Mestrado em Engenharia Biomédica – Informática Médica
1º Ano | 2022/2023 | 2º semestre

Processamento de inguagem Natural

Trabalho Prático de Grupo - 2

Ciarán McEvoy A87240
Gonçalo Carvalho PG50392

Introdução

Nos dias presentes, a maioria da informação disponível encontra-se em formato digital armazenados desde enciclopédias até *websites*. Tendo em conta a quantidade elevada de informação existente, torna-se assim importante ser capaz de encontrar e analisar a informação relevante. Assim sendo, quando se pretende criar um qualquer *website* a sua organização, e a da sua informação é fulcral.

Para este efeito, existem ferramentas que permitem retirar informação de páginas *web* mantendo apenas a informação informação relevante (*webs craping*), ser capaz de encontrar relações entre termos que possam ser importantes, e também organizá-los por categorias, assim como procurar, eliminar, editar e adicionar informação.

Web scraping

O primeiro passo neste trabalho passou pela realização de web scraping, ou seja, procurar e retirar a informação relevante de um dado site. Para este efeito, foram escolhidos 2 sites diferentes, um que continha uma lista com os ossos do corpo humano, e outro que continha doenças, sintomas, critérios de diagnóstico e tratamento.

Assim sendo, o processo de remoção para cada uma destas páginas foi diferente. No caso da lista de ossos, devido à simplicidade da página em questão. Primeiro, usou-se o *parser* e o link com a ajuda da biblioteca *BeautifulSoup*, para aceder à informação da página. Depois procedeu-se a encontrar a div que continha a informação relevante e simplesmente a iterar por esta até se encontrar a informação considerada relevante, como se demonstra na figura abaixo:

```
title = ""
for ele in soup:
    ul_elements = ele.find_all("ul")
    for ul in ul_elements:
        li_elements = ul.find_all("li")
        for li in li_elements:
            if li.text not in title:
                title = title + li.text + "\n"
```

Figura 1: Iterar por vários elementos *HTML* para remover informação

No fim, removeram-se porções de texto consideradas irrelevantes, para este ficar limpo.

Por outro lado, a página que continha a informação relativa a doenças provou-se algo mais complexa sendo que foi necessária uma perspetiva diferente. Para esse efeito foram retirados dois *urls*, um que continha a informação , como doenças de a-z e outro que era o url base do mdsaude. Retirou-se os termos de a-z disponíveis, bem como as respectivas páginas de informação acerca da doença. Das páginas de informação, retirou-se os headers e os parágrafos abaixo de cada um. Foi feito uso da biblioteca BeautifulSoup e do Regex para extrair o conteúdo desejado. Para a busca dos termos e das páginas html de cada termo primeiro foi feito um find_all das *div* com class=”entry-content”. Com essa *div* em posse, procedeu-se ao uso do regex para isolar a referência para a página de cada termo concatenando com o url base de modo a dar o link completo do termo, e por fim foi retirado o nome do termo em si para guardar em uma lista. Para a extração da informação geral das páginas dos termos em si, foi efetuado um .find(“div”, class_=”entry-content”). Para retirar todos os headings foi feito um .find_all(“h2). De modo a combinar os parâmetros certos com os headings certos, foi necessário recorrer ao regex em que de seguida a haver um heading, retiraram-se todos os parágrafos até ao próximo heading, sendo este processo repetido até a informação toda ser extraída.

Deteção de padrões

A deteção de padrões, nomeadamente entre palavras com similaridade elevada pode ser bastante útil para assegurar que conseguimos ter uma lista de termos semelhantes e que possam conter informação adicional relevante. Por outro lado também podemos conseguir encontrar sinónimos com estes métodos.

Para isto foi usada a biblioteca *Spacy*, com o objetivo de encontrar estas relações através da similaridade. Esta função vetorial vai ter em conta o contexto e o uso do termo para calcular esta similaridade.

Para isto são abertos dois ficheiros (ambos dicionários), o ficheiro *mdsaude.json* que foi criado anteriormente e o Final.json, que contém a informação retirada do trabalho anterior, e que terão os seus termos comparados entre si para este efeito.

Para este efeito, percorremos todas as chaves primeiramente do mdsaude, e verificando se qualquer uma destas chaves tem um factor de similaridade superior a 0.8

quando comparada com qualquer chave presente no ficheiro Final.json. Caso isto se verifique, estas chaves serão adicionadas ao termo num campo extra denominado "relações_sim".

Por fim também iremos comparar as chaves de mdsaude com as suas próprias chaves, seguindo os passos anteriores.

No fim disto, iremos guardar o nosso novo dicionário noutro ficheiro json.

Criação de categorias

Ao nosso ficheiro foram também adicionadas categorias. Para a criação destas, abrimos o ficheiro criado anteriormente, assim como os ficheiros Ossos.json (onde está a lista de ossos) e mdsaude.json. Iremos percorrer cada um destes dois ficheiros em combinação com o primeiro, e em cada iteração se encontrarmos o nome da mesma doença ou osso no nosso dicionário como *key*, ser-lhe-á adicionada uma nova entrada no formato "Categoria:" seguido de ou Osso, ou Doenca dependendo de se foi no ciclo dependente de mdsaude ou de Ossos.

Será de referir que no caso do ciclo que contém informação relacionada com os ossos, é utilizada a biblioteca *Spacy* para conseguir normalizar os termos. Isto ocorre porque em diversos casos, as duas palavras podem ser iguais excetuando a presença de acentos, e com a normalização conseguimos ultrapassar este problema.

Pesquisa de Sintomas

O algoritmo de procura de sintomas inclui um bocado de cada biblioteca. Este algoritmo está definido na própria rota do *flask.* Começa por normalizar o texto que o utilizador introduz através do *lemma.* Este texto normalizado depois percorre todos os textos dos termos que têm a secção de sintomas. Os textos dos sintomas também são normalizados antes de serem comparados. Duas condições são observadas para devolver um termo médico, sendo eles, que o score da similaridade entre os tokens do texto introduzido pelo utilizador e aquele dos sintomas tem que ser acima de 0.65, adicionalmente que o input normalizado faça match no texto normalizado dos sintomas. Apenas um destes dois tem que ser verdadeiro para que o termo seja devolvido. Observou-se que o similarity funcionou melhor quando o utilizador introduziu textos maiores dos possíveis sintomas que podia ter.

Website

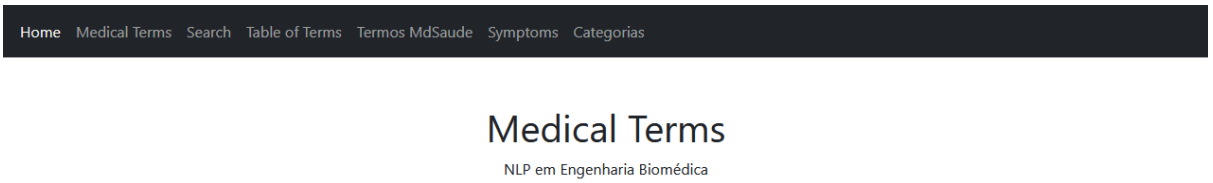

Figura 2: Página Inicial

O website está dividido em 6 categorias principais estando elas representadas na figura acima no header.

A primeira categoria, os medical terms, são todos os termos médicos acumulados até agora ordenados em lista. Cada termo é um hyperlink sendo possível carregar em cada um e aceder a página de informações acerca do respetivo. Na página dos termos médicos é também possível apagar e adicionar termos.

O *Search* permite ao utilizador introduzir um input, e receber todos os termos parecidos com esse input.

Designation	ENG	ES
abdução	abduction	abducción
abdómen	belly, abdomen	abdomen

Figura 3: Tabela dos termos com traduções

A página *Table of Terms* como vista na figura acima, tem uma tabela com os termos todos organizados em uma tabela com as suas respectivas traduções. Essas traduções incluem inglês e espanhol.

A seção dos *Termos MDSaude,* apresenta uma lista dos termos recolhidos do website MDSaude, sendo possível carregar em cada um dos termos e obter os dados acerca de cada termo de forma detalhada. Dentro da página de cada termo é também possível encontrar termos semelhantes a esse que possam existir nos acumulados.

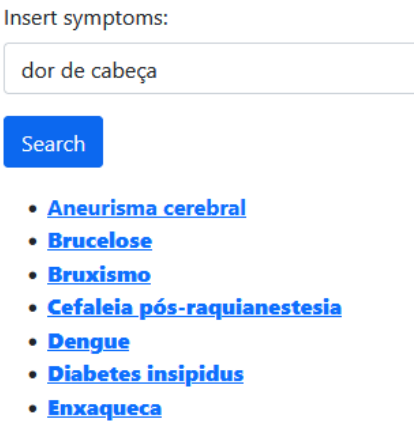

Figura 4: Sintomas de “dor de cabeça”

Na página *Symptoms,* o utilizador consegue introduzir sintomas que possa ter, e receber possíveis doenças, com as suas respectivas páginas detalhadas a explicar a doença. As doenças que podem ser obtidas vêm dos dados recolhidos do MDSaude.

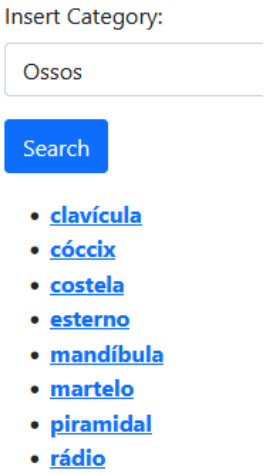

Figura 5: Categoria “Ossos” com resultados

Por fim na página das *Categorias,* novamente, o utilizador consegue fazer um input de uma categoria de termos médicos que possa querer, e receber uma lista de termos relacionados. Para este caso apenas foram criadas duas categorias, sendo ela “Ossos” ou “Doenças”.

Conclusões

A aplicação *flask* desenvolvida foi capaz de cumprir os requisitos desejados, desde a adição, edição, eliminação e procura de termos, até a organização por categorias, sendo também possível até a procura de uma doença tendo em conta os sintomas que o utilizador apresenta.

No entanto, tal como em qualquer aplicação, existem pontos que poderiam ser melhorados. Nomeadamente, a quantidade de categorias ainda se encontra bastante reduzida, sendo que foram usadas como *proof-of-concept* para este trabalho. Isto poderia ser resolvido procedendo à extração de informação de mais sites diferentes.

Por outro lado, seria também útil uma maior relação entre as diversas categorias, podendo ser interessante o uso de similaridade entre termos e as suas respectivas designações.

Será também de referir que apesar do funcionamento da procura de uma doença tendo em conta os seus sintomas, esta pesquisa é ainda algo rudimentar, sendo que os sintomas procurados têm de ser escritos de formas específicas para assegurar que a relação de similaridade será capaz de os encontrar.

Outra possível melhoria seria a utilização de outro tipo de algoritmos de relacionamento de termos, como por exemplo, efetuar uma *tokenização* diferente.

Apesar destas possíveis falhas, o funcionamento das funcionalidades básicas deste tipo de aplicações está assegurada, o que nos permite afirmar que a aplicação funciona devidamente.