MA5851 A3 Part2 - Web Crawler

December 5, 2021

1 Introduction

This notebook explains a pipeline for:

- Taking a URL as input
- Extracting all URLs found on that web page which match specified criteria
- Extracting the text on each of those sub-URLs
- Generating a data table of the url, the text and the first title located on that page
- Writing this data to a CSV, which is expected to then be used in a subsequent pipeline.

This is the abstract task which this pipeline accomplishes. In this scenario, we will configure the pipeline for a specific use case, which is:

- Input the URL for 'The Daily Mail UK' news publication's 'breaking News' page
- Extract every news article on the front page of the website
- Extract all text from each article, which will then be used in the 'NLP Analysis' notebook.

1.1 Use Case

In this use case, we are aiming to conduct a topic modelling and sentiment analysis on each article found by web scraping the Daily Mail's 'Breaking News' web page. We are interested to know if certain topics correlate with sentiment, which would indicate especially positive or negative discourse on that topic.

We are inputting in only the URL for the 'Breaking News - Daily Mail UK' web page. This pipeline will then automatically extract all URLs from that webpage and filter in only URLs which appear to be news articles (as opposed to URLs to images or ads). These news articles cover a range of topics including politics, news and entertainment. However, they are all 'current events' stories, meaning they are the 'headline' stories for the particular day this pipeline is run on. Therefore, this pipeline will produce different results when re-run.

While each news article web page contains many elements besides the text of the main story, the main story text is generally contained in predictable tags, in this case 'p' for 'paragraph text.' Undesired texts, such as previews for other stories, are usually wrapped in other tags.

1.1.1 Copyright considerations

The Daily Mail terms of use allow for non-commercial re-use of partial texts from their articles. The website also does not use cloudflare, which will allow us to conduct web scraping.(Daily Mail, 2011)

1.1.2 Metadata supplementation

This pipelines does more than extract links from a webpage, it also extracts the text found on each URL and provides it to us in a data table. We enrich this data in the following ways:
* Text cleaning by removing the following unwanted characters. Further cleaning is done in the subsequent pipeline with stop words and removing infrequent words which may be the result of residual html in the text strings.

- removed substrings are: "]",′"′,"′",".",",","[","/",">","<"
- In addition to extracting the raw text in the article, we also extract the title of the article by finding the first ′title′ tag that occurs on the page.
- The resultant data table of URL, text, title, are then stored in an output file. That file can then be consumed by the subsequent NLP pipeline.

1.1.3 Demonstration

What follows is a demonstration of setting up the required configurations for the pipeline (see the ′Configuration′ section) and then executing the pipeline with those configurations (see ′Execution′ section).

After executing the pipeline, we can also briefly review the data that had been outputted.

1.2 Extending and Scaling this Prototype

This pipeline contains functions and a template for a pipeline which can be used to accomplish this abstracted use case. This is intended as a prototype which could later be developed as:

- An application for extracting text from news articles on a series of news websites. This would require altering the current method which only evaluates URLs from the inputted webpage. This new implementation would extract every URL present on several domains which meet a specified criteria.
- Extracting news articles from across a time range, instead of just the news of today.
- This example will collected around 100-200 articles and takes approximately 2.5 minutes to run. The majority of computation time is from extracting text from the larger HTML text blobs. To scale the application to many thousands or millions of articles, could be accomplished by distributing out the processing for each sub-URL across parallel computers.

2 Configuration

In the following section, we:

- Import required modules
- Set the configuration constants, including the target URLs. This pipeline should be capable of performing the same task for different use cases by changing only the configurations. No changes to the code are required.
- Define the functions for use in the ′Execution′ section.
- Connect this runtime to file storage (Google Drive) to store the output file.

2.1 Modules

[ ]:
```python
import requests
from bs4 import BeautifulSoup
from pprint import pprint
from sklearn.pipeline import Pipeline
import pandas as pd
import re
import random
import os
import numpy as np
import time
```

2.2 Constants

The constants are the runtime variables which can be set here in this one code chunk. Future versions of this application could set these variables from an external config file. By changing the constants here, this pipeline can accomplish its task on a different use case, without any need to change code. These constants are:

- MAIN_URL - This is the inputted webpage from which we will extract the URLS found there. An example would be the home page of a news website.
- DOMAIN - URLs on pages are often partial urls and we will append this domain to the start to create the full URL.
- SCRAPE_OUTPUT_FILE - File location for the output file.
- ARTICLE_TAG = - the html tag we can use to identify when an element contains a URL we are interested in. For example 'a' tags contain articles, while other tags may contain URLS to ad sites.
- URL_HTML_TAG - Th etag which indicates the URL, this is commonly 'href'
- URLS_START_WITH - Filters in only URLS that start with this substring.
- URLS_NOT_END_WITH - FIlters out URLs that end with the substring.
- TEXT_TAG - The HTML tag which indicates the body text we want to extract.
- REMOVE_SUBSTRINGS List of characters and substrings which can be cleaned out of the extracted text.
- SEED - The random seed to be set for reproduceability.

The HTML tags and URL substrings to filter on, are determined by examining the HTML for the target pages directly. This will be demonstrated in the 'Execution' section below.

[ ]:
```python
MAIN_URL = 'https://www.dailymail.co.uk/news/breaking_news/index.html'
DOMAIN = 'https://www.dailymail.co.uk'
SCRAPE_OUTPUT_FILE = '/content/drive/MyDrive/MA5851_A3/scrape_results.csv'
ARTICLE_TAG = 'a'
URL_HTML_TAG = 'href'
URLS_START_WITH = ['https://www.dailymail.co.uk']
URLS_NOT_END_WITH = ['#video']
TEXT_TAG = 'p'
REMOVE_SUBSTRINGS = ["]",'"',"'",".",",","[","/",">","<"]
SEED = 42
```

2.3 File Storage

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

2.4 Functions

Where the imported modules' functionality isn't precisley suited for our use cases, it is appropriate to define some custome functions. These functions are re-useable for other use cases.

2.4.1 Utilities

[ ]:
```
def Set_All_Seeds(seed):
  """Aims to set all used random seeds in one place."""
  random.seed(seed)
  os.environ['PYTHONHASHSEED'] = str(seed)
  np.random.seed(seed)
  np.random.RandomState(seed)
```

2.4.2 HTML Element Selection

[ ]:
```
def Get_Soup(url: str):
  """Input a url and return the BeautifulSoup instance (aka: a 'soup')."""
  data = requests.get(url)
  html = BeautifulSoup(data.text, 'html.parser')
  return html
```

[ ]:
```
def Get_Links(soup: BeautifulSoup, find_tag: str, get_tag: str):
  """Extracts every URL found in the 'get_tag' of each 'find_tag' in the soup.
↪"""
  results = []
  for link in soup.find_all(find_tag):
    results.append(link.get(get_tag))
  return results
```

[ ]:
```
def Get_Content(soup: BeautifulSoup, tag: str):
  """Input a soup, return a list of strings which are the
  contents found between each tag"""
  results = []
  for p in soup.find_all(tag):
    results.append(p.contents)
  return results
```

2.4.3 Text Processing

[ ]:
```python
def Remove_Sub_Strings(string: str, remove: list):
  """User can input a list of substrings which will all be
  removed from the string"""
  for r in remove:
    assert isinstance(string, str)
    string = string.replace(r,"")
  return string

def Clean_String(s: str):
  """Cleans regex characters from string"""
  s = re.compile(r'<[^>]+>')
  return re.sub('(^|\s+)FIRST($|\s+)', '', s)

def Get_Text_From_Page(url: str):
  """Input a url and returns only the text found on the page.
  Reuires runtime variable 'TEXT_TAG' and 'REMOVE_SUBSTRINGS' to be defined."""
  web_text = Get_Content(soup = Get_Soup(url), tag=TEXT_TAG)
  return Remove_Sub_Strings(str(web_text), remove = REMOVE_SUBSTRINGS)
```

2.4.4 Link Selection

[ ]:
```python
def Select_Links_Starts_With(links: list, stem: str):
  """Input a list of URLS, returns the URLs which start with the stem"""
  results = []
  for link in links:
    if not isinstance(link, str):
      continue
    if link.startswith(stem):
      results.append(link)
  return results

def Select_Links_Ends_With(links: list, stem: str):
  """Input a list of URLS, returns the URLs which end with the stem"""
  results = []
  for link in links:
    if not isinstance(link, str):
      continue
    if link.endswith(stem):
      results.append(link)
  return results

def Remove_Links(func, links: list, stems: list):
  """Allows users to filter out URLs with a list of stems."""
  for s in stems:
    delta = func(links = links, stem=s)
    links = list(set(links) - set(delta))
  return links

def Append_Links(func, links: list, stems: list):
  """Allows users to filter in URLs with a list of stems."""
  results = []
  for s in stems:
    delta = func(links = links, stem=s)
    results.append(delta)
  return results[0]
```

3 Execution

[ ]:
```
Set_All_Seeds(SEED)
start_time = time.time()
```

3.1 Extract webpages

[ ]:
```
# Take the main URL and extract all desired URLs found on that webpage into a␣
  ↪list.
main_soup = Get_Soup(MAIN_URL)
URLs = Get_Links(soup = main_soup, find_tag = ARTICLE_TAG, get_tag =␣
  ↪URL_HTML_TAG)
URLs = Append_Links(func = Select_Links_Starts_With,links = URLs, stems =␣
  ↪URLS_START_WITH)
URLs = Remove_Links(func = Select_Links_Ends_With,links = URLs, stems =␣
  ↪URLS_NOT_END_WITH)
```

We can examine a sample of the main page HTML code here. From exploring the full HTML document, we can determine the tags and url stems needed to extract the desired URLS and set those porperties in the configuration.

[ ]:
```
str(main_soup)[0:300]
```

[ ]:
```
'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"//www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\n<html><head><script
type="text/javascript">\ntry {\n  Object.defineProperty(window, \'adverts\',
{configurable: false, value:{}});\n}\ncatch(error) {\n
console.error(error);\n}\n</script><lin'
```

Using the tags and stems we have inputted into the configuration, we can extract a list of URLs as follows. The number of URLs extracted will be detailed in the output section below.

[ ]:
```
URLs[0:4]
```

[ ]:
```
['https://www.dailymail.co.uk/news/article-10270907/Norway-Covid-60-infected-
Omicron-Christmas-party-mild-symptoms.html#comments',
 'https://www.dailymail.co.uk/news/article-10274357/Australian-cricket-coach-
Justin-Langer-Tim-Paine-sexting-scandal.html',
 'https://www.dailymail.co.uk/health/article-10268047/Second-case-Omicron-
COVID-19-variant-detected-Minnesota.html',
 'https://www.dailymail.co.uk/news/article-10272445/Biden-says-putting-plans-
deter-Putin-invading-Ukraine.html']
```

We can apply our custom function to extract all body text for each URL in the list. See in the below example, that some html code and unwanted sub strings are still in the text. These will be removed with stop words in the subsequent pipeline.'

[ ]:
```
#First 725 characters of article htmls are unhelful matter.
Get_Text_From_Page(URLs[1])[725:1500]
```

[ ]:
```
'31 View  br  comments  Australian cricket coach Justin Langer has gone in to
bat for former captain Tim Paine and knocked critics for six in the wake of the
ex-skippers sexting scandal\\xa0 Langer came out on the front foot to smash the
relentless attacks on Paine as a hypocritical pursuit of false perfectionism Our
captain one of the best made a mistake and is paying a heavy price for it Langer
slammed ahead of the first Ashes Test at the Gabba on Wednesday\\xa0 What I
continually see in this job and see in the society we live in  its brutal We
live in a world of perfectionism dont we? We are a very judgmental society\\xa0
Australian cricket coach Justin Langer (pictured) has slammed the attacks on Tim
Paine who stood down as Test captain in November after a sexting'
```

3.2 Extend the metadata with new features

From the list of URLs, we can develop more features and output a data table of:

- The URL
- The title of the article
- 'bag of words' which is the text in the article.

[ ]:
```
bag_of_words = []
for url in URLs:
  bag_of_words.append(Get_Text_From_Page(url)[600:])
```

[ ]:
```
titles = []
for url in URLs:
  titles.append(Get_Soup(url).find("title").contents[0])
```

[ ]:
```
output = pd.DataFrame({"URLS":URLs,"Bag Of Words":bag_of_words,"Title":titles}).
 ↪drop_duplicates()
output.to_csv(SCRAPE_OUTPUT_FILE)
execution_time = time.time() - start_time
```

3.3 Output Profiling

The data table has been written to the output file. We can finish this pipeline by providing some descriptive information about the corpus. The following is a preview of the data's first three rows.

[ ]:
```
output.head(3)
```

[ ]:
```
                                                  URLS  ...
Title
0  https://www.dailymail.co.uk/news/article-10270...  ...  Norway Covid: 60
infected with Omicron at Chri...
1  https://www.dailymail.co.uk/news/article-10274...  ...  Australian cricket
coach Justin Langer on Tim ...
2  https://www.dailymail.co.uk/health/article-102...  ...  Second US case of
Omicron COVID-19 variant det...

[3 rows x 3 columns]
```

Exploratory data analysis will be conducted on the corpus in the subsequent pipeline which performs NLP analysis. This pipeline is only responsible for extracting the text from the web pages. However, to ensure quaity data is sent to the next pipeline, we can explore:

- The number of articles extracted
- Execution time
- Check for rows missing text

The total number of articles extracted:

[ ]:
```
len(output)
```

[ ]:
```
155
```

The time taken to process this many articles (in seconds):

[ ]:
```
execution_time
```

[ ]:
```
5243.575374126434
```

Low word counts indicate a problem extracting text from HTML. We can check how many records have a small word count as follows.

[ ]:
```
ind = output["Bag Of Words"].str.len() < 100
len(output[ind])
```

[ ]:
```
0
```

4 Refferences

Reporter, A. Daily Mail. "Daily Mail - Terms." Mail Online, June 7, 2011. https://www.dailymail.co.uk/home/article-1388146/Terms.html.