Journal of Informetrics 11 (2017) 176–197

Contents lists available at ScienceDirect

Journal of Informetrics

journal homepage: www.elsevier.com/locate/joi

Regular article

Introducing *metaknowledge*: Software for computational research in information science, network analysis, and science of science☆

John McLevey[a,*], Reid McIlroy-Young[b]

[a] *University of Waterloo, Canada*
[b] *University of Chicago, United States*

ARTICLE INFO

Article history:
Received 20 July 2016
Received in revised form
14 December 2016
Accepted 14 December 2016

Keywords:
Informetrics
Scientometrics
Bibliometrics
Networks
Computational
Big data
Software
RPYS
Gender
Topic models
Burst analysis
Python

ABSTRACT

metaknowledge is a full-featured Python package for computational research in information science, network analysis, and science of science. It is optimized to scale efficiently for analyzing very large datasets, and is designed to integrate well with reproducible and open research workflows. It currently accepts raw data from the Web of Science, Scopus, PubMed, ProQuest Dissertations and Theses, and select funding agencies. It processes these raw data inputs and outputs a variety of datasets for quantitative analysis, including time series methods, Standard and Multi Reference Publication Year Spectroscopy, computational text analysis (e.g. topic modeling, burst analysis), and network analysis (including multi-mode, multi-level, and longitudinal networks). This article motivates the use of *metaknowledge* and explains its design and core functionality.

1. Introduction

Researchers in information science, network analysis, and science of science currently have access to an unprecedented volume of data. Researchers are increasingly working with datasets that include millions of observations (e.g. Börner, 2010, 2015; Boyack, Klavans, & Börner, 2005; Evans & Foster, 2011; Foster, Rzhetsky, & Evans, 2015; Rzhetsky, Foster, Foster, & Evans, 2015; Shi, Foster, & Evans, 2015; Sinatra, Deville, Szell, Wang, & Barabási, 2015; Skupin, Biberstine, & Börner, 2013; Sugimoto, Lariviere, Ni, Gingras, & Cronin, 2013; Uzzi, Mukherjee, Stringer, & Jones, 2013; Wang, Song, & Barabási, 2013). In 2015, there were more than 3.8 million records indexed in ProQuest Dissertations and Theses, more than 23 million in

☆ Thanks to Jillian Anderson, Steven McColl, Alexander Graham, Amelia Howard, and Pierson Browne for comments on an earlier version of this manuscript. Jillian Anderson developed the Javascript library *mkD3* that enables interactive visualizations of *metaknowledge* datasets. She is a *metaknowledge* developer on releases >3.1. *metaknowledge* is funded in part by a Social Science and Humanities Research Council of Canada grant and an Early Researcher Award from the Ministry of Research and Innovation in Ontario, both awarded to Dr. John McLevey.

* Corresponding author.
E-mail address: john.mclevey@uwaterloo.ca (J. McLevey).

http://dx.doi.org/10.1016/j.joi.2016.12.005
1751-1577/

PubMed, more than 60 million in Scopus, and the number of cited references indexed in the Web of Science surpassed 1 billion. The Scholarly Database – hosted by researchers at Indiana University – currently contains over 25 million records (LaRowe, Ambre, Burgoon, Ke, & Börner, 2009). The text and network datasets that can be extracted from these databases are often enormous. As de Solla Price (1963) predicted, we are in a period of abundant data, and more is being produced all the time.

In addition to being "bigger" than they used to be, bibliometric datasets are becoming more complex as researchers link them with data from online repositories, social media, blogs, surveys, and administrative data from institutions, granting agencies, and governments (e.g. Cronin & Sugimoto, 2014; Haustein, Peters, Sugimoto, Thelwall, & Larivière, 2014; Kronegger, Mali, Ferligoj, & Doreian, 2012; Sugimoto et al., 2013). Making the most of this abundant data requires access to sufficient infrastructure and software that scales efficiently, reduces opportunities for human error, and is compatible with open and reproducible workflows. Using these tools appropriately requires computing skills that have not traditionally been necessary for conducting sophisticated research on the structure, evolution, and content of science.

There are currently many excellent software options for constructing and analyzing bibliometric datasets, small or large. There is specialized software for historical bibliometrics (e.g. Garfield's (2009) HistCite, Van Eck and Waltmen's (2014) CiteNetExplorer, Thor, Marx, Leydesdorff, and Bornmann's (2016) CRExplorer, and Comins and Leydesdorff's (2016b) RPYS i/o) and for mapping the topic and network structures of science (e.g. Van Eck and Waltmen's (2010) VOSViewer, Chen's (2006) CiteSpace, and WoS2Pajek for Pajek (De Nooy, Mrvar, & Batagelj, 2011)). Katy Börner and her collaborators developed Sci^2 and the Network Workbench (NWB) as modular "plug and play" programs, intended to be collaboratively developed by scientometric researchers as the field evolves (Börner, 2011). All of these programs have their own parsers for converting raw data files into something useful for bibliometric and scientometric research. Most tend to focus on very specific research ends (e.g. creating topic maps) and attempt to cover an entire research workflow from parsing raw data to producing graphs intended for publication. They are all primarily graphical user interfaces (GUIs) with drop down menus that require repetitive user input.[1]

GUI systems dramatically lower the barriers to conducting bibliometric and scientometric research, but many of the most exciting and promising developments in the field require computing workflows that are better suited to scripted data analysis, for example in R, Python, or Stata. Almost all research workflows include many small sequential tasks, some of which have to be repeated many times. A GUI program can require hours of tedious and error prone user input every time the workflow is executed. This is a waste of researcher time and effort. It could be automated and made reproducible with data cleaning and analysis scripts. While we fully support efforts to empower as many researchers as possible to leverage access to data and computing power to advance research in information science, network analysis, and science of science, there is a trade-off. GUI software plays a central role in research, but we also require software that is optimized for scalability, speed, reproducibility, easily linking open data, and open workflows.[2]

This article introduces *metaknowledge*, a Python package for computational research in information science, network analysis, and science of science.[3] The package name is adopted from Evans and Foster's (2011) brief article in *Science*. In short, it accepts raw data inputs from the Web of Science, PubMed, Scopus, Proquest Dissertation and Theses, and administrative data from some funding agencies. It outputs tidy datasets for a wide range of quantitative analyses, including but not limited to longitudinal analysis, Standard and Multi Reference Publication Year Spectroscopy (RPYS), computational text analysis (e.g. topic modeling, burst analysis), and network analysis (including multi-mode, multi-level, and longitudinal networks). Although *metaknowledge* is aimed at researchers with some programming knowledge, who are working with large and complex bibliometric datasets and / or who are committed to open and reproducible research, it fits into any research workflow in bibliometrics and scientometrics. In the sections below, we discuss the design and core functionality of *metaknowledge*, explain how to get started, and demonstrate some of its most useful functions.

2. Design and general overview

metaknowledge was designed with open and reproducible research workflows in mind. First, it is open source (General Public License 2). All source code is easily available online, enabling other researchers to make modifications that are useful in their own work, such as by adding custom parsers to process administrative data from institutions in their own country. Second, as a Python package, *metaknowledge* is scriptable, meaning researchers write small amounts of code to process and analyze their data. These scripts can be re-run anytime, and all revisions can be tracked using version control systems such as *git* and hosted on online platforms such as Github, GitLab, or the Open Science Framework. Analyses can be automated using clearly documented dependencies between files, for example by using Makefiles

[1] One exception is Gagolewski's (2011) CITAN package for R, which is primarily focused on impact assessment, e.g. computing *h* index and *g* index.

[2] The availability of sophisticated libraries in R (e.g. statnet suite and igraph) and packages in Python (e.g. networkx), for example, has been enormously productive for social networks researchers despite the fact that GUIs like UCINet, Pajek, Visone, and Gephi are widely used.

[3] We chose to make *metaknowledge* a Python package because Python excels at cleaning and manipulating strings and is well-suited for intensive research computing.

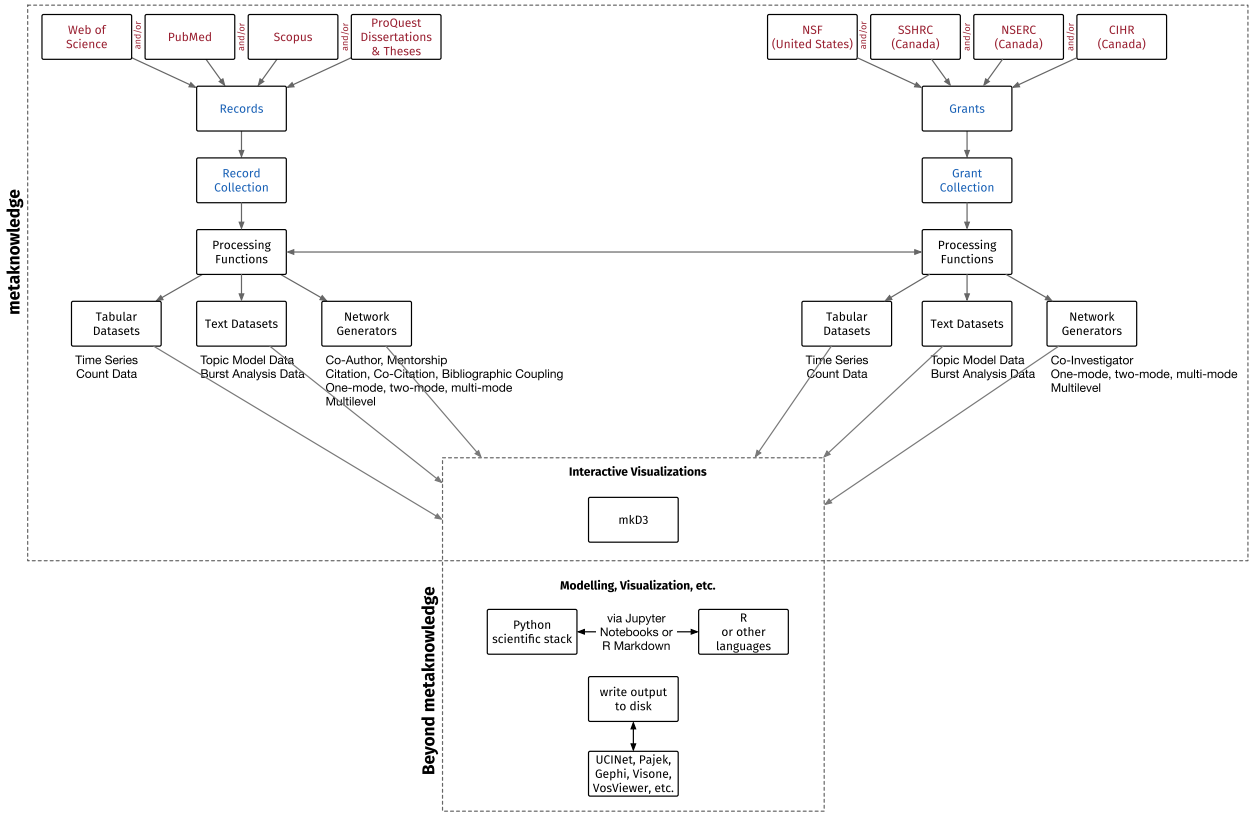


Fig. 1. Overview of information flow in *metaknowledge*. More technically-oriented readers may also be interested in the class hierarchy. A graph of the inheritance structure is available.

(Stodden, Leisch, & Peng, 2014).[4] Researchers can write their code and text alongside one another in Jupyter Notebooks (discussed below) to make their work reproducible. In short, *metaknowledge* fits comfortably into research workflows that are as open and reproducible as the researcher wants them to be (see Healy, 2011, on choosing workflow applications).

Relatedly, *metaknowledge* was designed to interface seamlessly with other widely-used data analysis software. Researchers can mix languages like Python and R in Jupyter Notebooks or R Markdown files, or they can write data to disk in formats designed for seamless import into other software, such as Pajek, Visone, Gephi, or UCINet. The basic idea, as shown in Fig. 1, is that *metaknowledge* should primarily be used to improve the extensive dataset construction and processing workflows that precede any modeling or other data analysis. In other words, it is primarily focused on the 80% of effort that goes into low-level data work (Light, Polley, & Börner, 2014) rather than tools for higher-level data analysis. Once this low-level work is done, *metaknowledge* offers tidy datasets for a wide variety of quantitative projects, including those using computational methods of text analysis or network analysis.

We have designed *metaknowledge* to be as easy to use as possible. We chose Python in part because it is an open source general purpose language designed to be easily human readable, and it is currently the most widely taught programming language in introductory computer science classes.[5] Although *metaknowledge* is more challenging to learn than GUI programs, we have intentionally designed it to have a low threshold and a high ceiling; while a little Python knowledge goes a long way, very little is required to use the most important features of *metaknowledge*.

Finally, *metaknowledge* works efficiently with very large datasets. It is able to dramatically outperform all other currently available software for working with bibliometric data. Custom data structures and caching make computing more efficient, dramatically reducing the time required to cycle through research workflows. Paired with the "low threshold and high ceiling" design approach, *metaknowledge* saves on both programming and processing time. This makes it well-suited to datasets of all sizes.

[4] Karl Broman has a useful blog post about using Makefiles for reproducible research. It is available here: http://kbroman.org/minimal_make/.

[5] A tutorial on Python programming itself is, of course, well-beyond the scope of this article. McKinney (2012) provides an overview of Python for data analysis, and there are many high-quality tutorials online.

3. Getting started

3.1. Installing Python, the Scientific Stack, and metaknowledge

As *metaknowledge* is a Python 3 package, the first step is to install Python 3. We strongly recommend downloading the Python 3 version of the Anaconda open data science platform from Continuum Analytics. It includes IPython (for interactive computing) and a variety of packages that are useful when working with data. This is a simple click-through installation with good default settings.

Once Python is installed, the most current official release of *metaknowledge* can be installed and upgraded using the native package manager for Python 3, `pip3`.[6] On the command line,[7] simply type `pip3 install metaknowledge`. It is also possible to update *metaknowledge* to the current version by typing `pip3 install metaknowledge --upgrade`. In addition to *metaknowledge* and the packages already included in the Anaconda distribution, researchers should use `pip3` to install *networkx*, *NLTK*, and, optionally, the data visualization package *seaborn*, which produces graphs with reasonable defaults (e.g. color-blind friendly palettes).

3.2. Writing metaknowledge code in Jupyter Notebooks

Many researchers will likely use *metaknowledge* by writing Python scripts saved in their project directory. Those scripts can be run from the command line and all relevant output – such as graphs and datasets – can be written to disk. This requires little knowledge of Python beyond the *metaknowledge* code introduced in this article. The researcher simply needs to have a basic idea of how to write and run a data analysis script. Researchers who write data analysis scripts in R, Stata, or SAS, to list just a few examples, will be familiar with this workflow.

Alternatively, researchers can weave together code and text in a Jupyter Notebook, which runs in a web browser as a light server process (Pérez & Granger, 2007). The Notebooks can easily be run locally (e.g. on a laptop) or on a remote server, which is especially useful for researchers working with very large datasets or running computationally-intensive analyses. The notebook server is started by typing `jupyter notebook` on the command line.[8] A web browser will open, showing files in the current directory. Getting started is as simple as clicking the drop down menu to start a new Python 3 notebook.

The process of working in Jupyter Notebooks will be familiar to any researcher who has worked in RStudio, especially those who make use of tools like *knitr* (Xie, 2015) to weave together text and code into a single dynamic document. Jupyter Notebooks are structured as a collection of cells containing either text or executable code.[9] Much like the interactive paradigm that dominates R programming, the code cells in a Jupyter Notebook can be run one at a time, enabling a researcher to iteratively check and revise her code as she works. They include useful tools for writing code, including integrated help documents and tab-completion. Working in a notebook has the added benefit of being able to effectively display dataframes and graphs inside the Notebook itself. We have written a series of Jupyter Notebooks as companions to this article, although the code in this article can also be typed into a standalone Python script. Researchers can run all of the code in this article using the same data we use. The Notebooks and code are available at https://github.com/mclevey/metaknowledge_article_supplement.git.

The first code cell in a Jupyter Notebook (or the first few lines of a script) usually imports the packages that will be used in the script. `import`, which is equivalent to the `library()` command in R, loads the packages into Python, providing access to their functions. We will import *metaknowledge*, *matplotlib*, *networkx*, *seaborn*, and *pandas*. To simplify our code, we will call *metaknowledge* `mk`, *networkx* `nx`, and so on. The line `%matplotlib inline` makes it possible to view graphs inline in Jupyter Notebooks and should not be used in scripts.

```
%matplotlib inline

import metaknowledge as mk
import matplotlib.pyplot as plt
import networkx as nx
import seaborn as sns
import pandas
```

We can now set the working directory, in this case to the directory above the one where our Jupyter Notebook is stored (using `..`).

```
import os
os.chdir('..')

# change default font size in seaborn plots
sns.set(font_scale=.7)
```

[6] It is possible that future releases of *metaknowledge* may introduce changes to core functionality. The version described in this article is 3.1.1. This version is available from https://github.com/networks-lab/metaknowledge/releases.

[7] Accessible with `cmd.exe` on Windows, `Terminal.app` on Mac OS, and various ways on Linux.

[8] Jupyter is part of the Anaconda installation and so is already available to researchers who have installed Python this way.

[9] In addition to Python, Jupyter Notebooks are capable of executing code in a range of other languages, including R.

4. Creating and processing record collections

metaknowledge currently reads plain text files from Web of Science, PubMed, Scopus, and ProQuest Dissertation and Theses.[10] It also accepts files containing data on grants from the National Science Foundation (NSF) and the Canadian tri-council agencies: the Social Science and Humanities Research Council of Canada (SSHRC), the National Engineering and Research Council of Canada (NSERC), and the Canadian Institutes of Health Research (CIHR). Since *metaknowledge* is open source, researchers can add support for administrative data from their own institutions and countries by writing custom data parsers.

Researchers can load individual files or an entire directory into *metaknowledge*. For example, the code below scans a directory containing plain text files downloaded from the Web of Science. Starting from the working directory we defined previously, the path to our data is `raw_data/imetrics/`. These files contain the meta-data for 8,140 articles published between 1978 and 2015 in *Journal of the Association for Information Science and Technology*/*Journal of the American Society for Information Science and Technology*, *Scientometrics*, and *Journal of Informetrics*. We can create a `RecordCollection` and call it anything we want; in this case, we will call it `RC`. `RecordCollections` are the main object researchers will interact with when using *metaknowledge*, as they contain most of the user-facing methods.

```
RC = mk.RecordCollection('raw_data/imetrics/')
```

In the above example, *metaknowledge* scans the directory `raw_data` for files produced by any of the supported databases (e.g. Web of Science). It determines an appropriate parser, and then identifies individual entries (e.g. articles) from each file. Each individual entry is separated from the others, tokenized, assigned a unique id, and is added to the `RecordCollection` as a `Record`.[11] The `RecordCollection` we created above, for example, contains all these `Record`s, regardless of which database they originated from.

If researchers are working with a large set of files, *metaknowledge* provides the option to cache `RecordCollections` and `GrantCollections` by setting the argument `cached = True`.[12] This creates a cache file that *metaknowledge* can read instead of rereading and parsing the same files each time the script is re-run. The caching mechanism always checks that the raw files have not changed since the last time it created a `RecordCollection`. If they have changed (e.g. some files were removed, others added), it will create the `RecordCollection` once again.

Many commonly used algorithms in bibliometric and scientometric research require citations to be systematically compared and matched across hundreds, thousands, or even millions of records. This process is sometimes very slow. But citations are uniquely implemented in *metaknowledge*, resulting in much faster processing speed. Citations are all stored – regardless of their source – in a unified `Citation` object. The author, year and journal are extracted and used to create an identification string that is used by all functions involving citations, including network generators. This reduces the runtime from hours to minutes for very large networks.

In addition to publication and citation data, *metaknowledge* is capable of reading administrative data from some granting agencies. Currently, parsers are available to create `GrantCollections` given data from the NSF, SSHRC, NSERC, and CIHR. Of course, there is no standard for how agencies release administrative data on the grants they award. Like raw data from each of the major citation databases, each requires a unique parser. Once the data have been parsed, individual `Grants` can be added to a `GrantCollection` object; these are equivalent in almost all cases to `Record`s and `RecordCollection`s. Most of the examples in this article use `RecordCollections`, but in most cases using data from a `GrantCollection` would be the same.

Occasionally, there are problems with the raw data provided by scholarly databases like the Web of Science or from granting organizations. While *metaknowledge* can deal with some deformities, sometimes the data is too deformed to parse. This tends to happen only when database files have been manually altered in some way. When an error is found during parsing, the entry will be flagged as "bad", the line number will be recorded, and *metaknowledge* will attempt to find the next entry in the file. The rest of the document will be marked as "bad" if this fails. Every "bad" `Record` is added to the `RecordCollection` alongside good ones, but they are clearly flagged. It is up to the user to decide what to do about "bad" `Record`s, which can be examined using the `badEntries()` method, or simply dropped using `dropBadEntries()`.[13]

[10] We see this as especially important because, as Morichika and Shibayama (2016) points out, scientometric research rarely tends to focus on educating and producing scientists. Although see Sugimoto, Li, Russell, Finlay, and Ding (2011) and McFarland et al. (2013).

[11] When parsing files, each tag (e.g. author, title, publication year) has its own processing function. This enables *metaknowledge* to handle complex fields like lists of authors and cited references easily. With the exception of extracting or assigning the identification number, all of this processing and cleaning is done just-in-time. In other words, if data on authors is not needed, it will not be extracted. This means the memory usage of `Record`s increases as new fields are added, but it will be less in total than if all the fields had been processed.

[12] Caching will only work if the raw data is stored in more than one text file.

[13] Most of the publication and citation databases have poor documentation. This means that distinguishing between malformed data and data that are presented in an unusual way is difficult. Even when the data are presumed to be well formed, different programs for reading the data will make slightly different decisions about how to read it and come up with different results. An example of this is the cited reference field in WOS. As there is no official documentation for how to accurately parse citations, *metaknowledge* will give slightly different number for a year when doing RPYS (discussed in Section 5.4) then most other programs. This issue could be solved if a canonical way of parsing citations were provided, such as in a BNF grammar notation.

Table 1
Overview of some important methods for processing and analyzing record collections. Networks generators are discussed separately in Section 6.

Important methods	Implemented for	Example
write to csv	`RecordCollection`, `GrantCollection`	`RC.writeCSV()`
write to bibtex	`RecordCollection`	`RC.writeBib()`
record collection to Python dictionary	`RecordCollection`, `GrantCollection`	`RC.makeDict()`
split by year range	`RecordCollection`, `GrantCollection`	`RC.yearSplit()`
create time series data	`RecordCollection`, `GrantCollection`	`RC.timeSeries()`
glimpse a collection	`RecordCollection`, `GrantCollection`	`RC.glimpse()`
arrange by counts	`RecordCollection`, `GrantCollection`	`RC.rankedSeries()`
standard and multi RPYS	`RecordCollection`	`RC.rpys()`
filter cited references	`RecordCollection`	`RC.getCitations()`
create data for topic modeling or other NLP	`RecordCollection`, `GrantCollection`	`RC.forNLP()`
create data for burst analysis	`RecordCollection`, `GrantCollection`	`RC.forBurst()`

5. Methods for analyzing record collections

Once a `RecordCollection` object has been created, the data can be manipulated using builtin functions, called methods. Methods are used by putting a dot (`.`) after the name of the object and before the name of the method. For example, the method `peek` can be used to display a single `Record` in a `RecordCollection`.[14] We can use it, for example, on the `RecordCollection` that we called `RC`.

```
RC.peek()
```

metaknowledge has many useful methods for working with a `RecordCollection`. The methods we will discuss in this section are listed in Table 1. The simplest method is to take the content of a `RecordCollection` and write it to file as a tabular dataset where every `Record` is a row and every available tag (e.g. author, publication year, title, abstract, etc.) is a column. This is done using the `writeCSV()` function. It is also possible to convert the `RecordCollection` to a dataframe that can be used by the *Pandas* and/or *statsmodels* packages for quantitative data analysis in Python (McKinney, 2012).

```
RC.writeCSV('generated_datasets/records.csv')
pandas.DataFrame(RC.makeDict())
```

metaknowledge includes methods for producing time series datasets of article publication years, quick summaries (e.g. most frequently occurring authors and journals), estimations of (co-)author gender user birth records data, datasets for Reference Publication Year Spectroscopy (RPYS), and datasets for computational text analyses such topic modeling or burst analysis. These methods can produce dataframes for analysis in Python or to be written to disk for analysis in other software. This gives researchers easy access to advanced methods without requiring them to be implemented in *metaknowledge*.

5.1. Creating time series datasets

There are two main ways that researchers can create time series datasets. The first and simplest way is to create a dataframe with the number of articles published in each year to screen. This output can be saved as a `csv` file.

```
# [2:] removes incomplete data from 2016
growth = pandas.DataFrame(RC.timeSeries('year', outputFile = 'growth.csv'))[2:]
```

These dataframes are easily visualized with Python or other software like R. Fig. 2 is a time series plot of the growth of publications in our information science and bibliometrics dataset.

```
with sns.axes_style('white'):
  plt.plot(growth['year'], growth['count'], color = 'gray')
  sns.despine()
  plt.tight_layout()
plt.savefig('figures/growth_chart.pdf')
```

It is also possible to plot the number of articles published in specific journals over time by providing `timeSeries()` with a list of specific journals. Like other *metaknowledge* output, this method produces dataframes structured according to the "tidy" standard described by Hadley Wickham (2014), which simplifies modeling and visualization. Fig. 3 is a time series plot of the subset of articles published in *Scientometrics* and *Journal of Informetrics*.

[14] Specifically, it displays the first record from the RecordCollection, which is unordered.

Fig. 2. Growth of publications in *Journal of the Association for Information Science and Technology*/*Journal of the American Society for Information Science and Technology*, *Scientometrics*, and *Journal of Informetrics*.

```
subset = ['SCIENTOMETRICS', 'JOURNAL OF INFORMETRICS']

growth_by_journal = pandas.DataFrame(RC.timeSeries('journal',
    outputFile = 'generated_datasets/growth_journals.csv', limitTo = subset))

with sns.axes_style('white'):
    fig, ax = plt.subplots()
    growth_sm = growth_by_journal[growth_by_journal['entry'] == 'SCIENTOMETRICS'][::-1]
    growth_joi = growth_by_journal[growth_by_journal['entry'] == 'JOURNAL OF INFORMETRICS'][::-1]
    growth_sm.plot(ax = ax, y ='count', x = 'year', label = 'SCIENTOMETRICS')
    growth_joi.plot(ax = ax, y ='count', x = 'year', label = 'JOURNAL OF INFORMETRICS')
    sns.despine()
plt.savefig('figures/growth_compare.pdf')
```

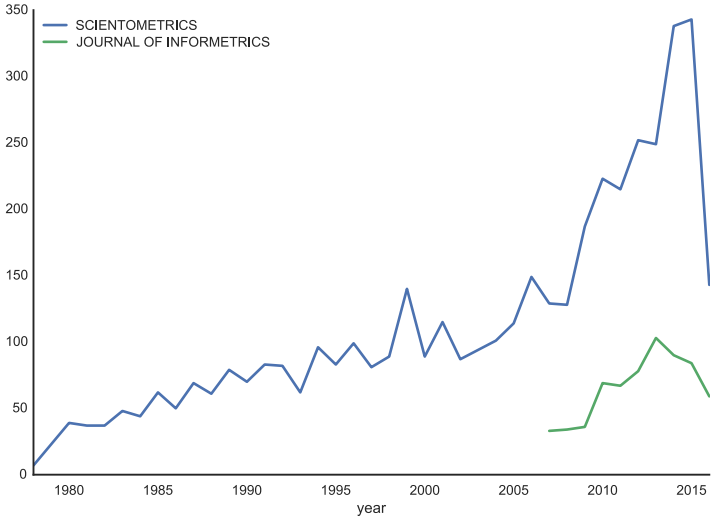


Fig. 3. Growth of publications in *Journal of Informetrics* and *Scientometrics*.

For modeling purposes, researchers may also want to divide their `RecordCollections` into a series of time slices. This can be accomplished using the `yearSplit()` function, which produces `RecordCollections` defined by a specific range of publication years. The following code will split the `RecordCollection` called `RC` into two new collections: `RC0509` and `RC1014`. The new `RecordCollection` will include the years given as arguments (i.e. 2005–2009 inclusive, 2010–2014 inclusive).

```
RC0509 = RC.yearSplit(2005, 2009)
pandas.DataFrame(RC0509.makeDict())
```

5.2. RecordCollections at a glimpse

It is possible to get a quick snapshot of a `RecordCollection` using the `glimpse()` method, which will return a string that contains a table ranking, by number of occurrences, the authors, journals, referenced citations, or any other field in the dataset.

```
overview = RC.glimpse()
print(overview)
```

5.3. Estimating (co-)author and researcher gender

One limitation of bibliometric datasets is that they do not include demographic data on co-authors or researchers unless there is a reliable way of merging bibliometric data with administrative or other additional data.[15] Most databases have incomplete address data, and many records use author first initials rather than first names. *metaknowledge* simplifies the process of adding one demographic dimension to bibliometric datasets: gender.

Good gender data is essential for many types of quantitative research on gender inequality and gender disparities in science, but for obvious reasons it is very challenging to obtain. *metaknowledge* takes an approach that has been used in recent studies of gender disparities in science (Sugimoto, 2016; Sugimoto et al., 2013; West, Jacquet, King, Correll, & Bergstrom, 2013), which is to estimate gender using country-specific data on birth records and names. *metaknowledge* downloads the Global Name Dataset from Open Gender Tracker's Github repository. This names dataset, which was initially developed for studying gender diversity in news content, uses data from the United States Social Security Administration, the United Kingdom Office of National Statistics, the Northern Ireland Statistics and Research Administration, and the Scotland General Register Office. Despite the name of the dataset, it is not yet a "global" dataset. The Open Gender Tracker team is currently working on adding new countries and therefore estimates for non-anglophone names. Similarly, we are currently working on collecting gender name data from other countries using the approach outlined in the online supplement of Sugimoto et al.'s (2013) "Global Gender Disparities in Science." In the meantime, researchers can make use of country specific gender name data without relying on the convenience functions in *metaknowledge*. A second limitation is that the birth record and name data prevents us from adopting a broader conceptualization of gender. In fact, an automated approach to gender classification is not well-suited to a more sophisticated understanding of gender, identity, and personhood. We see this as an important motivation to pair bibliometric research with survey and qualitative research on gender and science.

metaknowledge uses the open source gender classifier developed for the Open Gender Tracker, which in our case provides estimates of (co-)author or researcher gender when given first names.[16] This happens at the level of authorships (i.e. each author on each `record`), not authors. *metaknowledge* appends counts of likely male and female authors to each record in the `RecordCollection`, and uses the label "uncertain" if the name is not in their dataset, or if the classifier cannot make a reasonably good estimate. Because these estimates are available in the `RecordCollection`, they are also available as columns using the `writeCSV()` and `makeDict()` methods. Percentages can easily be computed by also using the `num-Authors` variable, which *metaknowledge* also computes automatically. A summary of the gender breakdown is available at the `RecordCollection` level using the `genderStats()` method.

```
gender_breakdown = RC.genderStats()
print(gender_breakdown)
```

```
p = pandas.DataFrame(RC.makeDict())
print(p[['num-Authors','num-Female','num-Male', 'num-Unknown']])
```

5.4. Standard and Multi Reference Publication Year Spectroscopy (RPYS)

metaknowledge includes methods for producing datasets for Standard and Multi Reference Publication Year Spectroscopy (RPYS). Standard RPYS was proposed by Marx, Bornmann, Barth, and Leydesdorff (2014) and Marx and Bornmann (2014) as a

[15] For example, see the discussion of the Slovenian researcher dataset used by Kronegger et al. (2012).
[16] The web of Science didn't include first names until 2008.

method for quantifying the impact of historical publications on research fields. It is the latest development in an established tradition of "algorithmic historiography" or "historical bibliometrics" (Garfield, Pudovkin, & Istomin, 2003; Garfield, Sher, & Torpie, 1964; Leydesdorff, Bornmann, Marx, & Milojević, 2014; Lucio-Arias & Scharnhorst, 2012), with early work primarily facilitated by the development of Garfield's (2009) HistCite software and more recently by van Eck and Waltman's (2014) CiteNetExplorer. The method is typically used to identify specific books and articles that have a durable influence over time.

5.4.1. Standard RPYS

Standard RPYS begins with mining the cited references from publications and plotting a frequency distribution of publication years sorted by date from earliest to most latest. Typically, this data is standardized by computing the extent to which the number of cited references from each publication year deviates from a five year median. These deviations from the median can be graphed as counts or percentages, revealing pronounced peaks during years when important books or articles where published. *metaknowledge* has the following functions for conducting Standard RPYS given our `RecordCollection`, `RC`.

```
stan_results = RC.rpys(1900,2000) # takes a min and max
```

This will produce a dataframe that includes data on the number of publications each year and the yearly deviation from the five year median. This dataframe can be used to easily produce graphs typically used in RPYS studies. If using Python, the standard RPYS plot, shown in Fig. 4, can be produced with the following code.

```
dev_line_color = sns.xkcd_rgb['pale red']

with sns.axes_style('white'):
    plt.plot(stan_results['year'],stan_results['abs-deviation'], color = dev_line_color)
    plt.plot([1900,2015], [0, 0], linewidth=1, color = 'black')
    sns.despine(offset=10, trim=True)
plt.savefig('figures/rpys_standard.pdf')
```

As in RPYS articles (e.g. Comins & Hussey, 2015b; Leydesdorff et al., 2014; Wray & Bornmann, 2015), researchers inspect the top publications in years where there is substantial deviation from the median. The `getCitations()` function simplifies

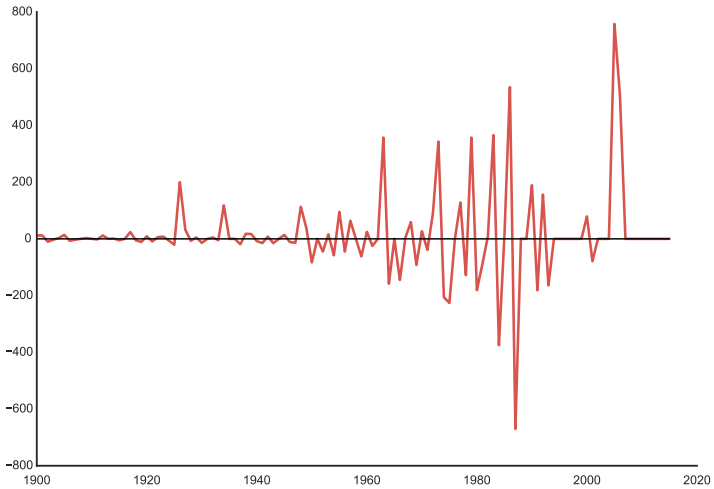


Fig. 4. Standard RPYS plot. Pronounced peaks represent years where citations to published books or articles deviate from a 5 year median.

looking up cited references from any particular year.

```
year_results = RC.getCitations('year', 1963)
```

5.4.2. Multi RPYS

Multi RPYS is an extension of the standard method (Comins & Hussey, 2015a). In short, it segments the original citing articles based on their publication years and conducts a Standard RPYS analyses for each. It is useful for differentiating between historical publications that have lasting impact versus those that are influential only within a short time frame (see Baumgartner & Leydesdorff, 2014; Comins & Leydesdorff, 2016a). It does this by rank transforming the data, which enables researchers to compare different time slices within the same field, or even across more than one field. The transformed data is then visualized as a heatmap, revealing a dynamic picture of changes in citations to historical publications over time. To that end, the code chunk below segments the `RecordCollection` by publication year and conducts an RPYS analysis for each. It stores the results in a Python dictionary, which is then used to create a table. The table is then plotted as a heatmap, shown in Fig. 5.

```
minYear = 1950
maxYear = 2015

years = range(minYear, maxYear+1)

dictionary = {'CPY': [],
              'abs-deviation': [],
              'num-cites': [],
              'rank': [],
              'RPY': []}

for i in years:
    try:
        RCyear = RC.yearSplit(i, i)
        if len(RCyear) > 0:
            rpys = RCyear.rpys(minYear=minYear, maxYear=maxYear)
            length = len(rpys['year'])
            rpys['CPY'] = [i]*length

            dictionary['CPY'] += rpys['CPY']
            dictionary['abs-deviation'] += rpys['abs-deviation']
            dictionary['num-cites'] += rpys['count']
            dictionary['rank'] += rpys['rank']
            dictionary['RPY'] += rpys['year']
    except:
        pass
```

```
multi_rpys = pandas.DataFrame.from_dict(dictionary)

hm_table = multi_rpys.pivot('CPY', 'RPY', 'rank')

with sns.axes_style('white'):
    sns.heatmap(hm_table, square = False, cmap='YlGnBu', cbar_kws={'orientation': 'horizontal'})
    plt.xlabel('Reference Publication Year', size = 8)
    plt.ylabel('Citing Publication Year', size = 8)
    sns.despine()
    plt.tight_layout()
plt.savefig('figures/rpys_multi.pdf')
```

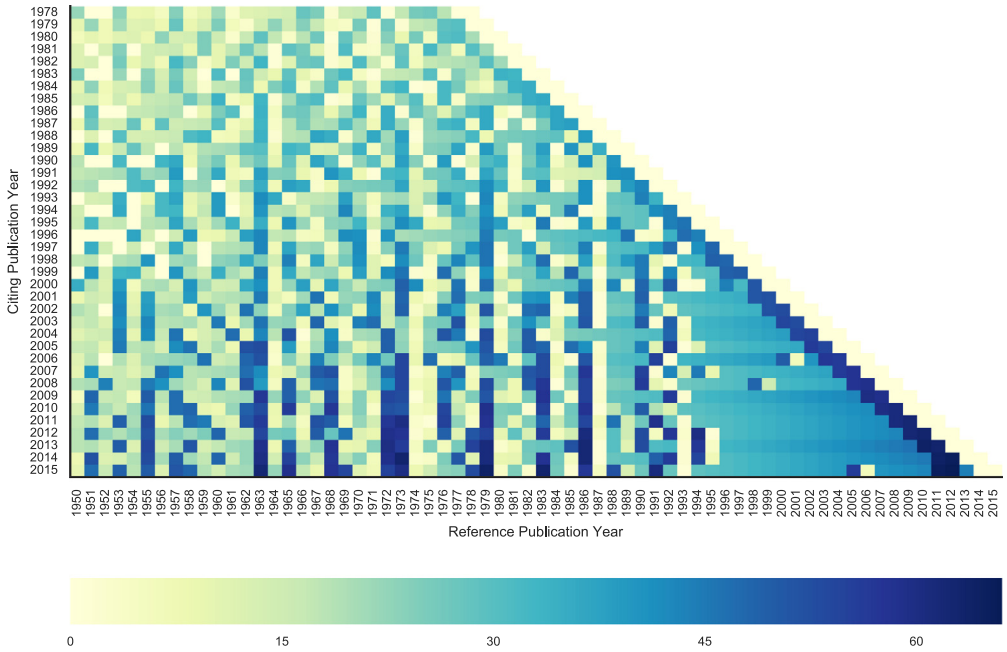


Fig. 5. Heatmap showing the results of a Multi RPYS analysis. Darker bands across both time periods indicate lasting influence. For example, *Big Science, Little Science* was published in 1963.

RPYS is subject to the same set of limitations as other historical bibliometric datasets and should therefore always be interpreted with caution. As discussed by Marx (2011), older bibliometric data have numerous problems, including poor coverage of journals and fields, database errors, translation problems, misspelling and other errors in handling the names of authors and journals, and the greater use of informal citations in older research.

5.5. Computational text analysis

metaknowledge can also extract datasets for computational text analysis, including topic modeling (Adams & Light, 2014; Blei, 2012; McFarland et al., 2013) and burst analysis (Kleinberg, 2003), both of which are becoming more common in scientometrics, information science, and sociology (Adams & Light, 2014; Bail, 2014a,b, 2015; DiMaggio, 2015; DiMaggio, Nag, & Blei, 2013; Griffiths & Steyvers, 2004; Light & Adams, 2016; Light et al., 2014; Mohr & Bogdanov, 2013; Nichols, 2014; Romo-Fernández, Guerrero-Bote, & Moya-Anegón, 2013; Song & Kim, 2013; Yau, Porter, Newman, & Suominen, 2014). The methods `forNLP()` and `forBurst()` produce datasets which can be analyzed using text analysis packages, including implementations of Latent Dirichlet Allocation (LDA) models in Python, R, and Java.[17] The supplementary Jupyter Notebooks include examples with more detail provided than we can go into below.

`forNLP` produces a `csv` file consisting of the unique id (e.g. Web of Science number) for each `Record`, the publication year, title, abstract, and keywords. Before writing to disk, *metaknowledge* will process all titles, abstracts, and keywords using methods that are widely-used used in computational text analysis. To that end, there are numerous optional arguments that, when enabled, will change text to lower case, stem words to their common roots, and remove numbers, white space, and delimiting characters. *metaknowledge* also uses *NLTK: The Natural Language Toolkit* package (Bird, Klein, & Loper, 2009) to specify lists of stopwords to be deleted. A variety of languages are supported. Finally, researchers can easily drop specified keywords if, for example, they are too general to be useful. This is done by identifying terms to drop in a Python list (called "stop_words" in the example below), and then telling `forNLP()` to drop the keywords provided in that list. The resulting dataset can be easily analyzed using text analysis packages, including topic modeling. Fig. 6 is a screenshot of a topic model visualization using the *pyLDAvis* package running in the supplementary Jupyter Notebook on text analysis.[18] The topic model itself was computed using the *gensim* package.

[17] There are many online resources in addition to the published literature. We suggest beginners consult the tutorial on topic modeling published by *The Programming Historian* (Graham, Weingart, & Milligan, 2012).

[18] *pyLDAvis* is Ben Mabey's port of Carson Sievert and Kenny Shirley's R library *LDAvis*.

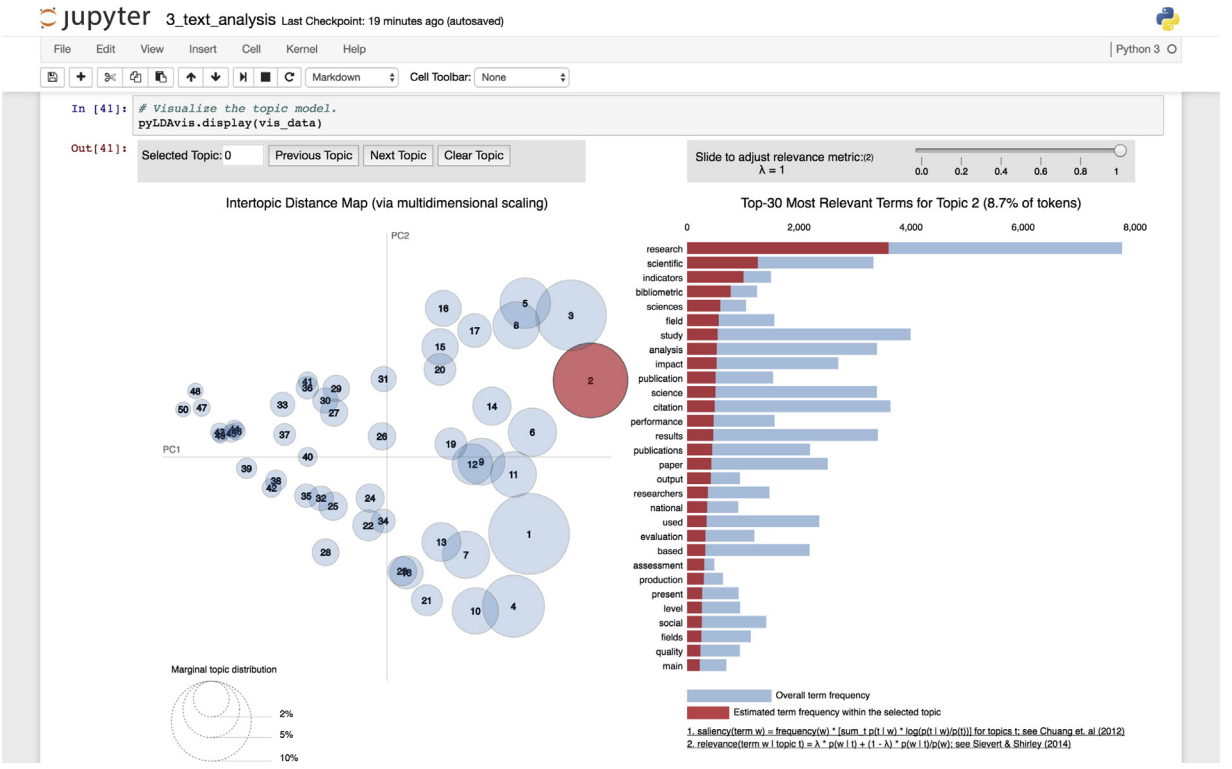


Fig. 6. Screenshot of a topic model visualization running in the supplementary Jupyter Notebook on text analysis. Hovering over topics dynamically updates the horizontal bar graph on the right.

```
stop_words = ['list', 'of', 'words', 'to', 'remove', 'from', 'abstracts']

RC.forNLP('generated_datasets/topic_model/', dropList = stop_words,
    lower = True, removeNumbers = True, removeNonWords = True,
    removeWhitespace=True, removeCopyright=False, stemmer = None)
```

Burst analysis (e.g. with Binder, 2015) requires slightly different data, structured as a two column "stream" of years and single words. This can be produced using the `forBurst` method. Again, *metaknowledge* will process the text by default, although all cleaning can be turned off by using the optional arguments.

```
RC.forBurst('keywords', outputFile = 'generated_datasets/topic_model/bursts.csv',
    dropList = 'stop_words', lower = True, removeNumbers = True, removeNonWords = True,
    removeWhitespace = True, stemmer = None)
```

6. Methods for creating and processing networks

Some of the most useful methods in *metaknowledge* generate and modify datasets on intellectual networks (White, 2011; Yan & Ding, 2012). The *metaknowledge* network generators make extensive use of the Python package *networkx* (Hagberg, Schult, & Swart, 2008).[19] *metaknowledge* currently has 10 network generators, summarized in Table 2. We will describe the specifics of each of the generators below, though they all follow the same general logic. It is almost always necessary to process these networks once they have been generated. While many processing functions are available in *networkx*, some critical to scientometric research are not. Most importantly, scientometric researchers may need to drop edges based on weight. In co-citation networks, for example, edges with a weight of 1 are best considered noise. While not necessarily recommended, researchers may also wish to drop nodes from the network if they fall below some specified degree threshold. Finally, researchers may wish to drop nodes based on some other quantitative attribute, such as an occurrence count. Edges and nodes can be dropped from an already created network according to these three attributes – edge weight thresholds, degree thresholds, count threshold – using the functions `dropEdges()`, `dropNodesByDegree()`, and `dropNodesByCount()`.

[19] We chose *networkx* over the Python version of *igraph* because it is easier to install and the difference in speed is minor for the work *metaknowledge* is doing.

Table 2
Overview of *metaknowledge* network generators.

Generator	Implemented for	Stemming	Usage
co-author	`RecordCollection`	No	`RC.networkCoAuthor()`
co-investigator	`GrantCollection`	No	`GC.networkCoInvestigator()`
institutions	`GrantCollection`	No	`GC.networkCoInvestigatorInstitution()`
citation	`RecordCollection`	No	`RC.networkCitation()`
co-citation	`RecordCollection`	No	`RC.networkCoCitation()`
bib coupling	`RecordCollection`	No	`RC.networkBibCoupling()`
one mode	`RecordCollection`, `GrantCollection`	Yes	`RC.networkOneMode()`
two mode	`RecordCollection`, `GrantCollection`	Yes	`RC.networkTwoMode()`
multi mode	`RecordCollection`, `GrantCollection`	Yes	`RC.networkMultiMode()`
multi level	`RecordCollection`, `GrantCollection`	Yes	`RC.networkMultiLevel()`

The first argument that the three drop functions take is the name of the graph they will be modifying. They modify the original graph in place rather than producing a new one, making computation more efficient when dealing with large networks. Then there are a series of optional arguments to control the conditions under which an edge or node will be dropped, including lower and upper thresholds, the name of attributes for counting, and whether or not to drop self-loops. For example, it is possible to remove all edges that are self-loops or that have a weight less than 2, and then to remove isolates. We demonstrate these methods in the sections on network generators below.

6.1. Co-authorship

Co-authorship networks are produced from `RecordCollections` with the `networkCoAuthor()` method. Currently, authors are matched based on the full author string, including middle initials if provided. Obviously this introduces the possibility that a single author's work could be incorrectly attributed to multiple "authors." On the other hand, matching authors based on only a first initial and last name increases the possibility that the work of multiple authors will be combined into the work of one "author." We have chosen the stricter requirement (full match) as the default, however researchers can switch to using the shortened names provided by some sources (e.g. AU instead of AF in the Web of Science). Other disambiguation methods may be implemented in the future, and researchers are free to code their own.[20]

We can easily create a co-authorship network from any `RecordCollection`. Below, we create a network from the full collection of 8,140 articles published between 1978 and 2015 in *Journal of the Association for Information Science and Technology*/*Journal of the American Society for Information Science and Technology*, *Scientometrics*, and *Journal of Informetrics*. The function `graphStats()` provides a summary of some useful network statistics, including the number of nodes and edges, network density, and transitivity.

```
coauth_net = RC.networkCoAuthor()
print(mk.graphStats(coauth_net))
```

Co-authorships are counted and assigned as edge weights,[21] making it simple to remove edges that fall below some specified edge threshold. The code block below illustrates how to do this using the `dropEdges` method. Further modifications can be made using *networkx* methods. In the example below, we restrict the network to the giant component, compute eigenvector centrality scores, and plot the network using Eigenvector centrality for node size.[22] The final result is shown in Fig. 7.

```
# drop edges with weight < 2, restrict to giant component
mk.dropEdges(coauth_net, minWeight = 2, dropSelfLoops = True)
giant = max(nx.connected_component_subgraphs(coauth_net), key=len)

# compute eigenvector centrality and use for node size
eig = nx.eigenvector_centrality(giant)
size = [2000 * eig[node] for node in giant]

# draw the graph
nx.draw_spring(giant,
    node_size = size,
    with_labels = True,
```

[20] *metaknowledge* handles disambiguation the same way for other strings, such as institution names.
[21] If it is necessary to produce a binary network, we can add the argument `weighted = False` and `count = False` to the function.
[22] While we have opted to work within Python for this example, researchers may prefer to use other software for visualizing and modeling networks.

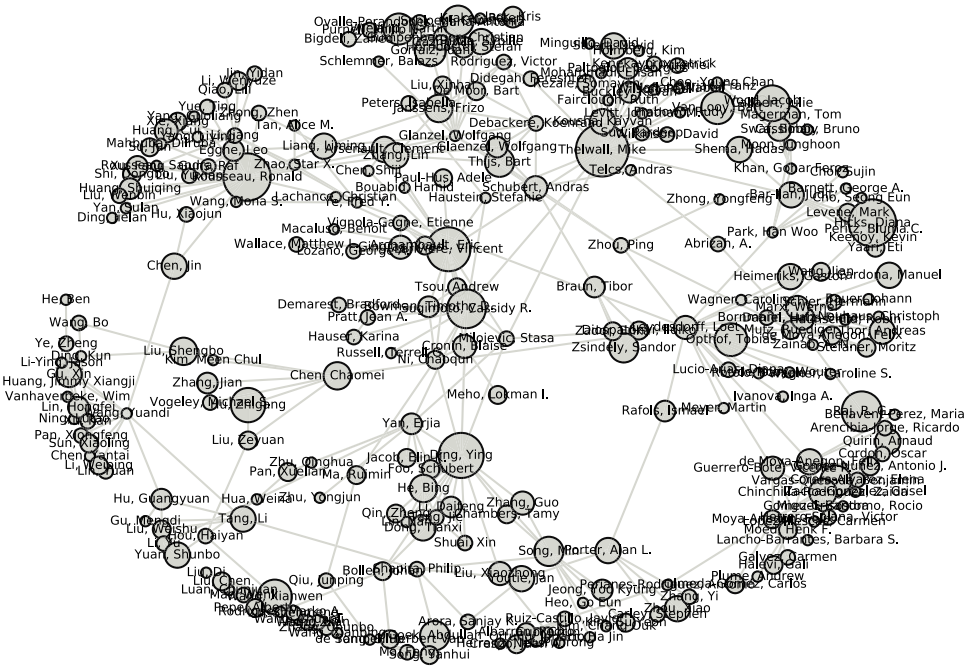


Fig. 7. Network of repeat coauthors on articles published between 1978 and 2015 in *Journal of the Association for Information Science and Technology*/*Journal of the American Society for Information Science and Technology*, *Scientometrics*, and *Journal of Informetrics*. Giant component only.

```
    font_size = 5,
    node_color = '#FFFFFF',
    edge_color = '#D4D5CE',
    alpha = .95)
plt.savefig('./figures/coauthors.pdf')
```

Researchers can detect communities using the Louvain method (Blondel, Guillaume, Lambiotte, & Lefebvre, 2008) implemented in the *community* package. The code block below shows how to do this by first computing the best partitions. The resulting visualization – shown in Fig. 8 – differs slightly from the first because the layout algorithms have different random seeds.

```
import community

partition = community.best_partition(giant)
modularity = community.modularity(partition, giant)
print('Modularity:', modularity)

colors = [partition[n] for n in giant.nodes()]
nx.draw_spring(giant,
    node_color=colors,
    cmap=plt.cm.Accent,
    node_size = degSize)
plt.savefig('./figures/coauthors_community.pdf', bbox_inches='tight')
```

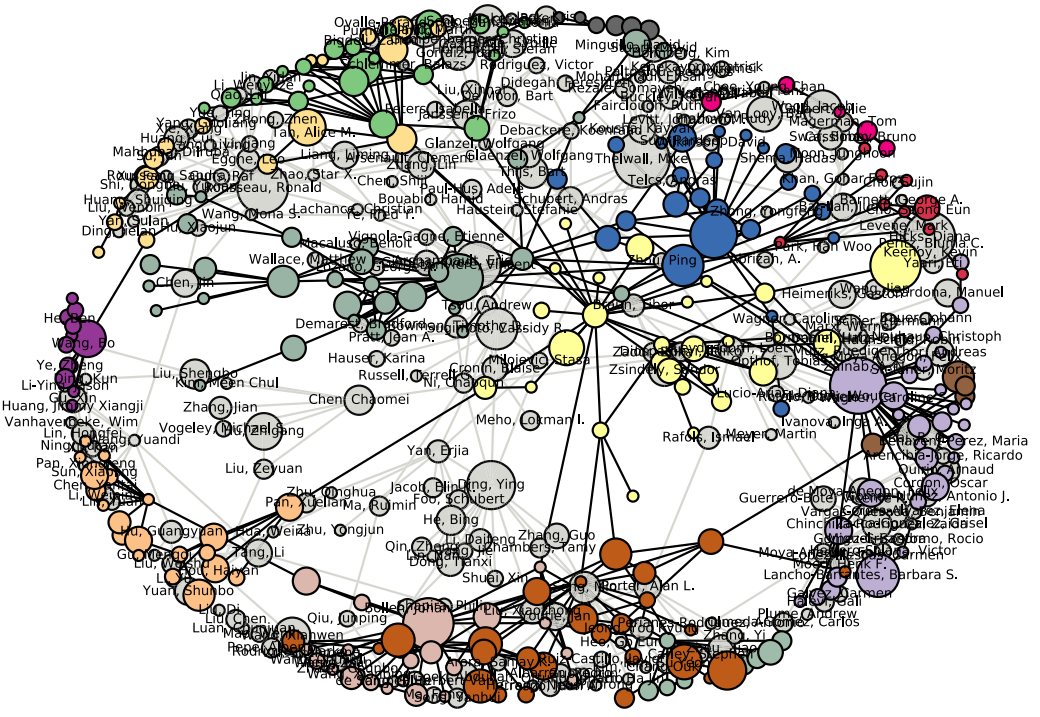


Fig. 8. Louvain community detection (modularity:.841) on giant component of repeat co-author network. As before, nodes are sized by eigenvector centrality.

The supplementary Jupyter Notebook includes some further analysis of this network. Researchers interested in doing more with *networkx* (Hagberg et al., 2008) should also consult the documentation and online tutorials.

6.2. Citation, co-citation, and bibliographic coupling networks

The other three specialized network generators for `RecordCollections` are for creating citation, co-citation (Small, 1973), and bibliographic coupling (Kessler, 1963) networks. Each of these by default creates networks using the `citations` extracted from the `Records` of a `RecordCollection`. The citations are tracked with unique IDs composed of their author, year and journal. By default all three methods will return networks at the document level. It is simple to return networks at the journal or author levels by using the optional `nodeType` argument. Generating these networks is straightforward. The example below (see Fig. 9) uses the `RecordCollection` containing articles published between 2010 and 2014.

```
document_cite = RC1014.networkCitation()
document_cocite = RC1014.networkCoCitation()
document_cocite_in_RC_only = RC1014.networkCoCitation(coreOnly = True)
journal_cocite = RC1014.networkCoCitation(nodeType = 'journal')
```

metaknowledge scans the cited references fields from `Records` when creating a co-citation network from a `RecordCollection`. The default behavior is to construct a co-citation network from all items in the reference lists regardless of where they were published. Unfortunately, the titles of journal articles are not included in the cited references meta-data. To make node information more useful, *metaknowledge* will attach article titles to the nodes if the article is included as an item in the `RecordCollection` itself (otherwise, *metaknowledge* has no way of knowing). If the researcher prefers not to include any article titles, they can add the argument `detailedCore = False`. This is only likely to be the case for extremely large datasets, as including titles has very little effect on performance in most cases.

It is also possible to construct a co-citation network where the nodes in the network are limited to `Records` from the `RecordCollection`. This is done by passing the optional argument `coreOnly = True`. By way of example, the following code creates an article co-citation network (article level is the default setting) from the `RC1014` record collection. The optional argument `coreOnly = True` means that only articles published in the seed journals used to create this `RecordCollection` will be nodes in the network. The edge threshold is set to 3 using the `minWeight` argument in `dropEdges`.

```
journal_cocite = RC1014.networkCoCitation(coreOnly = True)
mk.dropEdges(journal_cocite, minWeight = 3)

# visualize the giant component only
giant2 = max(nx.connected_component_subgraphs(journal_cocite), key=len)

partition = community.best_partition(giant2)
modularity = community.modularity(partition, giant2)
print(modularity)

colors = [partition[n] for n in giant2.nodes()]

nx.draw_spring(giant2,
    node_color=colors,
    with_labels = False,
    cmap=plt.cm.Accent,
    node_size = 100)
plt.savefig('./figures/cocite_uniform_community.pdf', bbox_inches='tight')
```

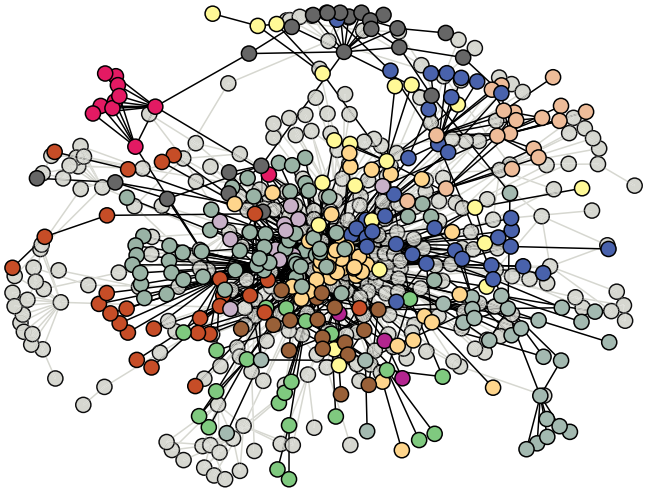


Fig. 9. Co-citation network of information science and scientometrics articles, 2010–2014. Only articles published in *Journal of the Association for Information Science and Technology*/*Journal of the American Society for Information Science and Technology*, *Scientometrics*, and *Journal of Informetrics* are included in the final network. Only the giant component is shown. Louvain community detection, modularity: 0.425.

6.3. Co-investigator and co-investigator institution networks

Co-investigator networks can be extracted from a `GrantCollection` using the `networkCoInvestigator()` method. For data from known granting agencies, *metaknowledge* will use the default grant parser to automatically identify investigator names and affiliations for constructing the networks. For unknown agencies, *metaknowledge* will use a `fallbackParser` to parse the grants and the researcher will have to supply the tags that should be used to construct the network using the `targetTags` argument.

```
nserc_grants = mk.GrantCollection('raw_data/grants/nserc/')
ci_nets = nserc_grants.networkCoInvestigator()
```

In some cases, researchers may want to construct networks of institutions rather than networks of researchers. This can be done with the `networkCoInvestigatorInstitution()` method. The behavior is the same as `networkCoInvestigator()`, except that the tag used stores information about the institution name rather than the researcher.

```
inst = nserc_grants.networkCoInvestigatorInstitution()
print(mk.graphStats(inst))
```

In addition, networks can be generated from grant data using the one mode, two mode, multi-mode, and multi-level functions outlined in Table 2.

6.4. One-mode networks (e.g. keyword)

In addition to network generators for co-authors, citation, co-citation, bibliographic coupling, and co-investigators, there are also four general purpose network generators. The first is `networkOneMode()`, which takes in one tag string and produces a network where the nodes are elements of the tags and edges as assigned based on co-occurrence. It is possible to use this generator to create a co-occurrence network for any term (e.g. co-occurrence of MeSH terms from PubMed records). This openness provides greater flexibility to the researcher, who simply has to generate a network that makes sense given their research questions.

```
keywords = RC1014.networkOneMode('keywords')
```

6.5. Two-mode networks

The second general purpose network generator constructs a two mode network, for example of authors and keywords, using the `networkTwoMode()` method. As with classical two-mode networks, edges are assigned across node types (e.g. author-keyword) but not within types (see the function `networkMultiLevel()` below). `networkTwoMode()` takes in two tags and produces a network with entries from one tag linked to the other. The graph produced is undirected by default, though it can be made directed by using the `directed = True` argument. If this is the case, the first tag will be the `source` and the second will be the `target`.

```
two_mode = RC1014.networkTwoMode('keywords', 'authorsFull')
mk.graphStats(two_mode)
```

6.6. Multi-mode networks

The `networkMultiMode()` generator is similar to `networkTwoMode()` in that nodes of the same type do not link to each other. However, `networkMultiMode()` can be given an arbitrary number of tags. Currently, studies of networks with 3 or more modes are very rare.

6.7. Multi-level networks

Finally, *metaknowledge* has a method for producing multilevel networks. In this case, edges are assigned across and within types (author-author, keyword-keyword, author-keyword). This is aligned with the recent development of theory and methods for multi-level network analysis (Bellotti, 2012; Bellotti, Guadalupi, & Conaldi, 2016; Lazega, Jourda, Mounier, & Stofer, 2008; Wang, Robins, Pattison, & Lazega, 2013). Researchers can produce network of this type with the method `networkMultiLevel()`.

```
multilevel = RC1014.networkMultiLevel('keywords', 'authorsFull')
```

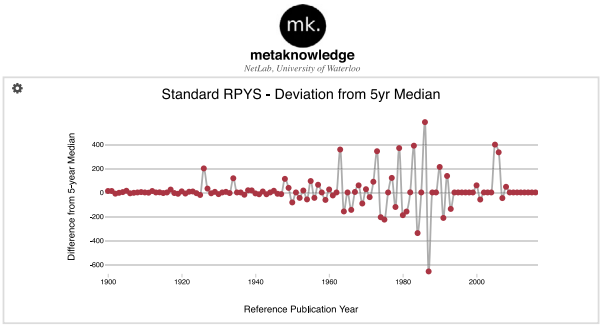
6.8. Writing networks to disk

Researchers have a range of options for writing network datasets to disk. First, using functions available in *networkx*, researchers can create `.graphml`, `.net` (Pajek), or `.gml` files.[23] Although these produce well-formatted and easy to use files, we found them to be slow for even moderately sized networks (e.g. 1000 nodes). To solve this problem, metaknowledge has a `writeGraph()` method that can produce weighted and time-stamped edge lists and node attribute files using the data structures required for two-mode, multi-level, and longitudinal / dynamic network analysis. `writeGraph()` will automatically write both to disk as `.csv` files. The resultant files have been tuned for reading by R (e.g. by *statnet* (Handcock, Hunter, Butts, Goodreau, & Morris, 2008)), although no other software has been found to have problems with them. The speed of `writeGraph()` is limited by the write speed of the host's disk and is recommend for large graphs.

```
mk.writeGraph(document_cocite, 'generated_datasets/journal_document_cocitework')
```

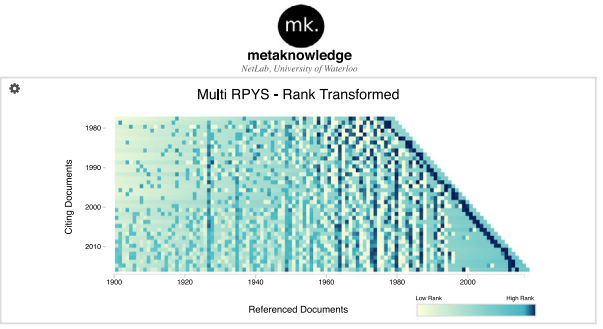
7. Interactive data visualizations

metaknowledge has an accompanying Javascript library called *mkD3* that takes datasets generated by *metaknowledge* and produces interactive *D3.js* graphs (Bostock, Ogievetsky, & Heer, 2011) with useful and informative default settings. Currently, *mkD3* can display interactive networks, standard RPYS, and multi RPYS graphs. Examples of how to create these graphs are included in the supplementary Jupyter Notebooks.



Rank	Author	Source Title	Year Published	Times Cited
1	Schubert A	SCIENTOMETRICS	1986	166
2	Callon Michel	MAPPING DYNAMICS SCI	1986	99
3	Vinkler P	SCIENTOMETRICS	1986	52
4	Price D	LITTLE SCI BIG SCI	1986	43
5	Healey P	RES POLICY	1986	35
6	Dervin B	ANNU REV INFORM SCI	1986	35
7	Lawani Sm	SCIENTOMETRICS	1986	34
8	Leydesdorff L	SCIENTOMETRICS	1986	33
9	Brooks Ta	J AM SOC INFORM SCI	1986	32
10	Schubert A	CZECH J PHYS	1986	31
11	Pao Ml	J AM SOC INFORM SCI	1986	29
12	Jaffe Ab	AM ECON REV	1986	28
13	Swanson Dr	PERSPECT BIOL MED	1986	25
14	Macroberts Mh	SOC STUD SCI	1986	24

(A)



RPY	Author	Source Title	Times Cited	CPY
2010	Moed Hf	J INFORMETR	27	2011
2010	Leydesdorff L	J AM SOC INF SCI TEC	26	2011
2010	Bornmann L	J INFORMETR	21	2011
2010	Opthof T	J INFORMETR	18	2011
2010	Egghe L	ANNU REV INFORM SCI	17	2011
2010	Van Raan Afj	J INFORMETR	15	2011
2010	Zitt M	J INFORMETR	12	2011
2010	Rafols I	SCIENTOMETRICS	11	2011
2010	Prathap G	SCIENTOMETRICS	11	2011
2010	Leydesdorff L	J INFORMETR	10	2011
2010	Franceschet M	J INFORMETR	10	2011
2010	Gonzalez-Pereira B	J INFORMETR	10	2011
2010	Alonso S	SCIENTOMETRICS	9	2011
2010	Franceschini F	EUR J OPER RES	9	2011

(B)

Fig. 10. (A) is a screenshot of an interactive standard RPYS plot in a browser window. Hovering over year produces a popover with information about the top publication that year and will automatically update the content of a table listing the top 15 publications in that year. Both tools can be easily turned on and off. (B) is an interactive heatmap of a Multi RPYS.

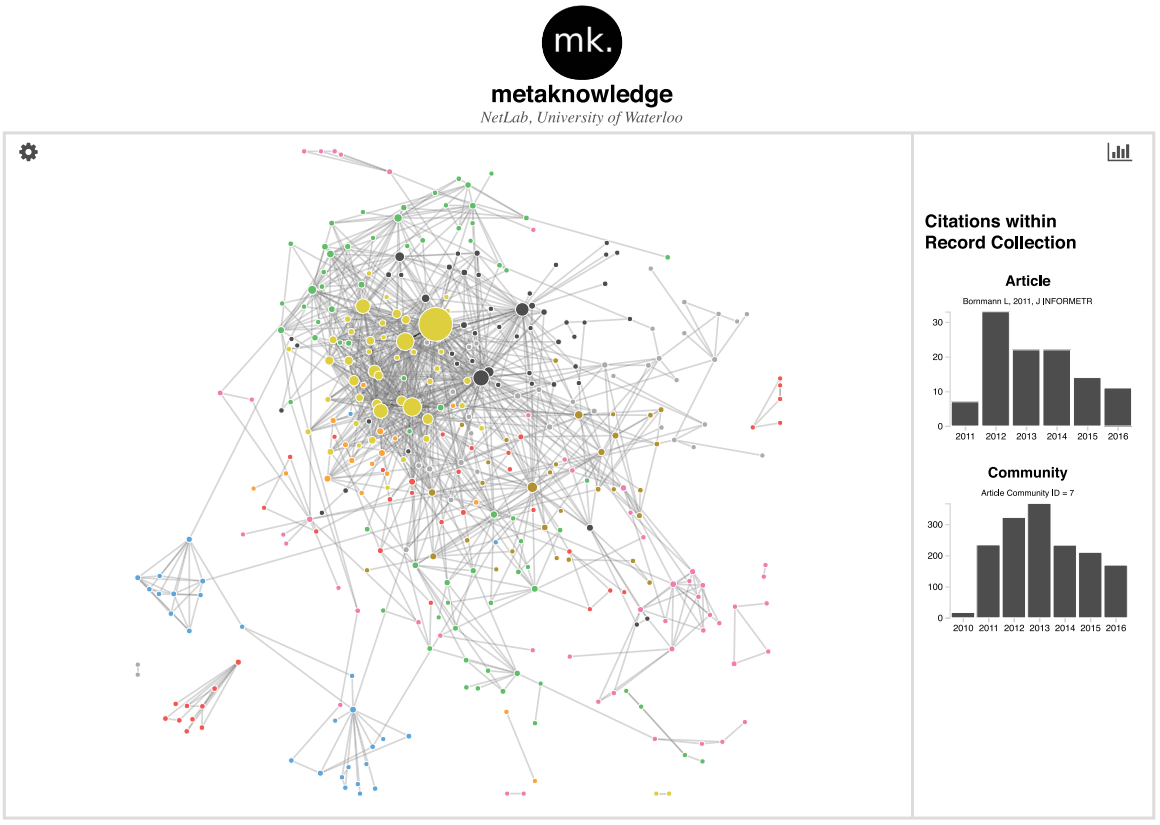
mkD3 graphs open in a browser window. For example, panel (A) of Fig. 10 is a screenshot of the results of a Standard RPYS. By default, hovering over a year will produce a small popover that displays the extent to which that year deviated from the five year median and information about the most frequently cited publications from that year. There is also a table underneath the main graph that updates dynamically based on the researcher's selections. The functionality is the same for the Multi RPYS, except that the default graph is a heatmap (as discussed in Section 5.4.2). A screenshot of the interactive heatmap is shown in the second panel (B) of Fig. 10. Similarly, hovering over nodes in a network graph – such as the one shown in Fig. 11 – reveals a small window with information about the node, and dynamically updates the information table below the graphs.

mkD3 graphs include several additional tools intended to facilitate data exploration. First, right clicking on any node in a network visualization will open up a new browser tab with the results of a PubMed or Google Scholar query for the specific article, author, or journal. Second, the bar graphs on the side of the main graph dynamically display citation counts each year for whatever node the mouse is hovering over. Importantly, these are citations within the `RecordCollection` only, which is what enables *metaknowledge* to compute counts for individual years. Third, the second bar graph aggregates citations within the `RecordCollection` for all nodes that are part of the same community.[24] Finally, *mkD3* can integrate co-citation network and RPYS analyses (see McLevey & Anderson, Working Paper). If there is data on community membership, researchers have the option of displaying RPYS graphs as small multiples (Few, 2012; Robbins, 2012; Tufte, 1990) below the main network graph, enabling researchers to identify historically important publications from each intellectual community in the network.[25]

8. Advanced usage

At this point, many researchers already have what they need: a reproducible way of producing well-structured datasets with very few lines of simple code, and some simple methods for inspecting and cleaning those datasets. We have also provided examples of using other packages (e.g. *networkx*) to analyze datasets produced by *metaknowledge*. There are, however, more advanced data processing functions available in *metaknowledge* that researchers may want or need to access. In this section we describe two such uses: (1) filtering `RecordCollections` and (2) processing very large network datasets in computationally efficient ways. These are only two of many possible advanced usages, and our discussion of them is compressed due to space considerations. The complete documentation has more information about both examples below.

[23] VOSViewer imports `.net` and `.gml` files.
[24] Of course, this requires that the dataset include a community membership variable.
[25] An important limitation is that the co-citation network must be created using the `coreOnly = True` argument, which restricts the nodes in the network to articles that were published in the original record collection. Otherwise, *metaknowledge* will not have access to the necessary cited reference data.



Bornmann L, 2011, J INFORMETR

Node A	Node B	Edge Weight
Bornmann L, 2011, J INFORMETR	Leydesdorff L, 2011, J AM SOC INF SCI TEC	77
Bornmann L, 2011, J INFORMETR	Waltman L, 2012, J AM SOC INF SCI TEC	27
Bornmann L, 2011, J AM SOC INF SCI TEC	Bornmann L, 2011, J INFORMETR	21
Waltman L, 2011, J INFORMETR	Bornmann L, 2011, J INFORMETR	20
Bornmann L, 2011, J INFORMETR	Bornmann L, 2013, J INFORMETR	19
Bornmann L, 2012, J INFORMETR	Bornmann L, 2011, J INFORMETR	19
Bornmann L, 2010, J INFORMETR	Bornmann L, 2011, J INFORMETR	18
Bornmann L, 2011, J INFORMETR	Rousseau R, 2012, J AM SOC INF SCI TEC	18
Opthof T, 2010, J INFORMETR	Bornmann L, 2011, J INFORMETR	18
Bornmann L, 2011, J INFORMETR	Moed Hf, 2010, J INFORMETR	17
Bornmann L, 2011, J INFORMETR	Leydesdorff L, 2010, J AM SOC INF SCI TEC	15
Gingras Y, 2011, J INFORMETR	Bornmann L, 2011, J INFORMETR	14
Bornmann L, 2011, J INFORMETR	Bornmann L, 2011, J INFORMETR	13
Leydesdorff L, 2012, J AM SOC INF SCI TEC	Bornmann L, 2011, J INFORMETR	12
Vinkler P, 2010, SCIENTOMETRICS	Bornmann L, 2011, J INFORMETR	12
Vinkler P, 2011, J AM SOC INF SCI TEC	Bornmann L, 2011, J INFORMETR	12

Fig. 11. Screenshot of an article-level co-citation network in *mkD3*. Hovering over a node (1) dynamically updates the table below the graph, which displays the edge weights for each adjacent node, and (2) updates the dynamic bar graphs in the right pane of the graph window. The first of the two graphs uses the `citation` object to search for the node in the cited reference fields of the raw data, producing citation counts within the `RecordCollection` over time. The second bar graph does the same, but aggregates all citations for nodes that have been identified as part of the same community.

This section assumes researchers are familiar with Python. Users without Python knowledge, or who are not interested in this more advanced functionality, can skip the section without missing key information about using *metaknowledge*.

8.1. Filtering record collections

Sometimes it is useful to filter `RecordCollection`s, for example by extracting `Record`s that were published in a specific journal or that include a certain keyword. `RecordCollection`s are mutable sets, which means they have all the features of Python builtin `set`s.[26] This enables *metaknowledge* users to iterate over all the `Record`s in the `RecordCollection`. The `Record`s act like Python's builtin `dict`s except for the fact that they cannot be modified. A `Record`'s keys are the names of the tags in the source, and the values are the processed content of the those tags.[27] So, for example, if we wanted to extract `Record`s containing the WOS keyword (`'ID'`) 'MODEL' from the `RecordCollection` we created above, we could use a for loop.

```
RC_filtered = mk.RecordCollection() # new blank collection for the Records
for R in RC:
    if 'MODEL' in R.get('ID', []): # get catches KeyError and replaces it with []
        RC_filtered.add(R)
# RC_filtered now has all the Records with keyword Model
```

Using the sources name for tags does not work when multiple sources are used. For example, author full names have the tag `'AF'` in the Web of Science, but are called `'FAU'` in PubMed's Medline format. For this reason, some tags have an alternate name that is shared across all sources. For example, `'AF'` from Web of Science becomes `'authorsFull'`. The full list of common names is in the documentation.

8.2. Processing large networks

Working with networks containing millions of nodes is very memory intensive. The default settings for most functions in *metaknowledge* provide large amounts of data, which can slow down processing time for very large networks. In many cases, researchers do not need all of the data that *metaknowledge* offers by default (e.g. volume and issue numbers, page counts, publisher addresses). Thus, if runtimes are too slow, using the optional arguments to restrict the data *metaknowledge* returns can speed up computing significantly.

Researchers can reduce *metaknowledge*'s memory usage when dealing with `Citation` objects by using the global flag `FAST_CITES`, which causes `Citation`s to be converted into their id strings upon creation. This reduces their memory usage by ~70%, making most operations much faster. Generating co-citation networks for collections of 300K records, for example, will take minutes instead of hours. The tradeoff is that all methods not provided by strings will raise an exception.

Finally, the `mergeGraphs()` method combines two networks without losing any information.[28] This means that instead of creating a single large graph at one time, researchers can break their datasets into separate smaller graphs, process those graphs, and then combine them. This workflow is less memory intensive.

9. Conclusion

This article introduced *metaknowledge*, a Python package for computational research in information science, network analysis, and science of science. *metaknowledge* is primarily designed to simplify the ~80% effort (Light et al., 2014) that goes into creating and preparing datasets for modeling and other analysis. Although *metaknowledge* appears to have a higher barrier to entry than most other existing software, it is designed so that researchers have to know very little Python to do most of what they would want to do. At the same time, even researchers with very little Python knowledge will find that they are limited only by their data, not their software.

Currently, *metaknowledge* accepts raw data from the Web of Science, PubMed, Scopus, ProQuest Dissertations and Theses, and administrative data from some granting agencies. The most important functions parse these raw data files to construct `RecordCollections` and `GrantCollections`, which can be processed and used to create tidy datasets for time series plots and longitudinal analysis, Reference Publication Year Spectroscopy for historical bibliometrics, text datasets for computational analysis using methods like topic modeling and burst analysis, and network datasets for a wide variety of networks, including multi-mode, multi-level, and longitudinal. *metaknowledge* is designed to output this data in ways that integrate seamlessly with other data analysis software. Our primary emphasis is on integration with other Python packages and R, where innovative new methods are being implemented at a pace that exceeds that of specialized scientometric research software. *metaknowledge* will also output datasets that can be easily analyzed with popular GUI software like Pajek, Gephi, and VOSViewer, and a new Javascript library (*mkD3*) for producing interactive *D3.js* graphs. *metaknowledge* is optimized to

[26] Python's official documentation gives the full list of features.
[27] If the tag is missing or blank it is considered missing so a `KeyError` will be raised if the `[]` syntax is used to access that tag within the record.
[28] The algorithm is described in the *metaknowledge* documentation.

scale well to very large datasets. It has custom data structures and can cache datasets, making computing much faster and more efficient on both small and large datasets.

Finally, and most importantly, *metaknowledge* is open source and is fully compatible with open and reproducible computing workflows. As an open source package, researchers are free to make changes to source code or extend core functionality. For example, researchers can add parsers to read administrative data from their own national granting agencies, and because *metaknowledge* is scriptable, it is better suited to open science practices and reproducible analyses than other scientometric software. We hope that it can facilitate the broader movement towards open science practices, which information science, science of science, and networks researchers are well-positioned to contribute to.

References

Adams, J., & Light, R. (2014). Mapping interdisciplinary fields: Efficiencies, gaps and redundancies in HIV/AIDS research. *PLoS ONE*, *9*, 1–13.
Bail, C. (2014a). The cultural environment: Measuring culture with big data. *Theory and Society*, *43*, 465–482.
Bail, C. (2014b). *Terrified: How anti-Muslim fringe organizations became mainstream*. Princeton: Princeton University Press.
Bail, C. (2015). Lost in a random forest: Using Big Data to study rare events. *Big Data & Society*, *2*, 1–3.
Baumgartner, S., & Leydesdorff, L. (2014). Group-based trajectory modeling (GBTM) of citations in scholarly literature: Dynamic qualities of "transient" and "sticky knowledge claims". *Journal of the Association for Information Science and Technology*, *65*, 797–811.
Bellotti, E. (2012). Getting funded. Multi-level network of physicists in Italy. *Social Networks*, *34*, 215–229.
Bellotti, E., Guadalupi, L., & Conaldi, G. (2016). *Comparing fields of sciences: Multilevel networks of research collaborations in Italian academia*. pp. 213–244. Springer International Publishing.
Binder, J. (2015). *Package 'bursts'*.
Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. O'Reilly.
Blei, D. (2012). Probabilistic topic models. *Communications of the ACM*, *55*, 77–84.
Blondel, V., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, *2008*, P10008.
Börner, K. (2010). *Atlas of science: Visualizing what we know*. Cambridge: MIT Press.
Börner, K. (2011). Plug-and-play macroscopes. *Communications of the ACM*, *54*, 60–69.
Börner, K. (2015). *Atlas of knowledge: Anyone can map*. Cambridge: MIT Press.
Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, *17*, 2301–2309.
Boyack, K., Klavans, R., & Börner, K. (2005). Mapping the backbone of science. *Scientometrics*, *64*, 351–374.
Chen, Chaomei. (2006). CiteSpace II: Detecting and visualizing emerging trends and transient patterns in scientific literature. *Journal of the American Society for Information Science and Technology*, *57*, 359–377.
Comins, J., & Hussey, T. (2015a). Compressing multiple scales of impact detection by Reference Publication Year Spectroscopy. *Journal of Informetrics*, *9*, 449–454.
Comins, J., & Hussey, T. (2015b). Detecting seminal research contributions to the development and use of the global positioning system by reference publication year spectroscopy. *Scientometrics*, *104*, 575–580.
Comins, J., & Leydesdorff, L. (2016). *Identification of long-term concept-symbols among citations: Can documents be clustered in terms of common intellectual histories?* , arXiv preprint arXiv:1601.00288.
Comins, J. A., & Leydesdorff, L. (2016). RPYS i/o: software demonstration of a web-based tool for the historiography and visualization of citation classics, sleeping beauties and research fronts. *Scientometrics*, *107*, 1509–1517.
Cronin, B., & Sugimoto, C. (2014). *Beyond bibliometrics: Harnessing multidimensional indicators of scholarly impact*. MIT Press.
De Nooy, Wouter, Mrvar, Andrej, & Batagelj, Vladimir. (2011). . *Exploratory social network analysis with Pajek* (Vol. 27) Cambridge University Press.
de Solla Price, D. (1963). *Little science, big science*. New York: Columbia University Press.
DiMaggio, Paul. (2015). Adapting computational text analysis to social science (and vice versa). *Big Data & Society*, *2*, 1–5.
DiMaggio, P., Nag, M., & Blei, D. (2013). Exploiting affinities between topic modeling and the sociological perspective on culture: Application to newspaper coverage of US government arts funding. *Poetics*, *41*, 570–606.
Evans, J., & Foster, J. (2011). Metaknowledge. *Science*, *331*, 721–725.
Few, S. (2012). *Show me the numbers: Designing tables and graphs to enlighten*. Analytics Press.
Foster, J. G., Rzhetsky, A., & Evans, J. A. (2015). Tradition and innovation in scientists' research strategies. *American Sociological Review*, *80*, 875–908.
Gagolewski, M. (2011). Bibliometric impact assessment with R and the CITAN package. *Journal of Informetrics*, *5*, 678–692.
Garfield, E. (2009). From the science of science to Scientometrics visualizing the history of science with HistCite software. *Journal of Informetrics*, *3*, 173–179.
Garfield, E., Pudovkin, A., & Istomin, V. (2003). Why do we need algorithmic historiography? *Journal of the American Society for Information Science and Technology*, *54*, 400–412.
Garfield, E., Sher, I., & Torpie, R. (1964). The use of citation data in writing the history of science. In *Technical report*. DTIC Document.
Graham, S., Weingart, S., & Milligan, I. (2012). *Getting started with topic modeling and MALLET*.
Griffiths, T., & Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National academy of Sciences*, *101*, 5228–5235.
Hagberg, A., Schult, D., & Swart, P. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)* (pp. 11–15).
Handcock, M., Hunter, D., Butts, C., Goodreau, S., & Morris, Martina. (2008). statnet: Software tools for the representation, visualization, analysis and simulation of network data. *Journal of Statistical Software*, *24*, 1548.
Haustein, S., Peters, I., Sugimoto, C., Thelwall, M., & Larivière, V. (2014). Tweeting biomedicine: An analysis of tweets and citations in the biomedical literature. *Journal of the Association for Information Science and Technology*, *65*, 656–669.
Healy, K. (2011). Choosing your workflow applications. *The Political Methodologist*, *18*, 9–18.
Kessler, M. M. (1963). Bibliographic coupling between scientific papers. *American Documentation*, *14*, 10–25.
Kleinberg, J. (2003). Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, *7*, 373–397.
Kronegger, L., Mali, F., Ferligoj, A., & Doreian, P. (2012). Collaboration structures in Slovenian scientific communities. *Scientometrics*, *90*, 631–647.
LaRowe, G., Ambre, S., Burgoon, J., Ke, W., & Börner, K. (2009). The Scholarly Database and its utility for scientometrics research. *Scientometrics*, *79*, 219–234.
Lazega, E., Jourda, M.-T., Mounier, L., & Stofer, R. (2008). Catching up with big fish in the big pond? Multi-level network analysis through linked design. *Social Networks*, *30*, 159–176.
Leydesdorff, L., Bornmann, L., Marx, W., & Milojević, S. (2014). Referenced Publication Years Spectroscopy applied to iMetrics: Scientometrics, Journal of Informetrics, and a relevant subset of JASIST. *Journal of Informetrics*, *8*, 162–174.
Light, R., & Adams, J. (2016). Knowledge in motion: The evolution of HIV/AIDS research. *Scientometrics*, *107*, 1227–1248.
Light, R., Polley, D., & Börner, K. (2014). Open data and open code for big science of science studies. *Scientometrics*, *101*, 1535–1551.

Lucio-Arias, D., & Scharnhorst, A. (2012). Mathematical approaches to modeling science from an algorithmic-historiography perspective. In *Models of science dynamics*. pp. 23–66. Springer.
Marx, W. (2011). Special features of historical papers from the viewpoint of bibliometrics. *Journal of the American Society for Information Science and Technology*, *62*, 433–439.
Marx, W., & Bornmann, L. (2014). Tracing the origin of a scientific legend by reference publication year spectroscopy (RPYS): The legend of the Darwin finches. *Scientometrics*, *99*, 839–844.
Marx, W., Bornmann, L., Barth, A., & Leydesdorff, L. (2014). Detecting the historical roots of research fields by reference publication year spectroscopy (RPYS). *Journal of the Association for Information Science and Technology*, *65*, 751–764.
McFarland, D. A., Ramage, D., Chuang, J., Heer, J., Manning, C. D., & Jurafsky, D. (2013). Differentiating language usage through topic models. *Poetics*, *41*, 607–625.
McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. O'Reilly Media, Inc.
McLevey, J., & Anderson, J. Visualizing dynamic knowledge domains with network analysis and RPYS. Working paper.
Mohr, J., & Bogdanov, P. (2013). Introduction—Topic models: What they are and why they matter. *Poetics*, *41*, 545–569.
Morichika, N., & Shibayama, S. (2016). Use of dissertation data in science policy research. *Scientometrics*, 1–21.
Nichols, L. (2014). A topic model approach to measuring interdisciplinarity at the National Science Foundation. *Scientometrics*, *100*, 741–754.
Pérez, F., & Granger, B. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, *9*, 21–29.
Robbins, N. (2012). *Creating more effective graphs*. Wiley.
Romo-Fernández, L., Guerrero-Bote, V. P., & Moya-Anegón, F. (2013). Co-word based thematic analysis of renewable energy (1990–2010). *Scientometrics*, *97*, 743–765.
Rzhetsky, A., Foster, J. G., Foster, I. T., & Evans, J. A. (2015). Choosing experiments to accelerate collective discovery. *Proceedings of the National Academy of Sciences*, *112*, 14569–14574.
Shi, F., Foster, J., & Evans, J. (2015). Weaving the fabric of science: Dynamic network models of science's unfolding structure. *Social Networks*, *43*, 73–85.
Sinatra, R., Deville, P., Szell, M., Wang, D., & Barabási, A.-L. (2015). A century of physics. *Nature Physics*, *11*, 791–796.
Skupin, A., Biberstine, J., & Börner, K. (2013). Visualizing the topical structure of the medical sciences: A self-organizing map approach. *PLoS ONE*, *8*, e58779.
Small, H. (1973). Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for information Science*, *24*, 265–269.
Song, M., & Kim, S. Y. (2013). Detecting the knowledge structure of bioinformatics by mining full-text collections. *Scientometrics*, *96*, 183–201.
Stodden, V., Leisch, F., & Peng, R. (2014). *Implementing reproducible research*. CRC Press.
Sugimoto, C., Lariviere, V., Ni, C., Gingras, Y., & Cronin, B. (2013). Global gender disparities in science. *Nature*, *504*, 211–213.
Sugimoto, C., Li, D., Russell, T., Finlay, C., & Ding, Y. (2011). The shifting sands of disciplinary development: Analyzing North American Library and Information Science dissertations using latent Dirichlet allocation. *Journal of the American Society for Information Science and Technology*, *62*, 185–204.
Sugimoto, C. R. (2016). Is science built on the shoulders of women? A study of gender differences in contributorship. *Acad Med*, *91*, 1136–1142.
Thor, A., Marx, W., Leydesdorff, L., & Bornmann, L. (2016). Introducing CitedReferencesExplorer (CRExplorer): A program for reference publication year spectroscopy with cited references standardization. *Journal of Informetrics*, *10*, 503–515.
Tufte, E. (1990). *Envisioning information*. Graphics Press.
Uzzi, B., Mukherjee, S., Stringer, M., & Jones, B. (2013). Atypical combinations and scientific impact. *Science*, *342*, 468–472.
Van Eck, N. J., & Waltman, L. (2010). Software survey: VOSviewer, a computer program for bibliometric mapping. *Scientometrics*, *84*, 523–538.
Van Eck, N. J., & Waltman, L. (2014). CitNetExplorer: A new software tool for analyzing and visualizing citation networks. *Journal of Informetrics*, *8*, 802–823.
Wang, D., Song, C., & Barabási, A.-L. (2013). Quantifying long-term scientific impact. *Science*, *342*, 127–132.
Wang, P., Robins, G., Pattison, P., & Lazega, E. (2013). Exponential random graph models for multilevel networks. *Social Networks*, *35*, 96–115.
West, J. D., Jacquet, J., King, M. M., Correll, S. J., & Bergstrom, C. T. (2013). The role of gender in scholarly authorship. *PLoS ONE*, *8*, e66212.
White, H. (2011). Scientific and scholarly networks. In *The Sage handbook of social network analysis*. pp. 271–285. London: SAGE Publications.
Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, *59*.
Wray, B., & Bornmann, L. (2015). Philosophy of science viewed through the lense of "Referenced Publication Years Spectroscopy"(RPYS). *Scientometrics*, *102*, 1987–1996.
Xie, Y. (2015). . *Dynamic Documents with R and knitr* (Vol. 29) CRC Press.
Yan, E., & Ding, Y. (2012). Scholarly network similarities: How bibliographic coupling networks, citation networks, cocitation networks, topical networks, coauthorship networks, and coword networks relate to each other. *Journal of the American Society for Information Science and Technology*, *63*, 1313–1326.
Yau, C.-K., Porter, A., Newman, N., & Suominen, A. (2014). Clustering scientific documents with topic modeling. *Scientometrics*, *100*, 767–786.