Automatic Sampling and Analysis of YouTube Data

Sentiment Analysis of User Comments

Julian Kohne
Johannes Breuer
M. Rohangis Mohseni

2021-02-25

Sentiment Analysis of User Comments

Setup

In previous versions of `R`, all strings were automatically translated to factors when reading data. This has been changed recently, but we still include the code to prevent this, in case you're working with an older version of `R`.

```
# Only necessary for R versions < 4.0.0
options(stringsAsFactors = FALSE)
```

Basic Sentiment Analysis

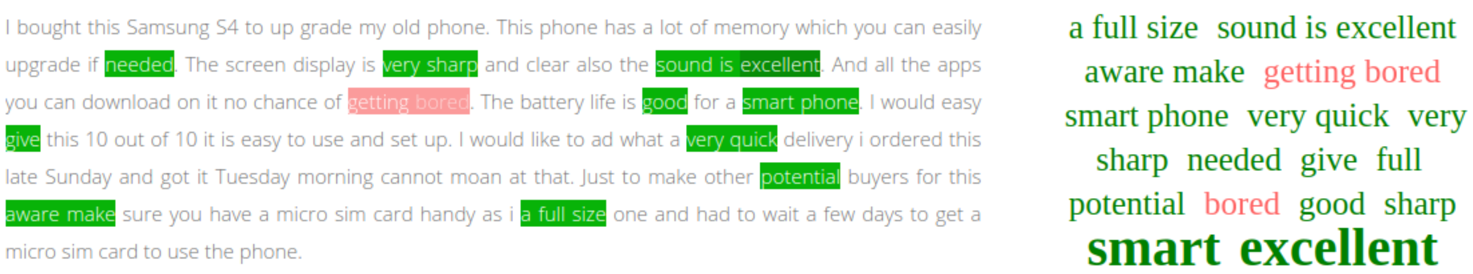
Sentiment Analysis

- The basic task of sentiment analysis is to detect the *polarity* of a sentence or a collection of sentences in terms of positivity and negativity
- Sentiment is often used in market research for product reviews
- For *YouTube* videos, look at the sentiment in the comments to quantify the valence of user reactions
- There are other methods for detecting:
    - emotional states
    - political stances
    - objectivity/subjectivity

Basic Idea of Sentiment Analysis

We compare each word in a sentence with a predefined dictionary

- Each word gets a score: For example a score between -1 (negative) and +1 (positive), with 0 being neutral
- We add up all the scores for a sentence to get an overall score for the sentence



Basic Sentiment Analysis

```
lexicon::hash_sentiment_jockers[sample(1:10738,10),]
```

```
##                x     y
##  1: coordinate  0.25
##  2:      grant  0.50
##  3:     unsure -0.50
##  4: painlessly  1.00
##  5:    savaged -1.00
##  6:   ordained  0.60
##  7:    scratch -0.60
##  8:   slowness -0.25
##  9:  innocuous  0.50
## 10: malodorous -1.00
```

Basic Sentiment Analysis

- This simple approach is usually only a crude approximation
- It is limited for a multitude of reasons:
  - Negations ("This video is not bad, why would someone hate it?")
  - Adverbial modification ("I love it" vs. "I absolutely love it")
  - Context ("The horror movie was scary with many unsettling plot twists")
  - Domain specificity ("You should see their decadent dessert menu.")
  - Slang ("Yeah, this is the shit!")
  - Sarcasm ("Superb reasoning, you must be really smart")
  - Abbreviations ("You sound like a real mofo")
  - Emoticons and emoji ("How nice of you... 😠")
  - ASCII Art ("(͡° ͜ʖ ͡°)")
- These limitations can lead to inaccurate classifications, for example:

Basic Sentiment Analysis

Classified as very negative

Fucking hilarious! And that guy could either do commercials or be an actor, I've never, in my entire life, heard anyone express themselves that strongly about a fucking hamburger. And now all I know is I have never eaten one of those but damned if I won't have it on my list of shit to do tomorrow! Hell of a job by schmoyoho as well, whoever said this should be a commercial hit it on the head.

Classified as very positive

Schmoyoho, we're not really entertained by you anymore. You're sort of like Dane Cook. At first we thought, "Wow! Get a load of this channel! It's funny!" But then we realized after far too long, "Wow, these guys are just a one trick pony! There is absolutely nothing I like about these people!" You've run your course. The shenanigans, the "songifies".. we get it. It's just not that funny man. We don't really like you. So please, for your own sake, go and actually try to make some real friends.

Sentiment Analysis of *YouTube* Comments

There are more sophisticated methods for sentiment analysis that yield better results. However, their complexity is beyond the scope of this workshop. We will do three things in this session and compare the respective results

1) Apply a basic sentiment analysis to our scraped *YouTube* comments

2) Use a slightly more elaborate out-of-the-box method for sentiment analysis

3) Extend the basic sentiment analysis to emoji

Word of Advice: Before using the more elaborate methods in your own research, make sure that you understand the underlying model, so you can make sense of your results. You should never blindly trust someone else's implementation without understanding it. Also: Always do sanity checks to see if you get any unexpected results.

1) Basic Comment Sentiments

First of all, we load our preprocessed comments and try out the built-in basic sentiment tagger from the `syuzhet` package

```
# loading data
comments <- readRDS("../../data/ParsedEmojiComments.rds")

# loading package
library(syuzhet)

# testing simple tagger
get_sentiment("Superb reasoning, you must be really smart")
```

```
## [1] 1.5
```

1) Basic Comment Sentiments

We can appy the basic sentiment tagger to the whole vector of comments. Remember that we need to use the text column without hyperlinks and emojis.

```
# creating basic Sentiment scores
BasicSentiment <- get_sentiment(comments$TextEmojiDeleted)

# summarizing basic sentiment scores
summary(BasicSentiment)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -34.4500  -0.5000   0.0000  -0.0741   0.2500  19.9500
```

Checking the documentation for the `get_sentiment()` function reveals that it can take different *methods* as arguments. These methods correspond to different dictionaries and might yield different results. The function also allows using a custom dictionary by providing a dataframe to the *lexicon* argument.

1) Basic Comment Sentiments

Lets compare the results of the different dictionaries

```
# computing sentiment scores with different dictionaries
BasicSentimentSyu <- get_sentiment(comments$TextEmojiDeleted,
                                   method = "syuzhet")
BasicSentimentBing <- get_sentiment(comments$TextEmojiDeleted,
                                    method = "bing")
BasicSentimentAfinn <- get_sentiment(comments$TextEmojiDeleted,
                                     method = "afinn")
BasicSentimentNRC <- get_sentiment(comments$TextEmojiDeleted,
                                   method = "nrc")
```

```
## Warning: `filter_()` is deprecated as of dplyr 0.7.0.
## Please use `filter()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

1) Basic Comment Sentiments

```
# combining them into a dataframe
Sentiments <- cbind.data.frame(BasicSentimentSyu,
                               BasicSentimentBing,
                               BasicSentimentAfinn,
                               BasicSentimentNRC,
                               1:dim(comments)[1])
# setting column names
colnames(Sentiments) <- c("Syuzhet",
                          "Bing",
                          "Afinn",
                          "NRC",
                          "Comment")
```

1) Basic Comment Sentiments

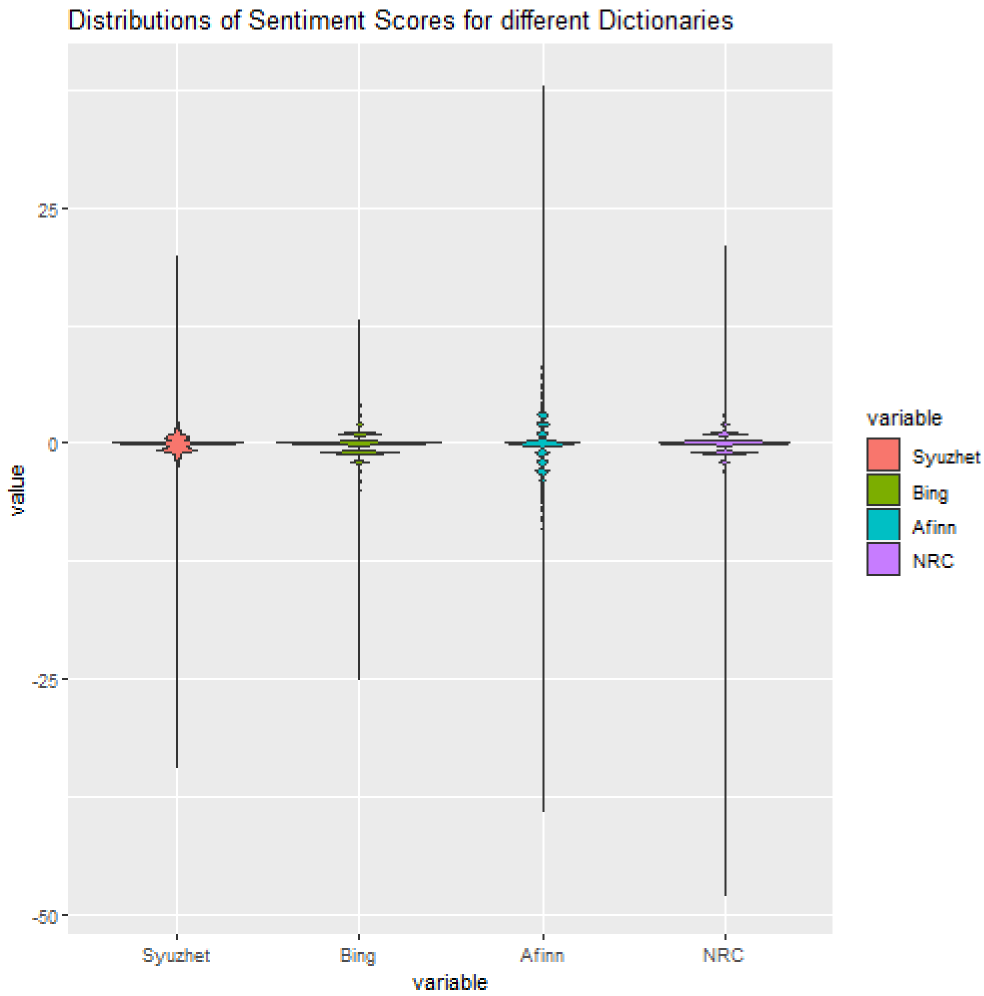
```r
# Correlation Matrix
cor(Sentiments[,c(-5)])
```

```
##           Syuzhet      Bing     Afinn       NRC
## Syuzhet 1.0000000 0.7932942 0.7529296 0.6997556
## Bing    0.7932942 1.0000000 0.7142517 0.5827967
## Afinn   0.7529296 0.7142517 1.0000000 0.5075133
## NRC     0.6997556 0.5827967 0.5075133 1.0000000
```

1) Basic Comment Sentiments

```
# load library
library(ggplot2)

# transform data into long format
SentimentsLong <- reshape2::melt(Sentiments,
                                 id.vars = c("Comment"))

# violin plot
ggplot(SentimentsLong, aes(x=variable,
                           y=value,
                           fill=variable)) +
    geom_violin() +
    ggtitle("Distributions of Sentiment Scores
            for different Dictionaries")
```

1) Basic Comment Sentiments



1) Basic Comment Sentiments

The choice of the dictionary can have an impact on your sentiment analysis. For this reason, it's crucial to select the dictionary with care and to be aware of how, by whom and for which purpose it was constructed. You can find more information on the specifics of the differnt dictionaries here.

In this session, we will continue with the `syuzhet` dictionary.

```
# Adding the Syuzhet Comments to our dataframe
comments$Sentiment <- Sentiments$Syuzhet
```

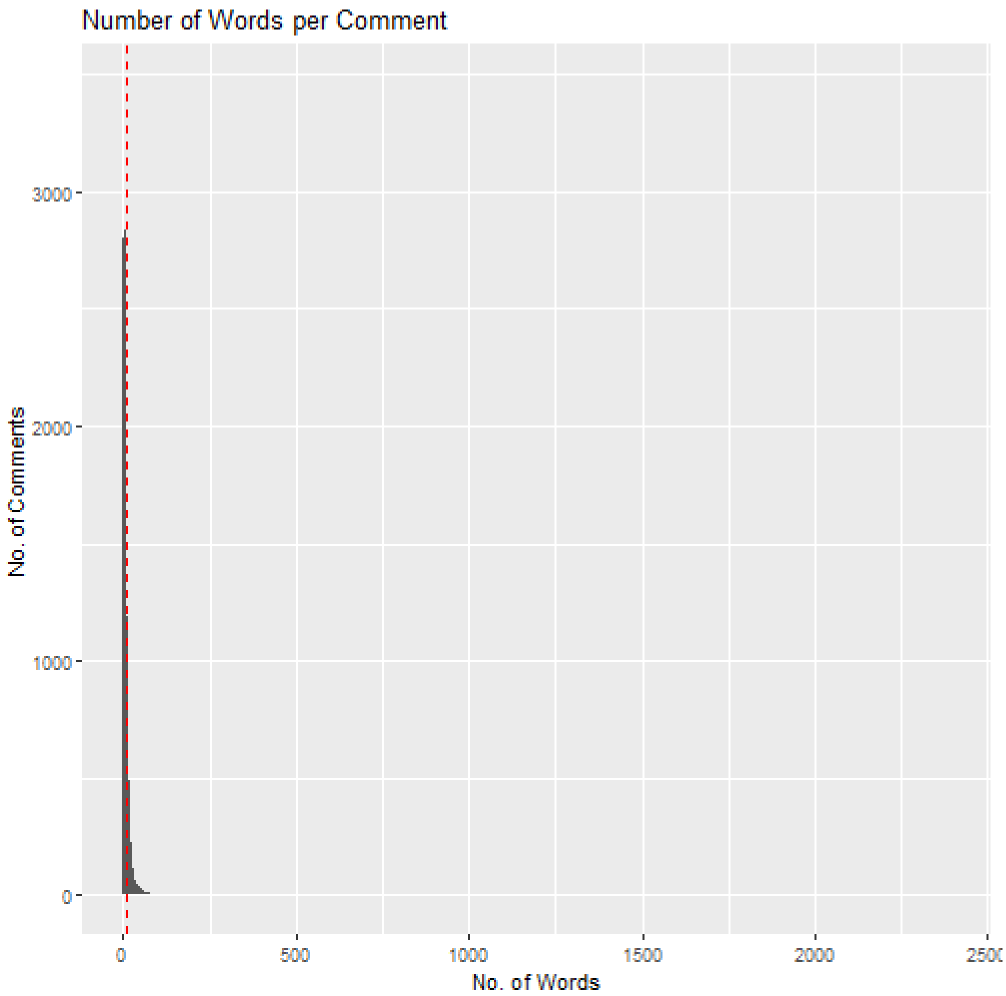
1) Basic Comment Sentiments

Another pitfall to be aware of is the length of the comments. Let's have a look at the distribution of words per comment

```
# Computing number of words per comment
comments$Words <- sapply(comments$TextEmojiDeleted,
                         function(x){length(unlist(strsplit(x,
                                                            " ")))})
```

```
# Histogram
ggplot(comments, aes(x=Words)) +
  geom_histogram(binwidth = 1) +
  geom_vline(aes(xintercept=mean(Words)),
             color="red",
             linetype="dashed",
             size = 0.5) +
  ggtitle(label = "Number of Words per Comment") +
  xlab("No. of Words") +
  ylab("No. of Comments")
```

1) Basic Comment Sentiments



1) Basic Comment Sentiments

Because longer comments also contain more words, they have a higher likelihood of getting more extreme sentiment scores. Lets' look at the most negative and the most positive comments.

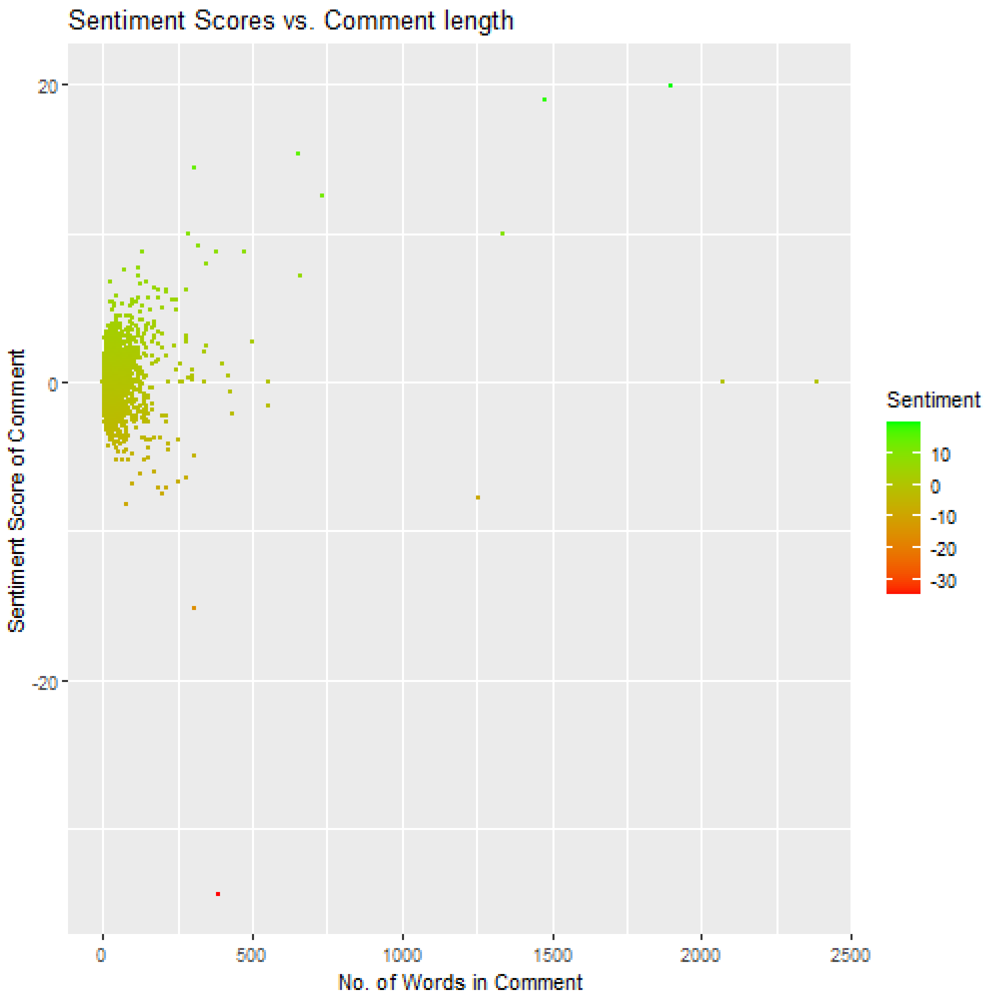
```r
# Most positive comment
strwrap(comments$TextEmojiDeleted[Sentiments$Syuzhet ==
                                  max(Sentiments$Syuzhet)], 79)

# Most negative comment
strwrap(comments$TextEmojiDeleted[Sentiments$Syuzhet ==
                                    min(Sentiments$Syuzhet)], 79)
```

1) Basic Comment Sentiments

Most positive Comment "I am going to watch but i am so scared you are going to fuck up for me the one civic duty i have done since i legally could...work the census. I am weirdly passionate that the numbers are correct. My favorite house i every had to verify was the counties burn house. I really do enjoy doing this and i sent in my application last week for work it for the 3rd time. Edit: thank you. I believe we can only make our country better is by having a well educated, healthy, food and housing secure population. Things are not perfect but i have hope and making sure that funds are properly distrubute as laws hopefully change (i have hope). Encourage eduation and open mindedness in this upcoming group.) is something i think is extremely important."

Most negative Comment "Please tell me how it's worse. From my perspective, it seems that the fostering of government dependence, unions, loss of jobs, stop and frisk, and gun restriction in detroit and chicago are more damaging than any republican policy. Giving tax cuts to walmart is bad policy, but its not destroying my neighborhoods. Forcing everyone to become TSA agents and city workers has definitely destroyed my neighborhoods. The failure of the party is so obvious, all you can do is pass off the blame by calling the opposition racist and claiming the party is reformed. What are the evil policies hurting blacks? I have some ideas."

1) Basic Comment Sentiments

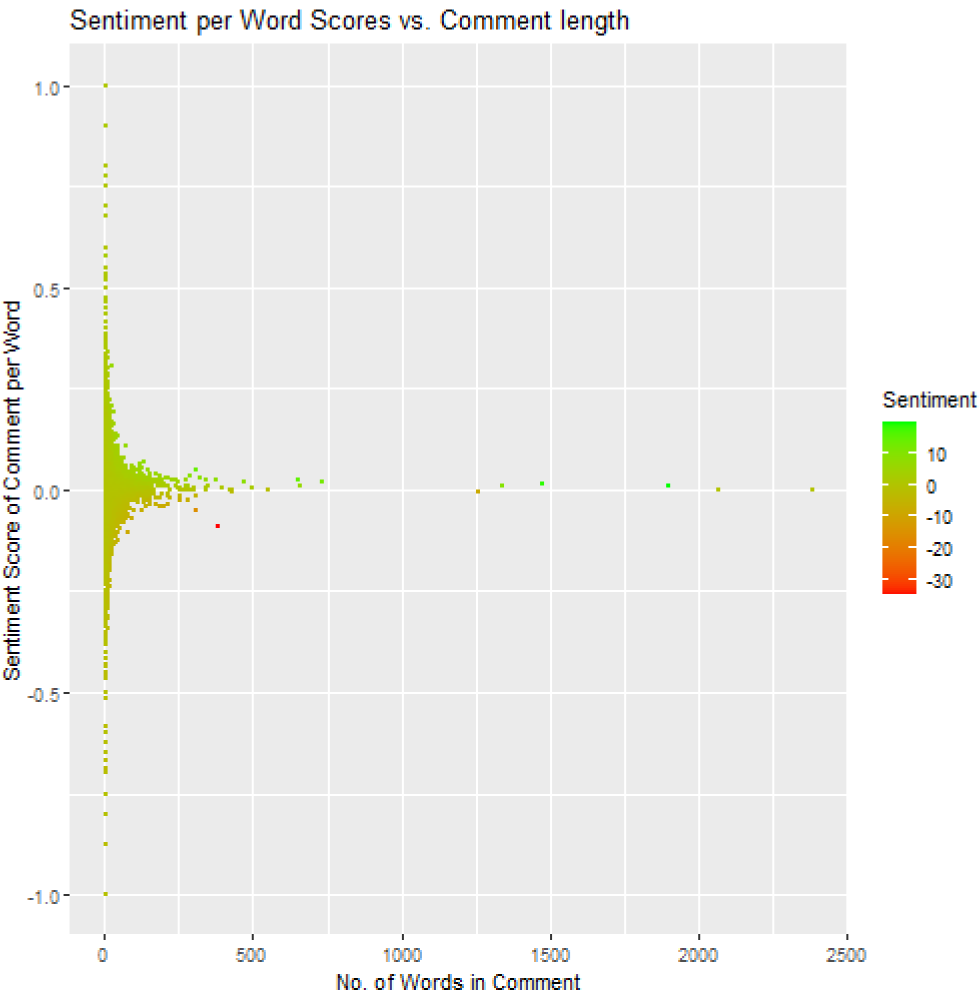


Basic Comment Sentiment

To control for the effect of comment length, we can divide the sentiment score by the number of words in the comment to get a new indicator: *SentimentPerWord*

```
# Normalizing for number of Words
comments$SentimentPerWord <- comments$Sentiment / comments$Words

# Most positive comment
head(comments$TextEmojiDeleted[comments$Sentiment ==
                                 max(comments$SentimentPerWord,
                                     na.rm = T)],1)

# Most negative comment
head(comments$TextEmojiDeleted[comments$Sentiment ==
                                 min(comments$SentimentPerWord,
                                      na.rm = T)],1)
```

Basic Comment Sentiment



More elaborate Method(s)

2) More elaborate Method(s)

Although no sentiment detection method is perfect, some are more sophisticated than others. Two of those options are

- the `sentimentR` package
- the **Stanford coreNLP** utilities set

`sentimentr` attempts to take into account:

- valence shifters
- negators
- amplifiers (intensifiers),
- de-amplifiers (downtoners),
- adversative conjunctions

Negators appear ~20% of the time a polarized word appears in a sentence. Conversely, adversative conjunctions appear with polarized words ~10% of the time. Not accounting for the valence shifters could significantly impact the modeling of the text sentiment.

2) More elaborate Method(s)

Stanford coreNLP utilities set:

- build in Java
- very performant
- tricky to get to work from `R`
- documentation

We will be using `sentimentR` for this session as it represents a good trade-offbetween usability, speed, and performance.

2) SentimentR

First, we need to install and load the package

```
if ("sentimentr" %in% installed.packages() == FALSE) {
  install.packages("sentimentr")
}

library(sentimentr)
```

then we can compute sentiment scores

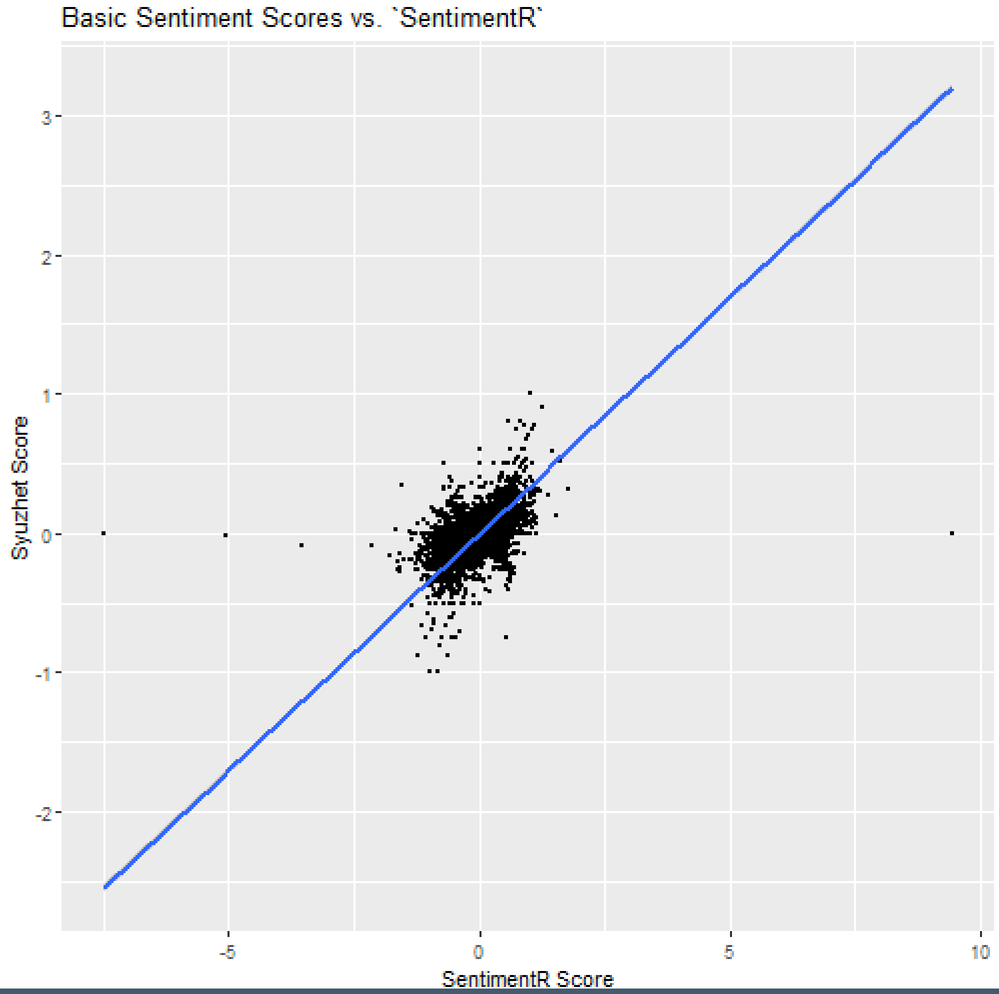
```
# computing sentiment scores
Sentences <- get_sentences(comments$TextEmojiDeleted)
SentDF <- sentiment_by(Sentences)
comments <- cbind.data.frame(comments,SentDF[,c(2,3,4)])
colnames(comments)[c(15,16,17)] <- c("word_count",
                                     "sd",
                                     "ave_sentiment")
```

2) SentimentR

Let's check if the sentiment scoring for sentimentR correlates with the simpler approach

```
# plotting SentimentPerWord vs. SentimentR
ggplot(comments, aes(x=ave_sentiment, y=SentimentPerWord)) +
    geom_point(size =0.5) +
    ggtitle("Basic Sentiment Scores vs. `SentimentR`") +
    xlab("SentimentR Score") +
    ylab("Syuzhet Score") +
    geom_smooth(method=lm, se = TRUE)
```

2) SentimentR

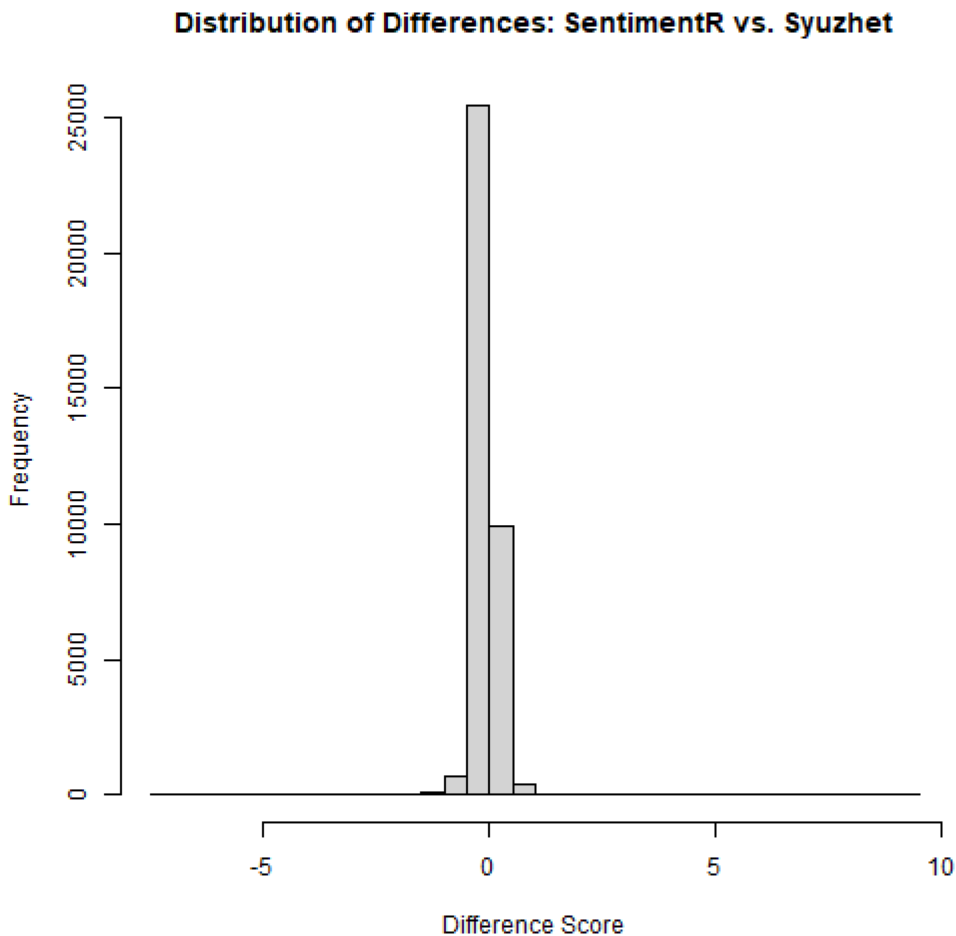


2) SentimentR

We can also look at the difference scores for the two methods

```
#computing difference score
comments$SentiDiff <- comments$ave_sentiment-
                      comments$SentimentPerWord

hist(comments$SentiDiff,
     main= "Distribution of Differences:
      SentimentR vs. Syuzhet",
     xlab = "Difference Score",
     ylab = "Frequency",
     breaks = 50)
```

SentimentR


Distribution of Differences: SentimentR vs. Syuzhet
Frequency
0
5000
10000
15000
20000
25000
-5
0
5
10
Difference Score

2) SentimentR

Let's check for which comments we get the biggest differences between the two methods. *Note*: A bigger difference means that `SentimentPerWord` is more positive than `SentimentR`

```
# top 5 maximum difference comments
strwrap(comments[order(comments$SentiDiff),c(2)][1:5],79)
```

```
##  [1] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [2] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [3] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [4] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [5] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [6] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [7] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [8] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
##  [9] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
## [10] "CRINGE"
## [11] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
## [12] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
## [13] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
## [14] "CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRINGE CRING
## [15] "CRINGE CRINGE CRINGE CRINGE CRINGE I need some clorox"
```

2) `SentimentR`

`SentimentR` is:

- better at dealing with negations
- better at detecting fixed expressions
- better at detecting adverbs
- better at detecting slang and abbreviations
- easy to implement
- quite fast

Sentiments for Emoji

3) Sentiments for Emoji

Emoji are often used to confer emotions (hence the name), so they might be a valuable addition to assess the sentiment of a comment. This is less straight-forward than assessing sentiments based on word dictionaries for multiple reasons:

- Emoji can have multiple meanings: 🙏
- They are highly context-dependent: 🍆
- They are culture-dependent: 🍑
- They are person-dependent: 🤣 😂

3) Sentiments for Emoji

In addition, emoji are rendered differently on different platforms, eliciting different emotions.

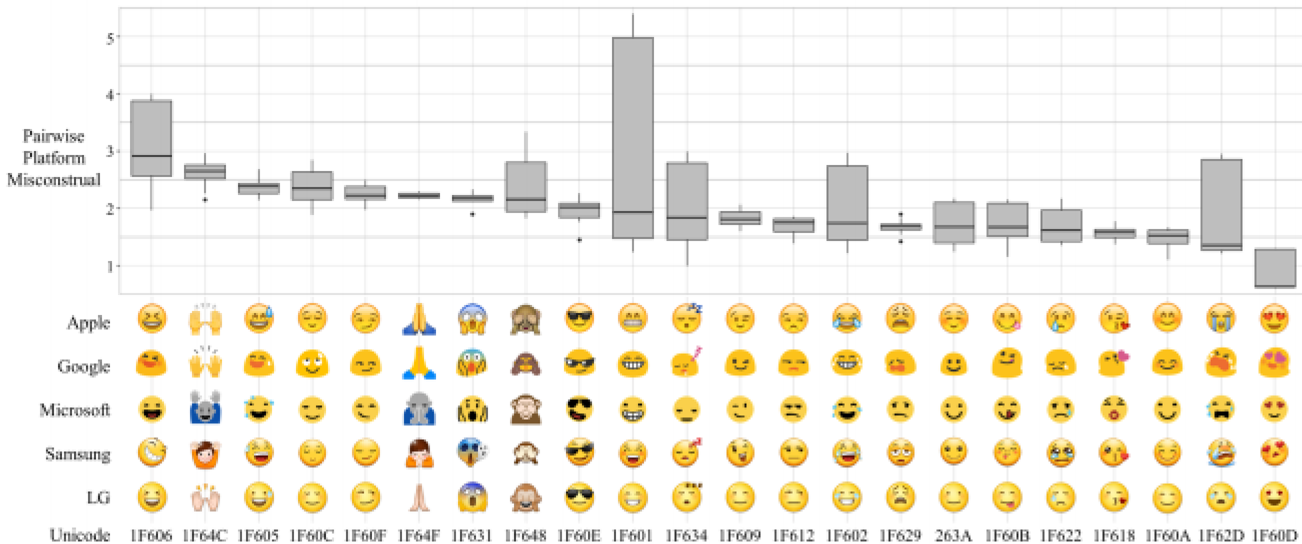


Figure 1. Across-platform sentiment misconstrual scores grouped by Unicode. Each boxplot shows the range of sentiment misconstrual scores across the five platforms. They are ordered by decreasing median platform-pair sentiment misconstrual, from left to right.

Source: Miller et al., 2016

3) Sentiments for Emoji

Emoji are also notoriously difficult to deal with from the technical side due to the infamous character encoding hell

- Emoji can come in one of multiple completely different encodings
- Your operating system has a default encoding that is used when opening/writing files in a text editor
- Your `R` installation has a default encoding that gets used when opening/writing files

If either of those mismatch at any point, you can accidentally overwrite the original encoding in a non-recoverable way. To us, this happened quite often with UTF-8 encoded files on Windows (the default encoding there is Latin-1252)

◉ Hex ○ Dec ○ Oct

Native	Symbola [1]	Code	UTF-8	UTF-16 LE	Surrogates	Name
😁	😁	`1F601`	`F0` `9F` `98` `81`	`3D` `D8` `01` `DE`	`D83D` `DE01`	GRINNING FACE WITH SMILING EYES

3) Sentiments for Emoji

Luckily, we already saved our emoji in a textual description format and can simply treat them as a character strings for our sentiment analysis. We can therefore proceed in 3 steps:

1) Create a suitable sentiment dictionary for textual descriptions of emoji

2) Compute sentiment scores for comments only based on emoji

3) Compare the emoji sentiment scores with the text-based sentiments

Emoji Sentiment Dictionary

We will use the emoji sentiment dictionary from the `lexicon` package. It only contains the 734 most frequent emoji, but since the distribution of emoji follows Zipf's Law, it should cover most of the used emoji.

```
# emoji Sentiments
EmojiSentiments <- lexicon::emojis_sentiment
EmojiSentiments[1:5,c(1,2,4)]
```

```
##                 byte                            name sentiment
## 1 <f0><9f><98><80>                   grinning face 0.5717540
## 2 <f0><9f><98><81>  beaming face with smiling eyes 0.4499772
## 3 <f0><9f><98><82>          face with tears of joy 0.2209684
## 4 <f0><9f><98><83>     grinning face with big eyes 0.5580431
## 5 <f0><9f><98><84> grinning face with smiling eyes 0.4220315
```

in comparison, our data looks like this:

```
# example from our data
comments$Emoji[85]
```

```
## [1] "EMOJI_RollingOnTheFloorLaughing EMOJI_RollingOnTheFloorLaughing EMOJI
```

Emoji Sentiment Dictionary

We bring the textual description in the dictionary in line with the formatting in our data so we can replace one with the other using standard text manipulation techniques

```
# chang formatting in the emoji dictionary
EmojiNames <- paste0("emoji_",gsub(" ","",EmojiSentiments$name))
EmojiSentiment <- cbind.data.frame(EmojiNames,
                                   EmojiSentiments$sentiment,
                                   EmojiSentiments$polarity)
names(EmojiSentiment) <- c("word","sentiment","valence")
EmojiSentiment[1:5,]
```

```
##                                 word sentiment  valence
## 1                  emoji_grinningface 0.5717540 positive
## 2  emoji_beamingfacewithsmilingeyes 0.4499772 positive
## 3            emoji_facewithtearsofjoy 0.2209684 positive
## 4       emoji_grinningfacewithbigeyes 0.5580431 positive
## 5 emoji_grinningfacewithsmilingeyes 0.4220315 positive
```

Emoji Sentiment Dictionary

```r
# tokenize the emoji-only column in our formatted dataframe
EmojiToks <- tokens(tolower(as.character(unlist(comments$Emoji))))
EmojiToks[130:131]
```

```
## Tokens consisting of 2 documents.
## text130 :
## [1] "emoji_facewithtearsofjoy" "emoji_facewithtearsofjoy"
##
## text131 :
## [1] "na"
```

Computing Sentiment Scores

We can now replace the emojis that appear in the dictionary with the corresponding sentiment scores

```
# Creating the dictionary object
EmojiSentDict <- as.dictionary(EmojiSentiment[,1:2])

# Replacing emoji with sentiment scores
EmojiToksSent <- tokens_lookup(x = EmojiToks,
                               dictionary = EmojiSentDict)
EmojiToksSent[130:131]
```

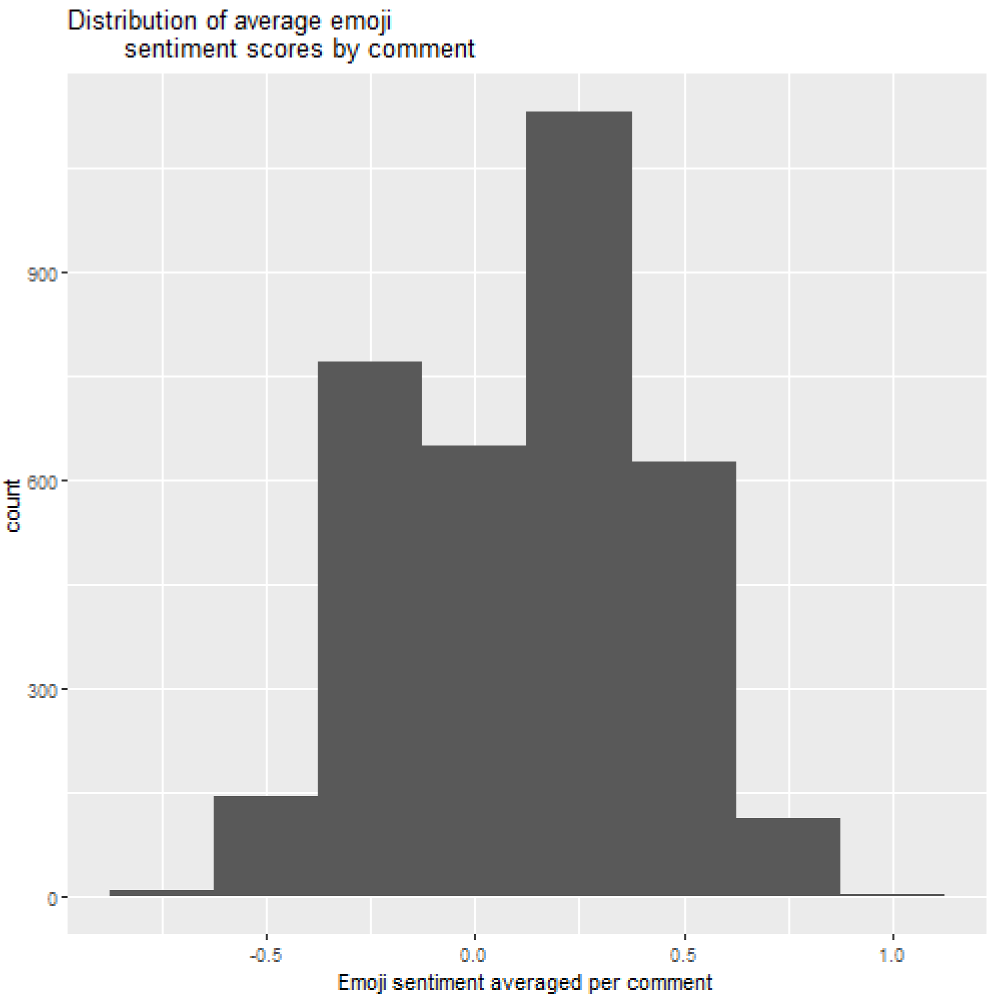
```
## Tokens consisting of 2 documents.
## text130 :
## [1] "0.220968403775133" "0.220968403775133"
##
## text131 :
## character(0)
```

Computing Sentiment Scores

```
# only keep the assigned sentiment scores for the emoji vector
AllEmojiSentiments <- tokens_select(EmojiToksSent,EmojiSentiment$sent
                                    "keep")
AllEmojiSentiments <- as.list(AllEmojiSentiments)

# define function to average emoji sentiment scores  per comment
MeanEmojiSentiments <- function(x){

  x <- mean(as.numeric(as.character(x)))
  return(x)

}

# apply the function to every comment that contains emojis
MeanEmojiSentiment <- lapply(AllEmojiSentiments,MeanEmojiSentiments)
MeanEmojiSentiment[MeanEmojiSentiment == 0] <- NA
MeanEmojiSentiment <- unlist(MeanEmojiSentiment)
MeanEmojiSentiment[130:131]
```

```
##   text130   text131
## 0.2209684       NaN
```

Emoji Sentiment Scores



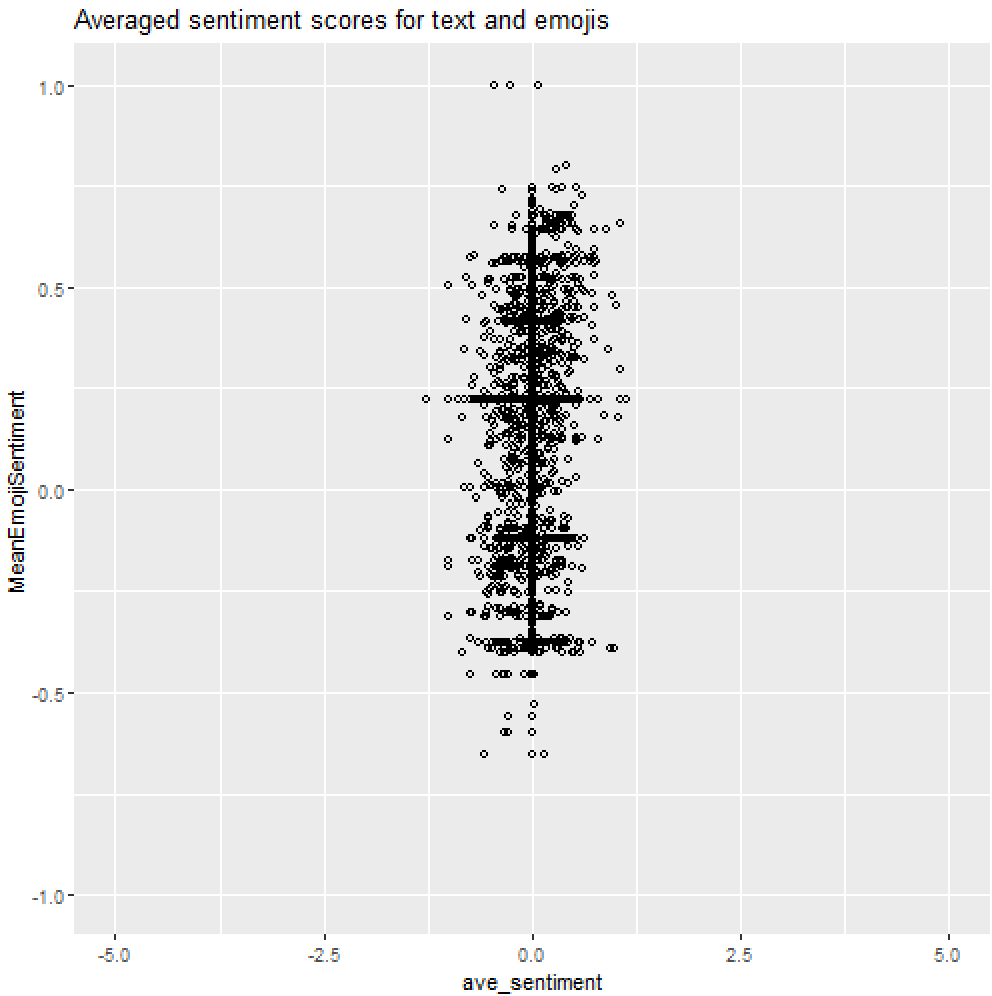
Emoji Sentiment vs. Word Sentiment

```r
comments <- cbind.data.frame(comments,MeanEmojiSentiment)

# correlation between averaged emoji sentiment score
#   and averaged text sentiment score
cor(comments$ave_sentiment,
    comments$MeanEmojiSentiment,
    use="complete.obs")
```

```
## [1] 0.1423646
```

```r
# plot the relationship
ggplot(comments, aes(x = ave_sentiment,
                     y = MeanEmojiSentiment))+
  geom_point(shape = 1) +
  labs(title = "Averaged sentiment scores for text and emojis") +
  scale_x_continuous(limits = c(-5,5)) +
  scale_y_continuous(limits = c(-1,1))
```

Emoji Sentiment vs. Word Sentiment



Emoji Sentiment vs. Word Sentiment

As we can see, there seems to be no meaningful relationship between the sentiment scores of the text and the sentiment of the used emojis. This can have multiple reasons:

- Comments that score very high (positive) on emoji sentiment typically contain very little text
- Comments that score very low (negative) on emoji sentiment typically contain very little text
- dictionary based bag-of-words/-emojis sentiment analysis is not perfect - there is a lot of room for error in both metrics
- most comment text and emoji sentiments are neutral
- emojis are very much context-dependent, but we only consider a single sentiment score for each emoji
- we only have sentiment scores for the most common emoji

Emoji Sentiment vs. Word Sentiment

The data is clustered around vertical and horizintal lines:

- skewed distribution of number of emojis per comment and types of emojis used (e.g., using the one emoji exactly once is by far the most common case for this particular video)
- most common average sentiment per word is zero

Exercise time 🏋️ 💪 🏃 🚴

Solutions