* [Thoughts](https://jtimm.net/2021/01/20/historic-memory/#thoughts)
* [Timeline content](https://jtimm.net/2021/01/20/historic-memory/#timeline-content)
* [Automated tweet-sizing of a thought](https://jtimm.net/2021/01/20/historic-memory/#automated-tweet-sizing-of-a-thought)
* [Hashtags sans supervision](https://jtimm.net/2021/01/20/historic-memory/#hashtags-sans-supervision)
* [Posting thread using rtweet](https://jtimm.net/2021/01/20/historic-memory/#posting-thread-using-rtweet)
* [Summary](https://jtimm.net/2021/01/20/historic-memory/#summary)

**Thoughts**

happened four years ago in the Trump presidency.

A fairly trivial task; what is a bit tricky, however, is (1) dealing with Twitter’s 280 character limit, and (2) automating the posting of **Twitter threads**, as Wikipedia’s daily accounting of the last four years can be fairly detailed. Via the rtweet package. For good measure, we demonstrate an unsupervised approach to adding hashtags to our tweet-threads in the form of named entities via spacy / spacyr.

**Timeline content**

The Trump timeline can be extracted using the uspols::uspols\_wiki\_timeline() function. A static table will shortly suffice.

# devtools::install\_github("jaytimm/uspols")

library(tidyverse)

locals1 <- uspols::uspols\_wiki\_timeline()

## feature-izing Events --

locals1$nsent <- tokenizers::count\_sentences(locals1$Events)

locals1$nchar1 <- nchar(locals1$Events)

## --

locals1 <- subset(locals1, nchar1 > 0)

Details of table content returned from uspols\_wiki\_timeline(): we consider the 699th day of the Trump presidency. A Thursday.

eg <- locals1 %>%filter(daypres == 699)

eg %>% select(quarter:dow) %>% slice(1) %>% knitr::kable()

| **quarter** | **weekof** | **daypres** | **date** | **dow** |
| --- | --- | --- | --- | --- |
| 2018\_Q4 | 2018-12-16 | 699 | 2018-12-20 | Thursday |

As detailed below, Wikipedia summarizes the day’s happenings as individual “Events.” Some days were more eventful than others during the past four years. I have added a bullet column for simple enumeration; day 699, then, included four events. Per above, we have added some Event-level features relevant to a tweet-bot: sentence count and character count.

eg %>% select(bullet, nsent, nchar1, Events) %>%

DT::datatable(rownames = FALSE,

options = list(dom = 't',

pageLength = nrow(eg),

scrollX = TRUE))

In this example, then, Events #3 and #4 will present problems from a character-limit perspective.

**Automated tweet-sizing of a thought**

**Sentence-based**

One approach to “tweet-sizing” Wikipedia-based events is simply to extract sentences up until we exceed some character threshold. An easy solution. Below we eliminate stops as markers of abbreviation (eg, honorifics Ms. or Dr.) – which makes sentence tokenizers infinitely more useful.

locals1$Events <- gsub('([A-Z])(\\.)', '\\1', locals1$Events)

The function below extracts sentences from a larger text until the cumulative character count exceeds some number, as specified by the chars parameter. As we want to add some affixal matter to our tweet threads, we set this at 250.

extract\_sent1 <- function(x, chars = 250) {

z1 <- data.frame(ts = tokenizers::tokenize\_sentences(x)[[1]])

z1$nchar\_sent <- nchar(z1$ts) + 1

z1$cum\_char <- cumsum(z1$nchar\_sent)

z2 <- subset(z1, cum\_char < chars)

paste0(z2$ts, collapse = ' ')

}

The table below details the Events of Day 699 in tweet-ready length per our sentence extraction procedure. Per this method, we lose some (perhaps useful) detail from our thread.

Events <- unlist(lapply(eg$Events, extract\_sent1, chars = 250))

data.frame(nsent = tokenizers::count\_sentences(Events),

nchar1 = nchar(Events),

Events = Events) %>%

DT::datatable(rownames = FALSE,

options = list(dom = 't',

pageLength = nrow(eg),

scrollX = TRUE))

**Via ellipses**

If, instead, we wanted to preserve full event content, one approach would be to split text into ~280 character chunks (ideally respecting word boundaries), and piece thread together via ellipses.

cumsum\_reset <- function(x, thresh = 4) {

ans <- numeric()

i <- 0

while(length(x) > 0) {

cs\_over <- cumsum(x)

ntimes <- sum( cs\_over <= thresh )

x <- x[-(1:ntimes)]

ans <- c(ans, rep(i, ntimes))

i <- i + 1

}

return(ans)

}

The second function implements cumsum\_reset in the context of counting characters at the word level – once a certain character-count threshold is reached, counting resets.

to\_thread <- function(x, chars = 250){ # no thread counts at present --

x1 <- data.frame(text = unlist(strsplit(x, ' ')))

x1$chars <- nchar(x1$text) + 1

x1$cs1 <- cumsum(x1$chars)

x1$sub\_text <- cumsum\_reset(x1$chars, thresh = chars)

x2 <- aggregate(x1$text, list(x1$sub\_text), paste, collapse = " ")

x2$ww <- 'm'

x2$ww[1] <- 'f'

x2$ww[nrow(x2)] <- 'l'

x2$x <- ifelse(x2$ww %in% c('f', 'm'), paste0(x2$x, ' ...'), x2$x)

x2$x <- ifelse(x2$ww %in% c('l', 'm'), paste0('... ', x2$x), x2$x)

paste0(x2$x, collapse = ' || ')

}

For a demonstration of how these functions work, we use a super long event from the Wikipedia timeline – from 12 August 2018 (day 573) – which is over 1200 characters in length.

eg1 <- locals1 %>% filter(nchar1 == max(nchar1))

**Function output is summarized below**. Also, we add a thread counter, and check in on character counts.

eg2 <- eg1 %>%

select(Events) %>%

mutate(Events = to\_thread(Events)) %>% ##### --- !!

separate\_rows(Events, sep = ' \\|\\| ')

eg3 <- eg2 %>%

mutate(Events = paste0(Events, ' [',

row\_number(), ' / ',

nrow(eg2), ']'),

nchar1 = nchar(Events)) %>%

select(nchar1, Events)

**Hashtags sans supervision**

A simple unsupervised approach to adding hashtags to tweets based on event content. Here, (1) extract named entities from tweet via the spacyr package, and (character-count depending) (2) add a randomly selected entity as hashtag to tweet.

ent1 <- spacyr::entity\_extract(spacyr::spacy\_parse(eg2$Events))

ent1 %>% slice(1:5) %>% knitr::kable()

| **doc\_id** | **sentence\_id** | **entity** | **entity\_type** |
| --- | --- | --- | --- |
| text1 | 1 | CIA | ORG |
| text1 | 1 | John\_Brennan | PERSON |
| text1 | 1 | Trump | ORG |
| text1 | 1 | New\_York\_Times | ORG |
| text1 | 1 | Trump | ORG |

ent2 <- ent1 %>%

select(doc\_id, entity) %>%

group\_by(doc\_id) %>%

distinct() %>%

mutate(n = n()) %>%

filter( !(entity == 'Trump' & n !=1) ) %>%

sample\_n(1) %>%

ungroup() %>%

mutate(doc\_id = as.integer(gsub('text', '', doc\_id)),

entity = paste0('#', gsub("\_|,|^the|'s", '', entity)))

eg3 <- eg3 %>%

mutate(doc\_id = row\_number()) %>%

left\_join(ent2, by = c('doc\_id')) %>%

mutate(txt3 = ifelse((nchar(Events) + nchar(entity)) > 280,

Events, paste0(Events, ' ', entity)),

nchar1 = nchar(txt3))

**Posting thread using rtweet**

The daily script (not detailed here) filters the Event timeline to the day’s date four years ago, and then applies our **tweet re-sizing functions** to address any potential character-count issues. The code below illustrates how threads are composed based on Event features (ie, sentence and character counts).

rowwise() %>%

mutate (THREAD = case\_when (nchar1 < 251 ~ Events,

nchar1 > 250 & nsent > 1 ~ extract\_sent1(Events),

nchar1 > 250 & nsent == 1 ~to\_thread(Events)

Lastly, we build threads by looping through our list of tweet-readied events, replying to each previously posted tweet via the in\_reply\_to\_status\_id parameter of the rtweet::post\_tweet() function.

rtweet::post\_tweet(tred2$txt3[1], token = tk)

if(nrow(tred1) > 1) {

for(i in 2:length(tred2$txt3)) {

Sys.sleep(1)

last\_tweet <- rtweet::get\_timeline(user = 'MemoryHistoric')$status\_id[1]

rtweet::post\_tweet(tred2$txt3[i],

in\_reply\_to\_status\_id = last\_tweet,

token = tk) }

} else {NULL}