Advanced text preprocessing for Latent Dirichlet Allocation topic modeling with R

Peter Andreas Mikkelsen Thramkrongart

## Abstract

Advanced text preprocessing tasks such lemmatization, part-of-speech tagging, and named-entity recognition can greatly improve natural language processing and text mining results, R programming language seems to lack a beginner-friendly fully-featured framework, that contains implementations for all such task. In this project, attempt to perform advanced text preprocessing using the ported R versions of the natural language processing frameworks *openNLP* and *SpaCy* and demonstrate the result with topic modeling. I conclude that SpaCy is very easy to use, but rather difficult to install with R and that openNLP is slower and conceptually more difficult to use, but simpler to install.

## Keywords:

Text mining; Preprocessing; Named-Entity Recognition; Lemmatization; Part-of-Speech Tagging; Latent Dirichlet Allocation

## 1 Introduction

In this age of information, immense amounts of data are being generated around us. Text is an incredibly valuable data type, but it is tricky to quantitatively with as is. It often needs to be pre-processed to usable for analysis and modeling. This project aims to explore how to perform advanced text pre-processing in the statistical programming language *R* (R Core Team 2017). Such preprocessing is usually performed with the general programming language *Python*, but many researchers, including myself, are more accustomed to R, as it is easier to use and more specialized for statistics and data analysis. Therefore, this project explores how advanced preprocessing can be performed in R to make fully streamlined projects kept entirely in R. Most researchers are not expert programmers, so accessible programming examples are always welcome when searching for how to perform tasks such as preprocessing, and hopefully, somebody will find this useful when searching for to perform advanced processing in R.

## 2 Problems and Background

For language data such as text and speech, text mining, the process of extracting meaningful information from unstructured text data using natural language processing techniques, is a fairly accessible sub-field of data science, that is applicable in many different fields such as biomedicine, humanities, and business. Text mining can help uncover latent topics, produce word clouds of frequent or important words, and model sentiment among many other things. Inconveniently, the language data has to be heavily preprocessed for most tasks. The very simple framework *Tidytext* and its accompanying book *Tidy Text Mining with R: A tidy approach* (Queiroz et al. 2020; Silge and Robinson 2017) is most novices’ first introduction to Text Mining because of its use of Tidy Data principles (Wickham 2014) and its accessibility. For preprocessing, it only features rudimentary pre-processing functions *tokenization* (dividing text into individual tokens such as words) and *TF-IDF-weighting* (a method for weighting tokens by their importance in a document compared to other documents). A sufficiently featured preprocessing pipeline usually features tokenization, *part-of-speech-tagging* (tagging words by their word-class such as noun or verb), *lemmatization* (transforming words to their dictionary form), *named-entity recognition* (e.g., tagging names and locations), removal of unnecessary words and commonly a type of vectorization or embedding (attempting to compress the meanings of words into vectors based on the contexts they appear in). I have not yet found a fully-featured software framework in R of text preprocessing (Wild et al. 2020). *SpaCyR* (Benoit, Matsuo, and Council (ERC-2011-StG 283794-QUANTESS) 2020) is an R wrapper for the fully-featured and fairly fast Python framework SpaCy (Honnibal et al. 2020). In Python, SpaCy is great because of its speed, ease of use, and compatibility with many languages. In R, while fast, it is not as beginner-friendly. It is difficult to install because of its need for a separate installation. Another option is the R wrapper *openNLP* for Apache openNLP in *Java* (Apache Software Foundation 2014; Hornik 2019). It has most features including models for named-entity recognition, but it lacks a lemmatizer and it does not have a vectorized processing pipeline. Therefore, one has to loop over each text to perform the preprocessing. This makes the framework slower and less beginner-friendly. Lastly, there is the popular and powerful framework *UDPipe* (Wijffels et al. 2020). Problematically, UDPipe does not named-entity recognition, which in contrast to lemmatization is much more troublesome to add afterward, as it essentially requires you to turn to another framework such as openNLP. To explore these frameworks, this project reports my attempt of performing advanced preprocessing with openNLP and SpaCyR.

## 3 Software Framework

This project was entirely performed on my Lenovo T410-I7 PC with an Intel Core i74600U CPU 2.10GHz 2.69GHz processor and 12GVB RAM which runs Windows 10 Enterprise operating system. The project was run on the desktop version of R (4.0.3)(R Core Team 2017) in *RStudio* (3.167) (RStudio Team 2020). For this project, the Python installation was installed according to the SpaCyR manual. Java was installed automatically with rJava (Urbanek 2020) dependency of openNLP in R. This document contains all the R code used for this project in code chunks. The majority of the code is commented out to avoid unnecessarily computing elements that take long to run multiple times. Instead, the saved outputs are merely reloaded and printed. A list of package versions is available in table 1 in the appendix. Below is the installation code for the installations.

#remotes::install\_github("paleolimbot/rbbt") #package for RMarkdown citations  
#install.packages("openNLP") ## properly install package with dependencies  
  
#Loading and installing packages with the pacman package  
#If p\_load() does not work, you will have to install the package manually  
pacman::p\_load(  
 tidyverse, #essential package for data wrangling and tidy syntax  
 LDAvis, #package for LDA vizualization  
 servr, #for hosting servers  
 NLP, #used for openNLP pipeline  
 gutenbergr, #used for retrieving texts from Project Gutenberg  
 text2vec, #used for LDA-modeling  
 textstem, #used for lemmatization  
 openNLP, #NLP preprocessing pipeline  
 tictoc, #used for timing  
 spacyr, #NLP preprocessing pipeline  
 tokenizers, #used for chunking  
 openNLPmodels.en, #english language model for openNLP  
 rbbt, #package for citations in RMarkdown,  
 sessioninfo, #For getting packageversions  
 flextable #for making tables  
 )  
#load("workspace.RData") #load workspace from ealier...  
  
#install.packages("openNLPmodels.en", dependencies=TRUE, repos = "http://datacube.wu.ac.at/")  
  
#spacy\_install() #this installs spacy in Python

## 4 Data Acquisition and Processing

### Retrieving the texts

The preprocessing was performed on horror books in English acquired from the digital archive *Project Gutenberg* using the R library *gutenbergr* (Robinson 2020). The text files came in individual lines that I collapsed into full texts and removed all punctuation but commas and punctuation marks. The code chunk below shows this process.

IDs <- #making a lists of the horror titles that we are going to use  
 gutenberg\_metadata %>%  
 filter(str\_detect(gutenberg\_bookshelf, "Horror") == T &  
 language == "en") %>% distinct(title, .keep\_all = T)  
  
IDs %>% glimpse()

## Rows: 41  
## Columns: 8  
## $ gutenberg\_id <int> 42, 209, 345, 375, 389, 1188, 2147, 2148, 2149,...  
## $ title <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde", ...  
## $ author <chr> "Stevenson, Robert Louis", "James, Henry", "Sto...  
## $ gutenberg\_author\_id <int> 35, 113, 190, 206, 214, 190, 481, 481, 481, 481...  
## $ language <chr> "en", "en", "en", "en", "en", "en", "en", "en",...  
## $ gutenberg\_bookshelf <chr> "Horror/Movie Books/Precursors of Science Ficti...  
## $ rights <chr> "Public domain in the USA.", "Public domain in ...  
## $ has\_text <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,...

texts <- #Downloading the data  
 gutenberg\_download(IDs$gutenberg\_id, meta\_fields = c("title", "author"))  
  
texts <- #collapsing data into single texts  
 texts %>% group\_by(title) %>% mutate(text = glue::glue\_collapse(text, " ")) %>% unique()  
  
texts$text <- #removing unnecessary characters and  
 texts$text %>% str\_replace\_all("[^[:alnum:][:space:]'\\.,]", "")  
  
texts %>% #save data  
 write\_csv("data/texts.csv")

texts <- read\_csv("data/texts.csv") #read the data again

##   
## -- Column specification --------------------------------------------------------  
## cols(  
## gutenberg\_id = col\_double(),  
## text = col\_character(),  
## title = col\_character(),  
## author = col\_character()  
## )

texts %>% glimpse() ##look at the data

## Rows: 41  
## Columns: 4  
## $ gutenberg\_id <dbl> 42, 209, 345, 375, 389, 1188, 2147, 2148, 2149, 2150, ...  
## $ text <chr> "STRANGE CASE OF DR. JE...  
## $ title <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde", "The Tu...  
## $ author <chr> "Stevenson, Robert Louis", "James, Henry", "Stoker, Br...

### Chunking

The texts were divided into smaller chunks of 100 words to lessen the strain on memory. The R library *tokenizers* was used for chunking (Mullen et al. 2018). It is a bit difficult to get the right chunk size. For some computations, larger chunks make things run faster, and vice versa. In my case, it generally made things run slower, but it lessened the risk of memory overload. Some chunking is needed to make SpaCy run properly.

tic()  
chunks <- #chunking data to smaller parts  
 chunk\_text(  
 texts$text,  
 chunk\_size = 100,  
 doc\_id = texts$title,  
 lowercase = F,  
 strip\_punct = F  
 )  
toc()#32.97 sec elapsed

## 11.74 sec elapsed

text\_tibble <- #turning list into tibble  
 tibble(  
 "title" = names(chunks) %>% str\_replace\_all("-\\d+", ""),  
 "titleChunked" = names(chunks),  
 "text" = chunks  
 )  
  
text\_tibble %>% glimpse() #looking at data

## Rows: 31,279  
## Columns: 3  
## $ title <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde", "The St...  
## $ titleChunked <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde-001", "Th...  
## $ text <named list> ["STRANGE CASE OF DR . JEKYLL AND MR . HYDE BY ...

#text\_tibble %>% write\_rds("data/text\_tibble.rds") #saving the data

### Lemmatizing with textstem only

For minimal preprocessing, which can be universal for most simple projects, the chunked texts were lemmatized with the *textstem* library (Rinker 2018). The functions of textstem are vectorized and acceptably fast, making it a rather beginner-friendly library, that I will argue that more beginners should know about. Textstem can easily be added to the simple tidytext framework. Below is the code chunk for the lemmatization. The preprocessing with SpaCy and openNLP was performed on the non-lemmatized texts.

tic() #lemmatizing strings...  
lemmatized\_tibble <- text\_tibble %>% mutate(text = text %>% lemmatize\_strings())  
toc()#897.18 sec elapsed  
  
lemmatized\_tibble$text <- lemmatized\_tibble$text %>% tolower() #transforming texts to lowercase  
  
lemmatized\_tibble %>% write\_csv("data/lemmatized\_tibble.csv")

lemmatized\_tibble <- read\_csv("data/lemmatized\_tibble.csv")#loading the lemmatized data   
  
lemmatized\_tibble %>% glimpse()

## Rows: 31,279  
## Columns: 3  
## $ title <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde", "The St...  
## $ titleChunked <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde-001", "Th...  
## $ text <chr> "strange case of dr. jekyll and mr. hyde by robert lou...

### Full preprocessing with SpaCy

The first full preprocessing framework I tried was SpaCy. It was a bit of a hassle to install, and it caused problems after the installation as well. *Reticulate* (Ushey, Allaire, and Tang 2020), the library for porting Python has some trouble with a module called *rpytools*. For some reason, this problem can be overruled by initializing SpaCy twice. The preprocessing itself is incredibly easy. Because it is fully vectorized, one only has to run the *spacy\_parse()* function once to tokenize, lemmatize, part-of-speech-tag, and named-entity-tag the whole data set. Additionally, it only took half an hour to run. Using the *tidyverse* library family (Wickham et al. 2019), I filtered and collapsed the texts. This took a bit over 10 minutes but using a faster library such as *data.table* (Dowle and Srinivasan 2020) and/over parallelizing the process could probably have shaved some minutes off the run time. The code chunk below shows the SpaCy preprocessing.

#you need to have spacy and spacyr properly installed for this chunk  
  
## I run spacy in a virtual environment  
# spacy\_initialize(condaenv = "C:/Users/thram\_000/anaconda3/envs/spacy\_condaenv",save\_profile = TRUE)  
  
spacey\_test <- #Because of an error with Reticulate, the library used for porting Python, I have to run this command twice to properly start the Python session  
 try(  
 spacy\_parse(#preprocess the two first chunks  
 unlist(  
 chunks[1:2])))

## spacy python option is already set, spacyr will use:  
## condaenv = "C:/Users/thram\_000/anaconda3/envs/spacy\_condaenv"

## Error in py\_module\_import(module, convert = convert) :   
## ModuleNotFoundError: No module named 'rpytools'

spacey\_test <- #test  
 try(  
 spacy\_parse(  
 unlist(  
 chunks[1:2])))

## Python space is already attached. If you want to switch to a different Python, please restart R.

## successfully initialized (spaCy Version: 2.3.5, language model: en\_core\_web\_sm)

## (python options: type = "condaenv", value = "C:/Users/thram\_000/anaconda3/envs/spacy\_condaenv")

try(spacey\_test %>%  
 glimpse())#this it what it looks like

## Rows: 201  
## Columns: 7  
## $ doc\_id <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde-001", "The...  
## $ sentence\_id <int> 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2...  
## $ token\_id <int> 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1...  
## $ token <chr> "STRANGE", "CASE", "OF", "DR", ".", "JEKYLL", "AND", "M...  
## $ lemma <chr> "strange", "case", "of", "DR", ".", "JEKYLL", "and", "M...  
## $ pos <chr> "ADJ", "NOUN", "ADP", "PROPN", "PUNCT", "PROPN", "CCONJ...  
## $ entity <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", "",...

#doing it on all the data. Note how long it takes  
tic()  
spacy\_lemmas <- spacy\_parse(unlist(chunks))  
toc()#1778.81 sec elapsed  
spacy\_lemmas %>% write\_csv("data/spacy\_lemmas.csv")  
  
#loading the lemmas. This file is too big for github  
spacy\_lemmas <- read\_csv("data/spacy\_lemmas.csv")  
  
tic() #filtering the lemmas  
spacy\_texts <- spacy\_lemmas %>%  
 filter(pos == "ADJ" |  
 pos == "ADV" |  
 pos == "NOUN" |  
 pos == "VERB") %>%  
 mutate(title = doc\_id %>%  
 str\_replace\_all("-\\d+", ""),  
 titleChunked = doc\_id) %>%  
 select(c("lemma", "title", "titleChunked")) %>%  
 group\_by(titleChunked) %>%  
 mutate(lemma = glue::glue\_collapse(lemma, " ")) %>% #collapsing the texts  
 unique() %>%  
 rename(text = lemma)  
toc() #775.12 sec elapsed  
  
spacy\_texts %>% write\_csv("data/spacy\_texts.csv")

spacy\_texts <- read.csv("data/spacy\_texts.csv")  
  
spacy\_texts %>% glimpse()

## Rows: 31,276  
## Columns: 3  
## $ text <chr> "strange case story lawyer man rugged countenance neve...  
## $ title <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde", "The St...  
## $ titleChunked <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde-001", "Th...

### Full preprocessing with openNLP

The preprocessing with openNLP was based on an example by L. Jockers & Thalken,(2020). The texts were tokenized into sentences and words, and the words, part-of-speech-tagged, named-entity-tagged, and filtered for appropriate word classes. Because openNLP is not fully vectorized, I had to loop over each chunk to perform the preprocessing pipeline. The preprocessing with openNLP took almost two hours. After that, chunks still had to be lemmatized as well. The code chunk below shows the openNLP preprocessing and the lemmatization with textstem.

#defining pipeline for POS tagging  
sent\_token\_annotator <- Maxent\_Sent\_Token\_Annotator() #sentence tokenizer  
word\_token\_annotator <- Maxent\_Word\_Token\_Annotator() #word tokenizer  
pos\_tag\_annotator <- Maxent\_POS\_Tag\_Annotator() #Part-of-speech-tagger  
  
  
tic() #preprocessing withopenNLP and textstem. It takes almost 2 hours to run  
openNLP\_texts <- NULL  
  
for (i in seq\_along(text\_tibble$titleChunked)) {  
  
 para\_text <- text\_tibble$text[i] #grapping text  
  
 text\_s <- as.String(para\_text) # turning into the right string format  
  
 #start annotating string  
 annotated\_string <- annotate(text\_s,  
 list(  
 sent\_token\_annotator,  
 word\_token\_annotator,  
 pos\_tag\_annotator  
 ))  
  
 word\_pos <- subset(annotated\_string, type == "word") #grapping position of words  
  
 tags\_v <- sapply(word\_pos$features, `[[`, "POS") #grapping pos tags  
  
 words\_v <- text\_s[word\_pos] #grapping pos tags  
 #assembling in dataframe  
 word\_pos\_df <- data.frame(Token = words\_v,  
 POS = tags\_v,  
 stringsAsFactors = FALSE)   
  
 filtered\_df <-  
 filter(word\_pos\_df,#we are only interested in nouns verbs, and adjectives  
 POS == "NN"| POS == "NNS" | str\_detect(POS, "VB") | str\_detect(POS, "JJ")| str\_detect(POS, "RB")) %>%  
 select(Token) %>%  
 mutate(Token = tolower(Token)) #to lower case  
  
 text\_v = paste(filtered\_df$Token, collapse = " ")  
  
 df = tibble(title = text\_tibble$title[i],  
 titleChunked = text\_tibble$titleChunked[i],  
 text = text\_v)  
 openNLP\_texts <- rbind(openNLP\_texts, df) #adding to the dataframe  
  
 cat("chunk", i,"out of",length(text\_tibble$titleChunked), "\r")#status text  
}  
toc()#6815.61 sec elapsed  
  
tic() #lemmatizing with textstem  
openNLP\_texts$text <- openNLP\_texts$text %>% lemmatize\_strings()  
toc()  
  
openNLP\_texts %>% write\_csv("data/openNLP\_texts.csv")

openNLP\_texts <- read\_csv("data/openNLP\_texts.csv")  
  
openNLP\_texts %>% glimpse() #this is what it looks like afterwards

## Rows: 31,279  
## Columns: 3  
## $ title <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde", "The St...  
## $ titleChunked <chr> "The Strange Case of Dr. Jekyll and Mr. Hyde-001", "Th...  
## $ text <chr> "dr story door lawyer be man rugged countenance be nev...

## 5 Topic Modeling

### Latent Dirichlet Allocation

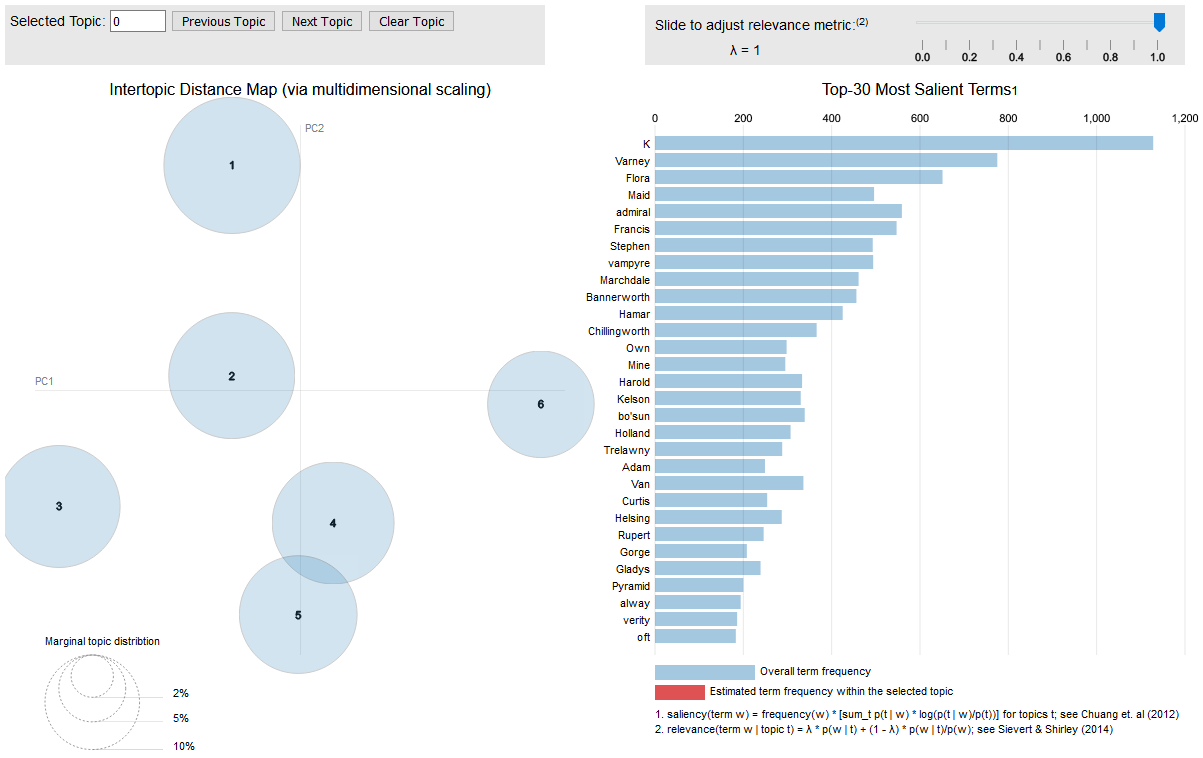
To demonstrate the effect of the preprocessing, I assembled a generic function to perform topic modeling using the implementation of Latent Dirichlet Allocation in the *text2vec* library (Blei, Ng, and Jordan 2003; Selivanov 2018). Topic modeling is a technique for discovering abstract topics from a collection of documents (Selivanov 2018). Latent Dirichlet Allocation is a probabilistic algorithm for clustering words in a number of latent topics. It has become the industry standard because of its robustness and speed (Blei, Ng, and Jordan 2003; Jelodar et al. 2019; Selivanov 2018). The technique is very useful to gain insight into the contents of a corpus and for assigning topics to documents when there are too many documents to read. My function was heavily based on the example from the webpage of text2vec (Selivanov 2018). For the sake of simplicity, I choose to use the same parameter values as the example. The function requires the number of topics to be specified. Because this project was about the preprocessing and not about the models, not much attention was turned to the fit of the models. The models were visualized using the R library *LDAvis* for making interactive topic modeling visualizations (Sievert and Shirley 2015) In this document, I have embedded static screen captures of the output. The *$plot()* method starts a server that hosts an interactive visualization. If you run the code, you can explore the model interactively yourself.

#defining a generic function for creating and fitting LDA models  
custom\_LDA\_func <- function(df, n) {  
 tokens = df$text %>% word\_tokenizer()  
   
 it = itoken(tokens, ids = df$title, progressbar = T)  
   
 v = create\_vocabulary(it,  
 stopwords = stopwords::stopwords()) %>%   
 prune\_vocabulary(v,  
 term\_count\_min = 5,  
 doc\_proportion\_max = 0.2)  
   
 vectorizer = vocab\_vectorizer(v)  
   
 dtm = create\_dtm(it, vectorizer, type = "dgTMatrix")  
   
 lda\_model = LDA$new(  
 n\_topics = n,  
 doc\_topic\_prior = 0.1,  
 topic\_word\_prior = 0.01  
 )  
   
 doc\_topic\_distr =  
 lda\_model$fit\_transform(  
 x = dtm,  
 n\_iter = 1000,  
 convergence\_tol = 0.001,  
 n\_check\_convergence = 25,  
 progressbar = T  
 )  
 return(lda\_model)  
}

### Modeling on raw texts

On the full raw texts, I was able to fit 6 topics. Because I have not read the texts, the topics were essentially meaningless to me, as they were only clusters of names of characters from books that I have not read. The code chunk below and the accompanying screen capture shows the fitting process.

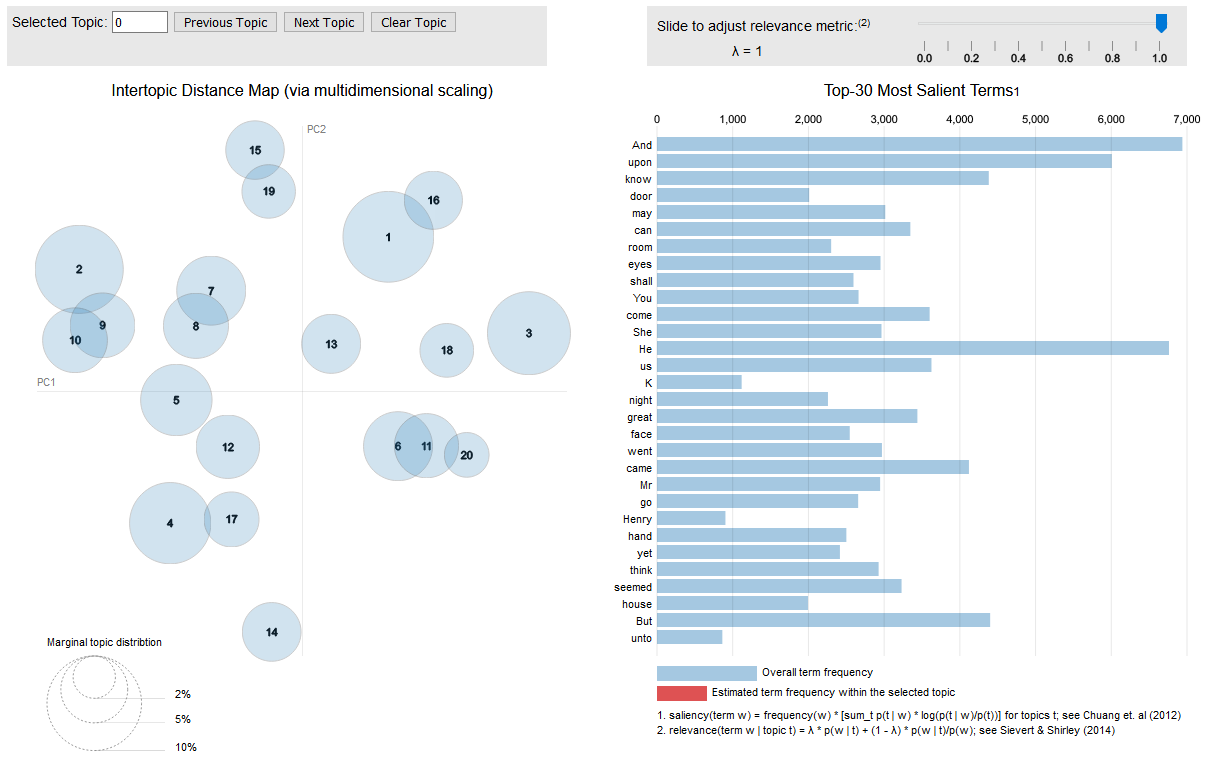
raw\_LDA <- custom\_LDA\_func(texts, 6) #fitting LDA on raw books  
  
# raw\_LDA %>% write\_rds("data/raw\_LDA.rds") #saving model  
#   
# raw\_LDA <- read\_rds("data/raw\_LDA.rds") #loading model  
  
#this part is not visible in the knitted document. It starts a server that hosts the visualization.  
raw\_LDA$plot(out.dir = "LDA\_files/raw\_LDAvis") #visualizing model with LDAvis



Screen capture of the interactive visualization of the LDA-Topic model fitted on the raw texts

On the chunked raw texts, I was able to fit 20 topics. The topics were not dominated by names, like on the full texts. This is because of the sampling technique used in the Latent Dirichlet Allocation implementation (Selivanov, Bickel, and Wang 2020). It uses WarpLDA sampling where run time does not depend on the number of topics. Fitting still depends on the number of documents, and because it treats each chunk as a document, the run time was considerably longer. The model is still far from perfect and the topics are still containing some words, that don’t carry much interesting meaning in a topic modeling context.

#fitting LDA on smaller chunks  
chunked\_LDA <- custom\_LDA\_func(text\_tibble, 20)  
  
# chunked\_LDA %>% write\_rds("data/chunked\_LDA.rds") #saving model  
#   
# chunked\_LDA <- read\_rds("data/chunked\_LDA.rds")  
  
chunked\_LDA$plot(out.dir = "LDA\_files/chunked\_LDAvis")

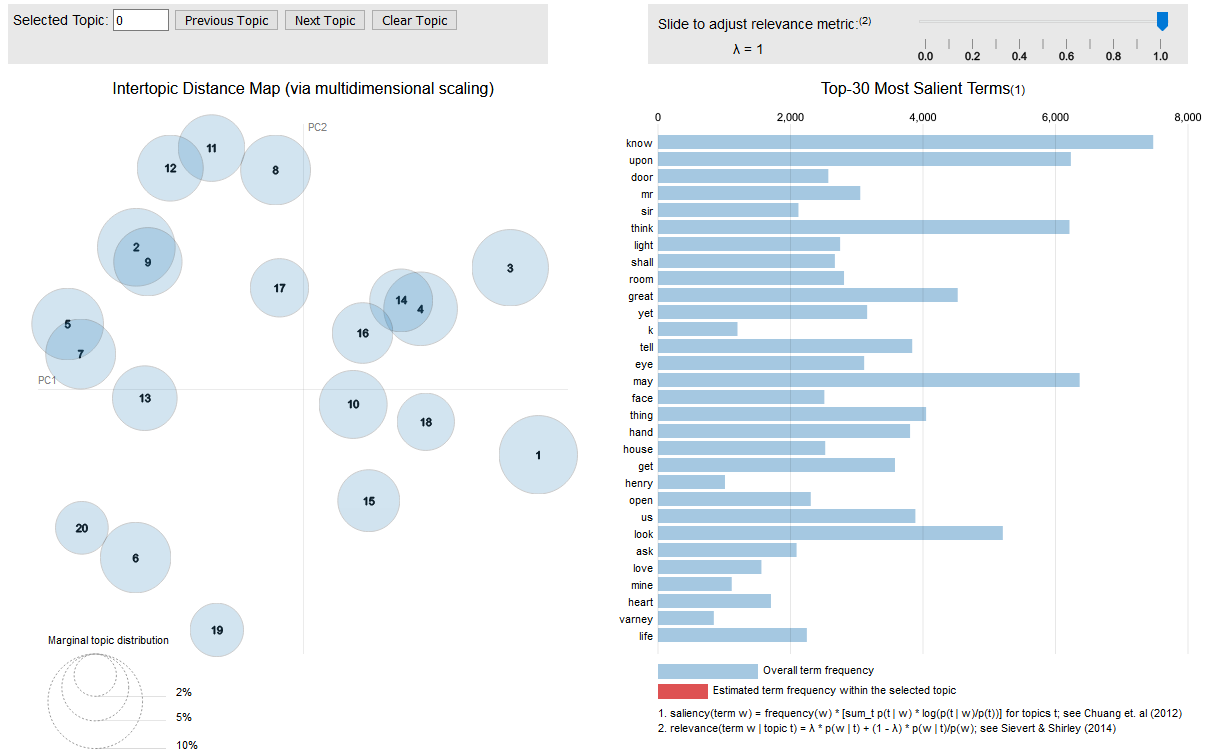


Screen capture of the interactive visualization of the LDA-Topic model fitted on the chunked texts

### Modeling the lemmatized texts

On the chunked and lemmatized texts, I was able to fit 20 topics. The model was further improved, but the topics still contain topics, that I would like to remove.

fitting model on lemmatized texts  
lemmatized\_LDA <- custom\_LDA\_func(lemmatized\_tibble, 20)  
  
# lemmatized\_LDA %>% write\_rds("data/lemmatized\_LDA.rds") #saving model  
#   
# lemmatized\_LDA <- read\_rds("data/lemmatized\_LDA.rds") #loading model  
  
lemmatized\_LDA$plot(out.dir = "LDA\_files/lemmatized\_LDAvis")

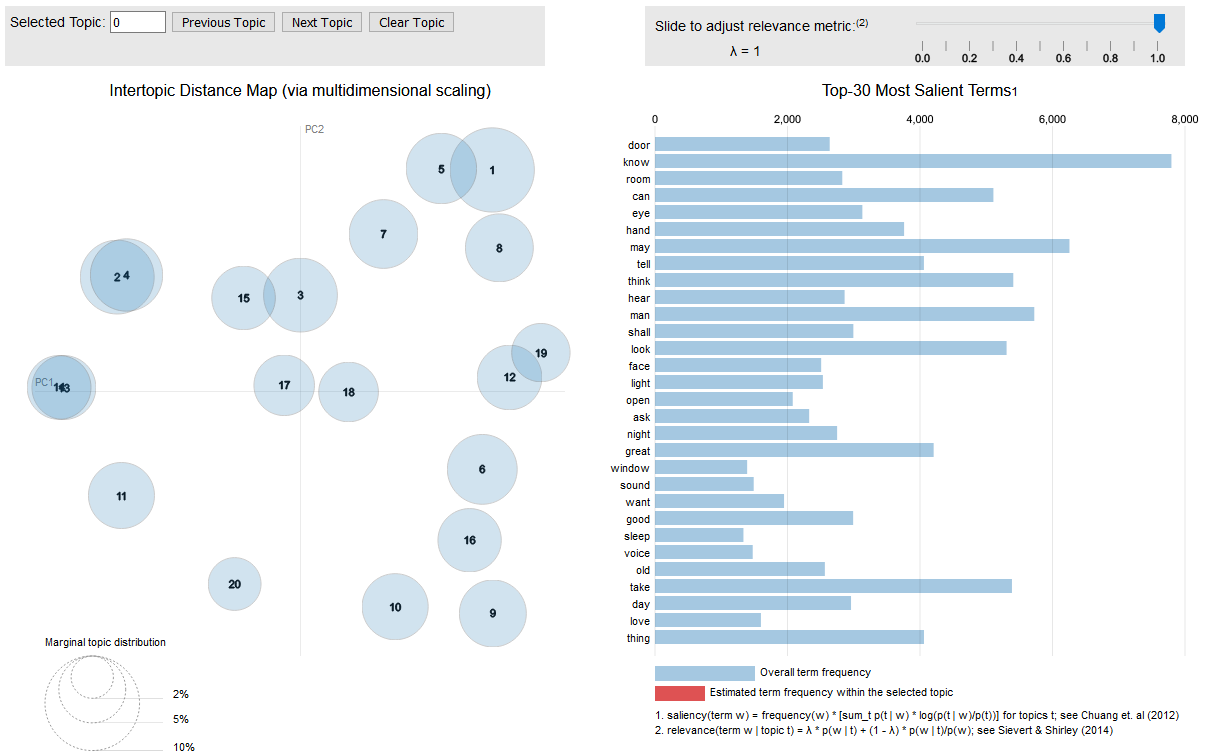


Screen capture of the interactive visualization of the LDA-Topic model fitted on the lemmatized texts

### Modeling on SpaCy texts

With SpaCy I was able to perform all the preprocessing would like to do for this task. Like before, I could fit a 20-topic model with ease, but this time, the topics were much more densely populated with words that carry meaning in a topic modeling context. With 20 topics, there was some overlap, but that can be fixed with better fitting parameters. In this case, we care more about the quality of the topics, rather than the number of topics. The named-entity tags were not used in this case, but they can be very useful for another task. One could imagine analyzing the words relating to each character, location, or something similar.

spacy\_LDA <- spacy\_texts %>% custom\_LDA\_func(20) #fitting model  
  
# spacy\_LDA %>% saveRDS("data/spacy\_LDA.rds") #saving model  
#   
# spacy\_LDA <- read\_rds("data/spacy\_LDA.rds") #loading model  
  
#wow it can fit 20 topics now!  
spacy\_LDA$plot(out.dir = "LDA\_files/spacy\_LDAvis")

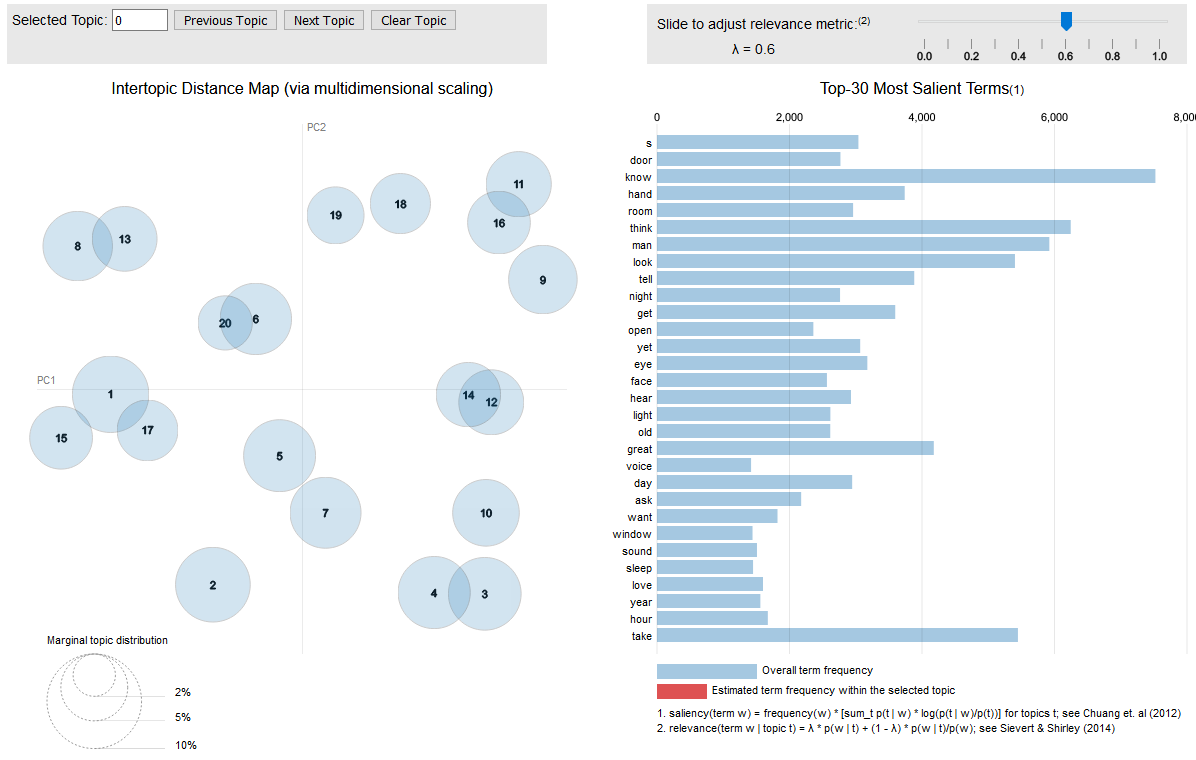


Screen capture of the interactive visualization of the LDA-Topic model fitted the data preprocessed with SpaCy

### Modeling on openNLP texts

As expected, the preprocessing with openNLP provided similar results to SpaCy. There was less topic overlap. I suspect the differences to be caused by differing language models, for the tagging.

openNLP\_LDA <- openNLP\_texts %>% custom\_LDA\_func(20) #running LDA  
  
# openNLP\_LDA %>% saveRDS("data/openNLP\_LDA.rds")  
#   
# openNLP\_LDA <- read\_rds("data/openNLP\_LDA.rds")  
  
openNLP\_LDA$plot(out.dir = "LDA\_files/openNLP\_LDAvis")



Screen capture of the interactive visualization of the LDA-Topic model fitted the data preprocessed with openNLP and textstem

## 6 Critical Evaluation

The purpose of this project was to try some of the options for advanced text preprocessing in R. Therefore, the data and modeling served essentially as stand-in dummies. I choose horror books from Project Gutenberg for three reasons: (1) It made the project more reproducible. (2) I have not read any of the books, but I expect some general themes to present across titles making the topic modeling more interesting. (3) The sizes of the text and the corpus itself caused some challenges that the preprocessing frameworks had to be able to deal with. When it comes to the preprocessing, the end products were very similar, as they were supposed to be. What mattered was the user experiences. Just lemmatizing the texts surprisingly easy effective. This could be a good option for beginners if paired with a good list of stop words. SpaCy was rather challenging to set up, and I never got it running without errors. When set up, it was fast and easy to implement. I would recommend this option, especially to users who have experience with Python. I will also note, that It is much easier to set up in Python, so if the rest of the project is in Python, I will wholeheartedly recommend just running the whole project with SpaCy in Python. The openNLP framework was easy to install but rather difficult to code and understand conceptually. I would not call this framework beginner-friendly, especially considering the extended run times, which can be problematic for beginners. For advanced users, my implementation can be improved considerably by finding the optimal chunk size parallelizing the loop, and using faster filtering techniques. Then again, Advanced users may just prefer to do the preprocessing in Python.

## 7 Conclusions

In this project, I have performed advanced text preprocessing in R with the SpaCyR and openNLP frameworks. I found SpaCyR to be difficult to install, but much better to use, when install. Apache’s openNLP framework was conceptually difficult to code and ran much slower. I additionally chunking with *tokenizers* and lemmatization with *textstem* surprisingly easy and effective. Until satisfactory framework is introduced for R, I may just perform preprocessing in Python.

## Appendix

# save.image("workspace.RData") #saving environment  
# load("workspace.RData") #loading environment  
  
#printing package versions

package\_info() %>% # printing package versions...  
 mutate(version = ondiskversion) %>%  
 select(package,version) %>%  
 regulartable() %>%   
 theme\_zebra() %>%   
 autofit() %>%   
 set\_caption("Table 1: R package versions")

Table 1: R package versions

| **package** | **version** |
| --- | --- |
| assertthat | 0.2.1 |
| backports | 1.2.1 |
| base64enc | 0.1.3 |
| broom | 0.7.3 |
| cellranger | 1.1.0 |
| cli | 2.2.0 |
| colorspace | 2.0.0 |
| crayon | 1.3.4 |
| curl | 4.3 |
| data.table | 1.13.6 |
| DBI | 1.1.0 |
| dbplyr | 2.0.0 |
| digest | 0.6.25 |
| dplyr | 1.0.2 |
| ellipsis | 0.3.1 |
| evaluate | 0.14 |
| fansi | 0.4.1 |
| flextable | 0.6.1 |
| float | 0.2.4 |
| forcats | 0.5.0 |
| fs | 1.5.0 |
| gdtools | 0.2.2 |
| generics | 0.1.0 |
| ggplot2 | 3.3.2 |
| glue | 1.4.1 |
| gtable | 0.3.0 |
| gutenbergr | 0.2.0 |
| haven | 2.3.1 |
| hms | 0.5.3 |
| htmltools | 0.5.0 |
| httpuv | 1.5.4 |
| httr | 1.4.2 |
| jsonlite | 1.7.2 |
| knitr | 1.30 |
| koRpus | 0.13.4 |
| koRpus.lang.en | 0.1.4 |
| later | 1.1.0.1 |
| lattice | 0.20.41 |
| LDAvis | 0.3.2 |
| lgr | 0.4.1 |
| lifecycle | 0.2.0 |
| lubridate | 1.7.9.2 |
| magrittr | 2.0.1 |
| Matrix | 1.2.18 |
| mlapi | 0.1.0 |
| modelr | 0.1.8 |
| munsell | 0.5.0 |
| NLP | 0.2.1 |
| officer | 0.3.15 |
| openNLP | 0.2.7 |
| openNLPdata | 1.5.3.4 |
| openNLPmodels.en | 1.5.1 |
| pacman | 0.5.1 |
| pillar | 1.4.7 |
| pkgconfig | 2.0.3 |
| promises | 1.1.1 |
| purrr | 0.3.4 |
| R6 | 2.5.0 |
| rbbt | 0.0.0.9000 |
| Rcpp | 1.0.4.6 |
| readr | 1.4.0 |
| readxl | 1.3.1 |
| reprex | 0.3.0 |
| reticulate | 1.18 |
| RhpcBLASctl | 0.20.137 |
| rJava | 0.9.13 |
| rlang | 0.4.9 |
| rmarkdown | 2.6 |
| rsparse | 0.4.0 |
| rstudioapi | 0.13 |
| rvest | 0.3.6 |
| scales | 1.1.1 |
| servr | 0.21 |
| sessioninfo | 1.1.1 |
| SnowballC | 0.7.0 |
| spacyr | 1.2.1 |
| stringi | 1.5.3 |
| stringr | 1.4.0 |
| sylly | 0.1.6 |
| sylly.en | 0.1.3 |
| systemfonts | 0.3.2 |
| text2vec | 0.6 |
| textstem | 0.1.4 |
| tibble | 3.0.4 |
| tictoc | 1.0 |
| tidyr | 1.1.2 |
| tidyselect | 1.1.0 |
| tidyverse | 1.3.0 |
| tokenizers | 0.2.1 |
| utf8 | 1.1.4 |
| uuid | 0.1.4 |
| vctrs | 0.3.6 |
| withr | 2.3.0 |
| xfun | 0.19 |
| xml2 | 1.3.2 |
| yaml | 2.2.1 |
| zip | 2.1.1 |

## References

Apache Software Foundation. 2014. “openNLP Natural Language Processing Library.” <http://opennlp.apache.org/>.

Benoit, Kenneth, Akitaka Matsuo, and European Research Council (ERC-2011-StG 283794-QUANTESS). 2020. *Spacyr: Wrapper to the “spaCy” “NLP” Library* (version 1.2.1). <https://CRAN.R-project.org/package=spacyr>.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. 2003. “Latent Dirichlet Allocation.” *The Journal of Machine Learning Research* 3 (null): 993–1022.

Dowle, Matt, and Arun Srinivasan. 2020. “Data.table: Extension of `data.frame`.” Manual. <https://CRAN.R-project.org/package=data.table>.

Honnibal, Matthew, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-Strength Natural Language Processing in Python*. Zenodo. <https://doi.org/10.5281/zenodo.1212303>.

Hornik, Kurt. 2019. “openNLP: Apache OpenNLP Tools Interface.” Manual. <https://CRAN.R-project.org/package=openNLP>.

Jelodar, Hamed, Yongli Wang, Chi Yuan, Xia Feng, Xiahui Jiang, Yanchao Li, and Liang Zhao. 2019. “Latent Dirichlet Allocation (LDA) and Topic Modeling: Models, Applications, a Survey.” *Multimedia Tools and Applications* 78 (11): 15169–15211. <https://doi.org/10.1007/s11042-018-6894-4>.

Mullen, Lincoln A., Kenneth Benoit, Os Keyes, Dmitry Selivanov, and Jeffrey Arnold. 2018. “Fast, Consistent Tokenization of Natural Language Text.” *Journal of Open Source Software* 3 (23): 655. <https://doi.org/10.21105/joss.00655>.

Queiroz, Gabriela De, Colin Fay, Emil Hvitfeldt, Os Keyes, Kanishka Misra, Tim Mastny, Jeff Erickson, David Robinson, Julia Silge [aut, and cre. 2020. *Tidytext: Text Mining Using “Dplyr”, “Ggplot2”, and Other Tidy Tools* (version 0.2.6). <https://CRAN.R-project.org/package=tidytext>.

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.

Rinker, Tyler W. 2018. “Textstem: Tools for Stemming and Lemmatizing Text.” Manual. Buffalo, New York. <http://github.com/trinker/textstem>.

Robinson, David. 2020. “Gutenbergr: Download and Process Public Domain Works from Project Gutenberg.” Manual. <https://CRAN.R-project.org/package=gutenbergr>.

RStudio Team. 2020. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, PBC. <http://www.rstudio.com/>.

Selivanov, Dmitriy. 2018. “Topic Modeling.” December 21, 2018. <http://text2vec.org/topic_modeling.html>.

Selivanov, Dmitriy, Manuel Bickel, and Qing Wang. 2020. “Text2vec: Modern Text Mining Framework for R.” Manual. <https://CRAN.R-project.org/package=text2vec>.

Sievert, Carson, and Kenny Shirley. 2015. *LDAvis: Interactive Visualization of Topic Models* (version 0.3.2). <https://CRAN.R-project.org/package=LDAvis>.

Silge, Julia, and David Robinson. 2017. *Text Mining with R: A Tidy Approach*. 1st ed. O’Reilly Media, Inc.

Urbanek, Simon. 2020. “rJava: Low-Level R to Java Interface.” Manual. <https://CRAN.R-project.org/package=rJava>.

Ushey, Kevin, JJ Allaire, and Yuan Tang. 2020. “Reticulate: Interface to ‘Python’.” Manual. <https://CRAN.R-project.org/package=reticulate>.

Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (1): 1–23. <https://doi.org/10.18637/jss.v059.i10>.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D’Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. “Welcome to the Tidyverse.” *Journal of Open Source Software* 4 (43): 1686. <https://doi.org/10.21105/joss.01686>.

Wijffels, Jan, BNOSAC, Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic, Milan Straka, and Jana Straková. 2020. *Udpipe: Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing with the “UDPipe” “NLP” Toolkit* (version 0.8.5). <https://CRAN.R-project.org/package=udpipe>.

Wild, Fridolin, Performance Augmentation Lab (PAL), Oxford Brookes University, and UK. 2020. “CRAN Task View: Natural Language Processing,” December. <https://CRAN.R-project.org/view=NaturalLanguageProcessing>.