Assignment 4: Pimp my streamer

In this assignment you will clean and reorganize your code to produce an efficient Twitter streamer.

NEW: 200% more emojis inside (+ native encodings)

For the later part of this assignment use the updated Emoji list including now more Emojis and native encodings. Searching for both native encodings and UTF-8 encoding should yield maximum results. Download the new Emoji list **here**.

1 Create addition access keys

Use your activated SIM cards to create three new twitter accounts. With each new account create a new app and retrieve the consumer key and secret. Create OAuth objects for each new app and authorize them using `my_oauth$handshake()`. Store the new, authorized OAuth objects together with your old one in a list and save the list in your projects folder.

2 Create streaming pipeline

2.1 Comment code

Go through your code of the last three assignments and commment out every bit. Comments are mainly for you to understand every element of the code whenever you need to get back it at a later point. However, it makes sense to think a potentially different reader in order to nudge yourself to be more verbose.

2.2 Streamline code

Now that you commented out your code and have a good representation of what each bit of code does, go through it and streamline it. Think about whether code bits could be reprogrammed in more elegant way. Often this means shortening the code and making it more generic, e.g., by using a loop rather than explicitly writing practically the same code multiple times. This also means adhering to a coherent formatting scheme for new lines and code indentations (Doesn't really matter how as long as it's consistent).

2.3 Write my streamer function

Now that the code is streamlined (and hopefully a bit shortened) create your own `my_streamer()` function. The function should take (at least) three arguments (`track`, `oauth`, `time`) and return the preprocessed and parsed tweets (like the `data_stream` object from assignment 1). Test the function and make sure it runs smoothly. This means that the function doesn't take forever and that it handles all things that could go wrong with the execution of the code. To test this run the function multiple times using differnet input arguments.

2.4 Implement oauth rotation

Now that you have a working function, implement an oauth rotation. The idea is that whenever the `time`-limit is reached rather than returning the result the function begins streaming again using a different OAuth object. This will enable streaming tweets on a single search term for hours or possibly days without Twitter limiting access.

To do this add a new argument to the function called, e.g., `nrep` (number of repetitions) that gives the number of times the streaming should be restarted. Then implement a loop inside your function that iterates over the sequence from 1 to `nrep` and in each cycle restarts the twitter streaming for the current `track` and `time` arguments. Every time the streaming is restarted choose a different OAuth object. An easy way to do this is using the modulo operator `%%`. The modulo operator returns the rest remaining from dividing the number left of the operator by the number right of the operator. For instance, `1%%4` is 1, `2%%4` is 2,`3%%4` is 3, `4%%4` is 0, and `5%%4` is again 1. Thus, the running index of the loop, e.g., `i`, can be be used to create a rotation within the range of 1 and 4 using `i%%4 + 1`. This can then be used to rotate OAuth objects.

```
# Define my streamer
my_streamer = function(track, nrep, time, sleep = 0){

  # load jsonlite
  if(!require(jsonlite)) stop('install jsonlite')
  if(!require(streamR)) stop('install streamR')
  if(!require(ROAuth)) stop('install ROAuth')
  if(!require(magrittr)) stop('install magrittr')

  # load oauths
  oauths = ('MyOAuthsPath.RDS')
  n_auth = length(oauths)

  # loop until nrep
  my_streams = list()
  for(i in 1:nrep){

    #stream
    my_streams[[i]] = filterStream(
      file.name = '',
      track = track,
      oauth = oauth[i %% n_auth + 1],
      timeout = time)

    # sleep for a while
    Sys.sleep(sleep)
    }

  # combine streams
  my_stream = do.call(c,my_streams)

  # throw error if nothing was found
  if(length(my_stream) == 0) stop('Nothing found')

  # parse
  parsed_stream = lapply(my_stream,function(x) {

    # parse JSON
    tweet = fromJSON(x)

    # test if tweet complete
    if('user' %in% names(tweet)){

      # extract user
      user = tweet[['user']]
```

```
    user = user[c('screen_name',
                  'location',
                  'description',
                  'followers_count',
                  'friends_count',
                  'statuses_count')]

    # extract other meta information
    meta = tweet[c('created_at',
                   'text',
                   'source',
                   'lang')]

    result = c(unlist(user),unlist(meta))
    } else {

    # retrieve only meta information
    result = c(unlist(tweet[c('created_at','text','source','lang')]))
    }
  return(result)
  })

# extract variable names
variable_names = unique(unlist(sapply(parsed_stream,names)))

# create named data frame
tmp_matrix = matrix(NA,ncol = length(variable_names), nrow=length(parsed_stream))
data_stream = data.frame(tmp_matrix)
names(data_stream) = variable_names

# fill data frame
for(i in 1:length(parsed_stream)){
  tweet = parsed_stream[[i]]
  data_stream[i,names(tweet)] = tweet
  }

# return data_stream
return(data_stream)
}
```

3 Rerun Assignment 3

3.1 Identify useful track term

Play around with your `my_streamer` function to identify a track term that produces (a) many tweets an (b) a large number of Emojis.

3.2 Run `my_streamer`

Run your `my_streamer` for at least 4 hours (sky is the limit) and save your results for later use.

3.3 Post Emoji-Zipfian

Redo the Emoji-Zipfian figure using (only) the new results and post the figure on Twitter.

```
library(png)
library(yarrr)
library(stringi)

# only needed for my code
library(Rcpp)
sourceCpp('~/Dropbox (2.0)/Work/Software/f.utils/src/readbig.cpp')
```

```
# ------ Get tweet text [this part should be different for you]

# read tweets
tweets = readbig('/Users/wulff/Dropbox (2.0)/Work/Teaching/2017 Summer/Naturallanguage/nlpSeminar/RiskS
# readbig is my own function - I need it because i used a larger
# database for my analysis containing over 2M tweets.


# collapse tweets
text = tweets[1:100000]
text = paste(text,collapse = ' ')

# ------ Count emojis

# extract emojis
emoji_ids = readRDS('advancedEmojiList.RDS')[-2283,]
# for some reason Emoji 2283 doesnt work

# get code point string and utf8
unis = as.character(emoji_ids$code_point)
utf8 = as.character(emoji_ids$utf8)

# count emojis using stringi - easy and fast
cnts = stri_count_regex(text,utf8)


# ------ Preprocess counts

# combine counts and code point labels
emj_cnts = data.frame(unis, cnts)
names(emj_cnts) = c('code_point','conts')

# normalize
emj_cnts[,2] = emj_cnts[,2] / max(emj_cnts[,2])

# order
emj_cnts = emj_cnts[order(emj_cnts[,2],decreasing = T),]


# ------ Helpers

# zipfian
```

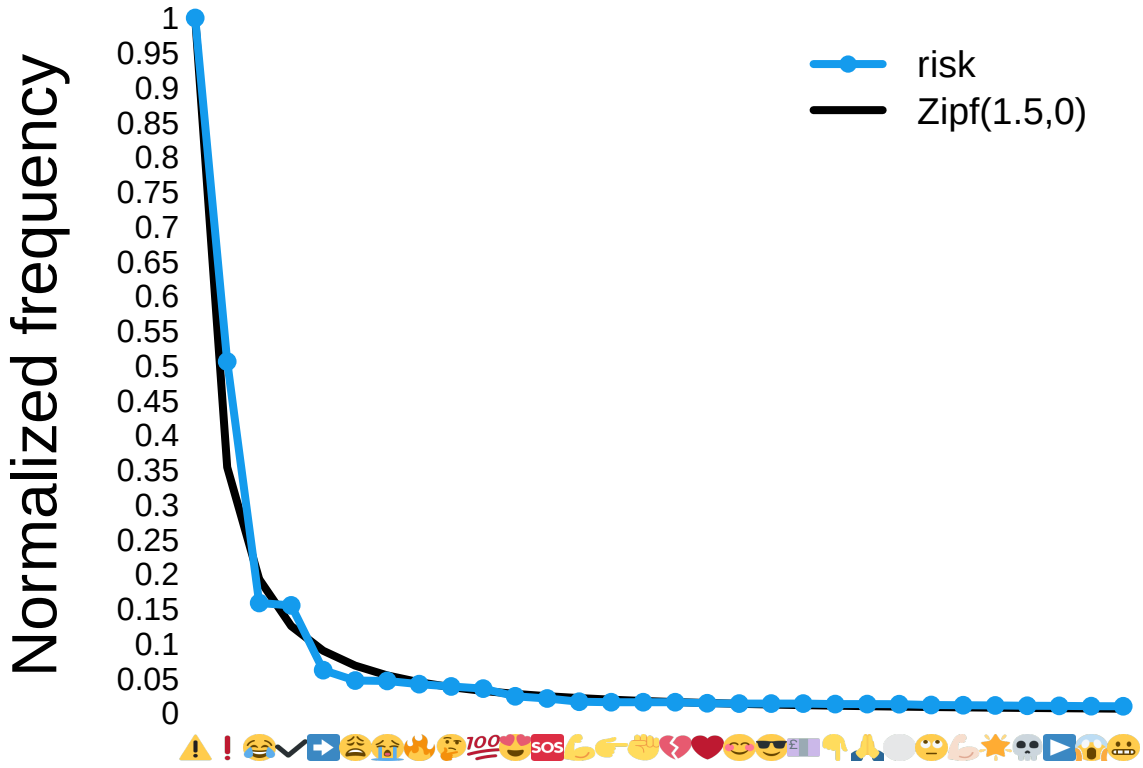
```r
zipfian = function(rank, alpha, beta) 1 / ((rank + beta)**alpha)

# add emojis
add_emoji = function(filename, x, y, cex, match = F){
  pic  = readPNG(filename)
  dims = dim(pic)[1:2]
  usr = par()$usr
  if(match == T) ar = diff(usr[3:4]) / diff(usr[1:2]) else ar = 1
  rasterImage(pic, x-cex/2, y-(ar*cex/2), x+cex/2, y+(ar*cex/2), interpolate=TRUE)
  }

# ------ Plot

#png('EmojiZipfian2.png',width = 720, height = 500)

xlim = c(.5,30.5)
ylim = c(-.1,1)
cols = piratepal("basel")[6]
pos = c(-.05)
pch = c(16)
plot.new()
par(mar = c(0,4.5,1,1))
plot.window(xlim = xlim, ylim = ylim)
mtext(seq(0,1,.05),at=seq(0,1,.05),las=1,side=2, line=-1)
mtext('Normalized frequency',side=2, line = 2, cex = 2, at = .5)
legend(20,1, legend = c('risk', 'Zipf(1.5,0)'), col = c(cols,'black'), lwd = 4, lty = 1, cex = 1.2, bty
lines(zipfian(1:30, 1.5, 0), lwd = 4, lty = 1, col = 'black')
tmp = emj_cnts[1:30,]
lines(tmp[,2],lty = 1, lwd = 4, col = cols)
points(tmp[,2],lty = 1, pch = pch, col = cols, cex = 1.3)
for(j in 1:30){
  path = paste0('emoji_imgs/',tmp[j,1],'.png')
  if(file.exists(path)){
    add_emoji(path,j,pos,1,match=T)
  } else {
    add_emoji('NAicon.png',j,pos,1,match=T)
  }
}
```


Normalized frequency
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
risk
Zipf(1.5,0)

```
#dev.off()
```