DSCI 558: Building Knowledge Graphs

HW1 – crawling + information Extraction

Released: Jan 23, 2025

Due: Jan 30, 2025 @ 23:59 PST

Summary

In this homework, you will use web crawlers to collect web pages and extract data from the **Open Library** website using the Python library **Scrapy**. Then, you will extract structured information from the crawled data using **SpaCy** with rule-based information extraction.

Ground Rules

- This homework must be done individually. You can seek help with tools but ensure that the submitted work is your own.

Submission Instructions

Submit the following files/folders in a single `.zip` archive named `Firstname_Lastname_hw01.zip`:

- JSON-Lines files:
  - `Firstname_Lastname_hw01_booklist.jsonl`
  - `Firstname_Lastname_hw01_book.jsonl`
  - `Firstname_Lastname_hw01_author.jsonl`
  - `Firstname_Lastname_hw01_authorIE.jsonl`
- Source Code:
  - Your Scrapy Crawler
  - Information Extraction code
- `Firstname_Lastname_hw01_report.pdf`: A PDF file with ans

Task 1: Crawling (Scrapy) - 4 points

This task requires you to crawl the **Open Library** and collect data on books and authors from the **Forever Trending Books** section. Familiarize yourself with the Python library Scrapy. We highly recommend analyzing which elements to select from a webpage using the **Inspect** tool in the Chrome browser and the `scrapy shell <url>` command.

First, you will create a Scrapy project as described in the Scrapy tutorial. It will create a project directory with the following contents:

```
tutorial/
    scrapy.cfg            # deploy configuration file
    tutorial/             # project's Python module, you'll import your code from here
        __init__.py
        items.py          # project items definition file
        middlewares.py    # project middlewares file
        pipelines.py      # project pipelines file
        settings.py       # project settings file
        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

In this project, you will:

1. Define appropriate items (key-value pair for structured data) in `items.py`.
2. Create a file-writing pipeline in `pipelines.py`.
3. Create a spider Python file in the `spiders/` directory.
4. Set the parameters in `settings.py` to control the "politeness" of your crawler (e.g., throttling, user-agent).

You will start with the following seed URL:
https://openlibrary.org/trending/forever

You will crawl three pieces of information starting from this seed URL.

Task 1-1: Forever Trending Books (1 point)

Store the list of books in the **Forever Trending Books** section in ascending order of their rank based on appearance in `Firstname_Lastname_hw01_booklist.jsonl`.

Requirements:

- **Data to collect**: The top 100 books from the section.
- **Output format**: Each JSON line must contain:
    - `title` (string): Title of the book.

    - `author` (string): Name(s) of the author(s).
    - `rank` (int): The position of the book based on its order in the "Forever Trending Books" section.
    - `url` (string): The URL to the book's Open Library page.
- **Deduplication**: Ensure there are no duplicate entries in the file.
- **Ranking**: Assign ranks sequentially based on the appearance of books in the list.

Task 1-2: Book Information (1 point)

Crawl individual pages of each book listed in Task 1-1. From these pages, collect and store detailed information in `Firstname_Lastname_hw01_book.jsonl`.

Requirements:

- **Data to collect**: For each book, crawl the Open Library page and extract:
    - `title` (string): Title of the book.
    - `author` (string): Name(s) of the author(s).
    - `genres` (List[string]): List of genres associated with the book (if available).
    - `description` (string): A complete description of the book (ensure it does not end with an ellipsis `...`).
    - `published_year` (int): The year the book was first published.
- **Deduplication**: Ensure there are no duplicate entries in the file.

Task 1-3: Author Information (2 point)

Crawl individual pages of authors listed in Task 1-2. From these pages, collect and store author information in `Firstname_Lastname_hw01_author.jsonl`. If the author's Open Library page does not contain values for `birthDate`, `deathDate`, or `biography`, and there is a Wikipedia link available, follow that link to extract the missing information.

Requirements:

- **Data to collect**: For each author, crawl their Open Library page and extract:
    - `name` (string): Full name of the author.
    - `birthDate` (Optional[string]): Date of birth (if available).
    - `deathDate` (Optional[string]): Date of death (if applicable, otherwise `null`).
    - `biography` (Optional[string]): A complete biography of the author (ensure it does not end with an ellipsis `...`).
    - **Secondary Data Source**: Wikipedia (only for missing fields).
        - Follow the Wikipedia link provided on the Open Library page, if available.

        - Extract missing information (`birthDate`, `deathDate`, or `biography`) and update the author profile.

- **Fallback**: If neither Open Library nor Wikipedia provides the information, populate missing fields with `null`.
- **Deduplication**: Ensure there are no duplicate entries in the file.

Task 2: Information Extraction (SpaCy) - 4 points

Once you have crawled the data, the next step is to extract structured information from the unstructured biographies of authors in your `Firstname_Lastname_hw01_author.jsonl` file. To do this, you will use **SpaCy**, a powerful library for NLP tasks. You will write rules to extract specific attributes from the text using pattern-matching and dependency parsing.

Requirements:

- Extract the following keys from each biography:
    - `birthplace`: Where the author was born.
    - `genres`: Notable genres associated with the author.
    - `notable_books`: Titles of important books mentioned in the biography.
    - `awards`: Achievements or awards won by the author (e.g., Pulitzer Prize).
    - `education`: Educational institutions attended, if mentioned.

- **Implement two extractors for each attribute:**
    - **Lexical Extractors**: Based on surface text patterns or regular expressions. Lexical extractors can use only surface-level information or exact text matching. For example, a phrase like "was born in" activates when a sentence includes "X was born in Y." In SpaCy, this corresponds to attributes like:
        - `ORTH`, `TEXT`, `LOWER`, `IS_ALPHA`, `IS_TITLE`, `IS_STOP`, etc.

    - **Syntactic Extractors:** Using parts of speech (POS), dependency parse-tree, and named entity recognition (NER). Syntactic extractors go beyond surface text to analyze sentence structures. For example, matching subject-verb-object relationships or navigating dependency trees. In SpaCy, this corresponds to attributes like:

        - `POS`, `TAG`, `DEP`, `ENT_TYPE`, `LEMMA`, etc.

Task 3: Describe Your Solutions - 2 point

Write a report to describe your approach to Tasks 1 and 2. Include answers to the following questions:

1. In Task 1, what did you configure in `settings.py` to ensure polite crawling?
    - Example: Did you use `AUTOTHROTTLE_ENABLED`? Did you disable `ROBOTSTXT_OBEY`?
2. What lexical rules did you implement for Task 2, and why?
3. What syntactic rules did you implement for Task 2, and why?