UNIVERSIDADE DO MINHO

Trabalho Prático 1

Elaborado no âmbito da Unidade Curricular de
Processamento de Linguagem Natural em Engenharia Biomédica

Grupo
André Sousa (PG52564)
Beatriz Macedo (PG55820)
Maria Vale (PG56144)

Docentes
José João Antunes Guimarães Dias Almeida
Luís Filipe Costa Cunha

abril de 2025

Índice

1 Introdução

O presente trabalho foi desenvolvido no âmbito da Unidade Curricular de Processamento de Linguagem Natural em Engenharia Biomédica e o seu objetivo consiste na aplicação dos conhecimentos adquiridos durante as aulas acerca do processamento de documentos e extração de informação dos mesmos.

Deste modo, este projeto consiste na extração de informação de documentos com vários tipos de informação médica. Os documentos foram processados, com o auxílio de expressões regulares. Posteriormente, essa informação foi armazenada em documentos de formato JSON.

2 Enquadramento

O processamento de linguagem natural (*Natural Language Processing - NLP*) é um campo da inteligência artificial que permite extrair vários tipos de informação de textos não estruturados.

As espressões regulares (também conhecidas como *regex*) têm-se mostrado fundamentais e eficazes no pré-processamento de textos e, por isso, úteis para o processamento de linguagem natural. O facto de encontrarem e extraírem padrões específicos em textos é uma mais valia para a preparação e limpeza de documentos.

3 Metodologias, Materiais e Métodos

Durante a realização deste trabalho foram utilizados diferentes materiais e métodos para atingir o objetivo pretendido.

Primeiramente, foi necessário definir os documentos que seriam utilizados durante o projeto. Para além do documento obrigatórios ("diccionari-multilinguee-de-la-covid-19.pdf" e "glossario_ neologismos_saude.pdf"), escolheram-se os documentos "medicina.pdf" e "glossario_ministerio_saude.pdf". Uma vez que os documentos obrigatórios contêm termos médicos e respetiva definição, achou-se pertinente explorar documentos que fossem constituídos por conceitos e respetivas traduções.

Uma vez que se trata de documentos em formato PDF, após a sua escolha, foi necessário convertê-los para documentos de texto. Assim, utilizou-se o comando *pdftohtml -xml* que permite a conversão dos documentos para um formato HTML e, posteriormente, em XML. Esta conversão foi fundamental para manter a estrutura e conteúdo do PDF e, ao mesmo tempo, facilitar a análise e extração de informações importantes presentes no documento.

A extração de informação, por sua vez, foi possível com recurso à linguagem *Python* e às bibliotecas *re* e *json*. A primeira biblioteca permite a utilização de expressões regulares para encontrar e alterar informação nos documentos XML. Por outro lado, a biblioteca *json* permite estruturar a informação extraída em formato JSON.

4 Implementação

Para alcançar os objetivos propostos neste estudo, foi realizada uma análise minuciosa de todos os documentos, tanto nas suas versões originais quanto após o processo de conversão. Esta etapa teve como finalidade compreender como os dados estavam organizados. Com

base nessa compreensão, os ficheiros txt e XML gerados pela conversão foram tratados utilizando expressões regulares, permitindo simultaneamente uma verificação da qualidade da informação extraída e do próprio processamento.

Nas próximas secções, serão detalhadas as opções metodológicas adotadas durante o tratamento dos diferentes ficheiros, bem como as abordagens utilizadas para localizar e extrair os conteúdos mais relevantes de cada um.

5 diccionari-multilinguee-de-la-covid-19.pdf

5.1 Análise do documento

Após a análise detalhada do documento, verificou-se que o mesmo apresenta 2 secções com informação bastante relevante que deveria ser extraída:

- **Abreviações** (página 27): Esta secção contém categorias de abreviações linguísticas e técnicas, estruturadas em quatro subseções:

  Categories lèxiques
  Abreviações gramaticais (n, adj, v tr, etc.)

  Indicadors de llengua
  Códigos de idiomas (pt, es, en, etc.)

  Altres codis
  Outras marcações linguísticas (sin., den. com., etc.)

  Remissions
  Referências cruzadas (sbl, nc, CAS)

Após esta análise do documento em formato PDF, foi também necessário observar como estava estruturado o documento XML obtido após a conversão, assim como o documento txt.

Observou-se que no documento XML alguns conceitos das abreviaturas encontravam-se na mesma linha que as suas definições. Esta ocorrência era comum para abreviações que apresentavam como conceito apenas uma letra ou palavra, como é o caso da primeira abreviação: *n nom*.

Assim sendo, no caso das abreviações, utilizou-se o documento txt para retirar os termos e os seus significados, dado que nesse formato as abreviações estavam bem divididas entre linhas.

No entanto, a ordem das abreviações e das classes correspondentes estavam desorganizadas, intercalando abreviações de classes diferentes. Foi preciso implementar um algoritmo para a sua organização.

- **Dicionário multilíngue** (páginas 30 a 180): Contém termos relacionados com a COVID-19 em múltiplos idiomas, estruturados da seguinte forma:

Número de Ordem
Denominação Catalã
Categoria léxica
Sinónimos complementares
Tradução
CAS
Área Temática
Definição
Nota

No Dicionário Multilingue, através da análise do seu documento XML, geraram-se regras *regex* que conseguiram organizar os conceitos na estrutura solicitada.

5.2 Processamento do documento

5.2.1 Abreviações

Como referido anteriormente, no caso das abreviações utilizou-se o ficheiro de extensão .txt para extrair os termos e o seu significado. Desse modo, o processamento foi a conversão de pdf para o ficheiro txt e posterior extração dos termos, com o tratamento de ignorar títulos e categorias indesejadas.

Assim, para a organização das abreviações por categorias, implementou-se um algoritmo que ordenou os termos e os seus significados, dado que seguiam um padrão de estarem associados de forma intercalada às categorias por ordem de ocorrência. Tratou-se ainda de uma exceção representada pela abreviação "pt" que obtinha mais que um significado, optou-se por criar uma lista para essa mesma chave, que representasse todos os possíveis significados dessa abreviação.

```python
# Distribuir os primeiros 21 pares (12+9) de forma intercalada
for i in range(22):
    abrev, desc = pares[i]
    if i % 2 == 0:
        destino = secao1
    else:
        destino = secao2

    if abrev == "pt":
        if "pt" not in resultado[secao2]:
            resultado[secao2]["pt"] = []
        resultado[secao2]["pt"].append(desc)
    else:
        resultado[destino][abrev] = desc

# A partir do par 21, alternar entre Altres codis e Remissions
for j, (abrev, desc) in enumerate(pares[22:]):
    destino = secao3 if j % 2 == 0 else secao4
    resultado[destino][abrev] = desc
```

Fig. 1: Código para a organização das abreviações por categorias.

5.2.2 Conceitos

Para a extração dos conceitos do dicionário analisou-se o ficheiro XML e tratou-se o mesmo. A limpeza teve como passos a remoção de cabeçalhos, tags irrelevantes, conteúdos vazios e números de páginas. Mais tarde, com as primeiras extrações fomos adicionando remoções e limpezas ao XML para melhorar a nossa extração. Foi importante neste processamento manter informações sobre tipo de fonte, dado que auxiliou na posterior limitação de termos e significados.

A maior dificuldade foi encontrar regras *regex* capazes de averiguar todos os conceitos e as suas particularidades.

```
# Limpeza geral de cabealhos, marcas e estrutura
doc = re.sub(r'.+page.+\n?', "", doc)
doc = re.sub(r'<\?xml.*?\?>\s*<!DOCTYPE pdf2xml[^>]*>\s*', '', doc, flags=re.
    DOTALL)
doc = re.sub(r'<fontspec[^>]*/>', '', doc)
doc = re.sub(r'</?pdf2xml[^>]*>', '', doc)
doc = re.sub(r'\n\s*\n', '\n', doc)
doc = re.sub(r'<page number[\w\W]+?<text>\d+</text>\n', "", doc)
doc = re.sub(r"</page>\n", "", doc)

# Simplificar tag <text> para manter s a fonte
doc = re.sub(r'<text[^>]*?font="([^"]+)"[^>]*?>(.*?)</text>', r'<text font
    ="\1">\2</text>', doc)

```

```
# Remover tags <text> com contedo vazio ou espaos
doc = re.sub(r'<text font="\d+">[\s\u00A0]*</text>\n?', '', doc)

# Remover <text> com apenas pontuao (ex: ; , .  ! ? etc.)
doc = re.sub(r'<text font="\d+">[;.,!?]+\s*</text>\n?', '', doc)

# Remover <text><b> </b></text> e <text><i> </i></text> (tags bold/italic
    vazias)
doc = re.sub(r'<text font="\d+"><[bi]>\s*</[bi]></text>\n?', '', doc)

# Remover textos com nmeros de pgina (em font 0 ou outras variantes)
doc = re.sub(r'<text font="0">.*?\d+.*?</text>\n?', '', doc)

# Remover a letra indicadora do dicionrio, como <text font="15">A</text>
doc = re.sub(r'<text font="15">[A-Z-]</text>\n?', '', doc)

# Remover o cabealho com "QUADERNS 50"
doc = re.sub(r'<text font="3"><b>QUADERNS 50\s*</b></text>\n?', '', doc)

# Remover o subttulo "DICCIONARI MULTILINGE DE LA COVID-19"
doc = re.sub(r'<text font="4">\s*DICCIONARI MULTILINGE DE LA COVID-19\s*</text
    >\n?', '', doc)
```

5.3 Estrutura JSON

5.3.1 Abreviações

No que diz respeito às abreviações, optou-se pela seguinte estrutura JSON:

```
{
    "Categories lxiques": {
        "n": "nom",
        "n pl": "nom plural",
        ...
    },
    "Indicadors de llengua": {
        "oc": "occit",
        "eu": "basc",
       ...
    },
    "Altres codis": {
        "v intr": "verb intransitiu",
        ...
    },
    "Remissions": {
        "sbl": "smbol",
        ...
    }
}
```

Nas figura seguinte pode verificar-se um excerto do documentos JSON obtidos relativo à estrutura anterior

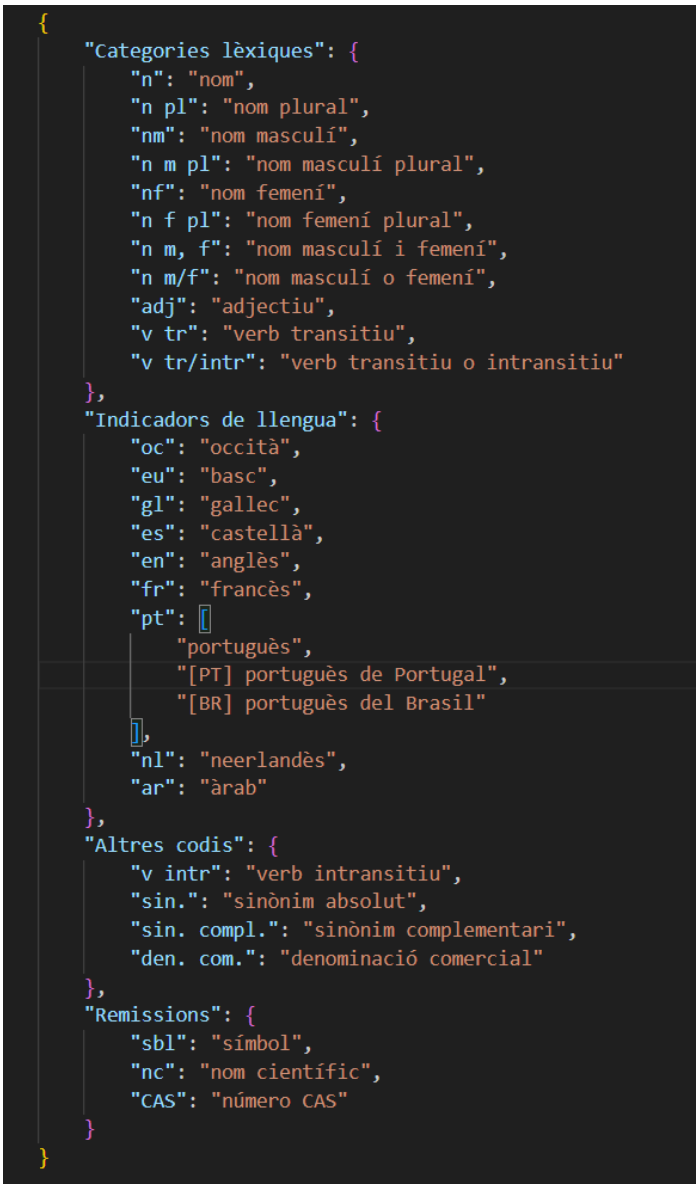

Fig. 2: Documento JSON com as abreviações do Dicionário Multilingue.

5.3.2 Conceitos

No que diz respeito aos conceitos, optou-se pela seguinte estrutura JSON:

```
{
  "id":
  "denominacao_catala":
  "categoria_lexica":
  "sinonimos_complementares":
  "traducao":
  "cas":
  "area_tematica":
  "definicao":
  "nota":
},
```

Na figura seguinte pode verificar-se um excerto do documento JSON obtido.

Fig. 3: Estrutura json de um Conceito do Dicionário Multilingue.

6 glossario_neologismos_saude.pdf

6.1 Análise do documento

Após a análise detalhada do documento "neologismos.pdf", verificou-se que o mesmo apresenta várias secções com informações relevantes que deveriam ser extraídas. As áreas consideradas relevantes foram:

- **Abreviaturas Utilizadas:** Esta secção lista as abreviaturas e siglas utilizadas ao longo do documento. É essencial para a compreensão dos termos abreviados que aparecem no glossário e em outras partes do documento. A estrutura é simples, apresentando a abreviatura seguida de seu significado.
- **Lista de Abreviaturas e Siglas:** Similar à secção anterior, esta lista fornece uma visão geral das abreviaturas e siglas utilizadas no documento. A lista é organizada alfabeticamente e inclui termos como "FAPESP" (Fundação de Amparo à Pesquisa do Estado de São Paulo) e "GNTSH" (Glossário de Neologismos Terminológicos da Saúde Humana).
- **Glossário:** Esta é a secção principal do documento, contendo os neologismos terminológicos da saúde humana. Cada entrada no glossário é estruturada da seguinte forma:
  -**Termo:** O termo neológico.
  -**Referências Gramaticais:** Indicando o substantivo do termo.
  -**Equivalências/Traduções:** Traduções do termo para o inglês [ing] e espanhol [esp].
  -**Definição/descrição:** Explicação do termo.
  -**Informação Enciclopédica:** Informações adicionais relevantes sobre o termo.
  -**Citação:** Contexto em que o termo é utilizado.
  -**Número de Identificação:** Referência ao número do artigo de onde o termo foi extraído.
- **Equivalências Inglês-Português:** Esta secção apresenta uma lista de termos em inglês com as suas respetivas traduções para o português. É útil para leitores que precisam entender os termos em ambos os idiomas.
- **Equivalências Espanhol-Português:** Similar à secção anterior, esta lista fornece as traduções dos termos do espanhol para o português. Facilita a compreensão dos termos para falantes de espanhol.
- **Anexos:** Os anexos incluem uma tabela com os títulos dos artigos revistos para a realização deste documento, dos quais foram extraídos os neologismos. Esta secção é importante para a rastreabilidade e verificação das fontes dos termos.

6.2 Exceções às regras

Foi possível reparar em algumas exceções às regras apresentadas anteriormente, que poderiam precisar de uma atenção especial, nomeadamente:

- **Glossário:**

  -Traduções que não apresentavam informação sobre a linguagem da tradução, não tinha presente o “[ing]” ou “[esp]”;
  -Erros de escrita em “[esp]”, onde aparecia só “[es”;
  -Quebras de linha a meio das traduções;

-Traduções apareciam geralmente em apenas uma linha e sempre com o mesmo formato de letra, contudo para o termo “coréia de sydenham” isto não se verificou. Neste caso é utilizado um estilo de letra não coerente com o resto do documento;

-Presença de notas, após as citações em casos esporádicos, como no caso do termo “biopsia óptica”.

- **Equivalências Inglês-Português**
  - Linhas mistas com os dois idiomas na mesma linha, isto é, algumas entradas contém o termo em inglês e a respetiva tradução portuguesa na mesma linha, separados por múltiplos espaços.
  - Certas traduções em português encontram-se repartidas por múltiplas linhas, o que exige a sua junção.
- **Equivalências Espanhol-Português**
  - Agrupamento de múltiplos termos numa mesma coluna, por vezes surgem dois ou mais termos espanhois ou portugueses seguidos, sem separação por linhas, dificultando a associação correta dos pares.
  - Traduções longas em português distribuídas por várias linhas, as traduções podem estar segmentadas em várias linhas consecutivas, exigindo concatenação.

6.3 Processamento do documento

Após identificar-se as exceções presentes no documento, foi possível passar ao processamento dos dados. Para facilitar este trabalho, cada uma das secções consideradas relevantes foi processada independentemente:

6.3.1 Abreviaturas Utilizadas

- **Leitura do Arquivo XML:** Primeiramente, o código define o caminho para o arquivo XML denominado `"neologismos.xml"` e abre-o em modo de leitura com a codificação UTF-8, para garantir que qualquer caracter especial presente no arquivo seja lido corretamente. O conteúdo do arquivo é armazenado na variável `'xml_data'`.

```
    # Caminho para o arquivo XML
    doc_text_path = "neologismos.xml"

    # Ler o arquivo XML
    with open(doc_text_path, 'r', encoding="utf-8") as file:
        xml_data = file.read()

```

- **Extração do Conteúdo da Página 10:** Em seguida, é utilizada uma expressão regular para extrair o conteúdo específico da página 10 do arquivo XML. Esta expressão regular procura a tag `<page>` com o atributo `number="10"` e captura todo o conteúdo entre `<page>` e `</page>`. A flag `re.DOTALL` é utilizada para permitir que o ponto (`.`) da expressão regular também corresponda a quebras de linha. Caso o conteúdo da página 10 não seja encontrado, o código imprime uma mensagem de erro e interrompe a execução. Caso contrário, o conteúdo completo da página 10 é armazenado na variável `pagina_10`.

```
    # 1. Extrair apenas o contedo da pagina 10 usando regex
```

```
    pagina_10_pattern = re.compile(
        r'<page number="10".*?>.*?</page>',
        re.DOTALL  # Permite que . corresponda a quebras de linha tambm
    )

    pagina_10_match = pagina_10_pattern.search(xml_data)
    if not pagina_10_match:
        print("Pagina 10 no encontrada no arquivo XML.")
        exit()

    pagina_10 = pagina_10_match.group(0)

```

- **Extração das Tags `<text>` da Página 10:** Com o conteúdo da página 10, o código procura todas as tags `<text>` dentro da página, utilizando uma expressão regular. A expressão definida extrai o conteúdo de cada tag `<text>`, ignorando os atributos das tags. A flag `re.DOTALL` é novamente utilizada para garantir que quebras de linha dentro das tags sejam capturadas corretamente.

```
    # 2. Extrair todas as tags <text> da pgina 10
    text_tags_pattern = re.compile(
        r'<text[^>]*>(.*?)</text>',
        re.DOTALL
    )

```

O código itera sobre cada ocorrência de tag `<text>` e extrai o seu conteúdo.

- **Identificação de Siglas e Significados:** Para identificar siglas e os seus respetivos significados, o código define uma expressão regular específica. Esta regex procura sequências de letras maiúsculas, números e símbolos como `&` e `.`, seguidos por um separador (como –, -, ou :) e o significado da sigla. A flag `re.IGNORECASE` permite que o padrão seja insensível a maiúsculas e minúsculas, e a flag `re.UNICODE` assegura que caracteres especiais sejam reconhecidos corretamente.

```
    # 3. Padrao para identificar siglas e significados
    sigla_pattern = re.compile(
        r'^([A-Z0-9&][A-Z0-9\s&\.]+[A-Z0-9]|et\s+al\.)\s*[\-:]\s*(.+)$',
        re.IGNORECASE | re.UNICODE
    )

```

Assim, o código verifica cada conteúdo extraído das tags `<text>` e, se corresponder ao padrão de sigla, extrai a sigla e o seu significado, armazenando-os num dicionário denominado `abreviaturas`.

```
    abreviaturas = {}

    for text_tag in text_tags_pattern.finditer(pagina_10):
        conteudo = text_tag.group(1).strip()

    # Remover tags HTML/bold se existirem
```

```
    conteudo_limpo = re.sub(r'<[^>]+>', '', conteudo).strip()

    # Verificar se  uma linha de sigla
    match = sigla_pattern.match(conteudo_limpo)
    if match:
        sigla = match.group(1).strip()
        significado = match.group(2).strip()
        abreviaturas[sigla] = significado

```

Antes de verificar se o conteúdo corresponde a uma sigla, o texto extraído das tags `<text>` é limpo, removendo quaisquer tags HTML (como `<b>` para negrito) com a expressão regular:

```
    r'<[^>]+>'

```

Isso garante que apenas o texto seja analisado para a detecção de siglas. O conteúdo limpo é então verificado para identificar se contém uma sigla e, se for o caso, a sigla e o seu significado são extraídos e armazenados no dicionário `abreviaturas`.

- **Ordenação das Siglas:** Após extrair todas as siglas e os seus significados, o código organiza o dicionário de siglas alfabeticamente, utilizando a função `sorted()`. O dicionário de siglas ordenadas é armazenado em `abreviaturas_ordenadas`.

```
    # Ordenar as siglas alfabeticamente
    abreviaturas_ordenadas = dict(sorted(abreviaturas.items()))

```

- **Criação de um Ficheiro JSON para Guardar as Siglas:** É criado um arquivo JSON denominado `"abreviaturas_page10.json"`, onde são guardadas as siglas ordenadas. A função `json.dump()` é utilizada para gravar o dicionário `abreviaturas_ordenadas` no arquivo.

```
    # Guardar em ficheiro JSON
    output_path = "abreviaturas_page10.json"
    with open(output_path, 'w', encoding='utf-8') as json_file:
        json.dump(abreviaturas_ordenadas, json_file, indent=2, ensure_ascii=
    False)

```

Este processo permitiu a extração e organização dos dados das abreviaturas de forma estruturada, facilitando a análise e utilização das informações. Cada etapa foi cuidadosamente planejada e executada para garantir a precisão e a integridade das informações extraídas do documento "neologismos.pdf".

6.3.2 Lista de Abreviaturas e Siglas

- **Leitura do Arquivo XML:** Primeiramente e novamente é aberto o arquivo XML denominado “neologismos.xml”. O conteúdo completo do arquivo é armazenado na variável xml_data, permitindo que seja manipulado nas etapas seguintes.
- **Extração do Conteúdo da Página 86:** O código utiliza uma expressão regular para localizar e extrair o conteúdo específico da página 86 dentro do ficheiro XML. Esta expressão regular

encontra a tag <page> com o atributo number="86" e captura todo o conteúdo entre <page> e </page>. A flag re.DOTALL é utilizada para permitir que o ponto (.) capture quebras de linha, garantindo que todo o conteúdo da página, mesmo distribuído em várias linhas, seja extraído corretamente. Se o conteúdo da página 86 não for encontrado, o código imprime uma mensagem de erro e encerra a execução. Caso contrário, o conteúdo da página 86 é armazenado na variável pagina_86.

```
# 1. Extrair apenas o contedo da pgina 86
pagina_86_pattern = re.compile(
    r'<page number="86".*?>.*?</page>',
    re.DOTALL  # Permite que . corresponda a quebras de linha tambm,
para consiguir capturar vrias linhas de contedo entre <page> e </page>
)

pagina_86_match = pagina_86_pattern.search(xml_data)
if not pagina_86_match:
    print("Pgina 86 no encontrada no arquivo XML.")
    exit()

pagina_86 = pagina_86_match.group(0)

```

- **Extração das Tags <text> da Página 86:** Depois do conteúdo da página 86 ser extraído, o código procura todas as tags <text> presentes nessa página, através de uma expressão regular. Esta regex extrai três partes principais de cada tag <text>, são elas a posição top, que representa a posição vertical da tag, a posição left, que representa a posição horizontal e o conteúdo da tag, que é o texto dentro de <text></text>. A flag re.DOTALL é novamente usada para garantir que quebras de linha dentro das tags sejam capturadas.

```
# 2. Extrair todas as tags <text> da pgina 86
text_tags_pattern = re.compile(
    r'<text top="(\d+)" left="(\d+)"[^>]*>(.*?)</text>',
    re.DOTALL
)
```

- **Processamento das Tags para Identificação de Abreviaturas e Significados:** O código percorre cada uma das tags <text> extraídas da página 86. Para cada tag, ele processa o conteúdo da seguinte forma:
  - **Extração e Limpeza do Conteúdo:** O conteúdo da tag é extraído e qualquer tag HTML dentro dele é removida usando a expressão regular seguinte que limpa o conteúdo de qualquer formatação HTML:

```
re.sub(r'<[^>]+>', '', match.group(3))
```

  - **Identificação da Abreviatura:** Se o conteúdo estiver localizado na coluna da esquerda (onde left="128" ou valor similar), o código entende que este conteúdo é uma sigla (abreviatura). A variável current_top é configurada para armazenar a posição vertical (top) da sigla, e a sigla é armazenada na variável abreviatura.
  - **Identificação do Significado:** Se o conteúdo estiver na coluna da direita (com left="181" ou left="234"), e o valor de top for o mesmo da sigla (indicando que a sigla e o significado estão na mesma linha), o código assume que o conteúdo é o significado da sigla. Quando a

sigla é identificada e existe um significado para ela, ambos são armazenados no dicionário abreviaturas como chave (sigla) e valor (significado).
- **Controlo de Fluxo entre Sigla e Significado:** Quando o significado é identificado, a sigla e o significado são adicionados ao dicionário abreviaturas, e a variável abreviatura é limpa, pronta para identificar a próxima sigla.

```
    # 3. Processar as tags para encontrar abreviaturas e significados
    abreviaturas = {}
    current_top = None
    abreviatura = None

    for match in text_tags_pattern.finditer(pagina_86): #percorre todas as
    tags <text> da pgina 86
        top = match.group(1)
        left = match.group(2)
        content = re.sub(r'<[^>]+>', '', match.group(3)).strip()

        # Se estiver na coluna da esquerda (left="128" ou prximo)
        if left == "128" and content and not content.isspace():
            current_top = top
            abreviatura = content #sigla est em left == "128"
        # Se estiver na coluna da direita (left="234" ou prximo) e tivermos
    uma abreviatura no mesmo top
        elif (left == "181" or left == "234") and current_top == top and
    abreviatura: #top igual se est na mesma linha, pq a sigla e o
    significado tem o mesmo top, apenas left diferente
            if content and not content.isspace(): #se a sigla j estiver
    guardada, guarda o par no dicionario #not content.isspace() - verifica
    se nao esta vazio
                abreviaturas[abreviatura] = content
                abreviatura = None
```

- **Ordenação das Abreviaturas:** Após a extração e o armazenamento das siglas e significados, o código organiza o dicionário abreviaturas de forma alfabética, utilizando a função sorted(). O dicionário resultante, agora ordenado, é armazenado em abreviaturas_ordenadas.

```

    # Ordenar as abreviaturas alfabeticamente
    abreviaturas_ordenadas = dict(sorted(abreviaturas.items()))
```

- **Criação de um ficheiro JSON para guardar as siglas:** O código guarda as abreviaturas e os seus significados num arquivo JSON designado "abreviaturas_page86.json", utilizando a função json.dump().

```

    # Guardar em ficheiro JSON
    output_path = "abreviaturas_page86.json"
    with open(output_path, 'w', encoding='utf-8') as json_file:
        json.dump(abreviaturas_ordenadas, json_file, indent=2, ensure_ascii=
    False)
```

Este processo permitiu a extração e organização dos dados das abreviaturas e siglas de forma estruturada, facilitando a análise e utilização das informações. Cada etapa foi cuidadosamente planejada e executada para garantir a precisão e a integridade das informações extraídas do documento "neologismos.pdf".

6.3.3 Glossário

- **Leitura do XML como Texto:** O primeiro passo foi ler o documento XML como texto. Isso permitiu a manipulação das informações contidas nele de forma mais flexível. A leitura do XML como texto bruto é essencial para garantir que todas as informações, incluindo aquelas dentro das tags, sejam capturadas para processamento posterior.

```
# 1) Ler o XML como texto
with open('neologismos_glossario.xml', encoding='utf8') as f:
    raw = f.read()

```

- **Extração do Conteúdo entre as Tags `<text>...</text>`:** Utilizou-se uma expressão regular para extrair o conteúdo que está entre as tags `<text>`. Esta etapa foi fundamental porque as informações relevantes, como os termos do glossário e suas definições, estão encapsuladas dentro dessas tags. A extração limpa destas informações facilita a remoção das tags que foram consideradas desnecessárias para o processamento do glossário, visto não ser possível distinguir os diferentes elementos de um termo pelas informações das tags, sendo apenas possível pela utilização de expressões regulares no conteúdo textual entre as tags.

```
# 2) Extrai o contedo que est entre os <text>...</text>
raw_lines = re.findall(r'<text[^>]*>(.*?)</text>', raw, re.DOTALL)

```

- **Limpeza das tags, descodificação dos termos, remoção de acentos e espaços e correção de etiquetas de tradução mal formatadas:** As restantes tags HTML foram removidas, os caracteres especiais foram descodificados, e os acentos e espaços foram removidos. Este passo é importante para normalizar os dados, garantindo que todos os termos e definições estejam num formato consistente e livre de caracteres indesejados que possam interferir na análise e respetivo processamento. Por fim corrige etiquetas de tradução como “[es” que contém erros.

```
# 3) Limpa tags, decodifica termos, limpa acentos e espaos
clean_lines = []
for frag in raw_lines:
    txt = re.sub(r'<[^>]+>', '', frag)      # remove tags <b>s e <i>s
    txt = html.unescape(txt)                # converte &#34; em ' " '
    txt = unidecode(txt).strip()            # tira os acentos e os espaos
    if txt:
        clean_lines.append(txt)


# Corrige etiquetas [esp] incompletas ou mal formadas
clean_lines = [re.sub(r'\[esp?\.?\]?', '[esp]', line, flags=re.
IGNORECASE) for line in clean_lines]

```

- **Junção do Termo + Substantivo na Mesma Linha:** Para facilitar a identificação do início de um novo conceito, os termos foram unidos aos seus respectivos substantivos. Isso foi feito porque os substantivos têm um formato consistente (como `s.m.` ou `s.f.`), o que permite identificar facilmente o início de um novo termo no glossário. Esta junção simplifica a estrutura dos dados e facilita a extração de informações específicas.

```
    # 4) Junta o termo + substantivo (ex: 'abeta' + 's.f.') na mesma linha
    #isto facilita a identificao do incio de um novo conceito, graas ao
    substantivo ter sempre o mesmo formato s.[mf].
    merged = []
    i = 0
    while i < len(clean_lines):
       if i + 1 < len(clean_lines) and re.match(r'^(s\.[mf]\.)$',
    clean_lines[i+1], re.IGNORECASE):
            merged.append(f"{clean_lines[i]} {clean_lines[i+1]}")
            i += 2
       else:
            merged.append(clean_lines[i])
            i += 1
    lines = merged
```

- **Junção das Traduções Inglês + Marcador + Espanhol em uma Linha:** As traduções para o inglês e espanhol foram unidas numa única linha. Isto foi necessário, pois, em alguns casos, as traduções estavam distribuídas em várias linhas, o que causava problemas no processo de identificação e extração das traduções corretas.

```
    # 5) Junta tradues ingls + marcador + espanhol em apenas 1 linha.
    # Tive de fazer isto por causa do sydenham que apresentava as tradues ao
     longo de 3 linhas diferentes
    entry_start_re = re.compile(r'^(.+?)\s+(s\.[mf]\.)$', re.IGNORECASE)
    normalized = []
    j = 0


    while j < len(lines):
       if entry_start_re.match(lines[j]) and j + 1 < len(lines):
            normalized.append(lines[j])  # mantm termo + s.f./s.m.


            # verificamos se as prximas linhas tm tradues separadas
            buffer = [lines[j + 1]]
            k = j + 2
            # junta at 3 linhas seguintes, para tentar apanhar [ing] + [esp]
    separados
            while k < len(lines) and len(buffer) < 3:
                buffer.append(lines[k])
                combined = ' '.join(buffer)
                if '[ing]' in combined.lower() and '[esp]' in combined.lower
    ():
```

```
                    normalized.append(combined)
                    j = k + 1
                    break
                k += 1
            else:
                # se no encontrou [ing] e [esp], adiciona a linha normal e avana s 1
                normalized.append(lines[j + 1])
                j += 2
        else:
            normalized.append(lines[j])
            j += 1




    lines = normalized

```

- **Identificação do Início de um Conceito:** Utilizou-se uma expressão regular para identificar o início de um conceito. Esta expressão regular foi projetada para reconhecer a estrutura específica de um termo seguido pelo respetivo substantivo. A identificação precisa do início de cada conceito é fundamental para garantir que todas as informações relacionadas a esse conceito sejam extraídas e processadas corretamente.
  Esta abordagem foi utilizada ao invés de apenas procurar pelo termo sozinho, já que o termo isolado poderia aparecer também no corpo das descrições, gerando falsos positivos ao dividir as entradas. Já o par `Termo + s.m./s.f.` é exclusivo do cabeçalho de cada conceito e não volta a ocorrer em nenhum outro lugar do texto. Assim, este marcador garante que só iniciamos um novo bloco quando realmente começamos um conceito novo.

```
    # Regex para identificar incio de um conceito: termo + subtantivo
    entry_start = re.compile(r'^(.+?)\s+(s\.[mf]\.)$', re.IGNORECASE)

```

- **Extração das Traduções Inglesa e Espanhola:** As traduções foram extraídas, mesmo que venham separadas em várias linhas, ou se algum dos identificadores de tradução estiver em falta. A ordem desses identificadores permite inferir qual é a linguagem de tradução. Este passo envolveu a combinação de linhas adjacentes para formar um bloco de tradução completo, garantindo que todas as traduções estejam corretamente associadas ao termo correspondente.
  Durante a extração, foi necessário verificar se havia "Siglas" inseridas junto à tradução em espanhol, pois a quebra de linha nos determinadores de conceitos causava problemas no processamento desses termos.

```
    entries = []
    i = 0
    n = len(lines)

```

```
    while i < n:
       # Encontra incio de um conceito
       m = entry_start.match(lines[i])
       if not m:
           i += 1
           continue
       termo = m.group(1).strip()
       genero = m.group(2).lower().strip()
       i += 1


       # 5) Extrai a traduo inglesa e espanhola, mesmo que venham separadas
    em vrias linhas
       full_trans = ''  # vai guardar as tradues
       max_lookahead = 3  # quantas linhas  frente vamos tentar colar at
    encontrar [ing]
       found = False  # flag para indicar se j encontramos o bloco de traduo


       for offset in range(max_lookahead):
           # evita ultrapassar o final da lista
           if i + offset >= n:
               break


           # constri um fragmento que junta a linha atual at a linha i+
    offset
           fragment = ' '.join(lines[i:i + offset + 1])


           # procura o marcador [ing] (de forma caseinsensitive)
           if '[ing]' in fragment.lower():
               full_trans = fragment.strip()  # guarda o texto completo at
    aqui
               i += offset + 1  # avana o cursor para logo depois do bloco
    encontrado
               found = True  # sinaliza que encontramos o fragmento
               break  # sai do for
```

- **Extração de Siglas:** As siglas foram extraídas quando presentes no conceito. As siglas são importantes para fornecer uma referência rápida e abreviada para termos mais longos e frequentemente utilizados. A extração de siglas garante que todas as abreviações relevantes sejam capturadas e incluídas no glossário. Essas siglas eram simples de identificar, visto que sempre eram precedidas por `"Sigla:"`. Porém ao longo do processamento estas foram apresentando particularidades de processamento bastante particulares, que necessitam de um processamento bastante minuncioso para cada caso.

```
    # 1) Extrai sigla inline de full_trans, se houver
    sigla_inline = ''
    sigla_match = re.search(r'\bSigla\s*:\s*(\w+)', full_trans, re.
    IGNORECASE)
    if sigla_match:
       sigla_inline = sigla_match.group(1).strip()
       # remove Sigla: XXX de full_trans para no poluir a descrio
       full_trans = re.sub(r'\bSigla\s*:\s*\w+', '', full_trans, flags=re.
    IGNORECASE).strip()


    # 2) Extrai sigla em linha separada, se vier numa linha prpria
    sigla_next = ''
    if i < n and re.match(r'^\s*Sigla\s*:\s*', lines[i], re.IGNORECASE):
       parts = lines[i].split(':', 1)
       candidate = parts[1].strip()
       if candidate:
           # Sigla veio na mesma linha: "Sigla: ABC"
           sigla_next = candidate
           i += 1
       elif i + 1 < n:
           # Sigla veio na linha seguinte:
           # linha atual = "Sigla:", prxima = "AVCI"
           sigla_next = lines[i + 1].strip()
           i += 2
       else:
           # s tinha "Sigla:" no fim do arquivo
           i += 1


    # 3) Escolhe a sigla que existir (prioridade para inline)
    sigla = sigla_inline or sigla_next
```

- **Extração da Descrição do Termo:** A descrição do termo foi extraída até encontrar uma citação, informação enciclopédica ou o início de um novo termo. A descrição fornece uma explicação detalhada do termo, incluindo o seu significado e contexto de uso. A extração precisa da descrição é crucial para garantir que os utilizadores do glossário compreendam completamente cada termo.
  De modo à descrição a não conter nem siglas, nem partes das traduções que estavam desconfiguradas no pdf, foi necessário um processamento particular para garantir a integridade destes elementos.

```
    # Extrai traduo ingls + espanhol do texto restante j sem a Sigla
    m2 = re.match(
       r'^(.+?)\s*\[ing\];?\s*(.+?)\s*\[esp\]\s*(.*)$',
       full_trans,
       re.IGNORECASE
    )
```

```
    if m2:
       termo_ing = m2.group(1).strip()
       termo_esp = m2.group(2).strip()
       # tudo aps [esp] vira incio da descrio
       desc_parts = [m2.group(3).strip()] if m2.group(3).strip() else []
    else:
       # fallback: no havia [esp], tudo depois de [ing]  termo_esp
       m2 = re.match(r'^(.+?)\s*\[ing\];?\s*(.+)$', full_trans, re.
    IGNORECASE)
       if not m2:
           raise ValueError(f"Formato inesperado para tradues: '{full_trans}
    '")
       termo_ing = m2.group(1).strip()
       termo_esp = m2.group(2).strip()
       desc_parts = []


    # continua o loop de descrio a partir de desc_parts inicial
    while i < n and not lines[i].lower().startswith('inf. encicl') \
           and not lines[i].startswith(('', '"')) \
           and not entry_start.match(lines[i]):
       desc_parts.append(lines[i])
       i += 1
    descricao = ' '.join(desc_parts).strip()


```

- **Extração da Informação Enciclopédica:** A informação enciclopédica foi extraída quando presente. Esta informação fornece detalhes adicionais sobre o termo, como o seu contexto histórico, científico ou cultural. Estas informações eram identificáveis pela expressão `"Inf.encicl.:"` quando presentes.

```
    # Informao enciclopdia
    info_enc = ''
    if i < n and lines[i].lower().startswith('inf. encicl'):
       info_enc = re.sub(r'(?i)^inf\. encicl\.:?\s*', '', lines[i]); i += 1
       while i < n and not lines[i].startswith(('','"')) and not entry_start
    .match(lines[i]):
           info_enc += ' ' + lines[i]; i += 1
       info_enc = info_enc.strip()

```

- **Extração da Citação e Números de Artigos:** A citação e os números dos artigos foram extraídos para fornecer exemplos de uso do termo e referências às fontes originais. As citações ajudam a ilustrar como os termos são utilizados em contextos reais, e eram identificadas por um início com aspas, sendo consideradas terminadas quando encontravam a zona dos números dos artigos, iniciada por `"("`.

```

    # --- Alteracao: captura citao apenas ate '(' indicando nmero ---
    citacao = ''
```

```
    nr_artigos = []
    if i < n and lines[i].startswith((’’,’"’)):
        citacao = lines[i]; i += 1
        # agrupa ate ver linha comeando por ’(’
        while i < n and not lines[i].startswith(’(’) and not entry_start.
    match(lines[i]):
            citacao += ’ ’ + lines[i]; i += 1
        if i < n and lines[i].startswith(’(’):
            nums = re.findall(r’\d+’, lines[i]); nr_artigos = [int(x) for x
    in nums]; i += 1


```

- **Gravação em JSON:** Finalmente, os dados processados foram gravados num arquivo JSON, garantindo a persistência dos dados extraídos e permitindo seu uso posterior em outras aplicações ou análises.

```
    entries.append({
        "Termo": termo,
        "Substantivo": genero,
        "Termo ing": termo_ing,
        "Termo esp": termo_esp,
        "Sigla": sigla,
        "Descricao": descricao,
        "Informacao Enciclopedia": info_enc,
        "citacao": citacao,
        "nr_artigos": nr_artigos
    })


    # Grava em JSON
    with open(’glossario.json’, ’w’, encoding=’utf8’) as out:
        json.dump(entries, out, ensure_ascii=False, indent=2)


    print(f"Processados {len(entries)} termos e gerado ’glossario.json’.")

```

Este processo permitiu a extração e organização dos termos do glossário de forma estruturada, facilitando a análise e utilização dos dados. Cada etapa foi cuidadosamente planejada e executada para garantir a precisão e a integridade das informações extraídas do documento "neologismos.pdf".

6.3.4 Equivalências Inglês-Português

- **Leitura do Ficheiro XML:** O conteúdo do ficheiro XML `"equivalencias_ing_pt.xml"` é lido utilizando a codificação UTF-8. A função `open()` abre o arquivo em modo de leitura e o conteúdo é armazenado na variável `xml_content`.

```
    with open("equivalencias_ing_pt.xml", "r", encoding="utf-8") as file:
```

```
    xml_content = file.read()

```

- **Extração das Linhas de Texto com Expressão Regular:** Uma expressão regular é usada para capturar as linhas de texto que estão dentro das tags `<text>` no conteúdo do XML. Esta expressão extrai duas partes: a posição horizontal (`left`) e o conteúdo da tag. O método `re.findall()` é utilizado para aplicar a expressão ao conteúdo XML, e os resultados são armazenados na lista `text_regex`, onde cada entrada contém a posição horizontal e o texto da linha.

```
    # Expresso regular para capturar todas as linhas de texto
    text_regex = re.findall(r'<text top="\d+" left="(\d+)"[^>]*>(.*?)<\/text
    >', xml_content)

```

- **Inicialização de Estruturas para Armazenamento das Linhas Agrupadas por Posição Horizontal:** São criadas variáveis para organizar as traduções extraídas:
  - `eng_to_pt`: Dicionário vazio que armazenará as equivalências entre as palavras ou expressões em inglês (como chave) e as respectivas traduções em português (como valor).
  - `current_english`: Variável que guarda a última linha identificada como sendo em inglês.
  - `portuguese_lines`: Lista que armazena as linhas subsequentes que são as traduções em português.
  - `is_reading_portuguese`: Variável booleana inicializada como False, que indica se o código está a ler traduções em português.

```
    eng_to_pt = {}
    current_english = ""
    is_reading_portuguese = False
    portuguese_lines = []

```

- **Processamento das Linhas Identificadas:** O código percorre cada linha extraída de `text_regex`, realizando os passos explicados seguidamente.

```
    for left, content in text_regex:
        left = int(left)
        content = re.sub(r"<.*?>", "", content).strip()  # Remove tags HTML
    internas
        print(f"left: {left}, content: {content}")
        # Ignora linhas vazias ou linhas que so cabealhos/ttulos
        if not content or content.upper() in ["INGLS", "PORTUGUS"] or "
    Equivalncias" in content:
            continue

        if left == 128:
            # Verifica se a linha contm ambos os idiomas (com 2 ou + espaos
    consecutivos)
            if re.search(r"\s{2,}", content):
                parts = re.split(r"\s{2,}", content)
```

```
                if len(parts) >= 2:
                    eng, pt = parts[0].strip(), parts[1].strip()
                    eng_to_pt[eng] = pt
                    current_english = ""
                    portuguese_lines = []
                    is_reading_portuguese = False
                    continue  # j foi tratado, passa  prxima linha

            # Caso normal (s ingls)
            if current_english and portuguese_lines:
                eng_to_pt[current_english] = " ".join(portuguese_lines).
    strip()
            current_english = content
            portuguese_lines = []
            is_reading_portuguese = False

        elif left == 473: #tradues correspondentes em portugues
            portuguese_lines.append(content)
            is_reading_portuguese = True

        elif left > 473 and is_reading_portuguese:
            # Continuao da traduo em portugus
            portuguese_lines.append(content)

```

- **Limpeza do Conteúdo:** As tags HTML internas são removidas com:

```
    re.sub(r"<.*?>", "", content)

```

O conteúdo é limpo de espaços em branco com `strip()` e armazenado na variável `content`.
- **Filtragem de Linhas Relevantes:** São ignoradas linhas vazias ou que contenham termos como "INGLÊS", "PORTUGUÊS" ou "Equivalências", utilizando:

```
    if not content or content.upper() in ["INGLES", "PORTUGUES"] or "
    Equivalencias" in content

```

- **Identificação de Traduções (Inglês e Português):**

– **Linha em Inglês (`left == 128`):** Se a posição for 128, a linha é considerada em inglês. O código verifica se a linha contém ambos os idiomas (inglês e português) usando:

```
        re.search(r"\s{2,}", content)

```

Assim, consegue detetar múltiplos espaços consecutivos e caso se verifique a existência de 2 idiomas, a linha é dividida em duas partes: a primeira parte é considerada como inglês (eng) e a segunda parte como português (pt). Ambas as partes são armazenadas no dicionário eng_to_pt. Se a linha não tiver ambos os idiomas e for apenas em inglês, o código armazena a linha em inglês em current_english e aguarda pelas linhas de tradução em português que seguirão.

- **Linha em Português (`left == 473`):** Se a posição horizontal for 473, então a linha corresponde a uma tradução em português. O conteúdo dessa linha é adicionado à lista "portuguese_lines", que armazenará as traduções associadas à última sigla em inglês encontrada.
- **Continuação de Traduções (`left > 473`):** Linhas com posição horizontal maior que 473, enquanto `is_reading_portuguese` estiver ativada, também são adicionadas à lista "portuguese_lines" de traduções em português. Isso permite que traduções longas ou divididas em várias linhas sejam agrupadas corretamente.
- **Armazenamento das Traduções:** Quando o código encontra uma linha de inglês ("current_english") seguida de uma ou mais linhas de tradução em português ("portuguese_lines"), ele adiciona essa tradução ao dicionário "eng_to_pt", com o conteúdo em inglês como chave e a tradução em português como valor.

- **Adição da Última Tradução:** No final do processamento, o código verifica se existe uma última sigla em inglês ("current_english") e uma lista de traduções em português ("portuguese_lines"). Caso afirmativo, a última tradução é adicionada ao dicionário "eng_to_pt".

```
    # No fim do loop, adiciona a ultima entrada valida pq ja nao ha mais
    entradas inglesas
    if current_english and portuguese_lines:
        eng_to_pt[current_english] = " ".join(portuguese_lines).strip()

```

- **Criação de um Ficheiro JSON para Guardar as Siglas:** Finalmente, o dicionário "eng_to_pt", que contém todas as equivalências de inglês e português, é salvo num ficheiro JSON chamado "ing_pt.json". A função json.dump() é utilizada para salvar o dicionário no arquivo.

```
    # Salva em JSON
    with open("ing_pt.json", "w", encoding="utf-8") as json_file:
        json.dump(eng_to_pt, json_file, ensure_ascii=False, indent=2)

```

Este processo permitiu a extração e organização dos dados das equivalências Inglês-Português de forma estruturada, facilitando a análise e utilização das informações. Cada etapa foi cuidadosamente planejada e executada para garantir a precisão e a integridade das informações extraídas do documento "neologismos.pdf".

6.3.5 Equivalências Espanhol-Português

- **Leitura do Ficheiro XML:** O conteúdo do ficheiro XML `"equivalencias_ing_pt.xml"` é lido utilizando a codificação UTF-8. A função `open()` abre o arquivo em modo de leitura e o conteúdo é lido na variável `xml_content`, permitindo que o código trabalhe com todo o conteúdo textual do arquivo XML.

```
    with open("equivalencias_es_pt.xml", "r", encoding="utf-8") as file:
    xml_content = file.read()

```

- **Extração dos Blocos de Texto com Expressão Regular:** Utilizando uma expressão regular, o código captura todas as tags `<text>` no arquivo XML. A expressão regular extrai três elementos principais de cada tag `<text>`, são eles:

  – `top`: a posição vertical da linha.
  – `left`: a posição horizontal da linha.
  – `content`: o conteúdo textual dentro da tag, que pode incluir tags HTML internas.

  A função `re.findall()` é utilizada para aplicar a expressão regular ao conteúdo do arquivo XML. O resultado é armazenado na lista `"text_blocks"`, onde cada item é um tuplo com os valores de `top`, `left` e `content` extraídos de cada tag `<text>`.

```
    # L o contedo do XML
    with open("equivalencias_es_pt.xml", "r", encoding="utf-8") as file:
        xml_content = file.read()

    # Expresso regular: extrai (top, left, content)
    text_blocks = re.findall(r'<text top="(\d+)" left="(\d+)"[^>]*>(.*?)<\/
    text>', xml_content)


```

- **Inicialização de Estruturas para Armazenamento:** Antes de processar os dados extraídos, são inicializadas algumas variáveis para organizar as traduções:

  – `es_to_pt`: Um dicionário vazio que armazenará as equivalências entre as palavras ou expressões em espanhol (como chave) e as respetivas traduções em português (como valor).
  – `current_spanish_parts`: Uma lista que armazenará temporariamente as partes do texto em espanhol.
  – `current_portuguese_parts`: Uma lista que armazenará temporariamente as partes do texto em português.
  – `collecting_portuguese`: Uma variável booleana inicializada como `False`, que indica se o código está a armazenar traduções em português.

```
    es_to_pt = {}
    current_spanish_parts = []
    current_portuguese_parts = []
    collecting_portuguese = False

```

- **Processamento das Linhas de Texto Extraídas:** O código percorre cada linha extraída das tags `<text>`, representada pelos valores de `top`, `left` e `content`. O conteúdo é então processado conforme a posição horizontal (`left`) da tag, identificando se a linha está em espanhol ou português.

```
    for top, left, content in text_blocks:
        top = int(top)
        left = int(left)
        content = re.sub(r"<.*?>", "", content).strip()

```

```
        # Ignora ttulos e vazios
        if not content or content in ["ESPANHOL", "PORTUGUS", "3.4.
    Equivalncias espanhol  portugus"]:
            continue

        if left < 431:
            if collecting_portuguese and current_spanish_parts and
    current_portuguese_parts:
                # Salva par anterior
                full_spanish = " ".join(current_spanish_parts).strip()
                full_portuguese = " ".join(current_portuguese_parts).strip()
                es_to_pt[full_spanish] = full_portuguese

                # Reset
                current_spanish_parts = []
                current_portuguese_parts = []
                collecting_portuguese = False

            current_spanish_parts.append(content)

        elif left >= 431:
            current_portuguese_parts.append(content)
            collecting_portuguese = True

```

- **Limpeza do Conteúdo:** Antes de processar o conteúdo das tags, o código realiza a limpeza do texto. As tags HTML internas são removidas utilizando a expressão regular:

```
    re.sub(r"<.*?>", "", content)

```

Espaços extras no início e no final do conteúdo são removidos com o método `strip()`.
- **Filtragem de Linhas Relevantes:** O código ignora certas linhas que não são relevantes para a extração de traduções, como títulos, cabeçalhos e outras linhas que contêm "ESPANHOL", "PORTUGUÊS" ou "3.4. Equivalências espanhol – português".
- **Identificação de Traduções em Espanhol e Português:** O código usa a posição horizontal da linha (`left`) para distinguir entre as traduções em espanhol e em português.
- **Texto em Espanhol (left < 431):** Quando a posição `left` é menor que 431, o conteúdo é interpretado como espanhol. Esse conteúdo é adicionado à lista `current_spanish_parts`. Se o código estava a armazenar uma tradução em português antes, ele guarda a tradução anterior como um par (espanhol, português) no dicionário `es_to_pt`, e reinicia as variáveis e listas para o próximo par de tradução.
- **Texto em Português (left >= 431):** Quando a posição `left` é maior ou igual a 431, o conteúdo é interpretado como português. O conteúdo é adicionado à lista `current_portuguese_parts`, e o sinalizador `collecting_portuguese` é definido como `True`, indicando que o conteúdo subsequente será a tradução em português de uma linha em espanhol.
- **Armazenamento das Traduções em Espanhol e Português:** Cada vez que o código encontra uma linha em espanhol seguida por uma ou mais linhas em português, ele cria um par de tradução (`espanhol`, `português`) e armazena no dicionário `es_to_pt`.

- **Adição da Última Tradução:** Após percorrer todas as linhas, o código verifica se há um par de tradução pendente, ou seja, se as variáveis `current_spanish_parts` e `current_portuguese_parts` ainda contêm conteúdo válido. Caso haja, o par final é adicionado ao dicionário `es_to_pt`.

```
    if current_spanish_parts and current_portuguese_parts:
    full_spanish = " ".join(current_spanish_parts).strip()
    full_portuguese = " ".join(current_portuguese_parts).strip()
    es_to_pt[full_spanish] = full_portuguese

```

- **Criação de um Ficheiro JSON para Guardar as Siglas:** Após o processamento, o dicionário `es_to_pt`, contendo todas as equivalências entre espanhol e português, é gravado num ficheiro JSON denominado `es_pt.json`, com a função `json.dump()`.

```
    with open("es_pt.json", "w", encoding="utf-8") as json_file:
        json.dump(es_to_pt, json_file, ensure_ascii=False, indent=2)

```

Este processo permitiu a extração e organização dos dados das equivalências Espanhol-Português de forma estruturada, facilitando a análise e utilização das informações. Cada etapa foi cuidadosamente planejada e executada para garantir a precisão e a integridade das informações extraídas do documento "neologismos.pdf".

6.3.6 Anexos

- **Leitura do XML inteiro como texto:** O primeiro passo foi ler o documento XML inteiro como texto. Isto permitiu a manipulação das informações contidas nele de forma mais flexível. A leitura do XML como texto bruto é essencial para garantir que todas as informações, incluindo aquelas dentro das tags, sejam capturadas para processamento posterior.

```
    with open('neologismos_anexos.xml', encoding='utf8') as f:
        raw = f.read()

```

- **Filtragem do conteúdo das páginas 218 até 230:** Utilizou-se uma expressão regular para filtrar apenas o conteúdo das páginas 218 até 230, visto que são nestas páginas que se encontram os Anexos.

```
    # 2) Filtra apenas o contedo das pginas 218 at 230
    pages = re.findall(r'<page number="(\d+)".*?>(.*?)</page>', raw, re.
    DOTALL) #retorna tuplos (nr pgina, conteudo pgina)
    filtered_pages = [content for num, content in pages if 218 <= int(num)
    <= 230] # agrupar o conteudo das diferentes pginas em uma lista, para
    retirar da forma de tuplos
    filtered_raw = '\n'.join(filtered_pages) # agrupa os elementos da lista
    em apenas 1 elemento

```

- **Extração do conteúdo entre as tags `<text>...</text>`:** Utilizou-se uma expressão regular para extrair o conteúdo que está entre as tags <text>. Esta etapa é importante para isolar

as informações textuais relevantes, como os títulos dos artigos e os detalhes das edições, que estão encapsuladas dentro dessas tags. Tal como no glossário, as tags não apresentavam informação relevante para a distinção dos diferentes campos da tabela dos anexos, logo apenas o texto englobado nestas era relevante para o processamento da informação.

```
# 3) Extrai o conteudo que esta entre os <text>...</text>
raw_lines = re.findall(r'<text[^>]*>(.*?)</text>', filtered_raw, re.
DOTALL)
print(raw_lines)

```

- **Limpeza das tags, descodificação dos termos, remoção de acentos e espaços:** As tags foram removidas, os caracteres especiais foram descodificados, e os acentos e espaços foram eliminados. Este passo é importante para normalizar os dados, garantindo que todas as informações estejam num formato consistente e livre de caracteres indesejados que possam interferir na análise e no processamento.

```
# 4) Limpa tags, descodifica termos, limpa acentos e espacos
clean_lines = []
for frag in raw_lines:
   txt = re.sub(r'<[^>]+>', '', frag)      # remove tags <b>s e <i>s
   txt = html.unescape(txt)                # converte entidades como
&#34; em "
   txt = unidecode(txt).strip()            # tira acentos e espacos
   if txt:
       clean_lines.append(txt)


print (clean_lines)

```

- **Parse da tabela:** A tabela foi analisada para extrair os registos individuais. Cada registo inclui o número do artigo, o título do artigo, o número da edição, o mês da edição e o ano da edição. A análise da tabela é fundamental para organizar as informações de forma estruturada e facilitar a criação do arquivo JSON. Após uma análise cuidada dos dados, percebeu-se que os diferentes campos de um registo apareciam sempre de forma sequencial, logo bastava encontrar o início de cada registo e percorre-lo de forma sequencial para extrair cada um dos campos.
  - **Número da linha:** Utilizou-se uma expressão regular para detectar o início de cada registo, que é um número de 2 a 3 dígitos. Este passo garante que cada registo seja identificado corretamente.
  - **Título do artigo:** O título do artigo pode ocupar várias linhas até encontrar o próximo número (número da edição). As linhas foram combinadas para formar o título completo do artigo. Este passo é importante para garantir que os títulos dos artigos sejam capturados integralmente.
  - **Número da edição:** O número da edição foi extraído e verificado para garantir que é um número. Esta informação é essencial para identificar a edição específica da revista em que o artigo foi publicado.
  - **Mês da edição:** O mês da edição foi extraído. Esta informação fornece contexto temporal para a publicação do artigo.

- **Ano da edição:** O ano da edição foi extraído. Esta informação é importante para completar a referência temporal da publicação.

```
    # 5) Parse da tabela
    entries = []
    i = 0
    n = len(clean_lines)


    # padrao para detectar o inicio de cada registro que e um numero de 2 a
    3 digitos
    row_re = re.compile(r'^\d{2,3}$')


    # pula ate o primeiro "01"
    while i < n and not row_re.match(clean_lines[i].strip()):
       i += 1


    while i < n:
       # 1) numero da linha
       num = clean_lines[i].strip()
       if not row_re.match(num): #verifica se  um nmero, atravs do regex que
     se fez em cima
            break
       i += 1


       # 2) titulo: pode ocupar varias linhas ate encontrar o proximo umero
       title_parts = []
       while i < n and not re.match(r'^\d+$', clean_lines[i].strip()):
           title_parts.append(clean_lines[i].strip())
           i += 1
       title = ' '.join(title_parts)


       # 3) numero da edicao
       if i < n and re.match(r'^\d+$', clean_lines[i].strip()): #garante que
      um nmero
           num_edic = clean_lines[i].strip()
           i += 1
       else:
           num_edic = ''


       # 4) mes da edio
       if i < n:
           mes_edic = clean_lines[i].strip()
       else:
           mes_edic = ''
```

```
        i += 1


        # 5) ano da edicao
        if i < n:
            ano_edic = clean_lines[i].strip()
        else:
            ano_edic = ''
        i += 1

```

- **Gravação em JSON:** Finalmente, os dados processados foram gravados em um arquivo JSON. O formato JSON foi escolhido por sua estrutura organizada e facilidade de uso em aplicações web e de análise de dados. A gravação em JSON garante que os dados dos anexos estejam armazenados de forma estruturada e acessível para futuras consultas e análises.

```
    entries.append({
            "Numero": num,
            "Titulo do Artigo": title,
            "Numero da Edicao": num_edic,
            "Mes da Edicao": mes_edic,
            "Ano da Edicao": ano_edic
        })


    # 6) Grava em JSON
    with open('anexos_pesquisa.json', 'w', encoding='utf8') as out:
        json.dump(entries, out, ensure_ascii=False, indent=2)


    print(f"Processados {len(entries)} artigos e gerado 'anexos_pesquisa.
    json'.")

```

Este processo permitiu a extração e organização dos dados dos anexos de forma estruturada, facilitando a análise e utilização das informações. Cada etapa foi cuidadosamente planejada e executada para garantir a precisão e a integridade das informações extraídas do documento "neologismos.pdf".

6.4 Estrutura JSON

6.4.1 Abreviaturas Utilizadas

A estrutura do ficheiro JSON apresentada na Figura 4 corresponde ao formato utilizado para representar as abreviaturas identificadas no texto e as respetivas formas completas. Cada entrada do dicionário associa uma sigla (como "FAPESP", "TCT" ou "et al.") à sua designação expandida, garantindo assim clareza.

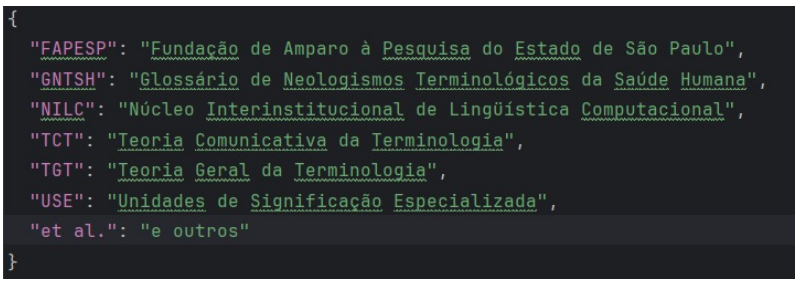

Fig. 4: Estrutura JSON associada às Abreviaturas.

6.4.2 Lista de Abreviaturas e Siglas

A estrutura do ficheiro JSON apresentada na Figura 5 corresponde ao formato utilizado para representar as abreviaturas e siglas identificadas no texto e as respetivas formas completas. Cada entrada do dicionário associa uma sigla/abreviatura à sua designação expandida, garantindo assim clareza.

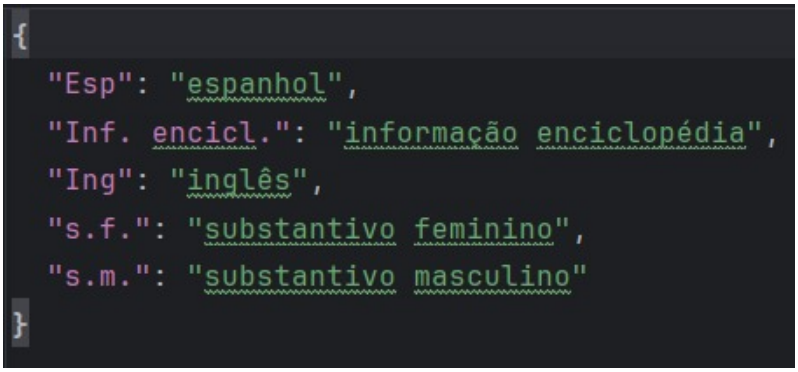

Fig. 5: Estrutura JSON associada às Abreviaturas e Siglas.

6.4.3 Glossário

No que diz respeito aos diferentes conceitos, retirados da parte do glossário, optou-se pela estrutura JSON apresentada na Figura 6.

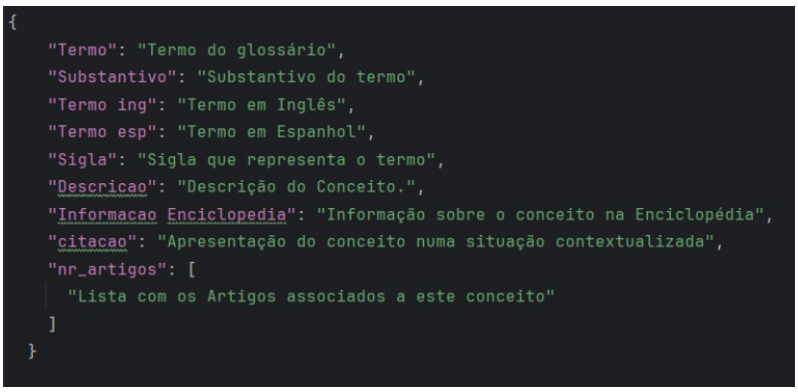

Fig. 6: Estrutura JSON associada aos conceitos do Glossário

É de destacar que esta estrutura engloba todos os possíveis parâmetros que um conceito, no glossário deste documento, pode ter. Os campos Sigla e Informação Enciclopedia só existe em alguns conceitos, sendo que os restantes estão presentes em todos. O campo nr_artigos, é uma lista com os números inteiros dos artigos associados ao conceito.

É importante referir que nesta estrutura JSON estão presentes todos os conceitos presentes no glossário do documento original, 306 conceitos, sendo que destes existem 14 conceitos com Siglas, e são todos corretamente identificados e processados pelo código.

6.4.4 Equivalências Inglês-Português

A estrutura apresentada neste ficheiro JSON representa pares de termos em inglês e as suas respetivas traduções para português. Cada entrada estabelece uma correspondência direta entre um conceito técnico ou biomédico identificado na língua inglesa e a sua tradução padronizada para o português, muitas vezes acompanhada por referências numéricas associadas à sua ocorrência ou uso. A estrutura deste ficheiro JSON pode ser observada na Figura 7.

```
{
  "abeta": "abeta (159)",
  "acetilsalicilic acid": "ácido acetilsalicílico (66)",
  "acute intermittent porphyria": "porfíria aguda intermitente (105)",
  "acute myeloid leukemia": "leucemia mielóide aguda (235)",
  "acute renal insufficiency": "insuficiência renal aguda (191)",
  "age-related macular degeneration": "degeneração macular relacionada à idade (162)",
  "alfa-tocoferal": "alfa-tocoferol (208)",
  "alzheimer's disease": "mal de alzheimer (63, 109, 97)",
  "amarílis vírus": "vírus amarílico (241)",
  "american tegumentar leishmaniasis": "leishmaniose tegumentar americana (186)",
  "american trypanosomiasis": "tripanossomíase americana (145)",
  "amperometric immunosensor": "imunossensor amperométrico (83)",
  "amyotrophic lateral sclerosis": "esclerose lateral amiotrófica (243)",
  "anterior cingulotomy": "cingulotomia anterior (61)",
  "apicoplast": "apicoplasto (35)",
  "apoptose": "apoptose (10, 84, 157, 256)",
  "araraquara": "araraquara (179)",
  "artificial neuronal net": "rede neural artificial (234)",
  "artificial skin": "pele artificial (56)",
  "artrodistrator": "artrodistrator (28)",
  "asian pneumonia": "pneumonia asiática (183)",
  "astemizol": "astemizol (105)",
  "asthmatic allergy": "alergia asmática (245)",
  "autologue vaccine": "vacina autóloga (79)",
  "averive stimulation": "estímulo aversivo (60)",
```


Fig. 7: Estrutura do ficheiro JSON associada às equivalências Inglês-Português.

6.4.5 Equivalências Espanhol-Português

A estrutura apresentada neste ficheiro JSON representa pares de termos em espanhol e as suas respetivas traduções para português. Cada entrada estabelece uma correspondência direta entre um conceito técnico ou biomédico identificado na língua espanhola e a sua tradução padronizada para o português, muitas vezes acompanhada por referências numéricas associadas à sua ocorrência ou uso. A estrutura deste ficheiro JSON pode ser observada na Figura 8.

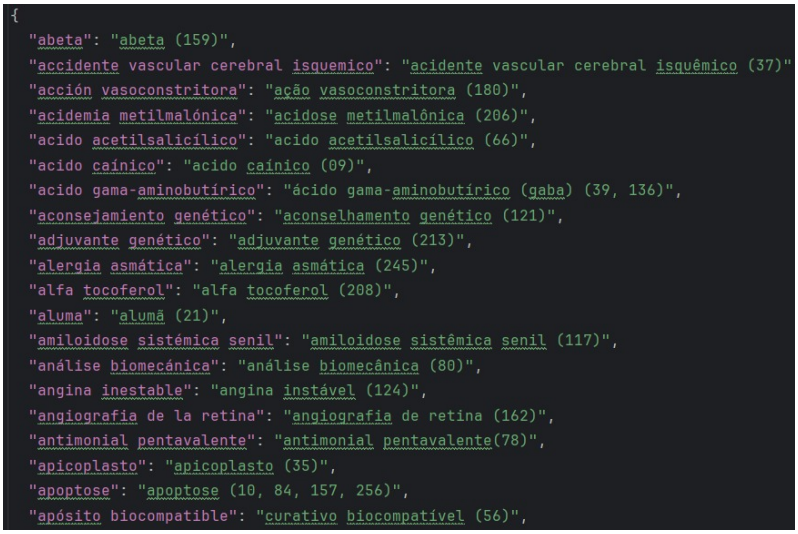

Fig. 8: Estrutura do ficheiro JSON associada às equivalências Espanhol-Português.

6.4.6 Anexos

Relativamente à secção dos anexos o JSON gerado contém os seguintes campos:

- Número: Número identificador do artigo.
- Título do Artigo: Título completo do artigo.
- Número da Edição: Número da edição em que o artigo foi publicado.
- Mês da Edição: Mês da edição em que o artigo foi publicado.
- Ano da Edição: Ano da edição em que o artigo foi publicado.

A estrutura deste ficheiro JSON é a que se encontra na Figura 9.

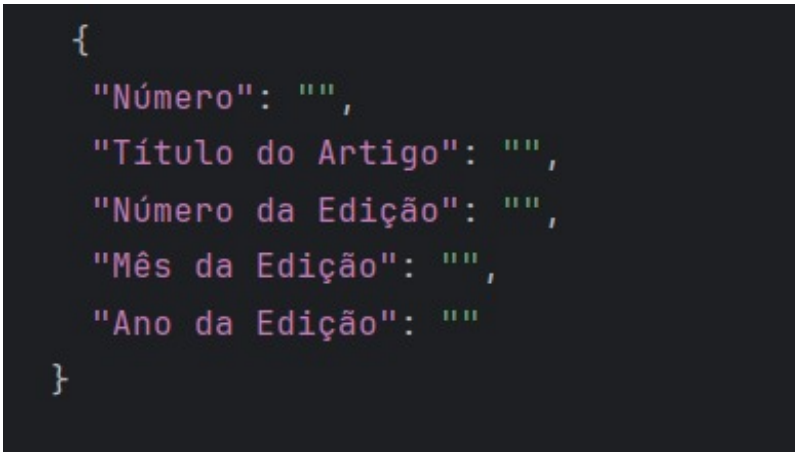

Fig. 9: Estrutura JSON associada aos Anexos

7 glossario_ministerio_saude.pdf

7.1 Análise do documento

Após a análise detalhada do documento, verificou-se que o mesmo apresenta algumas secções com informação considerada muito relevante que deveria ser extraída:

- **Siglas** (páginas 5 - 9): Estruturação da informação:

  SIGLA - Significado da Sigla

- **Glossário** (páginas 15 - 106): Apresenta conceitos e expressões médicas e respetiva definição. Em maior parte dos casos, apresenta também a categoria (Área temática da BVS Saúde Pública) a que o termo pertence. Quando isso não acontece, é recomendado ao leitor que consulte outro termo. Assim, há duas formas de estruturação:

  Conceito
  Categoria: Área temática da BVS Saúde Pública
  Definição do conceito.

  Conceito
  Ver outro conceito.

- **Áreas temáticas da BVS Saúde Pública** (páginas 107 - 112): Esta parte apresenta a descrição das diferentes categorias anteriormente utilizadas para agrupar os conceitos. Está estruturada da seguinte forma:

  Área/categoria
  Descrição da área temática ou categoria.

7.1.1 Exceções às regras

Verificou-se a existência de algumas exceções às regras apresentadas anteriormente, que poderiam precisar de uma atenção especial, nomeadamente:

- Falta de dois pontos após "*Categoria*", como é o caso dos conceitos "Solvente orgânico" e "Sistemas Formais de Cuidados";
- Categoria do conceito "Notificação de doenças" a negrito;
- Conceito "Sistema de Informação sobre Vigilância Alimentar e Nutricional (Sisvan)" escrito com tamanho de letra inferior;
- Conceitos que apresentam duas ou mais categorias, separadas pelo caracter ◉, por exemplo:

Categoria: Administração e Planejamento em Saúde ◉ Ciência e Tecnologia em Saúde

Depois de analisado o ficheiro em PDF, procedeu-se à observação da estrutura do documento convertido para o formato XML.

Verificou-se, de forma relativamente simples, que tanto os termos que pertencem ao glossário como aqueles associados às áreas temáticas utilizavam, de forma consistente, um tamanho fonte correspondente ao valor 21. É importante salientar, no entanto, que um caso específico – o termo "Sistema de Informação sobre Vigilância Alimentar e Nutricional (Sisvan)" – tinha uma dimensão de fonte inferior à dos restantes. Ao analisar este elemento, constatou-se que estava representado com um tamanho de fonte de 13.

Além disso, identificou-se que o símbolo ◉, anteriormente utilizado para separar diferentes categorias dentro de um mesmo conceito, foi convertido no XML como um elemento HTML do tipo text, com fonte de tamanho 24, mas sem conter qualquer conteúdo textual visível.

Para além disso, foi possível analisar que elementos HTML cujo parâmetro *top* era igual a 233, 240, 250, 247, 246, 238 ou 257 referiam-se aos cabeçalhos das páginas (Figura 10).

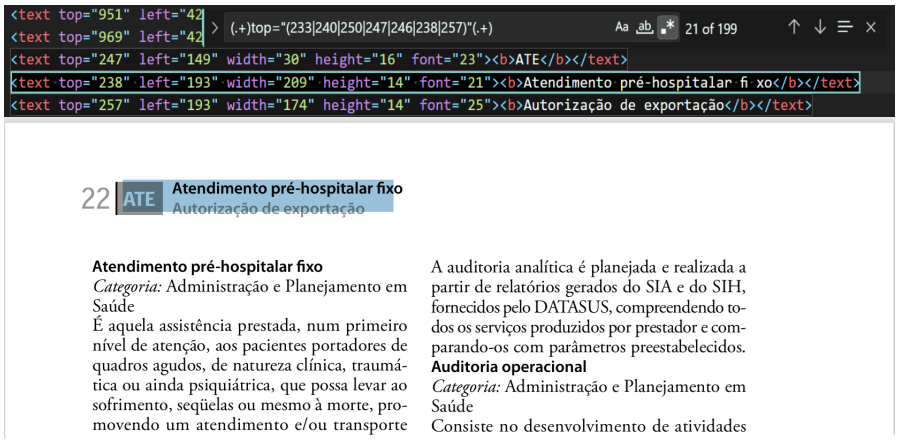

Fig. 10: Identificação do cabeçalho.

7.2 Processamento do documento

Como referido anteriormente, o processamento dos documentos foi efetuado com o recurso a expressões regulares.

O primeiro passo deste processamento foi a marcação das categorias, com a marca "@CAT". Para isso, utilizaram-se os seguintes padrões:

```
# casos em que categoria nao tem dois pontos
texto = re.sub(r"<.+?>Categoria<.+\n<.+>([^:\n]+)<.+?>", r"\n@CAT\1\n", texto)

# correcao do caso "notificacao de doencas" que tem a categoria a bold
texto = re.sub(r"<.+?>Categoria:.+\n<.+?>(.+)<.+", r"\n@CAT\1\n", texto)
```

Posto isto, o resultado obtido desta substituição e marcação já permitia identificar claramente uma separação entre os conceitos e as respetivas categorias (Figura 11).

```
<text top="363" left="427" width="237" height="14" font="21"><b>Acompanhamento do crescimento e </b></text>
<text top="381" left="427" width="162" height="14" font="21"><b>desenvolvimento infantil</b></text>

@CATAtenção à Saúde

<text top="416" left="427" width="289" height="16" font="14">Garantir a melhoria da qualidade de vida das </text>
<text top="434" left="427" width="285" height="16" font="14">crianças, permitindo pôr em evidência, pre-</text>
<text top="452" left="427" width="291" height="16" font="14">cocemente, qualquer transtorno que afete </text>
<text top="470" left="427" width="289" height="16" font="14">sua saúde e, fundamentalmente, sua nutrição </text>
<text top="488" left="427" width="157" height="16" font="14">e sua capacidade mental.</text>
<text top="507" left="427" width="110" height="14" font="21"><b>Aconselhamento</b></text>

@CATAtenção à Saúde

<text top="542" left="427" width="290" height="16" font="14">Processo de escuta ativa, individualizado e </text>
<text top="560" left="427" width="290" height="16" font="14">centrado no cliente. Pressupõe a capacidade </text>
<text top="578" left="427" width="289" height="16" font="14">de estabelecer uma relação de confi ança entre </text>
<text top="596" left="427" width="285" height="16" font="14">os interlocutores, visando ao resgate dos re-</text>
<text top="614" left="427" width="289" height="16" font="14">cursos internos do cliente para que ele mesmo </text>
<text top="632" left="427" width="290" height="16" font="14">tenha possibilidade de reconhecer-se como </text>
<text top="650" left="427" width="285" height="16" font="14">sujeito de sua própria saúde e transformação.</text>
<text top="669" left="427" width="164" height="14" font="21"><b>Aconselhamento coletivo</b></text>

@CATAtenção à Saúde
```

Fig. 11: Marcação da categoria.

Na Figura anterior, pode verificar-se que existiam alguns termos que ocupam duas linhas e, por isso, dois elementos *text* com tamanho de fonte igual a 21, consecutivamente (como é o caso do conceito "Acompanhamento do crescimento e desenvolvimento infantil"). Assim, estes casos foram corrigidos e marcados. Foi também corrigido o caso específico do conceito "Sisvan", referido anteriormente. Estes conceitos foram marcados com "@-".

```
# marcacao dos termos que ocupam duas linhas
texto = re.sub(r".+21\"><b>(.+)\n<.+21\"?><b>(.+)<.+", r"@-\1\2", texto)

# caso especifico do termo sisvan escrito com letra inferior
texto = re.sub(r".+13\"><b>(.+)\n<.+13\"?><b>(.+)<.+", r"@-\1\2", texto)
```

Parte do resultado desta correção está apresentado na Figura 12, onde se pode verificar a correção do conceito "Acompanhamento do crescimento e desenvolvimento infantil").

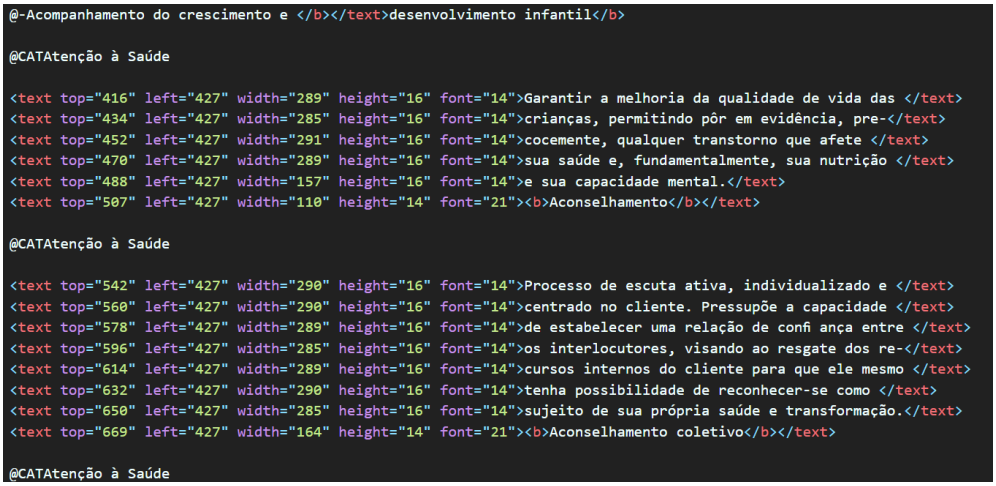

Fig. 12: Correção de termos que ocupam duas linhas.

Os seguintes passos do processamento consistiram na eliminação da numeração das páginas e de elementos itálicos, que iriam interferir na identificação de conceitos (como é o caso do conceito Equivalência in vitro). Para além disso, removeram-se também os conteúdos referentes ao cabeçalho das páginas (Figura 10).

```
# eliminacao da numeracao de paginas e indices
texto = re.sub(r"<.+>[0-9]{1,3}<.+>", r"\n", texto)

# remover italicos para nao interferir na sinalizacao de termos
# (ex: conceito "equivalencia in vitro")
texto = re.sub(r"<[/]?i>", r"", texto)

# remover cabecalhos das paginas com termos/conceitos
# (ex: ADJ Adjuvante farmaceutico Aids pediatrica)
texto = re.sub(r"(.+)top=\"(233|240|250|247|246|238|257)\"(.+)", r"", texto)
```

Posto isto, procedeu-se à marcação das restantes categorias e conceitos, tendo em atenção os casos específicos notados anteriormente.

```
# marcacao de termos que ocupam apenas uma linha
texto = re.sub(r"(.+)font=\"21\"><b>(.+)</b>(.+)", r"\n@-\2\n", texto)

# marcacao de categorias
texto = re.sub(r"<.+?>Categoria:.+\n<.+?>(.+)<.+", r"\n@CAT\1\n", texto)

# casos em que os dois pontos estao separados da palavra "Categoria"
# (ver termo CNS)
texto = re.sub(r"<.+?>Categoria.+\n<.+?>:(.+)<.+", r"\n@CAT\1\n", texto)
```

O resultado obtido destas modificações está apresentado na Figura 13. Pode verificar-se que a estrutura do documento apresenta melhorias significativas no que toca ao aspeto do glossário.

```
@-Gravidez de alto risco

<text top="326" left="176" width="167" height="16" font="14">Ver Gestação de alto risco.</text>

@-Grupo de apoio ao idoso


@CATAtenção à Saúde

<text top="380" left="176" width="285" height="16" font="14">Grupo que promove ações que visem à me-</text>
<text top="398" left="176" width="245" height="16" font="14">lhoria da qualidade de vida dos idosos.</text>

@-Grupo matricial


@CATCiências Sociais em Saúde

<text top="452" left="176" width="290" height="16" font="14">Grupo composto por lideranças lésbicas do </text>
<text top="470" left="176" width="285" height="16" font="14">país, fi liadas a ONGs que desenvolvem tra-</text>
<text top="488" left="176" width="289" height="16" font="14">balhos no âmbito da promoção da saúde, da </text>
<text top="506" left="176" width="289" height="16" font="14">visibilidade lésbica e do combate à epidemia </text>
<text top="524" left="176" width="285" height="16" font="14">do HIV/DST. Foi criado pela CN-DST/</text>
<text top="542" left="177" width="289" height="16" font="14">AIDS em 2001, para ações de prevenção das </text>
<text top="560" left="176" width="289" height="16" font="14">DST/aids junto às mulheres que fazem sexo </text>
<text top="578" left="176" width="146" height="16" font="14">com mulheres (MSM).</text>
```

Fig. 13: Marcação de categorias e conceitos.

Nesta fase, é importante verificar que há palavras que não cabem numa linha e, por isso, estão separadas por hífens (como é o caso da palavra "melhoria", da Figura 13). É importante lidar com estes casos de maneira diferente aos restantes. Assim, removeram-se os hífens e juntaram-se as diferentes partes com a seguinte expressão regular:

```
# tratamento de palavras com hifens
texto = re.sub(r"<.+?>(.+)-<.+\n", r"\1", texto)

# remover elementos html
texto = re.sub(r"<.+?>(.+)<.+\n", r"\1\n", texto)
```

Verifica-se, na Figura 14, que os hífens foram removidos e que as diferentes linhas que contêm as definições dos conceitos se encontram apenas à distância de um elemento *newline*.

O próximo passo consistiu na identificação de casos em que o mesmo conceito apresenta uma ou mais categorias. Nestes casos, como já foi visto anteriormente, as categorias são separadas por um elemento *text*, sem qualquer tipo de conteúdo, cujo tamanho da fonte é igual a 24. Assim, foi necessário implementar a seguinte substituição:

```
# termos com duas ou mais categorias
texto = re.sub(r"[^A-Za-z]?\n+<.+font=\"24\"></text>\n*[^A-Za-z]?(.+)", r"\
    n@CAT\1\n", texto)
```

Após esta substituição, verificou-se a existência de letras isoladas (por exemplo, A), referentes ao cabeçalho das páginas onde iniciam os termos que começam por essa letra. Um exemplo desse cabeçalho está na Figura 15.

Posto isto, foi necessário eliminar estes casos para não interferirem nos termos ou conceitos.

Para além disso, foram corrigidos casos em que as categorias se encontravam separadas, por ocuparem mais do que uma linha. Na Figura 16, pode verificar-se um desses casos.

```
@-Gravidez de alto risco

Ver Gestação de alto risco.

@-Grupo de apoio ao idoso


@CATAtenção à Saúde

Grupo que promove ações que visem à melhoria da qualidade de vida dos idosos.

@-Grupo matricial


@CATCiências Sociais em Saúde

Grupo composto por lideranças lésbicas do
país, fi liadas a ONGs que desenvolvem trabalhos no âmbito da promoção da saúde, da
visibilidade lésbica e do combate à epidemia
do HIV/DST. Foi criado pela CN-DST/
AIDS em 2001, para ações de prevenção das
DST/aids junto às mulheres que fazem sexo
com mulheres (MSM).
```

Fig. 14: Remoção de hífens e elementos html.

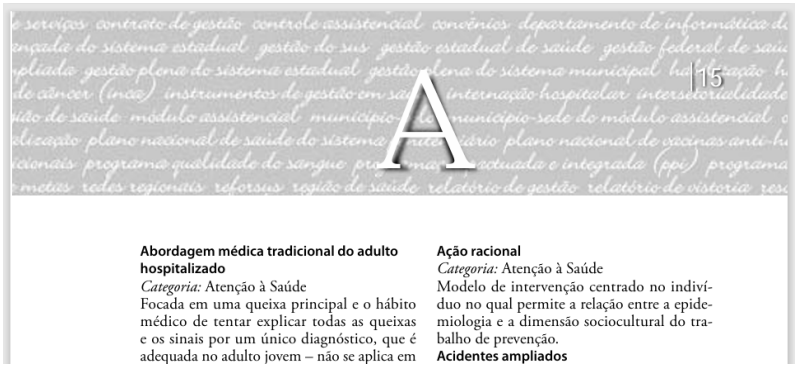

Fig. 15: Exemplo de cabeçalho.

Posto isto, foram implementadas as seguintes substituições:

```python
# eliminacao de cabecalhos
texto = re.sub(r"\n[A-Z]\n", r"\n", texto)

# correcao da categoria Recuros Humanos em Saude Pu-blica
texto = re.sub(r"@CAT(.+)-\n+([^\s\n@<]+)\n", r"@CAT\1\2\n", texto)

# correcao de outras categorias separadas
texto = re.sub(r"@CAT(.+)\n+([^\s\n@<]+)\n", r"@CAT\1\2\n", texto)

# correcao de casos em que a marca @CAT
# nao esta pegada a categoria
```

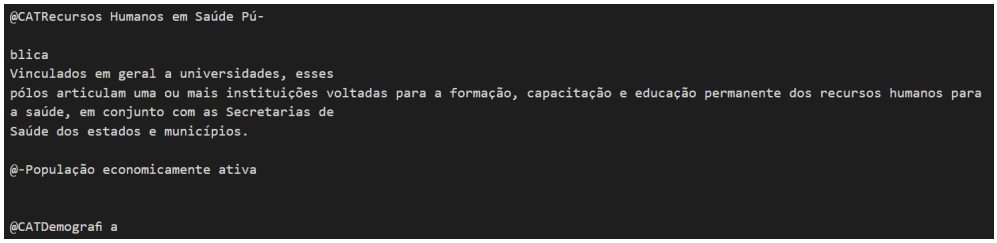

Fig. 16: Categoria "Recursos Humanos em Saúde Pública, dividido.

```
texto = re.sub(r"@CAT(.+)\n+@CAT\s+\n([^\n@<]+)\n", r"@CAT\1\n@CAT\2\n", texto
    )
```

Os próximos passos consistiram na correção de conceitos, categorias ou definições, que continham as expressões "in vivo" ou "in vitro".

```
# remocao do termo in vitro do cabecalho da pag 83
texto = re.sub(r"@CAT\s\n+<b>in vitro</b>", r"", texto)

# correcao dos restantes termos que usam "in vitro" ou "in vivo"
texto = re.sub(r"\n+(<.+>)?\n+<b>(in vi.+)</b>", r"\2", texto)
```

De seguida, foram realizadas algumas substituições em prol de corrigir erros existentes em algumas categorias.

```
# correcao de categorias como Atencao a Saude
texto = re.sub(r"@CAT([^\n]+)([\n]?)\n([^A-ZEOI@\n<])", r"@CAT\1\3", texto)

# remocao de espacos entre marca e categoria
texto = re.sub(r"@CAT\s(.+)", r"@CAT\1", texto)

# correcao da categoria Promocao e Educacao em Saude
texto = re.sub(r"Promocao\s*e\s*[\n]*Ed(.+)", r"Promocao e Ed\1", texto)

# correcao da categoria Medicamentos, Vacinas e Insumos
texto = re.sub(r"@CAT(.+),\s*\n(.+)", r"@CAT\1, \2", texto)
```

Neste ponto, apenas resta eliminar os elementos HTML ainda existentes no documento. No entanto, não serão apagados os elementos *bold*, já que estes irão permitir a identificação das siglas. No entanto, existem conceitos que apresentam elementos *bold* e que, por isso, não estão identificados. Na Figura 17, pode verificar-se um desses casos.

Posto isto, fizeram-se as seguintes substituições, juntamente com outras ainda acerca da correção de categorias e conceitos:

```
# eliminar todos elementos  HTML menos os bold
texto = re.sub(r"<[^b].+[^b]>", r"", texto)

# correcao e marcacao de termos com elementos a bold
texto = re.sub(r"\n<b>(.+)</b>\n+@CAT", r"\n@-\1\n@CAT", texto)

# correcao dos termos Profae e Alimento in natura
```

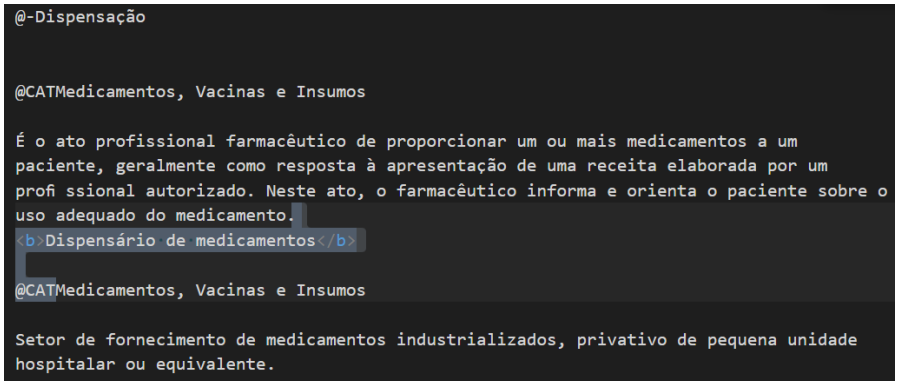

Fig. 17: Conceito com elemento bold.

```
texto = re.sub(r"@-(.+)[\n]*@-(.+)", r"@-\1\2", texto)

# correcao da categoria Vigilancia em Saude
texto = re.sub(r"@CAT \n+(.+)", r"@CAT\1\n", texto)

# correcao de um caso especifico com duas categorias ligadas
texto = re.sub(r"@CAT(.+)Atencao\sa\sSaude", r"@CAT\1\n@CATAtencao a Saude\n",
     texto)
```

Após todas estas substituições, os conceitos e respetivas definições estão prontos a ser extraídos para uma estrutura JSON, bem como as siglas. Para isso, foi utilizado o findall:

```
# termos com 1 categoria
conceitos1 = re.findall(r'@-([^@]+)@CAT(.+)[\n]*([^@]+)', texto)

# termos com 2 categorias
conceitos2 = re.findall(r'@-([^@]+)@CAT([^@]+)@CAT(.+)[\n]*([^@]+)', texto)

# termos com 3 categorias
conceitos3 = re.findall(r'@-([^@]+)@CAT([^@]+)@CAT([^@]+)@CAT(.+)[\n]*([^@]+)'
    , texto)

# termos sem categoria
conceitos4 = re.findall(r'@-(.+)[\n]+Ver([^@]+)', texto)

# siglas
siglas = re.findall(r"<b>(.+)\s-\s?</b>\n([^<]*)", texto)
siglas2 = re.findall(r"<b>(.+)</b>\n[\s]?-([^<]\*)", texto)
```

Após esta extração, as listas obtidas do código anterior foram transformadas em dicionários. Esses dicionários foram, por fim, carregados para um ficheiro JSON.

De seguida é explicada a sintaxe da estrutura de dados utilizada para representar a informação extraída do texto.

7.3 Estrutura JSON

No que diz respeito aos conceitos, categorias e definições retiradas da parte do glossário, optou-se pela seguinte estrutura JSON:

```
{
    "Conceito1": {
        "Categoria": ["Categoria1",
                      "Categoria2"],
        "Descricao": "Descricao do conceito1."
        },

    "Conceito2": {
        "Categoria": ["Categoria1"],
        "Descricao": "Descricao do conceito2."
        },

    "Conceito3": {
        "Descricao": "Ver conceito1."
        }
}
```

Pode verificar-se que esta estrutura aborda todas as formas diferentes sob as quais os conceitos e respetivas categorias e definições estão presentes no documento (ou seja, casos em que há nenhuma, uma ou mais categoria(s)).

Por outro lado, as siglas foram extraídas para um documento JSON com a seguinte sintaxe:

```
{
    "Sigla1": "Significado da Sigla1",
    "Sigla2": "Significado da Sigla2",
    "Sigla3": "Significado da Sigla3"
}
```

Nas Figuras 18 e 19 podem verificar-se excertos dos documentos JSON obtidos relativamente às siglas e aos conceitos presentes no glossário.

Verificou-se ainda que, no total, foram extraídos 705 conceitos e 224 siglas.

Para extrair as definições das diferentes áreas temáticas, foi duplicado o documento XML original e removidos os capítulos "Siglas", "Glossário", "Vocabulário Controlado do Ministério da Saúde", "Bibliografia consultada" e "Descritores organizados por categorias".

Esta duplicação e remoção foi necessária uma vez que o código anteriormente desenvolvido marcou com "@-" o nome das áreas temáticas, tanto no capítulo "Descritores organizados por categorias", como no capítulo "Áreas Temáticas da BVS Saúde Pública". Assim, por não ser possível distinguir que marcações estavam no capítulo das áreas temáticas e no capítulo dos descritores, achou-se pertinente apagar os capítulos acima referidos.

Posto isto, foi criado outro ficheiro python, que abre e processa o ficheiro XML com as partes removidas, com código muito semelhante ao anterior (com exceção das partes de marcação da categoria, visto que estas marcações não eram pertinentes). A estrutura JSON definida para armazernar a informação extraída acerca das áreas temáticas foi a seguinte:

```
{
```

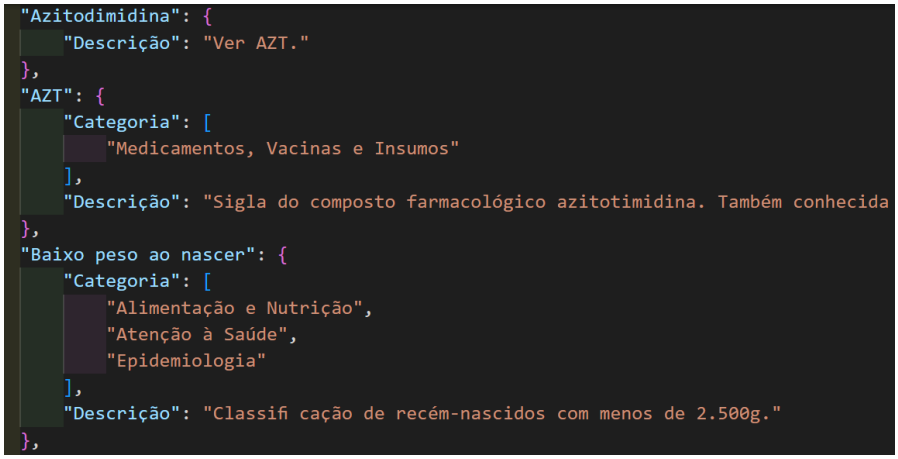

Fig. 18: Documento JSON com os conceitos e respetivas categorias e definições.

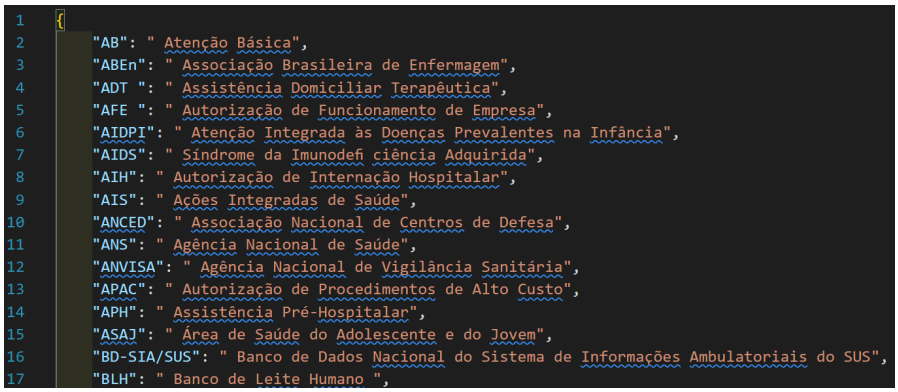

Fig. 19: Documento JSON com siglas e respetivos significados.

```
    "Area 1": "Descricao da Area1",
    "Area 2": "Descricao da Area2",
    "Area 3": "Descricao da Area3",
}
```

Na Figura 20 pode verificar-se um excerto do documento JSON obtido.

Foram extraídas, no total, 24 áreas temáticas do documento.

A mesma técnica foi utilizada para extrair os "Descritores organizados por categorias" (páginas 113 a 124, inclusive). Criou-se um novo ficheiro python e duplicou-se o documento XML original do qual se retiraram partes não relacionadas com o capítulo de interesse. O código desenvolvido para a extração da informação foi muito semelhante ao anterior.

A estrutura utilizada para esta informação foi a seguinte:

```
{
    "Categoria 1": ["Descritor 1", "Descritor 2", "Descritor 3"],
```

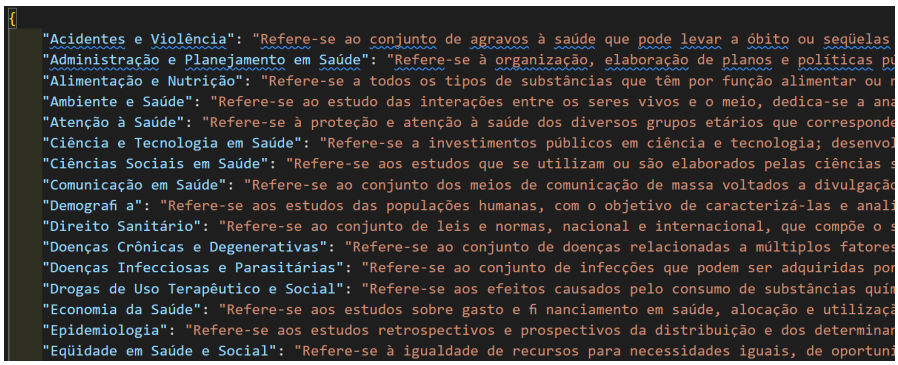

Fig. 20: Documento JSON com as áreas temáticas e respetivas definições.

```
    "Categoria 2": ["Descritor 1"],
    "Categoria 3": ["Descritor 2", "Descritor 4"]
}
```

Parte do resultado obtido está representado na Figura 21.

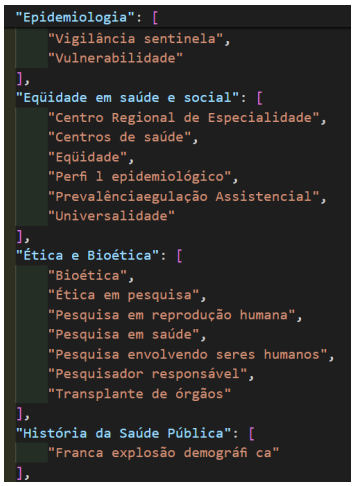

Fig. 21: Documento JSON com os descritores e respetivas categorias.

Deste modo, foram extraídos, no total, 22 descritores.

8 Conclusão

Considera-se que todos os objetivos deste trabalho foram atingidos, tendo sido possível aplicar e aprofundar conhecimentos relativos a expressões regulares abordados durante as aulas de Processamento de Linguagem Natural em Engenharia Biomédica.

Em suma extraiu-se toda a informação que se considerou relevante dos ficheiros em formato PDF escolhidos, tendo esta sido guardada no formato JSON, de acordo com as estruturas definidas.

A principal dificuldade associada ao desenvolvimento deste trabalho foi, de uma forma geral, lidar com os casos específicos encontrados em cada um dos documentos.

Foi possível verificar que existem diferentes formas de extração de informação dos documentos, nomeadamente abordagens em que se realiza (ou não) uma limpeza dos documentos antes do seu processamento.