Técnicas NLP para análisis de sentimiento y resumen de noticias financieras

Jorge Ballarín Corvinos

Trabajo de fin de máster en Modelización e Investigación Matemática, Estadística y Computación
Universidad de Zaragoza

Directores del trabajo:
Juan Luis Fernández-Martínez
Lucas Fernández-Brillet

4 de septiembre de 2023

Prólogo

Nuestra capacidad de comunicación a través del lenguaje nos permite expresar pensamientos o sentimientos; no parece disparatado pensar que la habilidad de las máquinas para comprenderlo y analizarlo ha sido un objetivo fundamental en la inteligencia artificial. El procesamiento natural del lenguaje (NLP) y el análisis de sentimiento son disciplinas clave que permiten a las computadoras entender y responder al lenguaje humano de manera más sofisticada.

Estas tecnologías tienen un amplio rango de aplicaciones, desde la comprensión de opiniones en redes sociales hasta la detección de noticias falsas, y están transformando la forma en que interactuamos con la información. Esto se sabe en el mundo financiero, donde la toma de decisiones informadas es crucial. Surge, por tanto, la necesidad de aprovechar la información disponible en noticias y redes sociales. Es por ello, que el propósito de este trabajo es desarrollar un modelo de lenguaje que funcione como un clasificador de noticias con análisis de sentimiento.

Para alcanzar este objetivo, se recurrirá a técnicas de vanguardia en procesamiento de lenguaje natural y análisis de sentimiento. El proceso comenzará con la adaptación de un modelo preexistente, FinBERT, a través de un entrenamiento utilizando datos de noticias financieras. A continuación, se aplicará una reducción de dimensionalidad mediante un análisis de componentes principales (PCA), para después desarrollar nuevos modelos utilizando algoritmos como XGBoost, k-NN de vecinos próximos y redes neuronales. Estos modelos se afinarán aún más utilizando algoritmos de optimización global. Posteriormente, se podrán comparar con el modelo BERT base, analizando las métricas de cada modelo. Además, se investigarán nuevas bases de datos que evaluarán el rendimiento del modelo con otros conjuntos de datos distintos.

El objetivo final es crear un clasificador especializado, eficiente y rápido en el ámbito financiero que pueda resultar valioso para diversas actividades dentro de este sector.

Abstract

Our ability to communicate through language allows us to express thoughts or feelings; It does not look crazy to think that the skill of machines to understand and analyze it has been a fundamental objective in artificial intelligence. Natural language processing (NLP) and sentiment analysis are key disciplines that allow computers to understand and respond to human language in more sophisticated ways.

These technologies have a wide range of applications, from understanding opinions on social networks to detecting fake news, and are transforming the way we interact with information. This is known in the financial world, where informed decision making is crucial. Therefore, the need arises to take advantage of the information available in news and social networks. That is why the purpose of this work is to develop a language model that works as a news classifier with sentiment analysis.

To achieve this objective, cutting-edge techniques in natural language processing and sentiment analysis will be used. The process will begin by adapting a pre-existing model, FinBERT, through training using financial news data. Next, dimensionality reduction will be applied using principal component analysis (PCA), and new models will then be developed using algorithms such as XGBoost, k-NN nearest neighbors and neural networks. These models will be further refined using global optimization algorithms. Subsequently, they can be compared with the BERT base model, analyzing the metrics of each model. In addition, new databases will be investigated that will evaluate the performance of the model with other different data sets.

The ultimate goal is to create a specialized, efficient and fast classifier in the financial field that can be valuable for various activities within this sector.

Índice general

Capítulo 1

Introducción al problema

1.1. Contexto en el ámbito científico

Los *algoritmos* gobiernan el mundo. Son conjuntos de instrucciones que nos permiten transformar una entrada de datos en la respuesta que buscamos. Cada día, sin percatarnos, seguimos rutinas y procedimientos para llevar a cabo tareas específicas; estas mismas tareas podrían ser fácilmente controladas por un algoritmo. Aunque suena simple, este concepto es la base de la programación y la informática, ya que define cómo las computadoras abordan problemas de manera eficaz y organizada.

Aquí es donde entra en juego el *machine learning*, o aprendizaje automático, un campo de estudio que ha experimentado un rápido crecimiento en las últimas décadas. Surgiendo como una rama de la inteligencia artificial, se centra en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender y tomar decisiones a partir de datos, sin necesidad de ser programadas explícitamente para ello. El avance del machine learning ha sido impulsado por el fenómeno llamado *big data*, término que se refiere a la gran cantidad de datos que actualmente disponemos, y a la necesidad creciente de recuperarlos y gestionarlos de manera automática.

No obstante, este aumento de datos ofrece también ciertos problemas, ya que si bien proporciona una gran cantidad de información, también presenta desafíos en términos de calidad, variabilidad y relevancia de los datos. La aplicación de métodos de machine learning a grandes bases de datos se llama *minería de datos*, cuya tarea es procesar y explorar grandes conjuntos de datos con el fin de obtener patrones en ellos.

Dentro del vasto campo del machine learning, se encuentra una rama que ha revolucionado la forma en que las máquinas comprenden y procesan datos: el *deep learning* o *aprendizaje profundo*. Esta técnica se caracteriza por la utilización de múltiples capas de procesamiento de datos, conocidas como redes neuronales profundas. A diferencia de las técnicas convencionales de machine learning, que suelen requerir una selección manual de características, el deep learning permite que los modelos extraigan automáticamente las características relevantes de los datos. Estas redes neuronales profundas son capaces de aprender y abstraer patrones y representaciones complejas a partir de datos sin procesar, lo que ha impulsado avances notables en áreas como visión por computadora, procesamiento de lenguaje natural y más.

En el contexto del aprendizaje automático, existen dos enfoques principales: el *aprendizaje supervisado* y el *aprendizaje no supervisado*. En el aprendizaje supervisado, los modelos se entrenan utilizando ejemplos etiquetados, es decir, se les proporciona un conjunto de datos en el que las respuestas correctas ya están etiquetadas. Los modelos aprenden a hacer predicciones basadas en estas respuestas previas y, una vez entrenados, pueden aplicar sus conocimientos para hacer predicciones sobre datos no vistos. Por otro lado, el aprendizaje no supervisado implica entrenar modelos en datos sin etiquetas. El objetivo aquí es encontrar patrones ocultos, estructuras y relaciones dentro de los datos, lo que permite agrupar, segmentar y comprender la estructura subyacente sin orientación explícita.

El lenguaje humano es un sistema complejo y esencial para la comunicación entre las personas. Es la forma fundamental en la que transmitimos información y nos comunicamos entre nosotros. En consecuencia, no es sorprendente que en los últimos años la tecnología haya avanzado en el tratamiento del lenguaje. Para abordar estos desafíos, surge el *procesamiento del lenguaje natural* (NLP por sus siglas en inglés *Natural Language Processing*), una rama de la inteligencia artificial que se enfoca en la interacción entre las computadoras y el lenguaje humano. El NLP busca desarrollar algoritmos y modelos que permitan a las máquinas comprender, interpretar y generar texto de manera similar a como lo hacen los humanos.

El procesamiento de lenguaje natural es uno de los campos que más ha crecido en los últimos años, y cada vez es más utilizado en diversas aplicaciones, como el *análisis de sentimientos* en textos. La capacidad de comprender y analizar el sentimiento expresado en un texto puede ser de gran utilidad en áreas como el análisis de opiniones de clientes, la detección de noticias falsas o para saber si se está hablando bien de una empresa o de una marca. Sin embargo, esta tarea no es tan sencilla debido a la naturaleza subjetiva del lenguaje, así como la manera en que depende del contexto.

Dentro del contexto de machine learning, y más concretamente dentro del deep learning, los *modelos de lenguaje* se enfocan específicamente en comprender y generar lenguaje natural. Estos modelos se entrenan en grandes cantidades de texto para capturar patrones y estructuras del lenguaje. Pueden utilizarse para tareas como reconocimiento de voz, traducción automática, generación de texto y análisis de sentimientos, entre otras. En este trabajo, se usarán modelos de lenguaje para el análisis de sentimiento en textos, además de resumen de noticias; todo ello aplicado al mundo financiero.

1.2. Contexto en el máster

La ejecución de este proyecto demanda una combinación de diversos conocimientos adquiridos en el máster, incluyendo el modelado de problemas, la programación y las técnicas de machine learning. Más específicamente, se han aplicado enfoques discutidos en asignaturas como Minería de Datos o Algoritmos Bioinspirados y Técnicas Evolutivas. También se han seguido procedimientos similares a los abordados en Modelización Matemática, además de llevar a cabo procesos de optimización que se asemejan a los enseñados en las áreas de logística y optimización. Todos estos elementos han contribuido al desarrollo del trabajo, junto con otros aspectos relevantes de diferentes asignaturas que resulta complicado detallar en su totalidad.

Esta base ha servido como cimiento, al que se le suma el posterior análisis en disciplinas adicionales como el deep learning, el análisis de sentimientos y las técnicas de procesamiento natural de lenguaje. Estos componentes crean un contexto adecuado para la realización de mi trabajo final de máster.

1.3. Objetivos

El objetivo principal de este trabajo es aplicar técnicas de machine learning para la clasificación de sentimientos en textos. Los sentimientos que se valoran son: positivo, neutral y negativo. Se busca mejorar la precisión y el rendimiento de los modelos mediante la optimización de parámetros, analizando la efectividad de diferentes métricas de evaluación en la tarea de clasificación. Para conseguirlo, se listan los objetivos específicos que se quieren alcanzar:

- Conocer los fundamentos teóricos en los que se basa el análisis de lenguaje y su contexto en el mundo científico.
- Estudiar a nivel teórico las herramientas y algoritmos de machine learning y deep learning cuando se trabaja con grandes cantidades de datos.

- Distinguir cómo funciona un modelo de lenguaje, analizando sus etapas y la manera en que se entrena y evalúa.
- Revisar el estado del arte actual del análisis de sentimiento, y poder trabajar para mejorar modelos existentes.
- Comprender las técnicas de machine learning empleadas para poderlas aplicar en un contexto adecuado.
- Diseñar modelos que, tras ser entrenados, comprendan conjuntos de texto y predigan sentimientos.
- Entender los resultados, saberlos interpretar y poder sacar conclusiones de ellos.
- Programar y diseñar los modelos descritos mediante un lenguaje de programación como Python, manejando correctamente los conjuntos de datos necesarios.
- Modificar y optimizar los modelos para obtención de mejores resultados.

1.4. Estructura de la memoria

Esta memoria tiene como objetivo presentar de manera detallada y descriptiva el trabajo realizado en este proyecto, así como los resultados obtenidos y las conclusiones finales. La memoria se estructura en 4 capítulos: el primero se ha dedicado a introducir el contexto y los objetivos del trabajo, el segundo a explicar las técnicas empleadas, el siguiente a explicar qué cosas se han realizado para la resolución del trabajo, y el último que resume las conclusiones obtenidas, posibles trabajos futuros y aplicaciones. Al final se adjuntan los anexos que complementan la memoria.

Tras el capítulo introductorio, en el capítulo 2 se desarrolla el marco teórico del trabajo, explicando desde conceptos básicos en el análisis de sentimiento (como modelo del lenguaje) hasta los métodos y las técnicas más sofisticadas que se han llegado a utilizar. También se echa un vistazo a los modelos de lenguaje empleados, así como las bases de datos que se han utilizado.

En el capítulo 3 se hace un desarrollo de toda la parte práctica del trabajo, empezando hablando del trabajo previo con el modelo FinBERT. También se muestran los resultados importantes de los modelos que se han ido obteniendo, explicando cómo se aplican los métodos y se utilizan las bases de datos. Se obtiene un clasificador de análisis de sentimiento del mundo financiero.

Por último, en el capítulo 4 se exponen las conclusiones obtenidas a raíz de los objetivos planteados, los resultados obtenidos y líneas futuras de trabajo, además de detallar las posibles aplicaciones.

En los anexos se incluye parte del código de programación desarrollado para este trabajo. En el anexo A.1 se adjuntan las funciones principales implementadas en Python para la creación de matrices; en el anexo A.2 el proceso llevado a cabo para hacer las PCA; en el anexo A.3 se detalla cómo se realiza la creación de los clasificadores, y en el anexo A.4 se muestran las funciones empleadas para la implementación del algoritmo de optimización RR-PSO.

Capítulo 2

Modelos de lenguaje y técnicas de clasificación

Se ha dedicado este capítulo con el objetivo de ofrecer un contexto general sobre el objeto de estudio y definir la terminología y métodos específicos que se utilizarán en apartados posteriores. Se explica también cómo se crea un modelo de lenguaje y sus métricas de evaluación.

2.1. Presentación del problema

En el contexto del análisis de sentimientos y la interpretación de noticias financieras, surge la necesidad de desarrollar modelos capaces de comprender y clasificar las emociones expresadas en el lenguaje humano. El presente trabajo se centra en la creación de un clasificador de sentimientos para noticias financieras, con el objetivo de automatizar la identificación y clasificación de las reacciones emocionales asociadas a eventos de temática financiera.

El problema que aborda este trabajo radica en la dificultad de interpretar y categorizar las opiniones y actitudes presentes en textos financieros de manera eficiente y precisa. La volatilidad y complejidad de los mercados financieros exigen una comprensión ágil de las noticias, pero esta tarea se torna desafiante debido a la diversidad de expresiones y matices en el lenguaje.

La formulación matemática de este problema se traduce en la creación de un modelo de lenguaje capaz de codificar el contenido de las noticias en representaciones numéricas y, posteriormente, clasificar estas representaciones en tres categorías de sentimiento: positivo, negativo o neutral. Matemáticamente, esto implica la implementación de un proceso de codificación y un clasificador multiclase que permita asignar una etiqueta de sentimiento a cada noticia.

La resolución de este problema involucra tanto técnicas de procesamiento del lenguaje natural (NLP) como conceptos de aprendizaje automático supervisado. A través de la formulación y desarrollo de un modelo de lenguaje personalizado, se busca ofrecer una solución efectiva para la clasificación automatizada de sentimientos en noticias financieras, permitiendo una toma de decisiones más informada y ágil en el ámbito económico. En el proceso de creación de este modelo, se explora una variedad de técnicas y algoritmos avanzados. Estas herramientas están diseñadas específicamente para descubrir patrones y relaciones dentro de las palabras y frases presentes en las noticias financieras.

En los siguientes apartados, se explorarán en detalle las etapas de codificación, entrenamiento de varios modelos y su evaluación, así como los resultados obtenidos en la tarea de clasificación de sentimientos en noticias financieras.

2.2. Creando un modelo de lenguaje en machine learning

En este proyecto, el enfoque se centra en la creación de un modelo de clasificación de noticias financieras, es decir, en la creación un modelo que tome una oración y tome como resultado el sentimiento de la oración, si es positiva, negativa o neutral. Esto se logra a través de unos procesos que se pueden agrupar en dos: la codificación de la frase a vector numérico y la clasificación de ese vector a uno de los sentimientos.

Codificación de la oración

Como se ha dicho, el primer objetivo es obtener una representación numérica que pueda ser procesada por el modelo. Este proceso consta de varias etapas secuenciales que aseguran que la información semántica y contextual se preserve de manera eficiente.

En primer lugar, se realiza la llamada tokenización, que es el proceso donde la oración se divide en unidades más pequeñas conocidas como *tokens*. Cada *token* representa una palabra específica, una parte de ella (si esta es larga) o un componente léxico dentro de la oración. Una vez que la oración ha sido tokenizada, a cada *token* se le asigna un número único, conocido como índice.

Después de ello, cada índice es representado como un vector numérico, llamado *embedding*. Los *embeddings* son vectores multidimensionales que codifican la semántica y la relación contextual de las palabras en el espacio numérico. La colección completa de *embeddings* resultante se introduce en el *encoder*. El *encoder* es una arquitectura compuesta por múltiples capas, cada una de las cuales realiza operaciones específicas para generar representaciones ricas y contextuales de los *tokens*. Estas representaciones vectoriales se actualizan en cada capa del *encoder*, sin modificar su dimensión, permitiendo que el modelo incorpore gradualmente información contextual más profunda y comprensiva.

El paso por el *encoder* no es la última parte, y es que el objetivo es obtener un único vector que capture la esencia de toda la frase. Para lograr esto, hay varias posibilidades; exploraremos las dos principales. La primera es calcular la media de cada componente en los *embeddings* generados para todos los *tokens* en la oración. Esta técnica promedia las características semánticas de todos los *tokens*, creando así un vector que representa la información general de la frase. La segunda opción implica una tokenización especial de la oración: la adición de dos *tokens* especiales llamados [CLS] y [SEP]. El *token* [CLS] se coloca al comienzo de la oración y el *token* [SEP] al final. Luego, se obtiene la representación vectorial del *token* [CLS], que para los procesos de clasificación de sentimientos, además de otras tareas como predicciones de textos, encapsula la esencia global de la oración.

Clasificación del sentimiento con un clasificador multiclase

En esta etapa, el vector de representación numérica de la oración, que fue creado en la fase de codificación, se introduce en un clasificador multiclase. Este clasificador está diseñado específicamente para asignar a la oración una de las tres categorías de sentimiento posibles: positivo, negativo o neutral.

La operación de clasificación es una tarea de aprendizaje automático supervisado, en la que el clasificador busca aprender patrones y relaciones en los datos de entrenamiento para poder generalizar y realizar predicciones precisas en nuevos ejemplos. El vector de representación de la oración se pasa a través del clasificador, que consiste en una red neuronal con una o más capas densamente conectadas. Cada capa realiza operaciones matemáticas y aplicaciones de funciones de activación para transformar la entrada y extraer características relevantes.

La salida del clasificador, en el caso de análisis de sentimiento, suele tener tres valores de probabilidad, cada uno asociado a una categoría de sentimiento específica: positivo, negativo y neutral. Estos valores de probabilidad indican la confianza del modelo en su predicción para cada categoría. Un valor

más alto de probabilidad para una categoría en particular sugiere que el modelo está más seguro de que la oración pertenece a esa categoría en términos de sentimiento.

2.2.1. Etapas de un modelo de lenguaje

Se ha hablado de que los modelos de lenguaje tienen la tarea de comprender y generar lenguaje humano de manera automática, y que para ello necesitan de ejemplos con los que puedan aprender los patrones necesarios para poder hacer un buen análisis de sentimiento. Estos ejemplos serán recogidos en un conjunto de datos, que será un conjunto de frases ya etiquetadas con su sentimiento que se le enseña al modelo.

El modelo se entrena utilizando un conjunto de datos como referencia, por lo tanto, es esencial disponer de uno suficientemente grande para que el modelo pueda aprender de manera efectiva. Este conjunto de datos se divide en tres partes distintas: el conjunto de entrenamiento, que generalmente abarca la mayor parte de los datos, representando aproximadamente el 70 % del total; el conjunto de prueba, que comprende la mayoría de los datos restantes; y finalmente, el conjunto de validación. La división de los datos generalmente se realiza de manera aleatoria para evitar cualquier sesgo y garantizar que el modelo se evalúe de manera imparcial.

La división del conjunto de datos en tres subconjuntos se debe a que el proceso de construcción de un modelo de lenguaje implica tres etapas distintas, y para cada una de estas etapas se utiliza un subconjunto de datos diferente. Aunque en ocasiones no se incluye un conjunto de validación porque esta fase no se realiza, las tres etapas que generalmente se llevan a cabo son:

1. Entrenamiento (*train*): el objetivo es ajustar los parámetros del modelo para que este pueda capturar y aprender los patrones y estructuras del lenguaje. Durante el entrenamiento, el modelo se expone a secuencias de palabras y se le enseña a predecir la probabilidad de que elija cada sentimiento. Para ajustar los pesos del modelo, se utilizan algoritmos de optimización que minimizan una función de pérdida o que maximizan la precisión de las predicciones. Durante esta etapa, el modelo aprende a asociar patrones específicos con palabras y a capturar la dependencia contextual entre ellas.

2. Validación: sirve para ajustar los hiperparámetros del modelo. Los hiperparámetros son configuraciones que no se aprenden durante el entrenamiento, pero que afectan el rendimiento del modelo, como la tasa de aprendizaje o la profundidad de los árboles. La validación se utiliza para encontrar la combinación óptima de hiperparámetros que maximice el rendimiento del modelo en datos no vistos. Se ajustan los hiperparámetros y se evalúa el rendimiento del modelo en el conjunto de datos de validación, iterando este proceso hasta obtener la configuración más adecuada.

3. Prueba (*test*): una vez finalizado el entrenamiento, se evalúa el rendimiento del modelo. Como el conjunto de datos utilizado no se ha usado todavía, permite medir cómo se generaliza el modelo a nuevas muestras de texto. Durante la prueba, el modelo realiza predicciones en el conjunto de datos de prueba y se comparan con las respuestas reales. Se calculan diversas métricas de evaluación, que ayudan a comprender qué tan bien el modelo generaliza y realiza predicciones precisas en datos no vistos previamente.

2.2.2. Métricas de evaluación

Las métricas son herramientas fundamentales para evaluar la calidad y el rendimiento de un modelo de lenguaje. A continuación, se profundizará en varias de las principales métricas con sus fórmulas correspondientes. Se definen las métricas para problemas binarios.

- Matriz de confusión: es una tabla que muestra el número de predicciones correctas e incorrectas realizadas por el modelo para cada clase de salida. Tiene la siguiente forma:

		Predicho	
		Positivo	Negativo
Real	Positivo	TP	FN
	Negativo	FP	TN

Cuadro 2.1: Matriz de Confusión con dos clases.

En esta matriz, las filas representan las clases reales y las columnas representan las clases predichas. Los elementos en la diagonal principal representan los verdaderos positivos, es decir, las instancias que fueron correctamente clasificadas. Los elementos fuera de la diagonal principal representan las instancias que fueron incorrectamente clasificadas como pertenecientes a una clase diferente.

- Exactitud (*accuracy*): es una métrica que mide la proporción de predicciones correctas realizadas por el modelo sobre el total de predicciones. Se calcula dividiendo el número de predicciones correctas (TP + TN) entre el número total de predicciones (TP + TN + FP + FN).

$$Accuracy = \frac{\text{Verdaderos Positivos} + \text{Verdaderos Negativos}}{\text{Total de Predicciones}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- Precisión: indica la proporción de predicciones positivas que son realmente positivas. Se calcula dividiendo el número de verdaderos positivos (TP) entre la suma de los verdaderos positivos y los falsos positivos (FP).

$$\text{Precisión} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

La precisión es especialmente relevante cuando se desea minimizar los falsos positivos, es decir, las instancias que fueron incorrectamente clasificadas como positivas.

- Sensibilidad (*Recall*): se define como la proporción de ejemplos positivos que son correctamente clasificados como positivos en relación con el total de ejemplos positivos reales.

$$Recall = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

La sensibilidad es particularmente útil cuando se desea minimizar los falsos negativos.

- *F1-score*: combina la precisión y el recall para proporcionar una medida global del rendimiento del modelo. Se calcula como la media armónica de la precisión y el recall.

$$F1\text{-}score = 2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FN} + \text{FP}}$$

El F1-score tiene en cuenta tanto la capacidad del modelo para evitar clasificar incorrectamente como positivas las instancias negativas (precisión) como su capacidad para clasificar correctamente las instancias positivas (sensibilidad). Proporciona una medida equilibrada del rendimiento del modelo.

- Pérdida de entropía cruzada (*Cross Entropy Loss*): es una medida de la discrepancia entre la distribución de probabilidad predicha por el modelo y la distribución de probabilidad real de los datos de entrenamiento. Se utiliza comúnmente en tareas de clasificación y se calcula mediante la siguiente fórmula:

$$\textit{Cross Entropy Loss} = -\sum_{i=1}^{N}\sum_{j=1}^{C} \omega_j \cdot y_j(i) \cdot \log(\hat{y}_j(i)),$$

donde N es el número de datos predichos, C es el número de clases, ω_j es el peso asociado a la clase j (para evitar el desequilibrio en la distribución de las clases en los datos o dar más importancia a una clase o a otra), $y_j(i)$ es un indicador de si la instancia i es de la clase j y $\hat{y}_j(i)$ es la probabilidad predicha por el modelo para la clase j y la predicción i. La pérdida de entropía cruzada se utiliza durante el entrenamiento para evaluar qué tan bien se ajusta el modelo a los datos y se busca minimizar esta pérdida para mejorar la calidad del modelo.

Por otro lado, este trabajo tiene tres clases: "Positivo", "Neutral" y "Negativo", y por tanto hay que adaptar estas métricas a un problema multiclase, no binario. Se tiene por tanto la siguiente matriz de confusión:

		Predicho		
		Positivo	Neutral	Negativo
Real	Positivo	C_1	C_2	C_3
	Neutral	C_4	C_5	C_6
	Negativo	C_7	C_8	C_9

Cuadro 2.2: Matriz de Confusión con tres clases.

En primer lugar, el *accuracy* va a ser igual que antes, es decir, número de predicciones correctas entre el número total de predicciones.

$$\textit{Accuracy} = \frac{C_1 + C_5 + C_9}{C_1 + C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8 + C_9}$$

La pérdida de entropía cruzada también tiene la misma fórmula, ya que la se ha generalizado para un número de clases C. Las fórmulas que se tienen que adaptar son la precisión, la sensibilidad y el F1-score para cada una de las clases. Se puede explicar este ajuste de las fórmulas con la clase "Positivo", aunque se haría de manera similar con el resto de clases.

Para empezar, la precisión es el número de verdaderos positivos entre la suma de verdaderos positivos y falsos positivos. Los verdaderos positivos para la clase "Positivo" son los que son de esta clase y están bien predichos (C_1); y los falsos positivos para la clase "Positivo" son los que se han predicho como "Positivo" pero en realidad no lo son ($C_4 + C_7$).

Los falsos negativos son los que en realidad eran "Positivo" pero no se han predicho así ($C_2 + C_3$); y como la sensibilidad es la proporción entre verdaderos positivos y la suma de verdaderos positivos y falsos negativos, ya la tenemos. Por otro lado, los verdaderos negativos serán por tanto los que se han dicho correctamente que no eran "Positivo" ($C_5 + C_6 + C_8 + C_9$).

Por último, el F1-score de cada clase se halla con la precisión y la sensibilidad de esa misma clase.

2.3. Procesos y técnicas de clasificación

A continuación la descripción de los procesos y métodos empleados en el trabajo.

2.3.1. Análisis de componentes principales (PCA)

El análisis de componentes principales (o PCA, por las siglas en inglés de *principal component analysis*) es una técnica de reducción de dimensionalidad utilizada en el análisis de datos y el aprendizaje automático. Su objetivo es transformar un conjunto de datos con muchas variables (dimensiones) en un

conjunto de datos con menos variables, manteniendo la mayor parte de la variabilidad en los datos originales. La PCA es particularmente útil cuando se trabaja con conjuntos de datos de alta dimensionalidad y se desea simplificar su representación sin perder demasiada información importante.

El proceso de PCA se inicia después de que las frases del conjunto de datos hayan pasado por el *encoder* y tengan asignado su vector correspondiente. En este punto, cada frase se representa como un vector en un espacio de alta dimensionalidad, donde cada dimensión corresponde a una característica.

La PCA se basa en la proyección de los datos en nuevas dimensiones que se definen por las componentes principales. Estas son combinaciones lineales de las variables originales que capturan la máxima variabilidad en los datos. Estas componentes principales se ordenan en función de su importancia, de modo que las primeras componentes capturan la mayor variabilidad y las últimas componentes capturan la variabilidad residual. Seleccionar un número adecuado de componentes principales implica un equilibrio entre reducir la dimensionalidad y mantener suficiente información para representar adecuadamente los datos.

Fundamento matemático

El cálculo de las componentes principales se realiza de la siguiente manera. Supóngase que existe una muestra con n individuos cada uno con p variables. Sean estas variables $(X_1, ..., X_p)$. Se supone que estas variables están estandarizadas, de manera que todas las variables estén en las mismas unidades. Para ello se restaría cada variable por su media, para después dividir la diferencia por su desviación típica.

Cada componente principal (Y_i, para todo $i = 1, ..., p$) se obtiene por combinación lineal normalizada de las variables originales, es decir, se tiene que:

$$Y_i = \phi_{1j} \cdot X_1 + \phi_{2j} \cdot X_2 + ... + \phi_{pj} \cdot X_p,$$

siendo los coeficientes ϕ_{ij}, para todo $i, j = 1, ..., p$, números reales que cumplen

$$\sum_{i=1}^{p} \phi_{ij} = 1.$$

Los términos ϕ_{ij} reciben en el nombre de *loadings* y son los que definen a la componente. Estos pueden interpretarse como la importancia que tiene cada variable i en cada componente j. Por otra parter, si denotamos ϕ_i al vector que contiene todos los *loadings* de la componente i-ésima, para todo $i = 1, ..., p$, y si $\mathbf{X}$ es el vector formado por las variables $(X_1, ..., X_p)$, entonces la componente principal i-ésima también se puede expresar de esta manera: $Y_i = \phi_i^T \cdot \mathbf{X}$, y la restricción de esta otra: $\phi_j^T \phi = 1$.

Teóricamente, en primer lugar se trata de encontrar los $\phi_{11}, ...\phi_{p1}$ de forma que se maximice la varianza del primer componente principal.

$$Var(Y_1) = Var(\phi_1^T \cdot \mathbf{X}) = \phi_1^T \cdot \Sigma \cdot \phi_1,$$

donde Σ es la matriz de varianzas-covarianzas. Esta es la función que se quiere maximizar, por lo tanto se plantea como el siguiente problema de optimización.

$$\begin{cases} max & \phi_1^T \cdot \Sigma \cdot \phi_1 \\ s.a.: & \phi_1^T \phi_1 = 1 \end{cases}$$

Aplicando el método de multiplicadores de Lagrange y con alguna operación algebraica, se ve que la solución tiene que cumplir que

$$(\Sigma - \lambda I_p)\phi_1 = 0.$$

ϕ_1 no es nulo, por lo que es la otra parte la que tiene que ser cero. Es facil ver que λ es un valor propio de Σ con vector propio asociado ϕ_1. Además, Σ es una matriz de dimensión $p \times p$, que tiene p valores

propios. El valor propio que tenga un valor más grande será la solución, ya que representa la varianza de la primer componente principal. Una vez escogido el valor propio, se procede a calcular el vector propio asociado ϕ_1.

Las otras componentes principales se calculan de forma similar, pero teniendo en cuenta que las componentes principales son ortogonales, por tanto, hay que añadir restricciones de no correlación. La variación total del conjunto de datos es la suma de los valores propios de la matriz, $\sum_{i=1}^{p} \lambda_i$.

2.3.2. Algoritmo XGBoost

XGBoost (*Extreme Gradient Boosting*) es un algoritmo de aprendizaje automático supervisado basado en *Gradient-Boosted Decision Trees* (GBDT) propuesto por Chen y Guestrin (2016). Se ha vuelto muy popular debido a su eficacia y versatilidad en una amplia gama de problemas de aprendizaje automático, especialmente en tareas de clasificación y regresión. Su éxito se encuentra en su capacidad para crear modelos de alta precisión al combinar árboles de decisión débiles como sus elementos base.

La idea clave de XGBoost se encuentra el concepto de *boosting*, una estrategia de ensamble que combina múltiples modelos débiles para formar un modelo más fuerte y preciso. La idea esencial del *boosting* es corregir los errores del modelo anterior mediante la construcción de nuevos modelos que se enfoquen en las áreas problemáticas. Esto lo logra mediante la construcción secuencial de árboles de decisión.

XGBoost inicia con un modelo inicial simple, comúnmente un árbol de decisión con una sola hoja o una predicción constante. Esta predicción básica sirve como línea de base para las iteraciones futuras. En cada iteración, se calculan los residuales, que representan los errores entre las predicciones acumuladas del modelo actual y los valores reales. Estos residuales cuantifican dónde el modelo necesita mejoras. Luego, se ajusta un nuevo árbol de decisión a los residuales calculados. Este árbol se construye específicamente para capturar y corregir los errores que el modelo anterior no pudo manejar. Las predicciones de este nuevo árbol se agregan a las predicciones acumuladas del modelo, refinando gradualmente la precisión a medida que se abordan los errores.

El modelo XGBoost utiliza un método de entrenamiento aditivo para optimizar la función objetivo, lo que significa que el proceso de optimización del paso posterior depende del resultado de su paso anterior. La función objetivo del modelo en el paso t puede expresarse como:

$$\text{obj}(t) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t) + constante, \tag{2.1}$$

donde x_i es el i-ésimo ejemplo de entrenamiento, $f_t(x_i)$ es la contribución del t-ésimo árbol de decisión para la predicción del i-ésimo ejemplo x_i, y_i representa el valor real para el i-ésimo ejemplo de entrenamiento, $\hat{y}_i^t$ se refiere a la predicción de la ronda t para el i-ésimo ejemplo, l representa el término de pérdida de la ronda t, $constante$ es un término constante, y Ω es el término de regularización del modelo, expresado como:

$$\Omega(f_t) = \gamma \cdot T_t + \frac{\lambda}{2} \sum_{j=1}^{T} w_j^2. \tag{2.2}$$

Tanto γ como λ son hiperparámetros, esto es, parámetros que se ajustan antes de entrenar un modelo de aprendizaje automático y que influyen en su comportamiento y rendimiento.

Si se realiza una expansión de Taylor de segundo orden en (2.1) se obtiene:

$$\text{obj}(t) = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constante, \tag{2.3}$$

donde g es la primera derivada, y h es la segunda derivada. Se pueden describir como:

$$g_i = \frac{\partial l(y_i, \hat{y}_i^{t-1})}{\partial \hat{y}_i^{t-1}} \tag{2.4}$$

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{t-1})}{\partial (\hat{y}_i^{t-1})^2} \tag{2.5}$$

Sustituyendo (2.2), (2.4), (2.5) en (2.3) y tomando la derivada, las soluciones son:

$$w_j^* = -\frac{\sum g_i}{\sum h_i + \lambda}$$

$$obj^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum g_i)^2}{\sum h_i + \lambda} + \gamma \cdot T$$

donde obj* representa la puntuación de la función de pérdida. Cuanto menor sea la puntuación, mejor será la estructura del árbol. w_j^* se refiere a la solución de los pesos.

Es muy importante hacer un buen ajuste de los hiperparámetros que dominan el algoritmo, tales como la tasa de aprendizaje, el número de árboles o la profundidad de los mismos. Una búsqueda exhaustiva de estos hiperparámetros es clave para realizar una implementación eficiente y precisa del algoritmo. En concreto, tienen especial interés los siguientes hiperparámetros, debido a su importancia en la clasificación. La información de cada uno está en la tabla 2.3.

Estimador	**Rango valores**	**Explicación**
n_estimators	$\mathbb{N}$	Número de iteraciones y de árboles.
max_depth	$\mathbb{N} \cup \{0\}$	Profundidad máxima de los árboles.
learning_rate	$[0,1]$	Cuánto se corrige en cada iteración (valores muy grandes hacen overfitting).
gamma	$[0,\infty)$	Controla cuándo se hace una partición en una hoja (si la ganancia es mayor que *gamma* se hace).
min_child_weight	$[0,\infty)$	Peso mínimo requerido para crear un nuevo nodo en el árbol.
subsample	$(0,1]$	Proporción de muestras utilizadas para entrenar cada árbol.
colsample_bytree	$(0,1]$	Proporción de características usadas para entrenar cada árbol.

Cuadro 2.3: Principales hiperparámetros del algoritmo XGBoost.

2.3.3. Algoritmo k-NN

El algoritmo k-NN (k-*Nearest Neighbors* o k vecinos más cercanos) es una técnica del campo del aprendizaje automático supervisado, cuyo enfoque se basa en el principio de que ejemplos similares tienden a compartir características similares. Es una técnica intuitiva y fácil de entender, que se utiliza tanto para clasificación como para regresión en tareas de análisis de datos y predicción.

Dado un conjunto de datos de entrenamiento con ejemplos etiquetados (en el caso de clasificación) o con valores objetivos conocidos (en el caso de regresión), el algoritmo k-NN opera de la siguiente manera. Para un nuevo punto de datos de entrada, se calcula la distancia con respecto a todos los puntos del conjunto de entrenamiento. Esto se hace utilizando una métrica de distancia, que puede ser la distancia

euclidiana, la distancia de Manhattan, la distancia de Chebychev, entre otras. La elección de la métrica depende del problema y de las características de los datos. A continuación, se seleccionan los k puntos de entrenamiento más cercanos al nuevo punto, según la métrica de distancia. Estos k vecinos conforman el vecindario del punto en cuestión. El último paso, en el caso de clasificación es asignar la etiqueta más común entre los k vecinos más cercanos al punto de datos; y para la regresión, se estima el valor objetivo del nuevo punto basándose en los valores de los k vecinos más cercanos.

El parámetro k es crucial en el algoritmo k-NN. Un valor pequeño de k tiende a dar un modelo más sensible a los datos, lo que puede llevar a sobreajuste si el ruido está presente. Por otro lado, un valor grande de k suaviza las decisiones y puede llevar a un bajo sesgo pero una mayor varianza. La elección adecuada de k es fundamental para obtener un equilibrio entre la precisión y la capacidad de generalización del modelo.

2.3.4. Redes neuronales

Una red neuronal es un modelo computacional inspirado en el funcionamiento del cerebro humano, diseñado para aprender y reconocer patrones complejos a partir de datos. Siguiendo esta imitación, consiste en un conjunto interconectado de unidades o nodos llamados neuronas artificiales, organizadas en capas, donde cada neurona procesa información y transmite señales a otras neuronas a través de conexiones.

Una red neuronal contará con al menos con dos capas, la capa de entrada y la de salida, y además podrán existir también capas ocultas entre ambas. La información fluye desde la capa de entrada (*inputs*), donde se ingresan los datos de entrada. El proceso sigue por las capas ocultas, donde se lleva a cabo el procesamiento y la extracción de características relevantes. Finalmente, llega la información a la capa de salida, donde se obtiene el resultado o predicción (*output*).

Para aplicar una red neuronal a un problema de clasificación, primero se debe diseñar la arquitectura de la red, que incluye el número de capas ocultas, la cantidad de neuronas en cada capa, la función de activación de cada neurona, y otros hiperparámetros. Luego, se debe preparar el conjunto de datos de entrenamiento y de prueba, donde cada muestra debe estar etiquetada con su respectiva clase.

Cada conexión que enlaza dos neuronas en una red neuronal tiene un valor llamado peso. Estos reflejan la relevancia de la conexión entre esas neuronas. Para determinar estos pesos, la red neuronal se somete a un proceso de aprendizaje o entrenamiento utilizando un conjunto de datos. Durante este proceso, la red ajusta gradualmente los pesos para reconocer patrones y características que sean relevantes para una tarea específica, como la clasificación en clases o la predicción de valores.

El proceso de actualización de pesos en una red neuronal, durante el entrenamiento, implica la optimización de los parámetros de la red para minimizar la función de pérdida. Comienza con los datos de entrada atravesando las capas ocultas hasta producir una predicción. Luego, se calcula la diferencia entre la predicción y el valor real mediante la función de pérdida. A continuación, mediante la retropropagación del error, se determina cómo cada peso contribuyó al error total. Utilizando un algoritmo de optimización, como el descenso de gradiente, se ajustan gradualmente los pesos en la dirección que reduce la pérdida. Este proceso iterativo actualiza los pesos en busca de una convergencia que resulte en un modelo mejor ajustado a los datos de entrenamiento. Se repite el proceso de iteración hasta que se alcance un criterio de parada, como un número máximo de iteraciones o cuando la pérdida converge a un valor mínimo.

Un elemento que tiene especial interés en las redes neuronales (como veremos más adelante con el modelo BERT), es la atención. Es una herramienta que permite al modelo enfocarse en ciertas partes específicas mientras le da menos importancia a otras, mejorando la captura de patrones relevantes. Tener varios mecanismos de atención en la red neuronal permite que el modelo capte distintas relaciones y dependencias en los datos de manera paralela y más efectiva, ganando eficiencia en el modelo, además

de tiempo.

Otros elementos esenciales en una red neuronal son las funciones de activación. Se asigna una función de activación a cada capa de una red neuronal, incluyendo las capas ocultas y la capa de salida. Esta función de activación de una neurona, que puede ser no lineal, se aplica a la suma ponderada de las entradas y los pesos de la neurona para producir su salida. Sin funciones de activación, las capas sucesivas de la red simplemente realizarían transformaciones lineales de las entradas, lo que limitaría su capacidad de aprendizaje y representación.

Existen varios tipos de funciones de activación utilizados en las redes neuronales. Algunos de los ejemplos más importantes son la función sigmoide, la función ReLU (*Rectified Linear Unit*), la función tangente hiperbólica o la función *Softmax*. Cada tipo de función de activación tiene sus propias propiedades y características, y la elección de la función adecuada depende del tipo de tarea y de la arquitectura de la red neuronal que se esté utilizando.

Por ejemplo, para las capas ocultas es bastante común usar la función ReLU, que para una entrada x se define como:

$$ReLU(x) = \text{máx}(0, x).$$

La función de activación *Softmax*, en cambio, es comúnmente utilizada en la capa de salida de una red neuronal para problemas de clasificación multiclase. Dada una entrada a una neurona $\mathbf{x}$, vector k-dimensional, se define la función de activación *Softmax* como una función S tal que $S : \mathbb{R}^k \longrightarrow [0,1]^k$, cumpliendo que, para todo $i = 1, ..., k$:

$$S_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}.$$

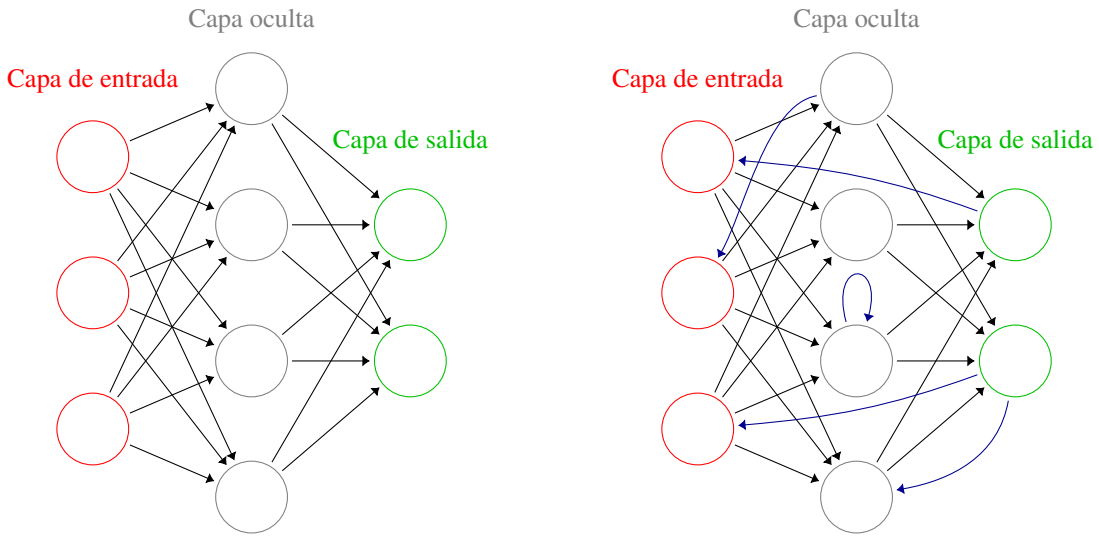
En la figura 2.1 - (a) se puede ver un ejemplo de las conexiones de una red neuronal con 2 capas ocultas, cada una de ellas con 3 neuronas, donde la capa de entrada tiene 2 neuronas y la de salida 1. Las flechas indican las conexiones neurona a neurona, y a cada una de ellas le irá asignado un valor, el peso.

Redes neuronales recurrentes

Un tipo concreto de red neuronal el cual también se va a trabajar es el de las redes neuronales recurrentes (RNN). Estas son un tipo de arquitectura de red neuronal diseñada para trabajar con secuencias de datos, como texto, audio o series temporales. A diferencia de las redes neuronales tradicionales, que tratan cada entrada de manera independiente, las RNN tienen la capacidad de capturar relaciones y dependencias en datos secuenciales al mantener una especie de "memoria" interna a medida que procesan cada elemento de la secuencia.

Una red neuronal normal (*Feedforward Neural Network*) procesa la entrada a través de capas de neuronas interconectadas, donde cada conexión entre neuronas tiene un peso asociado. La entrada fluye a través de estas capas en una sola dirección, desde la capa de entrada hacia la capa de salida, sin retroalimentación. Sin embargo, en una RNN, las neuronas tienen conexiones que forman bucles, lo que les permite mantener y actualizar información a lo largo del tiempo y las iteraciones. El propósito detrás de estas redes es tener la capacidad de disponer de una memoria que pueda retener datos anteriores y permitir consultas repetidas y razonamiento sobre esa información, a fin de emular de manera más precisa el funcionamiento de una red neuronal biológica.

En la figura 2.1 - (b) se puede ver un ejemplo de red neuronal recurrente.



(a) *Feedforward Neural Network*.

(b) Red neuronal recurrente.

Figura 2.1: Ejemplos de redes neuronales con una capa oculta.

2.3.5. Algoritmo PSO

La Optimización por Enjambres de Partículas, conocida como PSO por sus siglas en inglés (*Particle Swarm Optimization*), es una estrategia de optimización global que se basa en la observación del comportamiento de los insectos en la naturaleza. Imagina un enjambre de abejas buscando polen en un campo: cada abeja se moverá por el espacio en busca de polen, recordando los lugares por los que ha pasado y si ha estado cerca o lejos de conseguirlo. Mientras tanto, las abejas interaccionan entre si y aprenden de otrs abejas. A final, el conjunto se mueve hacia el lugar del campo donde hay más polen.

De manera similar, en PSO, se utilizan "partículas" que representan soluciones candidatas en un espacio de búsqueda. Cada partícula se mueve por el espacio de búsqueda, y su movimiento está influenciado por su posición actual, la mejor posición histórica que ha encontrado y la mejor posición que cualquier otra partícula ha encontrado. Esto fomenta la cooperación y la exploración conjunta del espacio de soluciones. A medida que las partículas interactúan y se ajustan sus movimientos, el enjambre converge hacia la solución óptima. El proceso se repite hasta que se alcanza un cierto criterio de convergencia o se agota el número máximo de iteraciones.

Esta técnica se describió por James Kennedy y Russell C. Eberhart (1995). Un esquema simplificado del algoritmo PSO podría ser el siguiente:

Algoritmo1: Algoritmo PSO Genérico

```
posicion, velocidad = generados aleatoriamente;
mejor_global = max(evaluar(posicion))
for(cada iteración i){
   for(cada partícula j){
     velocidad_j[i] = actualizar_velocidad(velocidad_j[i-1], mejor_personal,
                                           mejor_global)
     posicion_j[i] = actualizar_posicion(velocidad_j[i], posicion_j[i-1])
     mejor_personal = max{evaular(posicion_j[i]), mejor_personal}
   }
   mejor_global = max{evaluar(posicion), mejor_global}
}
return mejor_global
```

Se puede describir la trayectoria de cada partícula a través de un modelo continuo. Se define la ecuación diferencial que rige el comportamiento de la partícula i como:

$$x_i''(t) + (1-\omega)\,x_i'(t) + (\phi_1+\phi_2)\,x_i(t) = \phi_1\,g(t-t_0) + \phi_2\,l_i(t-t_0),$$

donde ω es la inercia del sistema, ϕ_1 y ϕ_2 dos parámetros vinculados a la aceleración global y local, $g(t)$ es el atractor global, l_i es el atractor local, y t_0 es un parámetro de desfase. Los parámetros ϕ_1 y ϕ_2 se definen como $\phi_1 = r_{i,1}a_g$ y $\phi_2 = r_{i,2}a_l$. Se tiene que a_g y a_l representan las aceleraciones global y local, y $r_{i,j}$ es un número aleatorio, con valores entre 0 y 1, correspondiente a la partícula i. ω, a_g, y a_l son parámetros globales del sistema.

Con la resolución de esta ecuación se calculan, a partir de los valores de posición y velocidad de cada partícula en un instante dado, los valores de posición y velocidad en el instante posterior. Esta resolución pasa por plantear métodos numéricos, y se puede realizar de varias formas. En función de la aproximación numérica utilizada, que proviene de la expansión en serie de Taylor, se puede clasificar a la aproximación de la derivada como diferencia hacia adelante (P), hacia atrás (R) o centrada (C). De esta forma, se tienen 9 variantes distintas del algoritmo dependiendo de cómo se aproximen las dos primeras derivadas: PP-PSO, PC-PSO, RP-PSO...

Algoritmo RR-PSO

En concreto para este trabajo, se va a usar el método RR-PSO, que utiliza la derivada primera y segunda hacia atrás. Se estudiará primero el caso general RR-GPSO y luego concretaremos el proceso en el RR-PSO.

En primer lugar, notar que para una función f (en este problema las funciones con las que planteamos estos métodos numéricos son la función posición x y la función velocidad v), la primera y segunda derivadas hacia atrás en un instante de tiempo t_0 son:

$$f'(t_0) = \frac{f(t_0) - f(t_0-h)}{h}$$

$$f''(t_0) = \frac{f(t_0) - 2f(t_0-h) + f(t_0-2h)}{h^2}$$

Para simplificar la notación, se define: $x = x(t)$, $x^+ = x(t+\Delta t)$, $x^- = x(t-\Delta t)$ y sus equivalentes para la velocidad v. Como $v^+ = \dfrac{x^+ - x}{\Delta t}$, es fácil ver que:

$$x^+ = v^+\Delta t + x \tag{2.6}$$

Que es la posición en el instante posterior que se buscaba. Para el cálculo de la velocidad en el instante posterior v^+, se tiene que conocer el valor de $v'(t+\Delta t)$, y sustituyendo términos y agrupando, se obtiene:

$$v^+ = \frac{\phi_1\,[g(t+\Delta t - t_0) - x]\,\Delta t + \phi_2\,[l_i(t+\Delta t - t_0) - x]\,\Delta t + v}{1 + (1-\omega)\Delta t + \Delta t^2\phi_1 + \Delta t^2\phi_2} \tag{2.7}$$

Finalmente, como se sabe que el algoritmo RR-PSO es un caso particular de la familia RR-GPSO en el que $\Delta t = 1$, basta con sustituir ese valor para conocer las fórmulas (2.6) y (2.7), que se emplean en la actualización de posición y velocidad.

2.4. Modelos y bases de datos empleadas

En la siguiente sección, se detallarán los modelos y bases de datos que han sido empleados en el presente estudio. Para comprender plenamente el alcance y la metodología de la investigación, es fundamental explorar las herramientas y recursos que han servido como base para el análisis y las conclusiones. En esta sección, se describirán los modelos teóricos y técnicos utilizados, así como las bases de datos seleccionadas para la obtención de datos empíricos.

2.4.1. Arquitectura Transformer

La arquitectura Transformer fue propuesta en el artículo *Attention is All You Need* publicado por Vaswani et al. (2017). En este artículo, se presentaba una nueva arquitectura de red neuronal para modelos de lenguaje natural que intentaba mejorar el trabajo de las redes neuronales recurrentes o convolucionales que se habían usado previamente, utilizando el concepto de atención.

Las redes recurrentes y convolucionales que se habían usado hasta entonces eran bastante efectivas, sin embargo, tenían limitaciones cuando procesaban secuencias largas o para recordar información a largo plazo. Además, para entrenar este tipo de redes se necesitaba emplear mucho tiempo y utilizaba muchos recursos.

La arquitectura Transformer aborda estos problemas utilizando múltiples capas de atención para procesar la entrada. Esta atención hace que la red se concentre en partes relevantes de la entrada en lugar de tratarla en su totalidad, mejorando la comprensión y reduciendo el ruido. También integra bloques residuales y normalización de capas para un aprendizaje más eficiente y rápido.

Desde su introducción, la arquitectura Transformer ha sido muy bien recibida en la comunidad de IA y es por eso que se ha aplicado en modelos tan importantes como BERT y GPT-3. Destaca por su efectividad en diversas tareas de procesamiento de lenguaje natural y sigue siendo un área activa de investigación en la comunidad de inteligencia artificial.

Se pueden implementar una gran cantidad de modelos con arquitectura Transformer a Python. Algunas de las bibliotecas que implementan métodos de *machine learning* que son útiles para el trabajo son Hugging Face Transformers, TensorFlow y PyTorch.

2.4.2. Modelos BERT y FinBERT

BERT, que significa *Bidirectional Encoder Representations from Transformers*, es un innovador modelo de lenguaje pre-entrenado desarrollado por Google por los investigadores Devlin et al. (2018). Su enfoque revolucionario radica en su capacidad para comprender y representar el contexto bidireccional de las palabras en un texto, en contraposición a los modelos de lenguaje anteriores, como ELMo y AWD-LSTM, que solo consideraban el contexto previo o posterior.

BERT es en esencia un modelo de lenguaje que consiste en un conjunto de *encoders* Transformer apilados uno encima del otro. Esta comprensión bidireccional del texto la realiza de forma que en lugar de predecir la siguiente palabra, enmascara un 15 % de todos los tokens seleccionados al azar. La ultima capa, que tiene una función de activación *Softmax*, es la que predice estos tokens. Una segunda tarea en la que BERT está entrenado es la predicción de la siguiente oración. Dadas dos oraciones, el modelo predice si estas dos frases realmente se suceden o no.

Por otra parte, en la fase de codificación de la frase, se agregan dos *tokens* indicados por [CLS] y [SEP] al principio y final de la secuencia de *tokens*, respectivamente. Para las tareas de clasificación, incluida la predicción de la siguiente oración, se utiliza el *token* [CLS], como se explica en la creación de modelos de lenguaje.

BERT tiene dos versiones: BERT-base, con 12 capas de *encoders*, tamaño de cada capa de 768, 12 conjuntos de cabezales de atención múltiple y 110 millones de parámetros en total; y BERT-large, con 24 capas de *encoders*, tamaño oculto de 1024, 16 conjuntos de cabezales de atención múltiple y 340 millones de parámetros. Ambos modelos han sido entrenados en *BookCorpus* y *Wikipedia* en inglés, que tienen en total más de 3500 millones de palabras. Se muestra un esquema con un ejemplo de como funciona un modelo de lenguaje que usa arquitectura BERT en la figura 2.2.

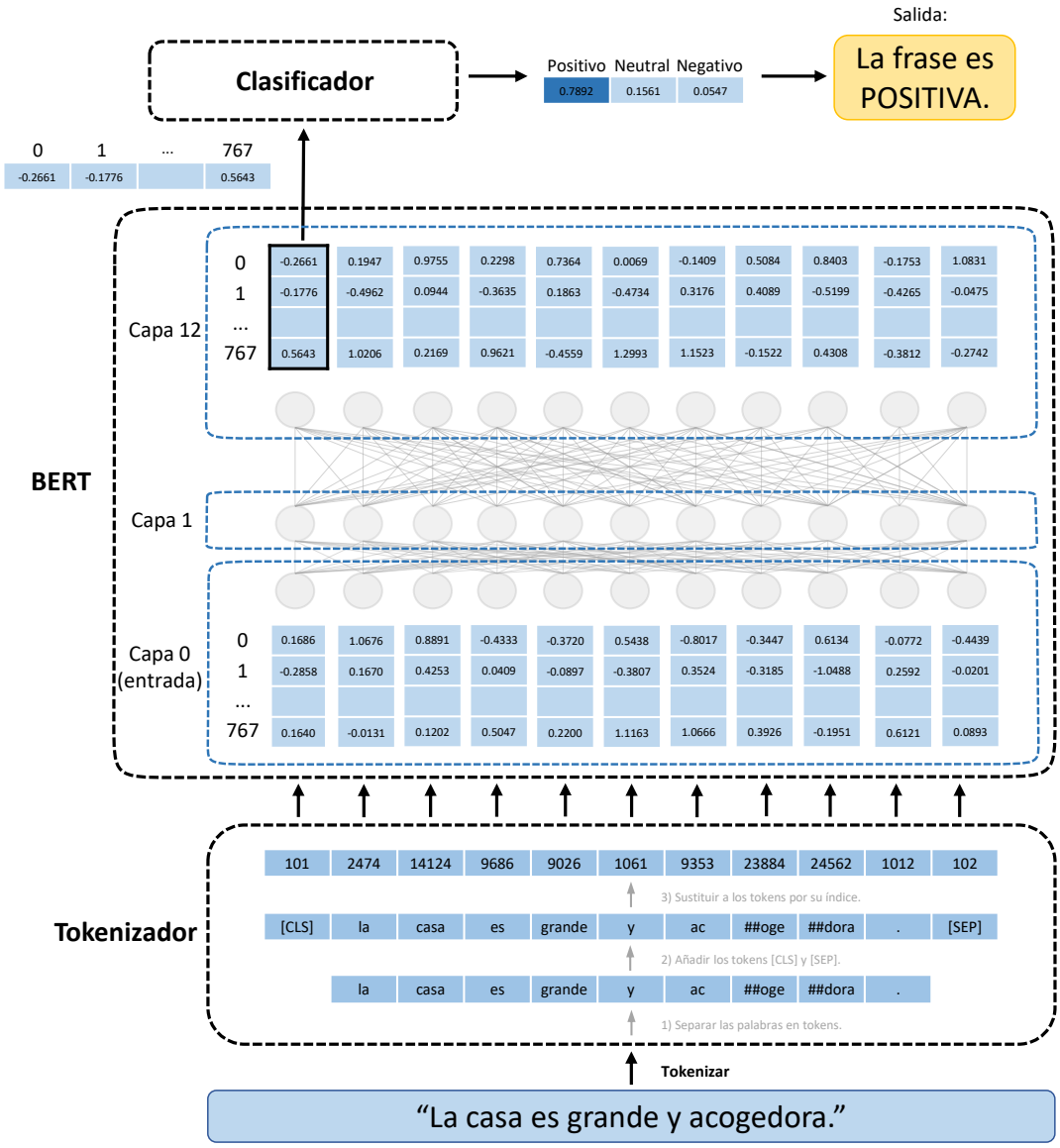


Figura 2.2: Esquema del funcionamiento de un modelo de lenguaje con arquitectura BERT.

Sin embargo, BERT no es completamente adecuado para todos los dominios, ya que su entrenamiento generalizado puede no abordar de manera óptima los matices y la jerga específica de ciertos campos, como las finanzas. Aquí es donde entra en juego FinBERT, un desarrollo posterior que aborda las necesidades específicas del análisis de texto financiero.

FinBERT, como una extensión especializada de BERT, fue introducido por Araci (2019) para abordar estos desafíos en el ámbito financiero. Este modelo se entrena en un corpus de datos financieros para adquirir un conocimiento profundo de la terminología, la jerga y los contextos únicos de las finanzas. Al hacerlo, FinBERT puede comprender mejor este lenguaje específico y analizar textos que hablen sobre

estos temas este dominio.

Un ejemplo destacado de la aplicación de FinBERT es el análisis de sentimiento en noticias financieras. Este modelo puede determinar si un artículo o titular es positivo, negativo o neutral, lo que puede proporcionar información clave para los inversores y tomadores de decisiones en el ámbito económico. Además de la clasificación de sentimiento, FinBERT también puede identificar palabras clave que se asocian con cada sentimiento, lo que brinda una visión detallada de las tendencias y los factores que influyen en los mercados.

2.4.3. Financial PhraseBank

Uno de los principales conjuntos de datos utilizado en este trabajo es Financial PhraseBank de Malo et al. 2014. Financial Phrasebank consta de 4838 frases en inglés seleccionadas al azar de noticias financieras encontradas en la base de datos LexisNexis. Luego, estas frases fueron comentadas por 16 personas con experiencia en finanzas y negocios. Se pidió a los expertos que etiquetaran según cómo creían que la información de la oración podría afectar el precio de las acciones de la empresa mencionada. La información podía afectar de manera positiva, negativa o neutral. El conjunto de datos también incluye información sobre el nivel que estaban de acuerdo los expertos para cada oración. La distribución de los niveles de acuerdo y las etiquetas de sentimiento se puede ver en la tabla 2.4.

Nivel de Acuerdo	Positivo	Neutral	Negativo	Total
100 %	25.2 %	61.4 %	13.4 %	2259
75 %-99 %	26.7 %	63.5 %	9.8 %	1189
66 %-74 %	36.7 %	51.0 %	12.3 %	763
50 %-65 %	31.1 %	54.5 %	14.4 %	627
Todas	28.2 %	59.4 %	12.5 %	4838

Cuadro 2.4: Tabla de nivel de acuerdo

2.4.4. Base de datos de tuits financieros

El otro conjunto de datos que se ha empleado ha sido un conjunto de tweets que tienen asociado un sentimiento, obtenido de Kaggle y hecho Rathi y Wallach (2019). Esta base de datos es una recopilación de 28440 tuits en inglés que cuenta con dos columnas: el tuit y su sentimiento asociado (-1 si el sentimiento es negativo, 0 se es neutral y 1 si es positivo). Estas frases corresponden a los mensajes publicados en la plataforma de Twitter y provienen de diversas cuentas de usuarios relevantes en el ámbito financiero. En particular, se han elegido aquellas frases que hacen referencia a una empresa en específico, es decir, que mencionan o se refieren a una compañía en particular. Estas selecciones se han realizado con el propósito de enfocarse en los comentarios y discusiones que están relacionados directamente con aspectos financieros y que involucran a una entidad empresarial en particular.

Los tuits que se utilizan son los publicados antes de la fecha 5 de septiembre de 2019, que son escritos por las siguientes cuentas: “MarketWatch”, “business”, “YahooFinance”, “TechCrunch”, “WSJ”, “Forbes”, “FT”, “TheEconomist”, “nytimes”, “Reuters”, “GerberKawasaki”, “jimcramer”, “TheStreet”, “TheStalwart”, “TruthGundlach”, “Carl_C_Icahn”, “ReformedBroker”, “benbernanke”, “bespokeinvest”, “BespokeCrypto”, “stlouisfed”, “federalreserve”, “GoldmanSachs”, “ianbremmer”, “MorganStanley”, “AswathDamodaran”, “mcuban”, “muddywatersre”, “StockTwits” y “SeanaNSmith”.

La distribución de los sentimientos es muy desigual, parecida al otro conjunto de datos. El porcentaje de tuits positivos es 29,9 %, el de neutrales es 60,9 % y el de negativos 9,1 %.

Capítulo 3

Resolución del trabajo

En este capítulo se presentan el trabajo previo que inspira el desarrollo del trabajo. También se detallan toda la resolución del trabajo, cómo se han creado los modelos, así como las implementaciones de los métodos empleados y resultados que se han ido obteniendo. Todos el código que se ha empleado se ha escrito con lenguaje Python.

3.1. Trabajo previo

Tomando como punto de partida el trabajo realizado para la creación de FinBERT, se ha usado también la base de datos que emplearon para entrenar el modelo, Financial PhraseBank. Por tanto, en primer lugar se debe separar esta base de datos en tres subconjuntos. De momento no se tiene en cuenta el porcentaje de expertos que está de acuerdo en la etiqueta de la frase. Se reserva el 20 % de todas las oraciones como prueba y el 10 % del resto como conjunto de validación. Al final, el conjunto de entrenamiento incluye 3488 frases y el de entrenamiento 970.

Se ejecutan las órdenes del archivo `finbert_training.ipynb` que se puede descargar desde el repositorio `https://github.com/ProsusAI/finBERT`, donde se encuentran los archivos. Tras importar el modelo con la biblioteca Transformers, se procede a entrenar, validar y testear dicho modelo FinBERT, además de tunear los hiperparámetros. Se obtiene una precisión del 86 %, con un valor para la pérdida de entropía cruzada de 0,32. El resto de métricas son las siguientes.

Clase	Precisión	Recall	F1-score	Elementos
Positivo	0,80	0,85	0,82	267
Neutral	0,92	0,85	0,88	575
Negativo	0,77	0,92	0,84	128
Media	0,83	0,87	0,85	970
Media ponderada	0,87	0,86	0,86	970

Cuadro 3.1: Métricas del modelo FinBERT de partida.

Son unos valores bastante buenos, se podría decir que realiza unas buenas predicciones. Partiendo de este modelo y con las herramientas que se han explicado anteriormente, se va a intentar mejorar estos valores, de manera que la clasificación del sentimiento sea aun más precisa.

3.2. Matriz de coordenadas y PCA

Se puede aprovechar este modelo para crear clasificadores más potentes. Para ello, se crea una nueva matriz, que será el conjunto de los 4838 vectores asociados a cada una de las frases de la base de datos

tras su codificación en el modelo FinBERT. Esta matriz, que se llama matriz de coordenadas, es de dimensión 4838×768 ya que la base de datos está formada por 4838 frases y el modelo emplea 768 variables para la codificaión.

La dimensión de esta matriz es muy grande, y podría venir bien reducir su tamaño para aligerar los procesos, siempre y cuando no se pierda información importante en los datos. Es por eso que el siguiente paso que se ha realizado ha sido el de hacer una PCA sobre esta matriz. De esta manera, se puede eliminar algún conjunto de variables que explique poca varianza, para poder procesar la información más rápidamente. En la gráfica 3.1 se muestra la evolución de la varianza explicada acumulada en relación al número de componentes principales empleadas, y marcado en rojo los puntos en donde la varianza acumulada alcanza los valores de 67 %, 80 %, 85 %, 90 % y 95 %.

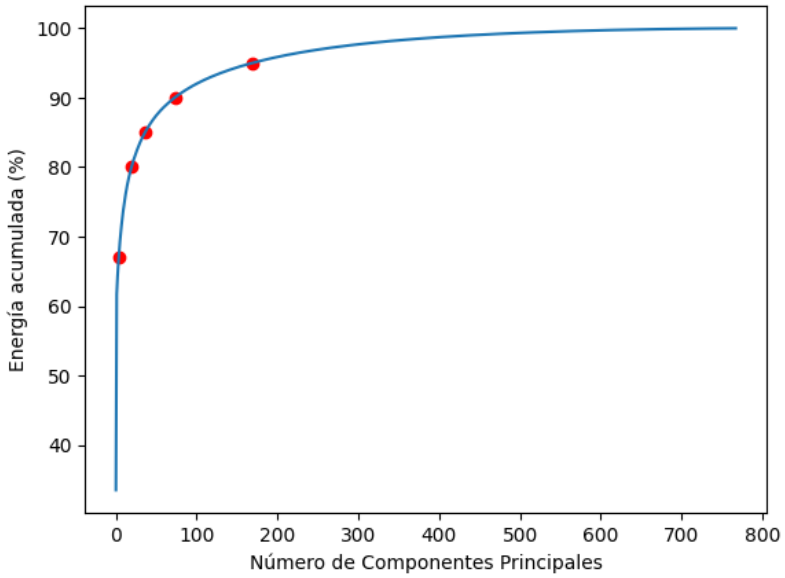


Figura 3.1: Evolución de la varianza explicada con el número de componentes principales.

También se muestra a continuación en la tabla 3.2 el mínimo número de componentes principales que se necesitan para alcanzar el porcentaje de varianza explicada acumulada dado.

Varianza explicada acumulada	67	80	85	90	95	100
Nº de componentes principales	4	20	36	73	169	768

Cuadro 3.2: Matrices de coordenadas según la varianza explicada.

Con esta información, se generan 6 conjuntos de matrices de componentes principales. Cada uno de estos conjuntos, que se denotan como `x67`, `x80`, `x85`, `x90`, `x95` y `x100`, donde el número que tiene cada matriz corresponde a un porcentaje particular de varianza explicada por sus componentes principales. Por ejemplo, `x67` representa el conjunto de componentes principales que explica el 67 % de la varianza total, mientras que `x100` abarca todos los componentes principales y explica el 100 % de la varianza total. Estos conjuntos nos permiten explorar y analizar diferentes niveles de reducción de dimensionalidad en nuestros datos, según cuánta varianza deseemos conservar en cada caso. El código empleado se puede encontrar en los anexos A.1 y A.2.

3.3. Creación de clasificadores

Una vez construídas estas matrices, se utilizan los tres métodos distintos de clasificación que se han explicado anteriormente para realizar el análisis de sentimiento de las frases; estos son, el algoritmo

XGBoost, el algoritmo *k*-NN de vecinos próximos y redes neuronales. El código que se emplea para esta tarea se encuentra en el anexo A.3.

La creación de estos clasificadores también se hace con ayuda de la base de datos Financial PhraseBank. Para estos modelos se ha dividido este conjunto de datos en dos, un 80 % de los datos para el entrenamiento y un 20 % para el testeo o prueba. En algún caso, como se indica, se realiza una validación cruzada también. Se describe a continuación cómo se ha construido cada clasificador.

En cuanto al algoritmo XGBoost, la clave de un buen funcionamiento del mismo es una rigurosa selección de hiperparámetros, que dependiendo de cada modelo o base de datos pueden variar bastante. Por ello, se ha realizado una búsqueda de dos formas distintas. La primera es con GridSearchCV, que ejecuta todas las posibles combinaciones de hiperparámetros que ofrezcas, y RandomizedSearchCV, que busca solamente un número de combinaciones dado, de entre las posibles opciones que ofrezcas. Además, en ambos casos, se realizan técnicas de validación cruzada de 3 veces.

En el algoritmo *k*-NN de vecinos próximos se ha tomado la distancia euclídea para la medida de distancias, y además se ha buscado el número *k* óptimo para cada conjunto de datos (menor que 20, para no sobreajustar).

Para las redes neuronales se utiliza una capa de entrada de 768 neuronas, 4 capas ocultas de 100 neuronas y la capa de salida, con 3 neuronas. Las capas ocultas utilizan función de activación *ReLU*, y la capa de salida una función de activación *Softmax*.

Estos métodos se aplican a cada una de las matrices obtenidas anteriormente, creando así 24 clasificadores distintos. Los resultados de precisión y tiempo empleado (en segundos, desde Google Colab) de cada uno de los modelos se muestran en las siguientes tablas.

	XGBoost GS	XGBoost RS	k-NN	Red Neuronal
x67	0,874	0,884	0,894	0,888
x80	0,893	0,890	0,894	0,882
x85	0,887	0,889	0,895	0,883
x90	0,886	0,887	0,892	0,872
x95	0,883	0,880	0,892	0,872
x100	0,883	0,897	0,890	0,885

Cuadro 3.3: Accuracy de cada uno de los clasificadores, modelo FinBERT.

	XGBoost GS	XGBoost RS	k-NN	Red Neuronal
x67	11	17	0,05	6,47
x80	47	70	0,08	6,53
x85	77	155	0,08	6,59
x90	154	271	0,15	6,57
x95	336	723	0,13	4,88
x100	1409	2182	0,26	11,64

Cuadro 3.4: Tiempo (en segundos) empleado para la creación de cada clasificador.

Si se compara con el modelo base, el accuracy está bastante bien, así que se mejora la precisión. Se observa que los valores que se han obtenido son bastante similares, apenas hay cambio entre matrices ni entre métodos. El valor más alto es de 0,897, pero el más bajo de 0,872. Se puede intentar mejorar

la eficiencia de los algoritmos. Para el algoritmo *k*-NN se podría cambiar la medición de las métricas, aunque es posible que apenas haya variaciones; quizás sería más interesante ajustar el número de capas para crear una buena red neuronal.

Por parte del algoritmo XGBoost, se han hecho dos búsquedas de hiperparámetros distintas, pero buscando solamente de entre unos valores concretos. La búsqueda ha sido muy restringida y se pueden intentar nuevas maneras de aproximarse a mejores valores. Además,se conoce al algoritmo XGBoost como uno de los mejores en tareas de clasificación, y al tener una búsqueda poco fructífera, se va a intentar realizar una más exhaustiva con un algoritmo de optimización global.

3.3.1. Algoritmo de optimización para la búsqueda de hiperparámetros

Se va en este caso a intentar perfeccionar más la búsqueda de hiperparámetros del algoritmo XGBoost, planteando esta búsqueda como un problema de optimización. Se puede incluir un algoritmo para la búsqueda, en concreto, se puede usar un algoritmo RR-PSO como el que se ha explicado antes. Esta parte corresponde al anexo A.4.

El propósito de este algoritmo es permitir una selección eficiente de hiperparámetros para el algoritmo XGBoost, con el fin de aumentar su precisión en la clasificación de frases. Por ello, la posición de una partícula corresponde a una combinación de valores de hiperparámetros del algoritmo XGBoost, mientras que la velocidad influye en cómo se mueve la partícula a través del espacio de búsqueda.

Para utilizar el algoritmo PSO en la elección óptima de hiperparámetros para el algoritmo XGBoost, primero se define la función objetivo a optimizar, en este caso la exactitud o *accuracy*. Luego, se define el espacio de búsqueda de hiperparámetros que hay que explorar, que en este caso solo se seleccionan unos cuantos, como el número de árboles, la profundidad máxima del árbol o la tasa de aprendizaje. A continuación, se inicializa un enjambre de partículas, cada una con una combinación aleatoria de valores de hiperparámetros dentro del espacio de búsqueda. El algoritmo RR-PSO se ejecuta para actualizar las posiciones y velocidades de las partículas en función del rendimiento de cada partícula en la función objetivo. Después de varias iteraciones, el algoritmo RR-PSO convergerá hacia una solución óptima o cerca de ella, que corresponderá a una combinación de hiperparámetros que optimice el rendimiento del algoritmo XGBoost en nuestro problema específico.

A continuación un esquema de cómo funciona este algoritmo, obtenido de Jiang et al. (2020).

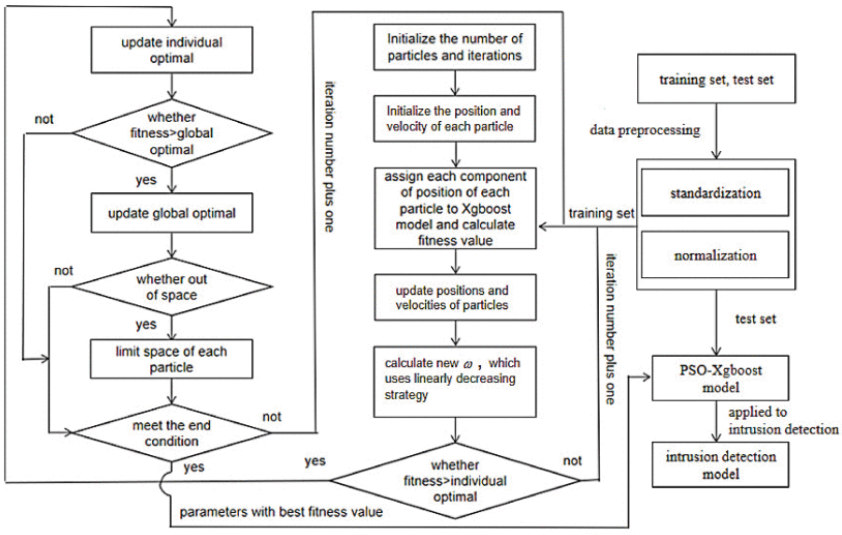


Figura 3.2: Un esquema del algoritmo RR-PSO - XGBoost propuesto.

En este caso, se ha probado la optimización con los hiperparámetros definidos en la tabla (2.3), pero el espacio de búsqueda se ha acotado en torno a unos rangos de valores que hacen que haya más posibilidades que la optimización sea óptima. Estos valores son los de la tabla (3.5).

Hiperparámetro	**Espacio de búsqueda**
n_estimators	50 - 200
max_depth	2 - 10
learning_rate	0,01 - 0,3
gamma	0 - 0,5
min_child_weight	0 - 1
subsample	0,4 - 1
colsample_bytree	0,6 - 1

Cuadro 3.5: Espacio de búsqueda de cada hiperparámetro.

Además, para agilizar el proceso se utiliza la matriz `x85`, ya que los resultados de eficacia y precisión han sido similares con la matriz `x100` que contenía toda la varianza explicada.

El algoritmo se somete a un proceso de optimización que consta de 20 iteraciones. En cada iteración, se maneja un conjunto de 20 individuos (o soluciones potenciales). En cada iteración, estos individuos se actualizan, y solo los mejores se conservan para la siguiente iteración. Este proceso de selección de los mejores individuos se repite 5 veces en total. El objetivo de realizar estas 5 repeticiones es obtener distintos valores óptimos, y al final, se selecciona el valor óptimo más grande en términos de precisión (*accuracy*) como resultado final.

Los datos recogidos de las 5 iteraciones son los siguientes:

Ejecución	**1**	**2**	**3**	**4**	**5**
n_estimators	200	190	80	123	159
max_depth	2	9	5	6	6
learning_rate	0.031677	0.278009	0.222686	0.207869	0.147154
gamma	0	0.466614	0.209504	0.344102	0.197121
min_child_weight	0	0.319747	0.733009	0.385500	0.473286
subsample	0.4	0.524400	0.519273	0.500303	0.719082
colsample_bytree	0.781129	0.886675	0.878353	0.963939	0.872354
Accuracy	0.923	0.924	**0.927**	0.924	0.923

Cuadro 3.6: Hiperparámetros óptimos en cada ejecución, con su *accuracy*.

El mejor resultado de *accuracy* obtenido es de 0,927. En particular, se va a estudiar el modelo de la tercera ejecución. Se analizan también el resto de métricas, que se muestran en la tabla 3.8.

Se ve que mejora en todos estos aspectos al anterior modelo. Solo tiene un inconveniente, y es que la pérdida de entropía cruzada en este modelo es 0,64. La pérdida de entropía cruzada mide la calidad de las predicciones en términos de cuan cerca están las distribuciones de probabilidad predichas de las distribuciones reales. Un valor más bajo de pérdida es preferible, ya que indica que el modelo está ajustando mejor sus predicciones a las etiquetas verdaderas.

Clase	Precisión	Recall	F1-score	Elementos
Positivo	0,93	0,86	0,89	295
Negativo	0,93	0,95	0,94	564
Neutral	0,93	0,97	0,95	109
Media	0,93	0,93	0,93	968
Media ponderada	0,93	0,92	0,93	968

Cuadro 3.7: Métricas del modelo XGBoost con ayuda de RR-PSO.

Por otro lado, es interesante también ver qué relación hay entre el porcentaje de expertos que están de acuerdo en la etiqueta puesta y la precisión del modelo. Se puede entrenar y evaluar el modelo solamente con las frases en las que el 100 % de los expertos están de acuerdo y ver si el modelo mejora con sus precisiones. Para ello se seleccionan únicamente las 2259 frases en las que todos los expertos están de acuerdo, se entrena con el 80 % de las frases (1807) y el resto sirven para probar las métricas.

Los resultados son muy positivos, de las 452 frases del conjunto de prueba solo yerra 8, es decir, el modelo tiene una precisión de 0,98. Además, también se logra un valor del F1-score de 0,98, así que el modelo está clasificando correctamente la gran mayoría de las frases positivas y negativas, minimizando tanto los falsos positivos como los falsos negativos. La parte menos buena vuelve a ser la pérdida de entropía cruzada, que aunque baja ligeramente, sus predicciones no son tan confiables como querríamos, con un valor de 0,57. La tabla con las métricas es la siguiente.

Clase	Precisión	Recall	F1-score	Elementos
Positivo	0,97	0,96	0,97	110
Negativo	0,99	0,99	0,99	274
Neutral	0,97	0,99	0,98	68
Media	0,98	0,98	0,98	452
Media ponderada	0,98	0,98	0,98	452

Cuadro 3.8: Métricas del modelo XGBoost con ayuda de RR-PSO, con 100 % fiabilidad.

3.4. Comparación con otros modelos de lenguaje

Con el objetivo de ver qué tan bueno es el anterior clasificador que se ha creado, y justificar que merece la pena haber entrenado el modelo FinBERT para hacer un modelo más eficiente en esta tarea específica, se importan el modelo de lenguaje BERT básico con la finalidad de comparar estos modelos con el que se ha creado.

Modelo BERT

Se importa es el modelo BERT base. Con este se tokenizan las frases y se obtienen sus representaciones numéricas. De esta forma, se obtiene la matriz de coordenadas de la base de datos Financial PhraseBank respecto del modelo BERT. Esta matriz también es de dimensión 4838×768, ya que el anterior modelo se basaba en esta codificación.

Se realizan los mismos pasos que para el anterior modelo, un análisis de reducción de la dimensionalidad PCA y se crean los clasificadores con los mismos métodos que antes (XGBoost, k-NN y redes neuronales), con el mismo porcentaje de frases para entrenar y como prueba. Además, se usa como la misma notación para las matrices PCA.

La tabla siguiente (3.9) muestra los *accuracys* que se han obtenido tras crear y entrenar los modelos con cada una de las matrices creadas en la PCA.

	XGBoost GS	XGBoost RS	k-NN	Red Neuronal
x67	0,597	0,606	0,585	0,493
x80	0,602	0,601	0,588	0,466
x85	0,602	0,601	0,596	0,467
x90	0,576	0,597	0,588	0,466
x95	0,590	0,598	0,588	0,474
x100	0,595	0,598	0,591	0,597

Cuadro 3.9: Accuracy de cada uno de los clasificadores, modelo BERT.

Estos valores son claramente peores que los anteriores, así que se pone en evidencia la clara mejoría de los modelos que hemos creado antes con la ayuda de FinBERT, y más todavía del último empleando el algoritmo RR-PSO.

Se podría intentar mejorar estos valores de la misma forma que antes, pero los resultados no son lo suficientemente buenos como para que con esta técnica se puedan igualar los resultados.

3.5. Análisis de la precisión con otras bases de datos

Volviendo al mejor modelo que se ha creado, ya se ha visto que este es muy bueno con la base de datos Financial PhraseBank. Vamos a ver cómo se comporta este modelo con otras bases de datos distintas, que también sean del ámbito financiero ya que se supone que tiene que estar entrenado para ello.

La base de datos escogida es un conjunto de tuits relacionados con el mundo financiero, como se ha descrito anteriormente. Se procede a separar de manera aleatoria el conjunto de datos, de manera que el 80 % de los tuits se utilizan para el entrenamiento y el resto para probar sus métricas.

Se obtiene un clasificador con un accuracy de 0,836, es decir, que aunque se tiene una precisión menor que con la base de datos de antes, este modelo clasifica correctamente la amplia mayoría de laas frases. Además, el resto de métricas que se obtienen son las siguientes, de la tabla 3.10:

Clase	**Precisión**	**Recall**	**F1-score**	**Elementos**
Positivo	0,85	0,71	0,77	1688
Negativo	0,83	0,94	0,88	3475
Neutral	0,89	0,54	0,67	525
Media	0,86	0,73	0,78	5688
Media ponderada	0,84	0,84	0,83	5688

Cuadro 3.10: Métricas del modelo XGBoost con ayuda de RR-PSO, base de datos de Twitter.

Vuelve a haber un número grande en la pérdida de entropía cruzada, con un valor de 0,80. Aunque sería preferible un valor más bajo, el resto de métricas son muy positivas.

Capítulo 4

Conclusiones

Finalmente, en este último capítulo, se van a exponer los resultados obtenidos en este trabajo y las conclusiones que se han sacado de ello. También señalar los objetivos que se han alcanzado, y analizar qué cosas han quedado pendientes y cual podría ser el trabajo posterior a este, además de las aplicaciones directas de este estudio.

4.1. Resumen de los resultados

En este estudio de investigación, se emprendió un análisis exhaustivo destinado a evaluar el rendimiento de modelos en el procesamiento de lenguaje natural, empezando con un modelo base que se ha ido tratando de mejorar. La tarea central de este análisis fue la clasificación de sentimientos en textos financieros, una tarea esencial en el campo del procesamiento de lenguaje natural que tiene un amplio rango de aplicaciones prácticas.

El modelo FinBERT fue el punto de partida de este estudio. Inicialmente, demostró ser sólido con una precisión del 86 % en la clasificación de sentimientos. Este resultado inicial ya era prometedor, pero el objetivo era mejorar aún más. Para lograrlo, se realizó una reducción de dimensionalidad para hacer un estudio de significación de componentes principales. Para varias matrices de coordenadas, se implementó un proceso de entrenamiento a través de varios métodos de clasificación, como el algoritmo XGBoost, el algoritmo k-NN y el uso de redes neuronales. Este entrenamiento fue llevado a cabo con la base de datos Financial PhraseBank. Todos estos procesos hicieron que el modelo FinBERT alcanzara una precisión del $88-89\,\%$, un progreso sustancial en la capacidad de clasificar con precisión los sentimientos expresados en textos.

Posteriormente, se exploró la aplicación de un algoritmo de optimización global para la optimización de los hiperparámetros del algoritmo XGBoost para intentar mejorar estos resultados. Se llevaron a cabo ajustes en los hiperparámetros utilizando el algoritmo RR-PSO. Esto resultó en un aumento adicional de la precisión, que alcanzó un impresionante 92,7 % de acierto. A pesar de este logro, es importante destacar que la pérdida de entropía cruzada se mantuvo en 0,64. Si bien esta pérdida no mejoró en comparación con FinBERT inicial, la mejora en la precisión es altamente notable y sugiere que el clasificador XGBoost podría ser una elección sólida en aplicaciones específicas de análisis de sentimientos, y más concretamente para el ámbito financiero.

Como comparativa, también se analizó el *accuracy* y otras métricas en modelos BERT no especializados en el mundo financiero, obteniendo unos resultados malos que muestran lo especializado y eficaz que es el modelo que se ha creado. Con un proceso similar al anterior, las precisiones de los modelos apenas alcanzaban el 60 % de precisión.

Se sometió al modelo creado a un entrenamiento con frases de Financial PhraseBank de una fiabilidad del 100 % de acuerdo entre expertos, consiguiendo mejorar aun más la precisión, hasta el 98 %, y

mejorando el valor de la pérdida de entropía cruzada hasta el 0,57. También se comparó este modelo con otras bases de datos, en este caso, con una recopilación de tuits, consiguiendo una precisión bastante buena, de 84 %.

En conclusión, se han obtenido unos resultados muy buenos que hacen del modelo creado un clasificador muy potente y efectivo para frases y noticias del mundo financiero.

4.2. Objetivos alcanzados

Los objetivos planteados en la investigación abarcaron una amplia variedad de aspectos relacionados con el análisis de sentimientos y el procesamiento de lenguaje natural. A continuación, se describen los objetivos cumplidos:

A lo largo del trabajo de investigación se han explorado los principios fundamentales del análisis de lenguaje, situándolo en el ámbito científico y tecnológico actual. Además, se han adquirido los conocimientos teóricos sobre las herramientas y algoritmos de machine learning y deep learning empleados, esenciales cuando se trabaja con grandes volúmenes de datos.

Por otro lado, también se ha hecho un estudio exhaustivo de las etapas de funcionamiento de un modelo de lenguaje, así como los procesos de entrenamiento y evaluación asociados. En cuanto al análisis de sentimiento, se ha investigado el estado del arte, y aunque no se ha podido trabajar con todos los modelos más novedosos, como los modelos GPT, sí se han usado otros muy importantes, como BERT y sus variantes.

A su vez, se ha tenido éxito en la creación de modelos, se han podido diseñar unos buenos modelos utilizando Python, con unas buenas predicciones y gestionando eficazmente los conjuntos de datos requeridos. También se ha logrado optimizar los modelos, con el uso de algoritmos como XGBoost o RR-PSO.

En resumen, la mayoría de los objetivos propuestos en la investigación se han cumplido con éxito. Se ha logrado mejorar y optimizar los modelos ya existentes, usando herramientas importantes de machine learning. Sin embargo, no todo el trabajo está hecho, y en este caso, como en la mayoría de las veces, todavía queda un camino que recorrer. Usar nuevos modelos, nuevas bases de datos o tratar de mejorar más las métricas de los modelos son solo algunas de las cosas que quedan pendientes como posible trabajo futuro.

4.3. Posible trabajo futuro

A pesar de los avances realizados en este trabajo, es importante reconocer que todavía existen áreas de oportunidad para futuras investigaciones y mejoras. Algunas de estas áreas a considerar incluyen:

- Optimización de Hiperparámetros: aunque se ha realizado una optimización de hiperparámetros en este trabajo, existe un espacio adicional para ajustar y sintonizar aún más estos parámetros. Una mejor y más extensa optimización de hiperparámetros puede desempeñar un papel crucial en el rendimiento de los modelos y podría llevar a mejoras adicionales en términos de precisión y de pérdida de entropía cruzada.
- Exploración de nuevos modelos: el campo del procesamiento de lenguaje natural está en constante evolución, y nuevas arquitecturas de modelos surgen con regularidad. Para mejorar aún más la precisión y el rendimiento de los modelos de análisis de sentimientos, se pueden explorar modelos más avanzados y recientes. Por ejemplo, considerar el uso de arquitecturas de redes neuronales más grandes, como modelos de transformers, que han demostrado un alto rendimiento en tareas de procesamiento de lenguaje natural. También se puede plantear la posibilidad de aprovechar

modelos de vanguardia como GPT, aunque hay que tener en cuenta que el uso de la API de estos modelos no es gratuito.

- Intentar predecir en bolsa: una vez se ha entrenado un modelo en el ámbito financiero y tenemos una clasificación bastante buena, se puede ir un paso más allá. Se podría tratar de almacenar el conjunto de noticias referidas a una empresa durante las últimas horas o días y hacer un análisis de sentimiento de esas noticias. Con ello, se podría saber si se está hablando bien o mal de una empresa, lo cual puede afectar a sus cotizaciones en bolsa.
- Análisis de sentimiento multiclase: si bien este trabajo se centró en la clasificación de sentimientos en tres clases (positivo, negativo y neutral), se podría considerar la expansión a un análisis de sentimientos multiclase con más categorías. Esto permitiría una comprensión más detallada de los matices en las opiniones expresadas en el texto.

4.4. Aplicaciones

Los resultados alcanzados en este trabajo no solo son relevantes desde una perspectiva académica, sino que también tienen importantes implicaciones en el mundo real:

- Aplicaciones en la industria: Los modelos de análisis de sentimientos mejorados pueden ser aplicados en diversas industrias, proporcionando una herramienta valiosa para comprender y evaluar la opinión pública y la percepción de los clientes. En la industria de la tecnología, por ejemplo, las empresas pueden utilizar estos modelos para monitorear las reacciones de los usuarios a sus productos y servicios en tiempo real. En el sector financiero, la capacidad de analizar los sentimientos expresados en noticias financieras y redes sociales puede ser crucial para la toma de decisiones de inversión.
- Detección de noticias falsas: En un mundo donde la desinformación es un problema creciente, los modelos de análisis de sentimientos pueden desempeñar un papel fundamental en la detección de noticias falsas. Estos modelos pueden evaluar la veracidad de las noticias al analizar el tono y la polaridad del contenido, identificando indicios de desinformación y ayudando a frenar la propagación de información falsa.
- Optimización de comunicaciones: Las organizaciones pueden utilizar estos modelos para optimizar sus estrategias de comunicación. Por ejemplo, pueden ajustar sus mensajes publicitarios y de marketing en función de las reacciones de los clientes, lo que mejora la efectividad de las campañas. También pueden anticipar y gestionar crisis de reputación al identificar rápidamente problemas de opinión pública.

Además de las aplicaciones mencionadas anteriormente, existen numerosas posibilidades adicionales en el campo del procesamiento natural del lenguaje y el análisis de sentimiento. Cada una de estas aplicaciones podría desencadenar líneas de estudio adicionales y oportunidades para futuros trabajos. Es importante destacar que este campo en constante evolución abre la puerta a una amplia gama de posibilidades de investigación, lo que significa que existen muchas más áreas por explorar y desarrollar para completar la investigación que se ha llevado a cabo.

Bibliografía

[1] ALAMMAR, J. (2018). *A Visual Guide to Using BERT for the First Time* [Blog post]. Extraído el 7 de julio de 2023 desde https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time/.

[2] ARACI, D. (2019). *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models*, University of Amsterdam. https://doi.org/10.48550/arXiv.1908.10063.

[3] BISHOP, C. M. (2006). *Pattern Recognition and Machine Learning*. New York: Springer.

[4] CHEN, T. & GUESTRIN, C. (2016). *XGBoost: A Scalable Tree Boosting System.* CoRR, volumen abs/1603.02754. https://doi.org/10.1145/2939672.2939785.

[5] DEVLIN, J., CHANG, M., LEE, K. & TOUTANOVA, K. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*.

[6] FERNÁNDEZ-MARTÍNEZ, J. L. & GARCÍA-GONZALO, E. (2012). *Stochastic stability and numerical analysis of two novel algorithms of the PSO family: PP-GPSO and RR-GPSO.* International Journal on Artificial Intelligence Tools. 21. 20. https://doi.org/10.1142/S0218213012400118.

[7] GOODFELLOW, I., BENGIO, Y. & COURVILLE, A. (2016). *Deep learning*. MIT Press.

[8] JIANG, H., HE, Z., YE, G., & ZHANG, H. (2020). *Network Intrusion Detection Based on PSO-Xgboost Model*. IEEE Access, 8, 58392-58401. https://doi.org/10.1109/ACCESS.2020.2982418.

[9] KENNEDY, J. & EBERHART, R. C. (1995). *Particle Swarm Optimization*. Proceedings of IEEE International Conference on Neural Networks (p./pp. 1942–1948), November, Washington, DC, USA: IEEE Computer Society.

[10] LIU, B. (2015). *Sentiment Analysis - Mining Opinions, Sentiments, and Emotions*. Cambridge University Press.

[11] MALO, P., SINHA, A., TAKALA, P., KORHONEN, P. & WALLENIUS, J. (2013). *FinancialPhraseBank-v1.0*.

[12] RATHI, V. & WALLACH, D. (2019, 5 de septiembre). *Sentiment Analysis on Financial Tweets*. Extraído el 15 de agosto de 2023 desde https://www.kaggle.com/datasets/vivekrathi055/sentiment-analysis-on-financial-tweets?datasetId=334892.

[13] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł. & POLOSUKHIN, I. (2017). *Attention is all you need.* Advances in Neural Information Processing Systems (p./pp. 5998–6008).

[14] WITTEN, I. H., FRANK, E. & HALL, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Amsterdam. Morgan Kaufmann.