Master's Degree Medical Informatics

1st year | 2022/2023 | 2nd semester

Natural Language Processing

TP2

Teachers:

José Almeida

Luís Cunha

Florian Hetzel EE10945

Hugo Silva PG50416

Tomás Lima PG50788

Introduction

The presented task targeted enriching the previous developed database and website of medical terms. The goal was to achieve a final product that could have a meaningful application on the real world and be easily handled. The group committed also to finding relations between terms, to finding a certain order among the immense sea of collected terms.

To achieve these goals, the skills acquired in new learnt tools, such as Spacy and BeautifulSoup, were put into practice. BeautifulSoup was used on web scraping tasks, to collect new terms, descriptions, or translations from different sources. Spacy was used on the relations identification process, to reduce the computational task in more than 50%. Google translator library was used once again to create German translations of the terms.

With the relations created, we intended to create a categorisation of the data, analysing which terms had more relations associated to them. The ones with the most will then be our top categories, the top of our data pyramid. The rest of the data will therefore unfold under them.

For a better end user experience, some functionalities were added, like the possibility to add or delete terms or categories. The initial given categories are the ones we identified as being the 15 most used in the medical lexical world. When a new term is added, its description is analysed so that relations can be created to the already existing terms in the database. It is also possible to search for information. The search engine looks for matches in the terms. The possibility to search for information in the translations or in the descriptions was not added, for the huge size of the database. This process would then put a lot of pressure in the server.

The built website was a continuation of the previous developed tool, using Flask framework.

Dataset Enrichment

Harvard Medical Dictionary of Health Terms

This online dictionary contains 2.042 medical terms. They are stored each letter on a separate page. So, to receive all terms, first it is necessary to get the anchors pointing to the individual pages. Therefore, the html code of the page of the character A was requested using the requests module. After, the html was parsed to Beautiful Soup. Next, it was found that the anchors are under a class with name "content-repository-content prose max-w-md-lg mx-auto flow-root getShouldDisplayAdsAttribute". By using .find_all() function together with the class this part of the html was retrieved. Then by using find_all() again, all anchors <a> were caught and the links to the individual pages could be extracted by using a['href']. The links were then stored in a list.

Next, a loop over the list of links was performed. By concatenating the base URL socket with the links, the individual pages for each letter of the alphabet could be requested and afterwards passed in the Beautiful Soup module. Again, first the main body of the page was caught with the same class as above. Next, all paragraphs <p> were listed by find_all(). Only the first two and the last paragraph did not contain relevant information, that's why they were omitted when looping the list. By splitting at ':' the term ([0] element) and the description ([1] element) could be saved to a JSON file.

Atlas da Saúde

The information from this source was collected during the classes, so we decided that since we had this information available in an already created JSON file, we could very easily incorporate it in our dataset.

Medical Dictionary Online

This online dictionary contained 28.471 terms. They were stored in individual pages for each starting letter. The anchors for the different pages were found to be stored in class “card-body”. The links to pages were extracted by using find_all(‘a’), to get all the anchors, then the URLs were caught by a[‘href’] for each a.

Following that, a loop over all the URLs was performed in which the individual pages were requested and parsed in the Beautiful Soup. Since all the terms are stored as list elements and link to their term page, find_all(‘li’) was used to get the terms. Then by performing li.a[‘href’] delivered the link to the term. All the links to terms were stored in a list.

Last, another for loop requested the html code of each term and parsed it to Beautiful Soup. In the term page the term itself was found to be in a <h2> and the description in the only <p> on the page. Two simple .find() operations retrieved the term and the description before saving them in a dictionary. Also, a list of relations was stored in the dictionary. The hyperlinks in the term description were used to create these. They were caught by using first find(‘p’) and then find_all(‘a’).

Finally, the dictionary was saved as a JSON file.

MedTerms Medical Dictionary A-Z List

This dictionary gave 12.326 terms. Again, the terms were in different pages depending on the starting character. By analysing the different links, it was found that only the requested letter changes in the link. A basic link was created:

https://www.medicinenet.com/script/main/alphaidx.asp?p={letter}_dict

In this, only the desired letter needed to be filled in. By looping over the alphabet, all pages could be requested on after another. Inside the page, the terms were stored in list objects <li> under the div with class AZ_results. The find function was used to get the div of class AZ_results before catching all <li> with find_all. Next, by looping over the list of <li> and extracting the link to the term page with li.a[‘href’] the individual term pages were requested. Then a find was used to get the div of class ‘pgContent’ to get the section of interest from the page. The term and description were then separated by a .split(‘:’) operation. The relations were retrieved by using find_all(‘a’) on the description text.

Finally, the term, description and relations were added to a dictionary, which was in the end stored as JSON file.

Dataset Cleaning

In the previous work, the keys of the dataset were the terms in Portuguese. However, since the availability of English dictionaries and glossaries in the web is much bigger, it was decided to swap the key by the English translation. This way, the dataset would then be consistent with the new added terms.

Since the data was coming from different sources, it was common that one dictionary contained, for example, the term “Diabetes” and another spelled it as “diabetes”. This way, when merging the info to a single dictionary, they were considered as being different keys (terms), resulting on the existence of duplicated information. This created the necessity of merging these duplicates under a single key, which is done by the *duplicates.py*, retaining the uppercase keys.

It was found that the Medical Dictionary Online contained a lot of countries and regional entries. Spacy was used to delete these entries, by searching for entities in the keys list and verifying if they were of the type “LOC”. This type was chosen, instead of “GPE”, for it being more accurate identifying real locations. “GPE” was matching with a lot of medical terms that we wanted to keep. In this step, a normalization of the keys was also performed, lowering all the letters of the terms. This process will be important for the following tasks of relations identification.

An attempt was made to remove singular-plural repetitions. For example, in the dataset, “bronchiole” and “bronchioles” exist as two separate terms, when they refer exactly to the same thing. As a strategy to get rid of this data repetition, a function was created that would merge the terms that had the same lemma, provided by spacy. However, this was aborted, since it was detected that, for example, the entry for the disease “aids” would be erased, by merging with its spacy lemma “aid”. Since more cases like this could be happening, we decided to not go ahead with this cleaning of the dataset.

Relations Identification

The first attempt of organizing the terms in categories was done by collecting the term topic from the Oxford Dictionary available online, for each available term of our database. However, the retrieved topics revealed to be very inaccurate and sometimes even contradicting. Therefore, this idea was abandoned after a few tries.

Two strategies were then performed to identify relations between terms. The final aim of this was to organize the dataset in a structure like the one of a graph, like shown in Figure 1.

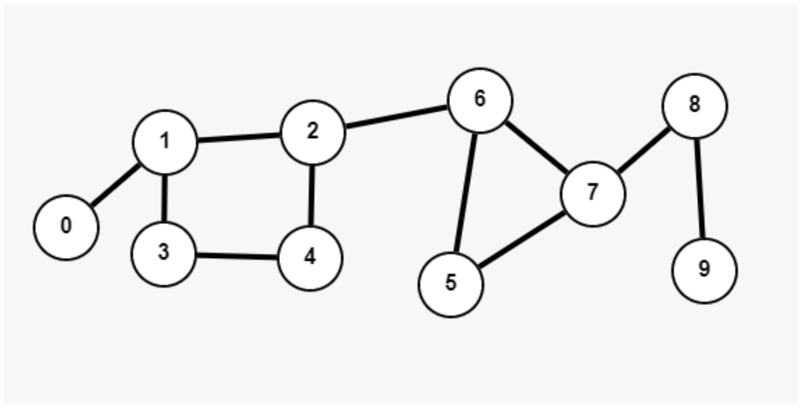

Figure 1 – Graph structure. Each point represents a term and the lines the existing relations between them.

This way, it would then be possible to identify the top "categories" of our dataset, being those the terms with the more relations associated to them.

Relations from terms

Terms that contain the same words, like "hafnia" and "hafnia alvei" must have a relation between them. Therefore, in our data structure it was important to relate them.

One of the strategies to find existing relations between terms was by comparing their keys. Remember that the keys of the dictionary are the terms in English. The process consisted in, for each term, tokenizing it on its words and verifying if each of them is a term existing in the dictionary. Additionally, adjacent combinations were also verified. This is, for example for the term “hand deformities, acquired” first every single word would be searched: “hand”, “deformities” and “acquired”, finding the relations to “hand”, “deformity” and “acquired” because those terms exist in the dictionary. The “deformity” relation is possible because we are using the spacy lemma to compare the single words. Next, “hand deformities” and “deformities acquired” are searched. The “hand deformities” is added to the relations because it exists as a term. Note that the relation is added on both ends. So, the “hand” term will have “hand deformities, acquired” added to its relations list and the term “hand deformities, acquired” has the term “hand” added to its relations list.

Relations from descriptions

It is very common to find existing term of our dictionary in the description of other terms. Once again, this means that there might be a relation between these terms.

This second strategy iterated over the descriptions text of each term, identifying all the nouns and adjectives using spacy. This is done so that now, only these words will be searched for in the dictionary keys. Since some descriptions are quite extensive, this process helps to reduce the computational task of comparing terms. If there is a match with a dictionary key, then the relations are added to both terms. For example, the description of the term “fungi” is “A kingdom of eukaryotic, heterotrophic organisms...”, which contains the term “eukaryotic”. As so, “eukariotic” is added to the relations list of “fungi”, and “fungi” is added to the relations list of “eukariotic”.

As you may notice, it is important that all the terms are lowered in the dictionary, so that the matches from the description words will always happen and compatibility problems will not arise.

Website

It was decided to reuse the website created in the first assignment, but with some adaptations and new features.

The search section was not altered, but the term list and the term pages were. The term page was updated to include all the possible elements that he has: the translations in Portuguese, Spanish and German, the descriptions in Portuguese and English, its relations to other terms on the database, and the possibility to delete him from the database by clicking a button.

In term list page, first it was made possible to add and delete terms. To add a term, there's a form, with the only required input being the term itself. Then, it is optional to input its translations and descriptions. The translations are achieved by the <input> element, but the descriptions are achieved by the <textarea> element. This is because the descriptions, in the database, are lists, so each line of the <textarea> is going to correspond to one element of that list (and for that, it is used the .split('\n') method). The relations are automatically calculated, reusing the code that was made to create them in the database.

Then, and because there are almost 47000 entries on the database, the page was changes so that only 100 entries were shown at a time, with buttons to move forwards and backwards on the list. Because moving 100 entries at a time was slow, especially for a database 47000 entries long, it was also added the option to search for an index, which makes it possible to go anywhere on the list (of course, this is serves more to move on the list than to search for a specific entry, but for that there's already the search page).

It was also created a new page, the categories page. This one was originally to show all categories of the database, and when once one of them was chosen, all the terms of that category. Since we didn't create a category attribute, we only display the most important terms, that could have been categories, like 'anatomy', 'bacteria', 'blood', and so on. When one is chosen, it links to the term page of that term, which shows all the relations that it has. Then, it was made possible to delete the category, which will only remove it from the category list, and not from the term list, and to add new categories, although it will only show up if there's a term that corresponds to it.

Future work

To increase the quality of our application, it could be important to remove the duplicate information contained in different similar-plural entries, as referred previously.

Also, a better categorization of the terms could improve the user experience. Maybe this can be achieved with a deeper language model or by further exploring the Oxford Dictionary.

Conclusion

A huge increase of the database was achieved, with the use of the tool BeautifulSoup learned during the NLP classes. The use of the language model Spacy was also very successful.

The goal of the categorization was compromised by the lack of accuracy of the Oxford Dictionary source. However, we managed to create our own strategy to achieve some sort of data organization, which leaded to a relations mapping between terms. This might not be the most user-friendly approach, but it is efficient.

The application created with Flask can present all the data in a more structured way than the previous assignment version. It is also possible to add and delete terms and “categories”. When a new term is added with a description, the relations to those term are instantly created, making the relations map of our database a dynamic structure.

In the end, despite the problems faced and the ideas that had to be abandoned, we are happy with the result, also because we were able to apply all the concepts learned during the PLN course.