Adding Neotoma Publications

Simon Goring

10 March, 2016

Abstract

This document details the process of pulling title strings from the Neotoma Database (using a MySQL dump of the database) and pushing the parsed titles into a large table. This parsed table is then used to pull close matches from the CrossRef API.

Contents

Project Purpose

Currently (09/03/2016) the Neotoma Database has limited information in the `publication` table. The information is stored as a wide table, but many records only contain a `PublicationID` key, year of publication and the full text citation, along with ContactIDs for the main authors. This links through the `publicationauthors` table, but in practice the fields there are not always filled (although the `ContactID` is filled).

PublicationID	Year	Citation	ArticleTitle
1	2006	International Organization for Standardization. 2006. Codes for the representation of names of countries and their subdivisions â try codes. ISO 3166-1, Geneva, Switzerland.	Codes for the representation of names of countries and their subdivisions â Part 1: Country codes Part 1: Coun-

Of all the 6288 records, 6065 have no existing DOI. Additionally, 4792 have no reliable title (except as part of the citation). No record (as far as I can tell) contains the full, rich data needed for bibJSON output.

bibJSON Briefly

A fuller definition of the standard is available online, but the idea is for a light & portable standard to represent bibliographic metadata using JSON (or JSON-LD). While bibJSON uses a default set of keys defined for `BibTeX`, part of bibJSON's beauty is that it has no defined namespace limits, so if we decide, as a community, to add extra data to the format (like the inclusion of IGSNs), then we can do that as well without breaking the standard.

The bibJSON Schema

The bibJSON format looks something like this (from http://okfnlabs.org/bibjson/):

```
{
    "title": "Open Bibliography for Science, Technology and Medicine",
    "author":[
        {"name": "Richard Jones"},
        {"name": "Mark MacGillivray"},
        {"name": "Peter Murray-Rust"},
        {"name": "Jim Pitman"},
        {"name": "Peter Sefton"},
        {"name": "Ben O'Steen"},
        {"name": "William Waites"}
    ],
    "type": "article",
    "year": "2011",
    "journal": {"name": "Journal of Cheminformatics"},
    "link": [{"url":"http://www.jcheminf.com/content/3/1/47"}],
    "identifier": [{"type":"doi","id":"10.1186/1758-2946-3-47"}]
}
```

That seems fairly straightforward. It maps well onto BibTeX and gives us a rich set of information that we can ingest into any number of other uses (like LiPD). Can Neotoma map to it easily?

How well does Neotoma Map onto bibJSON?

This is a fairly naive attempt to map on to bibJSON. I could try harder, but I think the point is made here. Let's assume we're using a Beringian paper as the test object. The paper is:

> Anderson, P.M., and A.V. Lozhkin. 2001. The latest Pleistocene interstade (Karginskii/Boutellier interval) of Beringia: variations in paleoenvironments and implications for paleoclimatic interpretations. *Quaternary Science Reviews*, **20**: 93-125.

The paper has a `PublicationID` of 848 in the Neotoma Database:

```
library(jsonlite)

bibJSON_test <- toJSON(list(title = dbGetQuery(mydb, "select ArticleTitle from publications \n
    author = dbGetQuery(mydb, "select concat(LeadingInitials, ', ', FamilyName) as name from contacts\n
    type = dbGetQuery(mydb, "select ArticleTitle from publications where PublicationID=848"),
    year = dbGetQuery(mydb, "select Year from publications where PublicationID=848"),
    journal = dbGetQuery(mydb, "select Journal from publications where PublicationID=848"),
    link = dbGetQuery(mydb, "select URL from publications where PublicationID=848"),
    identifier = dbGetQuery(mydb, "select DOI from publications where PublicationID=848")))
```

{“title”:[{}],“author”:[{“name”:“P.M., Anderson”},{“name”:“A.V., Lozhkin”}],“type”:[{}],“year”:[{“Year”:“2001”}],“journal”:[{

For many records even basic metadata is missing. There are 4792 records out of 6288 that are missing the article title. This isn’t the end of the world, but we want to start making a crack at fixing it.

The CrossRef Metadata API

Luckily, we can make use of the CrossRef Metadata API, which can return document metadata from DOIs, and we can use this to fill in our records. The CrossRef Metadata for the paper with DOI `10.1016/j.yqres.2009.05.006` (Eric’s conventional bulk sediment error paper) looks something like this:

```
{"status":"ok",
  "message-type":"work",
  "message-version":"1.0.0",
  "message":{"indexed":{"date-parts":[[2015,12,22]],
                        "date-time":"2015-12-22T11:07:51Z",
                        "timestamp":1450782471165},
             "reference-count":44,
             "publisher":"Elsevier BV",
             "issue":"2",
             "license":[{"content-version":"tdm",
                         "delay-in-days":0,
                         "start":{"date-parts":[[2009,9,1]],
                                  "date-time":"2009-09-01T00:00:00Z",
                                  "timestamp":1251763200000},
                         "URL":"http:\/\/www.elsevier.com\/tdm\/userlicense\/1.0\/"}],
             "published-print":{"date-parts":[[2009,9]]},
             "DOI":"10.1016\/j.yqres.2009.05.006",
             "type":"journal-article",
             "created":{"date-parts":[[2009,7,10]],
                        "date-time":"2009-07-10T09:40:02Z",
                        "timestamp":1247218802000},
                        "page":"301-308",
                        "source":"CrossRef",
                        "title":["The magnitude of error in conventional bulk-sediment radiocarbon date
                                  America"],
                        "prefix":"http:\/\/id.crossref.org\/prefix\/10.1016",
                        "volume":"72",
                        "author":[{"affiliation":[],"family":"Grimm","given":"Eric C."},
                                  {"affiliation":[],"family":"Maher","given":"Louis J.","suffix":"Jr."}
                                  {"affiliation":[],"family":"Nelson","given":"David M."}],
                        "member":"http:\/\/id.crossref.org\/member\/78",
                        "container-title":["Quaternary Research"],
                        "link":[{"intended-application":"text-mining",
                                 "content-version":"vor",
                                 "content-type":"text\/xml",
                                   "URL":"http:\/\/api.elsevier.com\/content\/article\
                                          /PII:S003358940900057X?httpAccept=text\/xml"},
                                    {"intended-application":"text-mining",
                                     "content-version":"vor",
                                     "content-type":"text\/plain",
                                     "URL":"http:\/\/api.elsevier.com\/content\/article\
                                            /PII:S003358940900057X?httpAccept=text\/plain"}],
                          "deposited":{"date-parts":[[2015,12,19]],
                                        "date-time":"2015-12-19T21:05:03Z",
                                        "timestamp":1450559103000},
                          "score":1.0,
                          "subtitle":[],
                          "issued":{"date-parts":[[2009,9]]},
                          "alternative-id":["S003358940900057X"],
                          "URL":"http:\/\/dx.doi.org\/10.1016\/j.yqres.2009.05.006",
                          "ISSN":["0033-5894"],
                          "subject":["Earth-Surface Processes","Earth and Planetary Sciences(all)"]}}
```

So . . . pretty rich.

We need to be able to pull out, at minimum information about article `"type"`, a link (`"created":"URL"`) & identifier (`"DOI"`), author names (`"created":"author"`), year of publication (`"created":"date-parts"`) & journal (`"created":"container-title"`). Given that I've spelled out the location for each of these in the CrossRef JSON schema it's likely that we'll be able to pull those in for articles contained in CrossRef (presuming we can find them).

Getting Started - Neotoma

To pull out bibliographic information from Neotoma we can try parsing individual text-string citations from Neotoma using regular expressions, or use a hybrid method, that both parses title or text strings from the text-string Neotoma citations and then makes use of of the CrossRef and JSTOR APIs to obtain more information about the individual records.

This workflow then looks like this:

1. Load in the Neotoma publication table
2. For records without "Title" fields or DOIs (but with full citations):

a. Use regular Expression matching to extract the likely "Title" field
b. Pass the string to the CrossRef API to get the five closest matches
c. All 5 matches go into a list called `results_list`
d. Closest match & DOI go into `results_frame` along with an estimate of distance.

To start, we create the variables (or load them if we've done partial runs earlier):

```r
library(devtools)
library(rcrossref)
library(jsonlite)
library(stringr)
library(stringdist)

# Some of this analysis is time consuming.  To save time in
# multiple runs we save as we go (see below), so the first
# thing we do is look for evidence that a past run has
# occurred.

publications <- dbGetQuery(mydb, "select * from publications")

if ("results_frame.csv" %in% list.files()) {
    results_frame <- read.csv("results_frame.csv")
    results_list <- readRDS("results_list.RDS")
} else {
    results_frame <- data.frame(hits = rep(NA, nrow(publications)),
        query = rep(NA, nrow(publications)), best_title = rep(NA,
            nrow(publications)), proximity = rep(NA, nrow(publications)),
        prox_pct = rep(NA, nrow(publications)), best_doi = rep(NA,
            nrow(publications)))
    results_list <- list()

    publications$Modified <- NA
}
```

We've now done a few things. One, I wrote the code so that it's ultimately going to spit out a table called `results_frame`, and, if that table exists, it will load it (so we can avoid running through records we've already checked). Then, for each record we use the `rcrossref` package's `cr_works_` function (which returns results in JSON format) to find the closest match.

There's a few other things that the code below does, in part because of the particulars of the `publications` table. For example, we pull the title out of the full citation string by searching for periods in the text string. We use regular expresson matching to match `'(18|19|20)\\d{2}\\. '` which should catch anything with the form `'1800. '`, basically, a year in the 19th, 20th or 21st century, followed by a period and a space.

The title (in general) should be bound by the date and the next period. In practice this doesn't exactly work, things like `U.S.A.` get truncated. To fix this we need to generate a more complex regular expression search (further down in the code block, where we use the `str_locate_all` block. Given the more complex search we get a long enough query that we can match a record in CrossRef. Once we get the match we can pull the DOI metadata and fill the fields properly.

There are still some problems, mostly date issues. For example, there are some papers listed as `n.d.`, or with no trailling period in the age field. In these cases I'm going to let them pass on as `NA` records. They ultimately need more looking into, so it's okay to let them screw up for now.

```r
# For each record:
for (i in 1:nrow(publications)) {
    # Only run for records without DOIs or filled results:
    if (is.na(publications$DOI[i]) & is.na(results_frame$hits[i])) {
        # If we need to strip out the title from the paper citation
        # (it hasn't been done yet in Neotoma):
        if (is.na(publications$ArticleTitle[i])) {
            ## We want to pull a title in from the full citation:
            year_text <- str_locate(publications$Citation[i],
                "(18|19|20)\\d{2}[a-zA-Z]{0,1}\\. ")

            # The regular expression string here searches for all
            # periods, except those preceeded by a St (shortening
            # 'Saint'), or preeceding an 'A.' (so we don't have to worry
            # about the 'S').  If the U.S.A is at the end of a title we
            # can't ignore the period, but we can if it is followed by a
            # comma or a lowercase letter.
            periods <- str_locate_all(publications$Citation[i],
                "(?<!(U|St))\\.(?!((A\\.)|,|:|( [a-z])))")[[1]]

            if (!any(is.na(year_text)) & max(year_text) < max(periods)) {
                # The date has valid information, then get the title (or at
                # least part of the title between a set of paired periods).
                # This chokes on things like 'U.S.', for now that's a small
                # subset.

                title <- substr(publications$Citation[i], start = year_text[2] +
                  1, stop = min(unlist(periods)[unlist(periods) >
                  max(year_text)]))
                year <- substr(publications$Citation[i], start = year_text[1],
                  stop = year_text[2] - 2)

                # I've added a flag for the base publication table so I can
                # differentiate between publications that have had their
                # titles entered by hand, and those I've done
                # programmatically.
                publications$Year[i] <- year
                publications$ArticleTitle[i] <- title
                publications$Modified[i] <- TRUE
            } else {
                # If there's no apparent date field.  Works out to ~ 210
                # records.
                title <- NA
                year <- NA
            }
        } else {
            ## If the ArticleTitle is there:
            title <- publications$ArticleTitle[i]
            year <- publications$Year[i]
        }

        # If we were able to recover a title to use in the search:
        if (!is.na(title)) {
            # and if there's a year assigned:
            if (!is.na(as.numeric(year))) {
                # This adds a filter to clean up the references, so that
                # we're only pulling the closest set:
                year_pub <- paste0(year, c("-01-01", "-12-31"))
                records <- jsonlite::fromJSON(cr_works_(query = title,
                  filter = c(from_print_pub_date = year_pub[1],
                    until_pub_date = year_pub[2]), limit = 5))
            } else {
                # If there's no year assigned we can't limit the search:
                records <- jsonlite::fromJSON(cr_works_(query = title,
                  limit = 5))
            }

            # We've created a search that looks for the title string, and
            # limits by publication date.  Because we're only looking for
            # very close matches we're just going to take the closest
```

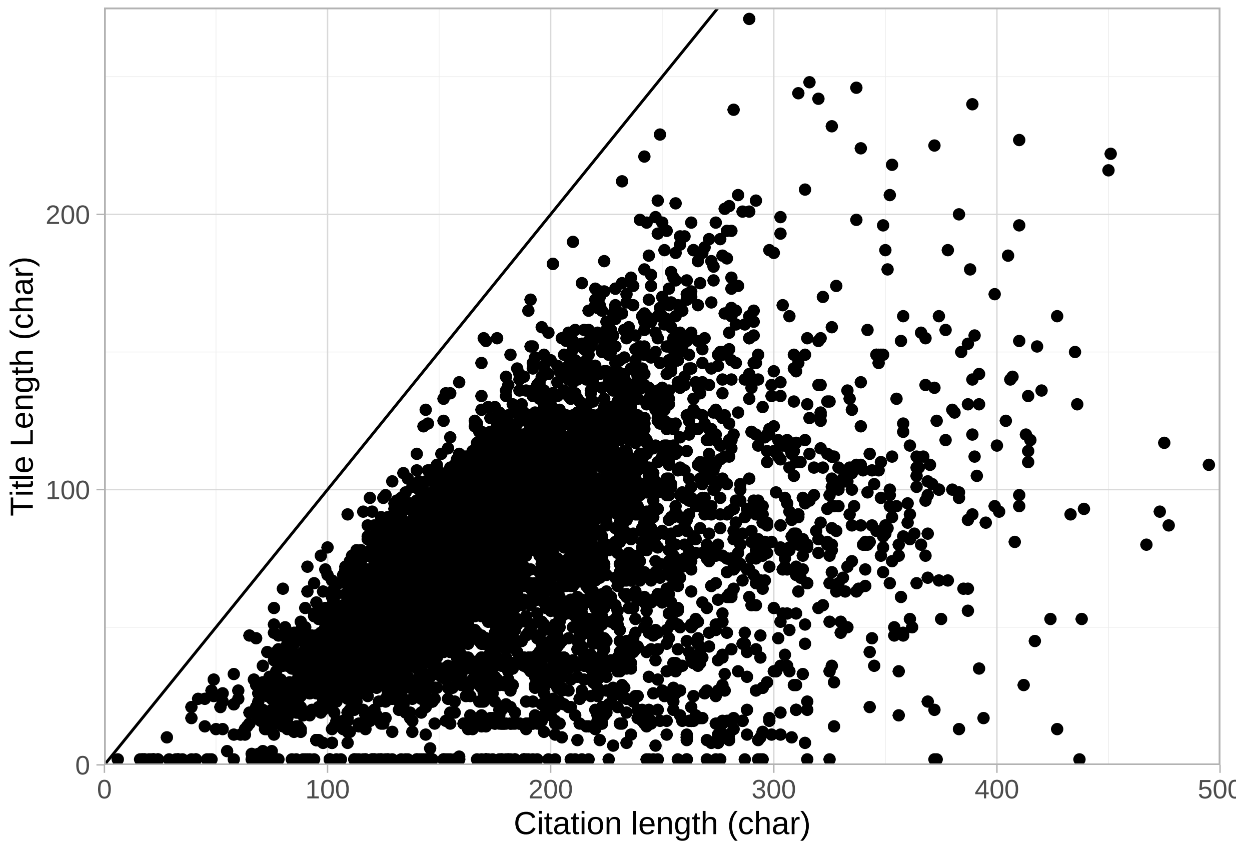
```
            if (records$status == "ok" & !is.null(records$message$items$title)) {
                # How many insertions or deletions are needed to find the
                # most similar title: for example: aaa to bbb should give 3,
                # while aba to bbb gives 2.  In general we should suspect
                # anything over ~10 or so. . .

                edit_dists <- sapply(records$message$items$title,
                  function(x) adist(tolower(x[1]), tolower(title)))
                matched_title <- which.min(edit_dists)

                if (!all(sapply(records$message$items$title,
                  length) == 0)) {
                  # We have a good record, now lets extract the information:
                  results_frame$hits[i] <- records$message$`total-results`
                  results_frame$query[i] <- records$message$query$`search-terms`
                  results_frame$proximity[i] <- edit_dists[matched_title]
                  results_frame$prox_pct[i] <- edit_dists[matched_title]/nchar(records$message$items$ti
                  results_frame$best_title[i] <- records$message$items$title[[matched_title]]
                  results_frame$best_doi[i] <- records$message$items$DOI[[matched_title]]

                  # We're storing this so that if there are bad matches we can
                  # see if it's because of the distance measure we're using or
                  # not.
                  results_list[[i]] <- unlist(records$message$items$title)
                }
            }
        }
        write.csv(results_frame, "results_frame.csv", row.names = FALSE)
        write.csv(publications, "pubs_edited.csv", row.names = FALSE)
        saveRDS(results_list, "results_list.RDS")
    } else if (!is.na(publications$DOI[i]) & is.na(results_frame$hits[i])) {
        # If there's already a record, but it's not in the
        # `results_frame`:
        results_frame$hits[i] <- NA
        results_frame$query[i] <- publications$ArticleTitle[i]
        results_frame$best_doi[i] <- publications$DOI[i]
    }
}
```

We're putting the output of the CrossRef API into a couple of `csv` files for readability & hand editing. The `pubs_edited.csv` file is the publications table. The `results_frame.csv` has more content, it includes:

- `hits`: The number of matching references found in CrossRef
- `query`: Equivalent to the `ArticleTitle` in the Publications table.
- `best_title`: The title CrossRef returns as the "best match".
- `proximity`: The number of edits required to convert one title to the next (*e.g.*, From "hat" to "hit" is 1 edit, from "hat" to "horse" is 4, from "Up" to "up" is 1).
- `prox_pct`: The number of edits as a function of the title length.
- `best_doi`: The DOI associated with the best match.

We also save the list of titles returned in the list `results_list`, just to check that "wrong" titles aren't getting picked over titles that seem more correct. We need to figure out a way to check these more reliably.

The regular expression searching works well in general. Most article titles are of reasonable length and appear to match well. This is a rough representation, but we see in general that titles are about as long as the full citation string, but somewhat shorter. There are a bunch of results with a length of 2. These are results for which the query is “NA”. I’m not entirely sure why that’s happening.



From this, we need to the go and identify which articles are good matches, by hand.