TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Detección de conductas negativas en chats de videojuegos competitivos

Estudiante: Nicolás Rivas Rodriguez

Dirección: Diego Fernández Iglesias

Fidel Cacheda Seijo

A Coruña, septiembre de 2024.

A mi familia y amigos que me han acompañado durante esta etapa

Agradecimientos

Quisiera agradecer tanto a mis padres y a todos mis familiares por apoyarme durante estos cuatro años, así como a Diego Fernández Iglesias y Fidel Cacheda Seijo por haber confiado en este proyecto y haberme guiado durante todo el proceso.

Resumen

El auge de los *eSports* ha contribuido al aumento de los jugadores en videojuegos como “Counter-Strike”, “Valorant” o “League of Legends”. Si bien esto es algo positivo, al juntar a personas de diferentes regiones como si de una red social se tratase, también arrastra consigo una nube de toxicidad que, por el bien de estas mismas comunidades, debe ser controlada.

En este trabajo se aborda este problema combinando el potencial que ofrece BERT, uno de los principales modelos de procesamiento del lenguaje natural, junto al uso de algoritmos de aprendizaje supervisado.

Para esto, siguiendo la metodología CRISP-DM como base y partiendo de un conjunto de datos que recoge mensajes enviados por jugadores del videojuego ”DoTA 2”, se han entrenado y evaluado, mediante una estrategia de validación cruzada anidada, dos modelos de clasificación basados en redes neuronales y un clasificador basado en aprendizaje máquina con la finalidad de comprobar si se trata de una tarea la cual se pueda resolver mediante estos métodos.

Abstract

The rise of *eSports* has contributed to de increase in players in videogames such as ”Counter-Strike”, ”Valorant”, and ”League of Legends”. While this seems to be positive, as it brings together people from different regions much like a social network, it also carries with it a large amount of toxicity that , for the sake of these communities, must be controlled.

This work aims to address this issue by combining the potential of BERT, one of the leading models in natural language processing, with the use of supervised learning algorithms.

To this end, following the CRISP-DM methodology as a basis and starting from a dataset that collects messages sent by players of the video game ”DoTA 2”, two neural network-based classification models and a machine learning classifier were trained and evaluated using a nested cross-validation strategy to determine whether this task can be effectively addressed using these methods.

Palabras clave:

- eSport
- Toxicidad
- Machine Learning
- BERT
- Aprendizaje Supervisado

Keywords:

- eSport
- Toxicity
- Machine Learning
- BERT
- Supervised Learning

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años, los videojuegos competitivos han aumentado en popularidad, así como lo han hecho en número de jugadores, llegando a cifras nunca antes vistas. Por poner algunos ejemplos, juegos como "Counter-Strike", "Valorant" o, el hasta ahora rey de los eSports, "League of Legends", han llegado a superar los dos millones y medio de jugadores [1].

Estos se caracterizan por enfrentar a varios jugadores entre sí, típicamente en dos equipos de cinco personas cada uno, para lograr cierto objetivo antes que el rival y así conseguir ganar la partida, buscando de esta forma asemejarse a los deportes tradicionales como el fútbol o el baloncesto. Lógicamente el poder ganar o no depende de que los integrantes de un equipo se coordinen y se comuniquen entre ellos, por lo que lo más habitual es que exista un medio de comunicación mediante el cual los jugadores puedan dar información a sus compañeros pero, muchas veces, estas herramientas se usan con otras intenciones. No es raro que las partidas se alarguen y generen situaciones de estrés y tensión, en las que los jugadores pueden perder fácilmente los papeles llevándolos a usar los canales de comunicación para atacar tanto a sus rivales como a sus compañeros de equipo. Estos comportamientos, conocidos como "tóxicos", se han convertido, a día de hoy, en el talón de Aquiles de las empresas desarrolladoras siendo cada vez más frecuentes. Sin ir más lejos, desde el año 2021 se ha visto un incremento de un 8% en el número de informes sobre estas actitudes, volviéndose el principal motivo por el cual los jugadores abandonan este tipo de juegos [2].

Por lo general este tipo de comportamientos se intentan evitar con sistemas que recompensan al usuario cuando muestra conductas positivas, como felicitar a tus compañeros cuando hacen algo bien, desear suerte a tus rivales o, si el juego lo permite, recomendar o dar "honor" a las cuentas de tus aliados, así como sistemas para detectar comportamientos negativos, siendo los más frecuentes los abusos verbales. Para esto último existen diversas estrategias, ya sean diccionarios con palabras restringidas o modelos de inteligencia artificial que, sumados

a un sistema de denuncias que permite a los propios jugadores informar sobre las actitudes de los demás, bloquean los canales de comunicación o, si se cruza cierta línea, inutilizan la cuenta del infractor. Aun con todo esto, las conductas "tóxicas" no hacen más que aumentar, pues ni a las empresas les interesa que estas herramientas sean completamente efectivas, ya que esto implicaría una gran pérdida de jugadores, ni los propios jugadores hacen esfuerzos por eliminarlas.

1.1 Conductas tóxicas y LLM

Los LLM (Large Language Models) son modelos de lenguaje formados por redes neuronales entrenadas con grandes cantidades de texto, lo cual les permite reconocerlo y generarlo. A raíz de estos, nacieron diferentes aproximaciones, como BERT (Bidirectional Encoder Representations from Transformers), herramienta creada por Google con la finalidad de comprender las búsquedas de los usuarios.

Por lo general, el abuso verbal es una de las manifestaciones más comunes de conductas tóxicas llevadas a cabo por los jugadores, por lo que un modelo como BERT se convierte en una solución ideal, ya que permite captar el contexto de una oración siendo capaz de entender el significado de una palabra basándose en las palabras que la rodean.

1.2 Objetivos

El objetivo principal del proyecto es utilizar un LLM para poder clasificar de forma precisa los mensajes enviados durante las partidas de videojuegos competitivos en línea, pudiendo decidir de manera automática la intención de estos, es decir, si son mensajes apropiados o no.

Para conseguirlo se plantean varios subobjetivos:

· Entender el dominio de la tarea.

· Seleccionar un conjunto de datos.

· Comprender el funcionamiento de las técnicas que se van a usar.

· Entrenar y conseguir un modelo que resuelva el problema.

· Evaluar el modelo contra un conjunto de datos de evaluación.

1.3 Estructura del documento

A lo largo de este documento se detallarán las fases de este proyecto. Primero se hará una breve introducción sobre la metodología escogida, CRISP-DM, explicando, de manera resumida, cada una de sus etapas. Luego de esto se profundizará en cada una de ellas, describiendo los procedimientos seguidos y las decisiones tomadas para completarlas. Finalmente, basándose en el desarrollo y los resultados, se expondrán las conclusiones y lecciones aprendidas, así como propuestas para el trabajo futuro.

La estructura será la siguiente:

- **Capítulo 2: Metodología: CRISP-DM**
- **Capítulo 3: Comprensión del Negocio**
- **Capítulo 4: Comprensión de los Datos**
- **Capítulo 5: Preparación de los Datos**
- **Capítulo 6: Modelado**
- **Capítulo 7: Evaluación**
- **Capítulo 8: Conclusiones**

1.4 Código asociado

Todo el código asociado a este proyecto, así como el conjunto de datos utilizado y los resultados obtenidos se encuentran en el siguiente repositorio de GitHub:

- TFG-Clasificacion-Mensajes-Toxicos

Capítulo 2

Metodología: CRISP-DM

Para asegurar una buena organización y estructuración del proyecto, se ha decidido seguir una metodología que se adapte al mismo, permitiendo establecer de forma clara los pasos a seguir para, de esta forma, maximizar las posibilidades de cumplir todos los objetivos establecidos y, a su vez, poder mantener un seguimiento de todos los procesos.

Existen una gran variedad de estrategias orientadas a proyectos de *Data Science*, siendo las más conocidas KDD (Knowledge Discovery in Databases), centrada en la extracción de conocimiento a partir de grandes conjuntos de datos revelando patrones, tendencias y relaciones ocultos en estos, SEMMA (Sample, Explore, Modify, Model, Assess), creada por el *SAS Institute* en 2012 y utilizada principalmente para *Data Mining*, consta de cinco fases mediante las cuales se espera descubrir patrones de negocio desconocidos, y CRISP-DM (Cross Industry Standard Process for Data Mining), que, dividida en seis fases, permite ajustar y mejorar modelos en función de los resultados obtenidos de forma iterativa. Se ha decidido seguir esta última por ser la que mejor se ajusta a este proyecto.

CRISP-DM tiene su origen a mediados de los años noventa [3], saliendo a la luz su primera versión en 1999. Con el paso del tiempo, acabó convirtiéndose en la metodología estándar para proyectos de Minería y Ciencia de Datos, sirviendo también como inspiración para nuevas propuestas como ASUM-DM de IBM o SEMMA, antes mencionada. A pesar de su longevidad, que abarca casi dos décadas, sigue siendo una de las metodologías más utilizadas en la actualidad, pudiendo aplicarse en gran cantidad de proyectos cantidad de proyectos de ciencia de datos contemporáneos.

Su ciclo de vida se divide en seis fases:

- 1-***Comprensión del negocio***: En esta etapa se pretende comprender los objetivos principales del proyecto desde diferentes perspectivas, es decir, tanto desde el punto de vista

del negocio (lo que propone el cliente) como desde el punto de vista de la minería de datos (cómo lograr lo que el cliente propone). Para ello, es necesario evaluar la situación actual y los recursos disponibles pudiendo así alinear los objetivos del negocio con los de minería de datos, estableciendo una planificación acorde al problema a solucionar.

- 2-***Comprensión de los datos***: Esta etapa se centra en dos puntos principales, conocer la estructura y relevancia de los datos, así como la calidad de los mismos. Para ello, una vez se tienen los datos, es necesario averiguar su naturaleza y comprobar la existencia de errores o ruido que pueda haber surgido durante el proceso de recopilación. Finalmente, teniendo en cuenta lo anterior, se debe de decidir si los datos pueden ser usados para abordar el problema.

- 3-***Preparación de los datos***: El objetivo de esta fase es el de obtener los datos finales sobre los que se va a trabajar. Para ello se deben seleccionar, limpiar y, si fuera posible, construir nuevas características para adaptarlos así a los diferentes modelos de Machine Learning.

- 4-***Modelado***: Esta fase tiene como objetivo construir uno o varios modelos que permitan alcanzar los objetivos del proyecto. Se deben seleccionar las técnicas que se van a aplicar y optimizar sus parámetros para proporcionar los mejores resultados posibles. Finalmente, se debe utilizar un conjunto de entrenamiento para estimar cuáles de estos modelos se comportan mejor, en función del problema que queremos solucionar.

- 5-***Evaluación***: Una vez entrenados los diferentes modelos, se evalúan sus resultados para comprobar el grado de ajuste a los objetivos de negocio utilizando conjuntos de prueba, es decir, muestras nunca antes vistas, con el fin de definir los beneficios e inconvenientes de cada uno en contraposición a los demás, basándose en diferentes métricas.

- 6-***Despliegue***: Es la fase final de la metodología. Consiste en planificar la implementación del modelo teniendo en cuenta las condiciones dadas por escenarios reales, monitorizando su ejecución. Debido a la dificultad de acceder a un escenario real, se ha decidido prescindir de esta última etapa.

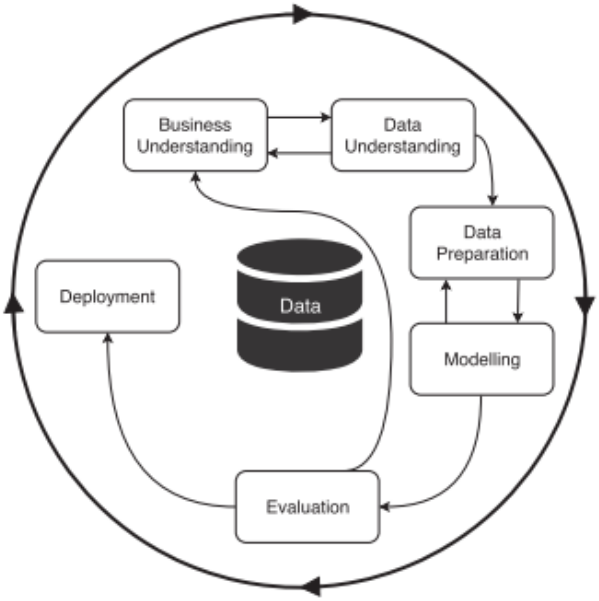

Figura 2.1: Ciclo de vida CRISP-DM.

Tal y como se puede apreciar en la figura 2.1, se trata de una metodología ágil, lo que permite volver sobre todas sus fases antes de llegar a la última de ellas. Esto facilita realizar varias iteraciones para comprender verdaderamente el objetivo a seguir y, de esta forma, obtener mejores resultados.

A lo largo de los apartados de esta memoria, se explicarán en detalle los pasos seguidos en cada una de estas fases.

Capítulo 3

Comprensión del Negocio

La fase de Comprensión del Negocio se centra principalmente en entender los objetivos del proyecto desde ambas perspectivas, es decir, desde el punto de vista del cliente y desde el punto de vista de la minería de datos, definiendo, a mayores, las tecnologías necesarias para alcanzarlos así como una planificación inicial, lo que ayudaría a evitar posibles contratiempos, favoreciendo el éxito del proyecto.

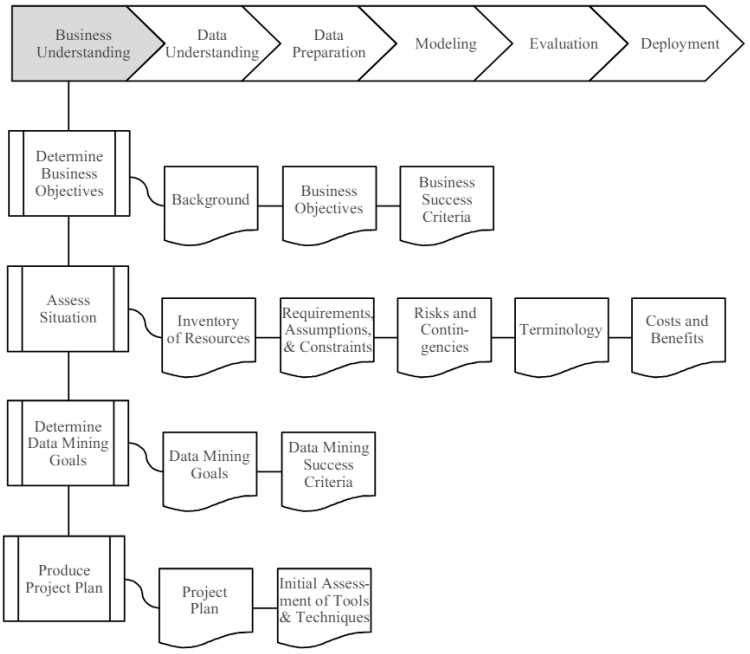

Figura 3.1: Comprensión del Negocio.

Para lograr esto se plantean cuatro tareas, las cuales se pueden apreciar en la figura 3.1 [4] y sobre las que se centrará este capítulo:

- ***Determinar los objetivos de negocio***:
  Se pretende determinar de manera precisa lo que el cliente quiere conseguir.
- ***Evaluar la situación***:
  Se pretenden analizar los recursos disponibles, las necesidades y los riesgos del proyecto, así como crear planes de contingencia, si fuera necesario.
- ***Determinar los objetivos de la minería de datos***:
  Se pretenden definir los objetivos del proyecto desde una perspectiva técnica.
- ***Elaborar un plan de proyecto***:
  Se pretende seleccionar las tecnologías y herramientas que se van a emplear, así como generar una planificación acorde a cada fase del proyecto.

3.1 Objetivos del negocio

Esta tarea tiene como finalidad entender el propósito real del proyecto. Para lograrlo, como se muestra en la figura 3.1, se deben de cumplir tres requisitos. En este caso, tanto los objetivos como el criterio de éxito ya han sido definidos previamente por lo que, a raíz de estos, debemos de deducir el trasfondo, es decir, conocer los datos y herramientas disponibles, así como comprender el funcionamiento de las mismas para, de esta forma, poder llevar a cabo el proyecto.

Ya que la meta de este trabajo es la clasificación de texto, es necesario tener claro cómo se procesa el lenguaje natural, para así contar con mayor conocimiento a la hora de entender nuestro conjunto de datos y el funcionamiento de los Large Language Models (LLM), centrándose concretamente en Bidirectional Encoder Representation from Transformer (BERT) al ser este el escogido para realizar el proyecto.

3.1.1 Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural (PNL) es una de las muchas áreas de la Inteligencia Artificial, centrada principalmente en construir modelos capaces de interpretar, manipular y comprender el lenguaje humano [5]. Estos modelos tienen como finalidad resolver una serie de tareas [6], entre las cuales se destacan:

- Traducción automática.
- Reconocimiento de comandos, tanto escritos como expresados de forma oral.
- Resumir grandes volúmenes de texto.

- Analizar la intención de un texto, es decir, si tiene una connotación positiva o negativa.
- Clasificar y extraer texto.

Como funciona el PNL: Text Embeddings

Para que una máquina sea capaz de procesar el lenguaje humano, primero es necesario traducirlo, es decir, transformar la información a un formato con el que sean capaces de trabajar. A lo largo de los años se han desarrollado diferentes técnicas, las cuales han evolucionado hasta llegar a los ”embeddings” [7].

- ***Bag fo Words***

  La técnica *Bag of Words* es una de las aproximaciones más simples para convertir el texto en vectores. Para ello, primero se deben de separar las diferentes palabras de una frase u oración y reducirlas a su forma raíz, por ejemplo, pasando los verbos conjugados a infinitivo. Una vez hecho esto, se calcula la frecuencia de cada una de estas palabras o *tokens*, generando así un vector con el que se podría trabajar. Sin embargo, el resultado es simple por lo que textos que a priori deberían de tener cierta relación semántica pueden ser codificados con valores bastante dispares.

- ***TF-IDF***

  TF-IDF (*Term Frequency — Inverse Document Frequency*) es otra técnica para codificar vectores de palabras [7]. En este caso se deben tener en cuenta tanto la frecuencia del término en el documento como la frecuencia inversa del término en todo el corpus[1]:

  $$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D) \tag{3.1}$$

  La frecuencia de un término (TF) se calcula dividiendo el número de veces que aparece en el documento, como se plantea en *Bag of Words*, entre el número de total de términos del mismo:

  $$TF(t, d) = \frac{\text{número de veces que un término t aparece en el documento d}}{\text{número de términos del documento d}} \tag{3.2}$$

  El inverso de la frecuencia de un término (IDF) sirve para cuantificar la información que ofrece una palabra. Se calcula como el logaritmo de la proporción entre el número

[1] Conjunto grande y estructurado de textos o datos lingüísticos

total de documentos y el número de documentos que contienen el término concreto:

$$IDF(t, D) = \log\left(\frac{\text{número total de documentos presentes en el corpus D}}{\text{número de documentos que contienen el termino t}}\right) \quad (3.3)$$

De esta forma, las palabras más comunes, como los adverbios o las conjunciones, tendrán menos relevancia que las palabras que son menos frecuentes, que en principio contienen más información sobre la temática del texto. Esto mejora la aproximación anterior, pero sin ser capaz aún de capturar su significado semántico y, a mayores de esto, genera vectores de gran tamaño y con pesos muy dispares.

- ***Word2Vec***

Word2Vec es una aproximación propuesta por *Google* que busca tanto predecir una palabra basándose en las que la rodean, *Continuous Bag of Words*, como predecir términos pueden estar relacionados con una palabra dada, *Skip-grams*. Esto, como se puede apreciar en la figura 3.2 [8], permite crear tanto un codificador, usando *Skip-gram*, como un decodificador, usando *Continuous Bag of Words*.

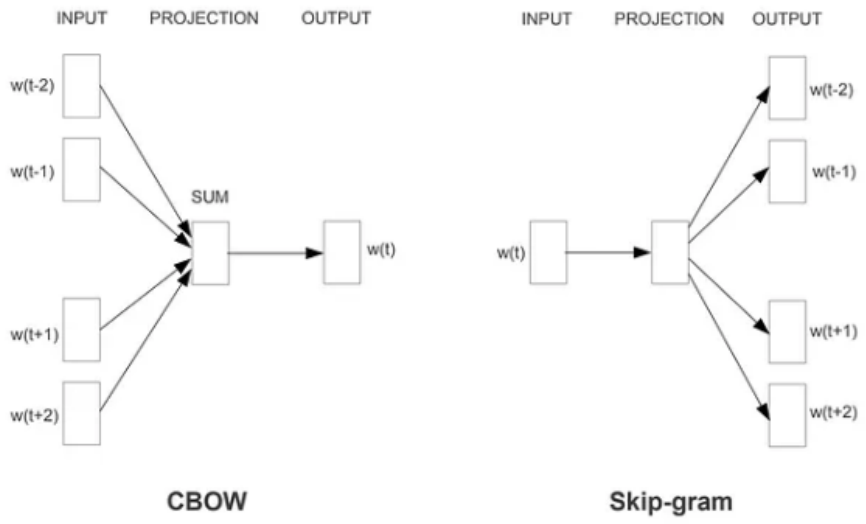

Figura 3.2: Arquitectura de Continuous Bag of Words y Skip-gram.

- ***Transformers and Sentence Embeddings***

Con la introducción de los *Transformers* surgió la posibilidad de generar vectores que, a mayores de almacenar información de palabras o frases también capturan el significado de estas, lo que dio lugar a los *Text Embeddings*. Los *Transformers* son modelos de inteligencia artificial, basados en redes neuronales, capaces de cambiar o transformar una secuencia de entrada en una de salida [9], pudiendo entender como se relacionan entre sí las diferentes partes de la entrada, captando el contexto de la misma y cuan importante es cada componente. Aún partiendo de representaciones vectoriales calculadas de manera similar a lo explicado antes, estos modelos cuentan con cabezas de autoatención. Cada una de estas "cabezas", mediante el uso de mecanismos de autoatención[2], genera, en paralelo con las demás, una representación diferente al vector inicial las cuales se combinarán con este consiguiendo que recoja mucha más información. Finalmente estos embeddings se pasan a una red neuronal feedforward[3] donde se refinan, dando como resultado un vector contextualizado.

3.1.2 Large Language Models

Un Large Language Model (LLM) es un modelo de inteligencia artificial basado en la arquitectura Transformer y entrenado con grandes cantidades de texto, capaz de resolver un amplio abanico de tareas. A diferencia de otros modelos que están diseñados exclusivamente para tareas concretas, los LLMs han revolucionando el campo del NLP [11]. Uno de los aspectos más destacables de los LLMs es su capacidad para ajustarse usando las propias respuestas de los usuarios como retroalimentación, aprendiendo así de sus errores. Esta capacidad de generalización ha hecho que su popularidad aumente, volviéndose una de las herramientas más usadas para tareas tanto de generación y comprensión de texto, como de clasificación o traducción de textos [12]. Actualmente, existen una gran variedad de modelos basados en LLM entre los cuales destacan GPT-3 o GPT-4 de *OpenAI* y el propio BERT de *Google*.

3.1.3 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers (BERT) es una técnica propuesta por Google en 2018 [13] cuyo objetivo original era comprender las búsquedas de los usuarios. A diferencia de otras herramientas de procesamiento de lenguaje natural, BERT permite captar el contexto de una oración, entendiendo el significado en función de las palabras que la rodean en ambas direcciones, fusionando el contexto de ambas. Esto es posible ya que está preentrenado usando *masked language models* (MLM) [14]. Un MLM es un modelo de lenguaje

[2] Un mecanismo de autoatención permite a los modelos centrarse en diferentes partes de la entrada, calculando la importancia de cada posición del la frase original [10].

[3] Red neuronal sin ciclos en el flujo de datos

el cual, de forma aleatoria, oculta algunas de las palabras de la entrada, forzando a predecir el vocabulario original basándose únicamente en el contexto.

Arquitectura

Existen dos modelos basados en una arquitectura Tansformer multicapa como codificador. $\text{BERT}_{\text{base}}$ cuenta con 12 capas, un tamaño de capa oculta de 768, un total de 12 cabezas de autoatención y 110 millones de parámetros, mientras que su hermano mayor, $\text{BERT}_{\text{large}}$ tiene 24 capas, un tamaño de capa oculta de 1024, 16 cabezas de autoatención y 345 millones de parámetros. A esto, como se puede apreciar en la figura 3.3 [15], se le suma una capa para clasificar las salidas previamente codificadas y una capa *softmax*[4] para, finalmente, calcular la probabilidad de cada palabra.

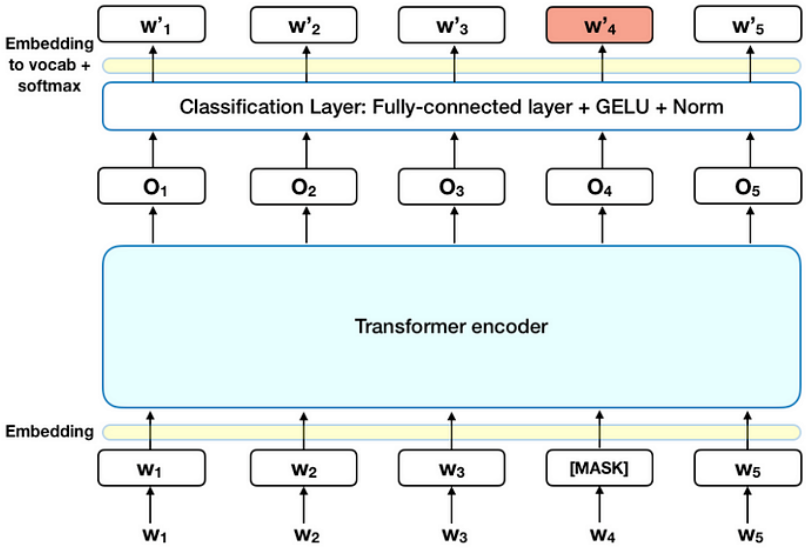

Figura 3.3: Arquitectura BERT.

3.2 Evaluación de la situación

La popularidad de los deportes electrónicos está en constante aumento, al igual que el número de jugadores, por lo que es necesario, tanto para las comunidades de estos juegos como para las propias empresas desarrolladoras, el conseguir herramientas que permitan controlar y penalizar, si fuera necesario, las conductas negativas.

[4] Una capa softmax transforma los resultados numéricos de una red neuronal en la probabilidad de una muestra a pertenecer a una clase

Por esta razón, existen varias fuentes de datos de uso libre, tanto etiquetadas como no, recogidas principalmente por los propios usuarios de diversas formas, siendo la más común el crawling en páginas como OpenDota las cuales, por desgracia, ya no están operativas por lo que los datos, a pesar de ser suficientes, son relativamente antiguos. En este caso, se han considerado diferentes conjuntos:

- GOSU: Dataset utilizado para entrenar un chatbot capaz de comunicarse de manera similar a lo que lo haría un jugador de *Dota*. Cuenta con un total de 20.663.261 mensajes escritos por los usuarios del mismo juego. Se trata de un conjunto de datos muy extenso pero tiene dos pegas, mezcla idiomas, en concreto el inglés (lenguaje general que se usa para comunicarse en los videojuegos) y el ruso (debido a la popularidad del juego en las regiones de habla rusa), y no está etiquetado. Dado que se pretende entrenar el modelo mediante aprendizaje supervisado y, por simplicidad, se usará el inglés como idioma principal, se ha descartado.

- GOSU AI: Se trata de una recopilación de mensajes únicamente en inglés del dataset anterior. Cuenta con un total de 3.268 mensajes, los cuales están separados en tres clases, tóxicos, algo tóxicos y no tóxicos. El principal problema de este conjunto, a mayores de que es relativamente pequeño, es que las etiquetas se asignaron manualmente, lo que puede afectar a su fiabilidad.

- League of Legends Tribunal Chatlogs: Conjunto de mensajes sacados de *LoL Tribunal*, herramienta del videojuego "League of Legends" que permitía a los propios jugadores juzgar si otro, el cual había sido reportado, merecía o no ser penalizado por su comportamiento. Este dataset contiene un total de 1.697.221 mensajes de texto en diferentes idiomas, ya que recoge mensajes de todos los servidores. No está etiquetado por lo que se ha descartado.

- Dota 2 Pro League Matches: Este dataset recoge información de partidas profesionales del videojuego *Dota 2* jugadas entre los años 2016 y 2024. Incluye mensajes enviados durante estas partidas. El conjunto es demasiado grande, cuenta con muchos campos que no son relevantes para el trabajo y no está etiquetado, por lo que se ha descartado.

- Dota 2 Matches: De la misma forma que el dataset anterior, recopila información de 50.000 partidas del juego *Dota 2* pero en este caso, de jugadores casuales. Contiene muchos datos innecesarios para el proyecto y no está etiquetado, por lo que se ha descartado.

- PUBG Simulated Chat Messages: Se trata de un conjunto de mensajes simulados basados en partidas del videojuego *PUBG: Battlegrounds*. Consta de 5.200 mensajes, clasificados

en optimistas y pesimistas, en función del resultado de las partidas, es decir, basándose en si el jugador ha ganado o si ha perdido. Al tratarse de mensajes simulados se ha descartado.

- Hopeful vs. Hopeless: Labeled Dota 2 Player Messages Dataset: Este dataset contiene mensajes anotados de partidas del juego *Dota 2*. Cuenta con un total de 8.000 mensajes, escritos principalmente en inglés clasificados en dos categorías, positivos y negativos, basándose en la actitud del jugador que lo escribe y el resultado de las partidas. Por estos motivos se ha decidido que será el dataset con el que se entrenara el modelo. Se profundizará sobre este en el siguiente capítulo.

3.2.1 Tecnologías escogidas

Para la realización del proyecto se ha decidido usar, basándose en Anaconda para poder construir el entorno de trabajo de manera óptima, una distribución libre de Python, más concretamente la versión 3.9.18. Esto se ha escogido así ya que, a mayores de ser uno de los lenguajes más utilizados en el ámbito de la Inteligencia Artificial y Machine Learning, se trata de un lenguaje simple, con un buen rendimiento y facilita el acceso y uso de las librerías necesarias para poder resolver el problema planteado.

Para generar el modelo deseado, como ya se ha explicado anteriormente, se ha planteado el uso de BERT, al cual se accederá desde las librerías ofrecidas por TensorFolow , en concreto, serán necesarias *tensorflow*, así como *tensorflow_hub* y *tensorflow_text*. A mayores de estas, será necesario usar librerías básicas del lenguaje como *pandas*, *numpy* y *Scikit-learn* para poder trabajar correctamente con los datasets y construir los modelos clasificadores.

3.3 Objetivos de la minería de datos

Desde el punto de vista de la minería de datos hay un objetivo claro, conseguir un modelo capaz de decidir si un texto tiene una intención negativa o positiva, siguiendo la metodología CRISP-DM hasta la penúltima de sus fases, la evaluación.

3.4 Planificación del proyecto

Ya que la metodología escogida es CRISP-DM, se debe de generar un plan de proyecto acorde a la misma. En este caso se ha decidido simplificar la estrategia, obviando la fase de despliegue, como ya se había planteado en el capítulo anterior.

Este proyecto será llevado a cabo por tres integrantes, Diego Fernández Iglesias y Fidel Cacheda Seijo, tutores del mismo, los cuales tomarán el rol de supervisores, y Nicolás Rivas Rodriguez, autor del trabajo, el cual asumirá el rol de analista y desarrollador.

3.4.1 Comienzo

El primer paso fue organizar una reunión, en la que se concretaron los aspectos principales del mismo, es decir, el problema a resolver, los conocimientos sobre el campo del *Machine Learning* y el *NLP*, la metodología a usar y los conjuntos de datos accesibles, así como las tecnologías disponibles y los objetivos.

Durante las tres siguientes semanas, se realizó un estudio del estado del arte para familiarizarse con el dominio especifico.

3.4.2 Planificación

Para asegurar la claridad del proyecto, se ha decidido usar la misma estructura y nomenclatura que ofrece la metodología para establecer las diferentes etapas. De esta forma, se pretende generar un plan sencillo de entender y de seguir, por lo que, en caso de que surgieran dificultades, no supondría un mayor problema el reubicarse en el mismo. A mayores de las etapas antes mencionadas, también se escribirá una memoria a modo de documentación, la cual se irá cubriendo de forma paralela para asegurar así su calidad. Con todo esto, se construyó un diagrama de Gantt, el cual se puede ver en la figura 3.4, que recoge la planificación inicial.

Figura 3.4: Diagrama de Gantt.

3.4.3 Costes

En esta clase de proyectos existen dos tipos de costes, los asociados a los recursos humanos, como pueden ser los salarios, y los asociados a los recursos materiales.

Para el primer caso, se cuenta con tres integrantes, los cuales, según se estipula en el BOE [16], se dividen de la siguiente forma:

- Dos integrantes que pertenecen al grupo A del área 3, para los que, desde el 1 de Enero de 2024, se estipula un sueldo total anual de 28.850,23. Siendo la jornada laboral anual de 1.800 horas, el coste se traduciría en unos 16,03€/hora.
- Un integrante, el cual pertenece al grupo B 2 del área 3 para el que, desde el 1 de Enero de 2024, se estipula un sueldo total anual de 27.147,12. Siendo la jornada laboral anual de 1.800 horas, el coste se traduciría en unos 15,08€/hora.

Teniendo en cuenta esto, junto a que se estima que el alumno dedique unas 304 horas para la realización del proyecto y que los tutores de este, entre el tiempo dedicado a correcciones y reuniones, dediquen al rededor de 20 horas, se prevé un coste total asociado a recursos humanos de unos 5.225,52€.

Los costes materiales se reducen al equipo necesario para realizarlo. Concretamente, se dispone de un ordenador valorado en 856,24€.Teniendo en cuenta que estos equipos se suelen amortizar en 4 años, y que la duración estimada del proyecto es de 2 meses, el coste real en este caso sería:

$$\frac{856,24}{48} \times 2 = 35,68€ \tag{3.4}$$

Poniendo todo en conjunto, se estima un coste de 5.225,52€ + 35,68€ = 5.261,19€.

Cabe destacar que todo el software y librerías utilizadas son de uso libre, por lo que no suponen costes adicionales y, al tratarse de un proyecto académico, no hay que tener en cuenta gastos indirectos como pueden ser el alquiler de oficinas, internet, luz o agua.

Capítulo 4

Comprensión de los Datos

El primer paso para conseguir un buen modelo, con el cual alcanzar los objetivos del proyecto, es partir de un conjunto de datos completo, preciso y de calidad, evitando así posibles contratiempos en las fases subsecuentes. Para esto, se deben analizar en profundidad tanto las diferentes alternativas existentes para obtener estos datasets, así como los rasgos de los mismos, asegurándose de que no presentan errores críticos y que se ajustan al problema que se espera resolver.

A lo largo de este capítulo se abordarán las diferentes fases, mostradas en la figura 4.1 [4], con la finalidad de realizar un análisis exhaustivo del conjunto de datos escogido.

4.1 Recolección de los datos

La tarea de recolección de datos relacionados con videojuegos no es sencilla, ya que las empresas desarrolladoras suelen mantener esta información en privado. A esto, hay que sumarle que, a día de hoy, no existen muchas herramientas que permitan extraer de forma rápida y eficaz los mensajes enviados por los jugadores en sus partidas, al tratase de información privada y personal, siendo la única forma de conseguirlos, si los desarrolladores lo permiten, solicitar la información de una cuenta que, como es lógico, debe ser propiedad del que realiza la solicitud. Aún existiendo esta posibilidad, la cual es lenta y depende de quien la realiza, al no contar con el tiempo y recursos suficientes, la tarea de etiquetado sería muy pesada y estaría sesgada.

Por esto se ha decidido buscar conjuntos de datos ya etiquetados recogidos en páginas como **Kagle** o **Harvard Dataverse**.

Durante este proceso se valoraron varias posibilidades, las cuales se pueden ver en el capítulo 3, siendo Hopeful vs. Hopeless: Labeled Dota 2 Player Messages Dataset la escogida

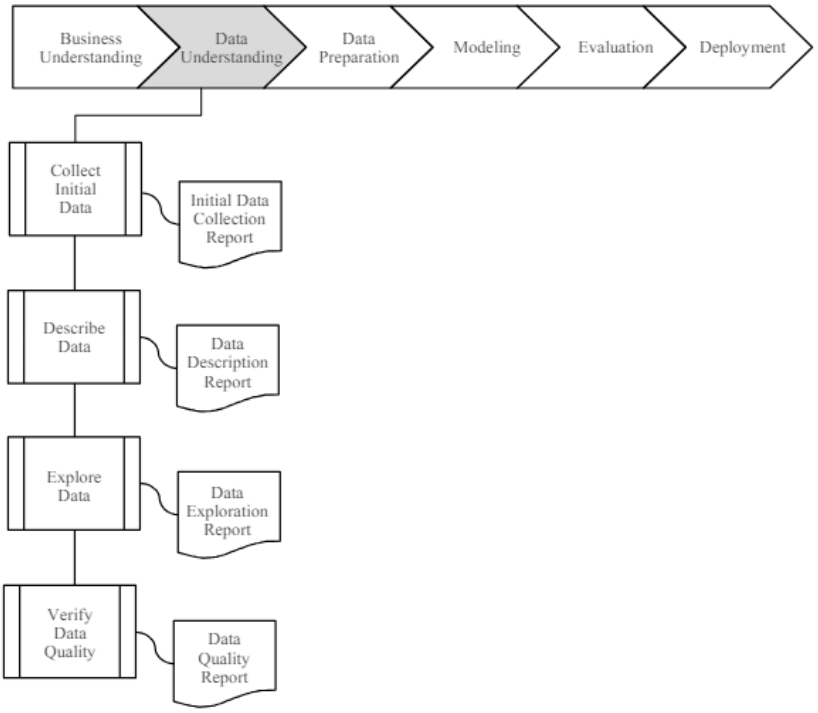

Figura 4.1: Comprensión de los Datos.

para resolver el problema propuesto. Se trata de una recopilación de mensajes, anotados directamente de partidas del videojuego *Dota 2* y etiquetados basándose en la intencionalidad del mensaje y el resultado final de la partida en la que fue enviado.

4.2 Descripción y exploración de los datos

El dataset escogido cuenta con un total de 8.000 muestras. Cada una de ellas se corresponde con un mensaje, escrito en inglés, enviado por algún jugador durante una partida. El etiquetado divide los datos en dos clases:

- Comentarios negativos: Mensajes con una intención negativa o pesimista. Un ejemplo de estos es ”yea just end this is a shit team”.
- Comentarios positivos: Mensajes con una intención positiva o optimistas. Un ejemplo de estos es ”gg thanks for the fun lane mid BH”.

Estas están distribuidas en una proporción uno a uno, habiendo 4.000 muestras etiquetadas

como negativas y 4.000 etiquetadas como positivas. Esto se puede observar en la figura 4.2.

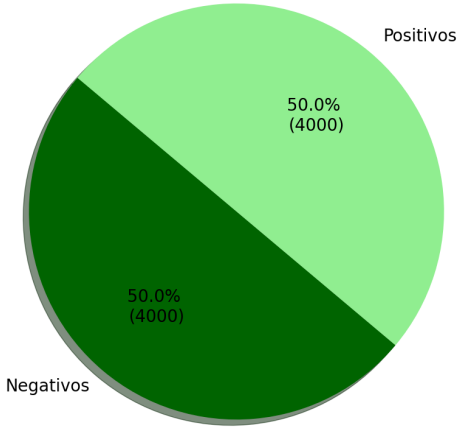

Figura 4.2: Distribución de las clases.

En lo que respecta a los mensajes, podemos destacar las siguientes características:

- **Longitud**: Una longitud media de 21,05 caracteres, lo que se traduce en unas 4,5 palabras cada uno. El más largo cuenta con 14 palabras mientras que el más corto tiene únicamente 1. En la figura 4.3 se puede ver la distribución de la longitud de los mensajes.

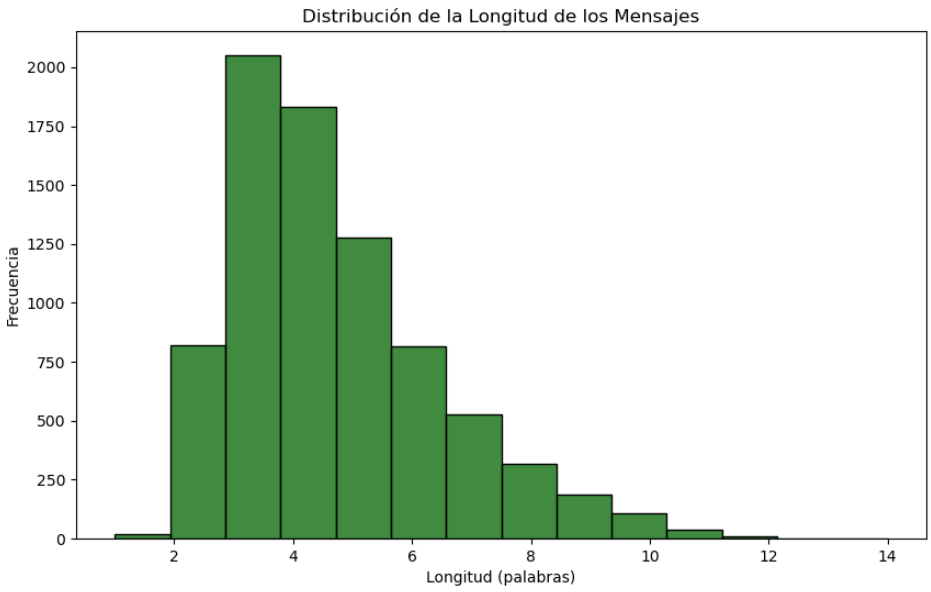

Figura 4.3: Distribución de la longitud de los mensajes.

- **Palabras más comunes**: Como era de esperar, al tratarse de mensajes directos, las

palabras más comunes son pronombres, siendo el más frecuente "i". En la figura 4.4 se pueden ver las 30 más recurrentes en el dataset.

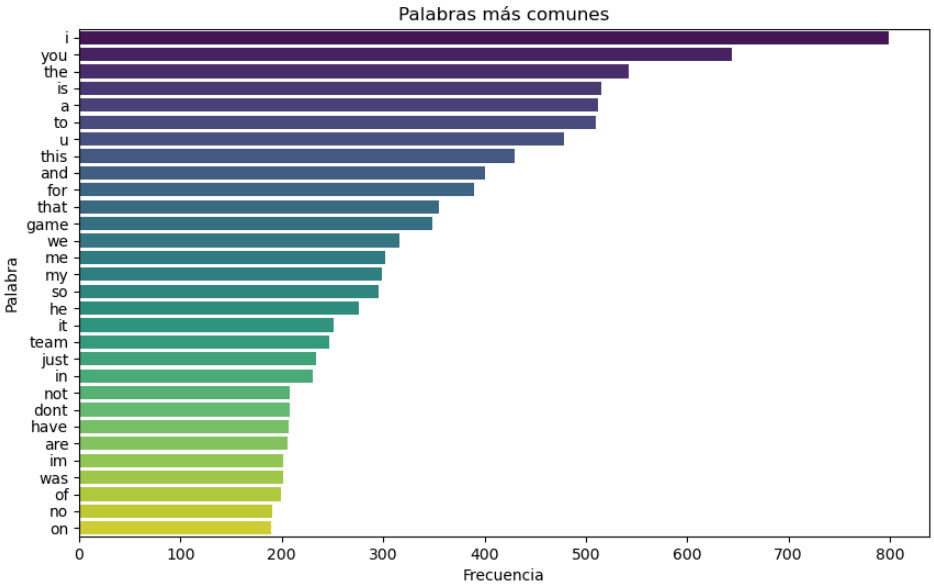

Figura 4.4: Top 30 palabras más comunes.

- **Palabras Clave**: Teniendo en cuenta el objetivo del proyecto, es importante conocer las palabras clave en el dataset escogido. Para obtenerlas se calculó el *Tf-idf* de cada uno de los términos presentes en el conjunto de datos. Obviando las palabras vacías, como los artículos o los pronombres, se pueden destacar las siguientes palabras:

Clase	Palabra	Frecuencia	Tf-idf
Negativo	game	232	0.019156
Negativo	fucking	154	0.015469
Negativo	bad	126	0.013488
Negativo	shit	124	0.012383
Positivo	nice	171	0.018307
Positivo	team	123	0.011706
Positivo	report	119	0.013461
Positivo	win	103	0.011093

Tabla 4.1: Palabras Clave.

- **Diversidad léxica**: Al tratarse de textos cortos, es lógico que no cuenten con una gran diversidad léxica. Aún así, se trata de una característica relevante de un texto, por lo que se han calculado tanto el índice Tipo-Token[1] y la Diversidad Léxica de cada una de las clases del conjunto de datos.

Clase	Índice Tipo-Token	Diversidad Léxica
Negativos	0.995293	2.136996
Positivos	0.991706	1.931139

Tabla 4.2: Diversidad Léxica.

En este caso, es suficiente con que el dataset inicial cuente únicamente con el texto y la clase a la que pertenece, ya que, antes de avanzar a las siguientes fases, se usará BERT para extraer un datapack con las características de cada una de las muestras.

4.3 Verificación de la calidad

A la hora de verificar la calidad de un conjunto de datos deben tenerse en cuenta posibles anomalías, la existencia de datos nulos y los posibles errores de medición.

En este caso al estar compuesta cada muestra únicamente por un texto y su clase, no deberían existir valores nulos. Aún así se hizo una comprobación, usando la librería *Pandas* para asegurarse a ciencia cierta de que no existieran estos valores la cual, como era esperable, confirmó esta hipótesis.

En lo que respecta a las anomalías y a los errores de medición, se presupone que no deberían de existir ya que, como se ha mencionado, no se trata de un dataset que recoja características, sino que, usando BERT, se extraerán para generar un datapack con el que entrenar y evaluar los modelos.

4.4 Creación del datapack

Para poder entrenar y evaluar el modelo se debe de generar un datapack que recoja las características presentes en el texto. Esto se consigue mediante un procesado de los mensajes,

[1] Relación entre el número de palabras únicas y el número total de tokens

generando el embedding[2] correspondiente a cada uno de ellos.

En este caso BERT, al ser un modelo preentrenado, permite procesar los datos, convirtiéndolos a un formato el cual pueda ser utilizado.

El primer paso es escoger uno de los modelos y, con este, preprocesar el texto. Para esto existen varias posibilidades, entre ellas se pueden destacar:

- **bert_en_uncased_L-12_H-768_A-12**: Modelo basado en la arquitectura $\text{BERT}_{\text{base}}$ que no distingue entre las mayúsculas y minúsculas.
- **bert_en_cased_L-12_H-768_A-12**: Modelo basado en la arquitectura $\text{BERT}_{\text{base}}$ que distingue entre las mayúsculas y minúsculas.
- **bert_en_uncased_L-24_H-1024_A-16**:Modelo basado en la arquitectura $\text{BERT}_{\text{large}}$ que no distingue entre las mayúsculas y minúsculas.
- **bert_en_cased_L-24_H-1024_A-16**: Modelo basado en la arquitectura $\text{BERT}_{\text{large}}$ que distingue entre las mayúsculas y minúsculas.

De entre todas las opciones, se ha escogido *bert_en_cased_L-12_H-768_A-12* por varios motivos. Como se explica anteriormente, se trata de un modelo capaz de diferenciar las mayúsculas y las minúsculas lo que, teniendo en cuenta que un mensaje escrito completamente en mayúsculas puede tener un significado diferente que uno que no lo está, puede ser relevante. Otro factor importante es que se basa en la arquitectura $\text{BERT}_{\text{base}}$, adaptándose mejor a los recursos disponibles, y, como se explica en la documentación [17], al no contar con parámetros entrenables puede usarse sin problema como extractor de características.

Para conseguir las representaciones vectoriales del texto se deben realizar dos procesos. Con el primero, se generan tres campos que servirán como entradas para extraer los atributos con el modelo seleccionado, estos son:

- **Words Ids**: Identificador único de cada palabra, recogido del propio diccionario de BERT.
- **Input Masks**: Máscaras que indican qué palabras pertenecen al texto y cuáles al relleno que se genera en caso de que la longitud de este no sea suficiente.
- **Input Type Ids**: Indican a qué parte del texto pertenece una palabra.

[2] Representación vectorial de una palabra o un conjunto de palabras

Originalmente los modelos de BERT están pensados para procesar grandes cantidades de texto, generando vectores de 128 tokens. Esto, al estar trabajando con frases cortas, añadiría mucha información innecesaria y redundante, por lo que antes de preprocesar los datos se debe ajustar el tamaño de estas estructuras. Teniendo en cuenta que la frase más larga tiene un total de 14 palabras, se ha establecido un tamaño de 34 tokens por vector, asegurando así pequeño margen.

Un ejemplo de cómo quedaría un mensaje después de este proceso sería el siguiente:

- Mensaje: ”gg thanks for the fun lane mid BH”
- Word Ids: [101 1043 2290 4283 2005 1996 4569 4644 3054 1038 2232 102 1043 2290 4283 2005 1996 4569 4644 3054 1038 2232 102 0 0 0 0 0 0 0 0 0 0]
- Input Mask: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]
- Type Ids: [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0]

Una vez hecho esto se deben traducir estas entradas de modo que puedan usarse para entrenar. Como resultado se obtendrían dos salidas:

- **Pooled Output**: Vector que representa todo el conjunto de texto.
- **Secuence Output**: Matriz que representa el contexto de cada uno de los tokens en el texto.

Ya que se va a realizar una tarea de clasificación, se ha decidido utilizar los pooled outputs al corresponderse directamente con el embedding del texto, resultando en un datapcak con 8.000 muestras, en concreto 4.000 de cada clase, y un total de 768 características cada una.

4.4.1 Correlaciones lineales 2 a 2

Con el datapack generado, es importante conocer las relaciones entre las diferentes características. Para esto se ha calculado la matriz de correlaciones lineales.

Una matriz de correlaciones recoge un valor el cual representa el coeficiente de correlación para cada par de atributos del conjunto de datos. Este coeficiente toma valores entre 1 y -1 y permiten extraer una serie de conclusiones:

- Los valores cercanos a 1 representan que dos atributos están altamente relacionados, es decir, cuando una de las dos variables aumenta, la otra también lo hace.
- Los valores cercanos a 0 indican una correlación neutral, es decir ninguna de las dos variables están relacionadas entre sí.

- Los valores cercanos a -1 muestran una alta correlación negativa, es decir a medida que una variable aumenta la otra disminuye.
- Los valores iguales a 1 representan la diagonal de la matriz, es decir la relación entre una variable consigo misma.

En este caso concreto, ya que el datapack cuenta con un total de 768 características, se ha calculado, por simplicidad, la cantidad de pares de atributos cuyo coeficiente de correlación es superior a 0.9 o inferior a -0.9 y se obtuvieron los siguientes resultados:

Umbral	Total
0.9	7544
-0.9	7535

Tabla 4.3: Número de pares altamente relacionados.

También se comprobó la existencia de atributos redundantes, es decir, con un coeficiente de 1 y que no pertenecen a la diagonal o con un coeficiente de -1, detectando un total de 6 variables con valor 1 y 8 con valor -1.

4.4.2 Visualización

Con la finalidad de poder visualizar el datapack, se aplicó el algoritmo **t-SNE** (t-distributed stochastic neighbor embedding) el cual permite mostrar en un espacio bidimensional un conjunto de datos de alta dimensionalidad. Para esto se siguen una serie de pasos, inicialmente, se calculan las probabilidades de similitud del espacio original[3], luego, se llevan los puntos originales a un espacio de baja dimensionalidad, en este caso de dos dimensiones, de forma que se minimicen las probabilidades, calculadas anteriormente, entre dos des estos puntos [18].

En la figura 4.5 se puede apreciar la distribución de los embeddings generados.

[3] Dos muestras son similares si, tras calcular su distancia euclídea, son cercanas la una a la otra

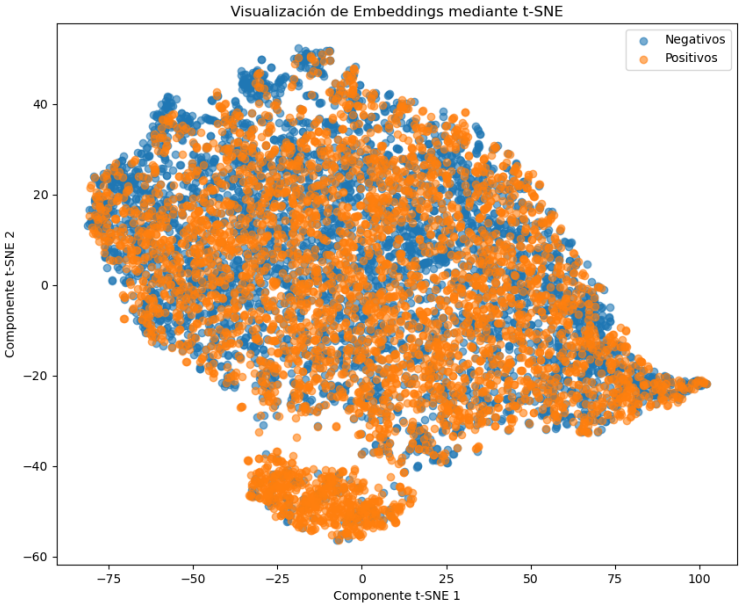

Figura 4.5: Visualización basada en t-SNE.

Capítulo 5

Preparación de los Datos

UNA vez generado el datapack se deben de revisar los datos, asegurándose de que cumplen unos estándares mínimos para, posteriormente, seleccionar las características que se van a utilizar en la fase de modelado.

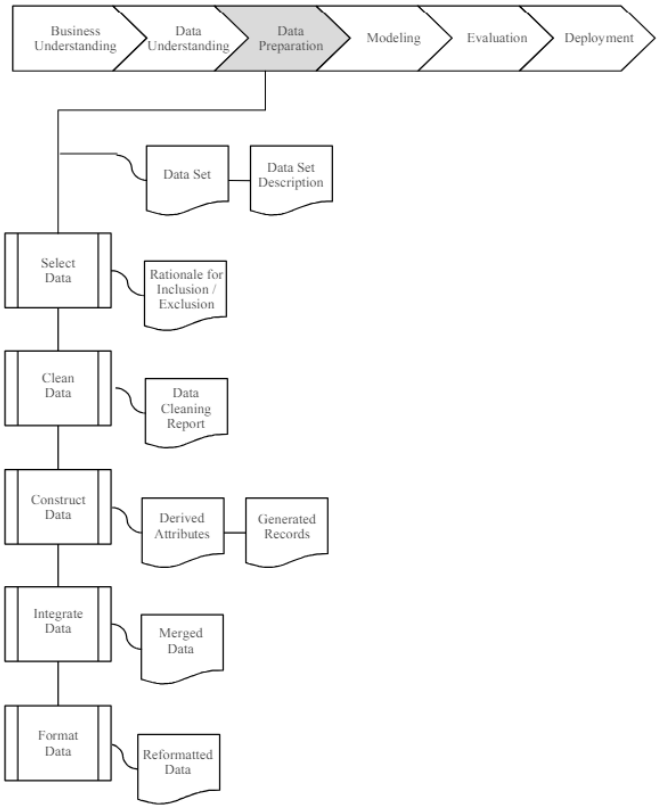

Figura 5.1: Preparación de los Datos.

Para conseguir esto se plantean una serie de tareas, que se pueden apreciar en la figura 5.1 [4], las cuales se abordarán durante este capítulo.

5.1 Limpieza de los datos

El primer paso es asegurarse de que los datos no presentan anomalías. En el capítulo anterior, se demostró que no existían datos nulos, por lo que este apartado se centrará en la detección de irregularidades.

Para realizar esta tarea, existen una gran variedad de técnicas, como pueden ser el uso de variables estadísticas, el cálculo del Rango Intercuartílico (IQR) o el uso de modelos de machine learning como Isolation Forest. Esta última es la escogida para abordar el proceso de limpieza.

5.1.1 Isolation Forest

Isolation Forest es una de las técnicas más populares y eficaces a la hora de detectar anomalías en conjuntos de datos.

Un punto anómalo es, a priori, más fácil de separar del resto de muestras. Aprovechándose de esto, el algoritmo genera particiones en el conjunto de datos a partir de una característica aleatoria del mismo y, en función de un valor de contaminación que representa la proporción esperada de valores atípicos, determina si existen o no muestras irregulares en el dataset.

En este caso, utilizando la clase *IsolationForest* del módulo *ensemble* de la librería *Scikit-learn*, se ha aplicado este algoritmo variando el valor de contaminación. Los resultados fueron los mostrados en la tabla 5.1. Tal y como se ha explicado anteriormente, este valor indica el número de anomalías esperadas en el conjunto de datos y, teniendo en cuenta que se han probado valores desde el 50% hasta el 0.1%, puede afirmarse que no existen muestras atípicas en el conjunto de datos.

Valor de Contaminación	Clase	Número de anomalías
0.5	Negativos	0
0.5	Positivos	0
0.1	Negativos	0
0.1	Positivos	0
0.01	Negativos	0
0.01	Positivos	0
0.001	Negativos	0
0.001	Positivos	0

Tabla 5.1: Resultados de Isolation Forest.

5.2 Construcción de datos

El siguiente paso, una vez eliminadas todas las anomalías, es la creación de nuevas características, a raíz de las ya existentes, para facilitar la comprensión de las mismas. En este caso concreto, BERT recoge en su totalidad las características presentes en el texto, por lo que el datapack generado en el capítulo anterior 4 es más que suficiente.

5.3 Formateo de los datos

Otro de los puntos importantes para poder avanzar a la siguiente etapa es asegurarse de que los datos estén presentados en un formato que pueda ser procesado por nuestro modelo. Por lo general se suelen usar dos técnicas para conseguir esto:

- ***Normalización***: La normalización pretende escalar los atributos para que todos compartan un rango común, reduciendo el impacto que pueda causar la diferencia en la escalas de los mismos y consiguiendo que todas las características tengan una influencia similar en el modelo.
- **Estandarización**: La estandarización pretende tipificar la distribución que sigue un conjunto de valores, es decir, conseguir que su media sea 0 y su desviación típica 1, restando a cada uno de los valores la media del conjunto.

Debido a las ventajas que ofrece la estandarización para una posterior reducción de la dimensionalidad, se ha decidido escoger este método sobre la normalización.

5.4 Selección de los datos

Una vez preprocesados los datos, se deben seleccionar aquellos con los que se va a trabajar. Esta tarea puede diferenciarse en dos subtareas, selección de muestras y selección de características.

5.4.1 Selección de muestras

Teniendo en cuenta la naturaleza de los datos, así como la inexistencia de nulos o errores, por lo explicado en el capítulo anterior, se ha decidido utilizar todas las filas del dataset.

A mayores de esto, también se trata de un dataset equilibrado, contando con el mismo número de muestras para cada una de las clases, por lo que se simplifica la tarea de selección de los conjuntos de entrenamiento, validación y prueba.

Para esto se plantea realizar dos iteraciones de validación cruzada, mediante las cuales separar, en primera, los conjuntos de entrenamiento y prueba, y, posteriormente, dividir cada conjunto de entrenamiento en entrenamiento y validación mediante una segunda iteración, tal y como se muestra en la figura 5.2.

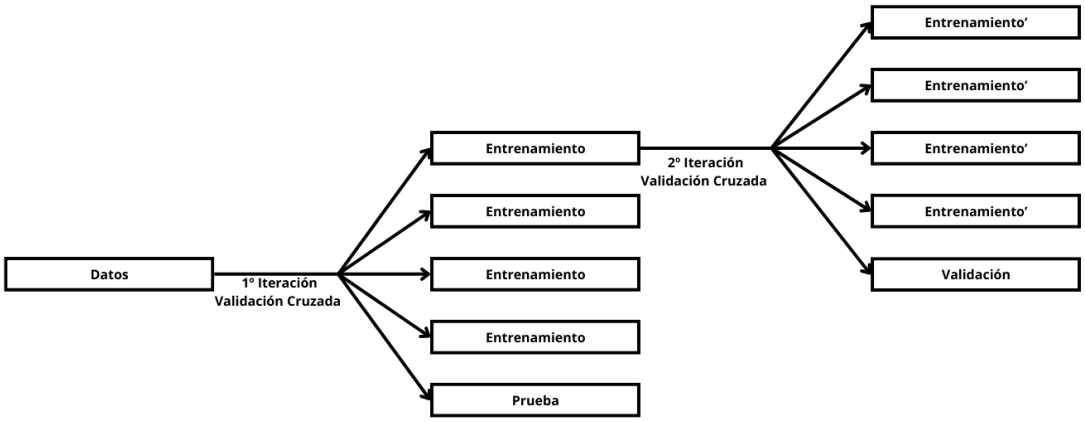

Figura 5.2: Selección de muestras.

Validación cruzada

La validación cruzada es un método que permite mejorar la generalización de los modelos de clasificación, evitando de esta forma el sobre-entrenamiento [19]. Se trata de una técnica

simple que plantea lo siguiente:

- Dividir el conjunto de datos en *n* partes iguales.
- En cada ciclo de entrenamiento usar una de esas partes como conjunto de validación y las *n-1* restantes como conjunto de entrenamiento.

5.4.2 Selección de características

Por lo general, cuanto mayor sea el número de características, más preciso va a ser el modelo a la hora de hacer predicciones pero, de la misma forma, mayores serán los tiempos de ejecución y mayores serán las posibilidades de que se produzca un sobre-ajuste. Por esto, reducir la dimensionalidad de un conjunto de datos es un paso importante.

Existen varias técnicas que permiten realizar este trabajo. Con la finalidad de comparar diferentes resultados se ha decidido usar tres de ellas:

- **PCA (Análisis de Componentes Principales)**
- **LDA (Análisis Discriminante Lineal)**
- **RFECV (Eliminación Recursiva de Características mediante Cross-Validation)**

PCA (Análisis de Componentes Principales)

PCA [20] es una estrategia de reducción de dimensionalidad basada en transformar un conjunto de variables inicial en un nuevo conjunto de variables no correlacionadas, denominadas componentes principales. Cada una de estas es una combinación lineal de las variables originales y se ordenan en función de su valor de variabilidad explicada, es decir, el primer componente será el que mayor valor tenga, lo que nos permite seleccionar las más relevantes.

Para conseguir esto, primero se calcula la matriz de covarianza, la cual permite identificar las relaciones lineales entre las variables. Luego, se determinan las direcciones de las componentes principales, así como la varianza que pueden explicar y, finalmente, se seleccionan tantas componentes como sea necesario.

En este caso, para asegurarse de que se mantiene la mayor cantidad de información, se ha estipulado que se debe explicar, al menos, el 95% de la variabilidad de los datos, es decir, el número de componentes principales seleccionado deberá de ser suficiente como para capturar el 95% de la varianza de los datos principales.

Una vez aplicado el algoritmo, se pasaría de las 768 características originales a 33.

LDA (Análisis Discriminante Lineal)

Junto con PCA, LDA [21] es uno de los métodos de reducción de dimensionalidad más utilizados. Esta técnica busca proyectar los datos a una baja dimensión, inferior a la de los originales, asegurando que se mantiene la máxima dispersión entre las clases, es decir, la distancia entre la media de cada una de las clases es máxima. En este caso concreto al contar con dos clases, estas se representarán en una única dimensión

Tras haber utilizado esta técnica, y como era esperado, se ha reducido el número de características a 1.

RFECV (Eliminación Recursiva de Características mediante Cross-Validation)

RFECV es una estrategia basada en la eliminación recursiva de características mediante el uso de validación cruzada.

Su funcionamiento consta de tres fases. En la primera de ellas, se van eliminando, iterativamente, las características menos relevantes. En la segunda, mediante validación cruzada, se evalúa el rendimiento con diferentes subconjuntos de características, y, finalmente, se selecciona la cantidad óptima de atributos.

Antes de aplicar esta técnica, se deben de eliminar los atributos redundantes, es decir aquellos que están altamente correlacionados entre sí, ya que podría afectar en gran medida al resultado final. Para esto se ha utilizado una función, extraída de [22], la cual a raíz de la matriz de correlaciones, calculada en el capítulo anterior, elimina una de las dos características cuyo coeficiente de correlación supera un umbral determinado, en este caso se ha utilizado 0.9. Tras aplicar la función al conjunto de datos, se eliminaron un total de 528 atributos, pasando de 768 a 240.

Con la redundancia eliminada, se debe establecer un modelo como estimador, para poder abordar la segunda etapa. En este caso se ha decidido utilizar un algoritmo de Regresión Logística, ya que ofrece buenos resultados a la hora de trabajar con conjuntos de alta dimensionalidad y es rápido de entrenar. A mayores de esto, también se deben estipular los siguientes parámetros:

- **Step**: Número de características que se eliminan en cada iteración. Se ha establecido a 1.
- **Cv**: Estrategia de validación cruzada escogida. En concreto se ha utilizado *StratifiedK-Fold*, desde la propia librería de *Scikit-Learn* con k = 5.

- **Scoring**: Métrica para evaluar la selección de características. Se ha escogido la exactitud. Al contar con un conjunto de datos equilibrado, esta métrica proporciona una buena idea general sobre el funcionamiento del modelo.
- **Njobs**: Cantidad de núcleos del procesador utilizados para realizar el proceso. Con el valor a -1 utiliza la CPU en su totalidad.

Después de aplicar la estrategia, como se puede ver en la figura 5.3, se ha comprobado que al superar las 150 características, la exactitud media del modelo estimador se empieza a estancar entre el 75% y el 77.5%, siendo el número óptimo de atributos 169.

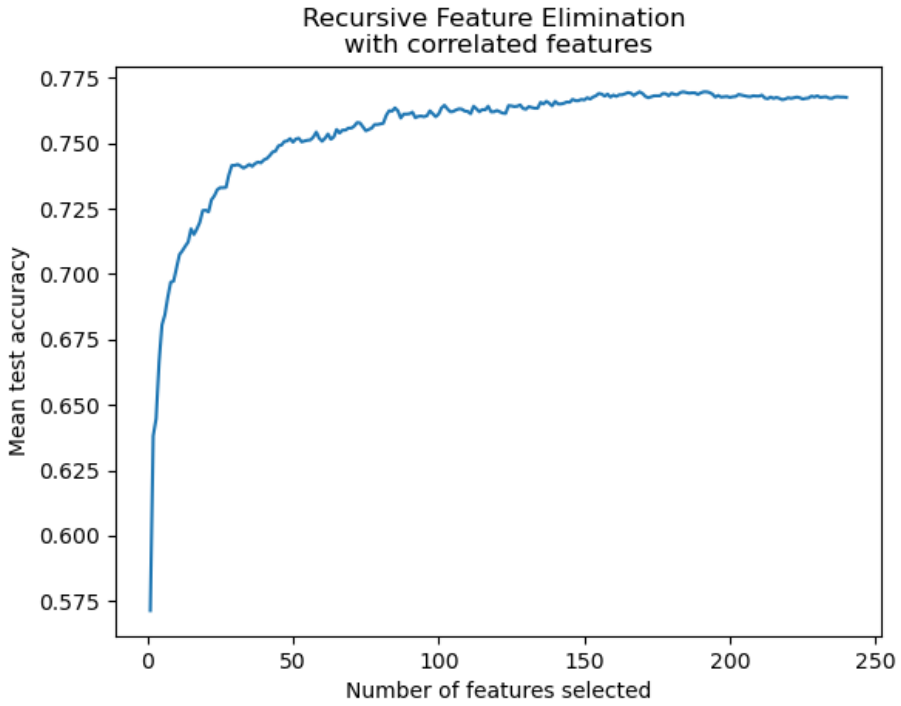

Figura 5.3: Número de Características vs. Exactitud.

Elección final

Una vez aplicados los tres métodos, se han comparado los resultados para seleccionar el que, a priori, es mejor. El caso de LDA, es bastante extremo, al reducir la dimensionalidad a una única característica. Por esto mismo se ha descartado, ya que se podría estar perdiendo bastante información sobre el conjunto de datos. En cambio tanto PCA como RFECV ofrecen una dimensionalidad que asegura que no se pierden atributos relevantes, siendo la primera de estas dos la que mayor número de características elimina.

Debido a que una reducción excesiva puede desembocar en peores resultados, se han escogido los atributos obtenidos por RFECV.

Capítulo 6

Modelado

Con los datos completamente procesados, es necesario construir y entrenar un modelo clasificador que sea capaz de abordar el problema inicial. Para esto se definen varias tareas, las cuales se pueden ver en la figura 6.1 [4] y sobre las que tratará este capítulo.

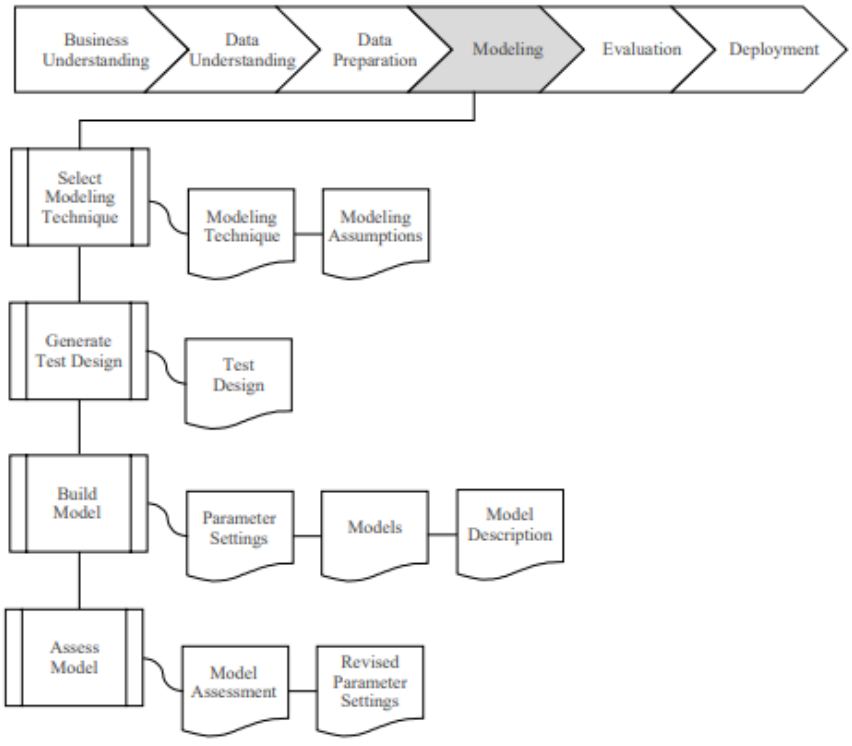

Figura 6.1: Modelado.

6.1 Selección de técnicas de modelado

En lo que se refiere a la clasificación de texto con BERT existen dos perspectivas:

- Usar BERT como un extractor de características.
- Ajustar BERT para realizar la clasificación.

En este caso se ha decidido seguir la primera de ellas. Durante el capítulo 4 se ha explicado el proceso de extracción de características, así como el modelo escogido para realizar este proceso.

Para abordar la tarea de clasificación se plantean tres modelos:

- **Modelo 1**: Un clasificador sencillo, compuesto por una capa *Dropout* y otra Densa con activación sigmoidal.
- **Modelo 2**: Un clasificador más complejo con tres capas Densas activadas mediante una función ReLU, una capa de normalización por lotes otra *Dropout* por cada una de las anteriores y un ultimo nivel Denso con activación sigmoidal.
- **Modelo 3**: Un SVM (Support Vector Machine). Un SVM es un algoritmo de aprendizaje supervisado, utilizado en tareas de clasificación [23], que tiene como objetivo encontrar el hiperplano que sea capaz de separar los datos de ambas clases, asegurando que la distancia entre los puntos mas cercanos de cada una de ellas sea la mayor posible.

Tanto el primer como el segundo modelo están construidos basándose en trabajos similares a este, como [24], [25], [26], [27]. En lo que respecta al tercero, se ha planteado para abordar el problema desde una perspectiva diferente.

En las figuras 6.2 y 6.3 se pueden apreciar las arquitecturas de los modelos 1 y 2 respectivamente.

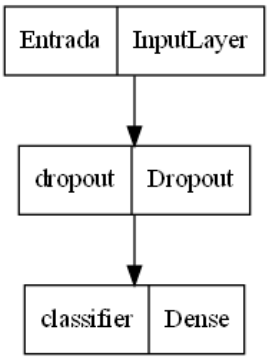

Figura 6.2: Arquitectura Modelo 1.

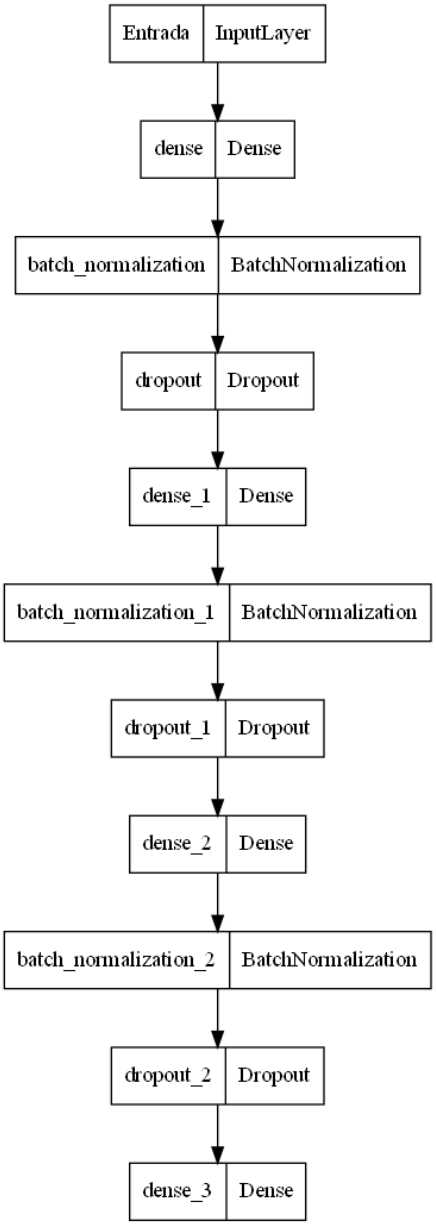

Figura 6.3: Arquitectura del Modelo 2.

6.2 Diseño de las pruebas

Existen varias métricas que nos permiten evaluar el funcionamiento de un modelo tales como la exactitud, la precisión o la sensibilidad y cada una de ellas ofrece información relevante sobre el comportamiento ante situaciones desconocidas. Estas se calculan a raíz de una matriz de confusión.

Una matriz de confusión es una representación tabular que permite visualizar, para cada una de las clases, las diferencias entre las predicciones obtenidas y los resultados esperados. En este caso, al tratarse de clasificación binaria, contaríamos con matrices de una dimensionalidad de 2x2, en las que cada una de las celdas representa lo siguiente:

- Celda superior izquierda: Mensajes negativos clasificados como negativos, es decir, **Verdaderos Negativos**
- Celda superior derecha: Mensajes negativos clasificados como positivos, es decir, **Falsos Positivos**
- Celda inferior izquierda: Mensajes positivos clasificados como negativos, es decir, **Falsos Negativos**
- Celda inferior derecha: Mensajes positivos clasificados como positivos, es decir, **Verdaderos Positivos**

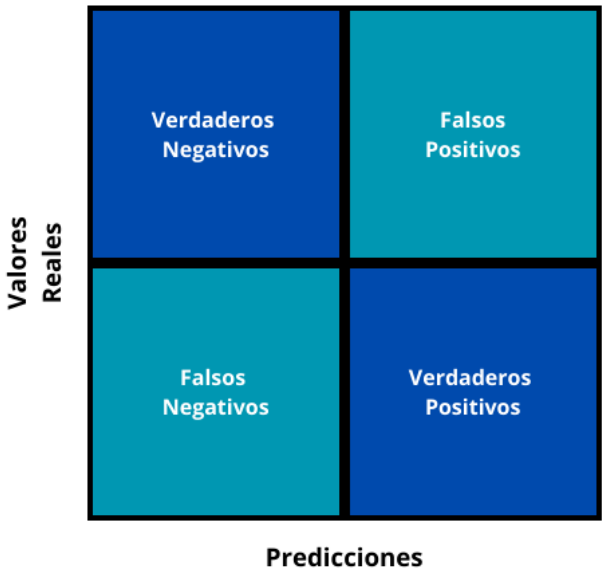

Figura 6.4: Matriz de confusión.

A priori, la exactitud puede no ser la mejor métrica a tener en cuenta para valorar el rendimiento de un modelo de clasificación, ya que tanto si se cuenta con un conjunto de datos desbalanceado como si se espera minimizar errores, reducir el número de falsos positivos o falsos negativos, puede resultar engañosa. Aún así, es una buena métrica para este proyecto ofreciendo una buena visión general sobre el funcionamiento del modelo al contar con un conjunto balanceado, existiendo la misma cantidad de muestras para cada una de las clases. Por esto mismo se usará como métrica principal.

Cabe destacar que, para poder conseguir más información, también se valorarán la precisión y la sensibilidad en la evaluación final como métricas secundarias.

6.3 Construcción del modelo

6.3.1 Selección de parámetros

Antes de entrenar es necesario conocer los hiperparámetros[1] correspondientes a cada uno de los modelos. Estos, a diferencia del resto, deben ser establecidos manualmente antes de realizar el entrenamiento, ya que dependiendo de sus valores se obtendrán mejores o peores resultados.

Los modelos 1 y 2 dependen de los siguientes:

- **Epochs**: Número de veces que se muestra el conjunto de entrenamiento completo al modelo. Para este hiperparámetro se han realizado pruebas con 10, 50 y 60 epochs.
- **Learning Rate**: Porcentaje de cambio con el que se actualizan los pesos del modelo en cada iteración. El valor estándar de este es de 0.001 y se han valorado también 0.01 y 0.0001
- **Batch Size**: Número de muestras que se introducen al modelo en cada iteración durante el entrenamiento. Se han probado 32, 64, 128 y 256.

El modelo 3, al tratarse de un SVM se utilizan los siguientes:

- **Parámetro de Regularización C**: Controla el equilibrio entre maximizar el margen entre las clases y minimizar el error de clasificación en el conjunto de entrenamiento. En este caso se han probado los valores 0.1, 1, 10 y 100.
- **Función del Kernel**: Especifica la función del kernel que se utiliza para transformar los datos en un espacio de características de mayor dimensión. Para este parámetro se han realizado pruebas con los kernels linear, rbf y poly.

[1] Parámetro cuyo valor se utiliza para controlar el proceso de aprendizaje

- **Gamma**: Controla la influencia de un solo ejemplo de entrenamiento. Se han probado scale y auto.

La forma más básica de determinar cuál es la mejor combinación de hiperparámetros es experimentar con las diferentes combinaciones, escogiendo la que, de entre todas, consiga los mejores resultados. Como es lógico, este proceso consume mucho tiempo ya que se deben probar, una a una, todas las combinaciones posibles. Con la idea de automatizar y agilizar este proceso, nacen diversas técnicas como *Grid Search* o *Random Search*. En este caso se ha decidido usar la primera de las dos.

Grid Search

Grid Search es una técnica, que, como se menciona anteriormente, permite determinar qué valores de los diferentes hiperparámetros consiguen mejores resultados. A raíz de una rejilla, la cual recoge todos estos valores, realiza una búsqueda exhaustiva hasta encontrar la combinación con la que se obtienen las predicciones que, en este caso, tengan una mayor exactitud para un conjunto de entrenamiento y validación concreto.

Para aplicar este algoritmo, se importa desde la librería *Scikit Learn* la clase *GridSearchCV* la cual, indicándole el modelo que se va a utilizar, la rejilla con los valores a probar, la métrica a valorar y la estrategia de validación cruzada, nos devuelve la mejor de todas las posibles combinaciones, así como una instancia del modelo ya entrenado con esos mismos parámetros.

Validación cruzada anidada

Un factor crucial a tener en cuenta a la hora de seleccionar los hiperparámetos es que existe la posibilidad de que, de manera indirecta, se enseñen al modelo algunas de las muestras que se usarán para la evaluación lo que, como es lógico, puede provocar un sobreentrenamiento. Con la finalidad de evitar esto, se plantea el uso de una estrategia de validación cruzada anidada la cual, como ya se sugirió en el capítulo anterior, propone separar los conjuntos de entrenamiento, validación y prueba mediante dos iteraciones de validación cruzada. Teniendo en cuenta que no existe un desbalance en las clases se ha decidido utilizar la clase *KFold* de la librería *Scikit Learn*, lo que se correspondería a una estrategia de validación cruzada simple, para ambas iteraciones.

Finalmente obtendríamos un proceso de entrenamiento y posterior evaluación como el que se muestra en la figura 6.5.

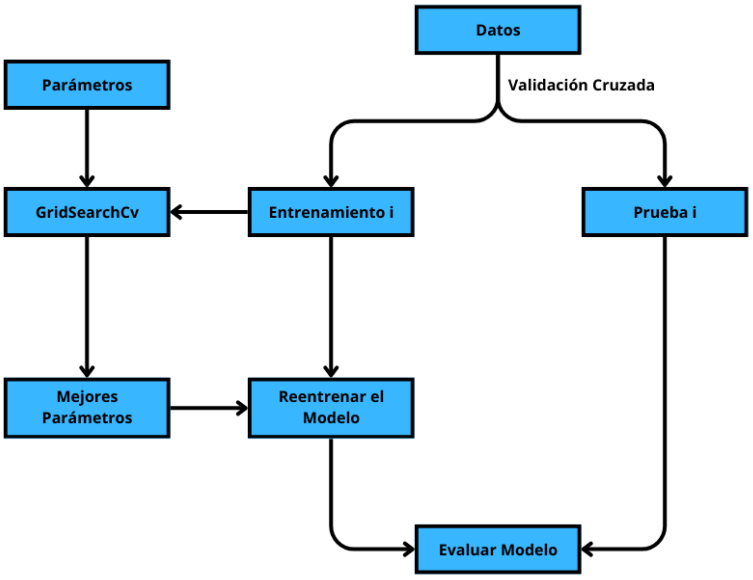

Figura 6.5: Algoritmo de validación cruzada anidada.

6.4 Resultados de hiperparametrización

Como ya se ha explicado anteriormente, la métrica principal a tener en cuenta para la evaluación es la exactitud por lo que, para seleccionar la mejor combinación de hiperparámetros, se plantea comparar tanto la exactitud media como la desviación típica de las mejores combinaciones devueltas para cada uno de los folds externos, es decir, para cada una de las 5 divisiones del conjunto de datos en entrenamiento y prueba.

6.4.1 Modelo 1

Fold Externo	Batch Size	Epochs	Learning Rate	Exactitud Media	Desviación Típica
1	32	60	0.001	76.26%	0.014
2	32	60	0.001	75.93%	0.012
3	128	50	0.01	76.45%	0.017
4	32	50	0.01	75.85%	0.008
5	32	50	0.01	76.01%	0.009

Tabla 6.1: Resultados de Entrenamiento.

Una vez aplicado el proceso de selección se obtuvieron para cada uno de los folds, los resultados mostrados en la tabla 6.1. Des estos mismos podemos concluir que los mejores

parámetros para este son 50 epochs, un batch size de 32 y un learning rate de 0.01 ya que, aún no siendo los que mejor exactitud media consiguen, son los más frecuentes y demuestran mayor estabilidad, al contar con una desviación típica menor.

6.4.2 Modelo 2

Fold Externo	Batch Size	Epochs	Learning Rate	Exactitud Media	Desviación Típica
1	64	60	0.001	78.14%	0.007
2	128	60	0.001	77.56%	0.011
3	256	60	0.001	78.26%	0.014
4	32	60	0.01	78.20%	0.011
5	256	60	0.01	77.73%	0.010

Tabla 6.2: Resultados de Entrenamiento.

En este segundo modelo, y tras seguir un procedimiento similar al utilizado en el anterior, los resultados, mostrados en la tabla 6.2, demuestran de una forma muy clara que en lo que respecta a los epochs, su mejor valor es de 60, siendo este el escogido en cada una de las divisiones. Para los otros dos parámetros la elección no es tan clara pero, como se puede apreciar, la combinación correspondiente a la primera fila, con un learning rate de 0.001 y un batch size de 64, si bien lo es la que mayor exactitud media consigue, si es la más estable de todas.

6.4.3 Modelo 3

Fold	C	Gamma	Kernel	Exactitud Media	Desviación Típica
1	0.1	scale	linear	77.06%	0.010
2	100	scale	linear	76.89%	0.013
3	10	auto	rbf	78.31%	0.012
4	10	auto	rbf	77.90%	0.004
5	1	scale	linear	77.31%	0.011

Tabla 6.3: Resultados de Entrenamiento.

Para este tercer modelo, de la misma forma que con los demás, una vez aplicado el algoritmo de selección, se puede concluir que un valor de 10 para el parámetro de regulación (C),

un kernel rbf y un gamma auto son la mejor combinación posible ya que, aún no siendo la más estable, obtiene los resultados de exactitud media más altos.

6.5 Entrenamiento final

Una vez seleccionados los hiperparámetros se entrenaron cada uno de los modelos para, posteriormente, poder evaluar su comportamiento y comprobar si se cumplen los objetivos propuestos.

Capítulo 7

Evaluación

Una vez entrenados los modelos, se debe evaluar tanto el funcionamiento del estos, como los procesos seguidos para obtener estos resultados, así como establecer las acciones que se deben realizar basándose en las conclusiones obtenidas, tal y como se muestra en la figura 7.1 [4].

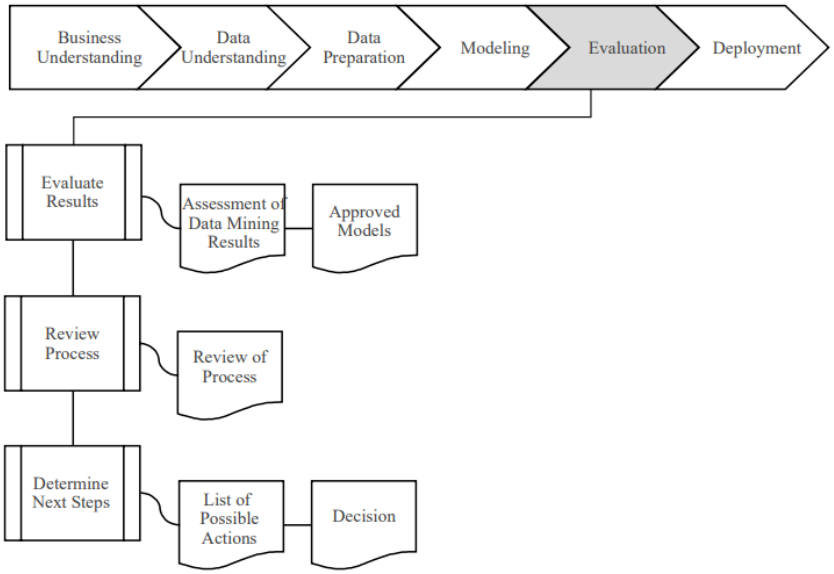

Figura 7.1: Evaluación.

7.1 Evaluación de los resultados

El primero de los pasos para valorar si se cumplen los objetivos establecidos es comprobar el funcionamiento de los modelos sobre los conjuntos de prueba. Para esto, se han calculado las matrices de confusión de cada uno de los modelos contra los conjuntos de prueba.

Como se explica en el capítulo anterior, estas matrices nos permiten calcular las diferentes métricas utilizadas para la evaluación de esta forma:

- **Exactitud**: Proporción de predicciones correctas sobre el total de predicciones realizadas.
$$\frac{VP + VN}{VP + VN + FP + FN} \tag{7.1}$$
- **Precisión**: Proporción de verdaderos positivos sobre el total de positivos predichos. Mide la calidad de las predicciones positivas.
$$\frac{VP}{VP + FP} \tag{7.2}$$
- **Sensibilidad**: Proporción de verdaderos positivos sobre el total de positivos reales. Mide la capacidad del modelo para detectar todos los casos positivos.
$$\frac{VP}{VP + FN} \tag{7.3}$$

En concreto, y como ya se ha planteado a la hora de diseñar las pruebas también en el capítulo 6, se evaluará, principalmente, la exactitud de los tres modelos así como su precisión y su sensibilidad.

A continuación se discutirán los resultados obtenidos para cada uno de los modelos.

7.1.1 Modelo 1

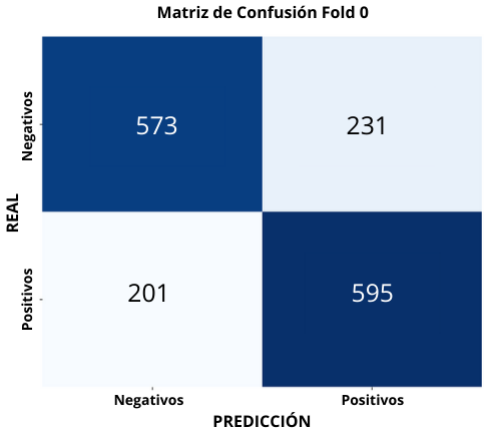

Figura 7.2: Matriz de confusión fold 1.

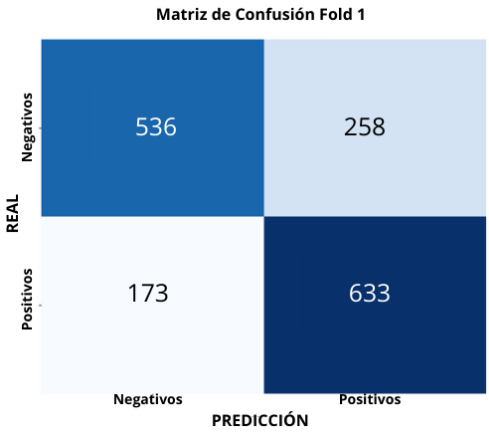

Figura 7.3: Matriz de confusión fold 2.

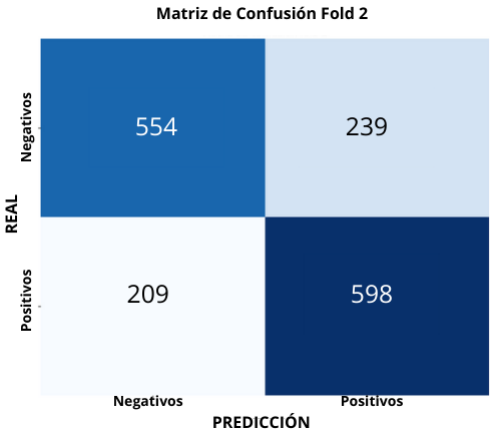

Figura 7.4: Matriz de confusión fold 3.

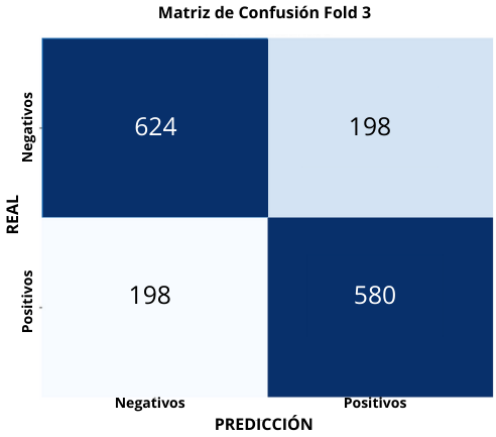

Figura 7.5: Matriz de confusión fold 4.

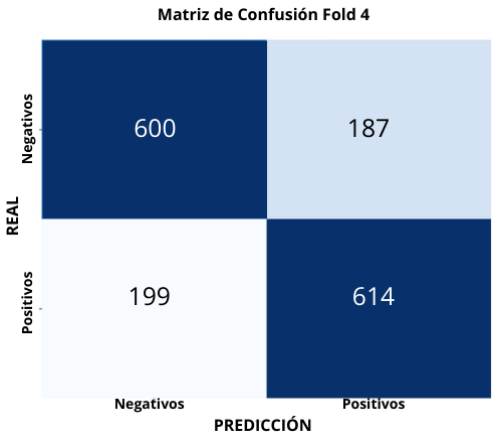

Figura 7.6: Matriz de confusión fold 5.

Una vez obtenidas las matrices de confusión, mostradas en las figuras 7.2, 7.3, 7.4, 7.5, 7.6, se calcularon las métricas correspondientes.

Al haber seguido una estrategia de validación cruzada, es relevante conocer el promedio de las métricas seleccionadas, así como sus desviaciones típicas. En el caso de este primer modelo, se obtuvieron unos resultados relativamente estables logrando, para cada una de las métricas, los promedios mostrados en la tabla 7.1.

Métrica	Media	Desviación Típica
Exactitud	73.83%	0.014
Precisión	73.15%	0.021
Sensibilidad	75.49%	0.015

Tabla 7.1: Métricas modelo 1

En concreto se alcanzó una exactitud máxima de 75.87% en el cuarto fold, siendo este también en el que mejores valores de precisión se obtienen, con un 76.65% , sin embargo, en lo que respecta a la sensibilidad se consiguió el mayor valor en el segundo fold, siendo esta del 78.53% .

7.1.2 Modelo 2

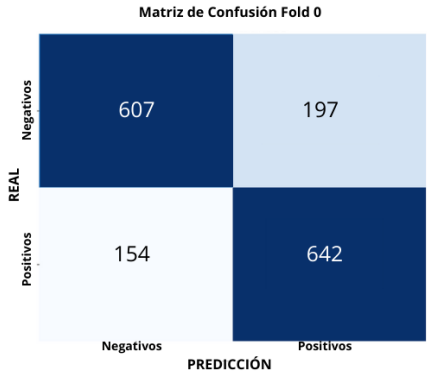

Figura 7.7: Matriz de confusión fold 1.

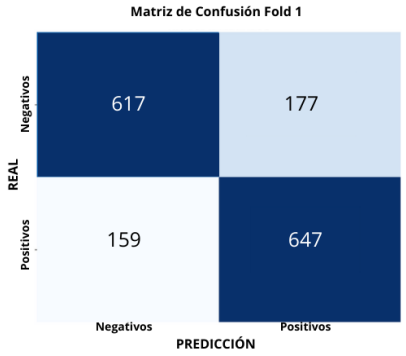

Figura 7.8: Matriz de confusión fold 2.

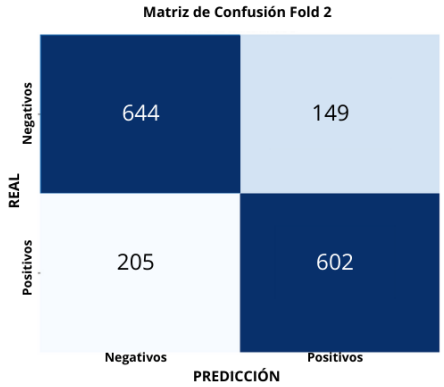

Figura 7.9: Matriz de confusión fold 3.

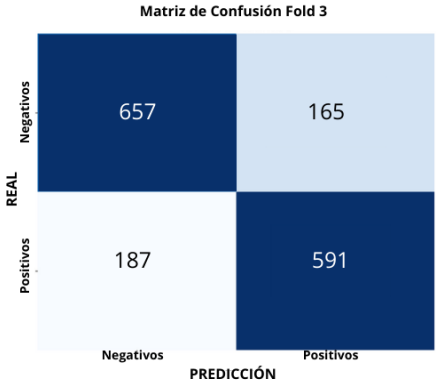

Figura 7.10: Matriz de confusión fold 4.

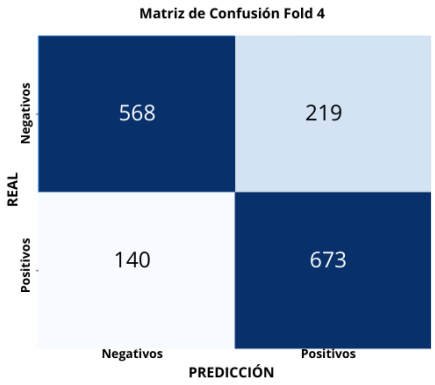

Figura 7.11: Matriz de confusión fold 5.

De forma similar al modelo anterior, una vez obtenidas las matrices de confusión, mostradas en las figuras 7.7, 7.8, 7.9, 7.10, 7.11, se calcularon las métricas correspondientes.

En este caso los resultados fueron los mostrados en la tabla 7.2, en los cuales se puede apreciar una alta estabilidad.

Métrica	Media	Desviación Típica
Exactitud	78.10%	0.004
Precisión	77.76%	0.016
Sensibilidad	78.85%	0.030%

Tabla 7.2: Métricas modelo 2

En lo que respecta a los valores máximos existe una mayor disparidad, al encontrarse cada uno en un fold diferente, concretamente la exactitud más alta corresponde al fold 1 con un 79%, la mayor precisión se corresponde al fold 2 alcanzando un 80.15% y el mejor valor de sensibilidad se corresponde al fold 4, siendo esta de un 82.77%.

7.1.3 Modelo 3

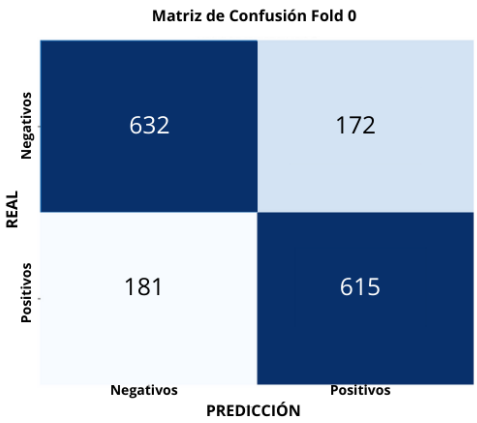

Figura 7.12: Matriz de confusión fold 1.

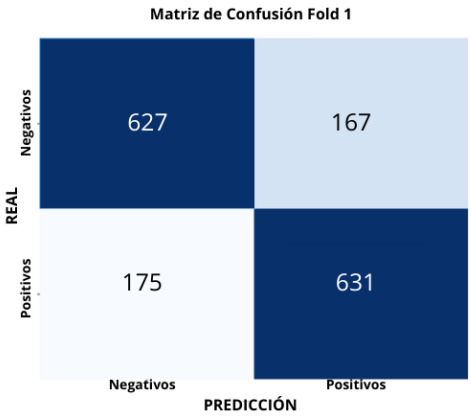

Figura 7.13: Matriz de confusión fold 2.

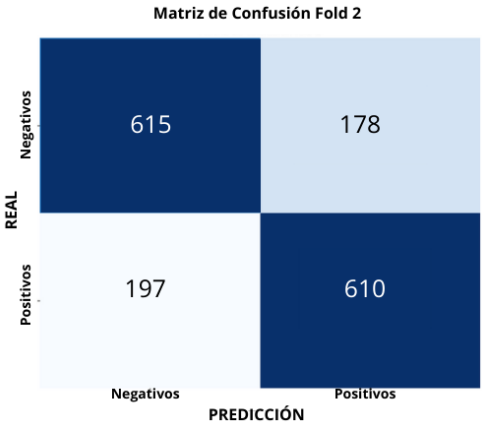

Figura 7.14: Matriz de confusión fold 3.

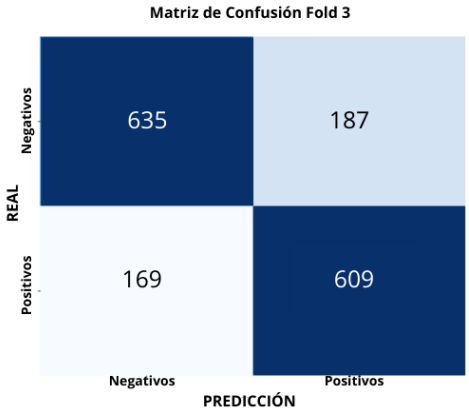

Figura 7.15: Matriz de confusión fold 4.

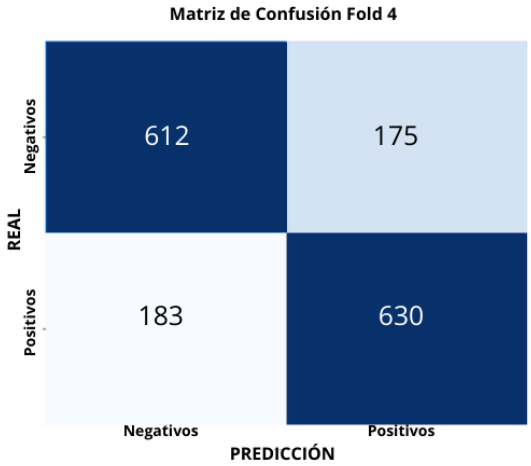

Figura 7.16: Matriz de confusión fold 5.

Con las matrices de confusión calculadas, mostradas en las figuras 7.12, 7.13,7.14, 7.15, 7.16, se calcularon las métricas correspondientes.

En este caso podemos afirmar que el comportamiento del modelo es bastante estable siendo, como se muestra en la tabla 7.3, el que menores desviaciones típicas presenta de los tres. Además, a diferencia del resto, los mejores valores de las métricas se corresponden al mismo fold, en concreto al segundo, llegando a alcanzar un 78.62%, 79.07% y 78.28% de exactitud, precisión y sensibilidad respectivamente.

Métrica	Media	Desviación Típica
Exactitud	77.70%	0.006
Precisión	77.88%	0.008
Sensibilidad	77.38%	0.009

Tabla 7.3: Métricas modelo 3

7.1.4 Evaluación final de los modelos

Con todas los valores calculados, y a modo de resumen y comparación final, se han contrastado los tres modelos.

Modelo	Exactitud	Precisión	Sensibilidad
1	73.83%	73.15%	75.49%
2	78.10%	77.76%	78.85%
3	77.70%	77.88%	77.38%

Tabla 7.4: Evaluación final de los modelos

Con esto podemos comprobar que, como era esperado aunque no por mucho, el modelo con un peor rendimiento es el primero de todos que es a su vez el más sencillo. Sin embargo, tanto el modelo 2 como el 3 obtienen resultados muy similares.

Como se puede apreciar claramente, existe un estancamiento en torno al 78% en todas las métricas analizadas lo cual puede estar producido por varias razones, siendo las más probable una posible ambigüedad en el dataset debido a un mal etiquetado, o a la propia naturaleza del problema pudiendo existir comentarios que, dependiendo del contexto, puedan dar la sensación de ser negativos cuando realmente no lo son y viceversa. Por desgracia, no existen otros trabajos que utilicen el mismo conjunto de datos para una tarea similar por lo que no es posible realizar comparaciones que corroboren esta hipótesis.

7.2 Evaluación del proceso

Para conseguir estos resultados se ha intentado seguir, de la forma más adecuada posible, cada una de las tareas propuestas en la metodología escogida pero, aún así, estos no son los

mejores. Esto puede ser consecuencia de diferentes razones:

- **Calidad del dataset**: Si bien no es complicado encontrar conjuntos de datos que recojan una gran cantidad de mensajes enviados por jugadores, aún no siendo los más actualizados, sí lo es el conseguir datasets etiquetados que puedan ser utilizados para poder cumplir los objetivos del negocio ya que, la mayoría de estos, suelen estar etiquetados de forma manual pudiendo existir un cierto sesgo a la hora de decidir si un comentario es o no tóxico. Esto es algo que ya se tuvo en cuenta a la hora de valorar los diferentes conjuntos y debido al tiempo que consumiría el seleccionar y etiquetar las muestras, se tomó la decisión de continuar con una de las opciones disponibles sobre construir un dataset propio.

- **Preprocesado del texto**: Por lo general, los mensajes con los que se trabaja, como se puede ver en el capítulo de Comprensión de los Datos 4, son cortos y están compuestos por una gran cantidad de palabras vacías como adverbios o artículos. A mayores también presentan faltas ortográficas, así como emoticonos que pueden afectar al significado del propio comentario. Además, para ajustar los tiempos, se tomó la decisión de no realizar ningún preprocesado sobre el texto con el cual poder solucionar estos problemas, por lo que existe la posibilidad de que se esté perdiendo información que pueda ser útil a la hora de realizar la clasificación.

- **Procesamiento de los datos**: En la mayoría de proyectos que utilizan BERT para clasificación, se suele utilizar el propio texto directamente, ajustando los pesos de BERT para adaptarlo a la tarea concreta. En este caso, y como se ha explicado en el capítulo de Modelado 6, se ha decidido utilizarlo como un extractor de características,"congelando" los pesos de este. Por esto, y para reducir la complejidad de los procesos de entrenamiento y evaluación, se ha realizado un procesado de los datos mediante el cual se han eliminado atributos que, a priori, no aportan información relevante sobre las muestras.

7.3 Acciones futuras

Teniendo en cuenta los puntos expuestos anteriormente, se deberían de tomar diferentes decisiones con el fin de abordar estos problemas e intentar mejorar los resultados obtenidos.

Principalmente se tendría que revisar la fiabilidad del dataset, así como realizar pruebas modificando el conjunto de datos para que las muestras ofrezcan la mayor cantidad de información posible.

Lo siguiente a tener en cuenta sería el realizar nuevos experimentos con modelos más complejos o, de la misma forma, ajustar los pesos de BERT y no usarlo únicamente como un extractor de características cosa que, a priori, debería de mejorar los resultados finales a cambio de sacrificar algo de tiempo en el proceso de entrenamiento.

Capítulo 8

Conclusiones

En este último capítulo, y a modo de cierre, se analizarán los resultados obtenidos, revisando cada una de las fases para comprobar el grado de compleción de los objetivos del proyecto. A mayores, se expondrá una breve explicación sobre las lecciones aprendidas y las posibles líneas de investigación futura que podrían surgir a partir de este trabajo.

8.1 Grado de compleción

Para valorar si se han cumplido los objetivos propuestos, tanto desde el punto de vista del negocio como desde la perspectiva de la minería de datos, se ha analizado cada una de las fases del proyecto.

Como primer paso, fue necesario familiarizarse tanto con la metodología a seguir como con las tecnologías a utilizar, al ser la primera vez que se trabajaba con estas. Esta fase, la cual comprende los capítulos 2 y 3, ha sido crítica ya que era necesaria para asentar las bases del trabajo. El segundo paso consistió en comparar los distintos conjuntos de datos disponibles y seleccionar el que, a priori, se ajustaba mejor al proyecto y, una vez escogido, seguir una serie de procedimientos para entenderlo y procesarlo, como se muestra en los capítulos 4 y 5.

Con las muestras preparadas, se avanzó a las dos últimas fases. En la primera de ellas, capítulo 6, se plantearon tres modelos, dos clasificadores basados en redes neuronales y una máquina de soporte vectorial, y, una vez definidos, se realizó el proceso de selección de hiperparámetros y posterior entrenamiento. Como fase final, abordada en el capítulo 7, se evaluó contra el conjunto de prueba.

Al haber seguido todas las fases alcanzado los objetivos intermedios propuestos en el capítulo 1, se considerar que, teniendo en cuenta el alcance y el propósito real del proyecto,

se han cumplido las expectativas iniciales.

8.2 Lecciones aprendidas

Durante la realización de este trabajo, se profundizó en las diferentes técnicas de procesamiento del lenguaje natural, así como en los diferentes procesos del *Machine Learning*. Principalmente ha servido como una pequeña introducción al mundo del NLP, ampliando los conceptos aprendidos en el grado, y aportando nuevos conocimientos como el funcionamiento de los grandes modelos de lenguaje (LLM), y como es lógico sobre cómo trabaja *BERT* a la hora de procesar mensajes de texto. Además de esto, también se ha adquirido experiencia en lo que respecta a la metodología CRISP-DM al ser la primera vez que se usa. Finalmente, se ha indagado en nuevas formas de tratar los datos, tanto en lo que se refiere a la comprensión como al procesamiento de los mismos, así como en varias estrategias de entrenamiento y evaluación, en concreto la validación cruzada anidada, mejorando los conocimientos vistos a lo largo de las asignaturas de la mención de *Computación*.

En resumen, el proyecto ha servido tanto para reforzar como para ampliar las aptitudes aprendidas a lo largo de la carrera en lo que se refiere al campo del *Machine Learning* y más concretamente al procesamiento del lenguaje natural y la clasificación de texto.

8.3 Líneas futuras

En el caso de querer seguir avanzando con este trabajo se deberían de tener en cuenta varias acciones a tomar.

Principalmente, y a modo de finalización, debería abordarse la fase de despliegue para comprobar si se cumplen los objetivos del negocio en un escenario realista. Esto podría hacerse en entornos distribuidos como Azure o Aws en los cuales simular el comportamiento de un canal de comunicación, que funcione de manera similar al de un videojuego, para poder valorar si la solución propuesta es válida o no.

Otras acciones a seguir, en caso de querer mejorar los resultados, serían las ya expuestas en la evaluación, capítulo 7, como el probar diferentes conjuntos de datos o, siguiendo con la idea de utilizar BERT como extractor de características, probar otros modelos de *Machine Learning* como árboles de decisión, Ramdom Forest o k-Nearest Neighbors y comparar sus resultados con los obtenidos en este trabajo, así como probar otros LLM como GPT de OpenAI.

Bibliografía

[1] Esports charts. Última comprobación: 24 April 2024. [En línea]. Disponible en: https://escharts.com/top-games?order=peak

[2] D. SYDORENKO. (2023) Toxicity in gaming: unveiling the 'closed club' and revolutionising cooperative play. Última comprobación: 24 April 2024. [En línea]. Disponible en: https://maddy.news/3tt1ZVv

[3] F. Martínez-Plumed, L. Contreras-Ochando, C. Ferri, J. Hernández-Orallo, M. Kull, N. Lachiche, M. J. Ramírez-Quintana, and P. Flach, "Crisp-dm twenty years later: From data mining processes to data science trajectories," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 8, pp. 3048–3061, 2021.

[4] P. C. Ncr, R. K. Ncr, J. Spss, T. Spss, T. Daimlerchrysler, and R. Daimlerchrysler, "The crisp-dm process model,(c)," *no. C*, 1999.

[5] ¿qué es el procesamiento de lenguaje natural (nlp)? Última comprobación: 02 Mayo 2024. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/nlp/

[6] What is natural language processing (nlp)? Última comprobación: 02 Mayo 2024. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/nlp/

[7] M. Mansurova. (2024) Text embeddings: Comprehensive guide. Última comprobación: 02 Mayo 2024. [En línea]. Disponible en: https://towardsdatascience.com/text-embeddings-comprehensive-guide-afd97fce8fb5

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[9] ¿qué son los transformadores en la inteligencia artificial? Última comprobación: 02 Mayo 2024. [En línea]. Disponible en: https://aws.amazon.com/es/what-is/transformers-in-artificial-intelligence/

[10] K. T. Chitty-Venkata, S. Mittal, M. Emani, V. Vishwanath, and A. K. Somani, "A survey of techniques for optimizing transformer inference," *Journal of Systems Architecture*, vol. 144, p. 102990, 2023. [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S1383762123001698

[11] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[12] Y. Chae and T. Davidson, "Large language models for text classification: From zero-shot learning to fine-tuning," *Open Science Foundation*, 2023.

[13] Wikipedia, "Bert (modelo de lenguaje) — wikipedia, la enciclopedia libre," 2024, [Internet; descargado 19-abril-2024]. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=BERT_(modelo_de_lenguaje)&oldid=159549162

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] R. Horev. (2024) Bert explained: State of the art language model for nlp. Última comprobación: 03 Mayo 2024. [En línea]. Disponible en: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[16] Disposición 17238 del boe núm. 177 de 2023. Última comprobación: 11 Mayo 2024. [En línea]. Disponible en: https://www.boe.es/boe/dias/2023/07/26/pdfs/BOE-A-2023-17238.pdf

[17] TENSORFLOW, "bert," https://www.kaggle.com/models/tensorflow/bert/tensorFlow2/en-cased-l-12-h-768-a-12/3?tfhub-redirect=true.

[18] t-sne. Última comprobación: 07 Septiembre 2024. [En línea]. Disponible en: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/t-sne

[19] D. Berrar *et al.*, "Cross-validation." 2019.

[20] F. González. (2023) Análisis de componentes principales (pca). Última comprobación: 9 Julio 2024. [En línea]. Disponible en: https://es.linkedin.com/pulse/an%C3%A1lisis-de-componentes-principales-pca-francisco-gonz%C3%A1lez?utm_source=share&utm_medium=guest_desktop&utm_campaign=copy

[21] A. J. Anaya-Isaza, D. H. Peluffo-Ordoñez, J. C. Alvarado-Pérez, J. Ivan-Rios, J. A. Castro-Silva, P. D. Rosero-Montalvo, D. F. Peña-Unigarro, and A. C. Umaquinga-Criollo, "Estudio

comparativo de métodos espectrales para reducción de la dimensionalidad: Lda versus pca," *INCISCOS 2016*, 2017.

[22] A. Siddiqui, "Feature-selection," https://github.com/siddiquiamir/Feature-Selection/blob/main/Multicollinearity.ipynb, 2022.

[23] MathWorks, "Introducción a support vector machine (svm)," Última comprobación: 07 Septiembre 2024. [En línea]. Disponible en: https://es.mathworks.com/discovery/support-vector-machine.html

[24] F. Ayık, "Mastering text classification with bert: A comprehensive guide," Última comprobación: 05 Agosto 2024. [En línea]. Disponible en: https://medium.com/@ayikfurkan1/mastering-text-classification-with-bert-a-comprehensive-guide-194ddb2aa2e5

[25] TENSORFLOW, "Classify text with bert," Última comprobación: 05 Agosto 2024. [En línea]. Disponible en: https://www.tensorflow.org/text/tutorials/classify_text_with_bert

[26] N. Sakhiya, "Text classification using bert," Última comprobación: 05 Agosto 2024. [En línea]. Disponible en: https://www.kaggle.com/code/nayansakhiya/text-classification-using-bert

[27] R. Cantini, "Bert_text_classification," Última comprobación: 05 Agosto 2024. [En línea]. Disponible en: https://github.com/rcantini/BERT_text_classification/blob/main/toxic_comments/BERT_toxic_comment.py