CHAPTER 1

INTRODUCTION

1.1 Text Data

Text data is basically verbal expression and communication. Words are how most humans form thoughts and communicate them. Text analysis is particularly useful for big data, either thousands of documents, or thousands of words in each document, or both. When the mass of information exceeds what a reader can read comprehensibly, computer-based, algorithmic text analysis using the internet and other technologies is particularly valuable. These big data records, comprised of thousands of documents and millions of words, are now more common than ever before. In this book, we study the corpus of 8,500 letters of the *Territorial Papers*, and another corpus consisting of more than 100,000 speeches given by members of the 39th United States Congress. Constructing larger and still larger data sets permits analysis on even larger degrees of scale.

Other examples of text information include tweets, messages, or emails of public figures, the copious WikiLeaks documents, the political debates, Supreme Court decisions, State of the Union speeches, the complete works of Shakespeare, poems of ancient Roman poets, and the Federalist papers. Many data sets include both text and numeric data. For example, movie and restaurant reviews include both text (the written review) and numeric rating scores (usually from 1 to 5, or 1 to 10) indicating how positive the reviewer feels about the subject. Other kinds of combined text and numeric data are medical chart data, with written notes describing a patient's medical condition and her associated numeric test results; and earnings reports of companies (the text), with associated numbers (e.g., sales, earnings, stock price, and their changes over each reporting interval). When the number of documents is limited and the documents are short, the information can be read and analyzed without needing special tools. But when the number and the length of documents exceed the capacity of readers, one seeks automatic computer-based methods to understand the relevant information.

Text mining can address interesting questions like contested authorships and similarities of genres: Which of Shakespeare's plays are similar, and which are different? Who was the author of certain disputed Federalist papers? Frederick Mosteller and David Wallace spent years combing over the written works of Madison and Hamilton in order to identify patterns in their writings, and they used these linguistic fingerprints to trace the origins

of the disputed Federalist papers back to their original author (Mosteller & Wallace, 1984). Text analysis permits even more nuanced inquiry such as whether the writing styles have changed over time in measurable ways. Are there gender, age, or other demographic differences in writing styles? The range of interesting questions that text analysis can address is virtually limitless.

1.1.1 Introducing the Definitions

Before we get into the mechanics of text analysis, let's start with a couple of definitions and illustrative examples. Understanding the component parts of text data itself requires some precision. Let's look at components of the text.

Text information is available on individual units of observation. The **unit** of observation may be an individual speech, letter, book, poem, movie review, patient record, company earnings report, or tweet. The basic unit of observation is also referred to as the **document**. The very fact that communications are recorded indicates that each unit can be called a document. Text information in each unit is comprised of a **string of words**. Words are typically separated by spaces (blanks). The string may be only a few words or thousands of words long. The collection of the text information from all the documents is referred to as the **corpus**.

Metainformation attached to a unit of information, or a document, refers to the context in which the unit of communication was created. Who authored these words, spoke this message, or chose to push the button that sent the message? When did the communication occur? Where did it occur? Who was the audience? Was the intended audience specified? Each document exists within a certain context, and the details that describe that context is the metadata of the text. Each document can be coded with these tags. One can also code demographic characteristics of a speaker, such as the speaker's age, party affiliation, residence, income level, or gender. One can also code the demographic characteristics of the audience. Coding metadata is useful for further analysis. Metadata may also suggest some category of subject matter, such as, the type of company (size or industry affiliation) described in the earnings report, or the genre of the movie (comedy, documentary, etc.) being reviewed. Actually, any characteristic that is worth breaking out as a category can give rise to a metadata variable that can be coded and then used to split documents into groups for meaningful separate or comparative analysis.

Text information for any given document is comprised of a string of words. The string includes words (e.g., "house", "war", "beautiful", "I", "am", "the", "also", "collusion"), punctuations (e.g., periods and commas),

special characters (e.g., # and &), and numbers. Some words are rare and occur only once in a document or in the entire corpus of documents; other words occur frequently.

If word order is not important, the text information of a document reduces to a collection of distinct words and their associated frequencies. Think about this as the relative word clouds that you have seen. However, when word order matters, one can use bigrams, trigrams, and/or even larger ***n*-grams** to systematically consider what words are adjacent to each other. A bigram is an ordered pair of two adjacent words. Since word order matters, the bigram "united states" is different from the bigram "states united". Any string of words can be represented by its distinct bigrams and their respective frequencies. Of course, this practice can be extended to trigrams (which are ordered three-word groups) or any number of multiples. Be warned that the process gets very complicated very fast as one moves to larger *n*-grams. The complications associated with *n*-grams are (a) their high dimension and (b) their sparsity. There will be a larger number of *n*-grams, but each *n*-gram does not occur very often. These features increase the dimensionality of the problem, thereby complicating the numerical text analysis.

There are useful and easy-to-use tools for dealing with text, even before you get too deeply into the analysis: the **Google Ngram Viewer** and **concordance** and **collocate** tools.

The Google Ngram Viewer (https://books.google.com/ngrams) is an online search engine that can chart the frequencies of any search word (or string of words), using a yearly count of *n*-grams found in sources printed between the year 1500 and the present day and part of Google's text corpora (in English, as well as other major languages). This search engine is a useful tool to learn about the (relative) occurrence of a specified word or phrase among all (Google) digitized books published during any given year. It utilizes the vast libraries of books that Google has digitized and finds the relative frequency of books that contain the phrase. The Google Ngram Viewer is a quick and useful tool for comparing the occurrences and trends of different expressions. You can also select the reference population, choosing either American or British English (or several other major languages). Note that it doesn't matter how often a phrase is mentioned in a given book.

The **key word in context** (commonly abbreviated as KWIC) is a search window for exploring the neighborhood where the key word or key phrase is located. You select a search window of a certain size around the KWIC to find the words immediately preceding and following that key word.

The contextual occurrence of a word is referred to as its concordance. **Concordance tools** provide you with all text snippets around a specified

KWIC. For example, in the Congressional debates that we explore in this book, you may want to see all text snippets in the corpus surrounding a word such as "republic" or around phrases such as "republican form of government" or "post office", as these snippets show you more about the context in which the word or phrase is used.

Collocate tools allow you to further quantify the information about the concordance. Again in the Congressional debates corpus, you may want to obtain counts on how often the words "servant" or "black" occur within a window of specified length around the KWIC word "slave". Free and easy to use software tools, such as **`AntConc`** (http://www.laurenceanthony.net/software.html) and **`Voyant`** (https://voyant-tools.org/) can provide this information. We provide examples in Section 1.4.

The statistical analysis of text data is challenging—certainly more challenging than the analysis of the typical numeric and categorical data encountered in standard statistical applications. Usually, the analysis of textual information starts with words (more generally, *n*-grams) and their frequencies. Next, the analysis factors in any associated metadata on the documents. Text analysis can further explore the relationships between text characteristics and document meta/covariate information. Metavariables that involve time, or space, allow the text data analyst to incorporate time-series and spatial components. Naturally, this complicates the analysis—but, of course, it also makes it more interesting and potentially more meaningful.

1.1.2 Types of Text Data

Some text data that you seek to analyze, particularly recent data, may have been created and collected electronically from the very start, for example, twitter data, earnings reports, or medical records. And if you collect the data yourself, you may want to design a structure for the data collection that takes into account the ease of analysis later. Designing the metadata information to select documents for targeted study will save you time in the long run; see Ledolter and Swersey (2007) for a discussion on useful principles of design of experiments.

But often the text data was created for some purpose other than the text analysis that you want to perform. Thanks to 20th-century uniform typeset, there is a vast amount of printed or typed information that can be transformed into an electronic textual record through a method called optical character recognition (OCR). The results of the OCR process on historical documents are textual data that are typically much noisier, in that they contain misspelled words, and unwanted words, that arise from scanned formats, such as headers, line breaks, page

numbers, and footnotes. Performing OCR on 19th-century printing is even trickier because of the irregular spacing and hand-carved type. Making this noisy data more useful for analysis requires time-consuming text cleaning, either through automatic or manual methods or both. This was a major problem with both of our historical corpora from the 19th century that we describe in Section 1.2. Printed book pages had to be digitized one page at a time. We will show you how we cleaned these noisy data sets in Chapter 3.

1.1.3 File Formats to Save and Store Text Information

Data can be stored in various formats. Here we mention five of them: (1) text, (2) XML, (3) Word, (4) Excel, and (5) PDF. First, because of their simplicity, **text files** (with extension .txt) are commonly used for storage of information that is structured as a sequence of lines of electronic text. Text files are also versatile in that they can be read into many different editing programs. Second, the information can be stored as an extensible markup language, or **XML file** (with extension .xml). XML files can be thought of as a text-based database; they are plain text files but also describe the file structure through custom tags that define objects and the data within each object. XML files open in Microsoft XML Notepad. Notepad++, a particularly useful text and source code editor, supports tabbed editing, which allows working with multiple open files in a single window. Third, the information can be stored as a Microsoft **Word file** (with extension .doc or .docx), or fourth, as a Microsoft **Excel file** (with extension .xls) or a Microsoft **Excel comma-separated** (also called, **comma-delimited**) **file** (with extension .csv). CSV is a simple file format used to store tabular data, such as a spreadsheet. The difference between the .csv and .xls file formats is that the .csv format is a plain text format in which values are separated by commas, while the .xls file format is a binary file format that holds information about all the worksheets in a file, including both content and formatting. Finally, files can be stored as Adobe Portable Document Format or **PDF files** (with extension .pdf). PDF files are easily viewed and manipulated using the free Acrobat Reader DC software. Because PDFs don't rely on the software that created them, nor do they depend on any particular operating system or hardware, they look the same no matter what device they're opened on.

Each formatting/storing system carries its own advantages and drawbacks; however, multiple tools are available to convert text data from one format to the other. We found that text files served our purposes best, but use the system that works for you.

1.2 The Two Applications Considered in This Book

The two applications in this book were selected because they were both representative and challenging. Each one represented a different social dynamic of communication.

One corpus, *The Territorial Papers of the United States*, contains the correspondence between the government and inhabitants of the U.S. territories over six decades from 1790 to 1850. The corpus of the *Territorial Papers* contains 8,500 letters of correspondence among ordinary and influential individuals during the westward expansion of the United States. These letters were not necessarily a sustained correspondence because they usually took weeks to reach their intended audience through the mail, and even longer to receive a response.

The U.S. *Territorial Papers* analyzed in Ledolter and Vandervelde (2019) cover 15 expertly curated volumes containing 8,500 entries (letters) written to and from Washington, D.C., the center of national power, and the nation's territories. The letters are arranged chronologically and regionally beginning in 1789 with the collective correspondence about the original Northwest Territory and ending in 1848 with the correspondence about Wisconsin (Volume 28, the last volume transcribed and published by Congress). The chronology and the regional coverage concerns the formation of new states to enter the union. We included all Northern states and those states west of the Mississippi river that joined before 1850. The coverage includes the regions from which the states of Indiana, Illinois, Louisiana, Missouri, Michigan, Wisconsin, and Iowa were formed. Analysis of this data reveals several things, first what the territorial residents want from their government, and what the government wants from its territories. Second, analyzing these letters can also reveal the sentiment of categories of letters; for example, whether the collective sentiment differs in letters that originate from the center of power or from the periphery of the expanding empire. Finally, analyzing these data can tell us about the relative importance of the topics that the letters address and how the discourse changed over time.

The dynamics of the other corpus, the Congressional debates from 1865 to 1867, were very different. The *Congressional Globe* (now called the *Congressional Record*), comprises all speeches given during a certain congress. We chose the 39th U.S. Congress, which met from 1865 to 1867, just after the Civil War. These texts are the transcripts of exactly what was said on the floor of each house as taken down by stenographers. There were more than 100,000 entries during these 2 years; some were brief remarks, but some were speeches lasting more than an hour. The purpose of this orally spoken word was to communicate directly to others in the chamber

in the hopes of persuading others and to ultimately pass laws. The purpose of the transcription and printing was ultimately to communicate with the American public, as the transcribed speeches were published, and disseminated throughout the nation. Only a discrete set of elected congressmen, no more than 200, and a handful of clerks were permitted to speak on the floor. Sometimes the congressmen were addressing members of the public in the galleries. Although parliamentary rules of order governed the overall process, responses were often immediate, speakers were frequently interrupted, speeches often got emotional and heated, and sometimes laughter broke out. All these utterances were faithfully recorded by stenographers.

The 39th U.S. Congress, meeting in Washington, D.C., from March 1865 to March 1867, covers an extremely active period of legal and Constitutional reform. The Thirteenth Amendment abolishing slavery had just been passed, the South had surrendered, and President Lincoln had just been assassinated and succeeded by the unpopular Vice President Andrew Johnson. In this study, we are interested in the language related to civil rights, in particular race, liberty, slavery, equality, governance and citizenship, and the occurrence of words such as "white", "black", "freedmen", "negro", "labor", "slaves", and so on. We are interested to learn which congressmen address which topics, whether the speakers who do so share common characteristics, and who were the leaders in the discussion. We are also interested in finding related words that were commonly used during the debates and whether certain topics dominated certain time periods.

We provide more detailed discussion and appraisal of these two digital data sets in Chapter 2.

You will undoubtedly want to select your very own text corpora and adjust the methods of text analysis to your specific interests. We provide several other text corpora that you can use for practice on the website for Chapter 11, including Donald Trump's 44,000 tweets, covering the period from May 2009 to December 2019. This data set includes text as well as emojis (pictorial representations of emphasis and sentiment). The date of each tweet is also available.

1.3 Introductory Example and Its Analysis Using the R Statistical Software

To show you a simple introductory example in detail, we consider just *four* speeches from the hundreds of thousands of speeches given during the 39th U.S. Congress. The text of all four short speeches is given below. The *Congressional Globe* gives the speaker's name (Representatives Thaddeus Stevens, James Brooks, and Philip Johnson, and the Clerk of the House of

Representatives) at the beginning of each speech. The speaker's name is the metavariable and must be separated from the textual information. We stored the information in four rows of a comma-delimited Excel file, **test.csv**. Each row represents a different speech. You can find the file on the book's website for Chapter 1.

```
Mr. STEVENS. The gentleman cannot have anything to
explain. As he has not spoken there is nothing to
explain, nothing to patch up.

Mr. BROOKS. I do not see the relevancy of that remark.
I desire to know, Mr. Clerk, first, whether I can yield
the floor temporarily to the gentleman from Pennsylvania
for the purpose of explanation; or second, whether I can
yield the remaining portion of my time to him.

The CLERK. Under the rules of the House the gentleman
from New York cannot yield the remaining portion of his
time to any member if objection is made to his doing so;
nor can he yield to any other member, except for purposes
of personal explanation in relation to the pending
proposition. The gentleman from Pennsylvania has stated
that he did not rise for the purpose of explanation.
Hence by his own statement he is precluded from taking
the floor.

Mr. JOHNSON. I do not desire the floor for the purpose
of personal explanation, but I desire to as an explanation
of the gentleman from New York.
```

The word file **ProgramTest** (which also can be found on our website for Chapter 1) includes the computer code that is needed to read in the information, to separate the metavariables from the text, and to analyze the text. In this book, we use the R computer software for analysis. The R Project for Statistical Computing (https://www.r-project.org/) provides a free software environment for statistical computing and graphics that includes many useful methods for text mining and the visualization of text information.

This book includes computer code quite sparingly for several reasons: The book's main focus is on the concepts of text analysis. The book describes and explains useful methods, discusses their appropriateness, and summarizes what can be learned from such analysis. As such, the discussion of concepts and methods should not be tied too tightly—too dependent on—to the software or software version that carries out the analysis. Computer code changes rapidly, and code that works well today is often outdated tomorrow.

But software is critical to carrying out text analysis, especially with large corpora comprised of millions of words. We want our readers to be able to use these methods and to gain valuable hands-on experience. The book will be much more useful if the reader can put these methods to use. This is the reason why you will find most of the code we use (the R code) on the book's accompanying website **https://www.biz.uiowa.edu/faculty/jledolter/analyzing-textual-information.** You can cut and paste snippets of the code to adapt our programs to your own analysis. The book and website are closely linked, with the overarching goal to make methods of textual analysis useful to the reader. Some readers may already be familiar with the R software, and for you, the coding in R may be easy. But for other readers, it may be your first exposure to R. The website contains R tutorials, links to supplementary materials, errata sheets, and our communication with you, the reader. The information on the website becomes a "living" document as compared with the printed book that intends to explain durable concepts but can only be frozen as of the time of publication.

For you to become more familiar with text data, we show slightly more R-code in this chapter than in the following chapters. We want to give you, the reader, a taste of the text data and the analysis tools. Note that R instructions are shown in boldface. The R output is indented and shown in regular font.

First, we input the data. We use the R function **`read.csv`** to input the information from a comma-delimited Excel file into a data frame, with rows corresponding to cases (documents, speeches) and columns (here there is just a single column) corresponding to the text and the metavariable. We strip out punctuation, periods, commas, and semicolons from the text, and we insert a hyphen to differentiate the state New York from the two adjoining words, "new" and "york". Next, we extract the metavariable (the name of the speaker), remove the metavariable from the text, and count the number of tokens (that is, the word length) of each speech. Note that the number of tokens in a speech is larger than the number of distinct words (as some words are repeated). The first speech, given by Representative Stevens, has length 21—that is, 21 words.

Here we apply useful R functions such as **`gsub`** (global text substitution to replace snippets of text), **`strsplit`** (to split a character string into words), and **`toString`** (to convert objects into a character string). These are useful functions for preprocessing text, which we need here for extracting the speakers. Otherwise, we could have skipped these steps and could have gone directly to creating the corpus. The website describes this more fully. You can also use the R help window to learn what these functions do and how to use them. We recommend that you double and triple check your code by trying it out on smaller chunks of your corpus, like we did for the four speeches, and convince

yourself that the code you are using is fit for more general analysis. Note that R uses the symbol ">" as the prompt for instructions. Text following "##" is ignored and is only there to comment on the code.

```
> ## read and clean data
>  data  =  read.csv('C:\\Users\\ledolter\\Desktop\\test.csv',
header=FALSE,stringsAsFactors=F)
> ## Note that we read the data from the directory C:\\Users\\
ledolter\\Desktop
> dim(data)
       [1] 4 1
> data[1:4,1]
       [1] "Mr. STEVENS. The gentleman cannot have anything to
       explain. As he has not spoken there is nothing to
       explain, nothing to patch up."
       [2] "Mr. BROOKS. I do not see the relevancy of that
       remark. I desire to know, Mr. Clerk, first, whether I
       can yield the floor temporarily to the gentleman from
       Pennsylvania for the purpose of explanation; or second,
       whether I can yield the remaining portion of my time to
       him."
       [3] "The CLERK. Under the rules of the House the
       gentleman from New York cannot yield the remaining
       portion of his time to any member if objection is made
       to his doing so; nor can he yield to any other member,
       except for purposes of personal explanation in relation
       to the pending proposition. The gentleman from
       Pennsylvania has stated that he did not rise for the
       purpose of explanation. Hence by his own statement he
       is precluded from taking the floor."
       [4] "Mr. JOHNSON. I do not desire the floor for the
       purpose of personal explanation, but I desire to as an
       explanation of the gentleman from New York."
> dim(data)[1] ## number of speeches
       [1] 4
> for (i in 1:dim(data)[1]) {
+ txt=data[i,1]
+ txt=tolower(txt)
+ txt=gsub("[.]","", ignore.case = TRUE,txt)
+ txt=gsub("[,]","", ignore.case = TRUE,txt)
+ txt=gsub("[;]","", ignore.case = TRUE,txt)
+  txt=gsub("new york","new-york", ignore.case = TRUE,txt) ##
the state of new york
+ data[i,1]=txt
+ }
> data[1:4,1]
       [1] "mr stevens the gentleman cannot have anything to
       explain as he has not spoken there is nothing to explain
       nothing to patch up"
       . . . .
```

```
        [4] "mr johnson i do not desire the floor for the
        purpose of personal explanation but i desire to as an
        explanation of the gentleman from new-york"
> ## omit meta variables from the text
> ## speaker in meta2; determine length of each speech
> ## works if there are no missing values in meta2
> len=dim(dim(data)[1])
> meta1=dim(dim(data)[1])
> meta2=dim(dim(data)[1])
> for (i in 1:dim(data)[1]) {
+ txt=data[i,1]
+ temp=strsplit(txt,"")[[1]]
+ len[i]=length(temp)-2
+ meta1[i]=temp[1]
+ meta2[i]=temp[2]
+ tempr=dim(len[i])
+ for (j in 1:len[i]) {
+ tempr[j]=temp[j+2]
+ }
+ data[i,1]=toString(tempr)
+ data[i,1]=gsub("[,]","", ignore.case = TRUE,data[i,1])
+ }
> data[1:4,1]
        [1] "the gentleman cannot have anything to explain as
        he has not spoken there is nothing to explain nothing
        to patch up"
        . . . .
        [4] "i do not desire the floor for the purpose of
        personal explanation but i desire to as an explanation
        of the gentleman from new-york"
> len
        [1] 21 47 77 24
> hist(len)
> boxplot(len)
> quantile(len)
      0%    25%    50%    75%   100%
   21.00  23.25  35.50  54.50  77.00
> meta2
        [1] "stevens" "brooks"  "clerk"   "johnson"
```

Next, we use the R package **tm** to create the corpus. The function **VCorpus** creates the corpus infrastructure necessary for the analysis. We strip all white space and transfer everything to lower case. We remove punctuation and numbers (this is not needed here as we already did that when preprocessing the text). We also omit stopwords. Stopwords are "filler" words that usually have little textual meaning, and these words are stripped from the text. You can check which words are included in the file stopwords("english"). If you want to add more stopwords or create your own stopwords, we show how you can do so easily.

We decided against **stemming** words. We leave the words as they are. Stemming strips off suffixes and some other word endings. You may already be familiar with stemming in search engines such as Hein on-line or Westlaw. The goal of stemming is to reduce inflectional forms and derivationally related forms of a word to a common base form, the word's stem. Stemming uses a heuristic process to identify and chop off the ends of words. The most common and empirically effective algorithm for stemming English is Porter's algorithm (Porter, 1980). The entire algorithm is long and intricate, consisting of several phases of word reductions applied sequentially. Within each phase, there are various conventions to select systematically among rules—for example, selecting the rule from each rule group that applies to the longest suffix.

At this point, the words and their frequencies are summarized in the **document-term matrix**, a large matrix with row dimension given by the number of documents and column dimension given by the number of distinct terms (words) in the corpus.

In our small four-speech example here, the document-term matrix is a matrix of four rows, and 43 columns because there are 43 distinct words appearing in the entire corpus of these four speeches, after having removed all stopwords. Note that very short words, words of length 2 or less, are automatically omitted. Also note that the word "new-york" has become "newyork" as we removed all punctuations when preprocessing the corpus. Many entries in the document-term frequency matrix are zero as words do not appear in every document.

Our document-term matrix has $4 \times 43 = 172$ cells: 61 of the cells have nonzero frequencies, while 111 cells are empty (zero frequencies). The sparsity of the matrix is expressed as the proportion of sparse elements (elements with zero frequency); here the sparsity is $100(111/172) = 65\%$. A sparsity of zero means that every word in the corpus occurs in every document.

Similarly, we can obtain the **term-document matrix**, the matrix transpose of the document-term matrix. Its row dimension is given by the number of distinct words in the corpus and its column dimension is given by the number of documents.

The document-term matrix helps us illustrate what stemming does because the matrix allows us to compare the words in our corpus without and with stemming. Stemming collapses the singular "purpose" and plural "purposes" into one word—which is excellent. However, stemming reduces the word "stated" to "state", which is not so good because the verb "stated" is not the same as the noun "state". Furthermore, stemming strips off "ing", so the stemmed word "noth" (from "nothing") and "pend" (from "pending") are difficult to interpret. Similarly, "explan", the stemmed version of "explanation", is unfortunately not merged with "explain". This simple comparison demonstrates that there is no easy way to decide whether to stem or not—each approach has its advantages and disadvantages.

We chose to leave words as they are but combine singular and plural words into the same root word. We try to do this for as many important words as possible. This can be done in two different ways. (1) One can substitute expressions in the text prior to creating the corpus and the document-term matrix. One can use the global text substitution command **gsub**: for example, the command txt=gsub(“ purposes ”,“ purpose ”, ignore.case = TRUE, txt) to globally replace all occurrences of the plural “purposes” with “purpose”, or the command txt=gsub(“ treaties ”, “ treaty ”, ignore.case = TRUE, txt) to replace the plural “treaties” with “treaty”. Make sure that you leave blank spaces before and after the expressions. Otherwise this command will also replace snippets within a larger word. (2) One can combine terms in the document-term matrix, using the function **combine_terms** that we wrote for this purpose. It’s on the website at https://www.biz.uiowa.edu/faculty/jledolter/analyzing-textual-information/.

Word frequencies are obtained and visualized, either through **bar-charts** or **word-clouds**. The bar chart shown here displays frequencies of words that occur at least three times in the corpus. The two different word clouds shown draw the size of the words proportional to the frequency of their occurrences.

The R code on the website will get you started. We recommend that you follow our template and use the R help window to learn about the R functions that we used. The more you practice, the better you will get writing your own code.

```
> library(tm) ## library tm needs to be installed (instructions
on WebAppendix)
> ## creating corpus
>  corpus  =  VCorpus(VectorSource(data[,1]),readerControl  =
list(reader = readPlain))
## this is how to create corpus
> corpus1 = tm_map(corpus, stripWhitespace)
> corpus2 = tm_map(corpus1, content_transformer(tolower))
> corpus3 = tm_map(corpus2, removePunctuation)
> corpus4 = tm_map(corpus3, removeNumbers)
> corpus5 = tm_map(corpus4, removeWords, stopwords("english"))
> corp.dtm = DocumentTermMatrix(corpus5,control=list(stemming=
FALSE))
## no stemming is the default
> corp.dtm
       <<DocumentTermMatrix (documents: 4, terms: 43)>>
       Non-/sparse entries : 61/111
       Sparsity            : 65%
       Maximal term length : 12
       Weighting           : term frequency (tf)
> corp.tdm = TermDocumentMatrix(corpus5,control=list(stemming=
FALSE))
```

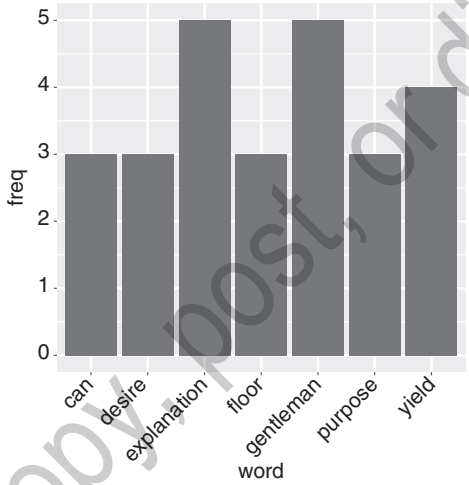
```
> corp.tdm
        <<TermDocumentMatrix (terms: 43, documents: 4)>>
        Non-/sparse entries : 61/111
        Sparsity            : 65%
        Maximal term length : 12
        Weighting           : term frequency (tf)
> corps.dtm = DocumentTermMatrix(corpus5,control=list(stemming
=TRUE))
> corps.dtm
        <<DocumentTermMatrix (documents: 4, terms: 42)>>
        Non-/sparse entries : 60/108
        Sparsity            : 64%
        Maximal term length : 12
        Weighting           : term frequency (tf)
> findFreqTerms(corp.dtm,1)
         [1] "anything"  "can"   "clerk"   "desire"  "except"
        . . .
> findFreqTerms(corps.dtm,1)
         [1] "anyth"     "can"   "clerk"   "desir"   "except"
        . . . .
> ## creating corpus
>
> stopwords("english")
         [1] "i"      "me"      "my"          "myself"   "we"
         [6] "our"    "ours"    "ourselves"   "you"      "your"
        . . . .
> ## adding your own stopwords
> stopwordsnew1=c(stopwords("english"),"occasionally")
##adding "occasionally"
> stopwordsnew2=c("perhaps","never")
> stopwordsnew2
        [1] "perhaps" "never"
>
> ## frequencies of words
> ## displaying frequencies
> library(ggplot2)
> dim(corp.dtm)
         [1] 4 43
> as.matrix(corp.dtm)
             Terms
        Docs anything can clerk desire except explain
           1        1   0     0      0      0       2
           2        0   2     1      1      0       0
           3        0   1     0      0      1       0
           4        0   0     0      2      0       0
        explanation first floor gentleman
                  0     0     0         1
                  1     1     1         1
                  2     0     1         2
                  2     0     1         1
```

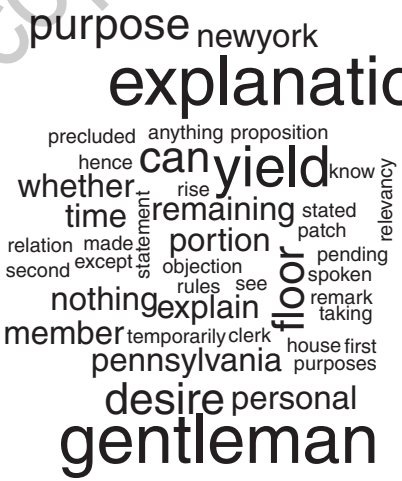
```
    Terms
Docs hence house know made member newyork
   1     0     0    0    0      0       0
   2     0     0    1    0      0       0
   3     1     1    0    1      2       1
   4     0     0    0    0      0       1
nothing objection patch pending
      2         0     1       0
      0         0     0       0
      0         1     0       1
      0         0     0       0
. . . .

> findFreqTerms(corp.dtm,1)
> findFreqTerms(corp.dtm,2)
 [1] "explain" "gentleman" "nothing" "can" "desire"
 [6] "explanation" "floor" "pennsylvania" "portion"
"purpose"
[11] "remaining" "time" "whether" "yield" "member"
[16] "newyork" "personal"
> freq=colSums(as.matrix(corp.dtm))
> ord=order(freq)
> freq[head(ord)]
anything    patch   spoken    clerk    first     know
       1        1        1        1        1        1
> freq[tail(ord)]
  desire    floor  purpose    yield gentleman explanation
       3        3        3        4        5           5
> freq=sort(colSums(as.matrix(corp.dtm)),decreasing=TRUE)
> head(freq,20)
gentleman explanation    yield      can   desire    floor
        5           5        4        3        3        3
purpose  explain  nothing pennsylvania  portion remaining
      3        2        2            2        2         2
   time  whether   member      newyork personal anything
      2        2        2            2        2         1
  patch   spoken
      1        1
> wf=data.frame(word=names(freq),freq=freq)
> head(wf)
                word freq
gentleman   gentleman    5
explanation explanation  5
yield         yield      4
can             can      3
desire       desire      3
floor         floor      3
> p=ggplot(subset(wf,freq>2),aes(word,freq))
> p=p+geom_bar(stat="identity")
> p=p+theme(axis.text.x=element_text(angle=45,hjust=1))
```

```
> p
> ## displaying frequencies

> ## displaying word clouds
> library(wordcloud)
> set.seed(142)
> wordcloud(names(freq),freq,min.freq=1)
> set.seed(142)
> dark2 = brewer.pal(6,"Dark2")
> wordcloud(names(freq),freq,max.words=7,rot.per=0.2,
  colors=dark2)
> ## displaying word clouds
> ## frequencies of words
```





We also assess whether the number of occurrences of one word (e.g., the word "explanation" or the word "gentleman") is related to (that is, correlated with) the number of occurrences of any of the other words in the very same document. Results with correlation +1 indicate that there is a positive and perfect linear association among the frequencies of the two words. Take the words "gentleman" and "except" for illustration. The frequencies in the four documents are 1 1 2 1 (for "gentleman") and 0 0 1 0 (for "except"). The frequencies are perfectly correlated: (frequencies for "gentleman") = 1 + 1 ∗ (frequencies for "except"). Hence, the correlation is 1. Of course, one should not read too much into correlations from just a few documents (like the four speeches in our example).

Often, the R function **`weightBin`** is useful. It transforms the number of occurrences (the frequency) of a word in a given document to an occurrence indicator of a word; that is, it creates a 0–1 indicator variable indicating whether the word has not (has) occurred in a given document.

```
> ## finding associations
> as.matrix(corp.dtm)
      Terms
Docs anything can clerk desire except explain
   1        1   0     0      0      0       2
   2        0   2     1      1      0       0
   3        0   1     0      0      1       0
   4        0   0     0      2      0       0
explanation first floor gentleman
          0     0     0         1
          1     1     1         1
          2     0     1         2
          2     0     1         1
. . . .
```

```
> findAssocs(corp.dtm, "explanation", 0.5)
$explanation
newyork       personal     floor   purpose     except  gentleman
0.90              0.90      0.87      0.87       0.52       0.52
hence            house      made    member  objection    pending
0.52              0.52      0.52      0.52       0.52       0.52
precluded  proposition  purposes  relation       rise      rules
0.52              0.52      0.52      0.52       0.52       0.52
stated       statement    taking
0.52              0.52      0.52
> findAssocs(corp.dtm, "gentleman", 0.5)
$gentleman
except       hence       house      made    member     objection
1.00          1.00        1.00      1.00      1.00          1.00
pending  precluded  proposition  purposes  relation          rise
1.00          1.00        1.00      1.00      1.00          1.00
rules       stated   statement    taking  newyork  pennsylvania
1.00          1.00        1.00      1.00      0.58          0.58
personal   portion   remaining      time     yield   explanation
0.58          0.58        0.58      0.58      0.58          0.52
> ## weightBin creates indicator variables for presence of term
> Bcorp.dtm=weightBin(corp.dtm)
> as.matrix(Bcorp.dtm)
           Terms
        Docs anything can clerk desire except explain
           1        1   0     0      0      0       1
           2        0   1     1      1      0       0
           3        0   1     0      0      1       0
           4        0   0     0      1      0       0
        explanation first floor gentleman
                  0     0      0         1
                  1     1      1         1
                  1     0      1         1
                  1     0      1         1
          . . . .
> findAssocs(Bcorp.dtm, "explanation", 0.5)
$explanation
        floor      purpose  can    desire   pennsylvania  portion
        1.00       1.00     0.58   0.58     0.58          0.58
        remaining  time     yield  newyork  personal
        0.58       0.58     0.58   0.58     0.58
> findAssocs(Bcorp.dtm, "gentleman", 0.5)
        $gentleman
        numeric(0)
> ## finding associations
```

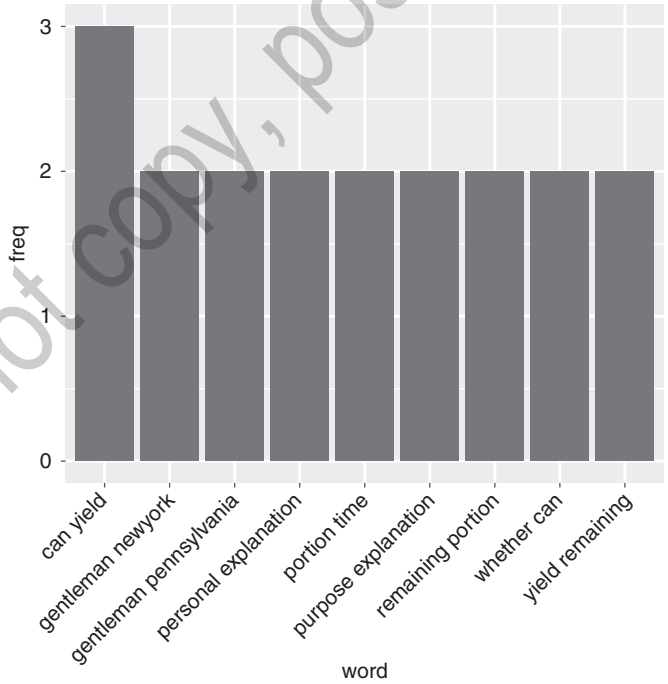
The document-term matrix shows the frequencies of individual words, and it does so for each document separately. What about frequencies of two-word phrases? What about bigrams? The R code shown below with

the function **BigramTokenizer** creates the document-term and the term-document matrices that contain frequencies of adjacent word combinations for each of the four documents. Here, there are 59 different bigrams. Bigrams that occur at least two times in the corpus are displayed in the attached bar chart. The extension to trigrams, which are three-word phrases, is straightforward. The complication with *n*-grams (as compared with the case of individual words) is the resulting large number of *n*-grams, with most (almost all) of the *n*-grams occurring with very low frequencies. This increases both the dimensionality and also the sparsity.

```
> ## bigrams
> BigramTokenizer = function(x)
+      unlist(lapply(ngrams(words(x),2),paste,collapse = " "),
use.names = FALSE)
> bi.dtm = DocumentTermMatrix(corpus5, control = list(tokenize
= BigramTokenizer))
> bi.dtm
        <<DocumentTermMatrix (documents: 4, terms: 59)>>
        Non-/sparse entries: 67/169
        Sparsity           : 72%
        Maximal term length: 22
        Weighting          : term frequency (tf)
> as.matrix(bi.dtm)
             Terms
        Docs anything explain can yield clerk first
           1                1         0           0
           2                0         2           1
           3                0         1           0
           4                0         0           0
        desire explanation desire floor
                         0            0
                         0            0
                         0            0
                         1            1
             Terms
        Docs desire know except purposes explain nothing
           1           0                0                1
           2           1                0                0
           3           0                1                0
           4           0                0                0
        explain spoken
                      1
                      0
                      0
                      0
        . . . .
> bi.tdm = TermDocumentMatrix(corpus5, control = list(tokenize
= BigramTokenizer))
```

```
> bi.tdm
       <<TermDocumentMatrix (terms: 59, documents: 4)>>
       Non-/sparse entries : 67/169
       Sparsity            : 72%
       Maximal term length : 22
       Weighting           : term frequency (tf)
> as.matrix(bi.tdm)
                                  Docs
       Terms                   1 2 3 4
          anything explain     1 0 0 0
          can yield            0 2 1 0
          clerk first          0 1 0 0
          desire explanation   0 0 0 1
          desire floor         0 0 0 1
          desire know          0 1 0 0
          . . . .
          time member          0 0 1 0
          whether can          0 2 0 0
          yield floor          0 1 0 0
          yield member         0 0 1 0
          yield remaining      0 1 1 0
> ## bigrams
>
> ## displaying bigram frequencies
> findFreqTerms(bi.dtm,1)
       [1] "anything explain" "can yield" "clerk first"
       [4] "desire explanation" "desire floor" "desire know"
       [7] "except purposes" "explain nothing" "explain
       spoken"
       . . . .
       [55] "time member" "whether can" "yield floor"
       [58] "yield member" "yield remaining"
> findFreqTerms(bi.dtm,2)
       [1] "can yield" "gentleman newyork" "gentleman
       pennsylvania"
       [4] "personal explanation" "portion time" "purpose
       explanation"
       [7] "remaining portion" "whether can" "yield remaining"
> freq=colSums(as.matrix(bi.dtm))
> ord=order(freq)
> freq[head(ord)]
anything explain          clerk first   desire explanation
               1                    1                    1
    desire floor          desire know     except purposes
               1                    1                    1
> freq[tail(ord)]
    portion time   purpose explanation    remaining portion
               2                    2                    2
     whether can       yield remaining           can yield
               2                    2                    3
```

```
> freq=sort(colSums(as.matrix(bi.dtm)),decreasing=TRUE)
> head(freq,20)
          can yield  gentleman newyork gentleman pennsylvania
                  3                  2                      2
personal explanation       portion time     purpose explanation
                  2                  2                      2
       . . . .
> wf=data.frame(word=names(freq),freq=freq)
> head(wf)
                                                      word freq
can yield                                     can yield    3
gentleman newyork                   gentleman newyork      2
gentleman pennsylvania gentleman pennsylvania              2
personal explanation     personal explanation              2
portion time                        portion time           2
purpose explanation       purpose explanation              2
>
> p=ggplot(subset(wf,freq>1),aes(word,freq))
> p=p+geom_bar(stat="identity")
> p=p+theme(axis.text.x=element_text(angle=45,hjust=1))
> p
> ## displaying bigram frequencies
```



1.4 The Introductory Example Revisited, Illustrating Concordance and Collocation Using Alternative Software

We chose the R Statistical Software for most of our textual analyses. However, we are eclectic in utilizing other software packages as long as they get the job done. The packages **AntConc** and **Voyant** are especially useful for their concordance and collocate tools. Concordance tools tell you about all text snippets around a specified key word in context (KWIC). You may want to see all text snippets in a corpus around a word such as "republic", or around phrases such as "republican form of government" or "post office", as these text selections will tell you about the context in which a word or phrase is used. **Collocate tools** allow you to further quantify the concordance information. For example, they can obtain counts on how often the word "black" or "indian" occurs within a window of specified length around the KWIC word "slave". Both software packages, **AntConc** and **Voyant**, are quite flexible, allowing you to change the length of the window and focus on words before or after the given KWIC word of interest.

We use the package **AntConc** to illustrate such analysis on the four speeches in the Excel file **test.csv**. The file is easily uploaded. The concordance tab allows us to select a KWIC word; we select "gentleman". We specify a search window size of 50 characters (letters), and ask the software to highlight the words that are within two words from the KWIC word "gentleman". We get five hits, and the five text snippets are displayed below. The KWIC word is in boldface, and the highlighted words around them are underlined:

1 Mr. Stevens. The **gentleman** cannot have anything

1 the rules of the House the **gentleman** from New York cannot

2 desire to as an explanation of the **gentleman** from New York.

3 to the pending proposition. The **gentleman** from Pennsylvania has

4 yield the floor temporarily to the **gentleman** from Pennsylvania

The collocates tab with the search word "gentleman" and with a specified window that asks for one word to the left and one word to the right reveals three collocates (neighboring words). The word "the" with five frequencies (5 to left and 0 to right), "from" with four frequencies (0 to left and 4 to the right), and "cannot" with one frequency (0 to left and 1 to the right). Changing the window to two words to the right and two words to the left gives us several more collocates for the KWIC "gentleman" such as

"Pennsylvania" with two frequencies (both to the right of the word "gentleman").

Concordance and collocates can also be displayed with the software package **Voyant. Voyant** has many other useful features such as automatic ways of omitting stopwords (e.g., "the" and "from") that are left in the **AntConc** output shown above. We refer the reader to the additional materials shown on our website at https://www.biz.uiowa.edu/faculty/jledolter/analyzing-textual-information/.

The analysis for this small example is straightforward. But imagine carrying out this analysis for a corpus with millions of words. Getting concordance information and collocates so quickly is a wonderful feature of these programs. Where do these terms appear in the text and which words are around them? Which are the most frequent words that co-occur in a neighborhood around a certain word and how can one get a word cloud for these given words? How does word usage change with author and date in case the metavariables author and date have been collected? The user wants an easy intuitive search engine to learn about a text corpus and easy ways for visualizing the information.

Two other useful software packages (with our apologies for having omitted many others) are as follows:

PhiloLogic4, developed at the University of Chicago Textual Optics Lab (https://artfl-project.uchicago.edu/philologic4). Go to https://artfl-project.github.io/PhiloLogic4 and explore this software using one of its many available data bases.

Distant Reader: A Tool for Reading, developed at the University of Notre Dame (https://distantreader.org). It transforms unstructured text into structured text and conducts useful analyses involving words and word counts and *n*-grams. It also parses the text through parts of speech (POS) analysis and determines automatically whether a word is a noun, verb, adjective, and so on.

POS tagging, also referred to as the parsing of text, is the process of relating each word in a text to a particular part of speech. In English, one distinguishes nine major POS: noun, verb, article, adjective, preposition, pronoun, adverb, conjunction, and interjection. However, there are clearly many more categories and subcategories. On nouns, for example, one can further distinguish the plural, possessive, and singular forms, and one can mark nouns for their "case" (role as subject, object, etc.), grammar, and so on. Verbs can be marked for tense, aspect, and other things. A tag set is a list of labels that describe the part of speech and its associated grammatical categories (case, tense, etc.) of words in a text corpus.

Tagging text "by hand" is not only tedious and slow, it also is not that easy because some POS are complex and some words can represent more than one part of speech at different times.

Once performed by hand, POS tagging is now carried out with computer algorithms. The field that deals with the set of methods that make human language accessible to computers is commonly referred to as **natural language processing**. The first major English corpus tagged by a computer algorithm is the Brown University Corpus developed in the mid-1960s. The corpus consists of complete sentences and more than 100,000 words, and a large tag set with almost 100 different tags. A somewhat simpler tag set for POS tagging of American English text is the Penn tag set, developed by the Penn Treebank project. Many algorithms for computer-based POS tagging are available today, and most achieve an accuracy above 95%.

1.5 Concluding Remarks

The analysis for the four-speech example in Section 1.3 is fairly simple: The number of words is not very large, the text is clean, and none of the words are misspelled. Nevertheless, even analyzing this simple case involved many decisions: The pair "new york" is recognized as the state and the two terms are joined. The words, "purpose" and "purposes" are recognized as the single and the plural forms of the same noun. We conclude that these two terms may be better combined during the next round of analysis, and we show how this can be done. Now imagine a much, much larger corpus, with millions of words and lots of misspellings. A single pass through the text can never be the end of the analysis; it cannot provide insightful solutions to vaguely formulated theories about the text. Any learning has to proceed iteratively, an approach that involves observing the results of each pass-through and refining the inquiry each time according to previous findings. Several (usually, many) iterative passes through the text are needed.

Statisticians have long been aware of this in the study of numbers. They customarily screen the data for unusual observations, and they transform and group data whenever needed. Processes in text analysis work the same way. Decisions about how to analyze the text data become clearer once the analysis has started. However, there is a big difference: Text has such a high number of dimensions that the discovery process is more difficult and time-consuming. But the challenge is rewarding.

1.6 References

Ledolter, J., & Swersey, A. J. (2007). *Testing 1–2–3: Experimental design with applications in marketing and service operations.* Stanford University Press.

Ledolter, J., & VanderVelde, L. (2019). A case study in text mining: Textual analysis of the *Territorial Papers*. *Digital Scholarship in the Humanities, 35*(1), 101–126. https://doi.org/10.1093/llc/fqz007

Mosteller, F., & Wallace, D. L. (1984). *Applied Bayesian and classical inference: The case of the Federalist Papers* (2nd ed.). Springer.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program, 14*(3), 130−137. https://doi.org/10.1108/eb046814

The R Project for Statistical Computing. https://www.r-project.org/

Voyant Tools: Open-source, web-based application for performing text analysis. https://voyant-tools.org/

AntConc: A freeware corpus analysis toolkit for concordancing and text analysis. https://www.laurenceanthony.net/software/antconc/

Distant Reader: A Tool for Reading: Developed at the University of Notre Dame; https://distantreader.org/

PhiloLogic4: Developed at the University of Chicago Textual Optics Lab; https://artfl-project.uchicago.edu/philologic4/