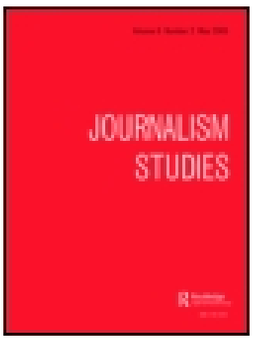

Journalism Studies

ISSN: 1461-670X (Print) 1469-9699 (Online) Journal homepage: http://www.tandfonline.com/loi/rjos20

Capturing and Analyzing Liquid Content

Rodrigo Zamith

To cite this article: Rodrigo Zamith (2016): Capturing and Analyzing Liquid Content, Journalism Studies

To link to this article: http://dx.doi.org/10.1080/1461670X.2016.1146083

Published online: 23 Feb 2016.

CAPTURING AND ANALYZING LIQUID CONTENT
A computational process for freezing and analyzing mutable documents

Rodrigo Zamith

As scholars take greater interest in analyzing digital content, they are being presented with novel methodological challenges. Among these challenges are dealing with the volume and mutable nature of digital content, and especially online content. The present work discusses a computational approach for "freezing" and analyzing aspects of a large volume of "liquid" content using an accessible combination of free software running on consumer hardware. The strengths and limitations of this approach are illustrated through an examination of the method employed in an analysis of more than 125,000 snapshots of the homepages of 21 news organizations over two months.

KEYWORDS algorithmic content analysis; computational content analysis; liquid news; online news; research methods; social science; Web analysis

Introduction

According to Shah, Cappella, and Neuman (2015), we are in the midst of a turn toward computational social science, a paradigmatic shift characterized by the use of large and complex datasets drawn from digital media sources that must be analyzed through the use of computational and algorithmic solutions. This is not to imply that small-scale or manual analyses are going away; they are not and in many cases offer the most appropriate approach (Crawford, Gray, and Miltner 2014). Rather, it points to a growing recognition of the benefits (though with their own unique limitations) of using computational tools to drive analyses (Zamith and Lewis 2015). Within the field of mass communication alone, a diverse array of social scientific research adopting computational methods has been published in recent years, involving the analysis of anywhere from tens of thousands to billions of units—tweets, forum postings, news articles, and the like (Grimmer and Stewart 2013; Hermida, Lewis, and Zamith 2014; Sormanen et al. 2016).

In conjunction with these empirical analyses, mass communication scholars have made a series of methodological contributions in recent years to guide automated and semi-automated analyses of content (Grimmer and Stewart 2013; Lewis, Zamith, and Hermida 2013; Sjøvaag and Stavelin 2012; Widholm 2016). One area in this stream of research that has received substantial attention in recent years has been the analysis of "liquid" content, or content that is not only mutable but constantly changing (Deuze 2008; Karlsson 2012; Karlsson and Sjøvaag 2016; Karlsson and Strömbäck 2010; see also Lowrey 2006; Matheson 2004; Sjøvaag, Stavelin, and Moe 2015). Of particular concern

Journalism Studies, 2016
http://dx.doi.org/10.1080/1461670X.2016.1146083

within that stream is how to effectively capture and analyze rapidly changing Web content, which introduces an array of methodological challenges (Lewis, Zamith, and Hermida 2013; Sjøvaag and Stavelin 2012).

This paper focuses on delineating a process for computationally capturing and analyzing aesthetic facets of rapidly changing Web content (e.g., where news items are placed relative to one another) using as a case study the analysis of the homepages of 21 US-based news organizations. It begins with a review of the literature on liquid content and the methodological challenges associated with studying it. Then, existing approaches to "freezing" such content and computationally analyzing Web documents are considered. Finally, these insights are synthesized and built upon as part of an analysis of more than 125,000 documents, with the advantages and disadvantages of the process discussed.

Background

As several scholars have observed, online news is distinct from its analog counterparts, with the characteristics of immediacy and interactivity often touted as key distinguishing characteristics (Boczkowski 2004; Karlsson 2011). Interactivity has been described as "the extent to which users can participate in modifying the form and content of a mediated environment in real time" (Steuer 1992, 84), and broadly refers to the additional control granted to the consumers of the content. This may include commenting on a news item, filtering the content that appears on a site, or choosing whether or not to play a video or maximize an image that is appended to an article. Interactivity effectively sets online journalism apart by enabling greater integration of user-generated content and by allowing users to engage with content in different ways.

Immediacy, in turn, refers to the absence of a delay between when producers create or update content and when consumers can view that content (Lim 2012). In a related context, immediacy may also refer to the expectation among online news consumers that content will be updated frequently to ensure that fresh material appears the next time a page is accessed (García Avilés et al. 2004). This, in turn, has altered journalism as it has become "less a product than a process, witnessed in real time and in public" (Tumber 2001, 98), therefore moving from a "black box" model to something less opaque (Deuze 2008; Karlsson 2011). Collectively, interactivity and immediacy have promoted the development of what researchers have varyingly termed "liquid," "fluid," and "dynamic" journalism (Deuze 2008; Karlsson and Strömbäck 2010).

Although there are various ways to study a website's liquidity (see Karlsson 2012; Widholm 2016), one approach is to focus on the extent to which content in key areas of a page changes. The homepage is one particular section of a website that news consumers would expect to be especially fluid through the addition of breaking news and updates to existing articles (Sjøvaag, Stavelin, and Moe 2015). Though homepages no longer serve as the primary point of entry for many news consumers, they are nevertheless generally viewed by news organizations as an important page through which to build an audience and engage in core functions of newswork, like communicating news priorities (Benton 2015).

However, as Lim (2012) notes, the notion that the homepages of news organizations are constantly changing has long been taken for granted by scholars, and that empirical work assessing the extent to which news homepages change over the course of the day

remains fairly limited. The prospect of constantly changing websites offers mass communication researchers several opportunities for studying the process of how particular news content evolves over short periods of time (Karlsson 2012; Lim 2012; Sjøvaag, Stavelin, and Moe 2015). However, that prospect also introduces a number of methodological challenges for the researcher (Karlsson and Sjøvaag 2016; Lewis, Zamith, and Hermida 2013; Sjøvaag and Stavelin 2012). In particular, as Deuze notes,

> the study of content has always rested on the premise that content actually exists, that it genuinely can be considered as a finished, static object of study. In the current media ecology of endless remixes, mashups, and continuous edits, that is a problematic assumption. (Deuze 2008, 861)

This begs the question: how should one capture dynamic objects that may constantly be in a state of flux?

Freezing Liquid Content

In order to analyze dynamic objects like homepages, it is often necessary to first "freeze" them (Karlsson and Strömbäck 2010, 16). In a manual content analysis, for example, turning dynamic objects into static ones is essential for ensuring that multiple coders are able to view the same content when establishing intercoder reliability. Similarly, in a computational content analysis, freezing objects is useful for training, tweaking, and testing an algorithm. In both instances, it is necessary to freeze content in order to reproduce the research (Zamith and Lewis 2015).

Unfortunately for researchers, the act of freezing objects must often be performed by the analyst. For example, although there are a few large online archives of websites, even the most robust of these, the Internet Archive, will, at most, take only a handful of electronic snapshots each day of the homepages of large news organizations like the *New York Times*. Furthermore, it will often fail to archive homepages of smaller organizations for several days —and even weeks—at a time. In the absence of a third-party archive, researchers must create their own archive for the content they wish to study.

Karlsson and Strömbäck (2010) point to three techniques for accomplishing this. The first technique is to simply take screenshots of the content. As they note, the advantage of using a screenshot is that it accurately captures the appearance of content as it is displayed under certain conditions. However, they argue that screenshots only capture what appears in a specific frame—that is, it fails to capture content that one would need to scroll up and down to see. More recently, Widholm (2016) developed a new application for his Regular-interval Content Capturing method that is capable of capturing the entire screen and outputting the result to a JPEG image. It is unclear, however, if that application works across popular operating systems, such as Windows or Linux. Furthermore, several pieces of information are lost when interactive objects are transformed into flat images (e.g., a JPEG), such as the URL that a headline links to.

The second technique is to print a copy of the object, either on paper or to a PDF document. As Karlsson and Strömbäck (2010) note, this allows the entire page to be captured as a single static object, but it often triggers a "printable version" of the page that excludes important information (e.g., certain images or interactive features). Moreover, when a printable version is not available or bypassed, aesthetic elements are often forcibly rearranged in the printed copy to ensure that the content fits a standard size (e.g.,

letter-size paper). As with the screenshots, a great deal of information is lost during this transformation.

The third technique is to "mirror" the page—that is, to download the source code necessary to render the page, including all of the associated media content such as the images and style sheets. One may do this manually by using the page-saving feature of their preferred browser. Alternatively, one may do this computationally through the use of free and commercial software, such as HTTrack or WebCopy. While this technique is advantageous in its ability to capture the greatest amount of information about the dynamic objects (e.g., information about the structure of the document, such as the styling of a particular element and link information), these approaches often fail to capture all of the information necessary to exactly replicate the page (e.g., downloading and relinking dependencies). That is, even when all the constituent elements are downloaded, they often fail to later reproduce the exact appearance of a page at the time it was mirrored.

The study of the liquidity of content often requires researchers to look at short periods of time (Lim 2012). For example, during a breaking news event, an article may be updated several times over the course of a day and an organization's homepage may change the art associated with that item repeatedly during the initial stages of reporting. This is an important consideration because research designs that involve short intervals tend to generate large datasets. Indeed, as Karlsson (2012) notes, the dearth of empirical analyses of the liquidity of content is likely due to the difficulty of the work. How, then, can such analyses, especially those aiming at a larger scale, be comprehensively performed through computational means?

Computationally Analyzing Homepages

In order to tackle larger-scale analyses, it is preferable to identify computational solutions (Shah, Cappella, and Neuman 2015). Computers are sequential, deterministic machines capable of processing vast amounts of numerical and textual data with exceptional speed and perfect reliability. These two features—speed and reliability—have long made computers an enticing aide for scholars interested in content analysis, with early applications of computer-assisted content analysis dating back to the 1950s and 1960s.

According to Zamith and Lewis (2015), computational approaches to content analysis offer a range of benefits, such as increased efficiency, transparency, and *post hoc* malleability, even as they are limited by the kind of content they are able to process. Indeed, computational aides for content analysts have been critiqued as often being difficult to use and yielding results of questionable validity (Mahrt and Scharkow 2013). This has led some scholars to argue that, in contexts where latent meanings are of interest, a hybrid approach that blends computational and manual traditions is favorable (Lewis, Zamith, and Hermida 2013; Sjøvaag, Moe, and Stavelin 2012). However, when the variables of interest are unambiguous, such as the presence or absence of a specific piece of computer code in the source code of a Web document or a change in the text of a document, computational analyses are preferable (Zamith and Lewis 2015).

The study of liquidity often centers on change—the addition of some elements and the removal of others—on digital documents like Web pages (Karlsson 2012; Karlsson and Strömbäck 2010; Lim 2012; Sjøvaag, Stavelin, and Moe 2015). Web pages are, by design, highly structured digital documents. Among the websites of news organizations, these

pages typically consist of a mixture of HyperText Markup Language (HTML) and JavaScript, a dynamic programming language that adds more sophisticated functions to a page through client-side scripts. Both the HTML and JavaScript code are interpreted by the browser when a user accesses a page and the browser renders that code into the audiovisual objects the user sees and hears when the page is loaded.

The nature of the Web provides the researcher with access to the source of the documents. This is important because it presents the researcher with semi-structured data. HTML, in particular, is written in the form of elements that consist of tags enclosed in angle brackets. For example, "<h1>" denotes the beginning of an important heading and "</h1>" denotes its end, with the text in between representing what the user would see, such as a headline. Furthermore, these elements can be stylized through the use of "class" and "style" attributes, and identified through the use of an "id" attribute.

Because each of these elements must be specified, it is possible to identify elements on a page based on their syntax. Specifically, it allows the use of selectors—pieces of code that can select other pieces of code based on features like the "id" attribute of an element. The combination of different selectors allows researchers to accurately identify and extract various features of a Web document (see Sjøvaag and Stavelin 2012).

The literature thus indicates that there are different strategies that a researcher can employ for computationally freezing and analyzing features of liquid content. However, how can these strategies be effectively leveraged to enable a comprehensive analysis of the liquidity of news websites, or to assess the relationships between features of dynamic documents like homepages? Furthermore, how can they be synthesized into a process that can account for large datasets covering multiple different websites using short intervals over a long period of time?

Case Study

In order to demonstrate a process for automatically capturing and coding structural features of liquid documents, a computational analysis of the homepages of 21 news organizations was performed over two months and using 15-minute intervals. The organizations analyzed were among the 50 largest US-based news organizations and are often studied by researchers interested in US news. The unit of analysis was the individual news item, appearing on the homepage of a given news organization at a given point in time. Of particular interest to this case study were two variables: the popularity of a news item, drawn from its ranking on a list of most-viewed items, and how prominent it was, if at all, based on its relative position on the homepage. A copy of the source code used as part of this case study is available on the author's website, http://www.rodrigozamith.com.

Freezing the Homepages

The first step in the process was to acquire the data. Because the homepages of the news organizations could change at any time as news items were either manually or automatically pushed onto the page and as automated functions were executed (e.g., computer scripts updating the list of most-viewed items), a data collection strategy with 15-minute intervals was chosen. Shorter intervals are generally preferable because they allow the researcher to later discard unnecessary data and because they reduce the likely impact

of missing data if a page cannot be loaded during a point in data collection (e.g., due to a network malfunction). However, shorter intervals also increase the amount of data that must be stored and processed.

Of particular importance to the ability to computationally identify and code for the popularity and prominence of individual news items is the acquisition of the computer code that enables the browser to render a homepage—that is, to display what the user ultimately views when he or she accesses a given homepage. While Karlsson and Strömbäck (2010) point to a manual approach or to the use of different third-party tools, neither option was found to be suitable for this analysis. The first option—manually saving each homepage—required human resources that were unavailable to the researcher (e.g., an assistant to simultaneously load 21 different homepages every 15 minutes for two months and organize each saved file).

The second approach—using third-party tools like HTTrack—was found to be unsatisfactory because many of the homepages made extensive use of client-side scripting languages like JavaScript to perform actions like loading and switching between lists of most-viewed stories. This code is designed to be loaded and processed by default, but it requires a modern browser to interpret and execute it. This limits the range of existing website mirroring tools, which typically use lightweight solutions that cannot perform those automated, client-side actions. Moreover, some homepages had multiple lists of top stories (e.g., "most e-mailed") and in order to access the appropriate list (the "most viewed" list), a mouse action had to be simulated to hover over that list and thus execute the script that would retrieve the appropriate information from the server and add it to the page. In light of this, a customized solution was deemed to be necessary.

Custom Python scripts were thus developed to freeze the homepages into organized sets of static files comprised of the page's source code and a screenshot of the entire page. Specifically, each script simulated a browsing session through the use of the Selenium framework, which enables the programmatic control of popular Web browsers. A new instance of Mozilla Firefox was initiated in a virtual environment for each news organization and the respective URL for the homepage was automatically entered. Then, there was a forced two-minute delay intended to allow all of the website's elements to finish loading and for any automated and scripted actions to take place (e.g., for interstitial advertisements to disappear). The computer script then performed any additional actions necessary to load all of the necessary components of the page (e.g., clicking on the appropriate list of most-viewed items or loading that information from a linked page). The HTML source code *processed by the browser* was then saved. The resulting file was then automatically named, tagged with a UTC date and time, and organized into an appropriate subdirectory.

It is important to note that the code saved in this process differs from a copy downloaded by a mirroring program like HTTrack because it includes all of the processed client-side actions. For example, instead of having the area with the list of most-viewed items comprise of a call to a script that would need to be executed (and thus not be ready to be parsed by most algorithms), it was a simple HTML object with a list of the most popular stories that could be readily parsed. Furthermore, because these scripts often involve requesting data from the server, executing them at a later point in time would likely fail to yield the desired result (i.e., obtaining data from the desired point in time).

Additionally, in order to aid in the development and assessment of the algorithms that would code each captured snapshot, a screenshot of the entire webpage was also taken. Although Karlsson and Strömbäck (2010) are correct that screenshots typically

only capture the visible portion of a page on a screen, the Selenium framework enabled the researcher to capture the entire page as a single, full-size image in the native resolution. Although these images were not used in the analysis by the algorithms, they were essential for creating the algorithms and, later, for ensuring that they accurately coded the content.

These scripts were run automatically every 15 minutes over a two-month period (from October 18 to December 20, 2014) using a dedicated Linux server with a quad-core, 3.1 gigahertz AMD processor, 16 gigabytes of RAM, and 1.5 terabytes of hard drive space, and running only free or open-source software. A total of 126,473 snapshots were captured, with the interpreted HTML source code taking up 40.5 gigabytes and the screenshots taking up 610.5 gigabytes of hard drive space.

Coding for Popularity and Prominence

Because most large news organizations—and all news organizations in this study—use content management systems, the structure of each news organization's homepage will generally only contain minor variations that are part of an otherwise consistent design. In the present analysis, each organization had at most a handful of distinct layouts. Thus, although the content on a given website would be in flux over the course of the day, the layout (and, most importantly, the HTML elements used to instruct the browser how to render that layout) would only have minor variations. It was therefore possible to leverage this uniformity to automate the content coding process by accounting for those variations and, for each variation, seeking out common patterns in the code.

With this consideration in mind, a second set of Python scripts were created to automatically code the frozen source code files. Individual scripts were developed for every news organization. For each snapshot, the BeautifulSoup library for Python was used to transform the source code into a navigable object that allowed specific elements to be located based on their attributes (see also Sjøvaag and Stavelin 2012; Sjøvaag, Stavelin, and Moe 2015). For example, with a navigable object, a script may easily locate a "div" element with a specific "id" attribute, and then locate all of the heading elements (e.g., "h1" and "h2") that appeared within that "div" element. For an illustration, see Figure 1.

Each script was designed to first identify the layout being used in the given snapshot based on predefined sets of structural attributes. For example, if a "div" element with a "class" attribute of "big-story clearfix two" was found in the source code of a snapshot for the *Plain-Dealer*, the procedure for the first layout variation was followed. Alternatively,

```
##### Locate the 'Most Viewed' Items and Store in List
try:
    mostviewed_1to5 = [link.find("a", href=link_pattern) for link in document_soup.find("div", class_="tab-content most-viewed").find_all("li")]
    mostviewed_linklist = []
    for link in mostviewed_1to5:
        try:
            link = link.get("href")
            mostviewed_linklist = parserfunctions.linklist_actions(link, mostviewed_linklist)
        except:
            pass
except:
    message = "Failed to retrieve the list of 'most viewed' links"
    seriousness = 2
    parserfunctions.error_log_entry(cur, conn, mysql_log_name, curr_time, pubshort, homepage, seriousness, message)
```

FIGURE 1
A sample of the code used to identify the "most-viewed" list on the homepage of the *New York Times*. A selector is used to identify the first "div" element with the "class" attribute of "tab-content most-viewed" and then identify all "li" child elements that have an "a" element with an "href" attribute matching the organization's link pattern. If the selector fails to identify those elements, the error is logged in a database

if a "div" element with a "class" attribute of "big-story clearfix topic-two" was found, the procedure for the second layout variation was followed.

In order to ensure that only news items were being coded, rather than links to an author's biography or to another website, the researcher identified distinct, publication-specific patterns in the URLs that would separate news content from non-news content. For example, all of the URLs pertaining to news content on the *Plain-Dealer*'s website either contained the string "/(YYYY)/(MM)/" and concluded with ".html" or contained the string "/news/article/".

The script then gathered the information necessary to assess an item's popularity. While there are several ways to operationalize the popularity of a news item (e.g., number of times it has been tweeted about), it was operationalized here as the number of times it had been viewed based on its position on the list of most-viewed items. This operationalization was preferred because it is not only consistent with relevant literature but also because the number of times an article has been viewed has been repeatedly found to be the most salient metric in newsrooms (Anderson 2013; Groves and Brown 2011; Lee, Lewis, and Powers 2014).

The script identified the region of the page containing the list of most-viewed items and extracted all of the relevant links appearing in it in the order they appeared. For example, a selector was written for the *New York Times* snapshots to locate the "div" element with a "class" attribute of "tab-content most-viewed" and then extracted all list elements ("li") appearing within it. Each of those elements was then scanned for the presence of an "a" child element that had a "href" attribute that matched the predefined URL pattern for news items. All matching URLs were then temporarily stored as a Python list object for popular items.

The next step was to gather the information necessary to code for the prominence of the items appearing on the page. The prominence of an item refers to its relative position on the homepage, with items appearing in more noticeable spots deemed to be more prominent and those appearing in less noticeable spots deemed less prominent (Lim 2012). To assign a prominence ranking for a given region, an F-shape pattern was systematically followed (see Boczkowski, Mitchelstein, and Walter 2011; Lee, Lewis, and Powers 2014; Lim 2010). This approach privileged distinct spots where items could be placed from left to right and then top to bottom. The present approach also privileged areas of the homepage that were clearly intended to draw readers, such as those that included large pictures and larger font sizes.

The five most prominent regions were identified for each variation of the layout for the 21 news organizations (for an example, see Figure 2). Based on the detected layout, the script would thus look for each one of those five regions using predefined selectors and identify the dominant item appearing within it. Through the use of different techniques, only the headlines for the main items appearing within the designated areas of prominence were extracted. For example, with the first layout variation of the *Plain-Dealer*, a selector as written to located the "div" element with an "id" attribute of "main" and all child header elements matching "h1", "h2", or "h3" were then extracted. Each of those elements was then scanned for the presence of an "a" child element that had a "href" attribute that matched the predefined URL pattern for news items. This procedure facilitated the exclusion of any "related items," since they are typically reserved for stories of lesser import or content from previous days. All matching URLs were then temporarily stored as a Python list object for prominent items.

FIGURE 2
A screenshot of the *New York Times*' homepage on December 9, 2014 at 00:15 UTC. The most prominent area is the top-left piece due to its large font size, followed by the middle item with its large accompanying picture, and then the items on the left bar due to their comparatively larger font sizes as well

After creating list objects for the popular items and the prominent items, all of the hyperlinks that matched the URL pattern for news items were extracted from the entire page and added to a third temporary Python list object. All three list objects were then combined into a single set of unique hyperlinks. Each of those unique hyperlinks was compared against the popularity and prominence list objects and subsequently coded based on if and where it appeared within those list objects. For example, if a link appeared as the second element in the prominence list object, it was assigned a value of 2 for prominence; if it did not appear on that list, it received a value of 0.

Each unique hyperlink was then stored as a separate row in a MySQL database. Although Sjøvaag and Stavelin (2012) advocate for the use of comma-separated values (CSV) files, an ACID-compliant, relational database was preferred because of its ability to handle multiple transactions at once (i.e., have multiple pieces of content be analyzed at the same time), its superior reliability, and its ability to easily filter and access specific

groups of entries. Additionally, although alternative database paradigms like NoSQL and Hadoop were available, MySQL was deemed to offer adequate performance with proper indexing, and provided a well-tested and well-documented system.

Verifying Coded Information

Computers are able to execute a given set of instructions with perfect reliability as a result of their deterministic nature. Consequently, there is no need to assess intercoder reliability when using an algorithm to perform the coding, a function that distinguishes computational approaches to content analysis from manual ones (Zamith and Lewis 2015). However, in order to ensure that the algorithm accurately coded the content, coded data are often compared against a "gold standard," which is typically a human-coded dataset that is presumed to represent the "correct" coding decisions (Grimmer and Stewart 2013).

Because of the straightforward and mechanical nature of the adopted approach, the researcher opted to follow an iterative process that involved multiple revisions to the algorithm to ensure that all data were coded properly. Heeding Sjøvaag and Stavelin's (2012) recommendation, all scripts included functions to log failures at every step in the process—that is, to store in a separate database an entry every time the script encountered something it was not programmed to handle. This enabled the researcher to quickly identify problematic snapshots and tune the algorithm to properly deal with them. Additionally, the researcher created an electronic interface that would display the screenshot associated with a given snapshot alongside the respective coding decisions made by the algorithm (see Figure 3). This enabled the researcher to quickly review and confirm that the correct coding decisions were made by the algorithm for randomized subsets of the data.

This iterative process was preferred because it allowed the researcher to quickly identify when and where the algorithm was failing and because it reduced the amount

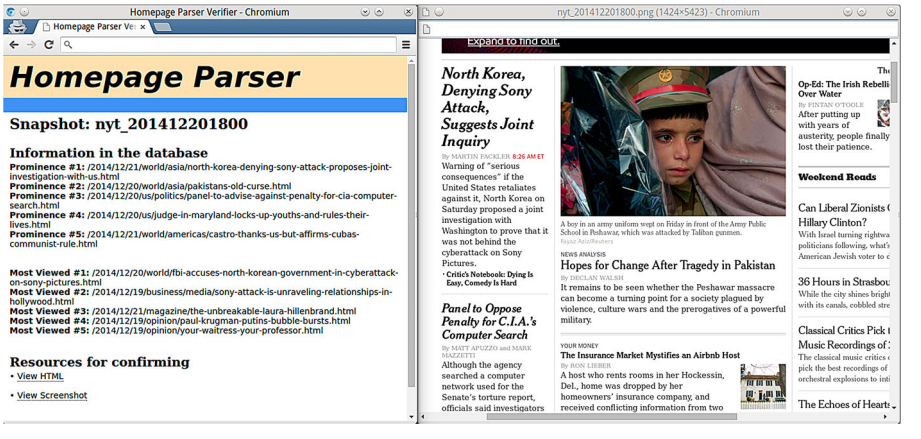

FIGURE 3
A screenshot of the researcher's interface for verifying the algorithm's coding decisions. On the left side, an electronic interface displaying the database information for the *New York Times* snapshot for December 20, 2014 at 18:00 UTC and links to the stored items. On the right side, a screenshot of that snapshot

of content that had to be manually reviewed for the researcher to have confidence in the accuracy of the algorithm. That is, if one were to review even just 10 percent of the snapshots analyzed—that would be 12,647 snapshots, each with hundreds of links—that individual would need to set aside several days if not weeks every time the algorithm was tuned. Instead, the error-logging functionality allowed the researcher to identify and diagnose problems, and the electronic interface allowed the researcher to manually verify that the correct coding decisions were being made.

After multiple revisions of the algorithm, the researcher ensured that there were no systematic errors being logged and then reviewed 50 random snapshots for each organization—1050 snapshots in all. Though this was just a fraction of the total number of snapshots, the fact that the variables analyzed were unambiguous and that the algorithm identified and coded all items as the researcher would have gave the researcher sufficient confidence to proceed with the machine-coded data.

Cleaning Data and Generating New Datasets

A total of 13,077,079 units were coded and entered into the database. Each of these records included information like the URL of the item, the publication associated with it, the timestamp of the snapshot both in UTC and adjusting for the organization's native time zone, whether it appeared in a prominent area or was popular, and, if so, the prominence and popularity ranking for that item.

These data were then cleaned of obvious noise—results of an array of different issues specific to the organizations—through the use of a last set of Python scripts. For example, in what was likely an unintentional programming error, the *Denver Post*'s website routinely included links to a small number of stories that were more than a year old. These stories were hidden from view through a styling attribute but were included in the source code. Additionally, a few links—typically, static objects that happened to fit the URL pattern for news items—were present in all of the snapshots. These entries were also removed from the database.

Some news organizations like the *St. Paul Pioneer Press* used symbolic URL paths, such that the same item would have distinct URLs, though a unique identifier was shared. For example, a story might have the string "ci_26749631" in its URL, but it would be prefaced by "/business/" in one instance and "/popular/" in another. This could be addressed computationally by reducing the URL to a string consisting only of the unique identifier.

Once the data had been cleaned, a range of new datasets could be quickly created from it. Because all data were stored in a relational database (MySQL) rather than a CSV file, the data could be easily and efficiently filtered and combined. Moreover, the database could be easily queried by Python scripts through the use of libraries like PyMySQL to create more sophisticated datasets for assessing different aspects pertaining to the liquidity of an item and of a page, such as the amount of time each news item spent in an area of prominence, for how long it was popular, and the highest and lowest rankings for an item's prominence and popularity across snapshots. These data could also be easily transformed and reshaped to fit the expectations of popular structural equation modeling software like MPlus and AMOS in order to assess complex relationships among variables. Finally, the data could be quickly extracted into smaller files for analysis by software like R, which stores data objects in the system's memory and poses challenges when the size of the dataset exceeds the amount of RAM a machine has.

Discussion

The liquid quality of online news offers researchers the ability to peer into the "black box" of journalism and assess how content changes and evolves (Deuze 2008; Karlsson 2011; Singer 2005). However, the prospect of mutable, and in some cases constantly changing, content introduces a range of methodological challenges for researchers (Lewis, Zamith, and Hermida 2013; Sjøvaag and Stavelin 2012). Indeed, the immediacy afforded by digital and networked technologies allows content producers to constantly produce new work on irregular schedules and update existing work. Furthermore, to meet consumer expectations of constantly updated content, such content is likely to become ever more liquid, especially as user-generated content becomes more popular.

Different scholars have proposed a number of strategies and techniques for addressing multiple challenges associated with "liquid" content, from how to freeze it to how to analyze it (Karlsson and Strömbäck 2010; Sjøvaag and Stavelin 2012; Widholm 2016). The present research has shown how those contributions could be effectively synthesized and built upon to develop a process for computationally capturing and analyzing large volumes of data using inexpensive, consumer-grade hardware that can be used to address important questions for mass communication research. For example, this process proved essential for an analysis that found considerable variability, across a number of US news organizations, in the rate of change of the list of most-viewed items and the median time it took for an item to appear on that list, indicating that such lists may not be readily comparable despite looking similar (Zamith 2015a). Additionally, this process was used to demonstrate that, for many news organizations, the editorial and audience agendas remain distinct, and that the impact of a story's popularity on its subsequent position in an area of prominence is smaller than is often assumed (Zamith 2015b).

While Karlsson and Strömbäck (2010) point to different ways of capturing data, from manual approaches like saving the pages with the browser to the use of third-party tools like HTTrack, the present work has shown that a superior approach is to use the Selenium framework. That framework allows modern browsers to process JavaScript code and other advanced Web features, which are becoming increasingly prevalent as the so-called "Web 2.0" proliferates and more of the interactive affordances of the technology are put to use. In particular, JavaScript is now routinely used to make asynchronous calls to the server to load and modify content within a page. These may include features that are of great interest to researchers, like the lists of most-viewed items (Boczkowski, Mitchelstein, and Walter 2011; Boczkowski and Peer 2011; Bright and Nicholls 2014), and, increasingly, core functionality like ESPN's and Facebook's automatic loading of new content when the user approaches the bottom of the page. In order to accurately freeze pages that make use of this advanced functionality and perform the actions necessary to have the document reach the state of interest for the researcher, it is necessary to adopt more robust solutions.

Additionally, adopting a technology like Selenium allows researchers to capture full-page screenshots of the content exactly as it appears on the browser screen. This effectively bypasses the limitations aptly noted by Karlsson and Strömbäck (2010) of traditional screenshots with the "print screen" and "screen grab" functions of popular operating systems (and the software that automate those functions) and offers a well-tested, cross-platform alternative to Widholm's (2016) method. Such screenshots can prove to be invaluable to the development and verification of algorithms for coding aesthetic features of documents, and can even serve as the content that human coders could then analyze.

Indeed, one could easily envision their use as complementary documents in a hybrid form of content analysis (see Lewis, Zamith, and Hermida 2013; Zamith and Lewis 2015).

However, although the Selenium framework offers several advantages, it must be noted that it brings with it considerable computational costs. Specifically, it requires a complete browser like Mozilla's Firefox or Google's Chrome to be loaded, which consumes far more RAM and CPU cycles than a simple document-mirroring tool like HTTrack. Although the hardware used in this case study is readily available to any consumer for a moderate cost, simultaneously loading the homepages of the 21 news organizations consumed nearly all of the available CPU and RAM during the brief periods of access—even after shifting the browsing sessions to a lighter virtualized environment. This computational cost thus impairs the scalability of this approach, drawing attention to the need to develop lightweight tools that can handle the advanced features found on modern websites.

The present research lends support to Sjøvaag, Moe, and Stavelin's (2012; Sjøvaag, Stavelin, and Moe 2015) advocacy for the use of selectors to locate and extract content from HTML pages. Python and the BeautifulSoup library were effectively used to perform syntactical analyses in order to identify regions of significance and select the items of interest. Moreover, the uniform nature of the content management systems used by the news organizations meant that only a handful of variations of a website's layout had to be accounted for, for each organization. This means that researchers only need to write a relatively small amount of computer code to accurately analyze large amounts of data. Moreover, the coupling of Python and BeautifulSoup was fairly efficient: over 13 million units of data could be extracted and content analyzed using consumer-grade hardware in less than a day. This offers promise to researchers interested in computationally assessing the liquidity of different content, and allows for the analysis of larger amounts of data with greater precision.

That the coding decisions made by the algorithm were not compared against an independent "gold standard" is a limitation of this research. Indeed, in an ideal world, an amount comparable to the conventions of intercoder reliability under a manual framework for content analysis would yield greater confidence in the accuracy of the algorithm. However, given that the analysis relied on the analysis of recurring syntax—HTML code generated by a content management system that had to be precise in order to render correctly—the efficiency gained through the procedure utilized greatly outweighed the potential loss of accuracy. Specifically, the error-logging mechanisms employed allowed the researcher to quickly identify the different variations of the layouts and, over time, minimize the number of instances in which the algorithms encountered a snapshot they were not programmed to handle. In order to ensure that the instructions were not too broad—resulting in miscoding items—the electronic interface made the confirmation of the algorithmic coding decisions expedient. The positive results from this process offer a pathway for researchers with limited resources.

As this analysis indicated, researchers must be careful to evaluate their data in different ways after the content has been collected and/or coded in order to identify anomalies that may point to broader issues with the source of the data. For example, the use of symbolic URL paths by the *St. Paul Pioneer Press* meant that a single news item might have been inappropriately treated as two separate news items in an analysis. Rather than recoding all of the items, a simple computer script could be used to automatically clean up that potential issue for all relevant items that had already been coded. This, among other examples,

indicates why in many instances it is preferable to store data directly in a formal database like MySQL rather than CSV files (cf. Sjøvaag and Stavelin 2012).

In conclusion, computational techniques can be effectively used to freeze and analyze liquid content like the homepages of news organizations. While the preferable solutions may require some custom programming, there are several free and open-source programs, libraries, and frameworks that can facilitate the creation of powerful scripts. In particular, the process discussed here, which combined Python, Selenium, Firefox, and BeautifulSoup with error-logging and computer-aided verification, enables rigorous analyses of a large amount of liquid content to be performed on consumer-grade hardware and with limited human resources. This, in turn, enables researchers to engage in computational social scientific inquiry, and in particular assess the evolution of journalistic content and relationships between key variables both in short intervals and over long periods of time.

DISCLOSURE STATEMENT

No potential conflict of interest was reported by the author.

REFERENCES

Anderson, C. W. 2013. *Rebuilding the News: Metropolitan Journalism in the Digital Age*. Philadelphia, PA: Temple University Press.

Benton, Joshua. 2015. "Gawker, Ever Restless in Restructuring Its Infrastructure, Is Stepping Back from the Stream." *Nieman Lab*, January 12. http://www.niemanlab.org/2015/01/gawker-ever-restless-in-restructuring-its-infrastructure-is-stepping-back-from-the-stream/.

Boczkowski, Pablo J. 2004. "The Processes of Adopting Multimedia and Interactivity in Three Online Newsrooms." *Journal of Communication* 54 (2): 197–213. doi:10.1111/j.1460-2466.2004.tb02624.x.

Boczkowski, Pablo J., Eugenia Mitchelstein, and Martin Walter. 2011. "Convergence across Divergence: Understanding the Gap in the Online News Choices of Journalists and Consumers in Western Europe and Latin America." *Communication Research* 38 (3): 376–396. doi:10.1177/0093650210384989.

Boczkowski, Pablo J., and Limor Peer. 2011. "The Choice Gap: The Divergent Online News Preferences of Journalists and Consumers." *Journal of Communication* 61 (5): 857–876. doi:10.1111/j.1460-2466.2011.01582.x.

Bright, Jonathan, and Tom Nicholls. 2014. "The Life and Death of Political News Measuring the Impact of the Audience Agenda Using Online Data." *Social Science Computer Review* 32 (2): 170–181. doi:10.1177/0894439313506845.

Crawford, Kate, Mary L. Gray, and Kate Miltner. 2014. "Critiquing Big Data: Politics, Ethics, Epistemology." *International Journal of Communication* 8 (June): 1663–1672.

Deuze, Mark. 2008. "The Changing Context of News Work: Liquid Journalism for a Monitorial Citizenry." *International Journal of Communication* 2 (July). http://ijoc.org/index.php/ijoc/article/view/290.

García Avilés, José Alberto, Bienvenido León, Karen Sanders, and Jackie Harrison. 2004. "Journalists at Digital Television Newsrooms in Britain and Spain: Workflow and Multi-skilling

in a Competitive Environment." *Journalism Studies* 5 (1): 87–100. doi:10.1080/1461670032000174765.

Grimmer, Justin, and Brandon M. Stewart. 2013. "Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts." *Political Analysis* 21 (3): 267–297. doi:10.1093/pan/mps028.

Groves, Jonathan, and Carrie Lisa Brown. 2011. "Stopping the Presses: A Longitudinal Case Study of the Christian Science Monitor Transition from Print Daily to Web Always." *#ISOJ* 1 (2): 95–134.

Hermida, Alfred, Seth C. Lewis, and Rodrigo Zamith. 2014. "Sourcing the Arab Spring: A Case Study of Andy Carvin's Sources on Twitter during the Tunisian and Egyptian Revolutions." *Journal of Computer-Mediated Communication* 19 (3): 479–499. doi:10.1111/jcc4.12074.

Karlsson, Michael. 2011. "The Immediacy of Online News, the Visibility of Journalistic Processes and a Restructuring of Journalistic Authority." *Journalism* 12 (3): 279–295. doi:10.1177/1464884910388223.

Karlsson, Michael. 2012. "Analysis of Online News." *International Communication Gazette* 74 (4): 385–402. doi:10.1177/1748048512439823.

Karlsson, Michael, and Helle Sjøvaag. 2016. "Content Analysis and Online News." *Digital Journalism* 4 (1): 177–192. doi:10.1080/21670811.2015.1096619.

Karlsson, Michael, and Jesper Strömbäck. 2010. "Freezing the Flow of Online News: Exploring Approaches to the Study of the Liquidity of Online News." *Journalism Studies* 11 (1): 2–19. doi:10.1080/14616700903119784.

Lee, Angela M., Seth C. Lewis, and Matthew Powers. 2014. "Audience Clicks and News Placement: A Study of Time-Lagged Influence in Online Journalism." *Communication Research* 41 (4): 505–530. doi:10.1177/0093650212467031.

Lewis, Seth C., Rodrigo Zamith, and Alfred Hermida. 2013. "Content Analysis in an Era of Big Data: A Hybrid Approach to Computational and Manual Methods." *Journal of Broadcasting & Electronic Media* 57 (1): 34–52. doi:10.1080/08838151.2012.761702.

Lim, Jeongsub. 2010. "Convergence of Attention and Prominence Dimensions of Salience among Major Online Newspapers." *Journal of Computer-Mediated Communication* 15 (2): 293–313. doi:10.1111/j.1083-6101.2010.01521.x.

Lim, Jeongsub. 2012. "The Mythological Status of the Immediacy of the Most Important Online News." *Journalism Studies* 13 (1): 71–89. doi:10.1080/1461670X.2011.605596.

Lowrey, Wilson. 2006. "Mapping the Journalism–blogging Relationship." *Journalism* 7 (4): 477–500. doi:10.1177/1464884906068363.

Mahrt, Merja, and Michael Scharkow. 2013. "The Value of Big Data in Digital Media Research." *Journal of Broadcasting & Electronic Media* 57 (1): 20–33. doi:10.1080/08838151.2012.761700.

Matheson, Donald. 2004. "Weblogs and the Epistemology of the News: Some Trends in Online Journalism." *New Media & Society* 6 (4): 443–468. doi:10.1177/146144804044329.

Shah, Dhavan V., Joseph N. Cappella, and W. Russell Neuman. 2015. "Big Data, Digital Media, and Computational Social Science Possibilities and Perils." *The ANNALS of the American Academy of Political and Social Science* 659 (1): 6–13. doi:10.1177/0002716215572084.

Singer, Jane B. 2005. "The Political J-Blogger: 'Normalizing' a New Media Form to Fit Old Norms and Practices." *Journalism* 6 (2): 173–198. doi:10.1177/1464884905051009.

Sjøvaag, Helle, Hallvard Moe, and Eirik Stavelin. 2012. "Public Service News on the Web: A Large-Scale Content Analysis of the Norwegian Broadcasting Corporation's Online News." *Journalism Studies* 13 (1): 90–106. doi:10.1080/1461670X.2011.578940.

Sjøvaag, Helle, and Eirik Stavelin. 2012. "Web Media and the Quantitative Content Analysis: Methodological Challenges in Measuring Online News Content." *Convergence: The International Journal of Research into New Media Technologies* 18 (2): 215–229. doi:10.1177/1354856511429641.

Sjøvaag, Helle, Eirik Stavelin, and Hallvard Moe. 2015. "Continuity and Change in Public Service News Online." *Journalism Studies*: 1–19. doi:10.1080/1461670X.2015.1022204.

Sormanen, Niina, Jukka Rohila, Epp Lauk, Turo Uskali, Jukka Jouhki, and Maija Penttinen. 2016. "Chances and Challenges of Computational Data Gathering and Analysis." *Digital Journalism* 4 (1): 55–74. doi:10.1080/21670811.2015.1096614.

Steuer, Jonathan. 1992. "Defining Virtual Reality: Dimensions Determining Telepresence." *Journal of Communication* 42 (4): 73–93. doi:10.1111/j.1460-2466.1992.tb00812.x.

Tumber, Howard. 2001. "Democracy in the Information Age: The Role of the Fourth Estate in Cyberspace." *Information, Communication & Society* 4 (1): 95–112. doi:10.1080/13691180122542.

Widholm, Andreas. 2016. "Tracing Online News in Motion." *Digital Journalism* 4 (1): 24–40. doi:10.1080/21670811.2015.1096611.

Zamith, Rodrigo. 2015a. "Deciphering 'Most Viewed' Lists: An Analysis of the Comparability of the Lists of Popular Items." Paper presented to the Communication Theory and Methodology Division of AEJMC, San Francisco, CA.

Zamith, Rodrigo. 2015b. "On Click-Driven Homepages: An Analysis of the Effect of Popularity on the Prominence of News." Paper presented to the Newspaper and Online News Division of AEJMC, San Francisco, CA.

Zamith, Rodrigo, and Seth C. Lewis. 2015. "Content Analysis and the Algorithmic Coder: What Computational Social Science Means for Traditional Modes of Media Analysis." *The ANNALS of the American Academy of Political and Social Science* 659 (1): 307–318. doi:10.1177/0002716215570576.

Rodrigo Zamith, Journalism Department, University of Massachusetts, USA.
E-mail: rzamith@umass.edu. Web: http://www.rodrigozamith.com