UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

SHALLOW PROCESSING OF PORTUGUESE:

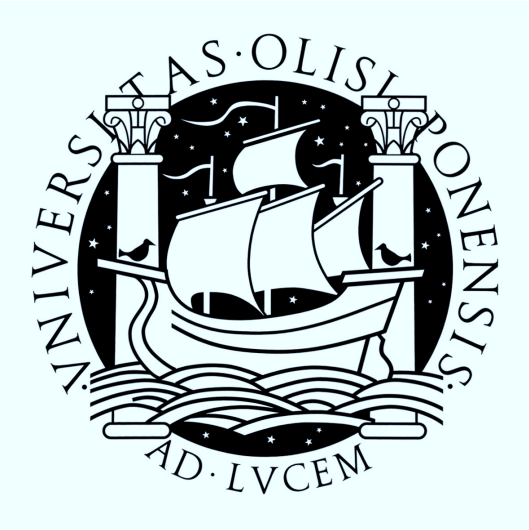
FROM SENTENCE CHUNKING TO NOMINAL LEMMATIZATION

João Ricardo Martins Ferreira da Silva

MESTRADO EM INFORMÁTICA

2007

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA



SHALLOW PROCESSING OF PORTUGUESE:

FROM SENTENCE CHUNKING TO NOMINAL LEMMATIZATION

João Ricardo Martins Ferreira da Silva

MESTRADO EM INFORMÁTICA

Dissertação orientada pelo Prof. Doutor António Horta Branco

2007

Resumo

Esta dissertação propõe um conjunto de procedimentos para o processamento computacional do Português. São cobertas cinco tarefas: Segmentação de Frases (*Sentence Segmentation*), Segmentação de Palavras (*Tokenization*), Anotação Morfossintáctica (*Part-of-Speech Tagging*), Traçamento Nominal (*Nominal Featurization*) e Lematização Nominal (*Nominal Lemmatization*).

Estas correspondem a alguns dos passos iniciais que produzem informação linguística, tal como categorias morfossintácticas ou lemas, informação esta que é importante para o processamento subsequente (e.g. análise sintáctica e semântica).

Sigo uma abordagem baseada em processamento superficial (*shallow processing*), segundo a qual a informação linguística é associada ao texto com base em informação local (i.e. usando uma palavra ou, no máximo, uma janela muito limitada de contexto que inclui apenas algumas palavras).

Começo por identificar e descrever as dificuldades encontradas em cada tarefa, com especial ênfase para aquelas que são específicas do Português.

Após uma panorâmica das abordagens e ferramentas já existentes, descrevo soluções para os problemas que foram apontados previamente. São também cobertas as implementações destas soluções que, após avaliação, revelam quer um desempenho ao nível do estado da arte quer, em alguns casos, um avanço no estado da arte.

O resultado desta dissertação é então tripartido: Uma descrição de alguns problemas chave encontrados no processamento superficial do Português, um con-

junto de algoritmos e as respectivas implementações para a resolução desses problemas, juntamente com a sua avaliação.

PALAVRAS-CHAVE: Processamento de linguagem natural, Processamento superficial, Segmentação de frases, Segmentação de lexemas, Anotação morfossintáctica, Análise morfológica, Lematização.

Abstract

This dissertation proposes a set of procedures for the computational processing of Portuguese. Five tasks are covered: Sentence Segmentation, Tokenization, Part-of-Speech Tagging, Nominal Featurization and Nominal Lemmatization.

These are some of the initial steps producing linguistic information — such as POS categories or lemmas — that is important to most subsequent processing (e.g. syntactic and semantic analysis).

I follow a shallow processing approach, where linguistic information is associated to text based on local information (i.e. using the word itself or perhaps a limited window of context containing just a few words).

I begin by identifying and describing the key problems raised by each task, with special focus on the problems that are specific to Portuguese.

After an overview of existing approaches and tools, I describe the solutions I followed to the issues raised previously. I then report on my implementation of these solutions, which are found either to yield state-of-the-art performance or, in some cases, to advance the state-of-the-art.

The major result of this dissertation is thus threefold: A description of the problems found in NLP of Portuguese, a set of algorithms and the corresponding tools to tackle those problems, together with their evaluation results.

KEY WORDS: Natural language processing, Shallow processing, Sentence segmentation, Tokenization, Morphosyntatcic annotation, Morphological analysis, Lemmatization.

Agradecimentos

O trabalho que aqui se apresenta é o resultado de vários meses de trabalho, ao longo dos quais tive o apoio — directo e indirecto — de várias pessoas e instituições, que merecem os meus agradecimentos.

Ao meu orientador, o Prof. António Branco, devo uma contribuição fundamental, por ter efectivamente orientado. Esteve sempre disponível para me apontar na direcção certa de todas as vezes que enveredei pelo caminho errado.

Agradeço ao Grupo NLX por me dar a disponibilidade necessária para me dedicar à dissertação e, através do Departamento de Informática, um local de trabalho. Adicionalmente, agradeço à FCT pelo financiamento dado no âmbito dos projectos TagShare e QueXting.

Agradeço também aos meus colegas, por ajudarem a manter um bom ambiente na sala (e por irem almoçar ou lanchar quando eu quero).

Finalmente, agradeço à minha família (gatos incluídos), pelo apoio incondicional durante todo este tempo em que tenho estado mais ausente.

Obrigado a todos. Embora o LaTeX seja sem dúvida uma grande ajuda, uma dissertação não se escreve sozinha.

Lisboa, Abril de 2007

João Ricardo Martins Ferreira da Silva

Aos meus familiares, amigos e, claro, gatos.

Contents

List of Figures

List of Tables

1

Introduction

The advent of digital computers in the early 1940's paved the way for a revolution in data processing by allowing complicated calculations to be performed with precision and speed.

Initially, the ongoing World War was the driving force behind most of the applications of computers (ballistics, development of the atomic bomb and cryptography). But soon after WWII, the field of Artificial Intelligence (AI) arose as an attempt to create intelligent programs.

Language is one of the most distinctive features of being human and a hallmark of intelligence. That is perhaps why Natural Language Processing (NLP) was one of the first fields tackled by AI research.

From a computational point of view, unprocessed text is nothing more than a string of characters, and a major instrumental goal of NLP is to associate linguistic information to this string. Such information may range from simple data about the boundaries of each sentence to a fully-fledged semantic analysis that supplies a representation of the meaning in the text.

Ultimately, the existence of a program that "understands" natural language would have a far-reaching technological and sociological impact, allowing for applications in many areas, such as improved human-computer interfaces, automatic translation, knowledge acquisition and information retrieval.

NLP shares a common trait with many other areas of AI that attempt to emulate some human capability (like Speech Recognition or Computer Vision): Processing a natural language or identifying an object in an image is something that comes easily (perhaps even innately) for humans, but it has proved to be very difficult for computers to do, as opposed to "number crunching" tasks, where computers undoubtedly excel.

Nonetheless, the difficulty (some would say impossibility) of accomplishing the ultimate goal of NLP — "understanding" natural language as a human does — should not drive our attention away from the fact that there are many useful applications of NLP that do not require a human-level understanding of language: Improving information retrieval, supporting the semantic web, rough translations, etc.

In particular, the NLP tasks covered in this dissertation could be considered to be very low-level as all of them fall below the level of fully-fledged syntactic processing. However, they are fundamental for the subsequent phases of processing and, even at this stage, already present non-trivial challenges.

1.1 Ambiguity

A major difficulty in processing natural language lies in the handling of ambiguity. This is a problem that, under different guises, arises in every stage of processing, and ranges from semantic ambiguity that requires real-world knowledge to be solved, to low-level (though not necessarily easier) issues like deciding whether a particular occurrence of the period symbol marks the end of a sentence or is part of an abbreviation.

There is a well-known anecdote (found, for instance, in (Pinker, 1994))

that serves as an illustrative example of the problems raised by ambiguity: One of the first parsers, developed at Harvard in 1960, was given the proverb "time flies like an arrow". The parser identified five different possible interpretations (arising from both lexical and structural ambiguity):

1. time (the abstract entity) goes by very quickly
2. time flies (a kind of insect) enjoy arrows
3. time (the act of measuring) flies that look like an arrow
4. time (the act of measuring) flies in the same way an arrow would
5. time (the act of measuring) flies like you would time an arrow

Naturally, the proverb is used in the first sense, but one normally does not realize that there are other interpretations without making a conscious effort or if the other interpretations happen to be highly favored by a rare context.

As human beings, we are permanently using inferences drawn from extensive world-knowledge and common-sense to discard many unsuitable interpretations. All of this is done unconsciously, which prevents us from explicitly accessing the "processing steps" that are used for those putative mental operations. So far, NLP programs lack this filtering, and generate too many interpretations (though there have been attempts to lessen this problem through common-sense ontologies, like in the Cyc project (Lenat *et al.*, 1990)).

If left unresolved, ambiguity will accumulate: An ambiguous linguistic construct will cause a branching into different possible cases which, in latter stages of processing, can also be specified into their own ambiguous

cases. For practical systems there is, therefore, a great need of curbing this explosion of alternatives.

1.2 Shallow Processing

Shallow processing (SP) is an approach to NLP that gained popularity as an efficient way of mitigating the problem caused by ambiguity, and delivering results that are useful without resorting to more complex and expensive syntactic and semantic processing.

The core rationale of SP is straightforward: It associates linguistic information to text based on local information (i.e. using just the word itself or perhaps a very limited window of context).

As a consequence of using only local information, some ambiguous constructs can be resolved quickly (often, as soon as one presents itself). Even if ambiguity cannot be fully resolved, the problem space can sometimes be reduced by restricting the amount of possible alternatives. In either case, the growth of alternatives is greatly reduced.

However, to achieve this, SP tasks tend to be highly specialized because they need to address specific different cases of ambiguity. So, instead of an all encompassing tool, SP uses tokenizers, stemmers, multi-word expression detectors, named entity recognizers, noun phrase chunkers, word-sense disambiguators, prepositional phrase attachment resolvers, etc.

For the same reason, SP often combines statistical and symbolical approaches, allowing for a much better adaptation of each tool to the specificity of the task to be handled.

1.3 Main Results

The main results of this dissertation will be:

- The identification and description of the key problems raised by each of the following tasks: Sentence Segmentation, Tokenization, POS Tagging, Nominal Featurization and Nominal Lemmatization, with special focus on those problems that are specific to Portuguese.
- An overview of current approaches and tools to tackle those tasks.
- The description and implementation of state-of-the-art, shallow processing algorithms to solve those problems, as well as their evaluation results.

1.4 Contents

The remainder of this dissertation is divided into four chapters:

- Chapter 2 describes the tasks at stake and explains why each one is relevant to NLP. The specific problems raised by each task are addressed, with special attention being given to issues that are specific to Portuguese.
- Chapter 3 provides background information on current approaches to the tasks described in the preceding chapter as well as an overview of the tools that are available to handle them.
- Chapter 4 describes the algorithms that were devised to tackle the problems described in Chapter 2 as well as evaluation results of the state-of-the-art tools that were developed to implement them.

- Finally, Chapter 5 offers some concluding remarks and discusses possible improvements as well as directions for future research.

2

Tasks and Problems

This chapter describes each task and discusses why its relevancy to NLP. The specific problems raised by each task are addressed, with special attention being given to issues that are specific to Portuguese.

2.1 Overview

The motivation for this work was a need to automatically associate linguistic information to text for use by subsequent NLP modules or other applications. While some tools and resources already exist, they were found not to be fully suitable or available,[1] which lead me to devise new approaches and/or tools.

There are five tasks covered in this dissertation: (i) Sentence Segmentation, (ii) Tokenization, (iii) Part-of-Speech Tagging, (iv) Nominal Featurization and (v) Nominal Lemmatization (the last two are addressed under the umbrella of Morphological Analysis).

This particular set of tasks was chosen because there is an inherent sequence to the tasks that are to be performed. For instance, morphological analysis of nominal tokens requires a previous step of morphosyntactic tagging that identifies nominal expressions, which in turn, requires an al-

[1] This issue will be addressed in further detail in Chapter 3.

ready tokenized text. Besides, as the work progressed, these tasks have shown to provide a lot of material to be covered by a MSc plan of work.

As mentioned in Section 1.1, ambiguity is pervasive in natural languages and it is the main source of difficulty in each task. This is confirmed by a recurring pattern: For each task, on average, most cases are easy to handle, but there is a number of difficult cases that are invariably caused by ambiguity. This is in line with a saying commonly used in Engineering according to which the first 80% of the work requires 20% of the effort, while the remaining 20% of the work requires 80% of the effort.

In fact, the resolution of ambiguity can be considered as being the main problem that is addressed by this dissertation, albeit it appears under different guises for each task.

2.2 General Remarks

Before describing each task in detail, there are some overarching remarks worth being made.

It is necessary to impose some restrictions over the type of input that will be considered. In particular, the tasks described here are meant to be applied over plain text. Note that this does not preclude the annotation of a text that uses HTML, or a similar XML-like markup scheme, but the information in the tags will be ignored.

An important requirement is that the tasks — and, consequently, the tools that implement them — must maintain a monotonic growth of information, i.e. information from previous processing phases or from previous results is not discarded, which means that relevant procedures are easily

reversible.[2] An additional advantage is that, by following this principle, there is no need to make any assumptions about what information will be needed by subsequent steps of processing as everything is preserved and can be used at will.

2.3 Sentence Segmentation

The sentence is the maximum syntactic element in natural languages. The practical implications of this is that one must know where each sentence begins and ends to be able to analyze it syntactically. However, in computer terms unprocessed text is simply a string of characters, with no marked boundaries for sentences. It is the task of a sentence segmenter to detect and mark the boundaries of each sentence and paragraph.

In Portuguese, as well as in other languages with similar orthographic conventions, the ending of a sentence is marked by the use of a punctuation symbol from a known set, such as `.` (period), `?` (question mark), `!` (exclamation mark) or `...` (ellipsis), which will be called terminator symbols. That, coupled with the convention that sentences must start with a capital letter, supports the detection of sentence boundaries for most cases, as in example (2.1).[3]

```
<s>Isto é uma frase.</s><s>Isto também é.</s>        (2.1)
```

Similarly, there are orthographic conventions for marking the ending of a paragraph, typically by inserting one or more newlines.

[2] The alternative is to save every intermediate result.

[3] Markup tags are used as delimiters: `<s>` for sentences and `<p>` for paragraphs. Note also that a mono-space font will be used for examples throughout the dissertation.

Consequently, for most cases, sentence segmentation is easy. There are, however, harder cases which must be resolved, including the ambiguous use of the dash symbol (especially in dialog) and the handling of the ambiguity pertaining to the period symbol. These cases will be discussed next.

2.3.1 Dialog and Dashes

Before tackling the segmentation of dialog, one must first uncover the orthographic conventions that it adheres to, which in some cases are specific to Portuguese. Some of these conventions are not explicitly presented in conventional grammars or "prontuários," and might have to be inferred through an analysis of examples that are adopted as best practice by book publishers and are intuitively considered as being correct.

The paragraph is used to delimit turn-taking, i.e. the change of speaker. The first sentence in a character's turn is marked by a — (dash) symbol. If there is no narrator's aside, the sentences in a given turn are separated following the conventions for non-dialog. Both these cases can be seen in example (2.2).

```
<p><s>— Como estás?</s></p>
<p><s>— Estou bem, obrigado.</s><s>E tu?</s></p>
```
(2.2)

A narrator's aside is signaled by the same dash symbol that marks the start of dialog but the aside is not considered as a separate sentence. Therefore, it does not necessarily begin with a capital letter. Additionally, if the speaker's own utterance was to be terminated with a period, that symbol is omitted. However, other terminator symbols, such as exclamation or

question marks, may be maintained for intonational purposes, as seen in example (2.3).

```
<p><s>— Bom dia — disse ele.</s></p>
<p><s>— Olá! — exclamou ela.</s></p>
```
(2.3)

The narrator's aside can be an interruption of the speaker's turn. In that case, the narrator's aside is followed by a dash and the speaker's sentence continues after the aside is over. On the other hand, if the narrator's aside concludes a sentence, any following sentence, even if uttered within the same turn, must be initiated by a dash.

```
<p><s>— Eu cá — disse — também.</s></p>
<p><s>— Não — respondeu.</s><s>— Eu não.</s></p>
```
(2.4)

In fact, the difficulty that is found in segmenting these cases stems from the ambiguity caused by using the dash symbol for various purposes. In dialog, this symbol is used to mark (i) the beginning of a speaker's turn, (ii) the beginning of a narrator's aside or (iii) the retaking of a speaker's turn after a narrator's aside. In addition, the dash may also be used outside of dialog for delimiting a parenthetical expression, as in (2.5).

```
<s>De pouco mais se falou durante a
primeira — e única — conferência de imprensa.</s>
```
(2.5)

2.3.2 Abbreviations

Similarly to what happens in many languages, Portuguese orthographic conventions state that abbreviated words must be ended by a period symbol. This causes a problem in those cases where the abbreviation is fol-

lowed by a word starting with a capital letter, as there is an ambiguity between the period marking the end of the sentence, as seen in example (2.1), or marking an abbreviation within a sentence, as seen in example (2.6).

```
<p><s>Eu vi um acidente na Av. Lusíada.</s></p>      (2.6)
```

2.3.3 Other Cases

There are still a few other cases that, for completeness sake, should also be handled.

Embedded Quotations

A case that needs careful handling is that of quotations embedded in a sentence. Such quotations may be formed by more than one sentence and may include a variety of terminator symbols, but these should not be taken into account when delimiting the sentence that includes the quotation. This is exemplified in example (2.7).

```
<p><s>O presidente aconselhou “Votem!” a
todos os presentes.</s></p>                          (2.7)
```

The segmentation of these embedded sentences is to be handled by some subsequent stage of syntactic processing.

Sentence wrapping

Sentence wrapping is a process that inserts line breaks into the text in order to confine it within a set of margins.

The segmenter must thus be able to handle sentences that span more than one line.

Besides delimiting such sentences, the segmenter can also, for convenience of subsequent processing, unwrap the sentences.

Note that this issue is related with word wrap and hyphenated words, which will be addressed in Section 2.4.4.

Titles, headers, etc.

It is common to omit the final period symbol from titles and headers, which might make it hard to discriminate between one of these cases and that of a sentence being wrapped over two lines.

Note that, however, there are usually cues in the text that indicate that a given string is a title, such as separating it from the rest of the text by one or more blank lines (i.e. separating it by a paragraph).

2.4 Tokenization

A tokenizer takes a string of characters and delimits the tokens that it includes, such as words, punctuation marks and other symbols. In this respect, tokenization can be seen as a normalization step through which we enforce that each token be delimited in the same way.

Before proceeding, I must define what I will consider as a token. What at first blush seems an easy concept to define — a token is what is informally thought as being a "word," — in fact has fuzzy boundaries and there is little consensus in the literature about the subject (Grefenstette and Tapanainen, 1994; Habert *et al.*, 1998).

Taking, as an illustrative example, the possessive in English, Grefen-

stette and Tapanainen (1994) point out that "(...) in the Brown corpus the word `governor's` is considered as one token (...) In the Susanne corpus the same string is divided into two tokens `governor` and `'s` (...)".

The criterion I will follow is to separate into individual tokens those strings that, in subsequent processing, are to be considered separately. This entails separating punctuation from words, expanding contractions into their components, etc.

Under this definition, tokenization is typically an easy task because, as seen in example (2.8), the white-space is already used in text as a delimiter.[4]

$$\cdots \texttt{ um exemplo simples } \cdots \Rightarrow \cdots \mid \texttt{um} \mid \texttt{exemplo} \mid \texttt{simples} \mid \cdots \qquad (2.8)$$

However, there are also several non-trivial issues that must be addressed, which will be discussed next.

2.4.1 Punctuation, Symbols and Numerals

Text is not formed only by alphabetic characters. There is a whole set of other characters — which, for simplicity, I will call "symbols" — that include punctuation, non-punctuation symbols (`@`, `$`, `&`, etc.) and Arabic numerals. Several of these symbols may occur adjacent to words.

The tokenization of symbols is typically an easy task, as the set of characters that can form words has no elements in common with the set of characters that are used for symbols.[5] For this reason, any symbol that oc-

[4] In the following examples, the | (vertical bar) symbol is used to help visualize the boundaries of each token. The $\cdots$ is used to indicate a sentence that continues, and should not be confused with the ... (ellipsis) punctuation symbol.

[5] With some exceptions, like the hyphen being used for hyphenated compound words.

curs adjacent to a word can be easily recognized and separated from that word.

$$\texttt{Um, dois e três}\cdots \Rightarrow |\texttt{Um}|,|\texttt{dois}|\texttt{e}|\texttt{três}|\cdots \tag{2.9}$$

The non-trivial issue regarding the tokenization of symbols comes from realizing that, when tokenizing, information about the adjacency of tokens is lost. In other words, in a trivially tokenized text no information is kept about how the symbols occurred originally. While this might not be problematic for example (2.9), as we know that orthographic rules force the comma to be adjacent to the word that precedes it, in example (2.10) the comma[6] is tokenized the same way in both cases, even though it originated from different adjacency situations. This clashes with the requirement for monotonic growth of information.

$$\begin{aligned} \texttt{1, 2 e 3}\cdots &\Rightarrow |\texttt{1}|,|\texttt{2}|\texttt{e}|\texttt{3}|\cdots \\ \texttt{1,20 Euro}\cdots &\Rightarrow |\texttt{1}|,|\texttt{20}|\texttt{Euro}|\cdots \end{aligned} \tag{2.10}$$

So, even if at this point one may not envision a reason for a need to differentiate the commas in example (2.10), that information should not be discarded.

Recognize or tokenize?

Note that the second line of example (2.10) the numeral (viz. `1,20`), is tokenized into several parts.

This is due to a decision that must be made at this point: Tokenize these strings as a single token or tokenize them into parts, which will be recognized as forming a numeral by a subsequent named-entity recogni-

[6]Note that, in Portuguese, the comma is used as the decimal separator while the period is the thousands separator.

tion task.

I have opted for the latter approach. It simplifies the tokenizer, as it avoids the need to recognize more complex patterns and handles numerals in a uniform fashion (namely, always tokenizing them into parts).

Mixed forms.

A special case is that of forms with a mix of alphabetic and non-alphabetic characters, such as alphanumeric forms (`A4`, `F1`, `F-117A`, `EXPO'98`, etc.) and "punctuated" acronyms, like `N.A.S.A.`

Again, it is necessary to decide how to handle these cases. I have decided to tokenize them as a single token as it simplifies the tokenizer: They are easy patterns to recognize and it avoids the need to preserve adjacency information for the types of characters involved (which would have to be done if one were to tokenize mixed forms into parts).

2.4.2 Contractions

A typical task of a tokenizer may be the expansion of contracted forms. These are strings that, syntactically, correspond to a sequence of words, even though they are written as a single token.

In Portuguese orthography, most of these strings correspond to the contraction of a Preposition — usually `de` or `em` — with the Article or Pronoun that follows it: `da` (`de` + `a`), `nele` (`em` + `ele`), `consigo` (`com` + `si`), etc. There are also some cases where two Clitic Pronouns may be contracted, e.g. `lho` (`lhe` + `o`).

Expanding contractions is important and useful for subsequent uniform syntactic processing, as it allows to more easily capture syntactic generalities.

When tokenizing contractions, there are two non-trivial issues: (i) Preventing information loss when expanding contractions and (ii) handling ambiguous strings. These will be discussed next.

Preserving information.

Expanding a contraction may lead to a loss of information, if in a tokenized text there is no way of knowing whether a contractible sequence of tokens originally occurred contracted or not.

For instance, the Preposition `de` and the Demonstrative `eles` can be contracted, giving `deles`. If this would hold in every occurrence, the expansion of the contraction `deles` would not lead to a loss of information because any occurrence of the two tokens |`de`|`eles`| would necessarily be the result of the expansion of `deles`. However, this is not the case, as there are situations where the contraction of `de` and `eles` is not allowed. In particular, when the complement of a `de` Preposition is an infinitive Verb, the Preposition `de` does not form a contraction with the word (Article, Demonstrative, etc.) that follows it. An example of this, from (Pinto, 2004, pp. 58), can be seen in (2.11).

```
A maneira de eles obterem bons resultados
é estimular o seu trabalho.
```
(2.11)

Therefore, to maintain a monotonic growth of information, any contractible sequence of tokens in the output of the tokenizer must include information about the format the string that originated it, i.e. whether the tokens originally occurred separated or contracted.

Note that apart from breaking the monotonic growth requirement, the loss of information that occurs when expanding contractions also has the

effect of increasing ambiguity.

Several of the tokens that can form contractible sequences, when taken without any context, can accept a variety of morphosyntactic tags. For instance, the expansion of da will originate the two tokens |de|a|. These tokens will be annotated with morphosyntactic tags by subsequent processing stages which will not be aware of the original format of the string and will have to disambiguate between the several tags those tokens can receive (viz. |a| can be a Definite Article, a Clitic, etc.).

However, one can take advantage of the fact that when the two tokens |de|a| originated from the expansion of da, they are to be tagged with Preposition and Definite Article, respectively.

Keeping with the shallow processing rationale, one should use this opportunity to resolve some ambiguity at an earlier stage (and without requiring extra processing).

There is yet another issue that might lead to a loss of information.

The use of capital letters can be used to convey extra information about the text (e.g. beginning of sentence, being part of a title, etc.) which might be of interest to subsequent processing.

Typically, this is not problematic, as capitalization can be preserved when tokenizing simply by not altering the token. However, when expanding a contraction, the resulting tokens must in some way preserve the capitalization of the contraction they have replaced, as exemplified in example (2.12).

$$\begin{aligned} \texttt{das} &\Rightarrow \texttt{|de|as|} \\ \texttt{Das} &\Rightarrow \texttt{|De|as|} \\ \texttt{DAS} &\Rightarrow \texttt{|DE|AS|} \end{aligned} \tag{2.12}$$

Note that one can conceivably encounter some extreme situations with

mixed-case tokens like DeSSaS or àqUELe, which would require a more complex scheme to fully preserve information.

Handling token-ambiguous strings.

Some strings can be tokenized in more than one way (more specifically, as a single token or as two tokens), depending on their occurrence.

For instance, deste can be tokenized as the single token |deste| if it is an occurrence of a form of the verb dar,[7] or as the two tokens |de|este| if it is an occurrence of the contraction of the Preposition de and the Demonstrative este.

Even though there are only 14 such strings in Portuguese (Table 2.1 shows the token-ambiguous strings and their expansions), handling these cases correctly is of critical importance as they amount to 2% (which is still a considerable amount) of the corpus used for training and evaluating the tools in this dissertation.[8] In addition, these strings and their expansions correspond mostly to words from the functional categories. It is therefore expected that errors in their tokenization would cause a considerable degradation of accuracy for subsequent stages.

The difficulty in handling these cases arises from the fact that the decision on how to tokenize ambiguous strings depends on the particular occurrence of that string and the surrounding context and, therefore, on its morphosyntactic category. However, at this stage of processing, morphosyntactic categories have not yet been assigned to tokens, as that task typically requires a previous tokenization step.

One is thus confronted with a problem of circularity: To decide how

[7] *Pretérito Perfeito* tense, second person, singular.
[8] A 260,000 token corpus which will be described in greater detail in Chapter 4.

single token	expanded
consigo	com si
desse	de esse
desses	de esses
deste	de este
destes	de estes
mas	me as
na	em a
nas	em as
nele	em ele
no	em o
nos	em os
pela	por a
pelas	por as
pelo	por o

Table 2.1: Token-ambiguous strings

to properly handle token-ambiguous strings, one needs morphosyntactic tags; but before assigning morphosyntactic tags one typically expect an already tokenized text.

2.4.3 Clitics

In Portuguese, whenever a clitic pronoun occurs to the right of a verb, the clitic is concatenated with the verb by means of an interposing hyphen (Bergström and Reis, 2004, pp. 108). Nevertheless, the resulting string still corresponds syntactically to a verb and a clitic.

As it is done for contractions, it is important to separate the clitic from the verb for the sake of better capturing syntactic generalities. It is worth noting that clitic pronouns can also be spelled in a contracted form. In such cases, the contracted clitic form should be expanded[9] after being detached

[9]For instance, the expansions `lhas` ⇒ | `lhe` | `as` | seen in example (2.13).

from the verb, as seen in (2.13).

$$\begin{aligned} \texttt{dar-me} &\Rightarrow \texttt{|dar|me|} \\ \texttt{dar-lhas} &\Rightarrow \texttt{|dar|lhe|as|} \\ \texttt{dar-se-lhas} &\Rightarrow \texttt{|dar|se|lhe|as|} \end{aligned} \tag{2.13}$$

When the verb form is either in the *Futuro* or in the *Condicional* tenses, the clitic is placed in mesoclisis, i.e. instead of being attached to the right of the verb form, the clitic is inserted into the verb and delimited by hyphens. The tokenization of these cases should extract the embedded clitic, as seen in example (2.14).

$$\begin{aligned} \texttt{dar-me-ia} &\Rightarrow \texttt{|daria|me|} \\ \texttt{dar-lhas-ia} &\Rightarrow \texttt{|daria|lhe|as|} \\ \texttt{dar-se-lhas-ia} &\Rightarrow \texttt{|daria|se|lhe|as|} \end{aligned} \tag{2.14}$$

There are three non-trivial cases (again due to ambiguity) regarding the tokenization of clitics: (i) Handling the form changes that occur when attaching clitics to verbs, (ii) preserving information about the string format prior to tokenization and (iii) deciding if a given string is the concatenation of a Verb and a Clitic or if it is a hyphenated compound word. These cases will be discussed next.

Changes in form.

The first issue is originated by the orthographic rules that, under certain circumstances, alter the canonical form of a clitic pronoun and of the verb form that it is attached to (Bergström and Reis, 2004, pp. 44). The clitic pronouns that are affected are `o`, `os`, `a` and `as`.

When the clitic pronouns `o`, `os`, `a` and `as` are associated to a verb form that ends in a nasal sound (i.e. terminating in `ão`, `õe` or `m`), they are prefixed by `n` (yielding `no`, `nos`, `na` and `nas` respectively). For instance:

`dão-no`	and not	`dão-o`
`põe-no`	and not	`põe-o`
`fazem-no`	and not	`fazem-o`

(2.15)

The clitic pronouns `o`, `os`, `a` and `as` are prefixed with an `l` (yielding `lo`, `los`, `la` and `las` respectively) when occurring after the `nos` or `vos` clitics[10] or when occurring after the form `eis`.[11] Note that, in any of these cases, the `vos`, `nos` and `eis` will lose the final `s`. For instance:

`deu-no-la`	and not	`deu-nos-a`
`trouxe-vo-los`	and not	`trouxe-vos-os`
`ei-los`	and not	`eis os`

(2.16)

The clitic pronouns `o`, `os`, `a` and `as` are also prefixed with an `l` when attached to verb forms that end in `r`, `s` or `z`. In addition, the final letter of the verb form is removed (note that, since this removal is purely orthographic in nature, it might require the addition of diacritics to the verb to maintain the stress of the last syllable). For instance:

`dá-lo`	and not	`dar-o`
`traze-lo`	and not	`trazes-o`
`fê-lo`	and not	`fez-o`

(2.17)

[10] Even when the `nos` or `vos` occur in proclisis, like in: `Ele só no-la deu ontem.`

[11] There are arguments over whether the form `eis` is an Adverb or a Verb, but this distinction is not relevant for the issue at stake here.

It is worth noting that there is a particular case where the alterations to the verb form are not limited to adding diacritics: When the verb form ends in `ens` or `éns`, stripping the final `s` would leave a word ending in `n`, which is invalid. In such cases, the `ns` is removed and replaced by `m`. For instance:

`tem-lo`	and not	`tens-o`	
`vem-lo`	and not	`vens-o`	(2.18)
`advém-lo`	and not	`advéns-o`	

Finally, when the clitic pronouns `o`, `os`, `a` and `as` occur in mesoclisis they are also prefixed by `l`. In this case, the `r` to the left of the clitic[12] will also be removed, which might require the addition of diacritics to the verb. For instance:

`dá-lo-ei`	and not	`dar-o-ei`	
`dá-lo-ia`	and not	`dar-o-ia`	(2.19)
`trá-lo-ei`	and not	`trar-o-ei`	

Despite the variety of form alterations described so far, the main difficulty they cause is that some of the form alterations may give rise to ambiguous forms. This happens in two of the cases described previously, namely: (i) When a clitic attached in enclisis position causes a form alteration in the verb form or (ii) when `nos` is found attached to a verb form that ends in a nasal sound.

First case. In the first case, the alteration to the verb form consists in removing the final letter and, in some cases, adding diacritics. The ambiguity is due to the fact that, after the alteration, the string bears no direct information about which was the letter (viz. `r`, `s` or `z`) that was removed from the verb form when attaching the clitic.

[12] Due to the way clitics are placed in mesoclisis, there has to be an `r` to the left of the clitic.

Taking the attachment of the clitic o as a running example: The *á-lo forms may be formed by attaching o to ar, az or ás.

comprá-lo	can be	comprar + o	or	compraz + o	(2.20)
dá-lo	can be	dar + o	or	dás + o	

The *ê-lo forms may be formed by attaching o to er, ez or ês.

vê-lo	can be	ver + o	or	vês + o	(2.21)

The *i-lo forms may be formed by attaching o to ir, is or iz.

bani-lo	can be	banir + o	or	banis + o	(2.22)
condi-lo	can be	condir + o	or	condiz + o	

The *í-lo forms may be formed by attaching o to ir or ís.

poluí-lo	can be	poluir + o	or	poluís + o	(2.23)

And the *ô-lo forms may be formed by attaching o to or, ôr or ôs.

pô-lo	can be	pôr + o	or	pôs + o	(2.24)
compô-lo	can be	compor + o	or	compôs + o	

Second case. In the second case, the alterations to the clitic cause a clash between two clitic forms. The clitic os (third person, masculine plural), when attached in enclisis position to a verb ending in a nasal sound, will be altered to nos. However, this altered form is equal to that of the (unaltered) clitic nos (first person, plural). Therefore, when detaching a nos, found in enclisis position, from a verb ending in a nasal sound, one has to

Normal forms		
Person	Sing.	Plu.
1st	me	nos
2nd	te	vos
3rd	lhe	lhes
	o	os
	a	as

Altered forms		
nos	→	no
vos	→	vo
o	→	no, lo
os	→	nos, los
a	→	na, la
as	→	nas, las

Contracted forms			
	me	te	lhe
o	mo	to	lho
os	mos	tos	lhos
a	ma	ta	lha
as	mas	tas	lhas

Table 2.2: Clitic forms

decide if the detached clitic is os or nos, as seen in (2.25).

```
dão-nos   can be  dão + os   or  dão + nos
põe-nos   can be  pôr + os   or  pôr + nos        (2.25)
vêem-nos  can be  vêem + os  or  vêem + nos
```

Note that it is not the tokenizer's responsibility to resolve these ambiguous cases or even to undo form alterations that might occur. The normalization of word forms is a task to be handled by a subsequent lemmatization process.

Nevertheless, the detailing of the form alteration rules is important because it allows to strictly define what are the strings that might be considered as a verb form attached to clitics.

A summary of clitic forms is shown in Table 2.2.

Preserving information.

The second issue is again related to the need of preserving information about the original string for use by the subsequent stages of processing.

Note that the tokenized forms in (2.13) and (2.14) maintain no information about their format prior to tokenization. For instance, as seen in (2.26),

the two tokens |deu|a| might have originated from a variety of strings.

(a) deu a prenda ⇒ |deu|a|prenda|
(b) deu à Maria ⇒ |deu|a|a|Maria| (2.26)
(c) deu-a a todos ⇒ |deu|a|a|todos|

If tokenization would proceed as depicted in (2.26), there would be a loss of information regarding the original format of the string. Thus, to enforce the requirement for monotonic growth of information, some way of differentiating the resulting tokenized strings is needed.

This loss of information also has the effect of increasing ambiguity. For instance, in each of the cases above, the token |a| that immediately follows |deu| has a different syntactic category: In (a) it is a Definite Article, in (b) it is a Preposition and in (c) it is a Clitic.

The disambiguation of |a| would normally be performed by a subsequent morphosyntactic tagging stage. However, the format of the string prior to tokenization allows us to decide with certainty the morphosyntactic category for two of these cases.

In (b), as à is a contraction, the two resulting tokens, |a|a|, can be categorized with certainty as a Preposition and a Definite Article, respectively. In (c), the first |a|, by virtue of originally occurring in enclisis position, can be categorized with certainty as a Clitic (and, in addition, deu as a Verb).

As before, following the shallow processing rationale, one should take advantage of any opportunity to resolve ambiguity at an earlier stage. Especially if this does not entail any extra cost.

Note that there might be also a loss of information when tokenizing clitics in mesoclisis position: When the verb form that was split is nor-

Futuro	*Condicional*
ei	ia
ás	ias
á	ia
emos	íamos
eis	íeis
ão	iam

Table 2.3: Verb endings for *Futuro* and *Condicional*

malized, information about the position where the clitic had been inserted might be lost. This issue is not as serious, as mesoclisis replaces enclisis just for *Futuro* and *Condicional* tenses, and it is possible through morphological analysis to decide the position where a clitic in mesoclisis position is to be inserted into a verb.[13] Nonetheless, preserving this information has, at the very least, the advantage of avoiding, at no additional cost, the need for that extra processing step.

Hyphenated compound words.

Finally, the third issue is related to a possible ambiguity of the hyphen symbol: It is used to attach clitics to verbs but also to form hyphenated compound words.

Consequently, any string that can be seen as being a clitic attached to a verb (in which case the clitic should be detached from the verb) can also be considered as a potential hyphenated compound word (in which case the string is to be tokenized as a single token).

However, what at first could be a very complicated issue, turns out not to be very serious because such strings would be exceedingly rare (and, most likely, be foreign words).

[13]Table 2.3 shows the verb endings for the *Futuro* and *Condicional* tenses. When in mesoclisis, the clitic is inserted before the corresponding ending.

2.4.4 Other Cases

There are still a few other, relatively rare cases that, for the sake of completeness, should also be handled.

Word wrap and hyphenated words.

As a consequence of word wrapping, it may happen that a word has to be split over two lines to maintain an aesthetically pleasing justification of the text. Split words have to be recognized as being a single token.

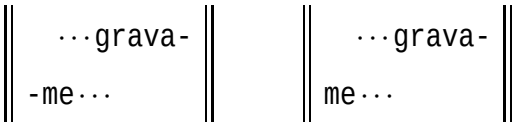
Detecting these cases is typically easy, as orthographic rules state that a hyphen must be appended to the first half of the split word. The difficulty that might arise from this issue is in the handling of those words that already include a hyphen and that are split over two lines. Not surprisingly, this difficulty stems from the ambiguity caused by the different purposes of the hyphen character in different occurrences.

If a hyphenated[14] word happens to be split in the same position where the hyphen occurs, the orthographic rules recommend using a second hyphen at the beginning of the second half of the split word to mark it. This, however, is not a mandatory rule (Pinto, 2004, pp. 179). So, when finding a split word marked with just one hyphen, one must decide if it is a hyphenated word or not.

Example (2.27) shows two situations where `grava-me` is being wrapped over two lines (the vertical double lines represent the margins). On the left, the orthographic recommendation was followed and a second hyphen was used to mark the second half of the split word (i.e. `-me`), while on the right a single hyphen was used, leading to an ambiguous situation (to make

[14] In an example of unfortunate ambiguous terminology, "hyphenated" is used to refer both to words that are connected by a hyphen and to words that are divided by a hyphen (when being split over two lines). To avoid confusion, I will not use "hyphenated" in the latter sense of the word.

matters worse, gravame is a word that actually exists in the Portuguese lexicon).



(2.27)

The "haver-de" forms.

When the preposition de occurs after a monossilabic, *Presente Indicativo* form of the verb haver, the preposition is attached to the verb by a hyphen (Bergström and Reis, 2004, pp. 31). Being separate syntactic elements, the preposition must be detached from the verb, as seen in (2.28).

```
hei-de  ⇒  |hei|de|
hás-de  ⇒  |hás|de|
 há-de  ⇒  |há|de|
hão-de  ⇒  |hão|de|
```

(2.28)

Note that, like what it is done with the tokenization of clitic pronouns, one should keep information about the format of the original string. If this is not done, there is no way of knowing, after tokenization, if the de preposition originally occurred attached to the verb form or not.

Example (2.29) shows two strings that, after being tokenized, would include the tokens |há|de|, even though only the first one corresponds to a "haver-de" form.

```
Sei que há-de chover.
Bilhetes, há de certeza.
```

(2.29)

Period ambivalence.

Whenever an abbreviation occurs at the end of the sentence, the period marking the end of the abbreviation is also used to mark the end of that sentence. This is done for aesthetic reasons, to avoid two consecutive period symbols. Nevertheless, one must consider that, in terms of tokens, two period symbols exist, viz. one marking the abbreviation and another marking the end of the sentence, as seen in (2.30).

$$\cdots\texttt{etc.</s>} \;\Rightarrow\; \cdots\,|\,\texttt{etc.}\,|\,\texttt{.}\,|\,\texttt{</s>} \tag{2.30}$$

Alternative terminations.

Another case is that of what will be called "alternative terminations," for lack of a better name. These are very particular constructs, usually found in text that is directed to unknown persons where one cannot use a specific gender or number in articles, nouns, adjectives, etc.

There are various ways of handling these cases. Example (2.31) shows four alternatives: (a) Consider the construct as a single token, (b) separate the termination, (c) separate everything, or (d) "expand" the construct.

$$\begin{aligned}
&\text{(a)}\;\texttt{Caro(a)} \;\Rightarrow\; |\,\texttt{Caro(a)}\,|\\
\text{or}\;&\text{(b)}\;\texttt{Caro(a)} \;\Rightarrow\; |\,\texttt{Caro}\,|\,\texttt{(a)}\,|\\
\text{or}\;&\text{(c)}\;\texttt{Caro(a)} \;\Rightarrow\; |\,\texttt{Caro}\,|\,\texttt{(}\,|\,\texttt{a}\,|\,\texttt{)}\,|\\
\text{or}\;&\text{(d)}\;\texttt{Caro(a)} \;\Rightarrow\; |\,\texttt{Caro}\,|\,\texttt{ou}\,|\,\texttt{cara}\,|
\end{aligned} \tag{2.31}$$

Note, however, that the last alternative falls outside the scope of tasks that are usually assigned to a tokenizer as it might require some sort of syntactic processing.

elided	full
d’	de
n’	em
pel’	por
m’	me
t’	te

Table 2.4: Elided forms

Apostrophe and letter suppression.

The apostrophe is used to indicate the suppression of letters in some very particular situations (Pinto, 2004, pp. 56).

As before, the normalization of the word form that suffers the elision is the responsibility of a subsequent lemmatization process. The relevant issue for the tokenization task is that some of these forms — listed in Table 2.4 — may occur attached to the following token.[15] In such cases, the elided form must be detached, as shown in (2.32).

$$\begin{aligned} \cdots\texttt{d'Os Lusíadas}\cdots &\Rightarrow \cdots\,|\,\texttt{d'}\,|\,\texttt{Os}\,|\,\texttt{Lusíadas}\,|\,\cdots \\ \cdots\texttt{n'A Selva}\cdots &\Rightarrow \cdots\,|\,\texttt{n'}\,|\,\texttt{A}\,|\,\texttt{Selva}\,|\,\cdots \end{aligned} \tag{2.32}$$

2.5 Part-of-Speech Tagging

One of the most well studied tasks in NLP is that of part-of-speech tagging, where tokens are automatically classified into morphosyntactic categories, or POS, such as Common Noun, Adjective, Verb, Preposition, Conjunction, etc. according to their context of occurrence.

This is an extremely important step in NLP, central to most applications: The linguistic information that is produced (morphosyntactic tags)

[15] Usually, a Definite Article in a title or denomination.

is highly useful for guiding the subsequent morphological and syntactic processing. For instance, syntactic processing without resorting to a previous morphosyntactic tagging step is possible, but the extremely high level of ambiguity would make parsing all but the most simple sentences a very expensive task in computational terms. In this respect, POS tagging can be envisaged as a shallow syntactic disambiguation task which, instead of performing a full-blown parse, just assigns syntactic categories (Manning and Schütze, 1999, pp. 341).

A given morphosyntactic category groups words that occur with the same syntactic distribution (Brants, 1995). This means that a token with a certain morphosyntactic category can be replaced in any context by another token with that same category and still the grammaticality of the relevant sentence is preserved.[16]

A brief exemplification of this can be found in (2.33), which mimics the card games that allow one to build sentences: Each card (the words delimited by brackets in the example) corresponds to a morphosyntactic category, and lists several words. To build a sentence, one needs only to pick a word from each card, following a given template. In (2.33), the template that is used is formed by four categories: Article, Noun, Adjective and Verb. By picking one word from each category in the template, one can create 24 ($= 2 \times 3 \times 2 \times 2$) different, grammatically correct sentences.

$$\left\{ \begin{array}{l} \texttt{O} \\ \texttt{Um} \end{array} \right\} \left\{ \begin{array}{l} \texttt{gato} \\ \texttt{cão} \\ \texttt{peixe} \end{array} \right\} \left\{ \begin{array}{l} \texttt{branco} \\ \texttt{rápido} \end{array} \right\} \left\{ \begin{array}{l} \texttt{saltou} \\ \texttt{fugiu} \end{array} \right\}. \tag{2.33}$$

The non-trivial issues that must be resolved for this task are (i) the

[16] Modulo subcategorization constraints.

tagging of those words that are lexically ambiguous with respect to POS, and in different occurrences can happen to be classified into different morphosyntactic categories (POS lexical ambiguity), (ii) the delimitation of the set of morphosyntactic categories (POS tagset selection) that will be used and (iii) the handling of categories that span more than one token (multi-word expressions). These issues will be discussed next.

2.5.1 POS Lexical Ambiguity

Ambiguity is present in virtually every natural language processing task, but it appears very prominently in the morphosyntactic tagging stage.

Several types belong to more than one morphosyntactic category and bear more than one POS tag. For instance, in different contexts of occurrence, the type `a` may be a Definite Article, a Preposition or a Clitic Pronoun.[17]

Naturally, a plain lexical look-up does not resolve this ambiguity as, at most, it can only return every possible category for the type corresponding to the relevant token. For instance, in (Branco and Silva, 2001, 2002), the authors are able to tag with a single POS tag ca. 64% of a corpus by using a plain lexical look-up for closed classes.[18] The remaining tokens, over one third of the corpus, are tokens whose type is lexically POS ambiguous.

Consequently, one needs a process for deciding which category should be assigned to a particular occurrence of a token whose type is lexically POS ambiguous. This decision is highly dependent on the context, possibly even influenced by long-distance syntactic relations.

An illustrative example can be obtained by envisaging a POS tagger

[17] And, if we wish to be fussy, the `a` may even be a Noun when referring to the letter "a" itself.
[18] Together with a few heuristics, such as detecting Adverbs by the `mente` termination.

that has to annotate the word `que` in (2.34).

In case (a), the token `que` remains ambiguous between Relative Pronoun and Conjunction, and the local context of `que` does not help to fully resolve the lexical POS ambiguity of the corresponding type `que`.[19] When adding an overt Subject to the sentence — the `ela` in cases (b) through (d) — the Relative Pronoun reading is dismissed. A Subject, however, can occur at an arbitrary distance of the word `que`, as seen in cases (c) and (d).

(a) ···disse ao amigo que saiu···
(b) ···disse ao amigo que ela saiu···
(c) ···disse ao amigo que ontem ela saiu···
(d) ···disse ao amigo que ontem à noite ela saiu··· (2.34)

This kind of decision tends to be particularly challenging for shallow processing approaches, which, by design, use only local information.

2.5.2 Tagset Selection

When selecting a tagset, a major issue is deciding its level of granularity.

In case one wishes to increase the discriminative power of the tagset, more fine-grained tags should be used. For instance, instead of having a single tag for Conjunction, the tagset may be further detailed into Coordinative and Subordinative Conjunctions. If an even finer granularity is desired, these tags can be further compounded, e.g. by dividing Coordenative Conjunctions into Additive, Adversative, Alternative, Conclusive and Explicative Conjunctions, as per (Cunha and Cintra, 1986, pp. 575).

However, one must also take into consideration how the tagset will

[19] But note that, with the context that is provided in example (2.34.a), not even humans can resolve this ambiguity.

be used and the type of application that has been envisaged for the tagger that uses that tagset. In particular, when doing shallow processing, taggers are often based on machine-learning algorithms. In this regard, having a large number of tags is undesirable as it may lead to higher data-sparseness and, consequently, a lower tagging precision (Brants, 1995).

The selection of a tagset must thus find a balance between these two opposing driving forces: Increasing the discriminative power of the tagset by using more refined tags and minimizing the data-sparseness by using a coarser tagset.

2.5.3 Multi-Word Expressions

Sag *et al.* (2002) roughly define multi-word expressions as "idiosyncratic interpretations that cross word boundaries". They also refer estimations that point out that the number of MWEs in a speaker's lexicon might even surpass the number of single words, meaning that this issue cannot be overlooked as being a minor problem.

A major issue here lies in what one should consider to be a MWE as there is a whole fuzzy continuum between compositional meaning and purely idiomatic (non-compositional) meaning.[20]

After deciding what is to be considered a MWE, an immediate approach is to build a list of such strings to match them while tagging, i.e. to consider MWE as being "words-with-spaces." For instance, "`visto que`" is to be tagged as a Conjunction and "`em direcção a`" as a Preposition. This approach, however, suffers from some drawbacks.

Firstly, some MWEs have internal inflection (i.e. one or more words

[20]Fixed, semi-fixed, syntactically-flexible, institutionalized, etc. See (Sag *et al.*, 2002) for an interesting overview of some types of MWE.

in them inflect). For example, the verb `pregar` in the MWE "`pregar um susto`" may inflect freely, wielding a great number of MWEs, a few of which are shown in example (2.35).

$$\left.\begin{array}{l}\texttt{preguei}\\ \texttt{pregas}\\ \texttt{pregamos}\\ \texttt{pregaria}\end{array}\right\}\ \texttt{um susto} \tag{2.35}$$

While not a fatal issue (as it could be handled by using larger lists of MWEs), it shows that this approach does not scale well and suffers from a lexicon proliferation problem.

Secondly, the tokens that form a MWE need not be consecutive. Example (2.36) shows how the MWE "`pregar um susto`" may accept a variety of "embedded" direct objects after `pregar`.

$$\texttt{pregar}\left\{\begin{array}{l}\texttt{-me}\\ \texttt{-lhe}\\ \texttt{-nos}\\ \vdots\end{array}\right\}\texttt{um susto} \tag{2.36}$$

This greatly increases the number of possible MWEs, because each alternative can be combined with each of the different inflections mentioned before. Nevertheless, the biggest difficulty caused by this is that, in some cases, what is embedded into the MWE may be arbitrarily complex, as seen in example (2.37). This shows that a simple string matching approach

is not powerful enough.

$$\texttt{pregar} \left\{ \begin{array}{c} \texttt{hoje} \\ \texttt{depois de amanhã} \\ \texttt{subitamente} \\ \texttt{um grande} \\ \vdots \end{array} \right\} \texttt{um susto} \tag{2.37}$$

Lastly, and more important, most strings that can form a MWE may also be considered as separate tokens, leading to ambiguity. Example (2.38) shows this with the string "`visto que`."

(a) `Vou dormir visto que já é noite.`
(b) `Encontrei o visto que perdi ontem.` (2.38)

This last problem entails that simply using a list of MWEs is not enough to resolve the issue.

In this dissertation, I will limit myself to those MWEs that belong to closed class categories, which greatly simplifies the problem as closed class MWEs do not allow for so much variety in terms of internal inflection and embedding.

MWEs from the open categories are to be handled by a task subsequent to POS tagging through the use of suitable techniques, like for instance the ones described in (Sag *et al.*, 2002).

2.6 Morphological Analysis

This dissertation only addresses the morphological analysis of tokens from the nominal categories: Adjectives, Determiners, Common Nouns, Pro-

nouns, etc. More specifically, it covers lemmatization and the assignment of inflection features and of a few derivational features as well.

Tokens from the verbal category will not be covered. This is due to the fact that the morphology of verbs is different and more complex (Branco *et al.*, 2006) than that of nominal categories and, under a shallow processing approach, it is sensible to isolate a simpler sub-problem to tackle it separately.

Note that the Participle is a verb form that can sometimes function syntactically as an Adjective (Bergström and Reis, 2004, pp. 61). These forms can also be partly handled at this stage, as if they were Adjectives.

Note also that, some authors (Mateus *et al.*, 2003, pp. 926) consider Gender to be a derivational, instead of an inflectional, process: Inflection is a systematic morphological process, i.e. if a category can inflect under a given feature, then all words in that category can inflect for that feature. Hence, Number is an inflection (every word has a singular and a plural form, except for some very rare cases) but not Gender (not every word has a feminine form, for example).

In my dissertation, it is not crucial to adhere to one of these views — whether Gender is an inflection or a derivation process — for addressing the NLP task at stake.

2.6.1 Nominal Featurization

Nominal featurization is the process of associating inflectional and derivational feature values to tokens of the nominal categories: Adjective, Common Noun and others that bear the same features (Article, Determiner, etc.)

In this dissertation, the featurizer thus handles Number and Gender.

It also handles some of the more regular and productive processes, like Diminutive and Superlative (the latter only applicable to Adjectives).

The key difficulties found in this task depend on the approach that is opted for. For a stochastic approach, the data-sparseness will be the key issue, while for a rule-based approach the ambiguity caused by invariant and lexically unknown words will come to be the critical problem.

Data-sparseness.

A possible approach to nominal featurization is to use the morphosyntactic tagger to assign feature tags together with the POS tags.

However, highly inflective languages may pose a problem for tagging technology based on statistics as, besides the usual POS tags, the tagger has to assign tokens with a great variety of additional information, such as the values for the features of Gender, Number, Degree, Case, Mood, Tense, etc. (Elworthy, 1995). Accordingly, this requires a much larger tagset, leading to a lower tagging precision due to the well known sparseness of data bottleneck: For the same amount of training data, a larger tagset will lead to the existence of more parameters for which there is no significant data available (Brants, 1995).

Another approach that may suffer from the same data-sparseness problems is that of automatically learning the morphology, be it unsupervised (Gaussier, 1999; Goldsmith, 2001) or supervised (Zajac, 2001).

Invariant and lexically unknown words.

If following a rule-based approach to nominal featurization, it is worth noting that getting the inflection feature values by mere lexical look-up is not enough. Following this option would require an exhaustive listing of

Word	Gender	Number
ermita	?	sing.
atletas	?	plu.
lápis	masc.	?
isósceles	?	?

Table 2.5: Sample lexicon entries for invariant words

the feature values for every possible word, which is not a viable approach as the lexicon, being permanently evolving and open to new words, can never be definitely complete.

Even worse, the existence of the so-called "invariant" words, which are lexically ambiguous with respect to inflection feature values, means that any approach that resorts to nothing more than a lexicon will not assign the relevant feature values to the occurrences of such words.

For example, the Common Noun ermita (*Eng.: hermit*) is singular, but it can be marked as masculine or feminine, depending on its specific occurrence. The Common Noun atletas (*Eng.: athletes*) is plural, but it can be marked as masculine or feminine, depending on its specific occurrence. The Common Noun lápis (*Eng.: pencil*) is masculine, but it can be marked as singular or plural, depending on its specific occurrence. The Adjective isósceles (*Eng.: isosceles*) can be marked with any combination of Gender or Number. By using nothing more than a simple lexical look-up, the output produced by a rule-based featurizer would always assign some underspecified feature values to such words.

2.6.2 Nominal Lemmatization

Lemmatization is the process by which a canonical, inflectionally normalized forms (the lemmas) are associated to tokens.

Word form	Lemma	Stem
lemma	lemma	lemm
lemmas	lemma	lemm
lemmatizer	lemmatizer	lemm
lemmatizers	lemmatizer	lemm
lemmatized	lemmatize	lemm
lemmatizing	lemmatize	lemm

Table 2.6: Lemmatization vs. stemming

The main motivation for doing lemmatization is to allow for more efficient searches in a lexicon: Instead of having to list every inflected form of a word, the lexicon only has to include the lemma.

In this regard, lemmatization is similar to the task of stemming, where several different word forms are also conflated into a single form. There is, however, a major difference that must be underlined: In stemming, it is not necessary for the resulting form (the stem) to be a genuine word, as stemming is typically used for Information Retrieval (IR), and its purpose is only to group "similar" words. As Porter states in (Porter, 1980), "[T]he suffixes are being removed simply to improve IR performance, and not as a linguistic exercise."

This difference is exemplified in Table 2.6 where 6 related words are shown being lemmatized and stemmed. The lemmatizer assigns the singular form to Nouns and the infinitive form to Verbs while a stemmer can possibly assign a same stem to all these different word forms.

In this dissertation, I will only perform lemmatization of content words that bear inflection affixes.[21] More specifically, I will perform nominal lemmatization, which targets words corresponding to the morphosyntac-

[21]Even though some of the words from other categories may also inflect, these constitute a closed list and can thus be explicitly listed. Lemmatization of such words can then be reduced to a simple table look-up.

tic categories of Adjective and Common Noun, assigning them the masculine singular form of the word, if it exists (this is also called the dictionary form).

To define the problem, it is important to precisely circumscribe the type of word forms that the nominal lemmatizer has to handle.

Since I am taking the lemma as being the "dictionary form," the lemmatizer has to handle Number and Gender, but not words that were obtained through derivation processes that create new words (i.e. new dictionary entries), like Adverb from Adjective, Noun from Verb, etc.

In addition, since the suffixes for Diminutive and Superlative forms are very regular and productive, I have decided to also handle those processes. This is in contrast with other processes (namely, augmentatives and prefixes) that have a much more irregular behavior and that often form new words.

As it happens with other NLP tasks, nominal lemmatization must handle ambiguous cases that exclude a mere lexical look-up as a viable approach. These will be discussed next.

Ambiguity.

The outcome of the lemmatization of a given word type may depend on the sense of its relevant token.

In some cases, such as with the word `ética`, the ambiguity can be resolved by knowing the POS of the token. For example, `ética`, when occurring as an Adjective, is the feminine singular form of *ethical* and should therefore be lemmatized into `ético`, the masculine singular form. However, when occurring as a Common Noun, `ética` has the meaning of *ethics*, and its lemma is `ética` instead.

There are cases, however, where different occurrences of a type, though bearing the same POS tag, can nonetheless receive different lemmas. For instance, the word `copas` may refer to the *hearts* suit of playing cards, in which case its lemma is `copas`, or it may be the plural form of `copa` (*Eng.: cupboard, treetop*), in which case it should be lemmatized into the singular form `copa`.

Note that ambiguous words such as `copas` cannot be resolved by any lemmatization process that is applied before a word sense disambiguation (WSD) stage. In fact, the existence of these words presents an inevitable upper bound for the shallow nominal lemmatization process, preventing this process from ever achieving total coverage of the targeted set of word forms.

3

Background and State-of-the-Art

This chapter provides background information on current approaches to the tasks described in the previous chapter as well as an overview of the tools that are available to handle them.

3.1 Sentence Segmentation

Sentence segmentation is a task that usually receives little attention in the literature (Mikheev, 2002), despite being one of the first tasks — if not the first — that is performed over raw text, and where annotation errors will have a detrimental impact over most of the subsequent processing stages.

There are two main approaches to Sentence Segmentation in the literature, viz. (i) manually built rules and (ii) machine-learning (ML) techniques (Mikheev, 2000).

This dichotomy presents the trade-off usually found in other uses of such approaches: Systems that use manually built rules can be tailored to handle very specific cases, but are hard to build and are not easily adaptable to other languages, while ML systems, though more robust and portable, require training data sets that are time-consuming to develop.

Note also that ML algorithms (decision trees, neural networks, etc.) can

typically be applied to a wide range of fields. To use them for a particular task it is necessary to define which features are relevant for that task. In a sentence segmentation task, these features would be facts such as "Is the following word uppercase?," "Is the current word an abbreviation?," etc. These features mirror, in a way, the same sort of cues that a rule-based system would look for in a text.

This is to say that, by taking an ML approach, one does not avoid having to come up with the features that are relevant for the task. This, in itself, can be difficult to achieve.

There are few rule-based tools dedicated solely to sentence segmentation since, in most cases, it is a functionality that is found embedded in some larger NLP tool, running as one of its first processing steps. The few standalone segmenters that I did find implement very trivial algorithms (e.g. always splitting when finding a period followed by a capital letter). A possible exception to this is the Perl[1] module Lingua::PT::PLN[2] for the NLP of Portuguese, but I was not able to run it.

There is a much larger amount of articles and tools available when searching for segmenters that use an ML approach.

Riley's classification tree.

In (Riley, 1989, pp. 351), a classification tree is used to detect the end of sentences, using features about the words on either side of punctuation marks to train the algorithm. This segmenter achieved 99.8% accuracy over the Brown corpus.

Note however that the model was trained over a large corpus with 25

[1] Perl — Practical Extraction and Report Language, a general-purpose programming language, widely used for text manipulation (Wall *et al.*, 2000).

[2] Cf. `http://search.cpan.org/~ambs/Lingua-PT-PLN-0.12/`

million words of news-wire texts.

SATZ.

SATZ (Palmer, 1994; Palmer and Hearst, 1997) is an efficient system that has shown good results over different languages: Typically it scores over 99% accuracy for English, German and French.

The main rationale of SATZ lies not so much in the ML algorithm that is used,[3] but in the use of POS information when analyzing the context that surrounds a potential sentence boundary.

By using POS to represent context — instead of individual words, — there is less data-sparseness and, consequently, SATZ can achieve good results with very small training corpora (e.g. a few hundred sentences).

In SATZ, each token receives a set of possible POS tags from a lexicon containing the prior probabilities of all POS tags for that token or, if a token is not present in the lexicon, based on a set of heuristics.[4]

This entails that while the training stage does not need large corpora, it requires nevertheless some additional work to collect POS probabilities for words (e.g. word lists with associated POS, a POS-annotated corpus, etc.)

MXTERMINATOR.

The MXTERMINATOR (Reynar and Ratnaparkhi, 1997) system is based on the same maximum entropy model used for POS tagging in (Ratnaparkhi, 1996).

[3]The authors evaluate two learning methods (neural networks and decision trees), finding similar scores for them.

[4]Common morphological endings, capitalized words, etc.

When evaluated over the Wall Street Journal Corpus, it scores 98.0% accuracy[5] after training over a set of approximately 39,500 sentences, although it can achieve already 96.5% with a much smaller, 500 sentence training corpus.

This segmenter is slightly worse than SATZ but it does not require POS tags or any other resource apart from a corpus annotated with sentence boundaries.

A potential disadvantage is that the feature templates used for training the algorithm (e.g. whether the word is an abbreviation, what is the word to the left, the word to the right, etc.) are fixed.

3.2 Tokenization

Like it happens with sentence segmentation, tokenization receives little attention and is often regarded as being a simple pre-processing step that is to be performed before the "real" linguistic processing begins. As a consequence, tokenization is often found embedded into a larger NLP task — as a pre-processing "cleanup" step — instead of being addressed independently (Grefenstette and Tapanainen, 1994; Habert *et al.*, 1998).

Tokenization is eminently a rule-based task which is typically implemented through the use of a finite state automaton (FSA).

Flex.

Tokenization is used outside of NLP proper. For instance, it is a fundamental step when building a compiler for a programming language.

[5] A better performing segmenter, with 98.8% accuracy, was obtained by creating a hand-crafted list of abbreviations.

Since building a FSA by hand is extremely hard, there have long existed tools to automatically generate a FSA from a set of rules. One of the most well known of these tools is Flex.[6]

In Flex a set of rules can be defined: These consist of patterns — described through regular expressions — that are associated with an action which is a piece of code in the C programming language. When the pattern matches a string in the input, the corresponding action is triggered (Paxson, 1988).

As the action may be any program, the only limit to the power of Flex lies in the fact that regular expressions have to be used to describe the patterns that fire the actions. Another possible disadvantage is that Flex is not specific to NLP, meaning that it does not provide any useful "shortcuts" that might be relevant in a specialized tool.

Freeling.

A good counterpoint to Flex can be found in Freeling, a set of open-source tools for NLP (Carreras *et al.*, 2004).

One of the modules in Freeling allows to easily create a tokenizer: By virtue of being a NLP-specific tool, the user has only to specify (again, through the use of regular expressions) which patterns correspond to tokens. The tool also provides a built-in support for abbreviations, by allowing the user to specify a list of abbreviations.

The shortcoming of Freeling comes from not allowing for generic actions to be triggered. In some cases (e.g. expanding contractions), it might be necessary to do more than simply delimiting the string that matches the

[6]Flex — Fast Lexical Analyzer Generator: A non-GNU, free, open-source implementation of the original Lex (Lesk, 1975) program.

pattern.

LT TTT.

The LTG's Text Tokenisation Toolkit (Grover *et al.*, 2000) provides a set of tools for text tokenization and mark-up.

Though its main purpose is to handle XML data, it can easily process plain text. In particular, it includes a general purpose transducer which rewrites strings according to a set of rules. This allows to easily model a tokenizer by, for instance, rewriting a matched word into itself or a contraction into its expansion.

In fact, the transducer is so general-purpose that it does not seem to provide any NLP-specific advantage over, for instance, Flex.

3.3 Part-of-Speech Tagging

POS tagging is probably the most well-studied problem from all the NLP tasks that are covered in this dissertation. Accordingly, there exists extensive literature about the subject as well as several available tools.

Given the large number of alternative approaches, I begin by grouping the type of approaches I will consider.

Firstly, I will consider machine-learning (ML) methods. The alternative would be to manually build sets of rules to assign POS tags, which is an extremely laborious task (Brill, 1995a,b). More important, even with the additional effort they require, manually built rules are not guaranteed to provide better results (Brill, 1992, 1995a).

For instance, Palavroso/MARv is a morphological analyzer (Palavroso) coupled with an ambiguity resolver (MARv) (Ribeiro *et al.*, 2003). After

training over a 230,000 token corpus, it scores 94.23% accuracy over a 60,000 token evaluation corpus. Under the same evaluation criteria, Brill's TBL scores 95.71% accuracy.

Note, however, that the two approaches are not always easily comparable. For instance, EngCG — a constraint-based morphological tagger, using hand-crafted rules — achieves very high scores for tagging precision but it does not resolve all ambiguities (Samuelsson and Voutilainen, 1997). If full disambiguation is required, EngCG alone is not enough.

Secondly, I will not consider unsupervised ML methods. An unsupervised POS tagger does not require a manually tagged training corpus. While this may be useful — as such a resource is hard and time-consuming to produce, — unsupervised POS taggers are less accurate than taggers built with supervised methods (i.e. using manually annotated training corpora) (Merialdo, 1994; Brill, 1995b). Since, for the present work, I have access to a manually POS-tagged corpus[7] for training and evaluation, i will take advantage of this fact by using supervised ML methods.

Supervised methods for POS tagging have been very successful. Even the extremely simple approach of assigning the most likely tag to each token can achieve 90% accuracy (Charniak *et al.*, 1993; Charniak, 1997).[8]

A possible explanation of this surprising result may be found in the uneven distribution of a word over the different POS that it may bear: Under a deterministic approach one can only state which POS a given word may bear (a binary, "true or false" value), while a statistical approach allows one to express the probability of a word belonging to a POS (Manning and Schütze, 1999, pp. 344).

For supervised methods, there is a large amount of tools available, such

[7] This corpus will be described in greater detail in Section 4.1.

[8] Given the simplicity of the approach, 90% is often considered the baseline for POS tagging.

as TBL (Brill, 1992, 1994, 1995a), TreeTagger (Schmid, 1994), MBT (Daelemans *et al.*, 1996), MXPOST (Ratnaparkhi, 1996), LT-POS (Mikheev, 1997), QTag (Tufis and Mason, 1998), TnT (Brants, 2000), the fnTBL toolkit (Florian and Ngai, 2001), one of the modules in FreeLing (Carreras *et al.*, 2004), SVMTool (Giménez and Màrquez, 2004), etc.

For this dissertation, I will only focus on three of them (viz. Brill's TBL, Brant's TnT and Ratnaparkhi's MXPOST). This decision is supported by a systematic evaluation and comparison of several well-known POS taggers, where these three are found to be the best (Megyesi, 2001).[9]

Brill's TBL.

Using ML methods does not preclude a rule-based approach. In fact, one of the most well known and widely used taggers is Brill's Transformation-Based Learner (Brill, 1992, 1994, 1995a).

The tagger begins by assigning each work its most likely POS tag (estimated from the training corpus). It then learns "patches" that are to be iteratively applied to the initial tagging. These patches are rules that change tags according to a template of the context, in view of improving overall accuracy.

This tagger has two main advantages over the more common stochastic taggers: (i) It requires less stored information as, instead of lengthy tables of stochastic parameters, it stores only a few transformation rules and (ii) the rules are a perspicuous form of storing linguistic information.

When evaluated over 5% of the Brown Corpus,[10] it achieved 95% accuracy after applying 71 patches to the initial tagging (Brill, 1992).

[9] Megyesi's results point to TnT as the best choice, with the highest overall accuracy, good handling of unknown words and the fastest training and tagging times.

[10] The Brown Corpus contains ca. 1.1 million words from various genres.

Brant's TnT.

Trigrams'n'Tags is a very efficient statistical POS tagger based on second-order Markov models (Manning and Schütze, 1999, pp. 317), using linear smoothing to mitigate data-sparseness and suffix analysis for handling unknown words (Brants, 2000).

Despite the simplicity of the underlying model, it achieves state-of-the-art performance, with an accuracy between 96% and 97%, depending on language and tagset.

Ratnaparkhi's MXPOST.

MXPOST attempts to improve accuracy by better use of context. For this, it uses a maximum entropy model (Ratnaparkhi, 1996). These models are able to combine diverse forms of contextual information in the form of feature templates like "current word," "previous tag," "next word," etc.

A limitation of this implementation is that these templates are fixed, i.e. the implementation does not allow for user-defined templates.

The tagger achieves 96.6% when tested over the Wall Street Journal corpus from the Penn Treebank. It was trained over ca. $960,000$ tokens and evaluation was performed over ca. $134,000$ tokens.

Despite its greater complexity, it does not score better than the simple Markov model in TnT. In fact, Ratnaparkhi (1996) compares MXPOST with other taggers (in particular, Brill's TBL) and finds that they all converge towards similar results, which leads the author to speculate that an accuracy limit has been reached for corpus-based algorithms, due to consistency problems in the corpus.

TBL	TnT	MXPOST
97.09%	96.87%	97.08%

Table 3.1: POS tagger accuracy

A better comparison.

The results shown so far are instructive, but cannot be easily compared, as the tagset, methods and corpora are different in each case.

In (Branco and Silva, 2004), the authors report on evaluation results for Brill's TBL, Brant's TnT and Ratnaparkhi's MXPOST. The accuracy measurements were obtained by averaging 10 test runs. In each run, the taggers were trained over a different 90% portion of a ca. $260,000$ token corpus and evaluated over the remaining 10%. The results that were obtained are summarized in Table 3.1.

Each of the tested taggers achieves state-of-the-art accuracy, in line with the results obtained for other languages when using similar tagging technology. Branco and Silva (2004) also report that, at the time, these appear to be the best results obtained for POS tagging of Portuguese.

3.4 Morphological Analysis

Morphological analysis consists in extracting the information that is encoded at the level of word-structure so that it may be used in subsequent processing stages.

According to this definition, I could include — as many authors do — the assignment of the base POS categories as part of this task. However, in this dissertation, I assume that such tags have already been assigned by a POS tagger that ran previously. In addition, as mentioned in Section 2.6,

this dissertation will only address the morphological analysis of tokens from the nominal categories. Consequently, I have to handle a much simpler sub-problem since verb inflection is much more complex (vd. (Branco *et al.*, 2006) for a characterization of the problem space when tackling verbal morphology).

3.4.1 Nominal Featurization

As mentioned in Section 2.6.1, the task of the Nominal Featurizer is to assign Number and Gender features to tokens that belong to the nominal categories (i.e. Adjectives, Common Nouns and others that can bear the same features).

As mentioned previously, a mere lexical look-up is not enough due to invariant and lexically unknown words.

An alternative, rule-based approach is to capture word ending regularities through rules. This is what is done, for instance, in JSpell (Almeida and Pinto, 1994; Simões and Almeida, 2000): It uses a dictionary of word roots, where each root is associated to a variety of information, such as POS, inflection features and lemma. Each root is also associated to a rule from a list of rules that allow to form new terminations and, in the process, alter the word's inflection feature values. To assign features to a given word, it looks for roots that, through their associated rules, can form the given word. The inflection feature values of the given word can then be determined through the knowledge of which rules were required to transforms the root into the given word. The authors do not provide evaluation results, and it is not clear how the system handles unknown words.[11]

[11]The JSpell system allows searching for similar words (i.e. as form of spellchecking), but that is not what is intended.

In many other systems, featurization is performed during the POS tagging stage, by a tagger using an extended tagset. A likely reason for this is that initial POS tagging research was mostly applied to English — a language with little inflection (Hajič, 2000) — where the issues raised by performing featurization during the POS tagging stage are not serious.

For languages with a richer inflection system, assigning inflection feature value tags with a POS tagger may raise a data-sparseness problem for ML approaches, as the base POS tags have to be greatly extended with a variety of feature information (Brants, 1995; Elworthy, 1995).

This problem is thus exacerbated as one moves towards massively inflected languages, such as Czech (Hajič and Hladká, 1997, 1998). Accordingly, most literature on this issue deals with the tagging of such languages. The most common approaches are (i) performing morphological analysis prior to POS tagging (Hajič and Hladká, 1998; Hajič, 2000) or (ii) tiered tagging (Tufis, 1999; Tufis *et al.*, 2000).

The level of inflection complexity of Portuguese, however, can be found between that of English and that of massively inflected languages. In addition, by restricting the task to nominal categories, I only have to handle a much simpler sub-problem. This militates against using the more complex methods for nominal featurization of Portuguese.

The task I address in this Section — that of dedicated nominal featurization — is very circumscribed. For that reason, I found no other tools for Portuguese with similar functionality.

3.4.2 Nominal Lemmatization

Often, the literature dealing with the subject of lemmatization is concerned with improving Information Retrieval (IR) Systems, where lemmatization

is used as a normalization step that conflates the various inflected forms of a word into a same base form, the lemma.

In this regard, lemmatization is similar to stemming. This seems to lead to some confusion in terminology, and sometimes a task is considered to be lemmatization when in fact, it is stemming. In (Silva and Oliveira, 2001), for instance, the authors claim to study several "lematizadores" for Portuguese which are actually stemmers.

This is an important distinction, as stemming is simpler than lemmatization. As mentioned in Section 2.6.2, the result from the stemming procedure needs not be a genuine word. Its only purpose is to gather "similar" words under a same form to improve IR performance (Porter, 1980).

Lemmatization, on the other hand, is more complex and follows linguistic criteria. The lemma is not simply a string that is common to various word forms, but a word form in a conventionalized "format." In Portuguese, for instance (vd. example (3.1)), Verbs are to be lemmatized into the infinitive form (case (a)), while the lemma for Adjectives and Common Nouns is the masculine singular form (case (b)), when it exists. If this form does not exist, the lemma is the singular form (case (c)). If even this form does not exist, the lemma matches the word form (case (d)).

(3.1)

(a)	`leio` (1st p. sing. *pres. ind.*)	→	`ler` (infinitive)
(b)	`gatas` (fem. plu.)	→	`gato` (masc. sing.)
(c)	`cobras` (fem. plu.)	→	`cobra` (fem. sing.)
(d)	`termas` (fem. plu.)	→	`termas` (fem. plu.)

There are three approaches that are used for lemmatization, and computational morphology in general: (i) using an extensive database that lists word forms, (ii) heuristic or rule-based affix stripping and (iii) finite state

approaches (Oflazer, 2006).

The first approach consists in listing word forms and their respective lemmas. Building such a lexicon is a hard and very expensive task, and it would never be a scalable approach as the lexicon is open to neologisms. In addition, as mentioned in Section 2.6.2, there are word types whose lemma depends on the POS — or even the sense — of its relevant token.

The second approach uses a set of rules or heuristics to "undo" morphographemic changes in word forms. This approach takes advantage of the regularities found in the inflection process, and can thus easily handle new words. Its downside is that it requires some effort to build rules and collect exceptions to those rules. However, most of this effort is only needed once, as the rules and exceptions rarely change (new words that enter the lexicon tend to follow the rules, instead of being exceptions).

The third approach uses finite state transducers that transform a string between its surface form and its lexical form.[12] This is a very powerful formalism that has been used for languages with complex morphology, like Turkish (Oflazer, 2006). It uses rules to transform the surface form into a sequence of morphemes and a lexicon to map that sequence to a lemma (and usually also to inflection feature values and POS, although, in this dissertation, these are handled in previous stages).

Similarly to what happens with featurization, circumscribing the target tokens to those from the nominal categories greatly eases the task. This militates against using the more complex methods.

For Portuguese lemmatization, JSpell (Almeida and Pinto, 1994; Simões and Almeida, 2000) — mentioned in the previous section — may be used, since the dictionary of word roots it stores is, in fact, a dictionary of lem-

[12] These transducers allow transformation to occur in both directions, i.e. analysis and synthesis (generation) of word forms.

mas. Its authors, however, do not provide evaluation results. In addition, this tool suffers from the same problems as the plain lexicon look-up approach: The handling of neologisms and of word types whose lemma depends on POS or sense.

4

Algorithms and Implementation

This chapter describes the algorithms that were devised to tackle the problems described in Chapter 2 as well as the evaluation results of the state-of-the-art tools that were developed to implement them.

Most of the work described in this dissertation was developed under the TagShare project.[1]

The project was undertaken by two academic research centers affiliated to the University of Lisbon: The Faculty of Sciences (FCUL), through the Natural Language and Speech Group of the Department of Informatics, and the Center of Linguistics (CLUL), through the Corpus Linguistics Group.

The main purpose of the project was to develop a set of linguistic resources (both tools and corpora) for the computational shallow processing of Portuguese.

Naturally, this dissertation deals mostly with the development of the algorithms and software tools. However, before describing the tools, a preliminary step should be the description of the corpus that was used for their training and evaluation.

[1]The project was funded by the Foundation for Science and Technology (FCT) of the Portuguese Ministry of Science and Technology (MCT) under the grant POSI/PLP/47058/2002.

4.1 Corpus

For the tools to be evaluated — and for them to be trained, as well, when applicable — there has to be properly prepared data sets. These data sets are known as a corpus.

The corpus used in this dissertation was prepared from a ca. $260,000$ token corpus provided by CLUL, composed mostly by excerpts from newspapers and fiction novels.

This corpus has evolved alongside the tools. For instance, its original POS annotation was changed to conform to the tagset used in this dissertation[2] and new annotation layers (inflection features, lemma, etc.) were added, and manually checked by trained linguists, as the corresponding tools were developed.

In this regard, it is important to note that, for most tools, there was not a suitable training corpus available for training ML approaches.

For instance, the corpus was not initially annotated with sentence and paragraph boundaries, invalidating a ML approach to sentence segmentation. Consequently, I had to opt for a rule-based segmenter. This segmenter could then be used to bootstrap a ML segmenter by providing an automatically segmented corpus that, after manual correction, can be used for training a ML segmenter.

4.2 General Remarks

Before entering into the details about algorithms and their implementation, there are some general remarks that can be made which are common to all SP tools addressed here.

[2] The tagset will be covered in Section 4.5.

4.2.1 On implementation

The general remarks about the implementation can be divided into four major issues: (i) the pattern-matching framework, (ii) the data structures used, (iii) the tool pipeline and (iv) the annotation scheme.

Pattern-matching.

Most of the tools described in this Chapter have, at their base, a procedure for pattern-matching in text.

The most usual approach to pattern-matching in text is to use a finite state machine that accepts the possible sequence of characters represented by the patterns.

Finite state machines are a well-studied subject in Computer Science, and there are already many utilities that ease their creation. One of such utilities, and also the one used for implementing the algorithms described in this dissertation, is Flex[3] (Paxson, 1988).

Flex receives a set of rules — pairs that assign regular expressions (patterns) to actions — and creates a source code file that, after being compiled, will produce an executable lexical analyzer (also known as a scanner). This scanner, when run, will find in its input text that matches one of the patterns and will trigger the corresponding action upon the matched text.

Note that when the text in the input is matched by more than one pattern, the pattern that matches the widest stretch of text is chosen (longest-match criteria).[4]

The power in Flex comes from the fact that the triggered action can be any generic piece of code written in the C programming language. Flex

[3] Flex — Fast lexical analyzer generator.

[4] In case of a tie, the pattern that was first declared in the Flex source file is chosen.

only hides from the programmer the complexity of the pattern-matching process.

Data structures.

Throughout this chapter, it will become clear that the tools will often have to access lists: Lists of abbreviations, lists of exceptions, etc.

However, the straightforward implementation of lists, the linked list, is a notoriously inefficient data structure and should be avoided.

There are several alternative data structures that have better performance than the basic linked list, such as red-black trees, splay trees, suffix trees, Patricia trees, tries, hashes, etc. (cf. (Cormen *et al.*, 2001), among others). Each has its own peculiarities. For instance, splay trees attempt to improve look-up time for frequently accessed items while tries optimize look-ups that are based on the prefix of words.

For this dissertation, I use AVL trees, which are an well-known implementation of balanced binary trees.

AVL trees are a good all-around data structure which are efficient in the operation that will be performed more often, the look-up of words.[5] As an additional advantage, I did not have to implement the data structure myself, thanks to Ben Pfaff's GNU libavl library.[6]

Tool pipeline.

The shallow processing rationale lends itself well to building small, specialized tools. In addition, there is a chain of dependency between the

[5]For this data-structure — the AVL tree — the look-up operation is $O(\log_2 n)$, where n is the size of the lexicon. This means that the size of the lexicon has to double before the search for an entry requires an additional step.

[6]Cf. `http://www.stanford.edu/~blp/avl/`

tasks, which means that these must run in sequence, with the output of each task being used as the input of the next one, in a pipeline scheme.

Accordingly, every tool runs on the command line under Linux,[7] where it is possible to take advantage of the easy pipelining that is provided by the | (pipe) operator. This pipeline of tools was named LX-Suite.

An on-line demo of LX-Suite can be found at the following website: `http://lxsuite.di.fc.ul.pt`. Note that LX-Suite includes more tools in the pipeline than the ones that are described in this dissertation. In particular, it includes a verbal morphological analyser, which assigns inflection features and lemmas to Verbs. A sample screenshot of the on-line demo of LX-Suite can be found in Annex A on page 143.

Annotation scheme.

One of the decisions that must be made is how to delimit tokens and how to represent the information that is assigned to them.

The whitespace character is used as the token delimiter. Linguistic information is assigned to tokens by appending tags to them, using special characters as delimiters: POS tags are appended to tokens using the / (slash) symbol as delimiter, inflection features are appended to POS tags with an intervening # (hash) symbol and lemmas are inserted between the word form and the POS tag,[8] using the / (slash) symbol as delimiter.

This is shown in example (4.1).

[7] The tools were only tested under Linux, but they should compile and run under other platforms provided the Flex libraries are made available.

[8] Having the lemma immediately follow the word form was found to facilitate the manual verification and annotation of the text.

`alta/ADJ`	`alta`, an Adjective	
`alta/ADJ#fs`	`alta`, a feminine singular Adjective	(4.1)
`alta/ALTO/ADJ#fs`	`alta`, whose lemma is `alto`	

This maintains a portable, plain text format and is easier to review manually than a markup-heavy format, such as XML.

Nonetheless, note that this is used as an internal format and it can be transformed by some tool further down the pipeline into a more sophisticated annotation scheme, such as one following CES, the Corpus Encoding Standard (Ide, 1998).

A corpus sample is shown in Annex B on page 145.

4.3 Sentence Segmenter

In this section I will describe the segmenter that I developed. The segmenter uses markup tags: `<s>` for sentence and `<p>` for paragraph.

As mentioned in Section 4.1, the corpus was not initially annotated with sentence and paragraph boundaries, which precluded a ML approach. Note also that the segmenter was used to provide an automatically segmented corpus that, after manual correction, was used to evaluate the tool.

4.3.1 Algorithm Outline

As mentioned in Section 2.3, sentence segmentation is, for most cases where a segmentation decision needs to be made, an easy task. The critical cases are the handling of abbreviations and the segmentation of dialog.

Initiator symbols		Terminator symbols	
A,B,C	capital letter	.	period
0,1,2	digit	?	question mark
...	ellipsis	!	exclamation mark
(...)	ellipsis	...	ellipsis
' "	begin quote	(...)	ellipsis
		' "	end quote

Table 4.1: Sentence initiators and terminators

Using Flex, one can define the set of terminator symbols — those that can end a sentence (e.g. period, question mark, etc.) — and the set of initiator symbols — those that mark the beginning of a sentence (e.g. capital letters). Table 4.1 lists these symbols.

To segment the easy cases, it is a simple matter of creating a Flex pattern that detects a terminator symbol followed by an initiator symbol. That pattern is then assigned to an action that outputs a sentence boundary between the terminator and the initiator symbols.

This is shown in example (4.2).

```
<p><s>Isto é uma frase.</s><s>Isto é outra frase,
pertencente ao mesmo parágrafo.</s></p>
<p><s>Isto é uma frase.</s><s>Isto é outra frase,
um pouco mais longa do que a anterior.</s></p>
```
(4.2)

The detection of paragraphs was designed to work in two alternative modes: Either a single newline is enough to mark a new paragraph (one-nl mode), or at least two consecutive newlines (i.e. a blank line) are required to mark a new paragraph (two-nl mode).

In one-nl mode, sentences in the same paragraph occur in the same

line.[9] Thus, a newline character signals a new paragraph.

Two-nl mode assumes that sentences may have been word-wrapped "manually," i.e. sentences may include non-consecutive newline characters. In this case, two consecutive newline characters (i.e. a blank line) are required to signal a new paragraph.

Example (4.3) shows a short excerpt in these two modes. The • symbol represents a newline character.

$$\left\| \begin{array}{l} \texttt{ Um parágrafo que se} \\ \texttt{prolonga por várias} \\ \texttt{linhas.•} \\ \texttt{ Um outro parágrafo} \\ \texttt{que se prolonga por} \\ \texttt{várias linhas.•} \end{array} \right\| \quad \text{or} \quad \left\| \begin{array}{l} \texttt{ Um parágrafo que se•} \\ \texttt{prolonga por várias•} \\ \texttt{linhas.•} \\ \texttt{•} \\ \texttt{ Um outro parágrafo•} \\ \texttt{que se prolonga por•} \\ \texttt{várias linhas.•} \end{array} \right\| \tag{4.3}$$

The same rule that delimits sentences is used to delimit paragraphs by using a more complex action that, instead of simply outputting a sentence boundary, also looks at the characters that separate the terminator and initiator symbols. If the terminator and initiator symbols are separated by enough newline characters to mark a paragraph, the action outputs a paragraph boundary together with the sentence boundary.

[9] The sentences occur in the same line in terms of file encoding. Visually (e.g. in a text editor), the sentences may span several lines due to line wrap.

Abbreviations.

Abbreviations are terminated by a period. Consequently, they raise an ambiguity issue when followed by an initiator symbol, such as a capital letter, as this matches the pattern that marks sentence boundaries.

```
···o Sr. João disse···
```
(4.4)

In example (4.4), the segmenter should not output a sentence boundary between `Sr.` and `João`, even though the string matches the pattern for a boundary.

For this purpose, a Flex pattern is created that matches an abbreviation[10] followed by an initiator symbol. Upon finding a string such as the one in (4.4), this pattern will match the abbreviation — together with the following initiator symbol — and trigger an action that outputs the matched string, unaltered (i.e. it does not emit a sentence boundary). Since the string matched by this pattern is necessarily larger than the string matched by the pattern for sentence boundary,[11] the longest-match criteria of Flex ensures that the action for the latter pattern will not be triggered.

It is important to note that abbreviations may raise another problem of ambiguity. There are ambiguous strings which can be seen as being a word followed by a period (i.e. a word at the end of a sentence) or as being an abbreviation. These string are shown in Table 4.2.

This ambiguity cannot be resolved at this early stage, since it needs additional information, such as POS tags. Thus, it was decided not to include these strings in the pattern that recognizes abbreviations. Consequently,

[10] A list of abbreviations can be found in Annex C on page 147.

[11] The string formed by an abbreviation followed by an initiator symbol is necessarily longer than a single period followed by an initiator symbol.

string	one token	two tokens	example
dez.	|dezembro|	|dez|.|	Flores, comprei dez.
dom.	|domingo|	|dom|.|	Ele tem um dom.
mar.	|março|	|mar|.|	Vou nadar no mar.
set.	|setembro|	|set|.|	Este é o último set.
ter.	|terça|	|ter|.|	E melhor ter que não ter.

Table 4.2: Ambiguous abbreviations

the segmenter adopts the simple heuristic of always interpreting them as being a word followed by a period.

Dialog.

The sentence segmentation of the narrative of dialog is another tricky issue for this task. A reason for this is the ambiguity of the dash, which can be used to either (i) mark the beginning of a turn by a given character in the narrative, (ii) the beginning of a narrator's aside or (iii) the resuming of the relevant character's turn. Example (4.5) shows a dialog excerpt with those three different uses of the dash. Note that, outside of dialog, the dash is also used to mark parenthetical expressions.

```
<p><s>— Eu cá — João afirmou
ponderadamente — também.</s></p>
```
(4.5)

Example (4.5) shows that immediate context might not be enough for segmentation. For instance, a dash followed by an initiator symbol does not necessarily mark the beginning of an utterance since a narrator's aside may begin with a capital letter (which is an initiator symbol).

Hence, to properly segment sentences in the narrative of dialogs, it is necessary to maintain some sort of information about the current state, as a dash and its immediate context might not be enough to make proper

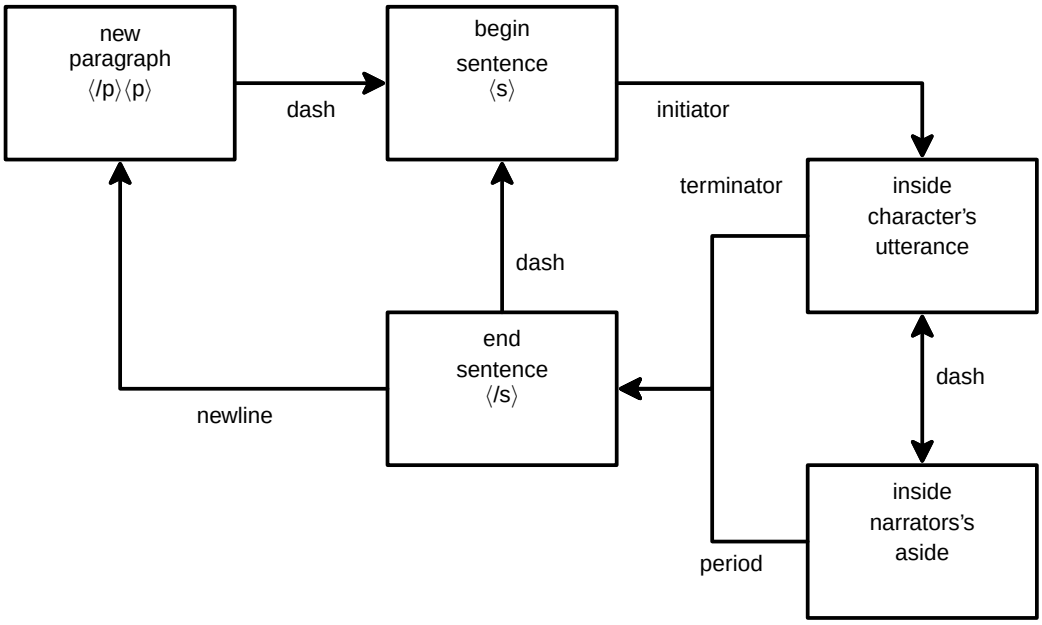


Figure 4.1: Turn segmentation in dialog

segmentation decisions.

For this purpose, the FSA switches between states that correspond to being in a character's utterance and being in a narrator's aside.[12] The outcome of finding a dash thus depends on the state the FSA is in.

Figure 4.1 illustrates this. Upon starting a new paragraph, a dash will signal an utterance. A second dash will mark a switch into a narrator's aside. From that state, a dash will indicate a switch back to the utterance.

Other cases.

There are a few other minor cases that are worth mentioning since the segmenter has special patterns to handle them.

[12]Flex provides a straightforward way of triggering actions that switch states. The state can influence which rules are applicable or allow for different actions for a same pattern.

Headers in enumerations. Headers are used to mark the items in enumerations and may appear in various formats, such as `3)`, `(iv)`, `b.`, etc. This latter form, in particular, may be problematic due to the period symbol.

To prevent the marking of a boundary after headers, these are identified by patterns that look for specific clues. For instance, a header must occur at the beginning of the line or be preceded by : (colon).

Ellipsis. The ellipsis poses a special problem since, as seen in Table 4.1, it may be used both as an initiator and as a terminator symbol. Consequently, finding an ellipsis followed by an initiator symbol might indicate a sentence boundary (4.6.a) or not (4.6.b).

(a) `···um grande mistério...</s><s>Como···`
(b) `···visto na escola de (...) Grbavci.</s>` (4.6)

The boundary is always placed after the ellipsis (i.e. this is taken as a terminator). When the ellipsis has the form `(...)` and it occurs between a terminator symbol and an initiator symbol it is taken as standing for a whole elided sentence and is thus segmented as `<s>(...)</s>`:

```
···é quase de terceiro mundo.</s>
<s>(...)</s>                                    (4.7)
<s>Mas a verdade é que···
```

4.3.2 Evaluation

For this task, I define recall as the proportion of boundaries that were correctly found and precision as the proportion of emitted boundaries that the segmenter got right.

The segmenter tool scored 99.94% recall and 99.93% precision over the

ca. 12,000 sentences in the 260,000 token corpus. This can also be seen in terms of error rates, with the segmenter having missed 0.06% boundaries and having emitted 0.07% extra boundaries.

These are very good values, better than the 98.0%–99.8% reported for the state-of-the-art ML tools mentioned in Section 3.1. This, however, is to be expected, since the rule-based segmenter was fine-tuned to the orthographic conventions of Portuguese, while the ML segmenters are general-purpose tools.

4.4 Tokenizer

For most tokenization decisions in a text, tokenization is an easy task and, in fact, one for which Flex is well suited since those decisions correspond to detecting and delimiting certain patterns in text.

4.4.1 Algorithm Outline

Since whitespace is already used in text as a token delimiter, most tokenization can in principle be done by defining a pattern that matches a sequence of non-whitespace characters and outputs it as a token, i.e. delimited by whitespace.[13]

There are, however, some cases where this is not enough namely, (i) the tokenization of punctuation and other symbols, (ii) the expansion of contractions and (iii) the detachment of clitic pronouns that are attached to verbs.

[13]The tokenizer uses the whitespace character as the token delimiter but, in the examples, the | (vertical bar) symbol is used to help visualize the token boundaries.

Punctuation and other symbols.

As mentioned in Section 2.4.1, the set of symbols has no elements in common with the set of characters that are used for words with some exceptions, like the hyphen being used for hyphenated compound words. For this reason, any symbol that occurs adjacent to a word, like a comma, can be easily recognized and separated from that word. The issue that remains to be solved is how to preserve information about the original setup in the input text.

Preserving information. A suitable solution is to tag the tokenized symbols with information about their original adjacency situation. More specifically, the tokenizer adds a mark to the left ($\backslash\star$) and/or the right side ($\star/$) of a given symbol if that side was originally adjoined to a blank character, as shown in example (4.8).

$$\begin{aligned} \texttt{1, 2 e 3}\cdots &\Rightarrow \quad \texttt{|1|,}\star\texttt{/|2|e|3|}\cdots \\ \texttt{1,20 Euro}\cdots &\Rightarrow \quad \texttt{|1|,|20|Euro|}\cdots \end{aligned} \tag{4.8}$$

Note that adjacency to the beginning or to the end of a line are also marked in this way. For instance, periods marking the end of a sentence are tokenized as $\texttt{|.}\star\texttt{/|}$.

Marking the tokens — e.g. the commas in example (4.8) — in such a way has the added benefit that, in future stages, like POS tagging or Named Entity Recognition, they will be considered as different tokens, capturing the fact that they may have different syntactic distributions and occur under different contexts.

Note that different symbols must be used to mark the left and the right sides. If the same symbol was used to mark either side, ambiguous situ-

ations might arise. For instance, if a $\star$ (asterisk) was used to mark space adjacency, marking a $\star$ (asterisk) symbol with either a left or a right space would produce the same result, viz. $\star\star$ (two consecutive asterisks).

Contractions.

To expand contractions, the tokenizer checks each word in its input against the list of contractions, which is a closed list (cf. Annex D on page 149) and, when finding a match, the expansion of the contraction is made to override the original word in the output.

As mentioned in Section 2.4.2, this raises two problems: (i) the preservation of information about the original input and (ii) resolving token-ambiguous strings.

The latter issue is important and complex enough to warrant its own section, and will be addressed in Section 4.4.2.

Preserving information. To preserve information, tokens originating from the expansion of a contraction are marked with a _ (underscore), as seen in example (4.9).

$$\begin{aligned} \texttt{da} &\Rightarrow \texttt{|de\_|a|} \\ \texttt{pela} &\Rightarrow \texttt{|por\_|a|} \end{aligned} \tag{4.9}$$

Note that marking only the first element of the expansion suffices. To recover the original string from the output of the tokenizer, one needs only to contract the marked token with the one following it. The set of contractible sequences and their results is known, and bears no ambiguity.

Note also that adding the contraction mark (_) might prove useful for the POS tagging stage, as the different resulting strings may better capture

differences in syntactic distribution.

To preserve capitalization of the expanded form, the tokenizer uses the heuristic depicted in (4.10): Lower case expands into lowercase, word initial capitalization into word initial capitalization, and all-caps into all-caps. Mixed-case is not fully preserved, as doing this would require a more complex annotation format.

```
das  ⇒  |de_|as|
Das  ⇒  |De_|as|                (4.10)
DAS  ⇒  |DE_|AS|
```

Clitics.

When in enclisis or mesoclisis position, clitics are separated from the verb to which they are attached.

To achieve this, the tokenizer looks for sequences of alphabetic characters with intervening hyphens. This is an overly general pattern that will match compound hyphenated words together with the strings that correspond to clitics attached to verbs.

Since the pattern itself imposes little restrictions, it is the triggered action that must perform some form of checking upon the matched string to prevent splitting a compound hyphenated word into parts. These checks fall into two types: (i) string matches and (ii) order constraints.

The triggered action begins by splitting the matched string at the positions where a hyphen occurs, as depicted in (4.11).

(a)	`dar-se-lhas-ia`	→	`[dar,se,lhas,ia]`	valid
(b)	`porta-voz`	→	`[porta,voz]`	invalid
(c)	`dia-a-dia`	→	`[dia,a,dia]`	invalid
(d)	`bruá-á`	→	`[bruá,á]`	invalid

(4.11)

Each of the split parts — except for the first one, which is assumed to be the verb form — is checked against a list of "clitic-like" strings, i.e. strings that appear in Table 2.2 (on page 25). To account for mesoclisis, the final part is also matched against the verb endings for the *Futuro* and *Condicional* tenses (cf. Table 2.3 on page 27).

For instance, in (4.11), cases (b) and (c) are deemed invalid, since `voz` and `dia` are not clitics or verb endings.

The simple string check eliminates most hyphenated compound words, but it might not be enough. Accordingly, the second check imposes restrictions on the order of the split parts. Namely: (i) a verb ending, if it exists, must be the last part and has to be preceded by some clitic and (ii) a contracted clitic may not be followed by another clitic.

For instance, in (4.11), case (d) is deemed invalid, since the verb ending `á` is not preceded by a clitic.

If the matched string fails the validity check, it is considered to be a hyphenated compound word, and is tokenized as a single token. Otherwise, the clitics are detached from the verb.

Preserving information. To preserve information, a similar approach as the one used for contractions is used, and a mark $(-)$ is prefixed to the detached clitic. Contracted clitics are expanded and marked.

$$\begin{aligned} \texttt{dar-me} &\Rightarrow |\texttt{dar}|-\texttt{me}| \\ \texttt{dar-lhas} &\Rightarrow |\texttt{dar}|-\texttt{lhe\_}|-\texttt{as}| \end{aligned} \tag{4.12}$$

Note that, besides preserving information, marking the detached clitics allows subsequent processes to differentiate between clitics in proclisis from clitics that occur attached to verbs (in enclisis or mesoclisis), like the Clitic a in example (4.13).

$$\begin{aligned} \texttt{não a deu} &\Rightarrow |\texttt{não}|\texttt{a}|\texttt{deu}| \\ \texttt{deu-a} &\Rightarrow |\texttt{deu}|-\texttt{a}| \end{aligned} \tag{4.13}$$

For mesoclisis, a mark ($-$CL$-$) is added to the verb in the position previously occupied by the clitic that was detached.

$$\begin{aligned} \texttt{dar-lhe-ia} &\Rightarrow |\texttt{dar}-\texttt{CL}-\texttt{ia}|-\texttt{lhe}| \\ \texttt{dar-lhas-ia} &\Rightarrow |\texttt{dar}-\texttt{CL}-\texttt{ia}|-\texttt{lhe\_}|-\texttt{as}| \end{aligned} \tag{4.14}$$

This mesoclisis mark could be omitted and still be possible to know where to reinsert the detached clitic. Nevertheless, placing a mark now avoids the need for that extra computation at no extra cost.

The tokenizer also marks (with a # symbol) those verb forms that may have undergone a form alteration due to clitization, i.e. verb forms that are attached to a lo, los, la or las Clitic.

$$\texttt{vê-las} \Rightarrow |\texttt{vê\#}|-\texttt{las}| \tag{4.15}$$

This is done so that a subsequent process that lemmatizes the verb form be informed that a form change occurred.

string	expansion	abs. freq.	as one token	as two tokens
consigo	com si	17	8	9
desse	de esse	33	6	27
desses	de esses	14	0	14
deste	de este	85	6	79
destes	de estes	35	0	35
mas	me as	1015	1015	0
na	em a	1314	2	1312
nas	em as	222	2	220
nele	em ele	11	0	11
no	em o	1450	14	1436
nos	em os	431	127	304
pela	por a	356	0	356
pelas	por as	69	0	69
pelo	por o	397	0	397
Total		5449	1180 21.66%	4269 78.34%

Table 4.3: Distribution of token-ambiguous strings

4.4.2 Token-ambiguous Strings

Table 4.3 shows the frequency of the ambiguous strings over the $260,000$ token corpus and their distribution in terms of how they are tokenized.

As pointed out previously, to properly handle ambiguous strings one has to know the POS of the token at stake. This information, however, is typically assigned after the tokenization step.

To break this circularity, I follow a two-stage approach to tokenization which also can be envisaged as POS tagging being interpolated into the tokenization process.

The basic rationale is as follows: Tokenization proceeds as normally, except that every token-ambiguous string is temporarily tokenized as a single token. After that step, the POS tagger runs over the result, assigning a POS tag to every token. Finally, a post-tagging tokenizer searches

tag	example
PREP+DA	no, pela
PREP+DEM	desse, deste
PREP+PRS	consigo
CL+CL	mas

Table 4.4: Portmanteau tags for token-ambiguous strings

specifically for the ambiguous strings and tokenizes them according to the POS tag they received.

To achieve this, the following steps are required:

1. The tagset has to be expanded with the new tags seen in Table 4.4. These are portmanteau tags that combine the POS tags that may occur in token-ambiguous strings.[14]

2. An adapted corpus is used for training the POS tagger. In this corpus, the token-ambiguous strings are always tokenized as a single token but receive a portmanteau tag when they correspond to a contraction. For instance, all occurrences of the token-ambiguous string deste are tokenized as a single token, but they receive the portmanteau tag PREP+DEM when they are an occurrence of the contraction (and, per usual, receive the tag for Verb, when occurring as such).

3. When the tokenizer first runs (the first tokenization stage), it does not alter the token-ambiguous strings. This produces a partly-tokenized text.

4. The POS tagger that was trained in step 2 is run over the partly-tokenized text, assigning a POS tag to each token.

[14] PREP: preposition, DA: definite article, DEM: demonstrative, PRS: personal pronoun, CL: clitic.

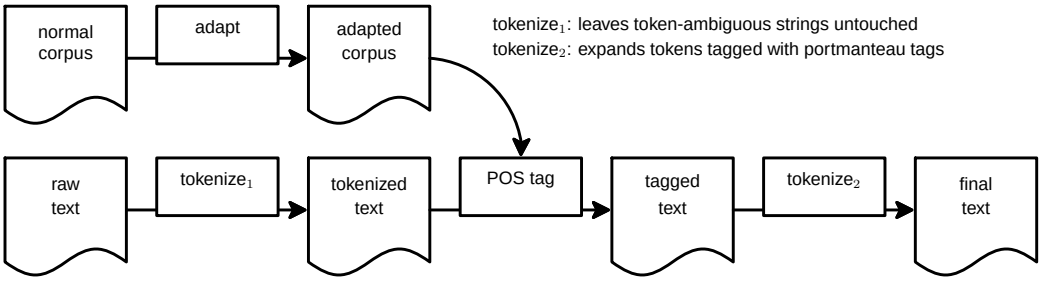


Figure 4.2: Two-stage tokenization

5. Finally, in a second tokenization stage, the post-tagging tokenizer looks for token-ambiguous strings tagged with portmanteau tags and expands them.

What this approach effectively does is to leave the decision on how to tokenize ambiguous strings up to the POS tagger.

A schematic display of the two-stage tokenization approach may be seen in Figure 4.2.

4.4.3 Other cases

The “haver-de” forms.

These forms are handled through the same mechanism that handles the tokenization of clitics: `de` is considered as being a special clitic form and the order constraints that verify the string’s validity do not allow any other clitics or a verb endings when `de` occurs. Consequently, the `de` is detached and marked, as if it would be a Clitic.

$$\begin{aligned} \texttt{hei-de} &\Rightarrow |\texttt{hei}|-\texttt{de}| \\ \texttt{hás-de} &\Rightarrow |\texttt{hás}|-\texttt{de}| \\ \texttt{há-de} &\Rightarrow |\texttt{há}|-\texttt{de}| \\ \texttt{hão-de} &\Rightarrow |\texttt{hão}|-\texttt{de}| \end{aligned} \tag{4.16}$$

Alternative terminations.

Alternative terminations are not handled in any special way, i.e. the tokenizer follows the usual criteria for the tokenization of punctuation and symbols, as seen in (4.17).

$$\texttt{Caro(a)} \Rightarrow |\texttt{Caro}|(|\texttt{a}|)\star/| \tag{4.17}$$

4.4.4 Remaining Issues

There are still some minor issues that the tokenizer does not handle.

Word wrap and hyphenated words.

The tokenizer does not reattach words that have been split over two lines.

$$\left\| \begin{array}{l} \cdots\texttt{grava-} \\ \texttt{-me}\cdots \end{array} \right\| \Rightarrow \cdots|\texttt{grava}|\texttt{-}\star/|\backslash\star\texttt{-}|\texttt{me}|\cdots \tag{4.18}$$

It thus assumes that the input was prepared to unwrap these cases.

Period ambivalence.

An ambivalent period is a period that concomitantly expresses two values: (i) the end of an abbreviation and (ii) the end of the sentence. In these

cases, the tokenizer does not output an extra period token to mark the end of the sentence.

$$\cdots\texttt{etc.</s>} \;\Rightarrow\; \cdots\,|\,\texttt{etc.}\,|\,\texttt{</s>} \tag{4.19}$$

Apostrophe and letter suppression.

The tokenizer handles the apostrophe symbol like any other symbol. Consequently, it does not consider strings with apostrophes as a single token, and the apostrophe is always tokenized as a separate token.

$$\begin{aligned} \cdots\texttt{d'Os Lusíadas}\cdots &\;\Rightarrow\; \cdots\,|\,\texttt{d}\,|\,\texttt{'}\,|\,\texttt{Os}\,|\,\texttt{Lusíadas}\,|\,\cdots \\ \cdots\texttt{n'A Selva}\cdots &\;\Rightarrow\; \cdots\,|\,\texttt{n}\,|\,\texttt{'}\,|\,\texttt{A}\,|\,\texttt{Selva}\,|\,\cdots \end{aligned} \tag{4.20}$$

Subsequent tasks should be aware of this. For instance, a syntactic analyzer must handle the two tokens $|\,\texttt{d}\,|\,\texttt{'}\,|$ as $|\,\texttt{de}\,|$.

Preservation of whitespace.

Regardless of how many spaces separated the tokens before tokenization, the latter will always be delimited by a single whitespace token delimiter character, as shown in example (4.21).

$$\begin{aligned} \texttt{um exemplo simples} &\;\Rightarrow\; |\,\texttt{um}\,|\,\texttt{exemplo}\,|\,\texttt{simples}\,| \\ \texttt{um exemplo simples} &\;\Rightarrow\; |\,\texttt{um}\,|\,\texttt{exemplo}\,|\,\texttt{simples}\,| \end{aligned} \tag{4.21}$$

To fully preserve information on whitespaces in the input text would require a more complex annotation scheme (e.g. by tagging each token with a number indicating how many adjoining spaces exist or by using

standoff annotation). I abstained from further pursuing this issue as I found no evidence of its relevance.

4.4.5 Evaluation

There are few sources for tokenization errors: (i) A compound hyphenated word — mistaken by a Clitic attached to a Verb — may be split into parts, (ii) a token-ambiguous string may be tokenized wrongly and (iii) any of the rare, not yet handled issues, may be found.

There are no occurrences of cases (i) and (iii) in the $260,000$ token corpus used for evaluation. Consequently, the evaluation of the tokenizer can be reduced to the evaluation of the two-stage approach to the tokenization of token-ambiguous strings.

Before evaluating the two-stage approach to tokenization, it is instructive to set baseline values for accuracy.

By looking back at Table 4.3 (page 79), a naive baseline measure can be immediately inferred: By always expanding ambiguous strings, the tokenizer achieves a precision of 78.34% over such cases.

However, an improved baseline can be easily reached by tokenizing to the most likely form (e.g. one token for `mas` but two tokens for `na`). With this technique, the tokenizer achieves 96.97% precision. This is a reasonably good value for a baseline that uses such a simple approach.

The highly skewed distribution seen in Table 4.3 seems to suggest that the two-stage approach will find it difficult to improve on this value: In the corpus that was used, some strings always tokenize to one of the forms (e.g. `mas` always as one token, `pelo` always as two tokens) and most of those that have occurred with both forms, present an uneven distribution (e.g. `na`, 2 times as one token and 1312 times as two tokens). Sparse data

such as this is typically bad for ML methods.

However, the two level approach performs extremely well. Using an auxiliary state-of-the-art POS tagger[15] the tokenizer implemented achieves 99.44% precision in the tokenization of ambiguous strings (i.e. it tokenizes wrongly ca. 30 of the 5449 ambiguous strings).

This high precision is likely due to the fact that ambiguous strings typically involve frequent tokens and that their local context tends to be quite different when that string is to be tokenized as a single token and when it is to be considered as a contraction.

If one considers these 30 errors over the whole $260,000$ token corpus, the tokenizer achieves 99.99% precision.

As mentioned in Section 3.2, tokenization is typically seen only as a pre-processing step within a larger NLP task and is not addressed independently. Consequently, I found no results for similar tokenizers against which to compare the precision of the tokenization tool developed here.

4.5 Part-of-Speech Tagger

The major issues dealt with in the development of the POS tagger were the selection of the tagset — including how to annotate multi-word expressions — and the POS tagging algorithm to be used.

For the present work, I had access to a manually POS-tagged corpus that can be used for evaluation and for training. I took advantage of that fact by using supervised ML methods.

[15]The POS tagger will be addressed in the next Section. It scores ca. 97% precision.

4.5.1 Tagset Selection

The selection of the tagset should take into account a balance between two opposing requirements: (i) The tagset should allow for high linguistic discrimination, by having a large tagset while (ii) avoiding data sparseness problems, which can be achieved by having a small tagset.

However, striking this balance cannot be reduced to blindly adjusting the number and assignment of tags. Syntactic categorization encodes basic linguistic generalizations about the distribution of lexemes, which by their own nature, are to be empirically uncovered, not superimposed in view only of stipulative convenience.

By definition, a morphosyntactic category identifies, under the same tag, tokens with identical syntactic distribution, i.e. tokens that in any occurrence receiving that tag, can replace each other while preserving the grammaticality of the corresponding linguistic construction, modulo the observation of suitable subcategorization constraints impinging over them. When the goal is the development of a top-accuracy and linguistically principled tagger that optimally supports subsequent syntactic parsing, this is a criterion that one cannot lose sight of in the choice of the tagset.

Taking the preceding considerations into account, there are possible "candidate" categories or subcategories that should not to be included in the tag set used to annotate the corpus over which the tagger is to be trained:

- Do not use different tags when there is no difference in syntactic distribution. An example of this are tags for the degree of an adjective, e.g. `alto/ADJ` (adjective) vs. `altíssimo/ADJ-sup` (superlative adjective).

- Do not use different tags when, even though there is a difference in distribution, the tag can be inferred from the form of the token at stake. For example, this is the case of tags indicating the polarity of an adverb, like `sim/ADV+` (positive adverb) and `nem/ADV-` (negative adverb), or tags indicating inflectional features, which can be subsequently determined from suffixes by a featurizer, as in the case of `alto/ADJms` (adjective, masculine singular) and `altas/ADJfp` (adjective, feminine plural).

- Also, when considering tag sets proposed in grammar textbooks of a more traditional, philological-oriented persuasion, it is not unusual to find categories aimed at indicating the constituency status of the phrase containing the relevant token. Such different tags encode information about whether the token at stake is a constituent of an elided or of a non-elided phrase but not an actual difference with respect to the syntactic distribution of that token. One example of this is the category "indefinite pronoun" versus some other category of closed classes. This category has been proposed for tagging articles, demonstratives or other pronominal items in headless Noun Phrases. For instance, according to such traditional views, the demonstrative `aquele` would receive `DEM` in the non-elliptical NP in (4.22.a) but it would receive a different tag in the corresponding elliptical NP in (4.22.b).

(4.22)
(a) `li [aquele/DEM livro]`$_{\text{NP}}$
(b) `li [aquele/INDPRON _]`$_{\text{NP}}$

Given that no difference with respect to syntactic distribution of items

like aquele is at stake (Branco and Costa, 2006), and in view of taming the data sparseness effect, the tags indicating the elliptical status of the containing phrase have no place in this tagset. Returning to the specific examples in (4.22), aquele receives the same tag (viz. DEM) on both cases.[16]

It is of note that the rationale discussed above and followed to circumscribe the tag set, not only helps to exclude possible candidate tags, but also to isolate and include categories that are usually not taken into account in a more traditional perspective.

Though being verbal forms, gerund, past participle and infinitive each have a distribution of its own due to the fact that they are the main predicators of subordinate clauses with specific distribution. Moreover, infinitival forms support nominative constituents (cf. (4.23.a)) and past participle can be used with adjectival force (such as in (4.23.b)). The tags GER, PTP and INF are thus included in the tag set to enhance the discriminative power of the tagger.

(a) [ouvir/INF música]$_{NP}$ diminui o stress
(b) o candidato eleito/PTP não chegou a tomar posse (4.23)

Other "non-canonical" tags are also included. These may be less interesting from a general linguistic point of view but they are important to improve also the contribution of the tagger for subsequent processing stages, in particular for named entity recognition (NER). These tags are shown in Table 4.5.

[16]Under more traditional approaches, single-word NPs like tudo are also proposed to receive the indefinite pronoun tag or a similar one. A IN (Indefinite Nominals) tag is included in the tagset to cover these cases.

Description	Tag	Examples
discourse marker	DM	adeus, olá, ...
social title	STT	Pres., Drª., prof., ...
part of address	PADR	Rua, Av., Rot., ...
month	MTH	Janeiro, jan., ...
day of the week	WD	Terça-feira, ter., Quarta, ...
measurement unit	UNIT	km, kg, b.p.m., ...
roman numeral	DGTR	MCV, LX, ...
fraction denominator	DFR	meio, terço, décimo, %, ...
order of magnitude	MGT	centenas, biliões, ...

Table 4.5: POS tags for NER

The full tagset is shown in Annex E on page 153.

Multi-word expressions.

In order to tag multi-word expressions from closed classes, a special tagging scheme is used where each component word in an expression receives the same tag (corresponding to the POS of the whole expression) prefixed by L, and followed by an index number corresponding to its position within the expression. This is shown in example (4.24).[17]

(a) apesar/LPREP1 de/LPREP2
(b) a/LCJ1 fim/LCJ2 de/LCJ3 (4.24)
(c) em/LADV1 o/LADV2 entanto/LADV3

This annotation scheme is well suited for shallow processing POS taggers, which use a limited window of context, since the special tag format used for MWE imposes strong co-occurrence constraints (e.g. a LCJ2 will not occur without a preceding LCJ1).

Note that, in some cases, when building the adapted corpus used for

[17] CJ: conjunction, ADV: adverb.

resolving token-ambiguous strings (cf. Section 4.4.2) it might be necessary to use portmanteau tags for tokens in multi-word expressions. For instance, the MWE seen in example (4.24.c) must be changed to that shown in (4.25) since `no` is a token-ambiguous string.

```
no/LADV1+LADV2 entanto/LADV3                    (4.25)
```

4.5.2 Tagger

Given the availability of a large number of ML, language-independent POS taggers, I used third-party tools instead of implementing a POS tagger from scratch.

More precisely, these third-party tools — namely, Brill's TBL (Brill, 1992, 1994, 1995a), Brant's TnT (Brants, 2000) and Ratnaparkhi's MXPOST (Ratnaparkhi, 1996) — are not simply POS taggers, but implementations of ML algorithms that allow one to build and run a POS tagger.

This distinction is reflected in the two modes of functioning that each tool has: (i) the learning mode (parameter generation), where several parameters are learned from the training corpus and (ii) the tagging mode, where these parameters are used by the tagging algorithm to annotate raw text.

As an example, I will describe in greater detail the parameters that are generated by TnT's learning component.

TnT's parameter generation

The learning component creates two files from the training corpus: (i) the lexicon, with the frequencies of words and their tags, and (ii) the n-gram file, containing contextual frequencies for uni-, bi- and trigrams.

```
-a       41   CL       41
compra   18   CN       16  V     2
consigo  17   PREPPRS  9   V     8
em_      666  LADV1    39  LCJ1  4  PREP  623
índice   20   CN       20
```

Figure 4.3: Part of TnT's lexicon

```
DA   25496
DA   CN     15740
DA   CN     PREP   5609
DA   CN     ADJ    1937
DA   CN     CJ     777
...
DA   ADJ    1114
```

Figure 4.4: Part of TnT's n-gram file

Figure 4.3 shows part of TnT's lexicon. The first column is the token, which is followed by its frequency in the training corpus. The remaining columns list the distribution of the token by POS.

For instance, the token `-a` occurs 41 times, always as a Clitic while `consigo` occurs 17 times, 9 of which as `PREPPRS` and 8 as a Verb.

Figure 4.4 shows an excerpt of TnT's n-gram file. Each line represents the frequency of the given sequence of POS tags.

For instance, the tag `DA` (unigram) occurs $25,496$ times. This tag is followed by a `CN` (bigram) $15,740$ of those times. The two tag sequence `DA-CN` is followed by `PREP` (trigram) in $5,609$ cases.

The algorithm (Brants, 2000) in the tagging component of TnT — i.e. the POS tagger *stricto sensu* — then applies the model parameters obtained in the previous step to the tagging of new text.

Integrating the taggers into the pipeline.

Each of the POS taggers that was used has different requirements on its input format and a particular output format. For instance, MXPOST requires one sentence per line and uses an underscore to separate a token from its tag, while TnT requires one token per line and uses a tab character as separator.

These formats differ from the adopted annotation scheme described in Section 4.2.1. Accordingly, to fit the POS tagger seamlessly into the pipeline, I implemented input and output "filters" for each tagger. These are simply text transformation scripts created using common text manipulation tools, such as sed (Dougherty and Robbins, 1997).

4.5.3 Evaluation

To evaluate the precision of the POS taggers, they were trained over 90% of the $260,000$ token corpus and evaluated over the remaining 10%. This was repeated over 10 test runs — each with a different 10% portion of the corpus being used for evaluation — and the results were averaged.

Evaluating over text that was not used in training allows for the occurrence of unknown words in the evaluation data. This is very important since it better mirrors "real" usage of the tagger, during which unknown words are likely to be found (Manning and Schütze, 1999, pp. 351).

Averaging the result of several runs over different partitions of the data is also important since it reduces the impact that a particular partition of the corpus might have.

Note also that each tagger has several command line parameters that can be adjusted. For instance, TnT allows to specify whether word cap-

TBL	TnT	MXPOST
97.09%	96.87%	97.08%

Table 4.6: POS tagger accuracy

italization should be taken into account, what method to use to estimate unknown words, etc. Naturally, adjustments to these parameters will also influence the precision of the taggers. For this evaluation, all taggers were used with their default settings.

The evaluation results are summarized in Table 4.6.

The taggers have similar accuracy values, with TnT scoring slightly worse than the other two. While this might appear unexpected — since in (Megyesi, 2001) TnT performed better — one must keep in mind that different algorithms may rank differently when tagging distinct languages and also display different sensitiveness to the size of training data.

Due to differences in corpora (language and size), algorithm, tagset size or evaluation procedure, it is not possible to do a direct comparison of scores against other POS taggers. A comparison is, nonetheless, instructive.

Comparison with the same tools for other languages

The scores that were obtained are in line with the current state-of-the-art results for POS tagging and, in particular, with the results reported for these three tools.

Table 4.7 compares the scores obtained here with the results reported on the corresponding tagger's reference paper. Since these papers differ in evaluation criteria, I also include the results reported by Megyesi (2001), who compared these taggers under the same terms.

TBL	TnT	MXPOST
97.09%	96.87%	97.08%
96.50% (Brill, 1995a)	96.70% (Brants, 2000)	96.60% (Ratnaparkhi, 1996)
92.39% (Megyesi, 2001)	95.31% (Megyesi, 2001)	93.49% (Megyesi, 2001)

Table 4.7: POS tagger comparison

The results obtained here are slightly better than the ones reported in the reference papers. The results in (Megyesi, 2001) are noticeably worse. The main causes for this are likely to be the target language in that paper (Swedish is a highly inflective language) and the training corpus dimension (only ca. 116,000 tokens).

Comparison with other POS taggers for Portuguese

When comparing with other POS taggers for Portuguese one finds that the taggers developed in this dissertation seem to currently be the best shallow processing POS taggers for Portuguese.

The Tycho Brahe tagger was developed by Alves (1999) using Brill's TBL as the underlying system. The corpus user for training has 130,000 tokens and the evaluation corpus has 45,000 tokens.[18] The tagger itself only achieves 88.24% precision, but additional refinement modules — e.g. morphological analysis, rule-based correction of common errors in the tagger's output, etc. — improve that score to 95.43% (Finger, 2000).

A similar approach is followed by Reis and Almeida (1998). They use TBL as the main tagger, which is coupled with JSpell (Almeida and Pinto, 1994) — a morphological analyzer that uses a dictionary and affix rules (al-

[18] I assume the training and evaluation corpora are disjoint, but this is never stated in the corresponding article.

ready mentioned in Section 3.4.1) — as a guesser for unknown words. The tagger was trained over a corpus with only 10,000 tokens. Their system is reported to achieve around 96% precision (nothing is specified about the evaluation method and corpus size).

Aires *et al.* (2000) evaluate four taggers (two of which — viz. TBL and MXPOST — are also covered in this dissertation). The taggers were trained over a 84,500 token corpus and evaluated over 10,000 tokens. The tagger with the best score is MXPOST, with 88.73% precision. This value is improved through a combination — a majority-wins voting strategy — of the four taggers, which achieves 89.42% precision. In her MSc dissertation, Aires (2000) improves on these results. MXPOST — which is still the tagger with the best standalone precision — achieves 89.66%. Combining taggers raises this score to 90.91%.

Marques and Lopes (2001) use a tagger based on neural networks. The tagger is further helped by a morphological analyzer that is capable of assigning all possible POS tags to over 1 million word forms. The corpus used has 18,675 tokens[19] from which 3 sentences are randomly removed to form an evaluation corpus. This process is repeated 10 times to get more stable results. The tagger scores ca. 96.3% precision.

Under the Lácio-Web project (Aluísio *et al.*, 2003) three taggers were trained over 80% of a corpus with ca. 1.2 million tokens. When evaluated over the remaining 20%, TBL scores 91.24%, TreeTagger scores 94.16% and MXPOST scores 95.53%. Note that these results were collected from the project's website[20] as I found no article containing this information.

For the sake of completeness, it is worth also referring taggers that use

[19]The authors report that the hand-tagged corpus has, on average, ca. 2.3 tags per word. It is not clear how this allows to evaluate the correctness of the tagging result.

[20]`http://www.nilc.icmc.usp.br/lacioweb/english/ferramentas.htm`

handcrafted rules, even though they fall outside the scope of this dissertation, which is concerned with shallow processing approaches.

These systems use a similar two-step approach: A morphological analyzer produces an initial annotation by assigning possible tags to tokens. A disambiguation procedure then chooses a single tag for each token.

PosiTagger (Aires, 2000, pp. 62) uses a set of handcrafted rules to correct an initial tagging performed through lexical look-up and affix rules. It achieves 82.65% precision. Note that it performs worse that the taggers based on ML methods that are also evaluated in (Aires, 2000).

The PALAVRAS tagger (Bick, 2000) is based on the Constraint Grammar approach introduced by Karlsson (1990). After a morphological analyzer assigns every possible POS tag to each token, the handcrafted rules are applied to select a particular tag or to eliminate some incorrect tags (which might leave some tokens with more than one POS). PALAVRAS was evaluated over several small texts in the range of $1,800$ to $4,800$ tokens, totaling ca. $14,000$ tokens. This tagger is reported to achieve high precision scores: When forcing "full disambiguation,"[21] its precision score ranges between 98.8% and 99.7% (Bick, 2000, pp. 187).

The Palavroso/MARv tagger (Ribeiro, 2003) is formed by a morphological analyzer (Palavroso) coupled with an ambiguity resolver (MARv). Palavroso assigns all possible POS tags to each token and MARv, using both handcrafted rules and probabilistic methods, chooses a single POS tag. The system scores 94.23% precision. Note that under the same training and evaluation conditions ($230,000$ tokens for training and $60,000$ tokens for evaluation), TBL is reported by Ribeiro (2003) to score 95.17%.

[21] Despite forcing "full disambiguation" (i.e. a single POS per token), it is reported that some tokens nevertheless remain with more than one POS in "cases of true ambiguity" (Bick, 2000, pp. 187). The author does not quantify how many such cases exist. Notice that all other figures by other authors reported in this dissertation concern actual full disambiguation in context.

4.6 Dedicated Nominal Featurizer

It is a common practice that the tagging of inflection features be done by a POS tagger with an extended tagset, where the base POS tags are enriched with information on a variety of inflection feature values. For instance, in such an extended tagset the base POS tag for Adjective would be replaced by four new tags, one for each combination of the two values for Gender (masculine and feminine) and Number (singular and plural). The number of new tags can even be higher if one wishes to consider additional features such as Diminutive, Superlative, etc.

`alto/ADJms`	Adjective, masc. sing.
`altas/ADJfp`	Adjective, fem. plu.
`altíssima/ADJfs-sup`	Adjective, fem. sing. superlative
`altinhos/ADJmp-dim`	Adjective, masc. plu. diminutive

(4.26)

When one considers all the POS categories whose words may receive inflection features, it becomes clear that the extended tagset will be much larger than the base tagset used in the previous Section (cf. Annex E on page 153).

This approach raises a problem for current stochastic taggers as an extended tagset is likely to lead to a lower tagging precision due to the data-sparseness problem: For the same amount of training data, as the tagset increases, there will be more parameters for whose estimation no significant data is available, with a decrease of the tagger's accuracy.

In this dissertation I address this problem by handling inflection tagging as a dedicated task, separated from that of POS tagging.

This rationale is partly inspired on the observation that, to some extent, a POS tag reflects a sentence-level syntactic distributional constraint, while an inflection tag reflects constraints that tend to be more markedly at the level of word-structure.

Of course, though POS and inflection can be seen as separate linguistic sub-systems, syntax and morphology are far from being independent of each other. My goal is thus not to argue for a reciprocal independence between morphology and syntax, but to study a different approach, empirically checking what can be gained in the overall tagging accuracy by following the rationale that inflection tagging is a separate linguistic system.

4.6.1 Algorithm Outline

The morphological regularities found in Portuguese suggest a straightforward rule-based algorithm for the autonomous assignment of inflection feature values given that word terminations are typically associated with a default feature value. For example, most words ending in `-ção`, like `canção` (Eng.: *song*) are feminine singular (vd. Annex F on page 157 for a listing of rules).

Any exceptions to this can then be easily found by searches in machine-readable dictionaries (MRD): The exceptions are words with the designed termination but with inflection features that do not match the default one. For instance, `coração` (Eng.: *heart*) is masculine singular.

Assigning inflection features can thus be done by simply searching for a suitable termination rule and assigning the corresponding default inflection tag if the input token is not one of the exceptions to that rule.

As mentioned in Section 2.6.1, the key issues that a rule-based to Nom-

inal Featurization faces are invariant and lexically unknown words.

Assigning inflection feature values through the method outlined above handles lexically unknown words in a straightforward manner, since new words that enter the lexicon tend to follow the rule instead of being exceptions. This leaves the handling of invariant words as the major difficulty that must be tackled. This issue will be addressed next.

4.6.2 Invariant words.

Invariant words are words that are lexically ambiguous with respect to inflection feature values.

For example, `ermita` (Eng.: *hermit*), depending on its specific occurrence, can be tagged as masculine or feminine. By using nothing more than rules and exceptions, the output produced by the rule-based featurizer would always be `ermita#?s` (i.e. singular, but with an underspecified value for Gender).[22]

Feature propagation.

To handle these cases, an algorithm can be envisaged that builds upon the fact that there is Gender and Number agreement in Portuguese, in particular within Noun Phrases (NPs): The Gender and Number feature values for the Common Noun in a NP determine the agreement of Determiners, Quantifiers, etc. within the same NP (Mateus *et al.*, 2003, pp. 330).

Consequently, we can determine the inflection feature values for an invariant Common Noun if we know the inflection feature values of other tokens with which the Common Noun agrees.

[22]Note that in these examples I will omit the POS tag for the sake of simplicity.

The procedure can be outlined as follows: All words from the closed classes that have inflection features (Demonstrative, Determiner, Quantifier, etc.) are collected together with their corresponding inflection tags.[23] During inflection analysis of a text, the inflection tags assigned to words from these closed classes are propagated to the words from open classes (Adjective and Common Noun) that immediately follow them. These may, in turn, propagate the received tags to other words.

Example (4.27) illustrates this procedure: The initial NP, shown in (a), is analyzed by the featurizer, which assigns underspecified features to the invariant words `ermita` and `humilde` (Eng.: *humble*), as seen in (b). The features assigned to the Definite Article `o` can then be propagated to `ermita` (c) and then to `humilde` (d).

(a) `o ermita humilde`
(b) `o#ms ermita#?s humilde#?s`
(c) `o#ms` → `ermita#ms humilde#?s`
(d) `o#ms ermita#ms` → `humilde#ms`

(4.27)

Note that example (4.27) suggests that this may be done in a two-pass procedure: In a first step, the featurizer assigns features based on termination rules and exceptions while in a second, separate step, propagation of features is performed.

This approach, however, would require two separate passes over the input file, which is not efficient. To avoid this, the featurizer algorithm is designed to perform a single pass, as shown in (4.28): As the featurizer scans the input, it maintains information about the feature that was last assigned (depicted by the tagged arrows). Upon reaching a token to which

[23] This can be done since the words from the closed classes form a closed list.

it cannot assign a fully specified feature tag through the use of rules (or their exceptions), it uses information about the last assigned feature.

(a) `o ermita humilde`
(b) `o#ms ermita humilde`
(c) `o#ms` $\xrightarrow{ms}$ `ermita#ms humilde`
(d) `o#ms ermita#ms` $\xrightarrow{ms}$ `humilde#ms`

(4.28)

As the featurizer begins to analyze the NP shown in (a), it assigns the masculine singular feature values to the Definite Article `o`, as seen in (b). This is done by a simple lexical look-up, since `o` belongs to a closed class. In (c), the featurizer finds an invariant word, `ermita`, but the feature values that have been assigned to `o` in the previous step allow the featurizer to assign a completely specified feature tag to `ermita`. In (d), a similar process occurs, this time using the feature values propagated from the assignment to `ermita`.

An additional optimization. The featurizer uses an optimized version of this algorithm. Upon finding a nominal token, the featurizer firstly tries to use the propagated features. Only if there is no propagated feature available — or if the propagated feature is itself underspecified — does it use the termination rules.

This optimization avoids the need for the lexicon look-up whenever a specified propagation exists, speeding up the featurization process. In addition, as it will be seen in Section 4.6.4, it can also increase the precision of the featurizer.

Propagation blocking.

The algorithm described above builds upon Gender and Number agreement within NPs. Consequently, in order to make use of this algorithm, one has to ensure that feature propagation occurs only within NP boundaries.

In addition, one must ensure that, even within a NP, propagation does not cross into other phrases found embedded in the NP. For instance, in (4.29), the Common Noun `história` is inside a Prepositional Phrase (PP) that is inside a NP and does not have to agree with `livros`.

$$[\texttt{livros\#mp}\ [\texttt{de}\ \texttt{história\#fs}]_{\text{PP}}]_{\text{NP}} \tag{4.29}$$

For this effect, some patterns of tokens and POS tags are defined such that, when they are found, tag propagation is prevented from taking place. The blocking patterns are very general, as I decided to follow a very conservative approach: It is preferable for a tag to remain unresolved due to a lack of propagation than to assign wrong feature values to tokens.

Some tokens are made to always block propagation. In particular, punctuation and tokens from categories that do not bear nominal inflection.

```
livros#mp de ermitas#?p
amarelos#mp e verdes#?p
```
(4.30)

There are, however, some cases where propagation must be blocked even among tokens from nominal categories. For those cases, the featurizer has 9 blocking patterns, which are listed below. I will assume an abuse of notation to describe these patterns, by taking advantage of a notation

similar to the one used for regular expressions.[24]

1. `CN ADJ? PREP PRENOM* CN` $\not\rightarrow$ `ADJ` — A Common Noun that is found within a prepositional phrase (or what may be one) will not propagate to the following Adjective.

   As shown in example (4.31), the invariant Adjective `laranja` (*Eng.: orange*) can agree with `dados` (*Eng.: dice*) or with `madeira` (*Eng.: wood*). As it is not possible to know at this shallow processing stage which is the correct agreement, the propagation of tags from `madeira` to `laranja` is blocked to prevent an erroneous tagging.

   $$\begin{aligned} &\texttt{[dados\#mp de madeira\#fs]}_{\text{NP}}\ \texttt{laranja\#mp} \\ &\texttt{dados\#mp de [madeira\#fs laranja\#fs]}_{\text{NP}} \end{aligned} \tag{4.31}$$

2. `PRENOM+ CN? ADJ? PREP PRENOM* CN` $\not\rightarrow$ `ADJ` — A Common Noun that is found within a modifier phrase (or what may be one) will not propagate to the following Adjective. This is similar to pattern 1, but it allows for the Common Noun to be elided.

   As shown in example (4.32), the invariant Adjective `laranja` can agree with `aqueles` (*Eng.: those*) or with `madeira`.

   $$\begin{aligned} &\texttt{[aqueles\#mp de madeira\#fs]}_{\text{NP}}\ \texttt{laranja\#mp} \\ &\texttt{aqueles\#mp de [madeira\#fs laranja\#fs]}_{\text{NP}} \end{aligned} \tag{4.32}$$

3. `CN ADJ? e|ou PRENOM* CN` $\not\rightarrow$ `ADJ` — A Common Noun that is found within a coordination structure (or what may be one) will not propagate to the following Adjective. This is similar to pattern 1, but with

[24] Capital letters for POS categories, *: zero or more, +: one or more, ?: zero or one, |: alternation, ( ): expression grouping, $\not\rightarrow$: does not propagate to.

`e` or `ou` instead of the Preposition.

When an NP is formed by a coordination of Nouns, a modifying Adjective will bear a plural inflection (Mateus *et al.*, 2003, pp. 330). As exemplified in (4.33), inflection features cannot be propagated from the final element of the coordination — `gata` (*Eng.: cat*) — to the modifying Adjective `pretos` (*Eng.: black*).

$$\texttt{[[o gato\#ms] e [a gata\#fs] pretos\#mp]}_{\text{NP}} \tag{4.33}$$

4. `CN` $\not\rightarrow$ `ADJ ADJ* CN` — A Common Noun will not propagate to an Adjective if that Adjective is followed by a Common Noun (with any number of Adjectives in between).

   This pattern accounts for the possibility of Adjectives occurring in prenominal position: While most Adjectives occur after the Noun they modify, some may occur before.

   As seen in example (4.34), the Adjective `grande` modifies and agrees with the Common Noun `miséria`, not `filhos`.

$$\texttt{provoca [em o filho\#ms]}_{\text{NP}}\ \texttt{[grande\#fs miséria\#fs]}_{\text{NP}} \tag{4.34}$$

5. `V ADV? PREP PRENOM+ ADJ?` $\not\rightarrow$ `CN` — When direct and indirect objects switch positions, the indirect object (IO) will not propagate to the bare NP direct object (DO) that follows it.[25]

   Usually the DO preceds the IO, but their position can be switched,

[25] A bare NP does not have a specifier preceding the head Noun, as in `Comprei flores/CN.`

as shown in (4.35). If the DO is a bare NP, as `flores` in case (b), propagation from the preceding IO must be prevented.

(a) `deu [a o alto#ms]`$_{\text{IO}}$ `[as flores#fp]`$_{\text{DO}}$
(b) `deu [a o alto#ms]`$_{\text{IO}}$ `[flores#fp]`$_{\text{DO}}$ (4.35)

6. `ADJ` $\not\rightarrow$ `ADJ` — An Adjective will not propagate to an immediately following Adjective.

   As with pattern 4, this accounts for the possibility of Adjectives occurring in prenominal position.

   As shown in example (4.36), `outra` modifies `vez` (but not `ricos`) while `grandes` modifies `elogios` (but not `alto`).

```
ficaram ricos#mp [outra#fs vez#fs]
fez a o alto#ms [grandes#mp elogios#mp]
```
(4.36)

7. `CN` $\not\rightarrow$ `ADJ (, ADJ)* e|ou ADJ` — A Common Noun will not propagate to the first element of a coordination of Adjectives.

   As shown in (4.37), there is a "distributive" property where a plural Common Noun will agree with a coordination of singular Adjectives.[26]

`as bolas#fp [laranja#fs , azul#fs e verde#fs]`$_{\text{AP}}$ (4.37)

8. `ORD+ e ORD+` $\not\rightarrow$ `ADJ|CN` — The final Ordinal in a coordination of Ordinals will not propagate to the Adjective or Common Noun that

[26] Cf. (4.37) with `a bola#fs laranja#fs, a bola#fs azul#fs e a bola#fs verde#fs.`

follows it.

As seen in example (4.38), the Common Noun has a plural inflection even though each Ordinal in the coordination is singular.

$$\texttt{[primeiro\#ms e segundo\#ms]}_{\text{ORD}}\ \texttt{lugares\#mp} \tag{4.38}$$

9. `CN` $\not\rightarrow$ `CN` — A Common Noun will not propagate to an immediately following Common Noun.

   As exemplified in (4.39), there are cases where consecutive Common Nouns do not agree. To account for this possibility, propagation is blocked.

```
o preço#ms base#fs
a pergunta#fs número#ms seis
```
(4.39)

This procedure is implemented by specifying the blocking patterns in Flex. If any of the patterns matches the input, it will trigger an action where propagation at the position marked with $\not\rightarrow$ is disabled (the other tokens involved in the pattern are handled normally).

Note that this action necessarily overrides the default action[27] of the lemmatizer since the string matched by any of the patterns seen above is longer than a single token.

Unresolved cases.

By using this featurization algorithm it is nevertheless possible for a token of an invariant word to end up tagged with an underspecified inflection

[27] The default action consists of matching a single nominal token and assigning a lemma to it.

tag. This happens not only due to some propagations being cautiously blocked, as discussed above, but also due to the so-called bare NPs, which do not have a specifier preceding the head Noun, as seen in (4.40.a). It can also occur in non-bare NPs, like (4.40.b), provided that the specifier is itself an invariant word, such as `cada#?s` (*Eng.: each*), which is lexically ambiguous with respect to its Gender feature value.

(a) `Ele detesta ermitas#?p`
(b) `Cada#?s ermita#?s` (4.40)

These remaining underspecified inflection feature tags cannot be accurately resolved at this SP stage.

At this point, one can follow the rationale that it is preferable to refrain from tagging than to tag incorrectly, and not attempt to resolve the underspecified tags without certainty.

The resolution of these tokens can be left to the subsequent phase of syntactic processing which, taking advantage of NP-external agreement,[28] may resolve some of these cases.

Alternatively, if a fully specified tagging is desired or needed at this stage, a naive but simple procedure is to assign the most likely feature value to all underspecified tokens that remain to be tagged or use a stochastic approach to disambiguate them.

These techniques that address the invariant words that remain will be described and evaluated in Section 4.6.4.

[28] Agreement holding between Subject and Verb, Subject and predicative complement in copular constructions with *be*-like verbs, etc.

	Compound word	Parts
(a)	surdo-mudo#ms	ms-ms
	surda-muda#fs	fs-fs
(b)	saia-casaco#ms	fs-ms
	bar-discoteca#ms	ms-fs
(c)	bomba-relógio#fs	fs-ms
	bombas-relógio#fp	fp-ms
(d)	abre-latas#m?	V-fp
	porta-voz#?s	V-fs
(e)	chapéu-de-chuva#ms	ms-PREP-fs
	casas-de-banho#fp	fp-PREP-ms

Table 4.8: Morphosyntactic composition

4.6.3 Remaining Issues

There is still a pending issue that the featurizer does not handle yet: The featurization of hyphenated compound words.

The difficulty in handling these cases stems from the fact that the featurizer is driven by a rule-based approach that uses the termination of words to determine the inflection feature tags. For hyphenated compound words, however, their termination might not be relevant for determining the inflection features.

The troublesome cases are words formed by morphosyntactic composition (Mateus *et al.*, 2003, pp. 978) and syntactic expressions that have been lexicalized.[29]

Table 4.8 shows some examples of hyphenated compound words that are formed by morphosyntactic composition.

The cases under (a) are not problematic. Since the inflection features of any of its parts can be used to tag the whole. This might suggest that

[29] Mateus *et al.* (2003) offer a distinction between morphosyntactic composition and lexicalized expressions. This distinction is not crucial for the task at stake here.

compound words might be analyzed simply by analyzing any of its parts. The remaining cases, however, show that this is not so.

The compound words shown in (b) receive the masculine Gender when their parts do not have the same feature values. In (c), the features of the compound are determined by the features of its first part. In (d) the first part of the compound is a verb form which, naturally, has no nominal inflection. The forms vary between being masculine with underspecified Number feature or having an underspecified Gender.[30]

At this shallow processing stage, it is not always possible for the lemmatizer to decide under which cases a given compound falls. Consequently, such words are explicitly listed and handled through a plain lexical look-up.

The notable exceptions are the cases under (e), where the features of the compound are determined by the features of its first part. These cases are rather easy to detect due to the presence of the Preposition `de` inside the compound.

4.6.4 Evaluation

The evaluation of the featurizer turns out to be less simple than at what first blush might seem, as it depends on (i) what one considers an error and it has also to take into account (ii) the correctness of input, which depends on the accuracy of the POS tagger that delivered the input to the featurizer.

[30]The criteria for the variation is of a semantic nature, viz. being instrumental or agentive. Naturally, this is a distinction that is well outside the capabilities of the shallow processing lemmatizer.

What is an error? As the rule-based featurizer does not necessarily assign a fully specified feature tag to every token, different scores for precision can be taken according to different ways of defining what should be counted as an error.

Mismatch method considers that the featurizer attempts to assign a tag to every token. Consequently, every tagged token that shows a mismatch between the assigned tag and the correct tag is counted as an error. As every nominal token is assigned a tag, there is full recall.

Specified method considers that underspecified tags are assigned when the featurizer abstains from making a decision. Therefore, only fully specified tags are considered when counting for errors. This now allows for a measure of recall, giving the proportion of the nominal tokens to which the featurizer assigned a completely specified tag. In this case, the precision of the featurizer is defined as the proportion of completely specified tags that have been correctly assigned. It is important to note that, under this method, and by the design of the algorithm, precision of 100% can be reached.

The differences between these two evaluation methods are exemplified in Table 4.9.

If the assigned tag matches the correct tag, both methods consider that as being a correct assignment. Also, if the featurizer assigns a fully-specified tag that is different from the correct tag, both methods consider that as being an error.[31]

The difference between the methods can be found when a underspecified tag is assigned. In such cases, the Mismatch method reports an error

[31] Note also that, even if both Gender and Number have been assigned incorrectly, it still counts as a single error.

Feature tag		Error?	
Correct	Assigned	Mismatch	Specified
ms	ms	no	no
ms	fs	yes	yes
ms	?s	yes	n/a

Table 4.9: Evaluation Methods

while the Specified method does not consider that particular occurrence as having been assigned a tag.

Correctness of input. As seen previously in the algorithm outline, the featurizer makes heavy use of the POS information in its input. It is therefore expected that POS tagging errors from the previous stage will have a negative impact on the performance of the featurizer.

To account for this — and measure its actual impact — the evaluation of the featurizer is taken over a correctly POS-tagged corpus but also over an automatically POS-tagged corpus.

Baseline, with lexicon-based featurizer (no propagation).

As always, it is instructive to set a baseline against which the featurizer will be compared.

This is accomplished by using the rule-based featurizer with propagation turned off. In this way, nominal feature values are assigned using only rules and exceptions. This permits to quantify the quality of the exceptions list and will also allow to measure the contribution of the propagation mechanism.

The evaluation results are summarized in Table 4.10.

The first result that is worthy of a remark is the 97.43% precision that is achieved by taking the Specified method over a correctly POS tagged

		Mismatch	Specified
Correct POS	Precision	86.62%	97.43%
	Recall	—	88.90%
Automatic POS	Precision	82.25%	92.51%
	Recall	—	88.91%

Table 4.10: Baseline for the rule-based featurizer

corpus: Since this baseline is for the lexicon-based featurizer and the value only considers fully specified tokens, it is effectively a measure of the quality of the lists of exceptions.

Also worthy of note is the 0.01% increase in recall when running over automatic POS. This (perhaps surprising) increase is a side-effect of wrong POS tags: It is caused by nominal tokens that would be assigned underspecified tags but were wrongly tagged (e.g. as Verbs). As a consequence, the featurizer will skip these tokens and, in the overall recall measure, there will be less tokens with underspecified tags.

The precision value using the Mismatch method is calculated over all nominal tokens — the fully specified and the underspecified. Thus, it is no surprise that this value is equal to the product of the precision and recall for the Specified method.[32]

Featurizer with propagation.

The use of feature propagation produces a clear improvement in precision and recall, as shown in Table 4.11.

Naturally, there is an increase in the recall of the Specified method as fewer tokens are now being tagged with underspecified features.

In addition, the precision of the Specified method also increases. The

[32] $97.43\% \times 88.90\% = 86.62\%$ and $92.51\% \times 88.91\% = 82.25\%$.

		Mismatch	Specified
Correct POS	Precision	94.18%	99.05%
	Recall	—	95.09%
Automatic POS	Precision	88.63%	93.23%
	Recall	—	95.06%

Table 4.11: Evaluation of the rule-based featurizer

reason for this is twofold: (i) the tokens tagged using propagation which previously were assigned underspecified features are always[33] correctly tagged, and (ii) since the propagated feature is preferred to the one assigned through rules,[34] many tokens that were tagged wrongly in the baseline now receive a correct tag through propagation.

The results that were obtained also show how POS tagging errors in the input have a detrimental effect over the resulting precision.

Disregarding the errors caused by erroneous POS tags, the featurizer achieves a very high precision, with a score of 99.05%.[35] Thus, the major cause for the 94.18% score under the Mismatch method is the relatively low recall of 95.09% caused by invariant words that remain with underspecified tags. This suggests that tackling these remaining cases is a good way of improving the overall performance of the featurizer.

Remaining cases (improving recall).

In this section I evaluate four approaches to tagging the invariant words that remained untagged so far: (i) The naive approach of assigning the most likely tag, (ii) relaxing the blocking rules, (iii) using a hybrid featur-

[33]The propagation mechanism, together with the very general blocking patterns, should preclude erroneous propagations.

[34]Cf. the optimization mentioned in Section 4.6.2.

[35]In principle, the only issue that prevents precision from reaching 100.00% is the quality of the lists of exceptions.

izer and (iv) resorting to a second pass with a syntactic analyzer.

These different approaches will be presented next. The evaluation results that were obtained may be seen in Table 4.12. Note that, since I am interested in measuring each method's coverage of invariant words, only the Specified method is applicable.

Naive. This is a straightforward, naive way of handling the invariant words that remain. It consists in assigning to them the most likely Gender feature value tags, viz. masculine for Gender (ca. 52% of the occurrences) and singular for Number (ca. 72% of the occurrences).[36]

Relaxing the blocking rules. By design, the propagation mechanism does not assign wrong tags. This is assured by the patterns that block feature propagation, which are intentionally very general.

This suggests that it might be possible to relax these patterns, allowing for the occurrence of more propagation and, consequently, a higher recall at the expense of some, not too large, decrease in precision.

The "relaxed" rule-based featurizer was obtained by dropping 4 of the 9 patterns used for blocking feature propagation. The chosen rules were the ones anticipated as likely producing the least number of wrong tags by propagation: Adjectives in the right periphery of coordination and of prepositional phrases modifying Nouns.

Hybrid featurizer. This approach uses a stochastic tagger to disambiguate those cases where the rule-based featurizer was unable to assign a fully specified feature tag.

[36]The most likely feature values were calculated over the ca. $51,000$ Adjectives and Common Nouns in the $260,000$ token corpus.

This requires that a stochastic tagger be trained on a corpus with inflection feature tags. Since the original corpus only had POS tags, this could only be done after building the featurizer: The featurizer was used to tag the 260,000 token corpus and the assigned feature inflection tags were manually verified. The resulting corpus was then used to train a stochastic POS tagger that is able to assign POS tags extended with inflection features.

The rule-based featurizer proceeds as usual but, upon an occurrence where it would assign a tag with underspecified features, it assigns the tag given by the stochastic tagger to that occurrence.

Syntactic analysis. When feature propagation is unable to assign features to invariant words, the featurizer abstains from "guessing" the inflection tag in the hope that a subsequent syntactic processing step, taking advantage of NP-external agreement, may resolve these cases.

To evaluate this, I took the output of the rule-based featurizer[37] and examined a sample of 113 tokens that remained with underspecified feature tags.

This analysis revealed that 97 of them could be resolved syntactically, leaving only ca. 16% of the formerly underspecified tags still unresolved.

Extrapolating[38] this result to the whole corpus suggests that a big increase in recall is possible by using a (fully precise and full coverage) syntactic analyzer.

Note, however, that I did not extrapolate this result over the automatically POS-tagged corpus, since it is difficult to predict how the POS errors

[37] Running over a correctly POS tagged corpus, with propagation activated and all blocking patterns active: 99.05% precision (cf. Table 4.11).

[38] As I do not have a syntactic analyzer, I have to extrapolate from a sample. This assumes that the portion of resolvable token will be roughly the same in the whole corpus.

		Naive	Relaxed	Hybrid	Syntactic
Correct POS	Precision	97.01%	98.78%	98.38%	98.85%
	Recall	100.00%	95.91%	100.00%	99.88%
Automatic POS	Precision	91.27%	93.07%	92.62%	—
	Recall	100.00%	95.79%	100.00%	—

Table 4.12: Methods for increasing recall (Specified method)

in the input would affect the parser: A single POS error in a sentence might be enough to invalidate the whole parse of that sentence.

Overall comparison.

The naive approach, not surprisingly, has the lowest precision of all the methods used to tackle the invariant words that the featurizer did not handle. This is likely due to the fact that, although masculine is the most frequent Gender feature, with ca. 52% of the occurrences, it is not much more common than feminine. Consequently, ca. 48% of the invariant words disambiguated by this method will be wrong.

When relaxing the rules that block propagation, there is a slight increase in recall and, as expected, a decrease in precision.

The hybrid method has a very good precision and does not leave any underspecified tags. Its main disadvantage is that it requires having a tagger that is able to assign inflection feature tags, which may not be available.

The method that uses syntactic disambiguation achieves the best precision although it still leaves a small amount of underspecified tags.

Precision and recall, by themselves, are not fully informative: One can get very high precision (but low recall) by doing little, like only assigning tags one is sure about. Conversely, one can achieve full recall (but low

Approach	Correct POS	Automatic POS
Baseline	92.97%	90.67%
Featurizer	97.03%	94.14%
Relaxed	97.32%	94.41%
Naive	98.48%	95.44%
Hybrid	99.18%	96.17%
Syntactic	99.36%	—

Table 4.13: Overall comparison (F-Measure)

precision) through a procedure as simple as assigning the same tag to each token.

When there is a trade-off involving precision and recall, it is useful to look at the F-Measure (Manning and Schütze, 1999, pp. 269), which combines precision and recall into a single value.

The F-measure is defined as follows:

$$f_\alpha = \frac{1}{\alpha\frac{1}{p} + (1-\alpha)\frac{1}{r}}$$

where p is precision, r is recall and α is a factor that weights the contribution of precision and recall to the overall score.

For this evaluation, I will give equal weights to precision and recall (i.e. $\alpha = 0.5$). In this case, the F-Measure simplifies to $f = \frac{2pr}{p+r}$, where p is precision and r is recall.

These results are shown in Table 4.13, ordered by F-Measure.

Every method that was used to raise recall improved on the F-Measure of the rule-based featurizer.

It is interesting to note that the Naive method, despite the name, has a better F-Measure score than the Relaxed method. This is due to the fact that the Relaxed method has a low recall, which counters its better preci-

sion score.

The Syntactic method has the best F-Measure score, but one must keep in mind that this was obtained through extrapolation, since a syntactic parser was not available. In addition, this method is likely to perform much worse when running over automatically POS tagger text. In this regard, the robustness of the Hybrid method is an advantage.

4.7 Nominal Lemmatizer

When addressing the task of nominal lemmatization, it is worth noting that the direct approach of assigning lemmas by means of a mere lexical look-up is far from being the most convenient methodology. Doing this would involve an exhaustive listing of every possible word form and its corresponding lemma, a task that is not practically viable as the lexicon, being open to the inclusion of new words, cannot be definitely complete.

So, we need an approach that is able to seamlessly handle unknown or new words.

4.7.1 Algorithm Outline

To implement the nominal lemmatizer, I build upon the morphological regularities found in word inflection and use a set of transformation rules that revert to the form of the lemma. As it can be seen in Figure 4.5, a single transformation rule can encapsulate a large number of lemma assignments, which would otherwise have to be explicitly listed.

A transformation rule will thus cover most cases for the corresponding termination. There will be, however, exceptions that should be accounted for. For example, the word `porta` (*Eng.: door*) is a feminine common noun

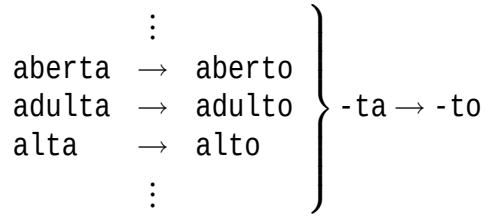


Figure 4.5: A transformation rule

whose lemma is `porta` but applying the rule from Figure 4.5 would give the lemma `porto`. Exceptions such as this have to be collected for each rule.

The basic rationale is thus to gather a set of transformation rules that, depending on the termination of a word, replace that termination by another, and complement this set of rules with a list of exceptions (vd. Annex G on page 159 for a listing of rules).

Neologisms are expected to comply with the regular morphology and are accounted for by the rules.

Transformation rules.

Transformation rules are replacement rules used to "undo" the changes caused by the inflection process. So, for instance, if words ending in `-to` are typically inflected into words ending in `-ta` to obtain the feminine singular form, the reverse transformation should be present in the lemmatization rules.

For example, doing this for all four possible Gender and Number combinations one can obtain for the words ending in `-to` leads to the set of rules exemplified in Figure 4.6.

These can be easily implemented by creating a procedure that, for each word, scans the set of rules for a match, i.e. a rule whose left-hand side

`-to`	(*masculine*, *singular*)		
`-tos`	(*masculine*, *plural*)	→	`-to`
`-ta`	(*feminine*, *singular*)	→	`-to`
`-tas`	(*feminine*, *plural*)	→	`-to`

Figure 4.6: A set of rules for `-to`

matches the termination of the word. Upon finding a rule, the corresponding termination of the word is replaced by the string on the right-hand side of the rule.

Exceptions.

To collect exceptions I resort to machine-readable dictionaries (MRD) that allow searching for words on the basis of their termination. Given that dictionaries only include lemmas, and never inflected forms, it is possible to collect the needed exceptions simply by searching for words with terminations matching the termination of inflected words. For instance, for the rule in Figure 4.5, I would search for words ending in `-ta`.

These collected exceptions are entered into an internal word list of the lemmatizer and each is coupled with its specific transformation rule.

Exceptions whose lemma matches the word form are associated to a "dummy" transformation that does not change the word form. For example, `porta` is associated with a `-ta` → `-ta` transformation.

The size of the different lists of exceptions may reach thousands of entries. Although this might seem at first blush a large size, one should bear in mind the following points:

- The number of words covered by a transformation rule is much larger than the number of exceptions one needs to collect for that rule.
- New words that enter the lexicon tend to follow the general rule for

inflection and, consequently, for lemmatization. This also implies that the list of exceptions is considerably stable and needs to be less frequently updated than an exhaustive look-up table of word forms and their lemmas.

4.7.2 Lexicon minimization

By using the algorithm outlined above, one drastically reduces the size of the lexicon that is necessary, since only exceptions have to be explicitly listed. Nevertheless, it is desirable to reduce the lexicon as much as possible. The techniques for doing this are addressed in this section.

Rule hierarchy.

The number of exceptions one needs to collect for a given rule can be decreased as extra, more specific rules can be used to capture regularities found within the exceptions themselves.

For instance, terminations matching `-ia` are usually transformed into `-io` (feminine into masculine). When collecting the exceptions for this rule one can find, among others, the whole family of words ending in the Greek suffix `-fobia` (*Eng.: -phobia*), whose lemma matches the word form. So, instead of listing them as exceptions for the `-ia` → `-io` rule, one can simply add the new rule `-fobia` → `-fobia` to handle all those cases.

In terms of the implementation of the algorithm, note that, to allow for this, the choice of which transformation rule to apply must use the longest match criteria, so that the longer, more specific termination will necessarily be chosen.

Recursive single-step rules.

The set of lemmatization rules shown in Figure 4.6 transforms any of the various inflected word forms from the `-to` "family" directly into the corresponding lemma.

By doing this transformation directly, special care must be taken when collecting the exceptions for the `-tas` → `-to` rule as a dictionary search for words ending in `-tas` will not provide all the necessary exceptions.

This happens because these exceptions must also include the words that are exceptions to the `-ta` → `-to` rule but when inflected for plural.

For example, the word `porta`, as seen before, is one of the exceptions to the `-ta` → `-to` rule. Accordingly, its plural form, `portas` (*Eng.: doors*), is an exception to the `-tas` → `-to` rule, but it is a word form that will not be listed in a dictionary.

Obtaining the exceptions in plural form is rather easy as one needs only apply a simple transformation to every exception in singular form. There is, however, an easier way of handling these cases, namely through the use of recursive single-step lemmatization rules, which is an approach that does not require extending the exceptions list.

Single-step rules. Single-step rules trigger transformations that only affect a single feature (Gender or Number). For instance, the last rule from the set shown in Figure 4.6 (viz. `-tas` → `-to`) is not a single-step rule, since it transforms a feminine plural form into a masculine singular form in a single transformation step.

The single-step rules seen in Figure 4.7 are similar to the set of rules in Figure 4.6, differing only in the rule for feminine plural word forms, which now transforms these words into their feminine singular form.

`-to`	(*masculine*, *singular*)		
`-tos`	(*masculine*, *plural*)	→	`-to`
`-ta`	(*feminine*, *singular*)	→	`-to`
`-tas`	(*feminine*, *plural*)	→	`-ta`

Figure 4.7: Single-step rules for `-to`

Recursive application of rules. Naturally, a single-step rule will not produce a lemma when the word to be lemmatized is in the feminine plural form.

Recursive lemmatization rules are repeatedly applied, transforming a word into another, until an exception has been found or until there is no rule that can be applied.

For example, `adultas` (*Eng.: adults*, feminine plural) would firstly be transformed into `adulta` (*Eng.: adult*, feminine singular) and, in a second step, into `adulto` (*Eng.: adult*, masculine singular) while `portas` would be transformed into `porta`, matching the exception.

By using single-step recursive rules, it is not necessary to extend the exceptions list with inflected forms of exceptions. In addition, it is straightforward to extend the algorithm to apply such rules: It suffices running the transformation procedure on its own output until no rule can be applied or until an exception is found.

Non-inflectional affixes.

Besides its various inflected forms, a word can also have non-inflectional affixes, greatly increasing the number of possible forms that the lemmatizer must handle.

In this section, I discuss how to handle non-inflectional prefixes and suffixes and how to resolve the complexity that arises when both occur

simultaneously in the same word form.

Prefixes. Prefixed words raise a difficulty when dealing with the search for exceptions to a transformation rule.

Taking again `porta` as an example of an exception, every word formed by combining it with a prefix is also an exception to the `-ta` → `-to` rule, like `anteporta`, `autoporta`, `superporta`, etc. This entails that, in addition to `porta`, all words formed by prefixing `porta` must also be included in the exceptions to the `-ta` → `-to` rule.

Adding all such prefixed words to the list of exceptions is not an appropriate approach, as the number of prefixes, even though not unlimited, is still very large and, more important, prefixes can be combined, as in `autosuperporta`.[39]

As a better solution to this problem, the algorithm is designed to temporarily remove prefixes from words when obtaining the lemma. After getting the lemma, the prefixes that were removed are returned back to their original place.

This requires a list of possible prefixes. When the starting characters of a word match one of the listed prefixes, these characters are temporarily removed. The process is repeated until it is not possible to remove any further prefixes.

It is important to note that, before removing any prefix, the word form is checked against the exceptions list. This is done to account for word such as `antena` (*Eng.: antenna*) which, although beginning with a sequence of characters matching a listed prefix (cf. `ante-`), it is not a prefixed word.

In Figure 4.8, this mechanism is illustrated by means of the processing

[39] Even though some of these prefixed words might be unusual, they are perfectly valid from a purely morphological point of view.

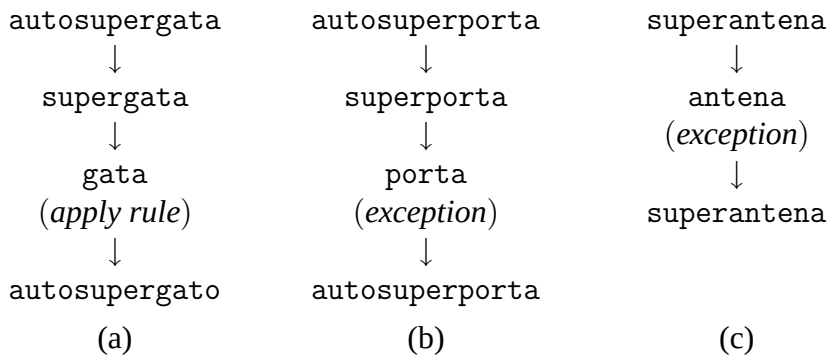


Figure 4.8: Prefix stripping

diagram of the following words:

- `gata` (*Eng.: cat*, feminine singular), a Common Noun which should be lemmatized into the masculine singular form `gato`. After stripping `auto-` and `super-`, one gets `gata`, which has no more prefixes to get stripped. The `-ta` → `-to` rule can then be applied, giving the result `gato`.

- `porta`, an exception, which should be lemmatized into `porta`. After stripping both prefixes, one gets `porta`, which is one of the listed exceptions.

- `antena`, an exception, which should be lemmatized into `antena`. This is similar to the previous case, but it illustrates a case where a possible prefix (`ante-`) was not removed, as the word form was listed in the exceptions list.

Degree suffixes. Degree suffixes (for Diminutive and Superlative) seem, at first, to be easily handled by the same mechanism used for inflection suffixes but specific difficulties should be taken into account.

For instance, `gatinho` (*Eng.: [small] cat*) is easily lemmatized into `gato` by a `-inho` → `-o` rule. Also, any exceptions to this rule, like `vizinho` (*Eng.: neighbor*) are easily collected by using MRDs. In addition, as seen before, to minimize the number of exceptions one needs to collect, recursive single-step rules can be used, such as `-inha` → `-inho`.

However, there are some situations that cannot be as easily handled by this mechanism. For the sake of concreteness, I will take the words `vizinha` (*Eng.: [feminine] neighbor*) and `portinha` (*Eng.: [small] door*) as an examples of such situations.

- Having the `-inha` → `-inho` rule described above avoids having to list `vizinha` explicitly as an exception. But, by using this rule, the word `portinha` would be transformed into `portinho` and, in a second step, lemmatized as `porto`. To prevent this from happening, one has to list `portinha` explicitly as an exception to the `-inha` → `-inho` rule.

- As an alternative, one could instead use `-inha` → `-a` as a default rule, which would correctly lemmatize `portinha`. However, in this case, we would need to list `vizinha` as an exception, to prevent it from being lemmatized into `viza`.

Accordingly, regardless of which rule is chosen to be the default one, one has to list extra exceptions. To avoid this, a possible approach is to allow for a rule to branch out into several possible transformations, thus giving rise to a branching search for the solution.

Branching search.

To allow branching search, the rules are extended to allow for various alternative transformations. Each alternative opens up a new search branch.

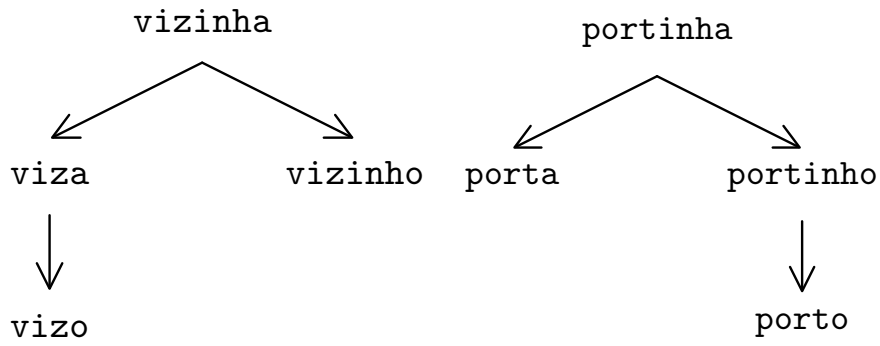


Figure 4.9: Branching search

The lemmatization proceeds in parallel through each branch until no further rules are applicable or until an exception has been found.

The heuristic for the choice of lemma takes the leaf node found at the lowest search depth (i.e. in the fewest steps).

For instance, to handle the problem described above, found when dealing with `portinha` and `vizinha`, a `-inha` → `-a`, `-inho` rule is created. For words ending in `-inha`, this rule creates two search branches. For the first branch, the `-inha` → `-a` is applied, while the search in the second branch proceeds under the `-inha` → `-inho` transformation. The result found at the lowest search depth (`vizinho` for `vizinha` and `porta` for `portinha`) is taken as being the lemma. Both these search trees can be seen in Figure 4.9.

The branching search has the added advantage of allowing to handle words that combine multiple prefixes and suffixes.

When several non-inflectional affixes occur in a single word, one must follow all possible paths of affix removal/transformation to ensure that an exception is not overlooked.

As an example, take the lemmatization of `anteninha` (*Eng.: [small] antenna*), illustrated in Figure 4.10.

Both `ante-` and `-inha` are possible affixes. In the first step, three search

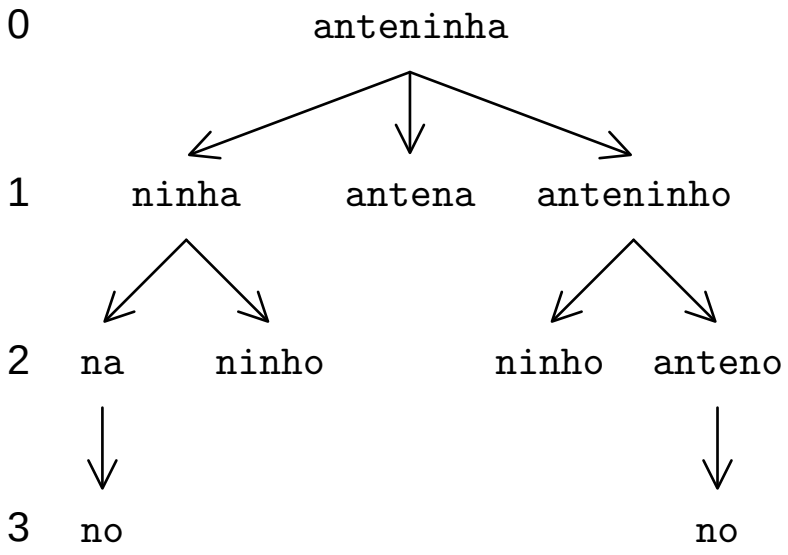


Figure 4.10: Search tree for `anteninha`

branches are opened. The first branch (`ninha`) corresponds to the removal of the `ante-` prefix, while the two other branches (`antena` and `anteninho`) have the result of applying the transformations in the `-inha` → `-a`, `-inho` rule. The lemma is the leaf node with the lowest search depth, `antena`.

The search under several branches seems to lead to a great performance penalty, but one must bear in mind that only a few words have non-inflectional affixes, and most of those have only one, in which case there is no branching at all. So, in practice, the branching search does not incur in a damaging performance penalty while allowing for arbitrarily complex affixed words.

4.7.3 Ambiguity

As it happens with most NLP tasks, there are ambiguous cases that must be dealt with.

Sense dependency. As mentioned in Section 2.6.2, there are cases where the lemma may be conditioned by the sense of the token at stake, as seen in example (4.41).

(a) `ética/ADJ` *(Eng.: ethical)* → `ético`
(b) `ética/CN` *(Eng.: ethics)* → `ética`
(c) `copas/CN` *(Eng.: cupboards)* → `copa`
(d) `copas/CN` *(Eng.: hearts suit)* → `copas`

(4.41)

Sometimes, the sense dependency can be resolved just by knowing the POS of the token at stake, as in cases (a) and (b). Since the lemmatizer is run after the POS tagger, these cases can be handled by allowing for exceptions that depend on POS. For instance, a `-ca` → `-co` rule can lemmatize the token `ética` when it occurs as an Adjective. The other sense of the word can be handled by listing `ética` as an exception that falls under the alternative `-ca` → `-ca` rule but only when the POS of the corresponding token is Common Noun.

The remaining cases — such as (c) and (d) — cannot be resolved without a previous word sense disambiguation (WSD) stage. Nonetheless, while a definite lemma cannot be assigned at this stage, ambiguity can be circumscribed by assigning all possible lemmas to such forms, e.g. assigning `copa` and `copas` to the word `copas`.

Word ending in `-e`. Words ending in `-e` pose a problem to the lemmatization task since these words do not have a preferential Gender feature (Mateus *et al.*, 2003, pp. 923) and their diminutive form has the same terminations as the diminutive for words ending in `-a` (viz. `-inha` or `-ita`) and `-o` (viz. `-inho` or `-ito`).

Word	Diminutive	Possible lemmas	Lemma
gato#ms	gatinho	gato or gate	gato
pente#ms	pentinho	pento or pente	pente
mesa#fs	mesinha	mesa or mese	mesa
parede#fs	paredinha	pareda or parede	parede

Table 4.14: Ambiguity of words ending in -e

Table 4.14 shows several words and some of their diminutive forms. When lemmatizing any of the diminutive forms, there are always two plausible transformations. For instance, for the -inho termination both -inho → -o and -inho → -e are plausible.

These problematic words are handled as exceptions — e.g. pentinho is listed as an exception to a -inho → -o rule, and associated with the -inho → -e transformation.

Affix ambiguity. We can take ambiguity to its utmost consequences and consider that every possible transformation introduces ambiguity.

For instance, we know that pentinho is the diminutive of pente, but there is no morphological reason to discard the lemma pento. In fact, there is even no reason to discard pentinho as the lemma. We only do so because we have learned that pento and pentinho are not "attested words."

To avoid this rampant ambiguity the lemmatizer follows a "least surprise" principle, according to which a word is only an exception if explicitly stated. Otherwise, the existing rules are applied.

There is, however, a remaining sub-case that is particularly problematic: Words that can also be considered as affixed exceptions.

An example is the word profunda, which can be seen as the feminine form of profundo or as the prefix pro attached to the word funda. The problematic issue arises due to the fact that funda is an exception to the

`-da` → `-do` rule and should be lemmatized to `funda`. As a consequence, `profunda` has two possible lemmas: `profundo` and `profunda`.

This kind of ambiguity is, in fact, another aspect of sense dependency and it thus require a WSD stage to resolve. Accordingly, such words are to receive all possible lemmas.

4.7.4 Remaining Issues

Similarly to what happens with Nominal Featurization, the main issue that remains to be addressed is the lemmatization of hyphenated compound words formed by morphosyntactic composition (Mateus *et al.*, 2003, pp. 978). For these words, replacing their termination might not be enough to obtain the lemma. The problematic cases, however, are not exactly the same that affected Nominal Featurization.

In many cases, the compound word can be lemmatized by lemmatizing each of its parts separately, as seen in (4.42).

```
surdos-mudos        → surdo-mudo
autora-compositora  → autor-compositor
aluna-modelo        → aluno-modelo
bomba-relógio       → bomba-relógio
```
(4.42)

However, there are some words formed by morphosyntactic composition where this approach does not work. The problematic cases are those where the hyphenated compound word is formed by a verb form — in the third person, singular, *Presente do Indicativo* tense — followed by a Common Noun or, rarely, an Adjective.[40] Some examples are shown in (4.43).

[40] Mateus *et al.* (2003) calls these cases “estruturas de reanálise.”

(a) `abre-latas` → `abre-latas`
(b) `conta-gotas` → `conta-gotas`
(c) `porta-vozes` → `porta-voz`
(d) `lava-louças` → `lava-louça` (4.43)

The first difficulty is in determining if a given compound falls under one of these cases since determining if the first part of the word is a verb form is not feasible for the shallow processing lemmatizer.[41]

Moreover, even if that were possible, the cases shown in (4.43) are not all handled in the same way. These forms have two types: (i) They are masculine with an underspecified Number feature — such as cases (a) and (b) — in which case the lemma matches the word form or (ii) they have an underspecified Gender — such as cases (c) and (d) — in which case plural forms are lemmatized into the singular form.

However, as mentioned in Section 4.6.3, the criteria for classification is of a semantic nature. Naturally, this is a distinction that well outside the capabilities of the shallow processing lemmatizer.

Consequently, to assign a lemma to these cases the lemmatizer has to use a plain lexical look-up.

4.7.5 Evaluation

In order to implement the algorithm, a list of 126 transformation rules was necessary. The list of exceptions to these rules amounts to 9,614 entries. Prefix removal is done resorting to a list of 130 prefixes.

The lemmatizer was evaluated over the 50,637 Adjectives and Common Nouns present in the vocabulary of the 260,000 token corpus. Note

[41] E.g. `conta`, `porta` and `lava` are verb forms but also Common Nouns.

Recall	Precision	F-Measure
99.60%	97.87%	98.73%

Table 4.15: Lemmatizer evaluation

that since the lemmatizer does not use a window of context around the target token, evaluating over a list of target tokens produces the same result as evaluating over the whole corpus. Note also that by doing this, I am evaluating the lemmatizer *stricto sensu*, i.e. the lemmatizer is run over a correctly tagged text where there are no errors caused by previous stages.

In the evaluation list there are 203 tokens that are lexically ambiguous with respect to lemmatization. As discussed before, these tokens cannot be lemmatized without a previous WSD step. Therefore, this figure helps to suggest an upper-bound of 99.60% to the recall of a lemmatizer based on shallow processing. On the remaining $50,434$ tokens, there were $1,072$ lemmatization errors, yielding a precision of 97.87%. The F-measure, with equal weights for precision and recall, yields 98.73%. These results are summarized in Table 4.15.

The errors are caused by words that are missing from the exceptions list and, therefore, receive the wrong lemma. Similarly to what happens with Nominal Featurization, the only issue that may prevent the lemmatizer from reaching full precision is the quality of the lists of exceptions.

5

Conclusions

The work in this dissertation lead me through several of the initial stages in NLP. After distilling everything, one thing becomes clear: The major difficulties were invariably caused by ambiguity.

I found shallow processing to be an efficient way of dealing with the problems caused by ambiguity: As a consequence of only using local information and specialized approaches — be them rule-based, stochastic or a combination of both, — it can resolve ambiguity without needing complex syntactic and semantic processing, and still deliver useful results.

5.1 Major contributions

The major contributions of this dissertation are twofold: (i) The identification of key issues for shallow processing (SP) and (ii) the design of state-of-the-art SP algorithms and the evaluation of the implementation of those algorithms.

5.1.1 Identification of key issues

This contribution is the identification of the key issues found in each task, in particular those that are specific to Portuguese. These are listed below.

- Sentence segmentation
  - Ambiguity and ambivalence of the period (full stop) symbol
  - Segmentation of the narrative of dialogs
- Tokenization
  - Token-ambiguous strings
- POS-tagging
  - Tagset design
  - POS-ambiguous tokens
- Nominal featurization
  - Invariant words
  - Hyphenated compound words
- Nominal lemmatization
  - Lemmas that depend on the sense of the word
  - Minimization of the lexicon
  - Hyphenated compound words

5.1.2 Algorithm design and evaluation

Each of the tasks was addressed through a state-of-the-art shallow processing algorithm.

Sentence segmentation

The algorithm is a finite-state automaton tuned to Portuguese orthographic conventions. Its implementation achieves 99.94% recall and 99.3% precision.

Tokenization

The algorithm is a finite-state automaton. The token-ambiguous strings are handled through a two-stage tokenization approach, where the POS tagging step is interpolated into tokenization to disambiguate the problematic strings. The implementation achieves 99.44% precision in handling the token-ambiguous strings.

POS tagging

I used several third-party, well-known tools to train ML taggers. Each resulting tagger has state-of-the-art performance, with precision varying between 96.87% and 97.09%. To the best of my knowledge, these are currently the best results for Portuguese POS tagging.

Nominal featurization

The featurization algorithm uses a rule-based approach that, based on the termination of a word, assigns inflection feature values to it.

To handle invariant words, a feature propagation mechanism was devised which takes advantage of agreement inside NPs. The propagation mechanism had to be supplemented with a set of blocking patterns that prevent propagation in some cases.

When running over a correctly POS tagged text and using the Hybrid approach to improve recall (cf. Table 4.12 on page 116), this algorithm achieves 98.38% precision at full recall (yielding a F-Measure score of 99.18%).

Nominal lemmatization

The lemmatization algorithm uses a rule-based approach that replaces the termination of words by other strings.

The lemmatization algorithm uses several methods to minimize the lexicon — formed by exceptions to the transformation rules — that is required, viz. (i) the use of hierarchical rules, (ii) applying single-step rules recursively, (iii) stripping non-inflectional affixes and (iv) using a branching search.

There are several cases where a lemma depends on the sense of the token at stake. Some of those cases can be handled by knowing the POS of that token. Since the lemmatizer runs after the POS tagger, these cases are easily resolvable. For the remaining ambiguous cases the lemmatizer assigns all possible lemmas, since they cannot be resolved without a previous WSD step.

This algorithm achieves 99.60% recall and 97.87% precision.

5.2 Future work and research

In future work, a natural path of research consists in adding new tools to the NLP pipeline. For instance, other shallow processing tools are currently being built upon the ones described in this dissertation, viz. a verbal morphological analyzer and a named-entity recognizer.

	Type	Token count		Total
Spoken	Informal	52.1%	165,838	318,593
	Formal	20.8%	66,274	
	Media	19.5%	62,116	
	Phone calls	7.6%	24,365	
Written	Newspaper	60.8%	432,232	711,135
	Book	25.7%	182,890	
	Magazine	8.5%	60,482	
	Miscellaneous	5.0%	35,531	
				1,029,728

Table 5.1: Composition of the CINTIL Corpus

The tools are also being used to extend the corpus that has been used throughout this dissertation.

Under the TagShare project, the initial 260,000 token corpus is being extended to 1 million tokens, with the tools providing a bootstrapping annotation that is reviewed by linguistically trained human annotators.

This 1 million token corpus is composed of several genres, including ca. one third of transcribed spoken materials. A detailed breakdown of the composition of the corpus may be found in Table 5.1.

But even without considering new tools, one can find possible improvements and research opportunities in the current tools. Some of these are itemized below.

- Sentence Segmentation
  - Now that there exists a segmented corpus to train on, one can experiment with some of the machine-learning techniques mentioned in Section 3.1 (page 45).
  - Automatic learning of new abbreviations: Words ending in a period that are not followed by an initiator symbol are collected

and, after manual verification, added to the list of known abbreviations.

  - Experiment with using interpolated POS tagging and portmanteau tags (the technique used for the two-stage tokenization approach) to resolve strings that are ambiguous between being an abbreviation and being a word followed by period (cf. Table 4.2 on page 70).

- Tokenization

  - Add features mentioned in Section 4.4.4 (page 82), such as the reattachment of words that have been split over two lines due to word wrap.

- POS Tagging

  - After the new, 1 million token corpus is ready, the POS taggers can be trained again to take advantage of the larger corpus.
  - Experiment with combining different POS taggers in view of possibly improving accuracy, cf. (Brill and Wu, 1998) and (van Halteren *et al.*, 2001).
  - Experiment with a rule-based "corrector." This tool would be run after the POS tagger to correct some common errors.
  - Experiment with using "clues" from the tokenization task. For instance, the tokenizer can assign POS tags with full certainty when expanding contractions and when detaching clitics from verbs. The POS tagger would not need to assign these tags and might be able to use them as "look-ahead clues" to guide and constrain tagging.

  - Experiment with *k*-best tagging, i.e. assigning more than one tag in certain situations in order to increase the probability of the correct tag being assigned (cf., for instance, (Brill, 1994, pp. 5)).

- Nominal Featurization

  - Extend the list of exceptions. At the moment, there are still some exceptions missing (as evidenced by the 97.43% precision of the lexicon-based featurizer seen in Table 4.10, on page 112).
  - Experiment with an exception "learner" tool. The purpose of this tool would be to help extend the list of exceptions by automatically collecting mismatches that are found in the text (i.e. words that, through propagation, are assigned features that conflict with those they receive through the lexicon). These mismatches would be manually reviewed to choose which — if any — should be incorporated into the list of exceptions.
  - Handle hyphenated compound words (cf. Section 4.6.3). It may be possible to create some heuristics to handle some of these cases, instead of having to list them explicitly in the lexicon.
  - Experiment with a reverse propagation step. This mechanism would propagate feature tags from right to left. Its purpose would be to cover a few more underspecified cases, like the one in the NP `Cada#?s humilde#?s pastor#ms` where the feature values from `pastor` could be used to specify the invariant words that precede it.

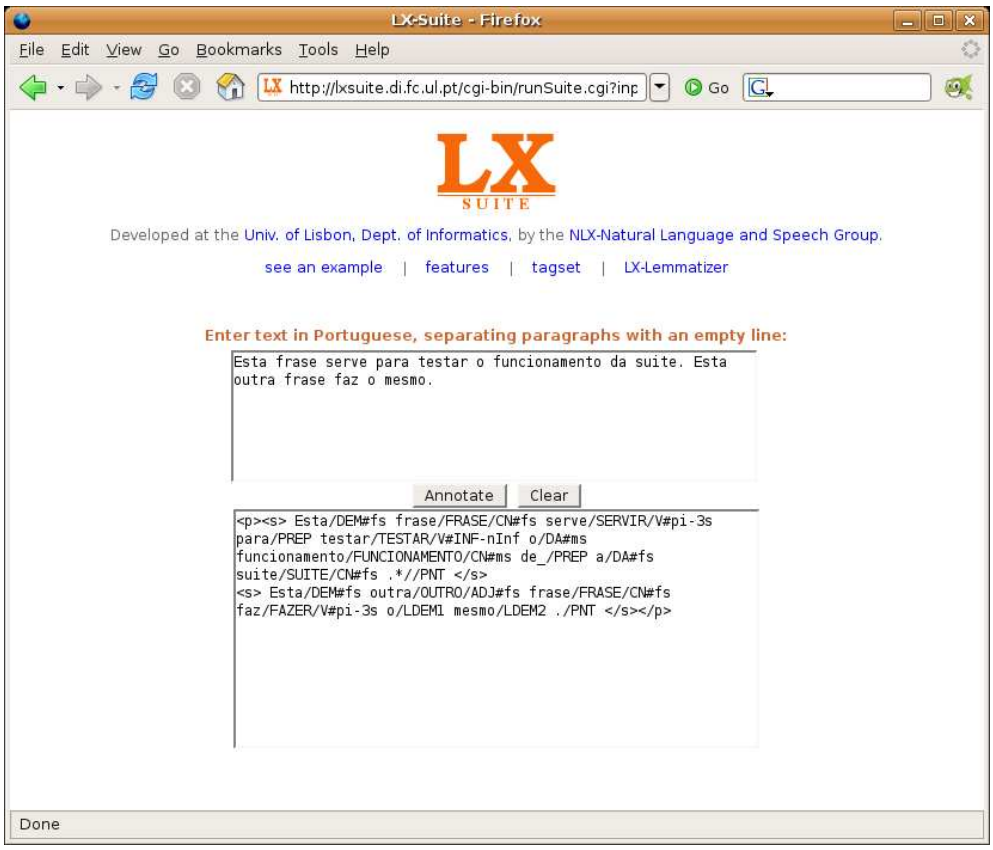
- Nominal Lemmatization

  - Extend the list of exceptions, including the ambiguous cases

mentioned in Section 4.7.3.

- Use transformation rules that depend not only on the termination of a given token but also on its POS.
- Handle hyphenated compound words (cf. Section 4.7.4).

A LX-Suite Screenshot



Some additional notes:

- The on-line demo may be found at: `http://lxsuite.di.fc.ul.pt`
- LX-Suite includes more tools than the ones described in this dissertation. In particular, it includes a verbal morphological analyzer, which assigns inflection features and lemmas to Verbs.

B Corpus Sample

```
A/DA#fs sessão/SESSÃO/CN#fs de_/PREP o/DA#ms dia/DIA/CN#ms
22/DGT começou/V com/PREP poemas/POEMA/CN#mp de_/PREP o/DA#ms
austríaco/AUSTRÍACO/CN#ms Hans/PNM Raimund/PNM ,*//PNT
lidos/LIDO/PPA#mp por_/PREP o/DA#ms próprio/PRÓPRIO/ADJ#ms
e/CJ por/PREP João/PNM Barrento/PNM ,*//PNT que/REL
traduzira/V parte/PARTE/CN#fs de_/PREP eles/PRS#mp3 para/PREP
o/DA#ms número/NÚMERO/CN#ms 6/DGT de_/PREP a/DA#fs
revista/REVISTA/CN#fs \*'/PNT Limiar/PNM '*//PNT \*(/PNT
1995/DGT )*//PNT e/CJ mais/ADV alguns/QNT#mp para/PREP a/DA#fs
antologia/ANTOLOGIA/CN#fs que/REL acompanhou/V o/DA#ms
Encontro/PNM .*//PNT
```

C List of Abbreviations

Latin forms	
cf.	*confer*
e.g.	*exempli gratia*
i.e.	*id est*
vs.	*versus*

Part of addresses			
Av.	Avenida	Estr.	Estrada
Lrg.	Largo	Lt.	Lote
Pr.	Praça	Rot.	Rotunda
Trav.	Travessa		

Months			
Jan.	Janeiro	Fev.	Fevereiro
Mar.†	Março	Abr.	Abril
Mai.	Maio	Jun.	Junho
Jul.	Julho	Ago.	Agosto
Set.†	Setembro	Out.	Outubro
Nov.	Novembro	Dez.†	Dezembro

Weekdays	
Seg.	Segunda
Ter.†	Terça
Qua.	Quarta
Qui.	Quinta
Sex.	Sexta
Sáb.	Sábado
Dom.†	Domingo

Sectioning and numbering					
Cap.	Capítulo	Ex.	Exemplo	Fig.	Figura
Fl.	Folha	Nº.	Número	No.	Número
Num.	Número	Pg.	Página	Pág.	Página
Pgs.	Páginas	Págs.	Páginas	Sec.	Secção
Séc.	Século	Tab.	Tabela	Vol.	Volume

Social titles and professions					
Sr.	Senhor	Snr.	Senhor	Srª.	Senhora
Snrª.	Senhora	Srs.	Senhores	Pres.	Presidente
Sec.	Secretário	Secº.	Secretário	Secª.	Secretária
Dir.	Director	Dirª.	Directora	Ab.	Abade
Fr.	Frade	Pe.	Padre	Prof.	Professor
Profª.	Professora	Dr.	Doutor	Drª.	Doutora
Arq.	Arquitecto	Arqº.	Arquitecto	Arqu.	Arquitecto
Arqª.	Arquitecta	Eng.	Engenheiro	Engº.	Engenheiro
Engª.	Engenheira				

Honorifics			
Em.	Eminência	Emª.	Eminência
Exº.	Excelentíssimo	Exo.	Excelentíssimo
Exmº.	Excelentíssimo	Exmo.	Excelentíssimo
Exª.	Excelentíssima	Exmª.	Excelentíssima
Exs.	Excelentíssimos	Exos.	Excelentíssimos
Exmos.	Excelentíssimos	Exas.	Excelentíssimas
Exmas.	Excelentíssimas	Excª.	Excelência
Exciª.	Excelência	Excia.	Excelência
Excas.	Excelências	Excias.	Excelências
Rev.	Reverência	Revª.	Reverência

Some additional notes:

- The † indicates an ambiguous abbreviation (cf. Table 4.2).
- Single letters followed by period are considered abbreviations.
- Honorifics can be combined with social titles, e.g. "Exº. Sr. Dr."
- The feminine indicator (ª) can be replaced by the letter "a."

D List of Contractions

a + X					
à	→	a a	às	→	a as
ao	→	a o	aos	→	a os
aonde	→	a onde	àquela	→	a aquela
àquelas	→	a aquelas	àquele	→	a aquele
àqueles	→	a aqueles			
com + X					
comigo	→	com mim	contigo	→	com ti
consigo†	→	com si	connosco	→	com nós
convosco	→	com vós			
de + X					
da	→	de a	dacolá	→	de acolá
daí	→	de aí	dalém	→	de além
dalgum	→	de algum	dalguma	→	de alguma
dalgumas	→	de algumas	dalguns	→	de alguns
dali	→	de ali	daquela	→	de aquela
daquelas	→	de aquelas	daquele	→	de aquele
daqueles	→	de aqueles	daquém	→	de aquém
daqui	→	de aqui	daquilo	→	de aquilo
das	→	de as	dela	→	de ela
delas	→	de elas	dele	→	de ele
deles	→	de eles	dentre	→	de entre
dessa	→	de essa	dessas	→	de essas
desse†	→	de esse	desses†	→	de esses

Continued on next page…

Continued from previous page

desta	→	de esta	destas	→	de estas
deste†	→	de este	destes†	→	de estes
disso	→	de isso	disto	→	de isto
do	→	de o	donde	→	de onde
dos	→	de os	doutra	→	de outra
doutras	→	de outras	doutrem	→	de outrem
doutro	→	de outro	doutros	→	de outros
dum	→	de um	duma	→	de uma
dumas	→	de umas	duns	→	de uns
em + X					
na†	→	em a	nalgum	→	em algum
nalguma	→	em alguma	nalgumas	→	em algumas
nalguns	→	em alguns	naquela	→	em aquela
naquelas	→	em aquelas	naquele	→	em aquele
naqueles	→	em aqueles	naquilo	→	em aquilo
nas†	→	em as	nela	→	em ela
nelas	→	em elas	nele†	→	em ele
neles	→	em eles	nessa	→	em essa
nessas	→	em essas	nesse	→	em esse
nesses	→	em esses	nesta	→	em esta
nestas	→	em estas	neste	→	em este
nestes	→	em estes	nisso	→	em isso
nisto	→	em isto	no†	→	em o
nos†	→	em os	noutra	→	em outra
noutras	→	em outras	noutrem	→	em outrem

Continued on next page...

Continued from previous page

noutro	→	em outro	noutros	→	em outros
num	→	em um	numa	→	em uma
numas	→	em umas	nuns	→	em uns
para + X					
prà	→	para a	pr′à	→	para a
pràs	→	para as	pr′às	→	para as
prò	→	para o	pr′ò	→	para o
pròs	→	para os	pr′òs	→	para os
por + X					
pela†	→	por a	pelas†	→	por as
pelo†	→	por o	pelos	→	por os

†: token-ambiguous string

E Base POS Tagset

Tag	Category	Examples
ADJ	Adjectives	bom, brilhante, eficaz, ···
ADV	Adverbs	hoje, já, sim, felizmente, ···
CARD	Cardinal Numerals	zero, dez, cem, mil, ···
CJ	Conjunctions	e, ou, tal como, ···
CL	Clitics	o, lhe, se, ···
CN	Common Nouns	cidade, gatos, ideia, ···
DA	Definite Articles	o, os, ···
DEM	Demonstratives	este, esses, aquele, ···
DFR	Denominators of Fractions	meio, terço, décimo, %, ···
DGTR	Roman Numerals	V, LX, MMIII, MCMXCIX, ···
DGT	Digits (arabic numerals)	0, 1, 42, 12345, 67890, ···
DM	Discourse Marker*	olá, pá, ···
EADR	Electronic Addresses	http://www.di.fc.ul.pt, ···
EL	Extra-linguistic*	*coughing*, *sneezing*, *etc.*
EMP	Emphasis*	[eu] cá, [tu] lá, ···
EOE	End of Enumeration	etc.
EXC	Exclamative	Que, Quantas, ···
FRG	Fragment*	gra[dual], manei[ra], ···
GER	Gerunds†	sendo, afirmando, vivendo, ···
GERAUX	Gerund "ter" or "haver"‡	tendo, havendo
IA	Indefinite Articles	uns, umas, ···
IND	Indefinites	tudo, alguém, ninguém, ···
INF	Infinitive†	ser, afirmar, viver, ···
INFAUX	Infinitive "ter" or "haver"‡	ter, terem, haver, ···

Continued on next page...

Continued from previous page

Tag	**Category**	**Examples**
INT	Interrogatives	quem, como, quando, ···
ITJ	Interjection	bolas, caramba, ···
LTR	Letters	a, b, c, ···
MGT	Magnitude Classes	dezena, dúzia, resma, ···
MTH	Months	Janeiro, Fevereiro, ···
NP	Noun Phrases	idem, ···
ORD	Ordinals	primeiro, segundo, último, ···
PADR	Part of Address	Rua, av., rot., ···
PL	Para-linguistic*	ah, hã, hmmm, oh, ···
PNM	Part of Name	Lisboa, António, João, ···
PNT	Punctuation Marks	., ?, (, ···
POSS	Possessives	meu, teu, seu, ···
PP	Prepositional Phrases	algures, ···
PPA	Past Participles†	sido, afirmados, vivida, ···
PPT	Past Participles‡	sido, afirmado, vivido, ···
PREP	Prepositions	de, para, ···
PRS	Personals	eu, tu, ele, ···
QNT	Quantifiers	todos, muitos, nenhum, ···
REL	Relatives	que, cujo, ···
STT	Social Titles	Presidente, dr^{a}., prof., ···
SYB	Symbols	@, #, &, ···
TERMN	Optional Terminations	(s), (as), ···
UM	“um” or “uma”	um, uma
UNIT	Measurement unit	kg., b.p.m., ···

Continued on next page...

Continued from previous page

Tag	Category	Examples
VAUX	Finite "ter" or "haver"‡	temos, haveriam, ···
V	Verbs§	falou, falaria, ···
WD	Week Days	segunda, terça-feira, sábado, ···

⋆: used in the transcription of spoken material

†: not in compound tenses

‡: in compound tenses

§: other than PPA, PPT, INF(AUX) or GER(AUX)

Some additional notes:

- The tags for multi-word expressions are formed as described in Section 4.5.1: Each component word in a MWE receives the same tag, formed by the POS of the whole expression prefixed by `L` and followed by an index number. For instance, the Preposition "em redor de" will be tagged as: `em/LPREP1 redor/LPREP2 de/LPREP3`.
- This table does not include the portmanteau tags used to resolve ambiguous strings (cf. Table 4.4 on page 80).

F Featurizer Rules

#fp (feminine plural)							
as	escas	ícas	inhas	âncias	ências	árias	delas
érrimas	íssimas	eiras	doras	uras	osas	itas	mentas
ivas	ezas	bes	ces	des	ies	ses	sões
ções	agens	ás	ãs	bés	cés	dés	iés
sés							

#fs (feminine singular)							
a	inha	érrima	íssima	ita	be	ce	de
ie	se	gem	são	ção	ez	iz	á
ã	bé	cé	dé	sé	cê		

#gp (plural, underspecified Gender)							
icidas	istas	ares	enses	is	ais		

#gs (singular, underspecified Gender)							
icida	ista	ense	al	vel	il	ar	

#mp (masculine plural)							
ícidas	emas	omas	es	hais	jais	oais	zais
éis	óis	ns	ens	éns	os	inhos	érrimos
íssimos	itos	us	cás	çás	fés	lés	nés
pés	rés	ís	ós	ús			

#ms (masculine singular)							
ícida	ema	oma	e	i	hal	jal	oal
zal	el	ol	ul	m	em	im	om
um	ém	n	o	inho	érrimo	íssimo	ito
p	er	ir	or	ur	és	ês	u
x	az	oz	uz	cá	çá	fé	lé
né	pé	ré	í	ó	vô	xô	ú

G Lemmatization Rules

-íssimo family (superlative)		
-íssima	→	-íssimo
-íssimo	→	-o
-císsimo	→	-z
-líssimo	→	-l
-abilíssimo	→	-ável
-ebilíssimo	→	-ével
-ibilíssimo	→	-ível
-obilíssimo	→	-óvel
-ubilíssimo	→	-úvel
-quíssimo	→	-co
-íssimas	→	-íssima
-íssimos	→	-íssimo

-inho family (diminutive)		
-inha	→	-a,-inho
-guinha	→	-ga,-guinho
-quinha	→	-ca,-quinho
-zinha	→	-za,-zinho
-inho	→	-o
-quinho	→	-co
-zinho	→	-
-inhas	→	-inha
-quinhas	→	-quinha
-zinhas	→	-zinha
-inhos	→	-inho
-quinhos	→	-quinho
-zinhos	→	-s,-zinho

-ito family (diminutive)		
-ita	→	-a,-ito
-quita	→	-ca,-quito
-zita	→	-zito
-ito	→	-o
-quito	→	-co
-zito	→	-
-itas	→	-ita
-quitas	→	-quita
-zitas	→	-zita
-itos	→	-ito
-quitos	→	-quito
-zitos	→	-zito

feminine forms								
-ã	→	-ão	-á	→	-á	-ba	→	-bo
-ça	→	-ço	-ca	→	-co	-da	→	-do
-ea	→	-eo	-fa	→	-fo	-ga	→	-go
-ha	→	-ho	-ia	→	-io	-fobia	→	-fobia
-ância	→	-ância	-ência	→	-ência	-eia	→	-eu
-heia	→	-heio	-grafia	→	-grafia	-filia	→	-filia
-metria	→	-metria	-ja	→	-jo	-la	→	-lo
-dela	→	-dela	-ola	→	-olo,-ol	-ma	→	-mo
-na	→	-no	-oa	→	-o	-pa	→	-po
-ra	→	-ro	-ora	→	-or	-ura	→	-ura
-sa	→	-so	-esa	→	-ês	-ta	→	-to
-menta	→	-menta	-ísta	→	-ísta	-ista	→	-ista
-ua	→	-u	-gua	→	-guo	-qua	→	-quo
-va	→	-vo	-xa	→	-xo	-za	→	-zo
-eza	→	-eza	-íza	→	-iz			

plural forms					
-ãs	→	-ã	-ás	→	-á
-as	→	-a	-és	→	-é
-es	→	-e	-ães	→	-ão
-ões	→	-ão	-eirões	→	-eirão
-res	→	-r	-ases	→	-ás
-ses	→	-s	-eses	→	-ês
-enses	→	-ense	-zes	→	-z
-ízes	→	-iz	-ís	→	-í
-is	→	-i	-ais	→	-al
-eis	→	-el	-éis	→	-el
-óis	→	-ol	-uis	→	-ul
-ns	→	-m	-os	→	-o
-ós	→	-ó	-us	→	-u

References

AIRES, RACHEL, 2000. *Implementação, Adaptação, Combinação e Avaliação de Etiquetadores para o Português do Brasil*. Master's thesis, Universidade de São Paulo, Brasil.

AIRES, RACHEL, SANDRA ALUÍSIO, DENISE KUHN, MARCIO ANDREETA AND OSVALDO OLIVEIRA, JR., 2000. Combining Multiple Classifiers to Improve Part of Speech Tagging: A Case Study for Brazilian Portuguese. In *Proceedings of the Brazilian AI Symposium*, pages 20–22.

ALMEIDA, JOSÉ AND ULISSES PINTO, 1994. Jspell — Um Módulo para Análise Léxica Genérica de Linguagem Natural. In *Proceedings of the 10th Encontro Anual da Associação Portuguesa de Linguística (APL)*.

ALUÍSIO, SANDRA, GISELE PINHEIRO, MARCELO FINGER, MARIA NUNES AND STELLA TAGNIN, 2003. The Lacio-Web Project: Overview and Issues in Brazilian Portuguese Corpora Creation. In *Proceedings of the Corpus Linguistics 2003 Conference*, pages 14–21.

ALVES, CARLOS, 1999. *Etiquetagem do Português Clássico Baseada em Corpus*. Master's thesis, Universidade de São Paulo, Brasil.

BERGSTRÖM, MAGNUS AND NEVES REIS, 2004. *Prontuário Ortográfico e*

Guia da Língua Portuguesa. Editorial Notícias, 47th edition. ISBN 972-46-0840-9.

BICK, ECKHARD, 2000. *The Parsing System PALAVRAS: Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework*. Ph.D. thesis, University of Århus, Denmark.

BOITET, CHRISTIAN AND PETE WHITELOCK, editors, 1998. *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th Conference on Computational Linguistics (COLING)*. ACL, Morgan Kaufmann Publishers.

BRANCO, ANTÓNIO AND FRANCISCO COSTA, 2006. Noun Ellipsis without Empty Categories. In *Proceedings of the 13th International Conference on Head-Driven Phrase Structure Grammar*, pages 81–101. CSLI Publications.

BRANCO, ANTÓNIO, FRANCISCO COSTA AND FILIPE NUNES, 2006. Processamento da Ambiguidade Flexional Verbal: Para uma Caracterização do Espaço do Problema. In *Proceedings of the 22nd Encontro Anual da Associação Portuguesa de Linguística (APL)*.

BRANCO, ANTÓNIO AND JOÃO RICARDO SILVA, 2001. EtiFac: A Facilitating Tool for Manual Tagging. In *Proceedings of the 17th Encontro Anual da Associação Portuguesa de Linguística (APL)*, pages 81–90.

BRANCO, ANTÓNIO AND JOÃO RICARDO SILVA, 2002. POS Tagging Without a Training Corpus or a Large-Scale Lexicon: How far can one get? In *Proceedings of the 18th Encontro Anual da Associação Portuguesa de Linguística (APL)*, pages 211–222.

BRANCO, ANTÓNIO AND JOÃO RICARDO SILVA, 2004. Evaluating Solutions for the Rapid Development of State-of-the-Art POS Taggers for

Portuguese. In *Proceedings of the 4th Language Resources and Evaluation Conference (LREC)*, pages 507–510.

BRANTS, THORSTEN, 1995. Tagset Reduction without Information Loss. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 287–289.

BRANTS, THORSTEN, 2000. TnT — A Statistical Part-of-Speech Tagger. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st North American Chapter of the Association for Computational Linguistics*, pages 224–231.

BRILL, ERIC, 1992. A Simple Rule-Based Part-of-Speech Tagger. In *Proceedings of the 3rd Applied Natural Language Processing Conference*, pages 152–155.

BRILL, ERIC, 1994. Some Advances in Transformation-Based Part of Speech Tagging. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, pages 722–727.

BRILL, ERIC, 1995a. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of Speech Tagging. *Computational Linguistics*, 21(4):543–565.

BRILL, ERIC, 1995b. Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In *Proceedings of the 3rd Workshop on Very Large Corpora*.

BRILL, ERIC AND JUN WU, 1998. Classifier Combination for Improved Lexical Disambiguation. In Boitet and Whitelock (1998), pages 191–195.

CARRERAS, XAVIER, ISAAC CHAO, LLUÍS PADRÓ AND MUNTSA PADRÓ, 2004. FreeLing: An Open-Source Suite of Language Analyzers. In *Proceedings of the 4th Language Resources and Evaluation Conference (LREC)*.

CHARNIAK, EUGENE, 1997. Statistical Techniques for Natural Language Parsing. *AI Magazine*, 18(4):33–44.

CHARNIAK, EUGENE, CURTIS HENDRICKSON, NEIL JACOBSON AND MIKE PERKOWITZ, 1993. Equations for Part-of-Speech Tagging. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI)*, pages 784–789.

CORMEN, THOMAS, CHARLES LEISERSON, RONALD RIVEST AND CLIFF STEIN, 2001. *Introduction to Algorithms*. The MIT Press, 2nd edition. ISBN 0-262-03293-7.

CUNHA, CELSO AND LINDLEY CINTRA, 1986. *Nova Gramática do Português Contemporâneo*. Edições João Sá da Costa, 3rd edition.

DAELEMANS, WALTER, JAKUB ZAVREL, PETER BERCK AND STEVEN GILLIS, 1996. MBT: A Memory-Based Part-of-Speech Tagger-Generator. In *Proceedings of the 4th Workshop on Very Large Corpora*, pages 14–27.

DOUGHERTY, DALE AND ARNOLD ROBBINS, 1997. *sed & awk*. O'Reilly & Associates, Inc., 2nd edition. ISBN 1-56592-225-5.

ELWORTHY, DAVID, 1995. Tagset Design and Inflected Languages. In *Proceedings of the Association for Computational Linguistics SIGDAT Workshop*.

FINGER, MARCELO, 2000. Técnicas de Otimização da Precisão Empregadas no Etiquetador Tycho Brahe. In *Proceedings of the 5th Encontro*

para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR), pages 141–154.

FLORIAN, RADU AND GRACE NGAI, 2001. Fast Transformation-Based Learning Toolkit. Technical report, John Hopkins University.

GAUSSIER, ÉRIC, 1999. Unsupervised Learning of Derivational Morphology from Inflectional Lexicons. In *Proceedings of the ACL Workshop on Unsupervised Methods in Natural Language Learning*.

GIMÉNEZ, JESÚS AND LLUÍS MÀRQUEZ, 2004. SVMTool: A General POS Tagger Generator Based on Support Vector Machines. In *Proceedings of the 4th Language Resources and Evaluation Conference (LREC)*.

GOLDSMITH, JOHN, 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics*, 27(2):153–198.

GREFENSTETTE, GREGORY AND PASI TAPANAINEN, 1994. What is a Word? What is a Sentence? Problems of Tokenisation. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX)*, pages 79–87.

GROVER, CLAIRE, COLIN MATHESON, ANDREI MIKHEEV AND MARC MOENS, 2000. LT TTT — A Flexible Tokenisation Tool. In *Proceedings of the 2nd Language Resources and Evaluation Conference (LREC)*.

HABERT, BENOÎT, GILLES ADDA, MARTINE ADDA-DECKER, PHILIPPE BOULA DE MARËUIL, SERGE FERRARI, OLIVIER FERRET, GABRIEL ILLOUZ AND PATRICK PAROUBEK, 1998. Towards Tokenization Evaluation. In *Proceedings of the 1st Language Resources and Evaluation Conference (LREC)*, pages 427–431.

HAJIČ, JAN, 2000. Morphological Tagging: Data vs. Dictionaries. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st North American Chapter of the Association for Computational Linguistics*, pages 94–101.

HAJIČ, JAN AND BARBORA HLADKÁ, 1997. Probabilistic and Rule-Based Tagger of an Inflective Language — A Comparison. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 111–118.

HAJIČ, JAN AND BARBORA HLADKÁ, 1998. Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. In Boitet and Whitelock (1998), pages 483–490.

IDE, NANCY, 1998. Corpus Encoding Standard: SGML Guidelines for Encoding Linguistic Corpora. In *Proceedings of the 1st Language Resources and Evaluation Conference (LREC)*.

KARLSSON, FRED, 1990. Constraint Grammar as a Framework for Parsing Running Text. In HANS KARLGREN, editor, *Proceedings of the 13th Conference on Computational Linguistics (COLING)*, volume 3, pages 168–173.

LENAT, DOUGLAS, RAMANATHAN GUHA, KAREN PITTMAN, DEXTER PRATT AND MARY SHEPHERD, 1990. Cyc: Towards Programs with Common Sense. *Communications of the ACM*, 33(8):30–49.

LESK, MIKE, 1975. LEX — A Lexical Analyzer Generator. Technical Report 39, AT&T Bell Laboratories.

MANNING, CHRISTOPHER AND HINRICH SCHÜTZE, 1999. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1st edition. ISBN 0-262-13360-1.

MARQUES, NUNO AND GABRIEL LOPES, 2001. Tagging with Small Training Corpora. In *Proceedings of the International Conference on Intelligent Data Analysis (IDA)*, number 2189 in Lecture Notes on Artificial Intelligence (LNAI), pages 63–72. Springer-Verlag.

MATEUS, MARIA HELENA MIRA, ANA MARIA BRITO, INÊS DUARTE, ISABEL HUB FARIA, SÓNIA FROTA, GABRIELA MATOS, FÁTIMA OLIVEIRA, MARINA VIGÁRIO AND ALINA VILLALVA, 2003. *Gramática da Língua Portuguesa*. Caminho, 5th edition. ISBN 972-21-0445-4.

MEGYESI, BEÁTA, 2001. Comparing Data-Driven Learning Algorithms for PoS Tagging of Swedish. In *Proceedings of the 6th Conference on Empirical Methods in Natural Language Processing*, pages 151–158.

MERIALDO, BERNARD, 1994. Tagging English Text with a Probabilistic Model. *Computational Linguistics*, 20(2):156–171.

MIKHEEV, ANDREI, 1997. LT POS — The LTG Part of Speech Tagger. Technical report, Language Technology Group, University of Edinburgh.

MIKHEEV, ANDREI, 2000. Tagging Sentence Boundaries. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st North American Chapter of the Association for Computational Linguistics*, pages 264–271.

MIKHEEV, ANDREI, 2002. Periods, Capitalized Words, etc. *Computational Linguistics*, 28(3):289–318.

OFLAZER, KEMAL, 2006. Computational Morphology. Foundational Course, Language and Computation Section, at the 18th European Summer School in Logic, Language and Information (ESSLLI).

PALMER, DAVID, 1994. SATZ — An Adaptive Sentence Segmentation System. Technical Report CSD-94-846, University of California at Berkeley.

PALMER, DAVID AND MARTI HEARST, 1997. Adaptive Multilingual Sentence Boundary Disambiguation. *Computational Linguistics*, 23(2):241–267.

PAXSON, VERN, 1988. *Flex — Fast Lexical Analyzer Generator*. Free Software Foundation.

PINKER, STEVEN, 1994. *The Language Instinct*. Harper Collins.

PINTO, JOSÉ MANUEL CASTRO, 2004. *Novo Prontuário Ortográfico*. Plátano Editora, 5th edition. ISBN 972-770-002-0.

PORTER, MARTIN, 1980. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137.

RATNAPARKHI, ADWAIT, 1996. A Maximum Entropy Model for Part-of-Speech Tagging. In ERIC BRILL AND KENNETH CHURCH, editors, *Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing*, pages 133–142. ACL.

REIS, RICARDO AND JOSÉ ALMEIDA, 1998. Etiquetador Morfo-sintáctico para o Português. In *Proceedings of the 13th Encontro Anual da Associação Portuguesa de Linguística (APL)*, pages 209–221.

REYNAR, JEFFREY AND ADWAIT RATNAPARKHI, 1997. A Maximum Entropy Approach to Identifying Sentence Boundaries. In *Proceedings of the 5th Applied Natural Language Processing Conference*, pages 16–19.

RIBEIRO, RICARDO, 2003. *Anotação Morfossintáctica Desambiguada do Português*. Master's thesis, Universidade Técnica de Lisboa — Instituto Superior Técnico.

RIBEIRO, RICARDO, LUÍS OLIVEIRA AND ISABEL TRANCOSO, 2003. Using Morphosyntatic Information in TTS Systems: Comparing Strategies for European Portuguese. In *Proceedings of the 6th Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR)*, number 2721 in Lecture Notes on Artificial Intelligence (LNAI), pages 143–150. Springer-Verlag.

RILEY, MICHAEL, 1989. Some Applications of Tree-Based Modelling to Speech and Language. In *Proceedings of the HLT'89: Workshop on Speech and Natural Language*, pages 339–352.

SAG, IVAN, TIMOTHY BALDWIN, FRANCIS BOND, ANN COPESTAKE AND DAN FLICKINGER, 2002. Multiword Expressions: A Pain in the Neck for NLP. In *Proceedings of the 3rd Conference on Intelligent Text Processing and Computational Linguistics*, pages 1–15.

SAMUELSSON, CHRISTER AND ATRO VOUTILAINEN, 1997. Comparing a Linguistic and a Stochastic Tagger. In PHILIP COHEN AND WOLFGANG WAHLSTER, editors, *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 246–253. ACL, Morgan Kaufmann Publishers.

SCHMID, HELMUT, 1994. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of the International Conference on New Methods in Language Processing*, pages 44–49.

SILVA, GILBERTO AND CLAUDIA OLIVEIRA, 2001. Lematizadores com Base em Léxico. Technical Report 069/DE9/01, Departamento de Engenharia de Sistemas do Instituto Militar de Engenharia.

SIMÕES, ALBERTO AND JOSÉ ALMEIDA, 2000. jspell.pm — Um Módulo de Análise Morfológica para Uso em Processamento de Linguagem Natural. In *Proceedings of the 16th Encontro Anual da Associação Portuguesa de Linguística (APL)*, pages 485–495.

TUFIS, DAN, 1999. Tiered Tagging and Combined Language Models Classifiers. In VÁCLAV MATOUSEK, PAVEL MAUTNER, JANA OCELÍKOVÁ AND PETR SOJKA, editors, *Proceedings of the 2nd Workshop on Text, Speech and Dialogue*, volume 1692 of *Lecture Notes in Computer Science (LNCS)*, pages 28–33. Springer-Verlag.

TUFIS, DAN, PÉTER DIENES, CSABA ORAVECZ AND TAMÁS VÁRADI, 2000. Principled Hidden Tagset Design for Tiered Tagging of Hungarian. In *Proceedings of the 2nd Language Resources and Evaluation Conference (LREC)*, pages 1421–1426.

TUFIS, DAN AND OLIVER MASON, 1998. Tagging Romanian Texts: A Case Study for QTAG, a Language Independent Probabilistic Tagger. In *Proceedings of the 1st Language Resources and Evaluation Conference (LREC)*, pages 589–596.

VAN HALTEREN, HANS, JAKUB ZAVREL AND WALTER DAELEMANS, 2001. Improving Accuracy in Word Class Tagging through the Combination of Machine Learning Systems. *Computational Linguistics*, 28(2):199–211.

WALL, LARRY, TOM CHRISTIANSEN AND JON ORWANT, 2000. *Programming Perl*. O'Reilly & Associates, Inc., 3rd edition. ISBN 0-596-00027-8.

ZAJAC, RÉMI, 2001. Morpholog: Constrained and Supervised Learning of Morphology. In *Proceedings of the 5th Conference on Natural Language Learning*, pages 90–96.

Errata

This errata lists some errors that were found in the version that was submitted for acceptance. For reader's convenience, these errors have already been corrected in the current version.

- page 85, footnote 15
  98% → 97%

- page 104, under item 4
  instabilidade → miséria

- page 137, under "sentence segmentation"
  99.4% precision → 99.93% precision