

Tesina

Licenciatura en Estadística

# Análisis de comentarios en redes sociales con latent Dirichlet allocation



Facultad de Ciencias Económicas y Estadística

Alumna: Alfonsina Badin

Director: Ignacio Evangelista

2025

# Índice

<b>1</b>	<b>Agradecimientos</b>	<b>3</b>
<b>2</b>	<b>Resumen</b>	<b>4</b>
<b>3</b>	<b>Introducción</b>	<b>5</b>
<b>4</b>	<b>Objetivos</b>	<b>7</b>
4.1	Objetivo general . . . . .	7
4.2	Objetivos específicos . . . . .	7
<b>5</b>	<b>Metodología</b>	<b>8</b>
5.1	Machine learning . . . . .	8
5.1.1	Aprendizaje supervisado . . . . .	8
5.1.2	Aprendizaje no supervisado . . . . .	8
5.2	Procesamiento de Lenguaje Natural (NLP) . . . . .	9
5.2.1	Recolección de datos . . . . .	10
	Uso de conjuntos de datos públicos . . . . .	10
	Obtención de datos de internet . . . . .	10
	Combinación de fuentes de datos . . . . .	10
5.2.2	Limpieza de texto . . . . .	10
	Pasos preliminares . . . . .	11
	Tokenización . . . . .	11
	Normalización de unicode . . . . .	11
	Eliminación de stopwords . . . . .	13
	Corrección ortográfica . . . . .	13
	Corrección ortográfica empleando distancia de Levenshtein-Damerau . . . . .	14
	Lematización . . . . .	16
5.2.3	Representación de texto . . . . .	17
<b>6</b>	<b>Aplicación práctica</b>	<b>18</b>
6.1	Recolección de datos: comentarios de YouTube . . . . .	18
6.2	Limpieza de texto . . . . .	19

6.3	Representación de texto . . . . .	20
<b>7</b>	<b>Bibliografía</b>	<b>21</b>
<b>8</b>	<b>Anexos</b>	<b>22</b>
8.1	Anexo: Recolección de datos: comentarios de YouTube . . . . .	22

# 1 Agradecimientos

## 2 Resumen

### 3 Introducción

A lo largo de los años, las redes sociales se han convertido en el medio de comunicación más utilizado. Diariamente, los usuarios ingresan a ellas para conversar con otras personas, dar a conocer sus puntos de vista, compartir experiencias, informarse sobre las últimas noticias y entretenerse.

En el año 2020, a raíz de la pandemia mundial por COVID-19, hubo un tipo de contenido que se popularizó con gran velocidad: el *streaming*. El *streaming* es una combinación de radio y televisión, que se transmite en vivo en una plataforma gratuita (YouTube o Twitch) y luego es publicado *On-Demand* para que el usuario pueda verlo fuera de su horario habitual si así lo desea. Por lo general el formato consta de un panel de conductores que charlan de tópicos comunes para la audiencia, generando debates interesantes con los que los usuarios se pueden llegar a identificar. Dentro de cada canal de *streaming* se transmiten distintos programas que tienen un perfil particular y objetivos diferentes ya que su contenido puede variar en entretenimiento, humor, chimentos, noticias, etc.

Su popularidad se debe a que es un contenido gratuito, cercano y que genera una **comunidad** entre los conductores y sus oyentes, quienes no sólo interactúan en vivo a través de un *chat* sino que dejan sus comentarios en otras redes sociales: Instagram, TikTok, YouTube, entre otros.

Al publicar los programas *On-Demand*, los canales de *streaming* acceden a un recurso valioso que refleja la opinión de sus oyentes: los comentarios de YouTube. El análisis de estos textos puede resultar en conclusiones interesantes para los canales ya que permite un entendimiento de los sentimientos que tiene su comunidad con respecto a cada contenido.

Para llevar a cabo este análisis, el presente informe utiliza **latent Dirichlet allocation (LDA)**, una técnica de aprendizaje automático no supervisado que clasifica automáticamente los textos en diferentes categorías o temas según las características del corpus. Esta metodología permite detectar patrones temáticos recurrentes en grandes volúmenes de datos textuales, como los comentarios de YouTube, y extraer información valiosa.

LDA se basa en un modelo probabilístico generativo y es especialmente útil para modelar datos discretos como textos. Es un modelo bayesiano jerárquico de tres niveles, en el cual cada documento (en este caso, cada comentario) se representa como una mezcla de temas, y cada tema se define como una mezcla de palabras. En este sentido, LDA no solo agrupa comentarios en función de temas compartidos, sino que también asigna probabilidades a cada palabra dentro de cada tema, proporcionando una representación explícita y estructurada del contenido del corpus.

Esta capacidad de modelar documentos como combinaciones de múltiples temas lo hace particu-

larmente adecuado para analizar la diversidad temática de los comentarios de YouTube. Por ejemplo, un comentario podría estar relacionado en un 60% con el tema “entretenimiento” y en un 40% con el tema “noticias”, lo que permite identificar la interacción entre los intereses de los usuarios.

Esta tesina tiene por objeto de estudio la técnica latent Dirichlet allocation. Se brinda una introducción al tema, abarcando las definiciones básicas, sus componentes, sus variantes, sus ventajas y limitaciones y sus campos de aplicación. Se incluyen también los conceptos teóricos necesarios para comprender la metodología. Adicionalmente, se presenta la aplicación del modelo en comentarios de redes sociales.

## 4 Objetivos

### 4.1 Objetivo general

El objetivo general del presente proyecto es profundizar en el estudio y aplicación del modelo latent Dirichlet allocation (LDA) para la identificación de tópicos o categorías.

### 4.2 Objetivos específicos

- Comprender las bases del procesamiento de lenguaje natural (NLP por sus siglas en inglés) y las técnicas de representación computacional de texto.
- Mencionar los desafíos y limitaciones que conlleva el ajuste de LDA y propuestas que han surgido para contrarrestar dichos inconvenientes.
- Aplicar LDA en un conjunto comentarios en español (argentino) en la red social YouTube para comprender las opiniones de los oyentes.



## 5 Metodología

### 5.1 Machine learning

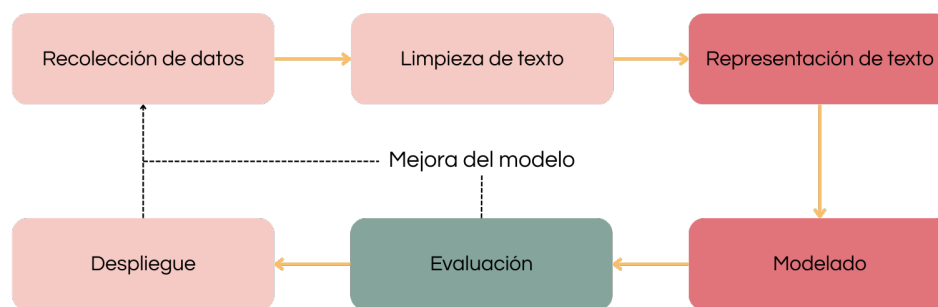
#### 5.1.1 Aprendizaje supervisado

#### 5.1.2 Aprendizaje no supervisado

## 5.2 Procesamiento de Lenguaje Natural (NLP)

El texto, tal como lo leemos y comprendemos los humanos, es rico en significado pero carece de una estructura intrínsecamente numérica, que es la base del procesamiento en los algoritmos de aprendizaje automático. El procesamiento de lenguaje natural es un área de investigación dentro de la informática y la inteligencia artificial (IA) que se ocupa de procesar el lenguaje humano. Este procesamiento generalmente implica traducir el lenguaje natural en datos (números) que una computadora puede usar para aprender sobre el mundo.

El proceso abarca tareas como la recolección, limpieza, normalización y tokenización de texto, estas últimas dos son las responsables de que la información textual pueda ser estructurada de manera que los modelos la interpreten y analicen eficazmente. En la Figura 1 (Vajjala et al. (2020)) se pueden observar los pasos que comprende el proceso.



*Figura 1: Etapas genéricas en NLP*

Aunque no sea evidente, el procesamiento de lenguaje natural (NLP) aparece en el día a día de las personas. Tiene una amplia variedad de aplicaciones prácticas, lo que lo convierte en una herramienta esencial en diversos campos. Por ejemplo:

- Mejoras en motores de búsqueda web.
- Corrección ortográfica durante la búsqueda.
- Revisión gramatical y ortográfica.
- Desarrollo de chatbots y asistentes virtuales.
- Generación de índices y tablas de contenido.
- Filtrado de spam en correo electrónico.
- Personalización en campañas de marketing.

Este amplio rango de usos subraya su importancia como puente entre los lenguajes humanos y las capacidades computacionales.

### 5.2.1 Recolección de datos

El primer paso esencial en cualquier proyecto de procesamiento de lenguaje natural es la recolección de datos. Esta etapa determina la calidad y relevancia de los resultados que se obtendrán al aplicar técnicas avanzadas como latent Dirichlet allocation (LDA). Existen diferentes enfoques para recopilar datos en NLP, dependiendo de los objetivos del análisis y la disponibilidad de recursos. A continuación, se describen las principales estrategias de recolección de datos:

#### Uso de conjuntos de datos públicos

Una opción inicial es buscar conjuntos de datos públicos que sean relevantes para la tarea específica. Si el conjunto de datos es adecuado, se puede proceder directamente a construir y evaluar un modelo. Sin embargo, cuando no se encuentra un dataset que cumpla los requisitos, se debe considerar generar datos personalizados.

#### Obtención de datos de internet

Otra estrategia consiste en identificar fuentes relevantes de datos en internet, como foros de consumidores o plataformas de discusión donde se publiquen consultas. Estos datos pueden ser extraídos automáticamente a través de técnicas de *web scraping* y, posteriormente, etiquetados manualmente por anotadores humanos.

#### Combinación de fuentes de datos

En la práctica, los conjuntos de datos suelen provenir de fuentes heterogéneas, como *datasets* públicos, datos etiquetados manualmente y datos aumentados. Esta combinación es especialmente útil para construir modelos en etapas iniciales, cuando los datos específicos para un escenario particular son limitados.

En esta tesina se optó por hacer uso de una *API* de Google para obtener los datos de entrada para el análisis, este proceso será detallado en el apartado de aplicación práctica

Esta etapa es crucial para garantizar la disponibilidad de datos limpios y útiles, que son la base para las siguientes etapas de procesamiento y modelado. Una vez que se cuenta con los datos recolectados, se procede al siguiente paso: la limpieza y el preprocesamiento de texto.

### 5.2.2 Limpieza de texto

La extracción y limpieza de texto se refiere al proceso de extraer de los datos de entrada el texto sin procesar, eliminando toda la información no textual, como marcas, metadatos, etc., y convirtiéndolo al formato de codificación requerido. La extracción de texto es un paso estándar en el manejo de datos y, usualmente, no emplea técnicas específicas de NLP. Sin embargo, es un paso importante que tiene implicaciones para todos los demás aspectos. Además, también puede ser la parte más demandante en términos de tiempo dentro de

un proyecto.

## **Pasos preliminares**

Previo a la limpieza del texto, es necesario llevar a cabo una serie de transformaciones básicas que garanticen la homogeneidad del corpus. En primer lugar, se convierte todo el texto a minúsculas con el fin de evitar que palabras idénticas sean tratadas como distintas debido a diferencias en el uso de mayúsculas, por ejemplo, “Video” y “video”. Otro paso fundamental consiste en la eliminación de signos de puntuación, caracteres especiales y espacios innecesarios, que no aportan información semántica al análisis y pueden generar una dispersión artificial en el vocabulario. En esta etapa también se contempla la reducción de espacios múltiples a un único espacio, asegurando así una identificación más precisa de las palabras.

Finalmente, se corrigen patrones de escritura característicos de entornos informales, como la repetición exagerada de letras para enfatizar una emoción (por ejemplo, “amoo” en lugar de “amo” o “encantaa” en lugar de “encanta”). Este tipo de modificaciones, frecuentes en comentarios de redes sociales, pueden inflar el vocabulario y dificultar la identificación de términos representativos. La normalización de estas formas al estándar correspondiente resulta clave para mejorar la coherencia interna del corpus y facilitar los pasos posteriores de análisis.

## **Tokenización**

La tokenización es un paso fundamental en el procesamiento del lenguaje natural que tiene como objetivo transformar el texto libre en una secuencia de unidades discretas que puedan ser procesadas computacionalmente. Consiste en segmentar el texto en unidades mínimas llamadas tokens, cuya definición depende de la granularidad aplicada en el proceso:

- A nivel de palabra: cada token corresponde a una palabra. Ejemplo: “Muy bueno el video” → [“Muy”, “bueno”, “el”, “video”].
- A nivel de sub-palabra: cada token es un fragmento de una palabra. Ejemplo: “jugando” → [“jug”, “ando”].
- A nivel de carácter: cada token es un único carácter (letra, signo de puntuación, número, etc.). Ejemplo: “sol” → [“s”, “o”, “l”].
- A nivel de oración: cada token corresponde a una oración completa. Ejemplo: “Hoy llueve. Mañana saldrá el sol.” → [“Hoy llueve.”, “Mañana saldrá el sol.”].

## **Normalización de unicode**

Al limpiar texto, especialmente que se ha obtenido a través de Web Scrapping, es posible que se encuentren varios caracteres Unicode, incluidos símbolos, emojis y otros caracteres gráficos, algunos ejemplos se muestran

en la Figura 2 (Vajjala et al. (2020)).



Figura 2: Caracteres unicode y su representación gráfica

Para interpretar estos símbolos no textuales y caracteres especiales se utiliza la normalización de *unicode*, que convierte el texto en alguna forma de representación binaria para ser almacenado en una computadora. Existen varios esquemas de codificación, y la codificación predeterminada puede variar según el sistema operativo.

El siguiente fragmento de código en R ejemplifica la limpieza de emojis en texto. Utilizando expresiones regulares, se eliminan todos los caracteres que no pertenezcan al alfabeto en minúsculas.

```
comentarios <- c(
  "Me encantó este video ", "Lo volvería a ver!!! ",
  "Qué aburrido no me gustó", "Muy bueno!!! "
)
comentarios_limpios <- gsub("[^a-záéíóúñ ]", " ", tolower(comentarios))
comentarios_limpios <- stringr::str_squish(comentarios_limpios)
comentarios_limpios
```

```
## [1] "me encantó este video"      "lo volvería a ver"
## [3] "qué aburrido no me gustó"  "muy bueno"
```

La salida es un conjunto de comentarios donde los emojis y otros símbolos fueron eliminados, manteniendo únicamente el texto en minúsculas y sin caracteres no deseados.

## Eliminación de stopwords

Los *stopwords* son palabras funcionales que, si bien cumplen un rol gramatical importante (como preposiciones, artículos, pronombres...), no aportan contenido semántico relevante al análisis del significado global del texto.

El objetivo de esta etapa es reducir el ruido lingüístico y centrar el análisis en las palabras que efectivamente comunican información. Por ejemplo, en frases como "gracias por el dato" o "esto es una joya", las palabras **por**, **el**, **es**, **una** son típicamente consideradas *stopwords*, mientras que **gracias**, **dato**, **joya** son términos de interés.

Para llevar a cabo esta tarea se utilizará el paquete **tm** (*Text mining*) que cuenta con un listado de *stopwords* (Tabla 1) y funciones como **removeWords** que permite eliminarlas de un texto dado.

Tabla 1: Algunas stopwords del paquete

de	la	que	el	en	y	a	los	del	se
las	por	un	para	con	no	una	su	al	lo

```
library(tm)
prueba <- c(
  "gracias por el dato",
  "esto es una joya"
)
removeWords(prueba, stopwords("es"))
```

```
## [1] "gracias  dato" "  joya"
```

## Corrección ortográfica

La corrección de errores en la escritura es un aspecto fundamental del procesamiento del lenguaje natural ya que los textos pueden contener diferentes tipos de fallos que afectan su comprensión y coherencia. Los errores en la redacción pueden clasificarse en tres categorías principales (*Moyotl-Hernández, 2016*):

- Errores ortográficos: ocurren cuando una palabra escrita no existe dentro del idioma.
- Errores gramaticales: Se presentan cuando las palabras utilizadas existen en el idioma, pero no son correctas en el contexto de la oración.
- Errores de estilo: se refieren a palabras redundantes, ambiguas o repetidas que afectan la claridad del texto.

En este sentido, un corrector ortográfico tiene como objetivo identificar palabras mal escritas dentro de un texto y sugerir la opción más adecuada a partir de un conjunto de términos válidos en el idioma.

### Corrección ortográfica empleando distancia de Levenshtein-Damerau

La distancia de Levenshtein o distancia de edición es una medida utilizada para calcular la similitud entre palabras, se trata de un conteo de operaciones requeridas para convertir una cadena de caracteres (una palabra) en otra. Las operaciones de edición son:

- Intersección de un caracter: hogar  $\rightarrow$  hogar (agregar ‘a’ entre la ‘h’ y la ‘g’).
- Eliminación de un caracter: argentina  $\rightarrow$  argentina (eliminar la ‘g’).
- Sustitución de un caracter por otro: numero  $\rightarrow$  número (reemplazar la ‘u’ por la ‘ú’).

Hay una generalización de esta medida que consideran el intercambio de dos caracteres como una operación: la distancia de Levenshtein-Damerau. En este método, aparece:

- Transposición de un caracter por otro: paelta  $\rightarrow$  paleta (intercambia el lugar de la ‘e’ y la ‘l’).

El método de corrección basado en la distancia de Levenshtein-Damerau se fundamenta en la búsqueda de la palabra más similar dentro de un diccionario para corregir una palabra mal escrita. Este procedimiento implica generar todas las posibles transformaciones de la palabra errónea mediante las operaciones de edición. Luego, cada una de estas transformaciones se compara con las palabras existentes en el diccionario, y aquellas que coincidan se agregan a una lista de sugerencias. Finalmente, la mejor corrección será aquella con la menor distancia a la palabra original.

En la Tabla 2 se muestran todas las palabras generadas a partir de la palabra errónea **burri** con una sola operación de edición.

Tabla 2: Posibles transformaciones de la palabra burri con distancia de edición uno

Operación	Palabras generadas
Eliminación	urri, brri, buri, bur
Inserción	aburri, bburri, cburri, ..., zburri, burris, burria, ..., burriz
Sustitución	aurri, burrs, murri, ..., burro, burrq, ..., burrz
Transposición	ubrri, bruri, burr, burir

El desafío radica en seleccionar la cantidad de operaciones admitidas para la corrección y cuál de estas opciones es la más apropiada. En el siguiente código se define un diccionario ficticio para este caso y

se calcula la distancia L-D haciendo uso del paquete `stringdist` para cada palabra del mismo. En la Tabla 3 se observan los resultados.

```
# Paquetes
library(stringdist)
library(dplyr)

# 1. Token a corregir
token <- "burri"

# 2. Diccionario ficticio
diccionario <- c("burro", "burrito", "barri", "burla", "buri",
                 "perro", "burris", "aburris", "aburro", "caballo")

# 3. Calculamos distancia DL
resultados <- tibble::tibble(
  candidato = diccionario,
  distancia = stringdist(token, diccionario, method = "dl")
) %>%
  arrange(distancia, candidato)
```

Tabla 3: Posibles correcciones para la palabra 'burri'

Candidato	Distancia	Candidato	Distancia
barri	1	aburro	2
buri	1	burla	2
burris	1	burrito	2
burro	1	perro	3
aburris	2	caballo	6

Si bien es fácil identificar que **perro** o **caballo** no son buenas correcciones, observando el resultado de la distancia L-D se llega a la misma conclusión. Para este conjunto de datos y el ejemplo en particular, hay correcciones mejores en función de este cálculo.

En síntesis, el uso de la distancia de Levenshtein-Damerau constituye una herramienta potente para la corrección ortográfica, ya que permite identificar palabras candidatas a partir de criterios formales de similitud. No obstante, el desafío principal radica en establecer un umbral adecuado de tolerancia en la distancia: una tolerancia demasiado estricta puede dejar sin corregir errores evidentes, mientras que una tolerancia demasiado laxa puede introducir correcciones incorrectas. De igual modo, disponer de un



diccionario amplio, representativo y ajustado al dominio del corpus resulta esencial para que las sugerencias tengan sentido lingüístico y contextual. En conjunto, ambos elementos, son determinantes para perfeccionar los procesos de corrección ortográfica y garantizar resultados confiables en etapas posteriores del análisis de texto.

## Lematización

La lematización es un proceso de normalización léxica en NLP cuyo objetivo es reducir las palabras a su forma canónica o “lema”. Al aplicar lematización:

- Se eliminan variaciones flexivas (tiempo verbal, número, género).
- Se conserva el significado léxico de la palabra.
- Se mejora la coherencia del vocabulario al reducir distintas formas a un único término.

El paquete `UDPipe` implementa modelos de *Universal Dependencies* que permiten realizar tokenización, etiquetado gramatical y lematización de textos en múltiples lenguas.

```
library(udpipe)
modelo <- udpipes_download_model(language = "spanish-gsd")
udmodel <- udpipes_load_model(file = modelo$file_model)
texto <- c("Los estudiantes estaban estudiando en la biblioteca.")
anot <- udpipes_annotate(udmodel, x = texto, doc_id = 1)
```

Tabla 4: Salida de `UDPipe`: tokens, lemas y categorías gramaticales

Token	Lema	UPOS
Los	el	DET
estudiantes	estudiante	NOUN
estaban	estar	AUX
estudiando	estudiar	VERB
en	en	ADP
la	el	DET
biblioteca	biblioteca	NOUN
.	.	PUNCT

En la Tabla 4 se observan los resultados de la lematización con una oración simple. El código devuelve palabras separadas, los lemas correspondientes y el tipo de palabra procesada, entre ellas:

- DET: determinante.
- NOUN: sustantivo.
- VERB: verbo.
- ADP: adposición.
- PUNCT: puntuación.

Estos son algunos de los tipos de palabras que el paquete `UDPipe` reconoce. Este etiquetado permite filtrar aquellas categorías con mayor carga semántica (sustantivos, verbos, adjetivos, adverbios y nombres propios) y así disminuir la dimensionalidad del *input* del modelo.

### 5.2.3 Representación de texto

La extracción de características relevantes de un texto es un paso clave para su procesamiento ya que incluso el mejor modelo posible, con información pobre, devuelve pobres resultados. En este apartado se desarrollarán algunas técnicas para transformar texto en una representación numérica que pueda alimentar un algoritmo de Machine Learning correctamente.









## 6 Aplicación práctica

En este apartado se expone la aplicación práctica de las técnicas de procesamiento de lenguaje natural (NLP) y del modelo latent Dirichlet allocation (LDA) sobre la base de comentarios recolectados de YouTube. El objetivo es mostrar, paso a paso, cómo los procedimientos teóricos previamente descritos se implementan en un corpus real, abarcando desde la carga inicial de los datos hasta la construcción de la matriz documento-término y el posterior modelado de tópicos.

### 6.1 Recolección de datos: comentarios de YouTube

Utilizando la API de YouTube brindada por el servicio de Google Cloud y con la herramienta App Scripts integrada en SpreadSheets (Google), se realizó una recolección de información básica de los videos publicados entre el 1 de enero del 2023 y el 21 de diciembre del 2024 de los canales más vistos del país: LuzuTV, Olga, Un Poco De Ruido, La Casa Streaming, Bondi Live, Vortexix, Urbana Play y Blender. Luego, filtrando sólo los videos con duración mayor a 10 minutos (para evitar incluir *shorts* o recortes), se utilizó la misma API conectada a R (versión 4.4.0) y el paquete **tuber** para la recolección de comentarios de los videos identificados en la primer instancia ([Anexo](#)). En la Tabla 5 se puede identificar la cantidad de información recolectada para cada canal de *streaming*.

Tabla 5: Información recolectada de cada canal

Foto	Canal	Videos	Comentarios	Comentarios recolectados
	LuzuTV	4.024	743.165	444.097
	Olga	1.184	228.781	217.972
	Un Poco De Ruido	108	101.975	90.738
	La Casa Streaming	434	7.822	7.524
	Bondi Live	769	23.712	21.711
	Vortexix	2.757	165.553	153.010
	Urbana Play	11.565	240.244	224.437
	Blender	2.261	180.773	166.544
	<b>Total</b>	<b>23.102</b>	<b>1.692.025</b>	<b>1.326.033</b>

Es de destacar que la API de YouTube cuenta con un límite de consulta para las respuestas a comentarios: se retienen únicamente los primeros 5 comentarios por cadena. Sin embargo, por estar dentro de un mismo “hilo”, estos comentarios no cambian drásticamente en su temática y no afecta los fines de esta investigación. Además, el límite mencionado no permite tomar toda la información de una sola corrida, hubo que realizar la recolección de a un día por canal usando dos conexiones en simultáneo. Esta diferencia en la recolección se mitiga filtrando los comentarios según su fecha de publicación previa al día 05/09/2025 (día de la primera recolección).

## 6.2 Limpieza de texto

En total se recopilaron 1.692.025 comentarios a los que se les aplicó una secuencia de preprocesamiento estándar en NLP:

1. Minúsculas: conversión completa del texto a minúscula para evitar duplicidad por capitalización.
2. Colapso de vocales repetidas: reducción de alargamientos (“amoo”, “encantaa” → “amo”, “encanta”), útil para comentarios informales.
3. Filtrado de caracteres: se conservaron únicamente letras del alfabeto español (incluyendo tildes y “ñ”) y espacios, eliminando números, emojis y signos.
4. Normalización de espacios: supresión de espacios múltiples.
5. Eliminación de stopwords: remoción de palabras que no aportan carga semántica.

Con el objetivo de asistir la corrección ortográfica y preservar la jerga propia del dominio, se construyó un diccionario combinado a partir de dos fuentes: por un lado, las 20.000 palabras más frecuentes del propio corpus de comentarios y, por otro, un listado exhaustivo de palabras en español extraído de la base *Kaikki.org*, que recopila el contenido de *Wiktionary*. De esta manera, el diccionario resultante no solo incorpora modismos, nombres propios y expresiones características del corpus analizado, sino también un repertorio amplio de formas válidas del idioma español, constituyendo así una base más robusta para la etapa de corrección ortográfica.

La estrategia propuesta para la corrección ortográfica es identificar *tokens* fuera del diccionario y proponer correcciones mediante la distancia de Leveinshtein-Damerau. En caso de empate en la distancia mínima, se seleccionará como corrección al candidato de mayor frecuencia en el diccionario, lo que favorece a las expresiones más habituales del corpus. Si bien Damerau sugiere que la gran mayoría de los errores se corrigen con una sola edición, en el presente trabajo se adoptó un umbral máximo de distancia menor o igual a 3. Esta decisión responde a la naturaleza del corpus, comentarios de YouTube, donde son habituales errores acumulados o repeticiones de caracteres.

### 6.3 Representación de texto

## 7 Bibliografía

- Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3, 993-1022.
- El Economista. (2024). *Streaming en Argentina: cuáles son los canales líderes y cómo monetizan sus contenidos*. <https://eleconomista.com.ar/sociedad-redes/streaming-2024-argentina-canales-lideres-figuras-estrategias-monetizacion-n74497>.
- Loo, M. van der. (2025). *stringdist: Approximate String Matching and String Distance Functions*. <https://cran.r-project.org/web/packages/stringdist/stringdist.pdf>
- Moyotl-Hernández, E. (2016). Método de corrección ortográfica basado en la frecuencia de las letras. *Research in Computing Science*, 124, 145-156.
- Vajjala, S., Majumder, B., Gupta, A., & Surana, H. (2020). *Practical Natural Language Processing*. O'Reilly Media.

## 8 Anexos

### 8.1 Anexo: Recolección de datos: comentarios de YouTube

A continuación se detalla el código con el que se tomó la información básica de cada video desde Google Apps Script. Para cada canal se configuró su ID y nombre.

- LuzuTV: UCTHaNTsP7hsVgBxARZTuajw
- Olga: UC7mJ2EDXFomeDIRFu5FtEbA
- Un poco de ruido: UCg6kTB4vw1XYFBR4TtHaBuQ
- La casa streaming: UC4u0BhsSi33PS20\_1JHiC5A
- Bondi: UCnZidingmuqNkaT9Wm64Xxg
- Vorterix: UCvCTWHCbBC0b9UIeLeNs8ug
- Ubana Play: UCC1kfsMJko54AqxteFEct-A
- BLender: UC6pJGaMdx5Ter\_8zYbLoRgA

```

/***** CONFIG *****/
const CHANNEL_ID = 'id_canal';
const SHEET_VIDEOS = 'nombre_canal';

const START_DATE_ISO = '2023-01-01T00:00:00Z';
const END_DATE_ISO   = '2024-12-22T00:00:00Z';

// Control de timeout
const PAGES_PER_RUN = 8;
const MINUTES_BETWEEN_RUNS = 1;
const SAFETY_MS = 5 * 60 * 1000;
const ROW_BUFFER = 200;

/*****/

function startVideoCrawl() {
  // limpiar triggers del batch
  ScriptApp.getProjectTriggers()
    .filter(t => t.getHandlerFunction() === 'crawlVideosBatch')
    .forEach(t => ScriptApp.deleteTrigger(t));
}
```

```

const ch = YouTube.Channels.list('snippet,statistics,contentDetails', { id: CHANNEL_ID });
if (!ch.items || !ch.items.length) throw new Error('Canal no encontrado');
const uploadsId = ch.items[0].contentDetails.relatedPlaylists.uploads;
const channelName = ch.items[0].snippet.title;
const subs = ch.items[0].statistics.subscriberCount;

const ss = SpreadsheetApp.getActiveSpreadsheet();
let sh = ss.getSheetByName(SHEET_VIDEOS);
if (!sh) sh = ss.insertSheet(SHEET_VIDEOS); else sh.clear();
sh.appendRow([
  'ChannelId', 'Canal', 'Suscriptores',
  'VideoId', 'Titulo', 'URL',
  'Duración ISO', 'Segundos', 'Fecha Publicación (UTC)',
  'Vistas', 'Likes', 'Comentarios(API)', 'Tipo live', 'Idioma'
]);

const props = PropertiesService.getScriptProperties();
props.deleteAllProperties();
props.setProperties({
  'v.uploadsId': uploadsId,
  'v.pageToken': '',
  'v.channelName': channelName,
  'v.subs': String(subs),
  'v.startISO': START_DATE_ISO,
  'v.endISO': END_DATE_ISO,
  'v.excludeShorts': String(EXCLUDE_SHORTS),
  'v.excludeLives': String(EXCLUDE_LIVES)
}, true);

Logger.log('startVideoCrawl OK. Canal=' + channelName);
crawlVideosBatch(); // primer lote
}

function crawlVideosBatch() {
  const props = PropertiesService.getScriptProperties();

```



```

const uploadsId    = props.getProperty('v.uploadsId');
if (!uploadsId) { Logger.log('ERROR: sin estado. Ejecutá startVideoCrawl().'); return; }
let pageToken      = props.getProperty('v.pageToken') || '';
const channelName  = props.getProperty('v.channelName');
const subs         = props.getProperty('v.subs');
const startISO     = props.getProperty('v.startISO');
const endISO       = props.getProperty('v.endISO');
const excludeShorts= props.getProperty('v.excludeShorts') === 'true';
const excludeLives = props.getProperty('v.excludeLives')  === 'true';

const startMs = Date.parse(startISO);
const endMs   = Date.parse(endISO);

const ss = SpreadsheetApp.getActiveSpreadsheet();
const sh = ss.getSheetByName(SHEET_VIDEOS);
const t0 = Date.now();

let pages = 0, rowsWritten = 0;

while (pages < PAGES_PER_RUN) {
  if ((Date.now() - t0) > SAFETY_MS) break; // margen anti-timeout

  const pl = YouTube.PlaylistItems.list('contentDetails', {
    playlistId: uploadsId, maxResults: 50, pageToken: pageToken || undefined
  });
  const items = pl.items || [];
  Logger.log(`Playlist page items=${items.length} token=${pageToken || '(first)'}`);
  if (!items.length) { finishVideosCrawl_(); return; }

  const ids = items.map(x => x.contentDetails.videoId).join(',');
  const vd  = YouTube.Videos.list('snippet,statistics,contentDetails', { id: ids });

  let pageMin = +Infinity, pageMax = -Infinity;
  const rows = [];

```

```

(vd.items || []).forEach(v => {

    const sn = v.snippet, st = v.statistics, cd = v.contentDetails;

    const pubMs = Date.parse(sn.publishedAt); // UTC

    if (pubMs < pageMin) pageMin = pubMs;
    if (pubMs > pageMax) pageMax = pubMs;

    // Si cae dentro del rango, aplico filtros y guardo
    if (pubMs >= startMs && pubMs < endMs) {
        const secs = iso8601ToSeconds_(cd.duration);
        const liveType = sn.liveBroadcastContent || 'none';
        if (excludeShorts && secs < 60) return;
        if (excludeLives && liveType !== 'none') return;

        rows.push([
            CHANNEL_ID, channelName, subs,
            v.id, sn.title, 'https://www.youtube.com/watch?v=' + v.id,
            cd.duration, secs, sn.publishedAt,
            Number(st.viewCount||0), Number(st.likeCount||0), Number(st.commentCount||0),
            liveType, sn.defaultAudioLanguage || ''
        ]);
    }
});

if (rows.length) { appendBatch_(sh, rows); rowsWritten += rows.length; }

if (pageMax < startMs) {
    Logger.log(`FIN por corte temprano: llegamos antes de ${START_DATE_ISO}. Páginas=${pages+1}, filas acumuladas`);
    finishVideosCrawl_();
    return;
}

pageToken = pl.nextPageToken || '';
pages++;
if (!pageToken) {
    Logger.log(`FIN: páginas=${pages}, filas=${rowsWritten}.`);
}

```

```

    finishVideosCrawl_();
    return;
  }
}

props.setProperty('v.pageToken', pageToken);
SpreadsheetApp.flush();
Logger.log(`Continuará: páginas=${pages}, filas=${rowsWritten}.`);
ScriptApp.newTrigger('crawlVideosBatch').timeBased().after(MINUTES_BETWEEN_RUNS * 60 * 1000).create();
}

function finishVideosCrawl_() {
  ScriptApp.getProjectTriggers()
    .filter(t => t.getHandlerFunction() === 'crawlVideosBatch')
    .forEach(t => ScriptApp.deleteTrigger(t));
  PropertiesService.getScriptProperties().deleteAllProperties();
  Logger.log('Crawl de VIDEOS finalizado.');
```

```

}
```

```

function appendBatch_(sheet, rows2D) {
  const needed = rows2D.length;
  const last   = sheet.getLastRow();
  const max    = sheet.getMaxRows();
  const cols   = rows2D[0].length;

  const free = max - last;
  if (free < needed) {
    const toAdd = (needed - free) + ROW_BUFFER;
    sheet.insertRowsAfter(max, toAdd);
  }

  sheet.getRange(last + 1, 1, needed, cols).setValues(rows2D);
}
```

```

function iso8601ToSeconds_(dur) {
  const m = dur && dur.match(/PT(?:\d+H)?(?:\d+M)?(?:\d+S)?/);
```

```

if (!m) return 0;

const h = parseInt(m[1]||'0',10), mi = parseInt(m[2]||'0',10), s = parseInt(m[3]||'0',10);

return h*3600 + mi*60 + s;
}

```

Mediante R, se hizo uso del siguiente código para recolectar los comentarios de cada video captado en el paso anterior. Notar que `cliend_id` y `client_secret` son las credenciales utilizadas para conectar al servicio de API de Google y es único por proyecto y por usuario.

```

library(tuber)
library(dplyr)
library(purrr)
library(stringr)
library(tidyr)
library(readr)
library(readxl)
library(janitor)

canal <- read_excel("canal.xlsx") %>%
  clean_names() %>%
  distinct()

yt_oauth(
  app_id      = client_id,
  app_secret  = client_secret,
  scope       = "ssl",    # <-- no pongas la URL; usa "ssl"
  token       = ""
)

video_ids <- canal$video_id

comentarios_canal <- data.frame()

for (i in 1:nrow(canal)) {
  all_comments <- get_all_comments(video_id = video_ids[i])
  comentarios_canal <- rbind(comentarios_canal, all_comments)
}

```

}