# Flujo de análisis en clasificación supervisada

Métodos supervisados

Laura Rodríguez Navas

Septiembre 2020

## Contents



Empezamos por cargar a nuestro espacio de trabajo los paquetes que usaremos:

- **tidyverse**, engloba otros paquetes (**dplyr**, **tidyr**, **ggplot**, etc.) que facilitan en gran medida el análisis exploratorio de los datos.
- **stringi**,
- **tm**, específico para minería de textos.
- **irlba**,
- **caret**,
- **doParallel**, proporciona computación paralela.
- **syuzhet**, específico para la extracción de sentimientos de textos.
- **ggcorrplot**, muestra visualizaciones gráficas de matrices de correlación usando *ggplot2*.
- **gbm**,

```
library(tidyverse)
library(stringi)
library(tm)
library(irlba)
library(RColorBrewer)
library(gridExtra)
library(caret)
library(doParallel)
library(syuzhet)
library(ggcorrplot)
library(gbm)
```

## Análisis Exploratorio de los Datos

Antes de entrenar un modelo predictivo, o incluso antes de realizar cualquier cálculo con un nuevo conjunto de datos, es muy importante realizar una exploración descriptiva de los datos. Este proceso nos permite entender mejor que información contienen cada variable, detectar posibles errores, etc. además, puede dar pistas sobre qué variables no son adecuadas para predecir un modelo.

Acorde a la realización del ejercicio propuesto se ha elegido la competición en Kaggle: **Real or Not? NLP with Disaster Tweets**. El dataset de la competición se puede encontrar en el siguiente enlace: https://www.kaggle.com/c/nlp-getting-started/data. Este dataset, con 10.876 instancias, contiene 4 variables explicativas: **id**, **keyword**, **location** y **text**, y dos valores en la variable clase **target** (0 y 1). Como podemos observar la variable clase es binaria, así que, durante este ejercicio vamos a aprender un modelo de *clasificación binaria*. El objetivo de este modelo será predecir si dado un tweet, éste trata sobre un desastre real o no. Si un tweet trata sobre un desastre real, se predice un 1. Si no, se predice un 0.

*La clasificación binaria es un tipo de clasificación en el que tan solo se pueden asignar dos clases diferentes (0 o 1).*

La métrica de evaluación esperada por la competición es **F1 score**. Para ver como se calcula consultar el siguiente enlace: https://www.kaggle.com/c/nlp-getting-started/overview/evaluation.

La partición inicial train-test, no se tiene que realizar, ya que las instancias de train y test ya vienen definidas en el dataset de la competición (descargar a nuestro espacio de trabajo los ficheros **train.csv** y **test.csv** de https://www.kaggle.com/c/nlp-getting-started/data).

Cargamos a nuestro espacio de trabajo los conjuntos de datos de train y test descargados, renombrando los valores perdidos como **NA** para que los podamos tratar más adelante. También mostramos las dimensiones de los conjuntos de datos usando la función **dim**.

```
train <- read.csv("train.csv", na.strings=c("", "NA"))
test <- read.csv("test.csv", na.strings=c("", "NA"))
dim(train)
```

```
## [1] 7613    5
```

```
dim(test)
```

```
## [1] 3263    4
```

Vemos que el conjunto de datos de train contiene 7613 instancias y el conjunto de datos de test contiene 3263 instancias. Cada instancia contiene las siguientes variables:

- **id**: un identificador único para cada tweet.
- **keyword**: una palabra clave del tweet.
- **location**: la ubicación desde la que se envió el tweet.
- **text**: el texto del tweet.
- **target**: solo en el conjunto de datos de train porqué es la variable clase a predecir. Indica si un tweet corresponde a un desastre real (1) o no (0).

```
## 'data.frame':    7613 obs. of  5 variables:
##  $ id      : int  1 4 5 6 7 8 10 13 14 15 ...
##  $ keyword : chr  NA NA NA NA ...
##  $ location: chr  NA NA NA NA ...
##  $ text    : chr  "Our Deeds are the Reason of this #earthquake May ALLAH Forgive "..
##  $ target  : int  1 1 1 1 1 1 1 1 1 1 ...
```

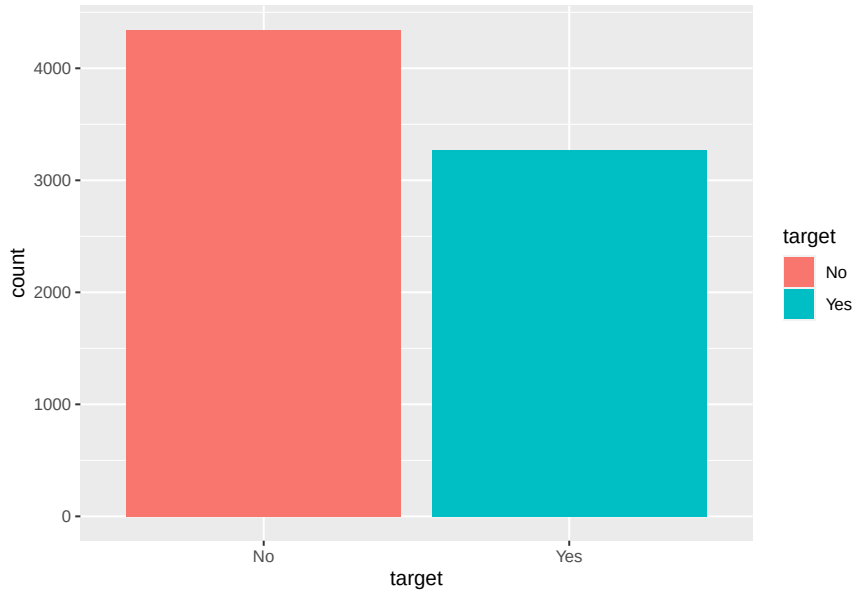
```
## 'data.frame':    3263 obs. of  4 variables:
##  $ id      : int  0 2 3 9 11 12 21 22 27 29 ...
##  $ keyword : chr  NA NA NA NA ...
##  $ location: chr  NA NA NA NA ...
```

```
##  $ text    : chr  "Just happened a terrible car crash" "Heard about #earthquake is"..
```

## Variable *target*

Como ya hemos comentado, la variable **target** es la variable a predecir. Es de tipo cuantitativa (de tipo entero) y conviene convertirla a variable cualitativa, almacenarla con el tipo *factor*. Para evitar errores, se recodifica para que sus dos posibles valores sean "Yes"-"No" y se convierte a *factor*.

```
train$target <- as.factor(ifelse(train$target == 0, "No", "Yes"))
ggplot(train, aes(x=target)) + geom_bar(aes(fill=target))
```




Cuando se crea un modelo, es muy importante estudiar la distribución de la variable clase, ya que, a fin de cuentas, es lo que nos interesa predecir.

Gráficamente observamos que la distribución de la variable a predecir no está muy sesgada y está relativamente equilibrada. Hay menos tweets que se refieren a desastres reales. Además, parece que no presenta un problema notable de *desbalanceo de clase*, porqué contamos con muchas observaciones del caso minoritario.

```
sum(train$target == "Yes") / dim(train)[1] * 100
```

```
## [1] 42.96598
```

```
sum(train$target == "No") / dim(train)[1] * 100
```

```
## [1] 57.03402
```

Para que un modelo predictivo nos sea útil tendremos que intentar superar el porcentaje mínimo dado que aproximadamente el 57% de los tweets no representan un desastre real (este porcentaje se recalculará únicamente con el conjunto de datos de train).

Como el objetivo del ejercicio es predecir que tweets pertenecen o no a un desastre real, el análisis que haremos a continuación se hace realiza de cada variable explicativa con relación a la variable a predecir **target**. Analizando de esta forma, se pueden extraer ideas sobre que variables están más relacionadas con los desastres reales.

### Variable *keyword*

La variable explicativa **keyword** representa una palabra clave en cada tweet. Vemos las 10 primeras del conjunto de datos de train.

```
train %>% select(keyword) %>% unique() %>% head(10)
```

```
##                 keyword
## 1                  <NA>
## 32               ablaze
## 68             accident
## 103          aftershock
## 137 airplane%20accident
## 172           ambulance
## 210         annihilated
## 244        annihilation
## 273          apocalypse
## 305          armageddon
```

Nuestro interés en la variable **keyword** dentro del análisis exploratorio de los datos es ver si existen correlaciones entre esta y la variable a predecir **target**. Para ello, y como estamos delante un ejercicio de *Procesamiento del Lenguaje Natural* realizaremos un análisis de sentimientos.

*El análisis de sentimientos es una técnica de Machine Learning, basada en el Procesado del Lenguaje Natural, que pretende obtener información subjetiva de una serie de textos. Su aplicación es este caso, consiste en resolver si un tweet es real o no en relación a un desastre.*

En el análisis de sentimientos usamos los paquetes de R: **syuzhet**, **ggcorrplot** y **doParallel**.

- El paquete **syuzhet** cuenta con la función **get_nrc_sentiment** que calculará la presencia de los diferentes sentimientos dado un conjunto de palabras clave. Los parámetros de esta función son:
  - **char_v**. Un vector de caracteres que en este caso contendrá todas las palabras clave.
  - **language**. Define el lenguaje. Como los tweets están en inglés, el lenguaje será el inglés.
  - **cl**. Para el análisis en paralelo. Es opcional, pero en este caso lo usaremos porqué hay muchas palabras clave.
- El paquete **doParallel** cuenta con las funciones:
  - **makePSOCKcluster**. Crea un clúster de sockets paralelos.
  - **registerDoParallel**. Registra el número de *cores* que usará el clúster creado.
  - **stopCluster**. Detiene la computación paralela.

La computación paralela la usaremos en muchas de las ejecuciones de este ejercicio ya que nos encontramos delante de un problema de *alta dimensionalidad*. Eso es, que la dimensionalidad de nuestros datos es muy elevada y puede reducir drásticamente la eficiencia de los algoritmos de clasificación supervisada que entrenaremos. Una de las técnicas más ampliamente utilizada y conocida para la reducción de la dimensionalidad es Principal Component Analysis (PCA) Esta técnica lleva a cabo la transformación de los datos generando unas nuevas variables.

La reducción de la dimensionalidad que aplicaremos en este ejercicio se realiza más adelante y se calculará teniendo en cuenta las palabras más frecuentes de los tweets en conjunto de datos.

El análisis de sentimientos de cada palabra clave, usando la función **get_nrc_sentiment**, consiste en extraer los sentimientos de cada palabra clave, guardarlos en un nuevo conjunto de datos (*emotion.df*), con el que calcularemos (paquete **cor**) y visualizaremos la matriz de correlaciones (paquete **ggcorrplot**) entre las palabras clave con relación a la variable a predecir. Es importante volver a transformar la variable a predecir para realizar los cálculos, cuando la variable es cualitativa.

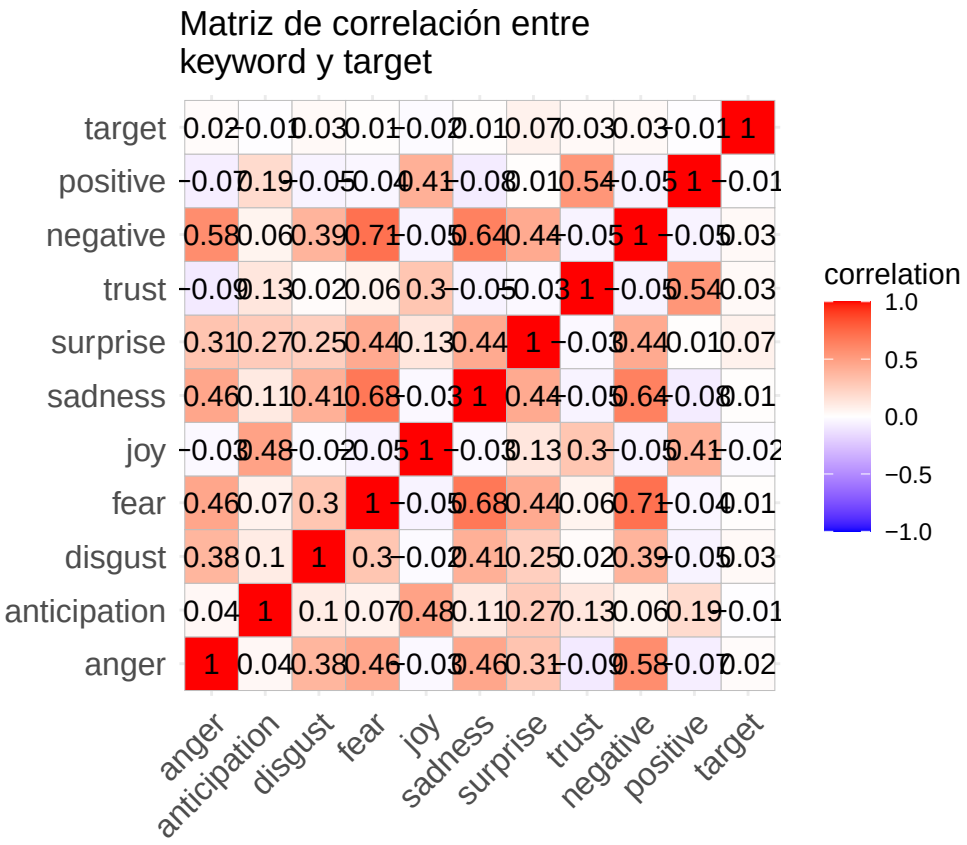
```
cl <- makePSOCKcluster(4, setup_strategy="sequential")
registerDoParallel(cl)
```

```r
emotion.df <- get_nrc_sentiment(char_v = gsub("_", " ", train$keyword),
                                language = "english", cl=cl)

emotion.df <- emotion.df %>% data.frame(target = train$target)

emotion.df$target <- as.numeric(emotion.df$target)

cor(emotion.df) %>%
  ggcorrplot(lab = TRUE,
             title = "Matriz de correlación entre \nkeyword y target",
             legend.title = "correlation")
```




```r
stopCluster(cl)
```

Parece que, al observar la matriz de correlaciones, existe una correlación nula entre las variables **keyword** y **target**. Al revisarlo con mayor detalle, podemos observar que la mayoría de las palabras clave no tienen un sentimiento positivo asociado. Las palabras clave asociadas a un sentimiento se asocian negativamente (miedo o tristeza), lo cual es bastante consistente con el problema, ya que intentemos predecir el desastre.

Acorde a nuestro criterio esta variable explicativa no es buena para hacer una predicción ya que no está realmente asociada con la variable a predecir. Así que la excluiremos del procesamiento de texto.

## Variable *location*

La variable explicativa **location** representa las ubicaciones desde donde se generaron los tweets. Vemos las 10 primeras y el número total de ubicaciones diferentes del conjunto de datos de train (3342 ubicaciones).

```
train %>% select(location) %>% unique() %>% head(10)
```

```
##                          location
## 1                            <NA>
## 32                     Birmingham
## 33 Est. September 2012 - Bristol
## 34                          AFRICA
## 35              Philadelphia, PA
## 36                    London, UK
## 37                      Pretoria
## 38                   World Wide!!
## 40                 Paranaque City
## 41                 Live On Webcam
```

```
count(train %>% select(location) %>% unique())
```

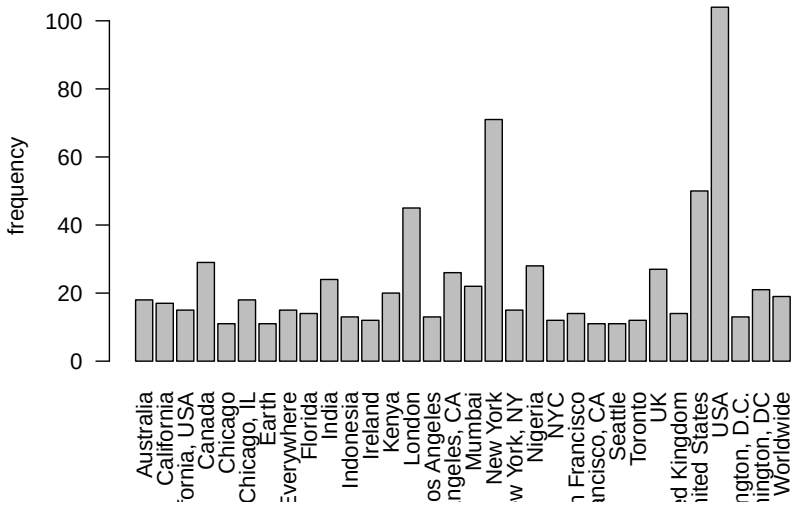
```
##      n
## 1 3342
```

A continuación, veremos las ubicaciones que se repiten más de 10 veces en el conjunto de datos de train.

```
location.freq <- table(unlist(train %>% select(location)))
location.freq[which(location.freq > 10)]
```

```
##
##         Australia        California   California, USA            Canada
##                18                17                15                29
##           Chicago       Chicago, IL             Earth        Everywhere
##                11                18                11                15
##           Florida             India         Indonesia           Ireland
##                14                24                13                12
##             Kenya            London       Los Angeles   Los Angeles, CA
##                20                45                13                26
##            Mumbai          New York      New York, NY           Nigeria
##                22                71                15                28
##               NYC     San Francisco San Francisco, CA           Seattle
##                12                14                11                11
##           Toronto                UK    United Kingdom     United States
##                12                27                14                50
##               USA  Washington, D.C.    Washington, DC         Worldwide
##               104                13                21                19
```

```
barplot(location.freq[which(location.freq>10)], las = 2,
        ylab = "frequency")
```




En el total de ubicaciones, 3342, la mayoría de ellas cuenta con menos de 10 observaciones. Acorde a nuestro criterio esta variable explicativa no es buena para hacer una predicción, ya que la variable tiene muy pocas observaciones, y puede ocurrir que, durante la validación cruzada o *bootstrapping*, algunas de las particiones no contengan ninguna observación de dicha variable, lo que puede dar lugar a errores.

### Variable *id*

La variable **id** es solo un identificador único, así que, no la analizaremos y procederemos a eliminarla de los conjuntos de datos de train y test.

```
train$id <- NULL
test$id <- NULL
```

### Conclusión análisis exploratorio

Llegados a este punto, parece que nuestro criterio en la exploración de los datos, el estudio de su distribución y sus posibles relaciones con la variable a predecir nos indica que las variables explicativas **keyword**, **location** y **id** no son buenas para hacer una predicción, así que nos centraremos en la variable **text** para hacer la predicción.

## Procesamiento de texto

Combinamos los conjuntos de datos de train y test para ahorrar esfuerzos en el preprocesado de datos. Para ello, usamos la función **bind_rows**, que nos permitirá enlazar de forma eficiente los conjuntos de datos por fila y columna. Podremos comprobar que la combinación se hace correctamente, sumando los elementos de train (7613) y de test (3263), el nuevo conjunto de datos (**complete_df**) tendrá 10876 observaciones, 3 variables explicativas (**keyword**, **location**, **text**) y la variable de clase **target**.

```
complete_df <- bind_rows(train, test)
str(complete_df, width = 85, strict.width = "cut")
```

```
## 'data.frame':    10876 obs. of  4 variables:
##  $ keyword : chr  NA NA NA NA ...
##  $ location: chr  NA NA NA NA ...
##  $ text    : chr  "Our Deeds are the Reason of this #earthquake May ALLAH Forgive "..
```

```
##  $ target  : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
```

El preprocesado de datos englobará las transformaciones de los textos, como, por ejemplo, la imputación de valores ausentes o la reducción de dimensionalidad.

Primero, miramos cuantos valores perdidos tiene nuestro conjunto de datos **complete_df**. La función **colSums** sumará los valores que la función **sapply** encuentre, en este caso, los valores perdidos.

```
colSums(sapply(complete_df, is.na))
```

```
##  keyword location     text   target
##       87     3638        0     3263
```

Identificamos que las variables explicativas **keyword** y **location** tienen valores perdidos. La variable explicativa **text** no tiene valores perdidos. Sobretodo hay una gran cantidad de tweets, para los cuales falta su ubicación. Los 3263 valores perdidos de la variable a predecir provienen del conjunto de datos de test. Nos ocuparemos de los valores perdidos más adelante.

## Corpus

Con nuestro nuevo conjunto de datos preparado (**complete_df**), procedemos a crear nuestro Corpus, es decir, el conjunto de textos de la variable **text** a analizar. En este caso, nuestro Corpus se compone de todos los textos de los tweets y los asignaremos al objeto *myCorpus* usando las funciones **VectorSource** y **Corpus**. La función **Corpus** creará el corpus a partir de un vector de textos. La función **VectorSource** interpretará cada mensaje de texto de los tweets como un elemento de ese vector de textos.

*Un corpus lingüístico se define como "un conjunto de textos de un mismo origen" y que tiene por función recopilar un conjunto de textos. El uso de un corpus lingüístico nos permitirá obtener información de las palabras utilizadas con más o menor frecuencia.*

```
myCorpus <- Corpus(VectorSource(complete_df$text))
myCorpus
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 10876
```

Como podemos ver, nuestro Corpus está compuesto por 10876 textos.

## Limpieza del texto

Necesitamos limpiar de los 10876 textos caracteres que son de poca utilidad. Empezamos por aseguramos de que no queden enlaces, con un poco de ayuda de las *regular expressions*. Para ello usaremos las funciones **gsub** y **tm_map**. La función **gsub** buscará y reemplazará desde la primera hasta la última de las coincidencias de un patrón (representado por una *regular expression*). La función **tm_map** será la encargada de aplicar las diferentes transformaciones de los textos a nuestro corpus.

*Una expresión regular (o en inglés regular expression) es una representación, según unas reglas sintácticas de un lenguaje formal, de una porción de texto genérico a buscar dentro de otro texto, como por ejemplo caracteres, palabras o patrones de texto concretos.*

```
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeURL))
```

Convertimos todo a minúsculas.

```
myCorpus <- tm_map(myCorpus, content_transformer(tolower))
```

Eliminamos los nombres de usuario.

```
removeUsername <- function(x) gsub("@[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeUsername))
```

Nos deshacemos de la puntuación, puesto que por ejemplo "fin" y "fin." son identificadas como palabras diferentes, lo cual no deseamos.

```
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeNumPunct))
```

Usamos **removeWords** con **stopwords("english")**, recordemos que los textos de los tweets están en inglés y cada idioma tiene sus propias palabras vacías; para eliminar palabras vacías, es decir, aquellas con poco valor para el análisis, que carecen de un significado por si solas, tales como artículos, preposiciones, conjunciones, pronombres, etc.

```
myStopWords <- c((stopwords('english')),
    c("really", "tweets", "saw", "just", "feel", "may", "us", "rt", "every", "one",
     "amp", "like", "will", "got", "new", "can", "still", "back", "top", "much",
     "near", "im", "see", "via", "get", "now", "come", "oil", "let", "god", "want",
     "pm", "last", "hope", "since", "everyone", "food", "content", "always", "th",
     "full", "found", "dont", "look", "cant", "mh", "lol", "set", "old", "service",
     "city", "home", "live", "night", "news", "say", "video", "people", "ill",
     "way",  "please", "years", "take", "homes", "read", "man", "next", "cross",
     "boy", "bad", "ass"))

head(myStopWords, 30)
```

```
##  [1] "i"          "me"         "my"         "myself"     "we"
##  [6] "our"        "ours"       "ourselves"  "you"        "your"
## [11] "yours"      "yourself"   "yourselves" "he"         "him"
## [16] "his"        "himself"    "she"        "her"        "hers"
## [21] "herself"    "it"         "its"        "itself"     "they"
## [26] "them"       "their"      "theirs"     "themselves" "what"
```

```
myCorpus <- tm_map(myCorpus, removeWords, myStopWords)
```

Además, podemos ver que se han añadido (aleatoriamente) más palabras vacías ("really", "tweets", "saw", etc.). Estas palabras vacías son de las más usadas en los mensajes de texto de los tweets (ver https://techland.time.com/2009/06/08/the-500-most-frequently-used-words-on-twitter/).

En este caso, removemos las palabras de una sola letra.

```
removeSingle <- function(x) gsub(" . ", " ", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeSingle))
```

Por último eliminamos los espacios vacíos excesivos, muchos de ellos introducidos por las transformaciones anteriores.

```
myCorpus <- tm_map(myCorpus, stripWhitespace)
```

# Creación de un modelo predictivo

## Preprocesado de los datos

Para la creación de un modelo predictivo, necesitamos construir una **Term Document Matrix** del conjunto de textos de la variable **text**, donde cada fila representará un texto y cada palabra única estará representada per una columna.

*Una Term Document Matrix es una matriz matemática que describe la frecuencia con la que se repiten una serie de palabras en una colección de documentos.*

Comenzaremos mapeando nuestro Corpus indicando que es una **Term Document Matrix**, de esta manera podremos realizar el preprocesado de datos. Sabemos que el preprocesado de datos engloba aquellas transformaciones de los datos con la finalidad de mejorar los resultados de la clasificación supervisada. Todo preprocesado de datos debe aprenderse de las observaciones de train y luego aplicarse al conjunto de train y de test. Esto es muy importante para no violar la condición de que ninguna información procedente de las observaciones de test influya en el ajuste del modelo.

Utilizaremos la función **TermDocumentMatrix** en nuestro Corpus y asignaremos el resultado al objeto *complete.tdm*. Con el parámetro **control** indicaremos que evaluaremos todos los textos de la matriz, con las características escogidas se evaluarán todas las palabras de los textos. Por defecto la función **TermDocumentMatrix** usa *tf-id*, que mide la importancia relativa de cada palabra en el conjunto de textos.

```
complete.tdm <- TermDocumentMatrix(myCorpus, control=list(wordLengths= c(4, Inf)))
complete.tdm
```

```
## <<TermDocumentMatrix (terms: 16880, documents: 10876)>>
## Non-/sparse entries: 76219/183510661
## Sparsity           : 100%
## Maximal term length: 49
## Weighting          : term frequency (tf)
```

Podemos observar que tenemos 16880 *terms*, esto quiere decir que tenemos 16880 palabras diferentes en nuestro Corpus. Lo cual es una cantidad considerable de vocabulario, pero no esperaríamos otra cosa de una red social como Twitter. La palabra más larga contiene 49 caracteres.

Usaremos la función **removeSparseItems** para depurar nuestra **Term Document Matrix** de aquellas palabras que aparecen con muy poca frecuencia, es decir, son dispersas. Porqué 16880 palabras son demasiadas palabras y es posible que no podamos entrenar nuestro modelo debido a restricciones computacionales.

Esta función requiere que especifiquemos el argumento **sparse**, que puede asumir valores de 0 a 1. Este valor representa la dispersión de las palabras que queremos conservar. Si lo fijamos muy alto (cerca de 1, pero no 1), conservaremos muchas palabras, casi todas, pues estamos indicando que queremos conservar palabras, aunque sean muy dispersas. Naturalmente, ocurre lo opuesto si fijamos este valor muy bajo (cerca de 0, pero no 0), pudiendo incluso quedarnos sin ninguna palabra, si las palabras en nuestros textos son dispersas en general.

En este caso, se decide fijarlo en *.9975*, conservando las palabras que aparecen en al menos el 0.25% de las observaciones.

```
complete.tdm <- removeSparseTerms(complete.tdm, sparse = .9975)
complete.tdm
```

```
## <<TermDocumentMatrix (terms: 582, documents: 10876)>>
## Non-/sparse entries: 31214/6298618
## Sparsity           : 100%
## Maximal term length: 17
## Weighting          : term frequency (tf)
```

De 16880 palabras que teníamos, nos hemos quedado con 582, lo cual reduce en gran medida la dificultad y complejidad del problema de *alta dimensionalidad*, lo cual es deseable. La palabra más larga contiene 17 caracteres.

Transformamos nuestra **Term Document Matrix** a un objeto de tipo **matrix** para así poder comprobar si nuestros datos aún contienen valores perdidos con la función **which**.

```
## integer(0)
```

Los datos ya no contienen valores perdidos.