The effects of transitions of power on the contents of municipal government websites

Markus Neumann Bruce Desmarais Hanna Wallach

October 16, 2017

Abstract

Websites have become a prominent source of data for automated text analysis in political science. However, extant research lacks common standards and often glosses over the details on how such analyses are conducted, exposing itself to potential pitfalls associated with this process and making replication difficult. We develop a set of guidelines and procedures to be followed in order to produce valid results. In order to develop a valid research design, the appropriate selection of cases and URLs is crucial. For the acquisition of website data, we cover several scraping methods and difficulties that can arise in the process. Pre-processing is a common step in text analysis, but when websites are concerned, additional measures need to be taken in order to guard against potential sources of bias. Finally, we cover several methods of analysis and validation appropriate for this kind of data. These steps are illustrated through our creation and exploitation of a new and innovative dataset - the websites of local governments in Indiana and Louisiana. We show that if our methodology is followed appropriately, an association between mayoral partisanship and the content of their cities' websites becomes visible.

1 Introduction

The analysis of entire websites has become more prominent recently, especially in the the text-as-data movement. We see great promise in this development, especially as it pertains to the study of governmental branches and agencies, which are often resilient to being studied by other methods. However, this line of inquiry comes with a set of particular challenges and pitfalls researchers need to be mindful of. We offer solutions to these problems at the four stages of such an analysis.

Contents

1 Introduction **1**

2 Research Design **3**
2.1 Sample Selection . . . 3
2.2 Finding & Verifying URLs . . . 3
2.3 Supporting Data . . . 3

3 Scraping Websites **3**
3.1 wget . . . 3
3.2 Parsing HTML with rvest/beautifulsoup . . . 3
3.3 Browser Automation with Selenium . . . 4
3.4 The Wayback Machine and other APIs . . . 4

4 Pre-processing **4**
4.1 Determining document filetype . . . 4
4.2 File conversion . . . 5
4.3 Conventional preprocessing (lowercase, numbers, punctuation, stopwords) . . . 5
4.4 Stemming and/or lemmatization . . . 5
4.5 Spellchecking . . . 5
4.6 Dealing with duplicate text & documents & html documents in particular . . . 5

5 Analysis **6**
5.1 LDA . . . 6
5.2 Other topic models . . . 6
5.3 Machine Learning Classifiers (SVM) . . . 6
5.4 Fightin' Words . . . 6
5.5 Non-bag of words models . . . 6

2 Research Design

2.1 Sample Selection

Running example: Why did we pick Indiana and Louisiana?

2.2 Finding & Verifying URLs

Running example: Our URLs are partially from the General Services Administration (GSA), and partially from Wikipedia. The GSA data can easily downloaded from GitHub, but contains some errors. To at least verify which parts of it work, we open each url in an automatized browser, and test whether it leads to a valid website as well as record the URL it redirects to. An explanation of this process is provided in section 3.3. Scraping URLs from Wikipedia will be explained in section 3.2.

Current scripts:

2.3 Supporting Data

In our case, this would be the Census and election data. However, this is incredibly case-dependent, so I am not sure whether this really needs its own section.

Current scripts: cityCoordinates.R (scrapes city coordinates from Wikipedia), combineURLs.R (combines Census with GSA data)

3 Scraping Websites

3.1 wget

wget arguments we are using, and thus might be worth discussing:
-r, –recursive, specify recursive download
-N, –timestamping, don't re-retrieve files unless newer than local; alternatively:
-nc, –no-clobber, skip downloads that would download to existing files (overwriting them)
-P, –directory-prefix=PREFIX, save files to PREFIX/..
-i, –input-file=FILE, download URLs found in local or external FILE
robots=off, we are NOT doing this currently, but I feel we should at least explain it
Functions: Do we write a wrapper for wget here?
Current scripts: wgetINLA.R

3.2 Parsing HTML with rvest/beautifulsoup

Running example: scraping city URLs from Wikipedia.

3.3 Browser Automation with Selenium

Running example: checking whether URLs from Wikipedia/, are valid, and/or whether they redirect somewhere else.

Also very useful for scraping websites that have some kind of scraping protection (we don't do this, but the reader might want to).

Functions: Note sure. There is an R implementation for selenium, but it is ridiculously difficult to install and still clearly inferior to the Python version.

Current scripts: govWebsitesVerification.py

3.4 The Wayback Machine and other APIs

This is not part of our running example. However, other people might still be able to use the Wayback Machine for their research designs, so we should at least describe it, especially now that we know its downsides.

To provide an example, we could do what we planned to do, but just for one large city, which has been scraped enough times by the WBM that it would be no problem. San Diego is currently the largest city with a Republican mayor, who got elected in 2014 (preceded by a Democrat), and there are enough snapshots around this time, so that would be one option. This would allow us to demonstrate how to use an API, which is possibly relevant to a lot of people (but then again, maybe not 100% relevant since we are focusing on scraping websites).

The implementation of this whole part would be a bit of a problem, since we are currently using a Ruby package, and from what I can tell looking at its code, I am not sure if I have the engineering skills to translate it to R.

4 Pre-processing

4.1 Determining document filetype

Determining the format of a file is usually as simple as looking at its file ending. However, not all files scraped from websites actually have endings. Simply discarding these files would lead to a large and completely unnecessary reduction in the size of the corpus. Similarly, xml files are sometimes labeled as html, which, if parsed as such, will lead to the introduction of a set of words into the corpus that have absolutely nothing to do with a website's content. Consequently it is imperative to determine a document's type by reading in its first couple of lines and extracting the tag revealing its type. In our running example, this leads to an increase of the corpus by X% for Indiana and Y% for Louisiana.

Function to include in the package: Read in a document, do a grep on its first few lines for either "doctype HTML" or "PDF"; then output the correct filetype, and if desired, change the file. Currently done in 'renameHTMLs.R'.

Note: I've had some nasty problems with text not being encoded properly, or at least consistently. The readtext[1] package seems to have a way for dealing with that, which I haven't looked into so far, but considering the new research design, this might be worth it.

Current scripts: detectOriginalExtensions.R, filetype_before_txt.R

4.2 File conversion

File conversion can either be done with various Unix packages, or Kenneth Benoit's readtext package. Currently, we are doing this with the former, using antiword, docx2txt, html2text and pdftotext (currently done with the shell script "websites/batchconversion.sh", which in turns calls four other shell scripts). If we are going to make an R package for our paper, it might be easier to do it with readtext instead. From what I can tell, readtext uses htmlTreeParse (XML package), pdftools (R package; but they previously used xpdf, which also relies on the same pdftotext we are using), antiword (which we are using too, but they are using the R wrapper for it) and xmlTreeParse for docx (XML package). I did try the readtext package at one point, and it lead to fairly comparable results, but I will need to do more testing if we do switch to it to make sure it doesn't mess anything up later.

Functions: wrapper for readtext

Current scripts: malletPreprocessing.R

4.3 Conventional preprocessing (lowercase, numbers, punctuation, stopwords)

Package: Wrapper for quanteda and/or tm package

Current scripts: All of the preprocessing from section 4.3 to 4.6 is currently done in malletPreprocessing.R (which doesn't require mallet anymore, so it's a bit of a misnomer).

4.4 Stemming and/or lemmatization

We're currently not doing this and I am wondering if we will get spanked for it by reviewers.

4.5 Spellchecking

Current scripts: malletPreprocessing.R

4.6 Dealing with duplicate text & documents & html documents in particular

Current scripts: malletPreprocessing.R

[1] http://cdn.rawgit.com/kbenoit/readtext/master/inst/doc/readtext_vignette.html#read-files-with-different-encodings

5 Analysis

5.1 LDA

Diagnostics: Topic coherence, optimal number of topics according to Arun et al. (2010).
Current scripts: exampleDocuments.R

5.2 Other topic models

STM, could also mention author-topic and dynamic topic model.

5.3 Machine Learning Classifiers (SVM)

Phil Schrodt and Glenn Palmer (+2 other people) wrote a whole paper on why SVM is good for text data, we should cite this here.

We can also mention l1/l2 penalty / elasticnet here.

Since the scikitlearn implementations of SVM and elasticnet seem much better than the ones available for R, what do we do here, package-wise?

Current scripts: svmSKLN.ipynb (Indiana), svmSKLNLA.ipynb (Louisiana), elasticNet.R (Indiana)

5.4 Fightin' Words

Make a table comparing the top 50 words for Democrats/Republicans produced by LDA, SVM and Fightin' Words. Probably goes in the appendix.

Package: Wrapper for Matt's speedReader package.
Current scripts: fightinWords.R, fightinWordsLA.R

5.5 Non-bag of words models

We're only using bag of words models, so do we focus only on them in this paper? Either way, we should at least point out that this would change some parts of these procedures, especially during preprocessing.