Automatic Sampling and Analysis of YouTube Data

Processing and Cleaning User Comments

Julian Kohne
Johannes Breuer
M. Rohangis Mohseni

2020-02-10

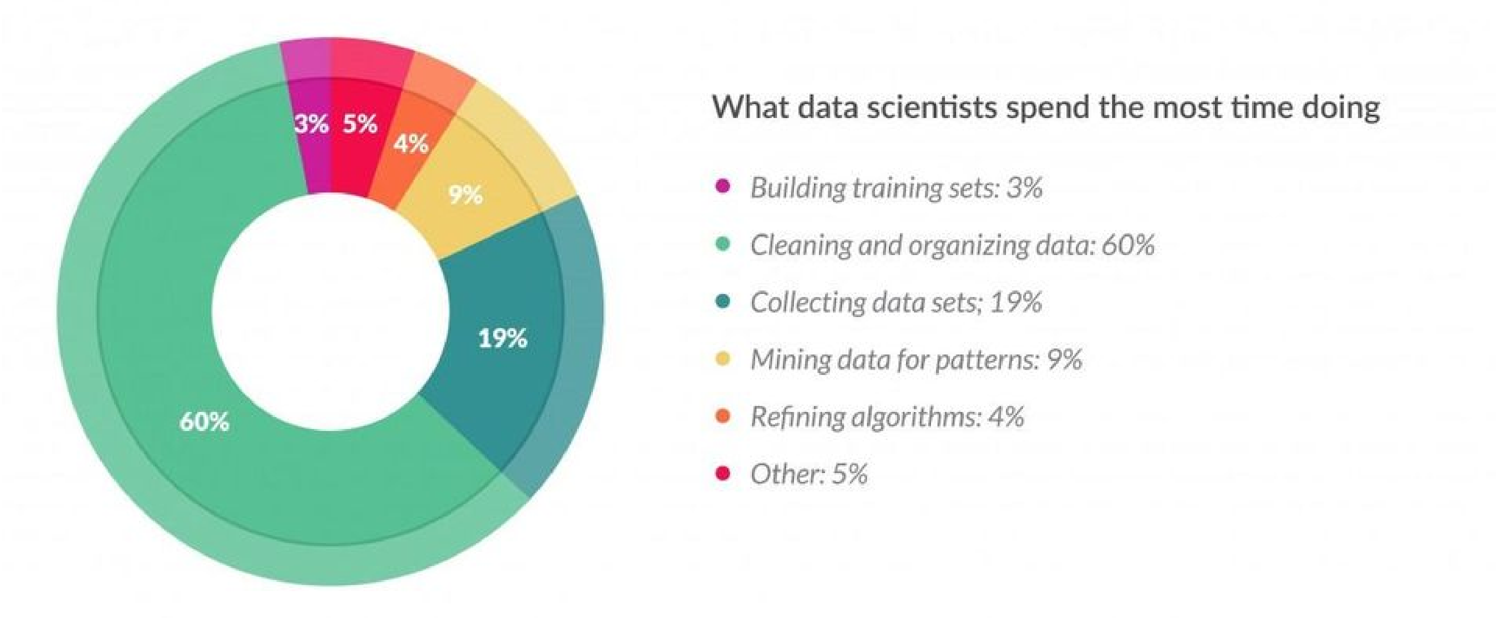
Processing and Cleaning User Comments

Preprocessing

- Preprocessing refers to all steps that need to be taken to make the data suitable for the actual analysis
- For webscraping data, this is often more tedious and time-consuming than for survey data because:
    - the data is not designed with your analysis in mind
    - the data is typically less structured
    - the data is typically more complex
    - the data is typically more heterogenous
    - the data is typically larger
- In addition, it's often necessary to work on Servers instead of regular PCs
- Even then, restructuring or transforming data can take days, so mistakes hurt more

Preprocessing

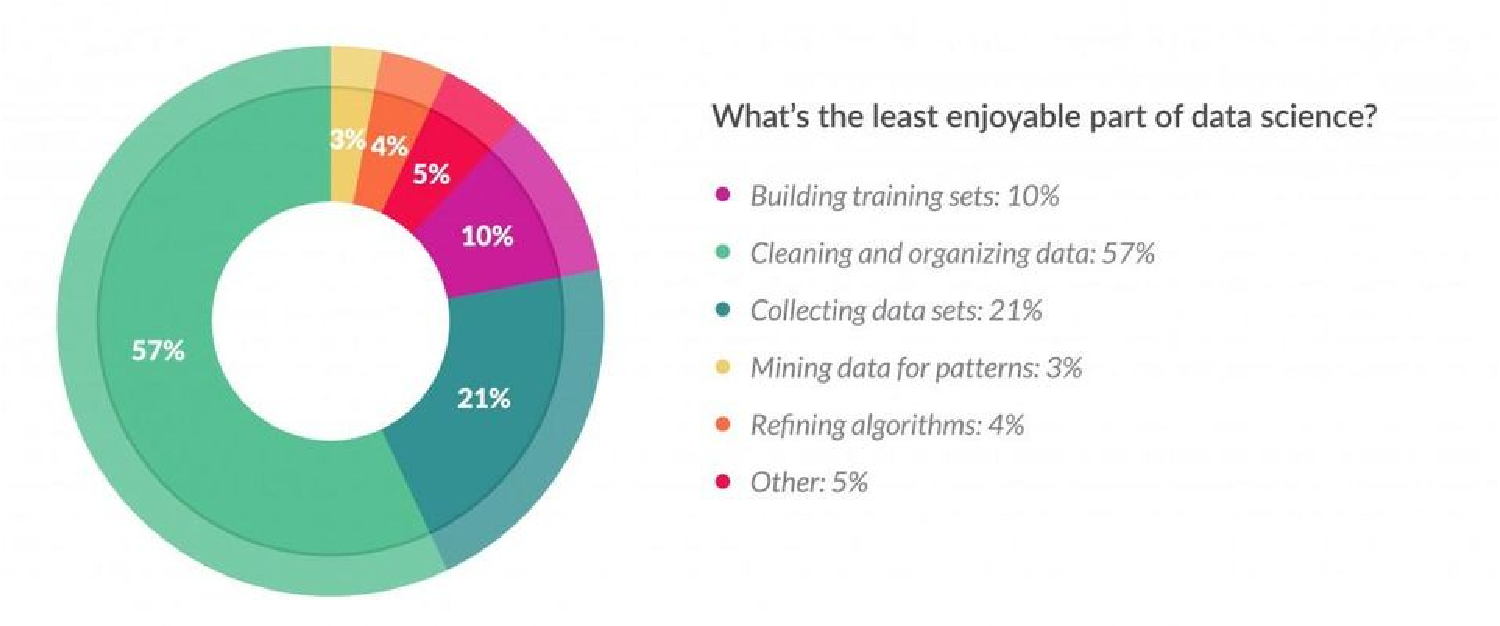
- In *Data Science*, most time is typically spent on the preprocessing rather than the actual analysis



Source: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#157890a96f63

Preprocessing

- Also, it is perceived as the least enjoyable part of the process



Source: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#157890a96f63

Preprocessing *YouTube* comments

- The `tuber` package already returns an R dataframe instead of a JSON
- We can already select which data we need by using the API through `tuber`
- For single videos, the data is small enough to be processed on a regular PC
- However, this doesn't mean that the data is already usable for all intents and purposes
- We still need to:
  - select
  - format
  - extract
  - link

the information that is relevant to us

Preprocessing *YouTube* Comments

Loading the unprocessed comments into R

```
# loading raw data (This is the BackUp file)
comments <- readRDS("../../data/RawComments.rds")
```

Understanding Your Data (1)

The first step is always to understand your data, this is especially crucial for *found data* because it was not designed with your analysis in mind

```
# listing all columns
colnames(comments)
```

```
##  [1] "authorDisplayName"     "authorProfileImageUrl" "authorChannelUrl"
##  [4] "authorChannelId.value" "videoId"               "textDisplay"
##  [7] "textOriginal"          "canRate"               "viewerRating"
## [10] "likeCount"             "publishedAt"           "updatedAt"
## [13] "id"                    "parentId"              "moderationStatus"
```

Luckily, the *YouTube* API is very well documented and provides brief explanations for all the variables you can extract from it

Understanding Your Data (2)

This information is valuable for understanding missing data

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

```
##
## FALSE  TRUE
##  1899  3878
```

A quick look into the documentation reveals:

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)

…or for knowing how specific datatypes are formatted

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

```
## [1] "2020-02-04T03:24:32.000Z" "2020-02-02T22:54:09.000Z"
## [3] "2020-02-02T21:22:39.000Z" "2020-02-02T07:21:58.000Z"
## [5] "2020-01-31T08:06:44.000Z" "2020-01-30T14:19:34.000Z"
```

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

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

A quick look into the documentation reveals:

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

Understanding Your Data (4)

…or how similar variables are different from each other

```
## [1] "... 0:39 Is that the Crying Indian in the back?"
```

```
## [1] "... <a href=\"https://www.youtube.com/watch?v=1aheRpmurAo&amp;t=0m39s
## [2] "Is that the Crying Indian in the back?"
```

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. (The comments.list and commentThreads.list methods both support a textFormat parameter, which specifies the desired text format). 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 (Don't) need

Now we can decide on what we need for further analysis

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

```
## [1] "authorDisplayName" "textDisplay"       "textOriginal"
## [4] "likeCount"         "publishedAt"       "updatedAt"
## [7] "id"                "parentId"
```

Word of advice: Always keep an unalterd copy of your raw data and don't overwrite it. You never know what kinds of mistakes/oversights you might notice down the line and you don't want to have to recollect everything. Save your parsed data in a seperate file (or in multiple steps if your preprocessing pipeline is complex).

Formatting your Data

By default, the data you get out of `tuber` is most likely not in the right format

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

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

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

```
##    Length     Class      Mode
##      5777 character character
```

```
## Error in unclass(e1) - e2: non-numeric argument to binary operator
```

Formatting `likeCount`

We want the `likeCount` to be a numeric variable and the timestamps to be datetime objects

```
# Transforming likeCount to numeric
# (carefull, this is overwriting the column)
Selection$likeCount <- as.numeric(Selection$likeCount)

# testing
summary(Selection$likeCount)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00    0.00   12.29    1.00 3903.00
```

We can now work with the number of likes as a numeric variable

Formatting your Timestamps (1)

Timestamps are extremely complex objects due to:

- Different calendars
- Different formattings
- Different origins
- Different time zones
- Historical anomalies
- Different resolutions
- Summer vs. Wintertime (different for each country and depending on hemisphere!)
- Leap years
- etc.

For these reasons, **never** try to code your own timestamp translations from scratch. Fortunately, R has several build in methods to deal with this madness. The most basic one as the `as.POSIXct()` function, the most convenient one is the `anytime()` function from the `anytime` package.

Formatting Timestamps (2)

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

```
## [1] "2020-02-04T03:24:32.000Z"
```

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

```
## [1] "2020-02-04 03:24:32 UTC"
```

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

```
## Time difference of 3.543911 days
```

This internal representation of time objects will be extremely important for plotting trends over time and calculating time differences

Formatting Timestamps (3)

A more convenient way to transform datetimes is the `anytime` package. Basically, it automatically tries to guess the format from the cahracter string, so you don't have to. This is especially handy for vectors of datetimes in multiple different formats.

```
# transforming datetimes using anytime()
library(anytime)
Selection$publishedAt <- anytime(Selection$publishedAt,
                                 asUTC = TRUE)
Selection$updatedAt <- anytime(Selection$updatedAt,
                               asUTC = TRUE)
sapply(list(Selection$publishedAt,Selection$updatedAt),class)
```

```
##      [,1]      [,2]
## [1,] "POSIXct" "POSIXct"
## [2,] "POSIXt"  "POSIXt"
```

Word of Advice: For datetime conversions, always do some sanity spotchecks, especially when you are using methods that automatically detect the format. Give special attention to the *timezone* in which your data is saved and compare it to the documentation of the standard.

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 04:24:32 in *our time*, we have no idea about the time at the location of the user. She might of made this comment at night or in the morning, depending on where she's from.

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

```
## [1] "2020-02-04 04:24:32 CET"
```

Extracting Information

After having formatted all our selected columns, we usually also want to create some TextEmoRep ones with information that is not directly available in the raw data. Consider for example our these comments:

```
# Example comments with extractable information
Selection$textOriginal[39]
```

```
## [1] "More bullshit 😷😠"
```

```
Selection$textOriginal[495]
```

```
## [1] "Sure, here you go: https://allthatsinteresting.com/iron-eyes-cody"
```

There are two issues exemplefied by these comments:

1) Comments contain emoji and hyperlinks that might distort our text analysis later

2) These are features that we'd like to have in a seperate column for our analysis

Extracting Hyperlinks (1)

We will start with deleting hyperlinks from our text and saving them in an additional column. We will use the textmining package `qdapRegex` for this, that has predefined routines for handling large textvectors and regular expressions. You can learn more about regular expressions here.

```r
# 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,2)
```

```
## [[1]]
## [1] NA
##
## [[2]]
## [1] "https://youtu.be/GWCySrYxov0"
```

Extracting Hyperlinks (2)

We get back 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. At the same time, the link was removed from the Selection dataframe.

```
LinkDel[495]
```

```
## [1] "Sure, here you go:"
```

```
Links[495]
```

```
## [[1]]
## [1] "https://allthatsinteresting.com/iron-eyes-cody"
```

Extracting Emoji (1)

The `qdapRegex` package has a lot of other different predefined functions to extract or remove 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 emoji, so we will have to use the `emo` package for removing the emoji and come up with our own method for extracting them.

Extracting Emoji (2)

First we want to replace the emoji with a textual description, so that we can treat it just like any other token in text mining. This is no trivial task, as we have to go through each comment and replace each emoji with it's respective textual description. Unfortuntely, we did not find a working, easy to use, out of the box solution for this. But we can always make our own!

Essentially, we want to replace this:

```
## 🐒
```

with this

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

Extracting Emoji (3)

First of all, we need a dataframe that contains the emoji as they are internally represented by R (this can be quite the hassle. Luckily, this is contained in the emo package

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

```
## # A tibble: 3 x 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 Emoji (4)

Next, we need to paste the names of the Emoji together while capitalizing every words first letter for better readibility

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

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

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

Extracting Emoji (5)

```r
# Applying the function to all the 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 x 3
##   runes emoji name
##   <chr> <chr> <chr>
## 1 1F600 😀    GrinningFace
## 2 1F601 😁    BeamingFaceWithSmilingEyes
## 3 1F602 😂    FaceWithTearsOfJoy
```

Extracting Emoji (6)

Next, we need to order our dictionary from longest to shortest, so that we can prevent partial matching of shorter strings later

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

```
## # A tibble: 5 x 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 is the `emoji` column, not the `text` or `runes` columns

Extracting Emoji (7)

Next, we need to `loop` through all of our emoji and replace them one after the other in each comment (this may take a while)

```
# Assigning the column to a TextEmoRep variable
TextEmoRep <- LinkDel

# Looping through all Emojis for all comments in LinkDel
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 Emoji (8)

As output, we get a large character vector with replaced emoji

```
TextEmoRep[39]
```

```
## [1] "More bullshit EMOJI_FaceWithMedicalMask EMOJI_NauseatedFace "
```

Extracting Emoji (9)

```
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 Emoji (10)

We can apply the function to get one vector containing only the emoji as textual descriptions

```
Emoji <- sapply(TextEmoRep,ExtractEmoji)
names(Emoji) <- NULL
LinkDel[39]
```

```
## [1] "More bullshit 😷🤢"
```

```
Emoji[39]
```

```
## [1] "EMOJI_FaceWithMedicalMask EMOJI_NauseatedFace "
```

Removing Emoji

In addition, we remove the emoji from our `LinkDel` variable to have one *clean* column that we can use for textmining later. This column will not contain hyperlinks or emoji.

```
# We take the LinkDel column and also delete the emoji from it
library(emo)
LinkDel[39]
```

```
## [1] "More bullshit 😷😧"
```

```
TextEmoDel <- ji_replace_all(LinkDel,"")
TextEmoDel[39]
```

```
## [1] "More bullshit "
```

Extracting Information

We now have different versions of our text column

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

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

3) One with only hyperlinks (`Links`)

4) One with only emoji (`Emoji`)

We want to integrate them in our dataframe

Linking everything back together

We can now recombine our dataframe with the additional columns we created to have the perfect starting point for our analysis! However, because we sometimes have more than two links or two emoji per comment, we need to use the `I()` function so we can put them in the dataframe `as is`. Later, we will have to unlist these columns rowwise if we want to use them.

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

Linking everything back together

At last, we can give the columns appropriate names and save the dataframe for later use

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

saveRDS(df, file = "../../data/ParsedComments.rds")
```

Exercise time ♀

Solutions