

MASTER FINAL WORK

Hate Speech Detection in Online Media Using the SVM Algorithm

UNED Master's Degree in Machine Learning – Academic Year 2024/2025

Author: Francisco Camacho

Summary

1. Introduction.....	4
1. Motivation.....	4
2. Alternatives and justification of the decision.....	5
2. Phase 1 - Collecting labeled datasets.....	7
1. Hatemedia Project.....	7
2. HuggingFace.....	7
3. Kaggle.....	7
4. Result.....	8
3. Phase 2 – SVM Training.....	9
1. Train SVM with each of the datasets.....	9
1. Kaggle Dataset.....	9
Step 1: Obtaining the dataset and preliminary processing.....	9
Step 2: Processing the dataset.....	10
Step 3: Model training and evaluation.....	13
2. HuggingFace Dataset.....	17
Step 1: Obtaining the dataset and preliminary processing.....	17
Step 2: Processing the dataset.....	19
Step 3: Model training and evaluation.....	20
3. Hatemedia Dataset.....	22
Step 1: Obtaining the dataset and preliminary processing.....	22
Step 2: Processing the dataset.....	23
Step 3: Model training and evaluation.....	25
2. Creating a complete dataset and training an SVM on it.....	29
1. Creating the total dataset.....	30
2. Creating the 100k observations dataset and training.....	31
Dataset processing.....	33
Model training and evaluation.....	35
3. Training with the total dataset.....	41
Dataset processing.....	41
Model training and evaluation.....	43
3. Comparison of results and conclusions.....	46
4. Phase 3 – Classification of new unlabeled comments.....	48
1. Getting new unlabeled comments.....	48
2. Classifying new unlabeled comments with pre-trained models.....	49
1. Read CSVs with reader comments.....	49
2. Loading saved models of phase 2.....	50
3. Classification of unlabeled comments.....	50
4. Results evaluation.....	52
3. Classifying new comments with models trained on them.....	54
4. Improving datasets with the new comments.....	56

1. Processing the datasets.....	56
2. SVM training and result evaluation.....	58
5. Comparative table of results and conclusions.....	61
5. ANNEX 1 – LIBLINEAR library, loss and regularization functions.....	64
6. ANNEX 2 – R code of common functions.....	66
7. ANNEX 3 – Code repository on GitHub.....	71
8. ANNEX 4 - Bibliography.....	72

1. Introduction

The goal of this final work of the UNED Master in Machine Learning (academic year 2024/2025), is to create a prototype that can classify as hate/non-hate the comments posted by readers on the online editions of Spanish newspapers. Obviously, once the model is trained, it can be used to classify comments regardless of whether they come from an online newspaper, a forum, or a social network.

(In fact, the comments of the found datasets have various sources, not just online newspapers.)

1. Motivation

When the Internet and social media emerged, the possibility of instant communication with people anywhere in the world became a reality (1). When these social media platforms became as commonplace in today's societies as the TV or the telephone (especially the mobile phone) became earlier, interactions between people became global, instantaneous... and multiple. The degree of interaction grew exponentially.

With this, the capacity to spread all kinds of messages, both positive and negative, also grew exponentially. Among these negative messages, the author of this work can distinguish two broad categories: hoaxes/fake news and hate speech.

During the exploration and information-seeking phase to decide about which of these two problems develop a final project for this master, I saw that hate speech is a truly global problem: I found papers written at universities in Peru, Nigeria, India, Ethiopia, Spain, Singapore, Australia... whose authors all show great concern about this phenomenon.

(1) The concept of the **global village** was coined by the Canadian sociologist Herbert Marshall McLuhan in the 1960s. McLuhan was referring to the influence that cinema, radio and television had on societies around the world. This concept, I remember, became popular again when the internet and social media started to expand and become very popular 25-30 years ago. The impact of the internet is incredibly greater in my opinion. But McLuhan died in 1980, when the internet was still in its first stage of evolution.

https://en.wikipedia.org/wiki/Global_village

2. Alternatives and justification of the decision

Once the "what" was decided, it was the turn to decide the "how." I initially considered both Naive Bayes and SVM. But after reviewing several papers of previous studies and comparisons, plus the experience gained from completing the previous assignments for this master's degree, I decided to apply SVM. (Due to advantages such as SVM's low tendency to overfitting.)

I initially tried with the LIBSVM library (R package e1071) and caret. But with the available datasets (especially the Hatemedia one), after running some tests, I found that execution times were very very long on the machine on which I was training the models.

In the document "A Practical Guide to Support Vector Classification" and on the website <https://www.csie.ntu.edu.tw/~cjlin/liblinear/>, I quickly found a very good alternative to LIBSVM for this problem: LIBLINEAR (from the same authors of 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."

(Bold as in the original).

I therefore decided to use the LIBLINEAR library instead of LIBSVM.

[

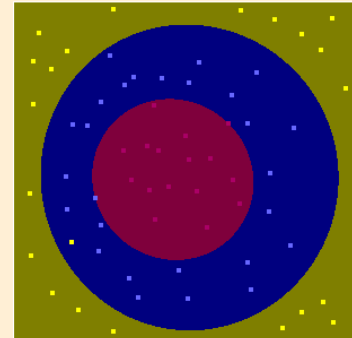
NOTE:

A small sample of the what the LIBSVM library has to offer.

<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:

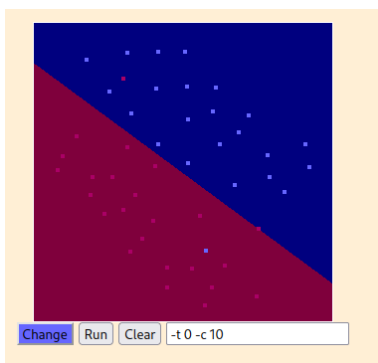
-s svm_type : set type of SVM (default 0)

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

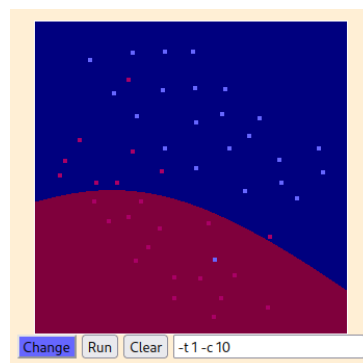
-t kernel_type : set type of kernel function (default 2)

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

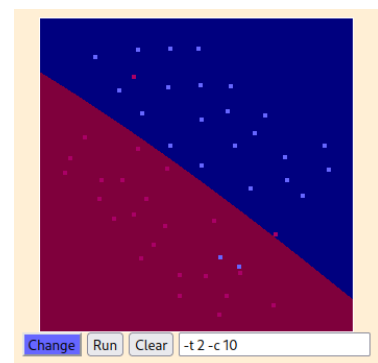
linear kernel:



polynomial kernel:



RBF kernel:



]

2. Phase 1 - Collecting labeled datasets

1. Hatemedia Project

The Hatemedia project was developed by UNIR, the International University of La Rioja:

“Hatemedia: Analysis and monitoring of hate speech in digital media in Spain.

Hatemedia is a space that showcases various projects carried out or participated in by the International University of La Rioja, related to the study of hate speech on social media and digital media in Spain.

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

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

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

At this last address you can access the file and find its description:

"The dataset used to train the hate/non-hate, intensity, and type algorithm models. The dataset consists of messages (news and comments) published by users in El Mundo, ABC, La Vanguardia, El País, and 20 Minutos, on Twitter and their websites, during January 2021."

"Dataset developed within the framework of the Hatemedia Project (PID2020-114584GB-I00), funded by MCIN/AEI /10.13039/501100011033, with the support of the collaborating company [Possible Inc.](#)"

This dataset is the largest found, and is already partially processed (stopwords, numbers, etc. have been removed).

2. HuggingFace

On the website of this AI and machine learning company, I found a super dataset of hate messages in Spanish made joining 5 datasets (2 from Spain - HateEval and Haternet-, another 2 from Chile and one from Mexico).

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

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

3. Kaggle

Kaggle is a platform and community for AI and machine learning experts and practitioners. I found here a multilingual dataset of hate speech.

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

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

This dataset comes separated into a training dataset and a test dataset.

Each of these datasets has its own particularities. The first dataset, from the Hatemedia project, is quite large (574,272 observations) and extremely unbalanced, with all the messages labeled as hate at the end. Furthermore, the dataset found is already partially processed. Comments are formatted as follows:

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

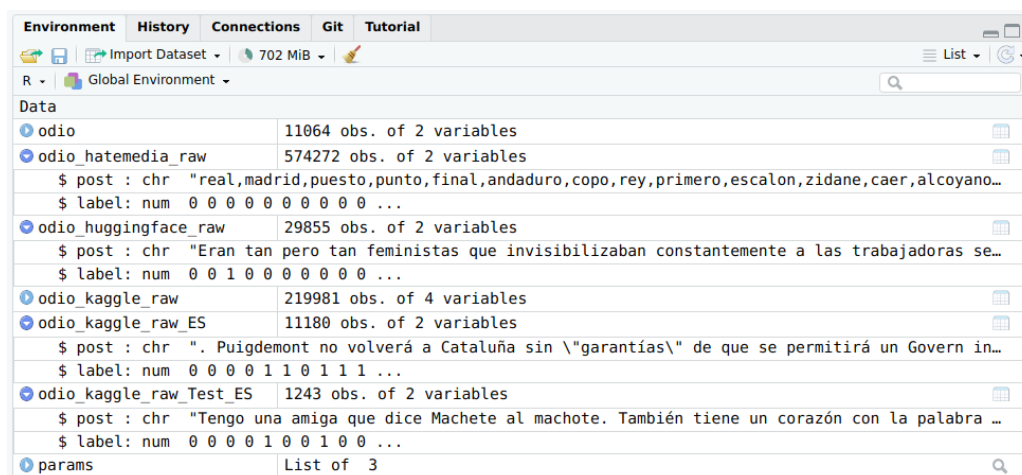
Kaggle datasets, being multilingual, must be preprocessed to extract only the messages in Spanish.

The final goal, it's to obtain 4 datasets with the same format, so that they can be combined into one if desired.

* More columns are available, but only "comment" and "label" are needed.

4. Result

The result of this first processing is 4 dataframes, all with the same structure (two columns: "post" and "label").



The screenshot shows the RStudio Environment pane with the following datasets loaded:

Dataset Name	Observations	Variables
odio	11064	2
odio_hatemedia_raw	574272	2
odio_huggingface_raw	29855	2
odio_kaggle_raw	219981	4
odio_kaggle_raw_ES	11180	2
odio_kaggle_raw_Test_ES	1243	2
params	List of 3	-

The total number of raw observations is:

```
nrow(hate_hatemedia_raw) + nrow(hate_huggingface_raw) + nrow(hate_kaggle_raw_ES)
+ nrow(hate_kaggle_raw_Test_ES)
## [1] 616550
```

I think this is a considerable amount of labeled comments to be able to properly train an SVM model.

3. Phase 2 – SVM Training

1. Train SVM with each of the datasets.

Since 3 different datasets have been found, the aim is to find out whether their characteristics (origin, size...) have any influence on the quality of the SVM that can be obtained when training this algorithm with these datasets.

1. Kaggle Dataset

Loading the common functions needed to process all datasets (code shown in the ANNEX 2).

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

Load packages required for various functions.

```
load_libraries ()  
  
## [1] "Loading libraries:"  
## [1] "Loading tm..."  
## [1] "Loading SnowballC..."  
## [1] "Loading textclean..."  
## [1] "Loading caret ..."  
## [1] "Loading LiblineaR..."  
## [1] "All libraries loaded."
```

Step 1: Obtaining the dataset and preliminary processing

```
#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
```

Dataset structure:

```
str(hate_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 ...
```

The label column is of type int. Since it is actually a 0/1 categorical variable, it is convenient to transform it into a factor:

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

We examine the result:

```
table(odio_kaggle_raw$label)  
##      0      1  
## 7365 3815  
  
prop.table(table(odio_kaggle_raw$label))  
##      0      1  
## 0.6587657 0.3412343
```

This dataset is unbalanced (although not as unbalanced as the other two).

```
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 provocais 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 homófobo,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
```

Step 2: Processing the dataset

As a first step, references to other users in comments will be removed.

Also, with the help of the *cleantext* library we examine the comments and do a first processing:

```
#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."
```

We check the result:

```
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

Create the corpus object with all the messages:

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

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

Cleaning the comments in the 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
```

The corpus is examined and nothing strange is detected.

```
#View(posts_corpus_clean)
```

Tokenization

Finally, the comments are tokenized . A DTM is obtained:

```
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)
```

Now we need to create the training and test sets.

Since this dataset is already separated into two sets, what we do is load the test dataset:

```
# 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]
```

We process it in the same way as the training set:

```
#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
```

Clean the test corpus as well:

```
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
```

Tokenization:

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

Now we need to obtain a list of the most frequently used words:

```
# 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 ]
```

And now we use that list to limit the number of columns/features in both the training and test sets:

```
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
```

Step 3: Model training and evaluation

```
# 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
```

A very important parameter in this call is "type." As we can read in the documentation, type can take the following values:

For multiclass classification

- 0 - logistic regression with L2 (primal) regularization
- 1 - "support vector classification" with L2 regularization and L2 (dual) loss function
- 2 - "support vector classification" with L2 regularization and L2 (primal) loss function
- 3 - "support vector classification" with L2 regularization and L1 (dual) loss function
- 4 - Crammer and Singer "support vector classification" (multiclass).
- 5 - "support vector classification" with L1 regularization and L2 loss function
- 6 - Logistic regression with L1 regularization
- 7 - logistic regression with L2 (dual) regularization

The authors of LIBLINEAR also say the following:

"When choosing between L1 and L2 regularization, we recommend trying L2 first unless the user needs a sparse model."

Therefore, tests have been carried out with options 1, 2, 3 and 5. Since times were quite similar, type = 3 (L2 regularization although we have sparse matrices*, and L1 loss function) has been chosen as it has given the best results.

*The Hatemedia dataset could only be processed using sparse matrices. For the Kaggle and HuggingFace datasets, there was no difference between using one type of matrix or the other.

```

# 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
##

```

I consider this first result simply acceptable. The accuracy is less than 80%, and the kappa is less than 0.5.

The same result is obtained using dense and sparse matrices. With this dataset, being small in size, there's no problem when using dense matrices. (This allows the heuristic function to be used for cost calculations.)

With the other datasets, it was not only essential to use sparse matrices, but also to convert the DTM by “chunks”, since otherwise the RStudio session would “explode” due to the physical memory limitations of the computer.

Model improvement

- We tried to improve the model using the cost calculated by the heuristicC function, as advised by the creators of the LIBLINEAR library:

```

# 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
##

```

Using the cost calculated by the heuristic function, the result was worse in our case. However, choosing a cost above 1 (1.5, 2, 3, 5, etc.) the result improves (only slightly). The best result found is with cost 10.

- We tried to improve the model also by using weights, to favor the detection of the minority class:

```

# 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
##

```

Testing different combinations of the cost parameter and the weights, the best values obtained are around 76-77% for accuracy and a kappa of 0.47.

Comparison table:

(Next page)

	Original	Cost 10	
Confusion matrix	Reference Pred. No Yes No 737 208 Yes 82 216	Reference Pred. No Yes No 734 201 Yes 85 223	
Accuracy	0.7667	0.7699	
Kappa	0.4409	0.4520	
F1	0.5983	0.6093	
MCC	0.4526		
	Cost 1, weights 1/3	Cost 10, weights 1/3	Cost 1, weights 2/3
Confusion matrix	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
Accuracy	0.6693	0.6685	0.7603
Kappa	0.3511	0.3531	0.4691
F1	0.6205	0.6234	0.6519
MCC			0.4677

2. HuggingFace Dataset

Since the process is 99% the same, only those parts of the code that are different or considered most interesting are shown.

Step 1: Obtaining the dataset and preliminary processing

```
# 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
```

We convert the label variable into a factor:

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

We examine the result:

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

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

This dataset is also quite unbalanced, as seen in the first phase (dataset collection) of this project.

We examine some of the comments of this 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 Zaloa, 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
```

The aforementioned peculiarity of this dataset is that it contains comments in different variants of Spanish: there are messages from Chile and Mexico, not just from Spain. This will probably produce a negative impact on the model's performance, as it increases the variety of expressions and idioms the algorithm must understand and be able to generalize in order to classify new texts (even if only comments in Spanish from Spain are classified).

Note:

For example, in messages 2 and 3, the word "flaiterio" is used. The author of this work was completely ignorant of the meaning of this word. According to the Association of Academies of the Spanish Language:

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

flaite.		
I.	1.	noun/adj. <i>Ch. juv.</i> A person of lower social class who tends to display aggressive behavior and dresses somewhat extravagantly. desp.
	2.	adj/noun <i>Ch. Referring to a person</i> with unrefined behavior.
	3.	mf. <i>Ch. Thief. delinc.</i>
	4.	adj. <i>Ch. Referred to something</i> in bad taste.
	5.	<i>Ch. Referring to something</i> of poor or low quality.

(Table translated from Spanish).

It seems a word candidate to appear in hate messages.

Step 2: Processing the dataset

The same processing is performed as for the previous dataset.

To avoid repetitions, only some fragments of the code are shown.

Corpus

Create the corpus object with all the messages:

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

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

This dataset is almost 3 times larger than the Kaggle one, so logically the corpus is almost 3 times larger as well.

Tokenization is also performed to create the DTM, and then the training and test sets are created. To do this, we use one of the functions in the common module:

```
#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
```

Let's check if we keep the (dis)proportion of non-hate messages/hate messages:

```
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
```

We still have 75/25 ratio (non-hate messages/hate messages).

Now we need to obtain a list of the most frequently used words to limit the number of columns/variables in the training and test sets:

```
#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
```

The previous dataset was too small. In this one, I played with the minimum frequency. Before using sparse matrices, this factor was quite important for the following reason: the lower the frequency required to select a word, the more words are logically selected, and the matrix grows proportionally. With matrices of a certain size, the RStudio session would crash. Once this problem was solved, I tried to choose the lowest possible frequency value. I only partially took into account the total number of observations. That's why, in this dataset, instead of choosing a minimum number of occurrences of 10, I chose 20.

Step 3: Model training and evaluation

It is the same code for all datasets, so the results are displayed directly:

First model (cost = 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
##
```

Comparative table of results:

	Cost 1, freq 20	Cost 1, freq 50	Cost 1, freq 100
Confusion matrix	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
Accuracy	0.8290	0.8258	0.7975
Kappa	0.4968	0.4560	0.3259
F1	0.6035	0.5569	0.4287
MCC	0.5035		
	Cost 1, freq 20 weights 1/3	Cost 0.289722 freq 20, weights 2/3	
Confusion matrix	Reference Pred. No Yes No 4474 487 Yes 1173 1329	Reference Pred. No Yes No 5002 694 Yes 645 1122	
Accuracy	0.7776	0.8206	
Kappa	0.4646	0.5083	
F1	0.6156	0.6263	
MCC		0.5088	

The more words the training vocabulary contains, the better the results seem to be. With the previous Kaggle dataset, the accuracy rate was 76-77% and kappa was 0.47. Now, the accuracy rate has increased to over 80%, with a better kappa of around 0.51.

3. Hatemedia Dataset

Step 1: Obtaining the dataset and preliminary processing

```
# 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
```

Given the large size of this dataset (and considering the physical limitations of the laptop), we can choose whether to process the entire dataset or select a subset of the non-hate messages plus all the hate messages. This is a way of doing **downsampling**, but selecting the number of non-hate messages, rather than automatically selecting the same number of messages from both classes.

```
# 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!!!"
```

Tests were performed with different dataset sizes. As can be seen in the comparison at the end of this section, the best result was obtained with the 100k observation dataset.

Dataset structure:

```
str(odio_hatemedia_raw)
## 'data.frame':    574272 obs. of  2 variables:
## $ post : chr
"real,madrid,puesto,punto,final,andaduro,coipo,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 ...
```

The "label" column is of type int. Since it is actually a 0/1 categorical variable, it is convenient to transform it into a factor:

```
#Convert label into a factor
odio_hatemediaw_label <- factor(odio_hatemediaw_label)
```

We already knew this was a very, very unbalanced dataset. The largest and most unbalanced of the three:

```
table(odio_hatemediaw_label)
##
##      0      1
## 563208 11064

prop.table(table(odio_hatemediaw_label))
##
##      0      1
## 0.98073387 0.01926613
```

The percentage of hate messages in this dataset is less than 2%.

In this dataset, the "," characters in the "post" variable must be replaced with spaces. If we don't do this with this dataset, we end up with each comment being a single, enormous string containing all the words in the comment concatenated. (This is because this Hatemediaw dataset was already partially processed, as we mentioned before).

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

Check the result:

```
head(odio_hatemediaw, )
##
## 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 ocasion 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
## ...
```

Step 2: Processing the dataset

In this dataset, messages in Catalan were found:

```
#odio_hatemediaw[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 ..." ...
```

The goal of this final work of the master is to detect hate speech in a single language, not multiple languages, even when they are as similar as Spanish and Catalan are.

Given the difficulty of detecting all the comments in Catalan in a dataset as large as this one, and given the fact that many Catalan citizens in Spanish online media write their comments in their native language, it has been decided not to delete for now these messages.

Corpus

Create the corpus object with all the messages:

```
#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
```

The number of documents is now lower than the initial number, as just under 4,000 records were deleted during this processing.

Once the corpus has been processed and **tokenization** was performed, the training and test sets are created:

```
#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)
```

We check that we keep the (dis)proportion of non-hate messages/hate messages:

```
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
```

We need now to obtain a list with the most frequently used words:

```
# 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
```

Different minimum frequencies were tested. Initially, higher minimum frequencies were chosen to limit the size of the DTM and the matrix passed as an argument to the training function. Once the entire dataset could be processed, frequencies as low as 100 (and even lower) were chosen for this dataset.

And now we use this list to limit the number of columns/features in both the training and test sets:

```
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
```

We can see that the training set has more than 428k observations, and goes from having 348k variables (terms) to only 16516.

Step 3: Model training and evaluation

In order to process the complete dataset, in addition to the mandatory use of sparse matrices, the DTMs had to be converted to said matrices in batches:

```
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)
```

Training (cost = 1) and evaluation:

```
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
```

The speed of this library compared to LIBSVM (e1071) can be easily appreciated. In just 10 seconds, an SVM model with over 400,000 observations was trained on a modest computer with an Intel Core i5 processor and 16 GB of RAM.

With the other “type” options, running times were very similar. (With LIBSVM/e1071, however, more than an hour after the process had started, it had not finished and I had to interrupt it).

```
# 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
##

```

Given the large size of the test dataset and the imbalance between the two classes, it's entirely logical that such (apparently) good accuracy is achieved, yet such low kappa and F1 values are obtained (and a balanced accuracy of less than 55%). Not to mention the recall (not even 10% of hate messages are detected)...

Obviously, this is not about creating a model that just classifies non-hate messages as non-hate messages.

I've run several tests with this dataset. A comparison is shown below. Results obtained with cost = 1, varying the dataset size (i.e., its degree of "balancing"), and the minimum frequency:

- Dataset with 300k observations:

Low freq (100,200,300,400): R session "explodes"

Intermediate freq (500): high accuracy (96%), low kappa (0.28)

Weights 1/2 -> 96% and 0.36

Weights 1/5 -> 95% and 0.38 (kappa improves a bit)

High freq (1000): "absurd" results (0 TP, 0 kappa, and weights cannot fix it)

- Dataset with 100k observations:

Low freq (100): high accuracy (92.5%) and acceptable kappa (0.58)

With weights 1/2, the accuracy remains the same (**92%**), and kappa is **0.61** .

- Dataset 30k observations (not so unbalanced: 19k non-hate vs 11k hate):

Low freq (100) -> High accuracy (83%), good kappa (0.62)
 high freq (500) -> low accuracy, negligible kappa

As we can see, with the dataset of 100k observations, a quite acceptable result is obtained.

It should be noted, however, that these results were obtained before I was able to process the entire dataset. We will try to improve the result with the full dataset (98% accuracy and kappa 0.15), since the final goal is to build the largest possible dataset by combining the three aforementioned datasets.

	Dataset 100k , cost 1, freq100, weights 1/2	Full dataset, cost 1, freq 100	Full dataset, cost 1, freq 100, weights 1/10
Confusion matrix	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
Accuracy	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	
	Full dataset, cost 1, freq 100, weights 1/5	Full dataset, cost 1, freq 100 weights 1/3	Full dataset, cost 1, freq 100 weights 1/2
Confusion matrix	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
Accuracy	0.9694	0.9757	0.9791
Kappa	0.3910	0.3934	0.3469
F1	0.4057	0.4058	0.3571
MCC		0.3935	

Since the dataset was so unbalanced, I decided to use a weight ratio of 1 to 10. This significantly improved kappa compared to the original model (from 0.15 to 0.35).

With weights 1/5 and 1/3, very similar accuracy, kappa, and F1 values are obtained. The choice is between detecting more PTs, at the expense of obtaining more FPs, or fewer PTs and also fewer FPs. In any case, the kappa values are not specially good, being just around 0.4.

Compared to previous datasets, accuracy has improved again, but kappa has dropped from 0.5 to 0.4.

(Unless we consider the dataset of 100k observations, which returns the best result of all).

Since kappa did not improve, and considering the possibility that choosing such a low frequency would produce some overfitting to the texts (being aware that SVM is not prone to this problem - one of the reasons why this technique was chosen), a higher minimum frequency was tested: 500 (instead of 100)

With freq = 500 the size of the DTMs and matrices decreased considerably:

```
[1] 428151 348382
[1] 428151    5886 ← 5886 columns/features instead of 16516 with freq 100
[1] 142717 348382
[1] 142717    5886 ← idem.
```

(Interestingly, training times became longer.)

But the results obtained are no better. Quite the opposite:

	Full dataset, cost 1, freq 100	Full dataset, cost 1, freq 500																								
Confusion matrix	<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																								
Accuracy	0.9811	0.9808																								
Kappa	0.1563	0.097																								
F1	0.1608	0.1003																								

The result is considerably worse than with freq = 100.

(With no weights it is possible to improve the results obtained with freq = 100 either).

2. Creating a complete dataset and training an SVM on it

The idea of bringing all the datasets together is to ensure that the SVM, by having more diverse (and larger, but above all diverse) sources of information, can be trained better and thus obtain the best possible results with the new unlabeled messages presented to it.

It will also be done in this phase of the project, given the good result obtained with the Hatemedia dataset of 100k observations (accuracy 92% and kappa 0.61), in addition to training the SVM model with the total dataset composed of the 3 datasets, to create from this “total” dataset another one of 100k comments in the same way: it will contain all the hate messages from the total dataset, and a certain number of non-hate messages until we reach those 100k records.

SVM will then be trained with both datasets, and the results will be compared (also with the previous ones).

1. Creating the total dataset

Global variables:

```
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
```

Import all CSV files:

```
# 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
```

We already know that the Hatemedia dataset is somewhat different. We'll remove the "," again to avoid problems.

```
# 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)
```

Now we can create a “total” dataset with all the available datasets:

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

We now have a dataset with a total of 616,550 comments from readers of online newspapers and other sources.

[
Remove datasets and variables that are no longer needed:

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

Check proportion of messages in this dataset:

```
table(hate_raw$label)  
##  
##      0      1  
## 593982 22568  
  
prop.table(table(hate_raw$label))  
##  
##      0      1  
## 0.96339632 0.03660368
```

Obviously, since the Hatemedia dataset is much larger than the other two and very unbalanced, the resulting dataset is also very unbalanced.

2. Creating the 100k observations dataset and training

Since we have 22,568 hate messages, we will randomly select the following number of non-hate messages (rounded to 78,000 as indicated in the “size” variable at the beginning):

```
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 whoooole dataset. Be patient!!!")  
}  
## [1] "Preparing a smaller dataset"
```

Check the proportion of messages in this dataset after downsampling:

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

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

The dataset is still unbalanced, but to a much lesser extent.

(With the Hatemedia dataset, no better result was obtained by further reducing the degree of imbalance by lowering the total number of non-hate observations to 30k).

Dataset structure:

```
str(hate_raw)
## 'data.frame':    100568 obs. of  2 variables:
## $ post : chr  "vender argentino " "odar 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 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 pifia 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 aparezco 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 ágil 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
```

Dataset processing

The "label" column is of type int. Since it is actually a 0/1 categorical variable, it is convenient to transform it into a factor:

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

We process comments in the usual way (remove references to other readers, remove emoticons ..):

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

During this process, some comments may be left blank. We delete them in that case:

```
# 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

We can now proceed to create the corpus object with all the messages:

```
#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
```

Clean the corpus:

Usual processing: remove capital letters, numbers...

```
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
```

Finally the comments are **tokenized**:

```
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)
```

[
At this point, delete the "corpus" objects. They are no longer needed.

```
rm(posts_corpus)
rm(posts_corpus_clean)
]
```

Now we need to create the training and test sets:

```
#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)
```

Let's check if the (dis)proportion of non-hate messages/hate messages is maintained:

```
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
```

Now we need to obtain the list of the most used words:

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

And now we use that list to limit the number of columns/features in both the training and test sets:

```
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
```

We had previously 137222 columns/features and now only 4,716.

Model training and evaluation

We need to convert the DTMs into matrices in order to train the model. Given the size of the dataset and the physical limitations of the equipment on which this process is performed, the DTMs are processed in batches, and the matrices are then combined to obtain the complete matrix.

```
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)
```

Training:

```
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
```

Prediction:

```
# 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
```

Result evaluation:

```
# 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  
##
```

When we have an unbalanced dataset, but in a ratio like this (3 to 1 in favor of non-hate messages), we get a more than acceptable result. An accuracy approaching **90%** and a kappa approaching **0.7** seem more than good to me (the rest of the indicators also seem quite good).

Considering also that this dataset of 100k observations seems quite realistic to me (the HuggingFace and Kaggle datasets considered separately are much smaller).

We can also compare this result of joining the 3 datasets and obtaining a dataset of 100k observations, with that of 100k observations from Hatemedia: kappa improves from 0.61 to 0.7.

Note:

Since we're using sparse matrices to handle a dataset of this size, we can't use LIBLINEAR's heuristic function to calculate a cost. However, after extensive testing, we've seen that weights help improve the result much more than using a cost other than 1.

- Improving the model using weights

It has been tested with 1/5, 1/3, and 1/2 ratios. Only with these last weights does the initial result improve:

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

Prediction:

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

Result evaluation:

```
#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
##
```

I consider this result to be quite good: accuracy of **90%** , kappa slightly higher than **0.7** , F1 close to **0.8** ...

Despite these results, I've decided to try two more techniques to see if they can be improved, even only slightly. We're going to try using the cross-validation offered by this library, and also using a grid search trying different parameters.

- Cross-validation

Let's use the cross-validation offered by LIBLINEAR to compare its results with what I've already obtained. As indicated in the documentation (and as seen in previous assignments in this master's degree), the accuracy metric isn't the most appropriate when you have a highly unbalanced dataset. But let's try it anyway:

```
#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

```

Note:

Out of curiosity, I tried comparing logistic regression (type = 0) with SVM. After an hour, it still was running. On the contrary, the four kinds of SVM could finish in less than 10 minutes.

```

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
```

After the tests that have been carried out, the result is not surprising: the best result is obtained with type 3 and cost 1.

- Grid search

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
```

By doing this grid search, a slightly different model is obtained, but the result is almost identical: accuracy of 90%, and kappa 0.71.

Since I consider this model more “complicated” (cost 10 and bias 10, instead of the default values), and its result is practically the same, I stick with the one previously obtained (type 3, weights 1/2).

We now move on to training SVM with the full dataset (more than 600k observations).

3. Training with the total dataset

Before processing the entire dataset, it has been preferred to work first with the dataset of 100k observations, and thus apply what has been learned to this dataset (for example, seeing the result of the cross-validation and the grid search, I think I can trust the results obtained with type 3 and cost 1).

Since the code is the same, only the parts that display information related to this dataset are shown:

Dataset processing

```
#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 (@)."
```

```
## [1] "Removed non ascii and emoticons."
## [1] "Removed lines with empty posts."
##      user      system elapsed
## 420.294    0.328 420.632

# 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

We can now proceed to create the corpus object with all the messages:

```
#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
```

After processing the corpus and performing tokenization, the training and test sets are created:

```
#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)
```

We check if we keep the (dis)proportion of non-hate messages/hate messages:

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

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

The list of most frequent words is generated, and we use it to limit the number of columns/features in both the training and test sets:

```
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
```

The number of columns/features has been decreased from 361,649 to 16,845, both in the training set and in the test set.

Model training and evaluation

We need to convert the DTMs into matrices in order to train the model. Given the size of the dataset and the physical limitations of the equipment on which this process is performed, the DTMs are processed in batches, and the matrices are then combined to obtain the final matrix.

```
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)
```

Training

```
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
```

Prediction:

```
# 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
```

Result evaluation:

```
# 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
##
```

The result is just acceptable. Kappa could certainly be better. An attempt will be made to find weights that can improve the detection of hate speech (and therefore kappa).

Note:

Since we're using sparse matrices to handle a dataset of this size, we can't use LIBLINEAR's heuristic function to calculate an optimal cost. However, after several tests, we've seen that the weights help improve the result much more than using a cost other than 1.

- Improving the model using weights

Cross-validation and grid search were tested. The best result was obtained using the latter:

```
[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
```

So we assign the weights 1/3 (type 3 and cost 1 as usual):

Training:

```
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
```

Prediction:

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

Result evaluation:

```
#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
##
```

Kappa has been improved from 0.39 to 0.51, while accuracy has only slightly worsened.

In any case, it can be easily seen that with both the Hatemedia dataset and the total dataset, better results are obtained when the model is trained with the subset of 100k observations instead of with the total dataset, which is extremely unbalanced.

3. Comparison of results and conclusions.

Comparison of the best results obtained with each dataset:

(cost = 1 for all, except dataset 28k: cost = 0.1

freq = 100 for all except Kaggle, freq = 50, HuggingFace and superdataset 50k, with freq = 20)

Dataset	Kaggle	HuggingFace	Hatemia weights 1/3	Hatemia 100k weights 1/2
Confusion matrix	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
Accuracy	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 weights 1/3	Superdataset 100k weights 1/2	Superdataset 50k weights 1/2
Confusion matrix		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
Accuracy		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

Comparing all the results so far, I see the following:

i) The larger the dataset, the higher the accuracy.

This is logical: since the datasets are very unbalanced, the model learns very well to detect the majority class.

ii) Since these are highly unbalanced datasets, accuracy is not the most important metric, but rather kappa. And kappa is better when the dataset is reduced to 100,000 observations, and even better with a dataset of 50,000 observations.

iii) With the same size (this is, with the same level of imbalance), the resulting dataset from mixing the three original datasets also yields better results. It seems that the original hypothesis that having different datasets would allow SVM to generalize better and obtain better results with new messages is confirmed.

iv) Putting all of the above together, the best result is logically the one obtained with the SVM trained on the dataset of 50,000 observations from the total dataset. This dataset contains 28,000 non-hate messages and 22,000 hate messages.

v) I have added at the end the Matthews correlation coefficient, but it does not provide more information than what kappa already gives (they are practically the same values).

I would like to emphasize again that the datasets contain not only comments in Spanish, but also in Catalan, and in the Spanish variants of Mexico and Chile, and that the sizes (50k, 100k and 600k observations) seem quite realistic to me (enough to validate in principle the drawn conclusions).

4. Phase 3 – Classification of new unlabeled comments

Until now, only datasets found on the internet have been used. This 3rd. phase involves classifying new unlabeled comments using the models trained in the previous phase.

Specifically, the following is intended:

- Obtain new comments
- Classify them with the previously trained models
- Evaluate the result

Depending on the result:

- Train and evaluate SVM with only new comments.
- Add the new comments to the total dataset, and train SVM with this “improved” dataset
- Compare the results

1. Getting new unlabeled comments

The online edition of El Mundo was chosen as the source of new comments. The main reason was its ease of access (and its wide distribution, which makes it easier to collect many comments per piece of news). To access reader comments in other newspapers, registration is required in most cases. However, in this medium, they are available without any barriers (you don't need to be a subscriber or even log in).

A total of 3,391 comments were obtained from 16 pieces of news:

Files

Plots

Packages

Help

Viewer

Presentation

New Folder

New File

Delete

Rename

More

Home

git

masterML

TFM_notLabelled

posts

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

National political news clearly predominate, as this is the kind of news with the greatest reader engagement.

Examples of some of those comments:

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_imparcial #55 Cerrar Aldama mente y sobreactúa. borrego detectado

...

The R code for web scraping is available here:

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

2. Classifying new unlabeled comments with pre-trained models

In this step, we will attempt to classify new comments using previously trained models.

1. Read CSVs with reader comments

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

We read the files with the new comments:

```
# 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
```

Rename variables:

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

Remove author column and add label column:

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

2. Loading saved models of phase 2

We proceed to load the two best models from the previous phase with which the best results were obtained (table page 46).

Note:

Given the results obtained with them, an attempt has also been made with a model trained with a very balanced dataset.

- "100k" Model:

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

- "600k" Model:

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

- "Best" model (dataset 50k observations, type 3, freq 20, cost 0.1, bias 10, weights 1/2):

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

3. Classification of unlabeled comments

- Choose comments randomly

We randomly selected 2000 comments from the total of 3000. With those 2000 comments we created 2 dataframes subset1 and 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, ]
```

- Classify messages with the two (three) models.

We chose 1000 comments (subset1) to perform this first test of classification of new unlabeled comments:

```
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
```

We make predictions with the 3 models:

```
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
```

We examine the result:

```
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
```

There are clearly too many positive/hateful comments. Interestingly, with the model trained on the most balanced dataset (the 50,000 observations dataset contains 28,000 non-hateful comments and 22,000 hateful comments), we get the worst result.

4. Results evaluation

To be able to obtain the different metrics, the new comments from El Mundo in June 2025 were manually classified by the author of this work.

That rating has been added to the label column, and the result of calling caret's confusionMatrix function for the 3 models is:

Confusion matrix for "model 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
##
```

Confusion matrix for “model 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
##
```

Confusion matrix for “model 50k better”:

```
##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
##
```

I think it's not just about the level of dataset balancing. I think vocabulary has as much influences, if no more, than the degree of balancing. The larger the vocabulary we train a model with, the better the results it returns.

3. Classifying new comments with models trained on them

Now we will train SVM only with the comments obtained in the first section of this chapter.

Initially, training was carried out with a dataset of 2,000 comments, to which another 500 were later added. It was found that the result improved significantly when these additional 500 comments were added.

With the training dataset with 2000 comments the best result obtained was:

```
[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 with default values)
```

Training SVM on a dataset with those 500 additional comments, the result is much better:

```
#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
##
```

Performing grid search:

```
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."
```

The best result we obtained is:

```
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
##
```

In this case the best result is obtained with SVM type 1.

It seems clear that by increasing the training set, the results could be improved even further.

4. Improving datasets with the new comments

The goal is to test whether "enhancing" the entire dataset made with the Hatemedia, HuggingFace and Kaggle datasets by adding the comments obtained in this phase yields a better result than in the previous section. (One might think that the result should be intermediate between the results obtained with only the datasets from the internet and with the dataset from the previous section, which only contains the new comments.)

1. Processing the datasets

In order not to repeat code, only some fragments are shown:

Initial variables:

```
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"
```

Check the proportion of messages in this dataset after performing the "downsampling" step:


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

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

Now we add the 2000 messages from El Mundo obtained in June 2025:

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

We added 500 more classified comments:

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

Note:

The process of classifying reader comments is quite laborious (and subjective, of course). I decided to do it in two steps in order to see how increasing the training set affects the final result.

Now we can create a complete dataset with all available datasets:

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

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

In the end we have a dataset with 53k comments, quite balanced (55.37% non-hate messages, and 44.63% hate messages).

We also read the comments of tests that are also already labeled:

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

Check proportion of messages in these datasets:

- training dataset:

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

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

Adding these 2,500 new comments only slightly changes the proportion already existing in the training dataset.

- test dataset:

```
table(posts_test_raw$label)
##
##      0      1
## 453   47
```

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

The dataset that we are going to classify with SVM is very unbalanced (according to the particular assessment of the author of this work).

The datasets are then prepared using the usual procedure: cleaning (removing references to other users, emoticons, etc.), creating the corpus, processing the corpus (removing capital letters, numbers, etc.), tokenizing, searching for the most frequent words, etc.

At the end of the process we get:

```
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
```

Now we can start training the SVM.

2. SVM training and result evaluation

We need to convert the DTMs into matrices in order to train the model. Given the size of the dataset and the physical limitations of the computer on which this process is carried out, the DTMs are processed in batches, and the matrices are then combined to obtain the complete matrix.

```
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)
```

Training (cost = 1)

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

#liblinear_svm_model
```

Prediction:

```
# 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
```

Result evaluation:

```
# 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
##
```

The result is slightly worse than what was obtained by training SVM only with the new comments (kappa 0.49).

We use grid search to try to find a better model than with the default parameters:

```
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
```

Best result obtained:

```
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
##

```

The result is again slightly worse than using only the comments from El Mundo. But in both cases, the results are much better than those obtained by training SVM only with the datasets found on the internet.

5. Comparative table of results and conclusions

On the next page you can find all the results of classifying the 500 new comments with all the models (trained with the internet datasets -first row-, with the new comments only -second row-, and with the “improved” datasets -third row-).

Models trained with internet datasets	Superdataset "600k" type 3 default values	Superdataset "100k" type 3 default values	Superdataset 50k "best"	
Confusion matrix	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	
Accuracy	0.643	0.312	0.219	
Kappa	0.2028	0.0494	0.0258	
F1	0.3544	0.2505	0.2351	
Models trained only with new comments	2000 comments type 3 default values	2000 comments "best"	2500 comments type 3 default values	2500 comments "best"
Confusion matrix	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
Accuracy	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
Models trained with "enhanced datasets"	Superdataset "600k" type 3 default values	Superdataset "600k best"	Superdataset "50k" type 3 default values	Superdataset "50k best"
Confusion matrix	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
Accuracy	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

"best": model found using grid search

"Enhanced dataset": dataset built with the 3 datasets found on the internet plus 2,500 labeled comments from El Mundo in June 2025

Conclusions

- Degree of balance vs minimum frequency

At first glance, it might seem that the degree of balancing is the most important factor influencing the quality of the results. But two results have led me to a different conclusion:

- When classifying new comments with models trained on the superdataset made with the three datasets from Hatemedia, HuggingFace and Kaggle, the result is better with the largest dataset, which is the most unbalanced (but is the one that has been built using a broader vocabulary).
- When testing with different minimum frequencies (thus limiting the columns/features of the DTMs) with the same dataset, regardless of its degree of balancing, the lower the minimum frequency, the better the results obtained.

Therefore, it seems clear that the minimum frequency of terms—which determines vocabulary size—is as a fundament factor as the degree of balance. (Since also with the same minimum frequency, the results are better with more balanced datasets.)

- Degree of "similarity" of the vocabularies

It was a rather big (and negative) surprise to get such bad results with the models trained with the datasets found on the internet (with the union of the three datasets). Numerous tests were performed, but all the results were very poor. This was quite unexpected, to be honest.

However, with datasets as small as those containing only comments from El Mundo obtained in June of this year, the results are very good. It can also be seen that by "enhancing" these datasets with the new comments, the results become "normal".

- Model configuration

In all cases, it can be seen that configuring the SVM parameters with values other than the defaults achieves much better results. Setting the weights appropriately is essential, as these are highly unbalanced datasets.

Possible improvements

Given the effect that adding new messages has on the datasets found on the Internet, and the significant difference between the dataset with new 2000 comments and the one with 2500, it seems clear that increasing the training set will have very positive effects.

It may also be interesting to vary more the type of news.

5. ANNEX 1 – LIBLINEAR library, loss and regularization functions

The “type” parameter in LIBLINEAR can produce 10 types of (generalized) linear models, combining various types of loss functions and regularization schemes. The regularization can be L1 or L2, and the losses can be the regular L2 loss for SVM (hinge loss), the L1 loss for SVM, or the log loss for logistic regression. The default value for “type” is 0. See below for details. Valid options are:

For multiclass classification:

- **0** – Logistic regression with L2 (primal) regularization
- **1** – Support vector classification with L2 loss and L2 (dual) regularization
- **2** – Support vector classification with L2 loss and L2 (primal) regularization
- **3** – Support vector classification with L1 loss and L2 (dual) regularization
- **4** – Support vector classification by Crammer and Singer
- **5** – Support vector classification with L2 loss and L1 regularization
- **6** – Logistic regression with L1 regularization
- **7** – Logistic regression with L2 (dual) regularization

For regression:

- **11** – Support vector regression with L2 loss and L2 (primal) regularization
- **12** – Support Vector Regression with L2 Loss and L2 (Dual) Regularization”

Note:

- L1 and L2 loss functions:

L1 loss function , also known as **the absolute loss** , is defined as the sum of the absolute differences between the predictions and the actual values. Mathematically, it can be expressed as:

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

where:

- y_i is the actual value.
- \hat{y}_i is the model prediction.
- n is the number of samples.

The L1 loss function is robust to outliers because it penalizes large and small errors equally. This means that large errors do not have a disproportionate impact on the overall loss.

L2 loss function , also known as **quadratic loss or mean squared error (MSE) loss** , is defined as the sum of the squares of the differences between the predictions and the actual values.

Mathematically, it can be expressed as:

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

where:

- y_i is the actual value.
- \hat{y}_i is the model prediction.
- n is the number of samples.

- L1 and L2 regularizations:

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

L1 regularization is also called LASSO regularization (“least absolute shrinkage and selection operator”):

- Adds the sum of the absolute values of the coefficients to the cost of the loss function.
- It tends to produce sparser models, where some coefficients may be exactly zero, which can lead to feature selection.
- It is useful when a more interpretable solution is desired and overfitting is to be reduced.

L2 regularization, also known as Ridge regularization:

- Add the sum of the squares of the coefficients to the cost of the loss function.
- It tends to distribute the error among all the coefficients, which can lead to smaller coefficients but not exactly zero.
- It is useful when you want to reduce overfitting and want all coefficients to contribute to the model.

In short, L1 and L2 regularization are techniques used to prevent overfitting in machine learning models. The choice between L1 and L2 depends on the characteristics of the problem and the objectives of the model.

6. ANNEX 2 – R code of common functions.

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 <- LiblineaR(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  
}
```

Note:

Checks have been added later for input parameters, and unit tests (using the *testthat* library) to validate those type checks.

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. ANNEX 3 – Code repository on GitHub

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

- Phase 1: Collecting labeled datasets

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

- Phase 2 - Application of 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

- Phase 3 - Classification of new unlabeled comments

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. ANNEX 4 - Bibliography

General:

- "Machine Learning with R", 4th 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/>