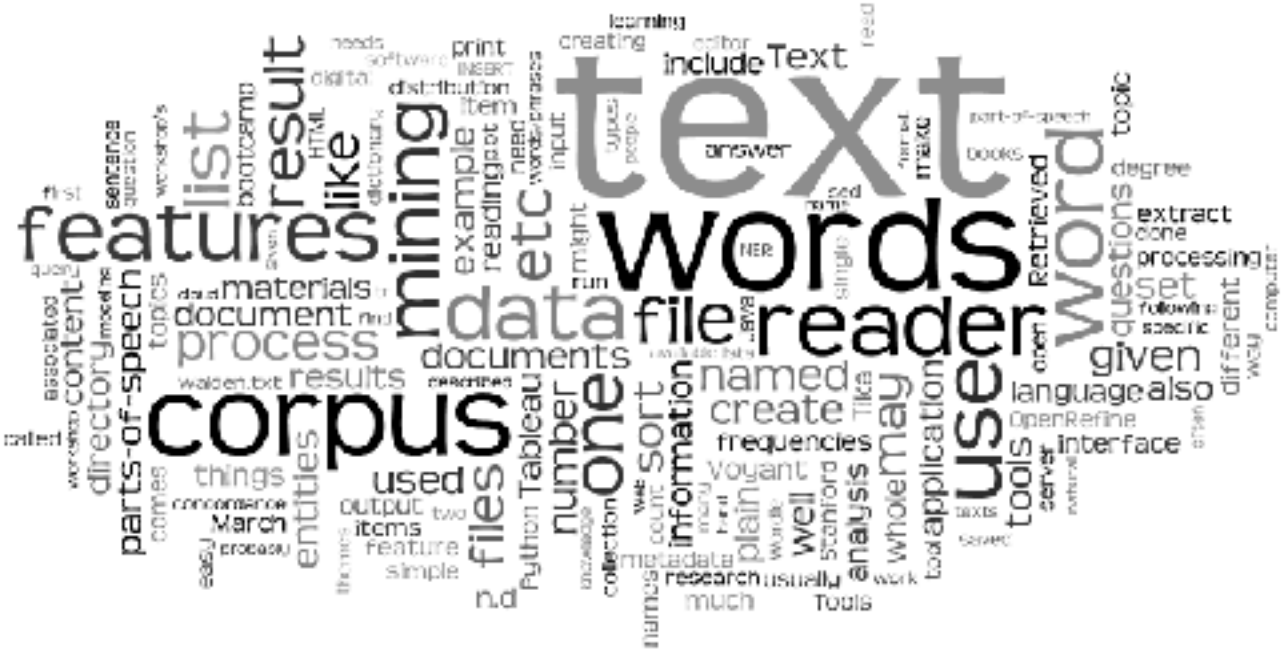


Text mining: Beyond the basics

Eric Lease Morgan (May 14, 2018)

Contents

Introduction

This workbook represents the printed handout for a text mining bootcamp originally slated for the European Library Automation Group (ELAG) 2018 meeting. The bootcamp is called "Text mining: Beyond the basics".

This three-hour, hands-on training session is intended & designed for librarians, programmers, and/or researchers of any type. It requires a computer with an Internet connection and a fully-functional text editor like NotePad++ or BBEdit. A local Java installation and familiarity with the command line are both desirable but not necessary. Programming experience is not a prerequisite, but knowledge of regular expressions is helpful. A website (GitHub repository is/ought to be available where the reader can download this handout as well as all of the supporting documents/scripts.[1]

Using freely available tools, it is possible to go beyond basic text mining, and instead, begin to read corpora at "scale". This bootcamp teaches how. At the end, participants will be able to:

- identify patterns, anomalies, and trends in a corpus
- practice both "distant" and "scalable" reading
- enhance & complement their ability to do "close" reading
- use & understand any corpus of poetry or prose

Activities in the bootcamp include:

- learning what text mining is, and why one should care
- creating a corpus
- creating a plain text version of a corpus
- creating simple word lists with a text editor
- creating advanced word lists with other tools
- cleaning & analyzing a corpus with OpenRefine
- charting & graphing a corpus with Tableau Public
- extracting parts-of-speech
- extracting named entities
- identifying topics and themes
- using a concordance to intelligently "read" a corpus

Anybody with sets of texts can benefit from this bootcamp. Any corpus of textual content is apropos: journal articles, books, the complete run of a magazine, blog postings, Tweets, press releases, conference proceedings, websites, poetry, etc. This bootcamp is operating system agnostic. All the software used in this workshop is freely available on the 'Net, or it is already installed on one's computer.

Let's get started!?

Eric Lease Morgan <eric_morgan@infomotions.com>
Notre Dame, Indiana (United States)

May 14, 2018

[1] GitHub repository - https://github.com/ericleasemorgan/textmining-bootcamp

What is text mining, and why should I care?

Text mining is a process used to identify, enumerate, and analyze syntactic and semantic characteristics of a corpus, where a corpus is a collection of documents usually in the form of plain text files. The purpose of this process it to bring to light previously unknown facts, look for patterns & anomalies in the facts, and ultimately have a better understanding of the corpora as a whole. Text mining makes the implicit explicit. It's process is to extract "features" from one or more documents to provide insight into a corpus. The result of text mining is akin to the development of tables-of-contents and back-of-the-book indexes manifesting themselves in books. The results of text mining enables a reader to use, navigate, and ultimately attempt to understand a corpus. Text mining only provides the reader with facts and measurements. It is up to the reader to interpret these facts for the purposes of extracting meaning. Text mining is akin to the use of a thermometer. The thermometer only gives you a temperature. It is up to the reader determine whether or not the outside temperature is hot or cold.

The simplest of text mining processes merely count & tabulate a document's "tokens" (usually words but sometimes syllables). The counts & tabulations are akin to the measurements and observations made in the physical and social sciences. Statistical methods can then be applied to the observations for the purposes of answering questions like:

- What is the average length of documents in the collection, and do they exhibit a normal distribution?
- What are the most common words/phrases in a document?
- What are the most common words/phrases in a corpus?
- What are the unique words/phrases in a document?
- What are the infrequent words/phrases in a corpus?
- What words/phrases exist in every document and to what extent?
- Where do given words/phrases appear in a text?
- What other words surround a given word/phrase?
- What words/phrases are truly representative of a document or corpus?
- If a document or corpus where to be described in a single word, then what would that word be? How about described in three words? How about describing a document with three topics where each topic is denoted with five words?

The answers to these questions bring to light a corpus's previously unknown features enabling the reader to use & understand a corpus more fully. Given the answers to these sorts of questions, a person can learn when Don Quixote actually tilts at windmills, to what degree does Thoreau's Walden use the word "ice" in the same breath as "pond", or how has the definition of "scientific practice" has evolved over time?

Given models created from the results of natural language processing, other characteristics (sentences, parts-of-speech, named entities, etc.) can be parsed. These values can also be counted & tabulated enabling the reader to answer new sets of questions:

- How difficult is a document to read?
- What is being discussed in a corpus? To what degree are the things the names of people, organizations, places, dates, money amounts, etc? What percentage of the personal pronouns are male, female, or neutral?
- What is the action in a corpus? What things happen in a document? Are things explained? Said? Measured?
- How are things in the corpus described? Overall, are the connotations positive or negative? Do the connotations evolve within a document?

The documents in a corpus are often associated with metadata such as authors, titles, dates, subjects/keywords, numeric rankings, etc. This metadata can be combined with measurements & observations to answer questions like:

- How have the use of specific words/phrases waxed & waned over time?
- To what degree do authors write about a given concept?
- What are the significant words/phrases described with a given genre?
- Are there correlations between words/phrases and given document's usefulness score?

Again, text mining is a process, and the process usually includes the following steps:

1. articulating a research question
2. amassing a corpus of study
3. coercing the corpus into a form amenable to computer processing
4. taking measurements and making observations
5. analyzing the results and drawing conclusions

Articulating a research question can be as informally stated as, "I'd like to know more about this corpus" or "I'd like to garner an overview of the corpus before I begin reading it in earnest." On the other hand, articulating a research question can be as formal as a dissertation's thesis statement. The purpose of articulating a research question -- no matter how formal -- is to give you a context for your investigations. Knowing a set of questions to answer helps you determine what tools you will employ in your inquires.

Creating a corpus is not always as easy as you might think. The corpus can be as small as a single document, or as large as millions. The "documents" in the corpus can be anything from tweets from a Twitter feed, Facebook postings, survey comments, magazine or journal articles, reference manuals, books, screen plays, musical lyrics, etc. The original documents may have been born digital or not. If not, then they will need to be digitized in one way or another. It is better if each item in the corpus is associated with metadata, such as authors, titles, dates, keywords, etc. Actually obtaining the documents may be impeded by copyrights, licensing restrictions, or hardware limitations. Once the corpus is obtained, it is useful to organize it into a coherent whole. There is a lot to think about when it comes to corpus creation.

Coercing a corpus into a form amenable to computer processing is a chore in & of itself. In all cases, the documents' text needs to be in "plain" text. This means the document includes only characters, numbers, punctuation marks, and a limited number of symbols. Plain text files include no graphical formatting. No bold. No italics, no "codes" denoting larger or smaller fonts, etc. Documents are usually manifested as files on a computer's file system. The files are usually brought together as lists, and each item in the list have many attributes -- the metadata describing each item. Furthermore, each document may need to be normalized, and normalization may include changing the case of all letters to lower case, parsing the document into words (usually called "features"), identifying the lemmas or stems of a word, eliminating stop/function words, etc. Coercing your corpus into coherent whole is not to be underestimated. Remember the old adage, "Garbage in, garbage out."

Ironically, taking measurements and making observations is the easy part. There are a myriad of tools for this purpose, and the bulk of this bootcamp describes how to use them. One important note: it is imperative to format the measurements and observations in a way amenable to analysis. This usually means a tabular format where each column denotes a different observable characteristic. Without formatting measurements and observation in tabular formats, it will be difficult to chart and graph any results.

Analyzing the results and drawing conclusions is the subprocess of looping back to Step #1. It is where you attempt to actually answer the questions previously asked. Keep in mind that human interpretation is a

necessary part of this subprocess. Text mining does not present you with truth, only facts. It is up to you to interpret the facts. For example, suppose the month is January and the thermometer outside reads 32º Fahrenheit (0º Centigrade), then you might think nothing is amiss. On the other hand, suppose the month is August, and the thermometer still reads 32º, then what might you think? "It is really cold," or maybe, "The thermometer is broken." Either way, you bring context to the observations and interpret things accordingly. Text mining analysis works in exactly the same way.

Finally, text mining is not a replacement for the process of traditional reading. Instead, it ought to be considered as complementary, supplemental, and a natural progression of traditional reading. With the advent of ubiquitous globally networked computers, the amount of available data and information continues to grow exponentially. Text mining provides a means to "read" massive amounts of text quickly and easily. The process is akin the inclusion of now-standard parts of a scholarly book: title page, verso complete with bibliographic and provenance metadata, table of contents, preface, introduction, divisions into section and chapters, footnotes, bibliography, and a back-of-the-book index. All of these features make a book's content more accessible. Text mining processes applied to books is the next step in accessibility. Text mining is often described as "distant" or "scalable" reading, and it is often contrasted with the "close" reading. This is a false dichotomy, but only after text mining becomes more the norm will the dichotomy fade.

Building a corpus

A corpus is a collection of one or more texts organized into a coherent whole. Building a corpus is very much like traditional library work. It requires a number of steps:

- articulating what you are going to collect; develop a "collection management policy"
- identifying qualified materials
- acquiring the materials
- describing the materials
- organizing the materials

But of course, "Whenever you have a hammer, everything begins to look like a nail."

What to collect or questions to answer

First, the reader needs to ask themselves, "What are the questions I'm trying to answer?" Articulating these questions is not always very easy. The questions can be as simple as, "What is this particular text about?" or "Can I get a summary of the text's action main ideas?" Other times, the questions can be as in-depth as a Ph.D. thesis statement, "How was Henry David Thoreau influenced by his contemporaries such as Ralph Waldo Emerson or Henry Wadsworth Longfellow?" or "How are the works of Aristotle & Plato both similar & different?" For the purposes of this bootcamp, the research questions include thing like the following:

- How do the tweets of academic institutions, art museums, and national news organization differ?
- How feasible it it to mine different forms of content: PDF, plain text, HTML, tweets, etc.
- To what degree is it possible to "read" an entire issue of a journal with the help of computation?
- What are some of the essential characteristics of early 19th Century American literature?
- Who are some of the major players in the field of digital humanities?

Identifying materials

Identifying qualifying materials for investigation requires the knowledge of the subject matter as well as the location of relevant content. Qualifying materials will take different forms. The more traditional forms are books & journals, which may or may not be available on the 'Net or in a library (in analog or digital formats). The materials may be HTML pages harvested from a Web server or merely available on a file system. The content may be survey results, and if the reader is lucky, then the results exist in a tab-delimited format whose columns are adequately enumerated & described. The answers to the research questions may exist in social media outlets such as the things of Twitter, Facebook, Instagram, etc.

At this point it is important to distinguish between the "types" of materials and their "formats" because the definitions of the two terms are often conflated. For the purposes of this bootcamp, "types" are denoted a genre of content such as but not limited to: books, scholarly articles, magazine articles, pictures, movies, sounds, tweets, blog postings, survey responses, etc. This is contrasted with "format" which denotes the way a genre of material is (digitally) manifested. Examples include: PDF, HTML, plain text, TIFF, JPEG, Excel spreadsheets, SQL, tab-delimited files, Microsoft Word, etc. With this in mind, a book can be manifested as a PDF file, an HTML file, or even a set of JPEG images. Similarly, an article may be manifested as a PDF file and/or an HTML file. Understanding the different between types and formats will assist the reader in the process of acquiring their materials as well as organizing them into a coherent collection/corpus.

For the purposes of this bootcamp, the qualifying materials include books by Aristotle, Plato, Austen, Bronte, Longfellow, Emerson, Thoreau, and Twain. There are articles from Digital Humanities Quarterly, First Monday, and Inside ASIS&T. There are tweets from art museums (the Metropolitan Art Museum, the Art Institute of Chicago, and the Detroit Art Museum), universities (Oxford, Cambridge, and Harvard), and news organizations (CBS News, Fox News, and NBC News). There are also various & sundry Web pages on the topics of the digital humanities, text mining, and American literature. On the whole, the qualifying materials are characterized by subject, type, and format.

Acquiring materials

Acquiring the materials for text mining is the next step in the process. The process may be as easy as borrowing books from a library or taking advantage of a library's book & article subscription databases. The reader may have the necessary materials already on their computer. Increasingly and more than likely, the desired content is available on the 'Net, which means in needs to be harvested -- locally downloaded. For anything but the most trivial of collections, harvesting content from the Web is not as easy as it seems. Sure, one or two items are easily obtained, but how does one systematically download a gross amount of content -- enough to give more than an overly qualified answer to a research question? The answer falls into a combination of three categories: 1) crawling, 2) screen scraping, or 3) exploiting an application programmer interface (API). Crawling is usually the most feasible. First, feed a URL to an application like curl or wget.[1] [2] The application will output the content at the other end of the URL. The reader can repeat this process both selectively as well as recursively. Screen scraping reads an HTML page and tries to extract its content sans any Javascript, navigation, or boiler plate information. Much of this work can now be done with an application called Tika.[3] Exploiting an API is probably the cleanest method for acquiring content from the 'Net, but it requires a knowledge of one or more programming languages, and not all websites support an API. Examples include OAI-PMH, linked data, or a site-specific API such as the ones offered by the Internet Archive.[4] Taking advantages of API's is beyond the scope of this workbook.

Almost all of the content for this bootcamp was harvested through the diligent use of the venerable wget get application.

Describing & organizing materials

When text mining is done against an individual item or even a small number of items, describing the items is not imperative, but as a collection gets larger, so does the need for metadata.

Meaningful text mining can be done against an individual item, such as a book. The process can yield knowledge of the book's action, reading difficulty, themes, etc. But as soon as the corpus includes more than one item, there will most likely be a desire to compare & contrast items. Yes, the items' action, reading difficulty, and themes can be made explicit, but as soon as they are the reader will probably desire to know to what degree each characteristic is applicable to each item. To satisfy this desire, each item needs to be associated with different forms of metadata, and the metadata will be dictated by the research question(s). For example, if there is a desire to learn how things evolve over time, then each item will need to be associated

---

[1] curl - https://curl.haxx.se

[2] wget - https://www.gnu.org/software/wget/

[3] Tika - http://tika.apache.org

[4] Internet Archive API's - http://blog.archive.org/developers/

with dates. If the reader desires to understand the differences in themes between authors, then the names of authors need to be associated with each item. If the collection only contains a single author and themes are to be compared & contrasted, then identifiers of the items (titles, keys, locations, etc) will need to be associated with each item. A whole litany of metadata possibilities exist: author/creator names, author/creator gender, titles, dates, literary genres, subjects/keywords, material type, material format, place of publication, distribution mechanism, denotation of authority (peer-reviewed or not), etc.

Keeping track of each item and its associated metadata can be as simple as a systematic file naming system or as complex as the exploitation of a relational database application. The later usually means associating each item in the collection with a unique identifier which points to a series of tables in a database where each table includes different metadata. The former approach has been used for this bootcamp. More specifically, each item in the collection has been saved in one or more directories/folders. Each directory/folder name denotes its subject, author, or material type (not format). Each file in each directory/folder has been given a name usually denoting authorship. Given this organizational scheme, it will be possible to text mine (make the implicit explicit) information against subjects (American literature, digital humanities, art, academia, and world affairs), authors/publishers, and types (books, journal & magazine articles, news, and tweets).

A recipe for creating a corpus

As an aside, here is a recipe for creating a corpus based on the content of a Web page.

Given an HTML file (file.html), the reader can extract (most) of the URLs it contains by chaining together a set of Unix/Linux commands:

```
$ # extract URLs
  cat file.html             | \
  egrep -o 'https?://[^ ]+' | \
  sed -e 's/https/http/g'   | \
  sed -e 's/>.*//g'         | \
  sed -e 's/\W+$//g'        | \
  sed -e 's/"//g'           | \
  sort                      | \
  uniq                      | \
  sort -bnr
```

The result can be saved to a file (urls.txt), and then uses as input to wget:

```
$ wget -i urls.txt
```

So, for example, the reader could peruse a Wikipedia article, save it locally, extract all of the URLs, filter out any undesirable URLs, and harvest the balance for more in-depth investigation. The result will be a set of content all but ripe for text mining.

A Bash shell script (./bin/harvest-wikipedia.sh) which is a part of this bootcamp, does just this. Given a URL and a directory, it will cache an HTML page, extract all of its URLs, and then retrieve the content of each one, giving them (somewhat) meaningful file names. For example, a collection of texts about book binding can be gotten in this way:

```
$ ./bin/harvest-wikipedia.sh \
  https://en.wikipedia.org/wiki/Bookbinding \
  ~/Desktop/bookbinding
```

Often times the articulation of the research question, the identification of the qualified materials, the acquisition the materials, describing them, and organizing them into a coherent whole go hand-in-hand and are interdependent. Each limits & informs the others. The process is usually not sequential but instead parallel and iterative. Ideally all would be factored & decided upon independently but such is generally not the case. The whole corpus-building process is a lot like deciding on what to have for dinner. The decision is limited by what one wants, what is affordable, what is in the refrigerator, and how everything can be combined into a delicious and satisfying meal.

Creating a plain text version with Tika

It is imperative to create plain text versions of corpus items.

Text mining can not be done without plain text data. This means HTML files need to be rid of markup. It means PDF files need to have been "born digitally" or they need to have been processed with optical character recognition (OCR), and then the underlying text needs to be extracted. Word processor files need to converted to plain text, and the result saved accordingly. The days of plain o' ASCII text files need to be forgotten. Instead, the reader needs to embrace Unicode, and whenever possible, make sure characters in the text files are encoded as UTF-8. With UTF-8 encoding, one gets all of the nice accent marks so foreign to United States English, but one also gets all of the pretty emoticons increasingly sprinkling our day-to-day digital communications. Moreover, the data needs to be as "clean" as possible. When it comes to OCR, do not fret too much. Given the large amounts of data the reader will process, "bad" OCR (OCR with less than 85% accuracy) can still be quite effective.

Converting harvested data into plain text used to be laborious as well as painful, but then a Java application called Apache Tika came on the scene. Tika comes in two flavors: application and server. The application version can take a single file as input, and it can output metadata as well as any underlying text. The application can also work in batch mode taking a directory as input and saving the results to a second directory. Tika's server version is much more expressive, more powerful, and very HTTP-like, but it requires more "under the hood" knowledge to exploit to its fullest potential.

For the sake of this workshop, versions of the Tika application and Tika server are included in the distribution, and they have been saved in the lib directory with the names tika-desktop.jar and tika-server.jar. The reader can run the desktop/GUI version of the Tika application by merely double-clicking on it. The result will be a dialog box.

Drag a PDF (or just about any) file on to the window, and Tika will extract the underlying text. To use the command-line interface, something like this could be run to output the help text:

```
$ java -jar ./lib/tika-desktop.jar --help
```

```
> java -jar .\lib\tika-desktop.jar --help
```

And then something like these commands to process a single file or a whole directory of files:

```
$ java -jar ./lib/tika-desktop.jar -t <filename>
```

```
$ java -jar ./lib/tika-desktop.jar -t -i <input directory> -o <output directory>

> java -jar .\lib\tika-desktop.jar -t <filename>

> java -jar .\lib\tika-desktop.jar -t -i <input directory> -o <output directory>
```

Try transforming a few files individually as well as in batch. What does the output look like? To what degree is it readable? To what degree has the formatting been lost? Text mining does not take formatting into account, so there is no huge loss in this regard.

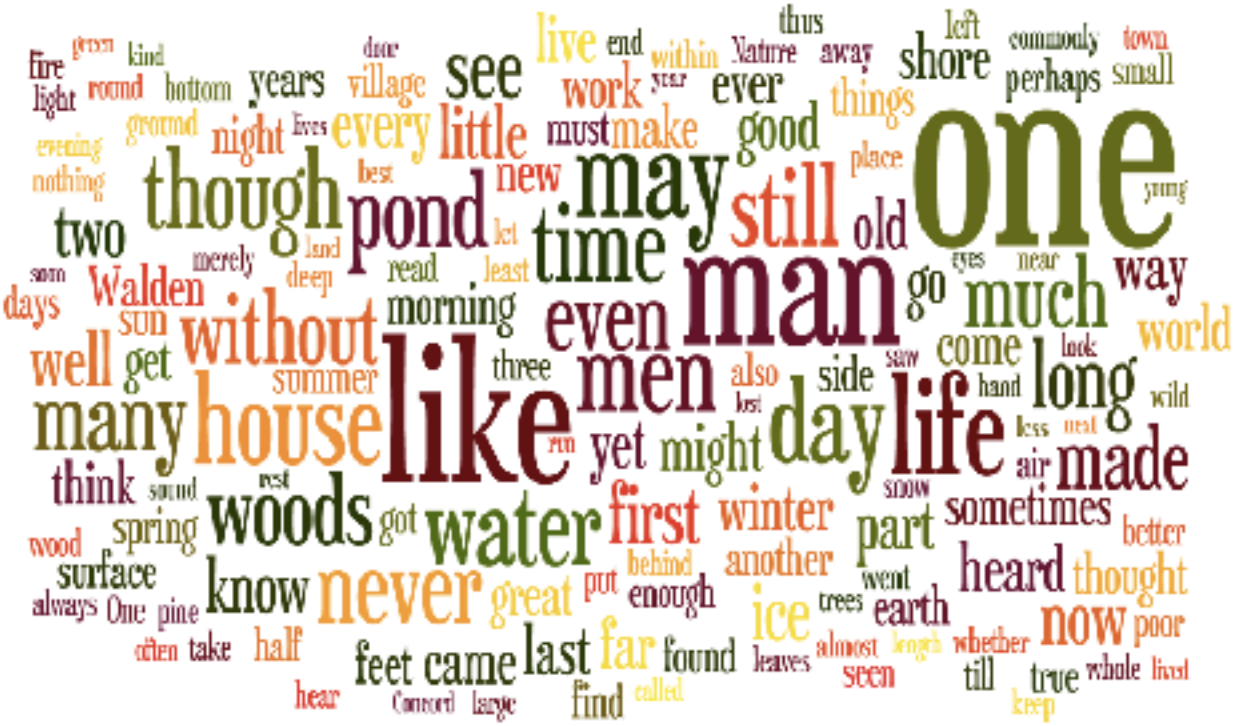
Without some sort of scripting, the use of Tika to convert harvested data into plain text can still be tedious. Consequently, the whole of the workshop's harvested data has been pre-processed with a set of Perl and bash scripts (which probably won't work on Windows computers unless some sort of Linux shell has been installed):

- ./bin/tika-server.sh - runs Tika in server mode on TCP port 8080, and waits patiently for incoming connections
- ./bin/tika-client.pl - takes a file as input, sends it to the server, and returns the plain text while handling the HTTP magic in the middle
- ./bin/file2txt.sh - a front-end to the second script taking a file and directory name as input, transforming the file into plain text, and saving the result with the same name but in the given directory and with a .txt extension

The entirety of the harvested data has been transformed into plain text for the purposes of this workshop. ("Well, almost all.") The result has been saved in the folder/directory named "corpus". Peruse the corpus directory. Compare & contrast its contents with the contents of the harvest directory. Can you find any omissions, and if so, then can you guess why/how they occurred?

Word clouds with Wordle

A word cloud, or sometimes called a “tag cloud” is a fun, easy, and popular way to visualize the characteristics of a text. Usually used to illustrate the frequency of words in a text, a word clouds make some features (“words”) bigger than others, sometimes colorize the features, and amass the result in a sort of “bag of words” fashion.



Many people disparage the use of word clouds. This is probably because word clouds may have been over used. The characteristics they illustrate are sometimes sophomoric. Or too much value has been given to their meaning. Despite these facts, a word cloud is an excellent way to initialize the analysis of texts.

There are many word cloud applications and programming libraries, but Wordle is probably the easiest to use as well as the most popular.[1] [2] To get started, use your Web browser and go to the Wordle site. Click the Create tab and type some text into the resulting text box. Submit the form. Your browser may ask for permissions to run a Java application, and if granted, the result ought to be simple word cloud. The next step is to play with Wordle’s customizations: fonts, colors, layout, etc. To begin doing useful analysis, open a file from the workshop’s corpus, and copy/paste it into Wordle. What does the result tell you? Copy/paste a different file into Wordle and then compare/contrast the two word clouds.

---

[1] Since Wordle is a Web-based Java application, the use of Wordle is also a good test case to see whether or not Java is installed and configured on your desktop computer.

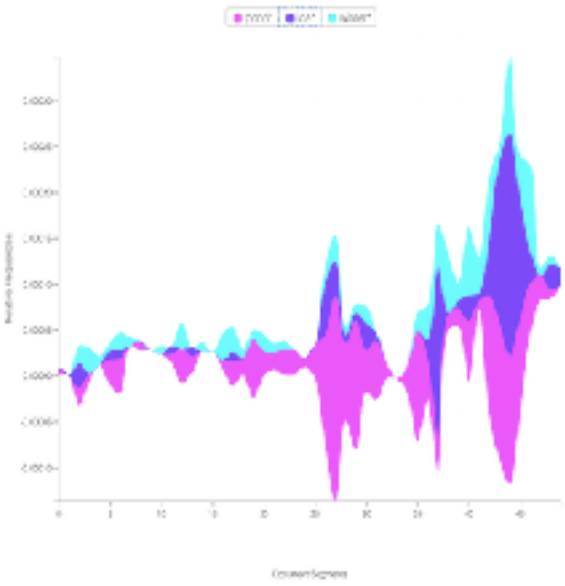
2 Wordle - http://www.wordle.net

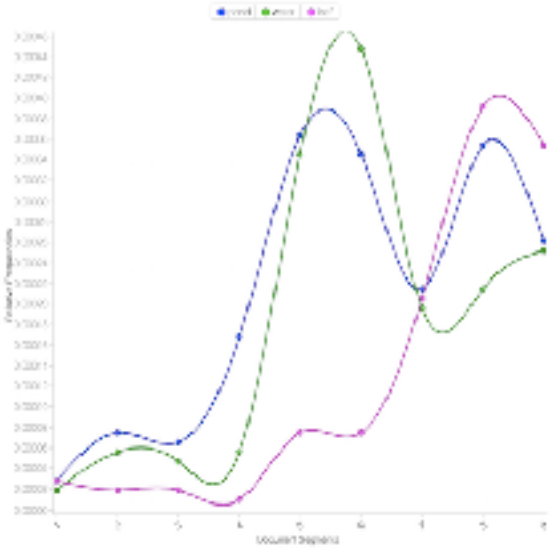
By default, Wordle make effort to normalize the input. It removes stop words, lower-cases letters, removes numbers, etc. Wordle then counts & tabulates the frequencies of each word to create the visualization. But the frequency of words only tells one part of a text's story. There are other measures of interest. For example, the reader might want to create a word cloud of ngram frequencies, the frequencies of parts-of-speech, or even the log-likelihood scores of significant words. To create the sorts of visualization as word clouds, the reader must first create a colon-delimited list of features/scores, and then submit them under Wordle's Advanced tab.

The challenging part of this process is created the list of features/scores, and the process can be done using a combination of the tools described in the balance of the workshop.

Using Voyant Tools to do some "distant reading"

Voyant Tools is often the first go-to tool used by either: 1) new students of text mining and the digital humanities, or 2) people who know what kind of visualization they need/want.[1] Voyant Tools is also one of the longest supported tools described in this bootcamp.

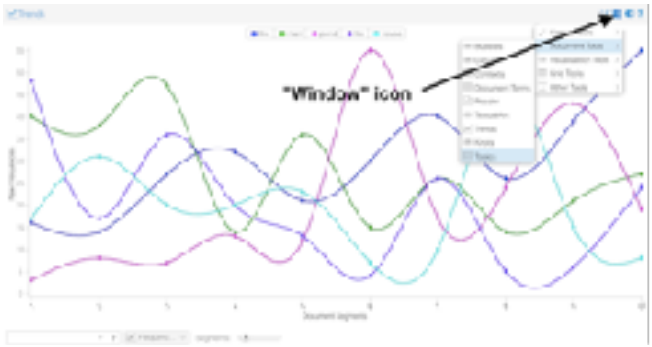




As stated the Tool's documentation: "Voyant Tools is a web-based text reading and analysis environment. It is a scholarly project that is designed to facilitate reading and interpretive practices for digital humanities

[1] Voyant Tools - https://voyant-tools.org/

students and scholars as well as for the general public." To that end it offers a myriad of visualizations and tabular reports characterizing a given text or texts. Voyant Tools works quite well, but like most things, the best use comes with practice, a knowledge of the interface, and an understanding of what the reader wants to express. To all these ends, Voyant Tools counts & tabulates the frequencies of words, plots the results in a number of useful ways, supports topic modeling, and the comparison documents across a corpus. Examples include but are not limited to: word clouds, dispersion plots, networked analysis, "stream graphs", etc.



Voyant Tools's initial interface consists of six panes. Each pain encloses a feature/function of Voyant. In the author's experience, Voyant Tools' is better experienced by first expanding one of the panes to a new window ("Export a URL"), and then deliberately selecting one of the tools from the "window" icon in the upper left-hand corner. There will then be displayed a set of about two dozen tools for use against a document or corpus.
Using Voyant Tools the reader can easily ask and answer the following sorts of questions:

- What words or phrases appear frequently in this text?
- How do those words trend throughout the given text?
- What words are used in context with a given word?
- If the text were divided into T topics, then what might those topics be?
- Visually speaking, how do given texts or sets of text cluster together?

After a more thorough examination of the reader's corpus, and after making the implicit more explicit, Voyant Tools can be more informative. Randomly clicking through its interface is usually daunting to the

novice. While Voyant Tools is easy to use, it requires a combination of text mining knowledge and practice in order to be used effectively. Only then will useful "distant" reading be done.

Extracting simple "features" with a text editor and command-line tools

In the world of text mining, natural language processing, and machine learning the word “features” is used to denote a set of textual characteristics. The reader’s text editor is the first tool to be used in this regard.

Features are denoted by a tokens (usually words) associated with some sort of measure. For example, a document may be 1,000 tokens (words) long. A list of all the words in a document, with duplicates removed, is called a dictionary, and the dictionary will be of a certain length. Sequential pairs of words (bigrams) can have frequencies. A third set of features may be a list of all the nouns, verbs, or adjectives with their individual counts & tabulations. A list of names, places, or organization (“named entities”) are another set of features.

Given a feature-rich text editor, the lengths of documents and the makings of a dictionary are easily within the reader’s grasp. There are a number of (Linux-based) tools that can take text as input, process it in a number of ways, and output features. If the reader is well-practices with these tools, then these sorts of tasks will increasingly become a part of one’s everyday reading experience.

Creating a dictionary -- a simple list of words in a work or corpus -- is an essential text mining process. There are many computer language modules and libraries which do this work for you, but learning how to create a dictionary with a text editor is simple and straight-forward. Once a dictionary is created, surprisingly interesting observations can be made. Here’s how to create a simple dictionary:

1. load a plain text file into your text editor
2. find & replace all space characters with carriage returns
3. find & replace leading and trailing white space characters with nothingness
4. find & replace leading or trailing punctuation marks with nothingness
5. sort the text

The result ought to be a long list of words, and many of them will be repeated. Search & browse the list. Ask yourself, “What words seem to occur frequently? Can I articulate a word of particular interest? If so, then does that word appear in the list and how often?”

At this point, the reader may want to save the file, and the open it again in OpenRefine, which will enable one to count & tabulate the words giving a frequency list. It will also enable the reader to count & tabulate sets of words with similar shapes such as book, books, book-making, books-binding, etc. Once these tabulations are done, the reader can copy the result, find & place tab characters with colons, and supply the result as input to Wordle for a visualization.

Dictionaries -- lists of words -- can include more than individual words. Dictionaries can also be lists of ngrams or specific types of words. For example, a dictionary may include all the two-word, three-word, four-word, etc., phrases in a text. Additionally, dictionaries may include all words of a given part-of-speech, named entities, roots (“stems”) of a word, or a word’s lemma (“canonical” form). Listing ngrams, parts-of-speech, named entities, roots, or lemmas are beyond the abilities of text editors. Instead, the techniques of machine learning are needed here. Creating such dictionaries is not within the bounds of text editors.

On the other hand, from the Unix/Linux command line, there are a number of "recipes" for doing some rudimentary analysis.[1] For example, given a file (file.txt), the reader can do things such as the following:

```
  $ # list all words
    cat file.txt                          | \
    tr -sc "[A-Z][a-z][0-9]'" '[\012*]' | \
    tr '[A-Z]' '[a-z]'                    | \
    tr '[:space:]' '[\n*]'                | \
    grep -v "^\s*$"                       | \
    sort                                  | \
    uniq -c                               | \
    sort -bnr

  $ # list all bigrams
    cat file.txt                                          | \
    tr -sc "[A-Z][a-z][0-9]'" '[\012*]'                   | \
    tr '[A-Z]' '[a-z]'                                    | \
    tr '[:space:]' '[\n*]'                                | \
    awk -- 'prev!="" { print prev,$0; } { prev=$0; }' | \
    sort                                                  | \
    uniq -c                                               | \
    sort -bnr

  $ # list all trigrams
    cat file.txt                           | \
    tr -sc "[A-Z][a-z][0-9]'" '[\012*]'  | \
    tr '[A-Z]' '[a-z]'                     | \
    tr '[:space:]' '[\n*]'                 | \
    awk -- 'first!=""&&second!="" { print first,second,$0; } { first=second;
second=$0; }' | \
    sort                       | \
    uniq -c                    | \
    sort -bnr

  $ # extract all URLs
    cat file.txt               | \
    egrep -o 'https?://[^ ]+' | \
    sed -e 's/https/http/g'    | \
    sed -e 's/\W+$//g'         | \
    sort                       | \
    uniq -c                    | \
    sort -bnr

  $ # extract domains from URLs
    cat file.txt               | \
    egrep -o 'https?://[^ ]+' | \
    sed -e 's/https/http/g'    | \
    sed -e 's/\W+$//g'         | \
    sed -e 's/http:\/\///g'    | \
    sed -e 's/\/.*$//g'        | \
    sort                       | \
    uniq -c                    | \
    sort -bnr

  $ # extract all email addresses
    cat file.txt                                                | \
    grep -i -o '[A-Z0-9._%+-]\+@[A-Z0-9.-]\+\.[A-Z]\{2,4\}' | \
    sort                                                        | \
    uniq -c                                                     | \
    sort -bnr
```

[1] See also "Unix for poets" (https://www.cs.upc.edu/~padro/Unixforpoets.pdf) and "Basic Text Analysis with Command Line Tools in Linux" (https://williamjturkel.net/2013/06/15/basic-text-analysis-with-command-line-tools-in-linux/)

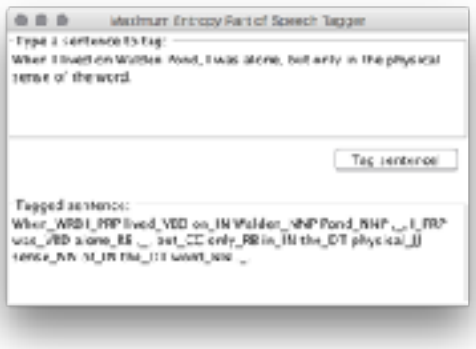
Extracting parts-of-speech and named entities with Stanford tools

Extracting specific parts-of-speech as well as "named entities", and then counting & tabulating them can be quite insightful.

Parts-of-speech include nouns, verbs, adjectives, adverbs, etc. Named entities are specific types of nouns, including but not limited to, the names of people, places, organizations, dates, times, money amounts, etc. By creating features out of parts-of-speech and/or named entities, the reader can answer questions such as:

- What is discussed in this document?
- What do things do in this document?
- How are things described, and how might those descriptions be characterized?
- To what degree is the text male, female, or gender neutral?
- Who is mentioned in the text?
- To what places are things referring?
- What happened in the text?

There are a number of tools enabling the reader to extract parts-of-speech, including the venerable Brill part-of-speech tagger implemented in a number of programming languages, CLAWS, the Apache OpenNLP, and a specific part of the Stanford NLP suite of tools called the Stanford Log-linear Part-Of-Speech Tagger.[2] Named entities can be extracted with the Stanford Named Entity Recognizer (NER).[3] This workshop exploits the Stanford tools.



The Stanford Log-linear Part-Of-Speech Tagger is written in Java, making it a bit difficult for most readers to use in the manner it was truly designed, the author included. Luckily, the distribution comes with a command-line interface allowing the reader to use the tagger sans any Java programing. Because any part-of-speech or named entity extraction application is the result of a machine learning process, it is necessary to use a previously created computer model. The Stanford tools comes quite a few models from which to choose. The command-line interface also enables the reader to specify different types of output: tagged, XML, tab-delimited, etc. Because of all these options, and because the whole thing uses Java "archives" (read programming libraries or modules), the command-line interface is daunting, to say the least.

After downloading the distribution, the reader ought to be able to change to the bin directory, and execute either one of the following commands:

```
$ stanford-postagger-gui.sh

> stanford-postagger-gui.bat
```

[2] Stanford Log-linear Part-Of-Speech Tagger - https://nlp.stanford.edu/software/tagger.shtml

[3] Stanford Named Entity Recognizer (NER) - https://nlp.stanford.edu/software/CRF-NER.shtml

The result will be a little window prompting for a sentence. Upon entering a sentence, tagged output will result. This is a toy interface, but demonstrates things quite nicely.

The full-blown command-line interface is bit more complicated. From the command-line one can do either of the following, depending on the operating system:

```
$ stanford-postagger.sh models/english-left3words-distsim.tagger walden.txt

> stanford-postagger.bat models\english-left3words-distsim.tagger walden.txt
```

The result will be a long stream of tagged sentences, which I find difficult to parse. Instead, I prefer the inline XML output, which is much more difficult to execute but much more readable. Try either:

```
$ java -cp stanford-postagger.jar: edu.stanford.nlp.tagger.maxent.MaxentTagger -model
models/english-left3words-distsim.tagger -outputFormat inlineXML -outputFormatOptions
lemmatize -textFile walden.txt

> java -cp stanford-postagger.jar: edu.stanford.nlp.tagger.maxent.MaxentTagger -model
models\english-left3words-distsim.tagger -outputFormat inlineXML -outputFormatOptions
lemmatize -textFile walden.txt
```

In these cases, the result will be a long string of ill-formed XML. With a bit of massaging, this XML is much easier to parse with just about any compute programming language, believe it or not. The tagger can also be run in server mode, which makes batch processing a whole lot easier. The workshop's distribution comes a server and client application for exploiting these capabilities, but, unfortunately, they won't run on Windows computers unless some sort of Linux shell has been installed. Some people can issue the following command to launch the server from the workshop's distribution:

```
$ ./bin/pos-server.sh
```

The reader can run the client like this:

```
$ ./bin/pos-client.pl walden.txt
```

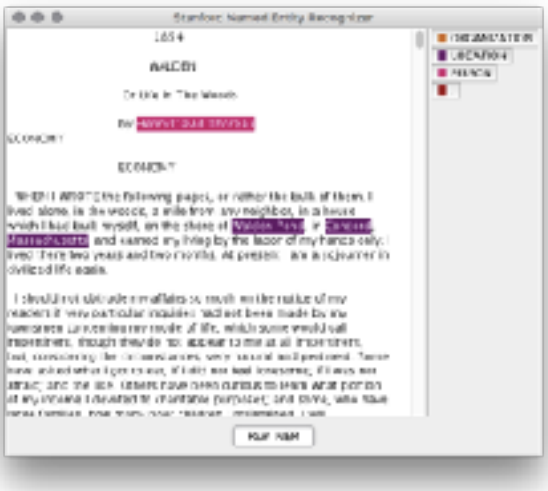
The result will be a well-formed XML file, which can be redirected to a file, processed by another script converting it into a tab-delimited file, and finally saved to a second file for reading by a spreadsheet, database, or data analysis tool:

```
$ ./bin/pos-client.pl walden.txt > walden.pos; ./bin/pos2tab.pl walden.pos > walden.tsv
```

For the purposes of this workshop, the whole of the harvested data has been pre-processed with the Stanford Log-linear Part-Of-Speech Tagger. The result is been mirrored in the parts-of-speech folder/directory. The reader can open the files in the parts-of-speech folder/directory for analysis. For example, you might open them in OpenRefine and try to see what verbs appear most frequently in a given document. My guess the answer will be the lemmas "be" or "have". The next set of most frequently used verb lemmas will probably be more indicative of the text.

The process of extracting features of name entities is very similar with the Stanford NER. The original Stanford NER distribution comes with a number of jar files, models, and configuration/parameter files. After downloading the distribution, the reader can run a little GUI application, import some text, and run NER. The result will look something like the following image.

The simple command-line interface takes a single file as input, and it outputs a stream of tagged sentences. For example:



```
$ ner.sh walden.txt

> ner.bat walden.txt
```

Each tag denotes an entity (i.e. the name of a person, the name of a place, the name of an organization, etc.). Like the result of all machine learning algorithms, the tags are not necessarily correct, but upon closer examination, most of them are pretty close. Like the POS Tagger, this workshop's distribution comes with a set of scripts/programs that can make the Stanford NER tool locally available as a server. It also comes with a simple client to query the server. Like the workshop's POS tool, the reader (with a Macintosh or Linux computer) can extract named entities all in two goes:

```
$ ./bin/pos-server.sh

$ ./bin/pos-client.pl walden.txt > walden.ner; \
  ./bin/pos2tab.pl walden.ner > walden.tsv
```

Like the workshop's pre-processed part-of-speech files, the workshop's corpus has been pre-processed with the NER tool. The preprocessed files ought to be in a folder/directory named... named-entities. And like the parts-of-speech files, the "ner" files are tab-delimited text files readable by spreadsheets, databases, OpenRefine, etc. For example, you might open one of them in OpenRefine and see what names of people trend in a given text. Try to create a list of places (which is not always easy), export them to a file, and open them with Tableau Public for the purposes of making a geographic map.

Extracting parts-of-speech and named entities straddles simple text mining and natural language processing. Simple text mining is often about counting & tabulating features (words) in a text. These features have little context sans proximity to other features. On the other hand, parts-of-speech and named entities denote specific types of things, namely specific types of nouns, verbs, adjectives, etc. While these thing do not necessarily denote meaning, they do provide more context than simple features. Extracting parts-of-speech and named entities is (more or less) a easy text mining task with more benefit than cost. Extracting parts-of-speech and named entities goes beyond the basics.

Cleaning & analyzing word lists with OpenRefine

OpenRefine is a system/application enabling the reader to analyze information and begin to turn it into knowledge.[1]

OpenRefine can read and parse a number of different data structures such as line-delimited files, tab-delimited files, a few flavors of XML, JSON, fixed-width files, some Google documents, and to some degree even MARC data. The imported data is then presented to the reader in a tabular format. Each column in the resulting table can be searched ("filtered"), browsed, faceted, normalized, or otherwise manipulated for analysis. Once analysis is done, the whole lot can be exported to other file formats for further refinement or visualization.

For example, examining the contents of a list of words (a "dictionary") can be done by:

1. saving a dictionary to a file
2. launching OpenRefine
3. creating a new project with the saved file
4. accepting the defaults, and continue to create the project
5. creating a "Text facet" on the list of words
6. sorting the list of facets by count

The result will be a list of words and their frequencies. The beginning of the list will be full of stop ("function") words. Ignoring the stop words, one can get a better idea of what words are mentioned more frequently, and therefore you can begin to enumerate a text's themes. Scroll to the end of the list to see the words occurring infrequently. Perusing the list of infrequent words can be just as insightful as perusing the list of frequent words.

OpenRefines supports the searching ("filtering") of columns, and searching is enhanced through the use of regular expressions. Using this feature the readers can begin to identify as well as compare & contrast themes. For example, you might want to compare & contrast the frequency of the words "man" and "god". Here's how:

1. remove all facets and filters
2. create a new "Text filter" on the list of words
3. enter "^man$|^god$" into the query field
4. make sure the "regular expression" check box is checked
5. create a "Text facet" on the list of words
6. sort the list of facets by count

The result is a tabulation of the words "man" and "god", and then you can the determine whether one theme is highlighted more often than another. Additional queries (replacements for step #3) might include:

- `^love.*|^war.*`
- `^man.*|^men.*|^woman.*|^women.*`
- `^man$|^men$|^woman$|^women$`
- `^mr$|^mrs$|^ms$`
- `^time$|^space$`

[1] OpenRefine - http://openrefine.org

The faceted search results can be ordered alphabetically or by count. Alphabetic listings are useful when it comes to browsing for specific words. Listings by count denote frequencies. In either case, the tabulations can be exported as a tab-delimited list, imported into a text editor, reformatted for the purposes of visualizations, and... visualized. For example, the following recipe creates a word cloud of colors:

1. remove all facets/filters
2. filter the dictionary for colors (`^black$|^white$|^gray$|^grey$|^brown$|^red$|^yellow$|^blue$|^ orange$|^green$|^purple$|^violet$`)
3. create a "Text facet" on the search results
4. select the number of choices
5. copy the result into your text editor
6. find & replace the tab character for a colon (:)
7. copy the list
8. use your browser to go to http://wordle.net
9. select the Advanced tab
10. paste the list into the input and Go

The result ought to look something like the figure.

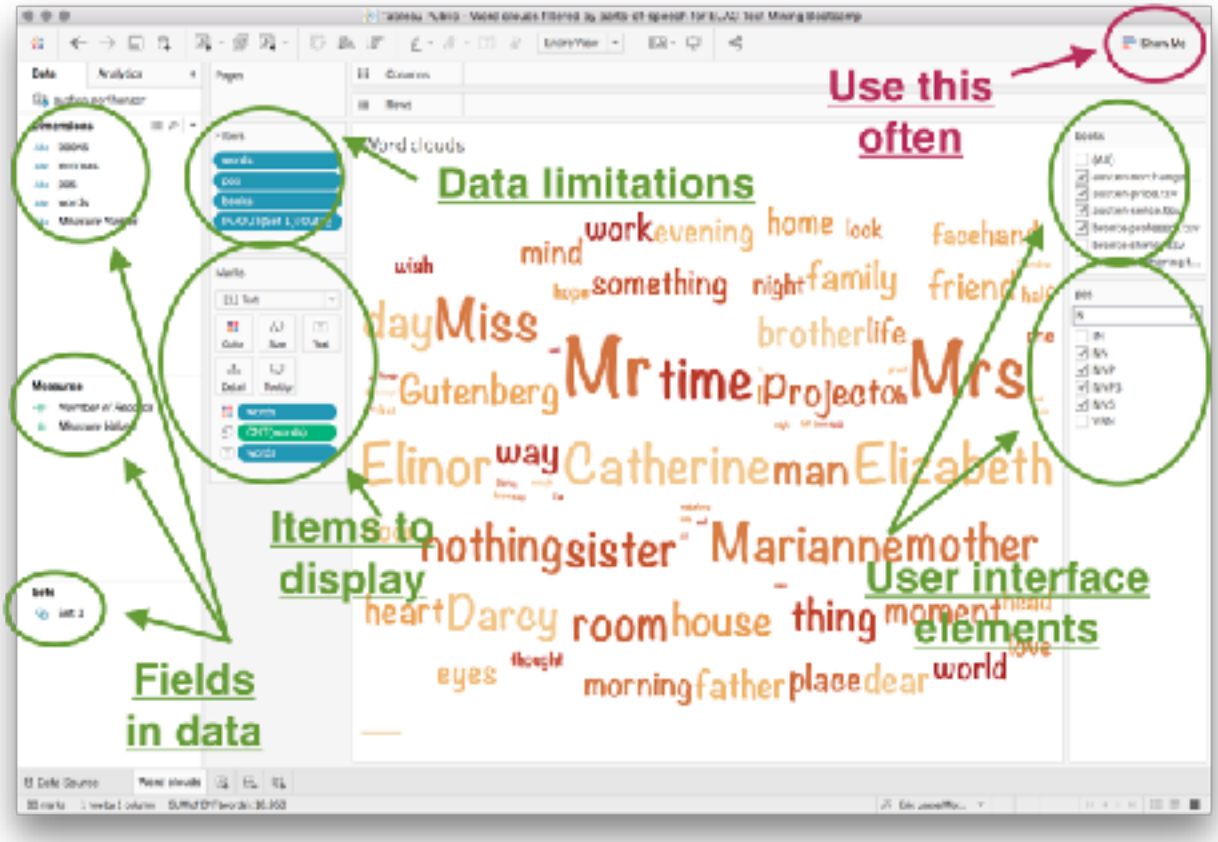
The dataset used in this tutorial is more than rudimentary because it only includes a single column. OpenRefine eats tab-delimited files for lunch, and consequently it supports tables with quite a few column. Moreover, there are a myriad of functions that can be applied to values in the columns, and there is a scripting language beneath the whole thing. Four of the more interesting functions include: 1) "clustering" -- a process to automatically correct differences in word spellings, 2) global find/replace operations, 3) "reconciliations" -- a process of creating keys/identifiers with your data in order to link & associate it with other data like cities & countries, and 4) graphing numeric columns.

Charting & graphing with Tableau Public

They say, “A picture is worth a thousand words”, and through use of something like Tableau this can become a reality in text mining.

After extracting features from a text, you will have almost invariably created lists. Each of the items on the lists will be characterized with bits of context thus transforming the raw data into information. These lists will probably take the shape of matrices (sets of rows & columns), but other data structures exist as well, such as networked graphs. Once the data has been transformed into information, you will want to make sense of the information -- turn the information into knowledge. Charting & graphing the data is one way to make that happen.

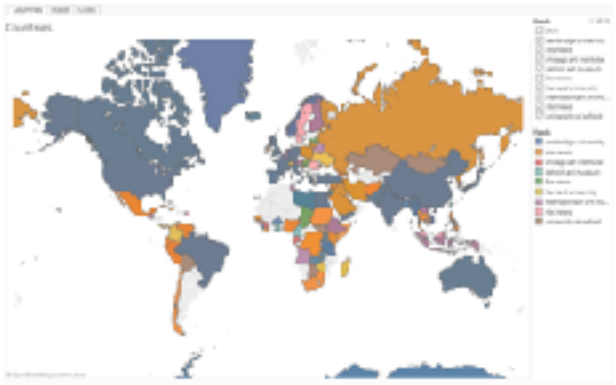
For example, the reader may have associated each word in a text with a part-of-speech, and then this association was applied across a corpus. The reader might then ask, “To what degree are words associated with each part-of-speech similar or different across items in the corpus? Do different items include similar or different personal pronouns, and therefore, are some documents more male, more female, or more gender neutral?” Alternatively, suppose the named entities have been extracted from items in a corpus, then the reader may want to know, “What countries, states, and/or cities are mentioned in the text/corpus, and to what degree? Are these texts ‘European’ in scope?”



A charting & graphing application like Tableau (or Tableau Public) can address these questions.[1] The first can be answered by enabling the reader to select one more more items from a corpus, select one or more parts-of-

[1] Tableau Public - https://public.tableau.com/

speech, counting & tabulating the intersected words, and displaying the result as a word cloud. The second question can be addressed similarly. Allow the reader to select items from a corpus, extract the names of places (countries, states, and cities), and plot the geographic coordinates on a global map. Once these visualizations are complete, they can be saved on the Web for others to use, for example, a word cloud or a world map.[2] [3]



Creating visualizations with Tableau (or Tableau Public) takes practice. Not only does the reader need to have structured data in hand, but one needs to be patient in the learning of the interface. To the author's mind, the whole thing is reminiscent of the venerable HyperCard program from the 1980's where one was presented with a number of "cards", and programming interfaces were created by placing "objects" on them.

This workshop comes with two previously created Tableau workbooks located in the etc directory (word-clouds.twbx and maps.twbx). Describing the process to create them is beyond the scope of this workshop, but an outline follows:

1. amass sets of data, like parts-of-speech or named entities
2. import the data into Tableau
3. in the case of the named entities, convert the data to "Geographic Roles"
4. drag data elements to the Marks, Rows, or Columns cards
5. make liberal use of the Show Me feature
6. drag data elements to the Filters card
7. observe the visualizations and turn your information into knowledge

Tableau is not really intended to be used against textual data/information; Tableau is more useful and more functional when applied to tabular numeric data. After all, the program is called... Tableau. This does not mean Tableau can not be exploited by the text miner. It just means it requires practice and an ability to articulate a question to be answered with the help of a visualization.

---

[2] word cloud - https://tabsoft.co/2JQimPd

[3] world map - https://tabsoft.co/2FJ6g83

Identifying themes and clustering documents using MALLET

Topic modeling is an unsupervised machine learning process. It is used to create clusters (read "subsets") of documents, and each cluster is characterized by sets of one or more words. Topic modeling is good at answering questions like, "If I were to describe this collection of documents in a single word, then what might that word be? How about two?" or make statements like, "Once I identify clusters of documents of interest, allow me to read/analyze those documents in greater detail." Topic modeling can also be used for keyword ("subject") assignment; topics can be identified and then documents can be indexed using those terms. In order for a topic modeling process to work, a set of documents first needs to be assembled. The topic modeler then, at the very least, takes an integer as input, which denotes the number of topics desired. All other possible inputs can be assumed, such as use of a stop word list or denoting the number of time the topic modeler ought to internally run before it "thinks" it has come the best conclusion.

MALLET is the grand daddy of topic modeling tools, and it supports other functions such as text classification and parsing. It is essentially a set of Java-based libraries/modules designed to be incorporated into Java programs or executed from the command line.[1] A subset of MALLET's functionality has been implemented in a program called topic-modeling-tool, and the tool bills itself as "A GUI for MALLET's implementation of LDA."[2] Topic-modeling-tool provides an easy way to read what possible themes exist in a set of documents or how the documents might be classified. It does this by creating topics, displaying the results, and saving the data used to create the results for future use. Here's one way:

1. create a set of plain text files, and save them in a single directory
2. run/launch topic-modeling-tool
3. specify where the set of plain text files exist
4. specify where the output will be saved
5. denote the number of topics desired
6. execute the command with "Learn Topics"

The result will be a set of HTML, CSS, and CSV files saved in the output location. The "answer" can also be read in the tool's console.

A more specific example is in order. Here's how to answer the question, "If I were describe this corpus in a single word, then what might that one word be?":

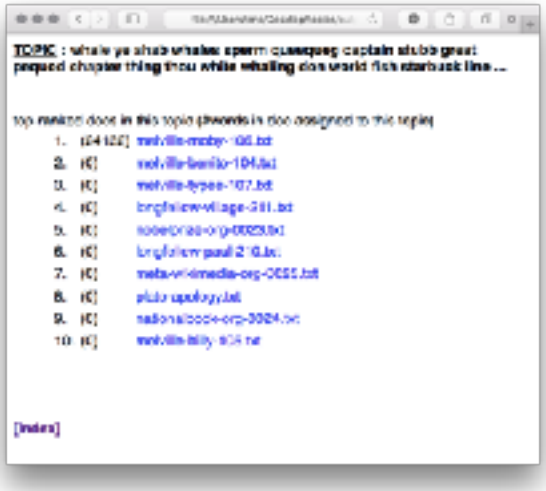
1. repeat Steps #1-#4, above
2. specify a single topic to be calculated
3. press "Optional Settings..."
4. specify "1" as the number of topic words to print
5. press okay
6. execute the command with "Learn Topics"

What word can be used to describe your collection?

---

[1] MALLET - http://mallet.cs.umass.edu/

[2] topic-modeling-tool - https://github.com/senderle/topic-modeling-tool



Iterate the modeling process by slowly increasing the number of desired topics and number of topic words. Personally, I find it interesting to implement a matrix of topics to words. For example, start with one topic and one word. Next, denote two topics with two words. Third, specify three topics with three words. Continue the process until the sets of words ("topics") seem to make intuitive sense. After a while you may observe clear semantic distinctions between each topic as well as commonalities between each of the topic words. Distinctions and commonalities may include genders, places, names, themes, numbers, OCR "mistakes", etc.

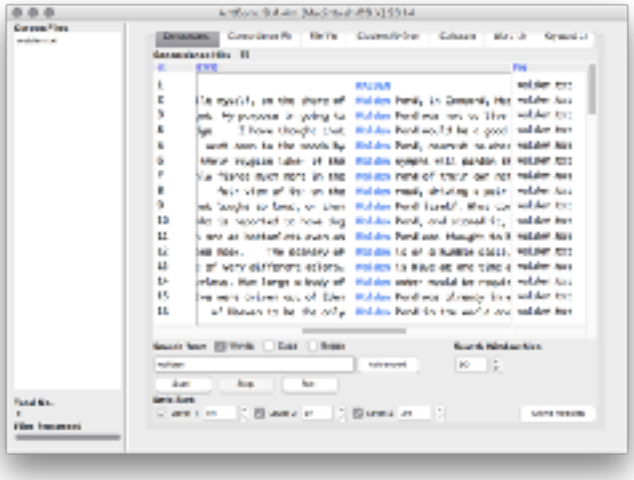
Using a concordance to facilitate searching keywords in context

A concordance is one of the oldest of text mining tools dating back to at least the 13th century when they were used to analyze and "read" religious texts. Stated in modern-day terms, concordances are key-word-in-context (KWIC) search engines. Given a text and a query, concordances search for the query in the text, and return both the query as well as the words surrounding the query. For example, a query for the word "pond" in a book called Walden may return something like the following:

```
1.   uilt myself, on the shore of Walden Pond, in Concord, Massachusetts, and earned
2.    got. My purpose in going to Walden Pond was not to live cheaply nor to live dea
3.   owledge. I have thought that Walden Pond would be a good place for business, not
4.   lives. The loon retires to solitary ponds to spend it. Thus also the snake casts
5.   nd went down to the woods by Walden Pond, nearest to where I intended to build m
6.   , through which I looked out on the pond, and a small open field in the woods wh
7.   s were springing up. The ice in the pond was not yet dissolved, though there wer
8.   d had placed the whole to soak in a pond-hole in order to swell the wood, I saw
9.    stray goose groping about over the pond and cackling as if lost, or like the sp
10.  ng the nails, and removed it to the pond-side by small cartloads, spreading the
11.  oads of stones up the hill from the pond in my arms. I built the chimney after m
```

The use of a concordance enables the reader to learn the frequency of the given query as well as how it is used within a text (or corpus).

Digital concordances offer a wide range of additional features. For example, queries can be phrases or regular expressions. Search results and be sorted by the words on the left or on the right of the query. Queries can be clustered by the proximity of their surrounding words, and the results can be sorted accordingly. Queries and their nearby terms can be scored not only by their frequencies but also by the probability of their existence. Concordances can calculate the position of a query i a text and illustrate the result in the form of a dispersion plot or histogram.



AntConc is a free, cross-platform concordance program that does all of the things listed above, as well as a few others.[1] The interface is not as polished as some other desktop applications, and sometimes the usability can be frustrating. On the other hand, given practice, the use of AntConc can be quite illuminating. After downloading and running AntConc, give these tasks a whirl:

1. use the File menu to open a single file
2. use the Word List tab to list token (word) frequencies
3. use the Settings/Tool Preferences/Word List Category to denote a set of stop words
4. use the Word List tab to regenerate word frequencies
5. select a word of interest from the frequency list to display the KWIC; sort the result
6. use the Concordance Plot tab to display the dispersion plot

[1] AntConc - http://www.laurenceanthony.net/software/antconc/

7. select the Collocates tab to see what words are near the selected word
8. sort the collocates by frequency and/or word; use the result to view the concordance

The use of a concordance is often done just after the creation of a corpus. (Remember, a corpus can include one or more text files.) But the use of a concordance is much more fruitful and illuminating if the features of a corpus are previously made explicit. Concordances know nothing about parts-of-speech nor grammar. Thus they have little information about the words they are analyzing. To a concordance, every word is merely a token -- the tiniest bit of data. Whereas features are more akin to information because they have value. It is better to be aware of the information at your disposal as opposed to simple data. Do not rush to the use of a concordance before you have some information at hand.

Epilog

The use to text mining in the humanities is simply the application of computing techniques to the liberal arts. Their use is similar to use of the magnifying glass by Galileo. Instead of turning it down to count the number of fibers in a cloth (or to write an email message), it is being turned up to gaze at the stars (or to analyze the human condition). What he finds there is not so much truth as much as new ways to observe.

Webliography

This is a list of useful links for the reader. They mostly point to software referenced in this workbook.

1. **AntConc** (http://www.laurenceanthony.net/software/antconc/) - "A freeware corpus analysis toolkit for concordancing and text analysis."

2. **Apache OpenNLP** (http://opennlp.apache.org/index.html) - "OpenNLP supports the most common NLP tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, language detection and coreference resolution."

3. **Basic Text Analysis with Command Line Tools in Linux** (https://williamjturkel.net/2013/06/15/basic-text-analysis-with-command-line-tools-in-linux/) - A useful webpage outlining how to do bits of text mining from the Unix/Linux command line.

4. **BBEdit** (https://www.barebones.com/products/bbedit/) - "BBEdit is the leading professional HTML and text editor for macOS. This award-winning product has been crafted to serve the needs of writers, Web authors and software developers, and provides an abundance of features for editing, searching, and manipulation of text. An intelligent interface provides easy access to BBEdit's best-of-class features, including grep pattern matching, search and replace across multiple files, project definition tools, function navigation and syntax coloring for numerous source code languages, code folding, FTP and SFTP open and save, AppleScript, macOS Unix scripting support, text and code completion, and of course a complete set of robust HTML markup tools."

5. **CLAWS part-of-speech tagger** (http://ucrel.lancs.ac.uk/claws/) - "Part-of-speech (POS) tagging, also called grammatical tagging, is the commonest form of corpus annotation, and was the first form of annotation to be developed by UCREL at Lancaster. Our POS tagging software for English text, CLAWS (the Constituent Likelihood Automatic Word-tagging System), has been continuously developed since the early 1980s. The latest version of the tagger, CLAWS4, was used to POS tag c.100 million words of the British National Corpus (BNC)."

6. **curl** (https://curl.haxx.se/) - "curl is used in command lines or scripts to transfer data. It is also used in cars, television sets, routers, printers, audio equipment, mobile phones, tablets, settop boxes, media players and is the internet transfer backbone for thousands of software applications affecting billions of humans daily."

7. **DiRT Directory** (http://dirtdirectory.org/) - "Bamboo DiRT is a tool, service, and collection registry of digital research tools for scholarly use... and makes it easy for digital humanists and others conducting digital research to find and compare resources ranging from content management systems to music OCR, statistical analysis packages to mindmapping software."

8. **Finding and Preparing Text** (http://hermeneuti.ca/finding-preparing-text) - "How do you find an electronic text of a work you want to study? How should you prepare it for study with a tool like Voyant? This appendix provides a short guide to finding and preparing an electronic text."

9. **Internet Archive API** (http://blog.archive.org/developers/) - "[The] Internet Archive encourages developers to [programmatically] add media to archive.org as well as to consume and repurpose metadata and media."

10. **Log-linear Part-Of-Speech Tagger** (https://nlp.stanford.edu/software/tagger.shtml) - "A Part-Of-Speech Tagger (POS Tagger) is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like "noun-plural". This software is a Java implementation of the log-linear part-of-speech taggers described in [a number of papers]..."

11. **MALLET** (http://mallet.cs.umass.edu/) - "MALLET is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text."

12. **Named Entity Recognizer** (https://nlp.stanford.edu/software/CRF-NER.shtml) - "Stanford NER is a Java implementation of a Named Entity Recognizer. Named Entity Recognition (NER) labels sequences of words in a text which are the names of things, such as person and company names, or gene and protein names. It comes with well-engineered feature extractors for Named Entity Recognition, and many options for defining feature extractors. Included with the download are good named entity recognizers for English, particularly for the 3 classes (PERSON, ORGANIZATION, LOCATION), and we also make available on this page various other models for different languages and circumstances, including models trained on just the CoNLL 2003 English training data."

13. **Natural Language Toolkit** (http://www.nltk.org/) - "NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum."

14. **Notepad++** (https://notepad-plus-plus.org/) - "Notepad++ is a free (as in 'free speech' and also as in 'free beer') source code editor and Notepad replacement that supports several languages. Running in the MS Windows environment, its use is governed by GPL License.""

15. **OpenRefine** (http://openrefine.org/) - "OpenRefine (formerly Google Refine) is a powerful tool for working with messy data: cleaning it; transforming it from one format into another; and extending it with web services and external data."

16. **Tableau Public** (https://public.tableau.com/s/) - "From connection through collaboration, Tableau is the most powerful, secure, and flexible end-to-end analytics platform for your data. Match the skills of any data worker with the capabilities they need. Prepare, create, explore, or view trusted data with subscriptions to Tableau's governed self-service analytics platform. Upgrade to Tableau Creator, a new subscription offering that gives you Tableau Desktop, Tableau Prep, and a server seat."

17. **Tika** (http://tika.apache.org/) - "The Apache Tika™ toolkit detects and extracts metadata and text from over a thousand different file types (such as PPT, XLS, and PDF). All of these file types can be parsed through a single interface, making Tika useful for search engine indexing, content analysis, translation, and much more."

18. **topic-modeling-too**l (https://github.com/senderle/topic-modeling-tool) - "A point-and-click tool for creating and analyzing topic models produced by MALLET."

19. **Unix for Poets** (https://www.cs.upc.edu/~padro/Unixforpoets.pdf) - A gentle introduction to counting & tabulating from the command line.

20. **Voyant Tools** (http://voyant-tools.org/) - "Voyant Tools is a web-based reading and analysis environment for digital texts."

21. **Wget** (https://www.gnu.org/software/wget/) - "GNU Wget is a free software package for retrieving files using HTTP, HTTPS, FTP and FTPS the most widely-used Internet protocols. It is a non-interactive command line tool, so it may easily be called from scripts, cron jobs, terminals without X-Windows support, etc."

Appendix: An Introduction to the NLTK

The venerable Python Natural Language Toolkit (NLTK) is well worth the time of anybody who wants to do text mining more programmatically.[1]

For much of my career, Perl has been the language of choice when it came to processing text, but in the recent past it seems to have fallen out of favor. I really don't know why. Maybe it is because so many other computer languages have some into existence in the past couple of decades: Java, PHP, Python, R, Ruby, Javascript, etc. Perl is more than capable of doing the necessary work. Perl is well-supported, and there are a myriad of supporting tools/libraries for interfacing with databases, indexers, TCP networks, data structures, etc. On the other hand, few people are being introduced to Perl; people are being introduced to Python and R instead. Consequently, the Perl community is shrinking, and the communities for other languages is growing. Writing something in a "dead" language is not very intelligent, but that may be over-stating the case. On the other hand, I'm not going to be able to communicate with very many people if I speak Latin and everybody else is speaking French, Spanish, or German. It behooves the reader to write software in a language apropos to the task as well as a language used by many others.

Python is a good choice for text mining and natural language processing. The Python NLTK provides functionality akin to much of what has been outlined in this workshop, but it goes much further. More specifically, it interfaces with WordNet, a sort of thesaurus on steroids. It interfaces with MALLET, the Java-based classification & topic modeling tool. It is very well-supported and continues to be maintained. Moreover, Python is mature in & of itself. There are a host of Python "distributions/frameworks". There are any number of supporting libraries/modules for interfacing with the Web, databases & indexes, the local file system, etc. Even more importantly for text mining (and natural language processing) techniques, Python is supported by a set of robust machine learning libraries/modules called scikit-learn. If the reader wants to write text mining or natural language processing applications, then Python is really the way to go.

In the etc directory of this workshop's distribution is a "Jupyter Notebook" named "An introduction to the NLTK.ipynb".[2] Notebooks are sort of interactive Python interfaces. After installing Jupyter, the reader ought to be able to run the Notebook. This specific Notebook introduces the use of the NLTK. It walks you through the processes of reading a plain text file, parsing the file into words ("features"). Normalizing the words. Counting & tabulating the results. Graphically illustrating the results. Finding co-occurring words,

[1] Python Natural Language Toolkit - http://nltk.org

[2] Jupyter - http://jupyter.org

words with similar meanings, and words in context. It also dabbles a bit into parts-of-speech and named entity extraction.

The heart of the Notebook's code follows. Given a sane Python installation, one can run this program by saving it with a name like introduction.py, saving a file named walden.txt in the same directory, changing to the given directory, and then running the following command:

```
$ python introduction.py
```

The result ought to be a number of textual outputs in the terminal window as well as a few graphics.

Errors may occur, probably because other Python libraries/modules have not been installed. Follow the error messages' instructions, and try again. Remember, "Your milage may vary."

```
# configure; using an absolute path, define the
# location of a plain text file for analysis
FILE = 'walden.txt'

# import / require the use of the Toolkit
from nltk import *

# slurp up the given file; display the result
handle = open( FILE, 'r')
data   = handle.read()
print( data )

# tokenize the data into features (words); display them
features = word_tokenize( data )
print( features )

# normalize the features to lower case and exclude punctuation
features = [ feature for feature in features if feature.isalpha() ]
features = [ feature.lower() for feature in features ]
print( features )

# create a list of (English) stopwords, and then
# remove them from the features
from nltk.corpus import stopwords
stopwords = stopwords.words( 'english' )
features  = [ feature for feature in features if feature not in stopwords ]

# count & tabulate the features, and then plot the
# results -- season to taste
frequencies = FreqDist( features )
plot = frequencies.plot( 10 )

# create a list of unique words (hapaxes); display them
hapaxes = frequencies.hapaxes()
print( hapaxes )

# count & tabulate ngrams from the features -- season
# to taste; display some
ngrams      = ngrams( features, 2 )
frequencies = FreqDist( ngrams )
frequencies.most_common( 10 )

# create a list each token's length, and plot the result;
# How many "long" words are there?
lengths = [ len( feature ) for feature in features ]
plot    = FreqDist( lengths ).plot( 10 )

# initialize a stemmer, stem the features, count & tabulate,
```

```
# and output
from nltk.stem import PorterStemmer
stemmer     = PorterStemmer()
stems       = [ stemmer.stem( feature ) for feature in features ]
frequencies = FreqDist( stems )
frequencies.most_common( 10 )

# re-create the features and create a NLTK Text object,
# so other cool things can be done
features = word_tokenize( data )
text     = Text( features )

# count & tabulate, again; list a given word -- season to taste
frequencies = FreqDist( text )
print( frequencies[ 'love' ] )

# do keyword-in-context searching against the text (concordancing)
print( text.concordance( 'love' ) )

# create a dispersion plot of given words
plot = text.dispersion_plot( [ 'love', 'war', 'man', 'god' ] )

# output the "most significant" bigrams, considering
# surrounding words (size of window) -- season to taste
text.collocations( num=10, window_size=4 )

# given a set of words, what words are nearby
text.common_contexts( [ 'love', 'war', 'man', 'god' ] )

# list the words (features) most associated with the given word
text.similar( 'love' )

# create a list of sentences, and display one -- season to taste
sentences = sent_tokenize( data )
sentence  = sentences[ 14 ]
print( sentence )

# tokenize the sentence and parse it into parts-of-speech,
# all in one go
sentence = pos_tag( word_tokenize( sentence ) )
print( sentence )

# extract named enities from a sentence, and print the results
entities = ne_chunk( sentence )
print( entities )

# done
quit()
```

About the author

Eric Lease Morgan has been writing software since 1976, and he has been working in libraries since 1984. He has a bachelor's degree in philosophy, and master's degree in information science. Over the years his research interests have included hypermedia, expert systems, information retrieval, recommender systems, information architecture, usability, linked data, and most recently text mining, natural language processing and machine learning. He has written more than a few books and many articles. He has given numerous presentation across North America and Europe. He has been practicing open access publishing and open source software distribution since before the phrases were coined. In his copious spare time, Eric plays guitar and recorder, maintains his outdoor aquarium, rows his boat, paints, does ceramics, and practices bookbinding. He is the sole owner of a tiny company, Infomotions LLC, providing consulting services. For more information, search the Web and the reader will probably learn more about Eric than he wishes to be known.