República Bolivariana de Venezuela
Universidad de Carabobo
Facultad Experimental de Ciencias y Tecnología
Departamento de Matemáticas

Modelo de aprendizaje automático basado en BERT en español para la clasificación multiclase de las comunicaciones de Sinco

Trabajo Especial de Grado presentado ante la ilustre Universidad de Carabobo por el Br. Margy Garzon para optar al título de Licenciado en Matemáticas.

Tutor: Prof. José Marcano

Tutor: Ing. Ricardo Mussett

Bárbula-Venezuela
6 de junio de 2024

Universidad de Carabobo

Departamento de Matemáticas
Bárbula-Venezuela

FACYT

ACTA

Reunidos en la sede del Decanato de la Facultad Experimental de Ciencias y Tecnología de la Universidad de Carabobo, los miembros del Jurado designados por el Equipo Directivo del Departamento de Matemáticas de la Facultad Experimental de Ciencias y Tecnologías (FACYT), para evaluar el Trabajo Especial de Grado titulado: **"Modelo de Aprendizaje Automático basado en BERT en español para la clasificación multiclase de las comunicaciones de Sinco"**, el cual fue presentado por la bachiller **Margy Yuliana Garzón García., C.I. V-27550125,** a fines de cumplir con los requerimientos legales para optar al título de **Licenciada en Matemática,** dejamos constancia de lo siguiente: Una vez leído el trabajo por cada uno de los miembros del jurado, se fijó el 06 de junio de 2024, para su defensa pública. Después de concluida ésta, el bachiller **Margy Y. Garzón G.,** respondió satisfactoriamente a las preguntas que le fueron formuladas por el Jurado. Posteriormente el Jurado deliberó y consideró **Aprobarlo**.

En Naguanagua, a los seis días de mes de junio del año dos mil veinticuatro.

Luis A. Rodríguez

Prof. Luis Rodríguez
C.I. 11678882
Jurado

Prof. César Seijas
C.I. 4567093
Jurado

Prof. José G. Marcano
C.I. 8254206
Presidente

Ing. Ricardo Mussett
C.I. 18104957
Tutor Externo

Dedicatoria

El presente trabajo se lo dedico a mis padres, Pedro y María, gracias a ustedes soy lo que soy y se merecen todos mis logros.

También se la dedico a mi hermana menor, Kristy, como recordatorio de que puedes hacer todo lo que te propongas. Vale la pena esforzarse por las cosas que nos apasionan en la vida.

Finalmente, me dedico este trabajo a mí misma por todo el esfuerzo realizado, todas las noches de desvelo, todos los momentos Eureka y por cada vez que creía que no podría. Querida Margy del primer semestre, hoy somos matemáticos.

Agradecimientos

Agradezco primeramente a mi familia, mis padres Pedro y María, por su apoyo incondicional en toda mi carrera, desde los desayunos hechos por mi mamá hasta el recorrido realizado por mi papá para llevarme a la universidad todos esos días.

Agradezco a cada uno de los profesores que me formaron en mi camino, a todos los recuerdo con especial cariño. Agradezco de forma especial al profesor Cesar Panza por sus enseñanzas no solo en mi formación profesional sino también en mi formación como persona.

Agradezco al departamento del que me sentí parte desde el primer instante: al departamento de matemáticas o, lo que es igual, a todas las personas que lo conforman. Gracias por ser mi familia por estos años.

Y por último, agradezco a mis compañeros, a mis amigos: Eric Sosa, Giovanni Peréz, Otto Ludewig, Abigail Medina y Jorge Perera por cada momento compartido, por cada ayuda brindada, por cada risa contagiada; hicieron de mis días más felices. Agradezco profundamente a mi mejor amiga Edurbelys Chávez. Te agradezco tus palabras, tu paciencia, tu comprensión, tu apoyo y tu cariño; la universidad no hubiese sido igual sin ti.

Agradezco enormemente a cada una de las personas mencionadas (y las no mencionadas también) por haberme brindado tanto, soy la persona más afortunada del planeta por tenerlos en mi vida, representan una gran fuente de inspiración para seguir adelante, los amo a todos.

Resumen

En este trabajo, se presenta el ajuste y adaptación de BERT, un modelo preentrenado basado en Transformers, con el objetivo de mejorar el rendimiento de clasificación en el procesamiento de lenguaje natural. El estudio se enfoca en un conjunto de datos obtenidos de Sinco, una plataforma de gestión de comunicaciones donde los usuarios plantean diversas problemáticas y situaciones. Un desafío importante identificado en el conjunto de datos fue el desbalance en la distribución de clases, debido a las diferentes situaciones presentadas. Para abordar este problema, se aplicaron técnicas de procesamiento de datos, como oversampling y undersampling, junto con la asignación de pesos, con el fin de equilibrar la distribución de ejemplos entre las clases. Además, se llevó a cabo la adaptación de BERT mediante la inclusión de capas de redes neuronales y se realizaron ajustes de hiperparámetros, como la tasa de aprendizaje y las funciones de activación. Se aplicó esta adaptación de BERT tanto en las clases balanceadas utilizando las técnicas previas mencionadas, como en los datos en su forma original. Tras evaluar el rendimiento de clasificación con cada variante de las clases modificadas, se determinó que el modelo que obtuvo los mejores resultados en términos de las métricas de evaluación fue aquel en el que se balancearon las clases mediante la asignación de pesos. Se logró una precisión del 79 %, un recall del 80 % y un valor de F1-score del 79 %. Estos resultados son prometedores y demuestran el potencial de esta metodología, lo cual es relevante dada la complejidad de los datos y la diversidad de las clases involucradas.

Abstract

In this paper, we present the tuning and adaptation of BERT, a pre-trained model based on Transformers, with the objective of improving classification performance in natural language processing. The study focuses on a dataset from Sinco, a communication management platform where users raise various issues and situations. A major challenge identified in the dataset was the imbalance in class distribution, due to the different situations presented. To address this problem, data processing techniques, such as oversampling and undersampling, along with weight assignment, were applied to balance the distribution of examples across classes. In addition, adaptation of BERT was carried out by including neural network layers and adjustments of hyperparameters, such as learning rate and activation functions. This BERT adaptation was applied both on the balanced classes using the previous techniques mentioned above and on the data in its original form. After evaluating the classification performance with each variant of the modified classes, it was determined that the model that obtained the best results in terms of the evaluation metrics was the one in which the classes were balanced by assigning weights. An accuracy of 79 %, a recall of 80 % and an F1-score value of 79 % were achieved. These results are promising and demonstrate the potential of this methodology, which is relevant given the complexity of the data and the diversity of the classes involved.

Índice general

Índice de figuras

Índice de tablas

Introducción

El procesamiento del lenguaje natural (PLN) y el aprendizaje automático han revolucionado la forma en que las organizaciones analizan y comprenden grandes cantidades de datos textuales. En este contexto, los avances en técnicas de aprendizaje automático, como el transformer BERT, han demostrado resultados prometedores en diversas tareas de PLN, incluida la clasificación de texto.

En el ámbito específico de las comunicaciones de Sinco, donde la información textual es abundante y varían en su contenido y tono, la clasificación multiclase se convierte en un desafío fundamental.

La contribución principal de este trabajo es el desarrollo de un modelo de aprendizaje automático basado en el ajuste fino de BERT en español, con el objetivo de mejorar la precisión y eficiencia en la clasificación multiclase de las comunicaciones de Sinco. A través de la implementación de esta propuesta, se busca no solo optimizar el proceso de clasificación de las comunicaciones, sino también explorar las capacidades y potencialidades de BERT en un contexto específico del idioma español y con unos datos que presentan desequilibrio de clases.

El presente trabajo de investigación está comprendido por cinco capítulos, distribuidos de acuerdo al siguiente esquema:

Capítulo 1 - El problema de la investigación: en este capítulo se profundiza en el problema que motivó esta investigación, así como la justificación que sostiene la misma. Además, se especifican cada uno de los objetivos a alcanzar.

Capítulo 2 - Marco Teórico: Este capítulo contiene los preliminares básicos relacionados al Procesamiento de Lenguaje Natural, el aprendizaje de máquina y la arquitectura Transformer en la cuál está basada el modelo BERT. Por otro lado, se explicará en qué consiste el ajuste fino de modelos preentrenados y se definirán algunas métricas usadas para evaluar problemas de clasificación multiclase.

Capítulo 3 - Preparación de los datos: En este capítulo se detalla el procedimiento de preparación de los datos con el objetivo de adaptarlos para un análisis posterior y asegurar su formato idóneo para su paso por el modelo. Además, se expone el análisis llevado a cabo sobre los datos, su contenido y las categorías a las que pertenecen, destacando el sesgo observado en la distribución de las clases. Para abordar esta desigualdad en la distribución de clases, se proponen en este capítulo técnicas de remuestreo y refinamiento en los algoritmos utilizados.

Capítulo 4 - Experimentación: En este capítulo exponemos la metodología a usar para lograr nuestros objetivos. También se explicará la arquitectura y el funcionamiento del modelo producto de realizar un ajuste fino a BETO. Por último, se presentan los resultados obtenidos luego de entrenar el modelo con las diferentes técnicas para datos desbalanceados

Capítulo 5 - Conclusiones y trabajo futuro: este capítulo presenta el cierre y la proyección de este estudio en el campo del procesamiento del lenguaje natural y el aprendizaje automático aplicado a la clasificación de textos en el contexto específico de las comunicaciones de Sinco en español. Se resumirán las principales contribuciones del trabajo y se discutirán posibles direcciones para investigaciones futuras.

Capítulo 1

El problema de la investigación

En este apartado se explicará brevemente el problema que motivó la presente investigación, los objetivos que se desean perseguir y la justificación de la misma.

1.1. Planteamiento del Problema

El Consejo Federal de Gobierno (CFG) es un órgano encargado de la planificación y coordinación de políticas y acciones para el desarrollo del proceso de descentralización y transferencias de competencia del nacional, el cual se efectúa por medio del financiamiento de proyectos a través de planes de inversión que son destinados por ley a las Gobernaciones, Alcaldías y Organizaciones de base del poder popular (Consejos Comunales, Comunas, Asociaciones, entre otros).

De esta forma, Sinco nace como una herramienta para el registro de proyectos de los consejos comunales de Venezuela para ser analizados, aprobados y gestionados por el Consejo Federal de Gobierno; así también, incorpora un módulo que permite la interacción entre las organizaciones y las instituciones, facilitando las comunicaciones de forma directa a través de la plataforma [10].

La problemática que enfrenta la plataforma Sinco se centra en la gran cantidad de comunicaciones internas que reciben de los usuarios, en las cuales predominan inconvenientes y problemas de diversa índole. El principal reto radica en la dificultad de gestionar y responder de manera oportuna a esta alta carga de comunicaciones de forma manual, y como consecuencia, esta plataforma se encuentra ante la imposibilidad de atender y solventar cada problemática de manera individual y eficiente sin el apoyo de herramientas tecnológicas adecuadas.

En la actualidad, un equipo compuesto por dos individuos es responsable de llevar a cabo la clasificación manual de las comunicaciones que se envían al CFG a través de Sinco, los cuales realizan lecturas y análisis de la información contenida en las comunicaciones, asignándoles etiquetas apropiadas. Este proceso tiene como objetivo remitir posteriormente las comunicaciones al departamento encargado de abordar las inquietudes específicas planteadas.

Por otro lado, la naturaleza de la plataforma da lugar a comunicaciones cuyo contenido presenta una notable complejidad y diversidad. En ella, los usuarios expresan sus dificultades, dudas y sugerencias relacionados no solo con el funcionamiento de la plataforma, sino también con la gestión de los proyectos comunitarios que se cargan en ella y con los recursos relacionados a dichos proyectos. Dado que el proceso de etiquetado es llevado a cabo de manera manual, comienza a surgir una dependencia del criterio de los analistas, lo que genera un sesgo entre las categorías, por lo que la falta de modelos de clasificación adaptados a este contexto y vocabulario específico dificulta aún más el

manejo de tales comunicaciones.

El desafío radica en encontrar una forma eficiente y precisa de clasificar estas comunicaciones en categorías o etiquetas relevantes. Dado que cada mensaje puede contener información única y diversa, se necesita un enfoque que pueda comprender la complejidad del lenguaje natural y asignar cada comunicación a la categoría correspondiente.

1.2. Justificación de la investigación

Sinco, como plataforma de participación digital, es un instrumento muy importante para conocer las opiniones y necesidades de los ciudadanos que integran las comunidades y consejos comunales a lo largo del país [11], por lo que utilizar un modelo de Procesamiento de Lenguaje Natural para clasificar las comunicaciones enviadas por sus usuarios representa un paso para mejorar la capacidad de respuesta del CFG y, a su vez, conduce a decisiones mejor informadas en pro de las políticas públicas.

También se debe tener en cuenta que un trabajador promedio dedica hasta el 30 % de su tiempo a documentar la información y a otras tareas administrativas básicas [12]. Al automatizar o evitar incluso una fracción de dichas tareas, la institución encargada de esta plataforma podría aumentar la eficiencia en el proceso, así como reorientar el enfoque hacia actividades más valiosas.

Para superar esta problemática, se propone utilizar un modelo de Procesamiento de Lenguaje Natural basado en la arquitectura Bidirectional Encoder Representations from Transformers (BERT) que permita agilizar el proceso de respuesta de manera automática a la hora de trabajar con estas grandes cantidades de textos que pueden provocar una sobrecarga de información, tarea que a un humano le llevaría mucho tiempo y costo.

La aplicación de un modelo basado en BERT para la clasificación de las comunicaciones de Sinco permitiría priorizar y atender de manera más eficiente aquellas comunicaciones que representen problemas más críticos o recurrentes para la institución, así como atender la expectativa de una respuesta rápida de parte de los usuarios, lo que resultaría en una mejora significativa en la calidad del servicio brindado por Sinco.

Además, esta solución tecnológica permitiría aprovechar al máximo los recursos y tiempos de respuesta, al centrar los esfuerzos humanos en las situaciones que requieran atención personalizada. La implementación de un modelo de inteligencia artificial para la clasificación de comunicaciones en Sinco, ayudaría a resolver los problemas de los usuarios de manera más eficiente, mejorando así la satisfacción de los usuarios y fortaleciendo la relación entre la plataforma y las comunidades para el desarrollo endógeno de las mismas.

1.3. Objetivos de la Investigación

En esta sección se definirán los objetivos planteados para esta investigación, resaltando los objetivos específicos que permitirán alcanzar el objetivo principal de este proyecto.

1.3.1. Objetivo General

"Crear un modelo de Aprendizaje Automático basado en BERT en español para la clasificación multiclase de las comunicaciones de Sinco".

Para lograr este objetivo general, es necesario considerar también los siguientes objetivos específicos que permitan preparar los datos, entrenar y evaluar el modelo para tener mejores resultados.

1.3.2. Objetivos Específicos de la Investigación

1. Investigar el funcionamiento e implementación de un modelo basado en BERT, con el objetivo de comprender su aplicación en la clasificación multiclase.
2. Investigar sobre técnicas para enfrentar problemas de clasificación multiclase con clases desbalanceadas.
3. Preprocesar los datos para poder utilizarlos en el análisis y modelado, aplicando técnicas de limpieza, tokenización y codificación a cada comunicación.
4. Realizar un análisis exploratorio detallado de los datos, con el objetivo de identificar patrones y tendencias relevantes para la clasificación de las comunicaciones.
5. Etiquetar una muestra de los datos para entrenar el modelo.
6. Aplicar ajuste fino para adaptar el modelo BERT en español a nuestra tarea específica.
7. Establecer métricas de comparación y evaluar los resultados de la clasificación de BERT.

Capítulo 2

Marco Teórico

El presente capítulo proporcionará las bases teóricas para fundamentar conceptualmente el estudio, vinculando la investigación con teorías, modelos y conceptos establecidos en la literatura académica. Primeramente, se darán detalles del campo del procesamiento de lenguaje natural (PLN), destacando su definición, aplicaciones y evolución a lo largo de los años y se explorará en detalle el funcionamiento de modelos de aprendizaje de máquina, centrándose en los hiperparámetros fundamentales para el rendimiento eficaz de dichos modelos. Luego de esto, se presentará la arquitectura transformer en la cuál se basa el modelo BERT, definido también en este capítulo. Por otro lado, se define el concepto de modelos preentrenados y su importancia en el campo de PLN y se explica en profundidad el proceso de ajuste fino de estos modelos. Por último, se aborda el crucial tema de las métricas de evaluación utilizadas para el entrenamiento de modelos.

2.1. Procesamiento del Lenguaje Natural

Este proyecto presenta un problema de clasificación multiclase de texto: se parte de un conjunto de mensajes clasificados o etiquetados con una de las clases posibles y se necesita crear un modelo capaz de asignar a un mensaje nuevo la clase a la cuál pertenece. Para entender y resolver este tipo de problema es necesario analizar la definición y aplicaciones del procesamiento de lenguaje natural, así como las técnicas usadas para pre-procesar el texto.

2.1.1. Definición

El procesamiento del lenguaje natural (PLN por sus siglas) es un campo de conocimiento de la inteligencia artificial y la lingüística aplicada que se ocupa del estudio de las interacciones mediante uso del **lenguaje natural** entre los seres humanos y las máquinas [13], además de desarrollar tecnologías y algoritmos que permiten a las máquinas comprender, interpretar y generar lenguaje natural de manera similar a como lo hacen los seres humanos. Entiéndase como lenguaje natural al medio que utilizamos de manera cotidiana para establecer comunicación con las demás personas [14].

El procesamiento del lenguaje natural combina diferentes disciplinas como la lingüística y la inteligencia artificial, utilizando modelos estadísticos y de aprendizaje automático para analizar y comprender el lenguaje humano, identificar patrones y extraer información relevante. Su objetivo es enseñar a las computadoras a comprender el significado y la intención detrás de las palabras y generar respuestas apropiadas, facilitando la comunicación entre máquinas y seres humanos a través del lenguaje natural.

Sin embargo, el procesamiento del lenguaje natural presenta desafíos, ya que el lenguaje humano es complejo y ambiguo: las palabras pueden tener múltiples significados y las oraciones pueden tener estructuras gramaticales diversas. Además, hay variaciones en la forma en que las personas hablan

y escriben, como el uso de jergas, sarcasmo o expresiones idiomáticas. Esto requiere el desarrollo de algoritmos y modelos sofisticados para interpretar y comprender correctamente el lenguaje humano.

2.1.2. Aplicaciones del PLN

- **Asistentes virtuales y chatbots:** desarrollar asistentes virtuales y chatbots que pueden interactuar con los usuarios de manera natural y comprender sus consultas y comandos en lenguaje humano. Estos asistentes pueden realizar tareas como responder preguntas, brindar información, realizar reservas y más.
- **Análisis de sentimiento:** El PLN permite analizar y comprender el sentimiento y la actitud expresados en textos, como reseñas de productos, comentarios en redes sociales o encuestas. Esto ayuda a las empresas a evaluar la satisfacción del cliente, identificar tendencias y tomar medidas para mejorar sus productos o servicios.
- **Traducción automática:** Mediante el uso de técnicas de PLN, se puede desarrollar software de traducción automática que permite convertir texto de un idioma a otro de manera rápida y eficiente. Esto facilita la comunicación entre diferentes idiomas y culturas [15].
- **Búsqueda semántica:** El PLN ayuda a mejorar los motores de búsqueda al comprender el contexto y la intención detrás de las consultas de los usuarios. Esto permite obtener resultados más relevantes y precisos al realizar búsquedas en internet. Para que el sistema funcione necesita datasets en el idioma específico, reglas de gramática, teoría semántica y pragmática (para entender el contexto e intencionalidad), etc.
- **Generación de contenido:** El PLN se utiliza para generar automáticamente contenido como noticias, informes o descripciones de productos e incluso imágenes. Esto puede ayudar a ahorrar tiempo y recursos en la creación de contenido.
- **Reconocimiento y síntesis del habla:** Los sistemas de reconocimiento de voz toman la voz humana como entrada, la convierten en texto y luego la interpretan para comprender su intención. Después de generar una respuesta en forma de texto, se vuelve a convertir en voz humana utilizando la síntesis de voz. Es a través de este proceso de síntesis de habla que las máquinas pueden generar y reproducir voz de manera natural.
- **Resumen y clasificación de textos:** El uso del procesamiento del lenguaje natural se aplica cada vez más para resumir automáticamente textos extensos o extraer palabras clave para clasificarlos. En muchos casos, estos sistemas resultan valiosos, especialmente en sectores como el jurídico, donde puede resultar difícil navegar por grandes volúmenes de documentación o textos extensos. Los sistemas de PLN ayudan a identificar secciones específicas dentro de textos legales o a resumir grandes cantidades de información. Otro de los usos que se le da a esta función de clasificación, es la de detección de spam.

2.1.3. Breve historia del PLN

La historia del procesamiento de lenguaje natural se remonta desde los primeros intentos de traducción automática en la década de 1950 hasta la era actual de modelos de lenguaje basados en inteligencia artificial.

La traducción automática se convierte en una de las primeras aplicaciones que requieren de investigaciones en este nuevo campo que se formaba. En 1949, el matemático Warren Weaver propone la aplicación de técnicas del **desciframiento criptográfico**, métodos estadísticos y teoría de la información para la traducción automática de textos [16]; por otro lado, Georgetown-IBM en 1954 experimenta con una máquina de traducción automática, uno de los primeros intentos en esta tarea.

En 1950, Alan Turing publicó un artículo titulado «Computing Machinery and Intelligence» que proponía lo que ahora se llama la **prueba de Turing** como criterio para evaluar la inteligencia de una máquina, siendo esto una de las primeras atribuciones al campo de la Inteligencia Artificial. El término ***inteligencia artificial*** fue usado por primera vez en 1956 en la conferencia «Dartmouth Summer Research Project on Artificial Intelligence» por John McCarthy.

En 1957, Noam Chomsky publicó «Computing Machinery and Intelligence», introduciendo a la comunidad lingüística un modelo generativo del lenguaje, dando origen a la conocida teoría estándar de la Gramática Generativo-Transformacional. Este enfoque gramatical prioriza la sintaxis como elemento central, relegando la semántica y la fonología a meros componentes interpretativos, lo que permite describir los aspectos regulares del lenguaje [17].

A partir de la década de los 60 y con el nacimiento de la Inteligencia Artificial, surge una nueva pregunta: ¿Existe la posibilidad de que un ordenador pudiera ser programado para utilizar una lengua, en concreto, el inglés? Así, surgen los primeros sistemas de comprensión del lenguaje natural como lo fue el programa **ELIZA**, desarrollado por Joseph Weizenbaum en 1966 cuyo algoritmo consistía en utilizar un conjunto de reglas de transformación, describiendo una serie de posibles patrones y un conjunto de posibles respuestas asociadas, sin fundamentarse en ningún modelo semántico. SHRDLU, desarrollado por Terry Winograd en el Instituto de Tecnología de Massachusetts (MIT) a finales de la década de 1960, fue otro sistema de comprensión del lenguaje natural destacado de la época.

En la década de 1980, el algoritmo de aprendizaje del **perceptrón multicapa** (MLP por sus siglas en inglés) se convirtió en un avance significativo para el procesamiento del lenguaje natural permitiendo el desarrollo de modelos de procesamiento de lenguaje basados en redes neuronales, lo que impulsó enormemente la capacidad de las computadoras para comprender y generar lenguaje humano de manera más precisa y gracias a esto, este campo toma un nuevo impulso.

Se lanza **WordNet** en 1990, una base de datos léxica que relaciona palabras en inglés [18] la cuál se originó en 1986 en la Universidad de Princeton, donde continuó siendo desarrollada y extendida. George A. Miller, un psicolingüista, se inspiró en experimentos de inteligencia artificial que intentaban comprender la memoria semántica humana y propuso WordNet como un mecanismo eficiente y económico de almacenamiento y acceso de palabras y conceptos.

A partir del año 2000 surgieron avances significativos en el procesamiento del lenguaje natural (PLN). Comenzaron a publicarse trabajos relevantes sobre «Word Embedding» y modelos de lenguaje basados en estadísticas. El «Word Embedding» es una técnica que representa palabras como vectores densos en un espacio matemático [19], lo que permite capturar relaciones semánticas y sintácticas entre palabras. Por otro lado, en 2009, Google presentó el modelo de lenguaje «Google n-gram», el cual utilizaba grandes volúmenes de datos para mejorar la calidad del procesamiento del lenguaje. Este modelo se basa en el análisis estadístico de secuencias de palabras, aprovechando la enorme cantidad de texto disponible en la web para comprender patrones lingüísticos y mejorar la precisión en diversas tareas de PLN, como la corrección gramatical, el análisis de contexto y la generación de texto.

En 2013, se publicó el modelo de lenguaje **Word2Vec** el cual se basaba en «Word Embedding» y permitió representar palabras como vectores matemáticos en un espacio multidimensional, donde la proximidad en el espacio representaba similitud semántica entre las palabras.

En 2018, se introdujo el **Transformer**, un modelo de aprendizaje profundo que revolucionó el procesamiento del lenguaje natural, marcando el comienzo de la era de los modelos de lenguaje basados en atención. Este nuevo modelo desarrollado por Google se destaca por su capacidad para proce-

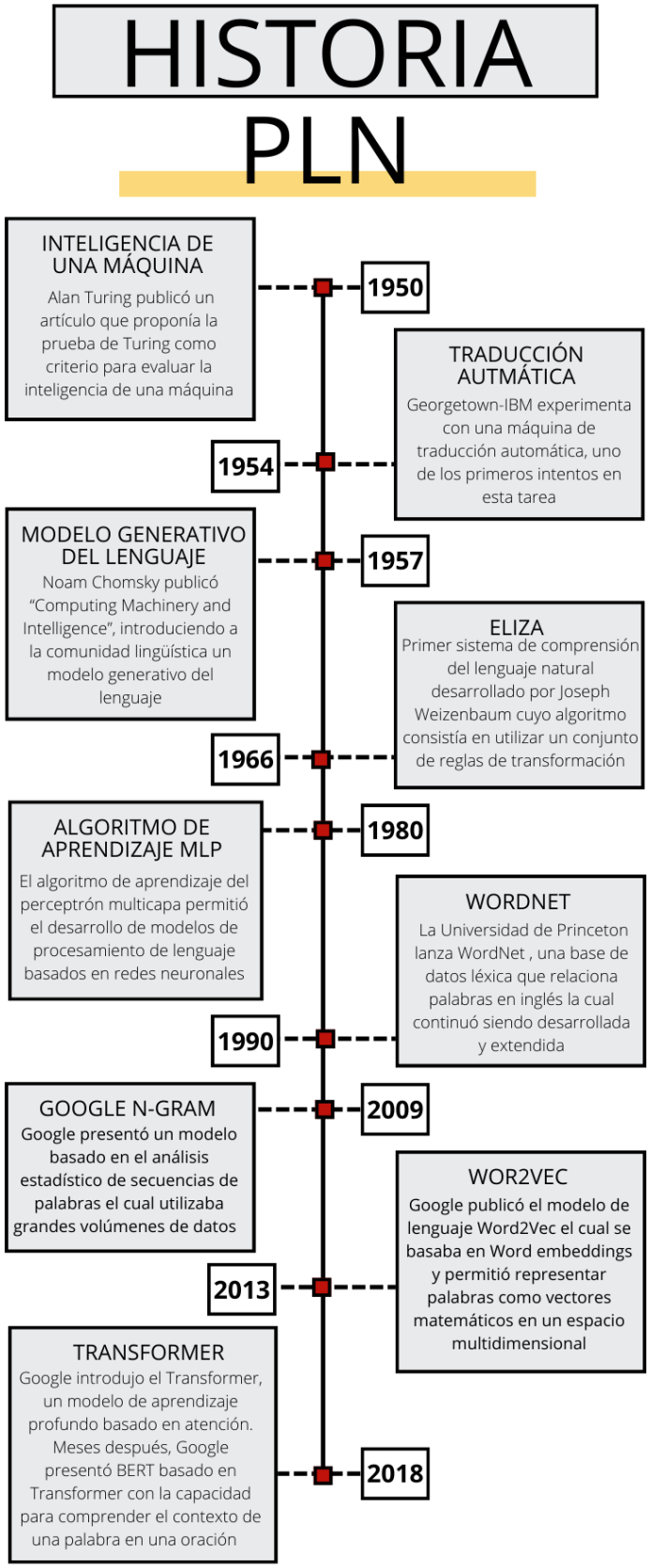

Figura 2.1: Línea del Tiempo de la historia del Procesamiento del Lenguaje Natural

sar secuencias completas de información a la vez, en lugar de depender de modelos recurrentes o convolucionales y surge debido que las redes neuronales recurrentes (RNN) y las redes neuronales convolucionales (CNN) utilizadas previamente en el PLN, tenían dificultades para capturar las relaciones entre palabras que estaban distantes entre sí en la secuencia debido a su dependencia de la posición y a la pérdida de memoria. En este mismo año, Google presentó un modelo de lenguaje basado en la arquitectura Transformer llamado **BERT** que difiere de los enfoques anteriores que utilizaban modelos de lenguaje unidireccionales o dependientes; BERT tiene la capacidad para comprender el contexto de una palabra en una oración a través del entrenamiento bidireccional, es decir, considerando tanto las palabras anteriores como las posteriores en una secuencia. En la figura 2.1 se puede observar un breve resumen de los hechos más importantes en la evolución del PLN a tráves de los años.

2.2. Aprendizaje de máquina

Según Goodfellow, Bengio y Courville [20], los algoritmos de aprendizaje de máquina son aquellos que tienen la capacidad de aprender de los datos para resolver una tarea específica y permiten abordar problemas que son altamente complejos de resolver con programas fijos, diseñados y escritos por seres humanos. Así, el **aprendizaje profundo** es un tipo de aprendizaje de máquina y de forma general, sus algoritmos están formados por una serie de capas compuestas de neuronas con parámetros conectados por funciones no lineales llamadas funciones de activación.

Dado un número de observaciones $X = x_1, x_2, ..., x_n$ con sus respectivas etiquetas $Y = y_1, ..., y_n$ se desea hallar la función $f(x) = y$ que sea capaz de predecir la etiqueta de una nueva observación, por lo tanto, estos algoritmos tienen un proceso de entrenamiento en el que se busca obtener la función f^* de tal forma que $f^* \approx f$ y $f^*(x;\theta) = \hat{y}$, donde θ son los parámetros que definen a f^* de tal forma que cumplan $\hat{y} = y$, los cuales serán aprendidos durante el entrenamiento del modelo o algoritmo [21]. Para encontrar estos parámetros, se emplea una función de pérdida o función objetivo $L(\hat{y}, y)$ que determina la diferencia entre la estimación f^* y el resultado esperado. A través de este proceso, el algoritmo calcula los gradientes de los parámetros y los ajusta utilizando el descenso estocástico de gradiente para minimizar L, que a su vez evalúa el error estimado de f^* en relación a un grupo de ejemplos aleatorios del conjunto de entrenamiento, denominado *batch*, siendo este error el promedio de la función de pérdida sobre estos ejemplos. Posteriormente, se determina el gradiente de los parámetros para llevar a cabo un paso de descenso de gradiente y actualizar θ con el propósito de minimizar el error. Estos pasos se repiten con nuevos *batch* hasta que se hayan empleado todos los datos de entrenamiento, generalmente en más de una ocasión.

2.2.1. Descenso del gradiente

El descenso del gradiente es un **algoritmo de optimización** para actualizar iterativamente los pesos o parámetros con el fin de minimizar la función de pérdida [22]. El proceso comienza con la inicialización de los parámetros del modelo con ciertos valores, luego, en cada iteración, se calcula el gradiente de la función objetivo con respecto a los parámetros actuales el cual indica la dirección en la que la función objetivo crece más rápidamente, por lo que el descenso del gradiente se mueve en dirección contraria al gradiente para reducir el valor de la función objetivo acercándose a su mínimo (véase la figura 2.2).

La ecuación para actualizar θ basada en el **gradiente** es la siguiente;

$$\theta_{n+1} = \theta_n - \eta \nabla L(\hat{y}, y) \tag{2.1}$$

donde η es la tasa de aprendizaje, la cuál se detallará más adelante, y $\nabla L(\hat{y}, y)$ se refiere al gradiente

de la función de pérdida con respecto a cada parámetro.

$$\nabla L(\hat{y}, y) = \begin{bmatrix} \frac{\partial}{\partial \theta_1} L(\hat{y}, y) \\ \frac{\partial}{\partial \theta_2} L(\hat{y}, y) \\ \vdots \\ \frac{\partial}{\partial \theta_n} L(\hat{y}, y) \end{bmatrix}$$

Esto se repite hasta que se alcanza un criterio de convergencia, como una cantidad máxima de iteraciones o una cierta tolerancia en el cambio de los parámetros.

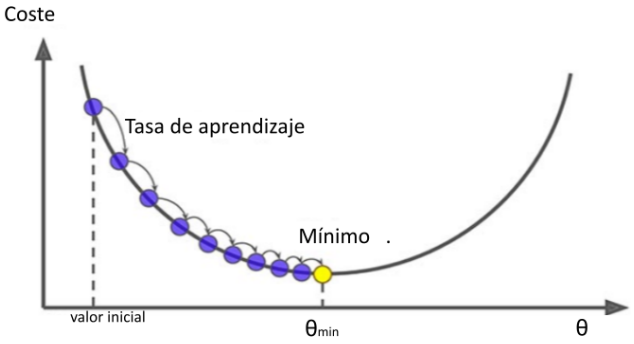

Figura 2.2: Actualización de parámetros mediante el descenso del gradiente para llegar al mínimo de la función de coste. Imagen presentada en [1]

2.2.2. Hiperparámetros

Mientras que los parámetros θ de un modelo de aprendizaje profundo son aquellos valores que el modelo debe modificar durante el entrenamiento, los hiperparámetros son configuraciones que se utilizan para controlar el proceso de entrenamiento de un modelo de aprendizaje automático y se establecen antes de iniciar el entrenamiento y no se optimizan directamente con los datos de entrenamiento. A continuación se establecerán los hiperparámetros más importantes para el entrenamiento del modelo.

Función de pérdida

La función de pérdida, también conocida como función de costo o función objetivo, es una medida que cuantifica qué tan bien el modelo de aprendizaje automático es capaz de predecir el resultado correcto. Esta función compara las predicciones del modelo con los valores reales de los datos y calcula una puntuación de error que penaliza a la red. De esta forma, la red es capaz de actualizar los parámetros para minimizar el error.

Existen diferentes tipos de funciones de pérdida [23]; cada una tiene sus propias características y se adapta mejor a diferentes tipos de problemas de aprendizaje automático. Las más comunes son:

- **Error cuadrático medio (MSE)**: Se utiliza para problemas de regresión donde se busca predecir un valor numérico.

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

- **Error absoluto:** Es una medida simple de la distancia entre las predicciones y los valores reales.

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

- **Pérdida de la regresión logística:** También conocida como entropía cruzada binaria, se utiliza para evaluar el rendimiento de un modelo de regresión logística.

$$L(y, \hat{y}_i) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

- **Pérdida de entropía cruzada:** Para problemas de clasificación multiclase con C clases, como es este caso, la función de pérdida usada es la función de *pérdida de entropía cruzada* o *cross entropy loss* [24] y se define cómo

$$L(x, y) = \frac{\sum_{n=1}^{N} l_n}{N}, \quad l_n = -\sum_{c=1}^{C} w_c \log\left(\frac{\exp(x_{n,c})}{\sum_{i=1}^{C}\exp(x_{n,i})}\right) y_{n,c} \tag{2.2}$$

  donde N es la cantidad de ejemplos, $x_{n,c}$ es la observación o ejemplo actual y $y_{n,c}$ es la etiqueta real correspondiente. El parámetro w_n es un peso asignado a cada clase y es útil para casos de datos desbalanceados, donde un mayor valor le dice al modelo que debe dar más importancia a las clases minoritarias.

Tasa de aprendizaje

Es un valor escalar que se utiliza para multiplicar el gradiente calculado en cada paso de entrenamiento. Se encarga de determinar el tamaño de los pasos que el algoritmo de optimización toma durante la actualización de los parámetros del modelo.

Este es un hiperparámetro crítico en el descenso del gradiente, ya que puede afectar la convergencia y la estabilidad del algoritmo. Si es demasiado alto, el algoritmo dará pasos excesivamente grandes causando divergencia. Por otro lado, si es demasiado bajo, los pasos serán demasiado pequeños y el proceso tomará mucho tiempo para alcanzar el mínimo.

Escoger un valor de tasa de aprendizaje dependerá de los datos aportados y la tarea en particular dado que no hay una fórmula definitiva para elegir el valor correcto. La práctica más extendida implica probar distintos valores mediante un proceso de prueba y error, aunque los valores típicos para este hiperparámetro suelen oscilar entre 0 y 1. Es común iniciar con una tasa de aprendizaje pequeña y luego aumentarla gradualmente en función de la iteración k del entrenamiento.

Optimizador

Los optimizadores son algoritmos que implementan variantes del descenso del gradiente, como el descenso del gradiente estocástico, para ajustar automáticamente los parámetros del modelo minimizando la función de pérdida durante el entrenamiento y mejorando el rendimiento del modelo.

El algoritmo de optimización ADAM se fundamenta en gradientes de primer orden de funciones estocásticas y utiliza estimaciones adaptativas de movimientos de orden inferior. Es una técnica fácil de implementar, eficiente desde el punto de vista computacional, con bajos requisitos de memoria, invariante al cambio de escala de los gradientes, y apta para abordar problemas de gran escala en términos de datos y/o parámetros [25].

Tamaño de lote

Durante el entrenamiento de un modelo de aprendizaje automático, los datos suelen dividirse en lotes de igual tamaño para facilitar el proceso de optimización. El tamaño del lote o *batch size* se refiere al número de ejemplos de entrenamiento que se utilizan en una iteración del algoritmo de entrenamiento [21].

Un tamaño de lote más grande puede tener ventajas en términos de eficiencia computacional, ya que permite una mayor paralelización del procesamiento de datos en hardware especializado como las unidades de procesamiento gráfico (GPU). Además, al considerar más ejemplos de entrenamiento simultáneamente, el modelo puede realizar actualizaciones de peso con más información y, en ocasiones, alcanzar una convergencia más estable o encontrar un óptimo local más preciso.

Por otro lado, un tamaño de lote más pequeño puede ofrecer ciertas ventajas, como una convergencia más rápida debido a las actualizaciones de peso más frecuentes. Sin embargo, es posible que el modelo no aprenda las características y detalles que pueden ser significativos en la predicción.

Número de épocas

Una época o *epoch*, en inglés, se refiere a una iteración completa a través de todo el conjunto de datos de entrenamiento. El número de épocas en un modelo de redes neuronales es la cantidad de veces que todo el conjunto de datos de entrenamiento se ha presentado al modelo durante el proceso de entrenamiento [21].

Un número insuficiente de épocas puede resultar en un modelo subentrenado, es decir, un modelo que no ha aprendido lo suficiente de los datos. Por otro lado, un exceso de épocas puede conducir a sobreajuste, donde el modelo empieza a memorizar los datos de entrenamiento en lugar de generalizar patrones.

Número de capas

Las capas se refieren a las unidades fundamentales dentro de la arquitectura de un modelo de aprendizaje profundo. Cada capa está compuesta por un conjunto de neuronas que realizan operaciones de transformación en los datos de entrada, permitiendo al modelo aprender representaciones cada vez más complejas a medida que pasa por estas capas. Cada capa en una red neuronal procesa y transforma la información de forma secuencial, lo que permite que el modelo aprenda representaciones jerárquicas de los datos a medida que avanza a través de las capas.

Función de activación

Las funciones de activación son funciones no lineales que se aplican a la salida de cada neurona con el fin de que la red pueda aprender y modelar relaciones y patrones complejos en los datos. Existen diversas funciones de activación, las cuales tienen diferentes ventajas o desventajas de acuerdo a nuestros datos y a la tarea empleada [26].

- **Función sigmoide:** la función sigmoide produce salidas en el rango de 0 a 1 lo cual es útil para comprimir valores atípicos hacia 0 o 1. Además, es diferenciable, lo cuál es importante para el descenso del gradiente. Es útil en problemas de clasificación binaria, ya que puede interpretarse como la probabilidad de que una determinada entrada pertenezca a una clase en particular. Se representa como:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Función ReLu:** La función *Rectified Linear Unit* toma un valor de entrada y devuelve cero si es negativo y el mismo valor de entrada si es positivo. Se define como:

$$f(x) = \text{máx}(0, x)$$

Esta función ofrece una gran eficiencia computacional y evita el problema del desvanecimiento del gradiente. Se ha demostrado que su uso acelera el proceso de entrenamiento en comparación con funciones sigmoideas o hiperbólicas ya que su derivada es muy similar a una función lineal. Esta función se comporta como una función rampa cuya derivada no existe en cero.

- **Función Softmax:** sea $X = (x_1, x_2, ..., x_n)$, la función Softmax es definida como:

$$\sigma(X)_i = \frac{exp\,(x_i)}{\sum_{k=1}^{n} exp\,(x_k)} \quad \forall i \leq n$$

Ella produce una distribución de probabilidad sobre varias clases mutuamente excluyentes. La función Softmax toma un vector $X \in \mathbb{R}^n$ y lo convierte en un vector de la misma longitud, donde cada elemento está en el rango de 0 a 1 y la suma de todos los elementos es igual a 1, es decir, normaliza el vector original.

- **Función Tangente Hiperbólica:** similar a la función sigmoide, la función *tanh*, definida como

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

produce salidas en el rango de -1 a 1. Esto puede ser útil para normalizar los datos y mejorar la capacidad del modelo para aprender representaciones más complejas. Tiene la ventaja de ser suavemente diferenciable y de mapear todos los valores hacia la media, dado que está centrada alrededor de cero.

2.3. Transformer

El *Tranformer* [2] es una arquitectura de red neuronal basada en **mecanismos de atención**, prescindiendo del uso de redes recurrentes y convolucionales y permitiendo la paralelización, lo que quiere decir que puede procesar múltiples partes de una secuencia de entrada al mismo tiempo. Inicialmente este modelo fue entrenado para la tarea de traducción, sin embargo, ha demostrado ser útil para otros problemas del PLN como procesamiento secuencial, predicción secuencia a secuencia y clasificación [27] obteniendo resultados superiores a los modelos que se usaban en el momento con un menor tiempo de entrenamiento.

De forma superficial y resumida, el Transformer mapea secuencias de vectores de entrada $(x_1, ..., x_n)$ en secuencias de vectores de salida de la misma longitud $(y_1, ..., y_n)$. Esta arquitectura se compone de dos partes principales: un codificador y un decodificador compuestos por una pila de capas idénticas. En la figura 2.3 se ilustra esta arquitectura; los autores usaron 6 capas idénticas. El codificador toma como entrada un texto y lo transforma en una representación de valores continuos en un espacio vectorial que contenga toda la información aprendida sobre esa entrada. El decodificador luego toma esta representación vectorial junto con la salida anterior y la utiliza para generar un solo texto de salida.

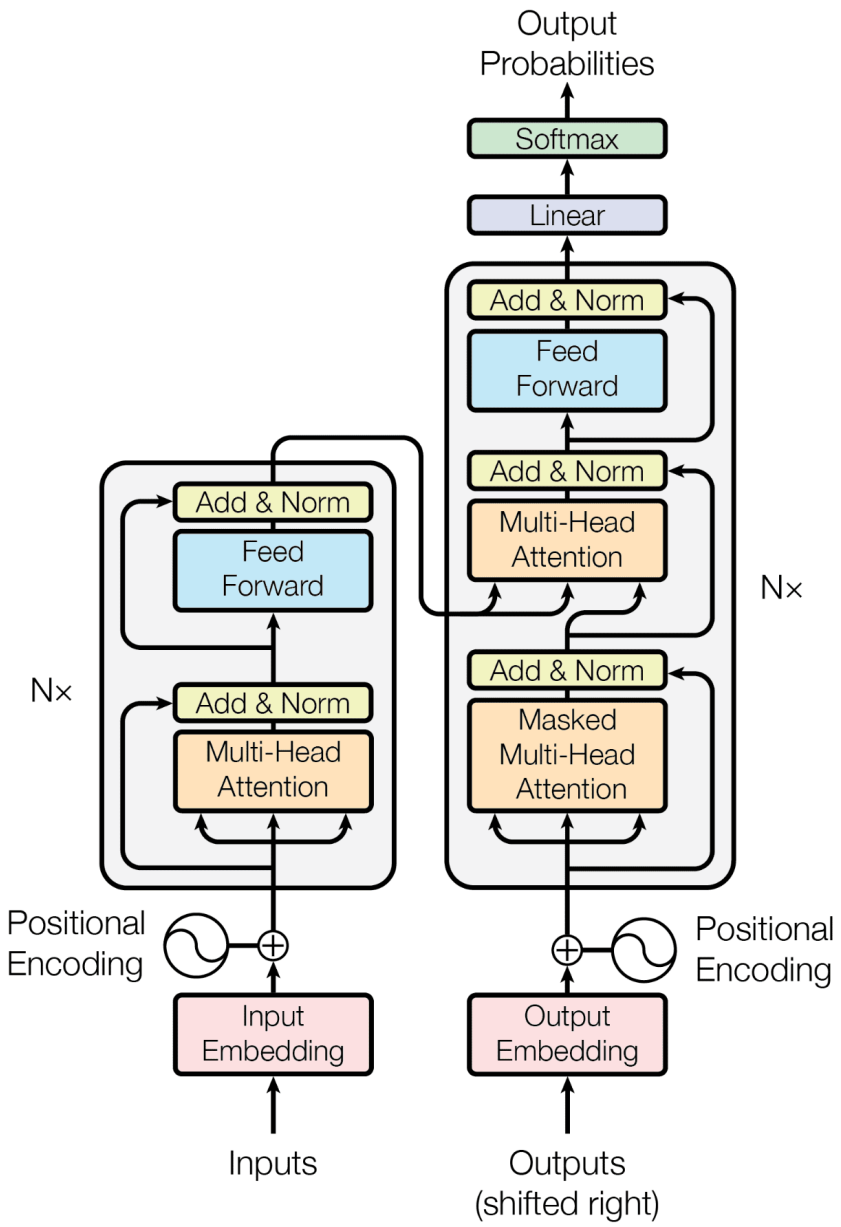

Figura 2.3: Arquitectura del transformer presentada en *attention is all you need* [2]

2.3.1. Embeddings

Inicialmente, el modelo recibe como entrada una frase. Esta es separada en unidades más pequeñas llamadas tokens que normalmente suelen ser palabras pero también pueden ser símbolos de puntuación, números o cualquier otra cosa que se desee considerar como una unidad significativa.

Este primer bloque es el encargado de aprender cómo representar cada token en el vocabulario como un vector v_e en R^n con $n = d_{model}$, donde d_{model} es la dimensión de la **representación vectorial** (el embedding) de un token. En [2] usan salidas de dimensión 512.

2.3.2. Codificación Posicional

Recordemos que la arquitectura transformer no usa recurrencias o convoluciones que determinen el orden de los elementos de las secuencias de entrada, por lo que de alguna manera se debe inyectar información relativa o absoluta del orden de los tokens o palabras dentro del texto.

Para codificar el orden en las secuencias, los autores del modelo [2] plantean sumar un vector a los embeddings para inyectar esta información. Dicho vector cuál se obtiene mediante la siguiente función:

Sea $f : \mathbb{N} \to \mathbb{R}^{d_{model}}$ la función que produce el vector v_{pe} de *Codificación Posicional*

$$v_{pe} = f(pos, i) = \begin{cases} sen\left(\frac{pos}{10000^{\frac{2k}{d_{model}}}}\right) & \text{si } i = 2k \\ cos\left(\frac{pos}{10000^{\frac{2k}{d_{model}}}}\right) & \text{si } i = 2k+1 \end{cases} \tag{2.3}$$

Donde *pos* es la posición del token en el texto e *i* es la dimensión. La intuición detrás de estas fórmulas es que, por cada posición par, se asigna un vector calculado con la función seno y, por cada posición impar uno con la función coseno. Se eligen estas funciones porque tienen propiedades lineales que facilitan al modelo el aprendizaje de atención.

Finalmente, se realiza la suma del vector embedding v_e y el vector v_{pe} que tiene la misma dimensión d_{model}, obteniendo así el vector de entrada x_i que contiene la información posicional de cada palabra, lo que permite paralelizar el entrenamiento.

2.3.3. Auto-atención

El bloque de auto-atención es el bloque principal del Transformer y es lo que lo hace diferente a otros modelos y está diseñado para capturar las relaciones de dependencia entre palabras en una oración o secuencia de texto. En lugar de utilizar capas recurrentes o convolucionales, el Transformer utiliza el mecanismo de atención para calcular las representaciones de palabras en un contexto dado.

Antes de explicar el módulo de auto-atención o *self-attention*, se debe entender el módulo de atención: en un enfoque basado en la atención, el núcleo radica en la capacidad de evaluar la importancia de un elemento en relación con otros elementos de un conjunto. En el caso de la auto-atención, estas evaluaciones se realizan con respecto a otros elementos dentro de una misma secuencia específica, considerando también el elemento de entrada (ver Fig 2.4).

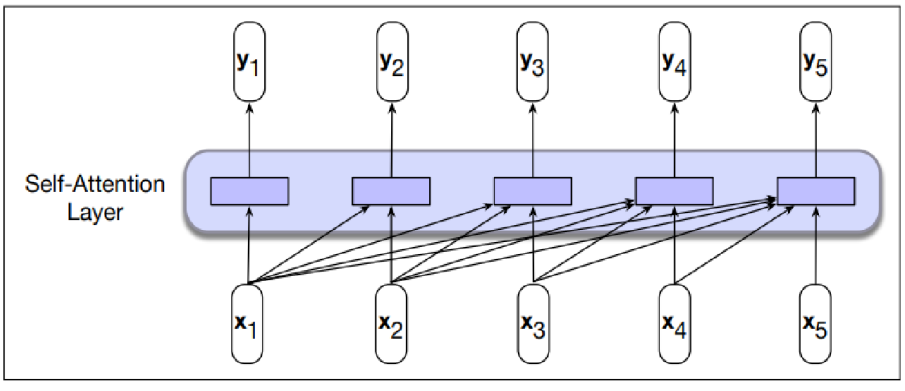

Figura 2.4: Flujo de información en un modelo de autoatención causal (o enmascarado). Al procesar cada elemento de la secuencia, el modelo atiende a todas las entradas hasta la actual, incluida. Imagen presentada en [3]

El resultado de estas comparaciones se usa para calcular una salida para la entrada actual. Por ejemplo, volviendo a la figura 2.4, al calcular y_3, se realizan comparaciones entre la entrada x_3 y sus

elementos precedentes x_1 y x_2, así como con x_3 mismo.

Así, para calcular la autoatención de cada elemento de la secuencia se sigue el siguiente proceso: definimos $y_i \in \mathbb{R}^d$ como el vector de salida, el cuál se busca actualizar en base a otros n vectores $X = \{x_1, ..., x_n\} \subset \mathbb{R}^d$. Considere además tres roles o funciones diferentes que se generan en cada incorporación de una entrada durante el curso del proceso de atención:

- **Consulta o *query*** q: representa la ubicación en la secuencia desde la cual se está tratando de aprender una relación.
- **Llave o *key*** k: se utiliza para calcular la relevancia de una consulta con respecto a las diferentes posiciones en la secuencia de entrada.
- **Valor o *value*** v**:** El valor representa la información específica asociada con cada posición en la secuencia de entrada.

Para calcular estas tres funciones, en [2] se introducen tres matrices de pesos $W^Q \in \mathbb{R}^{d\times d}$, $W^K \in \mathbb{R}^{d\times d}$ y $W^V \in \mathbb{R}^{d\times d}$ las cuales son aprendidas durante el proceso y usadas para proyectar cada vector de entrada x_i en una representación de su función:

$$\begin{aligned} q_i &= W^Q x_i \\ k_i &= W^K x_i \\ v_i &= W^V x_i \end{aligned} \tag{2.4}$$

Dadas estas proyecciones, se define la puntuación como la comparación entre un foco de atención actual x_i y un elemento en el contexto anterior x_j y consta de un producto escalar entre su vector de consulta q_i y los vectores clave del elemento anterior k_j

$$puntuacion(x_i, x_j) = q_i \cdot k_j^T \tag{2.5}$$

el cuál es un valor escalar que indica la similitud entre los elementos comparados.

Estas puntuaciones son regularizadas y normalizadas con una función Softmax para crear un vector de pesos α_{ij} llamado coeficientes de atención. Este indica la relevancia proporcional de cada entrada al elemento de entrada i que es el foco de atención actual.

$$\alpha_{ij} = \text{softmax}\left(\frac{q_i \cdot k_j^T}{\sqrt{d}}\right) = \frac{exp\left(\frac{q_i \cdot k_j^T}{\sqrt{d}}\right)}{\sum_{k=1}^{i} exp\left(\frac{q_i \cdot k_j^T}{\sqrt{d}}\right)} \quad \forall j \leq i \tag{2.6}$$

Los coeficientes de atención α_{ij} de (2.6) son usados para destacar los valores v de los x_i importantes por medio de una combinación lineal y se genera el valor de salida y_i de la siguiente manera:

$$y_i = \sum_{j\leq i} \alpha_{ij} v_j \tag{2.7}$$

Por último, este valor pasa por una última transformación aprendida para actualizar el valor de salida. Este último paso se muestra en la ecuación (2.8).

$$y_i' = \left(\sum_{j\leq i} \alpha_{ij} v_j\right) W^O + b^O, \quad con \quad W^O \in \mathbb{R}^{d\times d}, b^O \in \mathbb{R}^d \tag{2.8}$$

La descripción de este proceso ha sido desde la perspectiva de una sola salida en un solo paso de tiempo i. Sin embargo, como cada salida y_i se calcula de forma independiente, es posible paralelizar

este proceso implementando rutinas de multiplicación de matrices donde se empaquetan las entradas de los N tokens de la secuencia de entrada en una única matriz $X \in \mathbb{R}^{N \times d}$. Así, las ecuaciones (2.4) quedan de la forma:

$$\begin{aligned} Q &= XW^Q \\ K &= XW^K \\ V &= XW^V \\ con \quad Q, K, V &\in \mathbb{R}^{N \times d} \end{aligned} \tag{2.9}$$

De igual forma, la ecuación (2.7) se escribe:

$$Atencion(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.10}$$

la cuál es una matriz con los vectores salidas de cada elemento de X y contiene toda la información de atención aprendida en el proceso.

El bloque de auto-atención es el núcleo de lo que se llama el bloque transformador que se compone de tres subcapas principales (ver figura 2.5): la capa de atención múltiple o *Multi-Head attention*, la capa de retroalimentación y la capa de normalización:

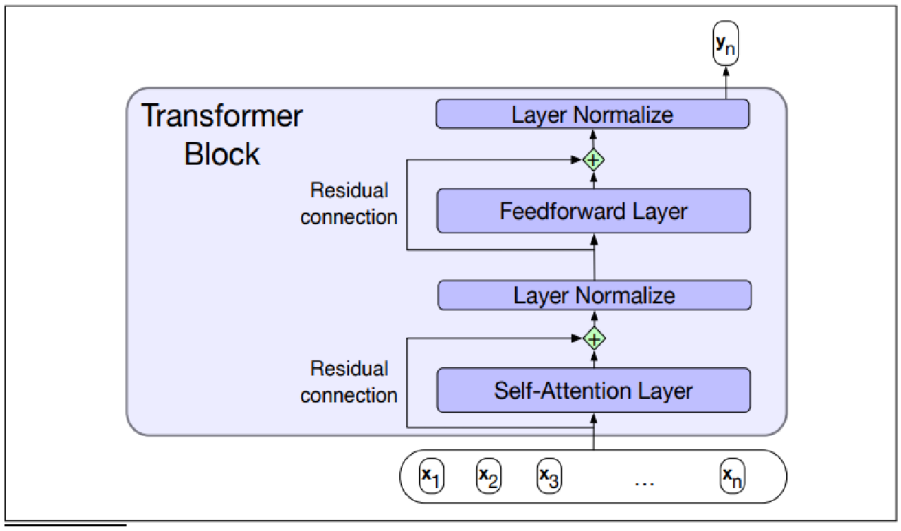

Figura 2.5: Bloque transformador. Imagen presentada en [3]

- **Multi-Head Attention:** Los módulos de auto-atención del Transformer también son Multi-Head, lo que significa que en lugar de realizar el proceso de auto-atención una vez para obtener una representación de dimensionalidad d, el proceso se repite h veces para vectores de dimensionalidad d/h, lo que permite al modelo atender a diferentes partes de la información en diferentes posiciones. Posteriormente, los h vectores resultantes se concatenan y se someten a la transformación descrita en la ecuación (2.11).

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, ..., head_h)W^O \\ \text{donde} \quad head_i &= Atencion(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.11}$$

- **Capa de retroalimentación o *feedfoward layer*:** después de obtener los pesos de atención, se realiza una operación de multiplicación entre los pesos y los valores de las palabras. Estos

resultados ponderados se concatenan y se pasan a través de una capa de red neuronal completamente conectada para obtener una representación combinada de las palabras en el contexto actual. Esto se logra con una función que consiste de dos transformaciones lineales y una ReLu de por medio.

- **Capa de normalización o *layer normalize*:** Por último, se aplica una técnica de normalización para estandarizar los valores de salida de la capa de red neuronal. Una opción común es la normalización por capa, que divide las salidas por su desviación estándar.

El bloque de atención se aplica de forma repetida en cada capa del Transformer, lo que permite que el modelo capture relaciones de dependencia de largo alcance en el texto.

2.4. Modelos preentrendos

Zhang (2021) en [4] afirma que «la brecha entre Transformers y las arquitecturas tradicionales se hizo especialmente amplia cuando se aplicó en este un paradigma de preentrenamiento y, por lo tanto, el ascenso de Transformers coincidió con el ascenso de modelos preentrenados a gran escala, ahora a veces llamados modelos base».

El preentrenamiento en el contexto del PLN se refiere al entrenamiento de un modelo en una gran cantidad de datos textuales sin ninguna tarea o cuestión específica en mente. Este preentrenamiento tiene el objetivo de permitir que el modelo «aprenda» acerca de las estructuras lingüísticas, las relaciones entre las palabras y la semántica del lenguaje de una manera general antes de ser adaptado a tareas específicas. Decimos que entrenamos previamente un modelo de lenguaje y luego llamamos a los modelos resultantes modelos de lenguaje previamente entrenados o preentrenados. Estos modelos preentrenados pueden luego ser afinados o adaptados para tareas de procesamiento de lenguaje natural específicas, como la traducción automática, el resumen de texto, el análisis de sentimientos, entre otros.

Algunos de estos grandes modelos son: Transformers, BERT (Bidirectional Encoder Representations from Transformers), GPT-3 (Generative Pre-trained Transformer 3), T5 (Text-to-Text Transfer Transformer) y XLNet. De esta forma, la utilización de modelos preentrenados o modelos base susceptibles de ser sometidos a ajustes finos permite abordar un desafío particular en el PLN con una cantidad limitada o moderada de datos, obteniendo resultados prometedores.

Algunos de estos grandes modelos preentrenados se pueden obtener a través de *Hugging Face* [28], una página web de inteligencia artificial centrada en el desarrollo de tecnologías de PLN y aprendizaje automático. A través de su plataforma y comunidad, *Hugging Face* ofrece una serie de recursos y herramientas que han revolucionado la forma en que se investiga, desarrolla y despliegan modelos de PLN.

2.5. BERT

La arquitectura del Transformer consta de un codificador para representar secuencias de entrada y un decodificador para generar secuencias de salida (ver Figura 2.3) lo que hace que este modelo pueda ser utilizado en tres modos diferentes: solo codificador, codificador-decodificador y solo decodificador. Un modelo generado a partir del codificador de Transformer es el conocido BERT, el cual es de interés en este trabajo de investigación.

«*Bidirectional Encoder Representations from Transformers*», o BERT por sus siglas, es un codificador transformador bidireccional multicapa basado en la implementación original descrita en [2], es decir, BERT está basado en la primera parte principal, el codificador, de la arquitectura del Transformer descrita

anteriormente. Fue desarrollada por investigadores de Google AI Language en 2018 [29] y ha tenido un gran impacto en el PLN.

Muchos modelos de lenguaje preentenados como ELMo (Peters et al., 2018a) y Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018) utilizan modelos de lenguaje unidireccionales para aprender representaciones generales del lenguaje procesando la información de manera secuencial, es decir, en una sola dirección, sin considerar información posterior. A diferencia de los modelos de lenguaje tradicionales que procesan el texto en un solo sentido (de izquierda a derecha o de derecha a izquierda), BERT utiliza un **enfoque bidireccional**, lo que significa que considera tanto el contexto anterior como el posterior de cada palabra ayudando a capturar la relación entre las palabras en función de su contexto, lo que ayuda a comprender el significado y la semántica de las oraciones de manera más precisa.

La arquitectura de BERT consiste en apilar L bloques de Codificadores de Transformer de A *heads* cada uno. Para el modelo base [29] se definió $L = 12$ bloques (o *layers*) de $A = 12$ *heads*. El estado oculto de cada *layer* es de tamaño $H = 768$. Así, después de cada *layer* se obtienen 12 representaciones de cada token de dimensión $768/12 = 64$, o equivalentemente, una con su concatenación.

Este modelo ha sido pre-entrenado sobre una cantidad inmensa de texto sin etiquetar. Este hecho es el que hace que BERT sea tan poderoso ya que con este entrenamiento es capaz de capturar gran cantidad de detalles y se logra un entendimiento del funcionamiento del lenguaje natural de manera general para luego agregar capas que permitan entrenarlo para tareas específicas [30], permitiéndole resolver una variedad de tareas de PLN como el análisis de sentimientos, el reconocimiento de entidades con nombre, el etiquetado de roles semánticos y la resolución de correferencias.

2.5.1. Preentrenamiento de BERT

Para el corpus del pre-entrenamiento de BERT se usó textos extraídos de Book Corpus (800M de palabras) y Wikipedia en inglés (2500M de palabras) en el cuál solo se extrajo texto, ignorando tablas, listas y encabezados. Entre ambos suponen un corpus total de entorno a tres mil millones de palabras.

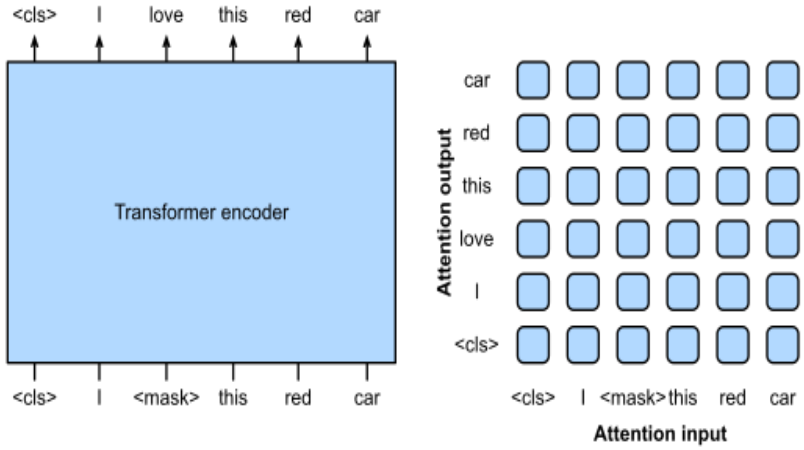

Figura 2.6: Izquierda: BERT previo al entrenamiento con modelado de lenguaje enmascarado . Derecha: patrón de atención en el codificador Transformer. Imagen presentada en [4]

Un primer procedimiento con el que se entrenó BERT es llamado «LM enmascarado» (MLM) el cual consiste en aplicar una máscara a un porcentaje de los tokens de entrada al azar (15 % en [29]) para luego predecir estos tokens enmascarados según el contexto dado por los términos no enmascarados. En la Figura 2.6 se observa un ejemplo donde el modelo debe predecir el token de *amor* enmascarado

analizando los tokens de entrada antes y después de este.

Otra estrategia de entrenamiento fue «Predicción de la Siguiente Oración» en el cual se eligieron oraciones A y B para cada ejemplo de preentrenamiento, donde el 50 % de las veces B es la oración real que le sigue a A y el resto de las veces es una oración aleatoria del corpus. Mediante este entrenamiento previo se buscó que el modelo comprenda las relaciones entre las oraciones, lo cual es beneficioso para tareas como la respuesta a preguntas (QA) o la inferencia del lenguaje natural (NLI).

2.5.2. BETO

La versión inicial de BERT introducida en 2018 fue pre-entrenada con textos en inglés, lo que limita su uso a tareas que requieren texto en dicho idioma como entrada.

BETO es un modelo BERT entrenado en un gran corpus de texto en español, en particular, se usó todo el texto en Wikipedia y todas las fuentes del proyecto OPUS que estuvieran en **español**. BETO tiene un tamaño similar a un BERT-Base y fue entrenado con la técnica de Máscara Completa de Palabras [31].

2.6. Ajuste Fino

En las secciones anteriores se ha mencionado una de las grandes ventajas de los grandes modelos pre-entrenados, la cuál es la capacidad de afinar o ajustar estos modelos para realizar tareas específicas.

El ajuste fino o «fine-tuning» en inglés, es una técnica del aprendizaje profundo que se fundamenta en el aprovechamiento de una red neuronal previamente entrenada. Este proceso implica la reconfiguración de algunos parámetros mediante un menor conjunto de datos o la adición de capas adicionales de redes neuronales que toma la capa superior de la red como entrada para realizar alguna tarea posterior, como el etiquetado de entidades nombradas, responder preguntas o correferencia [4].

El ajuste fino de un modelo pre-entrenado de Procesamiento de Lenguaje Natural (PLN) mediante la adición de capas adicionales es un enfoque común en el campo del aprendizaje automático. Consiste en tomar un modelo pre-entrenado, como BERT, GPT-3, o similares, y agregar capas adicionales de clasificación, regresión u otras tareas específicas para adaptar el modelo a una tarea o dominio de interés particular.

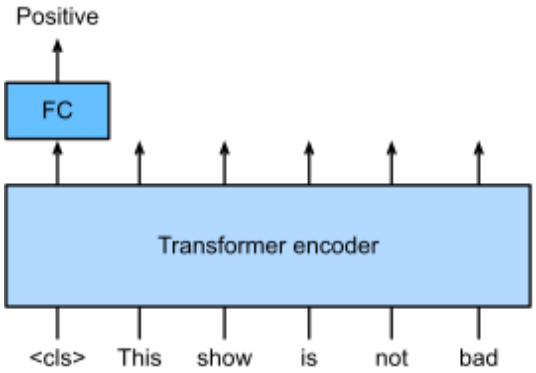

Figura 2.7: Ejemplo de ajuste fino. Imagen presentada en [4]

Cuando se entrena un modelo de PLN, se desarrolla un modelo de lenguaje que crea representaciones

profundas del significado de las palabras, por lo que el proceso de ajuste fino por lo general implica congelar los pesos del modelo pre-entrenado para evitar modificar sus representaciones aprendidas y agregar capas adicionales con datos específicos de la tarea. Esto permite al modelo pre-entrenado mantener el conocimiento general del lenguaje, mientras que las capas adicionales se ajustan a la tarea particular. En la imagen 2.7 se puede observar un ejemplo de ajuste fino, donde se toma un modelo pre-entrenado y se le agrega una capa adicional para realizar la tarea de análisis de sentimiento.

Citando a Zihao Fu et al en [32] «El ajuste fino de los parámetros del modelo para una tarea específica en un modelo pre-entrenado se ha convertido en una de las técnicas más prometedoras para el procesamiento del lenguaje natural (NLP) en los últimos años. Sin embargo, a medida que el número de parámetros crece de manera exponencial a miles de millones, se vuelve muy ineficiente guardar los parámetros completamente ajustados para cada tarea posterior. Muchos trabajos de investigación recientes proponen una forma de eficiencia de parámetros para resolver este problema ajustando solo una pequeña parte de los parámetros originales y almacenando los parámetros ajustados para cada tarea»

2.7. Métricas de evaluación

En el ámbito del aprendizaje automático y la inteligencia artificial, la evaluación precisa y objetiva del rendimiento de los modelos desempeña un papel crucial en la validación y comparación de enfoques propuestos. Las métricas de evaluación proporcionan una base cuantitativa para medir la eficacia de los algoritmos y modelos [33], permitiendo determinar su capacidad para resolver problemas específicos. Existe gran variedad de métricas importantes que se utilizan para medir el desempeño de un modelo; en este trabajo se usarán las más relevantes en problemas de clasificación multiclase.

2.7.1. Matriz de confusión

Sean N la cantidad de clases y sea $\mathcal{C} = \{C_1, C_2, ..., C_N\}$ las clases, la matriz de confusión es una matriz cuadrada $N \times N$ que muestra la cantidad de observaciones que fueron clasificadas correctamente e incorrectamente por el modelo en función de las clases reales y las predicciones realizadas por el modelo [5].

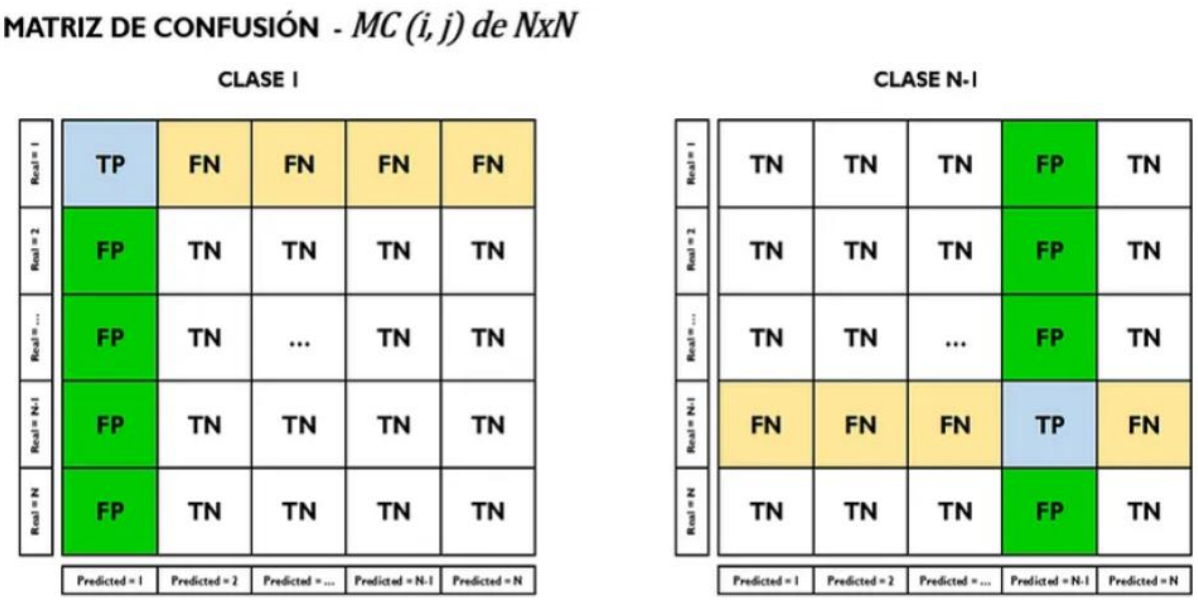

Figura 2.8: Matriz de confusión para un clasificador multiclase. Imagen presentada en [5]

Así, la matriz de confusión para la clase n (observar la fig. 2.8) muestra los siguientes valores:

- **Verdaderos positivos (VP):** el número de observaciones que fueron clasificadas correctamente en su clase por el modelo.
- **Falsos positivos (FP):** el número de observaciones que fueron incorrectamente clasificadas en la clase C_n por el modelo pero pertenecían a C_i con $i \neq n$.
- **Verdaderos negativos (VN)**: el número de observaciones que no pertenecían a la clase C_n y fueron clasificadas por el modelo en una categoría diferente a C_n.
- **Falsos negativos (FN):** el número de observaciones que no pertenecían a la categoría C_n y fueron incorrectamente clasificadas en dicha clase por el modelo.

2.7.2. Curva ROC

La curva ROC (Receiver Operating Characteristic) es una representación gráfica que ilustra el rendimiento de un modelo de clasificación para diversos umbrales de discriminación. Esta curva representa la tasa de verdaderos positivos (TPR), sensibilidad o recall en el eje y y la tasa de falsos positivos (FPR) o $1 - \textit{especifidad}$ en el eje x, la cuál se calcula como:

$$FPR = \frac{FP}{FP + TN}$$

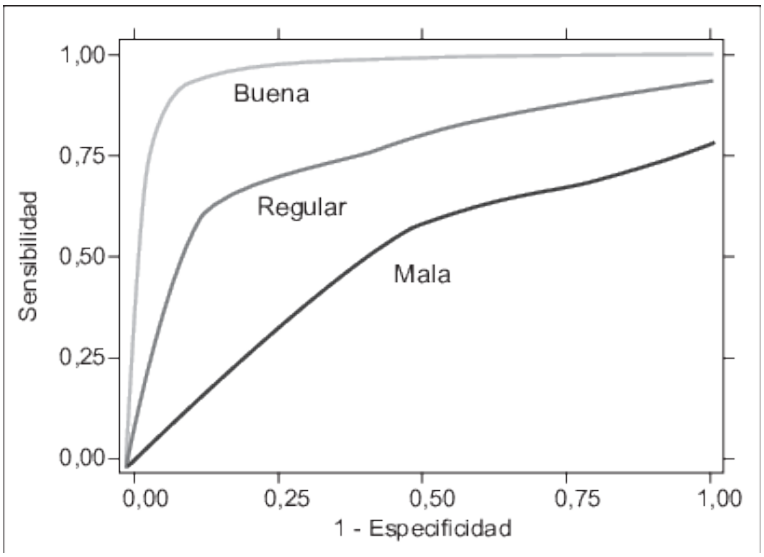

Figura 2.9: Ejemplo de curva ROC. Imagen presentada en [6]

A partir de la curva ROC se puede calcular el área bajo la curva (AUC-ROC), que proporciona una medida resumida del rendimiento del modelo, donde un valor más cercano a 1 indica un mejor rendimiento en la clasificación de los datos (ver 2.9)

2.7.3. Precisión, recall y F-score

La precisión y el recall proporcionan información sobre diferentes aspectos del rendimiento del modelo, y la puntuación F1 combina estas dos medidas en una métrica única.

- **La precisión:** mide la proporción de ejemplos correctamente clasificados para una clase específica entre todos los ejemplos que el modelo predijo como pertenecientes a esa clase. La precisión mide el nivel de calidad del modelo y su fórmula se puede observar en 2.1.

- **Recall:** el recall o sensibilidad indica la proporción de ejemplos correctamente clasificados para una clase específica entre todos los ejemplos reales que pertenecen a esa clase (observar su fórmula en la tabla 2.1), es decir, la capacidad del modelo de detectar una clase. En la práctica, es fundamental maximizar tanto la precisión como el recall, aunque esta tarea puede ser desafiante debido a su relación inversa. Aumentar la precisión puede resultar en una disminución del recall, y viceversa.
- **F1-score:** es una métrica que combina el recall y la precisión en una sola medida y proporciona una evaluación general del modelo. Se calcula como la media armónica ponderada de la precisión y el recall (véase 2.1) y es de gran utilidad en clases desbalanceadas.

Estos valores se calculan individualmente para cada clase, sin embargo, es posible resumir estos resultados mediante los micro y macro promedio (para cualquier métrica).

Métrica	**Fórmula**
$Precision_k$	$\frac{TP}{TP+FP} = \frac{MC(k,k)}{\sum_{i=1}^{N} MC(i,k)}$
$Recall_k$	$\frac{TP}{TP+FN} = \frac{MC(k,k)}{\sum_{i=1}^{N} MC(k,i)}$
$F1score_k$	$2\frac{precision_k * recall_k}{precision_k + recall_k}$

Tabla 2.1: Fórmulas de las métricas precisión, recall y F1-score por clase

Un promedio macro calculará la métrica de manera independiente para cada clase y luego tomará el promedio simple de estos valores. El micro promedio calcula las métricas para todas las etiquetas combinadas; se consideran como verdaderos positivos todas las predicciones que corresponden a verdaderos positivos para alguna etiqueta, es decir, los elementos en la diagonal de la matriz de confusión. Asimismo, se toman como falsos positivos todas las predicciones que no son verdaderos positivos para ninguna etiqueta, es decir los elementos fuera de la diagonal de la matriz de confusión.

Capítulo 3

Preparación de los datos

La preparación de datos desempeña un papel fundamental en el ámbito del Aprendizaje Automático, ya que la calidad de los datos resulta ser determinante en el desempeño y la eficacia de los modelos predictivos. En este capítulo, se llevará a cabo la preparación de los datos mediante técnicas de limpieza de texto y el Autotokenizer de BERT. Después, se realizará un análisis exploratorio de los datos para comprender su naturaleza e identificar patrones y relaciones relevantes entre las variables. A su vez, se implementarán estrategias como el oversampling y undersampling para abordar de manera efectiva el desequilibrio existente entre las diferentes clases presentes en los datos, que junto con técnicas mencionadas anteriormente, se encargarán de preparar de manera óptima los datos para el subsiguiente proceso de entrenamiento del modelo de Aprendizaje Automático.

3.1. Limpieza de los datos

Los datos se extraen directamente de un sistema de gestión de bases de datos MongoDB y consta de un dataframe que contiene dos columnas, la columna que contiene la comunicación o texto con etiquetas HTML y la columna con su respectiva etiqueta.

Para procesarlos se usó Python y los procesos aplicados a estos datos fueron los siguientes:

- Reemplazar espacios en blanco múltiples seguidos por uno solo.
- Remover todas las etiquetas HTML.
- Remover caracteres especiales.
- Aplicar tokenización y codificación del texto con el Autotokenizer de BERT.

3.1.1. Autotokenizer de BERT

La tokenización se define como el proceso de dividir un texto en unidades más pequeñas llamadas *tokens* que pueden ser palabras, símbolos de puntuación, números o cualquier otra cosa que se desee considerar como una unidad significativa; para nuestro caso conviene tomar las palabras como tokens. La tokenización ayuda a reducir la complejidad del texto al convertirlo en una secuencia de unidades más manejables que se pueden procesar computacionalmente.

El modelo BERT cuenta con su propio Autotokenizer [31], función integral que automatiza el proceso de tokenización el texto de entrada y lo codifica adecuadamente para que cumpla con el formato necesario al ingresar al modelo. BERT incluye dos modelos de tokenización:

- BERT Uncased: En la variante «Uncased» (en minúsculas) BERT trata todas las letras como minúsculas, sin tener en cuenta la capitalización original del texto de entrada.
- BERT Cased: el modelo conserva la capitalización original de las letras durante la tokenización

Durante la tokenización del texto para su análisis por BERT, se incorporan ciertos tokens especiales que son fundamentales para el proceso. Estos tokens son mencionados a lo largo del análisis y se detallan de la siguiente manera:

- Token [CLS]: Indica el inicio del texto y es el token que concentra la información principal relacionada con el contexto después del análisis.
- Token [SEP]: Marca la separación entre segmentos de texto o frases en el input.
- Token [MASK]: Es un token especial utilizado por BERT para hacer predicciones y encontrar palabras clave que aportan información significativa al contenido.
- Token [PAD]: Funciona como un token de relleno durante la etapa de tokenización del texto, facilitando el procesamiento.

3.2. Análisis exploratorio de los datos

Los datos se componen de algunas de las comunicaciones que los usuarios envían al CFG a través de la plataforma Sinco, abarcando un total de 31,174 mensajes distribuidos en 160 categorías distintas, etiquetados manualmente por un equipo de personas para ser posteriormente enviado al departamento encargado. Estas comunicaciones expresan en su mayoría una problemática relacionada tanto con la plataforma de Sinco como de los proyectos desarrollados en esta. Así, las clases o categorías suponen las situaciones más frecuentes presentadas en los datos.

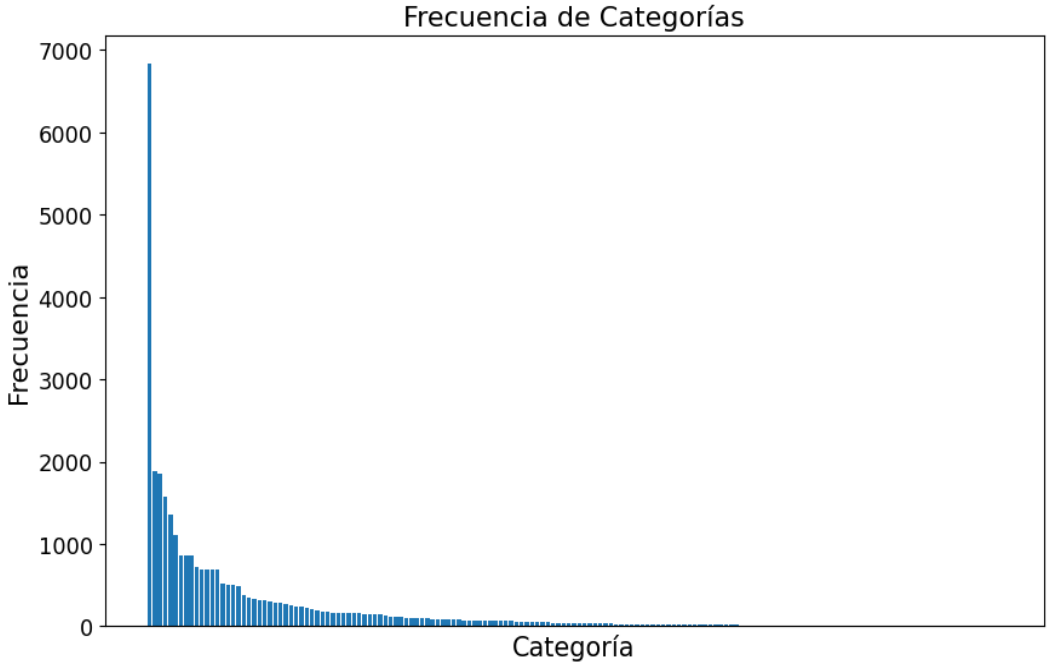

Figura 3.1: Gráfico de distribución de los mensajes por categoría

En la gráfica 3.1 podemos observar la cantidad de comunicaciones asociadas para cada categoría, en la cuál se destaca que la gran mayoría de comunicaciones se encuentran concentradas en unas pocas categorías. Por otro lado en la tabla 3.1 podemos observar el nombre de las categorías más frecuentes junto la frecuencia de mensajes en esa clase. Al analizar esta tabla se observan categorías redundantes o repetidas. Por ejemplo la clase «*CAMBIO DE VOCERIA RESPONSABLE DEL PROYECTO* » y «*Carga y estatus de proyectos / CAMBIO DE VOCERÍA RESPONSABLE*» contienen datos que tratan del mismo tema; aún así, fueron clasificadas con clases diferentes.

Esta situación se repite, por lo que se examinaron todas las categorías y su contenido con el fin de

unir aquellas que trataban del mismo tema en una sola clase; luego de este procedimiento se redujo la cantidad de categorías de 160 a 90. La clase «*Solicitud de activación usuario sinco* » es un un valor atípico ya que solo esta categoría contiene el 20 % de los datos, no obstante, debe considerarse que estas comunicaciones fueron comunes a inicios del lanzamiento de la plataforma debido a que muchos usuarios no tenían suficiente información para la activación de su usuario, situación que ya no es frecuente o relevante para el CFG. Por esta razón esta categoría no se tomará en cuenta para el entrenamiento del modelo.

Otro punto a considerar es que el nombre de la categoría 16 en la tabla 3.1 está vacío, lo cual no nos da ninguna información sobre el contenido que fue asignado a dicha categoría. Por esta razón se descartará del conjunto de datos.

Índice	Categoría	Frecuencia
0	SOLICITUD DE ACTIVACION USUARIO SINCO	6828
1	CAMBIO DE VOCERIA RESPONSABLE DEL PROYECTO	1889
2	PROCESO DE VALORACION DEL PROYECTO	1849
3	COMO CANCELAR UN PROYECTO REGISTRADO	1578
4	PASOS PARA REGISTRAR UN PROYECTO	1360
5	PROCEDIMIENTO PARA EL CAMBIO DE CORREO ELECTRÓNICO	1102
6	DESCONOCIMIENTO DEL PROCESO DE LA RENDICIÓN DE CUENTAS-VOCERIA	856
7	Carga y estatus de proyectos / PROCESO DE VALORACIÓN DEL PROYECTO	730
8	DIFICULTAD PARA EDITAR UN CAMPO EN EL MODULO DE REGISTRAR	686
9	Ejecución de Proyecto / REPORTE DE INSUMOS	659
10	Comunicaciones / RESPUESTA A COMUNICACIÓN INFORMATIVA	646
11	Carga y estatus de proyectos / CAMBIO DE VOCERO*A* RESPONSABLE	629
12	DIFICULTAD PARA EDITAR UN CAMPO EN EL MODULO DE RENDICION DE CUENTAS	516
13	Electricidad	502
14	Agrícola	500
15	Agua	482
16		414
17	Carga y estatus de proyectos / CANCELAR PROYECTO REGISTRADO NO APROBADO	372
18	Organizaciones / PROBLEMAS CON EL RIF O CÓDIGO SITUR	369
19	CAUSAS DE LA CANCELACION DEL PROYECTO POR EL FCI	349

Tabla 3.1: Primeras 20 categorías más comunes. La columna «categoría» contiene el nombre de la categoría y la columna «frecuencia» contiene la cantidad de mensajes que están clasificados en esa categoría

Luego de esta limpieza se disminuye considerablemente la cantidad de clases cuya distribución se observa en la figura 3.2. Podemos notar que nuestra distribución de las categorías sigue la distribución de Pareto [34] por lo que se segmentará las categorías en las más relevantes usando la Ley de Pareto o también llamada Regla 80/20, que postula que el 80 % de los resultados lo generan el 20 % de las causas [35].

Esta ley se utiliza en una variedad de campos, como en la economía en donde se observa que el 80 % de la riqueza de una nación suele estar en manos del 20 % de la población, en negocios donde se identifica que el 80 % de las ventas de una empresa provienen del 20 % de los clientes o en gestión

de proyectos, siendo el 80 % de los problemas y retrasos en un proyecto explicados por el 20 % de las tareas o situaciones.

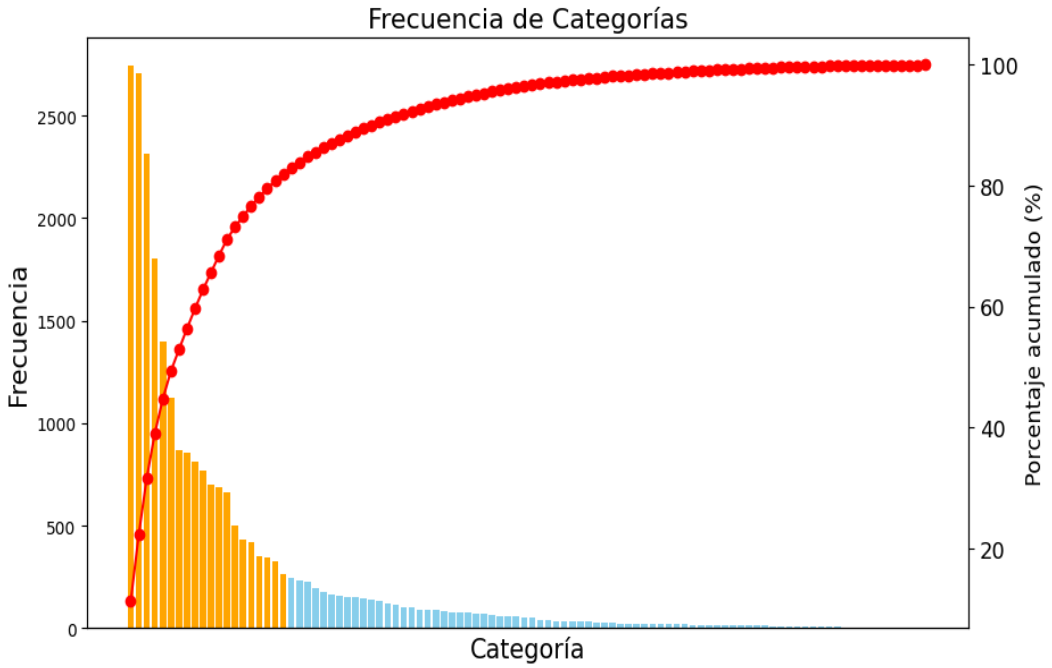

Figura 3.2: Gráfico de distribución de los mensajes por categoría junto con la curva de Pareto que muestra la frecuencia acumulada de las categorías.

De esta forma, no sorprende que de 90 categorías, en menos del 20 % de estas se encuentre aproximadamente el 80 % de los mensajes, así, nos concentraremos en esta segmentación y se trabajará con 15 categorías (las cuáles se especifican en la tabla 3.2) y un total de 17299 comunicaciones para el entrenamiento y evaluación del modelo cuya distribución por clase se aprecia en la figura 3.3.

Índice	Categoría	Frecuencia
0	Cambio de vocería	2687
1	Proceso de valorción del proyecto	2611
2	Cancelación de proyecto	2302
3	Rendición de cuentas	1797
4	Pasos para registrar un proyecto	1392
5	Cambio de correo electrónico	1126
6	Dificultad en el modulo de registrar	843
7	Electricidad	680
8	Ejecución de Proyecto / REPORTE DE INSUMOS	676
9	Reclamos, sugerencias y agradecimientos	671
10	Agua	634
11	Problemas con el RIF o código Situr	534
12	Agrícola	500
13	Estatus de desembolso	423
14	Problemas con cuentas bancarias	423

Tabla 3.2: Categorías que se usarán para crear el conjunto de datos

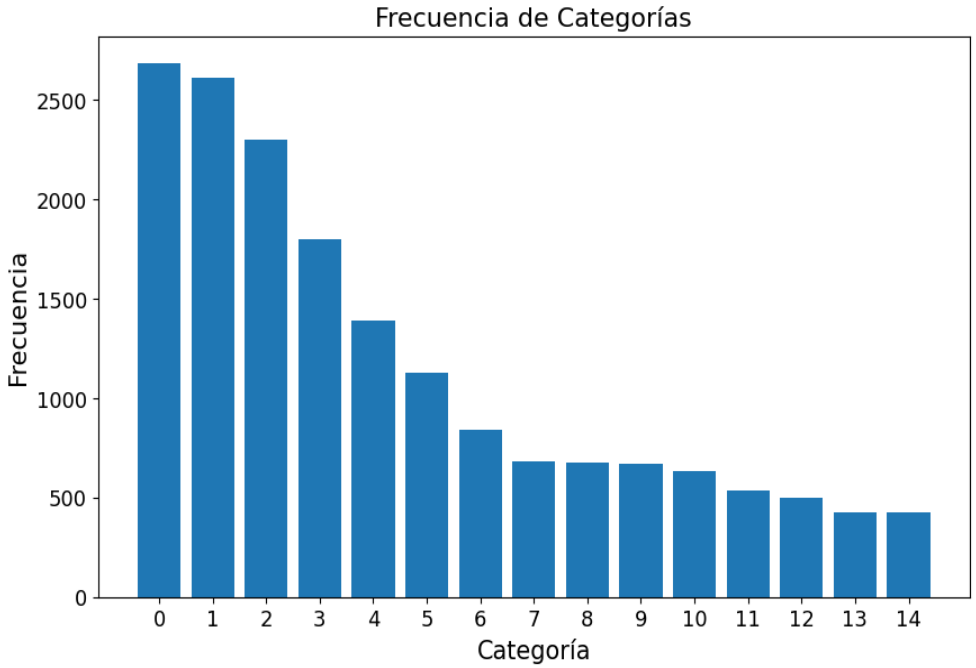

Figura 3.3: Gráfico de distribución de los mensajes por categoría luego de la reducción mediante la curva de Pareto.

3.2.1. Longitud de las comunicaciones

Un hiperparámetro importante a establecer es la longitud máxima o tamaño máximo de una secuencia de entrada al modelo, la cuál llamaremos *max_length* y se refiere a la cantidad de palabras o tokens por comunicación. En las comunicaciones de Sinco se puede usar la cantidad de palabras que el usuario desee, por lo que se cuentan con mensajes de diferente longitud. Sin embargo, se debe establecer un valor de longitud máxima para garantizar que todos los datos de entrada tengan la misma dimensión y que el modelo sea capaz de capturar toda la información relevante de la secuencia de entrada.

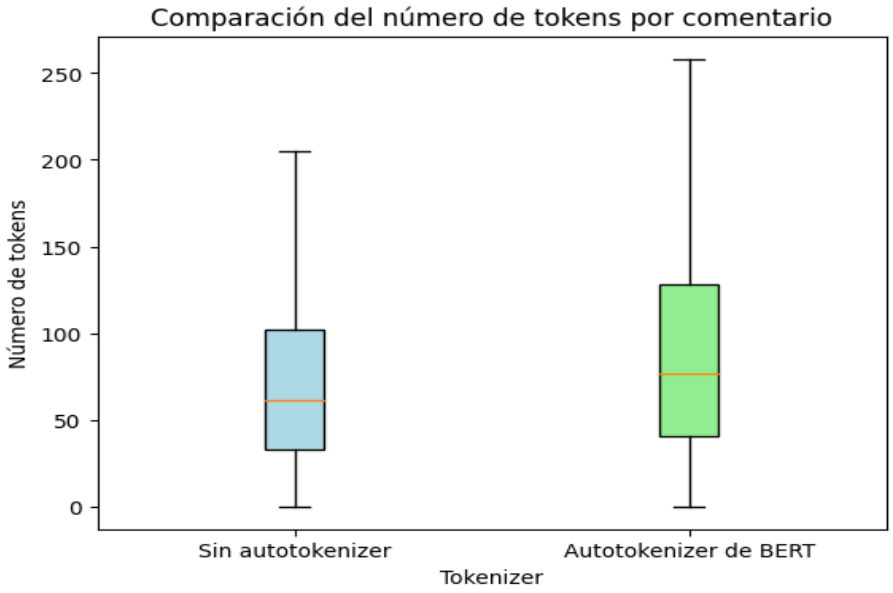

Figura 3.4: Diagrama de cajas que representan la distribución de la cantidad de tokens por datos sin aplicar el autotokenizer de BERT (diagrama de la izquierda) y luego de aplicar el autotokenizer de BERT (diagrama de la derecha)

En la figura 3.4 podemos observar la distribución de la cantidad de tokens por comunicación, así, al aplicar el Autotokenizer de BERT aumenta la cantidad de tokens por dato. Esto es debido a que muchas palabras de nuestro contexto son desconocidas para el modelo BERT por lo cual este procede a segmentar estas palabras desconocidas en palabras que estén dentro de su vocabulario.

Considerando el diagrama de caja luego de aplicar el Autotokenizer de BERT en 3.4 se establecerá $max\_length = 130$, que representa el tercer cuartil de este conjunto de datos, por lo que las comunicaciones que tengan más de 130 tokens serán truncadas en el proceso de tokenización y aquellas con menos de 130 palabras serán completadas con el Token especial [PAD].

3.3. Técnicas para trabajar con datos desequilibrados

El desequilibrio de clases dentro de un problema de clasificación multiclase en un modelo de aprendizaje de máquina se refiere a la situación en la que las clases en los datos de entrenamiento no están representadas de manera equitativa. En otras palabras, algunas clases pueden tener significativamente más ejemplos que otras, lo que puede afectar negativamente el rendimiento y la capacidad predictiva del modelo al volverse sesgado hacia las clases dominantes, lo que lleva a una menor precisión en la predicción de las clases minoritarias. Además, la métrica de precisión por sí sola puede ser engañosa en presencia de desequilibrios de clases, ya que un modelo que predice la clase mayoritaria en la mayoría de los casos aún puede tener una alta precisión.

Para abordar este problema, la comunidad investigadora [7] suele optar principalmente por estas técnicas:

- **Procesado**: consiste en procesar la muestra de datos antes del entrenamiento con el fin de reducir la disparidad entre las clases
- **Refinamiento de algoritmos:** consiste en modificar o ajustar los parámetros de los clasificadores para considerar el desequilibrio presente y/o ajustar la penalización por una clasificación incorrecta.

3.3.1. Sobremuestreo

El sobremuestreo u «oversampling» en inglés es una técnica utilizada en el procesamiento de datos que consiste en aumentar artificialmente el tamaño de la muestra de la clase minoritaria mediante la generación de nuevas muestras sintéticas, con el fin de equilibrar la distribución de las clases en un conjunto de datos desbalanceado.

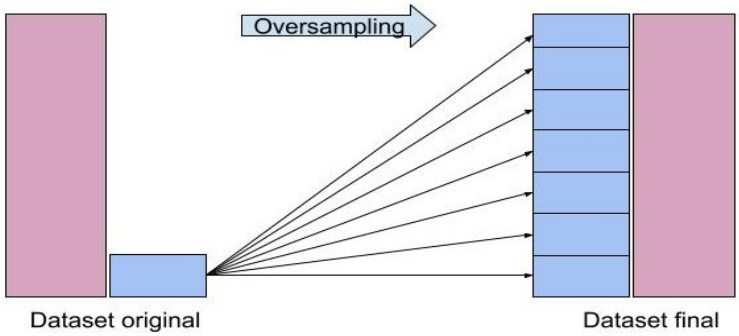

Figura 3.5: Sobremuestreo aleatorio. Imagen presentada en [7]

Este proceso puede ser por medio del **sobremuestreo aleatorio** que implica seleccionar muestras al azar con reemplazo de las clases minoritarias de forma independiente hasta tener la misma cantidad

de datos que las clases dominantes (ver la figura 3.5). Otra forma es generando muestras sintéticas utilizando técnicas como **SMOTE** (Synthetic Minority Over-sampling Technique) en el cuál se crean nuevas instancias de la clase menos representada a partir de ejemplos preexistentes; SMOTE identifica un punto de la clase minoritaria y elige uno de sus vecinos más cercanos para generar un nuevo dato sintético entre ellos (ver fig 3.6).

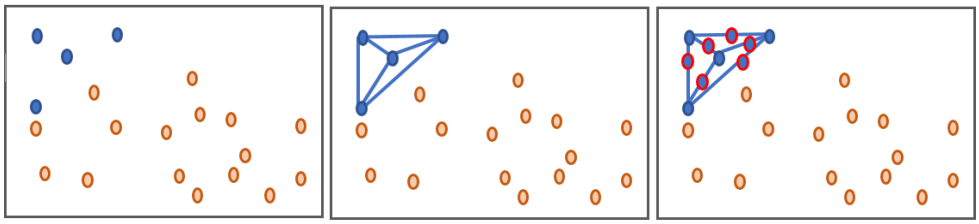

Figura 3.6: Funcionamiento SMOTE. Fuente [8]

Aumentar la muestra de las clases minoritarias permite equilibrar la distribución de clases, no obstante, el oversampling también presenta desventajas importantes a considerar. Uno de los principales riesgos es el **overfitting** o sobreajuste, donde el modelo puede ajustarse excesivamente a los datos generados de forma artificial durante el proceso de oversampling, lo que puede llevar a un rendimiento deficiente en datos nuevos. Asimismo, existe la posibilidad de distorsionar la distribución original de los datos y sesgar las predicciones, especialmente si no se gestiona adecuadamente la generación de nuevas muestras. Además, el aumento en el tiempo de entrenamiento puede ser significativo al incrementar el tamaño del conjunto de datos a través del oversampling, lo que podría resultar en procesos computacionalmente más intensivos.

La técnica más adecuada para aplicar a datos secuenciales, como el texto, es el sobremuestreo aleatorio, ya que resulta más sencilla su implementación en este contexto en comparación con la técnica SMOTE. Así, en la gráfica 3.3 vemos que la clase mayoritaria contiene 2.687 muestras por lo que al aplicar sobremuestreo aleatorio con la clase *RandomOverSampler* de la librería *imblearn* en Python todas las clases se equilibran obteniendo la misma cantidad de datos.

3.3.2. Submuestreo

De manera análoga, el submuestreo o «undersampling» implica reducir la cantidad de instancias de la clase mayoritaria para equilibrar la proporción entre las clases. Esta técnica es útil cuando el sobremuestreo no es factible o no es eficaz y se desea generar un conjunto de datos más equilibrado sin introducir instancias sintéticas.

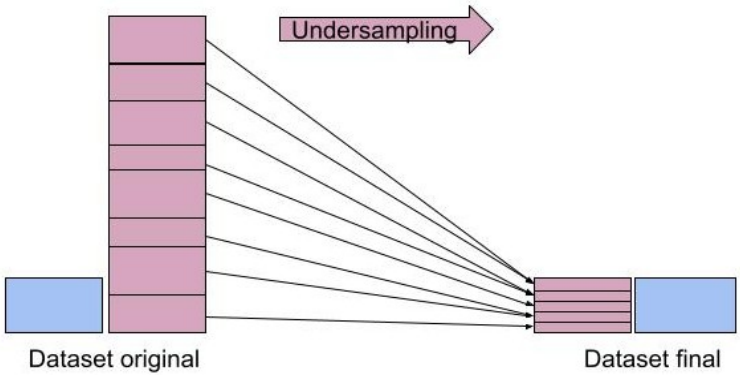

Figura 3.7: Submuestreo aleatorio. Imagen presentada en [7]

Como en el oversampling, destacamos dos maneras distintas de aplicarlo: submuestreo aleatorio y

el algoritmo Tomek Links. El **submuestreo aleatorio** consiste en eliminar muestras aleatorias de la clase mayoritaria hasta que se logra un equilibrio deseado entre las clases (ver fig 3.7), con el objetivo de mejorar la capacidad del modelo para predecir de manera precisa ambas clases. En cambio, el **algoritmo Tomek** se encarga de identificar pares de instancias pertenecientes a clases diferentes pero muy similares, llamados «tomek links», y eliminar la instancia correspondiente a la clase mayoritaria (observar fig. 3.8). Esta acción posibilita definir grupos claramente en el conjunto de entrenamiento y puede mejorar el rendimiento de la clasificación.

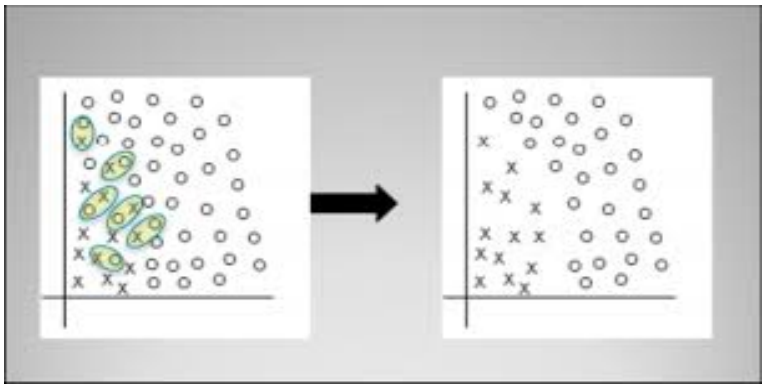

Figura 3.8: Funcionamiento del algoritmo Tomek Link. Imagen presentada en [9]

Al disminuir la cantidad de instancias de la clase mayoritaria, se reduce la influencia desproporcionada de esta clase en la toma de decisiones del modelo y se promueve un enfoque más equilibrado en la clasificación de datos, además, puede contribuir a mejorar la eficiencia computacional al trabajar con conjuntos de datos más pequeños y balanceados, lo que puede resultar en tiempos de entrenamiento más rápidos y un uso más óptimo de los recursos. Sin embargo, una de las principales desventajas de esta técnica es la pérdida de información potencialmente relevante al eliminar muestras de la clase mayoritaria, lo que puede resultar en la reducción de la representatividad de los datos y en la pérdida de detalles importantes para la clasificación.

En este trabajo se usó la clase *RandomUnderSampler* de la librería *imblearn* en Python para aplicar el submuestreo aleatorio a los datos, por lo que todas las clases disminuyeron sus muestras a 423, siendo esta la cantidad de muestras de la clase minoritaria (ver gráfica 3.3).

3.3.3. Refinamiento de algoritmos

Otro enfoque para abordar la problemática del desbalanceo es a través del aprendizaje sensible al coste. En este método el modelo se orienta a minimizar el coste, lo que implica una atención preferente a la clase minoritaria por medio del ajuste de los algoritmos. La técnica más común consiste en asignar pesos a cada clase en función de su desequilibrio en el conjunto de datos. Esto significa que el algoritmo da más importancia a la clase minoritaria al calcular la función de pérdida durante el entrenamiento asignando pesos más altos a la clase minoritaria y pesos más bajos a la clase mayoritaria, por lo que el modelo aprende a prestar más atención a la clasificación precisa de la clase minoritaria, lo que puede mejorar significativamente su capacidad para capturar patrones en los datos desbalanceados.

Sin embargo, si la proporción entre las clases es muy desigual, la ponderación de clases puede no ser suficiente para compensar la falta de representación de la clase minoritaria. Por otro lado, si los pesos se ajustan en exceso para la clase minoritaria, el modelo podría sobreajustarse a los datos de entrenamiento y no generalizar adecuadamente en datos nuevos.

3.4. Entorno y librerías

Para llevar a cabo de la clasificación, se optó por usar el lenguaje **Python** versión 3.10 en un *Jupyter Notebook* con un plan variable de consumo de RAM. Python es uno de los lenguajes de programación más destacados en el ámbito del Aprendizaje Automático debido a su condición de software libre, lo que ha generado la creación de una amplia variedad de bibliotecas que abarcan prácticamente todas las técnicas existentes en este campo. A continuación se presentan las principales bibliotecas usadas en el presente trabajo:

- **Transformer** (versión 4.4.2): Esta librería se enfoca en el procesamiento del lenguaje natural y proporciona una amplia gama de herramientas y modelos preentrenados[36].
- **Matplotlib** (versión 3.7.1): Ofrece una gran flexibilidad y control sobre la creación de gráficos estáticos, lo que resulta fundamental para visualizar datos y patrones en el contexto del aprendizaje automático [37].
- **Pytorch-lightning** (versión 2.2.4): es una estructura ligera que actúa como un envoltorio eficiente para proyectos de investigación en inteligencia artificial de alto rendimiento. Proporciona una estructura organizada para proyectos basados en PyTorch, lo que permite escalar los modelos sin necesidad de escribir una gran cantidad de código adicional. Esta estructura simplifica tareas comunes, como el entrenamiento y la validación de modelos, permitiendo a los investigadores concentrarse en el diseño y la experimentación de modelos en lugar de preocuparse por la infraestructura subyacente [38].
- **Imbalanced-learn** (versión 0.12.2) biblioteca basada en scikit-learn que proporciona herramientas específicas para abordar problemas de clasificación con clases desequilibradas [39].
- **Sklearn-pandas** (versión 2.2.0): el paquete sklearn-pandas proporciona una integración estrecha entre las bibliotecas pandas y scikit-learn, lo que resulta beneficioso al trabajar con conjuntos de datos al facilitar la conversión entre los DataFrames de pandas y las matrices NumPy requeridas por scikit-learn.

Capítulo 4

Experimentación

En este último capítulo, se abordará la fase crucial de experimentación, donde se detallarán la metodología empleada para entrenar el modelo basado en BERT, la aplicación de remuestreo de datos, entrenar y evaluar el desempeño del modelo, la descripción de la arquitectura resultante de un modelo base luego de la etapa de ajuste fino, la cuál será usada para generar modelos diferentes considerando técnicas para enfrentar el desequilibrio de las clases en los datos y, finalmente, se presentarán los resultados obtenidos junto con un análisis de los mismos.

4.1. Metodología

Después de procesar el conjunto de datos, podemos aplicar diversas técnicas para abordar el desequilibrio de los datos y desarrollar cuatro modelos distintos. Cada modelo se entrenará con: los datos originales, los datos generados mediante oversampling, los datos generados mediante undersampling y la técnica de asignación de pesos. Los pasos a seguir son:

1. Fijar las métricas a usar para evaluar nuestro modelo. En nuestro caso se observará la gráfica de pérdida, la matriz de confusión, el recall, la precisión y el F1-score.
2. Dividir el conjunto de datos en dos subconjuntos, uno para el entrenamiento del modelo y otro para la evaluación.
3. Establecer unos hiperpárametros de partida y entrenar el modelo una vez.
4. Evaluar el modelo con las métricas antes mencionadas usando los datos de validación cruzada.
5. Ajustar el algoritmo, modificando los hiperparámetros y entrenando nuevamente hasta obtener mejores resultados. Este modelo resultante será llamado *modelo 1*.
6. Entrenar el modelo con la arquitectura resultante en el paso anterior aplicando técnicas de oversampling, undersampling y asignación de pesos para obtener los parámetros del *modelo 2*, *modelo 3* y *modelo 4*, respectivamente. En este escenario, la técnica de oversampling y undersampling se implementará en las particiones utilizadas para entrenar el modelo, dejando que la partición de prueba conserve el desequilibrio original para ofrecer una evaluación más precisa del rendimiento.
7. Usar el conjunto de prueba para evaluar cada modelo y comparar los resultados obtenidos.

4.2. Remuestreo

El remuestreo, también conocido como **resampling** en inglés, es una estrategia en la cuál se genera un subconjunto de los datos de entrenamiento (usualmente de forma aleatoria) para ajustar el modelo,

y se evalúa con las observaciones restantes [40]. Recordemos que el objetivo de un modelo es predecir la clase de observaciones futuras u observaciones que el modelo no ha «visto» antes; después de entrenar un modelo, el error mostrado por defecto suele ser el error de entrenamiento y, aunque este error es útil para comprender cómo está aprendiendo el modelo (mediante el estudio de residuos), no proporcionan una estimación realista del comportamiento del modelo ante nuevas observaciones. De aquí, para obtener una estimación más precisa es necesario recurrir a un conjunto de pruebas o implementar estrategias de validación basadas en remuestreo.

El enfoque más sencillo de validación implica la distribución aleatoria de las observaciones disponibles en dos grupos: uno se utiliza para entrenar el modelo y el otro para evaluar su desempeño. De igual forma, este último subconjunto se subdivide en otros dos grupos: el conjunto de validación cruzada y el conjunto de prueba:

- **Conjunto de entrenamiento o train set:** se utiliza para ajustar los parámetros del modelo y permitirle aprender de los datos. Para este trabajo, este conjunto representará el 70 % de los datos disponibles para el entrenamiento.
- **Conjunto de validación cruzada o cross validation set**: es un subconjunto usado para ajustar los hiperparámetros del modelo y evaluar su rendimiento durante el proceso de ajuste y de esta forma escoger la mejor configuración de hiperparámetros. El 15 % de los datos serán usados para la validación cruzada.
- **Conjunto de prueba o test set**: es un subconjunto que se reserva completamente para evaluar el rendimiento final y la capacidad de generalización del modelo una vez que se ha ajustado y validado; para este conjunto se usará el 15 % de los datos restantes.

4.3. Arquitectura y funcionamiento del modelo

Luego de ajustar el algoritmo por medio de prueba y error, probando con diferentes configuraciones de hiperparámetros y evaluando con los datos de validación cruzada, se llegó a la arquitectura mostrada en la figura 4.1.

Esta arquitectura utiliza el modelo preentrenado de BERT en español (BETO) con sus parámetros originales. Se añade una capa de promedio a la salida de este modelo para resumir toda la información extraída de BETO mediante el cálculo del promedio simple del vector que representa cada token, lo que reduce la dimensionalidad de la salida. El ajuste fino aplicado a BETO consta de tres capas ocultas adicionales a la salida del modelo, con 758, 758 y 350 neuronas, respectivamente. Además, se introduce una última capa con 15 neuronas, la cual funciona como la capa clasificadora encargada de asignar los valores que luego se pasan a la función softmax, generando las probabilidades de que la frase de entrada pertenezca a cada clase.

A continuación se explicarán superficialmente las transformaciones que realiza el modelo a una frase de entrada. Como es sabido, los datos están compuestos por una secuencia de texto la cuál es pasada primeramente por el autotokenizer de BETO, que transforma las entradas en una matriz $X_{n\times 130}$, según lo explicado en la sección 3.1.1, donde n es el tamaño del lote que entran simultáneamente al modelo preentrenado BETO. Luego de salir de este gran modelo, la entrada X es ahora un arreglo tridimensional de tamaño $n \times 130 \times 768$ donde cada uno de los 130 tokens de las n observaciones son representados por un vector de 768 elementos que contiene la información aprendida durante su paso por BETO; seguidamente, al arreglo $X_{n\times 130\times 768}$ se le aplica el promedio simple para reducir la dimensionalidad a $n \times 768$.

Posteriormente, a la matriz $X_{n\times 768}$ se le aplica una transformación lineal al pasar por cada una de las cuatro capas ocultas de la figura 4.1 cuya función de activación es la función ReLu. Así, al salir de la

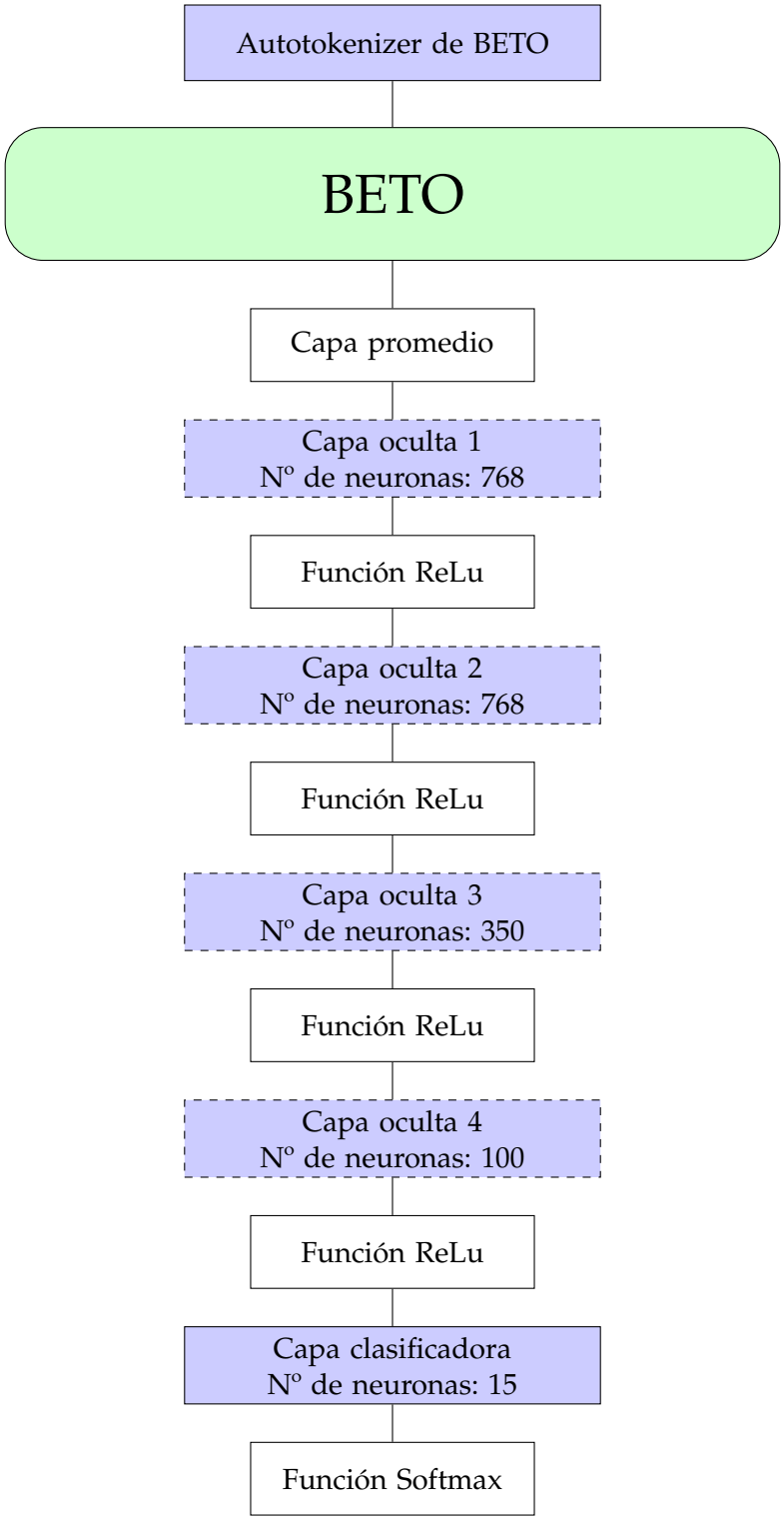

Figura 4.1: Ariquitectura principal del clasificador para las comunicaciones de Sinco.

capa 1, X tendrá un tamaño de $n \times 768$; lo mismo ocurre con la capa 2 y al salir de de la capa 3 la matiz X cambia a un tamaño de $n \times 350$ y por último, X reduce su tamaño a 350×100 al salir de la capa 4.

La última transformación lineal se aplica en la capa clasificadora que transforma la matriz a $X_{n \times 15}$ para ser pasada por una función Softmax que devuelve un vector $output = (p_1, p_2, ..., p_{15})$ por cada una de las n entradas, donde p_i es la probabilidad de que esa observación pertenezca a la clase i con $i = 1, ..., 15$.

Al momento del entrenamiento, además de pasar n cantidad de observaciones, también se deben pasar las etiquetas correspondientes $Y = (y_1, y_2, ..., y_n)$. Así, luego de que la entrada recorra todo el modelo descrito anteriormente se calcula la función de pérdida de entropía cruzada para actualizar los parámetros del modelo para la siguiente época. Este proceso se repite hasta que se procesen todas las observaciones y se cumplan todas las épocas.

4.4. Análisis de los resultados

Como se ha mencionado anteriormente, luego de obtener la arquitectura descrita en la sección anterior, el objetivo fue entrenar cuatro modelos diferentes usando la misma arquitectura base. El **modelo 1** se entrenó con los datos originales sin tomar en cuenta el desbalanceo de las clases, el **modelo 2** se entrenó con los datos a los cuales se les aplicó la técnica oversampling, por lo que fue entrenado con una mayor cantidad de datos. El **modelo 3** se entrenó con los datos luego de aplicar undersampling. Por último, llamaremos **modelo 4** a aquel que se entrenó con los datos originales pero asignó pesos a la función de pérdida, es decir, se designó un valor al hiperparámetro w_c en la ecuación 2.2 para cada clase, de tal manera que se penalice a las clases mayoritarias y se compensen las minoritarias en el momento del entrenamiento del modelo.

Para poder entender mejor los resultados presentados a continuación, es importante mencionar que las 15 clases están ordenadas de mayor a menor con respecto a la cantidad de observaciones por clase, siendo la clase 0 aquella con mayor cantidad de observaciones y la clase 14 la que tiene menor cantidad de datos.

4.4.1. Curva ROC

A continuación se muestran las gráficas resultantes de la curva ROC para cada modelo.

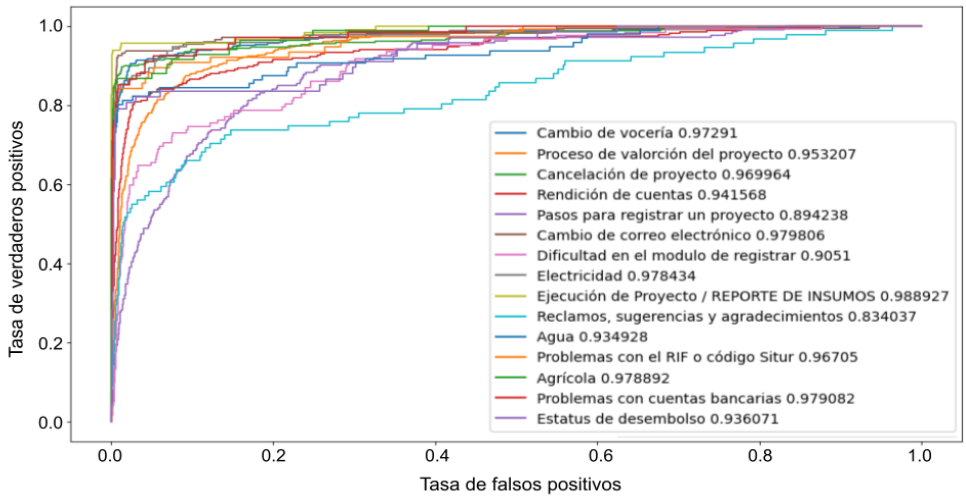

Figura 4.2: Curva ROC para el modelo 1.

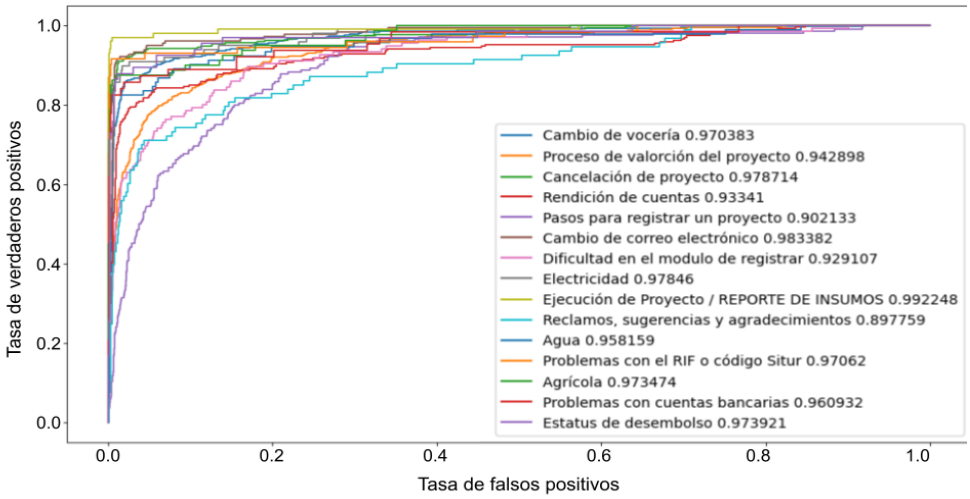

Figura 4.3: Curva ROC para el modelo 2.

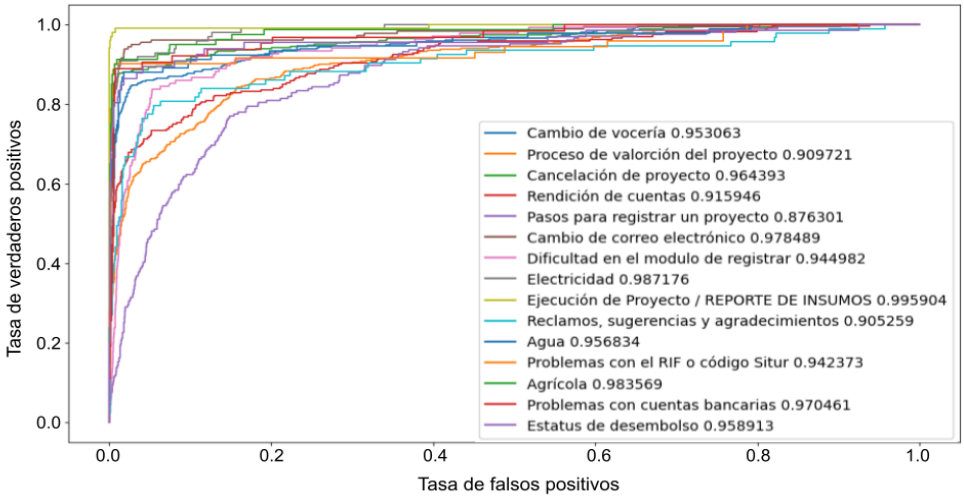

Figura 4.4: Curva ROC para el modelo 3.

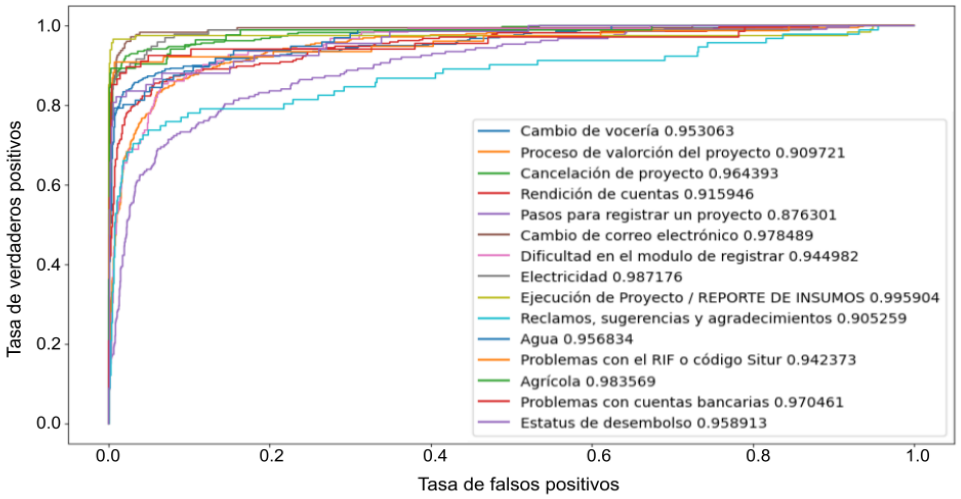

Figura 4.5: Curva ROC para el modelo 4.

Según estas gráficas, podemos deducir que en general los cuatro modelos tienen un comportamiento aceptable. En la gráfica 4.2 el modelo 1 presenta problemas con la clase «Pasos para registrar un proyecto» y «Reclamos, sugerencias y agradecimientos» cuyo valor AUC es de 0.89 y de 0.83 respectivamente, mientras que para el resto de las clases este valor esta sobre 0.90.

Según la gráfica 4.3 al aplicar oversampling sobre los datos, el modelo presenta una pequeña mejoría pues para cada clase se tiene un valor de AUC mayor a 0.90, siendo la clase «Reclamos, sugerencias y agradecimientos» la que tiene el menor valor de AUC por lo que este modelo parece tener problemas al clasificar esta clase. Ocurre un resultado similar para el modelo 3 y 4, cuyo menor valor de AUC es para «Pasos para registrar un proyecto» y «Reclamos, sugerencias y agradecimientos» cada una con un valor aproximado de 0.87.

Con estos resultados se ve claramente que hay dos clases las cuáles perjudican a los modelos.

4.4.2. Matiz de confusión

La matriz de confusión proporciona una visión detallada de cómo cada modelo clasifica las diversas clases y permite calcular varias métricas de rendimiento. Las figuras a continuación muestran matrices de confusión normalizadas por fila, donde las entradas representan la proporción de muestras en cada clase, en lugar del recuento de muestras.

La matriz de confusión de la figura 4.6 muestra que el modelo 1 falla al clasificar datos pertenecientes a las clases 4, 6, 9 y 14, es decir con las clases «Pasos para registrar un proyecto», «Dificultad en el modulo de registrar»,«Reclamos, sugerencias y agradecimientos» y «Estatus de desembolso» respectivamente, donde solo pudo clasificar correctamente un 60 % de las observaciones que pertenecían a la clase 6 y 14, un 50 % de la clase 4 y solo un 40 % de las muestras de la clase 9. A simple vista, no parece que el modelo falle al clasificar algunas clases por la falta de muestras en las últimas clases.

Observando la figura 4.7 podemos notar que la técnica del oversampling presenta mejores resultados que los mostrados en la matiz 4.6. El modelo 2, a comparación con el modelo 1, aprendió mejor de las clases 6, 8, 9, 11 y 14 ya que la cantidad de datos en estas clases era igual a las restantes. Sin embargo, presenta el mismo problema del modelo 1 al generalizar las clases 4 y 6.

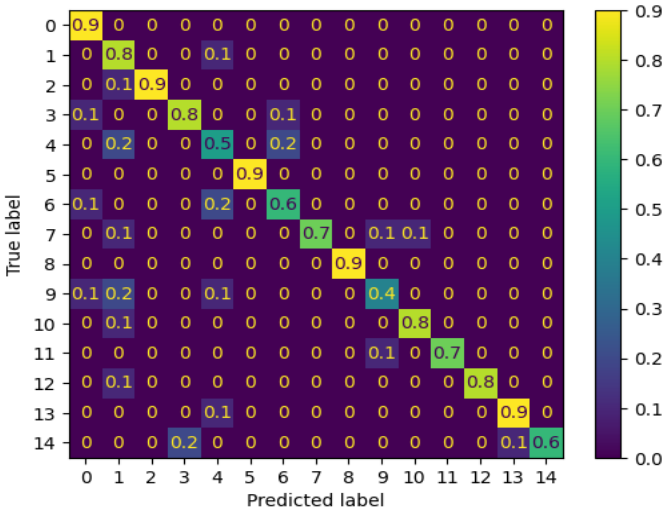

Figura 4.6: Matriz de confusión para el modelo 1.

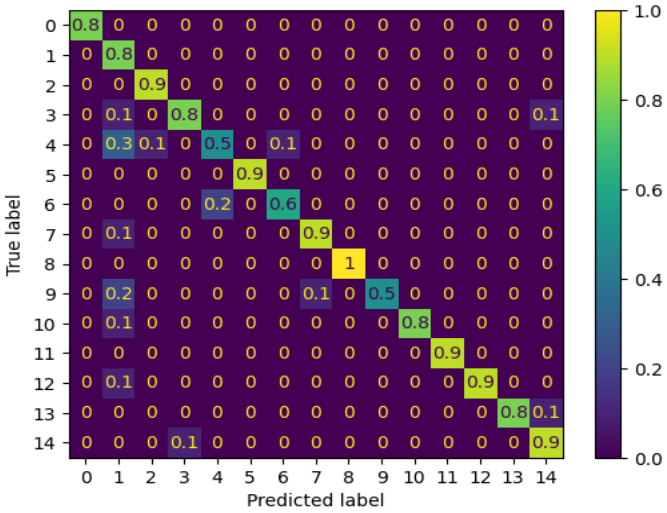

Figura 4.7: Matriz de confusión para el modelo 2.

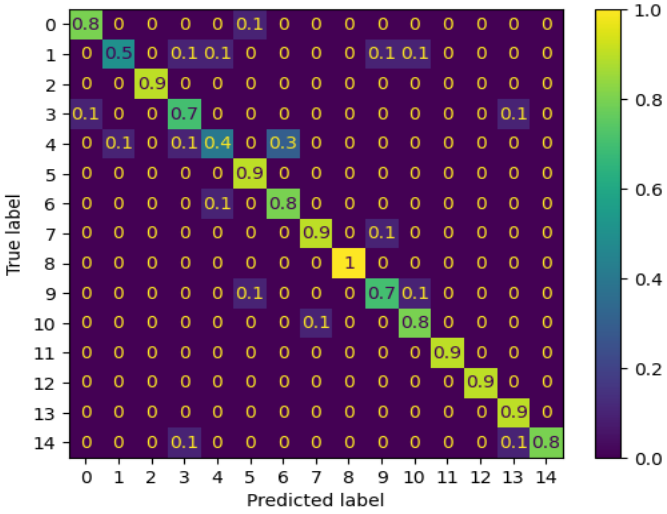

Figura 4.8: Matriz de confusión para el modelo 3.

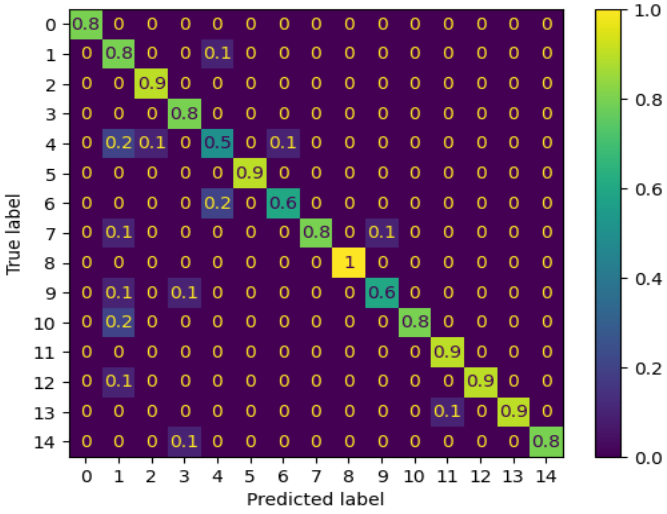

Figura 4.9: Matriz de confusión para el modelo 4.

El modelo 3 es mejor generalizando los datos de las clases 6 y 9 de acuerdo a la matriz 4.8. Sin embargo no es capaz de aprender los patrones de las clases 1 y 4. Finalmente, podemos concluir que el modelo 4 presenta los mejores resultados en comparación con los tres modelos anteriores, ya que el valor del porcentaje de datos clasificados correctamente por clase es más equilibrado en este modelo según los resultados mostrados en 4.9, aunque presenta mayor problema al reconocer observaciones de la clase 4, 6 y 9.

En todos los resultados anteriores, se ha encontrado la misma dificultad al identificar los patrones de la clase 4 «Pasos para registrar un proyecto», la 6 «Dificultad en el modulo de registrar» y la 9 «Reclamos, sugerencias y agradecimientos». Al revisar detalladamente estas categorías se llegó a la conclusión que estas clases no están bien representadas ya que muchas de las observaciones de estas clases fueron etiquetadas erróneamente y pertenecen o otras de las categorías restantes. Así, sin importar las mejoras que se puedan realizar a los modelos en un futuro, es posible que el impacto en estas clases sea la misma.

4.4.3. Precisión, recall y F1-score

Para calcular estas métricas, se usó el promedio macro descrito en la sección 2.7.3. En la siguiente tabla pueden verse los resultados obtenidos al evaluar cada modelo usando los datos de prueba:

	modelo 1	modelo 2	modelo 3	modelo 4
Precision	0.785559	0.776791	0.710992	0.793832
Recall	0.750356	0.790061	0.783281	0.801892
F1-score	0.763264	0.781563	0.734756	0.795622

Tabla 4.1: Valor de precision, recall y f1-score macro para cada modelo

En el cuadro 4.1 podemos observar que al entrenar el modelo sin considerar el desequilibrio de clases se tiene una precisión del 78.55 %, lo que significa que alrededor del 78.55 % de las predicciones positivas realizadas por este modelo fueron correctas. Con un recall del 75.03 %, el modelo 1 tiene una buena capacidad para encontrar todas las instancias relevantes de manera efectiva y el valor F1-score de 76.32 % indica un buen equilibrio entre la precisión y la exhaustividad del modelo, lo cuál es un buen resultado considerando el gran desequilibrio entre las clases.

Al aplicar oversampling, se ve una ligera mejoría en cuanto al recall con un valor del 79 % aproximadamente y el F1-score con un valor del 78.15 %. Sin embargo, el valor de la precisión es de 77.67 % lo que sugiere que la cantidad de predicciones correctas realizadas por el modelo en relación con el total de predicciones realizadas es menor con respecto al modelo 1.

El modelo 3 presenta el menor valor de F1-score (73.47 %) y de precisión (71.09 %), por lo que este modelo tiene baja capacidad de predicción, posiblemente causado por un sobreajuste al entrenarse con pocos datos luego del undersampling. Con una precisión del 79.38 %, un recall del 80.18 % y un F1-score del 79.56 %, el modelo 4 presenta los mejores resultados mostrando su capacidad no solo para predecir de forma correcta, sino también de identificar las características de las clases. Su valor de F1-score indica un buen equilibrio entre la precisión y el recall.

En resumen, los modelos 1 y 2 parecen tener un rendimiento similar, con puntajes consistentemente altos en precisión, recall y F1-score. El modelo 3 muestra puntajes ligeramente más bajos en comparación con los dos primeros modelos, especialmente en precisión, y el modelo 4 muestra los mejores resultados, por lo que se concluye que el refinamiento del algoritmo es una técnica útil al momento de trabajar con datos que presentan desequilibrio de clases.

Capítulo 5

Conclusiones y trabajo futuro

El presente capítulo proporciona un resumen integral del estudio realizado en este trabajo especial de grado, consolidando los hallazgos y destacando las contribuciones más relevantes. Además, se considerarán las limitaciones encontradas a lo largo de este trabajo y se ofrecerá una visión general de las perspectivas de futuro ofreciendo recomendaciones prácticas basadas en las conclusiones alcanzadas

5.1. Conclusiones

El estudio se centró en la investigación, implementación y evaluación de un modelo de aprendizaje automático basado en el ajuste fino de BETO para la clasificación multiclase de las comunicaciones de Sinco, con el propósito de abordar desafíos inherentes al PLN como lo es la clasificación de textos en un contexto de clases desbalanceadas. Durante este proceso, se lograron alcanzar varios objetivos específicos. En primer lugar, se logró comprender en profundidad el funcionamiento e implementación del modelo BETO, destacando su eficacia y versatilidad en la clasificación multiclase de textos en español. Asimismo, se investigaron y aplicaron diversas técnicas para enfrentar el desafío de clases desbalanceadas, lo que permitió mejorar la capacidad del modelo para detectar y clasificar las clases minoritarias con precisión. Además, se prepararon los datos para su posterior análisis y correcta representación en el formato adecuado para el entrenamiento del modelo. El análisis exploratorio detallado reveló la compleja naturaleza de los datos así como sus etiquetas correspondientes.

Uno de los principales objetivos fue enfrentar el reto de datos con un alto sesgo en las clases, problema que se abordó con el entrenamiento de cuatro modelos basados en una misma arquitectura adaptada de BERT pero con dos diferentes técnicas de procesado, como lo es el sobremuestreo y el submuestro, y la técnica de refinamiento de algoritmos.

Finalmente, se establecieron métricas de comparación y se evaluaron los resultados del rendimiento de los modelos en la tarea de clasificación multiclase. Se pudo observar que al entrenar los diferentes modelos, los resultados son prometedores comparándolos con resultados observados en otras investigaciones relacionadas. Por ejemplo, en [41] el autor obtuvo una precisión del 64 % al entrenar por 10 épocas un modelo basado en BERT-multilingual para la tarea de clasificación multiclase; Peréz J. en [42] usó diferentes modelos de clasificación multiclase para identificar la ideología política en tweets y, obtuvo un valor de F1-score de 40 % con BERT. Por último, en [43] se usaron diferentes versiones de BERT para la clasificación de texto de aplicaciones móviles educativas, donde el mejor resultado se consiguió con el modelo Roberta-base, con un F1-score del 80 %.

En nuestro caso, aunque los cuatro modelos arrojan resultados similares entre sí, podemos concluir que el oversampling, como estrategia para abordar el desequilibrio de clases, muestra una leve me-

jora en comparación con el modelo entrenado sin considerar el sesgo de clases. Por otro lado, el undersampling produjo los resultados menos significativos. En última instancia, el refinamiento de los algoritmos, el cual consiste en priorizar las clases minoritarias y penalizar las mayoritarias por medio del ajuste de algoritmos, se destacó como la técnica que generó los mejores resultados, lo que lleva a la conclusión de que su implementación adecuada puede potenciar el rendimiento de modelos de clasificación.

En conjunto, este trabajo proporciona una contribución significativa al campo del procesamiento del lenguaje natural (PLN) y la clasificación multiclase, al demostrar de manera concluyente la viabilidad y eficacia del modelo BERT mediante el ajuste fino para la clasificación de comunicaciones en un contexto corporativo como el de Sinco. Además, se evidenció que el proceso de etiquetado de los datos para el entrenamiento es igual de importante que la arquitectura del modelo para obtener resultados óptimos. Por último, las técnicas investigadas y aplicadas para el abordaje de clases desbalanceadas permiten anticipar el potencial impacto positivo de esta investigación en aplicaciones del mundo real.

5.2. Limitaciones y trabajo futuro

En este trabajo hemos visto una pequeña introducción al área de la clasificación de datos desbalanceados. Si bien es cierto que BERT ofrece buenos resultados, podemos aplicar ciertas técnicas que mejoran la calidad del modelo. Sin embargo, estas técnicas no mostraron mejoras muy significativas debido al ruido presentado en las etiquetas de las clases, esto debido a la presencia de sesgo por la interpretación subjetiva de los etiquetadores de los datos, que pueden tener diferentes opiniones o prejuicios al asignar etiquetas a los datos y llevar a que el modelo aprenda de manera incorrecta o sesgada, reproduciendo esos sesgos en sus predicciones.

Aún quedaría mucho trabajo futuro por hacer, por ejemplo, hacer un reetiquetado a las comunicaciones con el fin de disminuir el ruido en las categorías. Además, usar técnicas de remuestreo más complejas para equilibrar la distribución de las clases como SMOTE o Tomek Link que fueron mencionadas en el presente trabajo pero no pudieron ser tratadas por falta de tiempo. Otra posibilidad es realizar una búsqueda más minuciosa de hiperparámetros cuya configuración pueda presentar mejoras en los resultados. Por último, se recomienda desplegar el modelo en un equipo local para ayudar con la seguridad de los datos procesados

Bibliografía

[1] Conievte S. Bhattarai. What is gradient descent in machine learning? https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/, 2018. Accessed 14-May-2019.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[3] Dan Jurafsky and James H Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, chapter 10: Transformers and Pre-trained Language Models (3rd (draft) ed.)*. 2022.

[4] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[5] Wayner Barrios Bustamante. Calculando la precisión en un modelo de clasificación multiclase. https://wbarriosb.medium.com/calculando-la-precisi%C3%B3n-en-un-modelo-de-clasificaci%C3%B3n-multiclase-224d96f52043, 2024. Fecha de acceso: marzo 2024.

[6] MARÍA EUGENIA BURGOS and Carlos Manterola. Cómo interpretar un artículo sobre pruebas diagnósticas. *Revista chilena de cirugía*, 62(3):301–308, 2010.

[7] Blanca Abella Miravet et al. Mejora de las predicciones en muestras desbalanceadas. B.S. thesis, 2021.

[8] David Pugh. Balancing datasets and generating synthetic data with smote. https://datasciencecampus.github.io/balancing-data-with-smote/, 2019.

[9] KDnuggets. Learning from imbalanced classes. https://www.kdnuggets.com/2016/08/learning-from-imbalanced-classes.html/2/, 2016.

[10] SINCO. ¿qué es sinco? Recuperado de https://www.sinco.gob.ve/#:~:text=%C2%BFQu%C3%A9%20es%20SINCO%3F,de%20%C3%A9stas%20con%20la%20institucionalidad, s.f.

[11] Keegan McBride. Hola, mundo: La inteligencia artificial y su uso en el sector público. 2020.

[12] William D Eggers, David Schatsky, Peter Viechnicki, et al. How artificial intelligence could transform government. *https://www2.deloitte.com/us/en/insights/focus/cognitive-technologies/artificial-intelligence-government-summary.html*, 2017.

[13] Alexander Gelbukh. Procesamiento de lenguaje natural y sus aplicaciones. *Komputer Sapiens*, 1:6–11, 2010.

[14] Augusto Cortez Vásquez, Jaime Pariona Quispe, Ana Maria Huayna, et al. Procesamiento de lenguaje natural. *Revista de investigación de Sistemas e Informática*, 6(2):45–54, 2009.

[15] Alexander Gelbukh. Tendencias recientes en el procesamiento de lenguaje natural. *Proc. SICOM-2002*, 2002.

[16] Carlos Periñán Pascual. En defensa del procesamiento del lenguaje natural fundamentado en la lingüística teórica. *Onomázein*, (26):13–48, 2012.

[17] Leonardo Barón Birchenall and Oliver Müller. La teoría lingüística de noam chomsky: del inicio a la actualidad. *Lenguaje*, 42(2):417–442, 2014.

[18] Christiane Fellbaum. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, 2010.

[19] Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C-C Jay Kuo. Evaluating word embedding models: Methods and experimental results. *APSIPA transactions on signal and information processing*, 8:e19, 2019.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[21] Sebastián Alejandro Donoso Bustos. Entrenamiento y evaluación de modelos pequeños de lenguaje natural basado en métodos de autoatención. 2021.

[22] Dan Jurafsky and James H Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, chapter 5: Logistic Regression (3rd (draft) ed.)*. 2022.

[23] Rubén Rodríguez Abril. Aprendizaje de redes neuronales. `https://lamaquinaoraculo.com/deep-learning/aprendizaje-de-redes-neuronales/#:~:text=La%20funci%C3%B3n%20de%20p%C3%A9rdida%2C%20tambi%C3%A9n,se%20ha%20equivocado%20la%20red`. Fecha de acceso: marzo 2024.

[24] Crossentropyloss-pytorch. `https://pytorch.org/docs/master/generated/torch.nn.CrossEntropyLoss.html#crossentropyloss`. Accedido el 11/03/2024.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Dan Jurafsky and James H Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, chapter 7: Logistic Regression (3rd (draft) ed.)*. 2022.

[27] Diego Rivera López-Brea et al. No todo lo que necesitas es atención. 2022.

[28] Hugging Face. Hugging face: State-of-the-art natural language processing in ten lines of tensorflow, pytorch, and jax. `https://huggingface.co/`. Accessed: 2024-04-18.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[30] Iago Collarte González. Procesamiento del lenguaje natural con bert: Análisis de sentimientos en tuits. B.S. thesis, 2020.

[31] José Cañete, Gabriel Chaperon, Rodrigo Fuentes, Jou-Hui Ho, Hojin Kang, and Jorge Pérez. Spanish pre-trained bert model and evaluation data. In *PML4DC at ICLR 2020*, 2020.

[32] Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12799–12807, 2023.

[33] Rodrigo Adrián Fuentes Zúñiga. Prototipo de clasificador multiclase para relatos médicos. 2021.

[34] Vladimir Pokrovskii. The pareto distribution as a disguised gauss distribution. *arXiv preprint arXiv:2302.10751*, 2023.

[35] JI Gómez. Ley de pareto: 80/20. *Libros4economía*, 2007.

[36] Transformer. `https://huggingface.co/transformers/v4.4.2/installation.html`, 2020.

[37] Matplotlib. https://matplotlib.org/3.7.1/, 2012.

[38] pytorch-lightning. https://pypi.org/project/pytorch-lightning/, 2020.

[39] imbalanced-learn. https://imbalanced-learn.org/stable/, 2024.

[40] Joaquín Amat Rodrigo. Validación de modelos predictivos: Cross-validation, oneleaveout, bootstraping. https://cienciadedatos.net/documentos/30_cross-validation_oneleaveout_bootstrap. Accessed: 2024-05-01.

[41] Maximiliano Ponce Marquez, Samuel González-López, Aurelio López-López, and Guillermina Muñoz-Zamora. Método de recomendación para pruebas eléctricas fallidas en la industria de manufactura aplicando aprendizaje automático.

[42] Juan Carlos Pérez González. Clasificación de ideología política en textos. 2022.

[43] Anabel Pilicita and Enrique Barra. Using of transformers models for text classification to mobile educational applications. *IEEE Latin America Transactions*, 21(6):730–736, 2023.