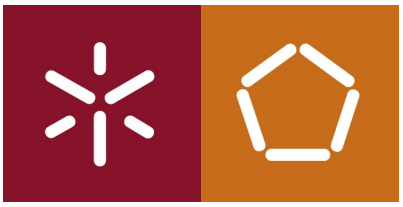

Universidade do Minho
mestrado integrado em engenharia informática

Introdução ao Processamento de Linguagem Natural

(1º Semestre / 4º Ano)

Beautiful Soup - Python Library

Henrique Pereira (a80261)
Diogo Braga (a82547)

Conteúdo

1 Introdução

Processamento de Linguagem Natural (PLN) é uma área baseada no desenvolvimento de aplicações e serviços capazes de entender e processar linguagens humanas naturais. Alguns exemplos práticos de PLN são o reconhecimento de voz, por exemplo: google voice search, compreensão do conteúdo ou análise de sentimentos, etc. [1]

Factualmente, existem milhões de *gigabytes* por dia gerados por *blogs*, sites sociais e páginas web. Existem muitas empresas que recolhem todos esses dados para entender as preferências dos utilizadores, fornecendo, desta forma, relatórios às empresas para que estas vão ajustando os seus planos. Por exemplo, uma pessoa que gosta de viajar e que procura regularmente um destino de férias. Estas pesquisas feitas pelo utilizador são usadas para fornecer anúncios relativos a aplicações de hotéis e reservas de voos online.

De seguida são apresentadas algumas das implementações bem sucedidas de PLN:

- Motores de busca;
- *Feeds* de redes sociais;
- Assistentes pessoal de voz;
- Filtros de spam.

Por vezes, os dados que um utilizador está à procura não estão prontamente disponíveis devido à sua natureza específica. Uma solução possível para esse problema é a ideia de **Web Scraping** ou a extração de informações de um site específico lendo cuidadosamente o HTML. [2]

Web Scraping é, então, uma técnica de extração de dados com capacidade para recolher dados de *sites*. Objetivamente, é realizada a filtragem dos dados que são interessantes para o utilizador. Esta capacidade pode ser aplicada em ambientes `Python` através duma biblioteca apresentada de seguida. [3]

Atualmente na quarta versão, o **Beautiful Soup** é uma biblioteca de `Python` que permite recolher informação de ficheiros HTML e XML de forma bastante simplificada. O Beautiful Soap é, portanto, muito utilizado para *web scraping*, isto é, para extrair dados relevantes de páginas web. Toda a sintaxe da biblioteca é descomplicada, facilitando a sua integração em scripts de `Python`, como iremos ver nos exemplos apresentados mais à frente neste trabalho.

2 Ferramenta

2.1 Funcionalidades

Como referido, o Beautiful Soup tem capacidade para extrair dados de páginas HTML e XML, permitindo obter informações relevantes e inserindo-as, por exemplo, num *dataset*. Resumidamente, o que esta biblioteca permite fazer é analisar um documento complexo e transformá-lo numa árvore de objetos `Python`. Porém, com o Beautiful Soup, apenas iremos lidar com quatro tipos de objetos: *Tag*, *NavigableString*, *BeautifulSoup* e *Comment*, que iremos explicitar já de seguida.

Para o Beautiful Soup funcionar é necessário utilizar juntamente com este um *parser*. A biblioteca suporta vários, tais como o *html.parser* do `Python`, o *lxml* (tanto para HTML como para XML) ou o *html5lib*. Todos possuem vantagens e desvantagens, mas recomenda-se o uso do *lxml*, por ser bastante rápido e flexível.

Com um vasto leque de funções, utilizando o Beautiful Soup é possível procurar, por exemplo, todos os elementos com uma dada classe num documento, ou selecionar os seus filhos. Além disso, é permitido também modificar a árvore, adicionando atributos e modificando valores. [5]

2.1.1 Criar a *soup*

Para analisar um documento, este é passado para o construtor BeautifulSoup. Tal pode ser feito através uma string ou um ficheiro aberto.

Primeiro, o documento é convertido em Unicode e as entidades HTML são convertidas em caracteres Unicode.

De seguida, o Beautiful Soup analisa o documento usando o melhor parser disponivel. Ele irá usar um parser de HTML, a menos que seja especificamente pedido para ser usado um parser XML.

2.1.2 Navegar na árvore

As tags podem conter strings e outras tags. Esses elementos são as tags "filho". O Beautiful Soup fornece muitos atributos diferentes para ir navegando e iterando sobre os filhos de uma tag.

A maneira mais simples de navegar na árvore de *parsing* é dizer o nome da tag desejada.

É possível utilizar esse método várias vezes para chegar a uma determinada parte da árvore. Existe também a capacidade de obter todas as tags de um determinado tipo.

2.1.3 Procurar na árvore

O Beautiful Soup define muitos métodos para pesquisar na árvore, mas são todos muito parecidos. Os dois métodos mais populares são o find() e o find_all(). Os outros métodos existentes usam quase exatamente os mesmos argumentos.

Ao passar um filtro para um argumento como find_all(), é possível chegar a determinadas partes do documento que interessam mais.

2.1.4 Modificar a árvore

A maior capacidade do BeautifulSoup está na pesquisa da árvore de *parsing*, mas também é possível modificar a árvore e gravar suas alterações como um novo documento HTML ou XML.

2.2 Características

O Beautiful Soup transforma um documento HTML complexo numa árvore complexa de objetos Python. São esses tipos os seguintes: [5]

Tag

Um objeto *Tag* corresponde a uma tag XML ou HTML no documento original. Os recursos mais importantes de uma tag são o respetivo nome e os seus atributos.

NavigableString

Uma string corresponde a um pouco de texto dentro de uma tag. O BeautifulSoup usa a classe *NavigableString* para incluir esses bits de texto.

Uma *NavigableString* é como uma string Python Unicode, diferenciando-se na capacidade que têm de "navegar"e procurar na árvore criada.

BeautifulSoup

O objeto *BeautifulSoup* representa o documento analisado como um todo. Isso significa que ele suporta a maioria dos métodos descritos para "navegação"e procura na árvore. Também existe a possibilidade de modificar a árvore.

Como o objeto *BeautifulSoup* não corresponde a um objeto real Tag HTML ou XML, não possui nome nem atributos.

Comment

Os objetos atrás referenciados cobrem quase tudo o que existe num arquivo HTML ou XML, mas existem alguns restos de bits. O único que necessita alguma preocupação diferente é o comentário.

O objeto *Comment* é apenas um tipo especial de *NavigableString*.

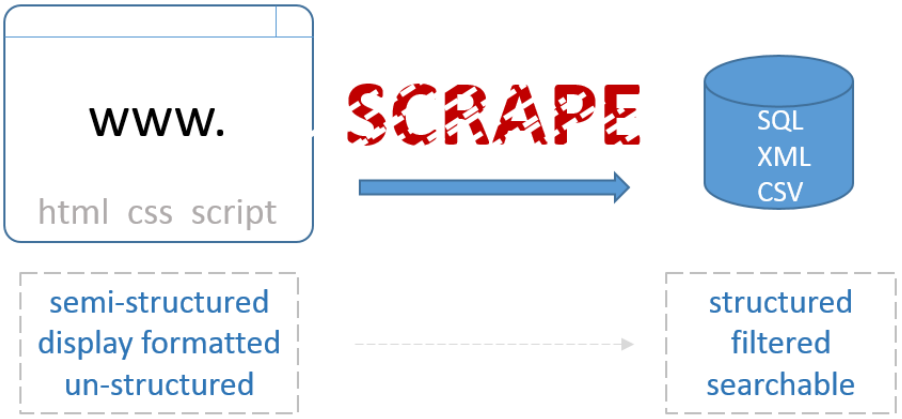

Figura 1: Scraping

3 Exemplo de utilização

Para demonstrar a funcionalidade do Beautiful Soup, montamos um exemplo que, a partir da página IMDb Top 250 TV do *IMDb*, obtemos a lista das 250 melhores séries e a sua classificação, de acordo com o website.

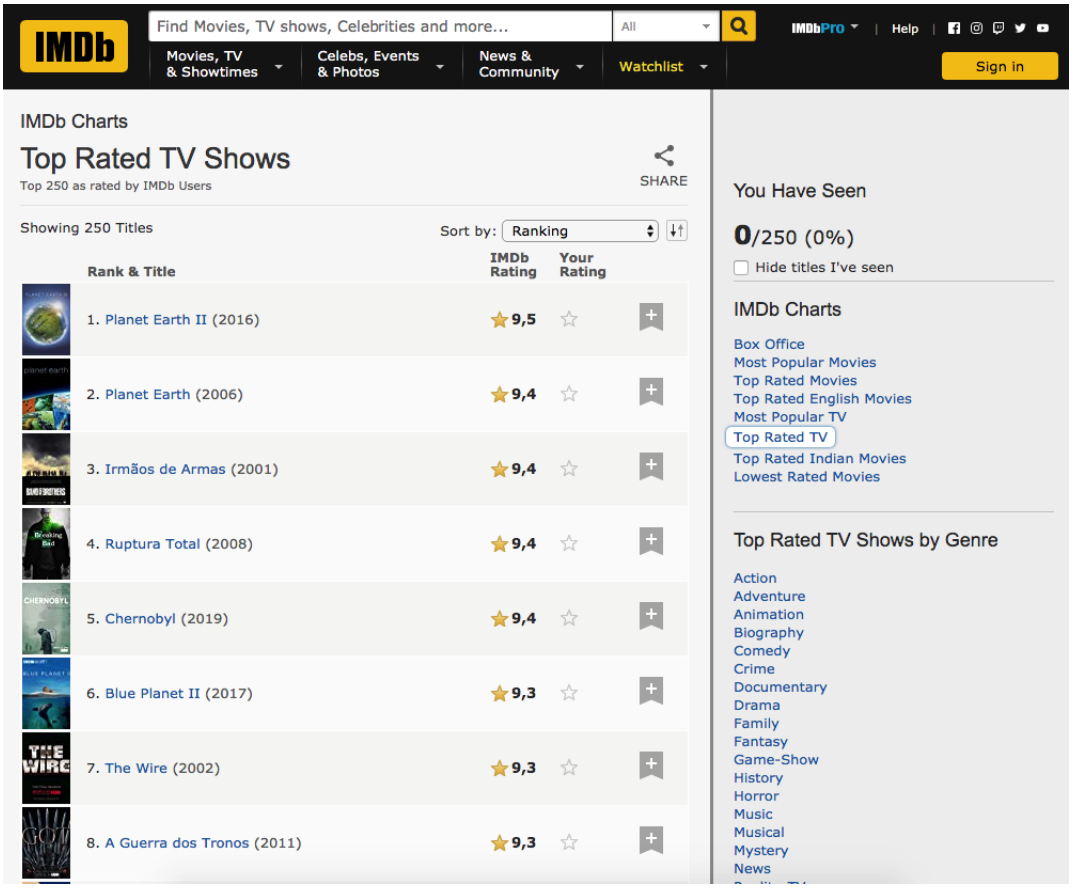

Figura 2: Página a analisar

Analisando a página, conseguimos perceber que os filmes estão inseridos numa *table*, estando cada filme numa linha *tr*, como mostram as imagens que se seguem:

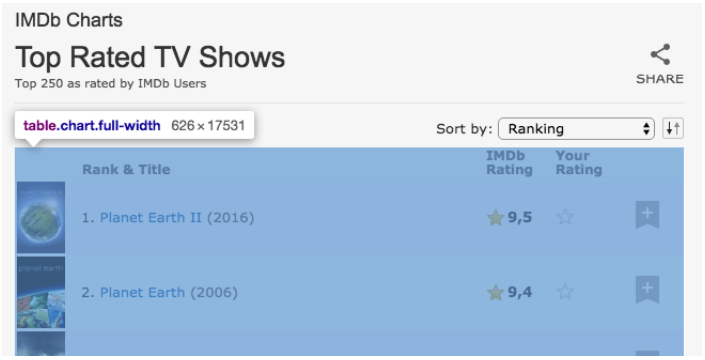

Figura 3: Séries estão dentro de uma *table*

Figura 4: Cada série está representada numa *tr - table row*

Figura 5: Elementos HTML com as informações a retirar dentro de cada *tr*

Desta forma, processar estes dados torna-se simples com o Beautiful Soup, como podemos ver no seguinte exemplo comentado:

```
from bs4 import BeautifulSoup
import requests
import re

# utilizando a biblioteca requests, vamos buscar o ficheiro HTML
# a pagina web
url = 'https://www.imdb.com/chart/toptv/'
content = requests.get(url).text
# inicializacao da lista de series
top250Series = []
# inicializacao do objeto BeautifulSoup com o parser lxml
# para obter os dados da pagina HTML
soup = BeautifulSoup(content, 'lxml')
# selecionar da "sopa" todas as table rows (tr) dentro do tbody
```

```
# cuja classe e "lister-list"
seriesList = soup.select('.lister-list tr')
for series in seriesList:
    # o titulo da serie esta dentro da tag <a> na celula titleColumn
    title = series.find('td', class_='titleColumn').find('a').get_text()
    # o ano da serie esta entre parentesis dentro da tag <span> na celula
                                                titleColumn
    year = series.find('td', class_='titleColumn').find('span').get_text()[1:5]
    # a classificacao esta dentro da tag <strong> na celula imdbRating
    rating = series.find('td', class_='imdbRating').find('strong').get_text()
    # os atores principais estao dentro da celula titleColumn, no atributo 'title'
    mainActors = series.find('td', class_='titleColumn').find('a')['title']
    # passar estes dados para um objeto e adiciona-lo a lista de series
    obj = {
        'title': title,
        'year': year,
        'rating': rating,
        'mainActors': mainActors
    }
    top250Series.append(obj)
# imprimir para o ecra a lista das 250 melhores series
print(top250Series)
```

4 Beautiful Soup no contexto de PLN

Como podemos perceber, o Beautiful Soup é uma potente ferramenta para a extração de informação de ficheiros HTML e XML e, aliada ao Processamento de Linguagem Natural, pode dar origem a análises interessantes. Por exemplo, o que iremos demonstrar de seguida é um *script* que permite analisar uma obra literária portuguesa em HTML (retirada do Project Gutenberg) e apresentar quais as palavras mais utilizadas, permitindo retirar os comuns artigos e preposições (*stopwords*). As *stopwords* podem ser definidas como palavras que surgem imensas vezes no texto, mas que não possuem interesse para a análise a realizar. O *script* que iremos apresentar pode ser corrido com qualquer página web, mas o resultados poderão ser deturpados pois este irá buscar todo e qualquer texto existente na página.

Para aplicar o Beautiful Soup ao Processamento de Linguagem Natural, decidimos criar um *web scraper* que nos indica, graficamente, quais as N palavras mais utilizadas nessa página web, permitindo retirar este tipo de informações dos livros existentes na Internet com o formato HTML. Para tal, seguimos inicialmente duas rotas distintas:

1. O scrapping é feito com o Beautiful Soup e a procura de palavras com o pacote *re* do Python para expressões regulares, sendo que de seguida contabilizamos todas as palavras e ordenamos a lista, passando esta de seguida para um *dicionário* e, por fim, pegando nos N elementos a apresentar e transformando-o esta informação num gráfico de barras.

2. O scrapping é feito com o Beautiful Soup mas, desta vez, toda a restante lógica é implementada recorrendo ao NLTK (*Natural Language Toolkit*).

Tendo estas duas abordagens, achamos que seria interessante apresentar uma comparação entre o desempenho das duas, assim permitir ao utilizador do script selecionar qual das opções queria. Desta forma, o utilizador pode correr o script com as seguintes flags:

- `-u` : de "url", corresponde ao link que queremos que seja usado como fonte.
- `-m` : de "Manual", corresponde à implementação sem NLTK.

- `-n` : de "NLTK", corresponde à implementação com NLTK.
- `-l` : de "lower case", transforma todas as palavras em minúsculas para uma comparação e contabilização mais precisa.
- `-s`: de "stopwords". Indica se se pretendem retirar as *stopwords* da lista de palavras ou não.
- `-N` : de "Número de stopwords". Opção para o modo sem NLTK, em que X pode ser 100, 200 ou 300 (obtidas com o Beautiful Soup da página. Linguateca). Esta flag apenas é utilizada caso `-s` tambem seja.
- `-p` : de "palavras", a apresentar no gráfico. Basicamente, é o TopX.

O código por nós desenvolvido está comentado e é apresentado no Anexo deste documento.

Com isto, pudemos retirar os seguintes resultados aplicando o script com as várias flags à obra "Dom Casmurro"de Machado de Assis, com o respetivo tempo de execução:

Sem NLTK, sem *lower case* e sem *stopwords*, 10 palavras

```
➜  TP2 git:(master) python3 script.py -m -u 'https://www.gutenberg.org/files/55752/55752-h/55752-h.htm' -p 10
STARTED PARSING WITHOUT NLTK
words:  70809
counting word frequency...
sorting by frequency...
Drawing graph...
Elapsed time:  103.44143295288086 seconds
```

Figura 6: Execução do comando

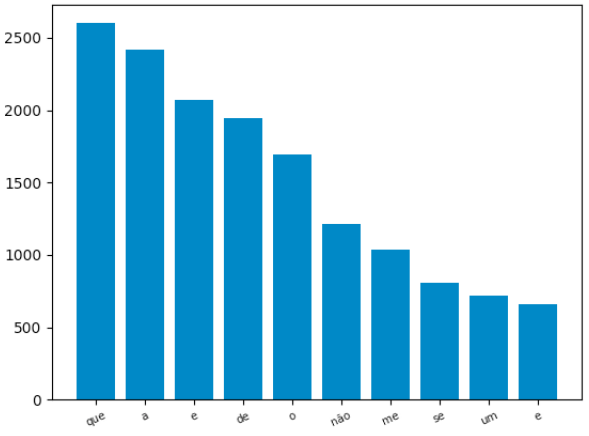

Figura 7: Sem NLTK, sem -l e sem -s, 10 palavras

Como podemos depreender pela análise do gráfico, e como seria de esperar, contabilizando as *stopwords*, as palavras mais utilizadas são os artigos ("a", "o") e as preposições/conjunções/pronomes ("que", "de", "me", etc). Desta feita, a análise não é muito interessante, pois seria já de esperar que estas palavras fossem utilizadas com abundância. De seguida, iremos analisar outro gráfico, desta vez com a opção de *lower case*, isto é, transformando todas as letras das palavras em minúsculas e contando de novo as ocorrências.

Sem NLTK, com *lower case* e sem *stopwords*, 10 palavras

```
→  TP2 git:(master) python3 script.py -m -u 'https://www.gutenberg.org/files/55752/55752-h/55752-h.htm' -l -p 10
STARTED PARSING WITHOUT NLTK
words:  70809
counting word frequency...
sorting by frequency...
Drawing graph...
Elapsed time:  105.59620404243469 seconds
```

Figura 8: Execução do comando

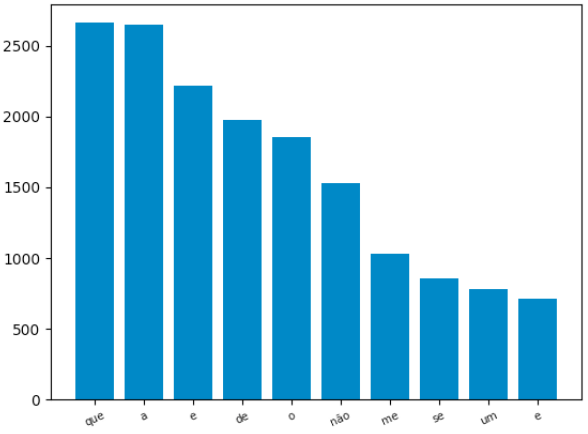

Figura 9: Sem NLTK, com -l e sem -s, 10 palavras

Ora, com a opção de *lower case*, as palavras mais utilizadas continuam a ser as mesmas, mas podemos notar um aumento da ocorrência das mesmas. Podemos ainda observar que a palavra "que"foi a que teve maior incidência no texto, com mais de 2500 ocorrências, enquanto que a menos utilizada (do Top 10) foi o "e"com cerca de 750.

Sem NLTK, com *lower case* e 300 *stopwords*, 10 palavras

```
→  TP2 git:(master) python3 script.py -m -u 'https://www.gutenberg.org/files/55752/55752-h/55752-h.htm' -l -s -N 300 -p 10
STARTED PARSING WITHOUT NLTK
words:  70809
stopwords:  300
clean words:  35840
counting word frequency...
sorting by frequency...
Drawing graph...
Elapsed time:  32.88286089897156 seconds
```

Figura 10: Execução do comando

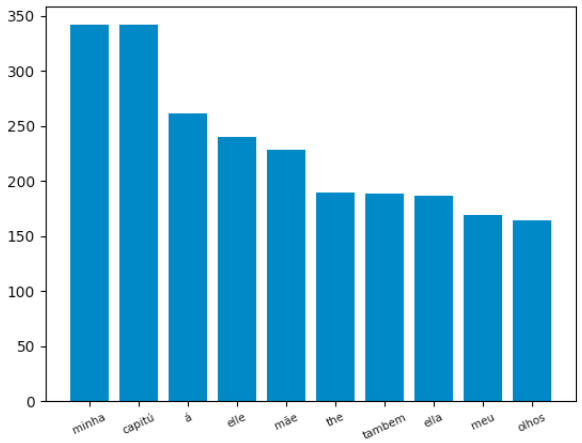

Figura 11: Sem NLTK, com -l, com -s e com 300 N, 10 palavras

Para obtermos resultados mais concretos, corremos novamente o *script*, desta feita com a opção de 300 *stopwords*, removendo-as da lista de palavras apanhada pelo *script*. Observando o gráfico obtido, muitas são as diferenças que se podem verificar: as palavras com maior incidência na obra são "minha"e "capitú"com cerca de 350 ocorrências e a palavra com menos no Top 10 é "olhos", contando-se cerca de 175 vezes ao longo do texto. Analisando a lista de palavras obtidas, podemos compreender o porquê de "capitú"ser uma das mais utilizadas: trata-se da personagem principal da obra. Além disso, podemos retirar de lá palavras como "elle"e "ella", ou seja, "ele"e "ela"na grafia da Língua Portuguesa moderna (a obra data de 1899). Da lista constam também as palavras "á"e "the". Relativamente à primeira, podemos indagar-nos se se tratará também de uma grafia antiga ou de um erro de tradução/adaptação para o Project Gutenberg, e portanto não faz parte das *stopwords*, por não se utilizar o "a"com acento agudo como palavra (sendo um erro gramatical de acentuação quando se verifica). O "the"deve-se à licença do Project Gutenberg existente no início e no final da obra, no ficheiro HTML desta, que não retiramos por acharmos que não teria influência nos resultados, mas, sendo uma *stopword* da Língua Inglesa, tem um peso grande na ocorrência das palavras. É de notar também que a lista de *stopwords* da Linguateca contempla pronomes (como podemos observar pela ocorrência de "minha"e "meu").

Com NLTK, com *lower case* e *stopwords* do dicionário Português, 10 palavras

```
→  TP2 git:(master) python3 script.py -n -u 'https://www.gutenberg.org/files/55752/55752-h/55752-h.htm' -l -s -p 10
STARTED PARSING WITH NLTK
words:  70809
stopwords:  204
clean words:  40132
Elapsed time:  2.355048894882202 seconds
```

Figura 12: Execução do comando

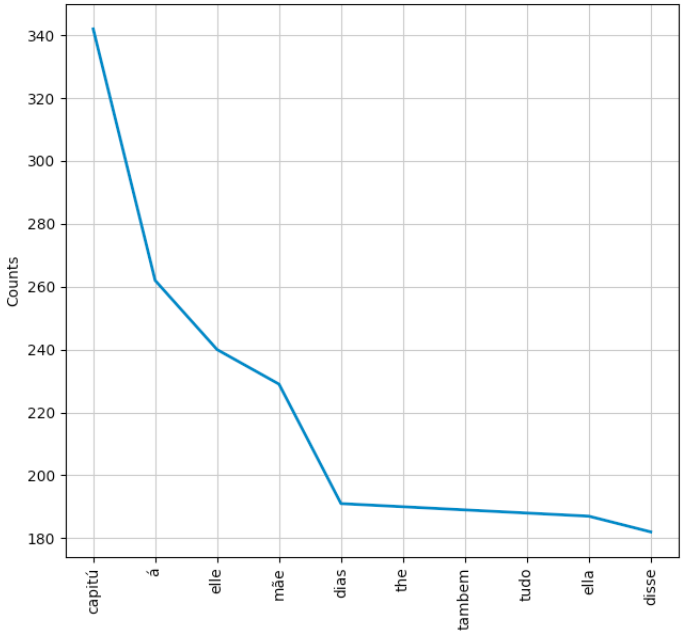

Figura 13: Com NLTK, com l e com s, 10 palavras

Para outra análise diferente, corremos o *script* com a opção que permite utilizar a biblioteca NLTK - Natural Language Toolkit. Ora, para isto, a lista de *stopwords* é fornecida pela *corpora* da biblioteca referida, e contempla 204 palavras. O gráfico final é também diferente, passando-se de gráficos de barras para um histograma. Observando-o, podemos depreender que é semelhante ao último gráfico obtido, com a exceção de que os pronomes observados anteriormente não se encontram neste. Podemos assim concluir que a lista de *stopwords* do NLTK é mais coerente, pois permite uma análise mais correta. Podemos, então, ver que o "minha"e o "meu"foram substituídos por palavras como "tudo"e "disse".

É de notar também a diferença nos tempos de execução (passando de cerca 30 segundos na versão sem NLTK e com 300 *stopwords* para 2 segundos na versão com NLTK), sendo que com NLTK o desempenho é imensamente superior, pois esta biblioteca apresenta funções predefinidas e otimizadas para o Processamento de Linguagem Natural.

4.1 Dependências necessárias para correr o *script*

Para executar o *script*, são necessárias as bibliotecas do `Beautiful Soup`, `matplotlib`, do `NLTK` e ainda do `request`. Para as instalar na máquina, basta correr os seguintes comandos:

```
pip install beautifulsoup4
pip install matplotlib
pip install nltk
pip install request
```

Além disso, para se obter o NLTK Data, ou seja, para se irem buscar as *stopwords* à corpora desta biblioteca, necessitamos de abrir o interpretador de Python e executar o seguinte:

```
>>> import nltk
>>> nltk.download()
```

De seguida, uma nova janela irá abrir e aí o utilizador poderá selecionar o que pretende baixar (neste caso, ou *all* ou *all-corpora*). A janela é a que se segue.

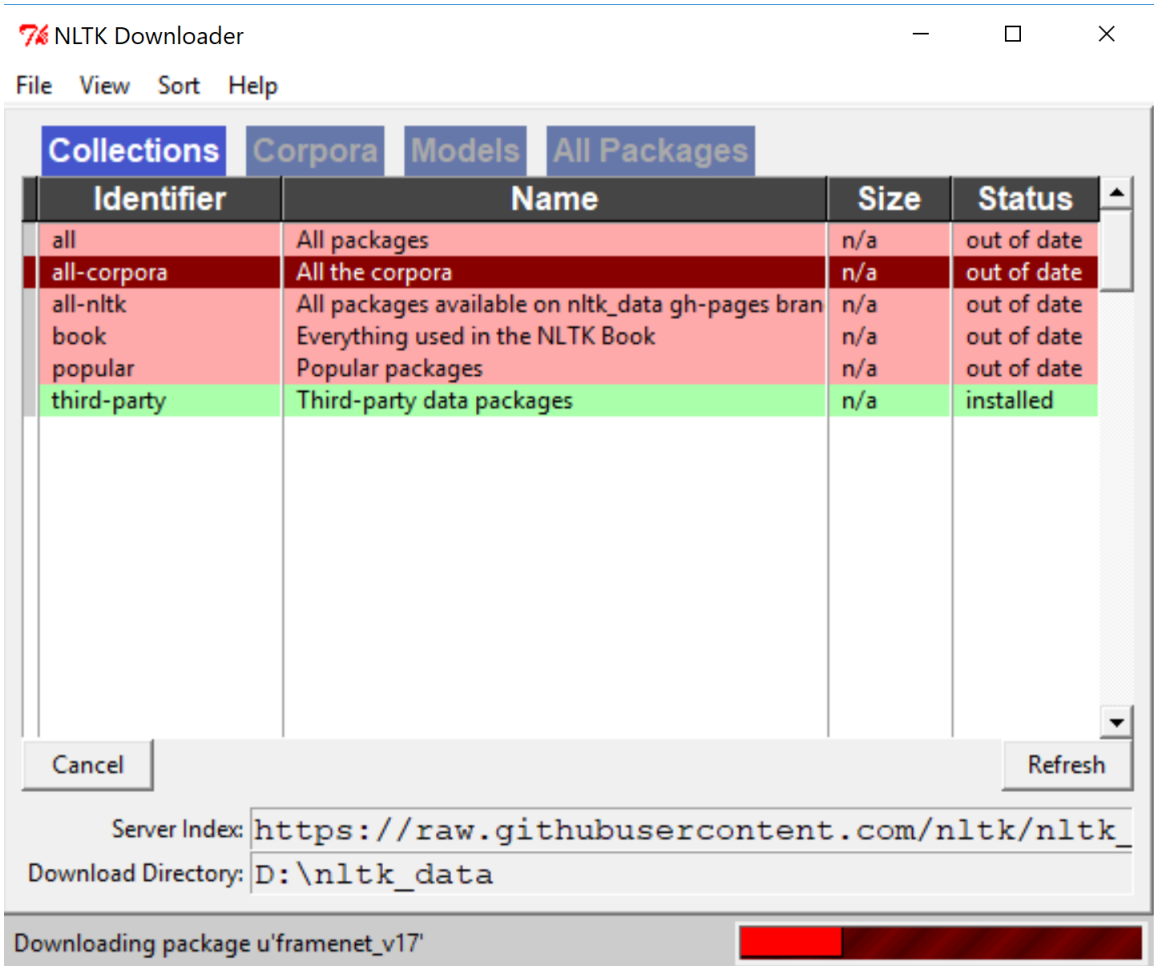

Figura 14: Janela de download do NLTK Data

5 Conclusão

Após realização das investigações e análises apresentadas no relatório, podemos concluir que qualquer página Web, contendo dados que o utilizador deseje, pode ser filtrada através do BeautifulSoup para posterior utilização.

Tal envolve uma examinação do código-fonte HTML da página, localizar pontos de referência que permitem localizar os dados de interesse dentro da página e depois extraí-los. O trabalho na página envolve o uso de métodos de navegação como find_all, parent e next_sibling. A extração dos dados envolve "desembrulhar"as tags para chegar ao texto, respeitando e mantendo a estrutura dos dados.

Esse processo pode ser meticuloso, mas no final, o trabalho com HTML é muito facilitado graças à utilização do BeautifulSoup.

A Anexo

Código Python do Script

```
from bs4 import BeautifulSoup
import requests
import re
import time
import matplotlib.pyplot as plt
from nltk.tokenize import RegexpTokenizer
import nltk
import sys
from getopt import getopt
from urllib.parse import urlparse


def parseHTML(url):
    text_url = url
    # Fazer pedido GET para obter o objecto HTML em formato texto
    text_content = requests.get(text_url).text
    # Criar objeto BS, usando o lxml
    text_soup = BeautifulSoup(text_content, 'lxml')
    # Obter apenas o texto da pagina HTML, sem as tags deste
    return text_soup.get_text()

def getStopwords(num):
    if num == 100:
        stopwords_url = 'https://www.linguateca.pt/chave/stopwords/publico.MF100.txt'
    elif num == 200:
        stopwords_url = 'https://www.linguateca.pt/chave/stopwords/publico.MF200.txt'
    else:
        stopwords_url = 'https://www.linguateca.pt/chave/stopwords/publico.MF300.txt'

    # fazer pedido GET para obter o texto do HTML com as stopwords
    stopwords_content = requests.get(stopwords_url).text
    # Criar objeto BS, usando o lxml
    stopwords_soup = BeautifulSoup(stopwords_content, 'lxml')
    # Obter a lista de stopwords, extraindo-as da div
    return stopwords_soup.get_text()

def lower(flag, list):
    # se a flag de lower estiver ativada,
    # transformar todas as letras das palavras em minusculas
    # caso contrario, devolve a lista que recebeu como parametro
    if flag:
        return [w.lower() for w in list]
    else:
        return list
```

```
def lower_res(tokens):
    words = lower(True, tokens)
    print('words: ', len(words))

    return words

def parseWordsNoNLTK(html):
    # iniciar contador de tempo
    start = time.time()
    print('STARTED PARSING WITHOUT NLTK')

    text = parseHTML(html)

    # obter todas as palavras presentes no texto
    # obtido do parseHTML, com a Regex \w+, no formato UNICODE
    words = re.findall(r"\w+", text, re.UNICODE)
    print('words: ', len(words))

    return words, start

def stopWordsNoNLTK(lower_words,num):
    stopwords_text = getStopwords(num)

    # extrai todas as palavras (\w+) do texto com as stopwords
    stopwords = re.findall(r'\w+', stopwords_text, re.UNICODE)
    print('stopwords: ', len(stopwords))

    # remove da lista de palavras aquelas que se encontram
    # na lista de stopwords
    clean_words = [w for w in lower_words if w not in stopwords]
    print('clean words: ', len(clean_words))

    return clean_words

def plotNoNLTK(clean_words,start,p):
    print('counting word frequency...')
    # conta as ocorrencias das palavras e coloca-as numa lista
    word_count = [clean_words.count(w) for w in clean_words]
    # transforma lista de ocorrencias e de palavras num dicionario
    dictionary = dict(list(zip(clean_words, word_count)))

    print('sorting by frequency...')
    # transforma o dicionario numa lista de pares e ordena-os por
    # numero de ocorrencias, revertendo no final a ordem para ordem
    # decrescente
    sorted_list = [(dictionary[key], key) for key in dictionary]
    sorted_list.sort()
    sorted_list.reverse()
```

```
    # obtem a informacao dos pares e coloca-a na respetiva lista
    # separando as palavras e as respetiva ocorrencias
    names = [val[1] for val in sorted_list]
    values = [val[0] for val in sorted_list]

    # desenhar o grafico com P barras, utilizando o matplotlib
    print('Drawing graph...')
    plt.bar(range(p), values[:p], tick_label=names[:p])
    plt.xticks(rotation=25, fontsize=7.5)

    # contabiliza o tempo que demorou a efetuar o processamento
    end = time.time()
    print('Elapsed time: ', end - start, 'seconds')

    # mostra o grafico
    plt.show()
    # close plot
    plt.clf()


def parseWordsNLTK(html):
    # inicializacao do temporizador
    start = time.time()
    print('STARTED PARSING WITH NLTK')

    text = parseHTML(html)

    # utilizando o tokenizer do NLTK, obtem a lista de
    # palavras existentes na obra
    tokenizer = RegexpTokenizer('\w+')
    tokens = tokenizer.tokenize(text)

    return tokens,start

def getStopwordsNLTK(words):
    # obter as stopwords da corpora do nltk
    stopwords = nltk.corpus.stopwords.words('portuguese')
    print('stopwords: ', len(stopwords))
    # remover da lista de palavras aquelas que se encontram na lista
    # de stopwords
    clean_words = [word for word in words if word not in stopwords]
    print('clean words: ', len(clean_words))

    # obter o grafico de frequencias das palavras da lista
    freqdist = nltk.FreqDist(clean_words)

    return freqdist
```

```
def plotNLTK(freqdist,num,start):
    # finalizacao do temporizador, imprimindo o tempo de execucao
    end = time.time()
    print('Elapsed time: ', end - start, 'seconds')

    # desenhar o grafico
    freqdist.plot(num)
    # close plot
    plt.clf()


def main():
    # obter argumentos do script
    opts, args = getopt(sys.argv[1:], "mnlsu:N:p:")
    # colocar argumentos num dicionario
    dop = dict(opts)

    #html = 'https://www.gutenberg.org/files/55752/55752-h/55752-h.htm';

    # flag para o url da obra a processar
    if "-u" in dop:
        html = dop.get('-u')

    # para versao sem NLTK
    if "-m" in dop:
        words,start = parseWordsNoNLTK(html)
        # flag lower / ignore case
        if "-l" in dop:
            words = lower_res(words)
        # numero de stopwords a utilizar (100, 200 ou 300)
        if "-N" in dop:
            numSW = int(dop.get('-N'))
        # por default, usa 100
        else: numSW = 100
        # flag de stopwords
        if "-s" in dop:
            words = stopWordsNoNLTK(words,numSW)
        # flag p - numero de barras a mostrar no grafico
        # por default, sao 10
        if "-p" in dop:
            numP = int(dop.get("-p"))
        else: numP = 10
        plotNoNLTK(words,start,numP)

    # para versao com NLTK
    elif "-n" in dop:
        words,start = parseWordsNLTK(html)
        # flag lower / ignore case
        if "-l" in dop:
```

```
            words = lower_res(words)
        # flag de stopwords
        if "-s" in dop:
            freqdist = getStopwordsNLTK(words)
        else:
            freqdist = nltk.FreqDist(words)
        # flag p - numero de barras a mostrar no grafico
        # por default, sao 10
        if "-p" in dop:
            numP = int(dop.get('-p'))
        else: numP = 10
        plotNLTK(freqdist,numP,start)

# correr o script
main()
```

Referências

[1] Raheel Shaikh. "Gentle Start to Natural Language Processing using Python", Medium, Towards Data Science, 20 Oct. 2018, https://towardsdatascience.com/gentle-start-to-natural-language-processing-using-python-6e46c07addf3. Accessed 15 Nov. 2019.

[2] Kamil Mysiak. "NLP Part 1— Scraping the Web using BeautifulSoup and Python", Medium, 21 Mar. 2019, https://medium.com/@kamilmysiak/scraping-the-web-using-beautifulsoup-and-python-5df8e63d9de3. Accessed 15 Nov. 2019.

[3] Matheus Budkewicz. "Como fazer webscraping com Python e Beautiful Soup", Medium, 16 Aug. 2018, https://medium.com/horadecodar/como-fazer-webscraping-com-python-e-beautiful-soup-28a65eee2efd. Accessed 15 Nov. 2019.

[4] Hugo Bowne-Anderson. "Web Scraping NLP in Python", DataCamp, 13 Oct. 2017, https://www.datacamp.com/community/tutorials/web-scraping-python-nlp. Accessed 21 Nov. 2019.

[5] Leonard Richardson. Launchpad, 11 Nov. 2019, https://bazaar.launchpad.net/ leonardr/beautifulsoup/bs4/files/head:/doc/. Accessed 28 Nov. 2019.