Research Project Report

Extraction, cleaning and storage papers from NeurIPS

Mehdi Inane - Amélie Leroy - Paul Caucheteux
Master IASD - Université PSL
Data acquisition, extraction and storage

January 7, 2024

1 Context

This project aims to design a solution to create a database of Machine Learning research articles published in a given conference. We chose to concentrate on the Neural Information Processing Systems (NeurIPS) conference for its reputation in publishing works in state of the art machine learning, and for the structure of its API that makes it easier to extract article information via scraping.
Research articles contain a lot of significant information to be stored in a database, and the nature of the extracted information depends on the use case. One can witness for instance works that performed NLP tasks on research articles to unveil the trendy topics in each conference year, thus requiring to get access to the content of the articles. Other works rather focused on influent authors, or institutions, and needed access to the article metadata.

2 Challenge and contribution

Our main contribution is an automated data extraction and storage pipeline. Given a debut year and an end year, our program is able to build a dataset listing the information that we consider relevant for most downstream tasks. This information is :

- Article metadata : title, date of publication, url of the article content, abstract, and its location in our local machine
- content of the article, saved locally as a pdf file
- author information : first name, middle name (if it exists), last name and email address.
- citations of this article

This information is chosen such as to maximize the usability of the dataset for further tasks, and is also coherent with the format of the data provided by NeurIPS API : the metadata is available in HTML form, so we could perform basic extractions using BeautifulSoup and xPath. The PDF content is retrieved by performing get requests, and the author information was trickier to get. One could extract the author names from the articles metadata page, but this would not feature their emails and middle names. This information is rather present in the PDF content of the article, so this is not eligible to the simple data extraction tools we mentioned so far. Another tool was used to fetch the information directly from the PDF content, and will be presented later on with more detail : Grobid [1]. The same tool was used to extract the citations of the articles. We wanted to extract the institution information from the PDFs, but this was a challenge we could not overcome. As an example of use case, we have extracted the corresponding information of the 2021 conferences.

3 Technology and API used

3.1 Extracting article metadata

The extraction of the metadata followed two steps : crawling the NeurIPS API to fetch the HTML content, then extracting the relevant information from it. The advantage of the NeurIPS website structure is its division in dates. This allowed us to create such an automated solution that could build a full data extraction and storage pipeline given conference years provided by the user. The start crawling link is `https://papers.nips.cc/paper_files/paper/` . Then, one can add a year of interest in the end of the link to fetch the web page listing all of the articles of the year. The function yielding all the links of conference years is get_conference_links(). One the spider gets the content of a given conference year link, it extracts the links to access each of the published articles in that year by parsing from the HTML content of that page, using BeautifulSoup. The links all feature within an <a >tag having a title attribute of the value 'paper_title'. The link of the corresponding paper is fetched from the attribute 'href' of this tag. The corresponding function for this is get_papers_year(). Finally, the spider gets the link of a given article and fetches its metadata to save it in a custom class NipsPaper that stores the data. The following content is extracted from the link, using again BeautifulSoup and its html5lib parser :

- The paper title is present within the first <title >tag.
- the authors, publication date and pdf url are found within the <meta >tag, having an attribute name of values respectively of : citation_author, citation_publication_date, citation_pdf_url
- The abstract is always written as an h4 tag and its content is typically the first paragraph following it.
- We have also extracted the paper ID provided by NeurIPS, and it is typically used for creating a unique URL for accessing the PDF. Note that this is different than the article DOI, and we chose to omit it in the SQL construction as we chose our own indexing. This ID is extracted by performing string operations on the URL.

With the fetched URLs, we could download the articles using basic GET requests. The storage solution is an indexed file system. This aforementioned data (except the pdf content) is stored in JSON format, and will be processed later on to build the final datasets. The code for this process, given an article URL, is provided in the function get_paper_info(). The function for downloading the pdfs of the articles given the extracted JSON file is download_papers().

3.2 Extracting information from PDF research articles

The remaining information we need to extract is authors emails, and the citations of research articles. This information is featured nowhere in the API, but is present in the article content. Thus, we needed to use a solution that could parse information from PDF documents. Our use case made it possible for us, as we are dealing with research articles, mostly written using LateX. This is what motivated our choice of using GeneRation Of BIbliographic Data (GROBID), which leverages machine learning to extract relevant information from scientific publications. GROBID provides an API to convert the content of a PDF article as an XML string, and thus subsequent parsing solutions can be used to extract the information of interest. GROBID also provides several machine learning models depending on the task.

3.2.1 Extracting author information

In order to extract header information, one should connect to the localhost:8070/api/processFulltextDocument API and provide a URL to the PDF document of interest. The output provided by GROBID is an XML representation of the PDF content, and the data of interest in our case is the author information, that one could find as one of the children of sourceDesc/biblStruct/. This contains all of the information of authors in a relatively structured fashion, except for the institution / affiliation of the author. One example output that we could have is the following, which explains the difficulty of getting the right institution information for each author.

```
<biblStruct>
<analytic>
    \<author>
        <persName>
            <forename type="first">Neil</forename><forename type="middle">M T</forename><surname>Houlsby</surname></persName>
                <email>neilhoulsby@google.com</email>
    </author>
    <author>
        <persName><forename type="first">David</forename><forename type="middle">M</forename><surname>Blei</surname></persName>
        <email>david.blei@colombia.edu</email>
        </author>
    <author>
        <affiliation key="aff0">
        <orgName type="department">Google Research Zurich</orgName>
        <address><country key="CH">Switzerland</country>
        </address>
        </affiliation>
    </author>
```

Figure 1: Faulty XML structure to get affiliations

Instead, we get the first, middle (if applicable), and last names and the email address if present, using XPath. The corresponding author information extraction function is given by get_authors_info().

3.2.2 Extracting citations

In order to extract the citations, another model is used by GROBID and requires to connect to the following API link : localhost:8070/api/processReferences. Similar to author extraction, an XML version of the PDF content is extracted, containing relevant information about the citations of the paper. We chose to extract the title and authors of the citations, that we found using XPath following the document tree structure : biblStruct/title for the titles, biblStruct/author for authors.
Note that some articles could not be treated by GROBID [1], since their PDF is not supporting textual format. They were thus ignored from the pipeline.

4 Transformation and structure of the dataset

Following the extraction process, the obtained data was represented in JSON format. In this section, we delve into the subsequent steps of cleaning and structuring the dataset. It's noteworthy that, in certain instances, there are missing values for attributes such as email and middle name. As these details may not be present in the original PDFs, we opted to leave these fields empty only for entries where the corresponding information was absent, ensuring that the dataset remains accurate and reflective of the available data.

4.1 Cleaning of the dataset

The sole cleaning task required pertained to the abstract section of our dataset. Our goal was to eliminate specific LaTeX characters, such as *mathcalS* and *ell_1*. Additionally, we addressed issues like double slashes, space normalization, and the removal of dollar signs. We made the deliberate choice to retain mathematical formulas, as their removal would not contribute significantly. We also kept some mathematical symbols in LaTeX, for example, we retain *mathcalR*: $\mathbb{R}$, because there is not really a readable way to transform it.

4.2 Data Storage Solution

In our quest for an efficient data storage solution, we considered various options and ultimately chose SQL for its robust relational database capabilities. SQL provides a standardized and powerful way to manage and retrieve data, making it a suitable choice for our project.

To structure our data systematically, we opted to create four distinct tables, each serving a specific purpose:

- one for the metadata of articles (see Table 1)
- another for authors (see Table 2)
- a third for citations (see Table 3)
- one for the relationship between articles and authors (see Table 4)

Our decision to create separate tables for each entity allows us to avoid redundancies and maintain a clear, organized structure. The relational model of SQL enables us to establish meaningful connections between these tables, facilitating efficient querying and analysis of our data.

By leveraging SQL and this well-thought-out table design, we anticipate streamlined data management, ease of retrieval, and enhanced overall system performance.

Table 1: Relation Articles

ID	Title	Year	URL PDF	Abstract
INTEGER PRIMARY KEY	TEXT	INTEGER	TEXT	TEXT

Table 2: Relation Authors

ID	First Name	Family Name	Middle Name	Mail
INTEGER PRIMARY KEY	TEXT	TEXT	TEXT	TEXT UNIQUE

[1] https://proceedings.neurips.cc/paper_files/paper/2021/file/250dd56814ad7c50971ee4020519c6f5-Paper.pdf

Table 3: Relation Citations

article_ID	Title	Authors
INTEGER FOREIGN KEY REFERENCES articles(ID)	TEXT	TEXT

Table 4: Relation Writing

Article_	authors_ID
INTEGER FOREIGN KEY REFERENCES articles(id)	INTEGER FOREIGN KEY REFERENCES authors(id)

5 Unsuccessful Strategies and Methods Explored

5.1 Scrapy

Initially, we attempted to utilize Scrapy for crawling and retrieving authors, titles, years, and abstracts. However, we encountered an obstacle – Scrapy did not organize the results chronologically by year. Although it provided all the necessary information, the lack of chronological ordering led us to opt for Beautiful Soup.

5.2 Arxiv

We also explored the use of Scrapy to gather subjects, such as keywords like machine learning, from Arxiv. Unfortunately, this approach proved unsuccessful because Scrapy cannot be applied to the following page: `https://arxiv.org/search/?query=something&searchtype=all&source=header`. This particular page is crucial for searching articles based on criteria like titles.

6 Conclusion

This project focused on extracting, cleaning and storing data from NeurIPS conference papers. Our main achievement was developing an automated pipeline to build a complete dataset. Despite challenges, such as difficulties in extracting author's affiliation details from PDFs, managing elaborated APIs like Grobid and handling complex data structures, we successfully created a comprehensive and useful database. This work demonstrates the feasibility and utility of such an automated approach in the context of scientific data extraction.

In future developments, we could consider leveraging Hadoop for more efficient processing of large-scale PDF datasets. Additionally, employing a NoSQL database might offer enhanced flexibility and scalability for handling our diverse and evolving data requirements. Furthermore, future work could explore integrating more advanced natural language processing techniques to further enrich the dataset and enables a deeper understanding of the content and themes of the articles.

References

[1] *GROBID*. https://github.com/kermitt2/grobid. 2008–2023. swh: 1:dir:dab86b296e3c3216e2241968f0d63b68e