# Text mining con R: ejemplo práctico Twitter

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Diciembre, 2017

## Tabla de contenidos

Versión PDF: Github

# Introducción

Twitter es actualmente una dinámica e ingente fuente de contenidos que, dada su popularidad e impacto, se ha convertido en la principal fuente de información para estudios de *Social Media Analytics*. Análisis de reputación de empresas, productos o personalidades, estudios de impacto relacionados con marketing, extracción de opiniones y predicción de tendencias son sólo algunos ejemplos de aplicaciones. Este tutorial pretende servir de **introducción** al análisis de texto y procesamiento de lenguaje natural con R. Para ello, se analizan las publicaciones que han hecho en Twitter diferentes personalidades. Los puntos tratados son:

- Automatización de la extracción de datos de Twitter.
- Análisis de las palabras empleadas por cada uno de los usuarios.
- Experimentos básicos de clasificación y predicción de la autoría.
- Análisis de sentimientos

Si bien Python es el lenguaje de programación que domina en este ámbito, existen nuevas librerías en R que facilitan y extienden sus capacidades como herramienta de análisis de texto. Las que se emplean en este tutorial son:

- `stringr`: paquete desarrollado por *Hadley Wickham* con multitud de funciones (división, búsqueda, reemplazo...) para trabajar con *strings*.
- `tidytext`: paquete desarrollado por *Julia Silge* y *David Robinson*. Los autores proponen una forma de trabajar con texto que sigue los principios de *tidy data*, lo que hace muy sencillo combinarlo con otros paquetes tales como `dplyr`, `broom`, `tidyr` y `ggplot2`.
- `quanteda`: paquete con multitud de funciones orientadas a *text mining*, algunas de ellas permiten crear *Term-Document Matrices*
- `purrr`: permite aplicar funciones a elementos de un vector o lista, por ejemplo, a los elementos de una columna de un *dataframe*.

# Extracción datos Twitter

Como ocurre en muchas redes sociales, la propia plataforma pone a disposición de los usuarios una API que permite extraer información. Aunque en la mayoría de casos se trata de *web services API*, con frecuencia existen librerías que permiten interactuar con la API desde diversos lenguajes de programación. En este caso, se emplea la librería `rtweet`, un *wrapper* R que se comunica con la API de Twitter.

Para poder extraer datos de Twitter es necesario estar registrado en la plataforma y, a partir de la cuenta, crear una *Twitter App* asociada. *Twitter App* es el mecanismo que proporciona Twitter para desarrolladores que quieran acceder a los contenidos de Twitter a través de programas. Al crear una *Twitter App*, Twitter proporciona una serie de claves y tokens de identificación que permiten acceder a la aplicación y extraer información. Se puede encontrar información detallada de cómo crear una APP en https://apps.twitter.com/app/new y de cómo acceder a ella a través de R en rtweet vignettes. Twitter tiene una normativa que regula la frecuencia máxima de peticiones, así como la cantidad máxima de tweets que se pueden extraer rate limiting. Es importante cumplir estos límites para evitar ser bloqueado.

A lo largo de este tutorial se analizan los tweets de:

- Elon Musk (@elonmusk) y Bill Gates (@BillGates), dos directivos de empresas tecnológicas.
- Mayor Ed Lee (@mayoredlee) alcalde de la ciudad de San Francisco.

Para el trabajo de análisis que se quiere realizar, es conveniente recuperar tantos tweets como sea posible, o al menos unos 1000. Como el número máximo de tweets recuperados por consulta es de 200 y existe una limitación de tiempo mínimo entre consulta y consulta, se sigue la siguiente estrategia:

1. Todo tweet tiene un ID global numérico que sigue un orden temporal, lo que permite identificar si un tweet es más reciente que otro.
2. Entre los argumentos de `api.GetUserTimeline()` se puede especificar el *`max_id`* para recuperar únicamente tweets más antiguos.
3. Antes de cada consulta, se lee el fichero donde se están almacenando los tweets y se identifica el ID del último tweet recuperado. Si no existe fichero de almacenamiento para el usuario en cuestión, se crea uno.
4. Se realiza una nueva consulta empleando como argumento *`max_id`* el ID recuperado en el paso anterior.
5. Se incorporar los nuevos datos al archivo de almacenamiento.

```r
library(rtweet)
library(tidyverse)
library(knitr)

# Identificación y obtención de tokens
appname <- "extrac-------eets"
key     <- "WXkVypH-----------------mY0"
secret  <- "EeMD2-------qYbYw5EFTKR7i3M"
twitter_token <- create_token(app = appname, consumer_key = key,
                              consumer_secret = secret)

extraccion_tweets <- function(usuario, maxtweets = 100, output_file_name = NULL){
  # Esta función extrae los tweets publicados por un usuario y los almacena en
  # un fichero csv. Si existe un fichero con el mismo nombre, lo lee, concatena
  # los nuevos tweets y lo sobrescribe.
  #
  # Argumentos:
  #   usuario: identificador del usuario de twitter
  #   maxtweets: número de tweets que se recuperan
  #   output_file_name: nombre del fichero de escritura

  # Si no se especifica el nombre del archivo de almacenamiento, se crea un
  # nombre por defecto
  if(is.null(output_file_name)){
        output_file_name <- paste0("datos_tweets_", usuario, ".csv")
  }

  # Si no existe el fichero de almacenamiento, se crea uno nuevo con los
  # resultados de la primera recuperación
  if(!(output_file_name %in% list.files())){
    datos_new <- get_timeline(user = usuario, n = maxtweets, parse = TRUE,
                              check = TRUE, include_rts = FALSE)
    write_csv(x = datos_new, path = output_file_name, col_names = TRUE)
    print("Nuevo fichero creado")
  }else{
    # Se leen los datos antiguos
    datos_old <- read_csv(file = output_file_name)
    # Se identifica el último Id recuperado
    ultimo_id <- tail(datos_old, 1)["status_id"] %>% pull()
    # Para no recuperar de nuevo el último tweet de la consulta anterior
    # se incrementa en 1 el Id
    ultimo_id = ultimo_id + 1
    # Para que no haya errores de compatibilidad, se convierten todas las
    # columnas numéricas a character
    datos_old <- map_if(.x = datos_old, .p = is.numeric, .f = as.character)
    # Extracción de nuevos tweets
    datos_new <- get_timeline(user = usuario, n = maxtweets, max_id = ultimo_id,
                              parse = TRUE, check = TRUE, include_rts = FALSE)
    datos_new <- map_if(.x = datos_new, .p = is.numeric, .f = as.character)
```

```
    # Concatenación de los datos nuevos, viejos y escritura en disco
    datos_concatenados <- bind_rows(datos_old, datos_new)
    write_csv(x = datos_concatenados, path = output_file_name, col_names = TRUE)
    print(paste("Número total de tweets:", nrow(datos_concatenados)))
    print(paste("Número de tweets nuevos:", nrow(datos_new)))
  }
}
```

Cada vez que se ejecuta la función `extraccion_tweets()` se recuperan nuevos tweets y se almacenan junto a los extraídos previamente.

```
extraccion_tweets(usuario  = "@elonmusk", maxtweets  = 200)
extraccion_tweets(usuario  = "@BillGates", maxtweets  = 200)
extraccion_tweets(usuario  = "@mayoredlee", maxtweets  = 200)
```

## Carga de datos

Los datos utilizados en este análisis se han obtenido mediante la función definida en el apartado anterior. Los ficheros *.csv* pueden encontrarse en mi repositorio de github.

```
tweets_elon       <- read_csv(file = "./datos/datos_tweets_@elonmusk.csv",
                              col_names = TRUE)
tweets_BillGates  <- read_csv(file = "./datos/datos_tweets_@BillGates.csv",
                              col_names = TRUE)
tweets_mayoredlee <- read_csv(file = "./datos/datos_tweets_@mayoredlee.csv",
                              col_names = TRUE)
# Se unen todos los tweets en un único dataframe
tweets <- bind_rows(tweets_elon, tweets_BillGates, tweets_mayoredlee)

tweets %>% group_by(screen_name) %>% summarise(numero_tweets = n())
```

```
## # A tibble: 3 x 2
##   screen_name numero_tweets
##         <chr>         <int>
## 1   BillGates          2087
## 2    elonmusk          2678
## 3  mayoredlee          2447
```

El número de tweets recuperados es superior a 2000 para todos los usuarios.

De entre toda la información disponible, en este análisis únicamente se emplea: autor del tweet, fecha de publicación, identificador del tweet y contenido.

```
Colnames(tweets)
```

```
##  [1] "screen_name"                  "user_id"
##  [3] "created_at"                   "status_id"
##  [5] "text"                         "retweet_count"
##  [7] "favorite_count"               "is_quote_status"
##  [9] "quote_status_id"              "is_retweet"
## [11] "retweet_status_id"            "in_reply_to_status_status_id"
## [13] "in_reply_to_status_user_id"   "in_reply_to_status_screen_name"
## [15] "lang"                         "source"
## [17] "media_id"                     "media_url"
## [19] "media_url_expanded"           "urls"
## [21] "urls_display"                 "urls_expanded"
## [23] "mentions_screen_name"         "mentions_user_id"
## [25] "symbols"                      "hashtags"
## [27] "coordinates"                  "place_id"
## [29] "place_type"                   "place_name"
## [31] "place_full_name"              "country_code"
## [33] "country"                      "bounding_box_coordinates"
## [35] "bounding_box_type"
```

```
# Selección de variables
tweets <- tweets %>% select(screen_name, created_at, status_id, text)

# Se renombran las variables con nombres más prácticos
tweets <- tweets %>% rename(autor = screen_name, fecha = created_at,
                            texto = text, tweet_id = status_id)
head(tweets)
```

```
## # A tibble: 6 x 4
##      autor               fecha     tweet_id
##      <chr>              <dttm>        <dbl>
## 1 elonmusk 2017-11-09 17:28:57 9.286758e+17
## 2 elonmusk 2017-11-09 17:12:46 9.286717e+17
## 3 elonmusk 2017-11-08 18:55:13 9.283351e+17
## 4 elonmusk 2017-11-07 19:48:45 9.279862e+17
## 5 elonmusk 2017-10-28 21:36:18 9.243894e+17
## 6 elonmusk 2017-10-28 21:30:41 9.243880e+17
## # ... with 1 more variables: texto <chr>
```

## Limpieza de texto y tokenización

El proceso de limpieza de texto, dentro del ámbito de *text mining*, consiste en eliminar del texto todo aquello que no aporte información sobre su temática, estructura o contenido. No existe una única forma de hacerlo, depende en gran medida de la finalidad del análisis y de la fuente de la que proceda el texto. Por ejemplo, en las redes sociales los usuarios pueden escribir de la forma que quieran, lo que suele resultar en un uso elevado de abreviaturas y signos de puntuación. En este ejercicio, dado que los principales objetivos son estudiar el perfil lingüístico de los tres usuarios, identificar la autoría de los tweets y analizar el sentimiento que transmiten, se procede a eliminar:

- Patrones no informativos (*urls* de páginas web)
- Signos de puntuación
- Etiquetas HTML
- Caracteres sueltos
- Números

Tokenizar un texto consiste en dividir el texto en las unidades que lo conforman, entendiendo por unidad el elemento más sencillo con significado propio para el análisis en cuestión, en este caso, las palabras.

Existen múltiples librerías que automatizan en gran medida la limpieza y tokenización de texto, por ejemplo, `tokenizers` o `quanteda`. Sin embargo, creo que se entiende mejor el proceso implemento una función propia que, si bien puede estar menos optimizada, es más transparente. Definir una función que contenga cada uno de los pasos de limpieza tiene la ventaja de poder adaptarse fácilmente dependiendo del tipo de texto analizado.

```
limpiar_tokenizar <- function(texto){
  # El orden de la limpieza no es arbitrario
    # Se convierte todo el texto a minúsculas
    nuevo_texto <- tolower(texto)
    # Eliminación de páginas web (palabras que empiezan por "http." seguidas
    # de cualquier cosa que no sea un espacio)
    nuevo_texto <- str_replace_all(nuevo_texto,"http\\S*", "")
    # Eliminación de signos de puntuación
    nuevo_texto <- str_replace_all(nuevo_texto,"[[:punct:]]", " ")
    # Eliminación de números
    nuevo_texto <- str_replace_all(nuevo_texto,"[[:digit:]]", " ")
    # Eliminación de espacios en blanco múltiples
    nuevo_texto <- str_replace_all(nuevo_texto,"[\\s]+", " ")
    # Tokenización por palabras individuales
```

```
    nuevo_texto <- str_split(nuevo_texto, " ")[[1]]
    # Eliminación de tokens con una longitud < 2
    nuevo_texto <- keep(.x = nuevo_texto, .p = function(x){str_length(x) > 1})
    return(nuevo_texto)
}

test = "Esto es 1 ejemplo de l'limpieza de6 TEXTO  https://t.co/rnHPgyhx4Z
@JoaquinAmatRodrigo #textmining"
limpiar_tokenizar(texto = test)
```

```
## [1] "esto"              "es"                "ejemplo"
## [4] "de"                "limpieza"          "de"
## [7] "texto"             "joaquinamatrodrigo" "textmining"
```

Puede observarse que la función `limpiar_tokenizar()` elimina el símbolo @ y # de las palabra a las que acompañan. En Twitter, los usuarios se identifican de esta forma, por lo que @ y # pertenecen al nombre. Aunque es importante tener en cuenta las eliminaciones del proceso de limpieza, el impacto en este caso no es demasiado alto, ya que, si un documento se caracteriza por tener la palabra *#datascience*, también será detectado fácilmente mediante la palabra *datascience*.

```
# Se aplica la función de limpieza y tokenización a cada tweet
tweets <- tweets %>% mutate(texto_tokenizado = map(.x = texto,
                                                   .f = limpiar_tokenizar))
tweets %>% select(texto_tokenizado) %>% head()
```

```
## # A tibble: 6 x 1
##   texto_tokenizado
##              <list>
## 1       <chr [13]>
## 2       <chr [15]>
## 3        <chr [2]>
## 4        <chr [6]>
## 5       <chr [18]>
## 6        <chr [9]>
```

Gracias a la característica de las *tibble* de poder contener cualquier tipo de elemento en sus columnas (siempre que sea el mismo para toda la columna), se puede almacenar el texto tokenizado. Cada elemento de la columna *texto_tokenizado* es una lista con un vector de tipo *character* que contiene los tokens generados.

```
tweets %>% slice(1) %>% select(texto_tokenizado) %>% pull()
```

```
## [[1]]
##  [1] "if"      "one"     "day"     "my"      "words"   "are"     "against"
##  [8] "science" "choose"  "science" "mustafa" "kemal"   "atatürk"
```

## Análisis exploratorio

En R, las estructuras por excelencia para el análisis exploratorio son el *DataFrame* y la *Tibble*, que es la forma en la que se encuentra almacenada ahora la información de los tweets. Sin embargo, al realizar la tokenización, ha habido un cambio importante. Previamente a la división del texto, los elementos de estudio (observaciones) eran los tweets, y cada uno se encontraba en una fila, cumplimento así la condición de *tidy data: una observación, una fila*. Al realizar la tokenización, el elemento de estudio ha pasado a ser cada token (palabra), incumpliendo así la condición de *tidy data*. Para volver de nuevo a la estructura ideal se tiene que expandir cada lista de tokens, duplicando el valor de las otras columnas tantas veces como sea necesario. Ha este proceso se le conoce como expansión o *unnest*.

```
tweets_tidy <- tweets %>% select(-texto) %>% unnest()
tweets_tidy <- tweets_tidy %>% rename(token = texto_tokenizado)
head(tweets_tidy)
```

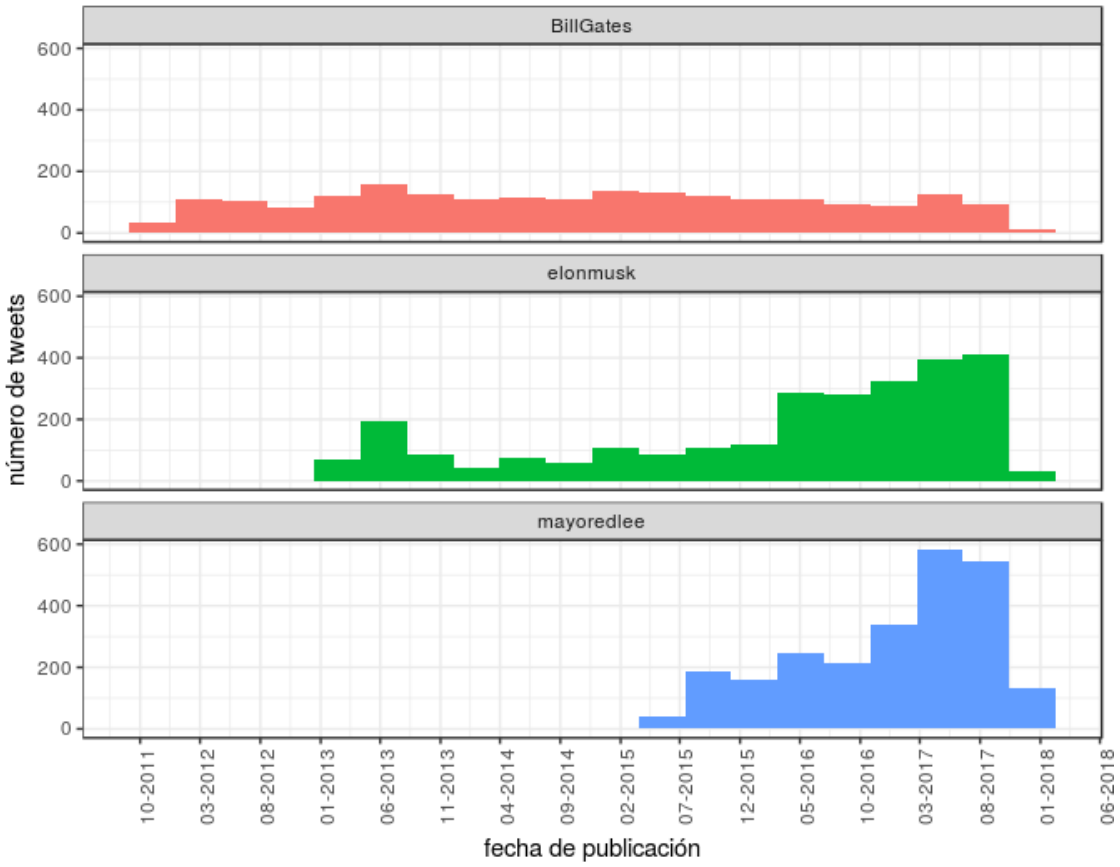
```
## # A tibble: 6 x 4
##      autor               fecha     tweet_id token
##      <chr>              <dttm>        <dbl> <chr>
## 1 elonmusk 2017-11-09 17:28:57 9.286758e+17    if
## 2 elonmusk 2017-11-09 17:28:57 9.286758e+17   one
## 3 elonmusk 2017-11-09 17:28:57 9.286758e+17   day
## 4 elonmusk 2017-11-09 17:28:57 9.286758e+17    my
## 5 elonmusk 2017-11-09 17:28:57 9.286758e+17 words
## 6 elonmusk 2017-11-09 17:28:57 9.286758e+17   are
```

Ahora que los la información está en formato *tidy*, se pueden realizar filtrados, sumatorios y representaciones con gran facilidad. La función `unnest_tokens()` del paquete `tidytext` permite, entre otras cosas, automatizar el proceso tokenización y almacenamiento en formato *tidy* en un único paso.

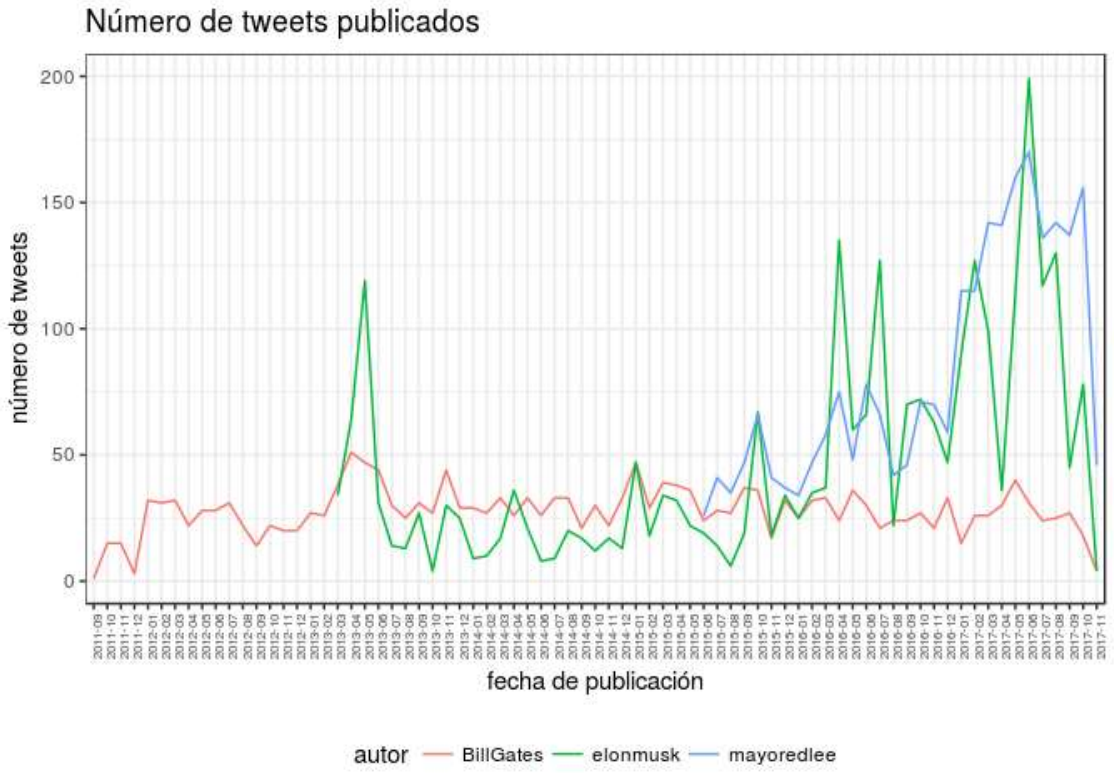
## Distribución temporal de los tweets

Dado que cada usuario puede haber iniciado su actividad en Twitter en diferente momento, es interesante explorar si los tweets recuperados solapan en el tiempo.

```
library(lubridate)

ggplot(tweets, aes(x = as.Date(fecha), fill = autor)) +
  geom_histogram(position = "identity", bins = 20, show.legend = FALSE) +
  scale_x_date(date_labels = "%m-%Y", date_breaks = "5 month") +
  labs(x = "fecha de publicación", y = "número de tweets") +
  facet_wrap(~ autor, ncol = 1) +
  theme_bw()+
  theme(axis.text.x = element_text(angle = 90))
```



```
tweets_mes_anyo <- tweets %>% mutate(mes_anyo = format(fecha, "%Y-%m"))
tweets_mes_anyo %>% group_by(autor, mes_anyo) %>% summarise(n = n()) %>%
  ggplot(aes(x = mes_anyo, y = n, color = autor)) +
  geom_line(aes(group = autor)) +
  labs(title = "Número de tweets publicados", x = "fecha de publicación",
       y = "número de tweets") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, size = 6),
        legend.position = "bottom")
```




Puede observarse un perfil de actividad distinto para cada usuario. Bill Gates ha mantenido una actividad constante de en torno a 30 tweets por mes durante todo el periodo estudiado. Elon Musk muestra una actividad inicial por debajo de la de Bill Gates pero, a partir de febrero de 2016 incrementó sustancialmente el número de tweets publicados. Ed Lee tiene una actividad muy alta sobre todo en el periodo 2017. Debido a las limitaciones que impone Twitter en las recuperaciones, cuanto más activo es un usuario, menor es el intervalo de tiempo para el que se recuperan tweets. En el caso de Ed Lee, dado que publica con mucha más frecuencia que el resto, con la misma cantidad de tweets recuperados se abarca menos de la mitad del rango temporal que con los otros.

## Frecuencia de palabras

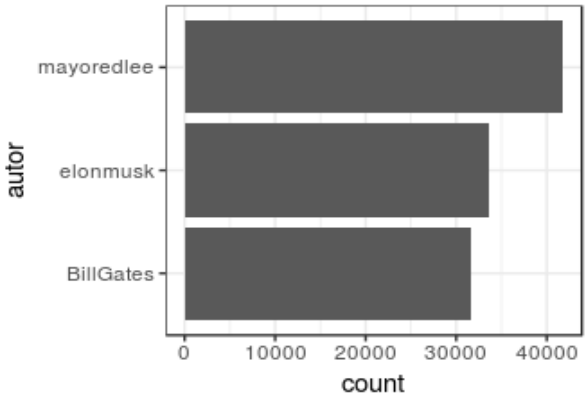
A la hora de entender que caracteriza la escritura de cada usuario, es interesante estudiar qué palabras emplea, con qué frecuencia, así como el significado de las mismas.

### Total de palabras utilizadas por cada usuario

```
tweets_tidy %>% group_by(autor) %>% summarise(n = n())
```

```
## # A tibble: 3 x 2
##        autor     n
##        <chr> <int>
## 1  BillGates 31572
## 2   elonmusk 33584
## 3 mayoredlee 41787
```

```
tweets_tidy %>%  ggplot(aes(x = autor)) + geom_bar() + coord_flip() + theme_bw()
```

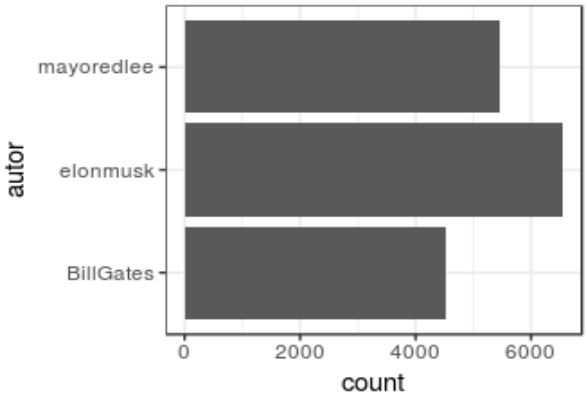



### Palabras distintas utilizadas por cada usuario

```
tweets_tidy %>% select(autor, token) %>% distinct() %>%  group_by(autor) %>%
            summarise(palabras_distintas = n())
```

```
## # A tibble: 3 x 2
##        autor palabras_distintas
##        <chr>              <int>
## 1  BillGates               4510
## 2   elonmusk               6552
## 3 mayoredlee               5471
```

```
tweets_tidy %>% select(autor, token) %>% distinct() %>%
                ggplot(aes(x = autor)) + geom_bar() + coord_flip() + theme_bw()
```



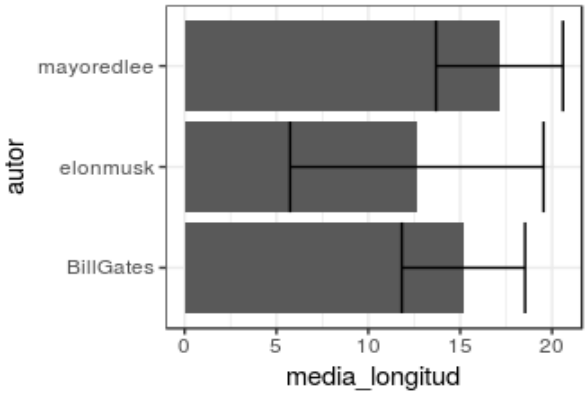

Aunque Elon Musk no es el que más palabras totales ha utilizado, bien porque ha publicado menos tweets o porque estos son más cortos, es el que más palabras distintas emplea.

## Longitud media de los tweets por usuario

```
tweets_tidy %>% group_by(autor, tweet_id) %>% summarise(longitud = n()) %>%
group_by(autor) %>% summarise(media_longitud = mean(longitud),
                              sd_longitud = sd(longitud))
```

```
## # A tibble: 3 x 3
##        autor media_longitud sd_longitud
##        <chr>          <dbl>       <dbl>
## 1  BillGates       15.17885    3.352719
## 2   elonmusk       12.63982    6.900113
## 3 mayoredlee       17.13987    3.453383
```

```
tweets_tidy %>% group_by(autor, tweet_id) %>% summarise(longitud = n()) %>%
group_by(autor) %>%
                summarise(media_longitud = mean(longitud),
                          sd_longitud = sd(longitud)) %>%
                ggplot(aes(x = autor, y = media_longitud)) +
                geom_col() +
                geom_errorbar(aes(ymin = media_longitud - sd_longitud,
                                  ymax = media_longitud + sd_longitud)) +
                coord_flip() + theme_bw()
```




El tipo de tweet de Bill Gates y Mayor Ed Lee es similar en cuanto a longitud media y desviación. Elon Musk alterna más entre tweets cortos y largos, siendo su media inferior a la de los otros dos.

## Palabras más utilizadas por usuario

```
tweets_tidy %>% group_by(autor, token) %>% count(token) %>% group_by(autor) %>%
            top_n(10, n) %>% arrange(autor, desc(n)) %>% print(n=30)
```

```
## # A tibble: 30 x 3
## # Groups:   autor [3]
##         autor token     n
##         <chr> <chr> <int>
##  1  BillGates   the  1195
##  2  BillGates    to  1117
##  3  BillGates    of   670
##  4  BillGates    in   591
##  5  BillGates    is   453
##  6  BillGates   and   439
##  7  BillGates   for   363
##  8  BillGates  this   345
##  9  BillGates    we   334
## 10  BillGates    on   333
## 11   elonmusk   the   988
## 12   elonmusk    to   916
## 13   elonmusk    of   638
## 14   elonmusk    is   543
## 15   elonmusk    in   478
## 16   elonmusk   for   400
## 17   elonmusk   and   367
## 18   elonmusk    it   351
## 19   elonmusk    on   344
## 20   elonmusk  that   315
```

```
## 21 mayoredlee    to  1693
## 22 mayoredlee   the  1355
## 23 mayoredlee   amp  1212
## 24 mayoredlee   our  1104
## 25 mayoredlee    sf   944
## 26 mayoredlee   for   822
## 27 mayoredlee    of   819
## 28 mayoredlee    we   782
## 29 mayoredlee    in   780
## 30 mayoredlee    is   452
```

## Stop words

En la tabla anterior puede observarse que los términos más frecuentes en todos los usuarios se corresponden con artículos, preposiciones, pronombres..., en general, palabras que no aportan información relevante sobre el texto. Ha estas palabras se les conoce como *stopwords*. Para cada idioma existen distintos listados de *stopwords*, además, dependiendo del contexto, puede ser necesario adaptar el listado. En la tabla anterior aparece el término *amp* que procede de la etiqueta html *&amp*. Con frecuencia, a medida que se realiza un análisis se encuentran palabras que deben incluirse en el listado de *stopwords*.

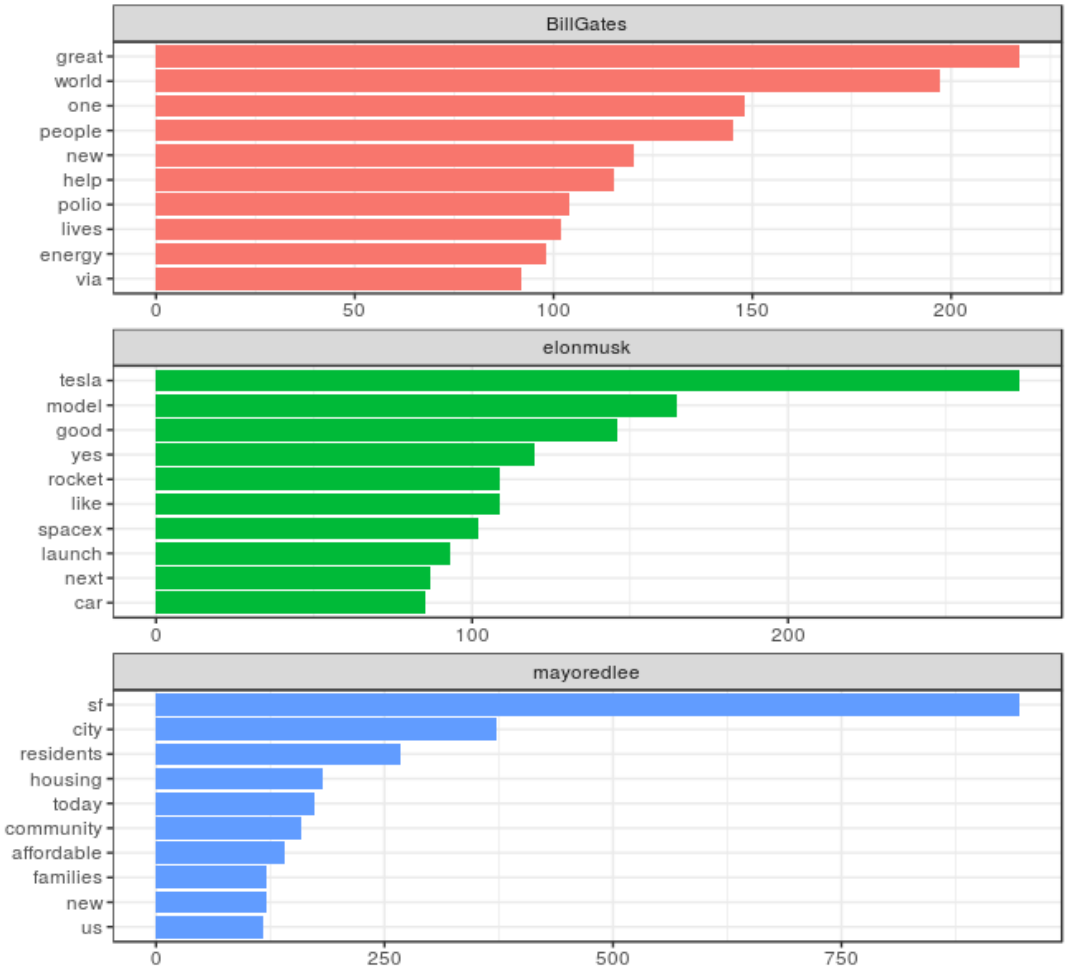
Dentro del paquete `tidytext` y `tokenizers` pueden encontrarse varios listados de *stopwords* para los idiomas "en", "da", "de", "el", "es", "fr", "it", "ru". En este caso, se emplea un listado de *stopwords* obtenido de la librería de Python `nltk.corpus`.

```
lista_stopwords <- c('me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
                'you','your', 'yours', 'yourself', 'yourselves', 'he', 'him','his',
                'himself', 'she', 'her', 'hers', 'herself', 'it', 'its', 'itself',
                'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which',
                'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',
                'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
                'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and',
                'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at',
                'by', 'for', 'with', 'about', 'against', 'between', 'into',
                'through', 'during', 'before', 'after', 'above', 'below', 'to',
                'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
                'again', 'further', 'then', 'once', 'here', 'there', 'when',
                'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
                'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own',
                'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
                'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've',
                'y', 'ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn',
                'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan',
                'shouldn', 'wasn', 'weren', 'won', 'wouldn','i')
```

```
# Se añade el término amp al listado de stopwords
lista_stopwords <- c(lista_stopwords, "amp")
# Se filtran las stopwords
tweets_tidy <- tweets_tidy %>% filter(!(token %in% lista_stopwords))
```

## Representación gráfica de las frecuencias

```
tweets_tidy %>% group_by(autor, token) %>% count(token) %>% group_by(autor) %>%
                top_n(10, n) %>% arrange(autor, desc(n)) %>%
                ggplot(aes(x = reorder(token,n), y = n, fill = autor)) +
                geom_col() +
                theme_bw() +
                labs(y = "", x = "") +
                theme(legend.position = "none") +
                coord_flip() +
                facet_wrap(~autor,scales = "free", ncol = 1, drop = TRUE)
```



Los resultados obtenidos tienen sentido si ponemos en contexto la actividad profesional de los usuarios analizados. Mayor Ed Lee es alcalde de San Francisco (*sf*), por lo que sus tweets están relacionados con la ciudad, residentes, familias, casas... Elon Musk dirige varias empresas tecnológicas entre las que destacan *Tesla* y *SpaceX*, dedicadas a los coches y a la aeronáutica. Por último, Bill Gates, además de propietario de *microsoft*, dedica parte de su capital a fundaciones de ayuda, de ahí las palabras mundo, polio, ayuda...

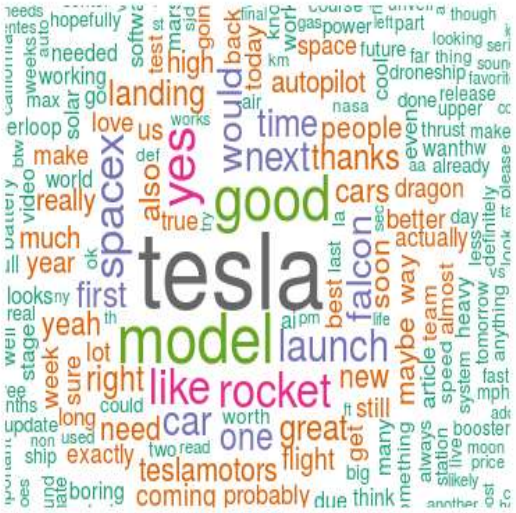
## Word Clouds

Otra forma visual de representar las palabras más frecuentes es mediante nubes de palabras (*word clouds*). En esta representación, las palabras más importantes tienen mayor tamaño.

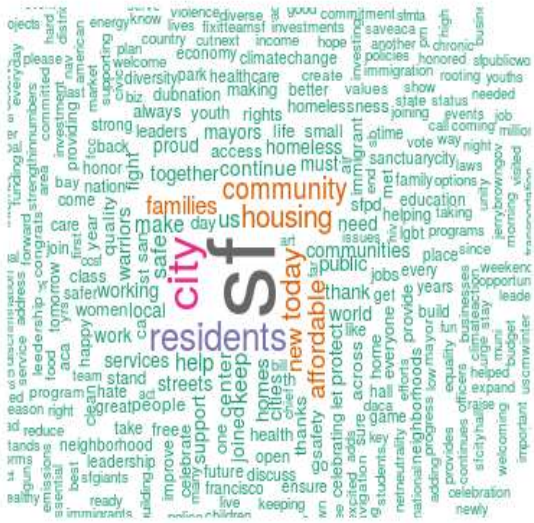
```
library(wordcloud)
library(RColorBrewer)

wordcloud_custom <- function(grupo, df){
  print(grupo)
  wordcloud(words = df$token, freq = df$frecuencia,
            max.words = 400, random.order = FALSE, rot.per = 0.35,
            colors = brewer.pal(8, "Dark2"))
}
df_grouped <- tweets_tidy %>% group_by(autor, token) %>% count(token) %>%
              group_by(autor) %>% mutate(frecuencia = n / n()) %>%
              arrange(autor, desc(frecuencia)) %>% nest()
walk2(.x = df_grouped$autor, .y = df_grouped$data, .f = wordcloud_custom)
```

```
## [1] "BillGates"
```

```
## [1] "elonmusk"
```




```
## [1] "mayoredlee"
```



## Correlación entre usuarios por palabras utilizadas

Una forma de cuantificar la similitud entre los perfiles de dos usuarios de Twitter es calculando la correlación en el uso de palabras. La idea es que, si dos usuarios escriben de forma similar, tenderán a utilizar las mismas palabras y con frecuencias similares.

Para poder generar gráficos de correlaciones se necesita disponer de cada variable en una columna. En este caso, las variables a correlacionar son los usuarios.

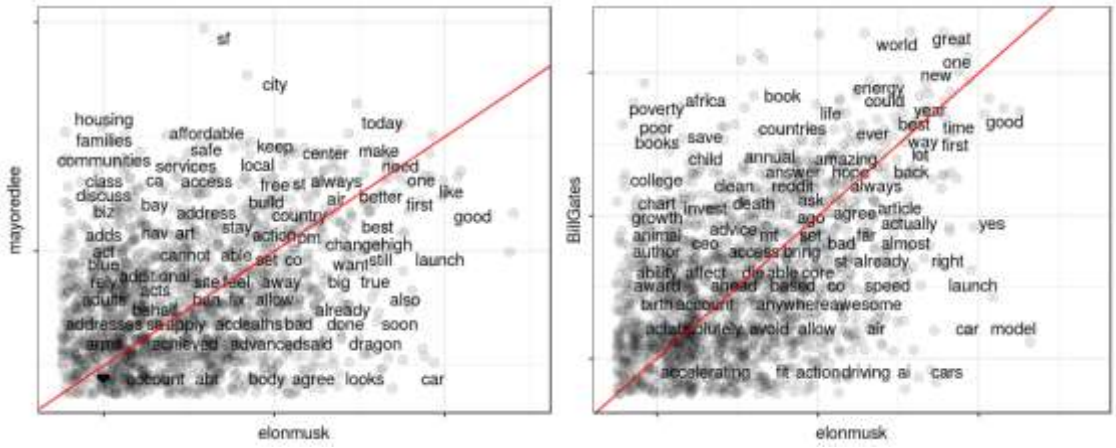
```
library(gridExtra)
library(scales)

tweets_spread <- tweets_tidy %>% group_by(autor, token) %>% count(token) %>%
                 spread(key = autor, value = n, fill = NA, drop = TRUE)

cor.test(~ mayoredlee + elonmusk, method = "pearson", data = tweets_spread)
```

```
##
##  Pearson's product-moment correlation
##
## data:  mayoredlee and elonmusk
## t = 6.4026, df = 1748, p-value = 1.958e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.1052625 0.1968356
## sample estimates:
##       cor
## 0.1513738
```

```
cor.test(~ BillGates + elonmusk, data = tweets_spread)
```

```
##
##  Pearson's product-moment correlation
##
## data:  BillGates and elonmusk
## t = 20.577, df = 1765, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4014594 0.4767108
## sample estimates:
##       cor
## 0.4398568
```

```
p1 <- ggplot(tweets_spread, aes(elonmusk, mayoredlee)) +
      geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
      geom_text(aes(label = token), check_overlap = TRUE, vjust = 1.5) +
      scale_x_log10(labels = percent_format()) +
      scale_y_log10(labels = percent_format()) +
      geom_abline(color = "red") +
      theme_bw() +
      theme(axis.text.x = element_blank(),
            axis.text.y = element_blank())

p2 <- ggplot(tweets_spread, aes(elonmusk, BillGates)) +
      geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
      geom_text(aes(label = token), check_overlap = TRUE, vjust = 1.5) +
      scale_x_log10(labels = percent_format()) +
      scale_y_log10(labels = percent_format()) +
      geom_abline(color = "red") +
      theme_bw() +
      theme(axis.text.x = element_blank(),
            axis.text.y = element_blank())

grid.arrange(p1, p2, nrow = 1)
```




Para poder valorar adecuadamente el nivel de correlación es interesante conocer el número de palabras comunes entre cada par de autores.

```
palabras_comunes <- dplyr::intersect(tweets_tidy %>% filter(autor=="elonmusk") %>%
                    select(token), tweets_tidy %>% filter(autor=="mayoredlee") %>%
                    select(token)) %>% nrow()
paste("Número de palabras comunes entre Elon Musk y Ed Lee", palabras_comunes)
```

```
## [1] "Número de palabras comunes entre Elon Musk y Ed Lee 1750"
```

```
palabras_comunes <- dplyr::intersect(tweets_tidy %>% filter(autor=="elonmusk") %>%
                    select(token), tweets_tidy %>% filter(autor=="BillGates") %>%
                    select(token)) %>% nrow()
paste("Número de palabras comunes entre Elon Musk y Bill Gates", palabras_comunes)
```

```
## [1] "Número de palabras comunes entre Elon Musk y Bill Gates 1767"
```

Aunque el número de palabras comunes entre Elon Musk y Bill Gates y entre Elon Musk y Ed Lee es similar, la correlación basada en su uso es mayor entre Elon Musk y Bill Gates. Esto tiene sentido si se contempla el hecho de que ambos trabajan como directivos de empresas tecnológicas.

## Comparación en el uso de palabras

A continuación, se estudia que palabras se utilizan de forma más diferenciada por cada usuario, es decir, palabras que utiliza mucho un autor y que no utiliza el otro. Una forma de hacer este análisis es mediante el log of odds ratio de las frecuencias. Esta comparación se hace por pares, en este caso se comparan Elon Musk y Mayor Ed Lee.

$$\text{log of odds ratio} = \log\left(\frac{\left[\frac{n_k+1}{\text{N}+1}\right]_{\text{Elon}}}{\left[\frac{n_k+1}{\text{N}+1}\right]_{\text{Edd}}}\right)$$

siendo $n_k$ el número de veces que aparece el término $k$ en los textos de cada autor y $N$ el número total de términos de cada autor.

Para realizar este cálculo es necesario que, para todos los usuarios, se cuantifique la frecuencia de cada una de las palabras que aparecen en el conjunto de tweets, es decir, si un autor no ha utilizado una de las palabras que sí ha utilizado otro, debe aparecer esa palabra en su registro con frecuencia igual a cero. Existen varias formas de conseguir esto, una de ellas es pivotar y despivotar el *dataframe* sustituyendo los *missing values* por cero.

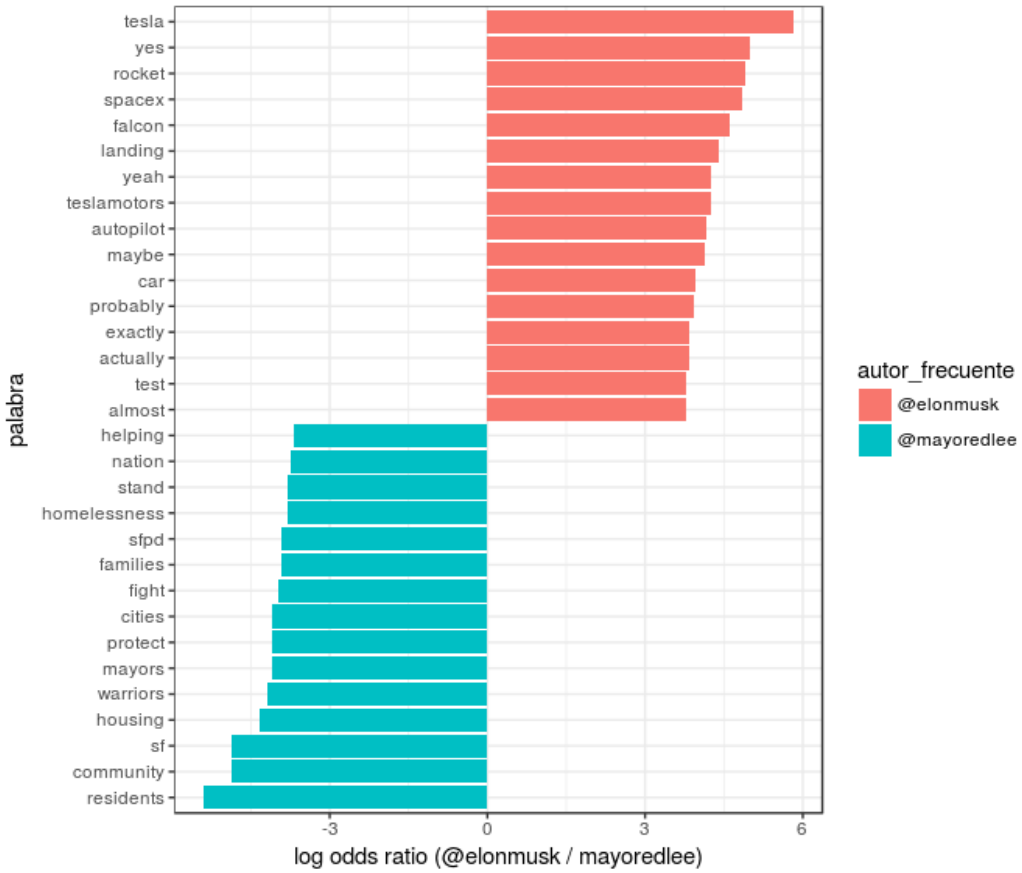
```
# Pivotaje y despivotaje
tweets_spread <- tweets_tidy %>% group_by(autor, token) %>% count(token) %>%
                 spread(key = autor, value = n, fill = 0, drop = TRUE)
tweets_unpivot <- tweets_spread %>% gather(key = "autor", value = "n", -token)

# Selección de los autores elonmusk y mayoredlee
tweets_unpivot <- tweets_unpivot %>% filter(autor %in% c("elonmusk",
                                                         "mayoredlee"))
# Se añade el total de palabras de cada autor
```

```
tweets_unpivot <- tweets_unpivot %>% left_join(tweets_tidy %>%
                                                 group_by(autor) %>%
                                                 summarise(N = n()),
                                                 by = "autor")
# Cálculo de odds y log of odds de cada palabra
tweets_logOdds <- tweets_unpivot %>%  mutate(odds = (n + 1) / (N + 1))
tweets_logOdds <- tweets_logOdds %>% select(autor, token, odds) %>%
                  spread(key = autor, value = odds)
tweets_logOdds <- tweets_logOdds %>%  mutate(log_odds = log(elonmusk/mayoredlee),
                                             abs_log_odds = abs(log_odds))
# Si el logaritmo de odds es mayor que cero, significa que es una palabra con
# mayor probabilidad de ser de Elon Musk. Esto es así porque el ratio sea ha
# calculado como elonmusk/mayoredlee.
tweets_logOdds <- tweets_logOdds %>%
                  mutate(autor_frecuente = if_else(log_odds > 0,
                                                   "@elonmusk",
                                                   "@mayoredlee"))
tweets_logOdds %>% arrange(desc(abs_log_odds)) %>% head()
```

```
## # A tibble: 6 x 6
## # Groups:   token [6]
##       token     elonmusk   mayoredlee  log_odds abs_log_odds
##       <chr>        <dbl>        <dbl>     <dbl>        <dbl>
## 1     tesla 1.265589e-02 3.786301e-05  5.811903     5.811903
## 2 residents 4.618938e-05 1.018515e-02 -5.395936     5.395936
## 3       yes 5.588915e-03 3.786301e-05  4.994566     4.994566
## 4    rocket 5.080831e-03 3.786301e-05  4.899256     4.899256
## 5 community 4.618938e-05 6.058082e-03 -4.876399     4.876399
## 6        sf 2.771363e-04 3.578055e-02 -4.860650     4.860650
## # ... with 1 more variables: autor_frecuente <chr>
```

Representación de las 30 palabras más diferenciadas

```
tweets_logOdds %>% group_by(autor_frecuente) %>% top_n(15, abs_log_odds) %>%
ggplot(aes(x = reorder(token, log_odds), y = log_odds, fill = autor_frecuente)) +
geom_col() +
labs(x = "palabra", y = "log odds ratio (@elonmusk / mayoredlee)") +
coord_flip() +
theme_bw()
```




Estas palabras posiblemente tendrán mucho peso a la hora de clasificar los tweets.

## Relación entre palabras

En todos los análisis anteriores, se han considerado a las palabras como unidades individuales e independientes. Esto es una simplificación bastante grande, ya que en realidad el lenguaje se crea por combinaciones no aleatorias de palabras, es decir, determinadas palabras tienden a utilizarse de forma conjunta. A continuación se muestran algunas formas de calcular, identificar y visualizar relaciones entre palabras.

La función `unnest_tokens()` del paquete `tidytext` permite dividir el texto por *n-gramas*, siendo cada *n-grama* una secuencia de *n* palabras consecutivas. Para conseguir los *n-gramas*, tiene que ser la función `unnest_tokens()` la que divida el texto, por lo que se elimina la tokenización de la función `limpiar_tokenizar`.

```
library(tidytext)
limpiar <- function(texto){
  # El orden de la limpieza no es arbitrario
    # Se convierte todo el texto a minúsculas
    nuevo_texto <- tolower(texto)
    # Eliminación de páginas web (palabras que empiezan por "http." seguidas
    # de cualquier cosa que no sea un espacio)
    nuevo_texto <- str_replace_all(nuevo_texto,"http\\S*", "")
    # Eliminación de signos de puntuación
    nuevo_texto <- str_replace_all(nuevo_texto,"[[:punct:]]", " ")
    # Eliminación de números
    nuevo_texto <- str_replace_all(nuevo_texto,"[[:digit:]]", " ")
    # Eliminación de espacios en blanco múltiples
    nuevo_texto <- str_replace_all(nuevo_texto,"[\\s]+", " ")
    return(nuevo_texto)
}

bigramas <- tweets %>% mutate(texto = limpiar(texto)) %>%
           select(texto) %>%
           unnest_tokens(input = texto, output = "bigrama",
                         token = "ngrams",n = 2, drop = TRUE)
# Contaje de ocurrencias de cada bigrama
bigramas  %>% count(bigrama, sort = TRUE)
```

```
## # A tibble: 67,288 x 2
##      bigrama     n
##        <chr> <int>
##  1    of the   345
##  2    in the   267
##  3     we re   206
##  4   for the   195
##  5      is a   183
##  6 the world   180
##  7    we are   163
##  8       i m   156
##  9      it s   154
## 10   will be   147
## # ... with 67,278 more rows
```

Los bigramas más frecuentes son los formados por *stopwords*. Como la relación entre estas palabras no aporta información de interés, se procede a eliminar todos aquellos bigramas que contienen alguna *stopword*.

```
# Separación de los bigramas
bigrams_separados <- bigramas %>% separate(bigrama, c("palabra1", "palabra2"),
                                           sep = " ")
head(bigrams_separados)
```

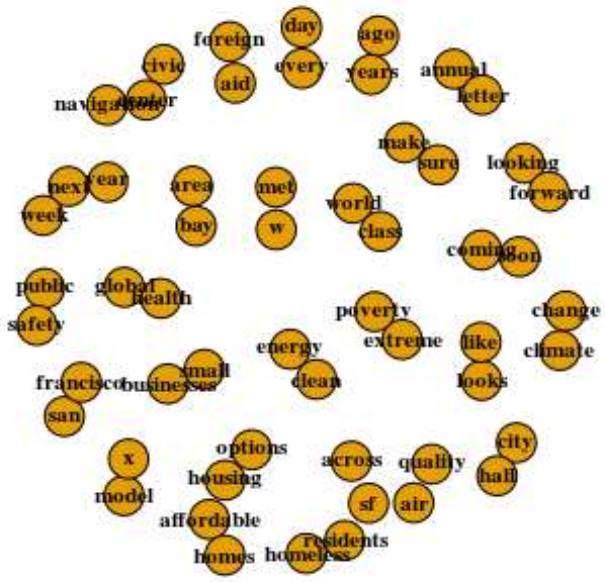
```
## # A tibble: 6 x 2
##   palabra1 palabra2
##      <chr>    <chr>
## 1       if      one
## 2      one      day
## 3      day       my
## 4       my    words
## 5    words      are
## 6      are  against
```

```
# Filtrado de los bigramas que contienen alguna stopword
bigrams_separados <- bigrams_separados  %>%
                     filter(!palabra1 %in% lista_stopwords) %>%
                     filter(!palabra2 %in% lista_stopwords)

# Unión de las palabras para formar de nuevo los bigramas
bigramas <- bigrams_separados %>%
            unite(bigrama, palabra1, palabra2, sep = " ")

# Nuevo contaje para identificar los bigramas más frecuentes
bigramas  %>% count(bigrama, sort = TRUE) %>% print(n = 20)
```

```
## # A tibble: 29,579 x 2
##                bigrama     n
##                  <chr> <int>
##  1 affordable housing    55
##  2       sf residents    50
##  3     climate change    41
##  4          make sure    39
##  5      san francisco    37
##  6        world class    33
##  7          city hall    31
##  8      public safety    30
##  9   affordable homes    29
## 10      annual letter    29
## 11          across sf    27
## 12    looking forward    27
## 13        air quality    26
## 14           bay area    26
## 15          every day    25
## 16      global health    25
## 17          next week    24
## 18            model x    23
## 19  navigation center    23
## 20    housing options    22
## # ... with 2.956e+04 more rows
```

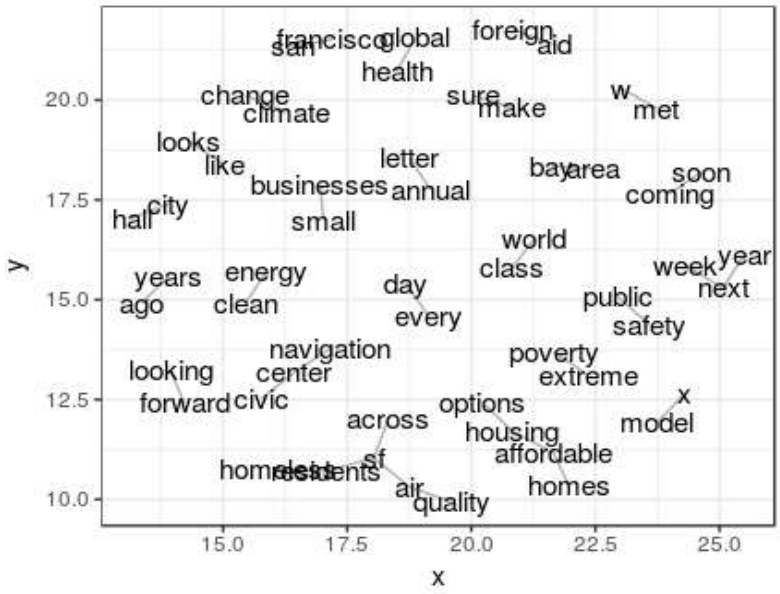
Una forma más visual e informativa de analizar las relaciones entre palabras es mediante el uso de *networks*. El paquete `igraph` permite crear *networks* a partir de *dataframes* que tenga una estructura de columnas de tipo: elemento_A, elemento_B, conexión.

```
library(igraph)
library(ggraph)
graph <- bigramas %>%
         separate(bigrama, c("palabra1", "palabra2"), sep = " ") %>%
         count(palabra1, palabra2, sort = TRUE) %>%
         filter(n > 18) %>% graph_from_data_frame(directed = FALSE)
set.seed(123)

plot(graph, vertex.label.font = 2,
     vertex.label.color = "black",
     vertex.label.cex = 0.7, edge.color = "gray85")
```




Con el paquete `ggraph` se pueden generar representaciones gráficas de *networks* basadas en `ggplot2`.

```
ggraph(graph = graph) +
  geom_edge_link(colour = "gray70")  +
  geom_node_text(aes(label = name), size = 4) +
  theme_bw()
```




## Term Frequency e Inverse Document Frequency

Uno de los principales intereses en *text mining*, *natural language processing* e *information retrieval* es cuantificar la temática de un texto, así como la importancia de cada término que lo forma. Una manera sencilla de medir la importancia de un término dentro de un documento es utilizando la frecuencia con la que aparece (*tf*, *term-frequency*). Esta aproximación, aunque simple, tiene la limitación de atribuir mucha importancia a aquellas palabras que aparecen muchas veces aunque no aporten información selectiva. Por ejemplo, si la palabra *matemáticas* aparece 5 veces en un documento y la palabra *página* aparece 50, la segunda tendrá 10 veces más peso a pesar de que no aporte tanta información. Para solucionar este problema se pueden ponderar los valores *tf* multiplicándolos por la inversa de la frecuencia con la que el término en cuestión aparece en el resto de documentos del corpus (*idf*). De esta forma, se consigue reducir el valor de aquellos términos que aparecen en muchos documentos y que, por lo tanto, no aportan información selectiva.

El estadístico tf-idf mide cómo de importante es un término en un documento teniendo en cuenta la frecuencia con la que ese término aparece en otros documentos.

*Nota: Existen implementaciones más sofisticadas de tf-idf*

## Term Frequency

$$\text{tf (término)} = \frac{n_{\text{término}}}{\text{longitud documento}}$$

```
# Número de veces que aparece cada término por tweet
tweets_tf <- tweets_tidy %>% group_by(tweet_id, token) %>% summarise(n = n())

# Se añade una columna con el total de términos por tweet
tweets_tf <- tweets_tf %>% mutate(total_n = sum(n))

# Se calcula el tf
tweets_tf <- tweets_tf %>% mutate(tf = n / total_n )
head(tweets_tf)
```

```
## # A tibble: 6 x 5
## # Groups:   tweet_id [1]
##       tweet_id       token     n total_n         tf
##          <dbl>       <chr> <int>   <int>      <dbl>
## 1 1.195196e+17     efforts     1      13 0.07692308
## 2 1.195196e+17 eradicating     1      13 0.07692308
## 3 1.195196e+17      finish     1      13 0.07692308
## 4 1.195196e+17         job     1      13 0.07692308
## 5 1.195196e+17       local     1      13 0.07692308
## 6 1.195196e+17        made     1      13 0.07692308
```

## Inverse Document Frequency

$$\text{idf (término)} = \log\left(\frac{n_{\text{documentos}}}{n_{\text{documentos con el término}}}\right)$$

```
total_documentos = tweets_tidy$tweet_id %>% unique() %>% length()
total_documentos
```

```
## [1] 7175
```

```
# Número de documentos en los que aparece cada término
tweets_idf <- tweets_tidy %>% distinct(token, tweet_id) %>% group_by(token) %>%
              summarise(n_documentos = n())

# Cálculo del idf
tweets_idf <- tweets_idf %>% mutate(idf = n_documentos/ total_documentos) %>%
              arrange(desc(idf))
head(tweets_idf)
```

```
## # A tibble: 6 x 3
##   token n_documentos        idf
##   <chr>        <int>      <dbl>
## 1    sf          906 0.12627178
## 2  city          362 0.05045296
## 3 great          318 0.04432056
## 4 world          294 0.04097561
## 5   new          286 0.03986063
## 6 today          277 0.03860627
```

## Term Frequency - Inverse Document Frequency

```
tweets_tf_idf <- left_join(x = tweets_tf, y = tweets_idf, by = "token") %>%
ungroup()
tweets_tf_idf <- tweets_tf_idf %>% mutate(tf_idf = tf * idf)
tweets_tf_idf %>% select(-tweet_id) %>% head() %>% kable()
```

| token | n | total_n | tf | n_documentos | idf | tf_idf |
|---|---|---|---|---|---|---|
| efforts | 1 | 13 | 0.0769231 | 31 | 0.0043206 | 0.0003324 |
| eradicating | 1 | 13 | 0.0769231 | 8 | 0.0011150 | 0.0000858 |
| finish | 1 | 13 | 0.0769231 | 7 | 0.0009756 | 0.0000750 |
| job | 1 | 13 | 0.0769231 | 36 | 0.0050174 | 0.0003860 |
| local | 1 | 13 | 0.0769231 | 84 | 0.0117073 | 0.0009006 |
| made | 1 | 13 | 0.0769231 | 67 | 0.0093380 | 0.0007183 |

Para el primer tweet, todos los términos que aparecen una vez tienen el mismo valor de *tf*, sin embargo, dado que no todos los términos aparecen con la misma frecuencia en el conjunto de todos los tweets, la corrección de *idf* es distinta para cada uno.

# Clasificación de tweets

Para poder aplicar algoritmos de clasificación a un texto, es necesario crear una representación numérica del mismo. Una de las formas más utilizadas se conoce como *Bag of Words*. Este método consiste en identificar el set formado por todas las palabras (tokens) que aparecen en el corpus, en este caso el conjunto de todos los tweets recuperados. Con este set se crea un espacio n-dimensional en el que cada dimensión es una palabra. Por último, se proyecta cada texto en ese espacio, asignando un valor para cada dimensión. En la mayoría de casos, ese valor es el *tf-idf*.

En el siguiente apartado se construye un modelo de aprendizaje estadístico basado en máquinas de vector soporte (*SVM*) con el objetivo de predecir la autoría de los tweets. En concreto, se comparan los tweets de Elon Musk y Mayor Ed Lee.

## Separación de los datos en entrenamiento y test

En todo proceso de aprendizaje estadístico es recomendable repartir las observaciones en un set de entrenamiento y otro de test. Esto permite evaluar la capacidad del modelo. Para este ejercicio se selecciona como test un 20% aleatorio de los tweets.

```
tweets_elon_ed <- tweets %>% filter(autor %in% c("elonmusk", "mayoredlee"))
set.seed(123)
train <- sample(x = 1:nrow(tweets_elon_ed), size = 0.8 * nrow(tweets_elon_ed))
tweets_train <- tweets_elon_ed[train, ]
tweets_test  <- tweets_elon_ed[-train, ]
```

Es importante verificar que la proporción de cada grupo es similar en el set de entrenamiento y en el de test.

```
table(tweets_train$autor) / length(tweets_train$autor)
```

```
##
##   elonmusk mayoredlee
##  0.5256098  0.4743902
```

```
table(tweets_test$autor) / length(tweets_test$autor)
```

```
##
##   elonmusk mayoredlee
##  0.5102439  0.4897561
```

## Vectorización tf-idf

Empleando los tweets de entrenamiento se crea un matriz *tf-idf* en la que cada columna es un término, cada fila un documento y el valor de intersección el *tf-idf* correspondiente. Esta matriz representa el espacio n-dimensional en el que se proyecta cada tweet. Las funciones `dfm()` y `tfidf()` del paquete `quanteda` automatizan la creación una matriz *df-idf* a partir de un corpus de documentos. Además, incorpora un tokenizador con múltiples opciones de limpieza.

```
library(quanteda)
texto <- paste0("Esto es 1 ejemplo de l'limpieza de6 TEXTO",
                "https://t.co/rnHPgyhx4Z @JoaquinAmatRodrigo #textmining")

matriz_tfidf <- dfm(x = texto, what = "word", remove_numbers = TRUE,
                    remove_punct = TRUE, remove_symbols = TRUE,
                    remove_separators = TRUE, remove_twitter = FALSE,
                    remove_hyphens = TRUE, remove_url = FALSE)
colnames(matriz_tfidf)
```

```
##  [1] "esto"               "es"                 "ejemplo"
##  [4] "de"                 "l'limpieza"         "de6"
##  [7] "textohttps"         "t.co"               "rnhpgyhx4z"
## [10] "@joaquinamatrodrigo" "#textmining"
```

Viendo los resultados de la tokenización realizada mediante `quanteda`, parece que la función de limpieza y tokenización definida en apartados anteriores funciona mejor para la limpieza de tweets.

```
limpiar_tokenizar(texto = texto)
```

```
## [1] "esto"              "es"                "ejemplo"
## [4] "de"                "limpieza"          "de"
## [7] "texto"             "joaquinamatrodrigo" "textmining"
```

La función `dfm()` interpreta cada elemento de un vector *character* como un documento distinto. Como el resultado de la función `limpiar_tokenizar()` trasforma cada tweet en un vector de palabras, es necesario concatenar el resultado para que formen de nuevo un único elemento.

```
paste(limpiar_tokenizar(texto = texto), collapse = " ")
```

```
## [1] "esto es ejemplo de limpieza de texto joaquinamatrodrigo textmining"
```

```
# Limpieza y tokenización de los documentos de entrenamiento
tweets_train$texto <- tweets_train$texto %>% map(.f = limpiar_tokenizar) %>%
                      map(.f = paste, collapse = " ") %>% unlist()

# Creación de la matriz documento-término
matriz_tfidf_train <- dfm(x = tweets_train$texto, remove = lista_stopwords)

# Se reduce la dimensión de la matriz eliminando aquellos términos que
# aparecen en menos de 5 documentos. Con esto se consigue eliminar ruido.
matriz_tfidf_train <- dfm_trim(x = matriz_tfidf_train, min_docfreq = 5)

# Conversión de los valores de la matriz a tf-idf
matriz_tfidf_train <- tfidf(matriz_tfidf_train, scheme_tf = "prop",
                            scheme_df = "inverse")
matriz_tfidf_train
```

```
## Document-feature matrix of: 4,100 documents, 1,698 features (99.6% sparse).
```

A la hora de trasformar los documentos de test, es importante proyectarlos en la misma matriz obtenida previamente con el set de entrenamiento. Esto es importante ya que, si en los documentos de test hay algún término que no aparece en los de entrenamiento o viceversa, las dimensiones de cada matriz no coincidirán. Para evitar este problema, al crear la matriz *tf-idf* de test, se pasa como argumento `dictionary` el nombre de las columnas de la matriz *tf-idf* de entrenamiento. El argumento `dictionary` tiene que ser de tipo *diccionario*, la transformación de un vector a un diccionario es un tanto compleja ya que tiene que convertirse primero a lista.

```
# Limpieza y tokenización de los documentos de test
tweets_test$texto <- tweets_test$texto %>% map(.f = limpiar_tokenizar) %>%
                     map(.f = paste, collapse = " ") %>% unlist()
# Identificación de las dimensiones de la matriz de entrenamiento
# Los objetos dm() son de clase S4, se accede a sus elementos mediante @
dimensiones_matriz_train <- matriz_tfidf_train@Dimnames$features
# Conversión de vector a diccionario pasando por lista
dimensiones_matriz_train <- as.list(dimensiones_matriz_train)
names(dimensiones_matriz_train) <- unlist(dimensiones_matriz_train)
dimensiones_matriz_train <- dictionary(dimensiones_matriz_train)

# Proyección de los documentos de test
matriz_tfidf_test <- dfm(x = tweets_test$texto,
                         dictionary = dimensiones_matriz_train)
matriz_tfidf_test <- tfidf(matriz_tfidf_test, scheme_tf = "prop",
                           scheme_df = "inverse")
matriz_tfidf_test
```

```
## Document-feature matrix of: 1,025 documents, 1,698 features (99.6% sparse).
```

Se comprueba que las dimensiones de ambas matrices son iguales.

```
all(colnames(matriz_tfidf_test) == colnames(matriz_tfidf_train))
```

```
## [1] TRUE
```

## Modelo SVM lineal

Como modelo de predicción se emplea un *SVM*. Este método de aprendizaje estadístico suele dar buenos resultados en clasificación. Para información más detallada consultar Máquinas de Vector Soporte (Support Vector Machines, SVMs)

### Ajuste del modelo

```
library(e1071)
modelo_svm <- svm(x = matriz_tfidf_train, y = as.factor(tweets_train$autor),
                  kernel = "linear", cost = 1, scale = TRUE,
                  type = "C-classification")
modelo_svm
```

```
##
## Call:
## svm.default(x = matriz_tfidf_train, y = as.factor(tweets_train$autor),
##     scale = TRUE, type = "C-classification", kernel = "linear",
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.0005889282
##
## Number of Support Vectors:  1469
```

## Predicciones

Empleando el modelo entrenado en el paso anterior se predice la autoría de los tweets de test.

```
predicciones <- predict(object = modelo_svm, newdata = matriz_tfidf_test)
```

## Error de predicción

```
# Matriz de confusión
table(observado = tweets_test$autor, predicho = predicciones)
```

```
##             predicho
## observado    elonmusk mayoredlee
##   elonmusk         498         25
##   mayoredlee        17        485
```

```
# Error de clasificación
clasificaciones_erroneas <- sum(tweets_test$autor != predicciones)
error <- 100 * mean(tweets_test$autor != predicciones)
paste("Número de clasificaciones incorrectas =", clasificaciones_erroneas)
```

```
## [1] "Número de clasificaciones incorrectas = 42"
```

```
paste("Porcentaje de error =", round(error,2), "%")
```

```
## [1] "Porcentaje de error = 4.1 %"
```

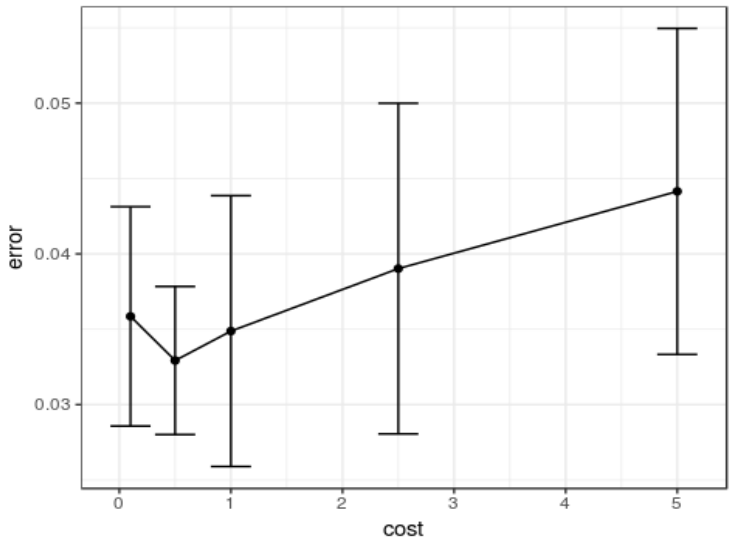
## Optimización de hiperparámetros

El método de *SVM* lineal tiene un único hiperparámetro *C* que establece la penalización por clasificación incorrecta, regulando así el balance entre bias y varianza. Al tratarse de un hiperparámetro, su valor óptimo no se aprende en el proceso de entrenamiento, para estimarlo hay que recurrir a validación cruzada.

```
set.seed(369)
svm_cv <- tune("svm", train.x =  matriz_tfidf_train,
               train.y = as.factor(tweets_train$autor),
               kernel = "linear",
               ranges = list(cost = c(0.1, 0.5, 1, 2.5, 5)))
summary(svm_cv)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.03292683
##
## - Detailed performance results:
##   cost      error  dispersion
## 1  0.1 0.03585366 0.007276309
## 2  0.5 0.03292683 0.004911807
## 3  1.0 0.03487805 0.008983648
## 4  2.5 0.03902439 0.010968079
## 5  5.0 0.04414634 0.010813314
```

```
ggplot(data = svm_cv$performances, aes(x = cost, y = error)) +
  geom_line() +
  geom_point() +
  geom_errorbar(aes(ymin = error - dispersion, ymax = error + dispersion)) +
  theme_bw()
```




```
svm_cv$best.parameters
```

```
##   cost
## 2  0.5
```

Acorde al error estimado por validación cruzada, el valor óptimo del hiperparámetro *C* es 0.5. Se reajusta el modelo con este valor.

```
modelo_svm <- svm(x = matriz_tfidf_train, y = as.factor(tweets_train$autor),
                  kernel = "linear", cost = 0.5, scale = TRUE)

predicciones <- predict(object = modelo_svm, newdata = matriz_tfidf_test)
table(observado = tweets_test$autor, predicho = predicciones)
```

```
##             predicho
## observado    elonmusk mayoredlee
##   elonmusk         503         20
##   mayoredlee        16        486
```

```
clasificaciones_erroneas <- sum(tweets_test$autor != predicciones)
error <- 100 * mean(tweets_test$autor != predicciones)
paste("Número de clasificaciones incorrectas =", clasificaciones_erroneas)
```

```
## [1] "Número de clasificaciones incorrectas = 36"
```

```
paste("Porcentaje de error =", round(error,2), "%")
```

```
## [1] "Porcentaje de error = 3.51 %"
```

Empleando un modelo de *SVM* lineal con hiperparámetro $C = 0.5$ se consigue un *test error* del 3.51%. Se trata de un porcentaje de error bastante bajo, aun así, explorando otros modelos como pueden ser *SVM* no lineales o *Random Forest* podrían alcanzarse mejores resultados.

## Análisis de sentimientos

Una forma de analizar el sentimiento de un de un texto es considerando su sentimiento como el la suma de los sentimientos de cada una de las palabras que lo forman. Esta no es la única forma abordar el análisis de sentimientos, pero consigue un buen equilibro entre complejidad y resultados. Para llevar a cabo esta aproximación es necesario disponer de un diccionario en el que se asocie a cada palabra un sentimiento o nivel de sentimiento. A estos diccionarios también se les conoce como *sentiment lexicon*. El paquete `tidytext` contiene 3 diccionarios distintos:

- `AFINN`: asigna a cada palabra un valor entre -5 y 5. Siendo -5 el máximo de negatividad y +5 el máximo de positividad.
- `bing`: clasifica las palabras de forma binaria positivo/negativo.
- `nrc`: clasifica cada palabra en uno o más de los siguientes sentimientos: *positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust*.

En este ejercicio se emplea la clasificación positivo/negativo proporcionada por el diccionario `bing`.

```
library(tidytext)
sentimientos <- get_sentiments(lexicon = "bing")
head(sentimientos)
```

```
## # A tibble: 6 x 2
##         word sentiment
##        <chr>     <chr>
## 1    2-faced  negative
## 2    2-faces  negative
## 3         a+  positive
## 4   abnormal  negative
## 5    abolish  negative
## 6 abominable  negative
```

Para facilitar el cálculo de sentimientos globales (autor, tweet…) se recodifican los sentimientos como +1 para positivo y -1 para negativo.

```
sentimientos <- sentimientos %>%
                mutate(valor = if_else(sentiment == "negative", -1, 1))
```

## Sentimiento promedio de cada tweet

Al disponer de los datos en formato *tidy* (una palabra por fila), mediante un *inner join* se añade a cada palabra su sentimiento y se filtran automáticamente todas aquellas palabras para las que no hay información disponible.

```
tweets_sent <- inner_join(x = tweets_tidy, y = sentimientos,
                          by = c("token" = "word"))
```

Se suman los sentimientos de las palabras que forman cada tweet.

```
tweets_sent %>% group_by(autor, tweet_id) %>%
                summarise(sentimiento_promedio = sum(valor)) %>%
                head()
```

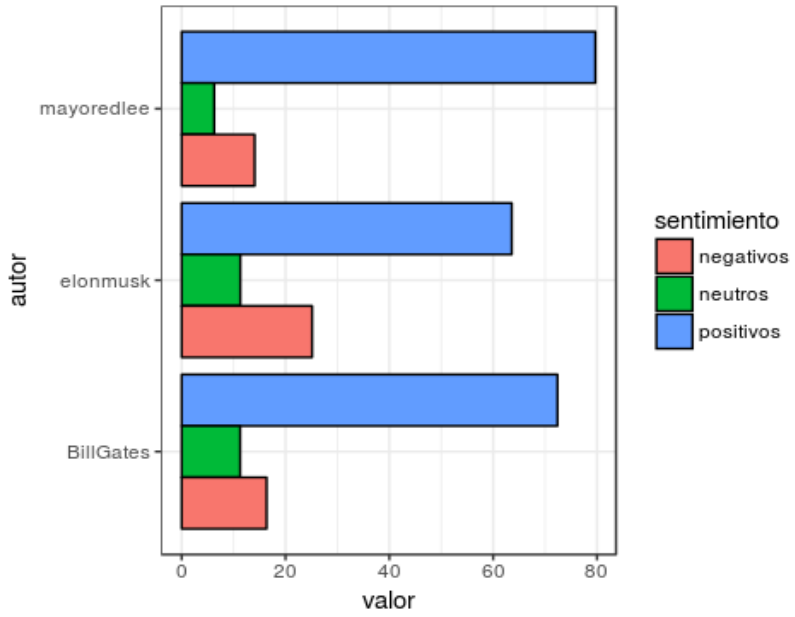
```
## # A tibble: 6 x 3
## # Groups:   autor [1]
##       autor     tweet_id sentimiento_promedio
##       <chr>        <dbl>                <dbl>
## 1 BillGates 1.195196e+17                    4
## 2 BillGates 1.213001e+17                   -1
## 3 BillGates 1.217459e+17                    2
## 4 BillGates 1.217460e+17                    3
## 5 BillGates 1.267828e+17                    1
## 6 BillGates 1.268263e+17                    1
```

## Porcentaje de tweets positivos, negativos y neutros por autor

```
tweets_sent %>% group_by(autor, tweet_id) %>%
                summarise(sentimiento_promedio = sum(valor)) %>%
                group_by(autor) %>%
                summarise(positivos = 100 * sum(sentimiento_promedio > 0) / n(),
                          neutros = 100 * sum(sentimiento_promedio == 0) / n(),
                          negativos = 100 * sum(sentimiento_promedio  < 0) / n())
```

```
## # A tibble: 3 x 4
##        autor positivos   neutros negativos
##        <chr>     <dbl>     <dbl>     <dbl>
## 1  BillGates  72.38994 11.257862  16.35220
## 2   elonmusk  63.58504 11.291461  25.12350
## 3 mayoredlee  79.68410  6.263617  14.05229
```

```
tweets_sent %>% group_by(autor, tweet_id) %>%
                summarise(sentimiento_promedio = sum(valor)) %>%
                group_by(autor) %>%
                summarise(positivos = 100*sum(sentimiento_promedio > 0) / n(),
                          neutros = 100*sum(sentimiento_promedio == 0) / n(),
                         negativos = 100*sum(sentimiento_promedio  < 0) / n()) %>%
              ungroup() %>%
              gather(key = "sentimiento", value = "valor", -autor) %>%
              ggplot(aes(x = autor, y = valor, fill = sentimiento)) +
              geom_col(position = "dodge", color = "black") + coord_flip() +
              theme_bw()
```



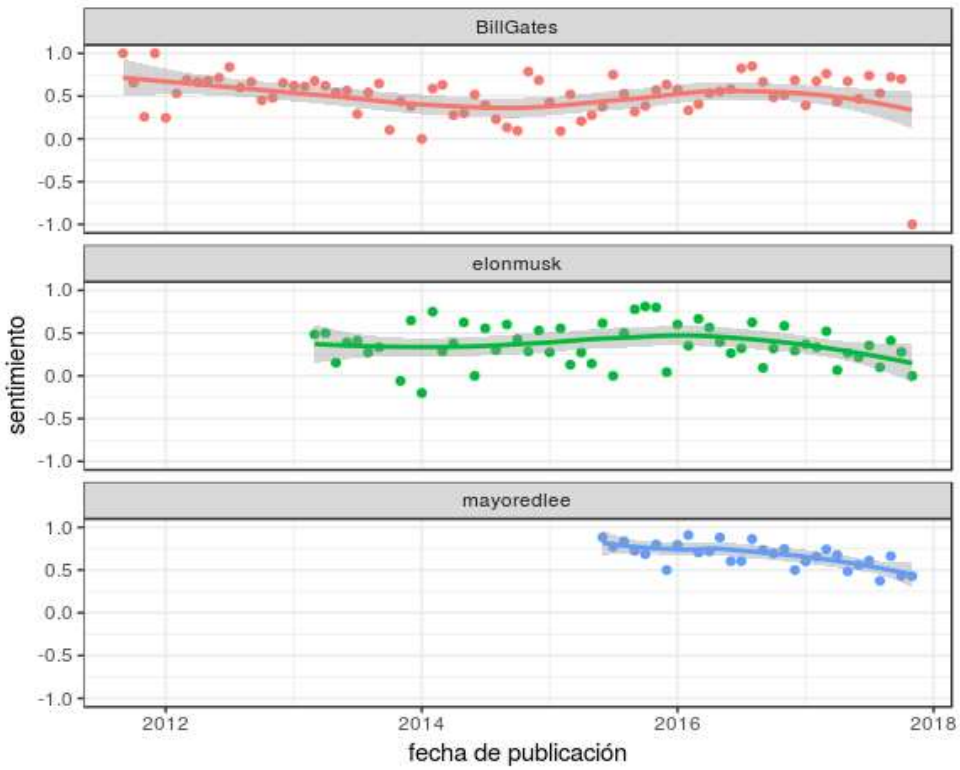

Los tres autores tienen un perfil muy similar. La gran mayoría de tweets son de tipo positivo. Este patrón es común en redes sociales, donde se suele participar mostrando aspectos o actividades positivas. Los usuarios no tienden a mostrar las cosas malas de sus vidas.

## Evolución de los sentimientos en función del tiempo

A continuación, se estudia como varía el sentimiento promedio de los tweets agrupados por intervalos de un mes para cada uno de los usuarios.

```
library(lubridate)
tweets_sent %>% mutate(anyo = year(fecha),
                       mes = month(fecha),
                       anyo_mes = ymd(paste(anyo, mes, sep="-"),truncated=2)) %>%
                group_by(autor, anyo_mes) %>%
```

```
            summarise(sentimiento = mean(valor)) %>%
            ungroup() %>%
            ggplot(aes(x = anyo_mes, y = sentimiento, color = autor)) +
            geom_point() +
            geom_smooth() +
            labs(x = "fecha de publicación") +
            facet_wrap(~ autor, ncol = 1) +
            theme_bw() +
            theme(legend.position = "none")
```




La distribución del sentimiento promedio de los tweets se mantiene aproximadamente constante para los 3 usuarios. Existen ciertas oscilaciones, pero todas ellas dentro del rango de sentimiento positivo.

# Bibliografía

*Text Mining with R A Tidy Approach by Julia Silge and David Robinson*

*Search Engines: Information Retrieval in Practice by Trevor Strohman, Donald Metzler, W. Bruce Croft*

http://varianceexplained.org/r/trump-tweets/

http://programminghistorian.github.io/ph-submissions/lessons/published/basic-text-processing-in-r

http://cfss.uchicago.edu/fall2016/text01.html This work by Joaquín Amat Rodrigo is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

This work by Joaquín Amat Rodrigo is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.