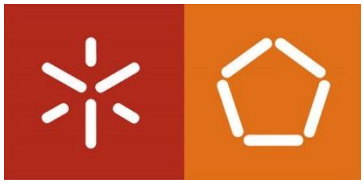

Master's Degree Medical Informatics

1st year | 2022/2023 | 2nd semester

Natural Language Processing

TP1

Teachers:

José Almeida

Luís Cunha

Florian Hetzel EE10945

Hugo Silva PG50416

Tomás Lima PG50788

Introduction

The presented task was to extract from a set of PDF documents the useful information about medical terms and procedures, such as its description and translation to other languages. For this, the acquired knowledge in Natural Language Processing was put into practice, in order to correctly select the chunks of important data.

Four documents were processed: *dicionario_termos_medicos_pt_es_en.pdf*, *anatomia geral.pdf*, *Dicionario_de_termos_medicos_e_de_enfermagem.pdf* and *Glossario de Termos Medicos Tecnicos e Populares.pdf*. It was soon clear that the best starting point to begin the data extraction was to first convert the PDF's into XML format, because when converting to TXT and HTML it was often seen that some terms and its corresponding descriptions would appear separated by other terms and descriptions. This would make almost impossible the correct data extraction, this without mentioning the extra text and handlers created when converting to HTML. This step of conversion to XML was achieved by running the command *pdftohtml –xml <filename>*.

After conversion, the main goal was to find patterns in the data that could be useful to extract the important data. Using regular expressions manipulation in several steps, it was possible to create a simple and workable format, over which the term identification and corresponding description/translation could easily be identified. The results were stored in a python dictionary, whiched would then be transferred to a JSON, using the in-built python function *json.dump()*.

The four JSON files originated from the four PDF documents were then combined into a single JSON, where the common keys were the merging factor. This is, if in one json file there was the same key as in another one, the resulting combination was one single key, with the joined information in its corresponding value, using a dictionary of dictionaries sort of structure.

Finally, in order to more easily share and present the achieved result, a website was created using the Flask framework.

Dicionário de Termos Médicos PT-ES-EN

To retrieve the information from this dictionary, the PDF file was converted to the XML format. Since the dictionary has three parts that differ in the origin language, only the part with portuguese as origin language was used, starting from page 252.

Next, the xml file was processed using the re.sub() function and then saved in a new XML file. First, the text flag was removed using the pattern "<text.+?>" and replacing by "". Likewise, the end text flag ("</text>), the page flag (<page.+>), page number (\d+) and the capital letters in the beginning of the sections ([A-Z]) were removed. Furthermore, in the PDF file a notch on the side of the pages can be found, which specifies the origin language and the translated languages (i.e. português – inglês – espanhol). In the XML these and the end page flag were removed together using the pattern "(.+\n)+</page>". Following that, the italic gender decleration of the terms (i.e. m – masculin, f – feminin) were eliminated because they were seen unnecessary. The pattern "\n.+" was used to identify and clear the XML from these. Additionally, it was found that term declarations were split into two lines if a gender was defined in italic in between the words of the term declaration (i.e. **aberração** *f* **cromossómica)**. To overcome that, the pattern "\n" was replaced by a blank space. To omit empty lines the pattern "\n\n" was substituted using "\n". Next, due to gender and adj. specifications, there could be structures found like "(adj. +)" or "(+)". To remove them the pattern "\(\n?[^\)]*\n?\+\n?[^\)]+\n?\)" was used. This pattern is very complex due to the number of exceptions. Therefore, this pattern will get a longer explanation here. It starts with the opening parenthesis, followed by an optional line break. After that there follow zero or more characters that are not a closing parenthesis and again an optional line break. Next, is the + followed by the next optional line break and zero or more characters that are not a closing parenthesis. Finally, there is another optional line break and the closing parenthesis.

The last re.sub() operation removes the end bolt flags with the pattern "". This leaves the XML to follow the pattern as:

```
<b> poruguese term declaration

U

English translation

E

Spanish translation
```

In a separate python file first a re.split() operation splits the XML content at “<b> to separate the terms and saves them in a list. Following that, for each list element a re.split() is performed, splitting at “\nU\n” an “\nE\n” to separate the declaration, english translation and spanish translation. Then, the parts are saved in a dictionary, using the portugues term as key, which gets another dictionary as value. Inside this second dictionary the english translation is assigned to the key “en” and the portuguese translation to the key “pt”.

Finally, the dicyionary is saved in a JSON file.

There are some exceptions, that are corrected manually. This is rational, because it is not useful to write a re.sub() operation for single appearance exceptions.

The first exception was to delete bloqueador beta from the page header (line 2936 in the new XML file).

The second exception was due to a mixture of bold and non-bold term because the pdf does not define closing parenthesis as bold (cisto (o quisto) - line 3958, dióxido de carbono (CO2) - line 6705, palato (bras.: palate) (duro/ mole) - line 16543.

Anatomia Geral

Analysing the .pdf and .xml files, we can detect some patterns. First, the pages 2, 4, 6, 8, 10, 12, 20, 22 and 24 only contain images, so they can be removed. This can be done using the following regular expression, that removes everything between a “<page...” and the following “</page>” of the number pages mentioned before

```
'<page number="([2468]|1[02]|2[024])".*\n(.*\n)*?</page>\n'
```

Some of the terms are in italic (surrounded by "<*i*>...</*i*>"), others in bold (surrounded by "<*b*>...</*b*>"), and some of the terms are just in plain text. This way, the only way to identify a term is through the font size of the text.

There are a total of 20 font sizes, from 0 to 19, and the fonts that are significant to us (they represent a term or a description) are 5, 6, 7, 12 and 18. The other ones identify other elements in the .pdf file, such as the page number, the title of the page, the chapter of the book, the group of the terms (insignificant to the processing we are making), and other elements.

This way, we use the following regular expressions to mark the text, removing the XML elements and adding a "#X=" marker to the start of the line, with the X being the font size of the text.

`'(?:<text.*?font="([0-9]*)".*?>(?:<.*?>)*\s*([^<>]+)\s*(?:<.*?>)*\n)'` to `'#\1=\2\n'`

After that, we remove all the lines that don't have markers, using the following regular expression:

```
'<.+\n'
```

Some of the lines now have a label at the end, such as "A", "B,", "C " or "D, ", that way, the following regular expression is used to remove them:

```
'([ABCD],? ?)+\n'
```

Now, there are some word that were separated in the end of a line, so there's 2 lines, the first one having a "-" at the end, that should be just one, so we can use the following regular expressions to join them:

`'(#.*?=.*)-\n#.*?=(.*)'` to `'\1\2'`

We can now use the following regular expression to remove the label of the terms (for example, the "14" in "14 Pescoço …"):

```
'#.+=[0-9]*\n'
```

Then, the descriptions are still separated in different lines, so we used the following regular expressions to join consecutive lines of descriptions (lines with "#3=" at the start):

`'(#3=.+)\n#3=(.+)'` to `'\1 \2'`

We then removed the insignificant font sizes with two regular expressions, the first one removing the insignificant text and their descriptions (some of them have one), and the second one removing the ones that don't have descriptions:

```
'#([0124]|1[0179])=.*\n #3=.+\n'
```

```
'#([0124]|1[0179])=.*\n'
```

Now we changed the "#3=" to "#D=", identifying the descriptions, and the rest of the "#X=" to "#T=", identifying the terms, using these regular expressions:

`'#3=(.+)' to '#D=\1'`

`'#(?:[567]|1[28])=(.+)'` to `'#T=\1'`

Then, we only searched for the terms with descriptions, since the ones without are not useful to create a dictionary, using the following regular expression:

```
'#T=(.+)\n#D=(.+)\n'
```

Then, we created a python dictionary and added all the results found with the previous regular expression, using the first capture group as the key and the second one as the value, and then we saved the dictionary in a .json file.

Dicionário de Termos Médicos e de Enfermagem

To retrieve the information from this dictionary, the PDF file was first converted to the XML format. The lines before the first term declaration "A, An" and after the description of the last term "ZUMBIDO" were deleted manually.

It was seen that in this document it would be useful to use the *font* information originated from the conversion to XML. Like so, the first step was to remove the information before it on every line, selecting it with the regular expression:

```
r'<text.+height="\d+" '
```

and substituting it by nothing, using the *re.sub()* function, like this:

```
new_xml = re.sub(r'<text.+height="\d+" ',"",xml)
```

Where *xml* is the data retrieved from the XML and the *new_xml* is the new variable to store the converted data. For the sake of simplicity, every time it will be said that a chunk of data was removed, this was the procedure adopted.

Next up, other XML props were tackled, such as the page opening, text closing and *fontspec* marks. They were promptly deleted using the following regular expressions:

```
r"</text>"

r"<page.+>"

r"\n?</page>\n?"

r"\n\s<fontspec.*"

r'font="\d+">Sou Enfermagem.*\n'
```

Since the font size was going to be used as the term and description identifiers, the italic and bold marks could be deleted, using the following regular expressions:

```
r"<b>(.+)</b>"

r"<i>(.*)</i>"
```

However, in this case, the text between marks is important, so it should be kept and not just substituted by an empty string "". To save the important data, a capture group for the information between marks was created, which would then be the replacer for the whole regular expression. The first capture group in a regular expression is addressed using the \1 identifier. Likewise, the substitution for the bold markers looks like:

```
new_xml = re.sub(r"<b>(.+)</b>",r"\1", new_xml)
```

The XML created some lines with dots, which were removed using:

```
r'font="\d+">o\n'
```

It was identified that, for some reason, all the words containing the the *fi* syllable would be separated, being that this sylable would appear in a line alone. For example, the word *identification* would show up as:

```
font=”12”>«some text before» identi

font=”12”>fi

font=”12”>cation «some text after»
```

To remove this, all the cases were identified using the regular expression:

```
r'\nfont="\d+">fi\nfont="\d+"> '
```

and substituted simply by the “fi” string.

The lines containing the chapter markers (3 capital letters, like ATE) were identified as being the only ones with font size 20. On the same way, the lines containing the page numbers were the only ones with font size 6. The lines containing a single “-” were the only ones with font size 19. These lines were deleted using the regular expression:

```
r'\nfont="(20|6|19)">.+'
```

Some empty lines were found. These were targeted with the regular expression:

```
r'\nfont="\d+"> \n'
```

and substituted by a “\n”.

Some words were separated in different lines, respecting the hyphenation rule. This is, they were divided and the break point identified with a hyphen. These cases were identified with the regular expression:

```
r'-\nfont="\d+">(.)'
```

and substituted by the first capture group “\1”, which was the first following letter on the next line.

The terms were identified as having a font size of either 10 or 24, and no other lines had this font size. This made it easy to correctly identify the terms. However, some of them were split in more than one line, so in order to join them, those cases were identified with the regular expression:

```
r'(font="(24|10)">.+)\nfont="(24|10)">(.+)'
```

where two following lines starting with a font size of 10 or 24 would be joined by replacing this with the first and fourth capture groups (`r"\1\4"`). First capture group keeps the first line with the font information, second capture group is the 10 or 24 in the first line font, third capture group is the 10 or 24 in the second line font and finally the fourth capture group is the term continuation in the second line.

The way found to easily identify term and corresponding description was to add a "@" after the term and join the description all together after it, in the same line. This was achieved in two steps. First, the "@" character was added after the terms, identified as having the fontsize 10 or 24 with the following regular expression:

```
r'font="(24|10)">(.+)'
```

and substituting it by second capture group followed by the "@" (`r"\2@"`). In this step we also finally delete the font information for the terms. The only font information remaining is the one of the descriptions, which is useful to easily identify them with:

```
r'\nfont="\d+">(.)'
```

and substituting it by the capture group, preceded by a whitespace (`r" \1"`). The whitespace is added to correctly separate the words which were separated in different lines before.

The *new_xml* was saved to a new file, that would then be opened by a separate script to make the dictionary and JSON file creation. This scrip simply had to read the file by lines and, for each line, split it on the "@" character. The term (key) would then be the first element of the resulting list and the description (value) the result of the strip of the second element, to remove the whitespace in the beginning and \n in the end.

The resulting dictionary was then dumped to a JSON file.

Glossário de Termos Médicos Técnicos e Populares

Analysing the XML file, we can detect some useful patterns. Each line of the PDF file is generally separated into 2 groups, the term, in bold ("*…*"), usually followed by an empty text or a comma, and the description, in italic ("*…*"),

usually followed by a text containing a “(pop)”. There are, however, some exceptions to this:

- the definition of the terms 'acne' and 'esfoliação' are not followed by a '(pop)', instead they are together in the same text (so we will need to separate the two).
- the terms 'acne', 'mediador', 'pico do débito', 'vulvovaginite' and 'anxiolítico' are not followed by an empty text or a comma (this won't be a problem, since the marking of the terms only uses the fact that they're bold).
- in some lines that end with a term and the next one begins with another term, the first one is not followed by an empty text or a comma, such as 'adolescente', 'receptor' and 'polidipsia' (this won't be a problem, since the marking of the terms only uses the fact that they're bold).
- in some lines, terms with 2 words are separated in the end, and they appear in 2 different XML elements, such as 'pré-medicação', 'infecção cruzada', 'efeito colateral' and 'tremor intencional' (so we will need to join them together).

It is essential to also notice that the descriptions don’t always follow the terms, so we can’t rely on the order of these two.

This way, we first make a pre-processing of the text, removing multiple spaces and new lines, removing punctuations in the start and the end of the lines, and removing the “\t” character, using the regular expressions:

`'([ ]){2,}'`, `'([\n]){2,}'`, `'>[ ,.]+'`, `'[ ,.]+<'` and `'\t'`

Now, we can start the processing of the file, fixing first the descriptions that have the “(pop)” in them, by use of the regular expressions:

`'(<.+?>)(.+?) \(pop\)(<.+?>)'` to `'\1<i>\2</i>\3\n\1(pop)\3'`

Then, we remove all non-text elements, with:

`'\n<[^t].+'` and `'^<[^t].+\n'`

We can then remove the maker of the sections for the manual search (for example, to search for “vacina”, we go to the “V” section), using the fact that they have a font size of 4 or 6:

`'<.+font="[46]".+\n'`

The, we can put the descriptions that are separated into different lines back in the same line, using:

`'(.+?<i>.+?)</i>.+?\n.+?<i>(.+?\n)'` to `'\1 \2'`

Now we can remove all the terms that don't have descriptions, such as "ambiente", using:

`'(<.+>)<b>(.+)</b>(.+)\n(<.+>\(pop\)<.+>)\n'`

We can now mark the terms with "#T=" and the descriptions with "#D=" using:

`'.+?<b>(.+)</b>.+?\n'` to `'#T=\1\n'` and `'.+?<i>(.+)</i>.+?\n'` to `'#D=\1\n'`

We can now remove the rest of the XML elements, by using:

`'<.+\n'`

After all of that, there's 5 lines that were in bold and were marked with "#T=" incorrectly, for they are not terms, but since they're the first 5 lines, they can easily be removed by using this regular expression 5 times:

`'#.+\n'`

Then, we need to fix the terms "pré-medicação", "infecção cruzada", "efeito colateral" and "tremor intencional", since they were separated into 2 different lines. We fixed them almost manually, creating a list of tuples, where the first element was the first word and the second one was the second word. This way, we can use the following regular expressions to fix them:

`'#T='+one+r"\n#T="+two` to `'#T='+one+r' '+two` (where "one" is the first word and "two" is the second one)

After that, we created a dictionary, and because the terms and descriptions are not in order, we made a while loop that took the first two lines of the text, and added them to the dictionary, with the line with the "#T=" as the key and the line with the "#D=" as the value, as long as they were different from each other, and removing them both from the text after that. If them both have "#T=" or "#D=", it only removes the first one and prints "ERROR" into the console (after all the processing we made, there were no prints). In the end, the dictionary is saved into a .json file.

JSON combining

The combination of the four resulting JSON's was handled by simple for loops. First a new dictionary was created by copying the dictionary of the translations JSON (*dicionario_termos_medicos_pt_es_en_new.json*). Then, looping through the three remaining JSON's, we would look for the existence of the key in the first dictionary. If it existed, the new information would be added. If not, a new key would be added with the new term and corresponding information. For the cases where more than one description would be available for a term, a list was created to store all possible descriptions.

In order to enrich the final dictionary, it was decided to add the English translation for the terms that didn't have the translation in the *dicionario_termos_medicos_pt_es_en* PDF file. To achieve this, the *GoogleTranslator* from the *deep_translator* tool was used. A loop was created over the dictionary, and for each term it was identified if the "en" key in the corresponding sub-dictionary. This was the key set to identify the English translation. If it was not present, it would be added and the corresponding value was the generated by the *GoogleTranslator*.

Website

To present the results of the assignment a website was created using Flask and Bootstrap 5. First, a layout.html was written to ensure a uniform style of the website. Inside the body of the layout.html a body block was inserted which was extended in the other HTMLs. A simple home page was matched to the route "/" with a short information text. The route "/terms" which renders the template terms.html provides a list of all keys from the JSON in a sorted order. Thereby, some exceptions came to light, that were manually corrected in the JSON. The exceptions were: 'blister', (d)escamação, (herpes) Zóster, .profilaxia, [[fossa retromandibular]], [[origem]], a f i n i d a d e, a(c)to obsessivo. These exceptions were mostly no mistakes in the processing step of the PDF, but instead like this in the dictionaries processed. ".profilaxia" and "a f i n i d a d e" were found to be due to bad pdf to xml transformation. All of these exceptions were manually corrected to ensure a pleasant presentation of the results.

By clicking on a presented term in the list of terms, the user gets redirected to the route "/term/<t>". The rendered template term.html presents a Bootstrap5 card with the stored information about this term, like English translation, Spanish translation and description(s).

The last route /search renders the template search.html which allows the user to search for terms in the dictionary. This was realized by a Boostrap5 form and input.group, allowing to save the user_input as variable *search_term* and performing a search for keys with that search_term in the dictionary. The matching results are stored in a list named *results* and then presented to the user offering hyperlinks to the detailed information about each term.

Conclusion

The final product was a JSON file with 10383 entries. Comparing the number of hand changes that had to be done and this really big number of final entries achieved, there is no other possible outcome than to be satisfied with the performance of the process.