UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

OPTIMIZACIÓN DEL SISTEMA DE ANÁLISIS Y CARACTERIZACIÓN DE DISCUSIONES EN TWITTER "TSUNDOKU"

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

NICOLÁS FRANCISCO GARCÍA RÍOS

PROFESOR GUÍA:
EDUARDO GRAELLS GARRIDO

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
FABIÁN VILLENA RODRÍGUEZ

Este trabajo ha sido realizado en cotutela con el Centro Nacional de Inteligencia Artificial

SANTIAGO DE CHILE
2023

OPTIMIZACIÓN DEL SISTEMA DE ANÁLISIS Y CARACTERIZACIÓN DE DISCUSIONES EN TWITTER "TSUNDOKU"

Tsundoku es un sistema de análisis y de caracterización de discusiones en *Twitter*, desarrollado por el profesor Eduardo Graells Garrido en conjunto con otros académicos del área de la computación. El objetivo principal de este sistema es la detección de postura de los usuarios en torno a algún tópico de discusión, junto con la detección de *bots* dentro de la misma. Este sistema ha sido ocupado para la caracterizar la discusión en torno al referéndum por una nueva constitución en Chile el año 2020, y sigue siendo ocupado por académicos de distintas áreas de estudio para otros tópicos de discusión.

Si bien *Tsundoku* es sumamente útil, también presenta ciertas limitantes que dificultan su uso. Principalmente se destaca el tiempo de ejecución excesivo en el preprocesamiento de *tweets*, influenciado por el manejo de datos con archivos en formato *json*, que son inadecuados para este tipo de estudios. Además, la clasificación del lugar de procedencia de los usuarios presenta resultados insatisfactorios, y muy por debajo de la evaluación de la predicción de postura. Así, nace la oportunidad de integrar herramientas de procesamiento masivo de datos dentro de *Tsundoku*, junto con algunas mejoras en torno a la clasificación que podrían explorarse.

Para lo anterior, se integró el uso de la librería *PyArrow* que permite la interoperabilidad de librerías de manejo de *dataframes* como *dask* y *pandas*, además de incluir métodos optimizados para la lectura y escritura de archivos especializados para el manejo de grandes cantidades de datos. El nuevo formato ocupado para el manejo de *dataframes* corresponde a ficheros *parquet*, creado especialmente para lecturas eficientes.

Además, se integraron *word embeddings* al proceso de clasificación, permitiendo añadir un acercamiento contextual al procesamiento de texto. Estos *embeddings* fueron obtenidos a partir de la arquitectura *Transformers*, ocupando el modelo pre-entrenado en español conocido como BETO y desarrollado por académicos de la Universidad de Chile.

Lo anterior resultó en una reducción del 92 % del tiempo de ejecución para el preprocesamiento de *tweets*, junto con la creación de un programa que permita a los usuarios transformar sus datos de *json* a *parquet* para un manejo eficiente de datos. Por otro lado, la integración de *word embeddings* a la clasificación permitió aumentar en un 7 % la exactitud de la clasificación del lugar de origen del usuario. Se espera que la exploración e incorporación de más herramientas de procesamiento de lenguaje natural pueda ampliar las oportunidades de estudio y mejorar la detección de postura con *Tsundoku*.

"Perhaps we can fly. All of us.
How will we ever know unless we leap from some tall tower?
No man ever truly knows what he can do unless he dares to leap"
- George R. R. Martin, A Feast For Crows

Agradecimientos

A mis padres, mis modelos a seguir. Gracias por su apoyo incondicional, por haberme enseñado sus valores, y por todo su esfuerzo que me han llevado hasta aquí.

A mi hermano Marcelo, mi compañero de vida y mejor amigo.

A mis amigos de la universidad, en especial a los 31 gramos. Compartir risas y buenos momentos han hecho mi paso por la universidad en una experiencia inolvidable.

A la Cami y al Aron, que aunque la distancia nos separe me han alentado a seguir adelante.

A mis amigos Mati y José. Su amistad y apoyo han sido esenciales en este trayecto.

Al Javo, por ser un buen amigo y mi apañe durante mi paso en la universidad. Extrañaré todos esos viajes en el mako, y todas las veces me has hecho rabiar por tus faltas ortográficas.

Por último, no puedo olvidarme del profesor Eduardo Graells. Gracias por su guía y consejos que han contribuido en mi crecimiento académico.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

Las redes sociales han adquirido un papel crucial en los procesos de creación de significado dentro de las comunidades [1], y a medida que la tecnología evoluciona, lo "social" se ha convertido en algo más que relaciones entre personas. Estas actualmente brindan una gran variedad de servicios de información que van desde entretenimiento, noticias, juegos, entre otros. Sin embargo, la raíz de este tipo de plataformas se ha mantenido constante, y esta corresponde a la comunicación de ideas entre sus usuarios.

Las plataformas web han permitido la manifestación y debates desde su misma génesis, y los avances de la tecnología no solo traen nuevas oportunidades de expresión con el resto del mundo, si no que además nos permiten registrar estas interacciones en distintos formatos. Durante el surgimiento de las primeras plataformas de tipo *blog* a mediados de los 90' como *link.net* o *Blogger* era común ver este tipo de interacciones en formatos puramente web, pero hoy en día y con el refinamiento de algunas plataformas como *Facebook* es posible tener mayor acceso a los datos de los participantes de una discusión.

Una de las redes sociales enfocada a este tipo de interacciones y que cuenta con una gran penetración en nuestro país y en el resto del mundo occidental es *Twitter*. Desde su fundación el año 2006 ha sido un referente en el mundo digital con su formato revolucionario de *microblogging*, consagrándose como una red de alto impacto en lo referido a medios de comunicación y de generación de discusiones. En consecuencia, *Twitter* ha llamado la atención de muchos académicos de distintas áreas de investigación, las que van desde ramas de la lingüística hasta ciencias de datos y estadística.

Twitter no solo es una red social, si no que también corresponde a un cúmulo de información que registra el cómo los seres humanos nos relacionamos digitalmente, lo que ha

suscitado una gran cantidad de preguntas de investigación relativas a la comunicación y a la extracción de datos a partir de texto. Lo anterior no es un tópico realmente novedoso, puesto que la extracción automática de información a partir de textos ha sido un tema de investigación tremendamente importante dentro del área del procesamiento de lenguaje natural por décadas.

Entre los principales problemas de investigación relacionadas con la extracción de información de texto son el análisis de sentimientos, el reconocimiento de emociones, la minería de argumentos y, más recientemente, la detección de postura. Esta última se considera un subproblema del análisis de sentimientos y busca identificar la opinión o actitud de un sujeto hacia un concepto en particular. A diferencia del análisis de sentimientos, la detección de postura se centra en detectar la alineación y evaluación del sujeto hacia un tema [2].

Aunque se han realizado investigaciones sobre la detección y clasificación de postura en *Twitter*, gran parte de ellas se han centrado en poblaciones anglosajonas y en su mayoría se basan en datos de ciudadanos estadounidenses. Por lo tanto, surge la oportunidad de investigar esta red social en el contexto de una población de habla hispana, específicamente en Chile, con el objetivo de obtener resultados más cercanos a la realidad de nuestro país.

Uno de los estudios más recientes que ha aprovechado esta oportunidad para investigar estas interrogantes en nuestro país es el trabajo titulado *'Every Colour You Are: Stance Prediction and Turnaround in Controversial Issues'* [3]. En este estudio, se presenta una metodología para medir y caracterizar la postura política de los usuarios que participan en discusiones específicas sobre el debate del aborto en Chile y Argentina.

Esta metodología permitió clasificar las posturas de las cuentas de *Twitter* a favor o en contra de la legalización del aborto libre en ambos países. Se basó en una medición de confianza que evalúa la probabilidad de adherencia a una postura por cada usuario. El proceso de clasificación se llevó a cabo mediante el uso de un clasificador basado en árboles de decisión secuenciales conocido como *XGBoost* [4]. Además de la postura en torno a un tema, también fue posible clasificar y caracterizar a los usuarios en diferentes atributos demográficos, como género, edad, lugar de procedencia y relevancia dentro de la discusión.

Por otro lado, en los últimos años se ha visto el surgimiento de nuevas entidades que generan contenido en las redes sociales, como lo son empresas o medios de comunicación. Sin embargo, también existen usuarios que no son administrados directamente por seres humanos. Estos usuarios son conocidos como *bots*[1] y el contenido que producen puede simular el comportamiento de un usuario común en la aplicación.

Aunque en general los *bots* en las redes sociales tienen comportamientos benévolos [5], en los últimos años ha habido casos en los que los *bots* han sesgado las opiniones de los

[1] *bots*: Programa que realiza tareas automatizadas dentro de una red (ver Glosario)

usuarios reales, manipulando y amplificando artificialmente el contenido compartido en las redes sociales [6]. Esta situación ha impulsado el desarrollo de métodos para detectar y caracterizar los *bots* en las redes sociales [7, 8], con el objetivo de comprender su papel en las interacciones mediadas por plataformas de *microblogging*.

Entre las últimas publicaciones relacionadas con la detección de *bots* se encuentra el estudio titulado *'Bots Don't Vote, but They Surely Bother!: A Study of Anomalous Accounts in a National Referendum'* [6], el cual se centró en la discusión política en torno al Referéndum Constitucional chileno del año 2020 en *Twitter*, y cuyo objetivo principal fue comprender el papel de los *bots* en la discusión constitucional chilena. Para lograr esto, se utilizó la metodología desarrollada en el trabajo previo mencionado, *'Every Colour You Are'* [3], y se adaptó para incluir la detección de *bots*.

El procedimiento para detectar los *bots* entre los numerosos usuarios que participan en una discusión se basa en la premisa de que este tipo de cuentas muestran comportamientos anómalos en comparación con el resto de la población [8]. Con esta suposición como base, se empleó el algoritmo de detección de anomalías denominado *Isolation Forest* [9, 10] para identificar comportamientos anómalos dentro de la plataforma y determinar qué usuarios podrían ser considerados como *bots*.

Así nace *Tsundoku*, un sistema de análisis y caracterización de las discusiones en *Twitter* implementado en lenguaje *Python*, que recopila la metodología desarrollada en los trabajos *'Every Colour You Are'* [3] y *'Bots Don't Vote'* [6] para el estudio de las discusiones en torno a un tema configurable. Este sistema ha sido creado por el profesor Eduardo Graells Garrido y ha sido utilizado por académicos de diversas áreas para analizar grandes volúmenes de datos de la plataforma *Twitter* [11]. El código de *Tsundoku* está disponible públicamente en *GitHub* [12] y sienta las bases de este trabajo.

1.2. Motivación

Si bien el trabajo *'Bots Don't Vote'* [6] logró estudiar el impacto que generaron los *bots* a la discusión constitucional, este sistema presenta algunas limitantes. Lo anterior se debe a que el sistema *Tsundoku* es bastante restrictivo en cuanto al tipo y a la estructura de datos que puede procesar, generando algunas dificultades de uso.

En primer lugar, el proceso de extracción de datos ocupa la API v1.1 de *Twitter*, que impone restricciones significativas en la cantidad de *tweets*[2] obtenidos. Para superar esta limitación, se desarrolló un programa que solicita constantemente datos a la plataforma y distingue entre *tweets* previamente obtenidos y los nuevos obtenidos de forma continua.

[2] *tweet*: un mensaje publicado en Twitter que contiene texto, fotos, GIF o video (ver Glosario)

El profesor Eduardo Graells ha desempeñado un papel fundamental en el desarrollo del proceso de obtención de datos, lo que ha resultado en una gran cantidad de información acumulada desde principios de 2020 hasta finales de 2022. En promedio, se recopilan un millón de *tweets* diarios, los cuales se almacenan continuamente en formato de *JavaScript Object Notation* (.json) y constituyen la base de datos principal del sistema *Tsundoku*.

Sin embargo, esta acumulación de datos ha presentado dificultades en su manejo debido al uso de archivos no óptimos para el procesamiento de datos a gran escala. El manejo de grandes volúmenes de *tweets* en formato *.json* ha ralentizado considerablemente las operaciones y la generación de tablas, lo que se convierte en el principal desafío a abordar.

Además, se ha observado un rendimiento insatisfactorio en algunos clasificadores de *Tsundoku*. Específicamente, los resultados obtenidos con el clasificador del lugar de procedencia de los usuarios no cumplen con las expectativas. El porcentaje de precisión[3] global del clasificador en este grupo de estudio es del 23 % con un promedio ponderado de 27 %, siendo las categorías de ruido (*noise*) con un *f1-score*[4] de 0.59 y las detecciones de la región metropolitana con 0.28 las que presentan los resultados más favorables.

Este bajo rendimiento del clasificador abre una oportunidad de mejora, permitiendo la integración de más datos de los usuarios o el uso de herramientas de procesamiento de lenguaje natural. Actualmente, el funcionamiento de *Tsundoku* se basa en un clasificador de árboles de decisión, que si bien es rápido en términos de entrenamiento y ejecución, limita el procesamiento de texto a nivel de tokens. El clasificador principal recibe como entrada una serie de matrices que contienen información de las interacciones de los usuarios como *retweets*[5] y el vocabulario utilizado por los usuarios dentro del conjunto de datos del estudio.

Hasta el momento, no se ha incorporado ninguna tecnología o herramienta de procesamiento de lenguaje natural basada en contexto. La integración de una nueva herramienta de *NLP*[6] podría conducir a mejores resultados en el proceso de clasificación de postura y caracterización de los usuarios, e incluso permitir la exploración de nuevas preguntas de investigación relacionadas con la plataforma. Esta incorporación de herramientas de *NLP* basadas en contexto tiene el potencial de enriquecer y mejorar significativamente las capacidades de clasificación y análisis actuales del sistema.

Tsundoku ya ha demostrado ser tremendamente útil tanto a nivel académico al permitir estudiar comportamientos dentro de la red social *Twitter*, como para análisis estadístico y de contingencia que ha llevado el trabajo a la prensa. Agilizar y facilitar su uso a nuevos investigadores puede potencialmente generar nuevas investigaciones en torno a las Ciencias Sociales Computacionales, un área de la Ciencia de la computación que está en auge.

[3] Precisión o *precision*: métrica de evaluación que mide la precisión de un clasificador (ver Glosario)
[4] Valor-F o *f1-score*: estadística que junta la precisión y la exhaustividad de un clasificador (ver Glosario)
[5] *retweets*: Corresponde a un *tweet* que es compartido o difundido públicamente (ver Glosario)
[6] *Natural Lenguage Processing* o Procesamiento de Lenguaje Natural en español

1.3. Objetivos

Vista la oportunidad de optimización y de mejoras al sistema de análisis y caracterización de discusiones *Tsundoku*, el objetivo principal de este trabajo es el de actualizar e integrar herramientas de procesamiento masivo de datos al sistema, que permitan una lectura y un manejo de archivos mucho más eficiente. A lo anterior se le suma también la exploración e incorporación de nuevas tecnologías al proceso de clasificación actual del sistema, que incluyan algún enfoque contextual del texto para la detección de postura y otras métricas demográficas.

Es posible desglosar el objetivo anterior en una serie de objetivos más concretos:

1. Integración de alguna herramienta de procesamiento masivo de datos, que permita trabajar tanto *dataframes* como archivos de forma más eficiente.
2. Actualizar el sistema actual para deprecar el uso de archivos *.json*, y sustituirlo a algún archivo de estructuras comprimidas, de acuerdo a la tecnología ocupada para el procesamiento de datos.
3. Extender el proceso de clasificación, integrando nuevas tecnologías de procesamiento de lenguaje natural o similares basadas en aprendizaje basado en contexto.
4. Documentar y distribuir el código del proyecto de forma ordenada, y que facilite su uso para aquellos nuevos usuarios que deseen ocupar el sistema.

Capítulo 2

Marco Teórico y Antecedentes

Antes de adentrarse en los detalles del funcionamiento y los antecedentes del sistema *Tsundoku*, es esencial comprender el concepto de clasificación, y el marco teórico relacionado con la detección de postura y el procesamiento de texto.

Un clasificador es un algoritmo de aprendizaje automático que se utiliza para asignar una etiqueta (*label*) o categoría a una entrada desconocida basándose en sus características observadas. Estos modelos se utilizan en una amplia variedad de aplicaciones para tomar decisiones y realizar predicciones basadas en datos.

El objetivo principal de un clasificador es aprender patrones y relaciones entre las características de los datos de entrenamiento para hacer predicciones precisas sobre nuevos datos no etiquetados. Estas predicciones se basan en la información adquirida durante el proceso de entrenamiento, donde el algoritmo ajusta sus parámetros para optimizar el rendimiento en la tarea de clasificación.

Dentro de este esquema, se encuentra el área del procesamiento de lenguaje natural, también conocido como *Natural Language Processing* o *NLP*. El *NLP* es un campo de las ciencias de la computación y la lingüística que se enfoca en permitir a las máquinas comprender, interpretar y generar lenguaje humano de manera natural.

El *NLP* implica una serie de tareas, incluyendo el procesamiento de texto, análisis de sentimientos, traducción automática, extracción de información y clasificación de texto. Respecto a los modelos de clasificación de texto, el objetivo es asignar una etiqueta o categoría específica a un texto dado, como identificar el sentimiento de una reseña, clasificar temas de artículos o detectar spam en correos electrónicos.

Aquí encontramos la detección de postura (también conocida como análisis de sentimiento de opinión o detección de polaridad). Esta es un área del procesamiento de lenguaje natural

que se enfoca en identificar y categorizar la actitud, emoción o sentimiento expresado en un texto hacia un tema o entidad específica [2].

La detección de postura es una tarea desafiante debido a la ambigüedad y la variabilidad lingüística del lenguaje humano. Las opiniones pueden expresarse de múltiples formas y pueden estar implícitas o explícitas en el texto. Además, el contexto y las palabras utilizadas pueden influir en la interpretación de la polaridad del sentimiento.

2.1. Modelos de Clasificación

2.1.1. Árboles de decisión (*Decision Trees*)

Un árbol de decisión (o *Decision Tree*) es un algoritmo de aprendizaje automático no paramétrico, que se utiliza tanto para tareas de clasificación como de regresión. Tiene una estructura jerárquica en forma de árbol, que consta de un nodo raíz, ramas, nodos internos y nodos hoja.

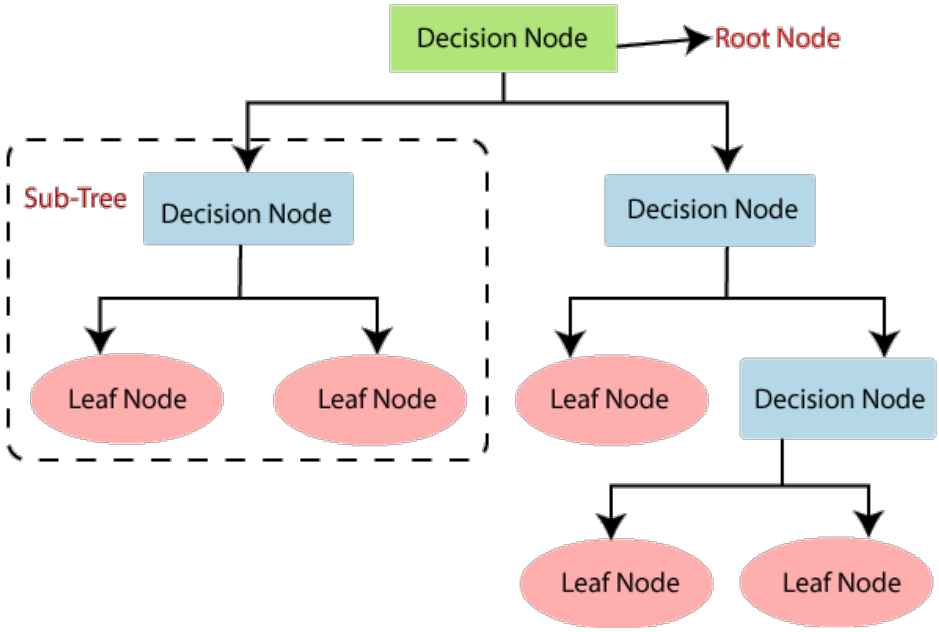

Figura 2.1: Diagrama resumen del funcionamiento de un *Decision Tree*

El algoritmo de árbol de decisión busca dividir el conjunto de datos de entrenamiento en subconjuntos homogéneos en función de las características. Comienza con el nodo raíz que representa todo el conjunto de datos y selecciona la característica que mejor divide los datos en subconjuntos más puros con respecto a las etiquetas. Esta característica se convierte en un nodo de decisión, y el conjunto de datos se divide en diferentes ramas del árbol.

El criterio de división de estos conjuntos suele basarse en la medida de Impureza de Gini, obtenido al sumar la probabilidad de cada elemento siendo elegido multiplicado por la probabilidad de un error en su categorización:

$$I_G(f) = \sum_{i=1}^{m} f_i(1 - f_i) = 1 - \sum_{i=1}^{m} f_i^2 \tag{2.1}$$

Donde f_i corresponde a una fracción del conjunto de datos etiquetados con un valor i, y m corresponde a la cantidad de etiquetas. Equivalentemente, existen otras variantes de árboles de decisión que suelen ocupar una medida de ganancia de información, la cual está basada en el concepto de entropía en teoría de la información:

$$I_E(f) = -\sum_{i=1}^{m} f_i \log_2 f_i \tag{2.2}$$

El aprendizaje del clasificador de árboles de decisión emplea una estrategia de dividir y conquistar (*divide and conquer*), al realizar una búsqueda ávida para identificar los puntos óptimos de división dentro de un árbol. Este proceso de división se repite de manera descendente y recursiva hasta que todos los registros han sido clasificados bajo etiquetas de clase específicas, o hasta que se alcanzan condiciones de parada previamente definidas, como el tamaño mínimo del subconjunto o la profundidad máxima del árbol.

2.1.2. *Random Forest*

Un *Random Forest* es un algoritmo de aprendizaje automático que se utiliza tanto para tareas de clasificación como de regresión. Es una técnica de ensamble que combina múltiples árboles de decisión independientes para mejorar la precisión y la generalización del modelo. Este ensamble se realiza calculando la ponderación de las respuestas de cada uno de estos árboles, o tomando la mayoría entre todos ellos.

Dado un conjunto de entrenamiento $X = x_1, ..., x_n$ con las etiquetas $Y = y_1, ..., y_n$, se crean muestras aleatorias con reemplazo del conjunto de entrenamiento original B veces. Este proceso de selección aleatoria es conocido como *bagging*.

Luego, para cada $b = 1, ..., B$:

1. Se selecciona la muestra con reemplazo de n ejemplos de entrenamiento de X e Y, llamados X_b, Y_b.
2. Se entrena un árbol de decisión (sección 2.1) f_b en X_b, Y_b.

Después del entrenamiento, las predicciones para muestras no etiquetadas x' se pueden hacer promediando las predicciones de todos los árboles de decisión individuales en x':

$$\hat{f}(x') = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{2.3}$$

Donde $\hat{f}(x')$ representa la predicción final promediada para la muestra x' y $f_b(x')$ es la predicción del árbol b para x'.

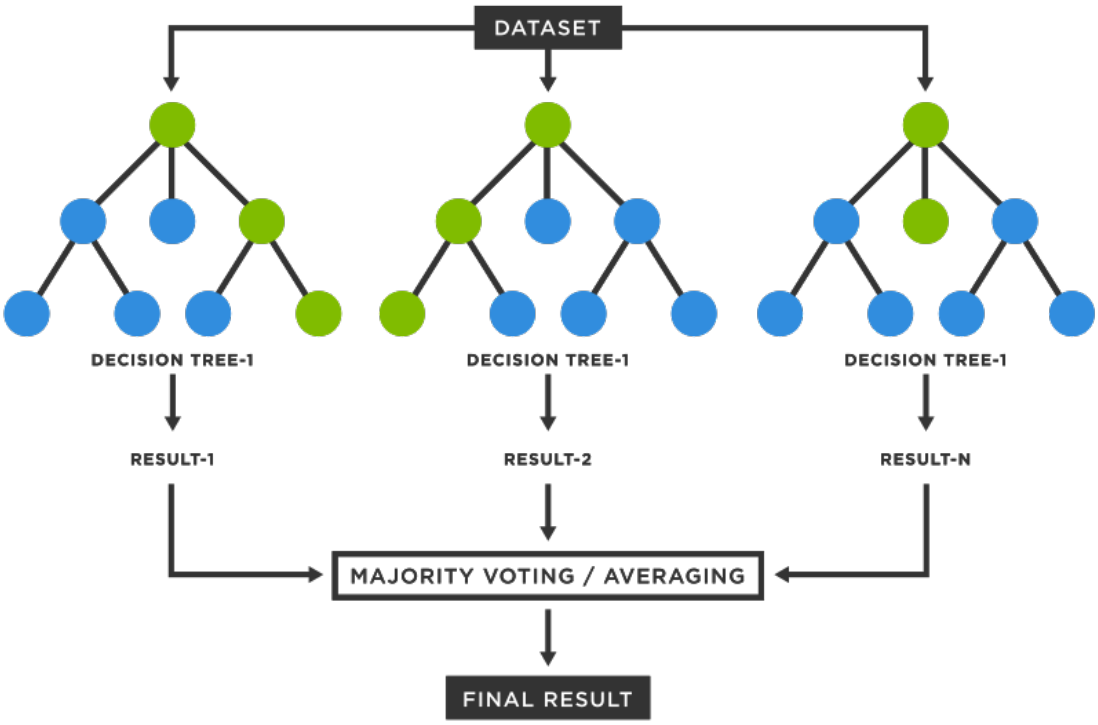

Figura 2.2: Diagrama resumen del funcionamiento del clasificador *Random Forest*

2.1.3. XGBoost (*Extreme Gradient Boosting*)

XGBoost [4] es un algoritmo de aprendizaje automático que integra *Random Forest* junto con la técnica de *Boosting*. Este modelo se basa en la construcción iterativa de árboles de decisión, donde cada árbol se enfoca en corregir los errores del modelo anterior.

Boosting es una técnica de modelado de conjuntos que intenta construir un clasificador fuerte a partir de varios clasificadores débiles. Esto se logra definiendo un único modelo utilizando varios modelos débiles, que a diferencia de la metodología de *Random Forest*, estos son entrenedados secuencialmente y no en paralelo.

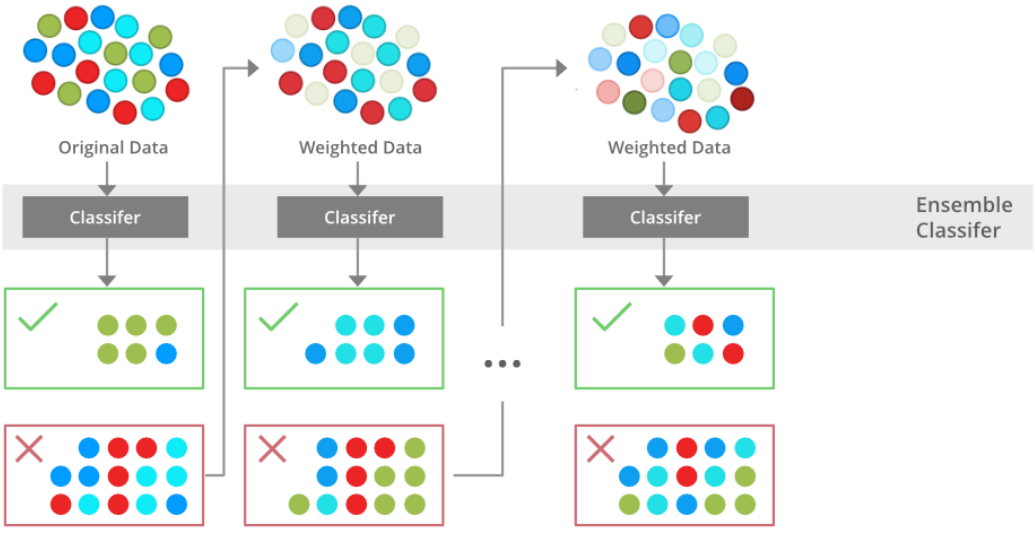

Figura 2.3: Diagrama resumen del funcionamiento del método de ensamblado *Boosting*

Primero, se construye un modelo a partir de los datos de entrenamiento. Luego se construye el segundo modelo, que intenta corregir los errores del modelo anterior. Este procedimiento se repite y se agregan modelos iterativamente hasta que todo el conjunto de datos de entrenamiento sea predicho correctamente, o se llegue a un número máximo de modelos.

En el algoritmo *XGBoost*, estos modelos son justamente árboles de decisión creados de forma secuencial. Los pesos juegan un papel importante, ya que estos se asignan a todas las variables independientes que luego se alimentan a cada árbol de decisión para el entrenamiento. El peso de las variables que son predichas incorrectamente por el árbol van aumentando en cada iteración, y estas variables son luego alimentadas al siguiente árbol de decisión.

Tal como lo indica la figura 2.5, el objetivo principal de *XGBoost* es el cálculo de la predicción de $\mathbf{x}_i$, la cuál puede definirse como:

$$\hat{y}_i = \phi\left(\mathbf{x}_i\right) = \sum_{k=1}^{K} f_k\left(\mathbf{x}_i\right), \quad f_k \in \mathcal{F} \tag{2.4}$$

Donde se define el dataset como $X = x_1, ..., x_n$ con las etiquetas $Y = y_1, ..., y_n$, y cada f_k corresponde a un modelo de árbol de decisión en $\mathcal{F}$, el espacio de modelos de *XGBoost*.

Debido a que cada uno de estos árboles no son independientes entre sí, no es posible calcular directamente este factor, si no que es necesario realizar una optimización de una función objetivo definida en función de cada f_k.

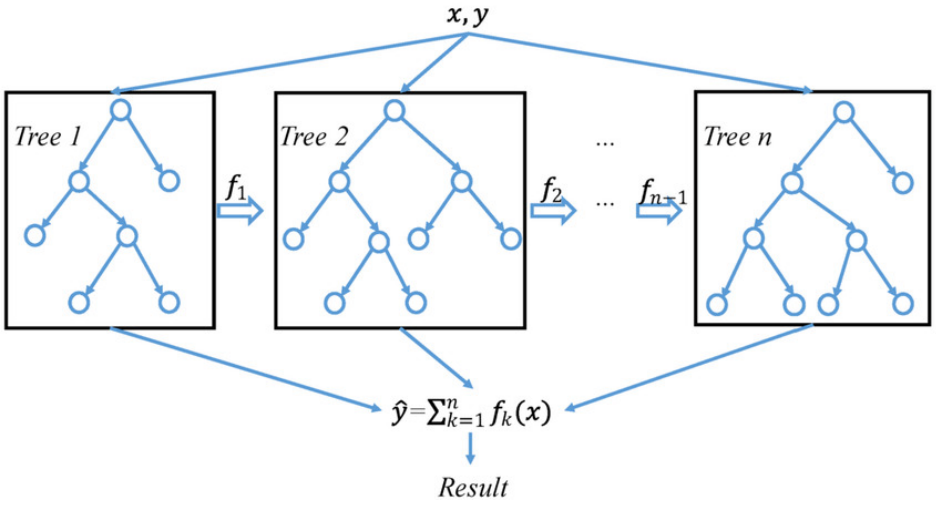

Figura 2.4: Diagrama resumen del funcionamiento del clasificador *XGBoost* [4], donde se observa que cada árbol se entrena secuencialmente.

Así, sea l una función de pérdida diferenciable, K un número de modelos débiles (en este caso *Decision Trees*), α una tasa de aprendizaje, y Ω un término de regularización de la complejidad del modelo. Se puede definir la función objetivo $\mathcal{L}$ como:

$$\begin{aligned} \mathcal{L}(\phi) &= \sum_i l\left(\hat{y}_i, y_i\right) + \sum_k \Omega\left(f_k\right) \\ \text{donde } \Omega(f) &= \gamma T + \frac{1}{2}\lambda \|w\|^2 \end{aligned} \tag{2.5}$$

Debido a que el modelo es entrenado de forma secuencial, se puede definir la función objetivo en términos de la predicción de la iteración t como sigue:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t\left(\mathbf{x}_i\right)\right) + \Omega\left(f_t\right) \tag{2.6}$$

A la cual se le calcula el segundo término de su Serie de Taylor[7] para poder optimizar $\mathcal{L}$ de forma más eficiente, obteniendo:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}^{(t-1)}\right) + g_i f_t\left(\mathbf{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right) \right] + \Omega\left(f_t\right) \tag{2.7}$$

donde $g_i = \partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$ corresponde al cálculo del gradiente, y $h_i = \partial^2_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right)$ corresponde al hessiano. Finalmente se puede obtener la función objetivo simplificada:

[7] Serie de Taylor: aproximación de funciones mediante una serie de potencias (ver Glosario)

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t\left(\mathbf{x}_i\right) + \frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right) \right] + \Omega\left(f_t\right) \tag{2.8}$$

Al expandir Ω en la ecuación 2.8 ya es posible determinar un factor de calidad, el cuál puede ser usado para el entrenamiento del clasificador. Este factor es equivalente a las métricas ocupadas para la división de subconjuntos en los árboles de decisión como la Impureza de Gini (ecuación 2.1, sección 2.1.1), pero derivado de muchas funciones objetivo:

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{2.9}$$

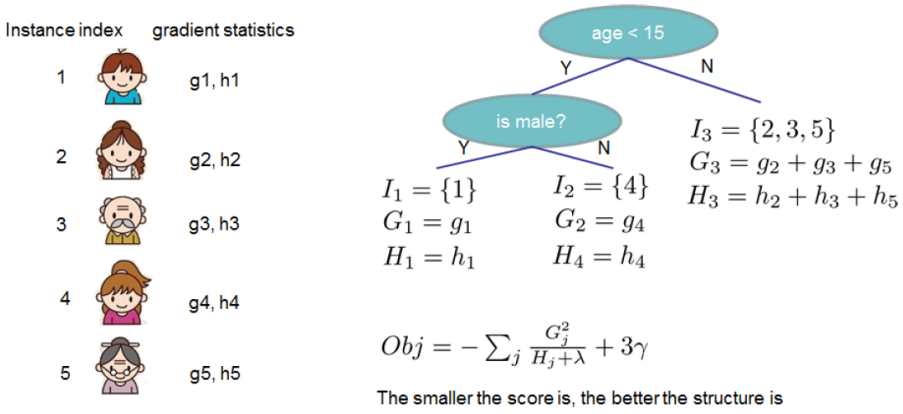

Figura 2.5: Diagrama de ejemplo del cálculo del factor de calidad presentado en el estudio de *XGBoost* [4]. Se puede ver que solo se necesita sumar el gradiente y el hessiano por cada hoja, y luego aplicar la fórmula para obtener el factor.

2.1.4. Transformers

Los *Transformers* [13] son una arquitectura de red neuronal que ha revolucionado el campo del procesamiento del lenguaje natural en los últimos años. Los *Transformers* han demostrado ser altamente efectivos al capturar relaciones de largo alcance en el texto, sin requerir una secuencia ordenada de entrada.

Lo que distingue a los *Transformers* de otras opciones de clasificación de texto es su capacidad para capturar la semántica global del texto. Esto se logra mediante el uso de mecanismos de atención, que permiten a la red enfocarse en partes relevantes del texto durante el proceso de codificación (*encoder*) y decodificación (*decoder*). Al asignar diferentes pesos a las palabras en función de su relevancia contextual, los *Transformers* pueden identificar las conexiones más importantes en el texto y comprender mejor su significado.

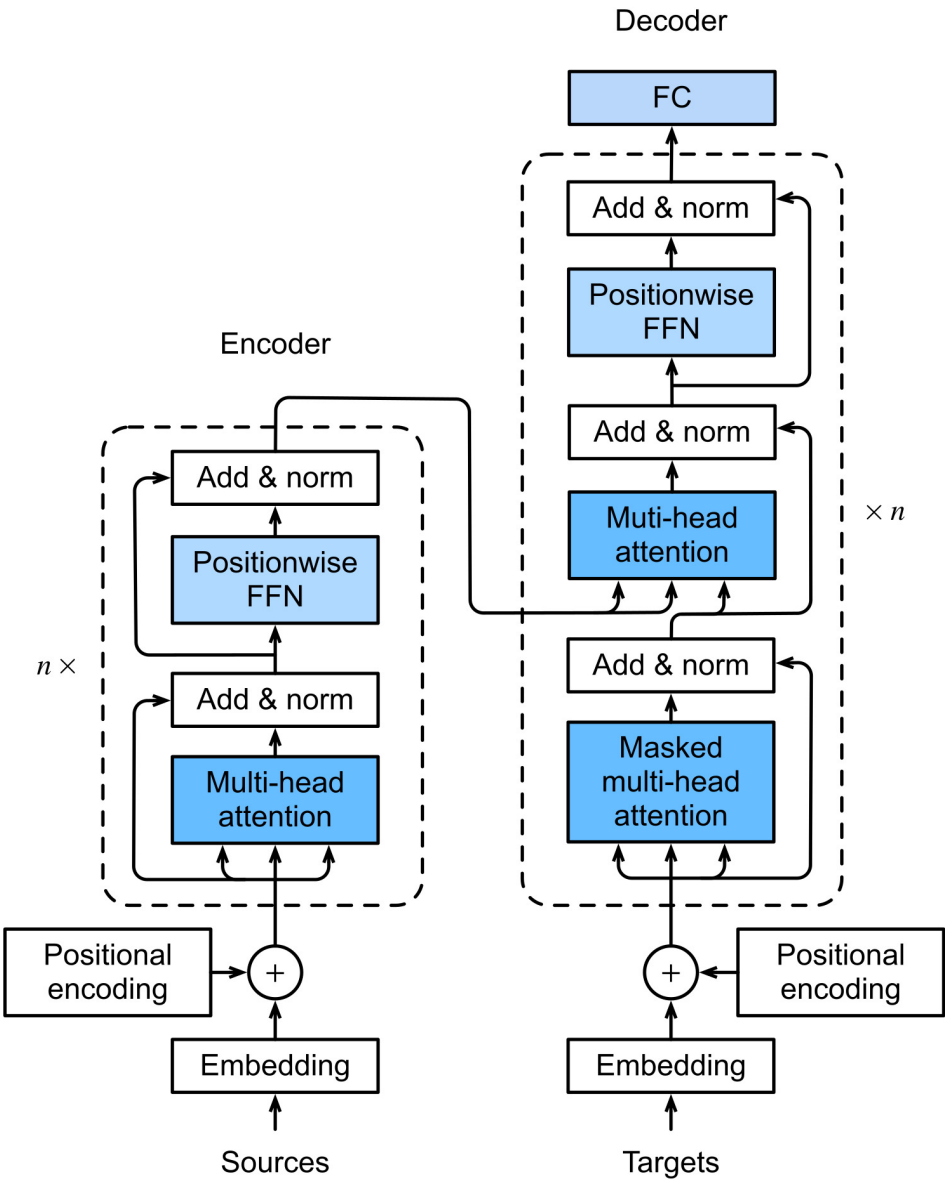

Figura 2.6: Diagrama resumen del funcionamiento de la arquitectura transfomer [13]

El *encoder* toma una secuencia de entrada y la procesa para extraer información útil y representativa. Está compuesto por varias capas idénticas, generalmente llamadas capas de atención o *self-attention*, que trabajan en paralelo para procesar la secuencia de entrada de manera no secuencial. Cada capa de atención en el *encoder* tiene dos subcapas:

1. *Self-Attention Layer*: En esta subcapa cada token de la secuencia de entrada se compara con todas las demás en la misma secuencia para calcular una puntuación de atención. Esta puntuación indica cuánta atención debe prestar el modelo a cada token en relación con las demás. Luego, se pondera la entrada según estas puntuaciones de atención para obtener una representación contextualizada.

2. *Feed-Forward Neural Network*: Después de la subcapa de *self-attention*, las representaciones contextualizadas pasan a través de una red neuronal de avance (*feed-forward*) para aprender representaciones más complejas.

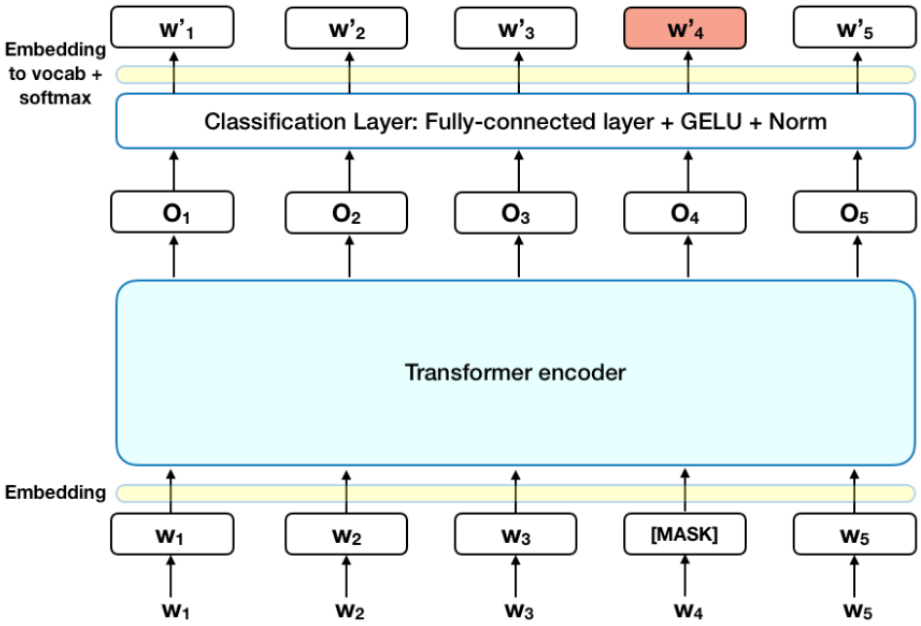

Figura 2.7: Diagrama resumen del funcionamiento el codificador (encoder) de la arquitectura transformer. El input corresponde a una lista de tokens, los cuales son integrados a una serie de vectores que luego son procesados por la red neuronal. El output es una serie de vectores cuyos indices se corresponden con los tokens del input original.

Por otro lado, el *decoder* también consta de múltiples capas de atención, pero su objetivo es generar una secuencia de salida basada en la información contextualizada del *encoder*. Funciona en un modo generativo, donde predice un token a la vez en la secuencia de salida. El *decoder* toma como entrada una secuencia de salida parcial y la procesa para generar palabras adicionales en cada iteración. Al igual que en el *encoder*, cada capa del *decoder* tiene dos subcapas:

1. *Masked Self-Attention Layer*: A diferencia del *encoder*, el *decoder* tiene una subcapa de *self-attention* con máscara que impide que las palabras futuras influyan en las palabras anteriores durante la generación. Esto garantiza que la generación sea autoregresiva, lo que significa que el modelo genera secuencialmente una palabra a la vez basada en las palabras generadas previamente.

2. *Encoder-Decoder Attention Layer*: En esta subcapa, el *decoder* calcula la atención sobre las representaciones contextualizadas del *encoder* para obtener información sobre el contexto relevante de la secuencia de entrada. Esta información ayuda al *decoder* a tomar decisiones más informadas durante la generación.

Esta arquitectura ocupa una herramienta sumamente potente para la clasificación de texto, conocida como *embedding*. Los *word embeddings* son representaciones numéricas que capturan el significado semántico de las palabras en un espacio vectorial. Estas representaciones permiten que los modelos de clasificación de texto comprendan las similitudes y relaciones entre las palabras, lo que mejora la precisión de las predicciones.

2.1.4.1. BERT

BERT (*Bidirectional Encoder Representations from Transformers*) [14] es un modelo de lenguaje basado en *Transformers* desarrollado por Google, y se compone de varias capas de codificadores (*encoders*). Este modelo ha sido revolucionario en el campo del procesamiento del lenguaje natural debido a su capacidad para capturar la semántica y el contexto de las palabras en un texto.

A diferencia de los modelos de lenguaje tradicionales que se entrenaban de manera unidireccional, BERT utiliza un proceso de entrenamiento de lenguaje previo masivo en el que se expone al modelo a grandes cantidades de texto en inglés sin etiquetas. Esto permite que el modelo aprenda representaciones contextuales de las palabras, lo que lo hace muy efectivo en tareas de clasificación de texto y comprensión del lenguaje.

Una de las principales ventajas de BERT es su capacidad de atención bidireccional. Esto significa que el modelo puede tener en cuenta tanto el contexto anterior como el posterior de una palabra al procesarla. Esto es especialmente útil en tareas de comprensión del lenguaje donde el significado de una palabra puede depender del contexto en el que se encuentra. Además, BERT es capaz de capturar relaciones sintácticas y semánticas complejas en el texto, lo que mejora su capacidad para entender y representar la estructura lingüística.

Otra ventaja importante de BERT es su capacidad para transferir el conocimiento aprendido de tareas de lenguaje previo a tareas específicas. Esto se logra mediante un proceso conocido como ajuste fino (*fine tunning*), en el que el modelo se entrena en un conjunto de datos etiquetados específico para la tarea que se desea abordar.

2.1.4.2. BETO

BETO [15] es una variante del modelo BERT específicamente adaptada para el idioma español. Desarrollado por investigadores del Departamento de Ciencias de la Computación de la Universidad de Chile, BETO ha sido entrenado en un gran corpus de textos en español y ha demostrado ser altamente efectivo en tareas de procesamiento del lenguaje natural en este idioma. Al ser una adaptación de BERT, BETO hereda las capacidades de atención bidireccional y la capacidad de capturar contextos y relaciones complejas en el texto.

2.2. Trabajos Relacionados

En los últimos años se han desarrollado diversos trabajos de investigación cuyo fin es la caracterización de postura en base a datos de texto, los que van desde metodologías de clasificación, o también estudios enfocados a algún público y tópico específicos. La mayoría de estos trabajos se han desarrollado bajo poblaciones angloparlantes, pero como se mencionó anteriormente, también se han hecho investigaciones en nuestro país.

2.2.1. 'Multilingual stance detection in social media political debates'

El artículo *'Multilingual stance detection in social media political debates'* [16] se enfoca en el desarrollo de un modelo avanzado para la detección de postura en debates políticos en redes sociales, desarrollado por académicos de la Università degli Studi di Torino, Italia. Los autores propusieron un modelo llamado *MultiTACOS*, que es una extensión de un modelo anterior desarrollado por los mismos autores el año 2016.

El modelo predecesor de *MultiTACOS* (denominado *iTACOS* [17]) utilizaba principalmente un clasificador probabilístico Naive Bayes para la detección de postura. El *dataset* bajo estudio correspondía a la discusión presidencial estadounidense del año 2016 (Hillary Clinton contra Donald Trump), demostrado ser altamente exitoso al coincidir con resultados de diversas encuestas de opinión de ese entonces.

Este modelo fue entrenado utilizando *metadata* de los usuarios que interfirieron en el debate presidencial y denominados *Features Estructurales*, correspondientes a sus nombres de usuario, descripciones, y por supuesto sus *tweets*. Además, se incluyeron datos de interacciones sociales denominados *Features Contextuales*, los cuales incluyen datos de los *retweets* y *quotes* para analizar su impacto en la detección de postura dentro del grupo de estudio.

Basándose en el éxito de *iTACOS*, los autores propusieron desarrollar una versión mejorada y ampliada del modelo. El enfoque principal de este último fue expandir el modelo original para funcionar en múltiples idiomas, recopilando recursos relacionados con temas políticos en inglés, francés, italiano, español y catalán. Esto incluyó la creación de nuevos corpus de texto, así como el uso de corpus de referencia existentes en tareas de detección de postura y evaluación de desempeño ampliamente conocidos en el área.

Si bien el trabajo *'Multilingual stance detection'* [16] no es de los primeras propuestas de detección de postura, si es una de las primeras en ocupar modelos de clasificación basados en tokens. Esta investigación ha demostrado el éxito de este tipo de sistemas, y su posible adaptabilidad a datos de nuestro propio país.

2.2.2. 'DeBot: Twitter Bot Detection via Warped Correlation'

El estudio *'DeBot: Twitter Bot Detection via Warped Correlation'* [7] fue llevado a cabo por académicos de la Universidad de Nuevo México en el año 2016. Su objetivo principal era detectar *bots* en *Twitter* mediante la identificación de cuentas altamente correlacionadas entre sí. Estas cuentas exhibían comportamientos similares en términos de diferencias temporales en sus publicaciones y contenido.

La premisa del estudio se basaba en la idea de que los usuarios auténticos en una red social no podrían mantener comportamientos similares durante períodos de tiempo extensos. Esta premisa se relaciona directamente con la noción de que los *bots* tienden a carecer de comportamientos orgánicos en comparación con los usuarios reales, lo que permitiría detectarlos al analizar su comportamiento dentro de un intervalo de tiempo definido.

Para lograr este objetivo, los investigadores propusieron un sistema que busca identificar grupos de usuarios altamente correlacionados en un intervalo de tiempo. Introdujeron un parámetro de correlación denominado *'warped correlation'*, el cual mide el grado de correlación y la frecuencia de publicación de los usuarios dentro del período de tiempo especificado.

Utilizando técnicas de aprendizaje no supervisado como *clustering*, se pudieron detectar y analizar grupos de usuarios con coeficientes de correlación similares. Los resultados obtenidos fueron altamente prometedores, alcanzando una precisión de hasta el 99.5 % en la detección de grupos de usuarios con comportamientos sincronizados.

2.2.3. 'Online Human-Bot Interactions: Detection, Estimation, and Characterization'

El artículo *'Online Human-Bot Interactions: Detection, Estimation, and Characterization'* [8] fue escrito por académicos de la Universidad de Indiana y la Universidad del Sur de California en 2017. Su objetivo principal era detectar *bots* mediante el estudio y análisis del comportamiento de los usuarios en la red social *Twitter*.

A diferencia del estudio anterior *'DeBot'*, la metodología utilizada en este artículo se adentró en las diferentes características que conforman un usuario dentro de la red social más allá de las posibles correlaciones temporales entre dos o más individuos. La premisa indicaba que no solo se podrían detectar usuarios artificiales mediante correlaciones entre sí, si no que estos presentan ciertas características que podrían identificarlos del resto.

Entre las características utilizadas para la detección de *bots* se incluyen los amigos de un usuario, los patrones de conexiones en la red, la actividad en series de tiempo y, por supuesto,

los tweets de cada usuario. Cada una de estas características fue procesada y utilizada para agrupar a los usuarios con parámetros similares mediante el uso de la técnica de aprendizaje no supervisado *clustering*.

El estudio analizó las interacciones en línea entre humanos y *bots*, con el objetivo de detectar patrones de comportamiento y distinguir entre cuentas auténticas y cuentas automatizadas. Se exploraron diferentes enfoques y técnicas, incluyendo el análisis de las características de los usuarios y la detección de anomalías en su comportamiento.

El uso del aprendizaje no supervisado y las técnicas de *clustering* permitieron identificar grupos de usuarios con comportamientos similares pero no necesariamente correlacionados, lo que ayudó a distinguir entre usuarios humanos y *bots*. Estos hallazgos proporcionaron una mejor comprensión de las interacciones en línea y contribuyeron al desarrollo de estrategias más efectivas para detectar y caracterizar la presencia de *bots* en *Twitter*.

2.2.4. 'Every Colour You Are: Stance Prediction and Turnaround in Controversial Issues'

Por otro lado, se tiene la publicación *'Every Colour You Are: Stance Prediction and Turnaround in Controversial Issues'* [3] desarrollada por los académicos Eduardo Graells-Garrido (Universidad de Pompeu Fabra), Ricardo Baeza-Yates (Universidad de Waterloo) y Mounia Lalmas (Universidad de Glasgow).

El objetivo principal de este estudio fue comprender y predecir la postura de los usuarios en relación con temas controversiales en la red social *Twitter*. El tópico en estudio fue la discusión del aborto en Chile y Argentina, enfocándose en analizar cómo los usuarios expresan su postura y cómo esta puede cambiar a lo largo del tiempo. Esto último lo convirtió en uno de los primeros trabajos en investigar la detección de postura en un contexto latino.

Para lograr esto, se ocupó el modelo de clasificación basado en árboles de decisión *XGBoost* (sección 2.1.3). Para lo anterior, se procesaron los *tweets* de los usuarios y se extrajeron características relevantes como el contenido textual y la información asociada con cada *tweet*, sus usuarios, y también las interacciones que estos tengan con el resto. Este procesamiento genera una serie de matrices que relacionan a cada usuario con los *tokens* que haya ocupado en su discurso, y también matrices que indican las interacciones entre los usuarios.

Además, si bien es posible obtener una gran cantidad de datos de los usuarios involucrados en el estudio, tanto el lugar de procedencia de cada uno de ellos como el género del usuario no es un parámetro obligatorio dentro de cada perfil. Por lo anterior, solo una pequeña parte de la fuente de datos principal de *Tsundoku* contiene este dato demográfico, llevando a incluir estas características al proceso de clasificación del sistema. Estas métricas podrían ser

relevantes para el estudio y segmentación de poblaciones al analizar los resultados obtenidos.

Es importante tener en cuenta que la información recopilada de *Twitter* puede estar sesgada a la población real, por lo que este sesgo fue medido y estudiado en otro trabajo relacionado realizado bajo el mismo tópico de discusión [18]. Sin embargo, los resultados obtenidos en este estudio fueron muy interesantes y proporcionaron una comprensión más profunda de la discusión sobre el tema del aborto en dichos países.

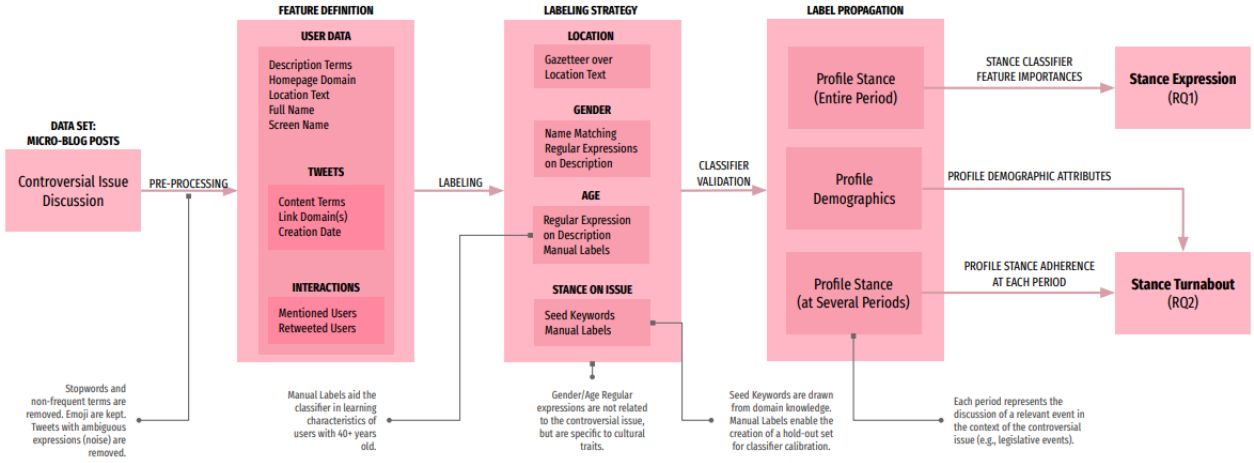

Figura 2.8: Diagrama de la metodología ocupada para el trabajo *'Every Colour You Are'* [3]

El diagrama de la ilustración 2.8 muestra de manera resumida cómo funciona el la arquitectura del sistema desarrollado en este trabajo. Este servirá como una referencia importante para la documentación y el entendimiento del sistema *Tsundoku* a lo largo de este trabajo, proporcionando una visión general de su funcionamiento y componentes principales.

2.2.5. 'Bots don't vote, but they surely bother!: A Study of Anomalous Accounts in a National Referendum'

El día 18 de Octubre del año 2019 en Santiago de Chile se presenció el llamado *'Estallido Social'*, lo que correspondió a una gran cantidad de manifestaciones a lo largo del país exigiendo equidad y dignidad en demandas tanto sociales como económicas. Este evento fue un hecho histórico en la sociedad chilena, y llevó a una discusión profunda en torno a la escritura de una nueva constitución.

Como resultado de esta discusión, el 25 de octubre de 2020 se llevó a cabo un referéndum constitucional sin precedentes en la historia de Chile. Este plebiscito generó intensas conversaciones en las redes sociales, caracterizadas por la desinformación y la polarización política extrema.

Con el objetivo de caracterizar a los usuarios involucrados en esta discusión en la plataforma de microblogging *Twitter*, se realizó el estudio *'Bots don't vote, but they surely bother'*, llevado a cabo por los mismos académicos que desarrollaron el trabajo mencionado anteriormente, *'Every Colour You Are'* (sección 2.2.4).

2.2.5.1. Objetivos y Metodología

El objetivo principal de *'Bots don't vote'* [6] fue caracterizar a los usuarios y la discusión en torno al plebiscito de 2020 en Chile, ocupando atributos demográficos como el vocabulario ocupado en la discusión. Además, se incorporó la detección de *bots* en la plataforma para determinar el impacto de este tipo de usuarios en la discusión y analizar cómo afectaron las interacciones en línea.

Para lograr esto, se amplió el sistema de caracterización desarrollado en el trabajo *'Every Colour You Are'* mediante la incorporación de técnicas de detección de anomalías. La premisa de este enfoque es similar a la del trabajo *'Online Human-Bot Interactions'* (sección 2.2.3), que propone identificar usuarios artificiales mediante la detección de características anómalas más allá de las correlaciones con otros usuarios.

Así nació *Tsundoku*, un sistema de caracterización de discusiones en *Twitter* que permite detectar la postura de los usuarios en relación con un tema de discusión y, a su vez, identificar usuarios anómalos dentro de esa discusión. Este trabajo fue desarrollado en lenguaje python, y los datos fueron recopilados por el mismo profesor Eduardo Graells.

El clasificador utilizado en *Tsundoku*, al igual que en *'Every Colour You Are'*, se basó en el modelo de clasificación *XGBoost* 2.1.3. Por otro lado, la detección de *bots* se realizó mediante el entrenamiento de un modelo conocido como *Isolation Forest* [9, 10], que se utiliza para la detección de anomalías.

El desarrollo de *Tsundoku* permitió un análisis más profundo de la discusión en *Twitter* relacionada con el plebiscito de 2020, y brindó una herramienta para identificar patrones de comportamiento de usuarios y la presencia de *bots* en la plataforma.

2.2.5.2. Resultados e Impacto

La aplicación de este sistema permitió identificar y separar de manera efectiva a los *bots* en el contexto de la discusión constitucional chilena en *Twitter*. Este estudio reveló que si bien la cantidad de *bots* dentro de la discusión constitucional en Chile era baja, su efecto y acción coordinada pudieron influir en el debate digital.

EL MERCURIO

Vida • Ciencia • Tecnología

vct@mercurio.cl @VCT_ElMercurio SANTIAGO DE CHILE, VIERNES 23 DE OCTUBRE DE 2020

Análisis de datos en más de 200 mil cuentas y 5 millones de tuiteos, entre agosto y septiembre:

Twitter da cuenta de un debate polarizado y grupos que no hablan entre sí frente al plebiscito

Usan palabras propias en sus mensajes, incluso se han "apropiado" de emojis que los representan. En el estudio hay dos comunidades que intercambian mensajes entre sus miembros con profusión, pero que no se relacionan con el otro segmento.

ALEXIS IBARRA O.

"Este estudio confirma lo que denuncia el documental 'El dilema de las redes sociales' y es que estas plataformas crean burbujas de información y que las personas que están en ellas creen que lo que leen ahí es la única verdad. En Chile hay una burbuja del Rechazo y otra del Apruebo, que no hablan entre sí, que se comparten información entre ellos, pero no con el otro grupo y, por lo tanto, no ven la realidad del otro lado", dice Ricardo Baeza-Yates, investigador del Instituto Milenio de Fundamento de los Datos y uno de los autores de la investigación sobre la actividad relacionada con el plebiscito en Twitter.

Este estudio tuvo como antecedente uno que analizó la discusión sobre las distintas posiciones sobre el aborto en Chile y Argentina, y otro similar sobre el estallido social del 18-O.

Eduardo Graells-Garrido, el otro autor de esta investigación, explica que desde 2015 tienen un *software* "robot" que "oye" lo que pasa en Twitter en relación con Chile.

"Para este estudio creamos un filtro que permitiera identificar mensajes que tuvieran que ver con el plebiscito. Encontramos que 500 mil cuentas hablaban de ello, pero había un referéndum en Italia que creaba ruido. Tras dejar solo las de Chile y eliminar las cuentas institucionales y las de parodia se llegó a 200 mil cuentas y a 5 millones de tuits".

De ese universo de 200 mil cuentas, el grupo más activo representa al 12% del total. "Son las cuentas que más comparten o las más citadas. Podríamos decir que corresponde a la élite de Twitter", dice Baeza-Yates.

Al hacer un *zoom*, se dieron cuenta de que solo el 5% de las 200 mil cuentas genera el 35% de los retuits relacionados con el plebiscito. Y si bien hay más cuentas relacionadas con el Apruebo que con el Rechazo, la actividad de ambos grupos se tiende a equiparar.

"Esto sucede porque las cuentas relacionadas con el Rechazo son 3,7 veces más activas en la cantidad de tuits que las del Apruebo", aclara Baeza-Yates.

Hashtag y emoticón

Entre los 50 términos más usados por ambas comunidades no hay ninguno en común.

Las palabras más utilizadas por los partidarios del Rechazo fueron "políticos de derecha", "chilenos", "patriotas" y "libertad". También hacen referencia a Argentina, Venezuela y otros términos, como "violencia", "terrorismo" y "destruir". Para referirse a otros usan palabras como "delincuentes", "zurdos" o "comunistas".

"En los perfiles vimos que la mayoría de los partidarios del Rechazo se identifica con el *emoji* de una bandera chilena al punto que aparece en la nube de palabras", agrega Graells-Garrido.

En el Apruebo, en tanto, las palabras más usadas fueron términos como "pueblo", "dignidad", "carabineros", "UDI", "Pinochet" y "dictadura". Además "hablan" de la franja electoral, de la participación el 25 de octubre y del segundo voto con el *hashtag* #ConvencionConstitucional (ver infografía).

Por su parte, el Apruebo usa el emoji de un payaso para referirse a los políticos, dice Graells-Garrido.

Otra particularidad es que en el análisis se vio que el Rechazo tenía referentes que eran muy retuiteados, como Sergio Melnick y José Antonio Kast. "Esos referentes no se ven en la comunidad del Apruebo. Hay algunos como Boric, Jackson y Vallejos, pero no se acercan al nivel de Kast o Melnick. Los liderazgos están más diseminados", dice Graells-Garrido.

En bloque

En ese sentido, el investigador dice que las subcomunidades dentro del Apruebo son más pequeñas, en cambio en el Rechazo hay menos subcomunidades, pero son más grandes. "Eso podría interpretarse en que en el Rechazo se retuitean más veces los mismos mensajes, provenientes de menos personas y se actúa en bloque. En el Apruebo, los mensajes y los referentes son más diversos y por eso hay más subcomunidades".

En los últimos años, el equipo ha generado herramientas que permiten identificar si una cuenta es de un hombre o una mujer, el lugar geográfico donde está el usuario, o su rango de edad.

"Esto es importante, porque nos permite nivelar la representación de algunos grupos en Twitter, ya que sabemos que hay más hombres, más personas de un nivel socioeconómico alto, con mayor educación. En tanto, personas de edad y de zonas rurales están subrepresentadas. Así, podemos ajustar la muestra a los datos del Censo", aclara Baeza-Yates.

Dos burbujas que no conversan

El gráfico de redes muestra los retuits entre personas que "hablaron" sobre el plebiscito entre agosto y septiembre. La imagen identifica dos grupos claramente definidos y que tiene escasos puntos de interrelación. En la nube de palabras el tamaño es proporcional a la frecuencia de uso.

Fuente Eduardo Graells-Garrido y Ricardo Baeza-Yates. EL MERCURIO

Investigadores

RICARDO BAEZA-YATES
Es director de los programas de posgrado en Ciencias de Datos de Northeastern University (Silicon Valley). Es académico del DCC de la U. de Chile e investigador *senior* del Instituto Milenio de Fundamento de los Datos. Trabajó en Yahoo! Labs desde 2006 a 2015, año en que dirigió Yahoo! Labs Londres. Su área de investigación son los sesgos algorítmicos, ciencia y visualización de datos, entre otros.

EDUARDO GRAELLS-GARRIDO
Trabaja en el Grupo de Visualización Científica en el Barcelona Supercomputing Center (BSC). Participa en el proyecto IoTwins en colaboración con el FC Barcelona. Es investigador en el Instituto de Data Science de la UDD. Sus temas de investigación son informática urbana, ciencia social computacional y visualización de información.

Figura 2.9: Artículo publicado en el diario *'El Mercurio'* el día 23 de Octubre de 2020 respecto al trabajo *'Bots don't vote'* [6], donde se destacan los resultados obtenidos en cuanto a la detección de *bots*, y al debate polarizado dentro de la discusión del referendum constitucional

Esto llevó a la publicación de los resultados obtenidos en el diario *'El Mercurio'* (Ilustración 2.9) el día 23 de Octubre de 2020, junto con la publicación de un artículo en el Instituto de Ética de Inteligencia Artificial de Montreal [19] dos años más tarde.

Los resultados obtenidos resultaron en una muy buena aproximación al resultado final del plebiscito del día 25 de octubre [20], en donde se detectó que el 80.78 % de los usuarios estaban a favor de la nueva constitución, mientras que en los resultados reales del plebiscito esta opción resultó ganadora con el 78.81 % de los votos.

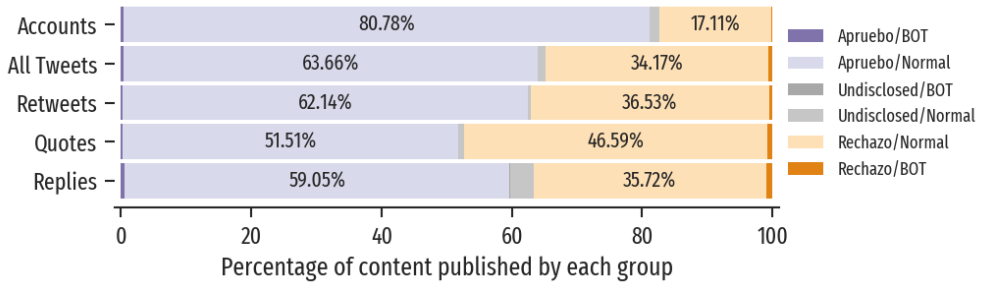

Figura 2.10: Resultados del análisis de la discusión en torno a una nueva constitución de *'Bots don't vote'* [6], donde se observa que el 80.71 % de los usuarios involucrados fueron clasificados con la postura *Apruebo*

Figura 2.11: Resultados oficiales informados por el Servicio Electoral de Chile del plebiscito constitucional del año 2020 [20]

La implementación del sistema *Tsundoku* se encuentra en la plataforma de control de versiones Github [12], y corresponde al código base para el desarrollo de este trabajo.

2.2.5.3. Limitaciones

Si bien el trabajo *'Bots Don't Vote'* [6] logró estudiar el impacto que generaron los *bots* a la discusión constitucional, este presenta algunas limitantes.

Por un lado, el proceso de extracción de datos para este estudio ocupa la *API* v1.1 de *Twitter*, que impone restricciones significativas en la cantidad de *tweets* obtenidos. Por consiguiente, fue necesario desarrollar un programa que solicite constantemente datos a la plataforma, permitiendo distinguir entre *tweets* previamente obtenidos y aquellos que se obtienen de forma continua. Además, se ha enfocado en recopilar datos exclusivamente de Chile, con el objetivo de realizar estudios centrados en nuestro país en esta plataforma.

El profesor Eduardo Graells ha desempeñado un papel fundamental en el desarrollo del proceso de obtención de datos, lo que ha llevado a la acumulación de una gran cantidad de información que abarca desde principios de 2020 hasta finales de 2022. En promedio, se recopilan un millón de *tweets* diarios, los cuales se almacenan continuamente utilizando el formato de *JavaScript Object Notation* (*.json*), y que corresponden a la fuente de datos principal del sistema *Tsundoku*.

Sin embargo, esta situación ha generado dificultades en el manejo de grandes volúmenes de datos internos del sistema. Si bien los archivos *JSON* son ampliamente utilizados para el intercambio de datos debido a su formato legible por humanos y su amplio soporte en diferentes lenguajes de programación, cuando se trata de manejar grandes cantidades de datos, los archivos *JSON* pueden tener algunas limitaciones:

En primer lugar, los archivos *JSON* pueden volverse bastante grandes cuando se almacena una gran cantidad de datos. Esto puede ocupar mucho espacio en disco y requerir más tiempo para leer o escribir el archivo completo.

En segundo lugar, y a medida que aumenta el tamaño del archivo, el rendimiento de las operaciones de lectura y escritura puede disminuir significativamente. Esto se debe a que se necesita tiempo para analizar y procesar todo el archivo, lo cual puede ser costoso en términos de tiempo de ejecución. Además, al leer un archivo *JSON* generalmente se carga todo el contenido en la memoria. Esto puede ser un problema si la cantidad de datos es extremadamente grande y supera la capacidad de memoria disponible, lo que podría provocar errores o ralentizaciones en la aplicación.

Finalmente, para las operaciones de búsqueda o acceso a datos específicos dentro de un archivo *JSON* grande se podría requerir un procesamiento adicional para encontrar y extraer los datos relevantes. Esto puede ser ineficiente en comparación con otras estructuras de datos diseñadas específicamente para operaciones de búsqueda eficientes, como las bases de datos o ficheros *Parquet*.

El uso de archivos no óptimos para el procesamiento de datos a gran escala ha ralentizado considerablemente las operaciones y la generación de *dataframes*, lo cual constituye el principal desafío a enfrentar. Por tanto, se presenta la oportunidad de optimizar y actualizar el manejo de *dataframes* de datos en *Tsundoku*, permitiendo la integración de herramientas

especializadas en el procesamiento masivo de datos, manteniendo el funcionamiento actual del toolkit y mejorando la usabilidad para futuros usuarios.

Por otro lado, se ha observado que el rendimiento de algunos clasificadores de atributos demográficos en *Tsundoku* no alcanza las expectativas en comparación con otros. Específicamente, se han obtenido resultados insatisfactorios en el clasificador relacionado con el lugar de procedencia de los usuarios.

Tabla 2.1: Métricas de evaluación del clasificador XGBoost usando validación cruzada, grupo de estudio *location*

	precision	recall	f1-score
antofagasta	0.13	0.31	0.18
araucania	0.12	0.18	0.14
arica	0.01	0.04	0.01
atacama	0.00	0.00	0.00
aysen	0.00	0.00	0.00
biobio	0.03	0.07	0.04
coquimbo	0.01	0.04	0.02
loslagos	0.02	0.06	0.03
losrios	0.03	0.06	0.04
magallanes	0.00	0.00	0.00
maule	0.03	0.07	0.04
noise	0.76	0.53	0.59
nuble	0.00	0.00	0.00
ohiggins	0.01	0.02	0.01
rm	0.55	0.19	0.28
tarapaca	0.17	0.31	0.21
valparaiso	0.10	0.15	0.12
accuracy			0.23
macro avg	0.12	0.12	0.10
weighted avg	0.42	0.23	0.27

Aunque la precisión de la clasificación de postura dentro del sistema es excepcionalmente alta, alcanzando una precisión global del 99 % con un promedio ponderado también de 99 %, el clasificador de lugar de procedencia ha mostrado un rendimiento deficiente en todas las categorías o regiones del país.

El porcentaje de precisión global del clasificador con este grupo de estudio es del 23 % con un promedio ponderado de 27 %, siendo las categorías de ruido (*noise*) con un *f1-score* de

0.59 y las detecciones de la región metropolitana con 0.28 las que presentan los resultados más favorables.

Este bajo rendimiento del clasificador abre la oportunidad de mejora, permitiendo la integración de más datos de los usuarios o el uso de herramientas de procesamiento de lenguaje natural adicionales. Actualmente, el funcionamiento de *Tsundoku* se basa en un clasificador de árboles de decisión, que si bien es rápido en términos de entrenamiento y ejecución, limita el procesamiento de texto a nivel de tokens.

El clasificador recibe como entrada una serie de matrices que contienen información de las interacciones de los usuarios y el vocabulario utilizado en el conjunto de datos de estudio, y hasta el momento no se ha incorporado ninguna tecnología o herramienta de procesamiento de lenguaje natural basada en contexto. La integración de una nueva herramienta de *PLN* podría conducir a mejores resultados en el proceso de caracterización de los usuarios, e incluso permitir la exploración de nuevas preguntas de investigación.

2.3. Herramientas de Procesamiento Masivo de Datos

El principal objetivo de este proyecto es implementar herramientas de procesamiento masivo de datos para mejorar el sistema *Tsundoku* (sección 2.2.5). Actualmente, el sistema se maneja utilizando únicamente librerías como *Dask* [21] y *Pandas* [22] para el procesamiento de tablas. Sin embargo, utilizar una librería especializada en el manejo de datos a gran escala podría significar una mejora sustancial en los tiempos de ejecución.

Una de las primeras herramientas evaluadas para mejorar el sistema fue la implantación de la librería *PySpark*, que proporciona una API de *Python* para *Apache Spark*. Esto permitiría realizar procesamientos en tiempo real a gran escala, aprovechando sistemas distribuidos y mejorando significativamente la eficiencia de los cálculos.

Sin embargo, surgió el problema de decidir dónde se implantaría este sistema, ya sea localmente o en un servidor en el futuro. Esto plantea desafíos en términos de adaptabilidad, ya que la implementación de *PySpark* es más compleja y no se ha considerado en los planes futuros de *Tsundoku*. Además, desplegarlo en un servicio externo sería más complejo que utilizar un sistema centralizado.

Ante esta situación se exploraron nuevas oportunidades, y una de las alternativas destacadas es la librería *PyArrow*. Esta librería ofrece una forma eficiente de transportar y procesar datos en memoria a través de diferentes sistemas, lo que podría ser beneficioso para el procesamiento masivo de datos en *Tsundoku*. Su integración podría proporcionar mejoras significativas en la velocidad y escalabilidad del sistema, sin los desafíos adicionales asociados con *PySpark*.

2.3.1. PyArrow: API de Python para Apache Arrow

Como se mencionó previamente, *PyArrow* [23] es una librería y API de *Python* que permite el uso de *Apache Arrow*. Esta librería se destaca por su alto rendimiento en operaciones de procesamiento y almacenamiento de datos, gracias a su utilización de estructuras de datos y algoritmos optimizados que aceleran el procesamiento y reducen la latencia.

Entre las principales ventajas de *PyArrow* se encuentra su integración con el ecosistema de *Python*. Está diseñada para trabajar de manera transparente con otras bibliotecas y herramientas populares como *Pandas*, *NumPy* y *scikit-learn* [24], permitiendo realizar análisis y manipulación de datos de manera conjunta.

Además, *PyArrow* ofrece soporte para diversos formatos de datos, lo que facilita su interoperabilidad con otras herramientas y sistemas sin depender de librerías externas para la lectura de archivos. Es capaz de leer y escribir datos en formatos como *Parquet*, *Arrow IPC*, *CSV*, *JSON*, entre otros, lo cual es especialmente relevante para las necesidades actuales del sistema.

En particular, el formato de archivo más recomendado por *PyArrow* es *Parquet*. Los archivos *Parquet* adoptan un enfoque columnar, lo que significa que organizan y almacenan los datos en columnas en lugar de filas. Esta estructura columnar permite una compresión más eficiente y un uso más óptimo del almacenamiento. Además, los datos similares se agrupan juntos, lo que reduce el tamaño total del archivo y permite una lectura más rápida de columnas específicas sin tener que leer todo el archivo.

La naturaleza columnar de *Parquet* también ofrece un procesamiento eficiente. Debido a esta estructura, los archivos *Parquet* son especialmente adecuados para consultas y análisis que involucran operaciones en columnas específicas. Al leer solo las columnas necesarias en lugar de todo el archivo, se logran mejoras significativas en el rendimiento y la velocidad de procesamiento.

Capítulo 3

Diseño de la Solución

El sistema *Tsundoku* sigue la metodología descrita en el trabajo *'Bots Don't Vote'* [6], mencionado anteriormente en la sección 2.2.5. Debido a que el objetivo principal de este trabajo es la optimización y mejoras del mismo, el diseño de la solución corresponde a la arquitectura actual del sistema con la incorporación de las actualizaciones necesarias a la misma. Es importante mencionar que actualmente no existe documentación del funcionamiento de *Tsundoku* además del trabajo *'Bots Don't Vote'*, por lo que este capítulo busca registrar y explicar detalladamente su funcionamiento y arquitectura

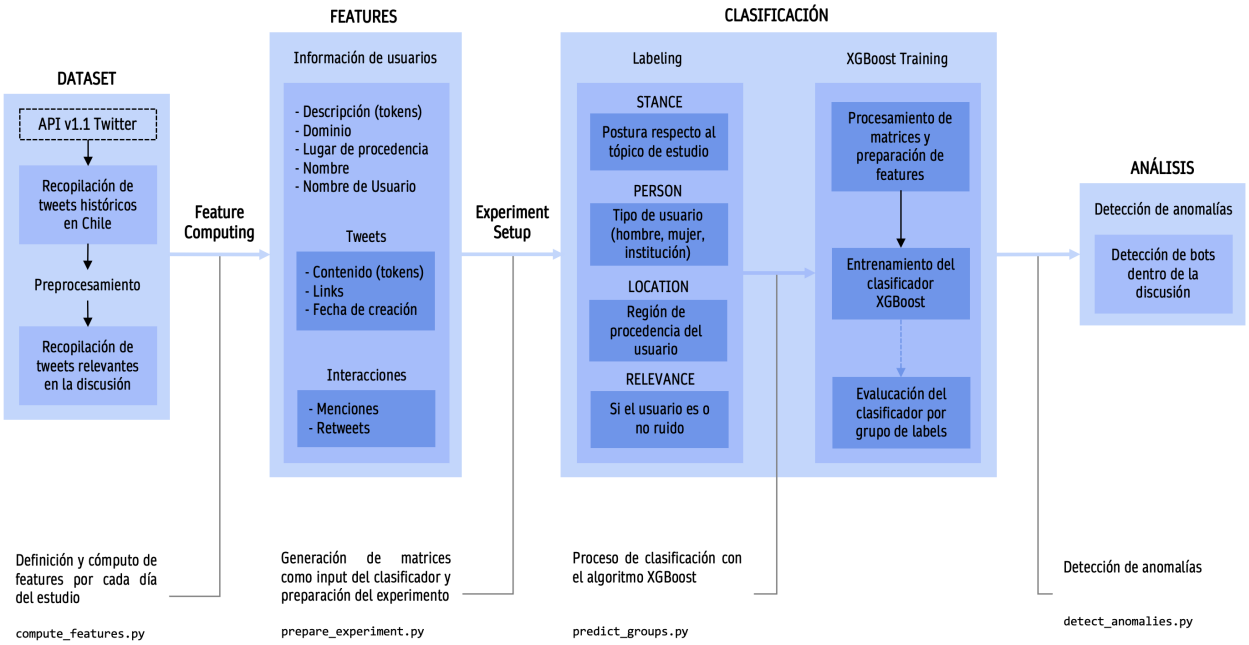

Figura 3.1: Diagrama resumen del diseño del sistema *Tsundoku* [6, 12]

El funcionamiento en detalle de toda la estructura y diseño de *Tsundoku* se explicará a lo largo de este capítulo, indicando en cada momento qué tipo de dato se está manejando y con

qué motivación. De cualquier modo, es posible segmentar el sistema en cuatro componentes claves, donde en cada uno se delega una funcionalidad base del manejo y cómputo del sistema:

1. El primer componente se encarga del manejo del dataset y el preprocesamiento de los datos para el estudio en particular. Su objetivo principal es filtrar el conjunto de *tweets* históricos recopilados por el profesor Eduardo Graells Garrido, adaptándolos al estudio específico que se desea realizar.

2. El segundo componente abarca la definición y el cálculo de las características relevantes de los usuarios para la clasificación. Su propósito es extraer la información más importante de cada día del experimento y transformarla en *dataframes* y matrices fáciles de manejar para el proceso de clasificación.

3. El tercer componente se refiere al proceso de clasificación, que implica la generación de entradas para el modelo y su entrenamiento. Para este propósito, se utiliza una implementación del algoritmo *XGBoost* 2.1.3, que se basa en árboles de decisión secuenciales y potenciación de gradiente.

4. Por último, el cuarto componente del sistema se centra en la consolidación y el análisis de los resultados, destacando el proceso de detección de bots dentro del grupo de estudio.

Antes de estudiar cada componente en detalle, primero debemos comprender el cómo funciona la configuración del sistema, y así mismo, la configuración del tópico en estudio que se quiera realizar con *Tsundoku*.

3.1. Configuración del sistema

El proceso de configuración de *Tsundoku* se realiza mediante archivos en formato *toml* (*Tom's Obvious Minimal Language*) para definir constantes utilizadas durante la compilación del sistema. Estos archivos recopilan variables de entorno importantes, como los directorios de origen de los *tweets* y los directorios donde se almacenarán los archivos generados durante la ejecución. También incluyen variables para experimentos y grupos utilizados en la clasificación, incluyendo las palabras clave relacionadas con el estudio.

Primero, la configuración de los directorios locales que contienen la base de datos de *tweets* recopilados históricamente, así como el directorio donde se almacenan los *dataframes* y archivos generados por *Tsundoku* durante su ejecución, se encuentran en el archivo de configuración principal `config.toml` (ver Anexo A.1). En este archivo también se definen variables como el nombre del proyecto, el archivo que contiene los términos relevantes del estudio, y también el archivo que contiene las *stopwords* a considerar para el preprocesamiento.

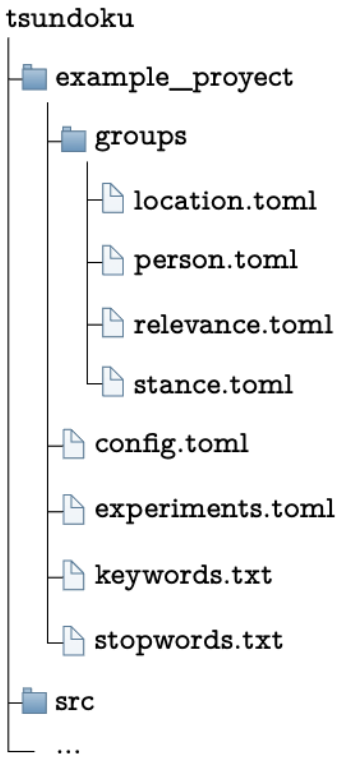

Figura 3.2: Estructura del directorio de configuración y sus archivos principales dentro del repositorio [12]

Por otro lado, en el archivo `experiments.toml` (ver Anexo A.2), se define la información base de los experimentos, las que incluyen las fechas que abarca el estudio y las constantes relacionadas con el filtrado de *tweets*. Además, en este archivo se establecen los parámetros del clasificador para cada grupo de estudio, tales como el `learning_rate`, `max_depth` y el tipo de resultado `objective`, que define si el grupo es binario, categórico, numérico, u otro.

Dado que el objetivo del proyecto es la caracterización de los usuarios en torno a un tema específico de discusión, es necesario configurar el sistema de acuerdo con el tópico. Una forma dinámica de lograrlo es mediante la definición de una lista de palabras clave, que sirvan como guía y de filtro para el proceso de filtrado de tweets. Estas palabras clave se obtienen de un archivo de texto en formato `keywords.txt` (ver Anexo A.3), que contiene los hashtags y expresiones regulares relevantes para la discusión y su manejo en el sistema.

Además, en el ámbito del procesamiento de lenguaje natural es bien sabido que es necesario realizar un preprocesamiento del texto para eliminar el ruido generado por palabras que no son relevantes para el análisis. Por esta razón, se define una lista de palabras conocidas como *stopwords* (`stopwords.txt`), que corresponden a palabras que aportan coherencia al texto pero que dificultan el proceso de clasificación. También se incluyen palabras que pueden no estar consideradas como *stopwords* por algunas librerías, pero que no guardan relación con el tópico en estudio.

Estos pasos de configuración y preprocesamiento son fundamentales para asegurar que el sistema de *Tsundoku* trabaje con la información relevante y minimice la influencia de palabras y elementos superfluos en el análisis de los *tweets*. Así, se garantiza una mayor precisión en la clasificación y caracterización de los usuarios en el contexto del tema de discusión.

Finalmente, es necesario definir características específicas para cada grupo o característica a clasificar. Dado que estos grupos pueden ser variables categóricas y el clasificador es un modelo supervisado, es importante tener una forma de etiquetar correctamente a los usuarios basándose en la información obtenida de los *tweets*. Para ello, se necesita de un archivo por cada grupo de estudio en el que se definen las posibles categorías, junto con los tokens que los identifican:

- **Stance** (Postura):
  En el caso de la postura, se definen dos categorías: `empathy` y `threat`. Por ejemplo, si el estudio se centra en la discusión del aborto [3], los *tokens* relacionados con `empathy` podrían ser *#abortolegal*, *elección* o *#abortolibre*, mientras que para `threat` podrían ser *#noalaborto*, *#salvemoslasdosvidas* o *vida*. Además, se considera otra categoría `undisclosed`, en caso de que no se pueda asignar ninguna de las opciones anteriores. Se incluye un ejemplo con datos reales en el Anexo A.4.
- **Location** (Lugar de procedencia):
  En el caso del lugar de procedencia, considerando que la población en estudio es chilena, se utilizan las 16 categorías correspondientes a las regiones de Chile. Estas son `arica`, `tarapaca`, `antofagasta`, `atacama`, `coquimbo`, `valparaiso`, `rm`, `ohiggins`, `maule`, `nuble`, `biobio`, `araucania`, `loslagos`, `losrios`, `aysen` y `magallanes`. También se incluye la categoría `noise`, que abarca a los usuarios provenientes de otros países distintos a Chile.
- **Relevance** (Relevancia):
  La relevancia mide cuán relacionado está el *tweet* con el tema de estudio, por lo que se utilizan dos categorías: `relevant` y `noise`.
- **Person** (Tipo de usuario):
  En esta categoría se clasifica el tipo de usuario que se está estudiando. Se definen tres categorías principales: `male` (usuario que se identifica con el género masculino), `female` (usuario que se identifica con el género femenino) e `institutional` (usuario que no corresponde a una persona, sino a una institución o grupo de personas que actúan como un usuario). También se incluye la categoría `undisclosed` en caso de que no se pueda determinar ninguna de las anteriores opciones.

Estas características por grupo de estudio se definen en archivos específicos para cada grupo, facilitando la identificación y clasificación de los usuarios dentro del sistema. Además, cada una de las palabras claves de cada categoría, como la definición de la lista de *keywords* principal del tópico en estudio son generados manualmente, por lo que es un proceso costoso.

3.2. Componente de preprocesamiento de datos

Tal como se mencionó anteriormente, la fuente principal de datos utilizada en este estudio proviene directamente del profesor Eduardo Graells, quien ha recopilado *tweets* históricos de *Twitter* desde el año 2020. Estos datos se encuentran almacenados en archivos en formato `.json.gz`, los cuales contienen objetos comprimidos en notación `JavaScript`. Estos archivos son generados mediante un programa y desarrollado específicamente para este propósito, el cual consulta constantemente cada cinco minutos a la API v1.1 de *Twitter* para obtener la información requerida.

Es importante destacar que, si bien el funcionamiento detallado de este programa es relevante para el proceso de obtención de los datos, no será abordado en el presente trabajo, ya que el enfoque se centra en otros aspectos del sistema. Por otro lado, cada uno de estos archivos se encuentra separado por fecha, siguiendo un formato de nombre específico:

`auroracl_<año><mes><día><hora><minuto>.data.json.gz`

Cada uno de estos archivos almacena una lista de *tweets*, los cuales acoplan un objeto con información referente al usuario y a los datos del *tweet* crudos que contienen los siguientes datos (ejemplo detallado en el Anexo A.5):

- **Metadata del tweet**:
  Esta incluye el texto plano del *tweet*, la fecha de creación, el idioma, menciones a otros *tweets* y entidades como *hashtags* contenidos dentro del mismo.
- **Metadata del usuario**
  Esta incluye el id del usuario, su descripción en texto plano, el lugar de procedencia, el nombre de usuario interno, el nombre público, y otras métricas como cantidad de seguidores, amigos, si está verificado, etc.
- **Metadata de las interacciones del *tweet***
  Esta incluye un booleano que indica si corresponde a un *retweet*, una mención a otro usuario, una respuesta a un *tweet*, y los usuarios que intervienen en cada una de las interacciones anteriores.

Esta información recopilada en una gran cantidad de archivos sirve como primer input del sistema. Estos son preprocesados de forma que, en primer lugar, se filtran todos los archivos dentro de un límite de fechas preconfigurado, y posteriormente se filtra cada archivo en base a las *keywords* definidas en el archivo de configuración `/<proyect_name>/keywords.txt` descrito en la sección anterior.

Este preprocesamiento genera en tiempo de ejecución nuevos archivos que recopilan los *tweets* ya filtrados, los cuales son almacenados en directorios dedicados a cada día involucrado dentro del estudio en particular. Estos archivos siguen el siguiente formato de nombre específico:

`raw/<año>-<mes>-<día>/tweets.partition.<número de partición>.json.gz`

Sin embargo, el tiempo de compilación del preprocesamiento del sistema y la generación de estos nuevos archivos con *tweets* filtrados es excesivamente lento. Utilizando el subsistema de Linux brindado por Windows (*WSL*) y con 16 giba-bytes de memoria RAM, este proceso puede tardar entre cuarenta segundos hasta diez minutos dependiendo de la cantidad de *tweets* recopilada en un mismo día.

Lo anterior se debe a que el formato ocupado para registrar los *tweets* no es apto para el manejo de grandes volúmenes de información, ni tampoco en un formato remendado para el tipo de estudio que se está realizando. Así, el tiempo de ejecución total del sistema genera bastantes dificultades de uso, y es la principal motivación de este trabajo.

Visto que el formato de los datos no permite ocupar el toolkit correctamente, nace la necesidad de migrar a alguno más apto para labores de procesamiento masivo de datos (*Big Data*). Tal como se mostró en la sección 2.3.1, la mejor opción es el uso de la librería *Apache Arrow* con su manejo simplificado de ficheros `.parquet`. Por esto, se requiere actualizar el funcionamiento actual del preprocesamiento a uno que considere la actualización del tipo de archivo que se maneje internamente.

3.2.1. Actualización del input del sistema actual

En primer lugar, se debe crear un nuevo programa que haga *parsing* de todos los archivos recopilados por el profesor Eduardo Graells. Para facilitar su uso, los archivos se elegirán con el mismo criterio con el que funciona actualmente el preprocesamiento, esto es, con un input de fechas y selección utilizando expresiones regulares.

Si bien es posible integrar dentro del preprocesamiento el *parsing* de estos archivos a uno especializado como lo es `.parquet`, esta no es una solución realmente conveniente para los usuarios. Lo anterior se debe a que de igual manera se estarían leyendo archivos en formato `.json` cada vez que se requiera realizar un nuevo estudio, y manteniendo la gran cantidad de *tweets* en un formato inadecuado.

Es por esto que la creación de un programa aislado que se encargue por sí solo de este proceso es una mejor solución, ya que evitará que siempre se haga *parsing* de todos los

archivos del conglomerado en cada ejecución del sistema. Esto también permite que el profesor Eduardo Graells o cualquier otro usuario de *Tsundoku* pueda mantener toda su base de datos en un formato acualizado sin tener que volver a solicitarlos.

Con lo anterior, el programa generará un nuevo directorio que simulará la base de datos del sistema, el cuál deprecará por completo el uso del conglomerado en formato `.json` y que reunirá los nuevos archivos con un formato de nombre equivalente al anterior:

`auroracl_<año><mes><día><hora><minuto>.data.parquet`

Esta actualización del manejo de datos puede verse gráficamente con la ilustración 3.3. En donde las líneas en gris indican el formato de archivo que se estará manejando en cada instante del componente, y el diagrama de decisión en amarillo es efectuado por el usuario.

Cabe mencionar que la Base de Datos no se refiere expresamente a un esquema de Base de Datos manejado con un gestor típico, si no que hace referencia a la fuente de datos que alimentará al sistema ubicado en un directorio determinado por el usuario. No está considerado dentro de este trabajo el despliegue de esta gran cantidad de *tweets* a algún servicio pagado, ni tampoco disponibilizar este sistema de forma pública a corto plazo.

Además, el proceso de preprocesamiento indicado antes de la recopilación de *tweets* en la ilustración 3.3 se detallará a continuación.

3.2.2. Importación y preprocesamiento de *tweets*

El proceso de importación consiste en filtrar el conglomerado de *tweets* obtenidos de forma cruda y transformarlos en una serie de directorios ordenados por cada uno de los días involucrados en el estudio. Esta transformación es crucial para garantizar que los datos se organicen de manera clara y sean manejables dentro del sistema *Tsundoku*. Además de la organización, este proceso se encarga de preprocesar y filtrar aquellos *tweets* que no sean relevantes para el trabajo de investigación, lo cual es fundamental para garantizar la calidad de los datos utilizados.

Para llevar a cabo este proceso, se tiene la implementación de la clase `Importer`, la cual desempeña un papel central en la gestión eficiente de este volumen de datos. Esta clase almacena todas las configuraciones de cada parámetro en estudio, lo que proporciona flexibilidad y control en el proceso de importación. Además, dentro de esta clase se definen los métodos necesarios para iterar sobre los archivos que cumplen con el patrón de fecha dentro del conglomerado de *tweets*, lo que garantiza la selección adecuada de los datos temporales requeridos para el estudio.

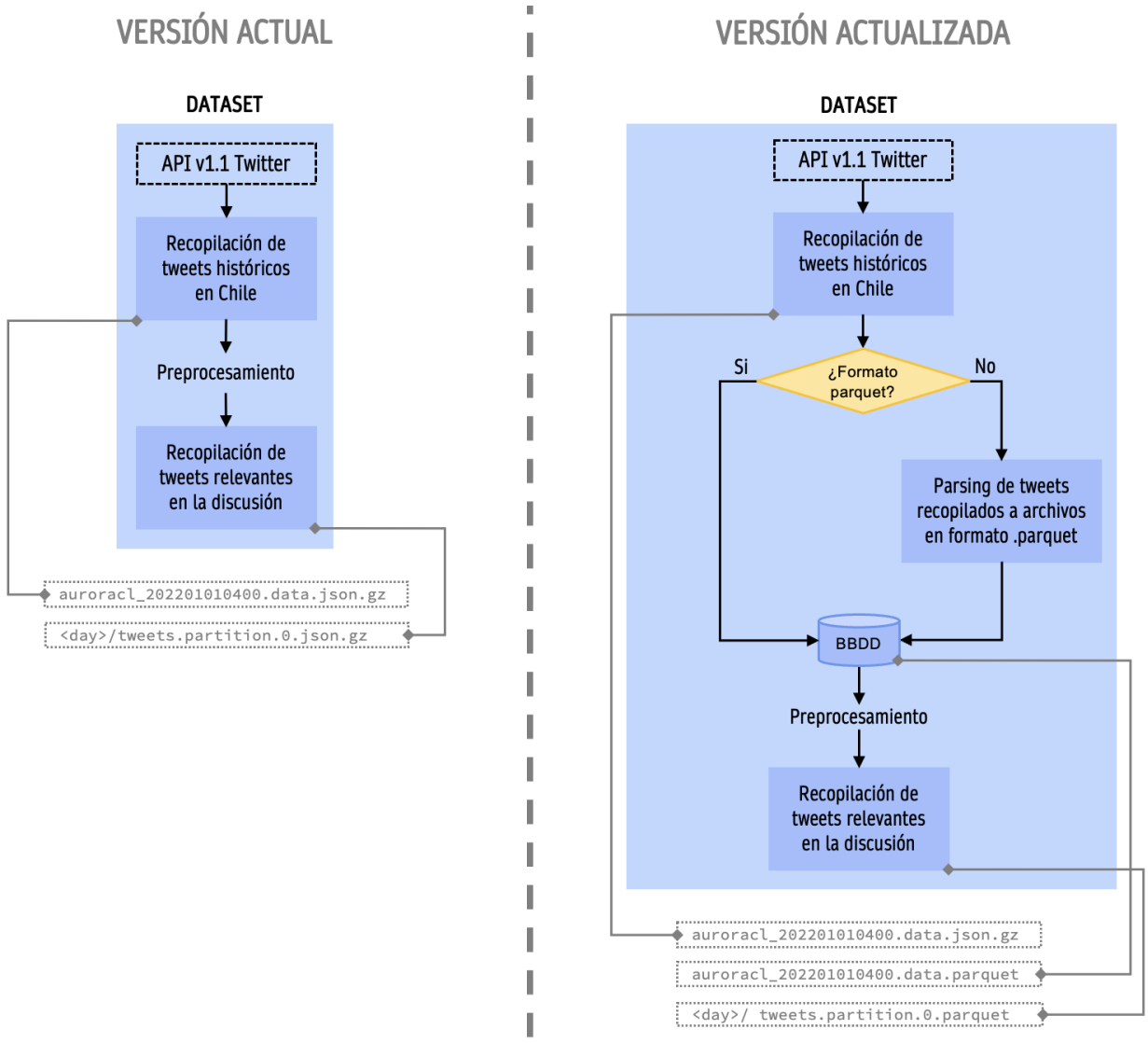

Figura 3.3: Diagrama de comparación del manejo y funcionamiento actual del dataset de *Tsundoku* versus el sistema actualizado

Dentro de la clase `Importer` también se establecen funciones y lógica específicas para leer y filtrar los *tweets* en base a la lista de *keywords* del archivo de configuración, lugares de procedencia y una lista negra. La lista negra tiene como objetivo restringir ciertos *tweets* que no son considerados relevantes para el análisis. Por ejemplo, se eliminan aquellos *tweets* que provienen de países distintos a Chile y no están dentro del alcance del estudio, como también se descartan *tweets* que fueron incluidos debido a las palabras clave, pero que pertenecen a otros temas debido a similitudes léxicas.

Durante el proceso de filtrado se utiliza la librería *ahocorasick* [25], la cual permite crear un autómata para la detección eficiente de patrones con las palabras clave definidas en el archivo de configuración. Esta elección se basa en la alta eficiencia que ofrece esta librería para realizar comparaciones de cadenas de texto. Además, se aprovechan herramientas de

programación funcional de alto nivel proporcionadas por la librería *cytoolz*, lo que facilita el manejo de los datos y agiliza el proceso de filtrado.

Durante la ejecución de este procesos se generan nuevos archivos con *tweets* filtrados, dentro de directorios dedicados a cada día del estudio y que tienen el siguiente formato:

`<día>-<mes>-<año>/tweets.partition.<número de partición>.json.gz`

Con la actualización del input del sistema, es necesario refactorizar todo el funcionamiento descrito anteriormente para generar archivos con el nuevo formato `.parquet`. Se puede ver un fragmento del proceso de filtrado de *tweets* en el Anexo A.6.

3.3. Componente de Cómputo de Features

Teniendo cada día del experimento separado en datos preprocesados y filtrados en *tweets* relevantes para el tópico en estudio, ya es posible comenzar a recopilar y anidar la información que nos importa de cada *tweet* a estructuras que sean de utilidad para su clasificación. Este proceso consiste en ir día por día y definir los datos claves de los usuarios y de los mismos *tweets*, de manera que podamos extraer características relevantes para la clasificación.

3.3.1. Cómputo de features por cada día del grupo de estudio

Toda la *metadata* de los usuarios tales como la descripción, su nombre de usuario, el lugar de procedencia y también el dominio que indiquen en su perfil nos permite caracterizar a cada uno de ellos. Es por esto que en cada iteración se debe crear un *dataframe* utilizando la librería *dask* que recopile todos estos datos y puedan ser manejados eficazmente.

No solo la *metadata* del usuario nos parece relevante para la clasificación, si no que también datos relacionados con los *tweets*. El contenido es fundamentalmente lo que estamos procesando por cada uno de ellos, junto con el periodo en el que se crearon, y los posibles enlaces que se compartan son indicadores para la detección de postura u otras métricas ya definidas.

Por otro lado, y debido que también es relevante conocer el comportamiento de los usuarios dentro de la discusión, es necesario mantener la información de las distintas interacciones que se tengan entre *tweets* a lo largo del estudio. Estas no solo sirven para caracterizar la discusión, si no que también para la detección de *bots* al observar anomalías en las interacciones.

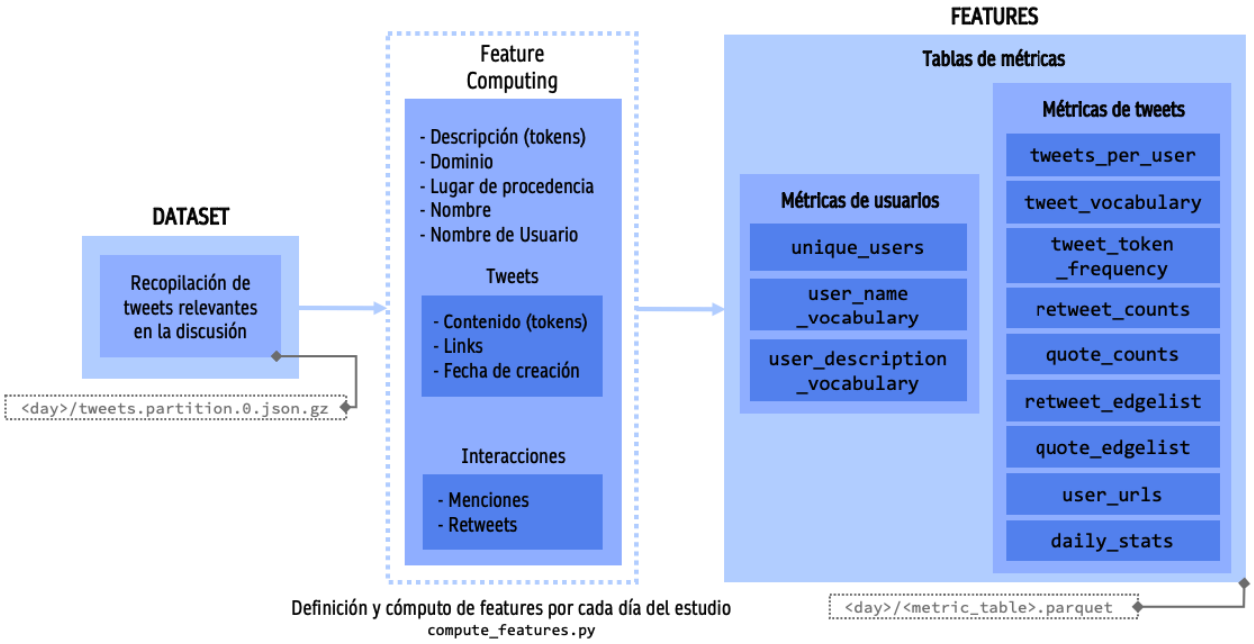

Figura 3.4: Diagrama de cómputo de features por día del estudio

En el diagrama de cómputo de features mostrado en la ilustración 3.4 se muestran todos los *dataframes* o tablas de métricas generadas en este proceso, las cuales se detallarán a continuación:

- **Tablas de métricas de los usuarios**:
  Las tablas de métricas de los usuarios se encargan de recopilar información relevante sobre los usuarios en la discusión, como su perfil y características distintivas. Estas tablas incluyen datos como la descripción y el lugar de procedencia de los usuarios, así como los *tokens* utilizados en sus nombres y descripciones, que pueden ser útiles para detectar *bots* y clasificar el tipo de usuario.

- **Tablas de métricas de los tweets**
  Por otro lado, las tablas de métricas de los *tweets* se centran en recopilar datos sobre los propios *tweets* y las interacciones asociadas. Estas tablas contienen información sobre la cantidad de *tweets* por usuario, el uso de *tokens* distintos en los *tweets* y la frecuencia de cada *token*.

  Además, se registran las interacciones realizadas, como *retweets*, citas y respuestas tanto a nivel individual de cada usuario como a nivel de interacciones entre usuarios. También se analizan los enlaces y dominios presentes en los *tweets*, lo cual puede ser relevante para comprender el tipo de usuario o su postura. Por último, se resumen estadísticas diarias de las interacciones, como menciones, respuestas recibidas, popularidad y métricas relacionadas con la cantidad de seguidores y amigos.

3.3.2. Construcción de tablas de métricas y matrices

Ya computadas las métricas más relevantes por cada día del estudio, es necesario agrupar todas estas mismas en tablas únicas para todo el experimento, y a su vez generar el input del clasificador. Este proceso sigue la misma metodología descrita en el trabajo *'Every colour you are'* [3], y a continuación se detallan los principales datos generados para este.

Primero, se toman todas las tablas generadas por cada día del experimento y se agrupan en *dataframes* más concisos y que incluyen:

- `user.unique`: Tabla que agrupa todos los usuarios únicos de todo el experimento, eliminando usuarios duplicados.
- `user.total.tweets`: Tabla que cuenta la cantidad de tweets total por usuario dentro de todo el periodo del experimento, sumando todos los tweets hechos por cada día.
- `user.profile_domains.relevant` y `user._domains.relevant`: Tablas que recopilan los dominios más relevantes de los usuarios, tanto los de perfil como los compartidos respectivamente. El nivel de relevancia se determina por la cantidad de repeticiones dentro del experimento, cuyo valor se define en la sección *'thresholds'* del archivo de configuración principal del experimento `experiments.toml`.
- `user.<interaction>_edges.all`: Tabla que registra la cantidad de interacciones por ambos usuarios involucrados en las interacciones del tweet, y que sirve como intermediaria para la generación de la red de interacciones. Las interacciones son `retweet`, `quote` y `reply`.
- `network.<interaction>.target_ids`: Tabla que recopila datos de las interacciones de los usuarios, las cuales como se ha mencionado incluyen `retweet`, `quote` y `reply`. Esencialmente registra la red de conexiones de todo el experimento, incluyendo también los usuarios que no han interactuado con otros.
- `user.tweet_vocabulary.all`, `user.name_tokens.all` y `user.description _tokens.all`: Tablas que registran todo el vocabulario ocupado por cada usuario. Estas incluyen los tweets, el nombre de usuario y la descripción respectivamente. Estas a su vez tienen sus versiones con el flag `relevant`, y al igual que el caso anterior, su valor se define en la sección *'thresholds'* del archivo de configuración principal del experimento.

Además del cómputo y agrupación de las distintas tablas, todos los datos de las interacciones entre usuarios y el vocabulario ocupado por cada uno en el estudio servirán como input del clasificador. Estos, sin embargo, serán matrices generadas a partir de la agrupación de las tablas computadas por cada día del experimento, mejor conocidas como matrices usuario-término y tweet-término descritas en el trabajo *'Every colour you are'* [3].

Las matrices generadas para el proceso de clasificación son las siguientes:

- **'user-term' matrix**:
  Estas corresponden a las llamadas matriz usuario-término, donde cada celda (i, j) codifica la cantidad de veces que cada usuario i ha ocupado un término o token j específico en el contenido de sus tweets publicados en el estudio. Estas matrices son `tweet_features`, `name_features` y `description_features`, donde cada una relaciona los términos ocupados en los tweets, nombres de usuario y descripciones respectivamente.

- **'one-hot encoded features' matrices**:
  Estas corresponden a matrices que incluyen metadata del usuario, donde se consideran los dominios, los dominios de los perfiles, etc.

- **'user-stance' matrices**:
  Esta matriz contiene el número de interacciones que un usuario ha tenido con otros usuarios con una postura pre-definida.

- **'adjacency' matrices**:
  Estas corresponden a tres matrices de adjacencia, donde cada celda (i, j) codifica las interacciones donde el usuario i ha interactuado con el usuario j. Como se mencionaron antes, estas interraciones son `retweet`, `quote` y `reply`.

Cada una de estas matrices relaciona dos métricas relevantes dentro del estudio, las cuales en su mayoría tienen en cuenta el vocabulario utilizado en la investigación o las interacciones dentro de *Twitter*. Sin embargo, esta metodología puede no ser óptima en términos de eficiencia de espacio y puede resultar en un uso innecesario de memoria. Esto se debe a que muchos valores dentro de cada matriz suelen ser 0, ya que es poco común que cada usuario utilice todo el vocabulario existente en un experimento o que hayan tenido interacciones con todos los demás usuarios involucrados.

Con el fin de abordar esta problemática, estas fueron implementadas utilizando objetos de matrices distribuidas (*Sparce Matrices*) de *Python*. Estas matrices se diseñan para registrar únicamente las coordenadas con valores distintos de cero, evitando así el almacenamiento de información redundante y optimizando el uso del espacio en memoria. Al utilizar matrices distribuidas, se logra reducir significativamente el consumo de recursos sin comprometer la integridad de los datos.

Finalmente, estas matrices al ser concatenadas las unas con las otras sirven como input al clasificador. El diagrama resumen del proceso de cómputo de *features* puede verse en la ilustración 3.5.

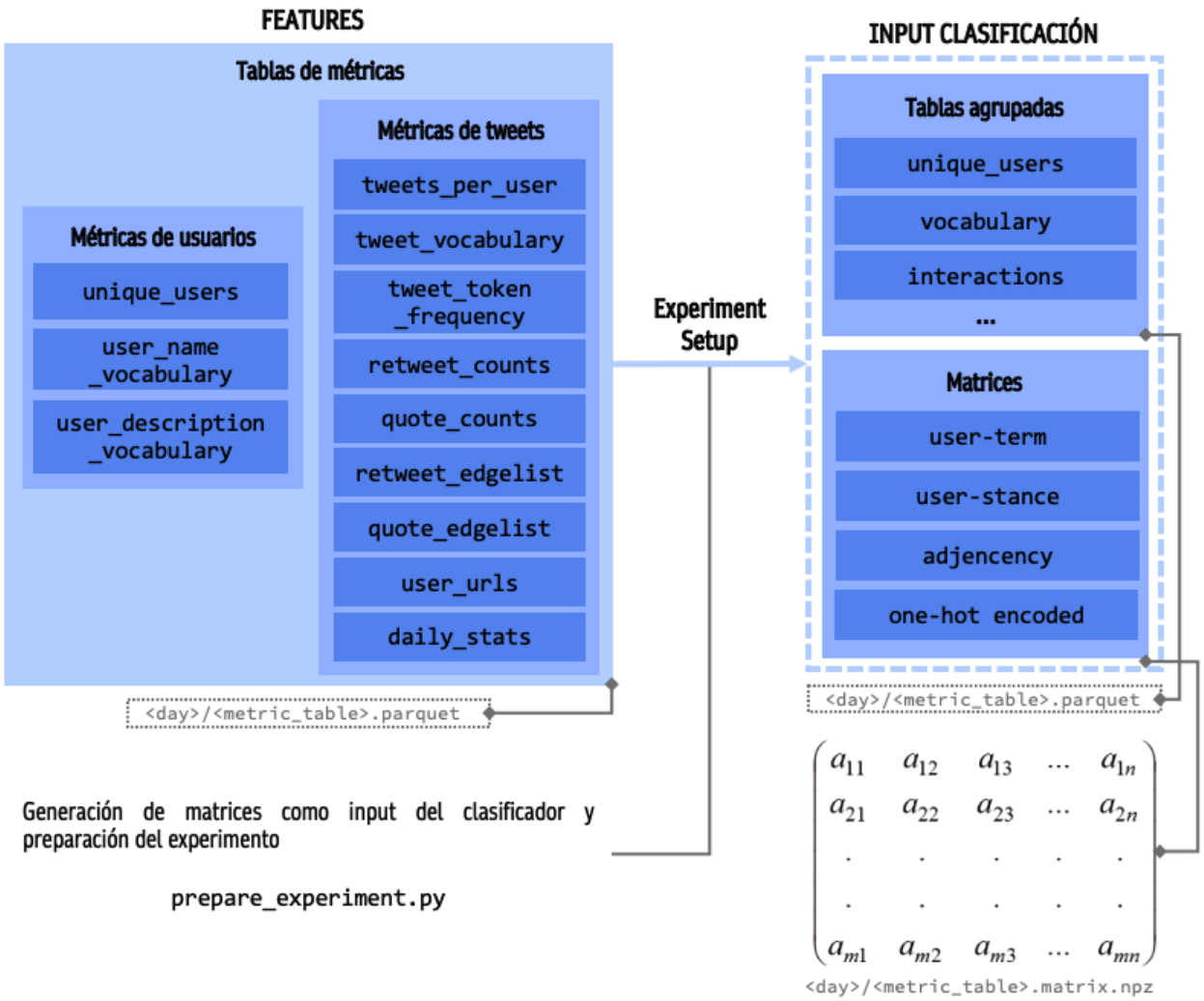

Figura 3.5: Diagrama de generación de matrices y tablas para el input del clasificador

3.4. Componente de Clasificación

Para el proceso de clasificación en *Tsundoku* se empleó el modelo *XGBoost* (sección 2.1.3). *XGBoost* es ampliamente reconocido por su capacidad para manejar conjuntos de datos complejos y grandes volúmenes de información, así como por su eficacia en problemas de clasificación y regresión.

Dado que *XGBoost* es un modelo de clasificación supervisado, surgen ciertas complicaciones respecto al dataset de entrenamiento y la definición de las etiquetas. En este contexto, se busca combinar la velocidad de entrenamiento inherente a un modelo basado en árboles, con la facilidad de poder entrenar este enorme conjunto de datos de manera no supervisada.

Con este objetivo en mente, el modelo desarrollado en *Tsundoku* se basa en un enfoque de

clasificación semi-supervisada. Esto implica utilizar las características de los propios datos para asignar automáticamente etiquetas a cada usuario, evitando así tener que realizar esta tarea manualmente para cada uno de ellos.

Esta estrategia resulta tremendamente útil, ya que la única dificultad radica en definir de manera clara qué usuarios y qué parámetros se deben utilizar para llevar a cabo el proceso de etiquetado automático. Para esto, se definió la clase `PartiallyLabeledXGB`, la cual encapsula las características clave del clasificador, incluyendo la definición del modelo utilizando la biblioteca *sklearn*, así como los métodos de entrenamiento y evaluación correspondientes.

3.4.1. Pipeline de clasificación con XGBoost

El proceso de clasificación con *XGBoost* se lleva a cabo mediante un pipeline que involucra diferentes etapas. Si bien el etiquetado de los usuarios no es completamente automático, se puede automatizar en gran medida gracias al uso de archivos de configuración específicos para cada grupo de estudio, tal como se describe en la sección 3.1. Estos archivos de configuración son fundamentales, ya que contienen los *tokens* y las variables categóricas necesarios para identificar y relacionar las características relevantes en cada caso.

1. **Preparación de features**:
   En esta etapa se procesan las matrices definidas en el componente anterior como entradas para el clasificador. Por cada grupo de estudio, se determinan los *tokens* relevantes que deben buscarse dentro de estas matrices a partir de las *keywords* de los archivos de configuración. Este proceso garantiza que se consideren únicamente las características y los datos relevantes para cada análisis específico.

2. **Pre etiquetado (pre-labeling)**:
   A continuación se asignan etiquetas preliminares a los usuarios que presentan una mayor concordancia con los *tokens* y los identificadores de usuario definidos en el archivo de configuración correspondiente. Estas etiquetas permiten tener un punto de partida para el proceso de clasificación y facilitan la propagación posterior de las etiquetas al resto del conjunto de datos.

3. **Propagación de etiquetas (*label-propagation*)**:
   Posteriormente se procede a extender estas etiquetas al resto del *dataset* utilizando técnicas de propagación de etiquetas como *'bootstrapped approach'*. Esta propagación tiene como objetivo asignar etiquetas a aquellos usuarios que no fueron pre-etiquetados, pero que presentan características similares a los usuarios ya etiquetados.

4. **Entrenamiento**:
   Con las etiquetas propagadas, se inicia el proceso de entrenamiento típico del clasificador con *XGBoost*.

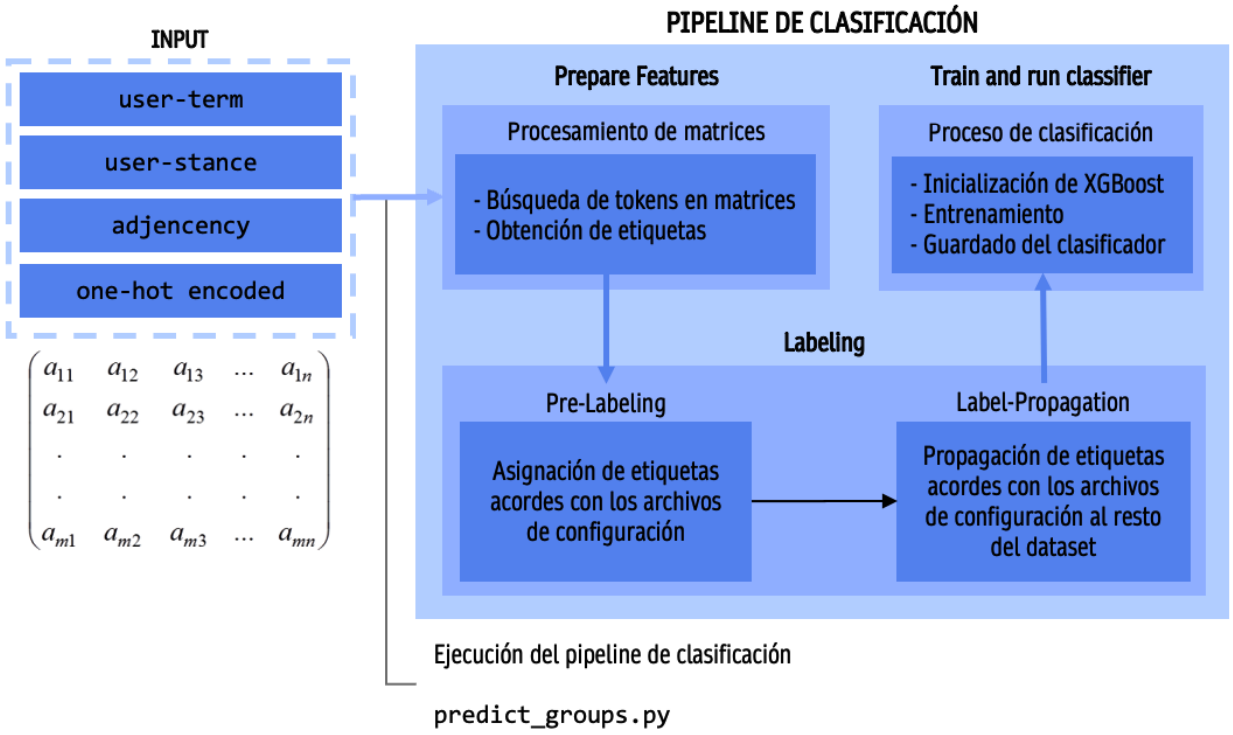

Figura 3.6: Diagrama del pipeline de clasificación

Este pipeline de clasificación permite automatizar y agilizar el proceso de etiquetado y clasificación de los usuarios en base a las características relevantes del estudio. Al combinar la preparación de características, el pre-etiquetado y la propagación de etiquetas, se logra una clasificación precisa y eficiente de los usuarios en función de las variables categóricas definidas en los archivos de configuración.

3.5. Componente de Consolidación de Resultados

Finalmente se tiene el componente de Consolidación de Resultados. Esta sección del sistema desempeña un papel crucial al finalizar el proceso de clasificación y se enfoca en dos aspectos clave: la evaluación del clasificador y la detección de *bots*.

3.5.1. Evaluación del clasificador

La evaluación del clasificador constituye una etapa fundamental para medir la efectividad y el desempeño del modelo en la tarea de clasificación en *Tsundoku*. Se lleva a cabo un análisis exhaustivo de las métricas de calidad, como la *precision*, *recall* [8], *support* y *f1-score*.

[8] Exhaustividad o *recall*: métrica de evaluación que indica los falsos negativos de la clasificación (ver Glosario)

Estas métricas permiten evaluar de manera cuantitativa la capacidad del clasificador para asignar correctamente las etiquetas a los usuarios y proporcionan información valiosa sobre su rendimiento.

Para calcular estas métricas se utiliza el método `classification_report` de la librería *sklearn.metrics*. Este método toma como entrada las etiquetas verdaderas y las etiquetas predichas por el clasificador, y genera un informe detallado que incluye las métricas mencionadas anteriormente para cada clase del problema de clasificación.

Es importante destacar que el proceso de evaluación se lleva a cabo mediante una validación cruzada. Esto implica dividir el conjunto de datos en k subconjuntos, donde el modelo se entrena k veces utilizando diferentes combinaciones de subconjuntos de entrenamiento y se evalúa en los subconjuntos de prueba correspondientes. Al realizar la evaluación de esta manera, se obtienen resultados más robustos y se reduce el impacto de la aleatoriedad en la evaluación del modelo.

3.5.2. Análisis de grupos y detección de anomalías

En esta sección se lleva a cabo un exhaustivo análisis de cada grupo de estudio, con el objetivo de determinar características clave como el vocabulario más utilizado por cada categoría, las redes que conforman sus interacciones y también los usuarios más relevantes dentro de cada grupo.

El análisis del vocabulario permite identificar las palabras y términos más frecuentes en los mensajes de cada grupo, lo cual brinda una comprensión más profunda de los temas y el lenguaje utilizado en las discusiones. Asimismo, se examinan las interacciones entre los usuarios dentro de cada grupo, lo cual permite identificar las principales redes y conexiones que se establecen entre ellos. Este último análisis revela patrones de comunicación, la formación de comunidades y la estructura de la red en cada grupo.

Por otro lado, el componente de detección de anomalías se centra en la identificación y caracterización de *bots*. Este proceso se lleva a cabo mediante la implementación de un árbol de decisión que segmenta a los usuarios en función de la anomalía de sus características en comparación con el resto de la muestra. Este enfoque se basa en el modelo de detección de anomalías conocido como *Isolation Forest*, el cual ha demostrado ser eficaz en la identificación de comportamientos atípicos en conjuntos de datos [9, 10].

El proceso de detección de anomalías tiene como objetivo identificar aquellos usuarios que presentan un comportamiento inusual en términos de sus interacciones, frecuencia de publicación, contenido de los mensajes y otras características relevantes como la fecha de creación y los tokens de su nombre de usuario. Estos usuarios anómalos son considerados

potenciales *bots* y su detección es uno de los principales objetivos del sistema *Tsundoku*.

Cabe destacar que el proceso de análisis de grupos y detección de anomalías ha sido desarrollado y explicado en mayor detalle en el trabajo titulado *"Bots don't Vote"* [6]. Este trabajo proporciona información detallada sobre la metodología utilizada, los algoritmos implementados y los resultados obtenidos. Para más detalles sobre el proceso de análisis en el contexto del sistema *Tsundoku*, se puede hacer referencia a la sección 2.2.5.

3.6. Integración de técnicas de procesamiento de lenguaje natural basados en contexto

Como se mencionó en la motivación de este trabajo, la integración de herramientas de Procesamiento del Lenguaje Natural (*NLP*) basadas en contexto puede brindar mejores resultados en el proceso de clasificación de postura y caracterización de los usuarios, al tiempo que abre nuevas oportunidades para la investigación relacionadas con la caracterización de discusiones en *Twitter*.

Dentro de estas herramientas, los *transformers* (sección 2.1.4) han demostrado ser especialmente útiles y han obtenido resultados altamente positivos en la clasificación de texto basada en contexto. Dos ejemplos destacados de modelos *transformers* son BERT y *ChatGPT*.

BERT, como fue mencionado en la sección 2.1.4.1 es un modelo de lenguaje basado en *transformers* que ha sido pre-entrenado en grandes corpus de texto y ha logrado avances significativos en diversas tareas de procesamiento del lenguaje natural, incluyendo la clasificación de textos. BERT tiene la capacidad de capturar las relaciones semánticas y sintácticas del texto al tomar en cuenta el contexto tanto anterior como posterior a cada palabra.

Por otro lado, ChatGPT también está basado en la arquitectura de *transformers*, y en el último año se ha destacado por su habilidad para generar respuestas coherentes y contextuales en conversaciones. A través del entrenamiento con grandes cantidades de datos, ChatGPT ha aprendido a comprender y generar texto de calidad, lo que lo convierte en una herramienta valiosa para abordar tareas de clasificación y caracterización de usuarios basadas en texto.

Sin embargo, es importante tener en cuenta que el entrenamiento de modelos como BERT y ChatGPT representa un desafío técnico y computacional. Estos modelos requieren un desarrollo de código adecuado y significativos recursos computacionales, particularmente en términos de memoria RAM necesaria para el entrenamiento. Dado que uno de los objetivos principales de este trabajo es optimizar los tiempos de ejecución, es necesario encontrar un equilibrio entre obtener buenos resultados al utilizar herramientas basadas en transformers y lograr un entrenamiento eficiente del modelo.

Teniendo en cuenta esta situación, se han estudiado las posibilidades de seguir dos enfoques distintos en cuanto a la integración de *transformers* en el sistema.

1. **Pipeline de clasificación con *transformers***
   El primer enfoque consiste en la implementación de un pipeline de clasificación paralelo al actual, aprovechando el potencial de los modelos de clasificación con *transformers*. Este nuevo pipeline se desarrollaría de manera independiente al funcionamiento actual del sistema, con el objetivo de proporcionar una alternativa al proceso de clasificación.

   De esta forma, se exploraría la posibilidad de mejorar los resultados obtenidos al considerar de manera más precisa el contexto en el análisis de los textos. Este enfoque busca aprovechar la capacidad de los modelos basados en *transformers* para capturar relaciones de dependencia y contextuales, lo que puede llevar a una mayor eficacia en la clasificación de posturas y caracterización de usuarios.

   Sin embargo, se ha determinado que este enfoque no resulta práctico en el contexto actual del sistema. Dado que uno de los objetivos del proyecto es mejorar los tiempos de ejecución de *Tsundoku*, la implementación de un pipeline de clasificación basado en *transformers* podría complicar excesivamente el proceso de clasificación y afectar negativamente la eficiencia del sistema.

   Teniendo en cuenta que el sistema actual funciona adecuadamente, se considera más conveniente mantener el enfoque actual de clasificación y buscar otras estrategias para mejorar la precisión y eficacia de los resultados.

2. **Integración de *word embeddings* a *XGBoost***
   El segundo enfoque surge como una respuesta al desafío que implica el entrenamiento de un nuevo clasificador basado en *transformers*. Para evitar entrenar directamente un modelo de redes neuronales, se considera la integración de una herramienta proporcionada por *transformers* denominada *word embeddings* al proceso de clasificación actual.

   Como se explicó en la sección 2.1.4, los *word embeddings* son vectores numéricos que asignan valores a las características semánticas del texto. Aunque a primera vista estos vectores pueden parecer extensos y carentes de significado, dentro del modelo es posible establecer relaciones entre ellos para determinar su similitud en un contexto específico. De esta manera, se logra capturar de forma más precisa y cuantitativa el aprendizaje contextual, enriqueciendo así el proceso de clasificación.

   La integración de *word embeddings* a *XGBoost* no implica una complejidad tan significativa al proceso actual de clasificación en comparación con la implementación de un pipeline secundario. Al aprovechar los *word embeddings*, se enriquece el modelo con información semántica adicional sin comprometer en gran medida la velocidad de ejecución.

   Esto representa una oportunidad prometedora para mejorar la precisión y la capacidad de generalización del sistema, abriendo nuevas posibilidades en el análisis de textos.

Para poder integrar este último enfoque se requiere de la creación de nuevos *dataframes* y matrices en los procesos de cómputo y clasificación. En particular, es necesario generar nuevas tablas que contengan el texto sin procesar de todos los *tweets* por usuario, ya que el proceso de tokenización utilizado hasta ahora en el sistema *Tsundoku* ha implicado la pérdida del contexto original de cada *tweet*:

3.6.1. Adiciones al componente de Cómputo de Features

Dado que la arquitectura de *transformers* contempla el preprocesamiento y la tokenización de texto de forma interna, es necesario generar tablas que agrupen todos los *tweets* realizados por cada usuario único en cada día del estudio. Es importante tener en cuenta que la cantidad de *tweets* por usuario es variable, Por lo que cada tabla debe incluir como columnas el ID del usuario y los *tweets* asociados a ese ID en una lista de strings.

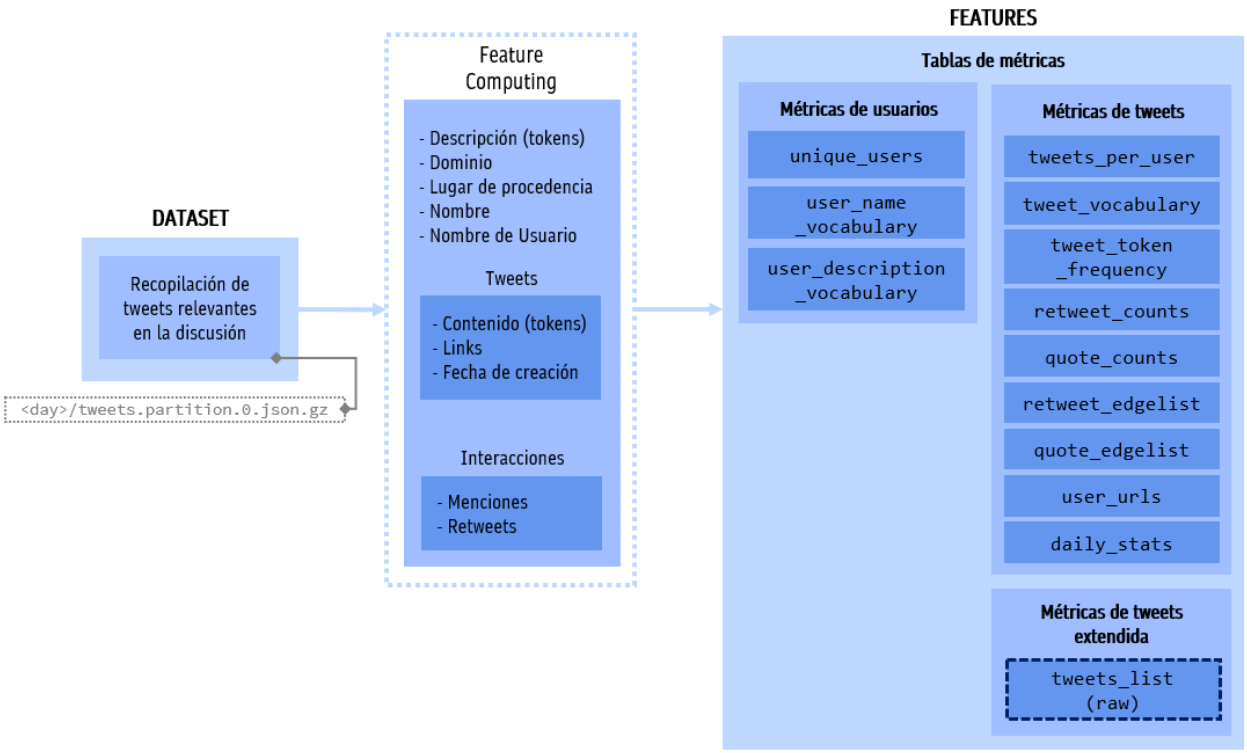

Figura 3.7: Diagrama resumen del funcionamiento extendido del componente de cómputo de *features*, donde se destaca la adición de un nuevo *dataset* de características correspondiente a la lista de *tweets* sin procesar

Luego se requiere agrupar todas estas tablas generadas en una sola. Se espera que este proceso no sea computacionalmente exigente, ya que es similar a otros procesos realizados en el mismo componente y el entrenamiento del modelo es principalmente un proceso cuya complejidad radica en la clasificación.

Además, es necesario generar una matriz que relacione a cada usuario con su *embedding*

para utilizarlo como entrada en el clasificador. La generación de los *word embeddings* es un proceso desafiante, ya que implica inicializar el modelo pre-entrenado de BETO junto con su tokenizador, y luego obtener los *embeddings* de cada uno de los *tweets* para cada usuario.

Una vez que se han generado los *embeddings* para cada *tweet* del usuario, existen dos posibles enfoques para la clasificación. La primera opción es almacenar todos estos *embeddings* en una matriz que relacione cada *tweet* con su correspondiente *embedding*, y a su vez, otra que relacione cada usuario con sus *tweets*. Aunque esta opción almacena una gran cantidad de detalles del estudio, no es la más óptima, ya que requeriría una gran cantidad de memoria para almacenar todos esos datos.

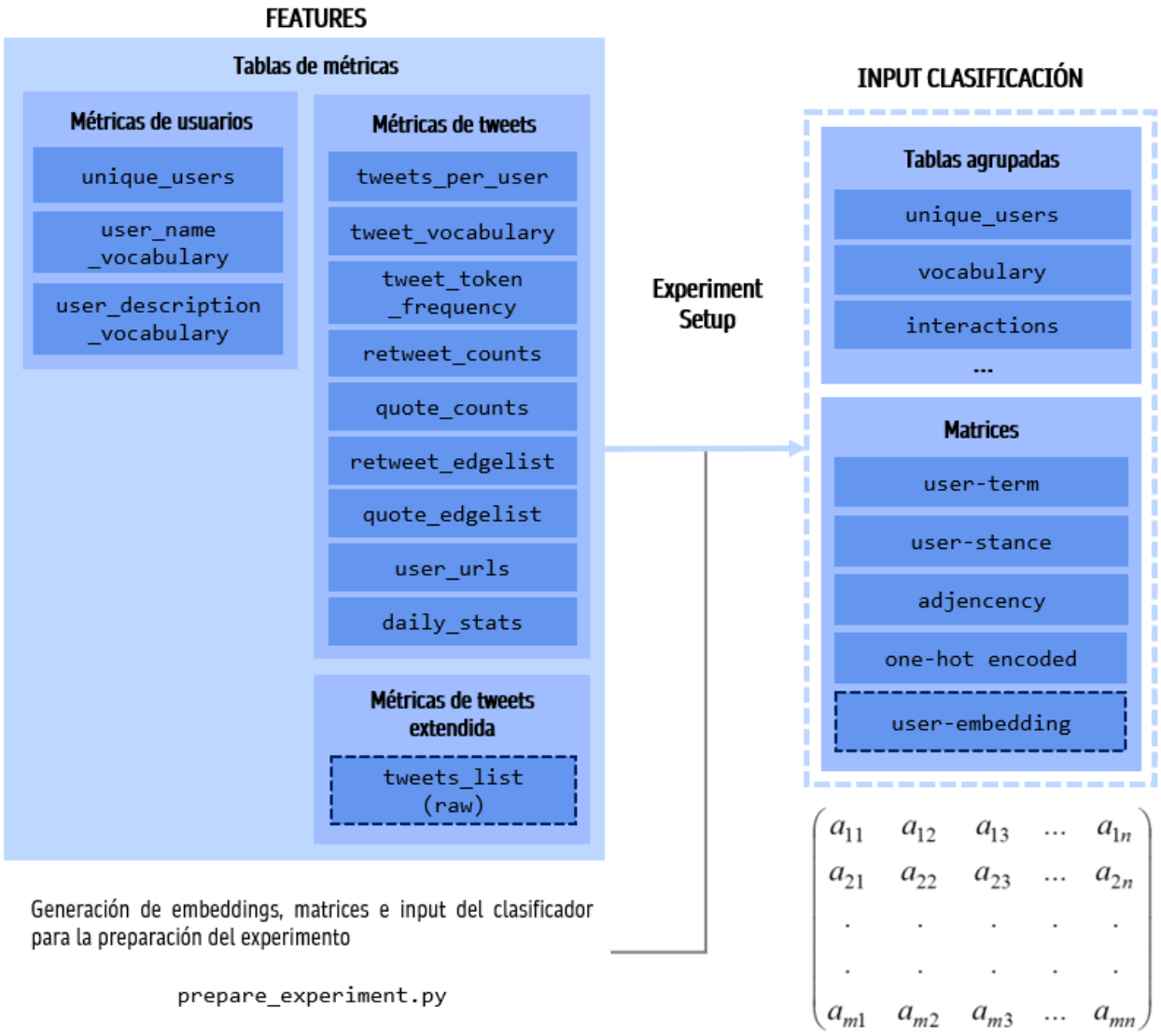

Figura 3.8: Diagrama resumen del funcionamiento extendido del componente de preparación del experimento, donde se muestra la generación de una nueva matriz *user-embedding* para el proceso de clasificación

Aquí surge la idea de calcular un vector denominado *user embedding* que represente al

usuario en su conjunto, en donde se promediaría o tomaría la mediana de los *tweet embedding* de un usuario específico. Esta idea tiene una justificación coherente en el contexto de nuestro experimento, puesto que si consideramos que cada *tweet* individual puede contener información relevante sobre las posturas, opiniones y características del usuario, promediar o tomar la mediana de los *embeddings* de todos los *tweets* nos permite obtener una representación sintáctica general del usuario en torno a un mismo tema.

Al calcular este *user embedding* estamos capturando las características sintácticas y semánticas más relevantes de los *tweets* de un usuario en un vector numérico compacto. Aunque perdemos ciertos detalles específicos de cada *tweet* individual, ganamos en eficiencia y memoria al representar al usuario de una manera más generalizada. Además, al promediar o tomar la mediana podemos mitigar el impacto de *tweets* ruidosos que podrían afectar la clasificación si se les diera un peso excesivo.

3.6.2. Adiciones al componente de Clasificación

A diferencia del proceso anterior, las adiciones y actualizaciones al componente de clasificación no resultan ser excesivamente complejas.

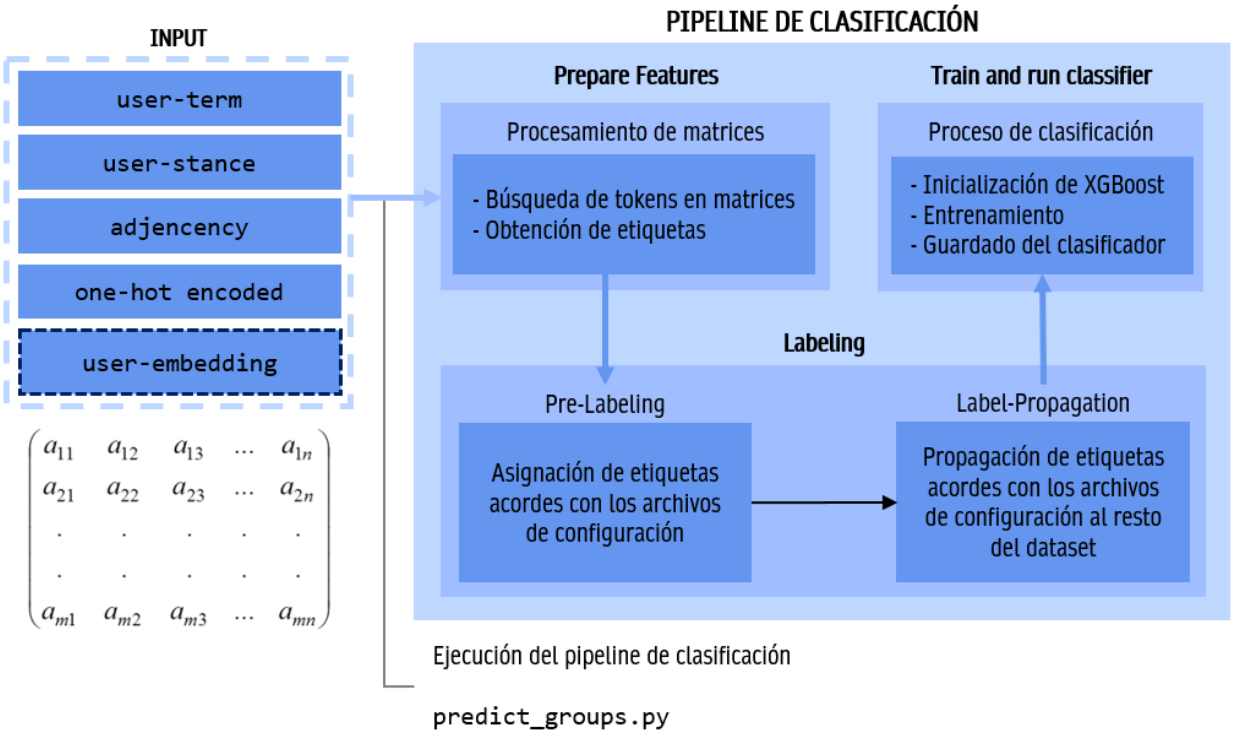

Figura 3.9: Diagrama resumen del funcionamiento extendido del componente de clasificación, donde se muestra la integración de la nueva matriz *user-embedding*

Debido a que ya se tendría una nueva matriz que relaciones a los usuarios con sus respec-

tivos *user embedding*, bastaría con integrarla al proceso de entrada del clasificador XGBoost. Esta matriz, como ya se explicó, contendría los vectores de *embeddings* correspondientes a cada usuario, y serían concatenados a las otras matrices para ser utilizados como datos de entrada para el clasificador.

Capítulo 4

Implementación de la Solución

El código fuente de sistema *Tsundoku* se encuentra con acceso público en la plataforma de control de versiones GitHub [12], por lo que todo el desarrollo se realizó sobre un fork del repositorio original en uno propio [26].

Por otro lado, la implantación del sistema de forma local presentó una serie de errores relacionados con la compatibilidad de algunas librerías. Es importante mencionar de que *Tsundoku* está desarrollado en un ambiente *Unix*, por lo que el sistema operativo con mayor compatibilidad de uso es *Ubuntu*. Se puede ocupar el Subsistema para Linux de Windows (WSL) [27] para poder ocupar y programar sobre el mismo en caso de que se utilice un ordenador con *Windows*.

Se recomienda además instalar el sistema de distribución *Anaconda* [28] para mejorar la gestión de las librerías y crear un entorno virtual más conveniente en el sistema *Tsundoku*. Anaconda proporciona una solución completa y útil para el manejo de paquetes y entornos de desarrollo en Python. Además, al crear un ambiente virtual específico para *Tsundoku* dentro de Anaconda se garantiza la coherencia y reproducibilidad del sistema al mantener sus dependencias aisladas.

4.1. Actualizaciones de formato en el preprocesamiento de datos

Recapitulando, la fuente principal de datos proviene directamente del profesor Eduardo Graells, quien ha recopilado *tweets* históricos de *Twitter* desde el año 2020. Este almacenamiento aglomera archivos en formato `.json.gz`, los cuales son generados por un programa que consulta constantemente a la API v1.1 de *Twitter*.

Recordar también que cada uno de estos archivos almacena una lista de *tweets*, los cuales acoplan un objeto con información referente al usuario y a los datos del *tweet* de forma cruda. Se puede encontrar estos datos con más detalle en la sección 3.2 y en el Anexo A.5.

El uso de archivos en formato `.json.gz` para registrar los *tweets* no es apto para el manejo de grandes volúmenes de información, ni tampoco en un formato remendado para el tipo de estudio que se está realizando. Esta situación se explica con mayor detalle en la sección 2.2.5.3, donde se mencionan las limitaciones actuales de *Tsundoku*.

Por lo anterior, el primer desarrollo sobre el sistema debe ser la creación de un script que permita actualizar el conglomerado actual de *tweets* a uno con un formato apto para le manejo de grandes volúmenes de datos. El formato elegido para esta actualización fue el fichero recomendado por la librería *PyArrow*, y corresponden a archivos .parquet.

4.1.1. Programa de parsing de archivos .json comprimidos a ficheros .parquet con librería pyarrow

Para llevar a cabo el proceso de parsing de los archivos recopilados por el profesor Eduardo Graells, se implementó un nuevo programa dentro del componente de preprocesamiento. Este se encarga de realizar el parsing de los archivos seleccionados de acuerdo al mismo criterio utilizado en el preprocesamiento actual, es decir, se especifican las fechas de interés en el archivo de configuración del experimento y se utilizan expresiones regulares para la selección de los archivos correspondientes.

La creación de este nuevo programa tiene como objetivo mejorar el rendimiento y la eficiencia del sistema. Al implementar un programa dedicado exclusivamente al proceso de parsing, se evita la necesidad de realizar el parsing de todos los archivos del conjunto de datos en cada ejecución del sistema.

Como resultado, este nuevo programa genera un nuevo directorio que simula la estructura de la base de datos del sistema, donde se almacenarán los archivos procesados con un formato de nombre equivalente al utilizado anteriormente:

`auroracl_<año><mes><día><hora><minuto>.data.parquet`

El proceso de parsing de los archivos al nuevo formato proporcionado por la librería *pyarrow* se basó en la lógica de filtrado de archivos definida en la clase `Importer` para la importación y filtrado de *tweets* en el sistema. Para ello, se implementó un nuevo método denominado `parse_date_data_to_parquet`.

Este método recibe como entrada la fecha, el patrón de los archivos junto con los directorios

de origen y destino, para luego realizar el parsing de los archivos `.json` a tablas gestionadas por la biblioteca *dask* en archivos `.parquet`. Se puede ver una pequeña sección de código de esta función en el Anexo A.7, el cuál contiene la función base para la transformación de estos datos usando *pyarrow*.

Es importante tener en cuenta que algunos archivos pueden no contener objetos *JSON* indexados, lo que podría generar errores relacionados con la compresión *zlib*. Además, es posible que algunos archivos estén corruptos y no cumplan con el formato esperado. Para manejar este tipo de situaciones, se hizo uso de sentencias `try-except` para controlar y gestionar los errores durante el proceso de parsing.

4.1.2. Actualizaciones en el preprocesamiento

Como resultado de la actualización de la base de datos, se requiere llevar a cabo modificaciones en el proceso de preprocesamiento, específicamente en el manejo de datos previo al filtro de *tweets* basado en palabras clave. Esta actualización presenta un desafío adicional en comparación con el desarrollo anterior, ya que los datos no se obtienen de archivos estructurados preexistentes.

En cambio, desde ahora se trabajará con archivos en formato *parquet*, lo que implica la necesidad de definir un esquema de datos compatible con la biblioteca *pyarrow*. Para lograr esto, se implementó un esquema que refleja la estructura de los datos en los archivos *parquet* que se leerán, los cuales reemplazan por completo la estructura sintáctica brindada por los objetos *json*. Se utilizaron los componentes `pa.schema` y `pa.field` proporcionados por *pyarrow* para facilitar este proceso de definición.

Estos componentes permitirán definir y validar la estructura de los datos cuando sea necesario, asegurando que cumplan con los requisitos aceptados por la biblioteca *pyarrow*. Así, el único mayor cambio relacionado con el preprocesamiento es la actualización de la lectura de los archivos importados para el filtrado de *tweets*, la cuál ahora transforma la estructura recibida desde *parquet* a la definida con las herramientas de *pyarrow*.

También se encontraron ciertas complicaciones con el manejo de fechas dentro de estos archivos almacenados en formato *parquet*. Estas al venir en un formato de string `'EEE MMM d HH:mm:ss Z yyyy'` que no estaba contemplado para el proceso de parsing llevó a algunos errores en el filtrado de datos. Sin embargo, esto pudo resolverse con las herramientras brindadas por *pandas*, que ya tenía integrados métodos como `to_datetime` para el formateo de fechas sobre *dataframes*.

En cuanto al proceso de filtrado de *tweets* este no se verá afectado por el formato de lectura y escritura de los datos. Este permanece sin cambios relevantes, ya que su lógica de

funcionamiento no depende del formato en el que se almacenan los datos.

Con estas actualizaciones en el preprocesamiento, se ha logrado la primera etapa para la adaptabilidad en el sistema para trabajar nuevos archivos óptimos, que garantizan la integridad y la compatibilidad de los datos.

4.2. Actualizaciones de formato en el proceso de cómputo de features

En contraste con el proceso de preprocesamiento, el componente de cómputo de características en *Tsundoku* se enfrenta al desafío de trabajar con una amplia variedad de *dataframes* y archivos para calcular las características necesarias de los usuarios en el proceso de clasificación.

Esta diversidad de tablas introduce un nivel adicional de dificultad, ya que es crucial garantizar la generación precisa de cada una de ellas y evitar posibles errores en su creación. Además, la implementación actual utiliza la librería *dask* para las operaciones de cómputo y combinación de tablas, cuyos objetos no son completamente compatibles con el formato de archivos *parquet* utilizado en el sistema actualizado.

Sin embargo, la librería *pyarrow* ofrece una solución valiosa al proporcionar una excelente interoperabilidad entre diferentes librerías de manejo de tablas. Específicamente, es posible transformar un objeto `dask.dataframe` a uno de pandas con un solo comando, lo que facilita la utilización de las funciones de manejo de archivos *parquet* de *pyarrow* sin complicaciones adicionales.

Aprovechando esta capacidad, se implementaron funciones dedicadas a la lectura y escritura de archivos *parquet* en un nuevo archivo de utilidades. Estas funciones simplifican el proceso de manejo y compilación de archivos *parquet*, evitando la repetición innecesaria de código y brindando una solución eficiente para trabajar con este tipo de archivos en el componente de cómputo de características de *Tsundoku*.

Los *dataframes* involucrados en este proceso fueron explicados a grandes rasgos en la sección 3.3.1, y estas son las siguientes:

- **Tablas de métricas de los usuarios**:
  Tal como lo indica su nombre, estas concentran las métricas más relevantes del usuario incluido en la discusión:
  - `unique_users`: contiene toda la metadata de un usuario único dentro del mismo día del estudio, tal como su descripción, lugar de procedencia, etc.

- `user_name_vocabulary`: contiene los tokens ocupados en los nombres de usuarios de los usuarios, lo cuál será util para la detección de bots y para la definición de qué tipo de usuario es.
- `user_description_vocabulary`: contiene los tokens ocupados en la descripción del usuario, lo cuál será útil para la definición de postura y otros.

- **Tablas de métricas de los tweets**
  Como lo indica su nombre, estas concentran las métricas más relevantes de los *tweets* y las interacciones que estos representan en la discusión:
  - `tweets_per_user`: esta tabla indica la cantidad de *tweets* obtenidos por cada usuario del dataset en el día de la iteración.
  - `tweet_vocabulary`: esta tabla recopila el uso de cada token distinto usado en el dataset, indicando el token y su frecuencia.
  - `tweet_token_frequency`: contiene la cantidad de repeticiones de cada token generado dentro del dataset en el contenido del *tweet*. Cada columna corresponde a un token específico, mientras que las filas indican la frecuencia de cada token ocupado por cada usuario.
  - `<interaction>_counts`: contiene la cantidad de interacciones por cada usuario del experimento. Las interacciones incluyen *retweet*, *quote* y *reply*.
  - `<interaction>_edgelist`: contiene la cantidad de interacciones por ambos usuarios involucrados en el *tweet* (por ejemplo, usuario mencionado y usario que hizo la mención). Las interacciones son las mismas que en el caso anterior: *retweet*, *quote* y *reply*.
  - `user_urls`: contiene todos los links y dominios enlazados en los *tweets* dentro del estudio, los cuales podrían ser útiles para determinar el típo de usuario o su postura.
  - `daily_stats`: resume las estadísticas de las interacciones en la discusión del usuario, como la medición de menciones, respuestas recibidas, popularidad, y también métricas como cantidad de seguidores y de amigos.

Una vez revisadas todos estos *dataframes*, se procedió a la preparación del experimento y a la agrupación de las tablas finales para la clasificación. Este proceso es similar al mencionado anteriormente, ya que la agrupación y generación de matrices son independientes del formato en el que se hayan computado estos *dataframes*.

Sin embargo, surge la dificultad de asegurar que la lectura de los archivos anteriores y la estructura esperada de cada una de ellos sean adecuadas, garantizando resultados consistentes con el sistema original. En este punto, cobra mayor importancia la función de lectura de archivos *.parquet*, ya que es necesario leer *dataframes* de cómputo de características correspondientes a cada grupo de estudio y a cada día del experimento. Los *dataframes* involucradas en este proceso fueron detallas anteriormente en la sección 3.3.2.

4.3. Actualizaciones de formato en los procesos de Clasificación y Consolidación de Resultados

Debido a que el proceso de clasificación se basa en la concatenación de matrices de adyacencia como input, este no se ve afectado por el cambio de ficheros de laos *dataframes*. Esto implica que los cambios relevantes en el uso de archivos *parquet* se limitan a los outputs y tablas de recopilación de resultados.

Los *dataframes* generados por el clasificador recopilan principalmente las predicciones de cada grupo de estudio del experimento. Estos *dataframes* pueden contener una gran cantidad de datos, por lo que resulta beneficioso almacenarlos en formato *parquet*.

En el proceso de consolidación de resultados, por otro lado, si se producen cambios significativos en la actualización del formato. Esto se debe a que el *script* de evaluación del clasificador y el de detección de anomalías reciben tablas generadas por la preparación del experimento y directamente por el clasificador, respectivamente.

En el caso de la evaluación del clasificador, se requiere la entrada de las predicciones generadas por este para realizar comparaciones y calcular métricas de desempeño. Estas predicciones se almacenan en tablas específicas en formato *parquet*, lo que facilita su posterior procesamiento y análisis.

Por otro lado, en la detección de anomalías para la identificación de *bots*, se utilizan las tablas que contienen información sobre los usuarios únicos y sus relaciones con otras métricas relevantes, como las interacciones y los enlaces compartidos. Estas tablas, también almacenadas en formato *parquet*, permiten detectar patrones anómalos y determinar la presencia de usuarios sospechosos o comportamientos inusuales en la discusión.

En este último, se destaca la lectura de *dataframes* que contienen información detallada de los usuarios únicos del experimento. Para garantizar la integridad y el formato de los datos, se ha implementado una nueva estructura que permite definir y validar la estructura de los datos de los usuario en formato *parquet*. De esta manera, se asegura que los datos sean interpretados correctamente por el sistema y se evitan posibles errores o inconsistencias en la lectura y manipulación de *dataframes*.

4.4. Integración de un modelo de clasificación basado en contexto con Transformers

Tal como se señaló en la motivación de este trabajo, uno de los objetivos consiste en integrar un modelo de clasificación basado en contexto. Para lograr esto, se realizó un proceso de selección y procesamiento de datos que permitiera incorporar los *tweets* por usuario de forma cruda y sin tokenizar, tal como se detalló en la sección 3.6.

La arquitectura de *transformers* y su implementación contemplan el preprocesamiento y la tokenización de texto de forma conjunta con el modelo. Con el fin de aprovechar herramientas preentrenadas como BETO, fue necesario recopilar estos datos en bruto y ponerlos a disposición del componente de preparación del experimento para el clasificador.

4.4.1. Selección y procesamiento de datos para la clasificación

En cuanto a la selección de datos, se llevó a cabo de manera directa considerando todos los *tweets* de la discusión agrupados por cada usuario único del experimento. Debido a que esta información no se encontraba generada dentro del proceso de cómputo de features en el sistema *Tsundoku*, fue necesario ampliar dicho proceso para incluir una nueva tabla.

Esta nueva tabla, denominada `tweets_list_per_user`, consta de dos columnas. La primera columna almacena el ID de cada usuario único, mientras que la segunda columna contiene la lista de *tweets* recopilados por cada usuario durante un día de discusión. Es importante resaltar que este proceso se realiza para cada día del experimento, asegurando así la inclusión de todos los *tweets* por usuario.

La incorporación de esta nueva tabla es fundamental en la etapa de clasificación, ya que proporciona los datos necesarios para realizar un análisis contextual más preciso. Al incluir los *tweets* sin procesar, se permite que el modelo de *transformers* utilice su propia metodología de preprocesamiento y tokenización, lo cual garantiza la compatibilidad con herramientas preentrenadas y maximiza el rendimiento del clasificador.

4.4.2. Integración de embeddings de modelos pre-entrenados de transformers a la clasificación

Una vez calculado el *dataframes* `tweets_list_per_user`, se procedió a la etapa de preparación del experimento. En esta etapa, se llevó a cabo la agrupación de estos *dataframes* en una única matriz que recopile todos los *tweets* por cada usuario del estudio. Para lograr esto, se extendió este proceso para incluir una nueva matriz denominada `user_embedding`.

Para generar esta matriz, fue necesario inicializar una instancia del modelo de clasificación basado en *transformers* BERT, utilizando el modelo preentrenado con un corpus en español denominado BETO. Además de la inicialización del modelo, se importó un tokenizador preentrenado que permitió asociar el vocabulario recopilado en todo el experimento con los correspondientes *word embeddings* del modelo BETO.

A continuación, se calcularon y obtuvieron los *embeddings* para cada serie de *tweets* de cada usuario único del experimento. Este proceso implicó calcular los embeddings para cada *tweet* obtenido del usuario, generando así un *sentence embedding* que resume el *tweet* en una representación matemática. Es de esperar que este cálculo sea costoso computacionalmente, pues además de estar trabajando con strings, se están operando varios tensores para la obtención de un *embedding* general del *tweet*

Finalmente, se promediaron todos los *sentence embeddings* de cada usuario y se agregaron a la nueva matriz `user_embedding` para su posterior integración en el clasificador.

Una vez obtenida la matriz `user_embedding`, se incorpora al proceso de clasificación. Para lograr esto, se creó un método denominado `process_embedding_matrix()`, el cual carga la matriz de *embeddings* y crea un dataframe de tokens que asigna un identificador a cada valor del vector del *embedding*. Esta etapa es necesaria para indicar al clasificador el tipo de input que recibirá, y se maneja internamente como `feature_names` del clasificador.

Luego, simplemente se concatena la nueva matriz `user_embedding` al input del clasificador actual, integrando así esta matriz de *embeddings* que representa de manera matemática el discurso de cada usuario en el experimento. Esta estrategia de integración de *embeddings* amplía y mejora el sistema de clasificación, al permitir una representación más rica y contextual de los *tweets*, lo cual facilita un análisis más preciso y efectivo de los mismos.

4.5. Otras implementaciones

Además de los desarrollos explicados a lo largo de este capítulo, se realizaron varias implementaciones y mejoras en el sistema que no están directamente relacionadas con la optimización del preprocesamiento.

En primer lugar, se llevó a cabo una reorganización de los métodos del sistema y se delegaron aquellos que son utilizados en todo *Tsundoku* a una nueva carpeta llamada *utils*. En esta carpeta se recopilaron todas las funciones auxiliares útiles, que no están directamente relacionadas con el pipeline del sistema.

Entre las implementaciones incluidas en la carpeta *utils* se encuentra la creación de una clase *Timer* para el cálculo de los tiempos de ejecución. También se implementaron las

funciones de lectura y escritura de tablas, las cuales son utilizadas a lo largo de todo el proceso de clasificación y preprocesamiento. Además, se crearon nuevas estructuras y esquemas de datos utilizando *pyarrow* para la lectura de tablas pre-estructuradas.

La reorganización y ordenamiento del sistema tiene varios beneficios. En primer lugar, sigue una metodología de desarrollo ágil, lo cual facilita la comprensión del código y mejora la escalabilidad del sistema. Además, al tener las funciones auxiliares en un lugar centralizado, se mejora la legibilidad y mantenibilidad del código, permitiendo a cualquier persona comprender rápidamente su funcionamiento.

Esta organización también permitió identificar y eliminar funciones distribuidas en otros procesos que no guardaban relación, lo cual contribuyó a una mayor claridad y coherencia en el código. También se eliminaron funciones que quedaron deprecadas como métodos para el manejo de archivos *json*.

Capítulo 5

Resultados

Para validar la solución desarrollada, es necesario medir los tiempos de procesamiento antes y después de la implementación mostrada en este trabajo. Además, se debe evaluar el comportamiento del clasificador, y si la adición de *embeddings* generó un impacto positivo en sus predicciones.

5.1. Medición del tiempo de ejecución

Primero, se implementó una clase denominada `Timer` que utiliza la librería *time* de *Python*. Esta clase tiene la finalidad de registrar los tiempos de ejecución de ciertas secciones de código para poder analizar y comparar estos valores con el funcionamiento previo de *Tsundoku*.

Adicionalmente, en algunas partes del sistema se incorporó la librería *tqdm* [29] de *Python*. Esta librería proporciona una interfaz de progreso visual en la consola, lo que permite monitorear el avance de las iteraciones dentro del sistema a tiempo real. Esta herramienta brinda transparencia durante la ejecución de algunas secciones de código que pueden requerir un tiempo prolongado de ejecución, y asegura al usuario que estas progresan sin errores.

5.2. Caso de estudio

Con el fin de realizar pruebas efectivas y exhaustivas del sistema se seleccionó un caso de estudio que abarque un gran periodo de tiempo, para poder asegurar una gran cantidad de datos y poner bajo estrés el sistema. El caso de estudio elegido para este propósito fue la discusión sobre la inmigración en Chile, abarcando cuatro meses de datos dentro del período comprendido desde el 1 de enero de 2022 hasta el 30 de abril del mismo año (120 días)

Primero, fue necesario generar todos los archivos de configuración requeridos para un tema de estudio (proceso explicado en la sección 3.1), así como disponer de una cantidad significativa de datos para realizar pruebas de rendimiento y estrés. Esta gran recopilación de *tweets* fue proporcionada por el Profesor Eduardo Graells, mientras que parte de los archivos de configuración fueron facilitados por la académica Yerka Freire-Vidal, quien actualmente está llevando a cabo un trabajo de tesis de doctorado entorno a este mismo tópico, y ya ha hecho estudios en el pasado con *Tsundoku* [11].

La cantidad de datos recopilados durante este período se puede observar en la ilustración 5.1. Este periodo comprende una cantidad total de 138.426.070 de *tweets*, donde cada día del estudio supera por lo general el millón de *tweets*.

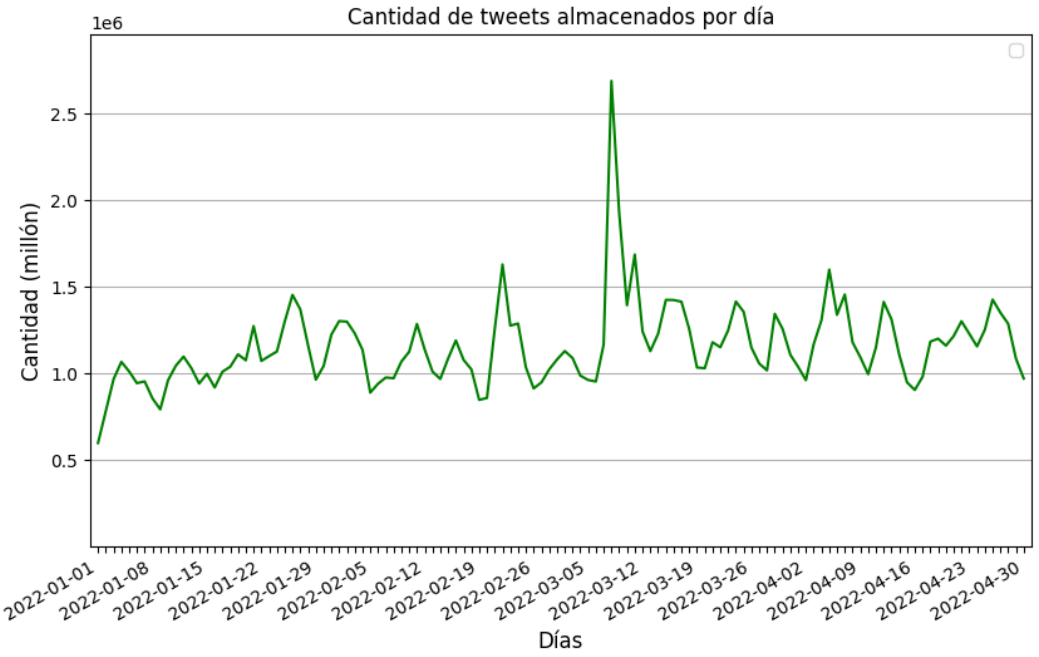

Figura 5.1: Gráfico de cantidad de *tweets* total por fecha, y previo al proceso e input al sistema *Tsudoku*

En la ilutración 5.1 se observa una serie de picos de datos a lo largo de todo el período de estudio, destacando especialmente un aumento significativo durante la semana del 5 al 12 de marzo. Este pico podría atribuirse al cambio de mando del expresidente Sebastián Piñera y la asunción del presidente Gabriel Boric, que se llevó a cabo en ese mismo periodo. Esto explicaría la gran actividad de los usuarios durante este período, y da a entender que estos los datos están efectivamente relacionados con la contingencia nacional.

5.3. Resultados

Los primeros resultados obtenidos tienen relación con el componente de preprocesamiento. Como se ha mencionado a lo largo de este trabajo, el principal objetivo es la reducción del tiempo de ejecución de este proceso, junto con la integración de un nuevo formato de almacenamiento del dataset.

5.3.1. Proceso de parsing

Primero se registró el tiempo de ejecución del proceso de parsing de todos los *tweets* que comprenden estos 120 días de estudio, los cuales pueden apreciarse en la ilustración 5.2.

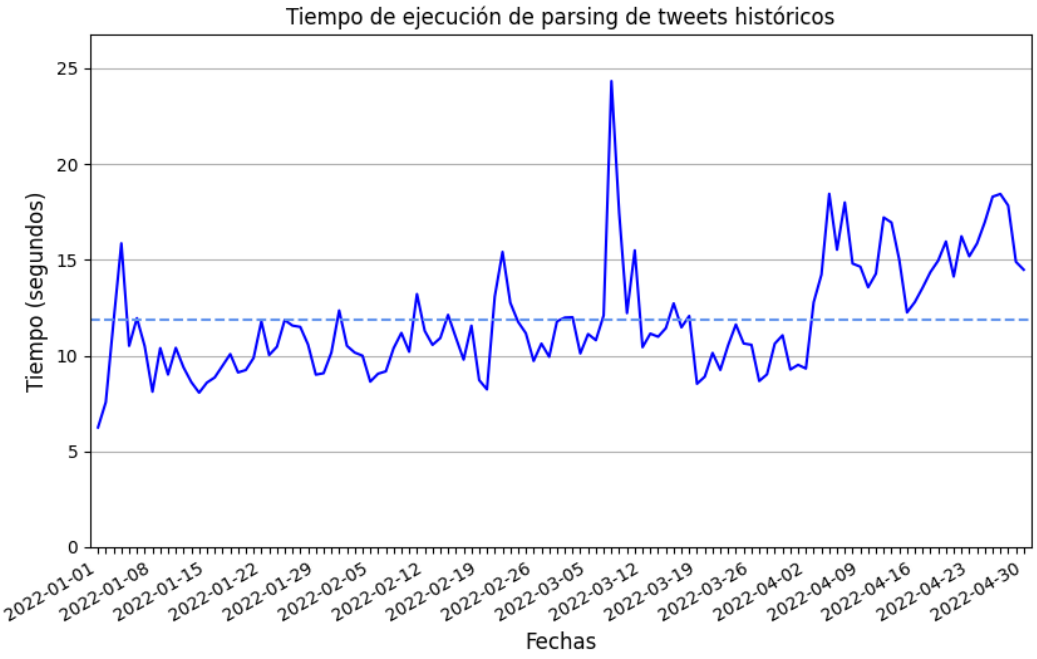

Figura 5.2: Gráfico de tiempo de ejecución del proceso de parsing por cada día del estudio

El tiempo total de ejecución fue de 1421.7 segundos, o 23 minutos con 41 segundos. Además, la media del tiempo de ejecución fue de 11.8 segundos.

Es posible notar que el tiempo de ejecución de este proceso de parsing si está relacionado con la cantidad de *tweets* recopilados históricamente dentro de este periodo. Lo anterior es de esperarse, pues a mayor cantidad de *tweets* es mayor el tiempo de procesamiento que debe realizar el sistema para la escritura de datos. En particular, llama la atención el pico de tiempo en el mismo periodo mencionado de la ilustración 5.1.

5.3.2. Preprocesamiento y filtrado de *tweets*

Ya medidos los tiempos de parsing del sistema actualizado, se procede a las mediciones relacionadas con el preprocesamiento y filtrado de *tweets* en base a *keywords*. Los tiempos de ejecución de este proceso son el principal motivo del desarrollo de este trabajo, y se espera que con el uso de un nuevo formato de lectura y escritura de datos se pueda reducir notablemente.

Así, se obtiene el siguiente gráfico de tiempos de ejecución por día:

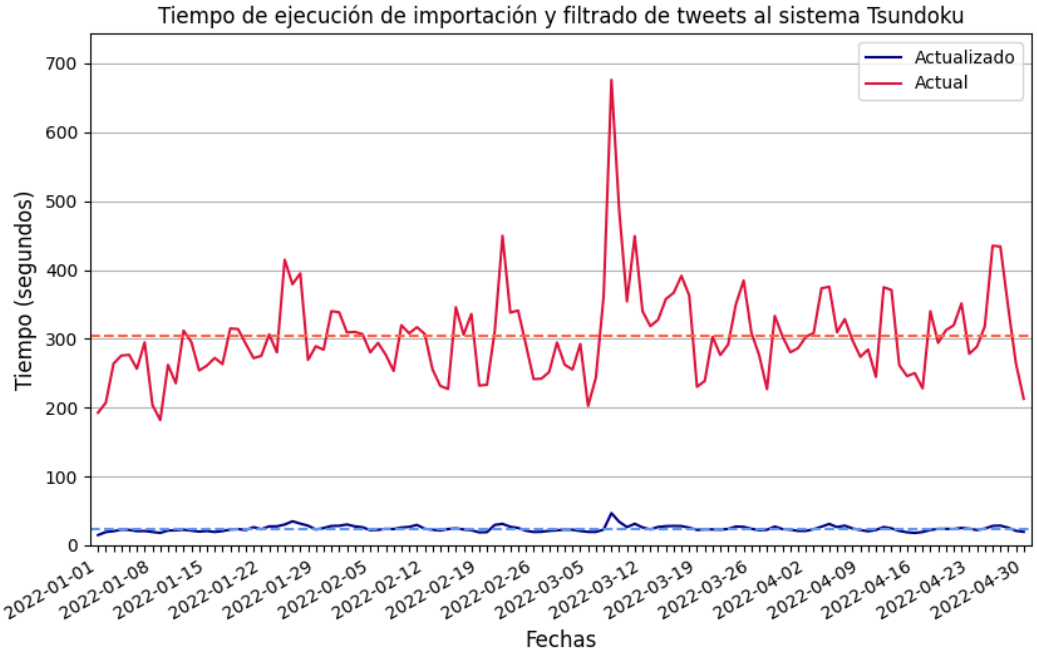

Figura 5.3: Gráfico de comparación en tiempos de procesamiento. Sistema actual versus Actualizado

En la ilustración 5.3 se destaca en color rojo los tiempos de procesamiento del sistema actual, mientras que en color azul los tiempos del sistema actualizado con el uso de archivos parquet. El tiempo de ejecución total del sistema actual fue de 36454.808 segundos (10 horas, 7 minutos, 35 segundos), mientras que en el sistema actualizado fue de 2909.617 segundos (48 minutos, 30 segundos).

La media del sistema actual y la del sistema actualizado fueron de 304 segundos (5 minutos, 4 segundos) y 24 segundos respectivamente.

Ya con esta información es posible notar una reducción sustancial del tiempo de ejecución del proceso de filtrado y preprocesamiento de *tweets*. Si bien era de esperarse que con la actualización del formato de lectura de archivos este tiempo se redujera, este resultó ser más

alto de lo que se había contemplado.

La reducción del tiempo de ejecución total fue de 92.01 % en comparación con el sistema actual. Si ponemos este número a un ejemplo práctico, podemos decir que lo que tarda el sistema actual en filtrar poco más de un mes de *tweets*, el nuevo sistema actualizado ya habrá terminado de procesar un año completo.

Además se puede apreciar que la variabilidad de los tiempos de filtrado del sistema actual es sumamente alta, y que junto con esto, la forma de este gráfico es muy similar a la de la cantidad de *tweets* históricos mostrada en la ilustración 5.1.

Lo anterior muestra efectivamente que el tiempo de este proceso se dedica en su mayoría a la lectura del corpus de *tweets* más que a la escritura de las tablas para el cómputo de features. Esto puede notarse más al medir la cantidad de *tweets* filtrados por día en el experimento, mostrado en la ilustración 5.4:

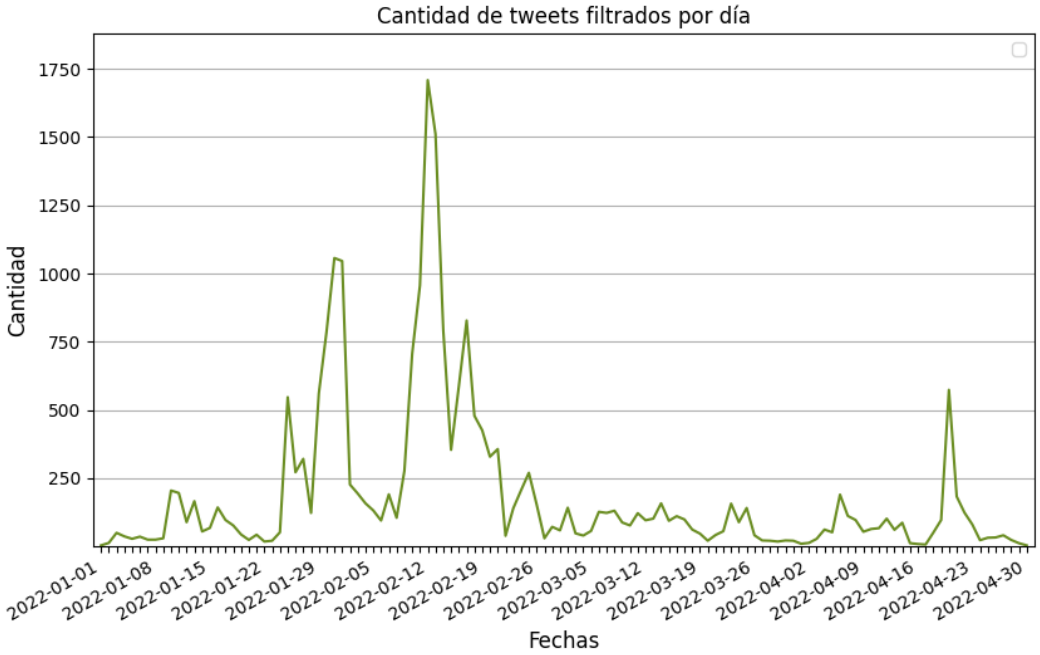

Figura 5.4: Gráfico de cantidad de *tweets* filtrados por día durante el pre-procesamiento por *keywords*

La cantidad de *tweets* filtrados, además de ser mucho más baja en comparación al corpus histórico, es variable e independiente de la cantidad de *tweets* recopilados por día. Es de esperar entonces que esta cantidad dependa de la contingencia nacional en torno al tópico de la migración en Chile.

En la ilustración 5.4 destacan tres aumentos sustanciales de datos, los cuales corresponden

a los periodos comprendidos entre finales de enero e inicios de febrero, medidados de febrero, y también finales de abril.

Si buscamos hechos relevantes ocurridos a nivel nacional dentro de estos periodos, podemos encontrar justamente eventos relacionados con el tópico de estudio. En particular, se destacan estos eventos que fueron informados en cadena nacional, obtenidos desde el noticiero CNN Chile [30]:

- **25 de Enero**: En la ciudad de Iquique, un grupo de cuatro migrantes venezolanos entre 20 y 25 años, agreden a unos Carabineros, quienes investigaban una supuesta venta de drogas en el lugar.
- **30 de Enero**: Se realizó una 'Marcha Antiinmigrante' que congregó a más de seis mil personas en Iquique.
- **11 al 14 de febrero**: Paro de camioneros producido por el asesinato de un camionero en las cercanías de Antofagasta por tres migrantes de nacionalidad venezolana.
- **21 de abril**: El presidente Gabriel Boric realiza su primera gira regional a la ciudad de La Serena, en la Región de Coquimbo.

5.3.3. Cómputo de features y preparación del clasificador

Ya estudiado el tiempo de ejecución del preprocesamiento, se pasa a medir los tiempos de ejecución de cómputo de features y la generación de matrices para el clasificador. Recordar que dentro de este proceso se leen los *tweets* filtrados por día y se generan *dataframes* de recopilación de características importantes.

A continuación se muestran las ilustraciones 5.5 y 5.6, las que grafican los tiempos de ejecución del proceso ya mencionado y muestran también el tiempo de cómputo de las dos distintas métricas descritas en la sección 3.3.

Por un lado, el sistema actual tardó en total unos 1658 segundos (27 minutos, 38 segundos) en computar las métricas por cada usuario, con una media de 13.8 segundos por día. Además, las métricas con mayor tiempo de ejecución resultaron ser las de *tweets* con unos 1500 segundos (25 minutos).

También es posible apreciar que el tiempo de ejecución depende directamente de la cantidad de *tweets* filtrados, puesto que se observan los mismos picos de datos en los periodos destacados de la ilustración 5.4.

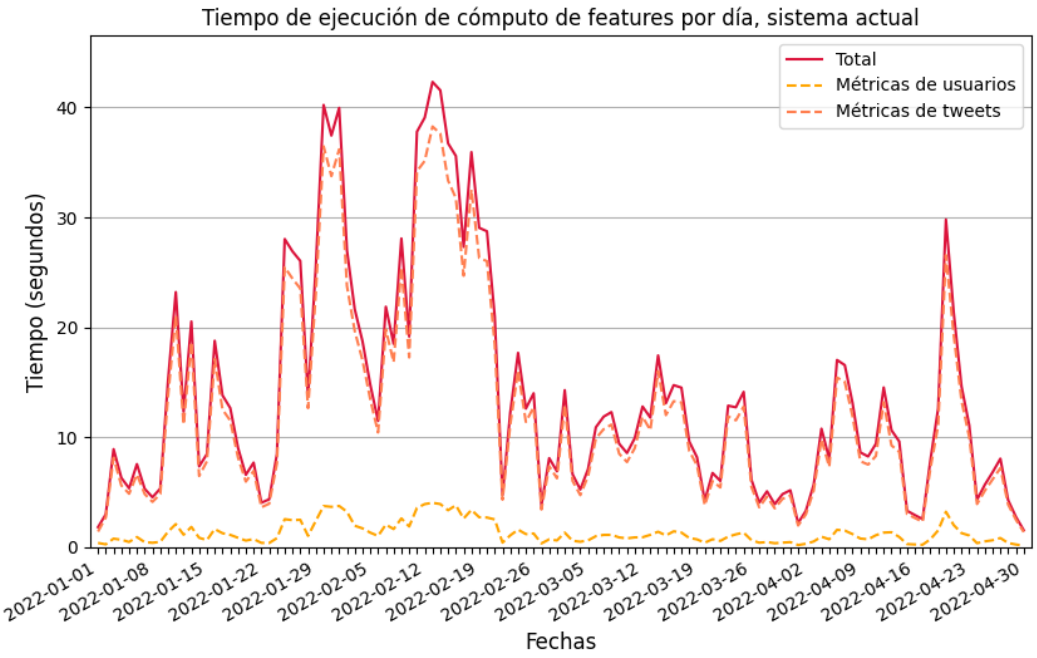

Figura 5.5: Gráfico de tiempos de ejecución en el cómputo de características por día, sistema actual

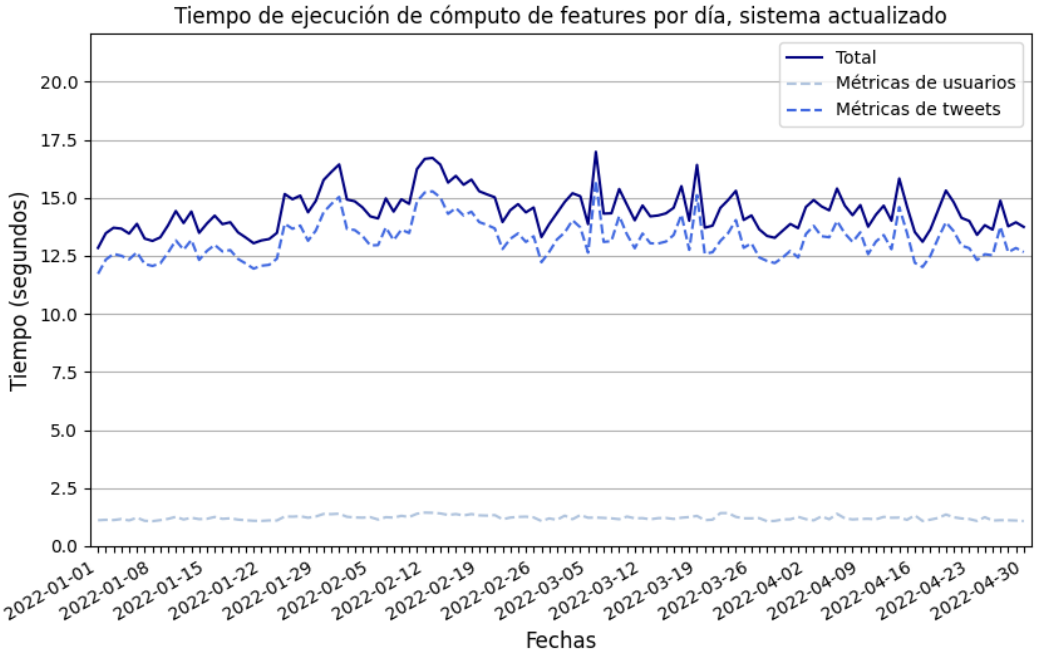

Figura 5.6: Gráfico de tiempos de ejecución en el cómputo de características por día, sistema optimizado

En el caso del sistema actualizado, este tardó en total unos 1732 segundos (28 minutos, 53 segundos) con una media de 14.4 segundos por día. Nuevamente las métricas con mayor tiempo de ejecución fueron las de *tweets* con unos 1586 segundos (26 minutos, 26 segundos) con una media de 13.2 segundos.

Si bien este proceso fue unos dos minutos más lento que el original, es necesario tener en cuenta que además de procesar las métricas del sistema actual, se integró el cómputo del listado de *tweets* por usuario para la generación de *embeddings* por cada usuario del estudio.

Junto con los datos anteriores, es posible notar que la variabilidad en los tiempos de cómputo de features del sistema actualizado es mucho menor a la del sistema actual. Esto puede apreciarse mejor en la ilustración 5.7, donde se grafican ambas curvas en conjunto:

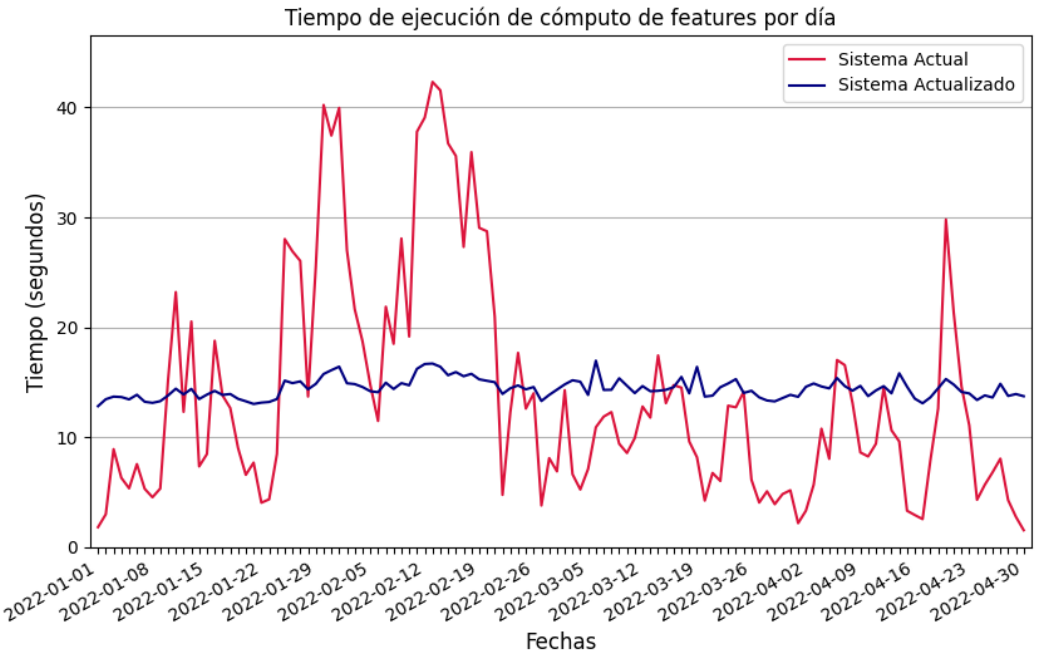

Figura 5.7: Gráfico de comparación de tiempos de ejecución en el cómputo de características

Esto muestra que, si bien en algunos casos con una baja cantidad de *tweets* el uso de tablas manejadas con archivos json es más eficiente que el actual con ficheros parquet, también el uso de archivos json es sumamente ineficiente en el caso contrario, con cantidades de *tweets* y features mayor. Así es posible apreciar la eficiencia del formato parquet, el cual permite operaciones de lectura y escritura estables.

Finalmente, es posible agrupar todas estas métricas y generar las matrices para el entrenamiento del clasificador. La agrupación de cada una de estas tablas es de forma secuencial, y se excluye dentro de la ilustración 5.8 la generación de la matriz `user_embedding`:

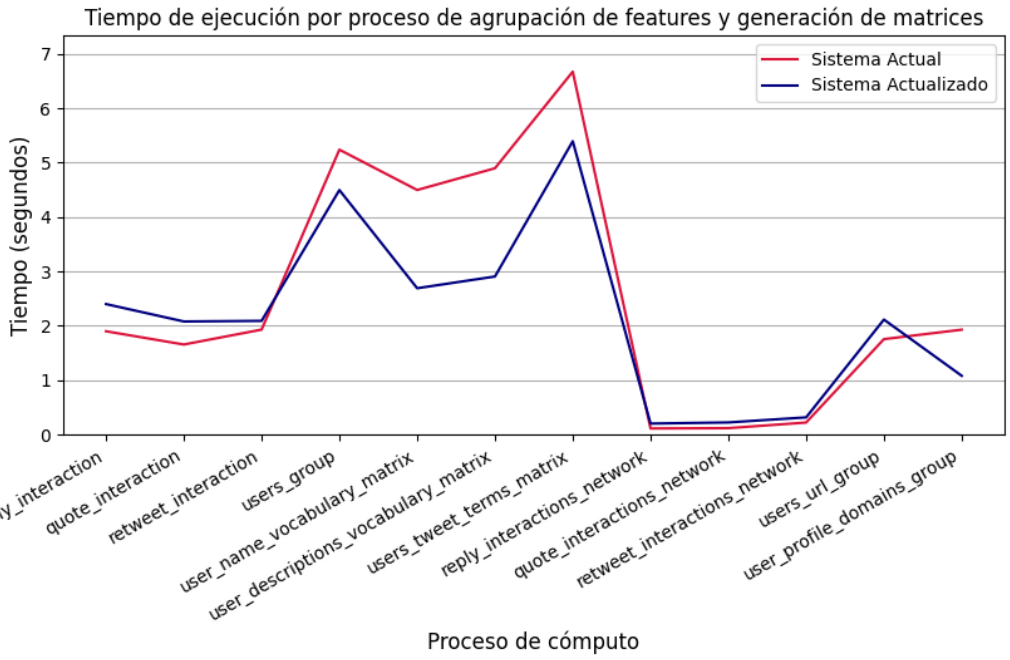

Figura 5.8: Gráfico de tiempos de ejecución por proceso de cómputo

La generación de la matriz `user_embedding` también se llevó a cabo dentro de este componente, y este proceso pudo generar un *embedding* de 756 dimensiones a 9587 usuarios únicos dentro del experimento. Este proceso, sin embargo, fue bastante costoso computacionalmente, pues fue necesario iterar por cada usuario y por cada uno de sus *tweets* realizando una serie de cálculos entre tensores. La generación de esta matriz tardó 2053 segundos (34 minutos, 13 segundos).

Es posible resumir y comparar los tiempos de ejecución totales del sistema en la tabla 5.1.

Tabla 5.1: Distribución de tiempos de ejecución por proceso en segundos

Proceso	**Sistema Actual (s)**	**Sistema Actualizado (s)**
Parsing de BBDD	-	1.421
Preprocesamiento	36.454	2.909
Cómputo de Features de Usuarios	158	146
Cómputo de Features de Tweets	1.499	1.586
Generación de Matrices	25	31
Generación de Matriz usuario-embedding	-	2.053
Tiempo total	**38.136**	**8.146**
Tiempo total en horas	**10:35:36**	**2:15:46**

El tiempo total de compilación del estudio de prueba utilizando el sistema actual fue de 38.136 segundos (10 horas, 35 minutos, 36 segundos), mientras que con el sistema actualizado se redujo a 8.146 segundos (2 horas, 15 minutos, 46 segundos).

Es evidente que el sistema desarrollado en este trabajo logró una enorme reducción en el tiempo de cómputo del preprocesamiento y filtrado de *tweets*. Sin embargo, es importante destacar que esta mejora también incluyó la integración de una nueva matriz de *embeddings*, lo cual implicó un costo significativo en términos de tiempo de compilación.

A pesar de esto, el tiempo de ejecución total del estudio fue un 78.6 % más eficiente que antes, alcanzando así los objetivos propuestos en este trabajo. Cabe mencionar que el proceso de parsing de los conjuntos de *tweets* es único para cada fecha registrada históricamente, lo que significa que cualquier estudio que utilice estos archivos no requerirá volver a ejecutar este proceso. Esto permite un mayor aprovechamiento de los recursos y una mayor eficiencia en futuros estudios y análisis de datos.

5.3.4. Evaluación del clasificador

Para la evaluación del clasificador *XGBoost* se llevó a cabo el cálculo de las métricas *precision*, *recall* y *f1-score*. Para esto se utilizó el método `classification_report` de la librería `sklearn.metrics`. Este método toma como entrada las etiquetas verdaderas y las etiquetas predichas por el clasificador, y genera un informe detallado con las métricas mencionadas para cada clase.

Es importante destacar que el proceso de evaluación se lleva a cabo mediante una validación cruzada. Esto implica dividir el conjunto de datos en k subconjuntos, donde el modelo se entrena k veces utilizando diferentes combinaciones de subconjuntos de entrenamiento y se evalúa en los subconjuntos de prueba correspondientes. Para este experimento, se ocuparon 5 folds para esta evaluación cruzada.

La tabla 5.2 entrega las métricas de evaluación del clasificador con el sistema actual, para el caso del grupo de estudio con el rendimiento más bajo del sistema correspondiente al de lugar de procedencia. Tal como se mencionó en la sección 2.2.5.3, este presenta precisión bastante baja comparado con los resultados de la medición de postura 5.3.

Tabla 5.2: Métricas de evaluación del clasificador actual, grupo de estudio *location*

	precision	recall	f1-score
antofagasta	0.13	0.31	0.18
araucania	0.12	0.18	0.14
arica	0.01	0.04	0.01
atacama	0.00	0.00	0.00
aysen	0.00	0.00	0.00
biobio	0.03	0.07	0.04
coquimbo	0.01	0.04	0.02
loslagos	0.02	0.06	0.03
losrios	0.03	0.06	0.04
magallanes	0.00	0.00	0.00
maule	0.03	0.07	0.04
noise	0.76	0.53	0.59
nuble	0.00	0.00	0.00
ohiggins	0.01	0.02	0.01
rm	0.55	0.19	0.28
tarapaca	0.17	0.31	0.21
valparaiso	0.10	0.15	0.12
accuracy			0.23
macro avg	0.12	0.12	0.10
weighted avg	0.42	0.23	0.27

Tabla 5.3: Métricas de evaluación del clasificador actual, grupo de estudio *stance*

	precision	recall	f1-score
empathy	0.97	0.97	0.97
threat	0.99	0.99	0.99
accuracy			0.99
macro avg	0.98	0.98	0.98
weighted avg	0.99	0.99	0.99

La tabla 5.4 por otro lado muestra las métricas de evaluación utilizando el sistema actualizado que incorpora *user_embeddings*.

Tabla 5.4: Métricas de evaluación del clasificador actualizado, grupo de estudio *location*

	precision	recall	f1-score
antofagasta	0.16	0.23	0.17
araucania	0.13	0.16	0.13
arica	0.00	0.00	0.00
atacama	0.41	0.00	0.00
aysen	0.61	0.00	0.00
biobio	0.01	0.02	0.01
coquimbo	0.03	0.07	0.04
loslagos	0.01	0.02	0.01
losrios	0.01	0.03	0.01
magallanes	0.10	0.03	0.05
maule	0.03	0.05	0.03
noise	0.68	0.51	0.56
nuble	0.00	0.00	0.00
ohiggins	0.02	0.04	0.02
rm	0.54	0.33	0.40
tarapaca	0.26	0.27	0.20
valparaiso	0.12	0.07	0.07
accuracy			0.30
macro avg	0.18	0.11	0.10
weighted avg	0.41	0.29	0.32

La precisión global aumentó en 7 puntos, mientras que la precisión ponderada aumentó en 5 puntos. Se observa que la precisión en la detección de algunas regiones fue considerablemente mejor en comparación con el sistema actual, especialmente en las regiones de *atacama* y *aysén*, mientras que la precisión en la detección de ruido fue ligeramente menor.

Estos resultados sugieren que, aunque se observó una mejora en el proceso de clasificación al incorporar representaciones semánticas con *embeddings*, esta mejora no fue suficiente para generar un impacto significativo en el sistema. Es posible que el uso de *embeddings* no sea realmente útil en clasificadores basados en árboles de decisión, y que sea necesario explorar otras herramientas o métodos, como el *fine-tuning* o incluso utilizar un clasificador basado en redes neuronales recurrentes.

Tabla 5.5: Métricas de evaluación del clasificador actualizado, grupo de estudio *stance*

	precision	recall	f1-score
empathy	0.95	0.97	0.96
threat	0.99	0.98	0.99
accuracy			0.97
macro avg	0.96	0.97	0.97
weighted avg	0.98	0.98	0.98

Por otro lado, no se observaron grandes cambios en la evaluación de los demás grupos de estudio. Solo se destaca una ligera disminución en la precisión global en la clasificación de postura, lo cual podría atribuirse al hecho de que la inclusión de un gran número de características en el clasificador no necesariamente se traduce en mejoras en la clasificación. Es necesario considerar otras estrategias o enfoques para abordar esta situación.

Capítulo 6

Discusión

6.1. Implicancias

El desarrollo de este trabajo conlleva una serie de implicancias tanto teóricas como prácticas. Este sistema ha sido utilizado por académicos de diversas áreas para analizar discusiones específicas en la plataforma. El ejemplo más destacado de su aplicación se evidencia en el trabajo titulado *'Bots Don't Vote'* presentado en detalle en la sección 2.2.5, donde se empleó *Tsundoku* para examinar el comportamiento de los usuarios en torno a la discusión de una nueva constitución en Chile.

Además de esto, la existencia de esta amplia conglomeración de datos recopilados por el profesor Eduardo Graells durante este período resulta de gran interés para sociólogos, lingüistas, científicos de datos y otros académicos de diversas disciplinas. Este período histórico constituye una oportunidad invaluable para futuras mejoras e implementaciones que permitan un mejor entendimiento del desarrollo de la discusión en *Twitter* y su impacto en la sociedad.

La disponibilidad de estos datos ofrece un potencial significativo para profundizar en el análisis y la comprensión de la evolución de dicha discusión en la plataforma, así como para investigaciones adicionales en el campo de la sociología y otras áreas relacionadas.

A lo anterior se suman los importantes cambios ocurridos en *Twitter* durante el último año, en particular en relación con la disponibilidad de datos históricos y la continuidad de la API de *Twitter* con sus nuevas políticas de difusión. Entre estos cambios destaca las nuevas limitaciones al servicio actual de la API v1.1, así como incrementar los valores de suscripción de la nueva API v2.0 (ver Anexo B.1).

Por lo tanto, la existencia de este conglomerado de *tweets* se vuelve sumamente atractiva

para aquellos interesados en estudiar este período de tiempo en la plataforma, ya que el acceso directo a estos datos en *Twitter* se ha vuelto en un problema. La posibilidad de utilizar este conjunto de datos dentro del sistema *Tsundoku* ofrece una valiosa oportunidad para investigar y analizar con mayor profundidad esta etapa histórica en Chile, brindando una perspectiva única y valiosa para futuras investigaciones académicas.

6.2. Limitaciones

Si bien *Tsundoku* ha demostrado ser una herramienta útil en el análisis de discusiones en *Twitter*, existen varias limitaciones que deben abordarse para mejorar aún más su usabilidad.

Una de las principales limitaciones del sistema *Tsundoku* radica en el acceso a los datos históricos de *Twitter*, lo cual es fundamental para realizar un estudio que abarque la mayor cantidad de usuarios y discusiones posibles. Esta limitación es de suma relevancia, ya que actualmente todos los datos son gestionados por el profesor Eduardo Graells y algunos otros académicos que han trabajado con *Tsundoku*. La dependencia de un grupo reducido de expertos en el manejo de los datos puede generar demoras y dificultades en la disponibilidad de los mismos, lo que puede obstaculizar el desarrollo de posibles investigaciones.

También, entre los aspectos técnicos del sistema, se encuentra en la generación de los archivos de configuración. Actualmente, esta tarea se realiza de forma manual, lo cual implica que la filtración de los tópicos de discusión depende de la intervención directa de los usuarios. La generación de una configuración adecuada que incluya todas las *features* y palabras clave necesarias para un filtrado preciso de la discusión requiere un proceso exhaustivo de investigación y generación de criterios, lo que puede resultar tedioso y afecta a la usabilidad del sistema.

Por otro lado, la integración de *word embeddings* al pipeline actual de detección de postura no implicó una mejora importante en el proceso de clasificación. Lo anterior podría deberse a múltiples factores, tales como que el modelo BETO no está entrenado para comprender sentencias cortas con formato de *microblogging* y sea necesario realizar un proceso de entrenamiento específico para este tipo de estudio (*fine tunning*). Debido al tiempo dedicado al resto de los objetivos principales de este trabajo, no fue posible realizar un análisis más exhaustivo al respecto.

6.3. Trabajo Futuro

Frente a las limitaciones actuales de *Tsundoku* y las implicancias mencionadas anteriormente, se identifican áreas de trabajo futuro que podrían contribuir a mejorar el sistema.

6.3.1. Trabajo a corto plazo

Una posible mejora importante sería el desarrollo de un programa capaz de generar tanto *keywords* relacionadas con un tópico específico como *keywords* que determinen las características de un grupo de estudio particular. Este programa recibiría como input una serie de palabras o frases relacionadas con el tópico y, en base a ello, generaría automáticamente los archivos de configuración necesarios.

Respecto al proceso de clasificación, sería beneficioso explorar alternativas al proceso de clasificación propuesto en este trabajo. Realizar un entrenamiento al modelo BETO para su especialización en la caracterización de sentencias cortas podría resultar en mejores resultados en la evaluación del clasificador. Esto abriría nuevas posibilidades de análisis, permitiendo abordar casos más complejos que requieren una comprensión más profunda del lenguaje y las interacciones en línea.

Por otro lado, se destacó la existencia de nuevas limitaciones en *Twitter* debido a los grandes cambios en la forma de obtener datos históricos. Sin embargo, estos cambios también abren la posibilidad de brindar acceso público a la gran cantidad de *tweets* recopilados durante los años 2020 y 2022 en Chile. Esto permitiría que académicos e investigadores interesados en utilizar este conjunto de *tweets* puedan acceder a ellos y realizar sus propias investigaciones.

Con lo anterior, nace la oportunidad de desplegar una posible base de datos, así como también el propio *Tsundoku* a un servidor público. Para lograr esto, sería necesario utilizar herramientas de contenedores como *Docker*, que permiten montar y configurar correctamente el sistema, y establecer la conexión con la base de datos.

El despliegue de *Tsundoku* en un servidor público es un proceso complejo que requiere tiempo y recursos para costear servicios brindados por plataformas como *Heroku*, *Firebase* o incluso *AWS* (Amazon Web Services), que son las que ofrecen servicios de alojamiento y administración de servicios computacionales. Por lo tanto, es importante considerar la disponibilidad de recursos financieros para mantener el funcionamiento del sistema y garantizar su accesibilidad a los usuarios interesados.

6.3.2. Visión

Aunque *Tsundoku* proporciona una sólida estructura para llevar a cabo diversas investigaciones en el ámbito de las discusiones en línea, es importante destacar que este sistema fue diseñado específicamente para la plataforma *Twitter*. Si bien esta adaptación asegura un enfoque coherente en el estudio de interacciones en dicha red, también introduce ciertas limitaciones.

La adaptación *ad hoc* de *Tsundoku* a *Twitter* conlleva restricciones en la generalización de sus resultados y en su aplicabilidad a otras plataformas o entornos de discusión en línea. Esto se debe a que cada red social posee características únicas en términos de formato de mensajes, tipos de interacciones y comportamientos de los usuarios.

Como visión, se podría considerar generalizar y extender el sistema para abarcar múltiples plataformas de discusión en línea. Al implementar un enfoque más flexible y modular que pueda acomodar diferentes formatos y características específicas de distintas redes sociales, se abriría la puerta a un análisis más completo y holístico de las interacciones en línea. Esto no solo permitiría una mayor diversidad de investigaciones, sino también una mejor comprensión de las similitudes y diferencias entre las distintas plataformas en términos de dinámicas de discusión, discurso y opinión.

Capítulo 7

Conclusiones

Tras revisar el trabajo presentado en este documento, se puede concluir que se logró una reducción significativa en el tiempo de compilación del sistema *Tsundoku*. La integración de herramientas de procesamiento masivo de datos con la librería *pyarrow*, y el uso de un nuevo formato de archivos *parquet* permitieron agilizar eficientemente los procesos de lectura y escritura de tablas en todos los componentes del sistema, lo que resultó en una disminución del 92 % del tiempo de ejecución asociado al filtrado de *tweets*.

La actualización prácticamente total del formato de archivos utilizado internamente resultó ser un desafío considerable, especialmente en términos de compatibilidad con bibliotecas de manejo de datos como *dask* y *pandas*. No obstante, se logró adaptar por completo el proceso de preprocesamiento y clasificación, manteniendo su funcionalidad esperada en términos de resultados, pero con tiempos de lectura mejorados.

Por otro lado, los diferentes enfoques aplicados al proceso de clasificación mediante el uso de nuevos modelos de lenguaje basados en contexto demostraron ser no tan efectivos. Pese a que el cálculo de *embeddings* con *transformers* requirió un uso considerable de recursos, los resultados de la integración de estos a la clasificación no revelaron una mejora notoria. Es necesario estudiar otros enfoques y nuevas herramientas que permitan mejorar el sistema actual.

Durante el desarrollo de este trabajo, se enfrentaron diversas dificultades al comprender y modificar el código escrito por terceros. Este proceso brindó una valiosa experiencia personal al tratar con sistemas complejos y subrayó la importancia de contar con una documentación clara y completa. Se pudo observar que la falta de documentación dificulta la comprensión de las funcionalidades y el propósito de cada componente del programa, lo cual puede obstaculizar su mantenimiento y adaptación en el futuro.

El desarrollo de este trabajo permitió registrar y organizar de manera detallada el funcionamiento de *Tsundoku*. Esto permitirá a los usuarios futuros comprender fácilmente cómo funciona el sistema y utilizarlo de manera efectiva para sus propios proyectos de investigación.

En resumen, este trabajo ha logrado optimizar y mejorar el sistema *Tsundoku* mediante la implementación de estrategias innovadoras. Se han reducido significativamente los tiempos de compilación y se ha mejorado la escalabilidad del sistema. Aunque algunos enfoques no fueron tan efectivos como se esperaba, este trabajo ha proporcionado valiosa experiencia y ha resaltado la importancia de la documentación y las buenas prácticas de programación. Estas mejoras sientan las bases para futuras investigaciones en la clasificación de discursos en redes sociales, ampliando las posibilidades del sistema *Tsundoku*.

Bibliografía

[1] Kavada, A., "Creating the collective: social media, the occupy movement and its constitution as a collective actor," Information, Communication & Society, vol. 18, no. 8, pp. 872–886, 2015, doi:10.1080/1369118X.2015.1043318.

[2] Küçük, D. y Can, F., "Stance detection: A survey," ACM Comput. Surv., vol. 53, 2020, doi:10.1145/3369026.

[3] Graells-Garrido, E., Baeza-Yates, R., y Lalmas, M., "Every colour you are: Stance prediction and turnaround in controversial issues," en 12th ACM Conference on Web Science, WebSci '20, (New York, NY, USA), p. 174–183, Association for Computing Machinery, 2020, doi:10.1145/3394231.3397907.

[4] Chen, T. y Guestrin, C., "Xgboost: A scalable tree boosting system," en Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, (New York, NY, USA), p. 785–794, Association for Computing Machinery, 2016, doi:10.1145/2939672.2939785.

[5] Aiello, L., Deplano, M., Schifanella, R., y Ruffo, G., "People are strange when you're a stranger: Impact and influence of bots on social networks," ICWSM 2012 - Proceedings of the 6th International AAAI Conference on Weblogs and Social Media, 2014.

[6] Graells-Garrido, E. y Baeza-Yates, R., "Bots don't vote, but they surely bother! a study of anomalous accounts in a national referendum," en 14th ACM Web Science Conference 2022, WebSci '22, (New York, NY, USA), p. 302–306, Association for Computing Machinery, 2022, doi:10.1145/3501247.3531576.

[7] Chavoshi, N., Hamooni, H., y Mueen, A. A., "Debot: Twitter bot detection via warped correlation," 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 817–822, 2016.

[8] Varol, O., Ferrara, E., Davis, C., Menczer, F., y Flammini, A., "Online human-bot interactions: Detection, estimation, and characterization," Proceedings of the International AAAI Conference on Web and Social Media, vol. 11, pp. 280–289, 2017, https://ojs.aaai.org/index.php/ICWSM/article/view/14871.

[9] Liu, F. T., Ting, K. M., y Zhou, Z.-H., "Isolation-based anomaly detection," ACM Trans. Knowl. Discov. Data, vol. 6, 2012, doi:10.1145/2133360.2133363.

[10] Liu, F. T., Ting, K. M., y Zhou, Z.-H., "Isolation forest," en 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422, 2008, doi:10.1109/ICDM.2008.17.

[11] Freire-Vidal, Y., Graells-Garrido, E., y Rowe, F., "A framework to understand attitudes towards immigration through twitter," Applied Sciences, vol. 11, no. 20, 2021, doi:10.3390/app11209689.

[12] Graells-Garrido, E., "Tsundoku: Bots don't vote, but they surely bother!," 2022, https://github.com/zorzalerrante/tsundoku.

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., y Polosukhin, I., "Attention is all you need," 2017, https://doi.org/10.48550/arXiv.1706.03762.

[14] Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K., "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019, https://doi.org/10.48550/arXiv.1810.04805.

[15] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., y Pérez, J., "Spanish pre-trained bert model and evaluation data," en PML4DC at ICLR 2020, 2020, https://github.com/dccuchile/beto.

[16] Lai, M., Cignarella, A. T., Hernández Farías, D. I., Bosco, C., Patti, V., y Rosso, P., "Multilingual stance detection in social media political debates," Computer Speech Language, vol. 63, p. 101075, 2020, doi:https://doi.org/10.1016/j.csl.2020.101075.

[17] Lai, M., Hernández Farías, D. I., Patti, V., y Rosso, P., "Friends and enemies of clinton and trump: Using context for detecting stance in political tweets," en Advances in Computational Intelligence (Sidorov, G. y Herrera-Alcántara, O., eds.), (Cham), pp. 155–168, Springer International Publishing, 2017.

[18] Graells-Garrido, E., Baeza-Yates, R., y Lalmas, M., "How representative is an abortion debate on twitter?," en Proceedings of the 10th ACM Conference on Web Science, WebSci '19, (New York, NY, USA), p. 133–134, Association for Computing Machinery, 2019, doi:10.1145/3292522.3326057.

[19] "Bots don't vote, but they surely bother! a study of anomalous accounts in a national referendum," Montreal AI Ethicts Institute, 2022, https://montrealethics.ai/bots-dont-vote-but-they-surely-bother-a-study-of-anomalous-accounts-in-a-national-referendum/.

[20] "Resultados plebiscito contitución política de chile año 2020," Servicio Electoral de Chile, 2020, https://app.powerbi.com/view?r=eyJrIjoiN2Y0ODM3MmUtZmY3YS00N2ZjLWJjNjMtM2Y4MjU3Y2UyNjEzIiwidCI6ImVhZjg3OWJkLWQzZWMtNDY1MC1iMTI5LTEzZGZkZjQ4NTlmZSJ9.

[21] "Dask library.", https://www.dask.org/.

[22] "Pandas libary.", https://pandas.pydata.org/.

[23] “Apache arrow.”, https://arrow.apache.org/.

[24] “Sckikit learn, machine learning library for python.”, https://scikit-learn.org/.

[25] “pyahocorasick library.”, https://pyahocorasick.readthedocs.io.

[26] García Ríos, N., “Tsundoku: Bots don’t vote, but they surely bother! new optimized version,” 2023, https://github.com/Nicolas-Francisco/tsundoku.

[27] “Windows subsystem for linux.”, https://learn.microsoft.com/en-us/windows/wsl/.

[28] “Anaconda for python.”, https://www.anaconda.com/.

[29] “tqdm library for python.”, https://tqdm.github.io/.

[30] “Cnn chile.”, https://www.cnnchile.com/.

Glosario

- ***tweet*:** Un *tweet* es un mensaje de estado en la plataforma *Twitter*, que puede tener imágenes, videos, enlaces y texto de hasta 280 caracteres. Los usuarios que estén interesados en los *tweets* de otras personas o empresas pueden convertirse en sus seguidores o *followers*. Estos corresponden a la forma más importante de comunicación en la plataforma, y son de motivo de estudio debido a las interacciones que generan.

Ejemplo de un *tweet*

- ***retweet*:** Un *retweet* corresponde a un *tweet* compartido públicamente por otro usuario. Es la forma más común de difundir noticias y opiniones dentro de la plataforma. Dentro de un *retweet* existen dos usuarios que interactúan: el usuario que publicó el *tweet* originalmente, y el usuario que lo difundió.

Ejemplo de un *retweet*

- ***bot*:** Un *bot* es un programa que realiza tareas repetitivas y predefinidas. Los *bots* están diseñados para imitar o sustituir el accionar humano, operando en forma automatizada. En el contexto de *Twitter*, estos son usuarios que generan contenido como *tweets* o *retweets*, pero que simulan ser personas reales

Ejemplo de un *bot* en *Twitter*

- **Precisión o *precision*:** La precisión es una métrica de evaluación de modelos de clasificación, y que mide que tan preciso es el modelo respecto a los resultados obtenidos. Se calcula midiendo la relación entre las etiquetas predichas correctamente respecto a

la cantidad total de etiquetas:

$$\text{Precision} = \frac{\text{Verdaderos Positivos}}{(\text{Verdaderos Positivos} + \text{Falsos Positivos})}$$

- **Exhaustividad o *recall*:** La exhaustividad es una métrica de evaluación de modelos de clasificación, y mide la proporción de etiquetas que el clasificador puede identificar. Se calcula midiendo la relación de verdaderos positivos frente a falsos negativos:

$$\text{Recall} = \frac{\text{Verdaderos Positivos}}{(\text{Verdaderos Positivos} + \text{Falsos Negativos})}$$

- **Valor-F o *F1-Score*:** El Valor-F corresponde a una métrica de evaluación de modelos de clasificación que integra la precisión y la exhaustividad en un solo valor. Este mide escencialmente que tan certero es el clasificador:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Serie de Taylor:** una Serie de Taylor es una aproximación de funciones mediante una serie de potencias o suma de potencias enteras de polinomios. Dicha suma se calcula a partir de las derivadas de la función para un determinado valor o punto a suficientemente derivable sobre la función y un entorno sobre el cual converja la serie:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

  Donde $f^{(n)}$ corresponde a la derivada n-ésima de la función f evaluada en el punto a, y $n!$ corresponde al factorial de n,

Anexos

Anexo A. Códigos fuentes

A.1. Archivo de configuración principal config.toml

Código A.1: snippet del archivo config.toml

```
[project]
name = 'Stress Test Project'

[project.path]
config = '/mnt/c/Users/nicol/Escritorio/Tsundoku/tsundoku/example_proyect'
data = '/mnt/c/Users/nicol/Escritorio/Tsundoku/data/tweets_data/2022_flattened'

[project.content]
term_files = ['keywords.txt']
accepted_lang = ['es']
timezone = 'America/Santiago'

[project.content.user_matrix]
stopwords_file = 'stopwords.txt'
lru_size = 500

[project.environment]
n_jobs = 3

```

A.2. Archivo de configuración del experimento y grupos de estudio

Código A.1: snippet del archivo experiments.toml

```
[experiments]

[experiments.test]
key = 'test'
folder_pattern = '*'
folder_start = '2022-01-25'
folder_end = '2022-01-31'
...

[experiments.test.topic_modeling]
n_topics = 200
min_tweets = 10
max_tweets_quantile = 0.98
min_users = 1000
...

[thresholds]
name_tokens = 50
description_tokens = 50
tweet_tokens = 50
tweet_domains = 50
...
edge_weight = 3

[stance]

[stance.xgb]
learning_rate = 0.15
max_depth = 3
...
seed = 42
objective = 'binary:logistic'

[stance.pipeline]
early_stopping_rounds = 10
eval_fraction = 0.1
threshold_offset_factor = 0.05
...

```

A.3. Archivo de configuración del tópico en estudio

Código A.1: snippet del archivo `keywords.txt`, ej: discusión de migración

```
#?inmigra[\w]+
#?migrac[\w]+
#?migrat[\w]+
[\w]+culturalidad
interculturalidad
xenofob[\w]+
racism[\w]+
(?:^|\s|\W|¡‘|¿‘)hait[\w]+
(?:^|\s|\W|¡‘|¿‘)[ck]reole?[\W]
(?:^|\s|\W|¡‘|¿‘)lepr[\w]+
promigrante
#migración
#migranteschile
#todossomosmigrantes
#redmigrante
#migracionpdi
#chilesinbarreras
#migraciónlaboral
#leydemigración
...

```

A.4. Archivo de configuración de grupo de estudio

Código A.1: snippet del archivo groups/stance.toml

```
[empathy]
title = 'Empathy'
order = ['empathy', 'undisclosed', 'threat']
undisclosed_title = 'Indecisos(as)/Sin Postura Explícita'
threshold = 0.3 # low value to avoid applying the threshold

[empathy.tweet_tokens]
must_have = ['#bienvenidosachile',
    '#chilesinbarreras',
    '#noalaxenofobia',
    '#interculturalidad',
    ...
]

[empathy.account_ids]
known_users = [
    3938222837, # MigrantesChile
    225445834, # camila_vallejo
    4119914644, # mbachelet
    ...
]

[threat]
title = 'Threat'

[threat.tweet_tokens]
must_have = [
    '#inmigrantesilegales',
    '#nomasinmigrantes',
    ...
]

[threat.account_ids]
known_users = [
    123955962, # joseantoniokast
    59815228, # AXELKAISER
    39157375, # vanrysselberghe
    302219334, # tere_marinovic
    ...
]

```

A.5. Ejemplo de un tweet almacenado en notación de objeto .json

Código A.1: snippet del archivo auroracl_202201010400.data.json.gz


```
{
  "id":1477127427268788229,
  "text":"Uno de mis deseos para este 2022 es ver a @sebastianpinera preso...",
  "created_at":"Sat Jan 01 03:59:57 +0000 2022",
  "lang":"es",
  "entities.urls":"",
  "entities.user_mentions":"13623532",
  "entities.hashtags":"",
  "user.id":2688421183,
  "user.description":"",
  "user.location":"Santiago, Chile",
  "user.name":"Zulema Pereira",
  "user.screen_name":"zpereiral",
  "user.url":"",
  "user.protected":false,
  "user.verified":false,
  "user.followers_count":400,
  "user.friends_count":928,
  "user.listed_count":4,
  "user.favourites_count":27470,
  "user.statuses_count":33498,
  "user.created_at":"Mon Jul 28 21:29:41 +0000 2014",
  "user.profile_image_url_https":"https://pbs.twimg.com/profile_images
↪ /493875577418309632/t2Pm3bjM_normal.jpeg",
  "user.default_profile":false,
  "user.default_profile_image":false,
  "is_retweet":true,
  "is_quote":false,
  "is_reply":false,
  "in_reply_to_user_id":0,
  "in_reply_to_status_id":0,
  "quote.id":0,
  "quote.user.id":0,
  "rt.id":1477121447885639681,
  "rt.user.id":209047109
}

```

A.6. Lógica del filtrado de tweets

Código A.1: snippet del archivo data/importer.py

```
...
def filter_dataframe(self, df):
    flag = df["id"].notna()
    if not self.location["accept_unknown"]:
        if self.location["patterns"]:
            flag = flag & (
                df["user.location"].str.contains(self.location["patterns"]) == True
            )
    if self.location["blacklist"]:
        flag = flag & ~(
            df["user.location"].str.contains(self.location["blacklist"]) == True
        )
    candidates = df[flag]
    if len(self.automaton):
        result = []
        for tuple in candidates.itertuples():
            findings = set(pluck(1, self.automaton.iter(getattr(tuple, "text"))))
            if self.terms["patterns"] is not None:
                # we have keywords:
                result.append(
                    "search-term" in findings and not "rejected-term" in findings
                )
            else:
                result.append(not "rejected-term" in findings)
        candidates = candidates[result]
    self.logger.info(f"Keyword filtering: {len(candidates)} from {len(df)} tweets")
    return candidates

```

A.7. Función de parsing de archivos json a parquet

Código A.1: snippet del archivo data/importer.py

```
def __parse__files__to__parquet(self, i, filename, target__path):
    try:
        df = json.read__json(filename)
    except zlib.error:
        self.logger.error(f"ZLIB EXCEPTION - (#{i}) corrupted file: {filename}")
        return 0
    except pa.lib.ArrowInvalid:
        self.logger.error(f"PYARROW EXCEPTION - (#{i}) corrupted file: {filename}")
        return 0

    target__file = target__path / f"{Path(filename).stem}.parquet"

    pq.write__table(df, target__file, use__dictionary=False)
    return df.num__rows

```

Anexo B. Imágenes

B.1. Niveles de acceso a la API de *Twitter*

Twitter API access levels and versions

While the Twitter API v2 is the primary Twitter API, the platform currently supports previous versions (v1.1, Gnip 2.0) as well. We recommend that all users start with v2 as this is where all future innovation will happen.

The Twitter API v2 includes a few access levels to help you scale your usage on the platform. In general, new accounts can quickly sign up for Basic access. Should you want additional access, you may choose to apply for Enterprise access.

	Free	Basic	Pro	Enterprise
Getting access	Get Started	Get Started	Get Started	Get Started
Price	Free	$100/month	$5000/month	
Access to Twitter API v2	✔ (Only Tweet creation)	✔	✔	
Access to standard v1.1	✔ (Only Media Upload and Login With Twitter)	✔ (Only Media Upload and Login With Twitter)	✔ (Only Media, Help, Rate Limit, and Login with Twitter)	
Project limits	1 Project	1 Project	1 Project	
App limits	1 App per Project	2 Apps per Project	3 Apps per Project	
Tweet caps - Post	1,500	3,000	300,000	
Tweet caps - Pull	✘	10,000	1,000,000	
Filteres stream API	✘	✘	✔	
Access to full-archive search	✘	✘	✔	
Access to Ads API	✔	✔	✔	

Figura B.1: Tabla de niveles de acceso a la API brindados por *Twitter* actualmente.