gesis Leibniz Institute for the Social Sciences

Automatic Sampling and Analysis of YouTube Data

Processing & cleaning user comments

Johannes Breuer, Annika Deubel, & M. Rohangis Mohseni

February 14th, 2023

Leibniz Association

Pre-processing

- Pre-processing refers to all steps that are necessary for preparing data for analysis
- For web data, this is often more involved than, e.g., for survey data because:
  - the data were not designed with our analysis in mind
  - the data are typically less structured
  - the data are typically more heterogeneous
  - the data are typically larger
- *Note*: In addition, with large amounts of data, it is often necessary to work on servers or clusters instead of regular desktop or laptop computers. Even then, depending on the data set size and the complexity of the pre-processing steps, these can potentially take several hours or even days.

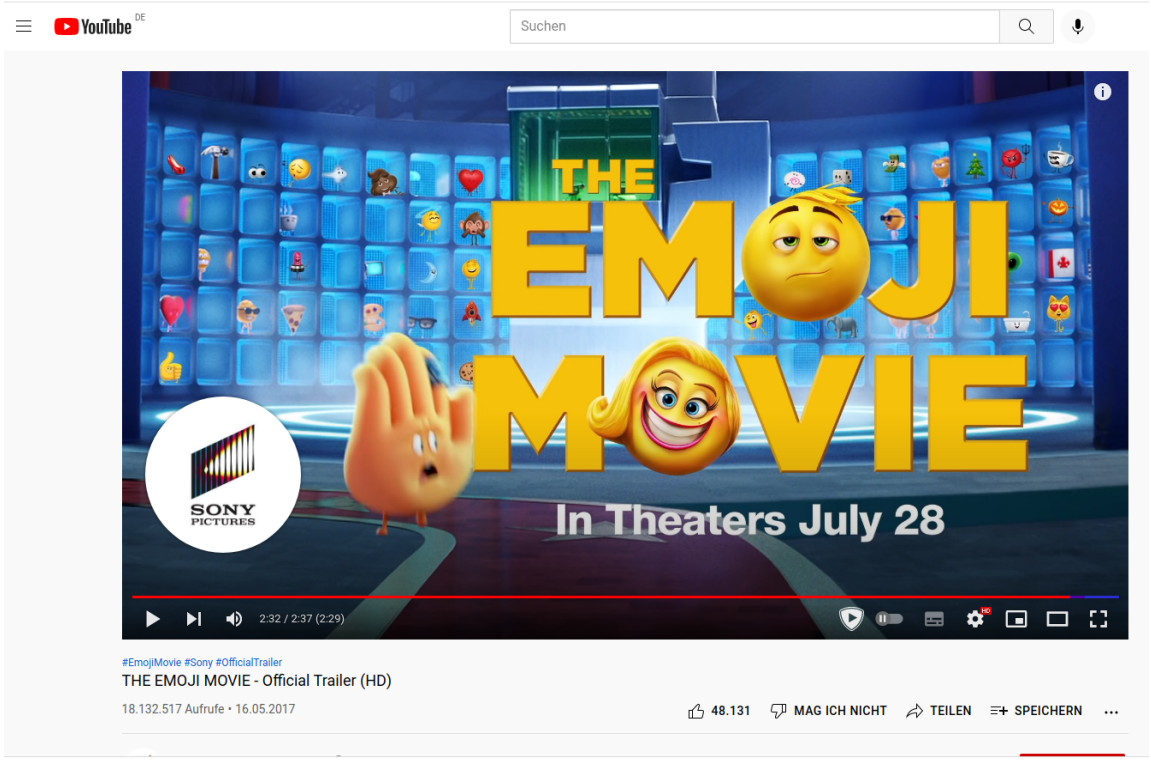
Pre-processing *YouTube* comments

- The `tuber` package returns an `R` dataframe instead of a `JSON` file (as the *YouTube* Data API v3 does)
- However, before we can analyze the comments, we need to:
  - select
  - format
  - extract
  - link

Note. For single videos, the data are small enough to be processed on a regular desktop/laptop computer, but this may not be the case if we have many videos with a high number of comments.

Pre-processing *YouTube* Comments

For this session, we will use comments from the Emoji Movie Trailer (https://www.youtube.com/watch?v=r8pJt4dK_s4)



Acknowledgment 

Almost all of the code we present in this session was written by Julian Kohne who taught the workshop with Johannes and Ro in 2020, 2021, and 2022.

Understanding Your Data (1)

The first step is always to explore your data. This is especially crucial for so-called *found data* because they were not designed for research purposes.

```
# load raw data
comments <- readRDS("./data/RawEmojiComments.rds")
# You may want/have to adjust the path (depending on your working directory)

# list all column names
colnames(comments)
```

```
##  [1] "videoId"               "textDisplay"           "textOriginal"          "authorDispla
##  [5] "authorProfileImageUrl" "authorChannelUrl"      "authorChannelId.value" "canRate"
##  [9] "viewerRating"          "likeCount"             "publishedAt"           "updatedAt"
## [13] "id"                    "parentId"              "moderationStatus"
```

Luckily, the *YouTube* Data API has good documentation that includes brief descriptions of all the pieces of information that you can extract from it.

Understanding Your Data (2)

This information is valuable for understanding what type of comments the dataframe contains...

```
table(is.na(comments$parentId))
```

```
##
## FALSE  TRUE
## 15620 22517
```

From the *YouTube* Data API documentation:

parentID: *The unique ID of the parent comment. This property is only set if the comment was submitted as a reply to another comment.*

Understanding Your Data (3)

... knowing how specific data types are formatted...

```
head(comments$publishedAt)
```

```
## [1] "2023-02-07T18:13:46Z" "2023-02-04T23:45:38Z" "2023-01-29T04:36:01Z" "2023-01-28T12:08
## [5] "2023-01-27T21:00:16Z" "2023-01-27T19:52:34Z"
```

```
class(comments$publishedAt)
```

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

From the *YouTube* Data API documentation:

publishedAt: *The date and time when the comment was originally published. The value is specified in ISO 8601 (YYYY-MM-DDThh:mm:ss.sZ) format.*

Understanding Your Data (4)

... or how similarly named variables differ from each other

```
comments$textOriginal[673]
```

```
## [1] "The best part 2:38"
```

```
comments$textDisplay[673]
```

```
## [1] "The best part <a href=\"https://www.youtube.com/watch?"
## [2] "v=r8pJt4dK_s4&amp;t=2m38s\">2:38</a>"
```

From the *YouTube* Data API documentation:

textOriginal: *The original, raw text of the comment as it was initially posted or last updated. The original text is only returned if it is accessible to the authenticated user, which is only guaranteed if the user is the comment's author.*

textDisplay: *The comment's text. The text can be retrieved in either plain text or HTML (...) Note that even the plain text may differ from the original comment text. For example, it may replace video links with video titles.*

Selecting What You Need

Now, we can select only those variables that we need for our analysis.

```
Selection <- subset(comments,select = -c(authorProfileImageUrl,
                                         authorChannelUrl,
                                         authorChannelId.value,
                                         canRate,
                                         viewerRating,
                                         moderationStatus))
colnames(Selection)
```

```
## [1] "videoId"           "textDisplay"       "textOriginal"      "authorDisplayName" "likeC
## [6] "publishedAt"       "updatedAt"         "id"                "parentId"
```

Word of advice: Always keep an unaltered copy of your raw data that you should not delete or overwrite. Save your processed data in a separate file (or in multiple steps and versions if your pre-processing pipeline is complex).

Formatting your Data

By default, the data you get with `tuber` is not in the right format for most analyses.

```
sapply(Selection, class)
```

```
##           videoId       textDisplay      textOriginal authorDisplayName         likeCount
##       "character"       "character"       "character"       "character"       "character"
##         updatedAt                id          parentId
##       "character"       "character"       "character"
```

```
# summary statistics for like counts
summary(Selection$likeCount)
```

```
##    Length     Class      Mode
##     38137 character character
```

```
# time difference between first comment and now
Sys.time() - Selection$publishedAt[1]
```

```
## Error in unclass(e1) - e2: nicht-numerisches Argument für binären Operator
```

Formatting the `likeCount`

We want the `likeCount` to be a numeric variable.

```
# transform likeCount to numeric
# (NB: this overwrites the original column)
Selection$likeCount <- as.numeric(Selection$likeCount)

# check
summary(Selection$likeCount)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    0.000    0.000    2.000    4.918    5.000 4287.000
```

Formatting Timestamps

The variables `publishedAt` & `updatedAt` should be datetime objects.

```
# transform timestamps to datetime objects
Selection$publishedAt[1]
```

```
## [1] "2023-02-07T18:13:46Z"
```

```
time_last_comment <- as.POSIXct(Selection$publishedAt[1],
                                format = "%Y-%m-%dT%H:%M:%OSZ",
                                tz = "UTC")
time_last_comment
```

```
## [1] "2023-02-07 18:13:46 UTC"
```

```
# test whether we can compute a difference
# with the datetime object
Sys.time() - time_last_comment
```

```
## Time difference of 6.959547 days
```

Word of Caution About Timestamps

Timestamps are extremely complex objects due to:

- Different calendars
- Different formatting
- Different time zones
- Historical anomalies
- Different resolutions
- Summer vs. winter time (differences between countries)
- Leap years
- etc.

Formatting Timestamps (3)

A convenient way of transforming datetime variables is the `anytime` package (another powerful option for dealing with times and dates in `R` is the `lubridate` package from the `tidyverse`). Its main function is also called `anytime()` and it automatically tries to guess the format from the character string.

```
# transform datetimes using anytime()
library(anytime)

Selection$publishedAt <- anytime(Selection$publishedAt,
                                 asUTC = TRUE)

Selection$updatedAt <- anytime(Selection$updatedAt,
                               asUTC = TRUE)
class(Selection$publishedAt)
```

```
## [1] "POSIXct" "POSIXt"
```

```
class(Selection$updatedAt)
```

```
## [1] "POSIXct" "POSIXt"
```

Formatting Timestamps (4)

Be aware of how to interpret your timestamps. Note that the date was interpreted as UTC but converted to our local CET timezone, which is 1 hour ahead of UTC. This comment was made at 18:13:46 in *our timezone*, but we have no idea about the time at the location of the user.

```
Selection$publishedAt[1]
```

```
## [1] "2023-02-07 19:13:46 CET"
```

Processing the Comments Text

After having formatted all our selected columns, we typically want to create new variables containing information that is not directly available in the raw data. For example, consider the following comments:

```
# Example comments with extractable information
strwrap(Selection$textOriginal[37245],79)
```

```
## [1] "Watch new Emoji movie [2017] Here: New Emoji movie 2017"
## [2] "https://www.clorox.com/"
```

```
Selection$textOriginal[691]
```

```
## [1] "Here him 😟😟😟😟😟😟"
```

Processing the Comments Text

Thes previous examples illustrate two things:

1) Comments can contain emojis and hyperlinks that might distort our text analysis.

2) These are features that we might want to have in separate variables for further analysis.

Extracting Hyperlinks (1)

We start with deleting hyperlinks from our text and saving them in a separate column. We will use the text mining package `qdapRegex` for this as it has predefined routines for handling large text vectors and regular expressions.

```
# Note that we are using the original text so we don't have
# to deal with the HTML formatting of the links
library(qdapRegex)
Links <- rm_url(Selection$textOriginal, extract = TRUE)
LinkDel <- rm_url(Selection$textOriginal)
head(Links[!is.na(Links)],3)
```

```
## [[1]]
## [1] "https://youtu.be/59Tr9NDr5N4\nI"
##
## [[2]]
## [1] "https://youtu.be/SgX3ggJv1Rw"
##
## [[3]]
## [1] "https://m.youtube.com/watch?v=BLUkgRAy_Vo"
```

Extracting Hyperlinks (2)

We get a list where each element corresponds to one row in the `Selection` dataframe and contains a vector of links that were contained in the `textOriginal` column. We also created the new vector `LinkDel`that contains the comment text without URLs.

```
strwrap(Selection$textOriginal[37245],79)
```

```
## [1] "Watch new Emoji movie [2017] Here: New Emoji movie 2017"
## [2] "https://www.clorox.com/"
```

```
LinkDel[37245]
```

```
## [1] "Watch new Emoji movie [2017] Here: New Emoji movie 2017"
```

```
Links[[37245]]
```

```
## [1] "https://www.clorox.com/"
```

Extracting Emojis (1)

The `qdapRegex` package has a lot of other different predefined functions for extracting or removing certain kinds of strings:

- `rm_citation()`
- `rm_date()`
- `rm_phone()`
- `rm_postal_code()`
- `rm_email()`
- `rm_dollar()`
- `rm_emoticon()`

Unfortunately, it does **not** contain a predefined method for emojis, so we will have to come up with our own method for extracting them.

Extracting Emojis (2)

The first step in our emoji processing is replacing the emojis with a textual description, so that we can treat it just like any other token in text mining.

Essentially, we want to replace this:

```
## 😄
```

with this

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

Extracting Emojis (3)

First of all, we need a dataframe that contains the emojis in the way in which they are internally represented by R (this means dealing with character encoding which can be a pain). Lucky for us, this information is contained in the emo package.

```
library(emo)
EmojiList <- jis
EmojiList[1:3,c(1,3,4)]
```

```
## # A tibble: 3 × 3
##   runes emoji name
##   <chr> <chr> <chr>
## 1 1F600 😀    grinning face
## 2 1F601 😁    beaming face with smiling eyes
## 3 1F602 😂    face with tears of joy
```

Extracting Emojis (4)

Next, we need to paste the names of the emojis together, capitalizing the first letter of every word for better readability.

```
# Define a function for capitalizing and pasting names together
simpleCap <- function(x) {

  # Split the string
  splitted <- strsplit(x, " ")[[1]]

  # Paste it back together with capital letters
  paste(toupper(substring(splitted, 1,1)),
        substring(splitted, 2),
        sep = "",
        collapse = " ")
}
```

Extracting Emojis (5)

```
# Apply the function to all emoji names
CamelCaseEmojis <- lapply(jis$name, simpleCap)
CollapsedEmojis <- lapply(CamelCaseEmojis,
                          function(x){gsub(" ",
                                           "",
                                           x,
                                           fixed = TRUE)})
EmojiList[,4] <- unlist(CollapsedEmojis)
EmojiList[1:3,c(1,3,4)]
```

```
## # A tibble: 3 × 3
##   runes emoji name
##   <chr> <chr> <chr>
## 1 1F600 😀    GrinningFace
## 2 1F601 😁    BeamingFaceWithSmilingEyes
## 3 1F602 😂    FaceWithTearsOfJoy
```

Extracting Emojis (6)

After that, we need to order our dictionary from the longest to shortest string to prevent partial matching of shorter strings later.

```
EmojiList <- EmojiList[rev(order(nchar(jis$emoji))),]
head(EmojiList[,c(1,3,4)],5)
```

```
## # A tibble: 5 × 3
##   runes                                          emoji name
##   <chr>                                          <chr> <chr>
## 1 1F469 200D 2764 FE0F 200D 1F48B 200D 1F469 👩‍❤️‍💋‍👩 Kiss:Woman,Woman
## 2 1F468 200D 2764 FE0F 200D 1F48B 200D 1F468 👨‍❤️‍💋‍👨 Kiss:Man,Man
## 3 1F469 200D 2764 FE0F 200D 1F48B 200D 1F468 👩‍❤️‍💋‍👨 Kiss:Woman,Man
## 4 1F3F4 E0067 E0062 E0077 E006C E0073 E007F 🏴 Wales
## 5 1F3F4 E0067 E0062 E0073 E0063 E0074 E007F 🏴 Scotland
```

Note that what we are ordering by the `emoji` column, not the `name` or `runes` columns.

Extracting Emojis (7)

Now we can `loop` through all emojis in the comments and replace them consecutively in each comment (*note*: this may take a while).

```
# Assign the comments vector without URLs to a new object
TextEmoRep <- LinkDel

# Loop over all emojis for all comments in the new object/vector
for (i in 1:dim(EmojiList)[1]) {

  TextEmoRep <- rm_default(TextEmoRep,
                  pattern = EmojiList[i,3],
                  replacement = paste0("EMOJI_",
                                       EmojiList[i,4],
                                       " "),
                  fixed = TRUE,
                  clean = FALSE,
                  trim = FALSE)
}
```

Extracting Emojis (8)

The resulting output is a large character vector in which the emojis are replaced by textual descriptions.

```
Selection$textOriginal[910]
```

```
## [1] "Current like to dislike ratio:\n👍47K  👎167K"
```

```
TextEmoRep[910]
```

```
## [1] "Current like to dislike ratio"             " EMOJI_ThumbsUp 47K EMOJI_ThumbsDown 167K
```

Extracting Emoji Descriptions

```
ExtractEmoji <- function(x){

  SpacerInsert <- gsub(" ","[{[SpAC0R]}]", x)
  ExtractEmoji <- rm_between(SpacerInsert,
                             "EMOJI_","[{[SpAC0R]}]",
                             fixed = TRUE,
                             extract = TRUE,
                             clean = FALSE,
                             trim = FALSE,
                             include.markers = TRUE)

  UnlistEmoji <- unlist(ExtractEmoji)
  DeleteSpacer <- sapply(UnlistEmoji,
                         function(x){gsub("[{[SpAC0R]}]",
                                          " ",
                                          x,
                                          fixed = TRUE)})

  names(DeleteSpacer) <- NULL
  Emoji <- paste0(DeleteSpacer, collapse = "")
  return(Emoji)
}
```

Extracting Emojis Function

We can apply the function to create a new vector containing only the emojis as textual descriptions.

```
Emoji <- sapply(TextEmoRep,ExtractEmoji)
names(Emoji) <- NULL
LinkDel[910]
```

```
## [1] "Current like to dislike ratio: 👍47K 👎167K"
```

```
Emoji[910]
```

```
## [1] "EMOJI_ThumbsUp EMOJI_ThumbsDown "
```

Removing Emojis

In addition, we remove the emojis from the `LinkDel` variable to get one "clean" text column (i.e., without hyperlinks & emojis).

```
# Remove emojis from the LinkDel vector
LinkDel[910]
```

```
## [1] "Current like to dislike ratio: 👍47K 👎167K"
```

```
TextEmoDel <- ji_replace_all(LinkDel,"") # function from the emo pkg
TextEmoDel[910]
```

```
## [1] "Current like to dislike ratio: 47K 167K"
```

Summary: Extracting Information

We now have different versions of our text column

1) The original one, with hyperlinks and emojis (`Selection$textOriginal`)

2) One with only plain text and without hyperlinks and emojis (`TextEmoDel`)

3) One with only hyperlinks (`Links`)

4) One with only emojis (`Emoji`)

Next, we want to integrate all of them into our dataframe.

Linking Everything Back Together

We can now combine our dataframe with the additional variables we have created. **NB**: because we sometimes have more than two links or two emojis per comment, we need to use the `I()` function so we can put them in the dataframe "as is" (for later analyses, we will have to unlist these columns rowwise if we want to use them).

```
df <- cbind.data.frame(Selection$videoId,
                       Selection$authorDisplayName,
                       Selection$textOriginal,
                       TextEmoRep,
                       TextEmoDel,
                       Emoji = I(Emoji),
                       Selection$likeCount,
                       Links = I(Links),
                       Selection$publishedAt,
                       Selection$updatedAt,
                       Selection$parentId,
                       Selection$id,
                       stringsAsFactors = FALSE)
```

Linking Everything Back Together

As a final step, we can give the columns appropriate names and save the dataframe for later use

```
# set column names
names(df) <- c("VideoID",
               "Author",
               "Text",
               "TextEmojiReplaced",
               "TextEmojiDeleted",
               "Emoji",
               "LikeCount",
               "URL",
               "Published",
               "Updated",
               "ParentId",
               "CommentID")
```

```
saveRDS(df, file = "./data/ParsedEmojiComments.rds")
# You may want/have to adjust the path (depending on your working directory)
```

One Function to Parse Them All!

We (i.e., mostly Julian Kohne) have created a function that combines all of the pre-processing steps presented in this session. It is stored in the file `yt_parse.R` (which you can find in the folder `content\R` in the workshop materials).

NB: This function will only work if the helper functions are in the same directory (i.e., the scripts named `CamelCase.R`, `ExtractEmoji.R`, & `ReplaceEmoji.R`).

Note: The function `yt_parse()` also contains an option for processing comments collected with the `vosonSML` package. To make use of this, the function argument `package` must be `="vosonSML`.

```
yt_parse(package = "vosonSML")
```

Exercise time 

Solutions