SEP-crawl-to-md

October 30, 2024

1 Code: Crawling the *Stanford Encyclopedia of Philosophy* and conversion to markdown

[1]:
```
# Load required packages

import requests
from bs4 import BeautifulSoup, Comment, Tag, NavigableString
from markdownify import markdownify as md
import os
import re

# set and print current working directory
cwd = os.getcwd()
print("Current Working Directory:", cwd)
```

```
Current Working Directory: /Users/rna/Documents/Daten/Wissen_neu/Data Science
und KI/Projekte Python Programmieren/Jupyterhub Uni MS/Crawl SEP/Test
```

[2]:
```
# define relative paths for saving the crawled and processed data
main_data_path = os.path.join(cwd, 'SEP_data')
html_compl_path = os.path.join(main_data_path, 'html_complete')
html_path = os.path.join(main_data_path, 'html')
md_path = os.path.join(main_data_path, 'md')
```

[5]:
```
# Define functions

def debug(iterable, iterable_string, i=3):
    '''Prints the first i elements of a sliceable iterable and gives its total␣
 ↪number'''
    i=3; print(i, 'of', len(iterable), iterable_string + ':', iterable[:i],␣
 ↪'\n')

def count_files_in_directory(directory_path):
    # Use listdir to get all entries in the directory
    all_entries = os.listdir(directory_path)
    # Filter entries to count only files
    file_count = sum(1 for entry in all_entries if os.path.isfile(os.path.
 ↪join(directory_path, entry)))
```

```
    return file_count

def progress_table_setup(*args):
    """
    Setup for a table that shows the progress of the download
    """
    # Define the format strings for the number (as an empty placeholder here) and theme
    number_format = "{:<7}"
    theme_format = "{:<35}"

    # Create a series of centered format placeholders for the additional args
    dynamic_formats = " ".join(["{:^12}"] * len(args))

    # Construct the full row format by combining them
    row_format = number_format + theme_format + dynamic_formats

    # Print header: Unpacking `args` with `*args` to match the dynamic placeholders
    print(row_format.format("", "Theme", *args))  # '*' unpacks args tuple into separate arguments

    # Calculate the total width for the separator based on the components
    total_width = 7 + 35 + 12 * len(args)
    print('-' * total_width)
    return row_format


def progress_table_fill(i, theme, file_path, row_format):
    """
    Setup for a table that shows the progress of the download
    """
    # Check if the file exists
    if os.path.exists(file_path):
        print(row_format.format(i, theme, '\u2713'))
    else:
        print(row_format.format(i, theme, '\u2718'))


def list_filenames(directory, strip_suffix=""):
    """List filenames without extensions and optional suffix."""
    filenames = []
    for entry in os.listdir(directory):
        if os.path.isfile(os.path.join(directory, entry)):
            filename_without_extension = os.path.splitext(entry)[0]
            filenames.append(filename_without_extension)
```

```
    return filenames

def compare_directories(dir1, dir2, strip_suffix=""):
    """Compare filenames between two directories disregarding the specified␣
 ↪suffix, and list what's unique to the first."""
    files_dir1 = set(list_filenames(dir1, strip_suffix))
    files_dir2 = set(list_filenames(dir2, strip_suffix))

    unique_to_dir1 = files_dir1 - files_dir2

    return unique_to_dir1
```

1.1 Crawling

We start by downloading article pages from the SEP in HTML format.

Check the current license conditions before choosing how many articles to download in line 14 of the last cell in this section (starting with "max_links").

[8]:
```
# defining functions

def crawl(index_url, base_url, main_data_path, html_compl_path, max_links=5,␣
 ↪fresh=False):
    try:
        # Find the links to the content pages on the overview page

        # Send request to the index page
        response = requests.get(index_url)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, 'html.parser')

        # Identify the main content area, e.g., <div id="content">
        main_content = soup.find('div', id='content')  # Adjust this line based␣
 ↪on actual HTML structure

        def has_correct_href(tag):
            # Check if the tag is an <a> and if the href attribute starts with␣
 ↪'entries/'
            return tag.name == 'a' and tag.get('href', '').startswith('entries/
 ↪')

        if main_content:
            links = main_content.find_all(has_correct_href)

        else:
            print("No main content div found")
```

```
        short_titles = {link['href'].split('/')[1] for link in links}

        short_titles_html_compl = set(list_filenames(html_compl_path))

        print(len(short_titles_html_compl), 'of', len(short_titles), 'articles␣
↪already downloaded.')

        if fresh==False:
            short_titles_missing = short_titles - short_titles_html_compl
            delta = len(short_titles_missing)
            if delta > 0:
                print(delta, 'html-files to download (complementary download).')
            #print(short_titles_missing)

        elif fresh==True:
            short_titles_missing = short_titles
            delta = len(short_titles_missing)
            if delta > 0:
                print(delta, 'html-files to download (complete fresh download).
↪')

        def print_result(i, delta, fresh):
            if fresh == False:
                fresh_text = 'complementary download'
            else:
                fresh_text = 'complete fresh download'
            print(i, 'of', delta, 'html-files downloaded in this run (' +␣
↪fresh_text + ').')

        if len(short_titles_missing) == 0:
            #print('Nothing to do.')
            i=0
            print_result(i, delta, fresh)
        else:
            print('')
            print(f'Saving files in directory: {html_compl_path}', "\n")

            row_format = progress_table_setup('html_complete')

            #links = ["entries/" + s + "/" for s in short_titles_missing]
            #print(links)

            i=1
            # Use each link to crawl the content pages
            # Limit to max_links, remove slice [:max_links] for all links
            for short_title in list(short_titles_missing)[:max_links]:
                #extract short_title
```

```
                #short_title = link.split('/')[1]
                url = base_url + '/' + "entries/" + short_title + "/"

                # Load the target page
                page_response = requests.get(url)
                page_response.raise_for_status()

                # Extract the last part of URL as filename
                #filename_base = url.split('/')[-2]
                html_compl_file_path = os.path.join(html_compl_path,␣
 ↪short_title + '.html')
                #print(html_compl_file_path)

                # Save the html-file
                with open(html_compl_file_path, 'w', encoding='utf-8') as file:
                    file.write(page_response.text)

                # Check if the file exists
                progress_table_fill(i, short_title, html_compl_file_path,␣
 ↪row_format)
                i+=1
            print('\n')
            print_result(i-1, delta, fresh)
    except Exception as e:
        print(f"An error occurred: {e}")
```

[ ]:
```
# Parameter for SEP
index_url = "https://plato.stanford.edu/contents.html"
base_url = "https://plato.stanford.edu"

# Create folder if it does not exist
os.makedirs(html_compl_path, exist_ok=True)

# Set the maximum number of articles to download
# ! check the actual license conditions before making a setting
# max_links = n -> download n articles
# max_links = 0 -> download nothing
# max_links = None -> download all articles
max_links = 5

# Start the Crawling and Saving
crawl(index_url, base_url, main_data_path, html_compl_path, max_links,␣
 ↪fresh=False)
```

1.2 Parsing html

Here we extract those parts of the saved HTML files that contain the article.

[13]:
```
# Defining functions

def print_lines(str_var, lines=5):
    """Print the first n lines of a string variable; for debugging purposes"""
    str_lines = str_var.strip().split('\n')
    for line in str_lines[:lines]:
        print(line)

def extract_div_content(html_content, div_id='article'):
    """Extracts and returns content of the div with a specific id from the␣
 ↪given HTML content."""
    soup = BeautifulSoup(html_content, 'html.parser')
    div_content = soup.find('div', id=div_id)
    return str(div_content) if div_content else ''


def modify_html(html_content, tags_to_process):
    # Parse the HTML content using BeautifulSoup
    soup = BeautifulSoup(html_content, 'html.parser')

    # Process each specified tag in the HTML
    for tag_name in tags_to_process:
        tags = soup.find_all(tag_name)
        for tag in tags:
            # Grab the inner HTML und clean it from line breaks
            inner_html = tag.decode_contents()
            modified_html = inner_html.replace('\n', ' ').strip() #' '.
 ↪join(inner_html.replace('\n', ' ').split())
            tag.clear()  # Lösche den bestehenden Inhalt des Tags
            tag.append(BeautifulSoup(modified_html, 'html.parser'))  # Add the␣
 ↪cleaned html
    return str(soup)


def process_directory(input_dir, output_dir, max_files=5, div_id='article',␣
 ↪fresh=False):
    """Process all HTML files in the input directory and save extracted content␣
 ↪to the output directory."""
    # Ensure the output directory exists, if not, create it
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    #print(count_files_in_directory(output_dir), 'of',␣
 ↪count_files_in_directory(input_dir), 'articles already downloaded.')
    print(count_files_in_directory(output_directory), 'of',␣
 ↪count_files_in_directory(input_directory), 'files already converted␣
 ↪(html_complete to html).')
```

```
    i=1
    # Loop through all files in the input directory
    if fresh==False:
        filenames = compare_directories(input_directory, output_directory)
        filenames = {s + ".html" for s in filenames}
        delta = len(filenames)
        print('Complementary conversion:', delta, 'articles to convert.')

    elif fresh==True:
        filenames = os.listdir(input_dir)[:max_files]
        delta = len(filenames)
        print('Complete fresh conversion:', delta, 'to convert.')

    if len(filenames) == 0:
        print('Nothing to do.')
        return
    else:
        print(f'Saving files in directory: {output_dir}', "\n")

        row_format = progress_table_setup('html_complete')

        #for filename in filenames:
        for filename in [f for f in list(filenames)[:max_files] if not f.
↪startswith('.')]:
            file_path = os.path.join(input_dir, filename)
            with open(file_path, 'r', encoding='utf-8') as file:
                html_content = file.read()

            # Extract the div content
            html_content = extract_div_content(html_content, div_id)

            # Tags to condense text
            tags = ['p', 'td', 'li', 'dd', 'blockquote']
            html_content = modify_html(html_content, tags)

            # Define the output file path
            output_file_path = os.path.join(output_dir, filename)

            # Save the extracted HTML to a new file in the output directory
            with open(output_file_path, 'w', encoding='utf-8') as output_file:
                output_file.write(html_content)

            # Check if the file exists
            progress_table_fill(i, filename, output_file_path, row_format)
            i+=1
    return i-1
```

[ ]:
```
max_files = None
#cwd = os.getcwd()
input_directory = html_compl_path  # Directory with original HTML files
output_directory = html_path       # Directory where processed HTML files will␣
 ↪be saved

i = process_directory(input_directory, output_directory, max_files, fresh=False)

print("\nNumber of html-files generated in this run:", i)
```

1.3 Parsing markdown

This section converts the articles in HTML format to markdown format.

[19]:
```
def preprocess_html(html_content):
    """
    Replace <sub> tags with a format compatible for markdownify, here␣
 ↪Pandoc-style subscripts
    Transform <sub>text</sub> to ~text~
    """
    processed_html = re.sub(r'<sub>(.*?)<\/sub>', r'~\1~', html_content)
    return processed_html

def modify_sec_academic_tools(text, bullet_symbol):

    in_academic_tools_section = False
    output_lines = []

    lines = text.splitlines()

    for line in lines:
        if line.strip() == "## Academic Tools":
            in_academic_tools_section = True
        elif line.strip().startswith("## ") and in_academic_tools_section:
            in_academic_tools_section = False

        if in_academic_tools_section and line.startswith("> | --- | --- |"):
            continue  # Überspringt das Hinzufügen dieser spezifischen Zeile zu␣
 ↪output_lines

        elif in_academic_tools_section and line.startswith("> "):
            #1==1
            line = re.sub(r'^> \|\s*(.*?)\s*\|\s*(.*?)\s*\|', bullet_symbol +␣
 ↪r' \1: \2', line)  # Entfernen von "> " am Anfang der Zeile
        output_lines.append(line)

    text = '\n'.join(output_lines)
```

```
    return text


def modify_sec_related_entries(markdown_text, bullet_symbol):
    # Regex pattern to find the "## Related Entries" and all subsequent content␣
 ↪until the next "##" heading or end of document
    rel_entries_pattern = r'(## Related Entries\n)([\s\S]*?)(?=\n## |\Z)'

    # Using re.search to locate the "## Related Entries" section
    rel_entries_match = re.search(rel_entries_pattern, markdown_text)
    if rel_entries_match:
        header = rel_entries_match.group(1)  # This is the "## Related Entries"␣
 ↪line
        content = rel_entries_match.group(2)  # This is the content following␣
 ↪the header till the next "##" or EOF

        # Split on "|" and format each entry as a list item, ensuring not to␣
 ↪split wrongly and trim spaces
        entries = content.split("|")
        formatted_entries = [bullet_symbol + " " + entry.strip() + "\n" for␣
 ↪entry in entries if entry.strip()]

        # Replace the old section content with the newly formatted entries
        markdown_text = markdown_text.replace(content, ''.
 ↪join(formatted_entries))

    return markdown_text

def additional_headings(markdown_text, bullet_symbol):
    # Replace specific copyright line with '## Authors'
    markdown_text = re.sub(r'\[Copyright © .*?\]\(\.\./\.\./info\.html#c\) by',␣
 ↪'## Authors', markdown_text)

    # Insert headings around 'First published'
    markdown_text = re.sub(r'(\*First published.*)', r'## Publication␣
 ↪information\n\n### Publication date\n\n\1\n\n## Abstract', markdown_text)

    # Insert heading TOC
    markdown_text = re.sub(bullet_symbol + r'( 1\. .*)', r'## TOC\n\n' +␣
 ↪bullet_symbol + r'\1', markdown_text)

    # Move the '## Authors' section and lower its heading level
    authors_pattern = r'(## Author(s)[\s\S]*?)(?=## |\Z)'  # Assumes '##' marks␣
 ↪the start of a new section or end of the document
    authors_section = re.search(authors_pattern, markdown_text)
    if authors_section:
```

```
        authors_content = authors_section.group(0).replace('## Authors', '###␣
↪Authors')

        # Remove the original '## Authors' section
        markdown_text = re.sub(authors_pattern, '', markdown_text, count=1)

        # Find the location of '## Abstract' and insert the modified '###␣
↪Authors' section before it
        abstract_pattern = r'(## Abstract)'
        markdown_text = re.sub(abstract_pattern, f'{authors_content}\n\n\\1',␣
↪markdown_text)

    return markdown_text


def modify_overall_md(markdown_text):
        # Delete all lines containing only --- (concerns one line after the TOC)
        markdown_text = re.sub(r'^---\s*$', '', markdown_text, flags=re.
↪MULTILINE)

        # Remove any trailing whitespace and newline characters
        markdown_text = markdown_text.rstrip()

        # Normalize blank lines between elements
        # Strip whitespace from lines that only contain whitespace
        markdown_text = re.sub(r'^[ \t]+$', '', markdown_text, flags=re.
↪MULTILINE)

        # Reduce multiple blank lines to a single blank line everywhere
        markdown_text = re.sub(r'(\n){3,}', '\n\n', markdown_text)

        # Ensure two blank lines before any heading, but not affecting␣
↪subsequent lines after a heading
        markdown_text = re.sub(r'([^\n])(\n+)(#+\s[^\n]+)', lambda m: m.
↪group(1) + '\n\n\n' + m.group(3), markdown_text)

        # Ensure at least one blank line after each heading
        markdown_text = re.sub(r'(#+\s[^\n]+)(\n*)', r'\1\n\n', markdown_text)

        #markdown_text = re.sub(r'(?<!\n)\n(?!\n)', ' ', markdown_text)
        # Merge lines that start with "> ",  adjusting subsequent "> " lines.
        def merge_blockquotes(match):
            # Capture the whole block of "> " lines and remove additional "> "␣
↪at the start of the lines
            block = match.group(0)
            lines = block.split('\n')
```

```
            first_line = lines[0]
            subsequent_lines = [line[2:] if line.startswith('> ') else line for␣
↪line in lines[1:]]
            return ' '.join([first_line] + subsequent_lines)

        markdown_text = re.sub(r'((^|(?<=\n))> [^\n]*([\n]> [^\n]*)+)',␣
↪merge_blockquotes, markdown_text, flags=re.MULTILINE)

        # Merge subsequent non-empty lines, but protect pairs of lines starting␣
↪with "* " or "+ " from merging
        # This regex finds pairs of lines that do not start with "* " or "+ "␣
↪or "-" or are not blank.
        markdown_text = re.sub(r'(?m)^(.+?)\n(?!\t*\* |\t*\+ |\t*- )(?!$)',␣
↪r'\1 ', markdown_text)

        # Clean up multiple spaces that might have been introduced
        markdown_text = re.sub(r'[ ]{2,}', ' ', markdown_text)

        markdown_text = markdown_text.strip()

        return markdown_text
```

[21]:
```
def process_directory_md(input_dir, output_dir, max_files=5, fresh=False):
    """Process all HTML files in the input directory and save as markdown files␣
↪in the output directory."""
    # Ensure the output directory exists, if not, create it
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    print(count_files_in_directory(output_directory), 'of',␣
↪count_files_in_directory(input_directory), 'files already converted (html to␣
↪md).')

    i=1
    # Loop through all files in the input directory
    if fresh==False:
        filenames = compare_directories(input_directory, output_directory)
        filenames = {s + ".html" for s in filenames}
        delta = len(filenames)
        print('Complementary conversion:', delta, 'articles to convert.')

    elif fresh==True:
        filenames = os.listdir(input_dir)[:max_files]
        delta = len(filenames)
        print('Complete fresh conversion:', delta, 'to convert.')
```

```
    if len(filenames) == 0:
        print('Nothing to do.')
        return
    else:
        print(f'Saving files in directory: {output_dir}', "\n")

        row_format = progress_table_setup('html -> md')

        # Loop through the files in the input directory up to max_files
        for filename in [f for f in list(filenames)[:max_files] if not f.
↪startswith('.')]:

            if filename.lower().endswith('.html'):
                file_path = os.path.join(input_dir, filename)

                with open(file_path, 'r', encoding='utf-8') as file:
                    #print(file_path)
                    html_content = file.read()

                html_content = preprocess_html(html_content)

                page_soup = BeautifulSoup(html_content, 'html.parser')

                # Convert HTML to Markdown
                markdown_text = md(str(page_soup), heading_style="ATX", bullets␣
↪= '-')

                markdown_text = modify_sec_academic_tools(markdown_text,␣
↪bullet_symbol)
                markdown_text = modify_sec_related_entries(markdown_text,␣
↪bullet_symbol)
                markdown_text = additional_headings(markdown_text,␣
↪bullet_symbol)
                markdown_text = modify_overall_md(markdown_text)

                # Define output file path
                output_file_name = filename.replace('.html', '.md')
                output_file_path = os.path.join(output_dir, output_file_name)

                with open(output_file_path, "w", encoding="utf-8") as file:
                    file.writelines(markdown_text)

                # Check if the file exists
                progress_table_fill(i, filename.replace('.html', ''),␣
↪output_file_path, row_format)
                i+=1
```

```
    return i-1
```

[ ]:
```
max_files = None
fresh = True
bullet_symbol = '-'
#cwd = os.getcwd()
input_directory = html_path  # Directory with original HTML files
output_directory = md_path       # Directory where processed files will be saved

i = process_directory_md(input_directory, output_directory, max_files, fresh)

print("\nNumber of md-files converted in this run:", i)
```