Hands-on with the Distant Reader: A Workbook

This workbook outlines sets of hands-on exercises surrounding a computer system called the Distant Reader -- https://distantreader.org.

By going through the workbook, you will become familiar with the problems the Distant Reader is designed to address, how to submit content to the Reader, how to download the results (affectionately called "study carrels"), and how to interpret them. The bulk of the workbook is about the later. Interpretation can be as simple as reading a narrative report in your Web browser, as complex as doing machine learning, and everything else in-between.

You will need to bring very little to the workbook in order to get very much out. At the very least, you will need a computer with a Web browser and an Internet connection. A text editor such as Notepad++ for Windows or BBEdit for Macintosh will come in very handy, but a word processor of any type will do in a pinch. You will want some sort of spreadsheet application for reading tabular data, and Microsoft Excel or Macintosh Numbers will both work quite well. All the other applications used in the workbook are freely available for downloading and cross-platform in nature. You may need to install a Java virtual machine in order to use some of them, but Java is probably already installed on your computer.

I hope you enjoy using the Distant Reader. It helps me use and understand large volumes of text quickly and easily.

Eric Lease Morgan <emorgan@nd.edu>
Notre Dame (Indiana)

January 25, 2020

What is the Distant Reader, and why should I care?

The Distant Reader is a tool for reading.

The Distant Reader takes an arbitrary amount of unstructured data (text) as input, and it outputs sets of structured data for analysis — reading. Given a corpus of any size, the Distant Reader will analyze the corpus, and it will output a myriad of reports enabling you to use & understand the corpus. The Distant Reader is intended to supplement the traditional reading process.

The Distant Reader empowers one to use & understand large amounts of textual information both quickly & easily. For example, the Distant Reader can consume the entire issue of a scholarly journal, the complete works of a given author, or the content found at the other end of an arbitrarily long list of URLs. Thus, the Distant Reader is akin to a book's table-of-contents or back-of-the-book index but at scale. It simplifies the process of identifying trends & anomalies in a corpus, and then it enables a person to further investigate those trends & anomalies.

The Distant Reader is designed to "read" everything from a single item to a corpus of thousand's of items. It is intended for the undergraduate student who wants to read the whole of their course work in a given class, the graduate student who needs to read hundreds (thousands) of items for their thesis or dissertation, the scientist who wants to review the literature, or the humanist who wants to characterize a genre.

How it works

The Distant Reader takes five different forms of input (described in the next section). Once the input is provided, the Distant Reader creates a cache — a collection of all the desired content. This is done via the input or by crawling the 'Net. Once the cache is collected, each & every document is transformed into plain text, and along the way basic bibliographic information is extracted. The next step is analysis against the plain text. This includes rudimentary counts & tabulations of ngrams, the computation of readability scores & keywords, basic topic modeling, parts-of-speech & named entity extraction, summarization, and the creation of a semantic index. All of these analyses are manifested as tab-delimited files and distilled into a single relational database file. After the analysis is complete, two reports are generated: 1) a simple plain text file which is very tabular, and 2) a set of HTML files

which are more narrative and graphical. Finally, everything that has been accumulated & generated is compressed into a single zip file for downloading. This zip file is affectionately called a “study carrel“. It is completely self-contained and includes all of the data necessary for more in-depth analysis.

What it does

The Distant Reader supplements the traditional reading process. It does this in the way of traditional reading apparatus (tables of content, back-of-book indexes, page numbers, etc), but it does it more specifically and at scale.

Put another way, the Distant Reader can answer a myriad of questions about individual items or the corpus as a whole. Such questions are not readily apparent through traditional reading. Examples include but are not limited to:

* How big is the corpus, and how does its size compare to other corpora?
* How difficult (scholarly) is the corpus?
* What words or phrases are used frequently and infrequently?
* What statistically significant words characterize the corpus?
* Are there latent themes in the corpus, and if so, then what are they and how do they change over both time and place?
* How do any latent themes compare to basic characteristics of each item in the corpus (author, genre, date, type, location, etc.)?
* What is discussed in the corpus (nouns)?
* What actions take place in the corpus (verbs)?
* How are those things and actions described (adjectives and adverbs)?
* What is the tone or “sentiment” of the corpus?
* How are the things represented by nouns, verbs, and adjective related?
* Who is mentioned in the corpus, how frequently, and where?
* What places are mentioned in the corpus, how frequently, and where?

People who use the Distant Reader look at the reports it generates, and they often say, “That’s interesting!” This is because it highlights characteristics of the corpus which are not readily apparent. If you were asked what a particular corpus was about or what are the names of people mentioned in the corpus, then you might answer with a couple of sentences or a few names, but with the Distant Reader you would be able to be more thorough with your answer.

The questions outlined above are not necessarily apropos to every student, researcher, or scholar, but the answers to many of these questions will lead to other, more specific questions. Many of those questions can be answered directly or indirectly through further analysis of the structured data provided in the study carrel. For example, each & every feature of each & every sentence of each & every item in the corpus has been saved in a relational database file. By querying the database, the student can extract every sentence with a given word or matching a given grammer to answer a question such as “How was the king described before & after the civil war?” or “How did this paper’s influence change over time?”

A lot of natural language processing requires pre-processing, and the Distant Reader does this work automatically. For example, collections need to be created, and they need to be transformed into plain text. The text will then be evaluated in terms of parts-of-speech and named-entities. Analysis is then done on the results. This analysis may be as simple as the use of concordance or as complex as the application of machine learning. The Distant Reader “primes the pump” for this sort of work because all the raw data is already in the study carrel. The Distant Reader is not intended to be used alone. It is intended to be used in conjunction with other tools, everything from a plain text editor, to a spreadsheet, to database, to topic modelers, to classifiers, to visualization tools.

I don’t know about you, but now-a-days I can find plenty of scholarly & authoritative content. My problem is not one of discovery but instead one of comprehension. How do I make sense of all

the content I find? The Distant Reader is intended to address this question by making observations against a corpus and providing tools for interpreting the results.

Five different types of input

The Distant Reader can take five different types of input, and this section describes in detail what they are:

1. a file
2. a URL
3. a list of URLs
4. a zip file
5. a zip file with a companion CSV file

Each of these different types of input are elaborated upon below.

A file

The simplest form of input is a single file from your computer. This can be just about file available to you, but to make sense, the file needs to contain textual data. Thus, the file can be a Word document, a PDF file, an Excel spreadsheet, an HTML file, a plain text file, etc. A file in the form of an image will not work because it contains zero text. Also, not all PDF files are created equal. Some PDF files are only facsimiles of their originals. Such PDF files are merely sets of images concatenated together. In order for PDF files to be used as input, the PDF files need to have been "born digitally" or they need to have had optical character recognition previously applied against them. Most PDF files are born digitally nor do they suffer from being facsimiles.

A good set of use-cases for single file input is the whole of a book, a long report, or maybe a journal article. Submitting a single file to the Distant Reader is quick & easy, but the Reader is designed for analyzing larger rather than small corpora. Thus, supplying a single journal article to the Reader doesn't make much sense; the use of the traditional reading process probably makes more sense for a single journal article.

A URL

The Distant Reader can take a single URL as input. Given a URL, the Reader will turn into a rudimentary Internet spider and build a corpus. More specifically, given a URL, the Reader will:

1. retrieve & cache the content found at the other end of the URL
2. extract any URLs it finds in the content
3. retrieve & cache the content from these additional URLs
4. stop building the corpus but continue with its analysis

In short, given a URL, the Reader will cache the URL's content, crawl the URL one level deep, cache the result, and stop caching.

Like the single file approach, submitting a URL to the Distant Reader is quick & easy, but there are a number of caveats. First of all, the Reader does not come with very many permissions, and just because you are authorized to read the content at the other end of a URL does not mean the Reader has the same authorization. A lot of content on the Web resides behind paywalls and firewalls. The Reader can only cache 100% freely accessible content.

"Landing pages" and "splash pages" represent additional caveats. Many of the URLs passed around the 'Net do not point to the content itself, but instead they point to ill-structured pages describing the content — metadata pages. Such pages may include things like authors, titles, and dates, but these things are not presented in a consistent nor computer-readable fashion; they are laid out with aesthetics or graphic design in mind. These pages do contain pointers to the content you want to read, but the content may be two or three more clicks away. Be wary of

URLs pointing to landing pages or splash pages.

Another caveat to this approach is the existence of extraneous input due to navigation. Many Web pages include links for navigating around the site. They also include links to things like “contact us” and “about this site”. Again, the Reader is sort of stupid. If found, the Reader will crawl such links and include their content in the resulting corpus.

Despite these drawbacks there are number of excellent use-cases for single URL input. One of the best is Wikipedia articles. Feed the Reader a URL pointing to a Wikipedia article. The Reader will cache the article itself, and then extract all the URLs the article uses as citations. The Reader will then cache the content of the citations, and then stop caching.

Similarly, a URL pointing to an open access journal article will function just like the Wikipedia article, and this will be even more fruitful if the citations are in the form of freely accessible URLs. Better yet, consider pointing the Reader to the root of an open access journal issue. If the site is not overly full of navigation links, and if the URLs to the content itself are not buried, then the whole of the issue will be harvested and analyzed.

Another good use-case is the home page of some sort of institution or organization. Want to know about Apple Computer, the White House, a conference, or a particular department of a university? Feed the root URL of any of these things to the Reader, and you will learn something. At the very least, you will learn how the organization prioritizes its public face. If things are more transparent than not, then you might be able to glean the names and addresses of the people in the organization, the public policies of the organization, or the breadth & depth of the organization.

Yet another excellent use-case includes blogs. Blogs often contain content at their root. Navigations links abound, but more often than not the navigation links point to more content. If the blog is well-designed, then the Reader may be able to create a corpus from the whole thing, and you can “read” it in one go.

A list of URLs

The third type of input is a list of URLs. The list is expected to be manifested as a plain text file, and each line in the file is a URL. Use whatever application you desire to build the list, but save the result as a .txt file, and you will probably have a plain text file.

Caveats? Like the single URL approach, the list of URLs must point to freely available content, and pointing to landing pages or splash pages is probably to be avoided. Unlike the single URL approach, the URLs in the list will not be used as starting points for Web crawling. Thus, if the list contains ten items, then ten items will be cached for analysis.

Another caveat is the actual process of creating the list; I have learned that is actually quite difficult to create lists of URLs. Copying & pasting gets old quickly. Navigating a site and right-clicking on URLs is tedious. While search engines & indexes often provide some sort of output in list format, the lists are poorly structured and not readily amenable to URL extraction. On the other hand, there are more than a few URL extraction tools. I use a Google Chrome extension called Link Grabber. [1] Install Link Grabber. Use Chrome to visit a site. Click the Link Grabber button, and all the links in the document will be revealed. Copy the links and paste them into a document. Repeat until you get tired. Sort and peruse the list of links. Remove the ones you don’t want. Save the result as a plain text file.‡ Feed the result to the Reader.

Despite these caveats, the list of URLs approach is enormously scalable; the list of URLs approach is the most scalable input option. Given a list of five or six items, the Reader will do quite well, but the Reader will operate just as well if the list contains dozens, hundreds, or even thousands of URLs. Imagine reading the complete works of your favorite author or the complete run of an electronic journal. Such is more than possible with the Distant Reader.

A zip file

The Distant Reader can take a zip file as input. Create a folder/directory on your computer. Copy just about any file into the folder/directory. Compress the file into a .zip file. Submit the result to the Reader.

Like the other approaches, there are a few caveats. First of all, the Reader is not able to accept .zip files whose size is greater than 64 megabytes. While we do it all the time, the World Wide Web was not really designed to push around files of any great size, and 64 megabytes is/was considered plenty. Besides, you will be surprised how many files can fit in a 64

megabyte file.

Second, the computer gods never intended file names to contain things other than simple Romanesque letters and a few rudimentary characters. Now-a-days our file names contain spaces, quote marks, apostrophes, question marks, back slashes, forward slashes, colons, commas, etc. Moreover, file names might be 64 characters long or longer! While every effort as been made to accomodate file names with such characters, your milage may vary. Instead, consider using file names which are shorter, simpler, and have some sort of structure. An example might be first word of author's last name, first meaningful word of title, year (optional), and extension. Herman Melville's Moby Dick might thus be named melville-moby.txt. In the end the Reader will be less confused, and you will be more able to find things on your computer.

There are a few advantages to the zip file approach. First, you can circumvent authorization restrictions; you can put licensed content into your zip files and it will be analyzed just like any other content. Second, the zip file approach affords you the opportunity to pre-process your data. For example, suppose you have downloaded a set of PDF files, and each page includes some sort of header or footer. You could transform each of these PDF files into plain text, use some sort of find/replace function to remove the headers & footers. Save the result, zip it up, and submit it to the Reader. The resulting analysis will be more accurate.

There are many use-cases for the zip file approach. Masters and Ph.D students are expected to read large amounts of material. Save all those things into a folder, zip them up, and feed them to the Reader. You have been given a set of slide decks from a conference. Zip them up and feed them to the Reader. A student is expected to read many different things for History 101. Download them all, put them in a folder, zip them up, and submit them to the Distant Reader. You have written many things but they are not on the Web. Copy them to a folder, zip them up, and "read" them with the… Reader.

A zip file with a companion CSV file

The final form of input is a zip file with a companion comma-separated value (CSV) file — a metadata file.

As the size of your corpus increases, so does the need for context. This context can often be manifested as metadata (authors, titles, dates, subject, genre, formats, etc.). For example, you might want to compare & contrast who wrote what. You will probably want to observe themes over space & time. You might want to see how things differ between different types of documents. To do this sort of analysis you will need to know metadata regarding your corpus.

As outlined above, the Distant Reader first creates a cache of content — a corpus. This is the raw data. In order to do any analysis against the corpus, the corpus must be transformed into plain text. A program called Tika is used to do this work. [2] Not only does Tika transform just about any file into plain text, but it also does its best to extract metadata. Depending on many factors, this metadata may include names of authors, titles of documents, dates of creation, number of pages, MIME-type, language, etc. Unfortunately, more often than not, this metadata extraction process fails and the metadata is inaccurate, incomplete, or simply non-existent.

This is where the CSV file comes in; by including a CSV file named "metadata.csv" in the .zip file, the Distant Reader will be able to provide meaningful context. In turn, you will be able to make more informed observations, and thus your analysis will be more thorough. Here's how:

1. assemble a set of files for analysis
2. use your favorite spreadsheet or database application to create a list of the file names
3. assign a header to the list (column) and call it "file"
4. create one or more columns whose headers are "author" and/or "title" and/or "date"
5. to the best of your ability, update the list with author, title, or date values for each file
6. save the result as a CSV file named "metadata.csv" and put it in the folder/directory to be zipped
7. compress the folder/directory to create the zip file
8. submit the result to the Distant Reader for analysis

The zip file with a companion CSV file has all the strengths & weakness of the plain o' zip file, but it adds some more. On the weakness side, creating a CSV file can be both tedious and daunting. On the other hand, many search engines & index export lists with author, title, and data metadata. One can use these lists as the starting point for the CSV file.☥ On the strength side, the addition of the CSV metadata file makes the Distant Reader's output immeasurably more useful, and it leads the way to additional compare & contrast opportunities.

Summary

To date, the Distant Reader takes five different types of input. Each type has its own set of strengths & weaknesses:

* a file — good for a single large file; quick & easy; not scalable

* a URL — good for getting an overview of a single Web page and its immediate children; can include a lot of noise; has authorization limitations

* a list of URLs — can accomodate thousands of items; has authorization limitations; somewhat difficult to create list

* a zip file — easy to create; file names may get in the way; no authorization necessary; limited to 128 megabytes in size

* a zip file with CSV file — same as above; difficult to create metadata; results in much more meaningful reports & opportunities

Submitting "experiments" and downloading "study carrels"

An introduction to study carrels

The structured data of study carrels; taking inventory through the manifest

The results of the Distant Reader process is the creation of a "study carrel" — a set of structured data files intended to help you to further "read" your corpus. Using a previously created study carrel as an example, this blog posting enumerates & outlines the contents of a typical carrel. A future blog posting will describe ways to use & understand the files outlined here. Therefore, the text below is merely a kind of manifest.

The Distant Reader takes an arbitrary amount of unstructured data (text) as input, and it outputs sets of structured data files for analysis — reading. Given a corpus of any size, the Distant Reader will analyze the corpus, and it will output a myriad of reports enabling you to use & understand the corpus. The Distant Reader is intended to supplement the traditional reading process. Given a question of a rather quantitative nature, a Distant Reader study carrel may very well contain a plausible answer.

The results of downloading and uncompressing the Distant Reader study carrel is a directory/folder containing a standard set of files and subdirectories. Each of these files and subdirectories are listed & described below:

* A1426341535 — This, or a very similarly named file, is an administrative file, a unique identifier created by the system (Airivata, https://airavata.apache.org) which managed the creation of the study carrel. In the future, this file may not be included. On the other hand, since the file's name is a unique identifier, then it could be exploited by a developer.

* adr — This subdirectory contains a set of tab-delimited files. Each file contains a set of email addresses extracted from the documents in your corpus. While the files' names end in .adr, they are plain text files that can be imported into for favorite spreadsheet, database, or analysis application. The files have two columns: 1) id, and 2) address. The definitions of these columns and possible uses of these files are described elsewhere, but in short, these files can humorously answer the question "Who are you gonna call?"

* bib — This subdirectory contains a set of tab-delimited files. Each file contains a set of rudimentary bibliographic information from a given document in your corpus. While the files' names end in .bib, they are plain text files that can be imported into for favorite

spreadsheet, database, or analysis application. The files have thirteen columns: 1) id, 2) author, 3) title, 4) date, 5) page 6), extension, 7) mime, 8) words, 9) sentences, 10) flesch, 11) summary, 12) cache, and 13) txt. The definitions of these columns and possible uses of these files are described elsewhere, but in short, these files help answer the question “What items are in my corpus, and how can they be described?”

* cache — This subdirectory contains original copies of the files you intended for analysis. It is populated by harvesting content from URLs or were supplied in the zip file you uploaded to the Reader. Each file is named with a unique and somewhat meaningful name and an extension. These files are intended for reading on your computer, or better yet, printed and then read in the more traditional manner.

* css — This subdirectory contains a set of cascading stylesheets used by the HTML files in the carrel. If you really desired, one could edit these files in order to change the appearance of the carrel.

* input.zip — This file, or something named very similarly, is the file originally used to create your study carrel. It has already served its intended purpose, but it is retained for reasons of provenance.

* ent — This subdirectory contains a set of tab-delimited files, and each file contains a set of named entities from a given document in your corpus. While the files’ names end in .ent, they are plain text files that can be imported into for favorite spreadsheet, database, or analysis application. The files have five columns: 1) id, 2) sid, 3) eid, 4) entity, and 5) type. The definitions of these columns and possible uses of these files are described elsewhere, but in short, these files help answer questions regarding who, what, when, where, how, and how many.

* etc — This subdirectory contains a set of ancillary files, and each are described below:

  o model-data.txt — the data file used by topic-model.htm, and it is essentially an enhanced version of reader.txt

  o queries.sql — a set of SQL queries used to generate report.txt, and this file is an excellent introduction to the use of reader.db

  o reader.db — an SQLite database file, and it is essentially the amalgamation of the contents of the adr, bib, ent, pos, urls, and wrd directories; the intelligent use of this file can be used to answer just about any question answerable by the carrel

  o reader.sql — a set SQL commands denoting the structure of reader.db

  o reader.txt — the concatenation of all files in the txt directory; a plain text version of the whole of the corpus is often used for other purposes and it is provided here as a convienence

  o report.txt — the result of applying queries.sql to reader.db; this file has the exact same content as standard-output.txt

  o stopwords.txt — a list of function words (i.e. “a”, “an”, “the”, etc.) used through the creation of the study carrel

* figures — This subdirectory contains a set of image files used by the carrel’s HTML files:

  o adjectives.png — a word cloud illustrating the most frequent adjectives in the corpus

  o adverbs.png — a word cloud illustrating the most frequent adverbs in the corpus

  o bigrams.png — a word cloud illustrating the most frequent bigrams (two-word phrases) in the corpus

  o flesch-boxplot.png — a box plot illustrating the average, quartile, and outlier readability scores of the items in the corpus

  o flesch-histogram.png — a histogram illustrating the distribution of readability scores of the items in the corpus

  o keywords.png — a word cloud illustrating the most frequent keywords (statistically significant unigrams) in the corpus

  o nouns.png — a word cloud illustrating the most frequent nouns in the corpus

    o pronouns.png — a word cloud illustrating the most frequent pronouns in the corpus

    o proper-nouns.png — a word cloud illustrating the most frequent proper nouns in the corpus

    o sizes-boxplot.png — a box plot illustrating the average, quartile, and outlier sizes of the items (measured in unigrams) in the corpus

    o sizes-histogram.png — a histogram illustrating the distribution of sizes of the items (measured in unigrams) in the corpus

    o topics.png — a pie chart illustrating how the corpus is subdivided if topic modeling were applied to the corpus, and the desired number of topics (latent themes) equals five

    o unigrams.png — a word cloud illustrating the most frequent unigrams (individual words) in the corpus

    o verbs.png — a word cloud illustrating the most frequent verbs in the corpus

  * htm — This subdirectory contains a set of interactive HTML files linked from the file named index.htm. The functionality of each file is outlined below:

    o adjective-noun.htm — search, sort, and browse adjective/noun combinations by adjective, noun, or frequency

    o adjectives.htm — search, sort, and browse adjectives and/or their frequency

    o adverbs.htm — search, sort, and browse adverbs and/or their frequency

    o bigrams.htm — search, sort, and browse bigrams (two-word phrases) and/or their frequency

    o entities.htm — search, sort, and browse named-entities, their type, and/or their frequency

    o keywords.htm — search, sort, and browse keywords (statistically significant unigrams) and/or their frequency

    o noun-verb.htm — search, sort, and browse noun/verb combinations by noun, verb, or frequency

    o nouns.htm — search, sort, and browse nouns and/or their frequency

    o pronouns.htm — search, sort, and browse pronouns and/or their frequency

    o proper-nouns.htm — search, sort, and browse proper nouns and/or their frequency

    o quadgrams.htm — search, sort, and browse quadgrams (four-word phrases) and/or their frequency

    o questions.htm — search, sort, and browse questions (sentences ending with a question mark) and from which items they were extracted

    o search.htm — a free text query interface based on the narrative summaries of each item in the corpus

    o topic-model.htm — a topic modeler; a tool used to enumerate as well as compare & contrast latent themes in the corpus

    o trigrams.htm — search, sort, and browse trigrams (three-word phrases) and/or their frequency

    o unigrams.htm — search, sort, and browse unigrams (individual words) and/or their frequency

    o verbs.htm — search, sort, and browse verbs and/or their frequencies

  * index.htm — This HTML file narratively reports on the content of your study carrel. It is the best place to begin once you have downloaded and unzipped the carrel.

  * MANIFEST.htm — This file, and it is the third best place to begin once you have downloaded and unzipped a carrel.

* job_1819387465.slurm — This file, or a very similarly named file, is the batch file used to initially create your study carrel. In the future, this file may be removed from the study carrel all together because it serves only an administrative purpose.

* js — This subdirectory includes a set of Javascript libraries supporting the functionality of index.htm as well as the HTML files in the htm directory. Because these files are here your computer does not need to be connected to the Internet in order to effectively read your carrel. Study carrels are designed to be stand-alone file systems usable for years to come.

* LICENSE — This is the license file; each study carrel is distributed under a GNU Public License.

* pos — This subdirectory contains a set of tab-delimited files, and each file contains a set of part-of-speech files from a given document in your corpus. While the files' names end in .pos, they are plain text files that can be imported into for favorite spreadsheet, database, or analysis application. The files have six columns: 1) id, 2) sid, 3) tid, 4) token, 5) lemma, and 6) pos. The definitions of these columns are described in another blog posting. The definitions of these columns and possible uses of these files are described elsewhere, but in short, these files help answer question regarding who, what, how, how many, and actions as well as grammer and style.

* README — This file contains the very briefest of introductions to the carrel.

* standard-error.txt — As each study carrel is being created, error and status messages are output to this file. It is a log file. If the creation of your study carrel fails, then this is a good place to look for clues on what went wrong. Send me this file if you are stymied.

* standard-output.txt — After your study carrel as been created and distilled into a database, sets of queries are applied against the database. This file is the second best place to begin once you have downloaded and unzipped a carrel.

* tsv — Except for one (questions.tsv), this subdirectory contains a set of frequency tables in the form of tab-delimited text files. The exception is a tab-delimited text file too, but it is just not a frequency file. All of these files can be imported into for favorite spreadsheet, database, or analysis application. Possible uses for these files are destined to be outlined in future postings, but in short, perusal of these files will help you answer questions regarding your corpus's "aboutness" as well as who, what, when, where, how, how many, and why questions. The structure of each file is listed below:

    o adjective-noun.tsv — three columns: 1) adjective, 2) noun, and 3) frequency where frequency denotes the number of times the given adjective appears immediately before the given noun in the corpus

    o adjectives.tsv — two columns: 1) adjective, and 2) frequency

    o adverbs.tsv — two columns: 1) adverb, and 2) frequency

    o bigrams.tsv — two columns: 1) bigram (two-word phrase), and 2) frequency

    o entities.tsv — three columns: 1) entity, 2) type, and 3) frequency

    o keywords.tsv — two columns: 1) keyword (statistically significant unigram), and 2) frequency

    o noun-verb.tsv — three columns: 1) noun, 2) verb, and 3) a frequency where frequency denotes the number of times the given noun appears immediately before the given verb in the entire corpus

    o nouns.tsv — two columns: 1) noun, and 2) frequency

    o pronouns.tsv — two columns: 1) pronoun, and 2) frequency

    o proper-nouns.tsv — two columns: 1) proper, and 2) frequency

    o quadgrams.tsv — two columns: 1) quadgram (four-word phrase), and 2) frequency

    o questions.tsv — two columns: 1) identifier, and 2) question where each question is a "sentence" ending in a question mark

    o trigrams.tsv — two columns: 1) trigram (three-word phrase), and 2) frequency

    o unigrams.tsv — two columns: 1) unigram (individual word), and 2) frequency

    o verbs.tsv — two columns: 1) verb, and 2) frequency

  * txt — This subdirectory contains plain text versions of the files stored in the cache directory. A plain text version of each & every item in the cache directory ought to exist in this directory. The contents of this directory is what was used to do the Reader's analysis. The contents of this directory are excellent candidates for further analysis with tools such as concordances, indexers, or topic modelers.

  * urls — This subdirectory contains a set of tab-delimited files, and each file contains a set of URLs from a given document in your corpus. While the files' names end in .url, they are plain text files that can be imported into for favorite spreadsheet, database, or analysis application. The files have three columns: 1) id, 2) domain, and 3) url. The definitions of these columns and possible uses of these files are described elsewhere, but in short, these files help answer questions regarding document provenance and relationships as well as addressing the perenial issue of "finding more like this one".

  * wrd — This subdirectory contains a set of tab-delimited files, and each file contains a set of computed keywords from a given document in your corpus. While the files' names end in .wrd, they are plain text files that can be imported into for favorite spreadsheet, database, or analysis application. The files have two columns: 1) id, and 2 keyword. The definitions of these columns and possible uses of these files are described elsewhere, but in short, these files help answer questions such as "What is this document about?"

Links

[1] Airivata — https://airavata.apache.org

Using combinations of desktop tools to analyze the data

This section first describes the types of desktop tools (computer programs) the student, researcher, or scholar will need in order to use a Distant Reader study carrel. This section then describes how some of the more specific tools can be used for the purpose of use & understanding.

Three essential types of desktop tools

There are three essential types of desktop tools you will need/want in order to use the content of a study carrel. These types include: text editors, spreadsheet/database applications, and analysis programs.

Text editors read and write plain text files -- files with no formatting and no binary characters. Plain text files usually have a .txt extension. Every single file in a Distant Reader study carrel (except one) is a plain text file, and therefore, every single file (except one) is openable by any text editor.

There are a few essencial tools you will want in a text editor, and the most important is a find/replace function. The function ought to allow you to find any character and change it to something else. This is useful for removing stopwords from a file. It is useful for removing carriage returns or newline characters, thus unwrapping your text. The text editor gets bonus points if the find/replace function supports "regular expressions". The second most important function of a text editor is a sorting feature. Each line in many text files is really an item in a list, and you will invariably want to sort the list. Another very useful function of a text editor, especially used for the purposes of text mining and natural language processing, is the ability to change the case of all letters to either their upper or lower-case forms. Such is the most basic of text normalization/cleaning processes. Religious wars are fought over text editors, and for the purposes of this workbook, only a number are listed, and not all of them support all the functions outlined above: Notepad, Wordpad, Text Edit, Notepad++, Atom, and BBedit.
The student, researcher, or scholar will want/need a plain text editor in order to truly exploit the use of the Distant Reader.

Spreadsheet/database applications are designed to read "delimited" files, plain text files where each line is a row in a matrix, and each item is punctuated by some special character such as a tab character or a comma. These items are the columns in the matrix. The whole file

is a kin to a spreadsheet or a database. Like a text editor, you will want to use the spreadsheet/database application to support sort. The spreadsheet/database application will need to be able to do arithmetic against items in the file. The spreadsheet/database application ought to include charting features. And the spreadsheet/database application ought to be able to save/export its data to other types of delimited files: tab-delimited files, CSV (comma-separated value) files, Excel workbook files, HTML tables, etc.

The majority of the files in a study carrel are delimited files, and these delimited files are really annotated lists. Examples include lists of word and their parts-of-speech, lists of documents and the URLs they contain, or lists of sentences and the named-entities they include. Given these files the student, researcher, or scholar can compare & contrast the ratio of named entities across a corpus, or they could plot the ebb & flow of an idea over time.

Analysis programs cover a wide spectrum of tools, and for the purposes of the workbook, these tools fall into a number of categories: counting & tabulating, concordancing, topic modeling, and visualizing. For the purpose of this workbook OpenRefine will be used to support counting & tabulating, as well as a few other things. Concordancing is really about find, and a program called AntConc is described. Topic modeling is the process of extracting latent themes from a body of text, and a GUI application called Topic Modeling Tool is demonstrated. Two visualization tools are useful: Wordle and Gephi. The former outputs tag clouds, and the later outputs network diagrams. The student, researcher, or scholar is expected to supplement visualization with the charting functions of spreadsheet/database applications.

Again, every single file (except one) making up a study carrel is a plain text file, and all those files are readable by a text editor. The majority of the files in a study carrel are delimited files, specifically, tab-delimited files, and they can be opened with a spreadsheet/database application. The student, researcher, or scholar will need to have these sort of programs at their disposal. The other, more specific application used in this workbook, are freely available for downloading as well as cross-platform in nature.

Wordle

Visualized word frequencies, while often considered sophomoric, can be quite useful when it comes to understanding a text, especially when the frequencies are focused on things like parts-of-speech, named entities, or co-occurrences. Wordle visualizes such frequencies very well. For example, the frequency of all words in the Iliad and the Odyssey, the frequency of all nouns, or the frequency of all words associated with the word ship.

[INSERT IMAGES HERE.]

[INSERT RECIPE HERE.]

Concordancing with AntConc

Concordancing is really the process of find, and AntConc is a very useful program for this purpose. Given one or more plain text files, AntConc will enable the student, researcher, or scholar to: find all the occurrences of a word, illustrate where the word is located, navigate through document where the word occurs, list word collocations, and calculate quite a number of useful statistics regarding a word. Concordancing, dating from the 13th Century, is the oldest from of text mining. Think of it as ^F (control-f) on steroids. AntConc does all this and more. For example, one can load all of the Iliad and the Odyssey into AntConc. Find all the occurrences of the word ship, visualize where ship appears in each chapter, and list the most significant words associated with the word ship.

[INSERT IMAGES HERE.]

[INSERT RECIPE HERE.]

Excel - charting tabular data

OpenRefine - fielded searching and grouping

OpenRefine eats delimited files for lunch. The student, researcher, or scholar can use OpenRefine to open one or more delimited file. OpenRefine will then parse the file(s) into fields. It can makes many things easy such as finding/replacing, faceting (think "grouping"), filtering (think "searching"), sorting, clustering (think "normalizing/cleannig"), counting &

tabulating, and finally, exporting data. OpenRefine is an excellent go-between when spreadsheets fail and full-blown databases are too hard to use.

[INSERT IMAGES HERE.]

[INSERT RECIPE HERE.]

Topic Modeling Tool - enumerating latent themes

Technically speaking, topic modeling is an unsupervised machine learning process used to extract latent themes from a text. Give a text and an integer, a topic modeler will count & tabulate the frequency of words and compare those frequencies with the distances between the words. The words form "clusters" when they are both frequent and near each other, and these clusters can sometimes represent themes, topics, or subjects. Topic modeling is often used to denote the "aboutness" of a text or compare themes between authors, dates, genres, demographics, other topics, or other metadata items.

Topic Modeling Tool is a GUI/desktop topic modeler based on the venerable MALLET suite of software. It can be used in a number of ways, and it is relatively easy to: list five distinct themes from the Iliad and the Odyssey, compare those themes between books, and compare those things over time (assuming each chapter happens chronologically).

[INSERT IMAGES HERE.]

[INSERT RECIPE HERE.]

Using command-line tools dig even deeper

Reader Extras - a set of tools for collections of carrels

Reader Lite - a tool for one-off items

Summary/conclusion

About the author

Eric Lease Morgan has been practicing librarianship since 1984, but he has been consistently writing software since 1976. He is currently employed at the University of Notre Dame where he works in the Navari Family Center for Digital Scholarship of where he provides text mining and natural language processing services to the University community.