Flujo de análisis en clasificación supervisada

Métodos supervisados

Laura Rodríguez Navas

Septiembre 2020

Contents

Comenzamos cargando los paquetes necesarios.

```
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

Para 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). La variable clase es binaria, así que, vamos a aprender un modelo de clasificación binaria. El objetivo de este modelo será predecir si dado un tweet, este tweet 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 métrica de evaluación esperada por la competición es F1. Y se calcula de la siguiente manera:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

donde:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

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 (ficheros **train.csv** y **test.csv**).

A continuación, cargaremos el conjunto de datos de train y test, nombrando los valores perdidos como **NA** para que los podamos tratar más adelante, y mostraremos sus dimensiones.

```
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
```

El conjunto de datos de train contiene 7613 instancias y el conjunto de datos de test contiene 3263 instancias. Cada instancia de estos conjuntos contiene la siguiente información:

- **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 es sobre un desastre real (1) o no (0).

```
str(train, width = 85, strict.width = "cut")
```

```
## '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 ...
```

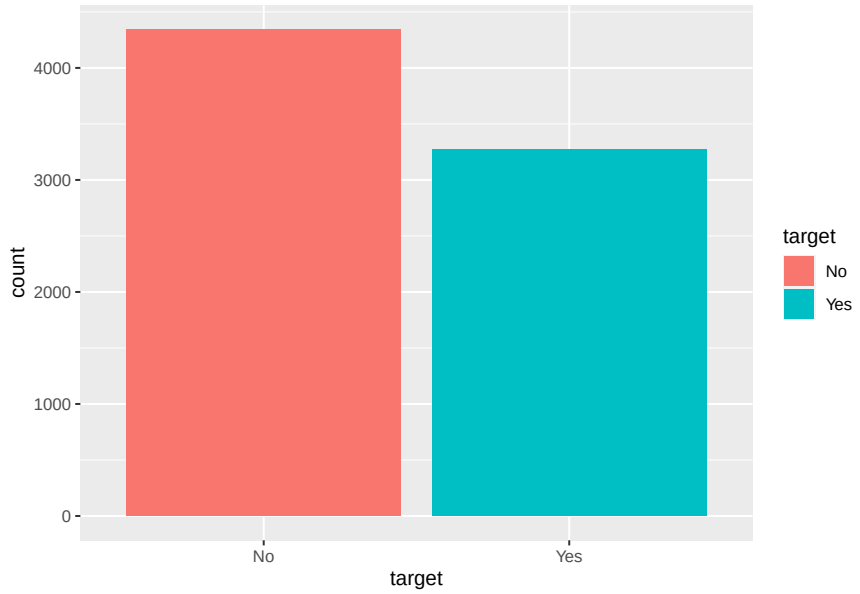
```
str(test, width = 85, strict.width = "cut")
```

```
## '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*

Categorizamos la variable a predecir, ya que inicialmente es de tipo entero, y observamos su distribución.

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



La distribución no está muy sesgada y vemos que hay menos tweets que se refieren a desastres reales. La distribución de la variable a predecir está relativamente equilibrada, donde el 43% de las observaciones son desastrosas y el 57% no.

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

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

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

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

Tampoco presenta un problema notable de *desbalanceo de clase* porqué contamos con muchas muestras del caso minoritario.

Variable *keyword*

La variable **keyword** representa una palabra representativa de cada tweet, se muestran las primeras 10.

```
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
```

Ahora veremos si la asociación de cada **keyword** con un sentimiento indica una relación con la variable a predecir. Para ello realizaremos un análisis de sentimientos de cada palabra clave.

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 real en relación a un desastre.*

Para ello usaremos los paquetes **syuzhet**, **ggcorrplot** y **doParallel**.

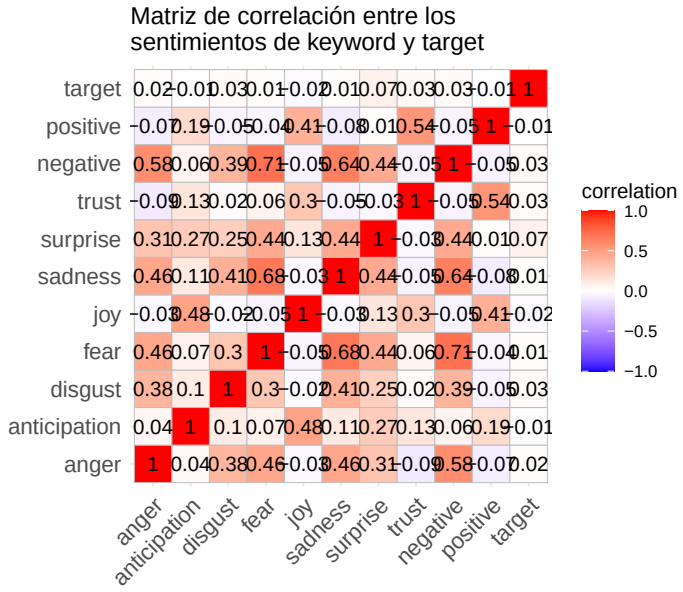
- **syuzhet** cuenta con la función **get_nrc_sentiment** que calcula la presencia de los diferentes sentimientos dado un conjunto de textos. Los parámetros de esta función son:
  - **char_v**. Un vector de caracteres que en este caso contiene todas las palabras clave.
  - **language**. Define el lenguaje.
  - **cl**. Para análisis paralelo. Es opcional, pero en este caso lo usaremos porqué hay muchas palabras clave.
- **ggcorrplot** muestra una visualización gráfica de una matriz de correlación usando *ggplot2*.
- **doParallel** proporciona una computación paralela. Los parámetros de esta función son:
  - **makePSOCKcluster**. Crea un clúster de sockets paralelos.
  - **registerDoParallel**. Registra el número de *cores* que usará el clúster creado.
  - **stopCluster**. Detiene la computación paralela.

Análisis de correlaciones entre **keyword** y **target**:

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

emocion.df <- emocion.df %>% data.frame(target = train$target)
emocion.df$target <- as.numeric(emocion.df$target)

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



Al observar la matriz de correlaciones, se observa una correlación nula con cada uno de los sentimientos. Esto hace que esta variable explicativa no sea buena para hacer una predicción.

Variable *location*

La variable **location** representa la ubicación desde donde se generaron los tweets, se muestran las primeras 10.

```
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())
```

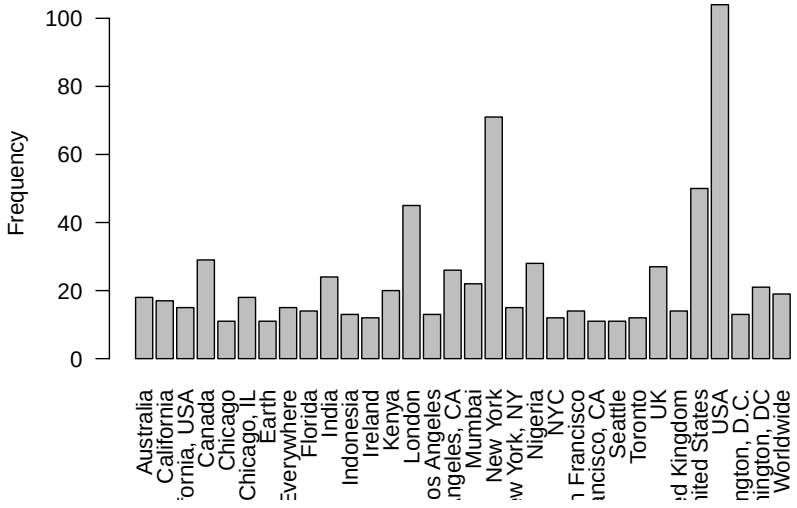
```
##      n
## 1 3342
```

En total hay 3342 ubicaciones. A continuación, mostramos las ubicaciones más frecuentes:

```
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")
```



Del total de ubicaciones (3342), la mayoría de ellas cuenta con menos de 10 observaciones. Esto hace que esta variable explicativa tampoco sea buena para hacer una predicción.

Variable *text*

Hemos considerado que las variables explicativas **keyword** y **location** no son buenas para hacer una predicción, así que nos centraremos en la variable **text**.

Llegados a este punto unimos los conjuntos de train y test (*7613 + 3263 observaciones*) para poder extraer los sentimientos más adelante.

```r
complete_df <- bind_rows(train, test)
dim(complete_df)
```

```
## [1] 10876     5
```

Echamos un vistazo más de cerca a las variables del nuevo conjunto de datos **complete_df**.

```r
summary(complete_df)
```

```
##        id          keyword            location            text
##  Min.   :    0   Length:10876       Length:10876       Length:10876
##  1st Qu.: 2719   Class :character   Class :character   Class :character
##  Median : 5438   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 5438
##  3rd Qu.: 8156
##  Max.   :10875
##   target
##  No  :4342
##  Yes :3271
##  NA's:3263
##
##
##
```

La variable **id** es solo un identificador único y la eliminaremos.

```r
complete_df$id <- NULL
```

Observamos si existen valores perdidos.

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

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

Las variables explicativas **keyword** y **location** contienen valores perdidos. Sobretodo hay una gran cantidad de tweets, para los cuales falta su ubicación. No existen valores perdidos para la variable explicativa **text**, tampoco para la variable a predecir **target**. Los 3263 valores perdidos de la variable a predecir provienen del conjunto de datos de test. Nos ocuparemos de los valores perdidos más adelante.

Parece que la variable explicativa **text** es una buena elección para una buena predicción y basaremos los siguientes pasos en ella.

Procesamiento de texto

Como en todo procesamiento de lenguaje natural, realizaremos el procesamiento de un conjunto de textos. En este caso realizaremos un procesamiento de los textos de los tweets y los prepararemos para el modelado. Comencemos por crear un corpus de los mensajes de texto de los tweets. Para ello usaremos la función **Corpus** del paquete **tm**, que creará nuestro corpus a partir de un vector de textos. La función **VectorSource** interpretará cada mensaje de texto de los tweets como un elemento del 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))
```

Durante el procesamiento de texto seguiremos la transformación de un mensaje de tweet específico para ver como se modifica a medida que avanzamos en el procesamiento de texto. Este mensaje es:

```
paste0(myCorpus[[400]])
```

```
## [1] "Jewish leaders prayed at the hospital where a Palestinian family is being treated after arson h
```

Dividimos el procesamiento de texto en 7 pasos.

1. Eliminar enlaces.

```
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeURL))
paste0(myCorpus[[400]])
```

```
## [1] "Jewish leaders prayed at the hospital where a Palestinian family is being treated after arson
```

Hemos eliminado: *http://t.co/Wf8iTK2KVx*.

La función **gsub** busca y reemplaza desde la primera hasta todas las coincidencias de un patrón (que normalmente representa una *regular expression*). La función **tm_map** es la encargada de aplicar las diferentes transformaciones de los textos al corpus creado.

2. Convertir a minúsculas.

```
myCorpus <- tm_map(myCorpus, content_transformer(stri_trans_tolower))
paste0(myCorpus[[400]])
```

```
## [1] "jewish leaders prayed at the hospital where a palestinian family is being treated after arson
```

3. Eliminar los nombres de usuario.

```
removeUsername <- function(x) gsub("@[^[:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeUsername))
paste0(myCorpus[[400]])
```

```
## [1] "jewish leaders prayed at the hospital where a palestinian family is being treated after arson
```

Hemos eliminado: *@huffpostrelig*.

4. Eliminar todo excepto el idioma y el espacio en inglés.

```
removeNumPunct <- function(x) gsub("[^[:alpha:][:space:]]*", "", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeNumPunct))
paste0(myCorpus[[400]])
```

```
## [1] "jewish leaders prayed at the hospital where a palestinian family is being treated after arson  v
```

No se observan cambios en en ejemplo.

5. Eliminar palabras irrelevantes (eliminación de redundancias).

```
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"))

myCorpus <- tm_map(myCorpus, removeWords, myStopWords)
paste0(myCorpus[[400]])
```

```
## [1] "jewish leaders prayed   hospital   palestinian family   treated  arson   "
```

Hemos eliminado: at the where a is being after via.

Las palabras irrelevantes que hemos eliminado se denominan *stop words o palabras vacías*. Cada idioma tiene sus propias palabras vacías. Como los textos están en inglés hemos eliminado los *stop words* que pertenecen al inglés usando la función **stopwords**, además hemos añadido aleatóriamente alguna de las palabras vacías 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/).

Las stop words o palabras vacías son todas aquellas palabras que carecen de un significado por si solas. Suelen ser artículos, preposiciones, conjunciones, pronombres, etc.

6. Eliminar palabras de una sola letra.

```
removeSingle <- function(x) gsub(" . ", " ", x)
myCorpus <- tm_map(myCorpus, content_transformer(removeSingle))
paste0(myCorpus[[400]])
```

```
## [1] "jewish leaders prayed hospital palestinian family treated  arson "
```

No se observan cambios en en ejemplo.

7. Eliminar espacios en blanco adicionales.

```
myCorpus <- tm_map(myCorpus, stripWhitespace)
paste0(myCorpus[[400]])
```

```
## [1] "jewish leaders prayed hospital palestinian family treated arson "
```

Terminamos con el procesamiento de texto. A continuación, crearemos dos *Term Document Matrix* (matriz que describe la frecuencia de las palabras que se producen en una colección de textos) para un análisis de sentimientos más detallado. Usaremos la función **TermDocumentMatrix** y dividiremos el corpus en dos, según el número de elementos de los conjuntos de datos train y test. Recordamos que el conjunto de datos de train contiene 7613 observaciones, y el conjunto de datos de test contiene 3263 observaciones. El parámetro **control** evalúa cada texto de la matriz, específicamente se evaluaran todas las palabras de cada texto (no aplicamos ningún filtro).

```
train_tdm <- TermDocumentMatrix(myCorpus[1:7613],
                                control= list(wordLengths= c(1, Inf)))
test_tdm <- TermDocumentMatrix(myCorpus[7614:10876],
                               control= list(wordLengths= c(1, Inf)))
train_tdm
```

```
## <<TermDocumentMatrix (terms: 14825, documents: 7613)>>
## Non-/sparse entries: 58707/112804018
## Sparsity           : 100%
## Maximal term length: 49
## Weighting          : term frequency (tf)
```

```
test_tdm
```

```
## <<TermDocumentMatrix (terms: 8966, documents: 3263)>>
## Non-/sparse entries: 25434/29230624
## Sparsity           : 100%
## Maximal term length: 35
## Weighting          : term frequency (tf)
```

La matriz para los datos de train contiene 14825 palabras y la matriz para los datos de test contiene 8966 palabras. Utilizaremos las matrices para ver las palabras más utilizadas en los tweets y predecir a partir de ellas.

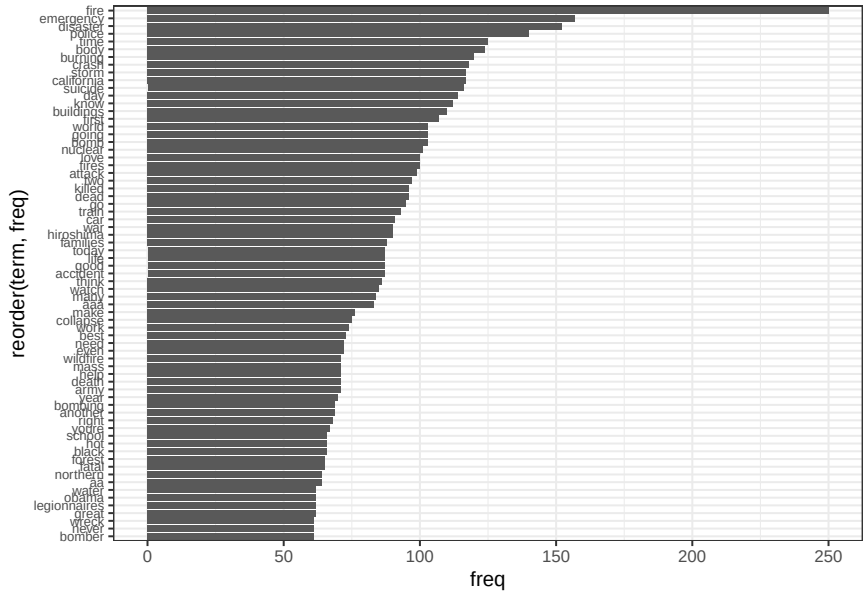
Palabras más frecuentes

- Palabras más frecuentes en los datos de train.

```
train.term.freq <- rowSums(as.matrix(train_tdm))
train.term.freq <- subset(train.term.freq, train.term.freq > 60)
freq_train_df <- data.frame(term = names(train.term.freq), freq= train.term.freq)
head(freq_train_df[order(-freq_train_df$freq), ])
```

```
##                term freq
## fire           fire  250
## emergency emergency  157
## disaster   disaster  152
## police       police  140
## time           time  125
## body           body  124
```

```
ggplot(freq_train_df, aes(reorder(term, freq), freq)) + theme_bw() +
  geom_bar(stat = "identity")  +
  coord_flip() +
  theme(axis.text.y = element_text(size=7))
```

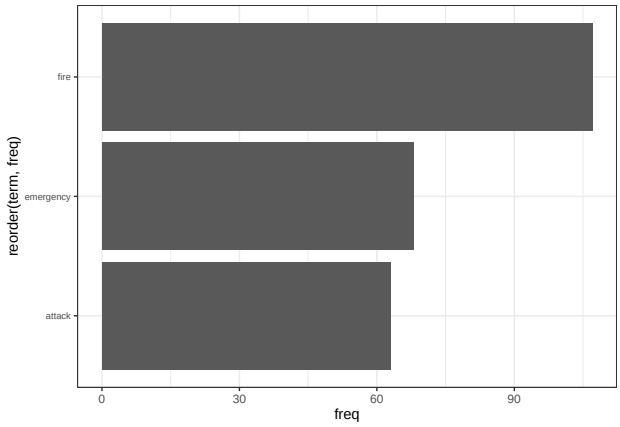


- Palabras más frecuentes en los datos de test

```
test.term.freq <- rowSums(as.matrix(test_tdm))
test.term.freq <- subset(test.term.freq, test.term.freq > 60)
freq_test_df <- data.frame(term = names(test.term.freq), freq= test.term.freq)
freq_test_df[order(-freq_test_df$freq), ]
```

```
##                term freq
## fire           fire  107
## emergency emergency   68
## attack       attack   63
```

```
ggplot(freq_test_df, aes(reorder(term, freq), freq)) + theme_bw() +
  geom_bar(stat = "identity")  +
  coord_flip() +
  theme(axis.text.y = element_text(size=7))
```



Las palabras *"fire"*, *"attack"* y *"emergency"* aparecen con alta frecuencia tanto en el conjunto de train como en el conjunto de test. Estas palabras son muy útiles para predecir un desastre real.

Modelado

Para la construcción de modelos, necesitamos construir una *Term Document Matrix* donde cada fila represente un texto y cada palabra única esté representada por una columna. Comencemos construyendo la matriz para nuestro corpus completo.

```
complete.tdm <- TermDocumentMatrix(myCorpus, control= list(wordLengths= c(4, Inf)))
complete.term.matrix <- as.matrix(t(complete.tdm))
dim(complete.term.matrix)
```

```
## [1] 10876 16891
```

Podemos ver que es una matriz dispersa con 10876 textos, es decir, tweets y 16880 palabras. Además, se generan algunos casos incompletos, que necesitaremos manejar antes de continuar con la construcción del modelo.

Sin embargo, si intentamos convertir esto para usar esta matriz para la construcción del modelo, es posible que no podamos entrenar nuestro modelo debido a restricciones computacionales. Necesitamos reducir la dimensión/características de la matriz.

Primero tratamos los casos incompletos.

```
incomplete.cases <- which(!complete.cases(complete.term.matrix))
complete.term.matrix[incomplete.cases,] <- rep(0.0, ncol(complete.term.matrix))
```

```
complete_irlba <- irlba(t(complete.term.matrix), nv = 150, maxit = 600)
complete_irlba$v[1:10, 1:5]
```

```
##               [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,] -0.000462870 -0.0003461531  9.263917e-06  1.074148e-04 -0.0006025571
##  [2,] -0.039611815  0.0191637843 -4.631903e-03 -4.704284e-03  0.0075975914
##  [3,] -0.002804300  0.0004191885 -8.592254e-06 -8.570421e-06 -0.0017859899
##  [4,] -0.006221545 -0.0006966814 -2.384731e-03 -2.622132e-04 -0.0034957244
##  [5,] -0.002844704  0.0003113251  1.691928e-04  2.684031e-05 -0.0016029824
```

```
##  [6,] -0.043686430  0.0177778044 -6.819700e-03 -4.737299e-03  0.0053901935
##  [7,] -0.010115221 -0.0248025873 -1.819646e-02 -2.408506e-04 -0.0006175213
##  [8,] -0.035196096  0.0178217283 -4.374870e-03 -4.425091e-03  0.0100228792
##  [9,] -0.009139736  0.0007828903 -4.421149e-06  2.046801e-04 -0.0070461856
## [10,] -0.001539314 -0.0001940795 -9.137939e-06  4.200374e-05 -0.0012366532
```

```
complete.svd <- data.frame(target = complete_df$target, complete_irlba$v)
train.df <- complete.svd[1:7613, ]
test.df <- complete.svd[7614:10876, -1]
dim(train.df)
```

```
## [1] 7613  151
```

```
dim(test.df)
```

```
## [1] 3263  150
```

```
names(train.df) <- make.names(names(train.df))
names(test.df) <- make.names(names(test.df))
```

```
set.seed(123)
cv.folds <- createMultiFolds(train.df$target, k=5, times=3)
cv.cntrl <- trainControl(method = "repeatecv", number = 5,
                         index = cv.folds, allowParallel = TRUE, returnResamp="final",
                         verboseIter=FALSE)
```

Regresión Logística

```
model_glmnet <- train(target ~ ., data=train.df,
                      method="glmnet",
                      metric="Accuracy",
                      trControl=cv.cntrl,
                      tuneLenght=15)
model_glmnet
```

```
## glmnet
##
## 7613 samples
##  150 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling:
## Summary of sample sizes: 6090, 6090, 6090, 6091, 6091, 6090, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda       Accuracy   Kappa
##   0.10   0.000214028  0.7790176  0.5386466
##   0.10   0.002140280  0.7788427  0.5378099
##   0.10   0.021402805  0.7766971  0.5302009
```

```
##   0.55   0.000214028  0.7791052  0.5387800
##   0.55   0.002140280  0.7787990  0.5370526
##   0.55   0.021402805  0.7567752  0.4811028
##   1.00   0.000214028  0.7790177  0.5385999
##   1.00   0.002140280  0.7778359  0.5343675
##   1.00   0.021402805  0.7291026  0.4146283
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.55 and lambda = 0.000214028.
```

Random Forest

```
model_rf <- train(target ~ ., data=train.df,
                      method="rf",
                      metric="Accuracy",
                      trControl=cv.cntrl,
                      tuneGrid=expand.grid(mtry=c(3, 4, 5, 7)))
model_rf
```

```
## Random Forest
##
## 7613 samples
##  150 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling:
## Summary of sample sizes: 6090, 6090, 6090, 6091, 6091, 6090, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   3     0.7738498  0.5308840
##   4     0.7736742  0.5306054
##   5     0.7741565  0.5320462
##   7     0.7750317  0.5338612
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 7.
```

Gradient Boosting

```
cv.grid <- expand.grid(interaction.depth=c(1, 2),
                       n.trees=100,
                       shrinkage=c(0.001, 0.01, 0.1),
                       n.minobsinnode=c(2, 5, 25))

model_gbm <- train(target ~ ., data=train.df,
                   method="gbm",
                   metric="Accuracy",
                   trControl=cv.cntrl,
```

```
                  tuneGrid=cv.grid,
                  distribution="adaboost",
                  verbose=FALSE)
model_gbm
```

```
## Stochastic Gradient Boosting
##
## 7613 samples
##  150 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling:
## Summary of sample sizes: 6090, 6090, 6090, 6091, 6091, 6090, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  Accuracy   Kappa
##   0.001      1                   2              0.5703402  0.0000000
##   0.001      1                   5              0.5703402  0.0000000
##   0.001      1                  25              0.5703402  0.0000000
##   0.001      2                   2              0.5703402  0.0000000
##   0.001      2                   5              0.5703402  0.0000000
##   0.001      2                  25              0.5703402  0.0000000
##   0.010      1                   2              0.6862809  0.3167993
##   0.010      1                   5              0.6856678  0.3152431
##   0.010      1                  25              0.6864123  0.3171660
##   0.010      2                   2              0.7105815  0.3835634
##   0.010      2                   5              0.7115881  0.3857311
##   0.010      2                  25              0.7118941  0.3865345
##   0.100      1                   2              0.7428076  0.4629660
##   0.100      1                   5              0.7437263  0.4649426
##   0.100      1                  25              0.7432891  0.4641821
##   0.100      2                   2              0.7558553  0.4936147
##   0.100      2                   5              0.7560303  0.4941869
##   0.100      2                  25              0.7574753  0.4969251
##
## Tuning parameter 'n.trees' was held constant at a value of 100
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 25.
```

K-Nearest Neighbor (kNN)

```
cv.grid <- expand.grid(k=1:3)

model_knn <- train(target ~ ., data=train.df,
                   method="knn",
                   metric="Accuracy",
                   trControl=cv.cntrl,
                   tuneGrid=cv.grid)
model_knn
```

```
## k-Nearest Neighbors
##
## 7613 samples
##  150 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling:
## Summary of sample sizes: 6090, 6090, 6090, 6091, 6091, 6090, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   1  0.7218769  0.4254800
##   2  0.7146533  0.4092179
##   3  0.7417117  0.4585926
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

SVM

```
cv.grid <- expand.grid(sigma=c(0.001, 0.01, 0.1, 0.5, 1),
                       C=c(1, 20, 50, 100))

model_svm <- train(target ~ ., data=train.df,
                     method="svmRadial",
                     metric="Accuracy",
                     trControl=cv.cntrl,
                     tuneLenght=15)
model_svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 7613 samples
##  150 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling:
## Summary of sample sizes: 6090, 6090, 6090, 6091, 6091, 6090, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7703040  0.5199771
##   0.50  0.7760834  0.5314795
##   1.00  0.7802865  0.5401797
##
## Tuning parameter 'sigma' was held constant at a value of 0.00820416
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.00820416 and C = 1.
```

Comparación de modelos

Métrica de validación

Evaluación de modelos mediante resampling

```
models <- list(KNN=model_knn, logistic=model_glmnet,
               RF=model_rf, boosting=model_gbm, SVMRadial=model_svm)

resamps <- resamples(models)
summary(resamps)
```

```
##
## Call:
## summary.resamples(object = resamps)
##
## Models: KNN, logistic, RF, boosting, SVMRadial
## Number of resamples: 15
##
## Accuracy
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## KNN       0.7220762 0.7327643 0.7385020 0.7417117 0.7490966 0.7715036    0
## logistic  0.7491793 0.7749671 0.7792378 0.7791052 0.7894911 0.7971110    0
## RF        0.7582129 0.7674868 0.7713535 0.7750317 0.7829941 0.7938280    0
## boosting  0.7275115 0.7500813 0.7614980 0.7574753 0.7645321 0.7780696    0
## SVMRadial 0.7655942 0.7724137 0.7793828 0.7802865 0.7833224 0.8030204    0
##
## Kappa
##                Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## KNN       0.4154222 0.4389489 0.4549650 0.4585926 0.4742860 0.5255358    0
## logistic  0.4772822 0.5281988 0.5391470 0.5387800 0.5599009 0.5797503    0
## RF        0.4989282 0.5169593 0.5258828 0.5338612 0.5492779 0.5765744    0
## boosting  0.4361559 0.4797778 0.5024020 0.4969251 0.5127961 0.5398718    0
## SVMRadial 0.5098819 0.5217476 0.5350176 0.5401797 0.5474368 0.5917576    0
```

Predicción y resultados

```
pred_svm <- predict(model_svm, test.df, type="raw")
```

```
submission <- read.csv("sample_submission.csv")
submission$target <- ifelse(pred_svm=="No", 0, 1)
head(submission)
```

```
##   id target
## 1  0      0
## 2  2      1
## 3  3      1
## 4  9      0
## 5 11      1
## 6 12      1
```

```r
write.csv(submission, "submission.csv", row.names=FALSE)
```

Conclusiones

Bibliografía

https://www.cienciadedatos.net/documentos/41_machine_learning_con_r_y_caret