Memoria de la Práctica Grupo 1

2024-11-29

Grupo 1

- Oliver Arnaldo Anderson Llorens - oliver.anderson@alumnos.upm.es
- Jose Ma - jose.ma@alumnos.upm.es
- Joaquín Daniel Negrete Saab - joaquin.negrete@alumnos.upm.es
- Anton Pogromskyy - anton.pogromskyy@alumnos.upm.es
- María del Mar Sierra Gil - mdm.sierra.gil@alumnos.upm.es
- David Gómez Martín - david.gomezm@alumnos.upm.es

Contents

1 Lista de tareas de cada uno de los integrantes y el porcentaje de contribución:

- Código Parte 1: María del Mar (90%), Anton Pogromskyy (5%) y Oliver Anderson (5%)
- Código Parte 2: Jose Ma (50%) y Joaquín Negrete (50%)
- Código Parte 3: Anton Pogromskyy (50%) y Oliver Anderson (50%)
- Pruebas Parte 2: Jose Ma (50%) y Joaquín Negrete (50%)
- Pruebas Parte 3: Anton Pogromskyy (100%)
- Memoria Pre: Anton Pogromskyy (90%) y Joaquín Negrete (10%)
- Memoria Parte 1: María del Mar (80%), Jose Ma (15%) y Anton Pogromskyy (5%)
- Memoria Parte 2: Jose Ma (50%) y Joaquín Negrete (50%)
- Memoria Parte 3: Anton Pogromskyy (50%) y Oliver Anderson (50%)

2 Requisitos

- Instalar la librería `testthat`: `install.packages("testthat")`
- Instalar la librería `R.utils`: `install.packages("R.utils")`
- Tener instalado `tar` para extraer el fichero tar
- Instalar `udpipe`: `install.packages("udpipe")`
- Instalar `spacyr` con el modelo `es_core_news_sm`
- Para ejecutar el código se debe estar trabajando en el directorio principal

3 Para ejecutar el proyecto

Primero, descargar el `.zip` de este enlace: https://drive.google.com/file/d/1FPnn2SCE76OrdVw22kuTmMZvCFvv6XeS/view?usp=sharing y descomprimir el *contenido* en el directorio `datos` del directorio principal proyecto. Debería quedar algo así:

```
.
|-- datos
|   |-- data_frame_summary_udpipe.rds
|   |-- ...
|-- fuentes
|   |-- train_analysis.R
|   |-- train_qcorpus.R
|   |-- train_stats.R
|-- ...
|-- resultados
```

Luego, se tienen que ejecutar los archivos `.R` del directorio `./fuentes` en el siguiente orden: 1. `train_qcorpus.R` 2. `train_stats.R` 3. `train_analysis.R`

4 Parte 1

4.1 Objetivo

1. Leer un archivo JSON lineal con datos textuales en español.
2. Crear un corpus utilizando la librería `quanteda`.
3. Enriquecer el corpus con variables documentales (docvars).
4. Guardar el corpus en un archivo `.rds`.

4.2 Diseño y ejecución:

En este primer programa se pide leer el fichero "spanish_train.json" y crear un corpus quanteda.

El primer paso fue descargar los datos de la página web de HuggingFace, para posteriormente descomprimir la carpeta obtenida y así poder usar el fichero "spanish_train.json".

En el código se importarán las librerías jsonlite y quanteda para la realización de esta primera parte. La función stream_in() del paquete jsonlile nos permite leer y cargar los datos del fichero previamente descargado en la variable data.

Creamos un corpus y asignamos docvars y el contenido del docvar usando el contenido de data (lectura del fichero jsonl)

Por último lo guardamos en un fichero para su posterior uso: fichero spanish_train.qcorpus.rds

Comprobamos que el corpus está correctamente creado:

```
head_corpus(corps, n = 5, preview_length = 50)
```

```
text1 :
" De no recibir más dinero, las raciones de la ONU e ..."

text2 :
" Manifestantes hindúes gritan consignas contra el g ..."

text3 :
" Así lo vemos en esta animación que muestra cómo el ..."

text4 :
" McDonald's restaba importancia a las protestas por ..."

text5 :
" ¿De dónde vienen y qué revelan las espantosas leye ..."

Package version: 4.1.0
Unicode version: 15.1
ICU version: 75.1

Parallel computing: disabled

See https://quanteda.io for tutorials and examples.
```

Las variables del corpus son:
id, url, title, summary

Variable: id
Ejemplos:
140930_ultnot_siria_onu_comida_ch, 130809_ultnot_protesta_cachemira_protesa_aa, media-37220890

Variable: url
Ejemplos:
https://www.bbc.com/mundo/ultimas_noticias/2014/09/140930_ultnot_siria_onu_comida_ch, https://www.bbc.com/mundo/ultimas_noticias/2013/08/130809_ultnot_protesta_cachemira_protesa_aa, https://www.bbc.com/mundo/media-37220890

Variable: title
Ejemplos:
ONU dice que peligran raciones de comida para Siria, Policía india dispersa protesta con gas lacrimógeno en Cachemira, El movimiento de los ríos en el Amazonas

Variable: summary
Ejemplos:
La directora de ayuda humanitaria de Naciones Unidas, Valerie Amos, advirtió que de no invertir más dinero, el Programa Mundial de Alimentos tendrá que detener sus operaciones en Siria en dos meses., La policía en la región de Cachemira administrada por India, lanzó gas lacrimógeno y disparó balas de goma para dispersar una protesa contra supuestas violaciones de los derechos humanos llevadas a cabo por las fuerzas del gobierno., La naturaleza es un hueso duro de roer.

5 Parte 2

5.1 Introducción

Este documento resume el proceso de análisis de un corpus en español mediante el uso de herramientas como `spacy`, `quanteda` y `udpipe`. El objetivo principal fue analizar las estructuras lingüísticas presentes en los títulos y resúmenes del corpus, generando información estadística sobre el número de tokens y su distribución.

5.2 Importación de Librerías y Datos

Primero, cargamos las librerías necesarias (`spacyr`, `quanteda` y `udpipe`) y los datos del corpus, que incluyen los campos de título y resumen. También cargamos un modelo lingüístico para el idioma español (`spanish-ancora`) compatible con `udpipe`.

5.3 Procesamiento con SpaCy

Utilizamos `spacy` para analizar los campos de título y resumen:

1. Parseamos los textos para extraer información lingüística, incluyendo el lema y la categoría gramatical de cada palabra.
2. Filtramos los resultados para eliminar signos de puntuación y enfocarnos en los lemas relevantes.

El resultado fue un subconjunto limpio de datos con los campos `doc_id` (identificador del documento) y `lemma` (forma lematizada de las palabras).

5.4 Creación de Listas de Tokens

Desarrollamos una función personalizada que agrupa los tokens por documento. Esto nos permitió contar el número total de palabras por documento tanto para los títulos como para los resúmenes. Al principio hicimos una primera función y parecía que no funcionaba correctamente, puesto que no cogía bien las palabras del último documento y tampoco teníamos en cuenta si le pasamos un dataframe sin valores. Aquí le muestro la primera función:

```
lista_agrupada <- function(dataframe_text_lemma) {
  titulos_unicos <- unique(dataframe_text_lemma$doc_id)
  lista_titulos <- list()
  pos_titulo_unico <- 1
  palabras <- 0
  for (i in 1:nrow(dataframe_text_lemma)) {
    if (dataframe_text_lemma[i, 1] == titulos_unicos[pos_titulo_unico]) {
      palabras <- palabras + 1
    } else if (dataframe_text_lemma[i, 1] != titulos_unicos[pos_titulo_unico]) {
      lista_titulos[titulos_unicos[pos_titulo_unico]] <- palabras
      pos_titulo_unico <- pos_titulo_unico + 1
      palabras <- 1
    }
  }
  return(lista_titulos)
}
```

Luego simplemente le añadimos la última fila y ya podrá tener en cuenta el último documento.

Estas listas se usaron posteriormente para generar estadísticas descriptivas.

5.5 Visualización de Resultados

Creamos histogramas para visualizar la distribución del número de tokens en los títulos y los resúmenes:

- Los histogramas de los títulos mostraron la mayoría de los documentos con menos de 50 tokens.
- Los histogramas de los resúmenes mostraron una mayor diversidad, con la mayoría de los documentos teniendo menos de 180 tokens.

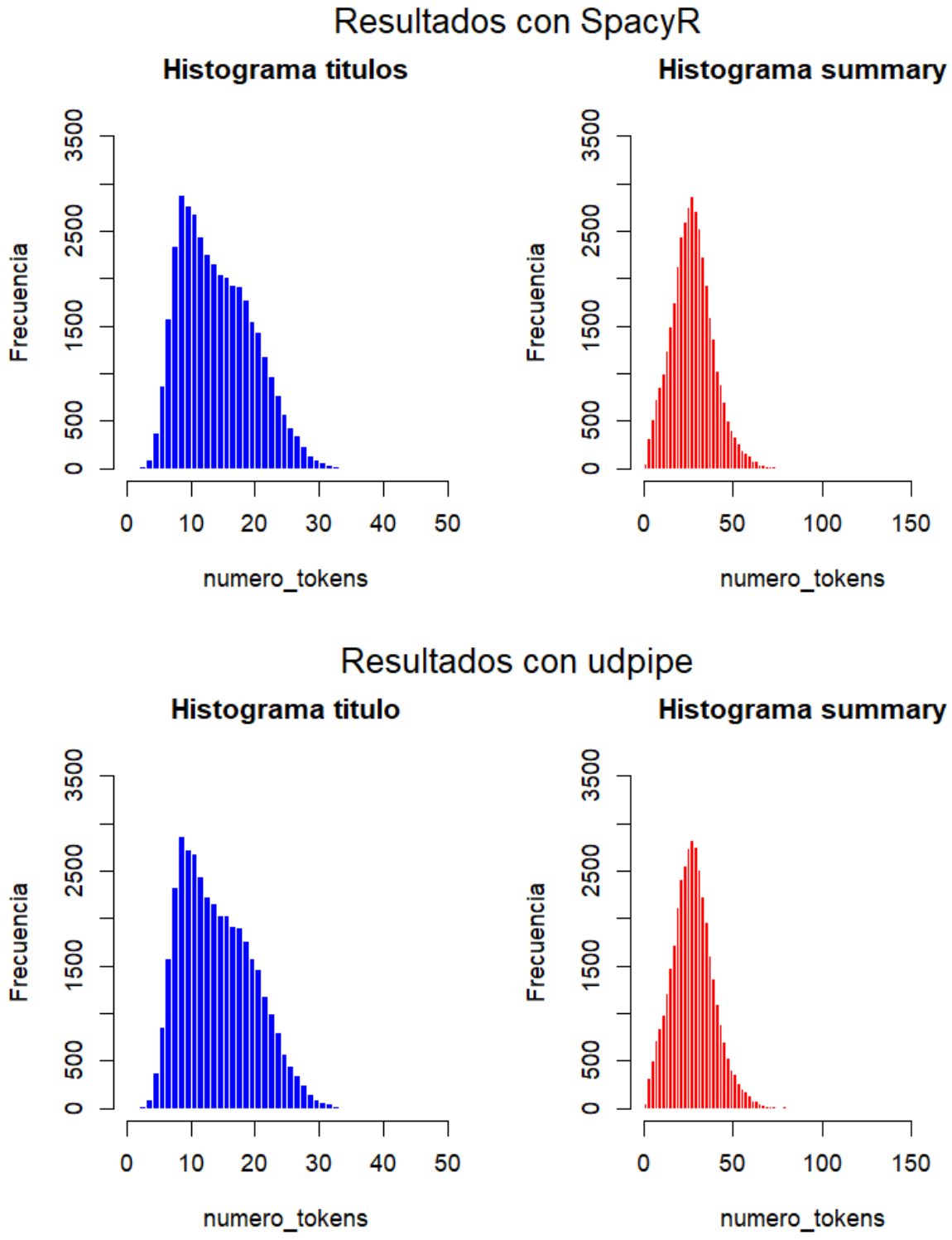
5.6 Procesamiento con UDPipe

Para complementar el análisis, utilizamos `udpipe` para realizar un análisis gramatical detallado. Este proceso incluyó:

1. Anotar los textos del corpus con el modelo `spanish-ancora`.
2. Guardar los resultados en ficheros para optimizar el tiempo de procesamiento.
3. Limpiar los datos eliminando puntuación y manteniendo solo los lemas relevantes.

5.7 Comparación de Resultados

Se utilizó la misma función para agrupar y contar los tokens en los datos procesados con `udpipe`. Finalmente, generamos histogramas para comparar la distribución de tokens en los títulos y resúmenes utilizando ambas herramientas.



Como podemos ver, ambos histogramas son casi idénticos, se pueden ver pequeñas diferencias como los valores máximos de cada uno de los tipos: el summary con spacy tiene 172 mientras que el de udpipe tiene 171; el titulo con spacy tiene 37 mientras que el de udpipe tiene 39. Estas diferencias son muy pequeñas y casi no se nota en las gráficas por lo que confirmamos que los resultados obtenidos con `udpipe` son consistentes con los obtenidos previamente con `spacy`.

Se puede ver que el número de palabras que más abundan en títulos está entre 5 y 20, mientras que en el summary están entre 10 y 40, pudiendo ver además que hay una asimetría a la derecha puesto que la mayoría de los datos se encuentran a la izquierda.

5.8 Prueba

Para llevar a cabo el testing para probar la función y la limpieza de los datos se puede ejecutar el fichero test_2.R ubicado en el directorio pruebas.

Esto es el resultado comentado:

Test 1: Importación del corpus Test passed

Test2: Prueba de la función lista_agrupada. La lista se compone de los dos docs y el resultado debe ser 2 y 3 $doc1 [1] 2

$doc2 [1] 3

[1] 2 [1] 3

Test passed

Test3: Prueba de la función lista_agrupada. el dataframe tiene sólo un elemento y se comprueba si da el mismo elemento. El nulo es porque se accede al segundo elemento de la lista el cual no existe $doc1 [1] 3

[1] 3

NULL Test passed

Test4: Prueba de la función lista_agrupada con un elemento nulo.

list() Test passed

Test5: Probamos si se guardan bien los resultados en un fichero Test passed

Test5: Probamos cómo elimina signos de puntuación del dataframe sample_data <- data.frame(doc_id = c("doc1", "doc1", "doc2"), lemma = c("palabra1", "palabra2", NA), pos = c("NOUN", "PUNCT", "PUNCT")) Además probamos el acceso al lemma limpio que debe dar "palabra1":

[1] 1 [1] "palabra1" Test passed

Test6: Probamos el funcionamiento del histograma y el resultado es el esperado de los breaks y count. [1] 0 1 2 3 4 5 [1] 1 2 3 0 0 Test passed

5.9 Retos y Soluciones

El principal inconveniente encontrado fue el tiempo de procesamiento de `udpipe`. Anotar los títulos y resúmenes del corpus tomó aproximadamente 30 minutos. Para mitigar este problema, guardamos los resultados procesados en ficheros RDS. Esto nos permitió cargar los datos rápidamente en ejecuciones futuras, reduciendo significativamente el tiempo requerido.

5.10 Conclusiones

Este análisis demostró la utilidad de herramientas lingüísticas avanzadas para procesar y analizar textos en español. Los histogramas generados permitieron visualizar la estructura del corpus y entender mejor la longitud promedio de los títulos y resúmenes.

6 Parte 3

6.1 Objetivos de la Tercera Parte

En esta parte del proyecto, se nos pide implementar un programa llamado `train_analysis.R` que realice el análisis del corpus obtenido en la primera parte, almacenado en el archivo `spanish_train.qcorpus.rds`. El programa debe calcular, para cada documento, lo siguiente:

1. **Frecuencia de verbos del título**: Contar cuántos verbos, convertidos a su forma en infinitivo, presentes en el campo `title` del documento también aparecen en el texto completo correspondiente.
2. **Frecuencia de verbos del resumen**: Contar cuántos verbos, convertidos a su forma en infinitivo, presentes en el campo `summary` del documento también aparecen en el texto completo correspondiente.

Estos resultados se presentan de manera visual mediante dos histogramas: - Un histograma para los verbos del `title`. - Otro histograma para los verbos del `summary`.

En los histogramas obtenidos, se muestra la distribución de las frecuencias de coincidencia y el número de documentos que corresponden a cada frecuencia. Esto permite visualizar de forma clara cómo se relacionan los verbos presentes en los `title` y `summary` con el contenido completo de los textos.

6.2 Importación de Código para Tests

Para mantener el documento más organizado y legible, importamos el código de tests desde otro archivo. Esto nos permite ejecutar y mostrar los resultados de las pruebas específicas de las funciones utilizadas en el programa principal sin añadir bloques de código extensos.

La importación se realiza con la función `source()` de R, que permite cargar y ejecutar el contenido de un archivo externo. En este caso, importamos el archivo `test_train_analysis.R`, donde se encuentran definidas las pruebas.

6.3 Uso de Archivos RDS para Optimizar el Procesamiento

Dado que el parseo de textos con la función `spacy_parse()` puede ser un proceso lento y costoso, decidimos implementar una estructura `if-else` para gestionar el almacenamiento y la carga de los datos procesados. La idea principal es evitar realizar el parseo en cada ejecución, almacenando los resultados en archivos `.RDS`, que son compactos y rápidos de cargar.

6.3.1 Estructura General

La estructura que utilizamos funciona de la siguiente manera:

- **Si el archivo RDS ya existe**:
  - Cargamos directamente los datos procesados con la función `readRDS()`.
  - Esto reduce significativamente el tiempo de ejecución en futuras sesiones.
- **Si el archivo RDS no existe**:
  - Parseamos los datos correspondientes con `spacy_parse()` y los guardamos en un archivo `.RDS` utilizando `saveRDS()`.
  - Esto asegura que el procesamiento solo se realiza la primera vez.

6.3.2 Función `spacy_parse()`

1. **Tokenización**:
   - Divide el texto en **tokens** (en nuestro caso las palabras de los textos).
2. **Etiquetado gramatical (POS tagging)**:
   - Asocia cada token con su categoría gramatical (por ejemplo, `VERB`, `NOUN`, `ADJ`).
3. **Lematización**:
   - Convierte las palabras a su forma raíz (por ejemplo, "corriendo" → "correr").

El propósito principal de usar `spacy_parse()` en este proyecto es que nos permita trabajar específicamente con los **verbos** en infinitivo de los textos. Esto incluye:

- Identificar todos los verbos en un texto (`pos == "VERB"`).
- Obtener los verbos en su forma **lema** (`lemma`), para evitar inconsistencias debido a conjugaciones.

6.4 Resultado de `spacy_parse()`

Tras realizar el `spacy_parse()` sobre las palabras de los campos `title`, `summary` y los textos completos, obtenemos una tabla estructurada que contiene información clave sobre los tokens del texto. En nuestro caso, nos quedamos con las columnas `lemma` (forma en infinitivo de los verbos) y `doc_id` (identificador único de cada documento).

El uso del `doc_id` es bastante importante, ya que nos permite asociar cada verbo con su documento correspondiente.

6.4.1 Ejemplo de los Resultados

Verbos del campo `title`:

```
   doc_id     lemma
2   text1     decir
4   text1  peligrir
28  text4 reconocer
40  text5  inspirar
57  text6     parar
70  text7     tener
```

Verbos del campo `summary`:

```
   doc_id    lemma
13  text1 advertir
17  text1 invertir
26  text1    tener
28  text1  detener
48  text2   lanzar
52  text2 disparar
```

Verbos de los textos completos:

```
   doc_id    lemma
3   text1  recibir
15  text1 terminar
32  text1    decir
47  text1 recortar
50  text1   llegar
61  text1    hacer
```

6.5 Función `verbs_create_list`

La función `verbs_create_list` organiza los verbos extraídos del corpus en una estructura más manejable, agrupándolos por documento. Su objetivo principal es asociar cada verbo a su documento correspondiente, utilizando el identificador único de cada documento (`doc_id`).

La función toma los verbos extraídos de un campo del corpus, como `title` o `summary`, y los organiza en una lista, donde cada elemento corresponde a un documento específico. Si un documento no contiene verbos en un campo determinado, se incluye en la lista con un vector vacío, asegurando que todos los documentos estén representados de manera consistente.

El resultado de esta función es una lista con nombres que corresponden a los `doc_id` del corpus y cuyo contenido son los verbos en su forma lematizada. Esta estructura facilita el acceso y análisis de los datos, permitiéndonos comparar los verbos presentes en los títulos o resúmenes con los del texto completo de cada documento.

Resultados tras aplicar la función sobre los dataframes obtenidos anteriormente:

Aplicando para `verbs_titles`:

```
$text1
[1] "decir"    "peligrir"

$text2
character(0)

$text3
character(0)

$text4
[1] "reconocer"

$text5
[1] "inspirar"

$text6
[1] "parar"
```

Para `verbs_summary`:

```
$text1
[1] "advertir" "invertir" "tener"    "detener"

$text2
```

```
[1] "lanzar"    "disparar"  "dispersar"

$text3
character(0)
```

Lo mismo con `verbs_text`:

```
$text1
[1] "recibir"  "terminar" "decir"    "recortar" "llegar"   "hacer"    "juntar"
[8] "proteger"

$text2
 [1] "hindúes"   "gritar"    "tener"     "decir"     "producir"  "volver"
 [7] "empezar"   "lanzar él" "resultar"  "arrestar"  "evitar"    "liderarar"

$text3
[1] "ver"     "mostrar" "buscar"  "perder"  "ganar"   "crear"
```

6.6 Eliminación de Verbos Duplicados

Como indica el enunciado, necesitamos trabajar con los **verbos únicos** de los campos `title` y `summary`, ya que no queremos contar verbos repetidos dentro de un mismo documento. Para lograr esto, definimos la función `make_unique()`, que elimina duplicados en cada elemento de una lista. Su objetivo es garantizar que cada sublista contenga solo valores **únicos**.

La función recorre cada elemento de la lista (correspondiente a un documento) y, si este no está vacío, utiliza la función `unique()` para filtrar los valores duplicados. De esta forma, el resultado es una lista en la que cada sublista contiene únicamente los verbos únicos asociados a su documento.

6.6.1 Ejemplo de Uso

Si observamos el contenido original de `list_verbs_summary`, en el documento **text__5** aparece el verbo `tener` dos veces.

```
$text5
[1] "infectar" "tener"    "tener"    "ver"
```

Después de aplicar `make_unique()`, el resultado muestra este verbo solo una vez.

```
$text5
[1] "infectar" "tener"    "ver"
```

6.7 Cálculo de Frecuencias de Verbos

Llegamos al núcleo del proyecto: **contar cuántas veces los verbos presentes en los campos `title` y `summary` aparecen en el texto completo de cada documento**. Para ello, definimos la función `freq_verbs`, que calcula la frecuencia total de los verbos en el texto correspondiente.

6.7.1 Explicación de la Función `freq_verbs`

La función `freq_verbs` toma tres argumentos:

1. **`text_list`**: Lista de verbos en los textos completos.
2. **`list_comp`**: Lista de verbos a comparar (pueden ser de `title` o `summary`).
3. **`corpus_ids`**: Identificadores únicos de los documentos.

El objetivo de esta función es recorrer los textos y contar cuántas veces los verbos presentes en `list_comp` aparecen en el texto completo correspondiente. Para ello:

- Se inicializa un vector vacío llamado `freq`, cuya longitud corresponde al número de documentos, y se asignan los `corpus_ids` como nombres.
- Para cada documento, se recorren los verbos de `list_comp` y se cuenta cuántas veces aparecen en el texto completo (`text_list`) utilizando `sum()`.
- Finalmente, el vector `freq` almacena la frecuencia total de los verbos por documento y se devuelve como resultado.

Aplicamos esta función y obtenemos las frecuencias de los verbos de cada `title` en su texto y de cada `summary` en su texto.

Veamos las frecuencias de los `titles` 50 - 70:

```
freq_verbs_titles[50:70]
```

```
text50 text51 text52 text53 text54 text55 text56 text57 text58 text59 text60
    14      0      0      0      4      0      1      6      2     13      4
text61 text62 text63 text64 text65 text66 text67 text68 text69 text70
     0      0      1      0      0      0      0      0      0      0
```

Veamos las frecuencias de los `summary` 50 - 70:

```
freq_verbs_summary[50:70]
```

```
text50 text51 text52 text53 text54 text55 text56 text57 text58 text59 text60
     0      0      0      0      8      0      0      5      2     19     11
text61 text62 text63 text64 text65 text66 text67 text68 text69 text70
     9      6      6      1     10      2      4      0     12      0
```

6.8 Testeos de la Función `freq_verbs`

Para confirmar que los resultados de la función `freq_verbs` son correctos, definimos una nueva función llamada `test_freq_verbs`. Esta función evalúa un subconjunto de documentos y verifica:

- La frecuencia de verbos encontrados en los títulos (`title`) y resúmenes (`summary`) comparados con el texto completo.
- Los verbos coincidentes entre los diferentes campos y las frecuencias correspondientes.

El objetivo de esta prueba es asegurar que los verbos se están contando correctamente y que los resultados coinciden con las expectativas.

Hacemos un test de los textos 56 - 58:

```
test_freq_verbs(list_verbs_text, list_verbs_title_unique, list_verbs_summary_unique, corpus_ids, TRUE)
```

—— Testing freq_verbs Function ——

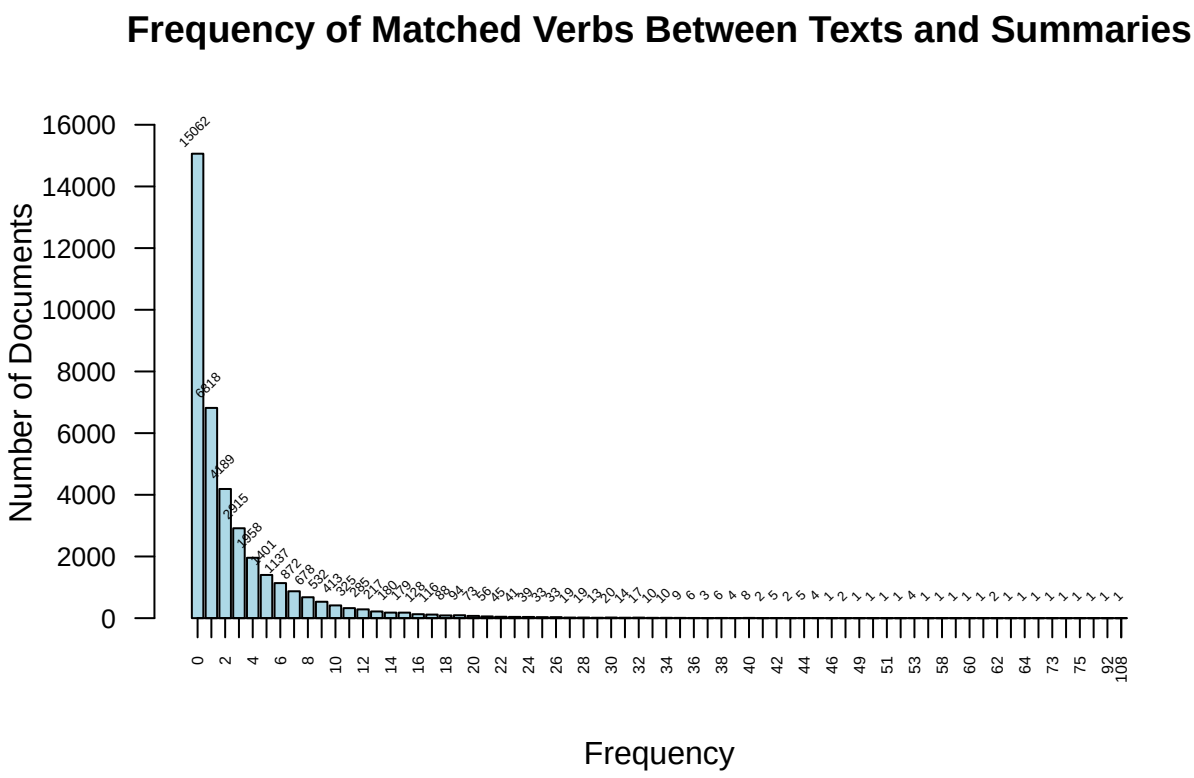
Document ID: text63
Text Verbs: convertir empezar hacer parecer apostar tener representar jugar conocer inclinar lograr incorporar rinda esperar interesar hablar dar descartar hacer crecer aumentar seguir consumir poner crear existir hacer mover manipular conseguir tener divulgar ocurrir apasionar identificar él afianzar ocurrir andar mantener ver jugar trasmitir contratar hablar tener dejar jugar ocurrir entrar trasmitar evitar trasmitir participar hablar quedar hacer él felicitar saber sacar saber anotar impedir conocer alcanzar decir – decir yo decir decir existir mantener hacer jugar jugar desempeñar pagar pagar creer desconocer realizar – cercenar sentir practicar sentir saber pertenecer gustar

Title Verbs: gustar
Matched Title Verbs: gustar
Matched Count (Titles): 1

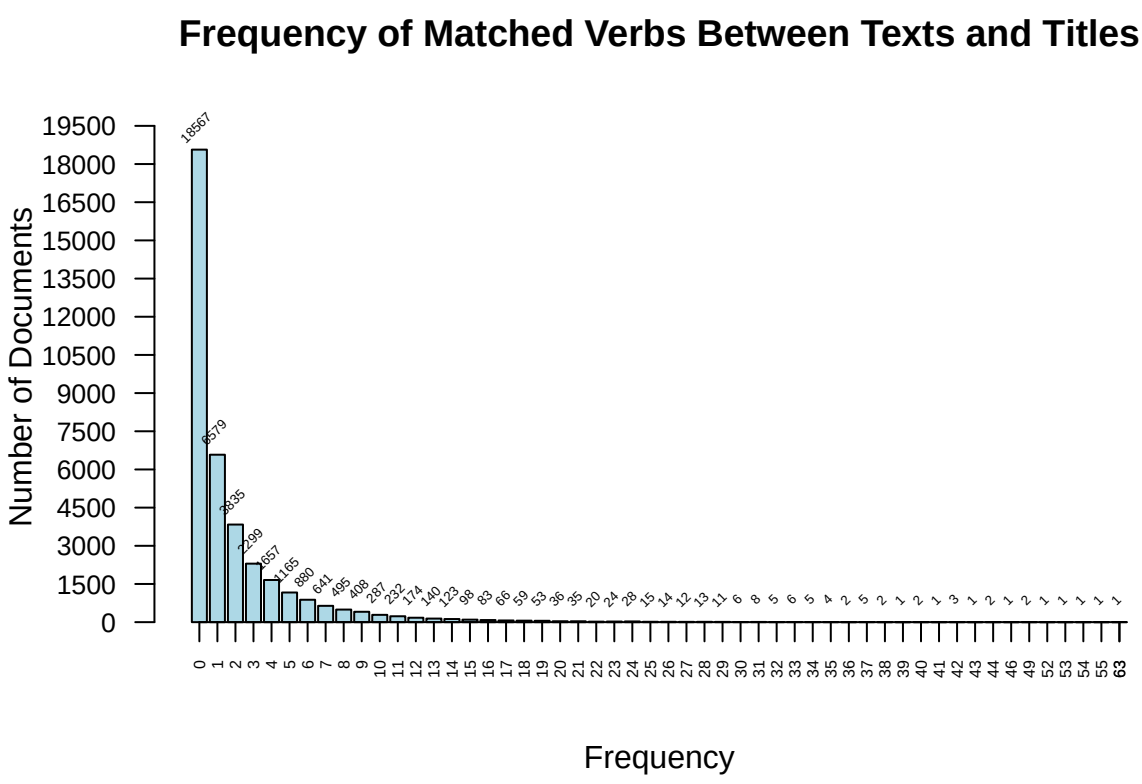
Summary Verbs: hacer colmar soler trasmitir superar
Matched Verbs (Summaries): hacer, trasmitir
Matched Count (Summaries): 6

6.9 Visualización de Frecuencias de Verbos

Para analizar la cantidad de documentos que tienen cierta frecuencia de coincidencia entre los verbos de los resúmenes (`summary`) y los textos completos, generamos un **histograma**. Este histograma nos permite observar cómo están distribuidas las frecuencias de coincidencia entre los documentos.



Histograma para ver las frecuencias de los verbos del campo **títulos**:



Como se puede observar hay gran cantidad de documentos que tienen frecuencia 0. Al principio esto nos pareció muy raro, pero tras realizar numerosas pruebas y verificar nuestros cálculos, hemos llegado a la conclusión de que estos son correctos. Nuestra hipótesis es que el problema radica en las limitaciones del modelo `es_core_news_sm` utilizado por `spacy_parse()`. Este modelo, aunque eficiente y rápido, no identifica correctamente ciertos verbos. Por ejemplo, no reconoce el verbo "ser", lo cual es especialmente significativo dada su alta frecuencia en el idioma español. Además, tampoco maneja bien los verbos pronominales, lo que afecta aún más la precisión de los resultados.

Creemos que si hubieramos usando un modelo de lenguaje español más grande que `es_core_news_sm` los resultados habrían sido más representativos y precisos.

6.10 Testeo para Frecuencias Específicas

Una vez calculadas las frecuencias de coincidencias de verbos entre los resúmenes (`summary`) y los textos completos, podemos realizar testeos para analizar documentos que tienen una frecuencia específica. Esto nos permite observar en detalle los verbos coincidentes en esos casos particulares y asegurarnos de que el cálculo se realiza correctamente.

6.10.1 Función `test_specific_frequency_summary`

La función `test_specific_frequency_summary` busca documentos que tienen una frecuencia específica de coincidencias en los resúmenes. Para cada documento encontrado: 1. Muestra los verbos del texto completo y del resumen. 2. Indica los verbos coincidentes y cuántas veces se encontraron en el texto.

Si no se encuentran documentos con esa frecuencia, la función informa que no hay coincidencias.

Hagamos un test, por ejemplo, con frecuencia **2**, solo debe aparecer 1 documento con la suma de frecuencias de cada verbo encontrado equivalente a 2. Como hay `4189` diferentes documentos con esta frecuencia, solo se va enseñar el primero encontrado. Para probar diferentes frecuencias o ver todos los documentos con una frecuencia determinada se puede ejecutar el fichero `test_train_analysis.R`.

=== Testing for Frequency: 2 ===

Document ID: text16
Text Verbs: firmar, aprobar, asegurar, tratar, lograr, englobar, enviar, apoyar, escribir, interesar, figurar, consistir, formar, consolidar, crear, respetar, señalar, indicar, contener, incluir, considerar, tratar, prever, eliminar, ahorrar, abrir, participar, proveer, anunciar, eliminar, incluir, dar, considerar, tratar, calificar, considerar, enviar, mido, decir, enviar, defendar, calificar, calificar, señalar, crear, aportar, resaltar, jugar, calificar, comenzar, eliminar, celebrar, generar, reconocer, tener, hacer, lograr, cerrar, negar, tener, hacer, garantizar, indicar, eliminar, decir, sumar, elevar, exportar, importar, considerar, tratar, englobar, seguir, entrar, refrendar, demostrar, vivir, cuestionar, aprobar, oponer, echar, estimar, mostrar él, aprueben, tener, validar, recibir, descargar, perdertir

Summary Verbs: tardar, lograr
Matched Verbs (Summaries) with Counts:
lograr (2)
Matched Count (Summaries): 1

Podemos hacer lo mismo con los **títulos**, por ejemplo, con los que tengan frecuencia **1**.

=== Testing for Frequency: 1 ===

Document ID: text1
Text Verbs: recibir, terminar, decir, recortar, llegar, hacer, juntar, proteger

Title Verbs: decir, peligrir
Matched Verbs (Titles) with Counts:
decir (1)
Matched Count (Titles): 1

6.11 Retos y Soluciones

6.11.1 Problema con el Cálculo de Frecuencias

Uno de los desafíos más importantes que enfrentamos fue calcular correctamente las frecuencias de coincidencia de verbos entre los textos y los campos `title` y `summary`. Inicialmente, los histogramas mostraban frecuencias máximas de solo 10, lo cual resultaba poco realista dado que muchos textos contienen cientos de verbos. Además, las frecuencias estaban incorrectamente distribuidas; por ejemplo, el histograma indicaba que había 4 documentos con una frecuencia de coincidencia de 8, pero al verificar manualmente, no había ninguno.

6.11.1.1 Identificación del Problema Después de analizar el código y realizar múltiples pruebas, descubrimos que el error se debía a que solo estábamos considerando los verbos únicos en lugar de sumar todas las veces que estos aparecían en los textos completos. Esto generaba resultados inconsistentes y poco representativos.

6.11.1.2 Solución Implementada La solución fue realizar una pequeña modificación en el cálculo de las frecuencias. En lugar de simplemente contar si un verbo estaba presente, sumamos todas las apariciones de cada verbo en el texto correspondiente. Esta corrección, aunque sencilla, tuvo un impacto significativo, permitiéndonos obtener histogramas precisos que reflejan correctamente la distribución de frecuencias.

6.11.1.3 Validación con Tests Para asegurarnos de que los cálculos eran correctos, utilizamos los tests desarrollados específicamente para esta tarea. Estos tests nos permitieron verificar las coincidencias verbo por verbo, documentando los resultados y confirmando que las frecuencias calculadas eran exactas. Gracias a estas pruebas, podemos afirmar con confianza que los resultados ahora son consistentes y precisos.