

ff

TFM - Detección de lenguaje de odio en los medios online mediante el algoritmo SVM

Master Machine Learning UNED – Curso 2024/2025

Autor: Francisco Camacho

Sumario

1. Introducción.....	4
1. Motivación.....	4
2. Alternativas y justificación de la decisión.....	5
2. Fase 1 - Recolección de datasets etiquetados.....	7
1. Proyecto Hatemedia.....	7
2. HuggingFace.....	7
3. Kaggle.....	7
4. Resultado.....	8
3. Fase 2 – Entrenamiento de SVM.....	9
1. Entrenar SVM con cada uno de los datasets.....	9
1. Dataset Kaggle.....	9
Paso 1: Obtención del dataset y procesado preliminar.....	9
Paso 2: Procesado del dataset.....	10
Paso 3: Entrenamiento del modelo y evaluación.....	13
2. Dataset HuggingFace.....	17
Paso 1: Obtención del dataset y procesado preliminar.....	17
Paso 2: Procesado del dataset.....	19
Paso 3: Entrenamiento del modelo y evaluación.....	20
3. Dataset Hatemedia.....	22
Paso 1: Obtención del dataset y procesado preliminar.....	22
Paso 2: Procesado del dataset.....	23
Paso 3: Entrenamiento del modelo y evaluación.....	25
2. Creación de un dataset total y entrenamiento de SVM.....	29
1. Creación del dataset total.....	30
2. Creación del dataset de 100k observaciones y entrenamiento.....	31
Procesado del dataset.....	33
Entrenamiento del modelo y evaluación.....	35
3. Entrenamiento con el dataset total.....	41
Procesado del dataset.....	41
Entrenamiento del modelo y evaluación.....	43
3. Comparación de resultados y conclusiones.....	47
4. Fase 3 – Clasificación de comentarios nuevos no etiquetados.....	49
1. Obtención de comentarios nuevos no etiquetados.....	49
2. Clasificación de comentarios nuevos no etiquetados con modelos ya entrenados.....	50
1. Leer CSVs con los comentarios de los lectores.....	50
2. Carga de los modelos guardados en la fase 2.....	51
3. Clasificación de comentarios no etiquetados.....	51
4. Evaluación de los resultados.....	53
3. Clasificación de comentarios nuevos con modelos entrenados con ellos.....	55

4. Mejora de los datasets con comentarios nuevos.....	57
1. Procesado de los datasets.....	57
2. Entrenamiento del SVM y evaluación del resultado.....	59
5. Tabla comparativa de resultados y conclusiones.....	62
5. ANEXO 1 – Librería LIBLINEAR, funciones de pérdida y regularización.....	65
6. ANEXO 2 – Código R de las funciones comunes.....	67
7. ANEXO 3 – Repositorio código en GitHub.....	72
8. ANEXO 4 - Bibliografía.....	73

1. Introducción

El objetivo de este TFM del Master de Machine Learning de la UNED (curso 2024/2025) es realizar un prototipo de un sistema que pueda clasificar como odio/no odio los comentarios escritos por los lectores de las ediciones online de periódicos españoles.

Evidentemente, una vez entrenado el modelo, éste puede ser utilizado para clasificar comentarios independientemente de si estos comentarios provienen de un periódico online, o de algún foro o red social.

(De hecho, los comentarios de los datasets encontrados tienen diversas fuentes, no solo periódicos online).

1. Motivación

Cuando internet y las redes sociales surgieron, la posibilidad de comunicarse desde cualquier parte del mundo con personas en cualquier otro lugar del planeta de manera instantánea se hizo realidad (1). Cuando esas redes sociales se convirtieron en una herramienta tan habitual en las sociedades actuales como lo hicieron el reloj o el teléfono (sobre todo el móvil) en su época, las interacciones entre personas se hicieron globales e instantáneas .. y múltiples. El grado de interacciones creció de manera exponencial.

Con ello creció por tanto también exponencialmente la capacidad de difundir toda clase de mensajes, tanto positivos como negativos. Entre estos mensajes negativos el autor de este trabajo puede distinguir dos grandes categorías: bulos (“fake news” en Inglés) y mensajes de odio (“hate speech”).

Durante la fase de exploración y de búsqueda de información para decidirme sobre cuál de estos dos problemas realizar el TFM, ví que los mensajes de odio son un problema efectivamente global: he encontrado papers hechos en universidades de Perú, Nigeria, India, Etiopía, España, Singapur, Australia ... cuyos autores todos muestran una gran preocupación por este fenómeno.

(1) El concepto de **aldea global** fue acuñado por el sociólogo canadiense Herbert Marshall McLuhan en los años 60 del S. XX. McLuhan se refería a la influencia que ejercían sobre las sociedades de todo el mundo el cine, la radio y la televisión. Este concepto recuerdo se volvió popular otra vez cuando internet y las redes sociales empezaron a expandirse y popularizarse hace 25-30 años. Su efecto es increíblemente mayor en mi opinión. Pero McLuhan falleció en 1980, cuando internet todavía estaba en un estado embrionario.

https://es.wikipedia.org/wiki/Aldea_global

2. Alternativas y justificación de la decisión

Una vez decidido el “qué”, había que decidir el “cómo”. Consideré primeramente tanto Naive Bayes como SVM. Pero tras ver varios papers de estudios y comparativas ya hechos, más la experiencia ganada al hacer las tareas previas de este máster, me decidí por SVM. (Debido a ventajas como lo poco que tiende SVM al sobreajuste).

Inicialmente probé con la librería LIBSVM (paquete e1071 de R) y con caret. Pero con los datasets encontrados (especialmente con el de Hatemedia), tras realizar algunas pruebas ví que los tiempos de ejecución eran muy grandes en el equipo disponible para entrenar los modelos.

En el documento “A Practical Guide to Support Vector Classification” y en la web <https://www.csie.ntu.edu.tw/~cjlin/liblinear/> , encontré rápidamente una muy buena alternativa a LIBSVM para este problema: LIBLINEAR (de los mismos autores de LIBSVM).

"When to use LIBLINEAR but not LIBSVM

There are some large data for which with/without nonlinear mappings gives similar performances. **Without using kernels**, one can quickly train a much larger set via a linear classifier. **Document classification** is one such application."

(Negritas así en el original).

Decidí por tanto utilizar la librería LIBLINEAR en lugar de LIBSVM.

[

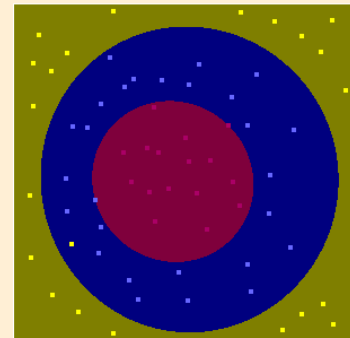
NOTA:

Una pequeña muestra de las posibilidades de la librería LIBSVM.

<https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

Graphic Interface

Here is a simple applet demonstrating SVM classification and regression.
Click on the drawing area and use ``Change`` to change class of data. Then use ``Run`` to see the results.



options:

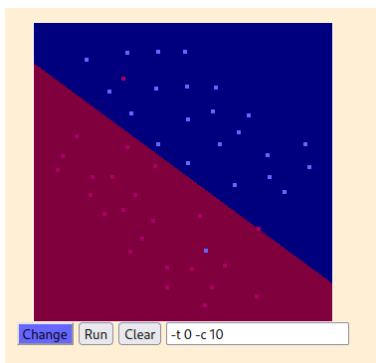
-s svm_type : set type of SVM (default 0)

- 0 -- C-SVC
- 1 -- nu-SVC
- 2 -- one-class SVM
- 3 -- epsilon-SVR
- 4 -- nu-SVR

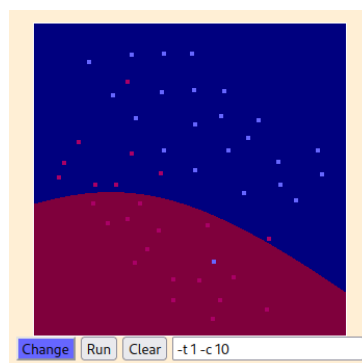
-t kernel_type : set type of kernel function (default 2)

- 0 -- linear: $u'v$
- 1 -- polynomial: $(\gamma u'v + \text{coef0})^{\text{degree}}$
- 2 -- radial basis function: $\exp(-\gamma |u-v|^2)$
- 3 -- sigmoid: $\tanh(\gamma u'v + \text{coef0})$

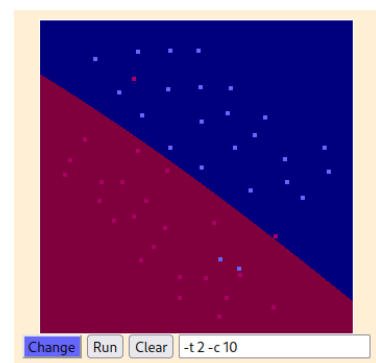
kernel lineal:



kernel polinomial:



kernel RBF:



]

2. Fase 1 - Recolección de datasets etiquetados

1. Proyecto Hatemedia

El proyecto Hatemedia ha sido elaborado por la UNIR, Universidad Internacional de la Rioja:

“Hatemedia: análisis y monitoreo del discurso de odio en los medios digitales en España.

Hatemedia es un espacio donde se muestran los diferentes proyectos realizados o que han hecho parte la Universidad Internacional de La Rioja, alrededor del estudio de las expresiones de odio en las redes sociales y medios digitales en España.”

URL: <https://hatemedia.es/>

GitHub: <https://github.com/esaidh266/Hate-Speech-Library-in-Spanish>

Fichero: <https://doi.org/10.6084/m9.figshare.26085700.v1>

En esta última dirección se puede acceder al fichero y encontrar la descripción del mismo:

"El dataset que se ha utilizado para el entrenamiento de los modelos de algoritmos de odio/no odio, intensidad y tipo. El datasets consta de mensajes (noticias y comentarios), publicados por en usuarios asociados a El Mundo, ABC, La Vanguardia, El País y 20 Minutos, en Twitter y sus portales webs, durante el mes de enero de 2021."

"Dataset desarrollado en el marco del proyecto Proyecto Hatemedia (PID2020-114584GB-I00), financiado por MCIN/AEI /10.13039/501100011033, con el apoyo de la empresa colaboradora [Possible Inc.](#)"

Este dataset es el más grande de los encontrados, y está ya parcialmente procesado (han sido eliminadas las "stopwords", los números ...)

2. HuggingFace

En la web de esta empresa del sector de la IA y el machine learning he encontrado un super dataset de mensajes de odio en español elaborado con 5 datasets (2 de España -HateEval y Haternet-, otros 2 de Chile y uno de México).

URL: <https://huggingface.co/>

Fichero: <https://huggingface.co/datasets/manueltonneau/spanish-hate-speech-superset>

3. Kaggle

Kaggle es una plataforma y comunidad de expertos y practicantes de IA y machine learning. Aquí he encontrado un dataset multiidioma de mensajes de odio.

URL: <https://www.kaggle.com/>

Fichero: <https://www.kaggle.com/datasets/wajidhassanmoosa/multilingual-hatespeech-dataset>

Este dataset viene separado en un dataset de entrenamiento, y un dataset de test.

Cada uno de estos datasets tiene sus particularidades. El primer dataset, del proyecto Hatemedia es de un tamaño bastante considerable (574272 observaciones), muy desbalanceado, con todos los mensajes etiquetados como odio al final. Además, el dataset encontrado ya está procesado parcialmente. Los comentarios vienen con el siguiente formato*

id	comentario	label
3	tambien,salar,ayuso,sumario,caray,parecer,ves,ayuso,sopa,	0
4	peperar,celula,sitio,	0
5	traer,flojo,resultado,señora,dar,talla,aludido,	0

Los datasets de Kaggle al ser multiidioma, tienen que ser preprocesados para sacar solo los mensajes en castellano.

En definitiva se trata de obtener 4 datasets con el mismo formato, para poder combinarlos en uno solo si así se desea.

* Vienen más columnas, pero solo "comentario" y "label" son necesarias.

4. Resultado

El resultado de este primer procesado son 4 dataframes, todos con la misma estructura (dos columnas: "post" y "label").

Environment	History	Connections	Git	Tutorial
<div> <div>Import Dataset</div> <div>702 MiB</div> <div>List</div> </div>				
R - Global Environment				
Data				
odio	11064 obs. of 2 variables			
odio_hatemedia_raw	574272 obs. of 2 variables			
<div> <div>\$ post : chr "real,madrid,puesto,punto,final,adaduro,coipo,rey,primero,escalon,zidane,caer,alcoyano...</div> <div>\$ label: num 0 0 0 0 0 0 0 0 0 0 ...</div> </div>				
odio_huggingface_raw	29855 obs. of 2 variables			
<div> <div>\$ post : chr "Eran tan pero tan feministas que invisibilizaban constantemente a las trabajadoras se...</div> <div>\$ label: num 0 0 1 0 0 0 0 0 0 0 ...</div> </div>				
odio_kaggle_raw	219981 obs. of 4 variables			
odio_kaggle_raw_ES	11180 obs. of 2 variables			
<div> <div>\$ post : chr ". Puigdemont no volverá a Cataluña sin \"garantías\" de que se permitirá un Govern in...</div> <div>\$ label: num 0 0 0 0 1 1 0 1 1 1 ...</div> </div>				
odio_kaggle_raw_Test_ES	1243 obs. of 2 variables			
<div> <div>\$ post : chr "Tengo una amiga que dice Machete al machote. También tiene un corazón con la palabra ...</div> <div>\$ label: num 0 0 0 0 1 0 0 1 0 0 ...</div> </div>				
params	List of 3			

El total de observaciones "raw" es:

```
nrow(odio_hatemedia_raw)+nrow(odio_huggingface_raw)+nrow(odio_kaggle_raw_ES)
+nrow(odio_kaggle_raw_Test_ES)
## [1] 616550
```

Creo que se trata de una cantidad considerable de comentarios etiquetados para poder entrenar de manera adecuado un modelo SVM.

3. Fase 2 – Entrenamiento de SVM

1. Entrenar SVM con cada uno de los datasets.

Dado que se han encontrado 3 datasets diferentes, se pretende averiguar si las características de los mismos (origen, tamaño ...) tienen alguna influencia en la calidad del SVM que se puede obtener al entrenar este algoritmo con esos datasets.

1. Dataset Kaggle

Carga de las funciones comunes usadas con todos los datasets (su código se muestra en el Anexo).

```
source('hate_speech_common.R')
```

Carga de paquetes que son necesarios para diversas funciones.

```
load_libraries()

## [1] "Loading libraries:"
## [1] "Loading tm ..."
## [1] "Loading SnowballC ..."
## [1] "Loading textclean ..."
## [1] "Loading caret ..."
## [1] "Loading Liblinear ..."
## [1] "All libraries loaded."
```

Paso 1: Obtención del dataset y procesamiento preliminar

```
# import the CSV file
odio_kaggle_raw <- read.csv(file.path("dataset_03_kaggle.csv"), sep=";")
odio_kaggle_raw <- odio_kaggle_raw[-1] # -> 11180 obs. (only spanish) of 2 variables
```

Estructura del dataset:

```
str(odio_kaggle_raw)

## 'data.frame':    11180 obs. of  2 variables:
## $ post : chr  ". Puigdemont no volverá a Cataluña sin \"garantías\" de que se permitirá un
Govern independentista? Sueñas puig"| __truncated__ "La gente que escribe como los indios me
mata. " "Desde cuándo eres tan negra? JAJAJAJA" "JACKIE Y HYDE MERECIAN ESTAR JUNTOS LA PUTA MADRE
QUIERO ROMPER TODO" ...
## $ label: int  0 0 0 0 1 1 0 1 1 1 ...
```

La columna “label” es de tipo int. Ya que se trata en realidad de una variable categórica 0/1, es conveniente transformarla en un factor:

```
#Convert class into a factor
odio_kaggle_raw$label <- factor(odio_kaggle_raw$label)
```

Examinamos el resultado:

```
table(odio_kaggle_raw$label)
##      0      1
## 7365 3815

prop.table(table(odio_kaggle_raw$label))
##      0      1
## 0.6587657 0.3412343
```

Este dataset está desbalanceado (aunque no de una manera tan exagerada como los dos siguientes).

```
head(odio_kaggle_raw)

##
post
## 1 . Puigdemont no volverá a Cataluña sin "garantías" de que se permitirá un Govern
independentista? Sueñas puig, nada mas pisar Spain creo iras para Parla de estremera
. . . . .
## 2 La gente que escribe como los indios me mata.
## 3 Desde cuándo eres tan negra? JAJAJAJA
## 4 JACKIE Y HYDE MERECIAN ESTAR JUNTOS LA PUTA MADRE QUIERO ROMPER TODO
## 5 desnudas provocáis al igual que un hombre, simplemente porque es un instinto humano.
## 6 "España devuelve a Marruecos a los 116 inmigrantes que saltaron valla de Ceuta" Por una vez (y
sin que sirva de precedente) aplaudo la decisión de Pedro Sánchez. No queremos negros delincuentes
en nuestro país. Venga ¡A vuestra putta casa!

##      label
## 1         0
## 2         0
## 3         0
## 4         0
## 5         1
## 6         1

tail(odio_kaggle_raw)

##
post
## 11175 Por eso si no están al nivel los criticaré, en ésta cuenta no estamos para comerle la polla
a nadie sin que lo merezca
## 11176 Ahora viene cuando me acusas de llamarte facha, no?
## 11177 homóforo,vale,mientras juegues bien,no importa? ¿Eres gay? A la calle,no quiero "maricones"
en mi equipo.
## 11178 Soy ese subnormal que habla con sus propios OC, sí.
## 11179 El PP catalán vuelve al mus francés, o al tute cabrón que es más divertido.
## 11180 Típico de minitas, si una chabona sube una foto con poca ropa o en bikini es una trola,puta
y que quiere provocar pero si lo hace ella está todo bien jajaja ni dos dedos de frente tienen
algunas

##      label
## 11175     0
## 11176     0
## 11177     0
## 11178     0
## 11179     0
## 11180     1
```

Paso 2: Procesado del dataset

Como primer paso, se van a eliminar las referencias a otros usuarios en los comentarios.

También, con ayuda de la librería cleantext examinamos los comentarios y haremos un primer procesado:

```
#check_text(odio_kaggle_raw$post) #output omitted for brevity

odio_kaggle <- preprocess_posts(odio_kaggle, odio_kaggle_raw)
## [1] "Removed references to users (@)."
```

```
## [1] "Removed non ascii and emoticons."
```

```
## [1] "Removed lines with empty posts."
```

Comprobamos el resultado:

```
odio_kaggle[1:5,1]

## [1] ". Puigdemont no volvera a Catalunya sin \"garantias\" de que se permitira un Govern independentista? Suenas puig, nada mas pisar Spain creo iras para Parla de estremera . . . . .\"
## [2] \"La gente que escribe como los indios me mata.\"
## [3] \"Desde cuando eres tan negra? JAJAJAJA\"
## [4] \"JACKIE Y HYDE MERECIAN ESTAR JUNTOS LA PUTA MADRE QUIERO ROMPER TODO\"
## [5] \"desnudas provocais al igual que un hombre, simplemente porque es un instinto humano.\"

#check_text(odio_kaggle$post)
```

Corpus

Creación del objeto corpus con todas los mensajes:

```
#create corpus
posts_corpus <- Vcorpus(VectorSource(odio_kaggle$post))

print(posts_corpus)
## <<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 11179
```

Limpieza de los textos del corpus:

```
system.time({
  posts_corpus_clean <- clean_corpus(posts_corpus)
})

## [1] \"#To lowercase\"
## [1] \"#Remove numbers\"
## [1] \"#Remove stopwords\"
## [1] \"#Remove punctuation signs\"
## [1] \"#Carry out the stemming\"
## [1] \"#Finally eliminate unneeded whitespace produced by previous steps\"

##      user  system elapsed
## 4.484    0.008    4.492
```

Se examina el corpus y no se aprecia nada extraño.

```
#View(posts_corpus_clean)
```

Tokenización

Finalmente se procede a la tokenización de los comentarios. Se obtiene una DTM:

```
system.time({
  posts_dtm <- DocumentTermMatrix(posts_corpus_clean)
})
##      user  system elapsed
## 1.331    0.008    1.339

posts_dtm

## <<DocumentTermMatrix (documents: 11179, terms: 19856)>>
## Non-/sparse entries: 133750/221836474
## Sparsity           : 100%
## Maximal term length: 93
## Weighting           : term frequency (tf)
```

Ahora hay que crear los conjuntos de entrenamiento y de test.

Dado que este dataset ya viene separado en dos conjuntos, lo que hacemos es cargar el dataset de test:

```
# import the CSV file
odio_kaggle_test_raw <- read.csv(file.path("dataset_04_kaggle.csv"), sep=";") # -> 1243 obs. (only
spanish) of 3 variables
odio_kaggle_test_raw <- odio_kaggle_test_raw[-1]
```

Lo procesamos de igual manera que el de entrenamiento:

```
#Convert label into a factor
odio_kaggle_test_raw$label <- factor(odio_kaggle_test_raw$label)

#process df test
odio_kaggle_test <- preprocess_posts(odio_kaggle_test, odio_kaggle_test_raw)

## [1] "Removed references to users (@)."
```

```
## [1] "Removed non ascii and emoticons."
```

```
## [1] "Removed lines with empty posts."
```

```
#create corpus test
posts_test_corpus <- Vcorpus(VectorSource(odio_kaggle_test$post))

print(posts_test_corpus)
## <VCorpus>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 1243
```

Limpieza del corpus de test también:

```
system.time({
  posts_corpus_test_clean <- clean_corpus(posts_test_corpus)
})

## [1] "#To lowercase"
## [1] "#Remove numbers"
## [1] "#Remove stopwords"
## [1] "#Remove punctuation signs"
## [1] "#Carry out the stemming"
## [1] "#Finally eliminate unneeded whitespace produced by previous steps"

## user system elapsed
## 0.511 0.000 0.511
```

Tokenización:

```
posts_dtm_test <- DocumentTermMatrix(posts_corpus_test_clean)
```

Se necesita ahora obtener un listado con las palabras más utilizadas:

```
# Data preparation - creating indicator features for frequent words
# the function findFreqTerms() in the tm package takes a DTM and returns a character vector
containing words that appear at least a minimum number of times
posts_freq_words_train <- findFreqTerms(posts_dtm, 50)

#posts_freq_words_train

[1] "¿cómo" "¿por" "¿que" "¿qué" "¿te" "abajo" "abierta" "abr"
[9] "absoluta" "abuela" "abuelo" "abuso" "acá" "acaba" "acabar" "acabo"
[17] "acaso" "ácido" "acoso" "acto" "acuerdo" "acusacion" "acusado" "ademá"
...
[993] "pued" "pueda" "pueden" "puedo" "puerta" "puesto" "puigdemont" "punto"
[ reached 'max' / getOption("max.print") - omitted 344 entries ]
```

Y ahora utilizamos ese listado para limitar el número de columnas/features tanto del conjuntos de entrenamiento como del de test:

```
dim(posts_dtm)
## [1] 11179 19856
posts_dtm_freq_train <- posts_dtm[, posts_freq_words_train]
dim(posts_dtm_freq_train)
## [1] 11179 396

dim(posts_dtm_test)
## [1] 1243 4792
posts_dtm_freq_test <- posts_dtm_test[, posts_freq_words_train]
dim(posts_dtm_freq_test)
## [1] 1243 396
```

Paso 3: Entrenamiento del modelo y evaluación

```
# dtm -> matrix

posts_freq_train_mat <- as.matrix(posts_dtm_freq_train)
#posts_freq_train_mat <- as(as(as(posts_freq_train_mat, "dMatrix"), "generalMatrix"),
"RsparseMatrix")

posts_freq_test_mat <- as.matrix(posts_dtm_freq_test)
#posts_freq_test_mat <- as(as(as(posts_freq_test_mat, "dMatrix"), "generalMatrix"), "RsparseMatrix")

# training
system.time({
  liblinear_svm_model <- LiblinearR(data=posts_freq_train_mat, target=odio_kaggle$label, type=3)
})
##      user  system elapsed
##    0.243    0.004    0.247
```

Un parámetro muy importante en esta llamada, es “type”. Como se puede ver en la documentación, type puede tomar los siguientes valores:

Para clasificación multiclase

- 0 - regresión logística con regularización L2 (primal)
- 1 - "support vector classification" con regularización L2 y función de pérdida L2 (dual)
- 2 - "support vector classification" con regularización L2 y función de pérdida L2 (primal)
- 3 - "support vector classification" con regularización L2 y función de pérdida L1 (dual)
- 4 - Crammer and Singer "support vector classification" (multiclase).
- 5 - "support vector classification" con regularización L1 y función de pérdida L2
- 6 - regresión logística con regularización L1
- 7 - regresión logística con regularización L2 (dual)

Los autores de LIBLINEAR dicen también lo siguiente:

“Para elegir entre la regularización L1 y L2, recomendamos intentar primero L2 a no ser que el usuario necesite un modelo disperso”.

Se han hecho por tanto pruebas con las opciones 1, 2, 3 y 5. Dado que los tiempos fueron bastante parecidos, se ha elegido type = 3 (regularización L2 aunque tengamos matrices dispersas*, y función de pérdida L1) por ser el que mejor resultado ha dado.

*El dataset de Hatemedia, solo se ha podido procesar utilizando matrices dispersas. Los datasets de Kaggle y de HuggingFace, no ha habido ninguna diferencia entre usar una u otra clase de matrices.

```

# prediction
system.time({
  prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)
})
##      user      system elapsed
##    0.006      0.000      0.006

# Confusion matrix
confusionMatrix(reference = as.factor(odio_kaggle_test$label), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")

## Confusion Matrix and Statistics
##
##              Reference
## Prediction    0    1
##           0  737 208
##           1   82 216
##
##              Accuracy : 0.7667
##              95% CI   : (0.7422, 0.79)
##      No Information Rate : 0.6589
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa   : 0.4409
##
##  Mcnemar's Test P-Value : 2.132e-13
##
##      Sensitivity : 0.5094
##      Specificity : 0.8999
##      Pos Pred Value : 0.7248
##      Neg Pred Value : 0.7799
##      Precision : 0.7248
##      Recall : 0.5094
##      F1 : 0.5983
##      Prevalence : 0.3411
##      Detection Rate : 0.1738
##      Detection Prevalence : 0.2397
##      Balanced Accuracy : 0.7047
##
##      'Positive' Class : 1
##

```

Considero este primer resultado simplemente aceptable. La exactitud no llega al 80%, y kappa no llega a 0.5

Se obtiene el mismo resultado usando matrices densas y dispersas. Con este dataset, al ser de un tamaño pequeño, no hay problema en utilizar matrices densas. (Esto permite utilizar la función heurística para el cálculo del coste).

Con los otros datasets, no solo ha sido imprescindible utilizar matrices dispersas, sino también realizar la conversión de la DTM en matriz “a trozos”, ya que si no la sesión del RStudio “explotaba” por las limitaciones de memoria física del equipo.

Mejora del modelo

- Intentamos mejorar el modelo usando el coste calculado por la función `heuristicC`, como aconsejan los creadores de la librería `LiblineaR`:

```

# For a sparse matrix is not possible to use this heuristic function.
c <- tryCatch({
  cost <- heuristicC(posts_freq_train_mat) #error if posts_freq_train_mat is a sparse matrix
  cost
}, error = function(err) { # error handler
  print(paste("ERROR: ", err))
})

```

```

1 #default cost
})
cat("c: ",c)
## c: 0.360131

system.time({
  liblinear_svm_model_c <- Liblinear(data=posts_freq_train_mat, target=odio_kaggle$label, type=3,
cost=10) # the best result is obtained with cost = 10

  prediction_liblinear_c <- predict(liblinear_svm_model_c, posts_freq_test_mat)
})
##      user  system elapsed
##   1.147    0.012    1.159

confusionMatrix(reference = as.factor(odio_kaggle_test$label), data =
as.factor(prediction_liblinear_c$predictions), positive="1", mode = "everything")

## Confusion Matrix and Statistics
##
##              Reference
## Prediction    0    1
##              0 734 201
##              1  85 223
##
##              Accuracy : 0.7699
##              95% CI : (0.7455, 0.7931)
##              No Information Rate : 0.6589
##              P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.452
##
##  Mcnemar's Test P-Value : 1.046e-11
##
##              Sensitivity : 0.5259
##              Specificity : 0.8962
##              Pos Pred Value : 0.7240
##              Neg Pred Value : 0.7850
##              Precision : 0.7240
##              Recall : 0.5259
##              F1 : 0.6093
##              Prevalence : 0.3411
##              Detection Rate : 0.1794
##              Detection Prevalence : 0.2478
##              Balanced Accuracy : 0.7111
##
##              'Positive' Class : 1
##

```

Utilizando el coste calculado por la función heurística, en nuestro caso el resultado fue peor. En cambio, eligiendo un coste por encima de 1 (1.5, 2, 3, 5 ..), se mejora el resultado (solo ligeramente). El mejor resultado encontrado es con coste 10.

- Intentamos mejorar el modelo usando pesos, para favorecer la detección de la clase minoritaria:

```

# Define class weights
class_weights <- c("0" = 2, "1" = 3) # Assign higher weight to the minority class
system.time({

  liblinear_svm_model_weights <- Liblinear(data = posts_freq_train_mat, target = odio_kaggle$label,
type = 3, cost = 1, w1 = class_weights)

  prediction_liblinear_weights <- predict(liblinear_svm_model_weights, posts_freq_test_mat)
})

##      user  system elapsed
##   0.552    0.012    0.564

```



```

#Confusion matrix
confusionMatrix(reference = as.factor(odio_kaggle_test$label), data =
as.factor(prediction_liblinear_weights$predictions), positive="1", mode = "everything")
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##      0  666 145
##      1 153 279
##
##              Accuracy : 0.7603
##              95% CI : (0.7355, 0.7838)
##      No Information Rate : 0.6589
##      P-Value [Acc > NIR] : 5.43e-15
##
##              Kappa : 0.4691
##
##  Mcnemar's Test P-Value : 0.6851
##
##      Sensitivity : 0.6580
##      Specificity : 0.8132
##      Pos Pred Value : 0.6458
##      Neg Pred Value : 0.8212
##      Precision : 0.6458
##      Recall : 0.6580
##      F1 : 0.6519
##      Prevalence : 0.3411
##      Detection Rate : 0.2245
##      Detection Prevalence : 0.3475
##      Balanced Accuracy : 0.7356
##
##      'Positive' Class : 1
##

```

Probando diferentes combinaciones del parámetro coste y de los pesos, los mejores valores obtenidos rondan el 76-77% para la exactitud y un kappa del 0.47.

Tabla comparativa:

(Página siguiente)

	Original	Coste 10	
Matriz de confusión	Reference Pred. No Yes No 737 208 Yes 82 216	Reference Pred. No Yes No 734 201 Yes 85 223	
Exactitud	0.7667	0.7699	
Kappa	0.4409	0.4520	
F1	0.5983	0.6093	
MCC	0.4526		
	Coste 1, pesos 1/3	Coste 10, pesos 1/3	Coste 1, pesos 2/3
Matriz de confusión	Reference Pred. No Yes No 496 88 Yes 323 336	Reference Pred. No Yes No 490 83 Yes 329 341	Reference Pred. No Yes No 666 145 Yes 153 279
Exactitud	0.6693	0.6685	0.7603
Kappa	0.3511	0.3531	0.4691
F1	0.6205	0.6234	0.6519
MCC			0.4677

2. Dataset HuggingFace

Dado que el proceso es el mismo en un 99%, se muestra solo aquellas partes del código que son diferentes o se consideran de más interés.

Paso 1: Obtención del dataset y procesamiento preliminar

```
# import the CSV file
odio_huggingface_raw <- read.csv(file.path("dataset_02_huggingface.csv"), sep=";")
odio_huggingface_raw <- odio_huggingface_raw[-1] # -> ~29000 obs. of 2 variables
```

Convertimos la variable label en un factor:

```
#Convert class into a factor
odio_huggingface_raw$label <- factor(odio_huggingface_raw$label)
```

Examinamos el resultado:

```
table(odio_huggingface_raw$label)
##
##      0      1
## 22590  7265

prop.table(table(odio_huggingface_raw$label))
##
##      0      1
## 0.7566572 0.2433428
```

Este dataset también está bastante desbalanceado, como ya se vio en la primera fase (recolección de datasets) de este proyecto.

Examinamos algunos de los comentarios de este dataset:

```
head(odio_huggingface_raw)
##
post
## 1 Eran tan pero tan feministas que invisibilizaban constantemente a las trabajadoras sexuales,
haciéndole creer al mundo que eran incapaces de decidir y que cada vez que ejercían su derecho a
hacerlo estaban siendo violadas.
## 2 @USER @USER @USER Me carga en lo q se convirtió la 2da vuelta a la gobernación...una flaiterio.
## 3 mapuchento , que si hay una cultura y sociedad absolutamente hetero patriarcal, de un machismo
extremo, es justamente la mapuche?
## 4 @USER @USER @USER @USER Una vecina que nada tenia que ver con este flaiterio de sus hijas

##   label
## 1     0
## 2     0
## 3     1
## 4     0

tail(odio_huggingface_raw)
##
post
## 29850 @USER @USER @USER @USER @USER @USER @USER Y de Asier Guridi Zalloa, Refugiado Político
Vasco, que hace unos meses realizó una nueva huelga de hambre por sus derechos y los de su hijo
Iván... Sabemos algo señores-as del consulado?
## 29851 Un debate interesante ¿las mujeres occidentales que deciden "libremente" ponerse la parte
de arriba del bikini para tapar sus tetas son más libres que las musulmanas que deciden "libremente"
llevar velo? LINK
## 29852 @USER @USER De todo esto y leyendo las respuestas solo puedo decir que:\\n1. Que madurez y
que mensaje tan correcto y sin papel.\\n2. Que buena es la afición del Sevilla y lo querido que eres
en ella.\\n3. Nos vamos a divertir mucho contigo en el campo en el Camp Nou.\\n\\n\\nBienvenido al
@USER
## 29853 @USER @USER la reina respeta la religión musulmana en Marruecos y no respeta la religión
Cristiana de su país. Con estos gestos de la reina creo que se acorta el tiempo de la monarquía en
España. La adscripción religiosa del monarca data de 1496. Explicárselo a la reina, por favor

##   label
## 29850   0
## 29851   0
## 29852   0
## 29853   0
```

La peculiaridad ya mencionada de este dataset, es que contiene comentarios en distintas variantes del Español: hay mensajes de Chile y de México, no solo de España. Esto seguramente tenga una influencia (para mal) en el rendimiento del modelo, ya que aumenta la variedad de expresiones y modismos que el algoritmo tiene que entender y ser capaz de generalizar para luego poder clasificar textos nuevos (aunque luego solo se clasifiquen comentarios en Español de España).

Nota:

Por ejemplo, en los mensajes 2 y 3 se ve la palabra “flaiterio”. El autor de este trabajo desconocía por completo el significa de esta palabra. Según la Asociación de Academias de la Lengua Española:

<https://www.asale.org/damer/flaite>

flaite.		
I.	1.	sust/adj. <i>Ch.</i> juv. Persona de clase social baja que suele mostrar un comportamiento agresivo y viste de forma un tanto extravagante. desp.
	2.	adj/sust. <i>Ch.</i> <i>Referido a persona</i> , de comportamiento poco refinado.
	3.	m-f. <i>Ch.</i> Ladrón. delinc.
	4.	adj. <i>Ch.</i> <i>Referido a cosa</i> , de mal gusto. desp.
	5.	<i>Ch.</i> <i>Referido a cosa</i> , de mala o poca calidad. desp.

Parece una palabra candidata a aparecer en mensajes de odio.

Paso 2: Procesado del dataset

Se realiza el mismo procesado que para el dataset anterior.

Para evitar repeticiones se muestran solo algunos fragmentos.

Corpus

Creación del objeto corpus con todas los mensajes:

```
#create corpus
posts_corpus <- VCorpus(VectorSource(odio_huggingface$post))

print(posts_corpus)
## <VCorpus>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 29854
```

Este dataset es casi 3 veces más grande que el de Kaggle, así que lógicamente el corpus resulta casi 3 veces más grande también.

Se realiza también la tokenización para crear la DTM, y luego se crean los conjuntos de entrenamiento y de test. Utilizamos para ello una de las funciones que están en el módulo común:

```
#Set seed to make the process reproducible
set.seed(123)

result <- train_test_split(odio_huggingface, posts_dtm, 0.75)
## train dtm nrow: 22391
## test dtm nrow: 7463
## length of train labels: 22391
## length of test labels: 7463

#create training set
posts_dtm_train <- result$dtm_train

#create testing set
posts_dtm_test <- result$dtm_test

#create labels sets
posts_train_labels <- result$train_labels
posts_test_labels <- result$test_labels
```

Vamos a comprobar si se mantiene la (des)proporción mensajes no de odio/mensajes de odio:

```
prop.table(table(posts_train_labels))
## posts_train_labels
##      0      1
## 0.756688 0.243312

prop.table(table(posts_test_labels))
## posts_test_labels
##      0      1
## 0.756662 0.243338
```

Se mantiene la proporción 75/25 (mensajes no de odio/mensajes de odio).

Se necesita ahora obtener un listado con las palabras más utilizadas para limitar el número de columnas/variables de los conjuntos de entrenamiento y de test:

```
#Data preparation - creating indicator features for frequent words
posts_freq_words_train <- findFreqTerms(posts_dtm_train, 20) # 10 -> ~3700 terms
# 20 -> ~2000
# 100 -> ~450

#tests
print("tonto" %in% posts_freq_words_train)
## [1] TRUE
print("imbecil" %in% posts_freq_words_train) # <- FALSE with freq 100
## [1] TRUE
```

El dataset anterior era demasiado pequeño. En este he jugado con la frecuencia mínima. Antes de utilizar las matrices dispersas, este factor era bastante importante. Por la siguiente razón: cuánto más baja es la frecuencia necesaria para seleccionar una palabra, lógicamente más palabras son seleccionadas, y la matriz crece de manera proporcional. Con matrices de cierto tamaño se llegaba a colapsar la sesión del RStudio. Una vez solventado este problema, he procurado elegir el valor de la frecuencia lo más bajo posible. Solo he tenido en cuenta un poco cuál era el número total de observaciones. Por eso en este dataset en lugar de elegir el mínimo número de ocurrencias 10, he elegido 20.

Paso 3: Entrenamiento del modelo y evaluación

Se trata del mismo código para todos los datasets, así que se muestran directamente los resultados:

Primer modelo (coste = 1)

```
## Confusion Matrix and Statistics
##
##      Reference
## Prediction  0    1
##      0 5216  845
##      1  431  971
##
##      Accuracy : 0.829
##      95% CI : (0.8203, 0.8375)
##      No Information Rate : 0.7567
##      P-Value [Acc > NIR] : < 2.2e-16
##
##      Kappa : 0.4968
```

```

##
## McNemar's Test P-Value : < 2.2e-16
##
##      Sensitivity : 0.5347
##      Specificity : 0.9237
##      Pos Pred Value : 0.6926
##      Neg Pred Value : 0.8606
##      Precision : 0.6926
##      Recall : 0.5347
##      F1 : 0.6035
##      Prevalence : 0.2433
##      Detection Rate : 0.1301
##      Detection Prevalence : 0.1879
##      Balanced Accuracy : 0.7292
##
##      'Positive' Class : 1
##

```

Tabla comparativa de resultados:

	Coste 1, freq 20	Coste 1, freq 50	Coste 1, freq 100
Matriz de confusión	Reference Pred. No Yes No 5216 845 Yes 431 971	Reference Pred. No Yes No 5346 999 Yes 301 817	Reference Pred. No Yes No 5384 1249 Yes 262 567
Exactitud	0.8290	0.8258	0.7975
Kappa	0.4968	0.4560	0.3259
F1	0.6035	0.5569	0.4287
MCC	0.5035		
	Coste 1, freq 20 pesos 1/3	Coste 0.289722 freq 20, pesos 2/3	
Matriz de confusión	Reference Pred. No Yes No 4474 487 Yes 1173 1329	Reference Pred. No Yes No 5002 694 Yes 645 1122	
Exactitud	0.7776	0.8206	
Kappa	0.4646	0.5083	
F1	0.6156	0.6263	
MCC		0.5088	

Cuantas más palabras tiene el vocabulario de entrenamiento, parece que mejores resultados se consiguen. Con el dataset anterior de Kaggle se consiguió una exactitud del 76-77%, y un kappa de 0.47. Ahora se ha conseguido pasar del 80% de exactitud, y un kappa mejor que ronda el 0.51.

3. Dataset Hatemedia

Paso 1: Obtención del dataset y procesamiento preliminar

```
# import the CSV file
odio_hatemedia_raw <- read.csv(file.path("dataset_01_hatemedia.csv"), sep=";")
odio_hatemedia_raw <- odio_hatemedia_raw[-1] # -> 574272 obs. of 2 variables
```

Dado el tamaño tan grande de este dataset (y teniendo en cuenta los límites físicos del equipo), se puede elegir si se procesa el dataset al completo, o si se quiere seleccionar uno que contenga un subconjunto de los mensajes no de odio, más todos los de odio. Es una manera de hacer **“downsampling”**, pero eligiendo el número de mensajes no de odio, en lugar de elegir automáticamente el mismo número de mensajes de las dos clases.

```
# Prepare a bit smaller dataset: 100k, 200k ..
if (!complete_dataset) {
  print("Preparing a smaller dataset")

  df_hate <- odio_hatemedia_raw[odio_hatemedia_raw$label == 1, ]
  df_no_hate <- odio_hatemedia_raw[odio_hatemedia_raw$label == 0, ]

  n <- nrow(df_no_hate)
  k <- size # number of random rows: 18936 (quite balanced) 88936 (total 100k, best outcome)
  288936 (total 300k) ...

  ids <- sample(n, size = k, replace = FALSE)
  df_no_hate_sample <- df_no_hate[ids, , drop = FALSE]

  # join sample dataset with no hate obs. with hate obs.
  odio_hatemedia_sample_k <- rbind(df_no_hate_sample, df_hate)

  # so we don't have to change variable names:
  odio_hatemedia_raw <- odio_hatemedia_sample_k

  # clean environment
  rm(df_hate, df_no_hate, df_no_hate_sample, odio_hatemedia_sample_k)
} else {
  print("Working with the whoooole dataset. Be patient!!!")
}

## [1] "Working with the whoooole dataset. Be patient!!!"
```

Se han hecho pruebas con diferentes tamaños del dataset. Como se puede ver en la comparativa al final de esta sección, el mejor resultado se obtuvo con el dataset de 100k observaciones.

Estructura del dataset:

```
str(odio_hatemedia_raw)
## 'data.frame':    574272 obs. of  2 variables:
## $ post : chr
"real,madrid,puesto,punto,final,andaluz,copo,rey,primero,escalon,zidane,caer,alcoyano,segundo,pesar
,empezar,gan"| __truncated__ "decir,coaccion,cifu,"
"tambien,salar,ayuso,sumario,caray,parecer,ves,ayuso,sopa," "peperar,celula,sitio," ...
## $ label: int  0 0 0 0 0 0 0 0 0 0 ...
```


La columna “label” es de tipo int. Ya que se trata en realidad de una variable categórica 0/1, es conveniente transformarla en un factor:

```
#Convert label into a factor
odio_hatemediaw_raw$label <- factor(odio_hatemediaw_raw$label)
```

Ya sabíamos que se trata de un dataset muy, muy desbalanceado. El más grande y más desbalanceado de los tres:

```
table(odio_hatemediaw_raw$label)
##
##      0      1
## 563208 11064

prop.table(table(odio_hatemediaw_raw$label))
##
##      0      1
## 0.98073387 0.01926613
```

El porcentaje de mensajes de odio en este dataset no llega al 2%.

En este dataset, en la variable “post” se debe reemplazar las “,” por espacios. Si no hacemos esto con este dataset, lo que se obtiene más tarde para cada comentario es una única cadena enorme que contiene todas las palabras concatenadas del comentario. (Esto es debido a que este dataset de Hatemediaw ya estaba parcialmente procesado como hemos dicho).

```
# Replace all "," in this dataset. If not, after processing it we get lines of only one "huge word"
odio_hatemediaw_raw$post <- gsub(',', ' ', odio_hatemediaw_raw$post)
```

Comprobación:

```
head(odio_hatemediaw_raw, )
##
## post
## 1 real madrid puesto punto final andaduro copo rey primero escalon zidane caer alcoyano segundo
## pesar empezar ganar jugar hombre menos prorrogar tecnico franz disponer equipo plagado menos
## habitual vinicius mariano ataque ninguno dos logro crear ocasión militao marco gol madrid justo
## descanso segundo parte intentar cerrar partido colmillo suficiente modesto alcoyano aprovechar
## corner empatar partido cinco minuto final empate sento jarro agua frio blanco intentar tiempo extra
## faltar cinco minuto casanova consiguio gol mas importante vida valer clasificacion octavo copa
## madrid zidane quedar apeado torneo vez franz quedar pelear unico titulo conseguir nunca asi contar
## minuto minuto partido directo
## 2 decir coaccion cifu
## ...
##      label
## 1      0
## 2      0
## ...
```

Paso 2: Procesado del dataset

En este dataset, se han encontrado mensajes en catalán:

```
#odio_hatemediaw_raw[17:37,1]
```

```
[1] "dimecr coolhunter preguntavar responsabilitat tenir trio mixt presentadors campanadser tot
## venir despr toni cruany ..."
[2] " avui podem jugar mama tots castellans aixi lamentar mes vegada fills jordi pujol marta
## ferrusola segons explicar ..." ...
```

El objetivo de este TFM es detectar mensajes de odio en un solo idioma, no en varios, aunque sean tan parecidos como son el Castellano y el Catalán.

Dada la dificultad de detectar todos los comentarios en Catalán en un dataset tan grande como este, y dado el hecho de que en los medios online españoles muchos ciudadanos catalanes escriben sus comentarios en su idioma materno, por ahora se decide no eliminar estos mensajes.

Corpus

Creación del objeto corpus con todas los mensajes:

```
#create corpus
system.time({
  posts_corpus <- VCorpus(VectorSource(odio_hatemedia$post))
})
##      user      system elapsed
## 21.342    1.260    22.601

print(posts_corpus)
## <VCorpus>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 570868
```

El número de documentos es ahora inferior al inicial, ya que durante el procesado, se eliminaron algo menos de 4000 registros.

Una vez procesado el corpus y realizada la **tokenización**, se procede a crear los conjuntos de entrenamiento y de test:

```
#Set seed to make the process reproducible
set.seed(123)

result <- train_test_split(odio_hatemedia, posts_dtm, 0.75)

## train dtm nrows: 428151
## test dtm nrows: 142717
## length of train labels: 428151
## length of test labels: 142717

#create training set
posts_dtm_train <- result$dtm_train

#create testing set
posts_dtm_test <- result$dtm_test

#create labels sets
posts_train_labels <- result$train_labels
posts_test_labels <- result$test_labels

rm(result)
rm(posts_dtm)
```

Vamos a comprobar que se mantiene la (des)proporción mensajes no de odio/mensajes de odio:

```
prop.table(table(posts_train_labels))
## posts_train_labels
##      0      1
## 0.98061899 0.01938101

prop.table(table(posts_test_labels))
```

```
## posts_test_labels
##      0      1
## 0.98061899 0.01938101
```

Se necesita ahora obtener un listado con las palabras más utilizadas:

```
# Data preparation - creating indicator features for frequent words
# the function findFreqTerms() in the tm package takes a DTM and returns a character vector
containing words that appear at least a minimum number of times
posts_freq_words_train <- findFreqTerms(posts_dtm_train, freq) # 100 -> ~17000 terms
                                                                # 500 -> ~3700
                                                                # 1000 -> ~2100

print("tonto" %in% posts_freq_words_train)
## [1] TRUE
print("imbecil" %in% posts_freq_words_train) # <- FALSE with freq 1000
## [1] TRUE
```

Se han hecho pruebas con diferentes frecuencias mínimas. Al principio se eligieron frecuencias mínimas más altas, para limitar el tamaño de la DTM y de la matriz pasada como argumento a la función de entrenamiento. Una vez que se pudo procesar el dataset completo, se pudo elegir una frecuencia tan baja como 100 (y menor) para este dataset.

Y ahora utilizamos este listado para limitar el número de columnas/features tanto del conjuntos de entrenamiento como del de test:

```
dim(posts_dtm_train)
## [1] 428151 348382

posts_dtm_freq_train <- posts_dtm_train[, posts_freq_words_train]
dim(posts_dtm_freq_train)
## [1] 428151 16516

dim(posts_dtm_test)
## [1] 142717 348382

posts_dtm_freq_test <- posts_dtm_test[, posts_freq_words_train]
dim(posts_dtm_freq_test)
## [1] 142717 16516
```

Se puede ver que el conjunto de entrenamiento tiene más de 428k observaciones, y pasa de tener 348k variables (términos) a tener 16516.

Paso 3: Entrenamiento del modelo y evaluación

Para poder procesar el dataset completo, además de utilizar obligatoriamente matrices dispersas, se ha tenido que realizar la conversión de las DTMs a dichas matrices por lotes:

```
chunk_size <- 10000

system.time({
  chunk_list_train <- creat_sparse_mat_in_chunks(posts_dtm_freq_train, chunk_size)
  posts_freq_train_mat_chunks <- do.call(rbind, chunk_list_train)
  rm(chunk_list_train)
})
```

```
## chunk 1 processed.
## chunk 2 processed.
## chunk 3 processed.
...
## chunk 41 processed.
## chunk 42 processed.
## chunk 43 processed.
## user system elapsed
## 57.373 18.546 79.351

system.time({
  chunk_list_test <- creat_sparse_mat_in_chunks(posts_dtm_freq_test, chunk_size)

  posts_freq_test_mat_chunks <- do.call(rbind, chunk_list_test)

  rm(chunk_list_test)
})

## chunk 1 processed.
## chunk 2 processed.
## chunk 3 processed.
...
## chunk 13 processed.
## chunk 14 processed.
## chunk 15 processed.
## user system elapsed
## 21.096 6.475 27.781

posts_freq_train_mat <- posts_freq_train_mat_chunks
rm(posts_freq_train_mat_chunks)

posts_freq_test_mat <- posts_freq_test_mat_chunks
rm(posts_freq_test_mat_chunks)
```

Entrenamiento (coste = 1) y evaluación:

```
system.time({
  liblinear_svm_model <- LiblinearR(data=posts_freq_train_mat, target=posts_train_labels, type=3)
})
## user system elapsed
## 10.343 0.004 10.435
```

Se puede apreciar la rapidez de esta librería comparada con LIBSVM (e1071). En apenas 10s se ha entrenado un modelo SVM con más de 400k observaciones en un modesto equipo con procesador Intel core i5 y 16 GB de RAM.

Con las otras opciones de “type”, los tiempos eran muy parecidos. (Con LIBSVM/e1071 en cambio, más de una hora después de iniciado el proceso, no había terminado).

```
# prediction
system.time({
  prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)
})
## user system elapsed
## 0.216 0.000 0.237

table(as.factor(prediction_liblinear$predictions))
##
## 0 1
## 142263 454

# confusion matrix
confusionMatrix(reference = as.factor(posts_test_labels), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")
```

```

##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0      1
##      0 139756  2507
##      1   195    259
##
##           Accuracy : 0.9811
##           95% CI : (0.9803, 0.9818)
##      No Information Rate : 0.9806
##      P-Value [Acc > NIR] : 0.1111
##
##           Kappa : 0.1563
##
##  Mcnemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.093637
##           Specificity : 0.998607
##           Pos Pred Value : 0.570485
##           Neg Pred Value : 0.982378
##           Precision : 0.570485
##           Recall : 0.093637
##           F1 : 0.160870
##           Prevalence : 0.019381
##           Detection Rate : 0.001815
##           Detection Prevalence : 0.003181
##           Balanced Accuracy : 0.546122
##
##           'Positive' Class : 1
##

```

Dado el tamaño tan grande del dataset de test, y el desequilibrio entre las dos clases, es completamente lógico que se obtenga una exactitud tan (aparentemente) buena, y un kappa y un valor de F1 tan bajos (y una exactitud balanceada que no llega al 55%). Por no hablar del recall (no se detectan ni un 10% de los mensajes de odio) ...

Lógicamente, no se trata de obtener un modelo que solo sirva para clasificar los mensajes no de odio como mensajes no de odio.

Con este dataset he hecho bastantes pruebas. Se muestra a continuación una comparativa de las mismas. Resultados obtenidos con $\text{coste} = 1$, variando el tamaño del dataset (es decir, su grado de “balanceo”), y la frecuencia mínima:

- Dataset con 300k observaciones:

freq baja (100,200,300,400): la sesión R “explota”

freq intermedia (500): alta exactitud (96%), kappa baja (0.28)

Pesos 1/2 -> 96% y 0.36

Pesos 1/5 -> 95% y 0.38 (se mejora algo kappa)

freq alta (1000): resultados “absurdos” (0 TP, kappa 0, y los pesos no lo arreglan)

- Dataset con 100k observaciones:

freq baja (100): exactitud alta (92.5%) y kappa aceptable (0.58)

Con pesos 1/2, la exactitud se mantiene (**92%**), y kappa es **0.61**.

- Dataset 30k observaciones (bastante balanceado: 19k no de odio vs 11k de odio):

freq baja (100) -> alta exactitud (83%), buen kappa (0.62)

freq alta (500) -> baja exactitud, kappa despreciable

Como se puede ver, con el dataset de 100k observaciones, se obtiene un resultado bastante aceptable.

Hay que mencionar no obstante, que estos resultados fueron obtenidos antes de que consiguiera procesar el dataset completo. Vamos a intentar mejorar el resultado con el dataset completo (exactitud 98%, y kappa 0.15), ya que el objetivo final es construir un dataset lo más grande posible juntando los 3 datasets mencionados.

	Dataset 100k , coste 1, freq100, pesos 1/2	Dataset completo, coste 1, freq 100	Dataset completo, coste 1, freq 100, pesos 1/10																																				
Matriz de confusión	<table><tr><td></td><td colspan="2">Reference</td></tr><tr><td>Pred.</td><td>No</td><td>Yes</td></tr><tr><td>No</td><td>21145</td><td>947</td></tr><tr><td>Yes</td><td>954</td><td>1819</td></tr></table>		Reference		Pred.	No	Yes	No	21145	947	Yes	954	1819	<table><tr><td></td><td colspan="2">Reference</td></tr><tr><td>Pred.</td><td>No</td><td>Yes</td></tr><tr><td>No</td><td>139756</td><td>2507</td></tr><tr><td>Yes</td><td>195</td><td>259</td></tr></table>		Reference		Pred.	No	Yes	No	139756	2507	Yes	195	259	<table><tr><td></td><td colspan="2">Reference</td></tr><tr><td>Pred.</td><td>No</td><td>Yes</td></tr><tr><td>No</td><td>139957</td><td>992</td></tr><tr><td>Yes</td><td>4994</td><td>1774</td></tr></table>		Reference		Pred.	No	Yes	No	139957	992	Yes	4994	1774
	Reference																																						
Pred.	No	Yes																																					
No	21145	947																																					
Yes	954	1819																																					
	Reference																																						
Pred.	No	Yes																																					
No	139756	2507																																					
Yes	195	259																																					
	Reference																																						
Pred.	No	Yes																																					
No	139957	992																																					
Yes	4994	1774																																					
Exactitud	0.9235	0.9811	0.9581																																				
Kappa	0.6138	0.1563	0.3544																																				
F1	0.6568	0.1608	0.3721																																				
MCC	0.6138	0.2258																																					
	Dataset completo, coste 1, freq 100, pesos 1/5	Dataset completo, coste 1, freq 100 pesos 1/3	Dataset completo, coste 1, freq 100 pesos 1/2																																				
Matriz de confusión	<table><tr><td></td><td colspan="2">Reference</td></tr><tr><td>Pred.</td><td>No</td><td>Yes</td></tr><tr><td>No</td><td>136865</td><td>1277</td></tr><tr><td>Yes</td><td>3086</td><td>1489</td></tr></table>		Reference		Pred.	No	Yes	No	136865	1277	Yes	3086	1489	<table><tr><td></td><td colspan="2">Reference</td></tr><tr><td>Pred.</td><td>No</td><td>Yes</td></tr><tr><td>No</td><td>138069</td><td>1583</td></tr><tr><td>Yes</td><td>1882</td><td>1183</td></tr></table>		Reference		Pred.	No	Yes	No	138069	1583	Yes	1882	1183	<table><tr><td></td><td colspan="2">Reference</td></tr><tr><td>Pred.</td><td>No</td><td>Yes</td></tr><tr><td>No</td><td>138903</td><td>1937</td></tr><tr><td>Yes</td><td>1048</td><td>829</td></tr></table>		Reference		Pred.	No	Yes	No	138903	1937	Yes	1048	829
	Reference																																						
Pred.	No	Yes																																					
No	136865	1277																																					
Yes	3086	1489																																					
	Reference																																						
Pred.	No	Yes																																					
No	138069	1583																																					
Yes	1882	1183																																					
	Reference																																						
Pred.	No	Yes																																					
No	138903	1937																																					
Yes	1048	829																																					
Exactitud	0.9694	0.9757	0.9791																																				
Kappa	0.3910	0.3934	0.3469																																				
F1	0.4057	0.4058	0.3571																																				
MCC		0.3935																																					

Al estar tan descompensado el dataset, pensé utilizar una relación de pesos 1 a 10. Esto mejoró bastante kappa con respecto al modelo original (de 0.15 a 0.35).

Con pesos 1/5 y 1/3, se obtienen valores de exactitud, kappa y F1 muy parecidos. Hay que elegir entre detectar más TP, a costa de obtener más FP, o menos TP y también menos FP.

En cualquier caso, los valores de kappa son bastante mediocres, rondando el 0.4

Si comparamos con los datasets anteriores, se ha vuelto a mejorar la exactitud, pero kappa ha retrocedido de 0.5 al 0.4.

(A no ser que considereremos el dataset de 100k observaciones, con el que se consigue el mejor resultado de todos).

Al no mejorar kappa, y pensando en la posibilidad de que elegir una frecuencia tan baja produjera algo de sobreajuste a los textos (siendo consciente de que SVM no es propenso a este problema -una de las razones por las que se ha elegido esta técnica), se probó con una frecuencia mínima más alta: 500 (en lugar de 100)

Con freq = 500 el tamaño de las DTMs y matrices disminuye bastante:

```
[1] 428151 348382
[1] 428151    5886    ← 5896 columnas/features en lugar de 16516 con freq 100
[1] 142717 348382
[1] 142717    5886    ← idem.
```

(Curiosamente los tiempos de entrenamiento son mayores).

Pero los resultados obtenidos, no son mejores. Todo lo contrario:

	Dataset completo, coste 1, freq 100	Dataset completo, coste 1, freq 500																								
Matriz de confusión	<table> <tr> <td></td><th colspan="2">Reference</th></tr> <tr> <th>Pred.</th><th>No</th><th>Yes</th></tr> <tr> <th>No</th><td>139756</td><td>2507</td></tr> <tr> <th>Yes</th><td>195</td><td>259</td></tr> </table>		Reference		Pred.	No	Yes	No	139756	2507	Yes	195	259	<table> <tr> <td></td><th colspan="2">Reference</th></tr> <tr> <th>Pred.</th><th>No</th><th>Yes</th></tr> <tr> <th>No</th><td>139819</td><td>2613</td></tr> <tr> <th>Yes</th><td>132</td><td>153</td></tr> </table>		Reference		Pred.	No	Yes	No	139819	2613	Yes	132	153
	Reference																									
Pred.	No	Yes																								
No	139756	2507																								
Yes	195	259																								
	Reference																									
Pred.	No	Yes																								
No	139819	2613																								
Yes	132	153																								
Exactitud	0.9811	0.9808																								
Kappa	0.1563	0.097																								
F1	0.1608	0.1003																								

El resultado es bastante peor que con freq = 100.

(Con ningún peso se consigue mejorar tampoco los resultados obtenidos con freq = 100).

2. Creación de un dataset total y entrenamiento de SVM

La idea de juntar todos los datasets, es intentar que el SVM, al tener fuentes de información más diversas (y más grandes, pero sobre todo diversas), pueda ser

entrenado de mejor manera y obtener así los mejores resultados posibles con los nuevos mensajes sin etiquetar que se le presenten.

También se va a hacer en esta fase del proyecto, dado el buen resultado obtenido con el dataset de Hatemedia de 100k observaciones (exactitud 92% y kappa 0.61), además de entrenar el modelo SVM con el dataset total compuesto de los 3 datasets, crear a partir de este dataset “total” otro de 100k comentarios de la misma manera: contendrá todos los mensajes de odio del dataset total, y un número tal de mensajes no de odio hasta llegar a esos 100k registros.

Luego se entrenará SVM con ambos datasets, y se compararán los resultados (también con los anteriores).

1. Creación del dataset total

Variables globales:

```
complete_dataset <- params$complete           # FALSE
crossValidation <- params$crossValidation       # TRUE
gridSearch <- params$gridSearch                # TRUE

# if no complete dataset, number of no hate messages to pick from the total dataset
size <- 78000 # 77432 number of random rows of no hate messages (value rounded)

# number of min. freq (the lower the size, the lower the freq)
freq <- 100

# assign higher weight to the minority class
class_weights <- c("0" = 1, "1" = 2)

random_seed <- 123 # 123, 9, 900 <- it has no influence at all in the results
```

Importar todos los ficheros CSV:

```
# import the CSV files
odio_hatemedia_raw <- read.csv(file.path("dataset_01_hatemedia.csv"), sep=";")
odio_hatemedia_raw <- odio_hatemedia_raw[-1] # -> 574272 obs. of 2 variables

odio_huggingface_raw <- read.csv(file.path("dataset_02_huggingface.csv"), sep=";")
odio_huggingface_raw <- odio_huggingface_raw[-1] # -> ~29855 obs. of 2 variables

odio_kaggle_raw <- read.csv(file.path("dataset_03_kaggle.csv"), sep=";")
odio_kaggle_raw <- odio_kaggle_raw[-1] # -> ~11180 obs. (only spanish) of 2 variables

odio_kaggle_test_raw <- read.csv(file.path("dataset_04_kaggle.csv"), sep=";")
odio_kaggle_test_raw <- odio_kaggle_test_raw[-1] # -> 1243 obs. (only spanish) of 2 variables
```

Ya sabemos que el dataset de Hatemedia es algo diferente. Procedemos a eliminar las “,” de nuevo para evitar problemas.

```
# Replace all "," in this dataset. If not, after processing it we get lines of only one "huge word"
odio_hatemedia_raw$post <- gsub(',', ' ', odio_hatemedia_raw$post)
```

Ahora ya se puede crear un dataset “total” con todos los datasets disponibles:

```
hate_raw <- rbind(odio_hatemedia_raw, odio_huggingface_raw, odio_kaggle_raw, odio_kaggle_test_raw)
dim(hate_raw)
## [1] 616550      2
```

Disponemos ahora de un dataset con un total de 616550 comentarios de lectores de periódicos online y otras fuentes.

[
Eliminar los datasets y variables que ya no son necesarios:

```
l_rm = ls(pattern = "^odio_")
rm(list=l_rm)
l_rm = ls(pattern = "^df_")
rm(list=l_rm)
]
```

Comprobar proporción de los mensajes en este dataset:

```
table(hate_raw$label)
##
##      0      1
## 593982 22568

prop.table(table(hate_raw$label))
##
##      0      1
## 0.96339632 0.03660368
```

Evidentemente, al ser el dataset de Hatemedia muchísimo mayor que los otros 2 y estar muy desbalanceado, el dataset resultante también resulta muy muy desbalanceado.

2. Creación del dataset de 100k observaciones y entrenamiento

Como disponemos de 22568 mensajes de odio, vamos a seleccionar de manera aleatoria el siguiente número de mensajes de no odio (se redondea a 78000 como se indica en la variable “size” del principio):

```
no_hate_n <- 100000 - 22568
cat("Number of no hate messages to be selected: ",no_hate_n,"\n")
## Number of no hate messages to be selected: 77432
set.seed(random_seed) # idea: repeat the process with different seeds,
                       # to see the influence of randomly chosen rows <- no influence seen

# Prepare a bit smaller dataset: 100k
if (!complete_dataset) {
  print("Preparing a smaller dataset")

  df_hate <- hate_raw[hate_raw$label == 1, ]
  df_no_hate <- hate_raw[hate_raw$label == 0, ]

  n <- nrow(df_no_hate)
  k <- size # number of random rows with no hate messages

  ids <- sample(n,size = k, replace = FALSE)
  df_no_hate_sample <- df_no_hate[ids, ,drop = FALSE]

  # join sample dataset with no hate obs. with hate obs.
  hate_raw_sample_k <- rbind(df_no_hate_sample, df_hate)

  #so we don't have to change all variable names
  hate_raw <- hate_raw_sample_k

  rm(df_hate,df_no_hate,df_no_hate_sample,hate_raw_sample_k)
} else {
```

```
print("Working with the whooole dataset. Be patient!!!")
}
## [1] "Preparing a smaller dataset"
```

Comprobar proporción de los mensajes en este dataset después del “downsampling”:

```
table(hate_raw$label)
##
##      0      1
## 78000 22568

prop.table(table(hate_raw$label))
##
##      0      1
## 0.7755946 0.2244054
```

El dataset sigue estando desbalanceado, pero en bastante menor grado.

(En el dataset de Hatemedia no se obtuvo mejor resultado al reducir más el grado de desbalanceo bajando el total de observaciones no de odio a 30k).

Estructura del dataset:

```
str(hate_raw)
## 'data.frame': 100568 obs. of 2 variables:
## $ post : chr "vender argentino " "odiar verbal mismo forma violencio gustado hoy " "athletic
remontar final barca conquista supercopa españa " "refrescant " ...
## $ label: int 0 0 0 0 0 0 0 0 0 ...

head(hate_raw)

post
## 188942 vender argentino
## 134058 odar verbal mismo forma violencio gustado hoy
## 124022 athletic remontar final barca conquista supercopa españa
## 226318 refrescant
## 365209 love nyc the homecoming concert ser cancelar huracan henri
## 193627 normal ver jan oblak bloquear balon poderoso mano balon aparentemente comodo pafia provoco
segundo jugada suponer mitrovic gol acabar racha partido getafe marcar gol atletico diego castro
noviembre simeonir iniciado mes despu balance gol cero favor rojiblanco certero cabezazo mitrovic
borde descanso simeonir dirigio vestuario cabizbajo mano bolsillo nuevo obligado corregir mal
primero tiempo equipo hablar segundo mitad crear mas allar dos gol mos ver equipo manera intensidad
mas dinamico mas contundente esperar formar cada jugador aparecezo analisis tecnico rojiblanco
victoria impulso simeonir explicar causa pensar equipo acabar romper espera asome mejor version
futbolista argumentacion salio defensa lustroso ramillete atacante hablar siempre delantero poder
representar pasar cuatro cunha llegado hacer jugar

##      label
## 188942    0
## 134058    0
## 124022    0
## 226318    0
## 365209    0
## 193627    0

tail(hate_raw)

post
## 616532 El 155 otra vez y convocar elecciones ad infinitum hasta que xenófobos, fascistas,
pesoeistas, o cualquier otro partido corrupto nacionalista español o catalán ganen.
## 616533 Definitivamente tu eres una perra
## 616534 Que bueno que no fui güerita, si prieta soy mamona, no quiero ni imaginarme güera jajajaa
## 616540 El que viene en yate de lujo no viene a pedir, si acaso a negociar. El que viene en patera
no viene a dar, viene a consumir los recursos sociales de los españoles. Si cada uno de los
4.000.000 de populistas que los quiere, se llevara un sólo inmigrante a su casa, asunto resuelto.
## 616543 Pero vamos mi novia es Sevillista y es una puta más .... SI LO ES SE DICE Y NO PASA NADA ,
es que disfruto más contigo que con mi mujer
## 616549 Odíame pero al mirarte a un espejo recuerda: yo soy agil y atractivo, tú eres una puta
```

cerda Y es que tienes tanta tontería y tienes tanta maldad, que en algún sitio de tu cuerpo se tiene que acumular

```
##          label
## 616532      1
## 616533      1
## 616534      1
## 616540      1
## 616543      1
## 616549      1
```

Procesado del dataset

La columna "label" es de tipo int. Ya que se trata en realidad de una variable categórica 0/1, es conveniente transformarla en un factor:

```
#Convert class into a factor
hate_raw$label <- factor(hate_raw$label)
```

Procesamos los comentarios de la manera habitual (eliminar referencias a otros lectores, eliminar emoticonos ..):

```
#check_text(hate_raw$post)
system.time({
  hate <- preprocess_posts(hate, hate_raw)
})
## [1] "Removed references to users (@)."
```

```
## [1] "Removed non ascii and emoticons."
## [1] "Removed lines with empty posts."
##      user  system elapsed
## 59.783   0.028   59.820
```

Durante este proceso puede suceder que algunos comentarios queden vacíos. Los eliminamos en ese caso:

```
# Number of rows deleted:
obs_removed <- nrow(hate_raw)-nrow(hate)

cat("Se han eliminado ", obs_removed, " líneas al procesar el dataset.\n")
## Se han eliminado 433 líneas al procesar el dataset.

rm(hate_raw)
```

Corpus de los textos

Ya se puede proceder a la creación del objeto corpus con todos los mensajes:

```
#create corpus
system.time({
  posts_corpus <- VCorpus(VectorSource(hate$post))
})
##      user  system elapsed
##   3.482   0.204   3.686

print(posts_corpus)
## <<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 100135
```

Limpieza de los textos del corpus:

Procesado habitual: eliminar mayúsculas, números ...

```

system.time({
  posts_corpus_clean <- clean_corpus(posts_corpus)
})
## [1] "#To lowercase"
## [1] "#Remove numbers"
## [1] "#Remove stopwords"
## [1] "#Remove punctuation signs"
## [1] "#Carry out the stemming"
## [1] "#Finally eliminate unneeded whitespace produced by previous steps"
##      user      system elapsed
## 46.487    0.032   46.524

```

Finalmente, se procede a la **“tokenización”** de los comentarios:

```

system.time({
  posts_dtm <- DocumentTermMatrix(posts_corpus_clean)
})
##      user      system elapsed
## 16.240    0.056   16.295

posts_dtm
## <<DocumentTermMatrix (documents: 100135, terms: 137222)>>
## Non-/sparse entries: 3387698/13737337272
## Sparsity           : 100%
## Maximal term length: 279
## Weighting           : term frequency (tf)

```

[
En este punto eliminar los objetos “corpus”. Ya no son necesarios.

```

rm(posts_corpus)
rm(posts_corpus_clean)
]

```

Ahora hay que crear los conjuntos de entrenamiento y de test.

```

#Set seed to make the process reproducible 123
set.seed(random_seed)

result <- train_test_split(hate, posts_dtm, 0.75)
## train dtm nrow: 75102
## test dtm nrow: 25033
## length of train labels: 75102
## length of test labels: 25033
#create training set
posts_dtm_train <- result$dtm_train

#create testing set
posts_dtm_test <- result$dtm_test

#create labels sets
posts_train_labels <- result$train_labels
posts_test_labels <- result$test_labels

rm(result)
rm(posts_dtm)

```

Vamos a comprobar si se mantiene la (des)proporción mensajes no de odio/mensajes de odio:

```

prop.table(table(posts_train_labels))
## posts_train_labels
##      0      1
## 0.7746398 0.2253602

prop.table(table(posts_test_labels))
## posts_test_labels
##      0      1
## 0.7746575 0.2253425

```

Se necesita ahora obtener el listado de las palabras más utilizadas:

```
posts_freq_words_train <- findFreqTerms(posts_dtm_train, freq) # 100 -> ~4700 terms
```

Y ahora utilizamos ese listado para limitar el número de columnas/features tanto del conjuntos de entrenamiento como del de test:

```
dim(posts_dtm_train)
## [1] 75102 137222
posts_dtm_freq_train <- posts_dtm_train[, posts_freq_words_train]
dim(posts_dtm_freq_train)
## [1] 75102 4716
dim(posts_dtm_test)
## [1] 25033 137222
posts_dtm_freq_test <- posts_dtm_test[, posts_freq_words_train]
dim(posts_dtm_freq_test)
## [1] 25033 4716
```

Pasamos de tener 137222 columnas/features a solo 4716.

Entrenamiento del modelo y evaluación

Necesitamos convertir las DTMs en matrices para poder entrenar el modelo. Dado el tamaño del dataset, y las limitaciones físicas del equipo en el que se realiza este proceso, se procesa en lotes las DTMs, y posteriormente se juntan las matrices para obtener la matriz total.

```
chunk_size <- 10000

system.time({
  chunk_list_train <- creat_sparse_mat_in_chunks(posts_dtm_freq_train, chunk_size)

  posts_freq_train_mat_chunks <- do.call(rbind, chunk_list_train)

  rm(chunk_list_train)
})
## chunk 1 processed.
## chunk 2 processed.
...
## chunk 7 processed.
## chunk 8 processed.
## user system elapsed
## 3.963 0.600 4.568

system.time({
  chunk_list_test <- creat_sparse_mat_in_chunks(posts_dtm_freq_test, chunk_size)

  posts_freq_test_mat_chunks <- do.call(rbind, chunk_list_test)

  rm(chunk_list_test)
})
## chunk 1 processed.
## chunk 2 processed.
## chunk 3 processed.
## user system elapsed
## 0.896 0.180 1.076

posts_freq_train_mat <- posts_freq_train_mat_chunks
rm(posts_freq_train_mat_chunks)

posts_freq_test_mat <- posts_freq_test_mat_chunks
rm(posts_freq_test_mat_chunks)
```

Entrenamiento:

```
system.time({  
  liblinear_svm_model <- LiblinearR(data=posts_freq_train_mat, target=posts_train_labels, type=3) # cost = 1  
})  
##      user      system elapsed  
##    2.471      0.000      2.471
```

Predicción:

```
# prediction  
system.time({  
  prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)  
})  
##      user      system elapsed  
##    0.017      0.000      0.017  
  
table(as.factor(prediction_liblinear$predictions))  
##  
##      0      1  
## 19916  5117
```

Evaluación del resultado:

```
# confusion matrix  
confusionMatrix(reference = as.factor(posts_test_labels), data =  
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")  
  
## Confusion Matrix and Statistics  
##  
##           Reference  
## Prediction      0      1  
##      0 18380  1536  
##      1  1012  4105  
##  
##              Accuracy : 0.8982  
##              95% CI : (0.8944, 0.9019)  
##      No Information Rate : 0.7747  
##      P-Value [Acc > NIR] : < 2.2e-16  
##  
##              Kappa : 0.6985  
##  
##  Mcnemar's Test P-Value : < 2.2e-16  
##  
##              Sensitivity : 0.7277  
##              Specificity : 0.9478  
##      Pos Pred Value : 0.8022  
##      Neg Pred Value : 0.9229  
##              Precision : 0.8022  
##              Recall : 0.7277  
##              F1 : 0.7632  
##              Prevalence : 0.2253  
##      Detection Rate : 0.1640  
##      Detection Prevalence : 0.2044  
##      Balanced Accuracy : 0.8378  
##  
##      'Positive' Class : 1  
##
```

Cuando se tiene un dataset desbalanceado, pero en una proporción como esta (3 a 1 a favor de los mensajes no de odio), se obtiene un resultado más que aceptable. Una exactitud rozando el **90%**, y un kappa rozando el **0.7**, me parecen más que buenos (el resto de indicadores también me parecen bastante buenos).

Teniendo en cuenta además que un tamaño de dataset de 100k observaciones, me parece bastante realista (los datasets de HuggingFace y Kaggle por separado son mucho más pequeños).

También se puede comparar este resultado de juntar los 3 datasets y sacar un dataset de 100k observaciones, con el de 100k observaciones de Hatemedia: kappa mejora del 0.61 al 0.7.

Nota:

Al estar utilizando matrices dispersas para poder manejar un dataset de este tamaño, no podemos utilizar la función heurística de Liblinear para calcular un coste. De todas maneras, después de bastantes pruebas, se ha visto que los pesos ayudan a mejorar el resultado mucho más que utilizar un coste distinto de 1.

- Mejora del modelo usando pesos

Se ha probado con relaciones 1/5, 1/3 y 1/2. Solo con estos últimos pesos se mejora el resultado inicial:

```
system.time({
  liblinear_svm_model_weights <- LiblinearR(data = posts_freq_train_mat, target = posts_train_labels,
                                             type = 3,
                                             w1 = class_weights)
})
##      user  system elapsed
##    3.194    0.000    3.194
```

Predicción:

```
# prediction
system.time({
  prediction_liblinear_weights <- predict(liblinear_svm_model_weights, posts_freq_test_mat)
})
##      user  system elapsed
##    0.016    0.000    0.016
```

Evaluación del resultado:

```
#Confusion matrix
confusionMatrix(reference = as.factor(posts_test_labels),
                 data = as.factor(prediction_liblinear_weights$predictions),
                 positive="1",
                 mode = "everything")
## Confusion Matrix and Statistics
##
##      Reference
## Prediction  0      1
##      0 17967 1180
##      1 1425 4461
##
##              Accuracy : 0.8959
##              95% CI : (0.8921, 0.8997)
##      No Information Rate : 0.7747
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.7065
##
```

```
## McNemar's Test P-Value : 1.747e-06
##
##      Sensitivity : 0.7908
##      Specificity : 0.9265
##      Pos Pred Value : 0.7579
##      Neg Pred Value : 0.9384
##      Precision : 0.7579
##      Recall : 0.7908
##      F1 : 0.7740
##      Prevalence : 0.2253
##      Detection Rate : 0.1782
##      Detection Prevalence : 0.2351
##      Balanced Accuracy : 0.8587
##
##      'Positive' Class : 1
##
```

Considero que este resultado es bastante bueno: se tiene una exactitud del **90%**, un kappa ligeramente superior a **0.7**, F1 cerca de **0.8** ...

A pesar de estos resultados, he pensado intentar 2 técnicas más por ver si se pueden mejorar, aunque sea ligeramente. Vamos a intentar utilizar la validación cruzada que ofrece esta librería, y también utilizar una búsqueda en rejilla con diferentes parámetros.

- Validación cruzada

Vamos a usar la validación cruzada que ofrece Liblinear para comparar su resultado con lo que ya he obtenido. Como se indica en la documentación (y se ha visto en tareas anteriores de este máster), la métrica exactitud no es la más adecuada cuando se tiene un dataset muy desbalanceado. Pero probamos igualmente:

```
# Find the best model with the best cost parameter via 10-fold cross-validations
system.time({

if (crossValidation) {

  print("Trying cross validation")

  tryTypes <- c(1,2,3,5)
  tryCosts <- c(0.1,1,10,100)

  bestType <- NA
  bestCost <- NA
  bestAcc <- 0

  for(ty in tryTypes){
    cat("Results for type = ",ty,"\n",sep="")
    for(co in tryCosts){
      acc=LiblinearR(data = posts_freq_train_mat, target = posts_train_labels,
                     type = ty, cost = co, bias = 1, cross = 10, verbose = FALSE)

      cat("Results for C=",co," : ",acc," accuracy.\n",sep="")

      if(acc>bestAcc){
        bestCost <- co
        bestAcc <- acc
        bestType <- ty
      }
    }
  }
}
```

```

    }
  }
})

[1] "Trying cross validation"
Results for type = 1
Results for C=0.1 : 0.8913648 accuracy.
Results for C=1 : 0.888809 accuracy.
Results for C=10 : 0.8868789 accuracy.
Results for C=100 : 0.8806091 accuracy.
Results for type = 2
Results for C=0.1 : 0.8906726 accuracy.
Results for C=1 : 0.8914979 accuracy.
Results for C=10 : 0.8908324 accuracy.
Results for C=100 : 0.8909255 accuracy.
Results for type = 3
Results for C=0.1 : 0.8930687 accuracy.
Results for C=1 : 0.8951054 accuracy.
Results for C=10 : 0.8917376 accuracy.
Results for C=100 : 0.8881301 accuracy.
Results for type = 5
Results for C=0.1 : 0.88821 accuracy.
Results for C=1 : 0.8899804 accuracy.
Results for C=10 : 0.8864396 accuracy.
Results for C=100 : 0.8861334 accuracy.

      user  system elapsed
525.389    0.190  525.730

```

Nota:

Por curiosidad, intenté comparar regresión logística (type = 0) con SVM. Después de una hora no había terminado. En cambio, como se puede ver, los 4 tipos de SVM se ejecutaron en menos de 10 minutos.

```

if (crossValidation) {
  print("Cross validation result: ")

  cat("Best model type is:",bestType,"\n")
  cat("Best cost is:",bestCost,"\n")
  cat("Best accuracy is:",bestAcc,"\n")
}

[1] "Cross validation result:" Best model type is: 3 Best cost is: 1 Best accuracy is: 0.8951054

```

Después de las pruebas que se han ido realizando, el resultado no es sorprendente: el mejor resultado se obtiene con type 3 y coste 1.

- Búsqueda en rejilla

```

# Find the best model combining type, cost, bias, and weights

system.time({
  if (gridSearch) {
    print("Doing grid search")

    tryTypes <- c(1,2,3,5)
    tryCosts <- c(0.1,1,10,100)
    tryBias <- c(-1,1,10)
    tryWeights <- list(c(1,2),c(1,3),c(1,5),c(1,10))

    bestType <- NA
    bestCost <- NA
    bestBias <- NA
  }
})

```

```

bestWeights <- NA

bestAcc <- 0
bestKappa <- 0

#
for(ty in tryTypes) {
  cat("Results for type = ", ty, "\n", sep="")
  for(co in tryCosts) {
    for(bi in tryBias) {
      for(w in tryWeights) {
        w <- setNames(w, c("0", "1"))
        liblinear_svm_model <- LiblinearR(data = posts_freq_train_mat,
                                           target = posts_train_labels,
                                           type = ty,
                                           cost = co,
                                           bias = bi,
                                           wi = w)

        prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)
        cm <- confusionMatrix(reference = as.factor(posts_test_labels), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")
        acc <- cm$overall[1]
        kap <- cm$overall[2]
        cat("Results for C = ", co, " bias = ", bi, " weights = ", w, ": ", acc, " accuracy, ", kap, "
kappa.\n", sep="")

        if(kap > bestKappa){ # kappa as criteria
          bestType <- ty
          bestCost <- co
          bestBias <- bi
          bestWeights <- w
          bestAcc <- acc
          bestKappa <- kap
        }
      }
    }
  }
}

})

[1] "Doing grid search"

Results for type = 1
Results for C = 0.1 bias = -1 weights = 12: 0.8566693 accuracy, 0.6305961 kappa.
Results for C = 0.1 bias = -1 weights = 13: 0.8464457 accuracy, 0.6164794 kappa.
...
Results for C = 100 bias = 10 weights = 15: 0.8740415 accuracy, 0.5904935 kappa.
Results for C = 100 bias = 10 weights = 110: 0.5404553 accuracy, 0.2265866 kappa.
Results for type = 2
Results for C = 0.1 bias = -1 weights = 12: 0.8561901 accuracy, 0.6297379 kappa.
Results for C = 0.1 bias = -1 weights = 13: 0.8486422 accuracy, 0.6214468 kappa.
...
Results for C = 100 bias = 10 weights = 15: 0.8797524 accuracy, 0.6845577 kappa.
Results for C = 100 bias = 10 weights = 110: 0.7922923 accuracy, 0.5354072 kappa.
Results for type = 3
Results for C = 0.1 bias = -1 weights = 12: 0.8683706 accuracy, 0.6568531 kappa.
Results for C = 0.1 bias = -1 weights = 13: 0.8549121 accuracy, 0.6349446 kappa.
...
Results for C = 100 bias = 10 weights = 15: 0.6742013 accuracy, 0.3672031 kappa.
Results for C = 100 bias = 10 weights = 110: 0.8617412 accuracy, 0.5274452 kappa.
Results for type = 5
Results for C = 0.1 bias = -1 weights = 12: 0.8582668 accuracy, 0.6348784 kappa.
Results for C = 0.1 bias = -1 weights = 13: 0.8482029 accuracy, 0.6219869 kappa.
...
Results for C = 100 bias = 10 weights = 15: 0.8602236 accuracy, 0.6393691 kappa.
Results for C = 100 bias = 10 weights = 110: 0.779393 accuracy, 0.5065301 kappa.

##      user  system elapsed
##  799.877    0.860  799.273

if (gridSearch) {
  print("gridSearch result: ")

```

```

cat("Best model type is:",bestType,"\n")
cat("Best cost is:",bestCost,"\n")
cat("Best bias is:",bestBias,"\n")
cat("Best weights are:",bestWeights,"\n")
cat("Best accuracy is:",bestAcc,"\n")
cat("Best kappa is:",bestKappa,"\n")
}

```

```
[1] "gridSearch result:"
```

```

Best model type is: 2
Best cost is: 10
Best bias is: 10
Best weights are: 1 2
Best accuracy is: 0.8982428
Best kappa is: 0.7103011

```

Al hacer esta búsqueda en rejilla, se obtiene un modelo algo diferente, pero el resultado es prácticamente idéntico: exactitud del 90%, y kappa 0.71.

Dado que este modelo lo considero más “complicado” (coste 10 y bias 10, en lugar de los valores por defecto), y su resultado es prácticamente igual, me quedo con el obtenido anteriormente (type 3, pesos 1/2).

Pasamos ahora a entrenar SVM con el dataset completo (más de 600k observaciones).

3. Entrenamiento con el dataset total

Antes de procesar el dataset total, se ha preferido trabajar primero con el dataset de 100k observaciones, y así aplicar lo aprendido a este dataset (por ejemplo, viendo el resultado de la validación cruzada y de la búsqueda en rejilla, creo que puedo confiar en los resultados que se obtienen con type 3 y con coste 1).

Dado que el código es el mismo, se muestran solo las partes que muestran información relativa a este dataset:

Procesado del dataset

```

#Convert class into a factor
hate_raw$label <- factor(hate_raw$label)
#check_text(hate_raw$post)

system.time({
  hate <- preprocess_posts(hate, hate_raw)
})
## [1] "Removed references to users (@)."
```

user	system elapsed
420.294	0.328 420.632

```

## [1] "Removed non ascii and emoticons."
## [1] "Removed lines with empty posts."
# Number of rows deleted:
obs_removed <- nrow(hate_raw)-nrow(hate)

```

```
cat("Se han eliminado ", obs_removed, " líneas al procesar el dataset.")
## Se han eliminado 3406 líneas al procesar el dataset.

rm(hate_raw)
```

Corpus de los textos

Ya se puede proceder a la creación del objeto corpus con todas los mensajes:

```
#create corpus
system.time({
  posts_corpus <- VCorpus(VectorSource(hate$post))
})
## user system elapsed
## 22.650 1.348 23.998

print(posts_corpus)
#<<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 613144
```

Después de procesar el corpus y realizar la tokenización, se crean los conjuntos de entrenamiento y de test:

```
#Set seed to make the process reproducible
set.seed(123)

result <- train_test_split(hate, posts_dtm, 0.75)
## train dtm nrow: 459859
## test dtm nrow: 153285
## length of train labels: 459859
## length of test labels: 153285
#create training set
posts_dtm_train <- result$dtm_train

#create testing set
posts_dtm_test <- result$dtm_test

#create labels sets
posts_train_labels <- result$train_labels
posts_test_labels <- result$test_labels

rm(result)
rm(posts_dtm)
```

Comprobar si se mantiene la (des)proporción mensajes no de odio/mensajes de odio:

```
prop.table(table(posts_train_labels))
## posts_train_labels
## 0 1
## 0.96319524 0.03680476

prop.table(table(posts_test_labels))
## posts_test_labels
## 0 1
## 0.96319927 0.03680073
```

Se genera el listado de palabras más frecuentes, y lo utilizamos para limitar el número de columnas/features tanto del conjunto de entrenamiento como del de test:

```
dim(posts_dtm_train)
## [1] 459859 361649
```

```
posts_dtm_freq_train <- posts_dtm_train[, posts_freq_words_train]
dim(posts_dtm_freq_train)
## [1] 459859 16845
dim(posts_dtm_test)
## [1] 153285 361649
posts_dtm_freq_test <- posts_dtm_test[, posts_freq_words_train]
dim(posts_dtm_freq_test)
## [1] 153285 16845
```

Se pasa de tener 361649 columnas/features a 16845, tanto en el conjunto de entrenamiento como en el de test.

Entrenamiento del modelo y evaluación

Necesitamos convertir las DTMs en matrices para poder entrenar el modelo. Dado el tamaño del dataset, y las limitaciones físicas del equipo en el que se realiza este proceso, se procesa en lotes las DTMs, y posteriormente se juntan las matrices para obtener la total.

```
chunk_size <- 10000

system.time({
  chunk_list_train <- creat_sparse_mat_in_chunks(posts_dtm_freq_train, chunk_size)

  posts_freq_train_mat_chunks <- do.call(rbind, chunk_list_train)

  rm(chunk_list_train)
})
## chunk 1 processed.
## chunk 2 processed.
## chunk 3 processed.
...
## chunk 45 processed.
## chunk 46 processed.
## user system elapsed
## 57.498 18.592 77.132

system.time({
  chunk_list_test <- creat_sparse_mat_in_chunks(posts_dtm_freq_test, chunk_size)

  posts_freq_test_mat_chunks <- do.call(rbind, chunk_list_test)

  rm(chunk_list_test)
})
## chunk 1 processed.
## chunk 2 processed.
## chunk 3 processed.
...
## chunk 14 processed.
## chunk 15 processed.
## chunk 16 processed.
## user system elapsed
## 17.727 4.896 23.006

posts_freq_train_mat <- posts_freq_train_mat_chunks
rm(posts_freq_train_mat_chunks)

posts_freq_test_mat <- posts_freq_test_mat_chunks
rm(posts_freq_test_mat_chunks)
```

Entrenamiento

```
system.time({
  liblinear_svm_model <- LiblinearR(data=posts_freq_train_mat, target=posts_train_labels, type=3) # cost = 1
})
## user system elapsed
## 10.446 0.091 10.543
```

```
#liblinear_svm_model
```

Predicción:

```
# prediction
system.time({

  prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)

})
##      user  system elapsed
##    0.223    0.000    0.227

table(as.factor(prediction_liblinear$predictions))
##
##      0      1
## 150872  2413
```

Evaluación del resultado:

```
# confusion matrix
confusionMatrix(reference = as.factor(posts_test_labels), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")

## Confusion Matrix and Statistics
##
##      Reference
## Prediction  0      1
##      0 146861  4011
##      1   783  1630
##
##              Accuracy : 0.9687
##              95% CI : (0.9678, 0.9696)
##      No Information Rate : 0.9632
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.3913
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##      Sensitivity : 0.28896
##      Specificity : 0.99470
##      Pos Pred Value : 0.67551
##      Neg Pred Value : 0.97341
##      Precision : 0.67551
##      Recall : 0.28896
##      F1 : 0.40477
##      Prevalence : 0.03680
##      Detection Rate : 0.01063
##      Detection Prevalence : 0.01574
##      Balanced Accuracy : 0.64183
##
##      'Positive' Class : 1
##
```

Se obtiene un resultado solo aceptable. Kappa podría ser mejor, ciertamente. Se intentará encontrar alguna relación de pesos que mejore la detección los mensajes de odio (y por tanto de kappa).

Nota:

Al estar utilizando matrices dispersas para poder manejar un dataset de este tamaño, no podemos utilizar la función heurística de LiblineaR para calcular un coste óptimo. De todas maneras, después de varias pruebas, se ha visto que

los pesos ayudan a mejorar el resultado mucho más que utilizar un coste distinto de 1.

- Mejora del modelo utilizando pesos

Se ha probado la validación cruzada y la búsqueda en rejilla. El mejor resultado obtenido se ha encontrado gracias a esta última:

```
[1] "gridSearch result: "  
Best model type is: 3  
Best cost is: 1  
Best weights are: 1 3  
Best accuracy is: 0.9619076  
Best kappa is: 0.5098516
```

Así que asignamos los pesos 1/3 (type 3 y coste 1 como siempre):

Entrenamiento:

```
system.time({  
  liblinear_svm_model_weights <- LiblinearR(data = posts_freq_train_mat, target = posts_train_labels,  
                                             type = 3,  
                                             wi = class_weights)  
})  
##      user  system elapsed  
## 20.398    0.071   20.498
```

Predicción:

```
# prediction  
system.time({  
  prediction_liblinear_weights <- predict(liblinear_svm_model_weights, posts_freq_test_mat)  
})  
##      user  system elapsed  
##  0.215    0.008    0.248
```

Evaluación del resultado:

```
#Confusion matrix  
confusionMatrix(reference = as.factor(posts_test_labels),  
                 data = as.factor(prediction_liblinear_weights$predictions),  
                 positive="1",  
                 mode = "everything")  
## Confusion Matrix and Statistics  
##  
##              Reference  
## Prediction      0      1  
##      0 144160  2355  
##      1   3484 3286  
##  
##              Accuracy : 0.9619  
##              95% CI : (0.9609, 0.9629)  
##      No Information Rate : 0.9632  
##      P-Value [Acc > NIR] : 0.9963  
##  
##              Kappa : 0.5099  
##  
##      McNemar's Test P-Value : <2e-16  
##  
##              Sensitivity : 0.58252  
##              Specificity : 0.97640  
##      Pos Pred Value : 0.48538  
##      Neg Pred Value : 0.98393  
##              Precision : 0.48538  
##              Recall : 0.58252  
##              F1 : 0.52953  
##      Prevalence : 0.03680
```

```
##      Detection Rate : 0.02144
##      Detection Prevalence : 0.04417
##      Balanced Accuracy : 0.77946
##
##      'Positive' Class : 1
##
```

Se ha conseguido mejorar kappa de 0.39 a 0.51, mientras que la exactitud ha empeorado un poco solamente.

De todas formas, se puede ver fácilmente que tanto con el dataset de Hatemedia como con el total, se obtienen mejores resultados cuando se entrena el modelo con el subconjunto de 100k observaciones en lugar de con el dataset total, que está exageradamente desbalanceado.

3. Comparación de resultados y conclusiones.

Comparativa de los mejores resultados obtenidos con cada dataset:

(coste = 1 en todos, excepto en dataset 28k: coste = 0.1

freq = 100 en todos, excepto para Kaggle, freq = 50, HuggingFace y superdataset 50k, con freq = 20)

Dataset	Kaggle	HuggingFace	Hatemia pesos 1/3	Hatemia 100k pesos 1/2
Matriz de confusión	Reference Pred. No Yes No 666 145 Yes 153 279	Reference Pred. No Yes No 5002 694 Yes 645 1122	Reference Pred. No Yes No 138069 1583 Yes 1882 1183	Reference Pred. No Yes No 21145 947 Yes 954 1819
Exactitud	0.7603	0.8206	0.9757	0.9235
Kappa	0.4691	0.5083	0.3934	0.6138
F1	0.6519	0.6263	0.4058	0.6568
MCC	0.4677	0.5088	0.3935	0.6138
Dataset		Superdataset 600k pesos 1/3	Superdataset 100k pesos 1/2	Superdataset 50k pesos 1/2
Matriz de confusión		Reference Pred. No Yes No 144160 2355 Yes 3484 3286	Reference Pred. No Yes No 17967 1180 Yes 1425 4461	Reference Pred. No Yes No 6167 582 Yes 796 5059
Exactitud		0.9619	0.8959	0.8907
Kappa		0.5099	0.7065	0.7797
F1		0.5295	0.7740	0.8801
MCC		0.5121	0.7067	0.7801

Comparando todos los resultados hasta ahora, observo lo siguiente:

i) Cuanto mayor es el dataset, se obtiene una exactitud más alta.

Esto es lógico: al ser datasets muy desbalanceados, el modelo aprende muy bien a detectar la clase mayoritaria.

ii) Dado que se trata de datasets muy desbalanceados, la exactitud no es la métrica más importante, sino kappa. Y kappa es mejor cuando el dataset es reducido a 100k observaciones, y mejor todavía con un dataset de 50k observaciones.

iii) Con el mismo tamaño (es decir, con el mismo nivel de desbalanceo), también se obtiene mejores resultados con lo dataset resultante de mezclar los 3 datasets originales. Parece que la hipótesis original de que al tener diferentes datasets, SVM podría generalizar mejor y obtener mejores resultados con nuevos mensajes, se confirma.

iv) Juntando todo lo anterior, el mejor resultado es lógicamente el obtenido con el SVM entrenado con el dataset de 50k observaciones obtenido del dataset total. Este dataset contiene 28k mensajes no de odio, y 22k de odio.

v) He añadido al final el cálculo del coeficiente de correlación de Matthews, pero no aporta más información de la que ya da kappa (son valores prácticamente iguales).

Me gustaría resaltar de nuevo que los datasets contienen no solo comentarios en castellano, sino también en catalán, y en el español de México y de Chile, y que los tamaños (50k, 100k y 600k observaciones), me parecen bastante realistas (lo suficiente como para dar validez en principio a las conclusiones extraídas).

4. Fase 3 – Clasificación de comentarios nuevos no etiquetados

Hasta ahora se ha utilizado solamente los datasets encontrados en internet. Se trata en esta fase de clasificar comentarios nuevos, no etiquetados, con los modelos entrenados en la fase anterior.

En concreto se pretende lo siguiente:

- Obtener comentarios nuevos
- Clasificarlos con los modelos anteriores
- Evaluar el resultado

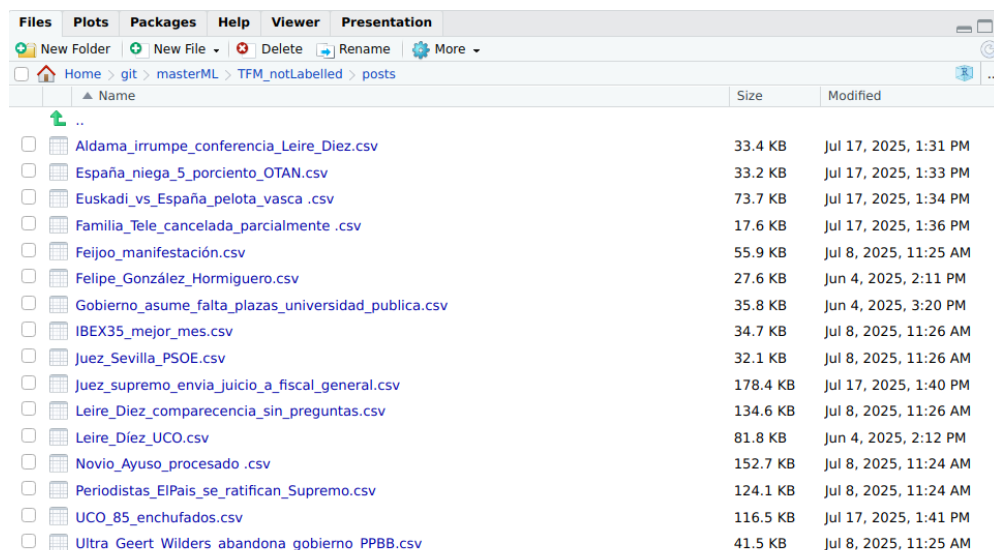
Dependiendo del resultado:

- Entrenar y evaluar SVM solo con los comentarios nuevos.
- Añadir los comentarios nuevos al dataset total, y entrenar SVM con este dataset “mejorado”
- Comparar los resultados

1. Obtención de comentarios nuevos no etiquetados

Como fuente de nuevos comentarios, se ha elegido la edición online de el periódico El Mundo. La razón principal ha sido la facilidad para obtenerlos (y su amplia difusión, lo que facilita recoger muchos comentarios por noticia). Para poder acceder a los comentarios de los lectores en otros periódicos, en la mayoría de los casos es obligatorio estar registrado. En cambio en este medio están disponibles sin ninguna barrera (no hace falta estar suscrito, y ni siquiera logarse).

En total se obtuvieron 3391 comentarios de 16 noticias:



Name	Size	Modified
..		
Aldama_irrumpe_conferencia_Leire_Diez.csv	33.4 KB	Jul 17, 2025, 1:31 PM
España_niega_5_por ciento_OTAN.csv	33.2 KB	Jul 17, 2025, 1:33 PM
Euskadi_vs_España_pelota_vasca .csv	73.7 KB	Jul 17, 2025, 1:34 PM
Familia_Tele_cancelada_parcialmente .csv	17.6 KB	Jul 17, 2025, 1:36 PM
Feijoo_manifestación.csv	55.9 KB	Jul 8, 2025, 11:25 AM
Felipe_González_Hormiguero.csv	27.6 KB	Jun 4, 2025, 2:11 PM
Gobierno_asume_falta_plazas_universidad_publica.csv	35.8 KB	Jun 4, 2025, 3:20 PM
IBEX35_mejor_mes.csv	34.7 KB	Jul 8, 2025, 11:26 AM
Juez_Sevilla_PSOE.csv	32.1 KB	Jul 8, 2025, 11:26 AM
Juez_supremo_envia_juicio_a_fiscal_general.csv	178.4 KB	Jul 17, 2025, 1:40 PM
Leire_Diez_comparecencia_sin_preguntas.csv	134.6 KB	Jul 8, 2025, 11:26 AM
Leire_Diez_UCO.csv	81.8 KB	Jun 4, 2025, 2:12 PM
Novio_Ayuso_procesado .csv	152.7 KB	Jul 8, 2025, 11:24 AM
Periodistas_ElPais_se_ratifican_Supremo.csv	124.1 KB	Jul 8, 2025, 11:24 AM
UCO_85_enchufados.csv	116.5 KB	Jul 17, 2025, 1:41 PM
Ultra_Geert_Wilders_abandona_gobierno_PPBB.csv	41.5 KB	Jul 8, 2025, 11:25 AM

Predominan claramente las noticias de ámbito político nacional, dado que son las noticias con más participación de los lectores.

Ejemplos de algunos de esos comentarios:

```
Andalu_ilustra0|En mi opinión, una de las mayores satisfacciones de la vida es ver que por mucho que imaginemos y elucubremos, la realidad siempre supera a la ficción.
tienesrazon|Casi el 80% de los españoles creían a Aldama frente a Sánchez cuando salió de la cárcel, bueno, los primeros los fiscales y jueces, si no no hubiera salido de la cárcel. Ahora seguro que son el 90%. Hasta Page ha pedido elecciones, como cualquier persona honesta. Los foreros rojos no porque no son honestos y además cobran de Ferraz la mayoría.
Goligo|@Objetivo_pero_no_impacial #55 Cerrar Aldama mente y sobreactúa. borrego detectado
...
```

El código R para hacer “web scraping” se encuentra disponible aquí:

https://github.com/fcamadi/masterML/blob/main/TFM_webscraping/Web_scraping_RSelenium_periodicos_online.Rmd

2. Clasificación de comentarios nuevos no etiquetados con modelos ya entrenados

En este paso se intentará clasificar los comentarios nuevos con los modelos entrenados anteriormente.

1. Leer CSVs con los comentarios de los lectores

```
source('hate_speech_03_common.R')
load_libraries()
```

Leemos los ficheros con los comentarios de los lectores:

```
# Set the directory containing the CSV files
directory <- "./posts"

# List all CSV files in the directory
csv_files <- list.files(path = directory, pattern = "*.csv", full.names = TRUE)

# Read all CSV files into a list of data frames
csv_data_list <- lapply(csv_files, function(file) read.csv(file, sep = "|", header = FALSE))

# Combine all data frames into a single data frame
posts_elmundo_June25 <- as.data.frame(do.call(rbind, csv_data_list))

cat("Number of posts by readers: ", nrow(posts_elmundo_June25))
## Number of posts by readers: 3391
```

Renombrar variables:

```
colnames(posts_elmundo_June25) <- c("author", "post")
```

Eliminar columna autor y añadir columna “label”:

```
posts_elmundo_June25$label <- NA
posts_elmundo_June25 <- posts_elmundo_June25[, -1]
```

2. Carga de los modelos guardados en la fase 2

Procedemos a cargar los dos mejores modelos de la fase anterior con los que se obtuvo un mejor resultado (tabla página 46).

Nota:

Dados los resultados obtenidos con ellos, se ha intentado probar también con un modelo entrenado con un dataset muy balanceado.

- Modelo “100k”:

```
# Load the model
svm_liblinear_100k <- readRDS("svm_liblinear_all_datasets_100k_freq100_weights_1_2.rds")
```

- Modelo “600k”:

```
# Load the model
svm_liblinear_600k <- readRDS("svm_liblinear_all_datasets_freq100_weights_1_3.rds")
```

- Modelo “mejor” (dataset 50k observaciones, type 3, freq 20, coste 0.1, bias 10, pesos 1/2):

```
# Load the model
svm_liblinear_50k_best <-
readRDS("svm_liblinear_all_datasets_50k_freq20_cost01_bias10_weights_1_2.rds")
```

3. Clasificación de comentarios no etiquetados

- Elegir comentarios de manera aleatoria

Elegimos aleatoriamente 2000 comentarios de los 3000. Con esos 2000 comentarios creamos 2 dataframes, subset1 y subset2:

```
set.seed(10)

# number of rows to select from the total (3391)
n <- 2000

# Randomly select rows
posts_2000 <- posts_elmundo_June25[sample(nrow(posts_elmundo_June25), n), ]

n <- 1000
# subset1
indices <- sample(nrow(posts_2000), n)

# first subset
subset1 <- posts_2000[indices, ]
# subset2
subset2 <- posts_2000[-indices, ]
```

- Clasificar mensajes con los dos (tres) modelos.

Elegimos 1000 comentarios (subset1) para realizar esta primera prueba de clasificación de comentarios nuevos no etiquetados:

```
train_vocab_100k <- colnames(svm_liblinear_100k$w)
subset1_sparse_matrix_100k <- process_unlabelled_posts(subset1, train_vocab_100k)
```

```
## [1] "Removed references to users (@)."
```

```
## [1] "Removed non ascii and emoticons."
```

```
## [1] "Removed lines with empty posts."
```

```
## [1] "#To lowercase"
```

```
## [1] "#Remove numbers"
```

```
## [1] "#Remove stopwords"
```

```
## [1] "#Remove punctuation signs"
```

```
## [1] "#Carry out the stemming"
```

```
## [1] "#Finally eliminate unneeded whitespace produced by previous steps"
```

```
## <<SimpleCorpus>>
```

```
## Metadata: corpus specific: 1, document level (indexed): 0
```

```
## Content: documents: 1000
```

```
train_vocab_600k <- colnames(svm_liblinear_600k$W)
```

```
subset1_sparse_matrix_600k <- process_unlabelled_posts(subset1, train_vocab_600k)
```

```
## [1] "Removed references to users (@)."
```

```
## [1] "Removed non ascii and emoticons."
```

```
## [1] "Removed lines with empty posts."
```

```
## [1] "#To lowercase"
```

```
## [1] "#Remove numbers"
```

```
## [1] "#Remove stopwords"
```

```
## [1] "#Remove punctuation signs"
```

```
## [1] "#Carry out the stemming"
```

```
## [1] "#Finally eliminate unneeded whitespace produced by previous steps"
```

```
## <<SimpleCorpus>>
```

```
## Metadata: corpus specific: 1, document level (indexed): 0
```

```
## Content: documents: 1000
```

```
train_vocab_50k_best <- colnames(svm_liblinear_50k_best$W)
```

```
subset1_sparse_matrix_50k_best <- process_unlabelled_posts(subset1, train_vocab_50k_best)
```

```
## [1] "Removed references to users (@)."
```

```
## [1] "Removed non ascii and emoticons."
```

```
## [1] "Removed lines with empty posts."
```

```
## [1] "#To lowercase"
```

```
## [1] "#Remove numbers"
```

```
## [1] "#Remove stopwords"
```

```
## [1] "#Remove punctuation signs"
```

```
## [1] "#Carry out the stemming"
```

```
## [1] "#Finally eliminate unneeded whitespace produced by previous steps"
```

```
## <<SimpleCorpus>>
```

```
## Metadata: corpus specific: 1, document level (indexed): 0
```

```
## Content: documents: 1000
```

Realizamos las predicciones con los 3 modelos:

```
predictions_100k <- predict(svm_liblinear_100k, subset1_sparse_matrix_100k)
```

```
#print(predictions_100k$predictions)
```

```
predictions_600k <- predict(svm_liblinear_600k, subset1_sparse_matrix_600k)
```

```
#print(predictions_600k$predictions)
```

```
predictions_50k_best <- predict(svm_liblinear_50k_best, subset1_sparse_matrix_50k_best)
```

```
#print(predictions_100k$predictions)
```

```
subset1$label_100k <- predictions_100k$predictions
```

```
subset1$label_600k <- predictions_600k$predictions
```

```
subset1$label_50k_best <- predictions_50k_best$predictions
```

Examinamos el resultado:

```
table(subset1$label_600k)
```

```
##
```

```
## 0 1
```

```
## 562 438
```



```
table(subset1$label_100k)
##
##    0    1
## 204 796

table(subset1$label_50k_best)
##
##    0    1
## 100 900
```

Hay claramente demasiados positivos/mensajes de odio. Curiosamente con el modelo entrenado con el dataset más balanceado (el dataset de 50k observaciones contiene 28k comentarios no de odio, y 22k comentarios de odio), el resultado es el peor.

4. Evaluación de los resultados

Para poder obtener las diferentes métricas, se ha procedido a clasificar “a mano” los comentarios obtenidos de El Mundo en junio de 2025.

Se ha añadido esa valoración en la columna “label”, y el resultado de llamar a la función confusionMatrix de caret para los 3 modelos es:

Matriz de confusión modelo “600k”:

```
#Confusion matrix
confusionMatrix(reference=as.factor(result1_1000$label),data=as.factor(result1_1000$label_600k),
                 positive="1", mode = "everything")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 545  24
##           1 333  98
##
##           Accuracy : 0.643
##           95% CI : (0.6124, 0.6727)
##           No Information Rate : 0.878
##           P-Value [Acc > NIR] : 1
##
##           Kappa : 0.2028
##
## Mcnemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.8033
##           Specificity : 0.6207
##           Pos Pred Value : 0.2274
##           Neg Pred Value : 0.9578
##           Precision : 0.2274
##           Recall : 0.8033
##           F1 : 0.3544
##           Prevalence : 0.1220
##           Detection Rate : 0.0980
##           Detection Prevalence : 0.4310
##           Balanced Accuracy : 0.7120
##
##           'Positive' Class : 1
##
```

Matriz de confusión modelo “100k”:

```
#Confusion matrix
confusionMatrix(reference = as.factor(result1_1000$label),
                 data = as.factor(result1_1000$label_100k),
                 positive="1",
                 mode = "everything")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 197    7
##           1 681 115
##
##           Accuracy : 0.312
##           95% CI : (0.2834, 0.3417)
##           No Information Rate : 0.878
##           P-Value [Acc > NIR] : 1
##
##           Kappa : 0.0494
##
##  Mcnemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.9426
##           Specificity : 0.2244
##           Pos Pred Value : 0.1445
##           Neg Pred Value : 0.9657
##           Precision : 0.1445
##           Recall : 0.9426
##           F1 : 0.2505
##           Prevalence : 0.1220
##           Detection Rate : 0.1150
##           Detection Prevalence : 0.7960
##           Balanced Accuracy : 0.5835
##
##           'Positive' Class : 1
##
```

Matriz de confusión modelo “50k mejor”:

```
#Confusion matrix
confusionMatrix(reference = as.factor(result1_1000$label),
                 data = as.factor(result1_1000$label_50k_best),
                 positive="1",
                 mode = "everything")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0  99    2
##           1 779 120
##
##           Accuracy : 0.219
##           95% CI : (0.1937, 0.2459)
##           No Information Rate : 0.878
##           P-Value [Acc > NIR] : 1
##
##           Kappa : 0.0258
##
##  Mcnemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.9836
##           Specificity : 0.1128
##           Pos Pred Value : 0.1335
##           Neg Pred Value : 0.9802
##           Precision : 0.1335
##           Recall : 0.9836
##           F1 : 0.2351
##           Prevalence : 0.1220
##           Detection Rate : 0.1200
##           Detection Prevalence : 0.8990
##           Balanced Accuracy : 0.5482
##
##           'Positive' Class : 1
##
```

Creo que no se trata solo del nivel de balanceo del dataset. Creo que el vocabulario influye tanto o más que el grado de balanceo. Cuanto mayor es el vocabulario con el que se ha entrenado un modelo, mejor es el resultado que este devuelve.

3. Clasificación de comentarios nuevos con modelos entrenados con ellos

Ahora se va a entrenar SVM únicamente con los comentarios obtenidos en el primer apartado de esta sección.

Inicialmente se realizó el entrenamiento con un dataset de 2000 comentarios, a los que se añadió posteriormente otros 500. Se comprobó que el resultado mejoró bastante al añadir estos comentarios adicionales.

Con el dataset de entrenamiento con 2000 comentarios el mejor resultado obtenido fue:

```
[1] "gridSearch result:"
Best model type is: 3
Best cost is: 0.1
Best bias is: 10
Best weights are: 1 5
Best accuracy is: 0.842
Best kappa is: 0.2943909 (0.25 con valores por defecto)
```

Entrenando SVM con un dataset con 500 comentarios adicionales, el resultado es mucho mejor:

```
#Confusion matrix
confusionMatrix(reference = as.factor(posts_test$label), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##      0  435  23
##      1   18  24
##
##              Accuracy : 0.918
##              95% CI : (0.8904, 0.9405)
##      No Information Rate : 0.906
##      P-Value [Acc > NIR] : 0.2013
##
##              Kappa : 0.4945
##
##  Mcnemar's Test P-Value : 0.5322
##
##      Sensitivity : 0.5106
##      Specificity : 0.9603
##      Pos Pred Value : 0.5714
##      Neg Pred Value : 0.9498
##      Precision : 0.5714
##      Recall : 0.5106
##      F1 : 0.5393
##      Prevalence : 0.0940
##      Detection Rate : 0.0480
##      Detection Prevalence : 0.0840
##      Balanced Accuracy : 0.7355
##
##      'Positive' Class : 1
##
```

Realizando una búsqueda en rejilla:

```
gridSearch <- TRUE

# Find the best model combining type, cost, bias, and weights
#
system.time({
  if (gridSearch) {
    tryTypes <- c(1,2,3,5)
    tryCosts <- c(0.1,1,10,100)
    tryBias <- c(-1,1,10)
    tryWeights <- list(c(1,2),c(1,3),c(1,5),c(1,10))

    grid_search_result <- grid_search(posts_freq_train_mat, posts_training$label, posts_test$label,
                                     tryTypes, tryCosts, tryBias, tryWeights)
  }
})

## [1] "Doing grid search ..."
## Results for type = 1
## Results for C = 0.1 bias = -1 weights = 12: 0.89 accuracy, 0.4532368 kappa.
## Results for C = 0.1 bias = -1 weights = 13: 0.88 accuracy, 0.4354323 kappa.
...
## Results for type = 2
## Results for C = 0.1 bias = -1 weights = 12: 0.888 accuracy, 0.4477535 kappa.
## Results for C = 0.1 bias = -1 weights = 13: 0.88 accuracy, 0.4354323 kappa.
...
## Results for type = 3
## Results for C = 0.1 bias = -1 weights = 12: 0.874 accuracy, 0.3836581 kappa.
## Results for C = 0.1 bias = -1 weights = 13: 0.862 accuracy, 0.3746375 kappa.
...
## Results for type = 5
## Results for C = 0.1 bias = -1 weights = 12: 0.87 accuracy, 0.3432221 kappa.
## Results for C = 0.1 bias = -1 weights = 13: 0.844 accuracy, 0.3369828 kappa.
...
## Results for C = 100 bias = 10 weights = 110: 0.922 accuracy, 0.5628979 kappa.
## [1] "Grid search finished."
```

El mejor resultado obtenido es:

```
if (gridSearch) {
  print(grid_search_result)
}

## $bestType
## [1] 1
##
## $bestCost
## [1] 100
##
## $bestBias
## [1] 1
##
## $bestWeights
## 0 1
## 1 2
##
## $cm
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 433  17
##           1  20  30
##
##               Accuracy : 0.926
##               95% CI   : (0.8994, 0.9474)
##               No Information Rate : 0.906
##               P-Value [Acc > NIR] : 0.06912
##
##               Kappa : 0.5776
##
## Mcnemar's Test P-Value : 0.74231
```

```
##
##          Sensitivity : 0.6383
##          Specificity : 0.9558
##          Pos Pred Value : 0.6000
##          Neg Pred Value : 0.9622
##          Precision : 0.6000
##          Recall : 0.6383
##          F1 : 0.6186
##          Prevalence : 0.0940
##          Detection Rate : 0.0600
##          Detection Prevalence : 0.1000
##          Balanced Accuracy : 0.7971
##
##          'Positive' Class : 1
##
```

En esta ocasión el mejor resultado se obtiene con un SVM de type 1.
Parece claro que aumentando el conjunto de entrenamiento, se podría mejorar los resultados todavía más.

4. Mejora de los datasets con comentarios nuevos

Se pretende comprobar si "mejorando" el dataset total obtenido con los datasets de Hatemedia, HuggingFace y Kaggle, añadiendo los comentarios obtenidos en esta fase, se obtiene un resultado mejor que en el apartado anterior. (Se podría pensar que el resultado debería ser intermedio entre los resultados obtenidos con solo los datasets obtenidos de internet, y el dataset del apartado anterior que solo contiene los comentarios nuevos).

1. Procesado de los datasets

Para no repetir código, se muestran solo algunas fragmentos:

Variables iniciales:

```
complete_dataset <- params$complete
crossValidation <- params$crossValidation
gridSearch <- params$gridSearch

#if no complete dataset, number of no hate messages to pick from the total dataset
size <- 28000 # number of random rows of no hate messages -> + 22k of hate posts

# min. freq (the lower the size, the lower the freq)
freq <- 10

# Assign higher weight to the minority class
class_weights <- c("0" = 1, "1" = 2)

random_seed <- 123

print(getwd())
## [1] "/home/francd/git/masterML/TFM_notLabelled"
```

Comprobar proporción de los mensajes en este dataset una vez realizado el paso del “downsampling”:

```
table(hate_raw$label)
##
##      0      1
## 28000 22568

prop.table(table(hate_raw$label))
##
##      0      1
## 0.5537099 0.4462901
```

Ahora añadimos los 2000 mensajes de El Mundo obtenidos en junio 2025:

```
subset1_labelled <- read.csv("subset1_labelled.csv", sep = "|", header = TRUE)
subset2_labelled <- read.csv("subset2_labelled.csv", sep = "|", header = TRUE)
```

Añadimos 500 más comentarios clasificados:

```
subset3_labelled <- read.csv("subset3_labelled.csv", sep = "|", header = TRUE)
```

Nota:

El proceso de clasificar los comentarios de los lectores es bastante laborioso (y subjetivo, por supuesto). He querido hacerlo en dos fases para ver cómo influye el aumento del tamaño de conjunto de entrenamiento en el resultado final.

Ahora ya se puede crear un dataset total con todos los datasets disponibles:

```
hate_raw <- rbind(subset1_labelled, hate_raw, subset2_labelled, subset3_labelled)

dim(hate_raw)
## [1] 53068      2
```

Disponemos al final de un dataset con 53k comentarios, bastante balanceado (55.37% de mensajes no de odio, y 44.63% de mensajes de odio).

También leemos los comentarios de test igualmente ya etiquetados:

```
posts_test_raw <- read.csv("posts_test_labelled.csv", sep = "|", header = TRUE)
```

Comprobar proporción de los mensajes en estos datasets:

- dataset entrenamiento:

```
table(hate_raw$label)
##
##      0      1
## 30180 22888

prop.table(table(hate_raw$label))
##
##      0      1
## 0.5687043 0.4312957
```

Añadir estos 2500 comentarios nuevos modifica muy ligeramente la proporción ya existente en el dataset de entrenamiento.

- dataset test:

```
table(posts_test_raw$label)
```

```
##
##    0    1
## 453  47

prop.table(table(posts_test_raw$label))
##
##    0    1
## 0.906 0.094
```

El dataset que vamos a clasificar con SVM está muy desbalanceado (según la valoración particular del autor de este trabajo).

A continuación se preparan los datasets con el procedimiento habitual: limpieza (eliminar referencias a otros usuarios, emoticonos, etc.), creación del corpus, procesado del corpus (eliminar mayúsculas, números ..), tokenización, búsqueda de las palabras más frecuentes ...

Al final del proceso se obtiene:

```
dim(posts_dtm_train)
## [1] 52921 89116
posts_dtm_freq_train <- posts_dtm_train[, posts_freq_words_train]
dim(posts_dtm_freq_train)
## [1] 52921 15187

#dim(posts_dtm_test)
#posts_dtm_freq_test <- posts_dtm_test[, posts_freq_words_train]
dim(posts_dtm_freq_test)
## [1] 500 15187
```

Y ya se puede pasar a entrenar SVM.

2. Entrenamiento del SVM y evaluación del resultado

Necesitamos convertir las DTMs en matrices para poder entrenar el modelo. Dado el tamaño del dataset, y las limitaciones físicas del equipo en el que se realiza este proceso, se procesa en lotes las DTMs, y posteriormente se juntan las matrices para obtener la matriz total.

```
chunk_size <- 10000

#Training
system.time({
  chunk_list_train <- creat_sparse_mat_in_chunks(posts_dtm_freq_train, chunk_size)

  posts_freq_train_mat_chunks <- do.call(rbind, chunk_list_train)

  rm(chunk_list_train)
})
## chunk 1 processed.
...
## chunk 6 processed.
## user system elapsed
## 5.598 1.496 7.095

#Test
system.time({
  chunk_list_test <- creat_sparse_mat_in_chunks(posts_dtm_freq_test, chunk_size)

  posts_freq_test_mat_chunks <- do.call(rbind, chunk_list_test)
```

```
rm(chunk_list_test)
})
## chunk 1 processed.
## user system elapsed
## 0.027 0.020 0.046

posts_freq_train_mat <- posts_freq_train_mat_chunks
rm(posts_freq_train_mat_chunks)

posts_freq_test_mat <- posts_freq_test_mat_chunks
rm(posts_freq_test_mat_chunks)
```

Entrenamiento (coste = 1)

```
system.time({
  liblinear_svm_model <- Liblinear(data = posts_freq_train_mat, target = hate$label, type = 3) # cost = 1
})
## user system elapsed
## 2.583 0.000 2.584

#liblinear_svm_model
```

Predicción:

```
# prediction
system.time({
  prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)
})
## user system elapsed
## 0.003 0.000 0.003

table(as.factor(prediction_liblinear$predictions))
##
## 0 1
## 454 46
```

Evaluación del resultado:

```
# confusion matrix
confusionMatrix(reference = as.factor(posts_test$label), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")
## Confusion Matrix and Statistics
##
##          Reference
## Prediction  0    1
##      0 430  24
##      1  23  23
##
##              Accuracy : 0.906
##              95% CI : (0.877, 0.9301)
##      No Information Rate : 0.906
##      P-Value [Acc > NIR] : 0.5387
##
##              Kappa : 0.4428
##
##  Mcnemar's Test P-Value : 1.0000
##
##      Sensitivity : 0.4894
##      Specificity : 0.9492
##      Pos Pred Value : 0.5000
##      Neg Pred Value : 0.9471
##      Precision : 0.5000
##      Recall : 0.4894
##      F1 : 0.4946
##      Prevalence : 0.0940
##      Detection Rate : 0.0460
##      Detection Prevalence : 0.0920
##      Balanced Accuracy : 0.7193
##
##      'Positive' Class : 1
##
```


El resultado es ligeramente peor que el obtenido entrenando SVM solo con los comentarios nuevos (se obtuvo un kappa de 0.49).

Utilizamos búsqueda en rejilla para encontrar un modelo mejor que con los parámetros por defecto:

```
# Find the best model combining type, cost, bias, and weights
#
system.time({
  if (gridSearch) {
    tryTypes <- c(1,3,5) # type 2 gives the worst results
    tryCosts <- c(1,10)
    tryBias <- c(1,10)
    tryWeights <- list(c(1,2),c(1,3),c(1,5)) #,c(1,10))

    grid_search_result <- grid_search(posts_freq_train_mat, hate$label, posts_test$label,
                                     tryTypes, tryCosts, tryBias, tryWeights)
  }
})
## [1] "Doing grid search ..."
## Results for type = 1
## Results for C = 1 bias = 1 weights = 12: 0.902 accuracy, 0.4966305 kappa.
## Results for C = 1 bias = 1 weights = 13: 0.894 accuracy, 0.4895994 kappa.
## Results for C = 1 bias = 1 weights = 15: 0.872 accuracy, 0.448371 kappa.
## Results for C = 1 bias = 10 weights = 12: 0.906 accuracy, 0.5089845 kappa.
..
## Results for type = 3
## Results for C = 1 bias = 1 weights = 12: 0.896 accuracy, 0.4788535 kappa.
## Results for C = 1 bias = 1 weights = 13: 0.884 accuracy, 0.4856879 kappa.
## Results for C = 1 bias = 1 weights = 15: 0.866 accuracy, 0.4343891 kappa.
..
## Results for type = 5
## Results for C = 1 bias = 1 weights = 12: 0.906 accuracy, 0.5089845 kappa.
## Results for C = 1 bias = 1 weights = 13: 0.896 accuracy, 0.4952828 kappa.
..
## Results for C = 10 bias = 10 weights = 13: 0.894 accuracy, 0.4895994 kappa.
## Results for C = 10 bias = 10 weights = 15: 0.88 accuracy, 0.4521749 kappa.
## [1] "Grid search finished."
## user system elapsed
## 134.618 0.160 134.787
```

Mejor resultado obtenido:

```
if (gridSearch) {
  print(grid_search_result)
}
## $bestType
## [1] 3
##
## $bestCost
## [1] 10
##
## $bestBias
## [1] 10
##
## $bestWeights
## 0 1
## 1 5
##
## $cm
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0    1
##           0 427  16
##           1  26  31
##
##
##           Accuracy : 0.916
##           95% CI : (0.8882, 0.9388)
```

```

##      No Information Rate : 0.906
##      P-Value [Acc > NIR] : 0.2487
##
##      Kappa : 0.5498
##
##      McNemar's Test P-Value : 0.1649
##
##      Sensitivity : 0.6596
##      Specificity : 0.9426
##      Pos Pred Value : 0.5439
##      Neg Pred Value : 0.9639
##      Precision : 0.5439
##      Recall : 0.6596
##      F1 : 0.5962
##      Prevalence : 0.0940
##      Detection Rate : 0.0620
##      Detection Prevalence : 0.1140
##      Balanced Accuracy : 0.8011
##
##      'Positive' Class : 1
##

```

El resultado vuelve a ser ligeramente peor que utilizando únicamente los comentarios de El Mundo. Pero en ambos casos, son resultados muchísimo mejores que los obtenidos al entrenar SVM únicamente con los datasets encontrados en internet.

5. Tabla comparativa de resultados y conclusiones

En la página siguiente se pueden encontrar todos los resultados de clasificar los 500 comentarios nuevos con todos los modelos (entrenados con los datasets de internet -primera fila-, con los comentarios nuevos solamente -segunda fila-, y con los datasets “mejorados” -tercera fila-).

Modelos entrenados con datasets de internet	Superdataset "600k" type 3 valores por defecto	Superdataset "100k" type 3 valores por defecto	Superdataset 50k "mejor"	
Matriz de confusión	Reference Pred. No Yes No 545 24 Yes 333 98	Reference Pred. No Yes No 197 7 Yes 681 115	Reference Pred. No Yes No 99 2 Yes 779 120	
Exactitud	0.643	0.312	0.219	
Kappa	0.2028	0.0494	0.0258	
F1	0.3544	0.2505	0.2351	
Modelos entrenados con nuevos comentarios	2000 comentarios type 3 valores por defecto	2000 comentarios "mejor"	2500 comentarios type 3 valores por defecto	2500 comentarios "mejor"
Matriz de confusión	Reference Pred. No Yes No 417 31 Yes 36 16	Reference Pred. No Yes No 397 23 Yes 56 24	Reference Pred. No Yes No 435 23 Yes 18 24	Reference Pred. No Yes No 433 17 Yes 20 30
Exactitud	0.8660	0.8420	0.9180	0.9260
Kappa	0.2491	0.2944	0.4995	0.5776
F1	0.3232	0.3780	0.5393	0.6186
Modelos entrenados con "datasets mejorados"	Superdataset "600k" type 3 valores por defecto	Superdataset "600k mejor"	Superdataset "50k" type 3 valores por defecto	Superdataset "50k mejor"
Matriz de confusión	Reference Pred. No Yes 444 39 9 8	Reference Pred. No Yes 423 24 30 23	Reference Pred. No Yes No 430 24 Yes 23 23	Reference Pred. No Yes No 427 16 Yes 26 31
Exactitud	0.9040	0.8920	0.9060	0.9160
Kappa	0.2106	0.4002	0.4428	0.5498
F1	0.2500	0.4600	0.4946	0.5962

"mejor": modelo encontrado con la búsqueda en rejilla

"dataset mejorado": dataset construido con los 3 datasets encontrados en internet más 2500 comentarios de noticias de El Mundo de junio de 2025

Conclusiones

- Grado de balanceo vs frecuencia mínima

A priori, puede parecer que el grado de balanceo es el factor más importante a la hora de influir en la calidad de los resultados. Pero dos resultados me han llevado a concluir de diferente manera:

- al clasificar los comentarios nuevos con los modelos entrenados con el superdataset hecho con los tres datasets de Hatemedia, HuggingFace y Kaggle, el resultado es mejor con el dataset más grande, que es el más desbalanceado (pero es el que ha sido construido usando un vocabulario más amplio).
- al realizar pruebas con diferentes frecuencias mínimas (a la hora de limitar las columnas/features de las DTMs) con el mismo dataset, independientemente de su grado de balanceo, cuanto menor es la frecuencia mínima, mejores resultados se han obtenido.

Por tanto parece claro que la frecuencia mínima de los términos -lo que determina el tamaño del vocabulario- es un factor tan fundamental como el grado de balanceo. (Ya que también a igual frecuencia mínima, el resultado es mejor con datasets más equilibrados).

- Grado de "parecido" de los vocabularios

Ha sido una sorpresa bastante grande (y negativa) obtener resultados tan malos con los modelos entrenados con los datasets encontrados en internet (con la unión de los 3 datasets). Se han realizado numerosas pruebas, pero todos los resultados han sido muy malos. Esto ha sido bastante inesperado, sinceramente.

En cambio, con datasets tan pequeños como los que contienen únicamente comentarios de El Mundo obtenidos en junio de este año, los resultados son muy buenos. También se puede apreciar que al "mejorar" esos datasets con los comentarios nuevos, los resultados pasan a ser "normales".

- Configuración de los modelos

En todos los casos se puede comprobar que cuando se configuran los parámetros de SVM con valores diferentes de los por defecto, se consiguen resultados mucho mejores. Configurar los pesos de manera adecuada es fundamental, ya que se trata de datasets muy desbalanceados.

Posibles mejoras

Dado el efecto que la adición mensajes nuevos producen en los datasets encontrados en internet, y a la diferencia tan notable entre el dataset de 2000 comentarios y el de 2500, parece claro que aumentar el conjunto de entrenamiento tendrá efectos muy beneficiosos.

También puede ser interesante variar más el tipo de noticias.

5. ANEXO 1 – Librería LIBLINEAR, funciones de pérdida y regularización

Traducción al castellano de la documentación de LIBLINEAR:

“El parámetro “type” en LIBLINEAR puede producir 10 tipos de modelos lineales (generalizados), combinando varios tipos de funciones de pérdida y esquemas de regularización. La regularización puede ser L1 o L2, y las pérdidas pueden ser la pérdida L2 regular para SVM (pérdida de bisagra -“hinge loss”), la pérdida L1 para SVM, o la pérdida logarítmica para la regresión logística. El valor predeterminado para “type” es 0. Consulta los detalles a continuación. Las opciones válidas son:

Para la clasificación multiclase:

- **0** – Regresión logística con regularización L2 (primal)
- **1** – Clasificación de soporte vectorial con pérdida L2 y regularización L2 (dual)
- **2** – Clasificación de soporte vectorial con pérdida L2 y regularización L2 (primal)
- **3** – Clasificación de soporte vectorial con pérdida L1 y regularización L2 (dual)
- **4** – Clasificación de soporte vectorial por Crammer y Singer
- **5** – Clasificación de soporte vectorial con pérdida L2 y regularización L1
- **6** – Regresión logística con regularización L1
- **7** – Regresión logística con regularización L2 (dual)

Para la regresión:

- **11** – Regresión de soporte vectorial con pérdida L2 y regularización L2 (primal)
- **12** – Regresión de soporte vectorial con pérdida L2 y regularización L2 (dual)”

Nota:

- Funciones de pérdidas L1 y L2:

La función de **pérdida L1**, también conocida como **pérdida absoluta**, se define como la suma de las diferencias absolutas entre las predicciones y los valores reales. Matemáticamente, se puede expresar como:

$$L1 = \sum_{i=1}^n |y_i - \hat{y}_i|$$

donde:

- y_i es el valor real.
- \hat{y}_i es la predicción del modelo.
- n es el número de muestras.

La función de pérdida L1 es robusta a los valores atípicos (outliers) porque penaliza los errores grandes y pequeños de manera uniforme. Esto significa que los errores grandes no tienen un impacto desproporcionado en la pérdida total.

La función de **pérdida L2**, también conocida como **pérdida cuadrática o pérdida de error cuadrático medio (Mean Squared Error, MSE)**, se define como la suma de los cuadrados de las diferencias entre las predicciones y los valores reales.

Matemáticamente, se puede expresar como:

$$L2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

donde:

- y_i es el valor real.
- \hat{y}_i es la predicción del modelo.
- n es el número de muestras.

- Regularizaciones L1 y L2:

Fuente: [https://en.wikipedia.org/wiki/Lasso_\(statistics\)](https://en.wikipedia.org/wiki/Lasso_(statistics))

La **regularización L1** es llamada también regularización LASSO (“least absolute shrinkage and selection operator”):

- Añade la suma de los valores absolutos de los coeficientes al costo de la función de pérdida.
- Tiende a producir modelos más escasos, donde algunos coeficientes pueden ser exactamente cero, lo que puede llevar a una selección de características.
- Es útil cuando se desea una solución más interpretable y se quiere reducir el sobreajuste.

La **regularización L2**, también conocida como regularización Ridge:

- Añade la suma de los cuadrados de los coeficientes al costo de la función de pérdida.
- Tiende a distribuir el error entre todos los coeficientes, lo que puede llevar a coeficientes más pequeños pero no exactamente cero.
- Es útil cuando se desea reducir la sobreajuste y se quiere que todos los coeficientes contribuyan al modelo.

En resumen, la regularización L1 y L2 son técnicas utilizadas para evitar el sobreajuste en modelos de aprendizaje automático. La elección entre L1 y L2 depende de las características del problema y de los objetivos del modelo.

6. ANEXO 2 – Código R de las funciones comunes.

hate_speech_common.R

```
# Código común para todos los datasets
# Common code needed to process all datasets

#####
#
# Load needed libraries
#
#####
load_libraries <- function() {

  print("Loading libraries:")

  print("Loading tm ...")
  if (!require(tm)) install.packages('tm', dependencies = T) # text mining
  library(tm)

  print("Loading SnowballC ...")
  if (!require(SnowballC)) install.packages('SnowballC', dependencies = T) # stemming
  library(SnowballC)

  print("Loading textclean ...")
  if (!require(textclean)) install.packages('textclean', dependencies = T)
  library(textclean)

  print("Loading caret ...")
  if (!require(caret)) install.packages('caret', dependencies = T)
  # data partitioning, confusion matrix
  library(caret)

  print("Loading mltools ...")
  if (!require(mltools)) install.packages('mltools', dependencies = T)
  # mcc -> Matthews correlation coefficient
  library(mltools)

  print("Loading tidyverse ...")
  if (!require(tidyverse)) install.packages('tidyverse', dependencies = T)
  library(tidyverse)

  # Liblinear instead of LIBSVM
  #
  # https://www.csie.ntu.edu.tw/~cjlin/liblinear/
  #
  # https://cran.r-project.org/web/packages/Liblinear/
  print("Loading Liblinear ...")
  if (!require(Liblinear)) install.packages('Liblinear', dependencies = T)
  library(Liblinear)

  print("All libraries loaded.")
}

#####
#
# Preprocess posts: remove emoticons, references to users ...
#
#####
preprocess_posts <- function(df, df_raw) {

  #remove references to other users
  df_raw$post <- gsub("@\\w+", "", df_raw$post)
  df_raw$post <- gsub("@ \\w+", "", df_raw$post)
  print("Removed references to users (@).")

  #remove non ascii characters and emoticons using textclean
  df <- df_raw
  df$post <- df_raw$post |>
  replace_non_ascii() |>
  replace_emoticon()
  print("Removed non ascii and emoticons.")
}
```

```

# Remove rows where the post column has empty or null values
result <- with(df, df[!(trimws(post) == "" | is.na(post)), ])
print("Removed lines with empty posts.")

result
}

#####
#
# Clean corpus
#
#####
clean_corpus <- function(corpus) {

  print("#To lowercase")
  posts_corpus_clean <- tm_map(corpus, content_transformer(tolower))

  print("#Remove numbers")
  posts_corpus_clean <- tm_map(posts_corpus_clean, removeNumbers)

  print("#Remove stopwords")
  # check words and languages with ?stopwords
  posts_corpus_clean <- tm_map(posts_corpus_clean, removeWords, stopwords())

  print("#Remove punctuation signs")
  posts_corpus_clean <- tm_map(posts_corpus_clean, removePunctuation)

  print("#Carry out the stemming")
  # To apply the wordStem() function to an entire corpus of text documents, the tm package includes
  # the stemDocument() transformation.
  posts_corpus_clean <- tm_map(posts_corpus_clean, stemDocument)

  print("#Finally eliminate unneeded whitespace produced by previous steps")
  posts_corpus_clean <- tm_map(posts_corpus_clean, stripWhitespace)

  posts_corpus_clean
}

#####
#
# train_test_split: create train and tests sets
#
#####
train_test_split <- function(df, dtm, percentage) {

  #Set seed to make the process reproducible
  set.seed(123)

  #partitioning data frame into training (75%) and testing (25%) sets
  train_indices <- createDataPartition(df$label, times=1, p=percentage, list=FALSE)

  #create training set
  dtm_train <- dtm[train_indices, ]

  #create testing set
  dtm_test <- dtm[-train_indices, ]

  #create labels sets
  train_labels <- df[train_indices, ]$label
  test_labels <- df[-train_indices, ]$label

  #view number of rows in each set
  cat("train dtm nrow: ", nrow(dtm_train), "\n") #
  cat(" test dtm nrow: ", nrow(dtm_test), "\n") #
  cat("length of train labels: ", length(train_labels), "\n") #
  cat(" length of test labels: ", length(test_labels), "\n") #

  return(list(dtm_train = dtm_train, dtm_test = dtm_test, train_labels = train_labels,
test_labels = test_labels))
}

```



```
#####
#
# creat_mat_in_chunks: create a huge matrix from a huge DTM in chunks
# so the R session does not "explode"
#
# (Don't tell anybody: I used duckduckgo.ia (mistral) to help
# develop this code :)
#
#####
creat_mat_in_chunks <- function(dtm, chunk_size) {

  n_docs    <- nrow(dtm)
  n_chunks  <- ceiling(n_docs / chunk_size)

  # helper function to get chunk [i] #
  get_chunk_i <- function(i) {
    start <- (i - 1) * chunk_size + 1
    end <- min(i * chunk_size, n_docs)

    # subset the DocumentTermMatrix
    sub_dtm <- dtm[start:end, ]

    # convert to a dense matrix
    mat <- as.matrix(sub_dtm)
    rm(sub_dtm) #to free space

    cat("chunk ", i, "processed. \n")
    return(mat)
  }

  # generate list of chunk-matrices
  chunk_list <- lapply(seq_len(n_chunks), get_chunk_i)
  # return it
  chunk_list
}

#####
#
# creat_sparse_mat_in_chunks: create a huge matrix from a huge DTM in chunks
# so the R session does not "explode"
#
# Now using sparse matrices, which are incredibly much smaller than dense
# matrices. This allows processing much bigger datasets.
#
#####
creat_sparse_mat_in_chunks <- function(dtm, chunk_size) {

  n_docs    <- nrow(dtm)
  n_chunks  <- ceiling(n_docs / chunk_size)

  # helper function to get chunk [i] #
  get_chunk_i <- function(i) {
    start <- (i - 1) * chunk_size + 1
    end <- min(i * chunk_size, n_docs)

    # subset the DocumentTermMatrix
    sub_dtm <- dtm[start:end, ]

    # convert to a sparse matrix
    mat <- as.matrix(sub_dtm)
    result <- as(as(as(mat, "dMatrix"), "generalMatrix"), "RsparseMatrix")
    rm(sub_dtm) #to free space
    rm(mat)
    cat("chunk ", i, "processed. \n")
    return(result)
  }

  # generate list of chunk-matrices
  chunk_list <- lapply(seq_len(n_chunks), get_chunk_i)
  # return it
  chunk_list
}
```

```
#####
#
# grid_search: grid search function using types (1,2,3,5), cost, bias, #
# and weights #
# #
#####
grid_search <- function(posts_freq_train_mat, training_labels, test_labels,
                        tryTypes, tryCosts, tryBias, tryWeights) {

  if (all(tryTypes %in% c(1,2,3,5))) {
    print("Doing grid search ...")
  } else {
    print("Wrong type parameter. Allowed values to use SVM: 1,2,3,5")
    return()
  }

  bestType <- NA
  bestCost <- NA
  bestBias <- NA
  bestWeights <- NA

  bestAcc <- 0
  bestKappa <- 0
  bestCm <- NA

  #
  for(ty in tryTypes) {
    cat("Results for type = ",ty,"\n",sep="")
    for(co in tryCosts) {
      for(bi in tryBias) {
        for(w in tryWeights) {
          w <- setNames(w, c("0","1"))
          liblinear_svm_model <- LiblinearR(data = posts_freq_train_mat, target = training_labels,
                                             type = ty, cost = co,
                                             bias = bi,
                                             wi = w)

          prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)
          cm <- confusionMatrix(reference = as.factor(test_labels), data =
as.factor(prediction_liblinear$predictions), positive="1", mode = "everything")
          acc <- cm$overall[1]
          kap <- cm$overall[2]
          cat("Results for C = ",co," bias = ",bi," weights = ",w," : ",acc," accuracy, ",kap,"
kappa.\n", sep="")

          if(kap>bestKappa){
            bestType <- ty
            bestCost <- co
            bestBias <- bi
            bestWeights <- w
            bestAcc <- acc
            bestKappa <- kap
            bestCm <- cm
          }
        }
      }
    }
  }

  print("Grid search finished.")
  result <- list(bestType = bestType, bestCost = bestCost, bestBias = bestBias,
bestWeights = bestWeights, cm = bestCm)
  result
}

```

Nota:

Se han añadido chequeos para los parámetros de entrada, y tests unitarios (usando la librería testthat) para validar esos chequeos de tipos.

hate_speech_common_03.R

```
# Código común para procesar comentarios sin etiquetar
# Common code needed to process unlabelled posts

source("hate_speech_common.R")

#####
#
# Preprocess unlabelled posts: remove emoticons, references to users ...
#
#####
process_unlabelled_posts <- function(new_text_df, train_vocab) {

  # Preprocess posts
  new_text_clean_df <- preprocess_posts(new_text_clean_df, new_text_df)

  # additional cleaning for unlabelled posts
  new_text_clean_df$post <- gsub("#\\d+ Cerrar", "", new_text_clean_df$post)
  new_text_clean_df$post <- gsub("# \\d+", "", new_text_clean_df$post)

  # Create corpus
  new_corpus <- Corpus(VectorSource(new_text_clean_df$post))

  # Clean new corpus
  new_corpus_clean <- clean_corpus(new_corpus)
  print("")
  print(new_corpus_clean)

  # Create DTM using the training vocabulary
  new_dtm <- DocumentTermMatrix(new_corpus_clean,
                                control = list(dictionary = train_vocab,
                                                wordLengths = c(2, Inf))
  )

  # Create matrix from the DTM
  new_matrix <- as.matrix(new_dtm)

  #check
  if (!all(colnames(new_matrix) %in% train_vocab)) {
    # should be TRUE
    print("WARNING: not all colnames are included in the training vocabulary!")
  }

  # Convert matrix into a sparse matrix
  new_sparse_matrix <- as(as(as(new_matrix, "dMatrix"), "generalMatrix"), "RsparseMatrix")

  # Return it to make the predictions
  new_sparse_matrix
}
```

7. ANEXO 3 – Repositorio código en GitHub

<https://github.com/fcamadi/masterML/tree/main/TFM>

- Fase 1: recolección de datasets etiquetados

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_01.Rmd

- Fase 2 - Aplicación de SVM

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_02_hatemedia.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_02_huggingface.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_02_kaggle.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_common.R

- Fase 3 - Clasificación de comentarios nuevos no etiquetados

https://github.com/fcamadi/masterML/blob/main/TFM/Web_scraping_RSelenium_periodicos_online.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_03_not_labelled_too_many_positives.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_03_not_labelled.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_03_not_labelled_ALL_100k.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_03_not_labelled_ALL_600k.Rmd

https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_03_common.R

8. ANEXO 4 - Bibliografía

General:

- Machine Learning with R, 4ª ed. - Brett Lantz

SVM:

- LIBSVM:

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>

LIBSVM implementation document is available at <http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>

- LIBLINEAR: <https://www.csie.ntu.edu.tw/~cjlin/liblinear/>

- "A Practical Guide to Support Vector Classification"
<https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>

- MIT OpenCourseWare Lecture Videos.

Lecture 16: Learning: Support Vector Machines

<https://ocw.mit.edu/courses/6-034-artificial-intelligence-fall-2010/resources/lecture-16-learning-support-vector-machines/>