

## MASTERABSCHLUSSARBEIT

# Erkennung von Hassrede in Online-Medien mithilfe des SVM Algorithmus

Abschlussarbeit des UNED-Masters in Maschinellem Lernen – Studienjahr 2024/2025

Autor: Francisco Camacho

## Index

1. Einleitung.....	4
1. Motivation.....	4
2. Alternativen und Begründung der Entscheidung.....	5
2. Phase 1 – Sammlung beschrifteter Datensätze.....	7
1. Hatemedia-Projekt.....	7
2. HuggingFace.....	7
3. Kaggle.....	7
4. Ergebnis.....	8
3. Phase 2 – SVM-Schulung.....	9
1. Trainieren Sie SVM mit jedem der Datensätze.....	9
1. Kaggle-Datensatz.....	9
Schritt 1: Beschaffung des Datensatzes und Vorverarbeitung.....	9
Schritt 2: Verarbeiten des Datensatzes.....	10
Schritt 3: Modelltraining und -evaluierung.....	13
2. HuggingFace-Datensatz.....	17
Schritt 1: Beschaffung des Datensatzes und Vorverarbeitung.....	17
Schritt 2: Verarbeitung des Datensatzes.....	19
Schritt 3: Modelltraining und -evaluierung.....	20
3. Hatemedia-Datensatz.....	22
Schritt 1: Beschaffung des Datensatzes und Vorverarbeitung.....	22
Schritt 2: Verarbeitung des Datensatzes.....	23
Schritt 3: Modelltraining und -evaluierung.....	25
2. Erstellung eines vollständigen Datensatzes und Trainierung eines SVM darauf.....	30
1. Erstellung des Gesamtdatensatzes.....	30
2. Erstellen des 100k-Beobachtungs-Datensatzes und Training.....	31
Datensatzverarbeitung.....	33
Modelltraining und -evaluierung.....	35
3. Training mit dem Gesamtdatensatz.....	41
Datensatzverarbeitung.....	42
Modelltraining und -evaluierung.....	43
3. Vergleich der Ergebnisse und Schlussfolgerungen.....	47
4. Phase 3 – Klassifizierung neuer unbeschrifteter Kommentare.....	49
1. Neue unbeschriftete Kommentare erhalten.....	49
2. Klassifizierung neuer unbeschrifteter Kommentare mit vortrainierten Modellen.....	50
1. CSV-Dateien mit Leserkomentaren auslesen.....	50
2. Laden gespeicherter Modelle der Phase 2.....	51
3. Klassifizierung unbeschrifteter Kommentare.....	51
4. Ergebnisauswertung.....	53
3. Klassifizierung neuer Kommentare mit darauf trainierten Modellen.....	55
4. Verbesserung des Datensatzes mit den neuen Kommentaren.....	57
1. Verarbeitung der Datensätze.....	57

2. SVM-Training und Ergebnisauswertung.....	59
5. Vergleichstabelle der Ergebnisse und Schlussfolgerungen.....	62
5. ANHANG 1 – LIBLINEAR-Bibliothek, Verlust- und Regularisierungsfunktionen.....	65
6. ANHANG 2 – R-Quellcode gemeinsamer Funktionen.....	67
7. ANHANG 3 – Code-Repository auf GitHub.....	72
8. ANHANG 4 - Bibliographie.....	73

## 1. Einleitung

Das Ziel dieser Abschlussarbeit des UNED-Masterstudiengangs in Maschinellern Lernen (Studienjahr 2024/2025) ist die Erstellung eines Prototyps, der die von Lesern in den Online-Ausgaben spanischer Zeitungen geposteten Kommentare als Hass- oder Nicht-Hass klassifizieren kann.

Sobald das Modell trainiert ist, kann es selbstverständlich zum Klassifizieren von Kommentaren verwendet werden, unabhängig davon, ob diese aus einer Online-Zeitung, einem Forum oder einem sozialen Netzwerk stammen.

(Tatsächlich stammen die Kommentare zu den verwendeten Datensätzen aus verschiedenen Quellen, nicht nur aus Online-Zeitungen.)

## 1. Motivation

Mit dem Aufkommen des Internets und der sozialen Medien wurde die Möglichkeit der sofortigen Kommunikation mit Menschen überall auf der Welt zur Realität (1). Als diese sozialen Medienplattformen in der heutigen Gesellschaft so alltäglich wurden wie früher das Fernsehen oder das Telefon (insbesondere das Mobiltelefon), wurde die Interaktion zwischen Menschen global, unmittelbar und vielfältig. Der Grad der Interaktion wuchs exponentiell.

Damit nahm auch die Fähigkeit zur Verbreitung aller Arten von Botschaften – sowohl positiver als auch negativer – exponentiell zu. Der Autor dieses Werks kann bei diesen negativen Botschaften zwei große Kategorien unterscheiden: Falschmeldungen/Fake News und Hassreden.

Während der Erkundungs- und Informationsbeschaffungsphase, in der ich überlegte, welches dieser beiden Probleme ich als Abschlussprojekt für diesen Master entwickeln sollte, wurde mir klar, dass Hassreden ein wahrhaft globales Problem sind: Ich fand Arbeiten von Universitäten in Peru, Nigeria, Indien, Äthiopien, Spanien, Singapur, Australien ..., deren Autoren alle große Besorgnis über dieses Phänomen äußern.

(1) Der Begriff des **globalen Dorfes** wurde in den 1960er Jahren vom kanadischen Soziologen Herbert Marshall McLuhan geprägt. McLuhan bezog sich damit auf den Einfluss von Kino, Radio und Fernsehen auf die Gesellschaft weltweit. Ich erinnere mich, dass dieser Begriff wieder populär wurde, als das Internet und die sozialen Medien vor 25 bis 30 Jahren begannen, sich auszubreiten und sehr populär zu werden. Meiner Meinung nach ist der Einfluss des Internets ungleich größer. McLuhan starb jedoch 1980, als sich das Internet noch in der Anfangsphase seiner Entwicklung befand.

[https://de.wikipedia.org/wiki/Globales\\_Dorf](https://de.wikipedia.org/wiki/Globales_Dorf)

## 2. Alternativen und Begründung der Entscheidung

Nachdem das „Was“ geklärt war, ging es an die Frage nach dem „Wie“. Zunächst hatte ich sowohl Naive Bayes als auch SVM in Betracht gezogen. Nach der Durchsicht mehrerer Arbeiten früherer Studien und Vergleiche sowie der Erfahrungen aus früheren Masterarbeiten entschied ich mich jedoch für SVM, aufgrund von Vorteilen wie der geringen Tendenz zur Überanpassung.

Ich habe es zunächst mit der LIBSVM-Bibliothek (R-Paket e1071) und Caret versucht. Aber mit den verfügbaren Datensätzen (insbesondere dem von Hatemedia) stellte ich nach einigen Tests fest, dass die Ausführungszeiten auf dem Computer, auf dem ich die Modelle trainierte, sehr, sehr lang waren.

Im Dokument „A Practical Guide to Support Vector Classification“ und auf der Website <https://www.csie.ntu.edu.tw/~cjlin/liblinear/> habe ich für dieses Problem schnell eine sehr gute Alternative zu LIBSVM gefunden: LIBLINEAR (von denselben Autoren wie LIBSVM).

"Wann sollte LIBLINEAR, aber nicht LIBSVM verwendet werden?"

Es gibt einige große Datenmengen, bei denen mit/ohne nichtlineare Abbildungen ähnliche Leistungen erzielt werden. **Ohne den Einsatz von Kernen** kann man mithilfe eines linearen Klassifikators schnell einen viel größeren Datensatz trainieren. **Die Dokumentenklassifizierung** ist eine solche Anwendung.

(Fett wie im Original).

Ich habe mich daher entschieden, die Bibliothek LIBLINEAR anstelle von LIBSVM zu verwenden.

[

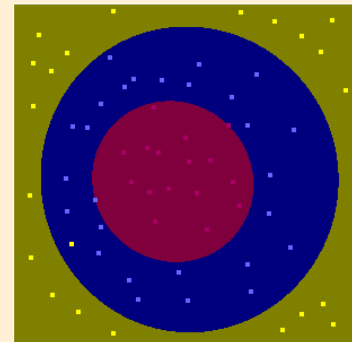
NOTIZ:

Eine kleine Auswahl dessen, was die LIBSVM-Bibliothek zu bieten hat.

<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.



Optionen:

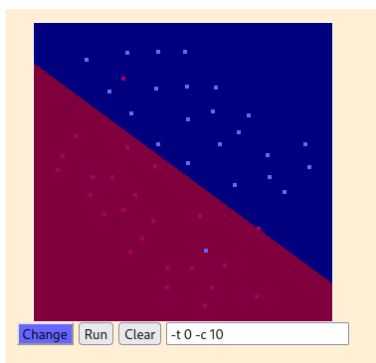
-s svm\_type: SVM-Typ festlegen (Standard 0)

- 0 - C-SVC
- 1 -- nu-SVC
- 2 - Ein-Klassen-SVM
- 3 - Epsilon-SVR
- 4 - nu-SVR

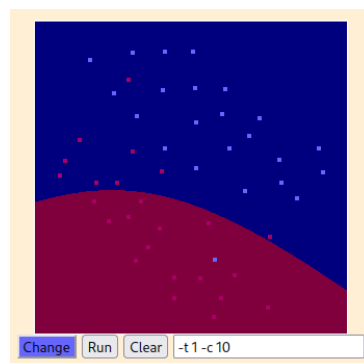
-t kernel\_type: Typ der Kernelfunktion festlegen (Standard 2)

- 0 -- linear:  $u'v$
- 1 - Polynom:  $(\gamma u'v + \text{coef0})^{\text{Grad}}$
- 2 - radiale Basisfunktion:  $\exp(-\gamma |uv|^2)$
- 3 - Sigmoid:  $\tanh(\gamma u'v + \text{coef0})$

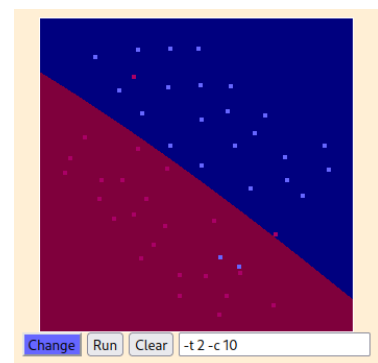
linearer Kernel:



polynomischer Kernel:



RBF-Kernel:



]

## 2. Phase 1 – Sammlung beschrifteter Datensätze

### 1. Hatemedia-Projekt

Das Hatemedia-Projekt wurde von UNIR, die Internationale Universität von La Rioja, entwickelt:

„Hatemedia: Analyse und Überwachung von Hassreden in digitalen Medien in Spanien.

Hatemedia ist ein Raum, in dem verschiedene Projekte präsentiert werden, die von der Internationalen Universität von La Rioja durchgeführt wurden oder an denen sie teilgenommen hat und die sich mit der Untersuchung von Hassreden in sozialen Medien und digitalen Medien in Spanien befassen.“

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

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

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

Unter dieser letzten Adresse kann man auf die Datei zugreifen und deren Beschreibung finden:

„Der Datensatz wurde zum Trainieren der Algorithmusmodelle für Hass/Nicht-Hass, Intensität und Typ verwendet. Der Datensatz besteht aus Nachrichten (Nachrichten und Kommentaren), die von Benutzern in El Mundo, ABC, La Vanguardia, El País und 20 Minutos im Januar 2021 auf Twitter und deren Websites veröffentlicht wurden.“

„Datensatz entwickelt im Rahmen des Hatemedia-Projekts (PID2020-114584GB-I00), finanziert von MCIN/AEI/10.13039/501100011033, mit Unterstützung des kooperierenden Unternehmens [Possible Inc.](#)“

Dieser Datensatz ist der größte, der gefunden wurde, und ist bereits teilweise verarbeitet (Stoppwörter, Zahlen usw. wurden entfernt).

### 2. HuggingFace

Auf der Website dieses KI- und Machine-Learning-Unternehmens habe ich einen Superdatensatz mit Botschaften auf Spanisch gefunden, der aus fünf Datensätzen besteht (zwei aus Spanien – HateEval und Haternet –, zwei weitere aus Chile und einer aus Mexiko).

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

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

### 3. Kaggle

Kaggle ist eine Plattform und Community für KI- und Machine-Learning-Experten sowie -Praktiker. Ich habe hier einen mehrsprachigen Datensatz mit Hassreden gefunden.

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

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

Dieser Datensatz ist in einen Trainingsdatensatz und einen Testdatensatz unterteilt.



Jeder dieser Datensätze weist seine Besonderheiten auf. Der erste Datensatz aus dem Hatemedia-Projekt ist recht umfangreich (574.272 Beobachtungen) und äußerst unausgewogen. Alle Nachrichten werden am Ende als Hass gekennzeichnet. Darüber hinaus ist der gefundene Datensatz bereits teilweise verarbeitet. Die Kommentare sind wie folgt formatiert:

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

Da Kaggle-Datensätze mehrsprachig sind, müssen sie vorverarbeitet werden, um nur die Nachrichten auf Spanisch zu extrahieren.

Das Endziel besteht darin, 4 Datensätze mit demselben Format zu erhalten, sodass sie bei Bedarf zu einem einzigen kombiniert werden können.

\* Es sind weitere Spalten verfügbar, aber nur „Kommentar“ und „Beschriftung“ sind erforderlich.

## 4. Ergebnis

Das Ergebnis dieser ersten Verarbeitung sind 4 Datenrahmen, alle mit derselben Struktur (zwei Spalten: „Post“ und „Label“).

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

Die Gesamtzahl der Rohbeobachtungen beträgt:

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

Ich denke, das ist eine beträchtliche Menge an gekennzeichneten Kommentaren, um ein SVM-Modell richtig trainieren zu können.



### 3. Phase 2 – SVM-Schulung

#### 1. Trainieren Sie SVM mit jedem der Datensätze.

Da 3 verschiedene Datensätze gefunden wurden, besteht das Ziel darin, herauszufinden, ob ihre Eigenschaften (Herkunft, Größe ...) einen Einfluss auf die Qualität des SVM-Modells haben, der beim Training dieses Algorithmus mit diesen Datensätzen erzielt werden kann.

##### 1. Kaggle-Datensatz

Laden der gemeinsamen Funktionen, die zur Verarbeitung aller Datensätze erforderlich sind (Code im ANHANG 2 dargestellt).

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

Laden Sie die für verschiedene Funktionen erforderlichen Pakete.

```
load_libraries ()

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

#### Schritt 1: Beschaffung des Datensatzes und Vorverarbeitung

```
#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
```

Datensatzstruktur:

```
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 ...
```

Die Beschriftungsspalte ist vom Typ int. Da es sich tatsächlich um eine kategoriale Variable vom Typ 0/1 handelt, ist es praktisch, sie in einen Faktor umzuwandeln:

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

Wir untersuchen das Ergebnis:

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

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

Dieser Datensatz ist unausgewogen (wenn auch nicht so unausgewogen wie die anderen beiden).

```
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
```

## Schritt 2: Verarbeiten des Datensatzes

In einem ersten Schritt werden Verweise auf andere Nutzer in Kommentaren entfernt.

Auch untersuchen wir mit Hilfe der *Cleantext -Bibliothek* die Kommentare und führen eine erste Verarbeitung durch:

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

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

Wir prüfen das Ergebnis:

```
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)
```

## Korpus

Erstellung des Korpus-Objekts mit allen Nachrichten:

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

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

Bereinigung der Kommentare im Korpus:

```
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
```

Der Korpus wird untersucht und es wird nichts Ungewöhnliches gefunden.

```
#View(posts_corpus_clean)
```

## Tokenisierung

Abschließend werden die Kommentare tokenisiert. Man erhält ein DTM:

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

posts_dtm

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

Jetzt müssen wir die Trainings- und Test-Sets erstellen.

Da dieser Datensatz bereits in zwei Teile aufgeteilt ist, laden wir den Testdatensatz:

```
# 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]
```

Wir verarbeiten es auf die gleiche Weise wie das Trainingsset:

```
#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
```

Reinigung des Test-Korpus:

```
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
```

Tokenisierung:

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

Jetzt müssen wir eine Liste der am häufigsten verwendeten Wörter erstellen:

```
# 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 ]
```

Und jetzt verwenden wir diese Liste, um die Anzahl der Spalten/Features sowohl im Trainings- als auch im Test-Set zu begrenzen:

```
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
```

### Schritt 3: Modelltraining und -evaluierung

```
# 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
```

Ein sehr wichtiger Parameter in diesem Aufruf ist „type“. Wie wir in der Dokumentation nachlesen können, kann „type“ die folgenden Werte annehmen:

Für die Mehrklassenklassifizierung

- 0 - logistische Regression mit L2 (primärer) Regularisierung
- 1 - „Support Vector Classification“ mit L2-Regularisierung und L2-(Dual-)Verlustfunktion
- 2 - „Support Vector Classification“ mit L2-Regularisierung und L2-(primärer) Verlustfunktion
- 3 - „Support Vector Classification“ mit L2-Regularisierung und L1-(Dual-)Verlustfunktion
- 4 - Crammer und Singer „Support Vector-Klassifizierung“ (Multiklassen).
- 5 - „Support Vector Classification“ mit L1-Regularisierung und L2-Verlustfunktion
- 6 - Logistische Regression mit L1-Regularisierung
- 7 - logistische Regression mit L2 (dualer) Regularisierung

Die Autoren von LIBLINEAR sagen außerdem Folgendes:

„Bei der Wahl zwischen L1- und L2-Regularisierung empfehlen wir, zuerst L2 auszuprobieren, es sei denn, der Benutzer benötigt ein spärliches Modell.“

Daher wurden Tests mit den Optionen 1, 2, 3 und 5 durchgeführt. Da die Zeiten ziemlich ähnlich waren, wurde Typ = 3 (L2-Regularisierung, obwohl wir spärliche Matrizen\* und L1-Verlustfunktion haben) gewählt, da er die besten Ergebnisse lieferte.

\*Der Hatemedia-Datensatz konnte nur mit dünn besetzten Matrizen verarbeitet werden. Bei den Kaggle- und HuggingFace-Datensätzen gab es keinen Unterschied zwischen der Verwendung des einen oder anderen Matrixtyps.

```

# 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
##

```

Ich halte dieses erste Ergebnis nur für akzeptabel. Die Genauigkeit liegt unter 80 % und der Kappa-Wert unter 0,5.

Das gleiche Ergebnis wird mit dichten und spärlichen Matrizen erzielt. Da dieser Datensatz klein ist, gibt es keine Probleme mit dichten Matrizen. (Dadurch kann die heuristische Funktion für Kostenberechnungen verwendet werden.)

Bei den anderen Datensätzen war es nicht nur wichtig, spärlichen Matrizen zu verwenden, sondern auch das DTM in „Chunks“ zu konvertieren, da die RStudio-Sitzung sonst aufgrund der physischen Speicherbeschränkungen des Computers „explodieren“ würde.

## Modellverbesserung

- Wir haben versucht, das Modell mithilfe der von der Funktion `heuristicC` berechneten Kosten zu verbessern, wie von den Entwicklern der Bibliothek `LIBLINEAR` empfohlen:

```

# 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
##

```

Bei Verwendung der mit der Heuristik berechneten Kosten war das Ergebnis in unserem Fall schlechter. Bei der Wahl eines Kostenwerts über 1 (1,5, 2, 3, 5 usw.) verbessert sich das Ergebnis jedoch (nur geringfügig). Das beste Ergebnis wurde mit einem Kostenwert von 10 erzielt.

- Wir haben versucht, das Modell auch durch die Verwendung von Gewichten zu verbessern, um die Erkennung der Minderheitsklasse zu begünstigen:

```

# 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, wi = 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
##
```

Beim Testen verschiedener Kombinationen des Kostenparameters und der Gewichte liegen die besten erzielten Werte bei einer Genauigkeit von etwa 76-77 % und einem Kappa von 0,47.

Vergleichstabelle:

(Nächste Seite)

	Original	Kosten 10	
Konfusionsmatrix	Reference Pred. No Yes No <b>737</b> 208 Yes 82 <b>216</b>	Reference Pred. No Yes No <b>734</b> 201 Yes 85 <b>223</b>	
Genauigkeit	0,7667	<b>0,7699</b>	
Kappa	0,4409	0,4520	
F1	0,5983	0,6093	
MCC	0,4526		
	Kosten 1, Gewichte 1/3	Kosten 10, Gewichte 1/3	Kosten 1, Gewichte 2/3
Konfusionsmatrix	Reference Pred. No Yes No <b>496</b> 88 Yes 323 <b>336</b>	Reference Pred. No Yes No <b>490</b> 83 Yes 329 <b>341</b>	Reference Pred. No Yes No <b>666</b> 145 Yes 153 <b>279</b>
Genauigkeit	0,6693	0,6685	0,7603
Kappa	0,3511	0,3531	<b>0,4691</b>
F1	0,6205	0,6234	<b>0,6519</b>
MCC			<b>0,4677</b>

## 2. HuggingFace-Datensatz

Da der Prozess zu 99 % identisch ist, werden nur die Teile des Codes angezeigt, die sich unterscheiden oder als am interessantesten erachtet werden.

### Schritt 1: Beschaffung des Datensatzes und Vorverarbeitung

```
# 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
```

Wir wandeln die Labelvariable in einen Faktor um:

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

Wir untersuchen das Ergebnis:

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

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

Dieser Datensatz ist auch ziemlich unausgewogen, wie in der ersten Phase (Datensatzsammlung) dieses Projekts zu sehen ist.

Wir untersuchen einige der Kommentare dieses Datensatzes:

```
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
```

Die bereits erwähnte Besonderheit dieses Datensatzes besteht darin, dass er Kommentare in verschiedenen Varianten des Spanischen enthält: Es gibt Nachrichten aus Chile und Mexiko, nicht nur aus Spanien. Dies wird sich wahrscheinlich negativ auf die Leistung des Modells auswirken, da es die Vielfalt der Ausdrücke und Redewendungen erhöht, die der Algorithmus verstehen und verallgemeinern können muss, um neue Texte zu klassifizieren (selbst wenn nur Kommentare auf Spanisch aus Spanien klassifiziert werden).

Notiz:

Beispielsweise wird in den Nachrichten 2 und 3 das Wort „flaiterio“ verwendet. Der Autor dieses Werkes kannte die Bedeutung dieses Wortes überhaupt nicht. Laut der Vereinigung der Akademien der spanischen Sprache:

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

flach.		
I.	1.	Substantiv/Adjektiv. <i>Ch.</i> juv. Eine Person aus einer niedrigeren sozialen Schicht, die zu aggressivem Verhalten neigt und sich etwas extravagant kleidet. Verachtung.
	2.	Adj./Substantiv <i>Ch.</i> <i>Bezieht sich auf eine Person</i> mit unkultiviertem Verhalten.
	3.	mf. <i>Ch.</i> Dieb. Delinquent.
	4.	Adj. <i>Ch.</i> <i>Bezieht sich auf etwas</i> Geschmackloses.
	5.	<i>Kap.</i> <i>Bezieht sich auf etwas</i> von schlechter oder minderer Qualität.

(Tabelle aus dem Spanischen übersetzt).

Es scheint ein Kandidat für das Wort zu sein, das in Hassbotschaften auftaucht.

## Schritt 2: Verarbeitung des Datensatzes

Es wird dieselbe Verarbeitung wie für den vorherigen Datensatz durchgeführt.

Um Wiederholungen zu vermeiden, werden nur einige Fragmente des Codes angezeigt.

## Korpus

Erstellung des Korpus-Objekts mit allen Nachrichten:

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

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

Dieser Datensatz ist fast dreimal größer als der von Kaggle, daher ist logischerweise auch das Korpus fast dreimal größer.

Zur Erstellung des DTM wird ebenfalls eine Tokenisierung durchgeführt, anschließend werden die Trainings- und Testdatensätze erstellt. Dazu verwenden wir eine der Funktionen im gemeinsamen Modul:

```
#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
```

Jetzt überprüfen wir, ob wir das (Un-)Verhältnis von Nicht-Hass-Nachrichten zu Hass-Nachrichten beibehalten:

```
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
```

Wir haben immer noch ein Verhältnis von 75/25 (nicht Hassbotschaften/Hassbotschaften).

Jetzt müssen wir eine Liste der am häufigsten verwendeten Wörter erstellen, um die Anzahl der Spalten/Variablen in den Trainings- und Testsätzen zu begrenzen:

```
#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
```

Der vorherige Datensatz war zu klein. In diesem habe ich mit der Mindesthäufigkeit experimentiert. Vor der Verwendung dünnbesetzter Matrizen war dieser Faktor aus folgendem Grund sehr wichtig: Je niedriger die zur Auswahl eines Wortes erforderliche Häufigkeit, desto mehr Wörter werden logisch ausgewählt und die Matrix wächst proportional. Bei Matrizen einer bestimmten Größe stürzte die RStudio-Sitzung ab. Nachdem dieses Problem gelöst war, versuchte ich, den niedrigstmöglichen Häufigkeitswert zu wählen. Die Gesamtzahl der Beobachtungen habe ich dabei nur teilweise berücksichtigt. Deshalb habe ich in diesem Datensatz statt einer Mindestanzahl von 10 Vorkommen 20 gewählt.

### Schritt 3: Modelltraining und -evaluierung

Es handelt sich um den gleichen Code für alle Datensätze, sodass die Ergebnisse direkt angezeigt werden:

#### Erstes Modell (Kosten = 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
##

```

### Vergleichstabelle der Ergebnisse:

	Kosten 1,Frequenz 20	Kosten 1,Frequenz 50	Kosten 1, Frequenz 100
Konfusionsmatrix	Reference Pred. No Yes No <b>5216</b> 845 Yes 431 <b>971</b>	Reference Pred. No Yes No <b>5346</b> 999 Yes 301 <b>817</b>	Reference Pred. No Yes No <b>5384</b> 1249 Yes 262 <b>567</b>
Genauigkeit	<b>0,8290</b>	0,8258	0,7975
Kappa	0,4968	0,4560	0,3259
F1	0,6035	0,5569	0,4287
MCC	0,5035		
	Kosten 1, Frequenz 20 Gewichte 1/3	Kosten 0,289722 Frequenz 20, Gewichte 2/3	
Konfusionsmatrix	Reference Pred. No Yes No <b>4474</b> 487 Yes 1173 <b>1329</b>	Reference Pred. No Yes No <b>5002</b> 694 Yes 645 <b>1122</b>	
Genauigkeit	0,7776	0,8206	
Kappa	0,4646	<b>0,5083</b>	
F1	0,6156	<b>0,6263</b>	
MCC		<b>0,5088</b>	

Je mehr Wörter das Trainingsvokabular enthält, desto besser scheinen die Ergebnisse zu sein. Mit dem vorherigen Kaggle-Datensatz lag die Genauigkeitsrate bei 76-77% und der Kappa-Wert bei 0,47. Jetzt ist die Genauigkeitsrate auf über 80% gestiegen, mit einem besseren Kappa-Wert von etwa 0,51.

### 3. Hatemedia-Datensatz

#### Schritt 1: Beschaffung des Datensatzes und Vorverarbeitung

```
# 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
```

Angesichts der Größe dieses Datensatzes (und der physischen Einschränkungen des Laptops) können wir wählen, ob wir den gesamten Datensatz verarbeiten oder eine Teilmenge der Nicht-Hassbotschaften plus alle Hassbotschaften auswählen möchten. Dies ist eine Möglichkeit, ein **Downsampling** durchzuführen, bei dem jedoch die Anzahl der Nicht-Hassbotschaften ausgewählt wird, anstatt automatisch die gleiche Anzahl von Nachrichten aus beiden Klassen auszuwählen.

```
# 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!!!"
```

Es wurden Tests mit unterschiedlich großen Datensätzen durchgeführt. Wie aus dem Vergleich am Ende dieses Abschnitts hervorgeht, wurde das beste Ergebnis mit dem 100.000 Beobachtungsdatensatz erzielt.

#### Datensatzstruktur:

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



Die Spalte „label“ ist vom Typ int. Da es sich tatsächlich um eine kategorische Variable vom Typ 0/1 handelt, ist es besser, sie in einen Faktor umzuwandeln:

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

Wir wussten bereits, dass dies ein sehr, sehr unausgewogener Datensatz war. Der größte und unausgewogenste der drei:

```
table(odio_hatemediaw$rowlabel)
##
##      0      1
## 563208 11064

prop.table(table(odio_hatemediaw$rowlabel))
##
##      0      1
## 0.98073387 0.01926613
```

Der Anteil der Hassbotschaften in diesem Datensatz beträgt weniger als 2 %.

In diesem Datensatz müssen die ","-Zeichen in der Variable "post" durch Leerzeichen ersetzt werden. Andernfalls erhält man jeden Kommentar als eine einzige, riesige Zeichenfolge, die alle Wörter des Kommentars aneinandergereiht enthält. (Das liegt daran, dass dieser Hatemediaw-Datensatz, wie bereits erwähnt, bereits teilweise verarbeitet wurde.)

```
# 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)
```

Wir überprüfen das Ergebnis:

```
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
## ...
```

## Schritt 2: Verarbeitung des Datensatzes

In diesem Datensatz wurden Nachrichten auf Katalanisch gefunden:

```
#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 ..." ...
```

Das Ziel dieser Abschlussarbeit des Meisters besteht darin, Hassreden in einer einzigen Sprache zu erkennen, nicht in mehreren Sprachen, selbst wenn diese sich so ähnlich sind wie Spanisch und Katalanisch.

Angesichts der Schwierigkeit, in einem so großen Datensatz alle Kommentare auf Katalanisch zu finden, und angesichts der Tatsache, dass viele katalanische Bürger in spanischen Online-Medien ihre Kommentare in ihrer Muttersprache verfassen, wurde entschieden, diese Nachrichten vorerst nicht zu löschen.

## Korpus

Erstellung des Korpus-Objekts mit allen Nachrichten:

```
#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
```

Die Anzahl der Dokumente ist mittlerweile geringer als die ursprüngliche Zahl, da im Zuge dieser Bearbeitung knapp 4.000 Datensätze gelöscht wurden.

Sobald der Korpus verarbeitet und **die Tokenisierung** abgeschlossen ist durchgeführt, werden die Trainings- und Testsätze erstellt:

```
#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)
```

Wir prüfen, ob wir das (Un-)Verhältnis von Nicht-Hass-Nachrichten zu Hass-Nachrichten beibehalten:

```
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
```

Wir müssen nun eine Liste mit den am häufigsten verwendeten Wörtern erstellen:

```
# 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
```

Es wurden verschiedene Mindestfrequenzen getestet. Zunächst wurden höhere Mindestfrequenzen gewählt, um die Größe des DTM zu begrenzen, und die Matrix wurde als Argument an die Trainingsfunktion übergeben. Nachdem der gesamte Datensatz verarbeitet werden konnte, wurden für diesen Datensatz Frequenzen von bis zu 100 (und sogar noch niedriger) gewählt.

Und jetzt verwenden wir diese Liste, um die Anzahl der Spalten/Features sowohl im Trainings- als auch im Testsatz zu begrenzen:

```
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
```

Wir können sehen, dass der Trainingssatz mehr als 428.000 Beobachtungen enthält und von 348.000 Variablen (Begriffen) auf nur 16.516 zurückgeht.

### Schritt 3: Modelltraining und -evaluierung

Um den gesamten Datensatz verarbeiten zu können, mussten die DTMs zusätzlich zur obligatorischen Verwendung von dünnbesetzter Matrizen stapelweise in diese Matrizen konvertiert werden:

```
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 (Kosten = 1) und 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
```

Die Geschwindigkeit dieser Bibliothek im Vergleich zu LIBSVM (e1071) lässt sich leicht einschätzen. In nur 10 Sekunden wurde ein SVM-Modell mit über 400.000 Beobachtungen auf einem einfachen Computer mit einem Intel Core i5-Prozessor und 16 GB RAM trainiert.

Bei den anderen „Typ“-Optionen waren die Laufzeiten sehr ähnlich. (Bei LIBSVM/e1071 war der Prozess jedoch mehr als eine Stunde nach seinem Start noch nicht abgeschlossen und ich musste ihn unterbrechen.)

```
# 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
##

```

Angehts der Größe des Test-Datensatzes und des Ungleichgewichts zwischen den beiden Klassen ist es völlig logisch, dass eine so (scheinbar) gute Genauigkeit erreicht wird, aber dennoch so niedrige Kappa- und F1-Werte (und eine ausgeglichene Genauigkeit von weniger als 55 %) erreicht werden. Ganz zu schweigen von der Recall-Rate, (nicht einmal 10 % der Hassbotschaften erkannt werden) ...

Offensichtlich geht es hier nicht darum, ein Modell zu erstellen, das Nicht-Hassbotschaften einfach als Nicht-Hassbotschaften klassifiziert.

Ich habe mehrere Tests mit diesem Datensatz durchgeführt. Ein Vergleich ist unten dargestellt. Ergebnisse, die mit Kosten = 1 erzielt wurden, wobei die Datensatzgröße (d. h. der Grad des "Klassen-Ungleichgewicht") und die Mindestfrequenz variiert wurden:

- Datensatz mit 300.000 Beobachtungen:

Niedrige Frequenz (100,200,300,400): R-Sitzung „explodiert“

Mittlere Frequenz (500): hohe Genauigkeit (96 %), niedriges Kappa (0,28)

Gewichte 1/2 -> 96% und 0,36

Gewichte 1/5 -> 95 % und 0,38 (Kappa verbessert sich etwas)

Hohe Frequenz (1000): „absurde“ Ergebnisse (0 TP, 0 Kappa und Gewichte können das nicht beheben)

- Datensatz mit 100.000 Beobachtungen:

Niedrige Frequenz (100): hohe Genauigkeit (92,5 %) und akzeptables Kappa (0,58)

Bei Gewichten von 1/2 bleibt die Genauigkeit gleich (**92%**) und Kappa **0,61**.

- Datensatz mit 30.000 Beobachtungen (nicht so unausgewogen: 19.000 Nicht-Hass vs. 11.000 Hass):

Niedrige Frequenz (100) -> Hohe Genauigkeit (83 %), gutes Kappa (0,62)

hohe Frequenz (500) -> geringe Genauigkeit, vernachlässigbares Kappa.

Wie wir sehen können, wird mit dem Datensatz von 100.000 Beobachtungen ein durchaus akzeptables Ergebnis erzielt. Es ist jedoch zu beachten, dass diese Ergebnisse erzielt wurden, bevor ich den gesamten Datensatz verarbeiten konnte. Wir werden versuchen, das Ergebnis mit dem vollständigen Datensatz (98 % Genauigkeit und Kappa 0,15) zu verbessern, da das Endziel darin besteht, durch die Kombination der drei oben genannten Datensätze den größtmöglichen Datensatz zu erstellen.

	Datensatz <b>100k</b> , Kosten 1, Frequenz 100, Gewichte 1/2	Vollständiger Datensatz, Kosten 1, Frequenz 100	Vollständiger Datensatz, Kosten 1, Frequenz 100, Gewichte 1/10																																																
Konfusions matrix	<table> <tr> <td></td><td>Pred.</td><td colspan="2">Reference</td></tr> <tr> <td></td><td></td><td>No</td><td>Yes</td></tr> <tr> <td>No</td><td></td><td><b>21145</b></td><td>947</td></tr> <tr> <td>Yes</td><td></td><td>954</td><td><b>1819</b></td></tr> </table>		Pred.	Reference				No	Yes	No		<b>21145</b>	947	Yes		954	<b>1819</b>	<table> <tr> <td></td><td>Pred.</td><td colspan="2">Reference</td></tr> <tr> <td></td><td></td><td>No</td><td>Yes</td></tr> <tr> <td>No</td><td></td><td><b>139756</b></td><td>2507</td></tr> <tr> <td>Yes</td><td></td><td>195</td><td><b>259</b></td></tr> </table>		Pred.	Reference				No	Yes	No		<b>139756</b>	2507	Yes		195	<b>259</b>	<table> <tr> <td></td><td>Pred.</td><td colspan="2">Reference</td></tr> <tr> <td></td><td></td><td>No</td><td>Yes</td></tr> <tr> <td>No</td><td></td><td><b>139957</b></td><td>992</td></tr> <tr> <td>Yes</td><td></td><td>4994</td><td><b>1774</b></td></tr> </table>		Pred.	Reference				No	Yes	No		<b>139957</b>	992	Yes		4994	<b>1774</b>
	Pred.	Reference																																																	
		No	Yes																																																
No		<b>21145</b>	947																																																
Yes		954	<b>1819</b>																																																
	Pred.	Reference																																																	
		No	Yes																																																
No		<b>139756</b>	2507																																																
Yes		195	<b>259</b>																																																
	Pred.	Reference																																																	
		No	Yes																																																
No		<b>139957</b>	992																																																
Yes		4994	<b>1774</b>																																																
Genauigkeit	0,9235	<b>0,9811</b>	0,9581																																																
Kappa	<b>0,6138</b>	0,1563	0,3544																																																
F1	<b>0,6568</b>	0,1608	0,3721																																																
MCC	<b>0,6138</b>	0,2258																																																	
	Vollständiger Datensatz, Kosten 1, Frequenz 100, Gewichte 1/5	Vollständiger Datensatz, Kosten 1, Frequenz 100 Gewichte 1/3	Vollständiger Datensatz, Kosten 1, Frequenz 100 Gewichte 1/2																																																
Konfusions matrix	<table> <tr> <td></td><td>Pred.</td><td colspan="2">Reference</td></tr> <tr> <td></td><td></td><td>No</td><td>Yes</td></tr> <tr> <td>No</td><td></td><td><b>136865</b></td><td>1277</td></tr> <tr> <td>Yes</td><td></td><td>3086</td><td><b>1489</b></td></tr> </table>		Pred.	Reference				No	Yes	No		<b>136865</b>	1277	Yes		3086	<b>1489</b>	<table> <tr> <td></td><td>Pred.</td><td colspan="2">Reference</td></tr> <tr> <td></td><td></td><td>No</td><td>Yes</td></tr> <tr> <td>No</td><td></td><td><b>138069</b></td><td>1583</td></tr> <tr> <td>Yes</td><td></td><td>1882</td><td><b>1183</b></td></tr> </table>		Pred.	Reference				No	Yes	No		<b>138069</b>	1583	Yes		1882	<b>1183</b>	<table> <tr> <td></td><td>Pred.</td><td colspan="2">Reference</td></tr> <tr> <td></td><td></td><td>No</td><td>Yes</td></tr> <tr> <td>No</td><td></td><td><b>138903</b></td><td>1937</td></tr> <tr> <td>Yes</td><td></td><td>1048</td><td><b>829</b></td></tr> </table>		Pred.	Reference				No	Yes	No		<b>138903</b>	1937	Yes		1048	<b>829</b>
	Pred.	Reference																																																	
		No	Yes																																																
No		<b>136865</b>	1277																																																
Yes		3086	<b>1489</b>																																																
	Pred.	Reference																																																	
		No	Yes																																																
No		<b>138069</b>	1583																																																
Yes		1882	<b>1183</b>																																																
	Pred.	Reference																																																	
		No	Yes																																																
No		<b>138903</b>	1937																																																
Yes		1048	<b>829</b>																																																
Genauigkeit	0,9694	0,9757	0,9791																																																
Kappa	0,3910	<b>0,3934</b>	0,3469																																																
F1	0,4057	<b>0,4058</b>	0,3571																																																
MCC		<b>0,3935</b>																																																	

Da der Datensatz so unausgewogen war, entschied ich mich für ein Gewichtsverhältnis von 1 zu 10. Dadurch wurde Kappa im Vergleich zum ursprünglichen Modell deutlich verbessert (von 0,15 auf 0,35).

Mit den Gewichten 1/5 und 1/3 werden sehr ähnliche Genauigkeits-, Kappa- und F1-Werte erzielt. Wir haben die Wahl zwischen der Erkennung von mehr PTs auf Kosten der Gewinnung von mehr FPs oder der Erkennung von weniger PTs und auch weniger FPs.

Die Kappa-Werte sind jedenfalls nicht besonders gut und liegen nur bei 0,4.

Im Vergleich zu früheren Datensätzen hat sich die Genauigkeit erneut verbessert, Kappa ist jedoch von 0,5 auf 0,4 gesunken.

(Es sei denn, wir betrachten den Datensatz mit 100.000 Beobachtungen, der das beste Ergebnis von allen liefert).

Da sich Kappa nicht verbesserte und die Möglichkeit bestand, dass die Wahl einer so niedrigen Frequenz zu einer Überanpassung der Texte führen würde (wobei man sich bewusst war, dass SVM nicht zu diesem Problem neigt – einer der Gründe, warum diese Technik gewählt wurde), wurde eine höhere Mindestfrequenz getestet: 500 (anstelle von 100).

Mit  $\text{freq} = 500$  verringerte sich die Größe der DTMs und Matrizen erheblich:

```
[1] 428151 348382
```

```
[1] 428151    5886 ← 5886 Spalten/Features statt 16516 mit Frequenz 100
```

```
[1] 142717 348382
```

```
[1] 142717    5886 ← idem.
```

(Interessanterweise wurden die Trainingszeiten länger).

Doch die erzielten Ergebnisse sind nicht besser. Im Gegenteil:

	Vollständiger Datensatz, Kosten 1, Frequenz 100	Vollständiger Datensatz, Kosten 1, Frequenz 500																								
Konfusions 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><b>139756</b></td><td>2507</td></tr> <tr> <th>Yes</th><td>195</td><td><b>259</b></td></tr> </table>		Reference		Pred.	No	Yes	No	<b>139756</b>	2507	Yes	195	<b>259</b>	<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><b>139819</b></td><td>2613</td></tr> <tr> <th>Yes</th><td>132</td><td><b>153</b></td></tr> </table>		Reference		Pred.	No	Yes	No	<b>139819</b>	2613	Yes	132	<b>153</b>
	Reference																									
Pred.	No	Yes																								
No	<b>139756</b>	2507																								
Yes	195	<b>259</b>																								
	Reference																									
Pred.	No	Yes																								
No	<b>139819</b>	2613																								
Yes	132	<b>153</b>																								
Genauigkeit	<b>0,9811</b>	0,9808																								
Kappa	0,1563	<b>0,097</b>																								
F1	0,1608	<b>0,1003</b>																								

Das Ergebnis ist noch schlechter als bei  $\text{freq} = 100$ .

(Ohne Gewichte ist es auch nicht möglich, die mit  $\text{freq} = 100$  erzielten Ergebnisse zu verbessern).



## 2. Erstellung eines vollständigen Datensatzes und Training eines SVM darauf

Die Idee, alle Datensätze zusammenzuführen, besteht darin, sicherzustellen, dass das SVM durch die Bereitstellung vielfältigerer (und größerer, aber vor allem vielfältigerer) Informationsquellen besser trainiert werden kann und somit die bestmöglichen Ergebnisse mit den ihm präsentierten neuen, unbeschrifteten Nachrichten erzielt.

Angesichts der guten Ergebnisse, die mit dem Hatemedia-Datensatz mit 100.000 Beobachtungen (Genauigkeit 92% und Kappa 0,61) erzielt wurden, wird in dieser Phase des Projekts zusätzlich zum Training des SVM-Modells mit dem Gesamtdatensatz, der aus den drei Datensätzen besteht, aus diesem „Gesamtdatensatz“ auf die gleiche Weise ein weiterer mit 100.000 Kommentaren erstellt: Er wird alle Hassbotschaften aus dem Gesamtdatensatz und eine bestimmte Anzahl von Nicht-Hassbotschaften enthalten, bis wir diese 100.000 Größe erreichen.

Anschließend wird SVM mit beiden Datensätzen trainiert und die Ergebnisse (auch mit den vorherigen) verglichen.

### 1. Erstellung des Gesamtdatensatzes

Globale Variablen:

```
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
```

Alle CSV-Dateien importieren:

```
# 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
```

Wir wissen bereits, dass der Hatemedia-Datensatz etwas anders ist. Wir entfernen das "," erneut, um Probleme zu vermeiden.

```
# 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)
```

Jetzt können wir einen „Gesamt“-Datensatz mit allen verfügbaren Datensätzen erstellen:

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

Wir verfügen nun über einen Datensatz mit insgesamt 616.550 Kommentaren von Lesern von Online-Zeitungen und anderen Quellen.

[  
Wir entfernen nicht mehr benötigte Datensätze und Variablen:

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

Wir überprüfen den Anteil der Nachrichten in diesem Datensatz:

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

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

Da der Hatemedia-Datensatz viel größer als die beiden anderen und sehr unausgewogen ist, ist der resultierende Datensatz offensichtlich auch sehr unausgewogen.

## 2. Erstellen des 100k-Beobachtungs-Datensatzes und Training

Da wir 22.568 Hassnachrichten haben, werden wir zufällig die folgende Anzahl von Nachrichten ohne Hass auswählen (gerundet auf 78.000, wie in der Variable „Größe“ am Anfang angegeben):

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

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

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

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

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

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

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

rm(df_hate,df_no_hate,df_no_hate_sample,hate_raw_sample_k)

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

Wir überprüfen den Anteil der Nachrichten in diesem Datensatz nach dem Downsampling:

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

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

Der Datensatz ist immer noch unausgewogen, allerdings in einem viel geringeren Ausmaß.

(Beim Hatemedia-Datensatz wurde kein besseres Ergebnis erzielt, indem der Grad des Ungleichgewichts weiter reduziert wurde, indem die Gesamtzahl der Nicht-Hass-Beobachtungen auf 30.000 gesenkt wurde.)

Datensatzstruktur:

```
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 Odiame pero al mirarte a un espejo recuerda: yo soy agil y atractivo, tú eres una puta
cerda Y es que tienes tanta tontería y tienes tanta maldad, que en algún sitio de tu cuerpo se tiene
que acumular

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

## Datensatzverarbeitung

Die Spalte „label“ ist vom Typ int. Da es sich tatsächlich um eine kategorische Variable vom Typ 0/1 handelt, ist es besser, sie in einen Faktor umzuwandeln:

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

Kommentare verarbeiten wir wie gewohnt (Verweise auf andere Leser entfernen, Emoticons entfernen..):

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

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

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

```
##      user      system elapsed
```

```
## 59.783    0.028    59.820
```

Dabei kann es vorkommen, dass Kommentare leer bleiben. Wir löschen diese dann:

```
# 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)
```

## Korpus

Wir können nun mit der Erstellung des Korpus-Objekts mit allen Nachrichten fortfahren:

```
#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
```

Reinigung des Korpus:

Übliche Verarbeitung: Großbuchstaben, Zahlen usw. entfernen.

```
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
```

Abschließend werden die Kommentare **tokenisiert**:

```
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: 3387698/1373733722
## Sparsity           : 100%
## Maximal term length: 279
## Weighting          : term frequency (tf)
```

[  
Löschen Sie an dieser Stelle die „Corpus“-Objekte. Sie werden nicht mehr benötigt.

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

Jetzt müssen wir die Trainings- und Testsätze erstellen:

```
#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)
```

Lassen Sie uns überprüfen, ob das (Un-)Verhältnis zwischen Hassbotschaften und Nicht-Hassbotschaften erhalten bleibt:

```
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
```

Jetzt müssen wir die Liste der am häufigsten verwendeten Wörter erstellen:

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

Und jetzt verwenden wir diese Liste, um die Anzahl der Spalten/Features sowohl im Trainings- als auch im Testsatz zu begrenzen:

```
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
```

Wir hatten vorher 137.222 Spalten/Features und jetzt nur noch 4.716.

### Modelltraining und -evaluierung

Um das Modell zu trainieren, müssen wir die DTMs in Matrizen umwandeln. Angesichts der Größe des Datensatzes und der physikalischen Einschränkungen der Geräte, auf denen dieser Prozess durchgeführt wird, werden die DTMs stapelweise verarbeitet und die Matrizen anschließend kombiniert, um die vollständige Matrix zu erhalten.

```
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

```

## Vorhersage:

```

# 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

```

## Auswertung des Ergebnisses:

```

# 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
##
```

Wenn wir einen unausgewogenen Datensatz haben, aber in einem solchen Verhältnis (3 zu 1 zugunsten von Nicht-Hassbotschaften), erhalten wir ein mehr als akzeptables Ergebnis. Eine Genauigkeit von nahezu **90 %** und ein Kappa von nahezu **0,7** erscheinen mir mehr als gut (auch die restlichen Indikatoren scheinen recht gut).

Wenn man auch bedenkt, dass dieser Datensatz mit 100.000 Beobachtungen für mich recht realistisch erscheint (die HuggingFace- und Kaggle-Datensätze sind einzeln betrachtet viel kleiner).

Wir können dieses Ergebnis der Zusammenführung der drei Datensätze und des Erhalts eines Datensatzes mit 100.000 Beobachtungen auch mit dem von 100.000 Beobachtungen von Hatemedia vergleichen: Kappa verbessert sich von 0,61 auf 0,7.

Notiz:

Da wir zur Verarbeitung eines Datensatzes dieser Größe dünn besetzte Matrizen verwenden, können wir die heuristische Funktion von LIBLINEAR nicht zur Berechnung der Kosten verwenden. Nach umfangreichen Tests haben wir jedoch festgestellt, dass Gewichtungen das Ergebnis deutlich stärker verbessern als die Verwendung eines anderen Kostenwerts als 1.

- Verbesserung des Modells durch Gewichte

Es wurde mit den Verhältnissen 1/5, 1/3 und 1/2 getestet. Nur mit diesen letzten Gewichten verbessert sich das anfängliche Ergebnis:

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

Vorhersage:

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

Auswertung des Ergebnisses:

```
#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
##

```

Ich halte dieses Ergebnis für ziemlich gut: Genauigkeit von **90 %** , Kappa etwas höher als **0,7** , F1 nahe **0,8** ...

Trotz dieser Ergebnisse habe ich beschlossen, zwei weitere Techniken auszuprobieren, um zu sehen, ob sie sich zumindest geringfügig verbessern lassen. Wir werden die von dieser Bibliothek angebotene Kreuzvalidierung und eine Rastersuche mit verschiedenen Parametern ausprobieren.

## - Kreuzvalidierung/Cross-validation

Nutzen wir die Kreuzvalidierung von LIBLINEAR, um die Ergebnisse mit meinen bisherigen Ergebnissen zu vergleichen. Wie in der Dokumentation (und auch in früheren Aufgaben dieses Masterstudiengangs) beschrieben, ist die Genauigkeitsmetrik bei stark unausgewogenen Datensätzen nicht die beste Wahl. Versuchen wir es trotzdem:

```

#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

```

## Notiz:

Aus Neugier habe ich versucht, die logistische Regression (Typ = 0) mit SVM zu vergleichen. Nach einer Stunde lief es immer noch. Im Gegensatz dazu konnten die vier SVM-Typen in weniger als 10 Minuten abgeschlossen werden.

```

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

```

Nach den durchgeführten Tests ist das Ergebnis nicht überraschend: Das beste Ergebnis wird mit Typ 3 und Kosten 1 erzielt.

## - Rastersuche/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

```

Durch diese Rastersuche erhält man ein etwas anderes Modell, das Ergebnis ist jedoch nahezu identisch: Genauigkeit von 90 % und Kappa 0,71.

Da ich dieses Modell für „komplizierter“ halte (Kosten 10 und Bias 10 anstelle der Standardwerte) und sein Ergebnis praktisch dasselbe ist, bleibe ich bei dem zuvor erhaltenen Modell (Typ 3, Gewichte 1/2).

Wir fahren nun mit dem Training von SVM mit dem vollständigen Datensatz (mehr als 600.000 Beobachtungen) fort.

### 3. Training mit dem Gesamtdatensatz

Vor der Verarbeitung des gesamten Datensatzes wurde bevorzugt, zunächst mit dem Datensatz der 100.000 Beobachtungen zu arbeiten und das Gelernte auf diesen Datensatz anzuwenden (wenn ich mir beispielsweise das Ergebnis der Kreuzvalidierung und der Rastersuche ansehe, denke ich, dass ich den mit Typ 3 und Kosten 1 erzielten Ergebnissen vertrauen kann).

Da der Code derselbe ist, werden nur die Teile angezeigt, die Informationen zu diesem Datensatz enthalten:

## Datensatzverarbeitung

```
#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)
```

## Korpus

Wir können nun mit der Erstellung des Korpusobjekts mit allen Nachrichten fortfahren:

```
#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
```

Nach der Verarbeitung des Korpus und der Tokenisierung werden die Trainings- und Testsätze erstellt:

```
#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)
```

Wir prüfen, ob wir das (Un)verhältnis von Nicht-Hass-Nachrichten zu Hass-Nachrichten beibehalten:

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

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

Die Liste der am häufigsten vorkommenden Wörter wird generiert und wir verwenden sie, um die Anzahl der Spalten/Features sowohl im Trainings- als auch im Testsatz zu begrenzen:

```
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
```

Die Anzahl der Spalten/Features wurde sowohl im Trainingssatz als auch im Testsatz von 361.649 auf 16.845 reduziert.

### Modelltraining und -evaluierung

Um das Modell zu trainieren, müssen wir die DTMs in Matrizen umwandeln. Angesichts der Größe des Datensatzes und der physikalischen Einschränkungen der Geräte, auf denen dieser Prozess durchgeführt wird, werden die DTMs stapelweise verarbeitet und die Matrizen anschließend kombiniert, um die endgültige Matrix zu erhalten.

```
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
```

## Vorhersage:

```
# prediction
system.time({
  prediction_liblinear <- predict(liblinear_svm_model, posts_freq_test_mat)
})
## user system elapsed
## 0.223 0.000 0.227
```

## Auswertung des Ergebnisses:

```
# 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
##
```



Das Ergebnis finde ich nur akzeptabel. Kappa könnte sicherlich besser sein. Es wird versucht, Gewichte zu finden, die die Erkennung von Hassrede (und damit Kappa) verbessern können.

Notiz:

Da wir zur Verarbeitung eines Datensatzes dieser Größe spärliche Matrizen verwenden, können wir die heuristische Funktion von LIBLINEAR nicht zur Berechnung optimaler Kosten nutzen. Nach mehreren Tests haben wir jedoch festgestellt, dass die Gewichtungen das Ergebnis deutlich stärker verbessern als die Verwendung eines anderen Kostenwerts als 1.

- Verbesserung des Modells durch Gewichte

Es wurden Kreuzvalidierung und Rastersuche getestet. Die besten Ergebnisse wurden mit letzterer Methode erzielt:

```
[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
```

Daher weisen wir die Gewichte 1/3 zu (Typ 3 und Kosten 1 wie üblich):

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
```

Vorhersage:

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

Auswertung des Ergebnisses:

```
#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 wurde von 0,39 auf 0,51 verbessert, während sich die Genauigkeit nur geringfügig verschlechtert hat.

In jedem Fall lässt sich leicht erkennen, dass sowohl beim Hatemedia-Datensatz als auch beim Gesamtdatensatz bessere Ergebnisse erzielt werden, wenn das Modell mit der Teilmenge von 100.000 Beobachtungen trainiert wird, anstatt mit dem Gesamtdatensatz, der äußerst unausgewogen ist.

### 3. Vergleich der Ergebnisse und Schlussfolgerungen.

Vergleich der besten Ergebnisse, die mit jedem Datensatz erzielt wurden:

(Kosten = 1 für alle, außer Datensatz 28k: Kosten = 0,1  
 freq = 100 für alle außer Kaggle, freq = 50, HuggingFace und Superdataset  
 50k, mit freq = 20)

Datensatz	Kaggle	HuggingFace	Hatemia Gewichte 1/3	Hatemia 100k Gewichte 1/2
Konfusionsmatrix	Reference Pred. No Yes No <b>666</b> 145 Yes 153 <b>279</b>	Reference Pred. No Yes No <b>5002</b> 694 Yes 645 <b>1122</b>	Reference Pred. No Yes No <b>138069</b> 1583 Yes 1882 <b>1183</b>	Reference Pred. No Yes No <b>21145</b> 947 Yes 954 <b>1819</b>
Genauigkeit	0,7603	0,8206	<b>0,9757</b>	0,9235
Kappa	0,4691	0,5083	0,3934	<b>0,6138</b>
F1	0,6519	0,6263	0,4058	<b>0,6568</b>
MCC	0,4677	0,5088	0,3935	<b>0,6138</b>
Datensatz		Superdataset 600k Gewichte 1/3	Superdataset 100k Gewichte 1/2	Superdataset 50k Gewichte 1/2
Konfusionsmatrix		Reference Pred. No Yes No <b>144160</b> 2355 Yes 3484 <b>3286</b>	Reference Pred. No Yes No <b>17967</b> 1180 Yes 1425 <b>4461</b>	Reference Pred. No Yes No <b>6167</b> 582 Yes 796 <b>5059</b>
Genauigkeit		<b>0,9619</b>	0,8959	0,8907
Kappa		0,5099	0,7065	<b>0,7797</b>
F1		0,5295	0,7740	<b>0,8801</b>
MCC		0,5121	0,7067	<b>0,7801</b>

Wenn ich alle bisherigen Ergebnisse vergleiche, stelle ich Folgendes fest:

i) Je größer der Datensatz, desto höher die Genauigkeit.

Das ist logisch: Da die Datensätze sehr unausgewogen sind, lernt das Modell sehr gut, die Mehrheitsklasse zu erkennen.

ii) Da es sich um stark unausgewogene Datensätze handelt, ist nicht die Genauigkeit das wichtigste Maß, sondern Kappa. Und Kappa ist besser, wenn der Datensatz auf 100.000 Beobachtungen reduziert wird, und sogar noch besser bei einem Datensatz mit 50.000 Beobachtungen.

iii) Bei gleicher Größe (d. h. bei gleichem Klassen-Ungleichgewicht) liefert der resultierende Datensatz aus der Mischung der drei ursprünglichen Datensätze ebenfalls bessere Ergebnisse. Die ursprüngliche Hypothese, dass unterschiedliche Datensätze SVM eine bessere Generalisierung und bessere Ergebnisse mit neuen Nachrichten ermöglichen, scheint sich zu bestätigen.

iv) Wenn man alle oben genannten Punkte zusammennimmt, ist das beste Ergebnis logischerweise dasjenige, das mit dem SVM erzielt wird, der auf dem Datensatz von 50.000 Beobachtungen aus dem Gesamtdatensatz trainiert wurde. Dieser Datensatz enthält 28.000 Nicht-Hassnachrichten und 22.000 Hassnachrichten.

v) Ich habe am Ende den Matthews-Korrelationskoeffizienten (MCC) hinzugefügt, aber er liefert nicht mehr Informationen als das, was Kappa bereits liefert (es sind praktisch die gleichen Werte).

Ich möchte noch einmal betonen, dass die Datensätze nicht nur Kommentare auf Spanisch, sondern auch auf Katalanisch und in den spanischen Varianten Mexikos und Chiles enthalten und dass mir die Größen (50.000, 100.000 und 600.000 Beobachtungen) recht realistisch erscheinen (genug, um die gezogenen Schlussfolgerungen grundsätzlich zu bestätigen).

## 4. Phase 3 – Klassifizierung neuer unbeschrifteter Kommentare

Bis jetzt wurden ausschließlich im Internet gefundene Datensätze verwendet. In dieser 3. Phase werden neue, unbeschriftete Kommentare klassifiziert mithilfe der in der vorherigen Phase trainierten Modelle.

Konkret ist folgendes beabsichtigt:

- Neue Kommentare erhalten
- Diese mit den zuvor trainierten Modellen klassifizieren
- Bewertung des Ergebnisses

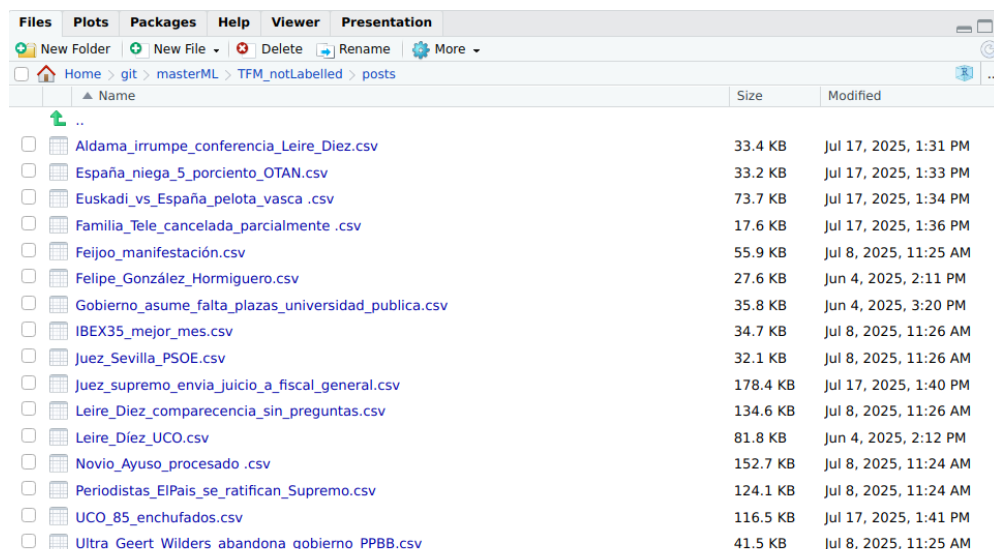
Abhängig vom Ergebnis:

- SVM nur mit neuen Kommentaren trainieren und bewerten.
- Fügen Sie die neuen Kommentare zum Gesamtdatensatz hinzu und SVM mit diesem „verbesserten“ Datensatz trainieren
- Die Ergebnisse vergleichen

### 1. Neue unbeschriftete Kommentare erhalten

Als Quelle für neue Kommentare wurde die Online-Ausgabe von El Mundo ausgewählt. Der Hauptgrund dafür war der einfache Zugang und die weite Verbreitung, die es einfacher macht, viele Kommentare pro Nachricht zu sammeln. Um auf Leserkommentare anderer Zeitungen zugreifen zu können, ist in den meisten Fällen eine Registrierung erforderlich. In diesem Medium sind sie jedoch ohne Barrieren verfügbar (man muss weder Abonnent sein noch sich anmelden).

Insgesamt wurden 3.391 Kommentare aus 16 Nachrichten erhalten:



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

Dabei überwiegen eindeutig nationale politische Nachrichten, da diese Nachrichtenart die größte Leserbeteiligung aufweist.

Beispiele für einige dieser Kommentare:

Andalu\_ilustra|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

...

Der R-Code für Web Scraping ist hier verfügbar:

[https://github.com/fcamadi/masterML/blob/main/TFM\\_webscraping/Web\\_scraping\\_RSelenium\\_periodicos\\_online.Rmd](https://github.com/fcamadi/masterML/blob/main/TFM_webscraping/Web_scraping_RSelenium_periodicos_online.Rmd)

## 2. Klassifizierung neuer unbeschrifteter Kommentare mit vortrainierten Modellen

In diesem Schritt versuchen wir, neue Kommentare mithilfe zuvor trainierter Modelle zu klassifizieren.

### 1. CSV-Dateien mit Leserkomentaren auslesen

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

Wir lesen die Dateien mit den neuen Kommentaren:

```
# 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
```

Variablen umbenennen:

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

Autorspalte entfernen und Spalte „Label“ hinzufügen:

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

## 2. Laden gespeicherter Modelle der Phase 2

Wir laden nun die beiden besten Modelle aus der vorherigen Phase, mit denen die besten Ergebnisse erzielt wurden (Tabelle Seite 46).

Notiz:

Angesichts der damit erzielten Ergebnisse wurde auch ein Versuch mit einem Modell unternommen, das mit einem sehr ausgewogenen Datensatz trainiert wurde.

- „100k“-Modell:

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

- „600k“-Modell:

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

- „Bestes“-Modell (Datensatz 50.000 Beobachtungen, Typ 3, Frequenz 20, Kosten 0,1, Bias 10, Gewichte 1/2):

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

## 3. Klassifizierung unbeschrifteter Kommentare

- Wir wählen Kommentare zufällig aus

Wir haben 2000 Kommentare zufällig aus insgesamt 3000 ausgewählt. Mit diesen 2000 Kommentaren haben wir 2 Datenrahmen erstellt, „subset1“ und „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, ]
```

- Wir klassifizieren Nachrichten mit den zwei (drei) Modellen.

Wir haben 1000 Kommentare (subset1) ausgewählt, um diesen ersten Test zur Klassifizierung neuer unbeschrifteter Kommentare durchzuführen:

```
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
```

## Wir treffen Vorhersagen mit den 3 Modellen:

```
Vorhersagen_100k <- vorhersagen (svm_liblinear_100k, subset1_sparse_matrix_100k)
```

```
#drucken(Vorhersagen_100k$Vorhersagen)
```

```
Vorhersagen_600k <- vorhersagen (svm_liblinear_600k, subset1_sparse_matrix_600k)
```

```
#drucken(Vorhersagen_600k$Vorhersagen)
```

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

```
#drucken(Vorhersagen_100k$Vorhersagen)
```

```
Teilmenge1 $ label_100k <- predictions_100k $ Vorhersagen
```

```
Teilmenge1 $ label_600k <- predictions_600k $ Vorhersagen
```

```
Teilmenge1 $ label_50k_best <- predictions_50k_best $ Vorhersagen
```

## Bewertung des Ergebnisses:

```
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
```

Es gibt eindeutig zu viele positive/hasserfüllte Kommentare. Interessanterweise erhalten wir mit dem Modell, das mit dem ausgewogensten Datensatz trainiert wurde (der Datensatz mit 50.000 Beobachtungen enthält 28.000 nicht hasserfüllte Kommentare und 22.000 hasserfüllte Kommentare), das schlechteste Ergebnis.

#### 4. Ergebnisauswertung

Um die verschiedenen Metriken erhalten zu können, wurden die neuen Kommentare von El Mundo im Juni 2025 vom Autor dieser Arbeit manuell klassifiziert.

Diese Bewertung wurde der Beschriftungsspalte hinzugefügt, und das Ergebnis des Aufrufs der ConfusionMatrix-Funktion von Caret für die drei Modelle lautet:

Konfusionsmatrix für das „600k“-Modell:

```
#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
##
```

## Konfusionsmatrix für das „100k“-Modell:

```
#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
##
```

## Konfusionsmatrix für das „50k“-Modell „besser“:

```
##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
##
```

Ich denke, es geht nicht nur um den Grad der Datenbalance. Ich denke, das Vokabular hat mindestens genauso viel Einfluss wie der Grad der Balance. Je größer das Vokabular ist, mit dem wir ein Modell trainieren, desto bessere Ergebnisse liefert es.

### 3. Klassifizierung neuer Kommentare mit darauf trainierten Modellen

Jetzt werden wir SVM nur mit den im ersten Abschnitt dieses Kapitels erhaltenen Kommentaren trainieren.

Das Training erfolgte zunächst mit einem Datensatz von 2.000 Kommentaren, zu dem später weitere 500 hinzugefügt wurden. Es zeigte sich, dass sich das Ergebnis durch die Hinzufügung dieser zusätzlichen 500 Kommentare deutlich verbesserte.

Mit dem Trainingsdatensatz mit 2000 Kommentaren war das beste erzielte Ergebnis:

```
[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)
```

Beim Training von SVM mit einem Datensatz mit diesen 500 zusätzlichen Kommentaren ist das Ergebnis viel besser:

```
#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
##
```

## Rastersuche durchführen:

```
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."
```

## Das beste Ergebnis, das wir erzielt haben, ist:

```
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 diesem Fall wird das beste Ergebnis mit SVM Typ 1 erzielt.

Es scheint klar, dass die Ergebnisse durch eine Erweiterung des Trainingsatzes noch weiter verbessert werden könnten.

#### 4. Verbesserung des Datensatzes mit den neuen Kommentaren

Ziel ist es zu testen, ob die „Erweiterung“ des gesamten Datensatzes aus den Datensätzen Hatemedia, HuggingFace und Kaggle durch Hinzufügen der in dieser Phase erhaltenen Kommentare zu einem besseren Ergebnis führt als im vorherigen Abschnitt. (Man könnte denken, dass das Ergebnis liegen sollte zwischen den Ergebnissen liegen, die nur mit den Datensätzen aus dem Internet und dem Datensatz aus dem vorherigen Abschnitt, der nur die neuen Kommentare enthält, erzielt wurden).

##### 1. Verarbeitung der Datensätze

Um den Code nicht zu wiederholen, werden nur einige Fragmente angezeigt:

Ausgangsvariablen:

```
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"
```

Wir überprüfen den Anteil der Nachrichten in diesem Datensatz, nachdem wir den Schritt „Downsampling“ durchgeführt haben:

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

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

Nun fügen wir die 2000 Leserkommentare von El Mundo hinzu, die im Juni 2025 eingegangen sind:

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

Wir haben 500 weitere Leserkommentare hinzugefügt:

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

Notiz:

Der Prozess der Klassifizierung von Leserkommentaren ist recht mühsam (und natürlich subjektiv). Ich habe mich für zwei Schritte entschieden, um zu sehen, wie sich die Erweiterung des Trainingsdatensatzes auf das Endergebnis auswirkt.

Jetzt können wir einen vollständigen Datensatz mit allen verfügbaren Datensätzen erstellen:

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

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

Am Ende haben wir einen Datensatz mit 53.000 Kommentaren, ziemlich ausgeglichen (55,37 % Nachrichten ohne Hass, und 44,63 % Hassnachrichten).

Wir lesen auch die Kommentare von Tests, die ebenfalls bereits beschriftet sind:

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

Wir überprüfen den Anteil der Nachrichten in diesen Datensätzen:

- Trainingsdatensatz:

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

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

Durch das Hinzufügen dieser 2.500 neuen Kommentare ändert sich der bereits im Trainingsdatensatz vorhandene Anteil nur geringfügig.

## - Testdatensatz:

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

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

Der Datensatz, den wir mit SVM klassifizieren werden, ist (nach der persönlichen Einschätzung des Autors dieser Arbeit) sehr unausgewogen.

Anschließend werden die Datensätze mit dem üblichen Verfahren aufbereitet: Bereinigen (Entfernen von Verweisen auf andere Benutzer, Emoticons usw.), Erstellung des Korpus, Verarbeitung des Korpus (Entfernung von Großbuchstaben, Zahlen usw.), Tokenisieren, Suchen nach den häufigsten Wörtern usw.

Am Ende des Prozesses erhalten wir:

```
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
```

Jetzt können wir mit dem Training des SVM beginnen.

## 2. SVM-Training und Ergebnisauswertung

Um das Modell zu trainieren, müssen wir die DTMs in Matrizen umwandeln. Angesichts der Größe des Datensatzes und der physikalischen Einschränkungen des Computers, auf dem dieser Prozess ausgeführt wird, werden die DTMs stapelweise verarbeitet, und die Matrizen werden anschließend kombiniert um die vollständige Matrix zu erhalten.

```
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 (Kosten = 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
```

## Vorhersage:

```
# 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
```

## Auswertung des Ergebnisses:

```
# 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
##
## Erkennungsprävalenz: 0,0920
## Ausgeglichene Genauigkeit: 0,7193
##
## „Positive“ Klasse: 1
##
```

Das Ergebnis ist etwas schlechter als das, was durch das Training von SVM nur mit den neuen Kommentaren erreicht wurde (Kappa 0,49).

Wir verwenden die Rastersuche um zu versuchen, ein besseres Modell zu finden als mit den Standardparametern:

```
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
```

Bestes erzieltetes Ergebnis:

```
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
##

```

Das Ergebnis ist erneut etwas schlechter, als wenn man nur die Kommentare aus El Mundo verwendet. In beiden Fällen sind die Ergebnisse jedoch deutlich besser als die, die man erhält wenn man SVM nur mit den im Internet gefundenen Datensätzen trainiert.

## 5. Vergleichstabelle der Ergebnisse und Schlussfolgerungen

Auf der nächsten Seite kann man alle Ergebnisse der Klassifizierung der 500 neuen Kommentare mit allen Modellen (trainiert mit den Internet-Datensätzen – erste Zeile –, nur mit den neuen Kommentaren – zweite Zeile – und mit den „verbesserten“ Datensätzen – dritte Zeile –) finden.

Mit Internet-Datensätzen trainierte Modelle	Superdatensatz "600k" Typ 3 Standardwerte	Superdataset "100k" Typ 3 Standardwerte	Superdataset "50k Beste"	
Konfusionsmatrix	Reference Pred. No Yes No <b>545</b> 24 Yes 333 <b>98</b>	Reference Pred. No Yes No <b>197</b> 7 Yes 681 <b>115</b>	Reference Pred. No Yes No <b>99</b> 2 Yes 779 <b>120</b>	
Genauigkeit	0,643	0,312	0,219	
Kappa	<b>0,2028</b>	<b>0,0494</b>	<b>0,0258</b>	
F1	0,3544	0,2505	0,2351	
Nur mit neuen Kommentaren trainierte Modelle	2000 Kommentare Typ 3 Standardwerte	2000 Kommentare "Beste"	2500 Kommentare Typ 3 Standardwerte	2500 Kommentare "Beste"
Konfusionsmatrix	Reference Pred. No Yes No <b>417</b> 31 Yes 36 <b>16</b>	Reference Pred. No Yes No <b>397</b> 23 Yes 56 <b>24</b>	Reference Pred. No Yes No <b>435</b> 23 Yes 18 <b>24</b>	Reference Pred. No Yes No <b>433</b> 17 Yes 20 <b>30</b>
Genauigkeit	0,8660	0,8420	0,9180	<b>0,9260</b>
Kappa	0,2491	0,2944	0,4995	<b>0,5776</b>
F1	0,3232	0,3780	0,5393	<b>0,6186</b>
Mit „erweiterten Datensätzen“ trainierte Modelle	Superdatensatz "600k" Typ 3 Standardwerte	Superdataset „600k Beste“	Superdataset "50k" Typ 3 Standardwerte	Superdataset „50k Beste“
Konfusionsmatrix	Reference Pred. No Yes <b>444</b> 39 9 <b>8</b>	Reference Pred. No Yes <b>423</b> 24 30 <b>23</b>	Reference Pred. No Yes No <b>430</b> 24 Yes 23 <b>23</b>	Reference Pred. No Yes No <b>427</b> 16 Yes 26 <b>31</b>
Genauigkeit	0,9040	0,8920	0,9060	0,9160
Kappa	<b>0,2106</b>	0,4002	0,4428	0,5498
F1	0,2500	0,4600	0,4946	0,5962

„Beste“: Modell mithilfe der Rastersuche gefunden

„Erweiterter Datensatz“: Datensatz, der mit den 3 im Internet gefundenen Datensätzen sowie 2.500 beschrifteten Kommentaren von El Mundo im Juni 2025 erstellt wurde

## Schlussfolgerungen

### - Grad der Ausgewogenheit gegenüber der Mindestfrequenz

Auf den ersten Blick mag es scheinen, dass der Grad der Ausbalancierung der wichtigste Faktor für die Qualität der Ergebnisse ist. Zwei Ergebnisse haben mich jedoch zu einer anderen Schlussfolgerung geführt:

- Beim Klassifizieren neuer Kommentare mit Modellen, die auf dem Superdatensatz trainiert wurden, der mit den drei Datensätzen von Hatemedia, HuggingFace und Kaggle erstellt wurde, ist das Ergebnis mit dem größten Datensatz besser, der am unausgewogensten ist (aber derjenige, der mit einem breiteren Vokabular erstellt wurde).

- Beim Testen mit unterschiedlichen Mindestfrequenzen (und damit Begrenzung der Spalten/Funktionen der DTMs) mit demselben Datensatz gilt unabhängig vom Grad der Ausgewogenheit: Je niedriger die Mindestfrequenz, desto besser die erzielten Ergebnisse.

Daher scheint es klar, dass die Mindesthäufigkeit von Begriffen – die die Größe des Wortschatzes bestimmt – ein ebenso grundlegender Faktor ist wie der Grad der Ausgewogenheit. (Denn auch bei gleicher Mindesthäufigkeit sind die Ergebnisse bei ausgewogeneren Datensätzen besser.)

### - Grad der „Ähnlichkeit“ der Vokabeln

Es war eine ziemlich große (und negative) Überraschung, mit den Modellen, die mit den im Internet gefundenen Datensätzen (mit der Vereinigung der drei Datensätze) trainiert wurden, so schlechte Ergebnisse zu erzielen. Es wurden zahlreiche Tests durchgeführt, aber alle Ergebnisse waren sehr schlecht. Das war, um ehrlich zu sein, ziemlich unerwartet.

Bei so kleinen Datensätzen wie denen, die nur Kommentare aus El Mundo enthalten und im Juni dieses Jahres abgerufen wurden, sind die Ergebnisse jedoch sehr gut. Es ist auch ersichtlich, dass die Ergebnisse durch die Erweiterung dieser Datensätze mit den neuen Kommentaren „normal“ werden.

### - Modellkonfiguration

In allen Fällen zeigt sich, dass die Konfiguration der SVM-Parameter mit anderen als den Standardwerten deutlich bessere Ergebnisse erzielt. Da es sich um stark unausgewogene Datensätze handelt, ist die richtige Gewichtung unerlässlich.

## Mögliche Verbesserungen

Angesichts der Auswirkungen, die das Hinzufügen neuer Nachrichten auf die im Internet gefundenen Datensätze hat, und des erheblichen Unterschieds zwischen dem Datensatz mit 2.000 neuen Kommentaren und dem mit 2.500 Kommentaren scheint es klar, dass eine Erweiterung des Trainingssatzes sehr positive Auswirkungen haben wird.

Es kann auch interessant sein, die Art der Nachrichten stärker zu variieren.

## 5. ANHANG 1 – LIBLINEAR-Bibliothek, Verlust- und Regularisierungsfunktionen

Der Parameter „Typ“ in LIBLINEAR kann zehn Typen (verallgemeinerter) linearer Modelle erzeugen, die verschiedene Verlustfunktionen und Regularisierungsschemata kombinieren. Die Regularisierung kann L1 oder L2 sein, und die Verluste können der reguläre L2-Verlust für SVM (Scharnierverlust), der L1-Verlust für SVM oder der Log-Verlust für die logistische Regression sein. Der Standardwert für „Typ“ ist 0. Details siehe unten. Gültige Optionen sind:

Für die Mehrklassenklassifizierung:

- **0** – Logistische Regression mit L2 (primal) Regularisierung
- **1** – Support-Vektor-Klassifizierung mit L2-Verlust und L2-(Dual-)Regularisierung
- **2** – Support-Vektor-Klassifizierung mit L2-Verlust und L2-(primärer) Regularisierung
- **3** – Support-Vektor-Klassifizierung mit L1-Verlust und L2-(Dual-)Regularisierung
- **4** – Support Vector Klassifizierung nach Crammer und Singer
- **5** – Support-Vektor-Klassifizierung mit L2-Verlust und L1-Regularisierung
- **6** – Logistische Regression mit L1-Regularisierung
- **7** – Logistische Regression mit L2 (dualer) Regularisierung

Für die Regression:

- **11** – Support-Vektor-Regression mit L2-Verlust und L2-(primärer) Regularisierung
- **12** – Support Vector Regression mit L2-Verlust und L2-(Dual-)Regularisierung“

Notiz:

### - L1- und L2-Verlustfunktionen:

**Die L1-** Verlustfunktion, auch als **absoluter Verlust bezeichnet** , ist definiert als die Summe der absoluten Differenzen zwischen den Vorhersagen und den tatsächlichen Werten. Mathematisch lässt sie sich wie folgt ausdrücken:

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

Wo:

- $y_i$  ist der tatsächliche Wert.
- $\hat{y}_i$  ist die Modellvorhersage.
- $n$  ist die Anzahl der Proben.

Die L1-Verlustfunktion ist robust gegenüber Ausreißern, da sie große und kleine Fehler gleichermaßen bestraft. Das bedeutet, dass große Fehler keinen unverhältnismäßigen Einfluss auf den Gesamtverlust haben.

**Die L2-** Verlustfunktion, auch als **quadratischer Verlust oder mittlerer quadratischer Fehler (MSE)-Verlust bekannt** , wird als Summe der Quadrate der Differenzen zwischen den Vorhersagen und den tatsächlichen Werten definiert.

Mathematisch kann es wie folgt ausgedrückt werden:

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

Wo:

- $y_i$  ist der tatsächliche Wert.
- $\hat{y}_i$  ist die Modellvorhersage.
- $n$  ist die Anzahl der Proben.

### - L1- und L2-Regularisierungen:

Quelle: [https://en.wikipedia.org/wiki/Lasso\\_\(statistics\)](https://en.wikipedia.org/wiki/Lasso_(statistics))

**Die L1-Regularisierung** wird auch LASSO-Regularisierung („Least Absolute Shrinkage and Selection Operator“) genannt:

- Addiert die Summe der Absolutwerte der Koeffizienten zu den Kosten der Verlustfunktion.
- Es werden tendenziell spärlichere Modelle erstellt, bei denen einige Koeffizienten genau Null sein können, was zu einer Merkmalsauswahl führen kann.
- Dies ist nützlich, wenn eine besser interpretierbare Lösung gewünscht wird und Überanpassung reduziert werden soll.

**L2-Regularisierung** , auch bekannt als Ridge-Regularisierung:

- Addieren Sie die Summe der Quadrate der Koeffizienten zu den Kosten der Verlustfunktion.
- Es neigt dazu, den Fehler auf alle Koeffizienten zu verteilen, was zu kleineren Koeffizienten führen kann, aber nicht genau Null.
- Dies ist nützlich, wenn Sie Überanpassung reduzieren und alle Koeffizienten zum Modell beitragen möchten.

Kurz gesagt handelt es sich bei der L1- und L2-Regularisierung um Techniken, die verwendet werden, um Overfitting in Machine-Learning-Modellen zu verhindern. Die Wahl zwischen L1 und L2 hängt von den Merkmalen des Problems und den Zielen des Modells ab.

## 6. ANHANG 2 – R-Quellcode gemeinsamer Funktionen.

### hate\_speech\_common.R

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

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

  print("Loading libraries:")

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

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

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

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

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

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

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

  print("All libraries loaded.")
}

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

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

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

```

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

result
}

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

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

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

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

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

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

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

  posts_corpus_clean
}

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

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

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

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

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

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

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

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

```



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```

Notiz:

Später wurden Prüfungen für Eingabeparameter und Unit-Tests (mithilfe der *Testthat* -Bibliothek) hinzugefügt, um diese Typprüfungen zu validieren.

## 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. ANHANG 3 – Code-Repository auf GitHub

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

### - Phase 1: Sammlung beschrifteter Datensätze

[https://github.com/fcamadi/masterML/blob/main/TFM/hate\\_speech\\_01.Rmd](https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_01.Rmd)

### - Phase 2 - Anwendung von 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_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_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_02_kaggle.Rmd)

[https://github.com/fcamadi/masterML/blob/main/TFM/hate\\_speech\\_common.R](https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_common.R)

### - Phase 3 - Klassifizierung neuer unbeschrifteter Kommentare

[https://github.com/fcamadi/masterML/blob/main/TFM/Web\\_scraping\\_RSelenium\\_periodicos\\_online.Rmd](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_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.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_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_not_labelled_ALL_600k.Rmd)

[https://github.com/fcamadi/masterML/blob/main/TFM/hate\\_speech\\_03\\_common.R](https://github.com/fcamadi/masterML/blob/main/TFM/hate_speech_03_common.R)

## 8. ANHANG 4 - Bibliographie

Allgemein:

- „Machine Learning with R“, 4. Auflage – 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/>