Text and data mining scientific articles with allofplos

Elizabeth Seiver[*], M Pacer[§], Sebastian Bassi[‡]

Abstract—Mining scientific articles is hard when many of them are inaccessible behind paywalls. The Public Library of Science (PLOS) is a non-profit Open Access science publisher of the single largest journal (*PLOS ONE*), whose articles are all freely available to read and re-use. allofplos is a Python package for maintaining a constantly growing collection of PLOS's 230,000+ articles. It also efficiently parses these article files into Python data structures. This article will cover how allofplos keeps your articles up-to-date, and how to use it to easily access common article metadata and fuel your meta-research, with actual use cases from inside PLOS.

Index Terms—Text and data mining, metascience, open access, science publishing, scientific articles, XML

Introduction

Why mine scientific articles?

Scientific articles are the standard mechanism of communication in science. They embody a clear way by which human minds across centuries and continents are able to communicate with one another, growing the total sum of knowledge. Scientific articles are unique resources, in that they are the material artifacts by which this cultural exchange is made concrete and persistent. They offer a unique source of insight into the history of carefully argued, hard-won knowledge. Accordingly because they are made of annotated text, they offer unique opportunities for well-defined text and data mining problems. Importantly, because PLOS represents the largest single journal in the history of publishing, it has collected an excellent corpus for this study, spanning seven journals that specialize in biology and medicine. Equally importantly, because PLOS is Open Access, the opportunity to use this data set is available to anyone capable of downloading and analyzing it. The allofplos library enables more people to do that more easily.

What is allofplos?

`allofplos` is a Python package for downloading and maintaining up-to-date scientific article corpora, as well as parsing PLOS XML articles in the JATS (Journal Article Tag Suite) [jats] format. It is available on PyPI [allofplospypi] as well as a GitHub repository [allofplosgh]. Many existing Python packages for parsing XML and/or JATS focus on defensive parsing, where the structure is assumed not to be reliable or the document is immediately converted to another intermediate format (often JSON) and XML is just a temporary stepping stone. allofplos uses lxml [lxml05], which is compiled in C, for fast XML parsing and conversion to familiar Python data structures like lists, dictionaries, and datetime objects. The intended audience is researchers who are familiar with scientific articles and Python, but may not be familiar with JATS XML. Other related tools include a parser from fellow Open Access publisher eLife [elife] as well as the Open Access subset for downloading OA articles in bulk from PubMed Commons (PMC) [pmc].

∗ *Corresponding author: elizabeth.seiver@gmail.com*
§ *Netflix*
‡ *Globant*

Functionality

The primary function of `allofplos` is to download and maintain a corpus of PLOS articles. To enable users to parse articles without downloading 230,000 XML files, allofplos ships with a starter directory of 122 articles (`starterdir`), and includes commands for downloading a 10,000 article demo corpus as well. The default path to a corpus is stored as the variable `corpusdir` in the Python program, and first checks for the environment variable `$PLOS_CORPUS` which overrides that default location. If you have used pip to install the program, specifying `$PLOS_CORPUS` will ensure that the article data will not be overwritten when you update the `allofplos` package, as the default location is within the package. (Forking/cloning the GitHub repository avoids this problem, because the default corpus location is in the `.gitignore` file.)

```python
import os
os.environ['PLOS_CORPUS'] = 'path/to/corpus_directory'
from allofplos import update
update.main()
```

Downloading new articles can also be accessed via the command line:

```
$ export PLOS_CORPUS="path/to/corpus_directory"
$ python -m allofplos.update
```

If no articles are found at the specified corpus location, it will initiate a download of the full corpus. This is a 4.6 GB zip file stored on Google Drive, updated daily via an internal PLOS server, that then is unzipped in that location to around 25 GB of 230,000+ XML articles. For incremental updates of the corpus, allofplos first scans the corpus directory for all DOIs (Digital Object Identifiers) [doi] of all articles (constructed from filenames) and compares that with every article DOI from the PLOS search API. The missing articles are then downloaded individually in a rate-limited fashion from links that are constructed using the DOIs. Those files are identical to the ones in the .zip file. The .zip file prevents users from needing to scrape the entire PLOS website for the XML files, and "smartly" scrapes only the latest articles. For a subset of provisional articles called "uncorrected proofs", it checks whether the final version is available, and downloads the updated version if so. The files are then ready for parsing and analysis.

Article corpora and parsing

To initialize a corpus (defaults to `corpusdir`, or the location set by the `$PLOS_CORPUS` environmental variable), use the `Corpus` class. This points allofplos at the directory of articles to be analyzed.

```
from allofplos import Corpus
corpus = Corpus()
```

To analyze the starter directory, also import `starterdir` and set `corpus = Corpus(starterdir)`. The number of articles in the corpus can be found with `len(corpus)`. The list of every DOI for every article in the corpus can be found at `corpus.dois`, and the path to every XML file in the corpus directory at `corpus.filenames`. To select a random Article object, use `corpus.random_article`. To select a random list of ten Article objects, use `corpus.random_sample(10)`. You can also iterate through articles as such:

```
for article in corpus[:10]:
    print(article.title)
```

Because DOIs contain semantic meaning and XML filenames are based on the DOI, if you systematically loop through the corpus, it will not be a representative sample but rather will implicitly progress first by journal name and then by publication date. The iterator for `Corpus()` puts the articles in a random order to avoid this problem.

The `Article` class

As mentioned above, you can use the Corpus class to initialize an Article() object without calling Article directly. An Article takes a DOI and the location of the corpus directory to read the accompanying XML document into lxml.

```
art = Article('10.1371/journal.pcbi.1004692')
```

The lxml tree of the article is memoized in `art.tree` so it can be repeatedly called without needing to re-read the XML file.

```
>>> type(art.tree)
lxml.etree._ElementTree
```

Article parsing in `allofplos` focuses on metadata (e.g., article title, author names and institutions, date of publication, Creative Commons copyright license [cc], JATS version/DTD), which are conveniently located in the `front` section of the XML. We designed the parsing API to quickly locate and parse XML elements as properties without needing to know the JATS tagging format.

```
>>> art.doi
'10.1371/journal.pcbi.1004692'
>>> art.title
'Ensemble Tractography'
>>> art.journal
'PLOS Computational Biology'
>>> art.pubdate
datetime.datetime(2016, 2, 4, 0, 0)
>>> art.license
{'license': 'CC-BY 4.0',
 'license_link':
     'https://creativecommons.org/licenses/by/4.0/',
 'copyright_holder': 'Takemura et al',
 'copyright_year': 2016}
>>> art.dtd
'JATS 1.1d3'
```

For author information, `Article` reconciles and combines data from multiple elements within the article into a clean standard form, including author email addresses and affiliated institutions. Property names match XML tags whenever possible.

Using XPath

While the Article class handles most basic metadata within the XML files, users may also wish to analyze the content of the article more directly. The XPath query language is built into lxml and provides a way to search for particular XML tags or attributes. (Note that XPath will always return a list of results, as element tags and locations are not unique.) You can perform XPath searches on `art.tree`, which also works well for finding article elements that are not Article class properties, such as the acknowledgments, which have the tag `<ack>`.

```
>>> acknowledge = art.tree.xpath('//ack/p')[0]
>>> acknowledge.text[:41]
'We thank Ariel Rokem and Jason D. Yeatman'
```

For users who are more familiar with XML or want to perform quality control checks on XML files, XPath searches can find articles that match a particular XML structure. For example, PLOS's production team needed to find articles that had a `<list>` item anywhere within a `<boxed-text>` element. They iterated through the corpus using `art.tree.xpath('//boxed-text//list')`.

Use case: searching Methods sections

We can put these pieces together to make a list of articles that use PCR (Polymerase Chain Reaction, a common molecular biology technique) in their Methods section (`pcr_list`). The body of an article is divided into sections (with the element tag `<sec>`) and the element attributes of Methods sections are either `{'sec-type': 'materials|methods'}` or `{'sec-type': 'methods'}`. In addition to importing allofplos, the `lxml.etree` module needs to be imported to turn XML elements into Python strings via the `tostring()` method.

```
import lxml.etree as et
pcr_list = []
for article in corpus.random_sample(20):

    # Step 1: find Method sections
    methods_sections = article.root.xpath(
        "//sec[@sec-type='materials|methods']")
    if not methods_sections:
        methods_sections = article.root.xpath(
            "//sec[@sec-type='methods']")

    for sec in methods_sections:

        # Step 2: turn the method sections into strings
        method_string = et.tostring(sec, method='text',
                                    encoding='unicode')

        # Step 3: add DOI if 'PCR' in string
        if 'PCR' in method_string:
            pcr_list.append(article.doi)
            break
        else:
            pass
```

Included SQLite database

The *allofplos* code includes a SQLite database with all articles in starter directory. In this release there are 122 records that represents a wide range of papers. In order to use the database, the user needs a SQLite client. The official client is command line based and can be downloaded from https://www.sqlite.org/download.html. The database can also be displayed on graphical viewers such as DB Browser for SQLite and SQLiteStudio. There are also some options to query the database online, without installing any software, like https://sqliteonline.com/ and http://inloop.github.io/sqlite-viewer/.

The main table of the database is *plosarticle*. It has the DOI, title, abstract, publication date and other fields that link to other child tables, like *articletype* and *journal_id*. The corresponding author information is stored in the *correspondingauthor* table and is linked to the *plosarticle* table using the relation table called *coauthorplosarticle*.

For example, to get all papers whose corresponding authors are from France:

```
SELECT DOI FROM plosarticle
JOIN coauthorplosarticle ON
coauthorplosarticle.article_id = plosarticle.id
JOIN correspondingauthor ON
(correspondingauthor.id =
coauthorplosarticle.corr_author_id)
JOIN country ON
country.id = correspondingauthor.country_id
WHERE country.country = 'France';
```

This will return the DOIs from three papers from the starter database:

```
10.1371/journal.pcbi.1004152
10.1371/journal.ppat.1000105
10.1371/journal.pgen.1002912
10.1371/journal.pcbi.1004082
```

The researcher can avoid using SQL queries by using the included Object-relational mapping (ORM) models. The ORM library used is *peewee*. A file with sample queries is stored in the repository with the name of allofplos/dbtoorm.py. Part of this file defines all Python classes that corresponds to the SQLite Database. These class definitions are from the beginning of the file until the comment marked as `# End of ORM classes creation.`

After this comment, there is an example of how to build a query. The following query is the *peewee* compatible syntax that constructs the same SQL query as outlined before:

```
query = (Plosarticle
      .select()
      .join(Coauthorplosarticle)
      .join(Correspondingauthor)
      .join(Country)
      .join(Journal,
            on=(Plosarticle.journal == Journal.id))
      .where(Country.country == 'France')
      )
```

This will return a *query* object. This object can be walked over with a for loop as any Python iterable:

```
for papers in query:
  print(papers.doi)
```

SQLite database constructor

There is a script at allofplos/makedb.py that can be used to generate the SQLite Database from a directory full of XML articles. This script was used to generate the included **starter.db**. If the user wants to make another version, from another subset (or from the whole corpus), this script will be useful.

To generate a SQLite DB with all the files currently in the *Corpus* directory, and save the DB as *mydb.db*:

```
$ python makedb.py --db mydb.db
```

There is an option to generate a DB with only a random subset of articles. For a DB with 500 articles randomly selected, use:

```
$ python makedb.py --random 500 --db mydb.db
```

Future directions

We also have plans for future updates to allofplos. First, we plan to make the article parsing publisher-neutral, allowing for reading JATS content from other publishers in addition to PLOS. Second, we want to improve incremental corpus updates so that all changes can be downloaded and updated via a standardized mechanism such as a hash table. This includes 'silent republications', where articles are updated online without an official correction notice (the substance of the article is unchanged, but the XML has been updated). While the local allofplos server has methods for catching these changes and updating the zip file appropriately, there is not currently a way to make sure a user's local corpus copy reflects all of those changes. Third, we want to expand the possibilities of multiple corpora and allow for article versioning, such as for comparing older and newer versions of articles instead of just replacing them entirely. And finally, we want to expand and integrate the functionality of the sqlite database so that selecting a subset of articles based on metadata criteria such as journal, publication date, or author is faster and easier than looping through each XML file individually.

Conclusions

As more scientific articles are published, it will become more important that these articles can be analyzed in aggregate. Tools like `allofplos` make such an effort much easier. With an intuitive and straightforward `Corpus()` and `Article()` APIs, `allofplos` avoids much of the complexity of parsing xml for new users, while still enabling XML experts the flexibility and power needed to accomplish their aims. By building in the ability to automatically update and maintain the corpus, people can trust that they have the most state-of-the-art data without needing to manually check the >230,000 articles (a task few would undertake). By connecting this information to database technologies, `allofplos` enables quickly accessing data when that efficient access is needed. By making strides in all of these directions `allofplos` demonstrates itself to be a valuable tool in the scientific python toolkit.

References

[lxml05] Behnel, S., Faassen, M. et al. (2005), lxml: XML and HTML with Python, http://lxml.de.

[cc] Creative Commons Licenses. https://creativecommons.org/licenses/

[allofplosgh] allofplos GitHub repository. https://github.com/PLOS/allofplos

[allofplospypi] allofplos PyPI repository. https://pypi.org/project/allofplos/

[jats] JATS NIH/NISO standard. https://jats.nlm.nih.gov/publishing/tag-library/1.1d3/chapter/how-to-read.html

[elife] elife-tools GitHub repository. https://github.com/elifesciences/elife-tools

[doi] Digital Object Identifiers. https://www.doi.org/doi_handbook/1_Introduction.html

[pmc] PMC Open Access Subset. https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/