Automatic Sampling and Analysis of YouTube Data

Sentiment Analysis of User Comments

Julian Kohne
Johannes Breuer
M. Rohangis Mohseni

2022-02-22

Setup

Note: In previous versions of `R`, all strings were automatically translated to factors when reading data. This has been changed with version 4.0.0.

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

# Load the data set
comments <- readRDS("../../data/ParsedEmojiComments.rds")
```

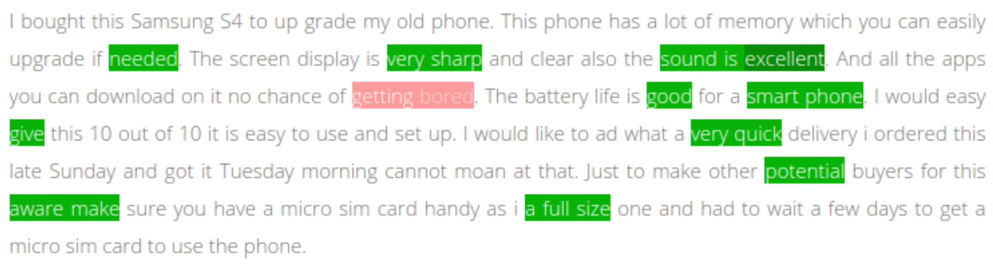
Sentiment Analysis

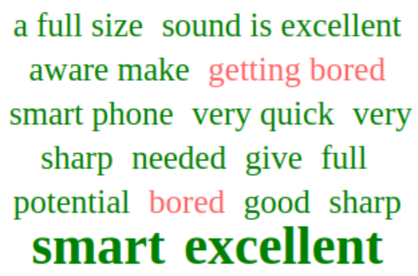
- The basic task of sentiment analysis is to detect the *polarity* of a sentence or a collection of sentences in terms of positivity and negativity
- Sentiment is often used in market research for product reviews
- For *YouTube* videos, we can look at the sentiment in the comments to quantify the valence of user reactions
- *Note*: There are also 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


I bought this Samsung S4 to up grade my old phone. This phone has a lot of memory which you can easily upgrade if needed. The screen display is very sharp and clear also the sound is excellent. And all the apps you can download on it no chance of getting bored. The battery life is good for a smart phone. I would easy give this 10 out of 10 it is easy to use and set up. I would like to ad what a very quick delivery i ordered this late Sunday and got it Tuesday morning cannot moan at that. Just to make other potential buyers for this aware make sure you have a micro sim card handy as i a full size one and had to wait a few days to get a micro sim card to use the phone.



Basic Sentiment Analysis

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

```
##              x     y
##  1:     esprit  0.80
##  2:    laidoff -0.60
##  3:       fine  0.25
##  4:  worrisome -1.00
##  5:  isolation -0.50
##  6:      guide  0.60
##  7: descendant  0.25
##  8:   countess  0.25
##  9: demonizing -1.00
## 10: blistering -0.50
```

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 negative (unpopular, hate)

```
comments$TextEmojiDeleted[6657]
```

```
## [1] "(unpopular opinion) Why does everyone hate this movie"
```

Classified as positive (genuinely, enjoy)

```
comments$TextEmojiDeleted[260]
```

```
## [1] "I don’t understand how people could genuinely enjoy this"
```

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.

Basic Comment Sentiments

First of all, we load the `syuzhet` package and try out the built-in basic sentiment tagger with an example sentence

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

# load package
library(syuzhet)

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

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

Basic Comment Sentiments

We can apply the basic sentiment tagger to the whole vector of comments in our data set. Remember that we need to use the text column without hyperlinks and emojis.

```r
# create basic sentiment scores
BasicSentiment <- get_sentiment(comments$TextEmojiDeleted)

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

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -34.45000  -0.50000   0.00000  -0.06833   0.25000  19.95000
```

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.

Basic Comment Sentiments

Let's compare the results of the different dictionaries

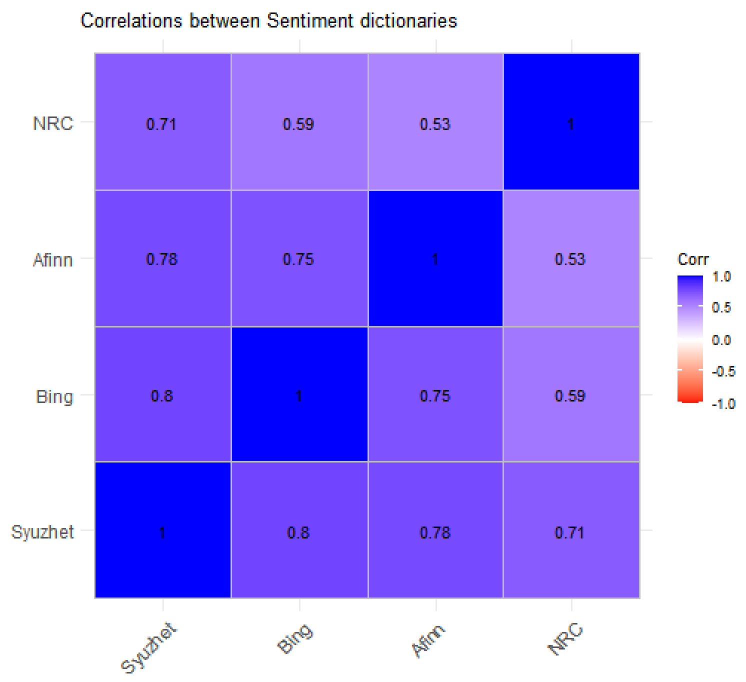
```r
# compute sentiment scores with different dictionaries
BasicSentimentSyu <- get_sentiment(comments$TextEmojiDeleted,
                                   method = "syuzhet")
BasicSentimentBing <- get_sentiment(comments$TextEmojiDeleted,
                                    method = "bing")
BasicSentimentAfinn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

Basic Comment Sentiments

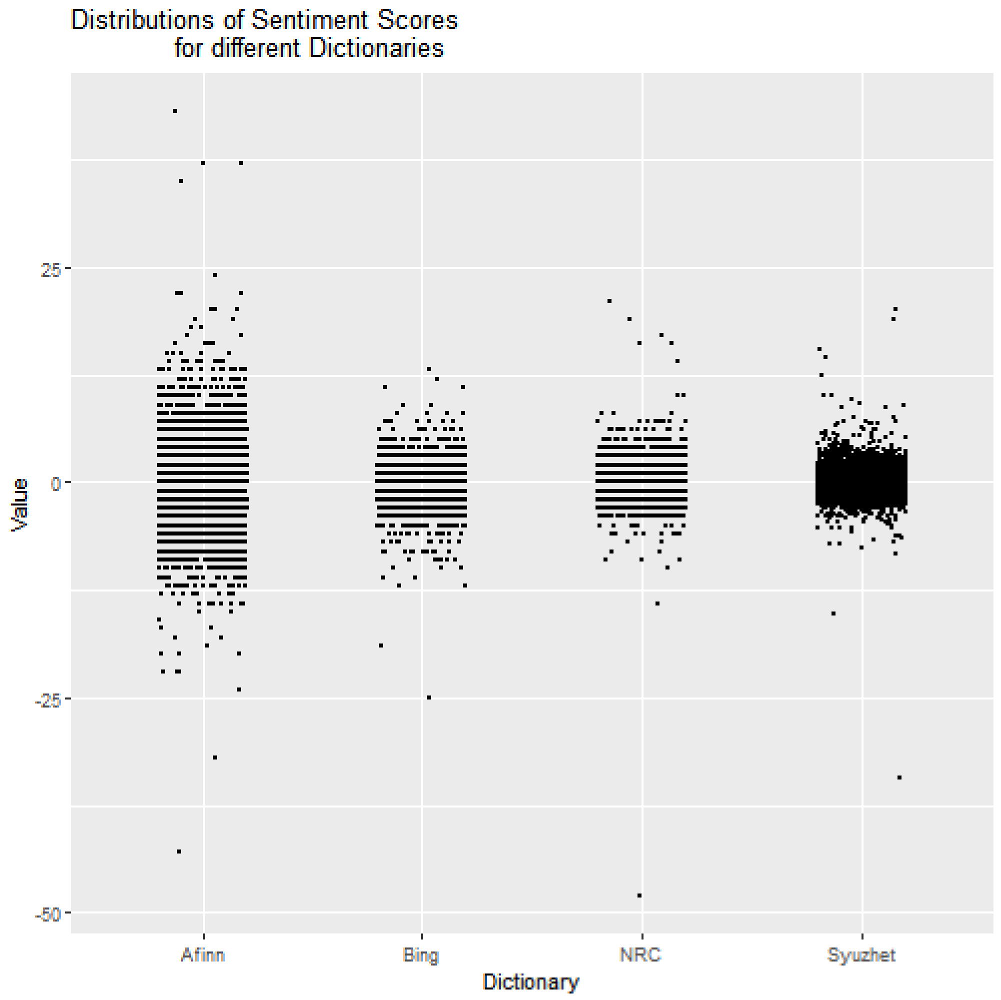
```r
# combine them into a dataframe
Sentiments <- cbind.data.frame(BasicSentimentSyu,
                               BasicSentimentBing,
                               BasicSentimentAfinn,
                               BasicSentimentNRC,
                               1:dim(comments)[1])
# set column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

Basic Comment Sentiments

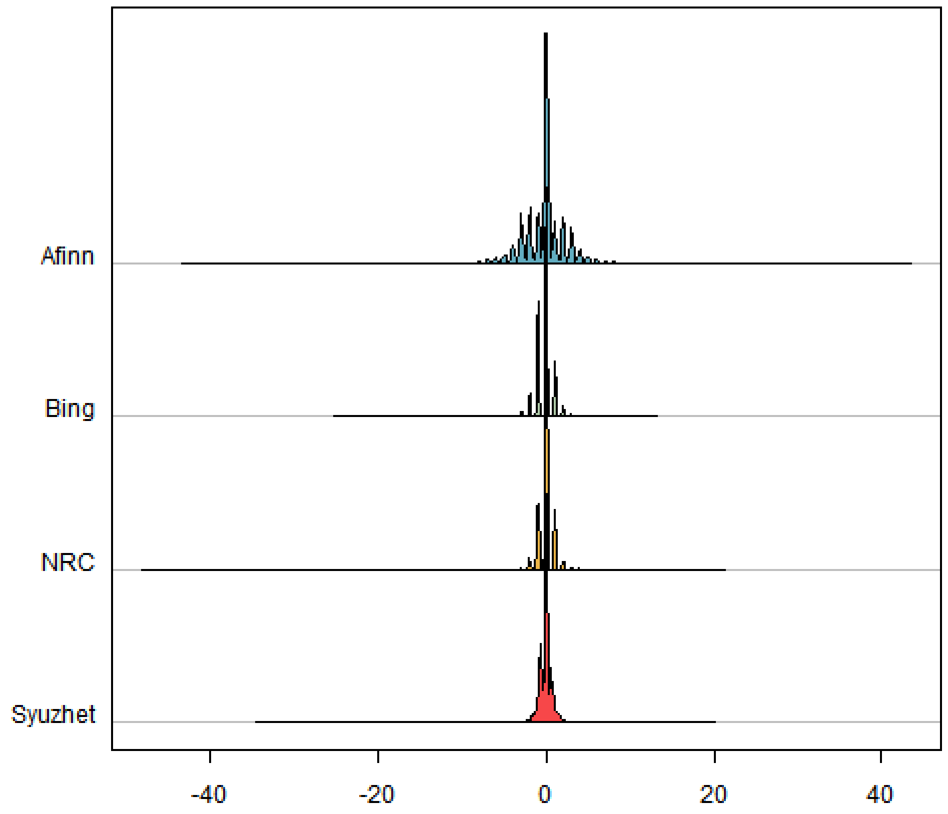
```r
# correlation matrix
library(ggcorrplot)
ggcorrplot(cor(Sentiments[,c(-5)]),
           color=c("red","white","blue"),
           title="Correlations between Sentiment dictionaries",
           lab=TRUE)
```



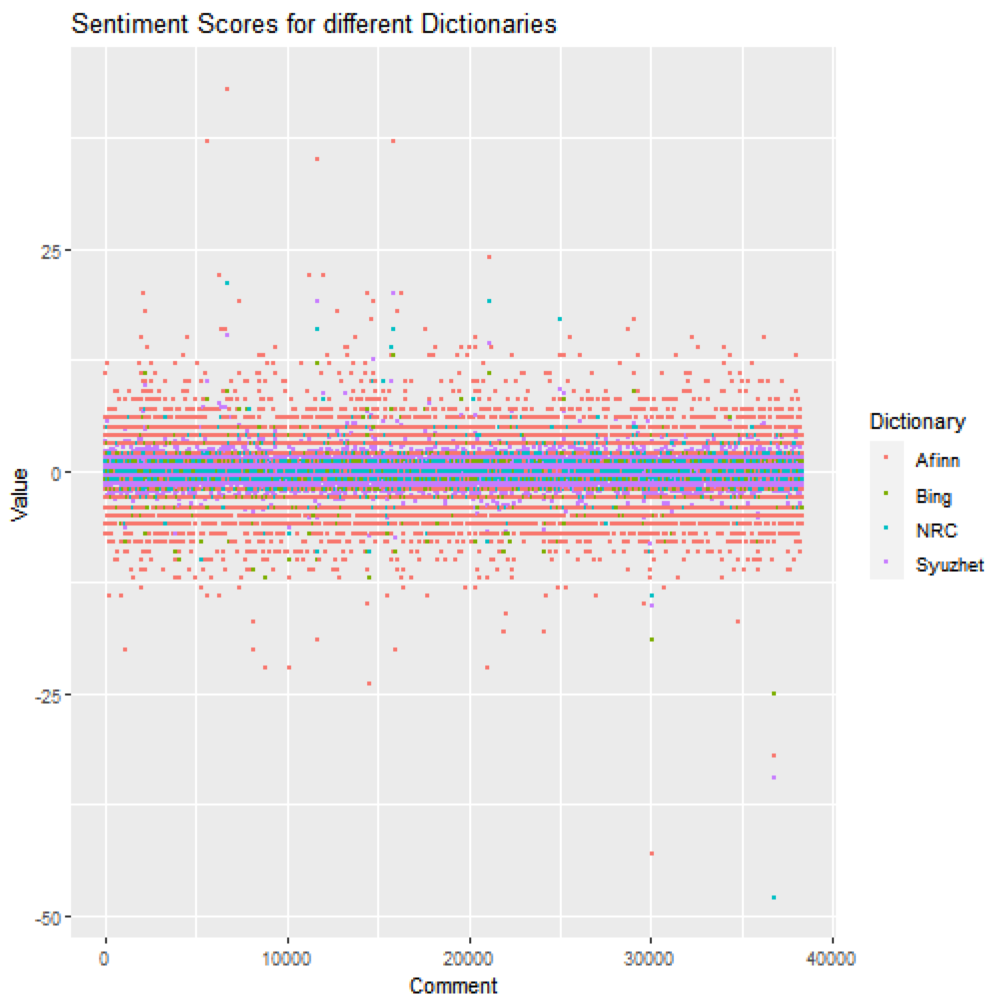
Basic Comment Sentiments: Jitter Plot



Basic Comment Sentiments: Ridgeline Plot



Basic Comment Sentiments: Scatter Plot



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.

```
# add the syuzhet sentiment scores to our dataframe
comments$Sentiment <- Sentiments$Syuzhet
```

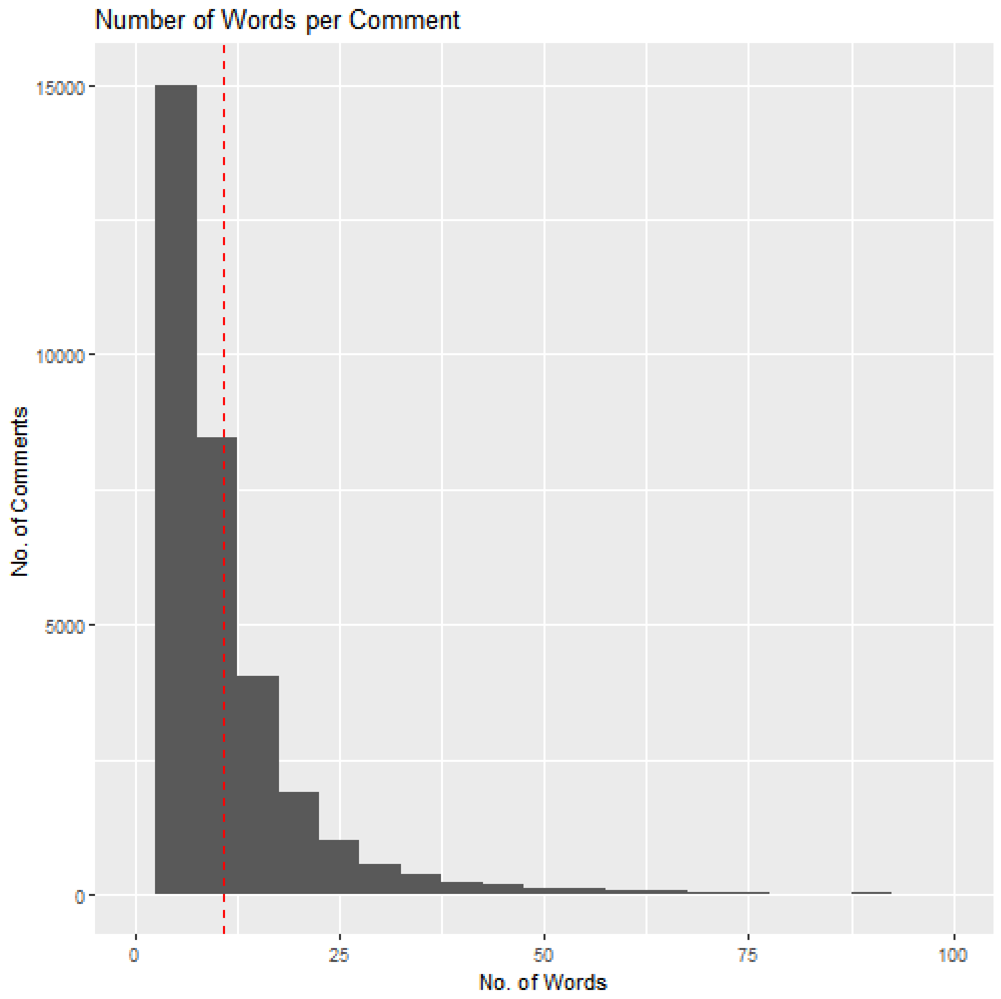
Basic Comment Sentiments

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

```
# compute 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 = 5) +
  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")+
  xlim(0, 100)
```

Basic Comment Sentiments



Basic Comment Sentiments

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

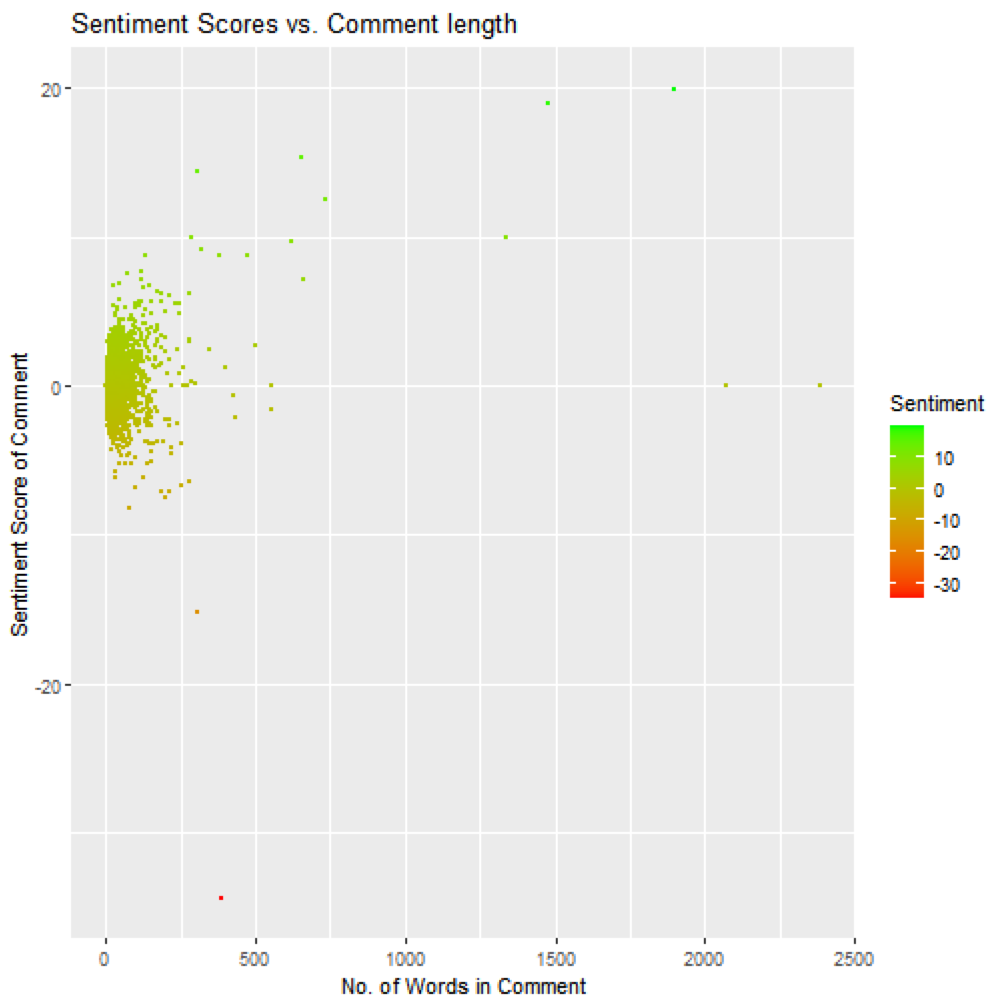
```
# Very positively scored comment
comments$TextEmojiDeleted[6704]

# Very negatively scored comment
comments$TextEmojiDeleted[29952]
```

Basic Comment Sentiments

Positively scored Comment "The emoji movie is a beautifully executed, well layered experience that has been the victim of a barrage of such biased reviews such as “The worst movie of 2017” , “ a very formulaic story” and “9% on Rotten Tomatoes.” So, in defense of this movie, i will give you, the reader, why The Emoji Movie is in fact the best movie of in 2017. Remember, though, that this essay has some of my opinions, and takes on certain parts of the film. The Story. The story is a beautifully written, homage to many fairy tales. If you have not watched this movie yet, here's the rundown of the story. Gene is a meh emoji in a world of other emoji’s on a teenager’s smartphone. Gene, not wanting to be like the other emoji’s, goes out to find himself. Along the way, […]"

Negatively Comment How far we've fallen from the light of Yeshua? Only for our arrogance were we to be consumed by the Leviathon and trapped within the belly of the beast as it slowly bleeds out as our wicked and unwashed cries of agony fall on deaf ears, cursed to toil within and gaze back on our decaying with envious eyes as death hadn't taken us first. Heh, I dunno… Emojis are pretty much a line of communication for minorities drug deals/hookups. […]"

Basic Comment Sentiments

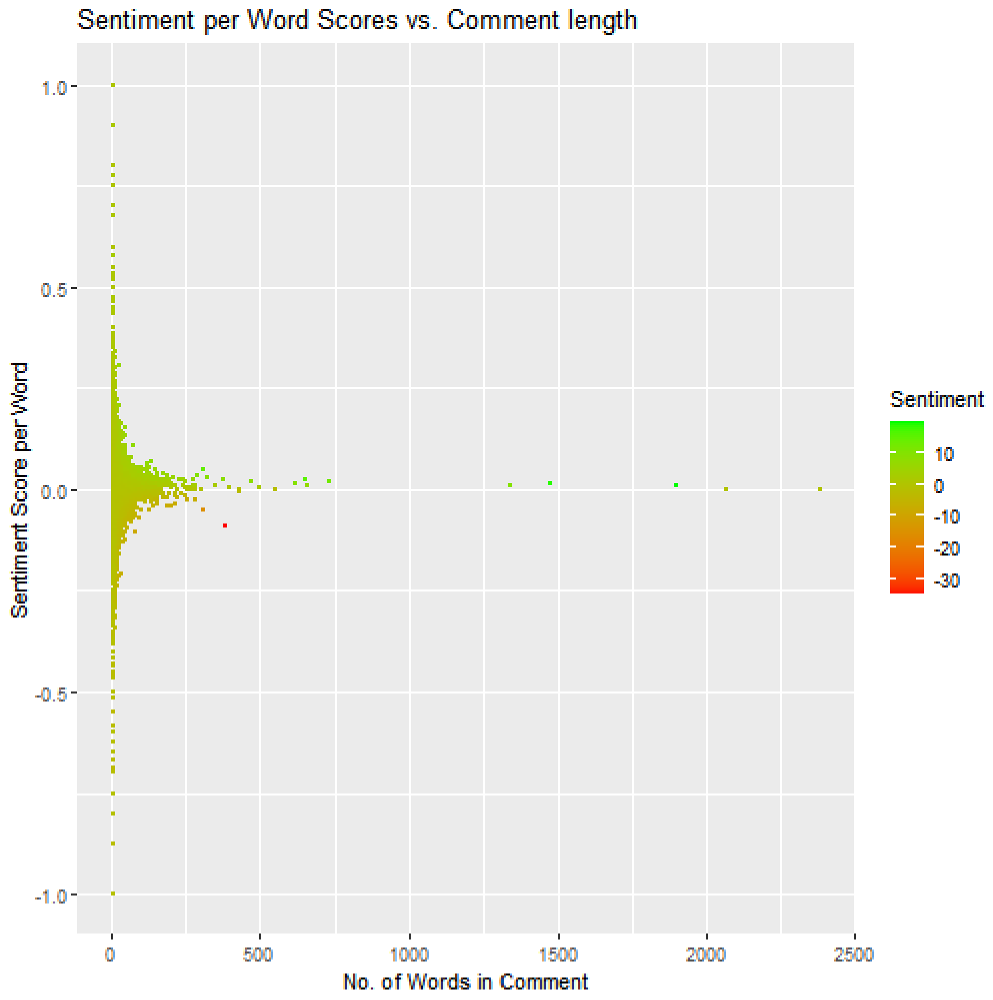


Basic Comment Sentiments

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*

```r
# normalize 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 Sentiments



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, hence, significantly impact the modeling of the text sentiment.

More elaborate Method(s)

Stanford coreNLP utilities set:

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

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

SentimentR

First, we need to install and load the package

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

library(sentimentr)
```

then we can compute sentiment scores

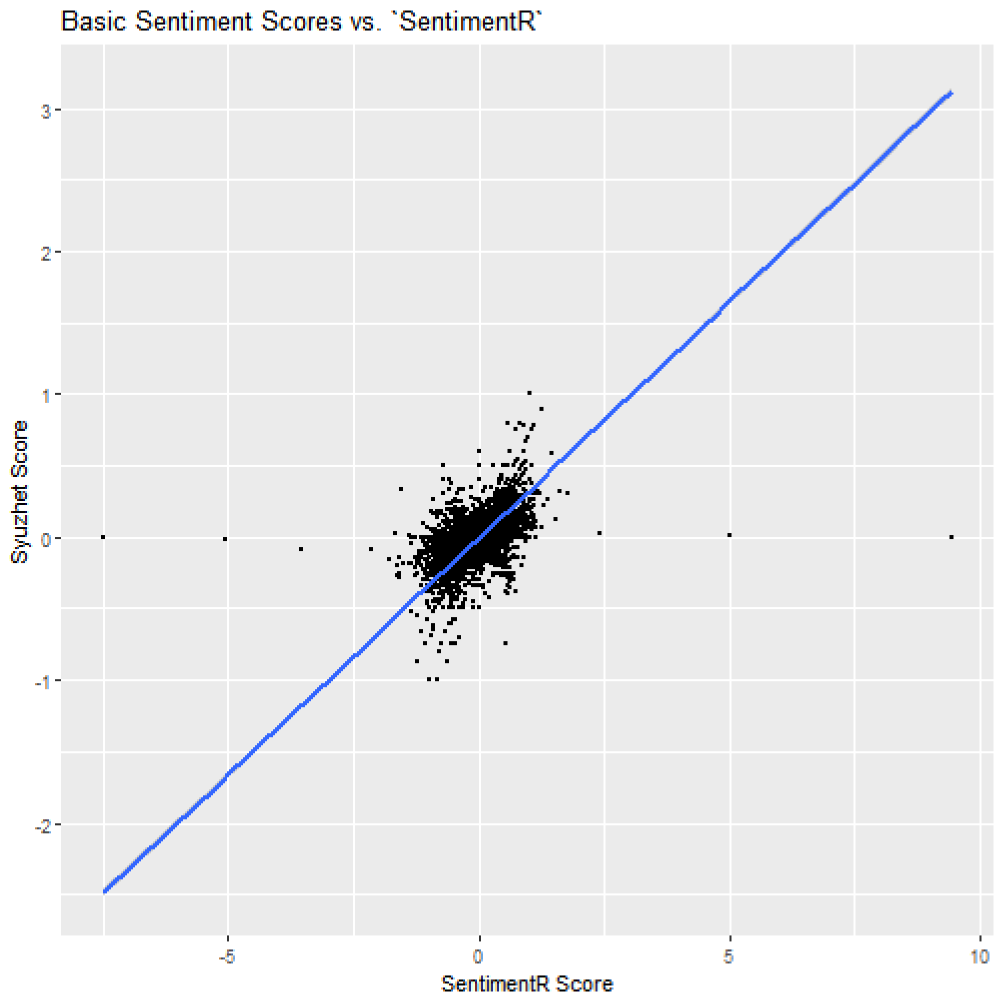
```r
# compute sentiment scores
Sentences <- get_sentences(comments$TextEmojiDeleted)
SentDF <- sentiment_by(Sentences)
comments <- cbind.data.frame(comments,SentDF[,c(2,3,4)])
colnames(comments)[c(15,16,17)] <- c("word_count",
                                     "sd",
                             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

SentimentR

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

```r
# plot 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)
```

SentimentR

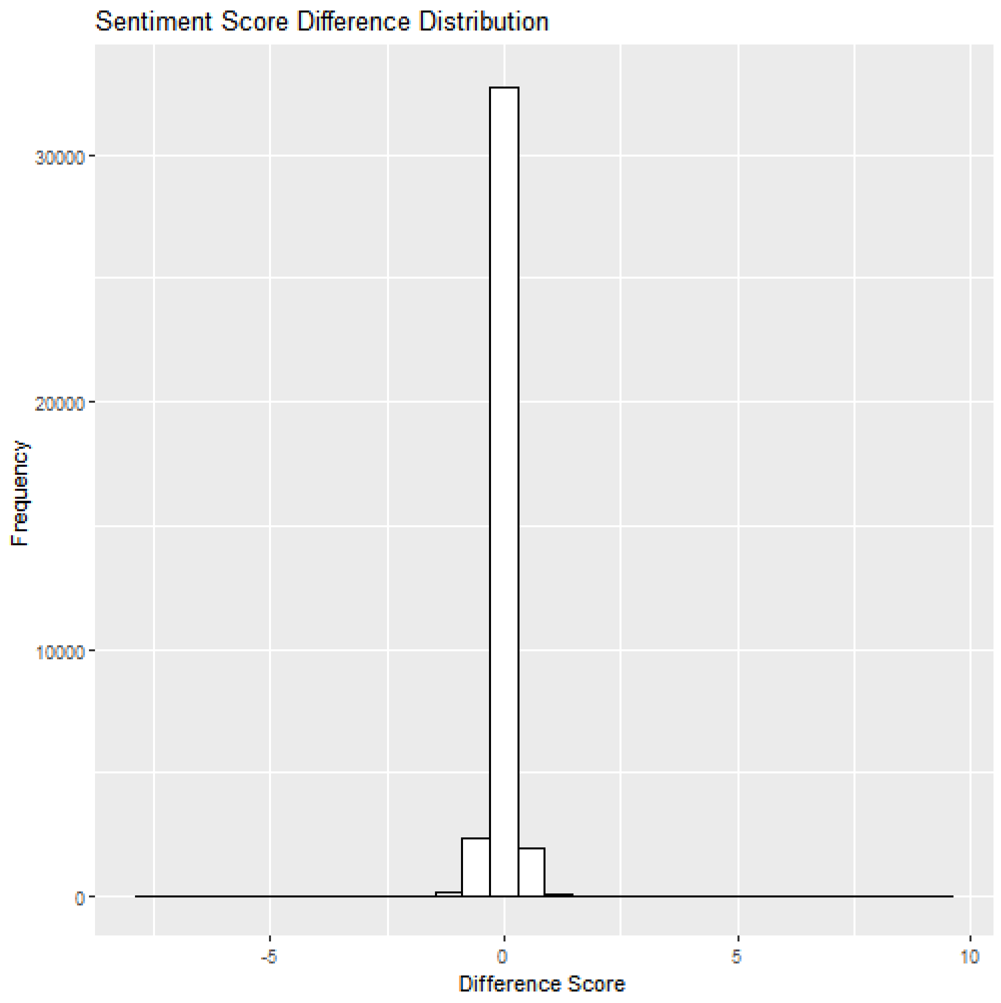

Basic Sentiment Scores vs. `SentimentR`
Syuzhet Score
SentimentR Score
3
2
1
0
-1
-2
-5
0
5
10

SentimentR

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

```r
#compute difference score
comments$SentiDiff <- comments$ave_sentiment-
                      comments$SentimentPerWord

ggplot(comments, aes(x=SentiDiff)) +
  geom_histogram(color="black", fill="white")+
  labs(title="Sentiment Score Difference Distribution",
       x="Difference Score",
       y = "Frequency")
```

SentimentR


Sentiment Score Difference Distribution
Frequency
30000
20000
10000
0
-5
0
5
10
Difference Score

SentimentR

Let's check for which comments we get large differences between the two methods. *Note*: We use an absolute difference score here and order from largest to smallest difference.

```
# illustrate scoring differences
Diff_comments <- comments[order(-abs(comments$SentiDiff)),
                          ][c(2,12:18)]
comments[c(4440,2839,37718,136),c(2, Syuzhet = 12, SentimentR = 17)]
```

```
##                                          Text Sentiment ave_sentiment
## 4440                                NOT FUNNY      0.80    -0.5656854
## 2839                               Yeah right      0.80    -0.7071068
## 37718 This isn't nearly as bad as the teaser.     -0.75     0.2651650
## 136                                       Wtf      0.00    -1.0000000
```

SentimentR

Compared to the basic approach, `SentimentR` is:

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

Sentiments for Emojis

Emojis 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:

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

Sentiments for Emojis

In addition, emojis are rendered differently on different platforms, meaning that they can potentially elicit different emotions.

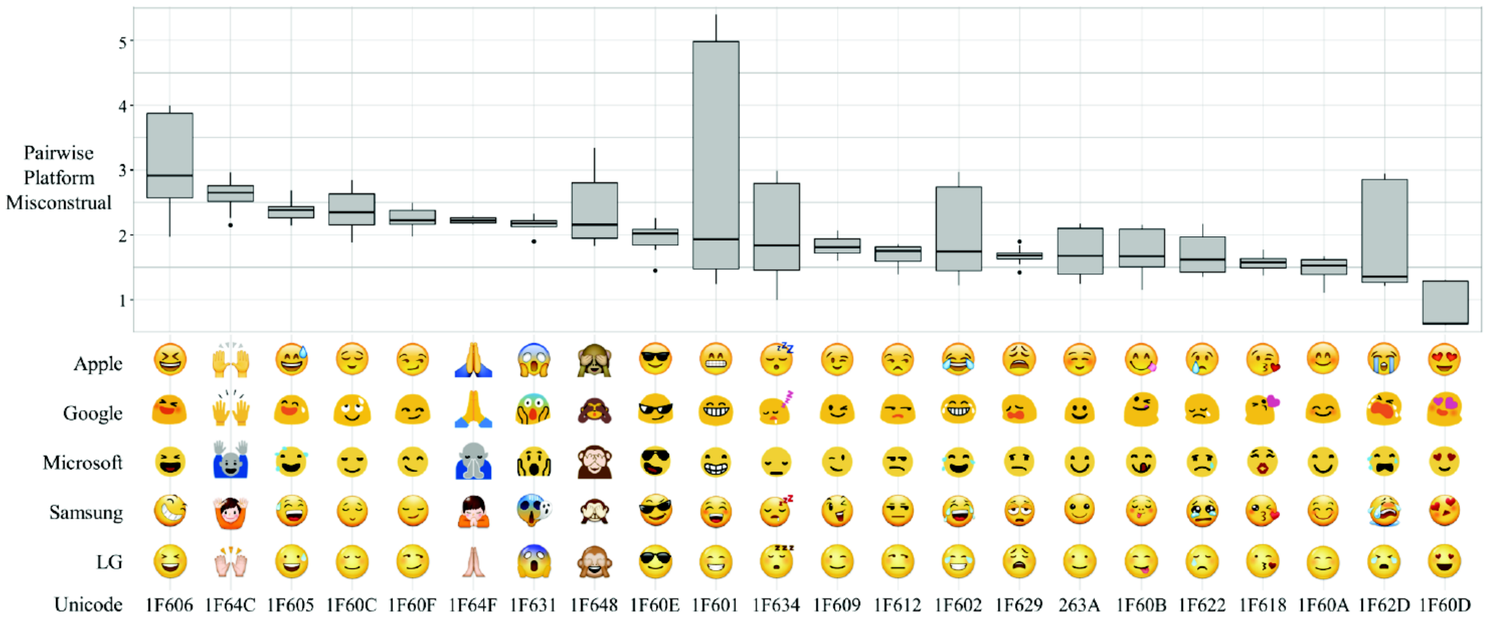


Figure 1. Across-platform sentiment misconstrual scores grouped by Unicode. Each boxplot shows the range of sentiment misconstrual scores across the five platforms. They are ordered by decreasing median platform-pair sentiment misconstrual, from left to right.

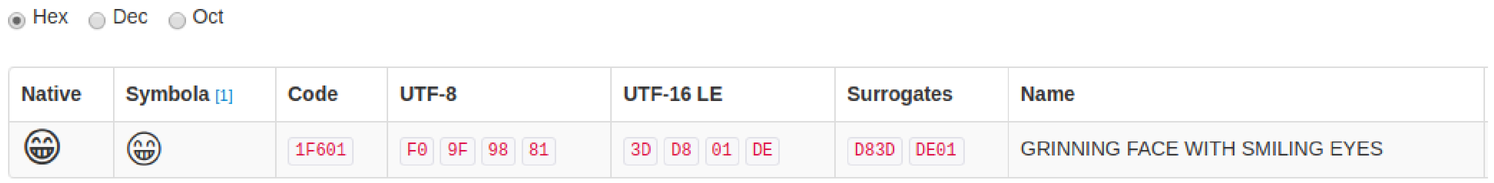
Source: Miller et al., 2016

Sentiments for Emojis

Emojis are also notoriously difficult to deal with from a technical perspective due to the infamous character encoding hell

- Emojis 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. For 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

Sentiments for Emojis

Luckily, we already saved our emojis 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 emojis

2) Compute sentiment scores for comments only based on emojis

3) Compare the emojis 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 emojis, but since the distribution of emojis follows Zipf's Law, it should cover most of the used emojis.

```
# 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
```

By comparison, our data looks like this:

```
# example from our data
comments$TextEmojiReplaced[5546]
```

```
## [1] "Amazing movieEMOJI_GrinningFace "
```

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.

```r
# change 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

```
# tokenize the emoji-only column in our formatted dataframe
EmojiToks <- tokens(tolower(as.character(unlist(comments$Emoji))))
comments$Text[11167]
```

```
## [1] "<U+0001F600>+<U+0001F3A5>=<U+0001F4A9>"
```

```
EmojiToks[11167]
```

```
## Tokens consisting of 1 document.
## text11167 :
## [1] "emoji_grinningface" "emoji_moviecamera"  "emoji_pileofpoo"
```

Computing Sentiment Scores

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

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

# replace emoji with sentiment scores
EmojiToksSent <- tokens_lookup(x = EmojiToks,
                               dictionary = EmojiSentDict)
comments$Text[11167]
```

```
## [1] "<U+0001F600>+<U+0001F3A5>=<U+0001F4A9>"
```

```
EmojiToksSent[11167]
```

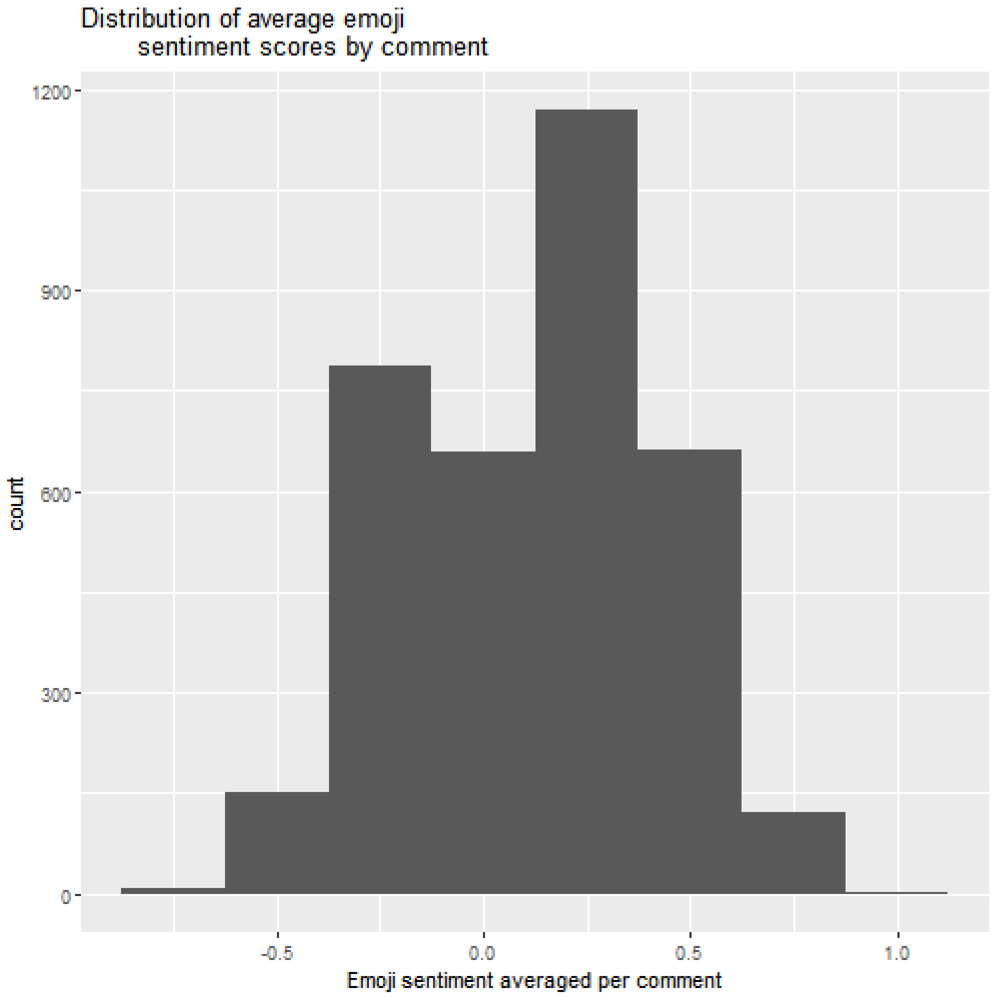
```
## Tokens consisting of 1 document.
## text11167 :
## [1] "0.571753986332574"  "0.303030303030303"  "-0.117903930131004"
```

Computing Sentiment Scores

```r
# only keep the assigned sentiment scores for the emoji vector
AllEmojiSentiments <- tokens_select(EmojiToksSent,EmojiSentiment$sentiment,
                                    "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[11167]
```

```
## text11167
## 0.2522935
```

Emoji Sentiment Scores



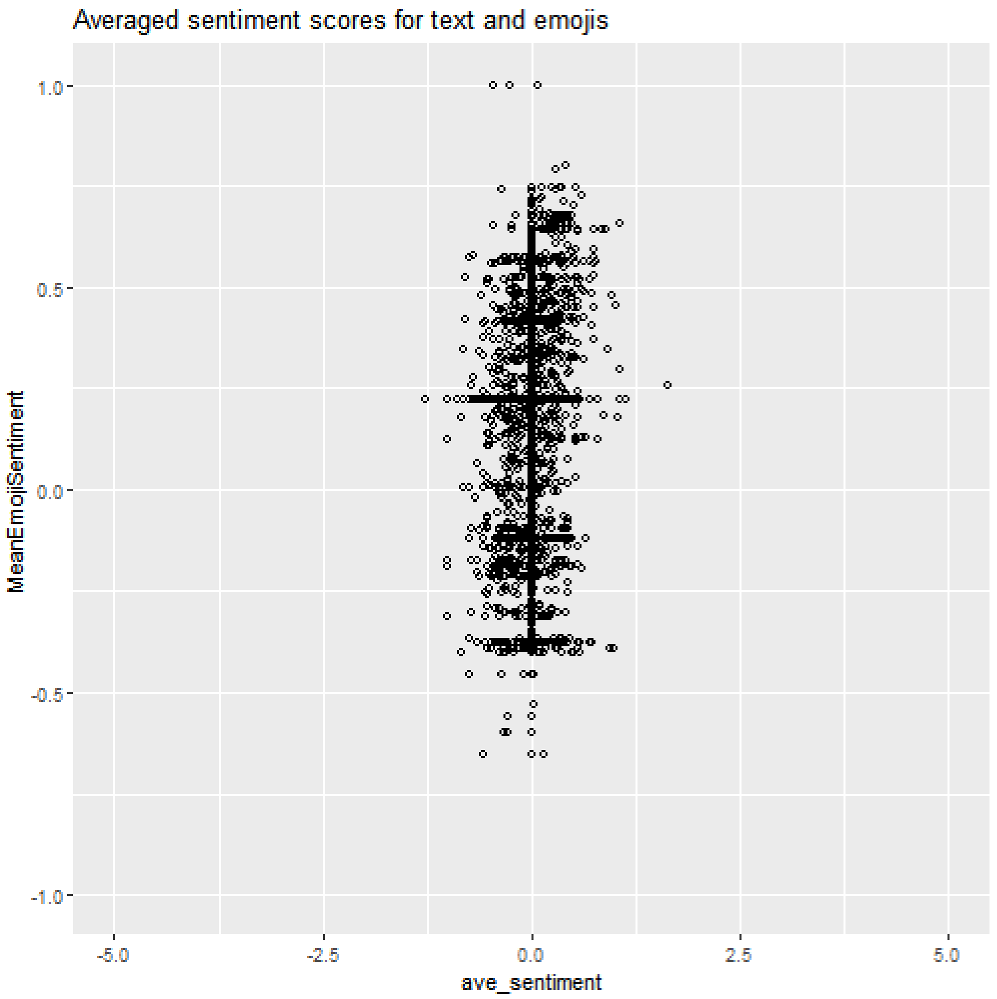
Emoji Sentiment vs. Word Sentiment

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

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

```
## [1] 0.1362876
```

```
# 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 texts are scored as 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
- If comments contain an emoji, it's usually exactly one emoji

Takeaway

From the examples in this data, we have seen multiple things:

- Sentiment detection is not perfect
- Bag-of-words approaches are often too simplistic
- Even more sophisticated methods can often misclassify comments
- The choice of dictionary and sentiment detection method is highly important and can change the results substantially
- Emojis seem to be even more challenging for classifying sentiment

Exercise time 🏋️ 💪 🏃 🚴

Solutions