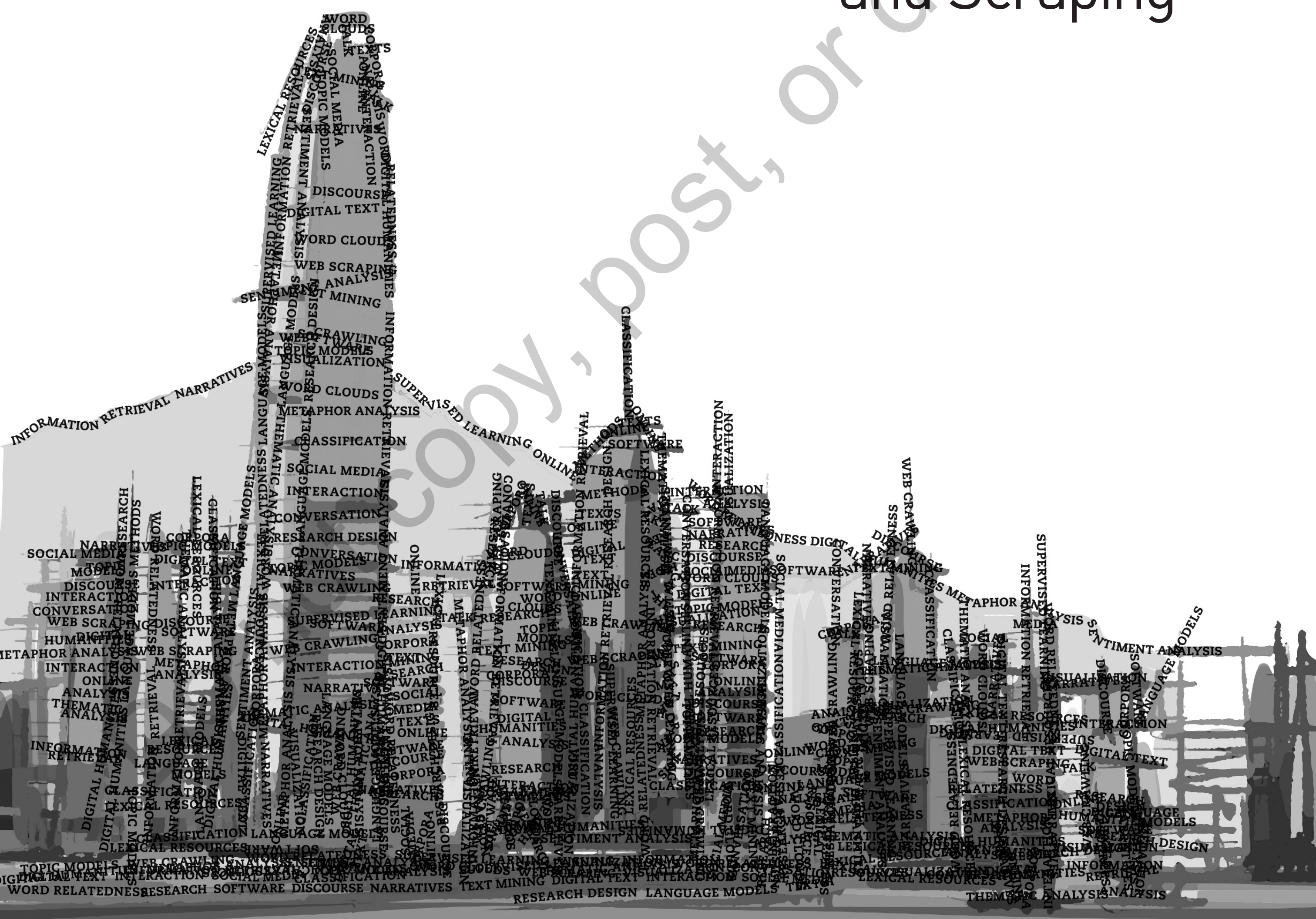
3

Web Crawling and Scraping



Learning Objectives

The goals of Chapter 3 are to help readers do the following:

1. Understand the basic organization of the web and learn about estimates of its size.
2. Learn about the main techniques for web crawling and scraping.
3. Learn about available software packages for automatically collecting textual data from webpages.

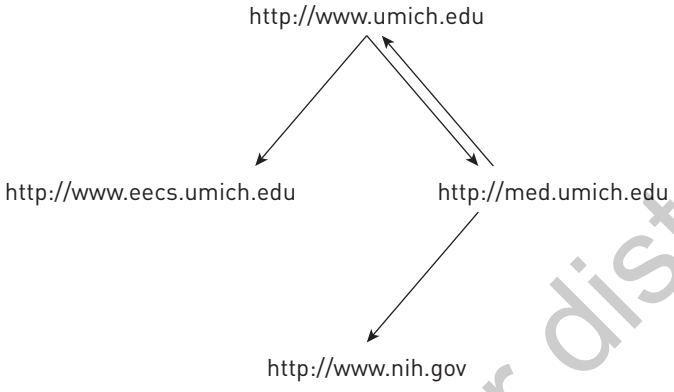
The web—a common abbreviation for the World Wide Web—consists of billions of interlinked hypertext pages. These pages contain text, images, videos, or sounds and are usually viewed using web browsers such as Firefox or Internet Explorer. Users can navigate the web either by directly typing the address of a webpage (the URL) inside a browser or by following the links that connect webpages between them.

In this chapter, we review Internet-based methods for crawling and scraping document collections for social science research. While these two terms are often used interchangeably, we use *crawling* to refer to the process of automatically identifying (via link navigation) the webpages that should be included in a collection and *scraping* to refer to the process of extracting the text from a collection of webpages.

The web can be visualized as a typical example of a graph, with webpages corresponding to vertices in the graph and links between pages corresponding to directed edges. For instance, if the page http://www.umich.edu includes a link to the page http://www.eecs.umich.edu and one to the page http://med.umich.edu, and the later page in turn links to the page of the National Institutes of Health (http://www.nih.gov) and also back to the http://www.umich.edu page, it means that these four pages form a subgraph of four vertices with four edges, as is illustrated in Figure 3.1.

In addition to "traditional" webpages, which account for a large fraction of the data that we currently find online, today's web also includes a number of other data sources, such as sites with user-contributed content (e.g., Wikipedia, Huffington Post), social media sites (e.g., Twitter, Facebook, Blogger), deep web data (e.g., data stored in online databases such as data.gov), or e-mail (e.g., Gmail, Outlook). While

FIGURE 3.1 ● Sample Web Graph



some of these sources may not be publicly available (for instance, e-mail is by definition private and so are a number of Facebook profiles), they still represent data in digital format that accounts for online traffic.

There are many challenges that come with the use of web data—many of which are highlighted in Chapter 13 (see the Web-Based Information Retrieval section). In addition, there are also challenges that are associated with crawling such data, which we address in this chapter.

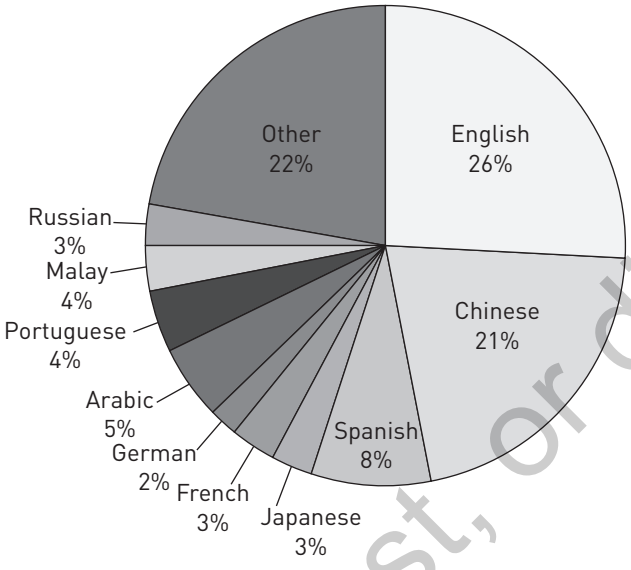
Web Statistics

While the size of the web is generally considered to be unknown, there are various estimates concerning the size of the indexed web—that is, the subset of the web that is covered by search engines. Web statistics compiled in 2014 by http://www.geekwire.com suggested 5 million terabytes of data online, out of which approximately 20% is textual data.

The web is estimated to include more than 600 million web servers and 2.4 billion web users, which includes about 1 billion Facebook users and 200 million Twitter users (http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers). Estimates of the number of e-mails come from http://www.radicati.com, which suggests that 154 billion e-mails are sent daily, of which more than 60% are spam.

An interesting statistic refers to the proportion of languages used on the web. Information collected by http://www.internetworldstats.com in 2015 showed the distribution of language use as illustrated in Figure 3.2.

FIGURE 3.2 ● Top Ten Languages on the Web in 2015



Source: http://www.internetworldstats.com/stats7.htm

Web Crawling

Web crawling is the process of building a collection of webpages by starting with an initial set of URLs (or links) and recursively traversing the corresponding pages to find additional links. A collection built this way can be used, for instance, to create an index for a search engine (see Chapter 13) and to perform information extraction (IE) and text mining (Chapter 12), text classification (Chapter 11), or any other process that requires textual data.

Processing Steps in Crawling

A crawler typically performs the following steps.

1. The crawler creates and maintains a list of URLs to be processed. This list is initially seeded with some manually selected URLs, and it is then iteratively grown into a large set of URLs.
2. The crawler selects a URL from the list (see below for selection strategies), marks it as "crawled," and it fetches the webpage from that URL. The page is processed, and links and content are extracted. This processing can be as simple as just extracting links using a regular expression that matches all the occurrences of

tags such as ←a href="http://...."→, followed by removal of all the HTML tags to obtain the content of the page. At times, a more sophisticated processing may be required, for instance when the links also include relative links or links to fragments or when the content of a page includes entire sections devoted to advertisements or other content that needs to be removed.

3. If the content has already been seen, it is discarded. If not, it is added to the collection of webpages to be further processed (e.g., indexed, classified, etc.)
4. For each URL in the new set of URLs identified on the page, a verification is made to ensure that the URL has not been seen before, that the page exists, and that it can be crawled. If all these filters are passed, the URL is added to the list of URLs at step 1, and the crawler goes back to step 2.

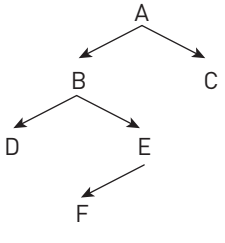
Traversal Strategies

An important aspect of any crawler is its web traversal strategies. As mentioned before, the web is a graph, and therefore, different graph traversal algorithms can be applied. One way of traversing the web is called breadth-first, where, given one webpage, we first collect and process all the pages that can be reached from URLs on that page before we move on to other pages. The second way of traversing the web is called depth-first, where, given one webpage, we extract one URL from that page, collect and process the page that can be reached from that one URL, extract again one URL on the page we just processed, and so on until we reach a "dead end." Only then do we backtrack and process additional URLs on the pages we have just visited. For instance, Figure 3.3 shows a simple web graph, along with the order of page traversal, starting with page A, for each of these two strategies.

Crawler Politeness

Most websites have a clear crawling policy that states which crawlers can or cannot traverse them and which parts of the site can be crawled. There are two main ways of indicating a crawling policy. The most commonly used one is robots.txt, which is a

FIGURE 3.3 ● Web Traversal Strategies



Breadth-first traversal order: A, B, C, D, E, F
Depth-first traversal order: A, B, D, E, F, C

file that is placed at the root of the website (e.g., http://www.cnn.com/robots.txt). This file can include a list of crawlers (or agents) that are disallowed for specific portions of the site. For instance, the following content of robots.txt indicates that all the crawlers are disallowed from traversing pages found under /tmp or /cgi-bin, and BadBot is disallowed from the entire site:

```
User-agent: *
Disallow: /tmp/
Disallow: /cgi-bin/
User-agent: BadBot
Disallow: /
```

Another way of providing a crawling policy is through meta tags included in the HTML of individual pages. There is a special meta tag called "robots," which can be used with combinations of values for two aspects: index or noindex (allow or disallow this webpage to be crawled), and follow or nofollow (allow or disallow the crawler to follow links on this webpage). For instance, a webpage could have a robots meta tag as follows:

```
←meta name="robots" content="index,nofollow"→
```

It states that this page can be crawled, but links on the page cannot be followed.

Writing a crawler requires some basic programming knowledge. Starting with a set of input URLs, the program will perform the four steps described previously to grow the collection of URLs. Alternatively, one can also use existing commands—for instance, the Linux command wget, which allows for recursive crawling with a prespecified depth of the crawl.

Web Scraping

Web scraping is used to extract text from webpages, be they pages that have been identified through crawling (e.g., all the pages found under a given website) or pages in digital archives, including news archives such as LexisNexis (http://www.lexisnexis.com) and Access World News (http://www.newsbank.com/libraries/schools/solutions/us-international/access-world-news) as well as digital archives of historical documents (Jockers & Mimno, 2013).

Web scraping software is designed to recognize different types of content within a website and to acquire and store only the types of content specified by the user. For instance, web scraping software allows a user to search a newspaper website and save only the names of article authors or to search a real estate website and save only the prices, addresses, or descriptions of listed properties.

Scraping involves using commercial software or programming languages such as Python, Java, or Perl to write programs for scraping "from scratch" or to write programs that make use of existing application program interfaces (APIs). The goal of web scraping is to identify the "useful" text within webpages. While text comes in many forms, the "usefulness" of the text to be gathered via scraping is defined by the needs of a project. For instance, if the goal is to build a collection of news stories, one would want to identify the text of the news articles included in the webpages found by a crawler on websites such as www.cnn.com or www.nbcnews.com, and at the same time, it will want to identify and ignore the text of the advertisements found on the same webpages.

Scraping from scratch (i.e., by using a programming language) involves a process of reversed engineering of the webpage structure, along with the use of regular expressions. Starting with one (or few) of the target webpages, the programmer of the web scraper will first look closely at the source of the webpage, using the Page Source menu option found under regular web browsers such as Firefox or Internet Explorer. The inspection of the page source will allow the programmer to identify patterns that can be used to locate the useful text. For instance, she could find that the text of the news articles of interest is surrounded by tags such as <article> . . . </article> or that the title of an article is marked by <h2> . . . </h2> tags that immediately follow a </date> tag. These patterns can then be used to write regular expressions in the programming language of choice which, when applied on a target webpage, will result in the desired text.

There are also websites that provide APIs, which also assume a minimum amount of programming knowledge but do not require the reverse engineering process described previously. The APIs define the structure of the data that can be extracted (or scraped) from a site and provide an easy interface to access this data. The use of APIs requires the programmer to first read the documentation of the API to understand what data is accessible via the API and how the API can be used. For instance, the following are two examples of APIs: the Twitter API https://dev.twitter.com/overview/api to access tweets based through search or as random subsets of the existing tweets; the Blogger API, https://developers.google.com/blogger/?hl=en to access the profile of the bloggers and the content of the blogs available on Google Blogger. It is important to note that many of these APIs have usage restrictions (e.g., the Blogger API requires an API key and restricts the access to a certain number of requests per day).

Commercial web scraping software is also available, and it is reasonably easy to use for nonprogrammers. The software works by running "scripts" written by the user. The scripts tell the software on which webpage to start, what kind of text to look for (e.g., text with a certain font size or formatting), what to do with the text that is found, where to navigate next once text is saved, and how many times to repeat the script. Saved text data can be downloaded in a convenient file form such as raw text, or a CSV (comma-separated values) file.

There are several useful freeware and commercial software products available on the market, and instructional videos for most of these are easy to find on YouTube

and other Internet video services or in the manual pages available on Linux via the *man* command. In our own research, we have used Lynx, which is a very simple Linux-based command-line browser that allows for automatic processing of a page, including link and content extraction. We have also used Helium Scraper (http://www.heliumscraper.com).

Software for Web Crawling and Scraping

Web crawling can be effectively performed using publicly available implementations such as the following:

- Scrapy (http://scrapy.org), a collection of Python scripts
- LWP (http://search.cpan.org/dist/libwww-perl/lib/LWP.pm), a Perl library

The following packages can be used for scraping:

- Helium Scraper (http://www.heliumscraper.com)
- Outwit (http://www.outwit.com)
- Outwit Hub (https://www.outwit.com/products/hub)
- FMiner (http://www.fminer.com)
- Mozenda (https://www.mozenda.com)
- Visual Web Ripper (http://www.visualwebripper.com)

For data collection from social media, it is typical to use the publicly available APIs made available by the social media platforms, such as the following:

- Twitter API (https://dev.twitter.com/overview/api), which can be used to search Twitter data or access their public stream of recent tweets
- Google Blogger API (https://developers.google.com/blogger/?hl=en), which provides access to the blogs and blogger profiles published on Blogger (https://www.blogger.com)