Lesson 13 Intro to Text Mining

Lusine Zilfimian

May 11 (Monday), 2020

Contents

- Intro Glossary
- String Manipulation
- Stringr vs Base
- RegEx
- Intro to TM

Bibliography

- See on Github: Handling and Processing Strings
- Character Strings - Chapter 2
- String Manipulation + stringr - Chapter 3, 4
- RegEx - Chapter 5

Intro Glossary

- A **token** is a meaningful unit of text, such as a word, that we are interested in using for analysis, and **tokenization** is the process of splitting text into tokens.
- **Semantics** means the meaning and interpretation of words, signs, and sentence structure.
- In **semantic analysis** we will care about the place and order of the words

Intro Glossary

- String manipulation - series of functions used to extract information from text variables.
- **Regex** is set of commands used to match a family of text (alphanumeric, digits, words) to detect string sequences in a large text data.
- *Regular expressions* to do more complicated tasks such as extract email IDs or date from a set of text.
- String manipulation functions are not customized however we can customize regular expressions in any way we want.

String Manipulation

- We can create strings with either single quotes or double quotes.
- Unlike other languages, there is no difference in behavior.
- To include a literal single or double quote in a string you can use \ to escape it.

```r
(x <- "This is my first character!") # double quotes
```

```
## [1] "This is my first character!"
```

```r
class(x) == typeof(x)
```

```
## [1] TRUE
```

```r
(x <- 'This is my second string') # single quotes
```

```
## [1] "This is my second string"
```

```r
(x <- "Why do we need the 'python'?")
```

```
## [1] "Why do we need the 'python'?"
```

```r
(x <- 'Why do we need the "python"?')
```

Working with strings

- If you try to include single quotes within single quotes, or double quotes within double quotes, it will not work

```
# Something went wrong
"Why call langauge "python" ? "
'Why call langauge 'python' ? '
```

```
## Error: <text>:2:21: unexpected symbol
## 1: # Something went wrong
## 2: "Why call langauge "python
##                         ^
```

- \ escaping character

```
(x <- 'Why do we need the \'python\'?')
```

```
## [1] "Why do we need the 'python'?"
```

```
is.character(x)
```

```
## [1] TRUE
```

- Paste() and its arguments
- Takes or more R objects, conver them to character and then concatenates them to character (one or more)

```
paste("Life of", pi)
```

```
## [1] "Life of 3.14159265358979"
```

- Element wise concatenation of two vectors

```
x <- 1:26
(a <- LETTERS[x[1:13]])
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"
```

```
(b <- letters[x[14:26]])
```

```
##  [1] "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```r
(char1 <- paste(x, c(a, b)))
```

```
##  [1] "1 A"  "2 B"  "3 C"  "4 D"  "5 E"  "6 F"  "7 G"  "8 H"  "9 I"  "10
## [11] "11 K" "12 L" "13 M" "14 n" "15 o" "16 p" "17 q" "18 r" "19 s" "20
## [21] "21 u" "22 v" "23 w" "24 x" "25 y" "26 z"
```

```r
length(char1)
```

```
## [1] 26
```

- sep = The element which separates every term.
- collapse = The element which separates every result.

```r
paste(x, c(a, b), sep = " - is - ") # optional - Define your own separator
```

```
##  [1] "1 - is - A"  "2 - is - B"  "3 - is - C"  "4 - is - D"  "5 - is - E
##  [6] "6 - is - F"  "7 - is - G"  "8 - is - H"  "9 - is - I"  "10 - is -
## [11] "11 - is - K" "12 - is - L" "13 - is - M" "14 - is - n" "15 - is -
## [16] "16 - is - p" "17 - is - q" "18 - is - r" "19 - is - s" "20 - is -
## [21] "21 - is - u" "22 - is - v" "23 - is - w" "24 - is - x" "25 - is -
## [26] "26 - is - z"
```

```r
(char2 <- paste(x, c(a, b), sep = " - is - ", collapse = " , ")) # optional
```

Basic Sting manipulations (Base)

- nchar() counts the number of characters (including whitespaces)
- for single character

```
nchar("Here are 16 chars:)")
```

```
## [1] 19
```

```
char1
```

```
##  [1] "1 A"  "2 B"  "3 C"  "4 D"  "5 E"  "6 F"  "7 G"  "8 H"  "9 I"  "10 
## [11] "11 K" "12 L" "13 M" "14 n" "15 o" "16 p" "17 q" "18 r" "19 s" "20 
## [21] "21 u" "22 v" "23 w" "24 x" "25 y" "26 z"
```

```
length(char2)
```

```
## [1] 1
```

```
nchar(char1)
```

```
##  [1] 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

- We can convert all characters of our string to lower case with the `tolower` command
- and to upper case with `toupper`

```
toupper("Here are 16 chars:)")
```

```
## [1] "HERE ARE 16 CHARS:)"
```

```
tolower(toupper("Here are 16 chars:)"))
```

```
## [1] "here are 16 chars:)"
```

```
chartr(old = "is", new = "is not", char2) # the size should be equal
```

```
## [1] "1 - is - A , 2 - is - B , 3 - is - C , 4 - is - D , 5 - is - E , 6
```

Substring + replacement based on the positions

- substr() and substring()
- These function can be used to overwrite or replace a part of character string.

```
x <- "Lusine Zilfimian"
substr(x, start = 8, stop = 7 + 5)
```

```
## [1] "Zilfi"
```

```
substring(x, first = 8) #  last = 1000000L
```

```
## [1] "Zilfimian"
```

```
substr(x, start = 14, stop = 14) <- "y"
x
```

```
## [1] "Lusine Zilfimyan"
```

Replacement and detection

```r
sub("Lusine Zilfimian", pattern = "i", replacement = "y") # find and replac
```

```
## [1] "Lusyne Zilfimian"
```

```r
gsub("Lusine Zilfimian", pattern = "i", replacement = "y")
```

```
## [1] "Lusyne Zylfymyan"
```

```r
grep(pattern = "Lusine", x =  c("Lusine Zilfimian", "David", "Mar")) #detec
```

```
## [1] 1
```

```r
grepl(pattern = "Lusine", x = c("Lusine Zilfimian", "David", "Mar"))
```

```
## [1]  TRUE FALSE FALSE
```

Abbreviations

- Abbreviate strings to at least minlength characters, such that they remain unique (if they were), unless strict = TRUE.

```
x <- c("Lusin ", " Lusine ", "Lusine Zilfimian", "Lusine Zilfi", "Lus - ",

(abb1 <- abbreviate(x, minlength = 4))
```

```
##            Lusin             Lusine Lusine Zilfimian       Lusine Zilfi
##          "Lusin"           "Lusine"         "LsnZlfm"         "LusnZlf"
##            Lus -        Lusine 1995
##           "Lus-"             "L199"
```

```
(abb2 <- abbreviate(x, minlength = 4, strict = T))
```

```
##            Lusin             Lusine Lusine Zilfimian       Lusine Zilfi
##           "Lusn"             "Lusn"            "LsnZ"            "LsnZ"
##            Lus -        Lusine 1995
##           "Lus-"             "L199"
```

Why is base not enough?

- The objects NULL and character(0) have zero length, yet when included inside paste() they are treated as an empty string:

```
paste("This", "is", NULL, character(0), "value")
```

```
## [1] "This is   value"
```

```
length(NULL)
```

```
## [1] 0
```

```
length(character(0))
```

```
## [1] 0
```

```
stringr::str_c("This", "is", NULL, character(0), "value", sep = " ")
```

```
## [1] "This is value"
```

- str_c() – zero argument are silently removed

Stingr

- All functions start with str_
- All functions take a vector of strings as the first argument
- `str_sub` works the same way as `substr()` but allows for negative subsetting as well

```
library(stringr)
str_sub("Joe Satriani", start = 5)
```

```
## [1] "Satriani"
```

```
str_sub("Joe Satriani", start = 5, end = 5+4)
```

```
## [1] "Satri"
```

```
str_sub("Joe Satriani", start = -8, end = -4)
```

```
## [1] "Satri"
```

- `str_detect()` - For detecting whether a pattern is present (or absent) in a string vector. You get a TRUE if a match is detected in a string, FALSE otherwise
- `str_subset()` - Keep strings matching a pattern
- `str_count()` - Count the number of matches in a string
- `str_split()` - Similar to `strsplit()` to separate a character vector into a number of pieces
- `str_replace()` - For replacing the first occurrence of a matched pattern in a string
- `str_exctact()` - For extracting a string containing a pattern

detect - bolean, extract – only pattern, subset – all the text.

Examples

```r
(geners <- c("Action, Adventure, Comedy", "Comedy",
  "Comedy, Drama, Drama, Romance", "Crime, Drama, History"))
```

```
## [1] "Action, Adventure, Comedy"     "Comedy"
## [3] "Comedy, Drama, Drama, Romance" "Crime, Drama, History"
```

```r
str_detect(string = geners,  pattern = "Drama")
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

```r
str_subset(string = geners,  pattern = "Action")
```

```
## [1] "Action, Adventure, Comedy"
```

```r
str <-c("123abd", "lz@yahoo.com", "Name567cd", "abc5.00", "lusinezilfimian@
str_subset(str, pattern="@yahoo.com")
```

```
## [1] "lz@yahoo.com"                 "lusinezilfimian@yahoo.com"
```

```r
str_count(string = geners,  pattern = "Drama")
```

```
## [1] 0 0 2 1
```

Examples

```
str_replace(string = geners, pattern = ",", replacement = " &")
```

```
## [1] "Action & Adventure, Comedy"      "Comedy"
## [3] "Comedy & Drama, Drama, Romance" "Crime & Drama, History"
```

```
sub(x = geners, pattern = ",", replacement = " &") # from base
```

```
## [1] "Action & Adventure, Comedy"      "Comedy"
## [3] "Comedy & Drama, Drama, Romance" "Crime & Drama, History"
```

```
gsub(x = geners, pattern = ",", replacement = " &") # from base
```

```
## [1] "Action & Adventure & Comedy"       "Comedy"
## [3] "Comedy & Drama & Drama & Romance" "Crime & Drama & History"
```

```
str_replace_all(string = geners, pattern = ",", replacement = " &")
```

```
## [1] "Action & Adventure & Comedy"       "Comedy"
## [3] "Comedy & Drama & Drama & Romance" "Crime & Drama & History"
```

Examples

```r
str_extract(string = geners, pattern = "Drama")
```

```
## [1] NA      NA      "Drama" "Drama"
```

```r
str_extract_all(string = geners, pattern = "Drama", simplify = T)
```

```
##      [,1]    [,2]
## [1,] ""      ""
## [2,] ""      ""
## [3,] "Drama" "Drama"
## [4,] "Drama" ""
```

Let's consider the example

```r
movies <- read.csv('movies3.csv', stringsAsFactors = F)
```

How many romcom movies are there?

```r
movies$Comedy <- str_detect(string = movies$Genre, pattern = "Comedy")
movies$rom <- str_detect(string = movies$Genre, pattern = "Romance")
table(movies$Comedy,movies$rom)
```

```
##
##          FALSE TRUE
##    FALSE  1478  202
##    TRUE    886  319
```

Split string into pieces

```r
(gen_m <- str_split(geners, ",", simplify = T))
```

```
##        [,1]     [,2]         [,3]       [,4]
## [1,] "Action" " Adventure" " Comedy"  ""
## [2,] "Comedy" ""           ""         ""
## [3,] "Comedy" " Drama"     " Drama"   " Romance"
## [4,] "Crime"  " Drama"     " History" ""
```

```r
trimws("  Lus     ine    ") # trim whitespaces
```

```
## [1] "Lus     ine"
```

- Sequance of characters that define a search pattern
- For instance, one could have the following pattern: 2 digits, 2 letters and 4 digits
- ?regex - to see more

Form of Regular Expressions

- 1. Literal characters which are characters that match themselves
- 2. Metacharacters - any character that is not a literal character . ^ $ + * ? | (), {}, [], -
- 3. Sequences
- 4. Character classes
- 5. Quantifiers
- 6. POSIX character classes
- If you want to match the metacharacter you need to use \ (escapes) before the metacharacter

Literal and Metacharacters

- Literal character - the simplest form of regular expressions are those that match a single charact
- Metacharacters - special characters that have a reserved status:

```
str <-c("Lusine1995^", "Say$1995-02(", "David.", "M*77", "M&95", "1995.27")
str_subset(string = str,  pattern = "Lusine") # literal
```

```
## [1] "Lusine1995^"
```

```
str_subset(string = str,  pattern = "Lusine|David|Say")
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."
```

```
str_subset(string = "money$",  pattern = "money\\$")
```

```
## [1] "money$"
```

```
str_subset(string = "This is money1",  pattern = "money$")
```

```
## character(0)
```

Metacharacters

```r
str_subset(string = str,  pattern = ".") # meta
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."       "M*77"
## [5] "M&95"         "1995.27"
```

```r
str_subset(string = "\n",  pattern = ".") # meta
```

```
## character(0)
```

```r
str_subset(string = " ",  pattern = ".") # meta
```

```
## [1] " "
```

```r
str_subset(string = "",  pattern = ".") # meta
```

```
## character(0)
```

```r
str_subset(string = str,  pattern = "\\.") # literal
```

```
## [1] "David."  "1995.27"
```

Metacharacters

```r
str_subset(string = str,  pattern = "^M") # meta
```

```
## [1] "M*77" "M&95"
```

```r
str_subset(string = str,  pattern = "\\^") # literal
```

```
## [1] "Lusine1995^"
```

```r
str_subset(string = str,  pattern = "\\$") # literal
```

```
## [1] "Say$1995-02("
```

Sequences

```r
str_subset(string = c("123", "4L", "M"),  pattern = "\\d") # [0-9]
```

```
## [1] "123" "4L"
```

```r
str_subset(string = c("123", "4L", "M"),  pattern = "\\D") # [^0-9]
```

```
## [1] "4L" "M"
```

```r
str_subset(string = c(" ", "l    ", "l"),  pattern = "\\s")
```

```
## [1] " "     "l    "
```

```r
str_subset(string = c(" ", "l ", "l"),  pattern = "\\S")
```

```
## [1] "l " "l"
```

```r
str_subset(string = c(" ", "l ", "l"),  pattern = "\\w") # word
```

```
## [1] "l " "l"
```

```r
str_subset(string = c(" ", "l ", "l"),  pattern = "\\W") # non-word
```

Character classes

```r
str_subset(string = str,  pattern = "[0-9]")
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "M*77"          "M&95"
## [5] "1995.27"
```

```r
str_subset(string = str,  pattern = "[f-z]")
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."
```

```r
str_subset(string = str,  pattern = "[a-zA-Z]")
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."        "M*77"
## [5] "M&95"
```

```r
str_subset(string = str,  pattern = "[aeuio]")
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."
```

```r
str_subset(string = c("AI", "ai"),  pattern = "[AEIOU]")
```

```
## [1] "AI"
```

Special meaning of ^

- Not in specified character

```
str_subset(string = c("ou", "lk"),  pattern = "[^aeuio]")
```

```
## [1] "lk"
```

```
str_subset(string = c("ou", "19", "-"),  pattern = "[a-zA-Z0-9]")
```

```
## [1] "ou" "19"
```

```
str_subset(string = c("ou", "19", "-"),  pattern = "[a-zA-Z]")
```

```
## [1] "ou"
```

```
str_subset(string = c("ou", "19", "-"),  pattern = "[^0-9]")
```

```
## [1] "ou" "-"
```

```
str_subset(string = str,  pattern = "[^1-9]")
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."       "M*77"
## [5] "M&95"         "1995.27"
```

Quantifiers

```
str_subset(string = c(" ", "l   ", "l"),  pattern = "\\s+")
```

```
## [1] " "    "l   "
```

```
str_subset(string = c(" ", "l   ", "l"),  pattern = "\\s*")
```

```
## [1] " "    "l   " "l"
```

```
str_subset(string = c(" ", "l   ", "l"),  pattern = "\\s?")
```

```
## [1] " "    "l   " "l"
```

```
str_subset(string = c(" ", "l   ", "l"),  pattern = "\\s{3}")
```

```
## [1] "l   "
```

```
str_subset(string = c(" ", "l   ", "l"),  pattern = "\\s{2,}")
```

```
## [1] "l   "
```

```
str_subset(string = c(" ", "l   ", "l"),  pattern = "\\s{1,3}")
```

POSIX character classes

```r
str_subset(string = str,  pattern = "[:alnum:]") # [a-zA-Z0-9]
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."       "M*77"
## [5] "M&95"         "1995.27"
```

```r
str_subset(string = str,  pattern = "[:alpha:]") # [a-zA-Z]
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."       "M*77"
## [5] "M&95"
```

```r
str_subset(string = str,  pattern = "[:digit:]") # [0-9]
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "M*77"         "M&95"
## [5] "1995.27"
```

```r
str_subset(string = str,  pattern = "[:lower:]") # [a-z]
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."
```

POSIX character classes

```r
str_subset(string = str,  pattern = "[:upper:]") # [A-Z]
```

```
## [1] "Lusine1995^"  "Say$1995-02(" "David."       "M*77"
## [5] "M&95"
```

```r
str_subset(string = str,  pattern = "[:space:]") # [\f\n\r\t\v]
```

```
## character(0)
```

```r
str_subset(string = str,  pattern = "[:punct:]")
```

```
## [1] "Say$1995-02(" "David."       "M*77"         "M&95"
## [5] "1995.27"
```

```r
str_subset(string = c("grey", "gray"), pattern = "gr(e|a)y")
```

```
## [1] "grey" "gray"
```

How many wins and nominations are there?

```r
movies$Awards[1:5]
```

```
## [1] "4 wins & 8 nominations."  "7 wins & 17 nominations."
## [3] "1 win & 2 nominations."   "9 wins & 28 nominations."
## [5] "2 wins & 67 nominations."
```

```r
movies$awards_num <- str_replace_all(movies$Awards, pattern="[^0-9]",
  replacement = " ")

movies$awards_num <- str_replace_all(movies$awards_num, pattern="\\s+",
  replacement = " ")
```

How many wins and nominations are there?

```
movies$awards_num <- trimws(movies$awards_num)
movies$awards_num[1:20]
```

```
##  [1] "4 8"        "7 17"       "1 2"        "9 28"       "2 67"       "1"
##  [7] "1 87 171" "2 3"        "2 4"        "1 5 7"      "1 1"        "1 3"
## [13] "1 1 3"      "1 2"        "2 11 5"     "1 3 31"     "2 2"        "2 2"
## [19] "1 3"        "1 5"
```

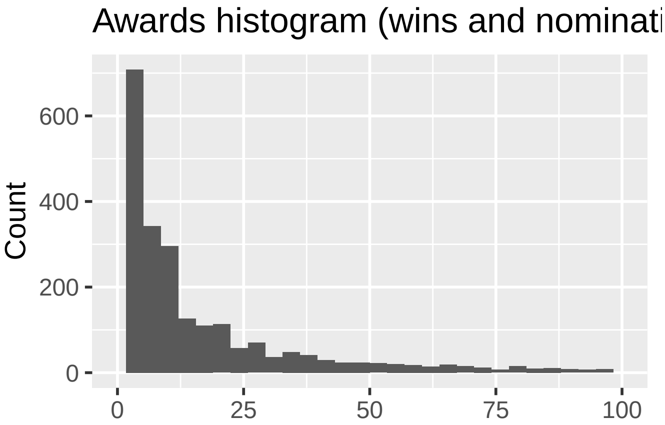
```
x1 <- str_split(movies$awards_num, pattern = " ",simplify = T)
x1 <- apply(x1, 2, as.numeric)
x1 <- rowSums(x1, na.rm=T)
x1[1:20]
```

```
##  [1]  12  24   3  37  69   1 259   5   6  13   2   4   5   3  18  35   4
## [18]   4   4   6
```

Visualization

```
movies$awards_num <- x1
ggplot(movies, aes(x = awards_num)) + geom_histogram() +
  xlim(c(0,100)) + labs(title = "Awards histogram (wins and nominations)",
    x = "Awards", y = "Count")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Wins only

- Possible strings are ...

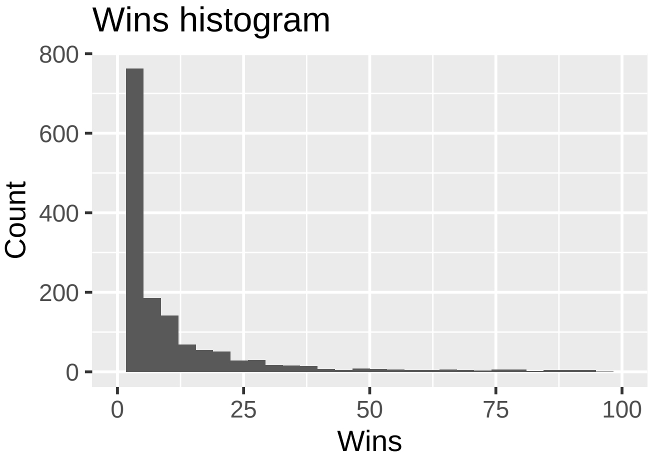
```
m <- movies$Awards[1:20]
m1 <- str_extract_all(movies$Awards, pattern="[0-9]+\\swin", simplify = T)
m1 <- str_remove_all(m1, pattern="[a-zA-Z]")
m1 <- as.numeric(m1)
movies$Wins <- m1
head(m1)
```

```
## [1]  4  7  1  9  2 NA
```

Visualization

```
ggplot(movies, aes(x = Wins)) + geom_histogram() +
  labs(title = "Wins histogram", x = "Wins", y = "Count")+
  xlim(c(0,100))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Task

- Read the file Text.txt without warnings. Use regular expressions to solve the problems:
- Show only the first line. Eliminate the periods before and after the words, but not periods inside the values and at the end of the statement.
- Show only the second line. Convert the GDP of Armenia into proper numbers, create a data frame with variables year and GDP.

Solution

```
readLines("Text.txt", warn=FALSE)
```

```
## [1] "..The. ...economy. ...of ..Armenia ...grew ..by. 5.2% in ..2018. an
## [2] "The GDP of Armenia was ?4,000,722.0 millions in 2012, ?4,555,638.2 
## [3] "Even in this small mountainous country, there are places where they
## [4] "Armenian cuisine is wonderful and varied, with only kharavts and sh
## [5] "The most popular tours in Armenia are tours to mountain lakes among
```

```
(text1 <- readLines("Text.txt", warn=FALSE)[1])
```

```
## [1] "..The. ...economy. ...of ..Armenia ...grew ..by. 5.2% in ..2018. an
```

```
words <- unlist(str_split(text1, pattern = " "))
words <- str_replace_all(string = words, pattern = "^\\.+", replacement = "
words <- str_replace_all(string = words, pattern = "\\.+$", replacement = "
paste(words, sep = " ", collapse = " ")
```

```
## [1] "The economy of Armenia grew by 5.2% in 2018 and reached a nominal G
```

Solution

```
(text2 <- readLines("Text.txt", warn=FALSE)[2])
```

```
## [1] "The GDP of Armenia was ?4,000,722.0 millions in 2012, ?4,555,638.2 r
```

```
words2 <- unlist(str_split(text2, pattern = " "))
numbers <- str_subset(words2, pattern = "[0-9]")
numbers <- str_replace(numbers, pattern = "\\.[0-9]", replacement = "")
numbers <- str_replace_all(numbers, pattern = "^\\?|,$|\\.$|[,]", replaceme
df = data.frame(matrix(numbers, ncol = 2, byrow = T))
colnames(df)= c("gdp", "year")
df
```

```
##       gdp year
## 1 4000722 2012
## 2 4555638 2013
## 3 4828626 2014
## 4 5032089 2015
```

Text Mining

- We will use the tm text mining package.

```
library(tm)
```

- Load the data into a Corpus (a collection of documents) which is the main data structure used by `tm`
- In order to create a `VCorpus` using `tm`, we need to pass a "Source" object as a paramter to the 'VCorpus
- First you need to create a `VectorSource` - A vector source interprets each element of the vector x as a document
- VectorSource is for only character vectors.

```
text <- c("The best guitar models are Gibson and Fender!",
  "My guitars are the best guitars.",
  "Are these guitars yours?")
vs <- VectorSource(text)
corpus <- VCorpus(vs)
```

- use [[ ]] to access single element in the corpus

```
inspect(corpus[[3]]) # like print
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 24
##
## Are these guitars yours?
```

- Access content directly for the second document

```
corpus[[2]][1]
```

```
## $content
## [1] "My guitars are the best guitars."
```

```r
corpus[[2]][2]
```

```
## $meta
##   author       : character(0)
##   datetimestamp: 2020-05-20 19:41:38
##   description  : character(0)
##   heading      : character(0)
##   id           : 2
##   language     : en
##   origin       : character(0)
```

- Metadata is used to annotate text documents or whole corpora with additional information. The easiest way to accomplish this with tm is to use the meta() function

```r
meta(corpus, tag = 'language')
```

```
## $`1`
## [1] "en"
##
## $`2`
## [1] "en"
##
```

- Create new metadata tag class for the first document and assign Guitar to it
- Same for IDs

```
meta(corpus[[1]], tag="class") <- "Guitar"
corpus[[1]][2]
```

```
## $meta
##   author       : character(0)
##   datetimestamp: 2020-05-20 19:41:38
##   description  : character(0)
##   heading      : character(0)
##   id           : 1
##   language     : en
##   origin       : character(0)
##   class        : Guitar
```

```
meta(corpus, tag = "ids") <- c("DOC1", "DOC2", "DOC3")
meta(corpus, "ids")
```

```
##    ids
## 1 DOC1
## 2 DOC2
## 3 DOC3
```

Document-term matrix or term-document matrix

- A matrix that describes the frequency of terms that occur in a collection of documents.
- In a document-term matrix, rows correspond to documents in the collection and columns correspond to

```
tdm <- TermDocumentMatrix(corpus)
inspect(tdm)
```

```
## <<TermDocumentMatrix (terms: 12, documents: 3)>>
## Non-/sparse entries: 17/19
## Sparsity           : 53%
## Maximal term length: 8
## Weighting          : term frequency (tf)
## Sample             :
##           Docs
## Terms      1 2 3
##   and      1 0 0
##   are      1 1 1
##   best     1 1 0
##   fender!  1 0 0
##   gibson   1 0 0
```

- Document term matrix is the transposed Term document matrix

```
dtm <- DocumentTermMatrix(corpus, control = list(weightTfIdf))
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 12)>>
## Non-/sparse entries: 17/19
## Sparsity           : 53%
## Maximal term length: 8
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs and are best fender! gibson guitar guitars guitars. models the
##    1   1   1    1       1      1      1       0        0      1   1
##    2   0   1    1       0      0      0       1        1      0   1
##    3   0   1    0       0      0      0       1        0      0   0
```

- Sparsity is defined as a percentage of element with value 0 in total number of elements in document term matrix

```
mm <- as.matrix(dtm)
# Sparsity
sum(mm == 0)/(3*12)
```

```
## [1] 0.5277778
```

Problems

- Guitar; Guitars, Guitar.
- One way would be to remove all punctuations before creating corpus, using regex

```
text2 <- str_remove_all(text, pattern = "[:punct:]")
text2
```

```
## [1] "The best guitar models are Gibson and Fender"
## [2] "My guitars are the best guitars"
## [3] "Are these guitars yours"
```

- Or alternatively use tm_map().
- tm_map() takes corpus and applies on it a function, in this case removePunctuation

```
corpus <- VCorpus(VectorSource(text))
corpus <- tm_map(corpus, removePunctuation)
dtm <- DocumentTermMatrix(corpus)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 11)>>
## Non-/sparse entries: 16/17
## Sparsity           : 52%
## Maximal term length: 7
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs and are best fender gibson guitar guitars models the these
##    1   1   1    1      1      1      1       0      1   1     0
##    2   0   1    1      0      0      0       2      0   1     0
##    3   0   1    0      0      0      0       1      0   0     1
```

TF–IDF

- Term frequency–inverse document frequency - numerical statistic that is intended to reflect how important a word is to a document tin a collection or corpus
- The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.
- TF - Term Frequency, which measures how frequently a term occurs in a document
- $TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ adocument}{Total\ number\ of\ terms\ in\ the\ document}$
- Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of normalization.

TF–IDF

- Tf – the importance of the term in the separate document.
- Inverse Document Frequency measures how important a term is.
- While computing TF, all terms are considered equally important. However it is known that certain terms, such as “is”, “of”, and “that”, may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
- $IDF(t) = log_2(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ in\ it})$
- IDF decrease the weight of frequent words.
- Could you say which words have the high tf-idf weight?
- The smaller the weight, the more common the term.

```r
dtm1 <- DocumentTermMatrix(corpus, control = list(weighting = weightTfIdf))
as.matrix(dtm1)
```

```
##     Terms
## Docs       and are       best    fender    gibson    guitar   guitars
##    1 0.1981203   0 0.07312031 0.1981203 0.1981203 0.1981203 0.0000000
##    2 0.0000000   0 0.11699250 0.0000000 0.0000000 0.0000000 0.2339850
##    3 0.0000000   0 0.00000000 0.0000000 0.0000000 0.0000000 0.1462406
##     Terms
## Docs    models        the     these     yours
##    1 0.1981203 0.07312031 0.0000000 0.0000000
##    2 0.0000000 0.11699250 0.0000000 0.0000000
##    3 0.0000000 0.00000000 0.3962406 0.3962406
```

- for guitar

```r
(1/8)*log(3/1, base=2)
```

```
## [1] 0.1981203
```

300,000 songs and their lyrics

- Let's consider the example from the file lyrics.csv

```
lyrics <- read.csv("lyrics.csv", stringsAsFactors = F)
summary(lyrics$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      67    2006    2008    2009    2014    2038
```

Data Cleaning

```
lyrics %>%
  group_by(year) %>%
  summarise(count = n()) %>%
  arrange(desc(year))
```

```
## # A tibble: 52 x 2
##     year count
##    <int> <int>
##  1  2038    10
##  2  2016 35042
##  3  2015 27159
##  4  2014 26393
##  5  2013 16245
##  6  2012 14581
##  7  2011 11387
##  8  2010 11374
##  9  2009 11333
## 10  2008 20375
## # ... with 42 more rows
```

Data Cleaning

```r
lyrics %>%
  group_by(year) %>%
  summarise(count = n()) %>%
  arrange(year)
```

```
## # A tibble: 52 x 2
##     year count
##    <int> <int>
##  1    67     1
##  2   112     4
##  3   702     1
##  4  1968     1
##  5  1970   421
##  6  1971   480
##  7  1972   496
##  8  1973   504
##  9  1974   395
## 10  1975   321
## # ... with 42 more rows
```

Data Cleaning

```
lyrics %>%
  group_by(artist) %>%
  summarise(n_songs=n()) %>%
  arrange(desc(n_songs))
```

```
## # A tibble: 17,088 x 2
##    artist           n_songs
##    <chr>              <int>
##  1 dolly-parton         755
##  2 american-idol        700
##  3 elton-john           680
##  4 b-b-king             667
##  5 chris-brown          655
##  6 eddy-arnold          628
##  7 barbra-streisand     624
##  8 ella-fitzgerald      623
##  9 bob-dylan            614
## 10 bee-gees             599
## # ... with 17,078 more rows
```

Data Cleaning

```
lyrics$char_num <- nchar(lyrics$lyrics)
summary(lyrics$char_num)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0   744.0   844.1  1212.0 42155.0
```

```
table(lyrics$genre)
```

```
##
##       Country    Electronic          Folk       Hip-Hop         Indie
##         16449         15446          3095         30644          5373
##          Jazz         Metal Not Available         Other           Pop
##         15659         26900         27984         22394         46371
##           R&B          Rock
##          5560        123402
```

```
sum(lyrics$char_num==0)
```

```
## [1] 89449
```

Data Cleaning

```r
lyrics <- lyrics %>%
  filter(char_num>100 & genre != "Not Available" & year > 1968 & year <= 20
dim(lyrics)
```

```
## [1] 221669      7
```

```r
table(lyrics$genre)
```

```
##
##     Country Electronic       Folk    Hip-Hop      Indie       Jazz
##       13710       7027       2006      22168       2888       6915
##       Metal      Other        Pop        R&B       Rock
##       21250       4838      37770       3198      99899
```

```r
summary(lyrics$char_num)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      101     681     986    1184    1419   42155
```

Data Cleaning

```
b <- "[Intro: BeyoncÃƒÂ©]"
iconv(b, to="ASCII", sub="")
```

```
## [1] "[Intro: Beyonc]"
```

```
lyrics$lyrics <- iconv(lyrics$lyrics, to="ASCII", sub="")
```

Let's analyze lyrics of Beyonce

```
beyonce <- lyrics[lyrics$artist =='beyonce-knowles',]
beyonce_vs <- VectorSource(beyonce$lyrics)
beyonce_corpus <- VCorpus(beyonce_vs)
```

Stopwords

```r
stopwords("english")
```

```
##   [1] "i"          "me"         "my"         "myself"     "we"
##   [6] "our"        "ours"       "ourselves"  "you"        "your"
##  [11] "yours"      "yourself"   "yourselves" "he"         "him"
##  [16] "his"        "himself"    "she"        "her"        "hers"
##  [21] "herself"    "it"         "its"        "itself"     "they"
##  [26] "them"       "their"      "theirs"     "themselves" "what"
##  [31] "which"      "who"        "whom"       "this"       "that"
##  [36] "these"      "those"      "am"         "is"         "are"
##  [41] "was"        "were"       "be"         "been"       "being"
##  [46] "have"       "has"        "had"        "having"     "do"
##  [51] "does"       "did"        "doing"      "would"      "should"
##  [56] "could"      "ought"      "i'm"        "you're"     "he's"
##  [61] "she's"      "it's"       "we're"      "they're"    "i've"
##  [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
##  [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
##  [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
##  [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
##  [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
##  [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
```

Stemming

```r
love <- c("love", "loving", "lovingly", "loved", "lover", "lovely", "love",
          "game", "gaming", "games")
stemDocument(love)
```

```
##  [1] "love"  "love"  "love"  "love"  "lover" "love"  "love"  "game"
##  [9] "game"  "game"
```

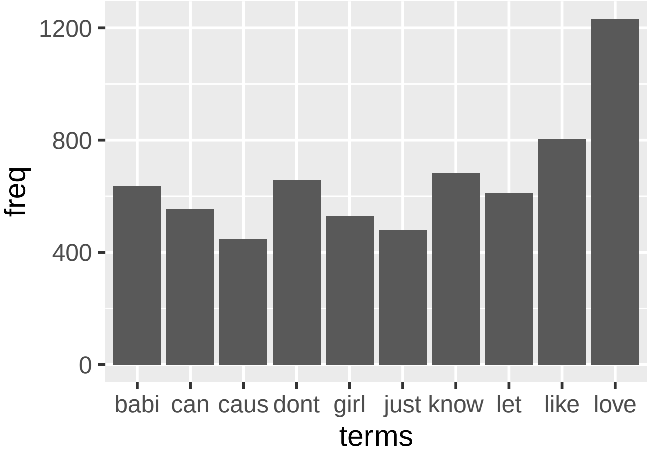
Analysis

```r
beyonce_tdm <- TermDocumentMatrix(beyonce_corpus,
               control=list(removeNumbers = T,removePunctuation = T,
                            stopwords = T, stemming = T))
dtm_mat <- as.matrix(beyonce_tdm)
freqs <- rowSums(dtm_mat)
df_freq <- data.frame(terms=rownames(dtm_mat),
                      freq = freqs, stringsAsFactors = F)
df_freq <- df_freq[order(df_freq$freq, decreasing = T),]
head(df_freq)
```

```
##      terms freq
## love  love 1233
## like  like  803
## know  know  684
## dont  dont  659
## babi  babi  638
## let    let  610
```

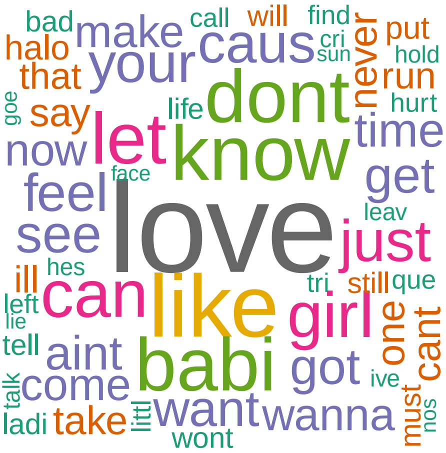
Analysis

```
df_top10 <- df_freq[1:10,]
ggplot(df_top10, aes(x=terms, y=freq)) + geom_bar(stat = 'identity')
```



Wordcloud

```r
set.seed(1)
wordcloud(words = df_freq$terms, freq = df_freq$freq, min.freq = 10,
max.words = 200, random.order = FALSE,  colors=brewer.pal(8, "Dark2"))
```



Tf-Idf

```r
beyonce_tdm <- TermDocumentMatrix(beyonce_corpus,
              control = list(removeNumbers = T, removePunctuation = T,
              stopwords = T, stemming = T, weighting = weightTfIdf))
tdm_mat <- as.matrix(beyonce_tdm)
freqs <- rowSums(tdm_mat)
df_freq <- data.frame(terms=rownames(tdm_mat),
                      freq = freqs)
df_freq <- df_freq[order(df_freq$freq, decreasing = T),]
head(df_freq)
```

```
##      terms     freq
## halo  halo 5.295742
## love  love 4.083825
## girl  girl 3.326538
## run    run 3.293408
## let    let 3.269829
## babi  babi 2.828966
```

```
set.seed(27)
wordcloud(words = df_freq$terms, freq = df_freq$freq, min.freq = 1,
    max.words = 90, random.order = FALSE, colors = brewer.pal(8, "Dark2"))
```

Task

- Extract the phone numbers in one column and the names in another:
- Combine them in the data frame

```r
phones <- c("Joe 027-789663",
  "Jimi 99656565",
  "Adri2 099-65-1995 GUITARIS")
```