11-2017

Text analysis in R

Kasper WELBERS

Wouter van Atteveldt

Kenneth BENOIT
Singapore Management University, kbenoit@smu.edu.sg

Citation

WELBERS, Kasper, van Atteveldt, Wouter, & BENOIT, Kenneth.(2017). Text analysis in R. *Communication Methods and Measures, 11(4)*, 245-265.
Available at: https://ink.library.smu.edu.sg/soss_research/3978

COMMUNICATION METHODS AND MEASURES
2017, VOL. 11, NO. 4, 245–265
https://doi.org/10.1080/19312458.2017.1387238

Text Analysis in R

Kasper Welbers[a], Wouter Van Atteveldt[b], and Kenneth Benoit[c]

[a]Institute for Media Studies, University of Leuven, Leuven, Belgium; [b]Department of Communcation Science, VU University Amsterdam, Amsterdam, The Netherlands; [c]Department of Methodology, London School of Economics and Political Science, London, UK

ABSTRACT
Computational text analysis has become an exciting research field with many applications in communication research. It can be a difficult method to apply, however, because it requires knowledge of various techniques, and the software required to perform most of these techniques is not readily available in common statistical software packages. In this teacher's corner, we address these barriers by providing an overview of general steps and operations in a computational text analysis project, and demonstrate how each step can be performed using the R statistical software. As a popular open-source platform, R has an extensive user community that develops and maintains a wide range of text analysis packages. We show that these packages make it easy to perform advanced text analytics.

With the increasing importance of computational text analysis in communication research (Boumans & Trilling, 2016; Grimmer & Stewart, 2013), many researchers face the challenge of learning how to use advanced software that enables this type of analysis. Currently, one of the most popular environments for computational methods and the emerging field of "data science"[1] is the R statistical software (R Core Team, 2017). However, for researchers that are not well-versed in programming, learning how to use R can be a challenge, and performing text analysis in particular can seem daunting. In this teacher's corner, we show that performing text analysis in R is not as hard as some might fear. We provide a step-by-step introduction into the use of common techniques, with the aim of helping researchers get acquainted with computational text analysis in general, as well as getting a start at performing advanced text analysis studies in R.

R is a free, open-source, cross-platform programming environment. In contrast to most programming languages, R was specifically designed for statistical analysis, which makes it highly suitable for data science applications. Although the learning curve for programming with R can be steep, especially for people without prior programming experience, the tools now available for carrying out text analysis in R make it easy to perform powerful, cutting-edge text analytics using only a few simple commands. One of the keys to R's explosive growth (Fox & Leanage, 2016; TIOBE, 2017) has been its densely populated collection of extension software libraries, known in R terminology as *packages*, supplied and maintained by R's extensive user community. Each package extends the functionality of the base R language and core packages, and in addition to functions and data must include documentation and examples, often in the form of vignettes demonstrating the use of the package. The best-known package repository, the Comprehensive R Archive Network (CRAN), currently has over 10,000 packages that are published, and which have gone through an extensive

CONTACT Kasper Welbers kasperwelbers@gmail.com Institute for Media Studies, University of Leuven, Sint-Andriesstraat 2 – box 15530, Antwerp 2000, Belgium.

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/hcms.

[1]The term "data science" is a popular buzzword related to "data-driven research" and "big data" (Provost & Fawcett, 2013).

screening for procedural conformity and cross-platform compatibility before being accepted by the archive.[2] R thus features a wide range of inter-compatible packages, maintained and continuously updated by scholars, practitioners, and projects such as RStudio and rOpenSci. Furthermore, these packages may be installed easily and safely from within the R environment using a single command. R thus provides a solid bridge for developers and users of new analysis tools to meet, making it a very suitable programming environment for scientific collaboration.

Text analysis in particular has become well established in R. There is a vast collection of dedicated text processing and text analysis packages, from low-level string operations (Gagolewski, 2017) to advanced text modeling techniques such as fitting Latent Dirichlet Allocation models (Blei, Ng, & Jordan, 2003; Roberts et al., 2014) — nearly 50 packages in total at our last count. Furthermore, there is an increasing effort among developers to cooperate and coordinate, such as the rOpenSci special interest group.[3] One of the main advantages of performing text analysis in R is that it is often possible, and relatively easy, to switch between different packages or to combine them. Recent efforts among the R text analysis developers' community are designed to promote this interoperability to maximize flexibility and choice among users.[4] As a result, learning the basics for text analysis in R provides access to a wide range of advanced text analysis features.

Structure of this Teacher's Corner

This teacher's corner covers the most common steps for performing text analysis in R, from data preparation to analysis, and provides easy to replicate example code to perform each step. The example code is also digitally available in our online appendix, which is updated over time.[5] We focus primarily on bag-of-words text analysis approaches, meaning that only the frequencies of words per text are used and word positions are ignored. Although this drastically simplifies text content, research and many real-world applications show that word frequencies alone contain sufficient information for many types of analysis (Grimmer & Stewart, 2013).

Table 1 presents an overview of the text analysis operations that we address, categorized in three sections. In the *data preparation* section we discuss five steps to prepare texts for analysis. The first step, *importing text*, covers the functions for reading texts from various types of file formats (e.g., txt, csv, pdf) into a *raw text* corpus in R. The steps *string operations* and *preprocessing* cover techniques for manipulating raw texts and processing them into *tokens* (i.e., units of text, such as words or word stems). The tokens are then used for creating the *document-term matrix* (DTM), which is a common format for representing a bag-of-words type corpus, that is used by many R text analysis packages. Other non-bag-of-words formats, such as the tokenlist, are briefly touched upon in the *advanced topics* section. Finally, it is a common step to *filter and weight* the terms in the DTM. These steps are generally performed in the presented sequential order (see Figure 1 for conceptual illustration). As we will show, there are R packages that provide convenient functions that manage multiple data preparation steps in a single line of code. Still, we first discuss and demonstrate each step separately to provide a basic understanding of the purpose of each step, the choices that can be made and the pitfalls to watch out for.

The *analysis* section discusses four text analysis methods that have become popular in communication research (Boumans & Trilling, 2016) and that can be performed with a DTM as input. Rather than being competing approaches, these methods have different advantages and disadvantages, so choosing the best method for a study depends largely on the research question, and

[2] Other programming environments have similar archives, such as pip for python. However, CRAN excels in how it is strictly maintained, with elaborate checks that packages need to pass before they will be accepted.

[3] The London School of Economics and Political Science recently hosted a workshop (http://textworkshop17.ropensci.org/), forming the beginnings of an rOpenSci special interest group for text analysis.

[4] For example, the tif (Text Interchange Formats) package (rOpenSci Text Workshop, 2017) describes and validates standards for common text data formats.

[5] https://github.com/kasperwelbers/text_analysis_in_R.

Table 1. An overview of text analysis operations, with the R packages used in this Teacher's Corner.

	R packages	
Operation	example	alternatives
Data preparation		
importing text	*readtext*	*jsonlite, XML, antiword, readxl, pdftools*
string operations	*stringi*	*stringr*
preprocessing	*quanteda*	*stringi, tokenizers, snowballC, tm, etc.*
document-term matrix (DTM)	*quanteda*	*tm, tidytext, Matrix*
filtering and weighting	*quanteda*	*tm, tidytext, Matrix*
Analysis		
dictionary	*quanteda*	*tm, tidytext, koRpus, corpustools*
supervised machine learning	*quanteda*	*RTextTools, kerasR, austin*
unsupervised machine learning	*topicmodels*	*quanteda, stm, austin, text2vec*
text statistics	*quanteda*	*koRpus, corpustools, textreuse*
Advanced topics		
advanced NLP	*spacyr*	*coreNLP, cleanNLP, koRpus*
word positions and syntax	*corpustools*	*quanteda, tidytext, koRpus*

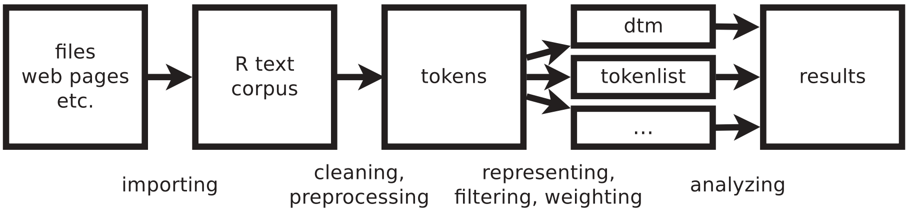

Figure 1. Order of text analysis operations for data preparation and analysis.

sometimes different methods can be used complementarily (Grimmer & Stewart, 2013). Accordingly, our recommendation is to become familiar with each type of method. To demonstrate the general idea of each type of method, we provide code for typical analysis examples. Furthermore, It is important to note that different types of analysis can also have different implications for how the data should be prepared. For each type of analysis we therefore address general considerations for data preparation.

Finally, the additional *advanced topics* section discusses alternatives for data preparation and analysis that require external software modules or that go beyond the bag-of-words assumption, using word positions and syntactic relations. The purpose of this section is to provide a glimpse of alternatives that are possible in R, but might be more difficult to use.

Within each category we distinguish several groups of operations, and for each operation we demonstrate how they can be implemented in R. To provide parsimonious and easy to replicate examples, we have chosen a specific selection of packages that are easy to use and broadly applicable. However, there are many alternative packages in R that can perform the same or similar operations. Due to the open-source nature of R, different people from often different disciplines have worked on similar problems, creating some duplication in functionality across different packages. This also offers a range of choice, however, providing alternatives to suit a user's needs and tastes. Depending on the research project, as well as personal preference, other packages might be better suited to different readers. While a fully comprehensive review and comparison of text analysis packages for R is beyond our scope here—especially given that existing and new packages are constantly being developed—we have tried to cover, or at least mention, a variety of alternative packages for each text analysis operation.[6] In general, these

[6] For a list that includes more packages, and that is also maintained over time, a good source is the CRAN Task View for Natural Language Processing (Wild, 2017). *CRAN Task Views* are expert curated and maintained lists of R packages on the Comprehensive R Archive Network, and are available for various major methodological topics.

packages often use the same standards for data formats, and thus are easy to substitute or combine with the other packages discussed in this teacher's corner.

Data preparation

Data preparation is the starting point for any data analysis. Not only is computational text analysis no different in this regard, but also frequently presents special challenges for data preparation that can be daunting for novice and advanced practitioners alike. Furthermore, preparing texts for analysis requires making choices that can affect the accuracy, validity, and findings of a text analysis study as much as the techniques used for the analysis (Crone, Lessmann, & Stahlbock, 2006; Günther & Quandt, 2016; Leopold & Kindermann, 2002). Here we distinguish five general steps: importing text, string operations, preprocessing, creating a document-term matrix (DTM), and filtering and weighting the DTM.

Importing text

Getting text into R is the first step in any R-based text analytic project. Textual data can be stored in a wide variety of file formats. R natively supports reading regular flat text files such as CSV and TXT, but additional packages are required for processing formatted text files such as JSON (Ooms, 2014), HTML, and XML (Lang & the CRAN Team, 2017), and for reading complex file formats such as Word (Ooms, 2017a), Excel (Wickham & Bryan, 2017) and PDF (Ooms, 2017b). Working with these different packages and their different interfaces and output can be challenging, especially if different file formats are used together in the same project. A convenient solution for this problem is the *readtext* package, that wraps various import packages together to offer a single catch-all function for importing many types of data in a uniform format. The following lines of code illustrate how to read a CSV file with the readtext function, by providing the path to the file as the main argument (the path can also be an URL, as used in our example; online appendix with copyable code available from https://github.com/kasperwelbers/text_analysis_in_R).

```r
install.packages(“readtext”)
library(readtext)

## url to Inaugural Address demo data that is provided by the readtext package
filepath <- "http://bit.ly/2uhqjJE?.csv"

rt <- readtext(filepath, text_field = “texts”)
rt
```

```
readtext object consisting of 5 documents and 3 docvars.
# data.frame [5 x 5]
         doc_id                      text     Year     President   FirstName
          <chr>                     <chr>    <int>         <chr>       <chr>
1  2uhqjJE?.csv.1   "\"Fellow-Cit\"..."     1789    Washington      George
2  2uhqjJE?.csv.2   "\"Fellow cit\"..."     1793    Washington      George
3  2uhqjJE?.csv.3   "\"When it wa\"..."     1797         Adams        John
4  2uhqjJE?.csv.4   "\"Friends an\"..."     1801     Jefferson      Thomas
5  2uhqjJE?.csv.5   "\"Proceeding\"..."     1805     Jefferson      Thomas
```

The same function can be used for importing all formats mentioned above, and the path can also reference a (zip) folder to read all files within. In most cases, the only thing that has to be specified is the name of the field that contains the texts. Not only can multiple files be references using simple, “glob”-style pattern matches, such as `~/myfiles/*.txt`, but also the same command will recurse through sub-directories to locate these files. Each file is automatically imported according to its format, making it very easy to import and work with data from different input file types.

Another important consideration is that texts can be represented with different character encodings. Digital text requires binary code to be mapped to semantically meaningful characters, but many different such mappings exist, with widely different methods of encoding “extended” characters, including letters with diacritical marks, special symbols, and emoji. In order to be able to map all known characters to a single scheme, the Unicode standard was proposed, although it also requires a digital encoding format (such as the UTF-8 format, but also UTF-16 or UTF-32). Our recommendation is simple: in R, ensure that all texts are encoded as UTF-8, either by reading in UTF-8 texts, or converting them from a known encoding upon import. If the encoding is unknown, *readtext’s encoding* function can be used to guess the encoding. *readtext* can convert most known encodings (such as ISO-8859-2 for Central and Eastern European languages, or Windows-1250 for Cyrillic—although there are hundreds of others) into the common UTF-8 standard. R also offers additional low-level tools for converting character encodings, such as a bundled version of the GNU *libiconv* library, or conversion though the *stringi* package.

String operations

One of the core requirements of a framework for computational text analysis is the ability to manipulate digital texts. Digital text is represented as a sequence of characters, called a string. In R, strings are represented as objects called “character” types, which are vectors of strings. The group of string operations refers to the low-level operations for working with textual data. The most common string operations are joining, splitting, and extracting parts of strings (collectively referred to as *parsing*) and the use of *regular expressions* to find or replace patterns.

Although R has numerous built-in functions for working with character objects, we recommend using the *stringi* package (Gagolewski, 2017) instead. Most importantly, because *stringi* uses the International Components for Unicode (ICU) library for proper Unicode support, such as implementing Unicode character categories (such as punctuation or spacing) and Unicode-defined rules for case conversion that work correctly in all languages. An alternative is the *stringr* package, which uses *stringi* as a backend, but has a simpler syntax that many end users will find sufficient for their needs.

It is often unnecessary to perform manual, low-level string operations, because the most important applications of string operations for text analysis are built into common text analysis packages. Nevertheless, access to low-level string operations provides a great deal of versatility, which can be crucial when standardized solutions are not an option for a specific use case. The following example shows how to perform some basic cleaning with *stringi* functions: removing boilerplate content in the form of markup tags, stripping extraneous whitespace, and converting to lower case.

```
library(stringi)
x <- c(“The first string”, ’ The <font size="6">second string</font>’)

x <- stri_replace_all(x, “”, regex = “<.*?>”)   ## remove html tags
x <- stri_trim(x)                               ## strip surrounding whitespace
x <- stri_trans_tolower(x)                      ## transform to lower case
x
```

```
[1] "the first string"  "the second string"
```

As with most functions in R, *stringi* operations are *vectorized*, meaning they apply to each element of a vector. Manipulation of vectors of strings is the recommended approach in R, since looping over each element and processing it separately in R is very inefficient.

Preprocessing

For most computational text analysis methods, full texts must be *tokenized* into smaller, more specific text features, such as words or word combinations. Also, the computational performance and accuracy of many text analysis techniques can be improved by normalizing features, or by removing "stopwords": words designated in advance to be of no interest, and which are therefore discarded prior to analysis. Taken together, these preparatory steps are commonly referred to as "preprocessing". Here we first discuss several of the most common preprocessing techniques, and show how to perform each technique with the quanteda package.

In practice, all of these preprocessing techniques can be applied in one function when creating a document-term matrix, as we will demonstrate in the *DTM* section. Here, we show each step separately to illustrate what each technique does.

Tokenization

Tokenization is the process of splitting a text into tokens. This is crucial for computational text analysis, because full texts are too specific to perform any meaningful computations with. Most often tokens are words, because these are the most common semantically meaningful components of texts.

For many languages, splitting texts by words can mostly be done with low-level string processing due to clear indicators of word boundaries, such as white spaces, dots and commas. A good tokenizer, however, must also be able to handle certain exceptions, such as the period in the title "Dr.", which can be confused for a sentence boundary. Furthermore, tokenization is more difficult for languages where words are not clearly separated by white spaces, such as Chinese and Japanese. To deal with these cases, some tokenizers include dictionaries of patterns for splitting texts. In R, the *stringi* package is often used for sentence and word disambiguation, for which it leverages dictionaries from the ICU library. There is also a dedicated package for text tokenization, called *tokenizers* (Mullen, 2016b).

The following code uses *quanteda*'s (Benoit et al., 2017) `tokens` function to split a single sentence into words. The tokens function returns a list whose elements each contain the tokens of the input texts as a character vector.

```r
install.packages("quanteda")
library(quanteda)

text <- "An example of preprocessing techniques"
toks <- tokens(text)  ## tokenize into unigrams
toks
```

```
tokens from 1 document.
text1 :
[1] "An"   "example"   "of"   "preprocessing"   "techniques"
```

Normalization: Lowercasing and stemming

The process of normalization broadly refers to the transformation of words into a more uniform form. This can be important if for a certain analysis a computer has to recognize when two words have (roughly) the same meaning, even if they are written slightly differently. Another advantage is that it reduces the size of the vocabulary (i.e., the full range of features used in the analysis). A simple but important normalization techniques is to make all text lower case. If we do not perform this transformation, then a computer will not recognize that two words are identical if one of them was capitalized because it occurred at the start of a sentence.

Another argument for normalization is that a base word might have different morphological variations, such as the suffixes from conjugating a verb, or making a noun plural. For purposes of analysis, we might wish to consider these variations as equivalent because of their close semantic relation, and because reducing the feature space is generally desirable when multiple features are in fact closely related. A technique for achieving this is *stemming*, which is essentially a rule-based algorithm that converts inflected forms of words into their base forms (stems). A more advanced technique is *lemmatization*, which uses a dictionary to replace words with their morphological root form. However, lemmatization in R requires external software modules (see the *advanced preprocessing* section for instructions) and for weakly inflected languages such as modern English, stemming is often sufficient. In R, the SnowballC (Bouchet-Valat, 2014; Porter, 2001) package is used in many text analysis packages (such as *quanteda* and *tm*) to implement stemming, and currently supports 15 different languages. Lowercasing and stemming of character, tokens, or feature vectors can be performed in *quanteda* with the `*_tolower` and `*_wordstem` functions, such as `char_tolower` to convert character objects to lower case, or `tokens_wordstem` to stem tokens.

```r
toks <- tokens_tolower(toks)
toks <- tokens_wordstem(toks)
toks
```

```
[1] "an"   "exampl"   "of"   "preprocess"   "techniqu"
```

In this example we see that the difference between "an" and "An" is eliminated due to lowercasing. The words "example" and "techniques" are reduced to "exampl" and "techniqu", such that any distinction between singular and plural forms is removed.

Removing stopwords

Common words such as "the" in the English language are rarely informative about the content of a text. Filtering these words out has the benefit of reducing the size of the data, reducing computational load, and in some cases also improving accuracy. To remove these words beforehand, they are matched to predefined lists of "stop words" and deleted. Several text analysis packages provide stopword lists for various languages, that can be used to manually filter out stopwords. In *quanteda*, the `stopwords` function returns a character vector of stopwords for a given language. A total of 17 languages are currently supported.

```r
sw <- stopwords("english")   ## get character vector of stopwords
head(sw)                     ## show head (first 6) stopwords
```

```
[1] "i"   "me"   "my"   "myself"   "we"   "our"
```

```r
tokens_remove(toks, sw)
```

```
text1 :
[1] "exampl"    "preprocess"    "techniqu"
```

Care should be taken to perform some preprocessing steps in the correct order, for instance removing stopwords prior to stemming, otherwise "during" will be stemmed into "dure" and not matched to a stopword "during". Case conversion may also create sequencing issues, although the default stopword matching used by *quanteda* is case-insensitive.

Conveniently, the preprocessing steps discussed above can all be performed with a single function that will automatically apply the correct order of operations. We will demonstrate this in the next section.

Document-term matrix

The document term matrix (DTM) is one of the most common formats for representing a text corpus (i.e. a collection of texts) in a bag-of-words format. A DTM is a matrix in which rows are documents, columns are terms, and cells indicate how often each term occurred in each document. The advantage of this representation is that it allows the data to be analyzed with vector and matrix algebra, effectively moving from text to numbers. Furthermore, with the use of special matrix formats for sparse matrices, text data in a DTM format is very memory efficient and can be analyzed with highly optimized operations.

Two of the most established text analysis packages in R that provide dedicated DTM classes are *tm* and *quanteda*. Of the two, the venerable *tm* package is the more commonly used, with a user base of almost 10 years (Meyer, Hornik, & Feinerer, 2008) and several other R packages using its DTM classes (*DocumentTermMatrix* and *TermDocumentMatrix*) as inputs for their analytic functions. The *quanteda* package is a more recently developed package, built by a team supported by an ERC grant to provide state-of-the-art, high performance text analysis. Its sparse DTM class, known as a `dfm` or *document-feature matrix*, is based on the powerful *Matrix* package (Bates & Maechler, 2015) as a backend, but includes functions to convert to nearly every other sparse document-term matrix used in other R packages (including the *tm* formats). The performance and flexibility of *quanteda*'s `dfm` format lends us to recommend it over the *tm* equivalent.

Another notable alternative is the *tidytext* package (Silge & Robinson, 2016). This is a text analysis package that is part of the *Tidyverse*[7]—a collection of R packages with a common philosophy and format. Central to the Tidyverse philosophy is that all data is arranged as a table, where (1) "each variable forms a column", (2) "each observation forms a row", and (3) "each type of observational unit forms a table" (Wickham et al., 2014, p. 4). As such, *tidytext* does not strictly use a document term *matrix*, but instead represents the same data in a long format, where each (non-zero) value of the DTM is a row with the columns document, term, and count (note that this is essentially a triplet format for sparse matrices, with the columns specifying the row, column and value). This format can be less memory efficient and make matrix algebra less easily applicable, but has the advantage of being able to add more variables (e.g., a sentiment score) and enables `the` use of the entire Tidyverse arsenal. Thus, for users that prefer the tidy data philosophy, *tidytext* can be a good alternative package to quanteda or tm, although these packages can also be used together quite nicely depending on the particular operations desired.

Consistent with the other examples in this teacher's corner, we demonstrate the creation of DTMs using the *quanteda* package. Its `dfm` function provides a single line solution for creating a DTM from raw text, that also integrates the preprocessing techniques discussed above. These may also be built up through a sequence of lower-level functions, but many users find it convenient to go straight from a text or corpus to a DTM using this single function.

[7] http://www.tidyverse.org/.

```
text <-  c(d1 = "An example of preprocessing techniques",
           d2 = "An additional example",
           d3 = "A third example")
dtm <- dfm(text,                           ## input text
           tolower = TRUE, stem = TRUE,    ## set lowercasing and stemming to TRUE
           remove = stopwords("english"))  ## provide the stopwords for deletion
dtm
```

```
Document-feature matrix of: 3 documents, 5 features (53.3\% sparse).
3 x 5 sparse Matrix of class "dfmSparse"
          features
docs        exampl preprocess techniqu   addit  third

  d1            1           1        1       0      0
  d2            1           0        0       1      0
  d3            1           0        0       0      1
```

The DTM can also be created from a *quanteda* `corpus` object, which stores text and associated meta-data, including document-level variables. When a `corpus` is tokenized or converted into a DTM, these document-level variables are saved in the object, which can be very useful later when the documents in the DTM need to be used as covariates in supervised machine learning. The stored document variables also make it possible to aggregate *quanteda* objects by groups, which is extremely useful when texts are stored in small units—like Tweets—but need to be aggregated in a DTM by grouping variables such as users, dates, or combinations of these.

Because *quanteda* is compatible with the *readtext* package, creating a corpus from texts on disk takes only a single additional step. In the following example we create a DTM from the *readtext* data as imported above.

```
fulltext <- corpus(rt)                              ## create quanteda corpus
dtm <- dfm(fulltext, tolower = TRUE, stem = TRUE,   ## create dtm with preprocessing
           remove_punct = TRUE,remove = stopwords("english"))
dtm
```

```
Document-feature matrix of: 5 documents, 1,405 features (67.9% sparse).
```

Filtering and weighting

Not all terms are equally informative for text analysis. One way to deal with this is to remove these terms from the DTM. We have already discussed the use of stopword lists to remove very common terms, but there are likely still other common words and this will be different between corpora. Furthermore, it can be useful to remove very rare terms for tasks such as category prediction (Yang & Pedersen, 1997) or topic modeling (Griffiths & Steyvers, 2004). This is especially useful for improving efficiency, because it can greatly reduce the size of the vocabulary (i.e., the number of unique terms), but it can also improve accuracy. A simple but effective method is to filter on document frequencies (the number of documents in which a term occurs), using a threshold for minimum and maximum number (or proportion) of documents (Griffiths & Steyvers, 2004; Yang & Pedersen, 1997).

Instead of removing less informative terms, an alternative approach is assign them variable weights. Many text analysis techniques perform better if terms are weighted to take an estimated information value into account, rather than directly using their occurrence frequency. Given a sufficiently large corpus, we can use information about the distribution of terms in the corpus to estimate this information value. A popular weighting scheme that does so is term frequency-inverse document frequency (*tf-idf*), which down-weights that occur in many documents in the corpus.

Using a document frequency threshold and weighting can easily be performed on a DTM. *quanteda* includes the functions `docfreq`, `tf`, and `tfidf`, for obtaining document frequency, term frequency, and *tf-idf* respectively. Each function has numerous options for implementing the SMART weighting scheme Manning et al. (2008). As a high-level wrapper to these, *quanteda* also provides the `dfm_weight` function. In the example below, the word "senat[e]" has a higher weight than the less informative term "among", which both occur once in the first document.

```
doc_freq <- docfreq(dtm)         ## document frequency per term (column)
dtm <- dtm[, doc_freq >= 2]      ## select terms with doc_freq >= 2
dtm <- dfm_weight(dtm, "tfidf")  ## weight the features using tf-idf
head(dtm)
```

```
Document-feature matrix of: 5 documents, 524 features (46.6% sparse).
(showing first 5 documents and first 6 features)
                  features
docs              fellow-citizen   senat     hous     repres         :       among
  2uhqjJE?.csv.1      0.2218487 0.39794  0.79588  0.4436975 0.2218487  0.09691001
  2uhqjJE?.csv.2      0.0000000 0.00000  0.00000  0.0000000 0.2218487  0.00000000
  2uhqjJE?.csv.3      0.6655462 0.39794  1.19382  0.6655462 0.0000000  0.38764005
  2uhqjJE?.csv.4      0.4436975 0.00000  0.00000  0.2218487 0.2218487  0.09691001
  2uhqjJE?.csv.5      0.0000000 0.00000  0.00000  0.0000000 0.0000000  0.67837009
```

Analysis

For an overview of text analysis approaches we build on the classification proposed by Boumans and Trilling (2016) in which three approaches are distinguished: *counting and dictionary* methods, *supervised machine learning*, and *unsupervised machine learning*. They position these approaches, in this order, on a dimension from most deductive to most inductive. Deductive, in this scenario, refers to the use of an *a priori* defined coding scheme. In other words, the researchers know beforehand what they are looking for, and only seek to automate this analysis. The relation to the concept of deductive reasoning is that the researcher assumes that certain rules, or premises, are true (e.g., a list of words that indicates positive sentiment) and thus can be applied to draw conclusions about texts. Inductive, in contrast, means here that instead of using an a priori coding scheme, the computer algorithm itself somehow extracts meaningful codes from texts. For example, by looking for patterns in the co-occurrence of words and finding latent factors (e.g., topics, frames, authors) that explain these patterns—at least mathematically. In terms of inductive reasoning, it can be said that the algorithm creates broad generalizations based on specific observations.

In addition to these three categories, we also consider a *statistics* category, encompassing all techniques for describing a text or corpus in numbers. Like unsupervised learning, these techniques are inductive in the sense that no *a priori* coding scheme is used, but they do not use machine learning.

For the example code for each type of analysis, we use the Inaugural Addresses of US presidents (N = 58) that is included in the *quanteda* package.

```
dtm <- dfm(data_corpus_inaugural, stem = TRUE, remove = stopwords("english"),
                                  remove_punct = TRUE)
dtm
```

```
Document-feature matrix of: 58 documents, 5,405 features (89.2% sparse).
```

Counting and dictionary

The dictionary approach broadly refers to the use of patterns—from simple keywords to complex Boolean queries and regular expressions—to count how often certain concepts occur in texts. This is a deductive approach, because the dictionary defines *a priori* what codes are measured and how, and this is not affected by the data.[8] Using dictionaries is a computationally simple but powerful approach. It has been used to study subjects such as media attention for political actors (Schuck, Xezonakis, Elenbaas, Banducci, & De Vreese, 2011; Vliegenthart, Boomgaarden, & Van Spanje, 2012) and framing in corporate news (Schultz, Kleinnijenhuis, Oegema, Utz, & Van Atteveldt, 2012). Dictionaries are also a popular approach for measuring sentiment (De Smedt & Daelemans, 2012; Mostafa, 2013; Taboada, Brooke, Tofiloski, Voll, & Stede, 2011) as well as other dimensions of subjective language (Tausczik & Pennebaker, 2010). By combining this type of analysis with information from advanced NLP techniques for identifying syntactic clauses, it also becomes possible to perform more fine-grained analyses, such as sentiment expressions attributed to specific actors (Van Atteveldt, 2008), or actions and affections from one actor directed to another (Van Atteveldt, Sheafer, Shenhav, & Fogel-Dror, 2017).

The following example shows how to apply a dictionary to a quanteda DTM. The first step is to create a dictionary object (here called myDict), using the dictionary function. For simplicity our example uses a very simple dictionary, but it is is also possible to import large, pre-made dictionaries, including files in other text analysis dictionary formats such as LIWC, Wordstat, and Lexicoder. Dictionaries can also be written and imported from YAML files, and can include patterns of fixed matches, regular expressions, or the simpler “glob” pattern match (using just `*` and `?` for wildcard characters) common in many dictionary formats. With the `dfm_lookup` function, the dictionary object can then be applied on a DTM to create a new DTM in which columns represent the dictionary codes.

```
myDict <- dictionary(list(terror = c("terror*"),
                          economy = c("job*", "business*", "econom*")))
dict_dtm <- dfm_lookup(dtm, myDict, nomatch = "_unmatched")
tail(dict_dtm)
```

```
Document-feature matrix of: 58 documents, 3 features (37.4% sparse).
(showing last 6 documents and last 3 features)
                     features
docs                  terror   economy      _unmatched
   1997-Clinton            2         3            1125
   2001-Bush               0         2             782
   2005-Bush               0         1            1040
   2009-Obama              1         7            1165
   2013-Obama              0         6            1030
   2017-Trump              1         5             709
```

Supervised machine learning

The supervised machine learning approach refers to all classes of techniques in which an algorithm learns patterns from an annotated set of training data. The intuitive idea is that these algorithms can learn how to code texts if we give them enough examples of how they should be coded. A straightforward example is sentiment analysis, using a set of texts that are manually coded as *positive*, *neutral*, or *negative*, based on which the algorithm can learn which features (words or word combinations) are more likely to occur in positive or negative texts. Given an unseen text (from which the algorithm was not trained), the sentiment of the text can then be estimated based on

[8] Notably, there are techniques for automatically expanding a dictionary based on the semantic space of a text corpus (see, e.g., Watanabe, 2017). This can be said to add an inductive layer to the approach, because the coding rules (i.e., the dictionary) are to some extent learned from the data.

the presence of these features. The deductive part is that the researchers provide the training data, which contains good examples representing the categories that the researchers are attempting to predict or measure. However, the researchers do not provide explicit rules for how to look for these codes. The inductive part is that the supervised machine learning algorithm learns these rules from the training data. To paraphrase a classic syllogism: if the training data is a list of people that are either mortal or immortal, then the algorithm will learn that all men are extremely likely to be mortal, and thus would estimate that Socrates is mortal as well.

To demonstrate this, we train a model to predict whether an Inaugural Address was given before World War II—which we expect because prominent issues shift over time, and after wars in particular. Some dedicated packages for supervised machine learning are *RTextTools* (Jurka, Collingwood, Boydstun, Grossman, & Van Atteveldt, 2014) and *kerasR* (Arnold, 2017b). For this example, however, we use a classifier that is included in *quanteda*. Before we start, we set a custom seed for R's random number generator so that the results of the random parts of the code are always the same. To prepare the data, we add the document (meta) variable `is_prewar` to the DTM that indicates which documents predate 1945. This is the variable that our model will try to predict. We then split the DTM into training (`train_dtm`) and test (`test_dtm`) data, using a random sample of 40 documents for training and the remaining 18 documents for testing. The training data is used to train a multinomial Naive Bayes classifier (Manning et al., 2008, Ch. 13) which we assign to `nb_model`. To test how well this model predicts whether an Inaugural Address predates the war, we predict the code for the test data, and make a table in which the rows show the prediction and the columns show the actual value of the `is_prewar` variable.

```r
set.seed(2)
## create a document variable indicating pre or post war
docvars(dtm, "is_prewar") <- docvars(dtm, "Year") < 1945

## sample 40 documents for the training set and use remaining (18) for testing
train_dtm <- dfm_sample(dtm, size = 40)
test_dfm <- dtm[setdiff(docnames(dtm), docnames(train_dtm)), ]

## fit a Naive Bayes multinomial model and use it to predict the test data
nb_model <- textmodel_NB(train_dtm, y = docvars(train_dtm, "is_prewar"))
pred_nb <- predict(nb_model, newdata = test_dtm )

## compare prediction (rows) and actual is_prewar value (columns) in a table
table(prediction = pred_nb$nb.predicted, is_prewar = docvars( test_dtm , "is_prewar"))
```

```
               is_prewar
prediction     FALSE   TRUE
       FALSE       8      0
       TRUE        0     10
```

The results show that the predictions are perfect. Of the eight times that FALSE (i.e. Inaugural Address does not predate the war) was predicted, and the 10 times that TRUE was predicted, this was actually the case.

Unsupervised machine learning

In unsupervised machine learning approaches, no coding rules are specified and no annotated training data is provided. Instead, an algorithm comes up with a model by identifying certain patterns in text. The only influence of the researcher is the specification of certain parameters, such as the number of categories into which documents are classified. Popular examples are topic modeling for automatically classifying documents based on an underlying topical structure (Blei et al., 2003; Roberts et al., 2014) and the “Wordfish” parametric factor model (Proksch & Slapin, 2009) for scaling documents on a single underlying dimension, such as left-right ideology.

Grimmer and Stewart (2013) argue that supervised and unsupervised machine learning are not competitor methods, but fulfill different purposes and can very well be used to complement each other. Supervised methods are the most suitable approach if documents need to be placed in predetermined categories, because it is unlikely that an unsupervised method will yield a categorization that reflects these categories and how the researcher interprets them. The advantage of the somewhat unpredictable nature of unsupervised methods is that it can come up with categories that the researchers had not considered. (Conversely, this may also present challenges for post-hoc interpretation when results are unclear.)

To demonstrate the essence of unsupervised learning, the example below shows how to fit a topic model in R using the *topicmodels* package (Grun & Hornik, 2011). To focus more specifically on topics within the inaugural addresses, and to increase the number of texts to model, we first split the texts by paragraph and create a new DTM. From this DTM we remove terms with a document frequency of five and lower to reduce the size of the vocabulary (less important for current example) and use *quanteda*’s convert function to convert the DTM to the format used by *topicmodels*. We then train a vanilla LDA topic model (Blei et al., 2003) with five topics—using a fixed seed to make the results reproducible, since LDA is non-deterministic.

```r
install.packages("topicmodels")
library(topicmodels)

texts = corpus_reshape(data_corpus_inaugural, to = "paragraphs")

par_dtm <- dfm(texts, stem = TRUE,              ## create a document-term matrix
               remove_punct = TRUE, remove = stopwords("english"))
par_dtm <- dfm_trim(par_dtm, min_count = 5)     ## remove rare terms
par_dtm <- convert(par_dtm, to = "topicmodels") ## convert to topicmodels format

set.seed(1)
lda_model <- topicmodels::LDA(par_dtm, method = "Gibbs", k = 5)
terms(lda_model, 5)
```

```
         Topic 1       Topic 2      Topic 3       Topic 4       Topic 5
[1,]    "govern"      "nation"      "great"          "us"       "shall"
[2,]     "state"         "can"        "war"       "world"     "citizen"
[3,]     "power"        "must"      "secur"         "new"       "peopl"
[4,] "constitut"       "peopl"    "countri"    "american"        "duti"
[5,]       "law"       "everi"       "unit"     "america"     "countri"
```

The results show the first five terms of the five topics. Although this is a basic example, the idea of “topics” being found bottom-up from the data can be seen in the semantic coherence of terms within the same topic. In particular, topic one seems to revolve around governance, with the terms “govern[ance]”, “power”, “state”, “constitut[ion]”, and “law”.

Statistics

Various statistics can be used to describe, explore and analyze a text corpus. An example of a popular technique is to rank the information value of words inside a corpus and then visualize the most informative words as a word cloud to get a quick indication of what a corpus is about. Text statistics (e.g., average word and sentence length, word and syllable counts) are also commonly used as an operationalization of concepts such as readability (Flesch, 1948) or lexical diversity (McCarthy & Jarvis, 2010). A wide range of such measures is available in R with the *koRpus* package (Michalke, 2017). Furthermore, there are many useful applications of calculating term and document similarities (which are often based on the inner product of a DTM or transposed DTM), such as analyzing semantic relations between words or concepts and measuring content homogeneity. Both techniques are supported in *quanteda, corpustools* (Welbers & Van Atteveldt, 2016), or dedicated packages such as *textreuse* (Mullen, 2016a) for text overlap.

A particularly useful technique is to compare the term frequencies of two corpora, or between two subsets of the same corpus. For instance, to see which words are more likely to occur in documents about a certain topic. In addition to providing a way to quickly explore how this topic is discussed in the corpus, this can provide input for developing better queries. In the following example we show how to perform this technique in quanteda.

```
## create DTM that contains Trump and Obama speeches
corpus_pres = corpus_subset(data_corpus_inaugural,
                            President %in% c("Obama", "Trump"))
dtm_pres = dfm(corpus_pres, groups = "President",
               remove = stopwords("english"), remove_punct = TRUE)

## compare target (in this case Trump) to rest of DTM (in this case only Obama).
keyness = textstat_keyness(dtm_pres, target = "Trump")
textplot_keyness(keyness)
```

```
## results in Figure 2.
```

Here, the signed χ^2 measure of association indicates that “america”, “american”, and “first” were used with far greater frequency by Trump than Obama, while “us”, “can”, “freedom”, “peace”, and “liberty” were among the words much more likely to be used by Obama than by Trump.

Advanced topics

The data preparation and bag-of-words analysis techniques discussed above are the basis for the majority of the text analysis approaches that are currently used in communication research. For certain types of analyses, however, techniques might be required that rely on external software modules, are more computationally demanding, or that are more complicated to use. In this section we briefly elaborate on some of these advanced approaches that are worth taking note of.

Advanced NLP

In addition to the preprocessing techniques discussed in the data preparation section, there are powerful preprocessing techniques that rely on more advanced natural language processing (NLP). At present, these techniques are not available in native R, but rely on external software modules that often have to be installed outside of R. Many of the advanced NLP techniques are also much more computationally demanding, thus taking more time to perform. Another complication is that these techniques are language specific, and often only available for English and a few other big languages.

Several R packages provide interfaces for external NLP modules, so that once these modules have been installed, they can easily be used from within R. The *coreNLP* (Arnold & Tilton, 2016) package provides bindings for Stanford CoreNLP java library (Manning et al., 2014), which is a full NLP parser for English, that also supports (albeit with limitations) Arabic, Chinese, French, German, and Spanish. The *spacyr* (Benoit & Matsuo, 2017) package provides an interface for the spaCy module for Python, which is comparable to CoreNLP but is faster, and supports English and (again with some limitations) German and French. A third package, *cleanNLP* (Arnold, 2017a), conveniently wraps

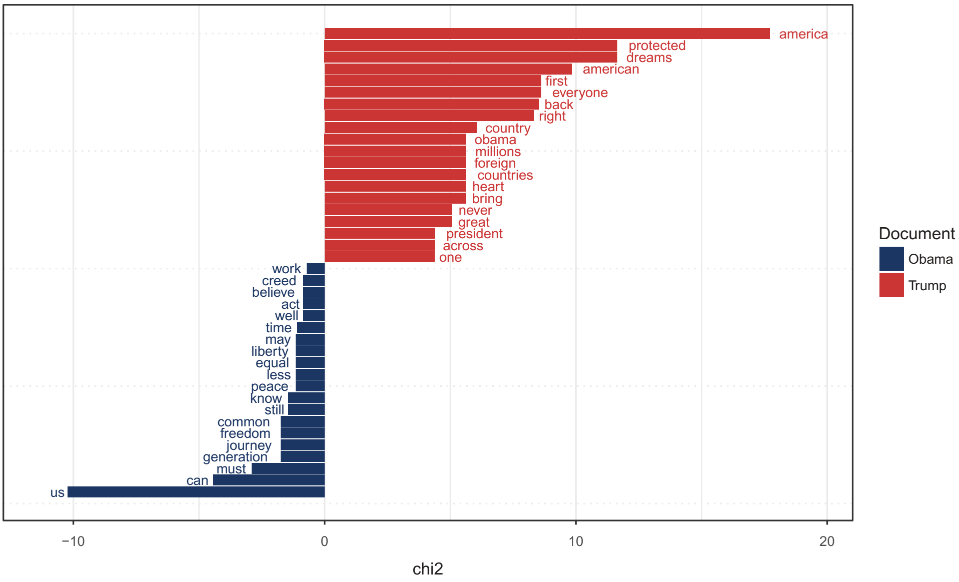

Figure 2. Keyness plot comparing relative word frequencies for Trump and Obama.

both CoreNLP and spaCy, and also includes a minimal back-end that does not rely on external dependencies. This way it can be used as a swiss army knife, choosing the approach that best suits the occasion and for which the back-end is available, but with standardized output and methods.

Advanced NLP parsers generally perform all techniques in one go. In the following example we use the *spacyr* package to parse a sentence, that will be used to illustrate four advanced NLP techniques: lemmatization, part-of-speech (POS) tagging, named entity recognition (NER) and dependency parsing.

```
## first install spaCy. See instructions on spacyr GitHub page:
## https://github.com/kbenoit/spacyr
install.packages("spacyr")
library(spacyr)

spacy_initialize()
d = spacy_parse("Bob Smith gave Alice his login information.", dependency = TRUE)
d[,-c(1,2)]
```

```
token_id        token        lemma    pos  head_token_id    dep_rel    entity
       1          Bob          bob  PROPN              2   compound  PERSON_B
       2        Smith        smith  PROPN              3      nsubj  PERSON_I
       3         gave         give   VERB              3       ROOT
       4        Alice        alice  PROPN              3     dative  PERSON_B
       5          his       -PRON-    ADJ              7       poss
       6        login        login   NOUN              7   compound
       7  information  information   NOUN              3       dobj
       8            .            .  PUNCT              3      punct
```

Lemmatization

Lemmatization fulfills a similar purpose as stemming, but instead of cutting off the ends of terms to normalize them, a dictionary is used to replace terms with their lemma. The main advantage of this approach is that it can more accurately normalize different verb forms—such as “gave” and “give” in the example—which is particularly important for heavily inflected languages such as Dutch or German.

Part-of-speech tagging

POS tags are morpho-syntactic categories for words, such as nouns, verbs, articles and adjectives. In the example we see three proper nouns (PROPN), a verb (VERB) an adjective (ADJ), two nouns (NOUN), and punctuation (PUNCT). This information can be used to focus an analysis on certain types of grammar categories, for example, using nouns and proper names to measure similar events in news items (Welbers, Van Atteveldt, Kleinnijenhuis, & Ruigrok, 2016), or using adjectives to focus on subjective language (De Smedt & Daelemans, 2012). Similarly, it is a good approach for filtering out certain types of words, such as articles or pronouns.

Named entity recognition

Named entity recognition is a technique for identifying whether a word or sequence of words represents an entity and what type of entity, such as a person or organization. Both “Bob Smith” and “Alice” are recognized as persons. Ideally, named entity recognition is paired with co-reference resolution. This is a technique for grouping different references to the same entity, such as anaphora (e.g., he, she, the president). In the example sentence, the word “his” refers to “Bob Smith”. Co-reference resolution is currently only supported by Stanford CoreNLP, but discussion on the spaCy GitHub page suggests that this feature is on the agenda.

Dependency parsing

Dependency parsing provides the syntactic relations between tokens, which can be used to analyze texts at the level of syntactic clauses (Van Atteveldt, 2008). In the *spacyr* output this information is given in the *head_token_i* and *dep_rel* columns, where the former indicates to what token a token is related and the latter indicates the type of relation. For example, we see that “Bob” is related to “Smith” (*head_token_i* 2) as a compound, thus recognizing “Bob Smith” as a single entity. Also, since “Smith” is the nominal subject (nsubj) of the verb “gave”, and Alice is the dative case (dative) we know that “Bob Smith” is the one who gives to “Alice”. This type of information can for instance be used to analyze who is attacking whom in news coverage about the Gaza war (Van Atteveldt et al., 2017).

Word positions and syntax

As discussed above, the bag-of-words representation of texts is memory-efficient and convenient for various types of analyses, and this often outweighs the disadvantage of losing information by dropping the word positions. For some analyses, however, the order of words and syntactical properties can be highly beneficial if not crucial. In this section we address some text representations and analysis techniques where word positions are maintained.

A simple but potentially powerful solution is to use higher order *n-grams*. That is, instead of tokenizing texts into single words (n = 1; *unigrams*), sequences of two words (n = 2; *bigrams*), three words (n = 3; *trigrams*) or more are used.[9] The use of higher order n-grams is often optional in tokenization functions. quanteda makes it possible to form n-grams when tokenizing, or to form ngrams from tokens already formed. Other options include the formation of “skip-grams”, or n-grams from words with variable windows of adjacency. Such non-adjacent collocations form the

[9] The term *n-grams* can be used more broadly to refer to sequences, and is also often used for sequences of individual characters. In this teacher’s corner we strictly use *n-grams* to refer to sequences of words.

basis for counting weighted proximity vectors, used in vector-space network-based models built on deep learning techniques (Mikolov, Chen, Corrado, & Dean, 2013; Selivanov, 2016). Below, we illustrate how to form both tri-grams and skipgrams of size three using a vector of both 0 and 1 skips.

```
text <- "an example of preprocessing techniques"
tokens(text, ngrams = 3, skip = 0:1)
```

```
tokens from 1 document. text1 :
[1] "an_example_of"                    "an_example_preprocessing"
[3] "an_of_preprocessing"              "an_of_techniques"
[5] "example_of_preprocessing"         "example_of_techniques"
[7] "example_preprocessing_techniques" "of_preprocessing_techniques"
```

The advantage of this approach is that these n-grams can be used in the same way as unigrams: we can make a DTM with n-grams, and perform all the types of analyses discussed above. Functions for creating a DTM in R from raw text therefore often allow the use of n-grams other than unigrams. For some analysis this can improve performance. Consider, for instance, the importance of negations and amplifiers in sentiment analysis, such as “not good” and “very bad” (Aue & Gamon, 2005). An important disadvantage, however, is that using n-grams is more computationally demanding, since there are many more unique sequences of words than individual words. This also means that more data is required to get a good estimate of the distribution of n-grams.

Another approach is to preserve the word positions after tokenization. This has three main advantages. First, the order and distance of tokens can be taken into account in analyses, enabling analyses such as the co-occurrence of words within a word window. Second, the data can be transformed into both a fulltext corpus (by pasting together the tokens) and a DTM (by dropping the token positions). This also enables the results of some text analysis techniques to be visualized in the text, such as coloring words based on a word scale model (Slapin & Proksch, 2008), or to produce browsers for topic models (Gardner et al., 2010). Third, each token can be annotated with token specific information, such as obtained from advanced NLP techniques. This enables, for instance, the use of dependency parsing to perform an analysis at the level of syntactic clauses (Van Atteveldt et al., 2017). The main disadvantage of preserving positions is that it is very memory inefficient, especially if all tokens are kept and annotations are added.

A common way to represent tokens with positions maintained is a data frame in which rows represent tokens, ordered by their position, and columns represent different variables pertaining to the token, such as the literal text, its lemma form and its POS tag. An example of this type of representation was shown above in the advanced NLP section, in the *spacyr* token output. Several R packages provide dedicated classes for tokens in this format. One is the *koRpus* (Michalke, 2017) package, which specializes in various types of text statistics, in particular lexical diversity and readability. Another is *corpustools* (Welbers & Van Atteveldt, 2016), which focuses on managing and querying annotated tokens, and on reconstructing texts to visualize quantitative text analysis results in the original text content for qualitative investigation. A third option is *tidytext* (Silge & Robinson, 2016), which does not focus on this format of annotated tokens, but provides a framework for working with tokenized text in data frames.

For a brief demonstration of utilizing word positions, we perform a dictionary search with the *corpustools* package (Welbers & Van Atteveldt, 2016), that supports searching for words within a given word distance. The results are then viewed in key word in context (KWIC) listings. In the example, we look for the queries “freedom” and “americ*” within a distance of five words, using the State of the Union speeches from George W. Bush and Barack Obama.

```
install.packages("corpustools")
library(corpustools)


tc <- create_tcorpus(sotu_texts, doc_column = “id”)
hits <- tc$search_features(’"freedom americ*"~5’)
kwic <- tc$kwic(hits, ntokens = 3)
head(kwic$kwic, 3)
```

```
[1] ...making progress toward <freedom> will find <America> is their friend...
[2] ...friends, and <freedom> in Iraq will make <America> safer for generations...
[3] ...men who despise <freedom>, despise <America>, and aim...
```

Conclusion

R is a powerful platform for computational text analysis, that can be a valuable tool for communication research. First, its well developed packages provide easy access to cutting edge text analysis techniques. As shown here, not only are most common text analysis techniques implemented, but in most cases, multiple packages offer users choice when selecting tools to implement them. Many of these packages have been developed by and for scholars, and provide established procedures for data preparation and analysis. Second, R’s open source nature and excellent system for handling packages make it a convenient platform for bridging the gap between research and tool development, which is paramount to establishing a strong computational methods paradigm in communication research. New algorithms do not have to be confined to abstract and complex explanations in journal articles aimed at methodology experts, or made available through arcane code that many interested parties would not know what to do with. As an R package, algorithms can be made readily available in a standardized and familiar format.

For new users, however, choosing from the wide range of text analysis packages in R can also be daunting. With various alternatives for most techniques, it can be difficult to determine which packages are worth investing the effort to learn. The primary goal of this teacher’s corner, therefore, has been to provide a starting point for scholars looking for ways to incorporate computational text analysis in their research. Our selection of packages is based on our experience as both users and developers of text analysis packages in R, and should cover the most common use cases. In particular, we advise users to become familiar with at least one established and well-maintained package that handles data preparation and management, such as *quanteda, tidytext* (Silge & Robinson, 2016) or *tm* (Feinerer & Hornik, 2017). From here, it is often a small step to convert data to formats that are compatible with most of the available text analysis packages.

It should be emphasized that the selection of packages presented in this teacher’s corner is not exhaustive, and does not represent which packages are the most suitable for the associated functionalities. Often, the best package for the job depends largely on specific features and problem specific priorities such as speed, memory efficiency and accuracy. Furthermore, when it comes to establishing a productive workflow, the importance of personal preference and experience should not be underestimated. A good example is the workflow of the *tidytext* package (Silge & Robinson, 2016), which could be preferred by people that are familiar with the tidyverse philosophy (Wickham et al., 2014). Accordingly, the packages recommended in this teacher’s corner provide a good starting point, and for many users could be all they need, but there are many other great package out there. For a more complete list of packages, a good starting point is the CRAN Task View for Natural Language Processing (Wild, 2017).

Marshall McLuhan (1964) famously stated that “we shape our tools, and thereafter our tools shape us”, and in science the same can be said for how tools shape our findings. We thus argue that the establishment of a strong computational methods paradigm in communication research goes hand-in-hand with embracing open-source tool development as an inherent part of scientific practice. As such, we conclude with a call for researchers to cite R packages similar to how one would cite other scientific work.[10] This gives due credit to developers, and thereby provides a just incentive for developers to publish and maintain new code, including proper testing and documentation to facilitate the correct use of code by others. Citing packages is also paramount for the transparency of research, which is especially important when using new computational techniques, where results might vary depending on implementation choices and where the absence of bugs is often not guaranteed. Just as our theories are shaped through collaboration, transparency and peer feedback, so should we shape our tools.

Declaration of interest

The authors report no conflicts of interest. The authors alone are responsible for the content and writing of the article.

ORCID

Kenneth Benoit http://orcid.org/0000-0002-0797-564X

References

Arnold, T. (2017a). *cleannlp: A tidy data model for natural language processing* [Computer software manual] (R package version 1.9.0). Retrieved from https://CRAN.R-project.org/package=cleanNLP

Arnold, T. (2017b). *kerasR: R interface to the keras deep learning library* [Computer software manual] (R package version 0.6.1). Retrieved from https://CRAN.R-project.org/package=kerasR

Arnold, T., & Tilton, L. (2016). *coreNLP: Wrappers around Stanford CoreNLP tools* [Computer software manual] (R package version 0.4-2). Retrieved from https://CRAN.R-project.org/package=coreNLP

Aue, A., & Gamon, M. (2005). Customizing sentiment classifiers to new domains: A case study. In *Proceedings of Recent Advances in Natural Language Processing (RANLP)*. Retrieved from http://research.microsoft.com/pubs/65430/new_domain_sentiment.pdf

Bates, D., & Maechler, M. (2015). *Matrix: Sparse and dense matrix classes and methods* [Computer software manual] (R package version 1.2-3). Retrieved from https://CRAN.R-project.org/package=Matrix

Benoit, K., & Matsuo, A. (2017). *spacyr: R Wrapper to the spaCY NLP Library* [Computer software manual] (R package version 0.9.0). Retrieved from https://CRAN.R-project.org/package=spacyr

Benoit, K., Watanabe, K., Nulty, P., Obeng, A., Wang, H., Lauderdale, B., & Lowe, W. (2017). *quanteda: Quantitative analysis of textual data* [Computer software manual] (R package version 0.99). Retrieved from http://quanteda.io

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research*, *3*, 993–1022.

Bouchet-Valat, M. (2014). *SnowballC: Snowball Stemmers based on the C Libstemmer UTF-8 Library* [Computer software manual] (R package version 0.5.1). Retrieved from https://CRAN.R-project.org/package=SnowballC

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant automated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, *4*(1), 8–23. doi:10.1080/21670811.2015.1096598

Crone, S. F., Lessmann, S., & Stahlbock, R. (2006). The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, *173*(3), 781–800. doi:10.1016/j.ejor.2005.07.023

De Smedt, T., & Daelemans, W. (2012). "vreselijk mooi!" (terribly beautiful): A subjectivity lexicon for dutch adjectives. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC)*, Istanbul, May 2012, 3568–3572.

Feinerer, I., & Hornik, K. (2017). *tm: Text mining package* [Computer software manual] (R package version 0.7-1). Retrieved from https://CRAN.R-project.org/package=tm

Flesch, R. (1948). A new readability yardstick. *Journal of Applied Psychology*, *32*(3), 221–233. doi:10.1037/h0057532

Fox, J., & Leanage, A. (2016). R and the journal of statistical software. *Journal of Statistical Software*, *73*(2), 1–13.

Gagolewski, M. (2017). *R package stringi: Character string processing facilities* [Computer software manual]. Retrieved from http://www.gagolewski.com/software/stringi/

[10] To view how to cite a package, the *citation* function can be used—e.g., `citation("quanteda")` for citing quanteda, or `citation()` for citing the R project. This either provides the citation details provided by the package developer or auto-generated details.

Gardner, M. J., Lutes, J., Lund, J., Hansen, J., Walker, D., Ringger, E., & Seppi, K. (2010). The topic browser: An interactive tool for browsing topic models. In *Nips workshop on Challenges of Data Visualization*. Retrieved from http://cseweb.ucsd.edu/~lvdmaaten/workshops/nips2010/papers/gardner.pdf

Griffiths, T. L., & Steyvers, M. (2004). Finding scientific topics. In *Proceedings of the National Academy of Sciences*, 5228–5235. doi:10.1073/pnas.0307752101

Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, *21*(3), 267–297. doi:10.1093/pan/mps028

Grun, B., & Hornik, K. (2011). topicmodels: An R package for fitting topic models. *Journal of Statistical Software*, *40* (13), 1–30. doi:10.18637/jss.v040.i13

Günther, E., & Quandt, T. (2016). Word counts and topic models: Automated text analysis methods for digital journalism research. *Digital Journalism*, *4*(1), 75–88. doi:10.1080/21670811.2015.1093270

Jurka, T. P., Collingwood, L., Boydstun, A. E., Grossman, E., & Van Atteveldt, W. (2014). *RTextTools: Automatic text classification via supervised learning* [Computer software manual] (R package version 1.4.2). Retrieved from https://CRAN.R-project.org/package=RTextTools

Lang, D. T., & the CRAN Team. (2017). *XML: Tools for parsing and generating XML within R and S-plus* [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=XML

Leopold, E., & Kindermann, J. (2002). Text categorization with support vector machines. How to represent texts in input space? *Machine Learning*, *46*(1), 423–444. doi:10.1023/a:1012491419635

Manning, C. D., Manning, C. D., Raghavan, P., Raghavan, P., Schütze, H., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge, UK: Cambridge University Press. doi:10.1017/cbo9780511809071

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (pp. 55–60). doi: 10.3115/v1/p14-5010

McCarthy, P. M., & Jarvis, S. (2010). Mtld, vocd-d, and hd-d: A validation study of sophisticated approaches to lexical diversity assessment. *Behavior Research Methods*, *42*(2), 381–392. doi:10.3758/brm.42.2.381

McLuhanm, M. (1964). *Understanding Media: The Extensions of Man*. New York: Pinguin Press.

Meyer, D., Hornik, K., & Feinerer, I. (2008). Text mining infrastructure in r. *Journal of Statistical Software*, *25*(5), 1–54. doi:10.18637/jss.v025.i05

Michalke, M. (2017). *koRpus: An R package for text analysis* [Computer software manual] (Version 0.10-2). Retrieved from https://reaktanz.de/?c=hacking&s=koRpus

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of International Conference of Learning Representations. arXiv preprint arXiv:1301.3781*, Scottsdale, Arizona, May 2013.

Mostafa, M. M. (2013). More than words: Social networks' text mining for consumer brand sentiments. *Expertat Systems with Applications*, *40*(10), 4241–4251. doi:10.1016/j.eswa.2013.01.019

Mullen, L. (2016a). *textreuse: Detect text reuse and document similarity* [Computer software manual] (R package version 0.1.4). Retrieved from https://CRAN.R-project.org/package=textreuse.

Mullen, L. (2016b). *tokenizers: A consistent interface to tokenize natural language text* [Computer software manual] (R package version 0.1.4). Retrieved from https://CRAN.R-project.org/package=tokenizers

Ooms, J. (2014). *The jsonlite package: A practical and consistent mapping between json data and r objects* [Computer software manual]. Retrieved from https://arxiv.org/abs/1403.2805

Ooms, J. (2017a). *antiword: Extract text from microsoft word documents* [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=antiword

Ooms, J. (2017b). *pdftools: Text extraction, rendering and converting of pdf documents* [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=pdftools

Porter, M. F. (2001). *Snowball: A language for stemming algorithms*. Retrieved from http://snowball.tartarus.org/texts/introduction.html

Proksch, S.-O., & Slapin, J. B. (2009). How to avoid pitfalls in statistical analysis of political texts: The case of germany. *German Politics*, *18*(3), 323–344. doi:10.1080/09644000903055799

Provost, F., & Fawcett, T. (2013). Data science and its Relationship to Big Data and Data-Driven Decision Making. *Big Data*, *1*(1), 51–59. doi:10.1089/big.2013.1508

R Core Team. (2017). *R: A language and environment for statistical computing* [Computer software manual]. Vienna, Austria. Retrieved from https://www.R-project.org/

Roberts, M. E., Stewart, B. M., Tingley, D., Lucas, C., Leder-Luis, J., Gadarian, S. K., … Rand, D. G. (2014). Structural topic models for open-ended survey responses. *American Journal of Political Science*, *58*(4), 1064–1082. doi:10.1111/ajps.12103

rOpenSci Text Workshop. (2017). *tif: Text interchange format* [Computer software manual]. Retrieved from https://github.com/ropensci/tif

Schuck, A. R., Xezonakis, G., Elenbaas, M., Banducci, S. A., & De Vreese, C. H. (2011). Party contestation and Europe on the news agenda: The 2009 European Parliamentary Elections. *Electoral Studies*, *30*(1), 41–52. doi:10.1016/j.electstud.2010.09.021

Schultz, F., Kleinnijenhuis, J., Oegema, D., Utz, S., & Van Atteveldt, W. (2012). Strategic framing in the BP crisis: A semantic network analysis of associative frames. *Public Relations Review*, *38*(1), 97–107. doi:10.1016/j.pubrev.2011.08.003t

Selivanov, D. (2016). *text2vec: Modern text mining framework for R* [Computer software manual] (R package version 0.4.0). Retrieved from https://CRAN.R-project.org/package=text2vec

Silge, J., & Robinson, D. (2016). tidytext: Text mining and analysis using tidy data principles in R. *Journal of Open Source Software*, *1*, 3. doi:10.21105/joss.00037

Slapin, J. B., & Proksch, S.-O. (2008). A scaling model for estimating time-series party positions from texts. *American Journal of Political Science*, *52*(3), 705–722. doi:10.1111/j.1540-5907.2008.00338.x

Taboada, M., Brooke, J., Tofiloski, M., Voll, K., & Stede, M. (2011). Lexicon-based methods for sentiment analysis. *Computational Linguistics*, *37*(2), 267–307. doi:10.1162/coli_a_00049

Tausczik, Y. R., & Pennebaker, J. W. (2010). The psychological meaning of words: LIWC and computerized text analysis methods. *Journal of Language and Social Psychology*, *29*(1), 24–54. doi:10.1177/0261927X09351676

TIOBE. (2017). *The R programming language*. Retrieved from https://www.tiobe.com/tiobe-index/r/

Van Atteveldt, W. (2008). Semantic Network Analysis: Techniques for Extracting, Representing, and Querying Media Content (Dissertation). Charleston, SC: BookSurge.

Van Atteveldt, W., Sheafer, T., Shenhav, S. R., & Fogel-Dror, Y. (2017). Clause analysis: Using syntactic information to automatically extract source, subject, and predicate from texts with an application to the 2008–2009 Gaza War. *Political Analysis*, *25*(2), 207–222. doi:10.1017/pan.2016.12

Vliegenthart, R., Boomgaarden, H. G., & Van Spanje, J. (2012). Anti-immigrant party support and media visibility: A cross-party, over-time perspective. *Journal of Elections, Public Opinion & Parties*, *22*(3), 315–358. doi:10.1080/17457289.2012.693933

Watanabe, K. (2017). The spread of the Kremlin's narratives by a western news agency during the Ukraine crisis. *The Journal of International Communication*, *23*(1), 138–158. doi:10.1080/13216597.2017.1287750

Welbers, K., & Van Atteveldt, W. (2016). *corpustools: Tools for managing, querying and analyzing tokenized text* [Computer software manual] (R package version 0.201). Retrieved from http://github.com/kasperwelbers/corpustools

Welbers, K., Van Atteveldt, W., Kleinnijenhuis, J., & Ruigrok, N. (2016). A Gatekeeper among Gatekeepers: News Agency Influence in Print and Online Newspapers in the Netherlands. *Journalism Studies*, 1–19 (online first). doi:10.1080/1461670x.2016.1190663

Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, *59*(10), 1–23. doi:10.18637/jss.v059.i10

Wickham, H., & Bryan, J. (2017). *readxl: Read excel files* [Computer software manual]. Retrieved from https://CRAN.R-project.org/package=readxl

Wild, F. (2017). Cran task view: Natural language processing. *CRAN*. Version: 2017-01-17. Retrieved from https://CRAN.R-project.org/view=NaturalLanguageProcessing.

Yang, Y., & Pedersen, J. O. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning* (*ICML*) (pp. 412–420), Nashville, TN, July 1997.