Text Mining

Contents

Part 1: to be completed at home before the lab

During this practical, we will cover an introduction to text mining. Topics covered are how to pre-process mined text (in both the tidy approach and using the `tm` package), different ways to visualize this mined text, creating a document-term matrix and an introduction to one type of analysis you can conduct during text mining: text classification. As a whole, there are multiple ways to mine & analyze text within `R`. However, for this practical we will discuss some of the techniques covered in the `tm` package and in the `tidytext` package, based upon the `tidyverse`.

You can download the student zip including all needed files for this lab here.

Note: the completed homework has to be **handed in** on Black Board and will be **graded** (pass/fail, counting towards your grade for individual assignment). The deadline is two hours before the start of your lab. Hand-in should be a **PDF** file. If you know how to knit pdf files, you can hand in the knitted pdf file. However, if you have not done this before, you are advised to knit to a html file as specified below, and within the html browser, 'print' your file as a pdf file.

For this practical, you will need the following packages:

```r
# General Packages
library(tidyverse)

# Text Mining
library(tidytext)
library(gutenbergr)
library(SnowballC)
library(wordcloud)
library(textdata)
library(tm)
library(stringi)
library(e1071)
library(rpart)
```

For the first part of the practical, we will be using text mined through the Gutenberg Project; briefly this project contains over 60,000 freely accessible eBooks, which through the package `gutenberger`, can be easily accessed and perfect for text mining and analysis.

We will be looking at several books from the late 1800s, in the mindset to compare and contrast the use of language within them. These books include:

- *Alice's Adventures in Wonderland by Lewis Carroll*
- *The Picture of Dorian Gray by Oscar Wilde*
- *Magic of Oz by Frank Lyman Baum*
- *The Strange Case of Dr. Jekyll and Mr. Hyde by Robert Louis Stevenson*

Despite being old books, they are still popular and hold cultural significance in TV, Movies and the English Language. To access this novel suitable for this practical the following function should be used:

```r
AAIWL <- gutenberg_download(28885) # 28885 is the eBook number of Alice in Wonderland
PODG  <- gutenberg_download(174)   # 174 is the eBook number of The Picture of Dorian 
MOz  <- gutenberg_download(419)   # 419 is the eBook number of Magic of Oz
SCJH  <- gutenberg_download(43)    # 43 is the eBook number of Dr. Jekyll and Mr. Hyde
```

After having loaded all of these books into your working directory (using the code above), examine one of these books using the `View()` function. When you view any of these data frames, you will see that these have an extremely *messy* layout and structure. As a result of this complex structure means that conducting *any* analysis would be extremely challenging, so pre-processing must be undertaken to get this into a format which is usable.

Pre-Processing Text: Tidy approach

In order for text to be used effectively within statistical processing and analysis; it must be pre-processed so that it can be uniformly examined. Typical steps of pre-processing include:

- Tokenization
- Removing numbers
- Converting to lowercase
- Removing stop words
- Removing punctuation
- Stemming

These steps are important as they allow the text to be presented uniformly for analysis (but remember we do not always need all of them); within this practical we will discuss how to undergo some of these steps.

Step 1: Tokenization, un-nesting Text

When we previously looked at this text, as we discovered it was extremely *messy* with it being attached one line per row in the data frame. As such, it is important to un-nest this text so that it attaches one word per row.

Before un-nesting text, it is useful to make a note of aspects such as the line which text is on, and the chapter each line falls within. This can be important when examining anthologies or making chapter comparisons as this can be specified within the analysis.

In order to specify the line number and chapter of the text, it is possible to use the `mutuate` function from the `dplyr` package.

---

1. **Apply the code below, which uses the `mutate` function, to add line numbers and chapter references one of the books. Next, use the `View()` function to examine how this has changed the structure.**

```r
library(tidyverse)

# Text Mining
library(tidytext)
library(gutenbergr)
library(SnowballC)
library(wordcloud)
library(textdata)
library(tm)
library(stringi)
library(e1071)
library(rpart)


# Example for Alice's Adventures in Wonderland
tidy_AAIWL <- AAIWL %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]", ignore_case = TR

# View the modified structure
View(tidy_AAIWL)
```

From this, it is now possible to pass the function `unnest_tokens()` in order to split apart the sentence string, and apply each word to a new line. When using this function, you simply need to pass the arguments, `word` (as this is what you want selecting) and `text` (the name of the column you want to unnest).

---

2. **Apply unnest_tokens to your tidied book to unnest this text. Next, once again use the `View()` function to examine the output.**

Hint: As with Question 1, ensure to use the piping operator (`%>%`) to easily apply the function.

```r
# Tokenize the text
tidy_AAIWL <- tidy_AAIWL %>%
  unnest_tokens(word, text)

# View the tokenized structure
View(tidy_AAIWL)
```

This results in one word being linked per row of the data frame. The benefit of using the `tidytext` package in comparison to other text mining packages, is that this automatically applies some of the basic steps to pre-process your text, including removing of capital letters, inter-word punctuation and numbers. However additional pre-processing is required.

Intermezzo: Word clouds

Before continuing the pre-processing process, let's have a first look at our text by making a simple visualization using word clouds. Typically these word clouds visualize the frequency of words in a text through relating the size of the displayed words to frequency, with the largest words indicating the most common words.

To plot word clouds, we first have to create a data frame containing the word frequencies.

---

3. **Create a new data frame, which contains the frequencies of words from the unnested text. To do this, you can make use of the function `count()`.**

Hint: As with Question 1, ensure to use the piping operator (`%>%`) to easily apply the function.

```r
# Create a data frame with word frequencies
word_freq <- tidy_AAIWL %>%
  count(word, sort = TRUE)


word_freq
```

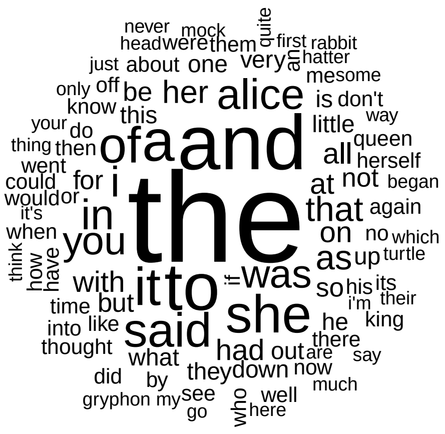
```
## # A tibble: 2,919 x 2
##    word      n
##    <chr> <int>
##  1 the    1676
##  2 and     899
##  3 to      757
##  4 a       649
##  5 she     543
##  6 it      539
##  7 of      523
##  8 said    466
##  9 alice   391
## 10 i       391
## # i 2,909 more rows
```

---

4. **Using the `wordcloud()` function, create a word cloud for your book text. Use the argument `max.words` within the function to set the maximum number of words to be displayed in the word cloud.**

Hint: As with Question 1, ensure to use the piping operator (%>%) to easily apply the function. *Note*: Ensure to use the function `with()`, is used after the piping operator.

```
# Generate the word cloud
wordcloud(words = word_freq$word, freq = word_freq$n, max.words = 100, random.order = FA
```



---

5. **Can you easily tell what text each word clouds come from, based on the popular words which occur?**

```
# It is more-less challenging to determine the text source as many common stop words l
```

Part 2: to be completed during the lab

Pre-Processing Text: Tidy approach - continued

Step 2: Removing stop words

As discussed within the lecture, stop words are words in any language which have little or no meaning, and simply connect the words of importance. Such as *the*, *a*, *also*, *as*, *were*... etc.

To understand the importance of removing these stop words, we can simply do a comparison between the text which has had them removed and those which have not been.

To remove the stop words, we use the function `anti_join()`. This function works through *un-joining* this table based upon the components, which when passed with the argument `stop_words`, which is a table containing these words across three lexicons. This removes all the stop words from the presented data frame.

---

6. **Use the function `anti_join()` to remove stop words from your tidied text attaching it to a new data frame.**

Hint: As with Question 1, ensure to use the piping operator (%>%) to easily apply the function.

```r
tidy_AAIWL_nostop <- tidy_AAIWL %>%
  anti_join(stop_words)
```

In order to examine the impact of removing these filler words, we can use the `count()` function to examine the frequencies of different words. This when sorted, will produce a table of frequencies in descending order. An other option is to redo the wordclouds on the updated data frame containing the word counts of the tidied book text without stop words.

---

7. **Use the function `count()` to compare the frequencies of words in the dataframes containing the tidied book text with and without stop words (use `sort = TRUE` within the `count()` function), or redo the wordclouds. Do you notice a difference in the (top 10) words which most commonly occur in the text?**

Hint: As with Question 1, ensure to use the piping operator (%>%) to easily apply the function.

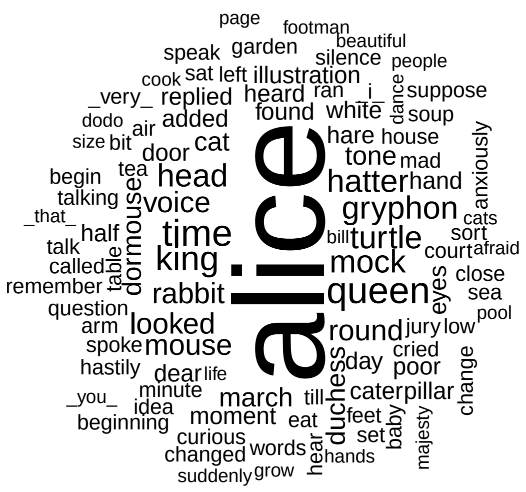
```r
# Count word frequencies before and after removing stop words
word_freq_with_stop <- tidy_AAIWL %>%
  count(word, sort = TRUE)

word_freq_without_stop <- tidy_AAIWL_nostop %>%
  count(word, sort = TRUE)

# Display top 10 words with and without stop words
list(
  with_stop = word_freq_with_stop %>% top_n(10),
  without_stop = word_freq_without_stop %>% top_n(10)
)
```

```
## $with_stop
## # A tibble: 10 x 2
##    word      n
##    <chr> <int>
##  1 the    1676
##  2 and     899
##  3 to      757
##  4 a       649
##  5 she     543
##  6 it      539
##  7 of      523
##  8 said    466
##  9 alice   391
## 10 i       391
##
## $without_stop
## # A tibble: 10 x 2
##    word        n
##    <chr>   <int>
##  1 alice     391
##  2 queen      73
##  3 time       73
##  4 king       61
##  5 mock       59
##  6 turtle     58
##  7 gryphon    55
##  8 hatter     55
##  9 head       53
## 10 rabbit     49
```

```r
wordcloud(words = word_freq_without_stop$word, freq = word_freq_without_stop$n, max.wor
```



Vector space model: document-term matrix

In this part of the practical we will build a text classification model for a multiclass classification task. To this end, we first need to perform text preprcessing, then using the idea of vector space model, convert the text data into a document-term (dtm) matrix, and finally train a classifier on the dtm matrix.

The data set used in this part of the practical is the BBC News data set. You can use the provided "news_dataset.rda" for this purpose. This data set consists of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004 to 2005. These areas are:

- Business
- Entertainment
- Politics
- Sport
- Tech

---

8. **Use the code below to load the data set and inspect its first rows.**

```r
load("data/news_dataset.rda")
head(df_final)
```

```
##   File_Name
## 1   001.txt
## 2   002.txt
## 3   003.txt
## 4   004.txt
## 5   005.txt
## 6   006.txt
##
## 1 Ad sales boost Time Warner profit\n\nQuarterly profits at US media giant TimeWarner
## 2
## 3
## 4
## 5
## 6
##   Category Complete_Filename
## 1 business  001.txt-business
## 2 business  002.txt-business
## 3 business  003.txt-business
## 4 business  004.txt-business
## 5 business  005.txt-business
## 6 business  006.txt-business
```

---

9. **Find out about the name of the categories and the number of observations in each of them.**

```
df_final %>%
  group_by(Category) %>%
  summarize(count = n())
```

```
## # A tibble: 5 x 2
##   Category      count
##   <chr>         <int>
## 1 business        510
## 2 entertainment   386
## 3 politics        417
## 4 sport           511
## 5 tech            401
```

---

10. **Convert the data set into a document-term matrix using the function `DocumentTermMatrix()` and subsequently use the `findFreqTerms()` function to keep the terms which their frequency is larger than 10. A start of the code is given below. It is also a good idea to apply some text preprocessing, for this inspect the `control` argument of the function `DocumentTermMatrix()` (e.g., convert the words into lowercase, remove punctuations, numbers, stopwords, and whitespaces).**

```
## set the seed to make your partition reproducible
set.seed(123)

df_final$Content <- iconv(df_final$Content, from = "UTF-8", to = "ASCII", sub = "")

docs <- Corpus(VectorSource(df_final$Content))

# alter the code from here onwards
dtm <- DocumentTermMatrix(...
                          ))
```

```
# Set the seed to make your partition reproducible
set.seed(123)

df_final$Content <- iconv(df_final$Content, from = "UTF-8", to = "ASCII", sub = "")

docs <- Corpus(VectorSource(df_final$Content))

# Create Document-Term Matrix
dtm <- DocumentTermMatrix(docs, control = list(
  tolower = TRUE,
  removePunctuation = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  stripWhitespace = TRUE
))

# Keep terms with frequency greater than 10
freq_terms <- findFreqTerms(dtm, lowfreq = 10)
dtm <- dtm[, freq_terms]
```

---

11. **Partition the original data into training and test sets with 80% for training and 20% for test.**

```r
# Partition the data
set.seed(123)
train_indices <- sample(seq_len(nrow(df_final)), size = 0.8 * nrow(df_final))

train_data <- df_final[train_indices, ]
test_data <- df_final[-train_indices, ]
```

12. **Create separate document-term matrices for the training and the test sets using the previous frequent terms as the input dictionary and convert them into data frames.**

```r
# Create DTM for training and test sets
train_dtm <- DocumentTermMatrix(Corpus(VectorSource(train_data$Content)), control = lis
  dictionary = freq_terms,
  tolower = TRUE,
  removePunctuation = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  stripWhitespace = TRUE
))

test_dtm <- DocumentTermMatrix(Corpus(VectorSource(test_data$Content)), control = list(
  dictionary = freq_terms,
  tolower = TRUE,
  removePunctuation = TRUE,
  removeNumbers = TRUE,
  stopwords = TRUE,
  stripWhitespace = TRUE
))

# Convert to data frames
train_dtm_df <- as.data.frame(as.matrix(train_dtm))
test_dtm_df <- as.data.frame(as.matrix(test_dtm))
```

13. **Use the `cbind` function to add the categories to the train_dtm data and name the column cat.**

```r
train_dtm_df$cat <- train_data$Category
```

14. **Use the `rpart()` function from the `rpart` library to fit a classification tree on the training data set. Evaluate your model on the training and test data. What is the accuracy of your model?**

```r
# Train classification tree
tree_model <- rpart(cat ~ ., data = train_dtm_df, method = "class")

# Predict on training data
train_preds <- predict(tree_model, newdata = train_dtm_df, type = "class")
train_accuracy <- mean(train_preds == train_dtm_df$cat)

# Predict on test data
test_preds <- predict(tree_model, newdata = test_dtm_df, type = "class")
test_accuracy <- mean(test_preds == test_data$Category)

list(
  train_accuracy = train_accuracy,
  test_accuracy = test_accuracy
)
```

```
## $train_accuracy
## [1] 0.7713483
##
## $test_accuracy
## [1] 0.752809
```