UNIVERSITAT DE VALÈNCIA

Máster en Inteligencia Artificial Avanzada y Aplicada

Impacto del Análisis de Sentimientos en la Predicción de Precios Financieros: Comparativa entre LSTM, BERT y FinBERT

ALUMNO:

Francisco de Borja Gregori García

TUTOR:

Valero Laparra Pérez-Muelas

CURSO ACADÉMICO:

2023/2024

Resumen

En el contexto económico actual, cada vez son más las personas que se interesan por el mundo de las inversiones, impulsadas por una mayor facilidad de acceso a plataformas digitales y por la creciente globalización de los mercados. Este fenómeno no solo involucra a grandes fondos, sino también a pequeños inversores que buscan rentabilizar su capital frente a la incertidumbre económica, la inflación y los bajos tipos de interés. Esta tendencia ha generado una creciente demanda de herramientas que permitan tomar decisiones financieras más rápidas, precisas e informadas.

En esta línea, el presente trabajo tiene como objetivo analizar el impacto del análisis de sentimientos en la predicción de precios financieros, integrando datos históricos del mercado con información textual extraída de noticias económicas. Para ello, se comparan tres enfoques distintos: (1) un modelo LSTM basado únicamente en series temporales de precios históricos; (2) un modelo LSTM combinado con vectores de sentimiento generados mediante BERT; y (3) un modelo LSTM que integra el análisis de sentimientos producido por FinBERT, una variante de BERT entrenada específicamente en el ámbito financiero.

La recopilación de datos se realiza mediante las APIs de Yahoo Finance (para precios históricos) y Finnhub (para noticias financieras). Los textos son procesados utilizando técnicas de análisis de sentimientos con los modelos de lenguaje BERT y FinBERT. Los resultados obtenidos se incorporan como variables de entrada al modelo LSTM junto con los datos históricos. Finalmente, se aplican métricas cuantitativas para evaluar la precisión de cada enfoque y determinar cuál ofrece mejores resultados en la predicción de movimientos del mercado.

Palabras clave: acciones, noticias, finanzas, análisis de sentimientos, predicción, series temporales, modelos, procesamiento del lenguaje natural.

Abstract

In the current economic context, an increasing number of people are becoming interested in the world of investments, driven by easier access to digital platforms and the growing globalization of markets. This phenomenon involves not only large funds, but also small investors seeking to make their capital profitable in the face of economic uncertainty, inflation and low interest rates. This trend has generated a growing demand for tools that allow faster, more accurate and better-informed financial decisions.

In this regard, this paper aims to analyze the impact of sentiment analysis on financial price prediction, integrating historical market data with textual information extracted from economic news. For this purpose, three different approaches are compared: (1) an LSTM model based exclusively on historical price time series; (2) an LSTM model combined with sentiment vectors generated using BERT; and (3) an LSTM model integrating sentiment analysis produced by FinBERT, a variant of BERT trained specifically in the financial domain.

Data collection is performed using Yahoo Finance (for historical prices) and Finnhub (for financial news) APIs. The texts are processed using sentiment analysis techniques with the BERT and FinBERT language models. The results obtained are incorporated as input variables into the LSTM model along with the historical data. Finally, quantitative metrics are applied to evaluate the accuracy of each approach and determine which one offers better results in predicting market movements.

Keywords: stocks, news, finance, sentiment analysis, prediction, time series, models, natural language processing.

Índice

Figuras

Tablas

1.- Introducción

En este capítulo se presenta el propósito y estructura del estudio, cuyo objetivo es comparar técnicas de series temporales y de análisis de sentimientos en la predicción de precios financieros. Comenzará con una visión general del contexto financiero y la motivación del proyecto. También se presentarán las metodologías empleadas y un esquema con los pasos que se seguirán para su desarrollo.

1.1.- Contexto y justificación

En los últimos años, el uso de técnicas de inteligencia artificial y aprendizaje profundo en el ámbito financiero han demostrado su potencial para mejorar la toma de decisiones en inversiones y predicción de precios de activos. Los modelos de redes neuronales recurrentes, en concreto las Redes de Memoria a Corto Largo Plazo (LSTM) son efectivos en el análisis técnico y series temporales capturando patrones en datos secuenciales. Sin embargo, con el auge de datos no estructurados, como las noticias o redes sociales, se ha incrementado el enfoque en nuevas fuentes que puedan influir en el comportamiento del mercado. Es aquí donde cobran relevancia los modelos que integran el análisis de sentimientos en BERT.

El análisis de sentimientos se establece como una metodología que pretende evaluar el estado de ánimo y percepción del mercado usando el procesamiento de lenguaje natural (NLP), permitiendo explorar en mayor profundidad la influencia del sentimiento en las fluctuaciones de los precios financieros. FinBERT es una variante de BERT adaptada al lenguaje financiero, diseñada para mejorar la precisión del análisis en textos económicos. Este trabajo combina las ventajas de LSTM para las series temporales con BERT y FinBERT para el análisis de sentimientos, mejorando la precisión en las predicciones de los precios de acciones. Se hará uso de datos de Yahoo Finance y *Finnhub* API para la recopilación de información de mercado y noticias.

1.2.- Planteamiento del problema

Los mercados financieros están influenciados por una constante evolución en los precios de los activos, resultando un desafío constante para los inversionistas y analistas financieros. La falta de precisión en los modelos de predicción puede ocasionar decisiones erróneas que impacten de manera negativa a las carteras de inversión aumentando el

riesgo financiero. Dicho problema se agrava omitiendo factores externos, como las noticias o reportes, que afectan directamente al comportamiento de los activos.

Por ello surgió un enfoque en el que el modelo no solo tome en cuenta el historial de precios, sino también el sentimiento del mercado que busca medir la emoción o estado de ánimo reflejado en las noticias y redes sociales. Este estudio se centra en determinar si la integración del análisis de sentimientos basados en BERT y FinBERT mejora el rendimiento de los modelos LSTM para la predicción de precios financieros.

1.3.- Objetivos del trabajo

El objetivo principal es desarrollar un modelo de predicción de series temporales basado en redes neuronales LSTM que integre análisis de sentimientos usando modelos de lenguajes preentrenados como BERT y FinBERT. Se llevará a cabo una comparación del rendimiento de los modelos LSTM con y sin el uso de estas variantes, evaluando su impacto en la predicción de precios financieros integrando datos textuales.

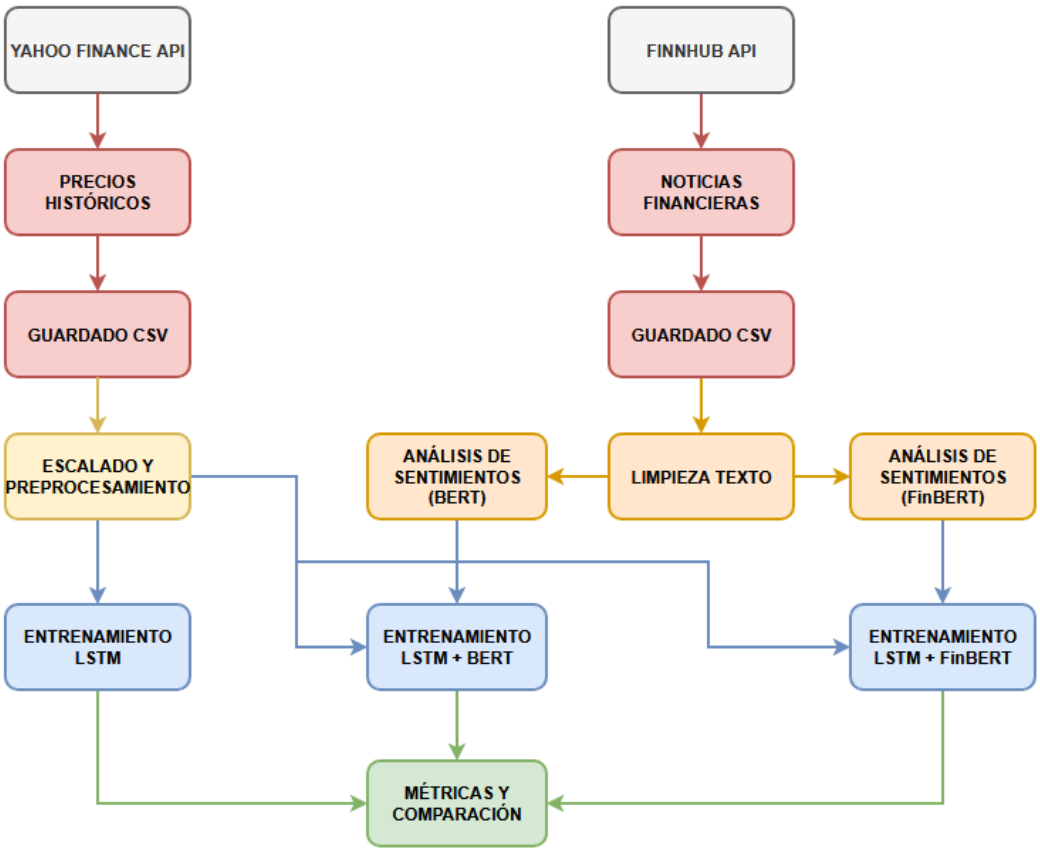

Figura 1: Esquema del modelo de predicción

Para cumplir con el objetivo general descritos en la *Figura 1* se plantean los siguientes objetivos específicos:

- ➔ Implementar un modelo LSTM para la predicción de series temporales de precios de acciones haciendo uso de datos históricos extraídos de Yahoo Finance.
- ➔ Integrar en dicho modelo, análisis de sentimientos basado en BERT, usando información textual de noticias financieras utilizando *Finnhub* API.
- ➔ Implementar su variante FinBERT, optimizada para el análisis de textos financieros y evaluar la efectividad de su predicción frente a BERT.
- ➔ Hacer uso de métricas de precisión como RMSE y MAE para realizar la comparación entre los tres enfoques: LSTM sin análisis de sentimientos, LSTM con análisis de sentimientos de BERT y LSTM con análisis de sentimientos de FinBERT.

2.- Redes neuronales y modelos para series temporales

A lo largo de este capítulo se ofrece una visión general de las redes neuronales artificiales (RNA), introduciendo su estructura y describiendo diferentes arquitecturas junto a sus principales aplicaciones. A continuación, se profundiza en las redes LSTM, detallando el funcionamiento de sus puertas para preservar información a corto y largo plazo en tareas de series temporales.

2.1.- Redes neuronales artificiales

Las redes neuronales artificiales han supuesto uno de los mayores avances para el campo del aprendizaje automático cuya estructura y funcionamiento están inspirados en el cerebro humano. Las RNA están formadas por numerosos nodos conocidos como neuronas que se distribuyen en capas para procesar y transformar la información a través de pesos sinápticos ajustables. Cada neurona recibe una serie de entradas ponderadas, las cuales procesa a través de una función de activación y genera una salida.

Con el aumento de la capacidad computacional y desarrollo de nuevos algoritmos, las RNA han demostrado un alto rendimiento en tareas como clasificación, regresión, predicción y generación de contenido [1].

2.1.1.- Concepto y arquitectura general de redes neuronales

Como se ha mencionado, las RNA se basan en una estructura de capas conectadas: una capa de entrada, una o más capas ocultas y una capa de salida tal y como se muestra en la *Figura 2*. Estas capas contienen neuronas donde cada una de ellas tiene un peso que representa la importancia de esa conexión en el proceso de aprendizaje.

- ➔ La capa de entrada es la que recibe los datos en crudo y los envía a las capas ocultas.
- ➔ Las capas ocultas procesan la información mediante operaciones matemáticas, ajustando sus pesos a través de un proceso de optimización. Dependiendo del problema a resolver se hace uso de un tipo de red que varía en la cantidad y estructura de las capas.
- ➔ La capa de salida proyecta el resultado final pudiendo resultar en una clasificación o predicción.

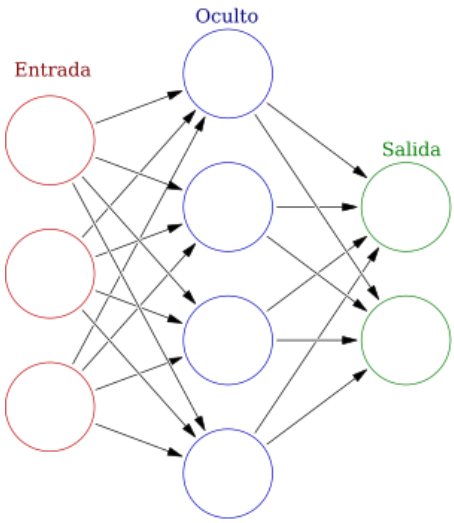

Figura 2: Red Neuronal Artificial

Las redes aprenden ajustando sus pesos mediante un proceso llamado retropropagación, el cual consiste en minimizar la función de pérdida permitiendo mejorar su capacidad para realizar predicciones o clasificaciones.

2.1.2.- Tipos de redes neuronales y su aplicabilidad

Actualmente existe una gran variedad de arquitecturas de redes neuronales, cada una de ellas adaptada a un tipo de problema en concreto. A continuación, se presentan algunas de las más relevantes y su aplicabilidad en diferentes contextos [2]:

- **Redes neuronales *feedforward* (FFNN):** son las RNA más básicas ya que la información se mueve en una única dirección, de entrada a salida. Demostraron ser eficientes para tareas de clasificación y regresión. Sin embargo, no son adecuadas para trabajar con secuencias de datos por falta de memoria.
- **Redes neuronales convolucionales (CNN):** tipo de redes especializadas en datos con estructura espacial, como imágenes, pero no recomendables para series temporales. Las CNN son usadas en los campos de visión, reconocimiento de imágenes y procesamiento de señales.
- **Redes neuronales recurrentes (RNN):** en estas redes la salida de una neurona puede volver como entrada de la siguiente iteración, permitiendo recordar información y manejar datos con secuencias. Es decir, son adecuadas para predicción de series temporales, aunque presentan limitaciones por la dificultad de retener información distante en el tiempo siendo un problema para manejar datos de secuencias largas.
- **Redes *Long Short-Term Memory* (LSTM):** es una variante de las RNN que permite mantener dependencias a largo plazo en datos secuenciales. Su arquitectura permite almacenar y recuperar información a lo largo de secuencias extensas, lo que las hace ideales para predicción de series temporales en finanzas o procesamiento del lenguaje natural.
- **Redes generativas adversarias (GAN**): están formadas por dos redes donde el generador es el encargado de crear datos falsos y el discriminador intenta distinguirlos de los reales. Se emplean en la generación de imágenes, vídeos y otros contenidos creativos.
- **Redes neuronales de transformación (*Transformers*):** estas redes no procesan secuencias como las RNN y LSTM, sino que utilizan un mecanismo de autoatención para evaluar el contexto de una palabra en relación con todas las demás palabras de una oración. Además, han revolucionado el campo del procesamiento del lenguaje natural (NLP).

2.2.- Redes LSTM

Las redes LSTM son un tipo de red neuronal recurrente diseñadas para procesar y predecir secuencias de datos y dependencias a largo plazo, como texto, audio o video, debido a su capacidad de recordar patrones complejos.

2.2.1.- Fundamentos y arquitectura de LSTM

Una de las características más importantes de las redes neuronales LSTM es que a diferencia de las RNN tradicionales, permite mitigar el desvanecimiento de gradientes, la cual causa retención de información relevante en secuencias largas.

Su estructura interna se basa en unidades de memoria llamadas "celdas de memoria" que permiten almacenar y recordar información durante períodos de tiempo extensos, haciendo ideales dichas redes para tareas que requieran procesamiento de secuencias de datos con dependencias temporales.

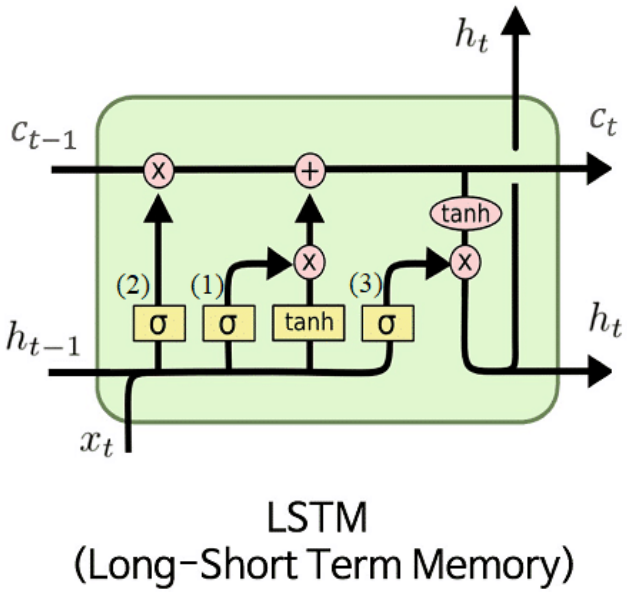

Figura 3: Diagrama de una célula de memoria LSTM

Como se muestra en la *Figura 3*, las celdas regulan el flujo de información a través de tres tipos de puertas: la puerta de entrada, la puerta de olvido y la puerta de salida. h_t es el estado oculto de las RNN que representa la memoria corta de la neurona, pero en las redes LSTM añadimos un segundo estado llamado c_t que representa la memoria a largo plazo.

➔ La puerta de entrada (1) determina la información que necesita ser actualizada en la celda de memoria. La información de entrada actual x_t y del estado previo de

la celda h_{t-1} genera un vector de activación que representa la información a añadir a la celda c_{t-1}. Esta adición de información resulta en una operación realizada entre dicho vector de activación y el estado previo c_{t-1}.

- ➔ La puerta de olvido (2) decide qué información de la celda es obsoleta y procede con su eliminación. Se recibe h_{t-1} y x_t y se genera un vector de activación que determina qué información desaparecerá. Cualquier valor cerca de 1 se mantendrá y cualquier valor cerca de 0 desaparecerá. Esto también resulta en una operación entre el segundo vector de activación y el estado previo c_{t-1}. A partir de las dos operaciones previas sobre c_{t-1}, se obtiene c_t.
- ➔ La puerta de salida (3) determina qué información debe ser usada para generar la salida del modelo. Para obtener la información final, se hace uso de la entrada actual x_t y del estado actual c_t. para generar un vector de activación que representa la salida del LSTM, obteniendo h_t.

Dicha estructura permite que las LSTM mantengan y actualicen información durante secuencias temporales largas, superando la incapacidad de gestionar dependencias de largo plazo de las RNN tradicionales. En relación con el precio de las acciones, el diseño de las LSTM captura patrones a lo largo de días, semanas e incluso meses resultando esencial para predicciones precisas [3].

2.2.2.- Aplicación de LSTM en la predicción de series temporales

Las LSTM son usadas para la predicción de precios de acciones donde han demostrado ser efectivas frente a la volatilidad de las acciones y la dependencia del tiempo de los mercados financieros. El uso de redes LSTM no solo se ha utilizado ampliamente en el ámbito de las acciones a largo plazo sino también en activos como criptomonedas, divisas e incluso en el corto plazo como análisis de volatilidad.

Estos activos son influenciados por numerosos eventos económicos globales, así como el sentimiento del mercado. Es por ello que se hace uso de las LSTM con capacidad de manejar datos secuenciales y considerar dependencias temporales permitiendo predecir tendencias futuras a partir de los datos históricos.

A todo esto, las LSTM también han demostrado tener éxito en la predicción de series temporales en otros sectores como la previsión de demanda y consumo de energía donde son empleadas para anticipar picos de demanda y optimizar la distribución en redes eléctricas. Al igual que en el sector energético, se hace lo mismo para el ámbito de

logística y comercio, utilizadas para prever la demanda de productos y optimizar inventarios.

Por último, en el contexto industrial las LSTM pueden anticipar fallos en maquinaria y equipos mediante el análisis de datos de sensores a lo largo del tiempo, reduciendo así el coste de mantenimiento y mejorando la eficiencia operativa.

2.2.3.- Ventajas y limitaciones de LSTM

Como se ha mencionado las LSTM son un modelo recurrente diseñado específicamente para procesar secuencias temporales largas, lo que le permite aprender relaciones complejas entre los datos históricos. La principal ventaja de LSTM es su capacidad para mantener la memoria de eventos pasados a través de sus celdas de memoria, capturando las dependencias a largo plazo que son cruciales en los mercados financieros. Además, las LSTM son eficaces para modelar la naturaleza secuencial de los datos financieros, donde los precios actuales dependen de las tendencias pasadas.

A todo esto, hay que sumarle que las LSTM no solo son útiles para predicción de series temporales sino también otro tipo de secuencias como procesamiento de lenguaje natural como traducción automática, generación de texto, reconocimiento de voz y vídeos, etc.

Sin embargo, LSTM tiene varias limitaciones. En primer lugar, suele necesitar volúmenes de datos enormes para poder aprender patrones significativos. Las LSTM aplicadas a series temporales con pocos datos históricos, hace que su rendimiento decrezca en comparación con los modelos más simples.

Además, aunque puede manejar datos temporales, no está diseñado para trabajar con datos textuales de manera eficiente. Cuando se trata de integrar análisis de sentimientos a partir de textos, LSTM requiere de mecanismos adicionales para procesar esta información. También, la naturaleza secuencial de LSTM implica un procesamiento más lento, ya que no puede aprovechar el paralelismo de la misma manera que los Transformers, lo que lo hace menos eficiente en términos computacionales [4].

3.- Procesamiento de lenguaje natural (NLP) y modelos transformadores

En este capítulo se presenta el procesamiento del lenguaje natural (NLP) como pilar para convertir el lenguaje humano en representaciones computacionales, detallando sus fases esenciales desde el preprocesamiento hasta la extracción de información. También, se examinan los principales tipos de modelos de NLP y su aplicabilidad práctica. Por último, se desglosa la arquitectura *Transformer* y se profundiza tanto en el modelo BERT como en su variante FinBERT.

3.1.- Introducción al procesamiento de lenguaje natural (NLP)

El NLP es una disciplina de la inteligencia artificial que extiende un puente entre la comunicación humana y la comprensión informática. Es un ámbito multidisciplinar que otorga a los ordenadores la capacidad de interpretar, manipular y comprender el lenguaje humano [5].

La importancia del NLP se ve reflejado en sus aplicaciones generalizadas, formando parte de la vida cotidiana de muchas personas como motor de búsqueda, atención al cliente automatizado, asistentes digitales, …

Cada vez supone un papel más importante para las empresas ayudándoles a agilizar y automatizar operaciones de negocio, aumento de la productividad y simplificar procesos [6].

3.1.1.- Concepto y fundamentos del NLP

El NLP es una rama de la inteligencia artificial que se centra en la interacción entre ordenadores y humanos a través del lenguaje natural. Tiene como objetivo la programación de los ordenadores para procesar y analizar el lenguaje humano, que es inherentemente ambiguo y variado. Dicho procesamiento del lenguaje consta de varias etapas [7]:

- **Preprocesamiento:**
  - Tokenización: es el proceso para dividir el texto en palabras, frases, símbolos u otros elementos, conocidos como tokens.
  - Normalización: convertir texto a minúsculas y eliminar puntuación o *stopwords*.

- **Análisis Léxico y Morfológico:**
  - Lematización y *stemming*: técnicas que llevan las palabras a su forma raíz, permitiendo que los modelos NLP traten palabras relacionadas como equivalentes.
  - Etiquetado gramatical: cada palabra es analizada para saber su función gramatical en la oración, ayudando a entender la estructura y simplificar el texto.
- **Análisis Sintáctico y Semántico:** determina la estructura gramatical y la relación contextual entre las palabras y oraciones. Esto permite que los modelos entiendan el contexto y las relaciones entre palabras.
- **Extracción de Información y Aplicaciones:** se extraen patrones y se aplican técnicas permitiendo a los modelos trabajar con extensos volúmenes de datos para analizar sentimientos, realizar clasificaciones, generar texto o responder preguntas.

3.1.2.- Tipos de modelos de NLP y su aplicabilidad

Existen varios tipos de modelos de NLP, cada uno de ellos con sus particularidades y aplicados según el problema y contexto:

- **Modelos estadísticos:** modelos que utilizan técnicas probabilísticas para predecir una secuencia de palabras. Son rápidos y simples para la optimización de tareas sencillas, pero carecen de comprensión textual profunda y captura de dependencias a largo alcance. Algunos ejemplos son el modelo de bolsa de palabras (*BoW*) y el modelo de palabras incrustadas (*Word2Vec*).
- **Modelos basados en redes neuronales:** al igual que las redes neuronales recurrentes y sus variantes, permiten capturar patrones complejos y dependencias a largo plazo en los datos textuales. Esto permite buenos resultados en tareas como la traducción automática y análisis de sentimientos.
- **Modelos de *Transformers***: utilizan mecanismo de atención para procesar palabras de una oración en paralelo y capturar relaciones a largo plazo más eficaces. Esto les da capacidad a los modelos a ser muy precisos en el análisis de sentimientos, clasificación y extracción de entidades. Un ejemplo que destacar es BERT (*Bidirectional Encoder Representations from Transformers*) que ha demostrado tener un gran rendimiento.

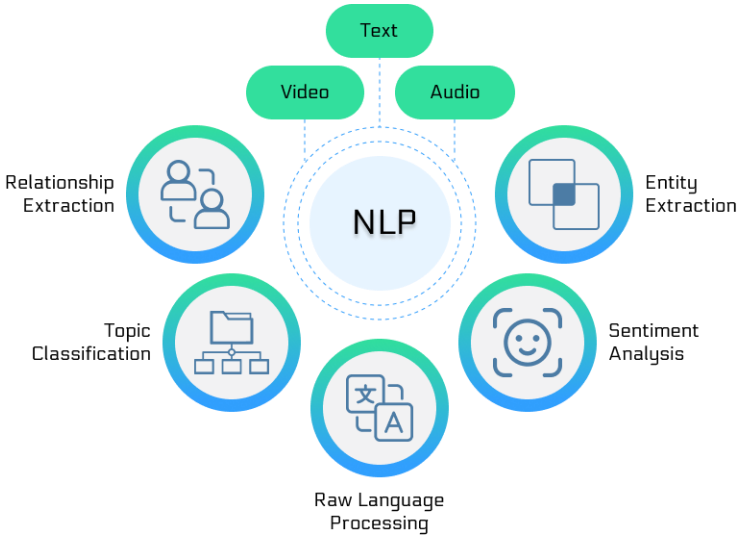

Figura 4: Aplicabilidad modelos NLP

La *Figura 4* muestra diferentes aplicabilidades de los modelos NLP, donde cada uno de ellos tiene sus ventajas y limitaciones basados en velocidad, precisión, captura de contexto y dependencias a largo plazo [8 – 9].

3.2.- Transformers y aplicaciones de BERT/FinBERT

Los *Transformers* han supuesto una revolución en el procesamiento de lenguaje natural gracias a la capacidad de capturar relaciones contextuales sin necesidad de conexiones recurrentes. El modelo BERT y su variante FinBERT están basados en una arquitectura de *Transformers*, convirtiéndose en herramientas poderosas para el análisis de sentimientos y contexto de textos financieros, mejorando la capacidad para predecir movimientos en el mercado.

3.2.1.- Fundamentos y arquitectura de Transformers

Un *Transformer* se basa en un mecanismo de atención, permitiendo a los modelos evaluar una palabra en relación textual con todas las demás en una frase de forma simultánea. Es decir, a diferencia de las RNN, los *Transformers* pueden analizar dependencias a largo plazo sin necesidad de procesar el texto secuencialmente, otorgando una mayor eficiencia y precisión en tareas de NLP como la traducción automática y el análisis de sentimientos.

La arquitectura *Transformer* mostrada en la *Figura 5* está formada por dos partes principales: el encoder y el decoder. Cabe destacar que modelos como BERT, solo hacen

uso de la parte del encoder, ya que están diseñados específicamente para comprender el contexto en secuencias de texto.

- ➔ **Encoder:** a través de múltiples capas de autoatención y capas de *feedforward* totalmente conectadas se encargan de transformar una secuencia de entrada, en una representación interna. Cada capa de encoder es idéntica en cuanto a estructura, pero los parámetros son propios de cada una siendo capaces de aprender patrones y características del texto diferentes.
- ➔ **Decoder:** toma la representación del encoder y hace uso de ella para producir una secuencia de salida.

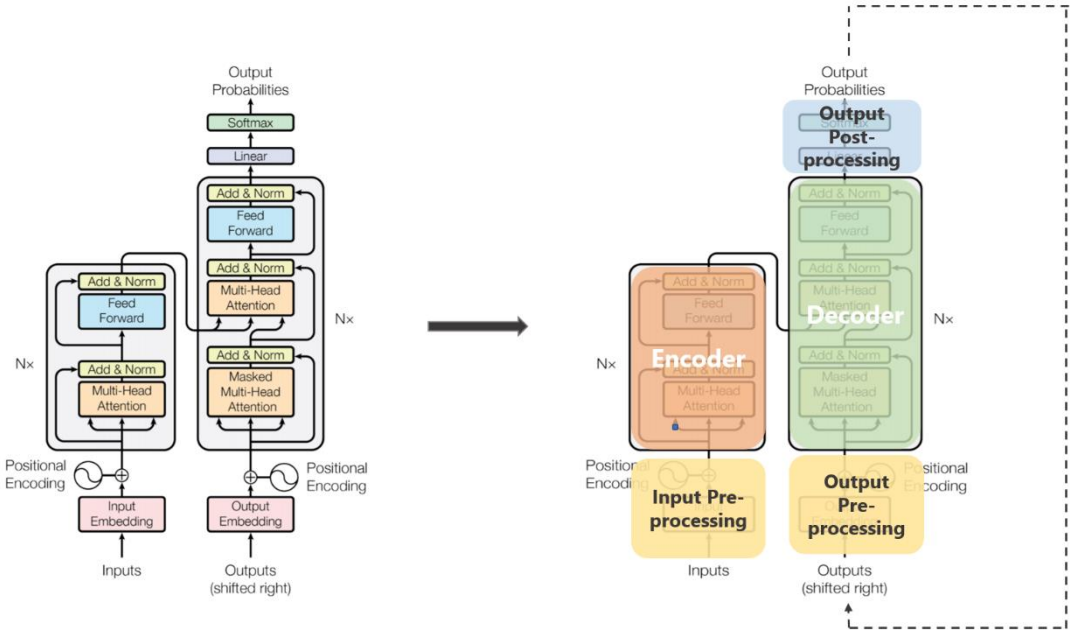

Figura 5: Arquitectura de los Transformers

Respecto al mecanismo de autoatención, es el núcleo de la arquitectura *Transformer* y permite que el modelo asigne un peso a cada palabra en relación con las demás palabras del texto. Esto permite al modelo entender de forma global el contexto y el significado de cada palabra [10 – 11].

3.2.2.- Introducción a BERT

El modelo preentrenado BERT (*Bidirectional Encoder Representation from Transformers*) ha transformado el campo del procesamiento de lenguaje natural que, a diferencia de las redes neuronales tradicionales, BERT utiliza una arquitectura bidireccional que permite en oraciones comprender el contexto de una palabra en ambos sentidos tanto hacia delante como atrás, ideal para la clasificación de texto y análisis de sentimientos tal y como se aprecia en la *Figura 6*.

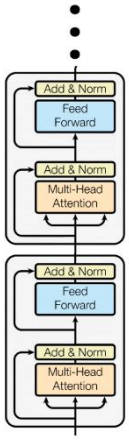

Figura 6: Esquema BERT

El preentrenamiento de BERT se realiza a partir de dos tareas: enmascaramiento de palabras y predicción de la siguiente oración. Este enfoque le permite identificar relaciones complejas y patrones contextuales que mejoran en precisión las tareas de NLP [12].

3.2.3.- FinBERT: adaptación de BERT para el análisis de sentimiento financiero

FinBERT es una variante de BERT que ha sido preentrenada específicamente para el análisis de textos financieros. Fue entrenado a partir de noticias económicas y reportes corporativos, convirtiéndola en una herramienta más precisa para interpretar lenguaje técnico y especializado en este campo. La integración de esta variante es adecuada para tareas como la clasificación de noticias financieras e identificar el riesgo asociado a eventos en el mercado.

3.2.4.- Ventajas y limitaciones de BERT y FinBERT en aplicaciones financieras

Tanto el modelo BERT como su variante FinBERT, ofrecen ventajas para analizar textos financieros, sobre todo en clasificación de documentos, análisis de sentimientos y extracción de información clave en noticias o reportes. Algunas ventajas son [13]:

- ➔ Como se ha mencionado, BERT permite capturar el contexto bidireccional teniendo en cuenta el significado de cada palabra en relación con las palabras posteriores como anteriores.
- ➔ FinBERT al ser una variante entrenada con un gran volumen de datos específicos en el sector financiero, ofrece una mayor precisión en el análisis de sentimientos de textos financieros.

- ➔ Ambos modelos tienen destreza en tareas de extracción de entidades y clasificación de documentos, como es el identificar una empresa, cifras económicas, eventos de mercado y nombres de productos.
- ➔ Los modelos basados en *Transformers* pueden ajustarse y adaptarse a tareas en específico haciendo *fine-tuning* (ajuste fino), lo que permite flexibilidad y adaptabilidad a nuevas necesidades. En relación con lo dicho, pueden manejar grandes cantidades de datos de texto en paralelo, crucial para el análisis a tiempo real de noticias las cuales pueden tener un impacto inmediato en el mercado.

Sin embargo, también presentan limitaciones debido principalmente a su complejidad y requisitos computacionales. Algunas limitaciones son:

- ➔ Ambos modelos requieren de grandes cantidades de memoria y capacidad de procesamiento para entrenarse y operar. El análisis en tiempo real es complejo debido a la latencia y costos computacionales lo que limita su aplicación a sistemas con altos recursos.
- ➔ FinBERT es menos efectivo en campos que requieren interpretaciones fuera del lenguaje financiero, lo que puede reducir su precisión al igual que dificultades para captar el sarcasmo, ironía o matices complejos. También cabe mencionar que BERT y FinBERT son modelos de caja negra, en otras palabras, sus decisiones y predicciones son difíciles de interpretar. En el sector financiero se requiere transparencia para entender cómo los factores textuales impactan en el análisis, por lo que la falta de interpretabilidad puede ser un problema.
- ➔ El lenguaje financiero está en constante evolución como es la introducción de nuevos términos o siglas, por eso es necesario reentrenar periódicamente con datos actuales del sector financiero, lo que requiere una inversión en términos de tiempo y recursos.

4.- Metodología del modelo de predicción

En el siguiente capítulo se describe el flujo para construir el sistema de predicción de precios. Empieza con la importación de librerías necesarias y definición de las variables clave. A continuación, se recopilan y preparan los datos financieros y de noticias. Luego, se aplica BERT y FinBERT para cuantificar el sentimiento diario,

seguido de la implementación de unos componentes adicionales. Por último, se definen los entrenamientos de los modelos y las métricas para evaluar los resultados obtenidos.

4.1.- Configuración inicial

Como se observa en la *Figura 7*, se declaran al principio del código diferentes tipos de librerías:

- ➔ Librerías para el manejo de datos y visualización
- ➔ Librerías para la obtención de datos
- ➔ Librerías para el procesamiento de texto
- ➔ Librerías para procesamiento y análisis de datos
- ➔ Librerías para *Deep Learning*

```python
# Declaración de librerías generales
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Obtención de datos
import time
import yfinance as yf
import requests
import logging
from datetime import datetime, timedelta

# Limpieza y procesamiento de texto
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# nltk.download('stopwords')
# nltk.download('wordnet')

# Preprocesamiento y métricas
from transformers import pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Deep Learning
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional, Input, Layer
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras import regularizers
```

Figura 7: Declaración de librerías

También al inicio del código establecemos un *logging* para que se muestre la información útil durante la ejecución del programa. Esto permite detectar errores o tener un mejor seguimiento del estado del código. Posteriormente, se definen 5 variables las cuales son:

- ➔ API_KEY: contiene la clave de la API de *Finnhub* para autorizar las solicitudes y obtener noticias financieras.
- ➔ SYMBOL: representa el símbolo de cotización de la empresa (en este caso Apple) en el mercado. Se utiliza para identificar que datos financieros descargar.
- ➔ START_DATE: fecha de inicio que se obtienen los datos financieros y noticias.
- ➔ END_DATE: fecha final que se obtienen los datos financieros y noticias.
- ➔ LOOK_BACK: establece el número de registros históricos que se utilizarán para predecir el siguiente precio en los modelos.

4.2.- Recopilación y preparación de datos

En este apartado se van a relatar las funciones para la obtención de datos para el estudio mediante el uso de las APIs y el preprocesado realizado a dichos datos antes del análisis de sentimientos.

4.2.1.- Obtención de Datos Financieros y de Noticias

La función *historical_data* mostrada en la *Figura 8* se encarga de descargar, procesar y almacenar los datos históricos de un activo financiero, en este caso se utiliza la acción de la compañía *Apple para un rango de fechas especificado*. La ejecución del código sigue los siguientes pasos:

1.- Muestra un mensaje de *log* que indica el símbolo y el rango de fechas que se va a procesar.

2.- Mediante *yf.download* se obtiene un *DataFrame* con las columnas de *Yahoo Finance* y reinicia el índice para que *Date* pase a ser columna. En caso de error, la función retorna un *DataFrame* vacío.

3.- Se convierte la columna *Date* a tipo fecha sin componente horario.

4.- Se filtran las columnas relevantes para el análisis (*Date*, *Open*, *High*, *Low*, *Close*, *Volume*).

5.- Se añade la columna NSI mediante la función *nsi(stock_data)* sobre el *DataFrame*, asignando valores de {-1, 0, 1} según la evolución del precio.

6.- Se establece el nombre y formato del archivo, devolviéndolo ya procesado.

```python
# Función para obtener precios históricos
def historical_data(symbol, start_date, end_date):
    # Inicio de la búsqueda de datos
    logging.info(f"Buscando precios de {symbol} desde {start_date} hasta {end_date}...")
    try:
        # Descarga los datos de Yahoo Finance
        stock_data = yf.download(symbol, start=start_date, end=end_date)
        stock_data.reset_index(inplace=True)
    except Exception as e:
        logging.error(f"Error al descargar datos para {symbol}: {e}")
        return pd.DataFrame()
    # Asegura que la columna Date sea solo fecha
    stock_data['Date'] = pd.to_datetime(stock_data['Date']).dt.date
    # Se filtran las columnas de interés
    stock_data = stock_data[['Date','Open','High','Low','Close','Volume']]
     # Calcula y añade el indicador NSI
    stock_data['NSI'] = nsi(stock_data)
    # Nomenclatura y formato del archivo de salida
    filename = f"{symbol}_historical_data.csv"
    try:
        logging.info(f"Guardando {filename}...")
        stock_data.to_csv(filename, index=False)
    except Exception as e:
        logging.error(f"Error al guardar {filename}: {e}")

    return stock_data
```

Figura 8: Función de descarga de históricos

Más adelante, el código comprueba si el fichero existe en disco para evitar descargas innecesarias, mostrando un mensaje de log donde los datos han sido cargados. En caso contrario, se pondrá a descargar, procesar y guardar datos devolviendo el *DataFrame* con las columnas *Date*, *Open*, *High*, *Low*, *Close*, *Volume* y *NSI*.
Finalmente, al llamar a *stock_data.shape* se obtienen las dimensiones del *DataFrame* donde para este estudio muestra por pantalla las dimensiones (266 filas, 7 columnas), correspondiendo las filas a los días en que el mercado estuvo abierto dentro del rango de fechas especificado.

La función *news_data* ilustrada en la *Figura 9* consulta la API de *Finnhub* para obtener las noticias del rango de fechas especificados de la empresa *Apple*. A continuación, son los pasos que sigue dicha función:
1.- Se convierten las cadenas *start_date* y *end_date* a tipo *datetime*, también se define *all_news* como lista vacía y *call_count* como contador para el control de peticiones.

2.- Se crea una lista *apple_keywords* para filtrar noticias relevantes de la compañía.

3.- Se itera día a día donde se genera el registro de búsqueda de noticias a través de la URL de la API, completando los parámetros requeridos y se realiza la llamada HTTP. Para cada elemento JSON devuelto, extrae el campo *headline* añadiéndolo a *all_news* en caso de contener alguna palabra clave. Se incrementa el contador en 1 y cuando alcanza 55 llamadas, se pausa durante un minuto para no exceder el límite.

4.- Tras completar el bucle, se convierte la lista *all_news* en un *DataFrame*.

5.- Se establece el nombre y formato del archivo.

```python
# Función para obtener noticias financieras
def news_data(symbol, start_date, end_date, api_key):
    # Se cargan fechas y contador de llamadas
    current_date = datetime.strptime(start_date, "%Y-%m-%d")
    end_date_dt   = datetime.strptime(end_date,   "%Y-%m-%d")
    all_news, call_count = [], 0
    # Palabras clave para filtrar noticias
    apple_keywords = ['Apple','iPhone','Mac','iPad','Tim Cook','iOS','macOS','Apple Watch','AirPods']
    # Se itera día a día en el rango solicitado
    while current_date <= end_date_dt:
        date_str = current_date.strftime("%Y-%m-%d")
        logging.info(f"Buscando noticias para {symbol} en {date_str}...")
        url    = 'https://finnhub.io/api/v1/company-news'
        params = {'symbol': symbol, 'from': date_str, 'to': date_str, 'token': api_key}
        try:
            resp = requests.get(url, params=params); resp.raise_for_status()
            for item in resp.json():
                hl = item.get('headline','')
                if any(k.lower() in hl.lower() for k in apple_keywords):
                    all_news.append({
                        'datetime': datetime.fromtimestamp(item['datetime']).date(),
                        'headline': hl})
        except Exception as e:
            logging.error(f"Error buscando noticias para {symbol} en {date_str}: {e}")
        # Gestión de límite de llamadas por minuto
        call_count += 1
        if call_count >= 55:
            logging.info("Límite de llamadas alcanzado; esperando 60s...")
            time.sleep(60); call_count = 0
        current_date += timedelta(days=1)
    # Nomenclatura y formato del archivo de salida
    news_df = pd.DataFrame(all_news)
    filename = f"{symbol}_news_data.csv"
    try:
        logging.info(f"Guardando {filename}...")
        news_df.to_csv(filename, index=False)
    except Exception as e:
        logging.error(f"Error al guardar {filename}: {e}")

    return news_df
```

Figura 9: Función de descarga de noticias

Al igual que la función anterior, el código gestiona la carga o descarga de los datos de noticias financieras. Si el fichero existe se muestra por mensaje su carga, pero en caso de que el archivo no exista o falle se invoca a la función *news_data* para obtener las noticias desde la API, almacenando la información en un archivo con las columnas *datetime* y *headline*. Por último, se muestran por pantalla las dimensiones del *DataFrame* donde en este estudio resultan ser (6737 filas, 2 columnas), correspondiendo las filas al número de titulares sobre la compañía durante el rango de fechas considerado.

4.2.2.- Preprocesamiento y Limpieza de Texto

Como se aprecia en la *Figura 10* se declara la función *process_text* con la finalidad de realizar la limpieza y preprocesado de texto, donde se integran varias etapas para obtener un texto depurado, que será utilizado para el análisis de sentimientos.

Empieza por la verificación de cadenas de caracteres como entrada. De lo contrario se devuelve la cadena vacía. Se eliminan las URL y etiquetas HTML, quitando la información irrelevante y que pueda afectar al análisis. Se normalizan los espacios y todo el texto se transforma a minúsculas para mantener la uniformidad en el procesamiento.

También se realiza un filtrado de caracteres donde únicamente pasan las letras, números y algunos caracteres especiales que son importantes (como %, $, #, @), los demás son considerados ruido. Posteriormente, el texto se divide en palabras o tokens facilitando el procesamiento individual de cada término. Se eliminan palabras comunes las cuales son aportan significado relevante al análisis, pero se mantienen palabras como "*not*" o "*never*" para captar matices negativos.

Por último, se aplica lematización para que las palabras estén a su forma base y reconstruir el texto como una cadena limpia.

```python
# Función para limpieza y procesamiento de texto
def process_text(text):
    if not isinstance(text, str):
        return ""

    # Eliminar URLs y etiquetas HTML
    text = re.sub(r'http\S+|www\.\S+', '', text)
    text = re.sub(r'<.*?>', '', text)
    # Eliminar espacios extra y recortar
    text = re.sub(r'\s+', ' ', text).strip()
    # Convertir a minúsculas
    text = text.lower()
    # Filtrar caracteres: se permiten letras, números, espacios y algunos caracteres especiales
    text = re.sub(r'[^a-zA-Z0-9\s%$#@]', '', text)
    # Tokenización
    words = text.split()
    # Eliminar stopwords, conservando "not" y "never"
    stop_words = set(stopwords.words('english')) - {'not', 'never'}
    words = [word for word in words if word not in stop_words]
    # Lematización
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    return ' '.join(words)
```

Figura 10: Función de limpieza y preprocesado del texto

4.3.- Análisis de Sentimientos

Se declara la función *sentiment_score* como una función auxiliar que interpreta el resultado obtenido del análisis de sentimientos. Recibe como entrada el resultado e identifica la etiqueta asociada y, en función de ello, devuelve un *score* positivo o negativo usando el valor de *score* proporcionado. Si la etiqueta no coincide con ninguno de estos casos, retorna 0. Permite transformar el análisis en valores numéricos para que se pueda interpretar en los modelos predictivos.

Con las funciones *analyze_sentiment_bert* y *analyze_sentiment_finbert* representadas en la *Figura 11* se realiza el análisis de sentimientos de los titulares de noticias usando un modelo preentrenado de BERT y otro de FinBERT, los cuales aplican varios pasos.

```python
# Análisis de sentimientos con BERT
def analyze_sentiment_bert(news_df):
    logging.info("Analizando sentimientos con BERT...")
    sentiment_analyzer = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

    # Preprocesamiento de solo titulares
    news_df['cleaned_headline'] = news_df['headline'].apply(process_text)

    # Analizar titulares
    headlines = news_df['cleaned_headline'].tolist()
    headlines_results = sentiment_analyzer(headlines, truncation=True)
    news_df['headline_sentiment'] = [sentiment_score(res) for res in headlines_results]

    daily_sentiments = news_df.groupby('datetime')['headline_sentiment'].mean().reset_index()
    logging.info("Análisis de sentimientos con BERT completado.")
    return daily_sentiments

# Análisis de sentimientos con FinBERT
def analyze_sentiment_finbert(news_df):
    logging.info("Analizando sentimientos con FinBERT...")
    sentiment_analyzer = pipeline("sentiment-analysis", model="ProsusAI/finbert")

    # Preprocesamiento de solo titulares
    news_df['cleaned_headline'] = news_df['headline'].apply(process_text)

    # Analizar titulares
    headlines = news_df['cleaned_headline'].tolist()
    headlines_results = sentiment_analyzer(headlines, truncation=True)
    news_df['headline_sentiment_finbert'] = [sentiment_score(res) for res in headlines_results]

    daily_sentiments_finbert = news_df.groupby('datetime')['headline_sentiment_finbert'].mean().reset_index()
    logging.info("Análisis de sentimientos con FinBERT completado.")
    return daily_sentiments_finbert
```

Figura 11: Funciones de análisis de sentimientos

Primero, aplican la función de *process_text* a cada titular para limpiarlo y normalizarlo. Luego, se hace uso del pipeline de análisis de sentimientos para evaluar cada titular, que junto a la función *sentiment_score* genera un resultado numérico. Dichos resultados se promedian de forma diaria produciendo un índice de sentimiento combinado para cada día.

4.4.- Complementos

Se crea una clase *AttentionLayer* para definir una capa de atención personalizada cuya función principal es ponderar cada paso temporal de una secuencia permitiendo a los modelos concentrase en las partes más relevantes de la información.

Se calcula un puntaje para cada paso aplicando la siguiente fórmula:

$$e = \tanh(x \cdot W + b)$$

Donde:

- ➔ x es la entrada.
- ➔ W es una matriz de pesos.

➔ b es un sesgo entrenable.

Luego se aplica *softmax* para normalizar los puntajes y los pesos resultantes se multiplican por la entrada y se suman a lo largo del tiempo para generar una salida resaltando la información relevante.

La función *nsi* calcula el *Numerical Sentiment Index* (NSI) basado en el movimiento de los precios. Se calcula el retorno mediante la siguiente fórmula [14]:

$$return = \frac{Close_{t+k} - Open_t}{Open_t}$$

Donde:

➔ $Close_{t+k}$ es el precio de cierre en el instante $t + k$.

➔ $Open_t$ es el precio de apertura en el instante t.

Con el retorno calculado, se permite asignar un valor positivo, negativo o nulo al índice representando respectivamente un movimiento alcista, estable o bajista siendo s por defecto 0.01.

$$NSI = \begin{cases} 1 \; si \; return > s \\ 0 \; si \; -s \leq return \leq s \\ -1 \; si \; return < -s \end{cases}$$

La función *lstm_data* prepara y utilizan los datos para entrenar modelos LSTM organizando los datos escalados en secuencia temporales de longitud definidas por el parámetro *look_back*.

4.5.- Modelo Predictivo con Redes LSTM

***train_lstm*:** en la *Figura 12* muestra cómo se utiliza una red LSTM bidireccional para predecir precios financieros basándose en el precio de cierre de una acción. Debajo se muestran los pasos seguidos:

1.- Se extrae la columna Close del DataFrame stock_data y se guarda en raw_data. También se llama a la función lstm_data que transforma la serie temporal en pares de entrada y salida para el modelo.

2.- Los datos son divididos en tres subconjuntos:

➔ Entrenamiento (80%): para ajustar los parámetros del modelo.

- ➔ Validación (10%): para evaluar el rendimiento durante el entrenamiento y poder evitar el sobreajuste.
- ➔ Test (10%): para comprobar la capacidad predictiva de datos no vistos.

3.- Se crean dos escaladores *MinMaxScaler* independientes: uno para las entradas (X) y otro para las salidas (y). Se aplica *fit_transform()* solo sobre los datos de entrenamiento (*X_train_raw* y *y_train_raw*) y luego se aplican los mismos parámetros de escalado a los datos de validación y test para evitar el *data leakage*.

4.- Se define la arquitectura del modelo que se compone de:

- ➔ Una capa de entrada atendiendo únicamente al precio del cierre.
- ➔ Consta de dos capas bidireccionales que permiten capturar información tanto del pasado como del futuro.
- ➔ Capas de *Dropout* para prevenir el sobreajuste.
- ➔ Capa de atención para resaltar las partes más relevantes de la secuencia temporal.
- ➔ Capas Densas para transformar la información que se ha extraído en capas anteriores en la predicción final. Junto a la función de ReLU se introduce no linealidad, permitiendo al modelo capturar relaciones complejas entre los datos.

5.- El modelo se compila con el optimizador Adam con una tasa de aprendizaje del 0,001 y utiliza la función de pérdida de error cuadrático medio (RMSE). Durante el entrenamiento se implementa el *EarlyStopping* para detener el entrenamiento si la pérdida de validación no mejora durante 10 épocas consecutivas y el *ReduceOnPlateau* para reducir la tasa de aprendizaje si la mejora de la validación se estanca.

6.- La función devuelve el modelo entrenado, los escaladores, conjuntos de validación y test y historial de entrenamiento.

```python
# Función de entrenamiento LSTM
def train_lstm(stock_data, look_back):
    # Preparar datos sin escalar
    raw_data = stock_data[['Close']].values
    X, y = lstm_data(raw_data, look_back)
    # Dividir los datos en conjuntos de entrenamiento, validación y test
    train_size = int(len(X) * 0.8)
    val_size = int(len(X) * 0.1)
    X_train_raw, y_train_raw = X[:train_size], y[:train_size]
    X_val_raw, y_val_raw = X[train_size:train_size+val_size], y[train_size:train_size+val_size]
    X_test_raw, y_test_raw = X[train_size+val_size:], y[train_size+val_size:]
    # Creamos escaladores separados para X e y
    scaler_X = MinMaxScaler(feature_range=(0, 1))
    scaler_y = MinMaxScaler(feature_range=(0, 1))

    # Ajuste y transformación de X
    X_train = scaler_X.fit_transform(X_train_raw.reshape(-1, X_train_raw.shape[-1])).reshape(X_train_raw.shape)
    X_val = scaler_X.transform(X_val_raw.reshape(-1, X_val_raw.shape[-1])).reshape(X_val_raw.shape)
    X_test = scaler_X.transform(X_test_raw.reshape(-1, X_test_raw.shape[-1])).reshape(X_test_raw.shape)
    # Ajuste y transformación de y
    y_train = scaler_y.fit_transform(y_train_raw.reshape(-1, 1)).flatten()
    y_val = scaler_y.transform(y_val_raw.reshape(-1, 1)).flatten()
    y_test = scaler_y.transform(y_test_raw.reshape(-1, 1)).flatten()

    # Definición del modelo
    model_price = Sequential([Input(shape=(look_back, 1)),
        Bidirectional(LSTM(128, return_sequences=True, kernel_regularizer=regularizers.l2(0.01))), Dropout(0.2),
        Bidirectional(LSTM(128, return_sequences=True, kernel_regularizer=regularizers.l2(0.01))), Dropout(0.2),
        AttentionLayer(), Dense(64, activation='relu'), Dense(1)])
    model_price.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

    # Callbacks
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)

    logging.info("Entrenando modelo LSTM (solo precios)…")
    history_price = model_price.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100, batch_size=16, verbose=1, callbacks=[early_stopping, reduce_lr])

    return model_price, scaler_X, scaler_y, X_val, y_val, X_test, y_test, history_price
```

Figura 12: Modelo LSTM

***train_lstm_bert*:** en la Figura 13 se muestra un modelo LSTM bidireccional que integra tres variables: el precio de cierre, el sentimiento derivado de un modelo BERT y un índice de sentimiento de noticias (NSI).

1.- Se alinean las fechas de los sentimientos de noticias con las fechas del *stock_data*. En caso de que haya fechas sin sentimientos se rellenan con 0.

2.- La variable raw_data contine las tres variables mencionadas (Close, Sentiment_BERT y NSI) por tiempo.

3.- Los datos son divididos de la misma manera que el modelo LSTM puro:

- ➔ Entrenamiento (80%)
- ➔ Validación (10%)
- ➔ Test (10%)

4.- Se escala *X* y *y* usando solo los datos de entrenamiento con fit_transform(), luego se transforman los demás.

5.- La arquitectura del modelo es similar a train_lstm pero ahora la entrada de dimensión es (look_back, 3) debido a las 3 variables.

6.- Los valores tanto de Dropout como learning_rate para este modelo en comparación al LSTM puro son más altos, 0,3 y 0,0017 respectivamente. Además, se implementa únicamente *ReduceLROnPlateau*. También, se incrementa el número de épocas a 150 y el batch_size a 64.

7.- La función devuelve el modelo entrenado, los escaladores, conjuntos de validación y test y historial de entrenamiento para un análisis posterior.

```
# Función de entrenamiento LSTM + BERT
def train_lstm_bert(stock_data, news_sentiments_bert, look_back):
    # Añadir sentimiento
    stock_data['Sentiment_BERT'] = stock_data['Date'].map(news_sentiments_bert.set_index('datetime')['headline_sentiment'])
    stock_data.fillna(0, inplace=True)
    # Preparar datos
    raw_data = stock_data[['Close', 'Sentiment_BERT', 'NSI']].values
    X, y = lstm_data(raw_data, look_back)
    # Dividir los datos en conjuntos de entrenamiento, validación y test
    train_size = int(len(X) * 0.8)
    val_size = int(len(X) * 0.1)
    X_train_raw, y_train_raw = X[:train_size], y[:train_size]
    X_val_raw, y_val_raw = X[train_size:train_size+val_size], y[train_size:train_size+val_size]
    X_test_raw, y_test_raw = X[train_size+val_size:], y[train_size+val_size:]
    # Escaladores separados
    scaler_X = MinMaxScaler()
    scaler_y = MinMaxScaler()

    # Transformación X
    X_train = scaler_X.fit_transform(X_train_raw.reshape(-1, X_train_raw.shape[-1])).reshape(X_train_raw.shape)
    X_val = scaler_X.transform(X_val_raw.reshape(-1, X_val_raw.shape[-1])).reshape(X_val_raw.shape)
    X_test = scaler_X.transform(X_test_raw.reshape(-1, X_test_raw.shape[-1])).reshape(X_test_raw.shape)
    # Transformación y
    y_train = scaler_y.fit_transform(y_train_raw.reshape(-1, 1)).flatten()
    y_val = scaler_y.transform(y_val_raw.reshape(-1, 1)).flatten()
    y_test = scaler_y.transform(y_test_raw.reshape(-1, 1)).flatten()

    # Definición del modelo
    model_bert = Sequential([Input(shape=(look_back, 3)),
        Bidirectional(LSTM(128, return_sequences=True, kernel_regularizer=regularizers.l2(0.01))), Dropout(0.3),
        Bidirectional(LSTM(128, return_sequences=True, kernel_regularizer=regularizers.l2(0.01))), Dropout(0.3),
        AttentionLayer(), Dense(64, activation='relu'), Dense(1)])
    model_bert.compile(optimizer=Adam(learning_rate=0.0017), loss='mean_squared_error')

    # Callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)

    logging.info("Entrenando modelo LSTM + BERT...")
    history_bert = model_bert.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=150, batch_size=64, verbose=1, callbacks=[reduce_lr])

    return model_bert, scaler_X, scaler_y, X_val, y_val, X_test, y_test, history_bert
```

Figura 13: Modelo LSTM + BERT

***train_lstm_finbert*:** este modelo sigue exactamente los mismos pasos que el modelo de BERT pero cambiando el análisis de sentimiento el cual será dado por un modelo preentrenado de FinBERT tal y como ilustra la *Figura 14*.

```python
# Función de entrenamiento LSTM + FinBERT
def train_lstm_finbert(stock_data, news_sentiments_finbert, look_back):
    # Añadir sentimiento FinBERT
    stock_data['Sentiment_FinBERT'] = stock_data['Date'].map(news_sentiments_finbert.set_index('datetime')['headline_sentiment_finbert'])
    stock_data.fillna(0, inplace=True)
    # Preparar datos
    raw_data = stock_data[['Close', 'Sentiment_FinBERT', 'NSI']].values
    X, y = lstm_data(raw_data, look_back)
    # Dividir los datos en conjuntos de entrenamiento, validación y test
    train_size = int(len(X) * 0.8)
    val_size   = int(len(X) * 0.1)
    X_train_raw, y_train_raw = X[:train_size], y[:train_size]
    X_val_raw,   y_val_raw = X[train_size:train_size+val_size], y[train_size:train_size+val_size]
    X_test_raw,  y_test_raw = X[train_size+val_size:], y[train_size+val_size:]
    # Escaladores separados
    scaler_X = MinMaxScaler()
    scaler_y = MinMaxScaler()

    # Transformación X
    X_train = scaler_X.fit_transform(X_train_raw.reshape(-1, X_train_raw.shape[-1])).reshape(X_train_raw.shape)
    X_val = scaler_X.transform(X_val_raw.reshape(-1, X_val_raw.shape[-1])).reshape(X_val_raw.shape)
    X_test = scaler_X.transform(X_test_raw.reshape(-1, X_test_raw.shape[-1])).reshape(X_test_raw.shape)
    # Transformación y
    y_train = scaler_y.fit_transform(y_train_raw.reshape(-1, 1)).flatten()
    y_val = scaler_y.transform(y_val_raw.reshape(-1, 1)).flatten()
    y_test = scaler_y.transform(y_test_raw.reshape(-1, 1)).flatten()

    # Definición del modelo
    model_finbert = Sequential([Input(shape=(look_back, 3)),
        Bidirectional(LSTM(128, return_sequences=True, kernel_regularizer=regularizers.l2(0.01))), Dropout(0.3),
        Bidirectional(LSTM(128, return_sequences=True, kernel_regularizer=regularizers.l2(0.01))), Dropout(0.3),
        AttentionLayer(), Dense(64, activation='relu'), Dense(1)])
    model_finbert.compile(optimizer=Adam(learning_rate=0.0017), loss='mean_squared_error')

    # Callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)

    logging.info("Entrenando modelo LSTM + FinBERT...")
    history_finbert = model_finbert.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=150, batch_size=64, verbose=1, callbacks=[reduce_lr])

    return model_finbert, scaler_X, scaler_y, X_val, y_val, X_test, y_test, history_finbert
```

Figura 14: Modelo LSTM + FinBERT

4.6.- Evaluación y visualización de resultados

Declaramos la función *mape* que se encarga de calcular el *Mean Absolute Percentage Error* (MAPE) la cual indica el error porcentual medio entre las predicciones y los valores reales. La fórmula es la siguiente:

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Donde:

- ➔ y_i son los valores reales.
- ➔ $\hat{y}_i$ son las predicciones.
- ➔ n es el número de muestras.

La función *plot_predictions* realiza predicciones tanto para la fase de validación como la de test en los tres modelos. Cada modelo realiza una predicción de valores los cuales se pasan a un vector unidimensional. Previamente los valores se habían desnormalizado por lo que ahora dichos datos se vuelven a su escala original.

Se calculan las siguientes métricas para cada uno de los tres modelos:

➔ RMSE (Error Cuadrático Medio): calcula la raíz cuadrada del promedio de los errores al cuadrado entre los valores reales y las predicciones. Un valor de RMSE menor indica que las predicciones se acercan más a los valores reales [15].

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

➔ MAE (Error Absoluto Medio): calcula el promedio de las diferencias absolutas entre los valores reales y las predicciones. Se mide el error sin importar la dirección [16].

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

➔ R^2 (Coeficiente de Determinación): mide que tan bien se ajustan los datos a la línea de regresión, tomando valores entre 0 y 1 donde 1 es un ajuste perfecto.

Se genera un gráfico visualizado en la *Figura 15* que muestra los precios reales y las predicciones tanto en fase de validación como test. Esto permite comparar el desempeño de los tres modelos a lo largo del tiempo.

```
# Gráfico comparativo
plt.figure(figsize=(14,7))
dates = stock_data['Date'][-(len(y_val_price)+len(y_test_price)):]
real_prices = np.concatenate([y_val_price_unscaled, y_test_price_unscaled])

plt.plot(dates, real_prices, label='Precios Reales', color='blue')
plt.plot(dates[:len(y_val_price)], price_predictions_val_unscaled, label='LSTM (Validación)', color='green')
plt.plot(dates[:len(y_val_bert)], bert_predictions_val_unscaled, label='LSTM+BERT (Validación)', color='red')
plt.plot(dates[:len(y_val_finbert)], finbert_predictions_val_unscaled, label='LSTM+FinBERT (Validación)', color='orange')
plt.plot(dates[len(y_val_price):], price_predictions_test_unscaled, label='LSTM (Test)', linestyle='--', color='darkgreen')
plt.plot(dates[len(y_val_bert):], bert_predictions_test_unscaled, label='LSTM+BERT (Test)', linestyle='--', color='darkred')
plt.plot(dates[len(y_val_finbert):], finbert_predictions_test_unscaled, label='LSTM+FinBERT (Test)', linestyle='--', color='darkorange')
plt.xlabel("Fecha")
plt.ylabel("Precio")
plt.title("Predicciones de precios (Validación y Test)")
plt.legend()
plt.show()
```

Figura 15: Gráfico de predicciones

La función *plot_curves* ilustrada en la *Figura 16*, permite visualizar las curvas de entrenamiento y validación para cada uno de los modelos. Útil para detectar sobreajuste o subajuste ya que permite observar cómo evolucionan las pérdidas a medida que se entrena el modelo.

```python
# Visualizar curvas de pérdida de entrenamiento y validación
def plot_curves(histories, model_names, log_scale=False):

    for history, name in zip(histories, model_names):
        loss = history.history['loss']
        val_loss = history.history['val_loss']
        epochs = range(1, len(loss) + 1)

        plt.figure(figsize=(12, 5))
        plt.plot(epochs, loss, label='Entrenamiento')
        plt.plot(epochs, val_loss, label='Validación')
        plt.xlabel('Épocas')
        plt.ylabel('Pérdida')
        plt.title(f'Curva de Entrenamiento y Validación - {name}')
        if log_scale:
            plt.yscale('log')
        plt.legend()
        plt.grid(True)
        plt.show()
```

Figura 16: Gráfico de curvas entrenamiento y validación

5.- Experimentación

En este capítulo se lleva a cabo un estudio sobre como las distintas decisiones de diseño afectan al rendimiento del modelo genérico BERT en la predicción de precios, evaluado mediante RMSE y MAE. Además, se ilustrarán los diferentes enfoques en gráficas comparativas.

5.1.- Impacto de las características

Durante la programación del código se utilizan diversos enfoques para mejorar la predicción de precios. Entre ellos destaca el impacto de las características del código en los resultados obtenidos para el modelo BERT al ser el genérico.

NSI: Se evalúa tanto la utilización de NSI como sin NSI, donde se comparan sus predicciones contra los precios reales en los dos conjuntos de validación y test. La integración del NSI dentro de un modelo de predicción de precios para capturar de forma numérica la percepción en el mercado que no se ve reflejado en los datos únicamente numéricos muestra en los resultados como reduce el error de predicción, es decir, mejora la aproximación de los valores reales.

En el gráfico de la *Figura 17* se observa como en ciertos tramos la diferencia entre con NSI y sin NSI puede ser mínima, lo que sugiere que el impacto del NSI varía según la magnitud de los cambios de sentimiento y su alineación con los movimientos del precio.

NSI	RMSE	MAE
Sin NSI	5,34	4,01
Con NSI	3,43	2,47

Tabla 1: Comparativa Numerical Sentiment Index

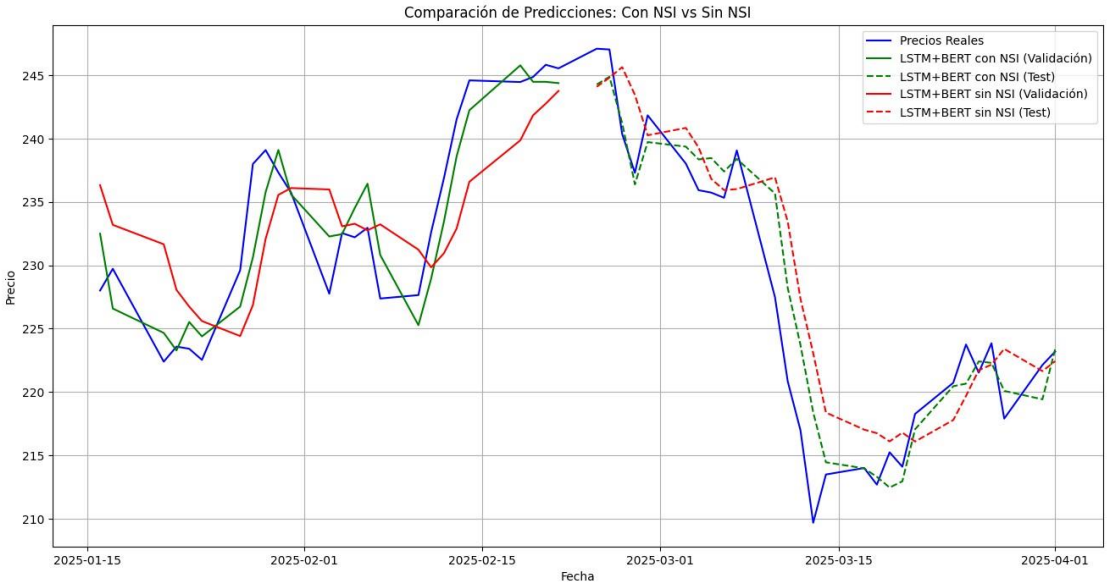

Figura 17: Comparativa desempeño NSI

Look Back: se utilizan tres configuraciones distintas de *look_back* (5, 7 y 10) para comparar su desempeño en las predicciones de precios mostrado en la *Figura 18*, destacando la influencia de la cantidad de días previos de información. A continuación, se presentan los siguientes valores de RMSE y MAE donde se aprecia como los valores de look back 5 tienen los valores más bajos, correspondiéndose en el gráfico con un mejor ajuste al comportamiento real de los precios.

LOOK BACK	RMSE	MAE
5	4,91	3,68
7	5,33	4,01
10	6,17	4,74

Tabla 2: Comparativa Look Back

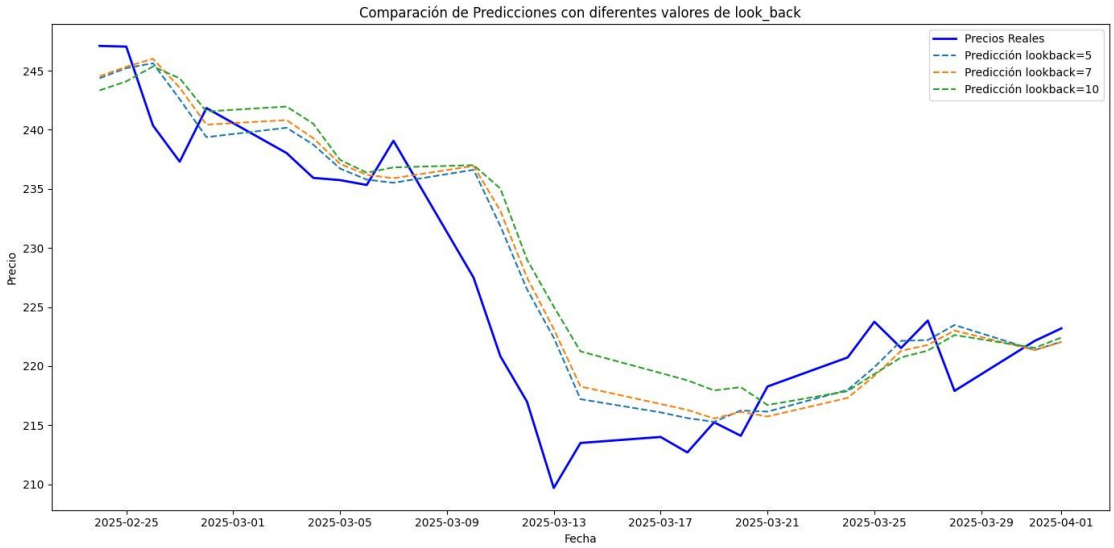

Figura 18: Comparativa desempeño Look Back

5.2.- Impacto de los hiperparámetros

Se realiza otro estudio alrededor de los hiperparámetros comparando el desempeño del modelo con análisis de sentimientos BERT con los diferentes valores de los mismos. Del estudio mencionado se destacan los siguientes hiperparámetros:

Capa de atención: La incorporación de mecanismos de atención tiene como objetivo aliviar la carga que tienen las LSTM para capturar de forma uniforme toda la información relevante en la secuencia. Con su adicción, la capa de atención aprende a asignar pesos mayores a los ramos temporales más informativos para predecir el valor futuro donde en la *Figura 19* se aprecia como las predicciones son más ajustadas. Al comparar las métricas, el modelo con atención presenta una pequeña reducción en el error resultado en decisiones de inversión más precisas.

ATENCIÓN	RMSE	MAE
Sin Capa	3,64	2,66
Con Capa	3,40	2,46

Tabla 3: Comparativa Capa de Atención

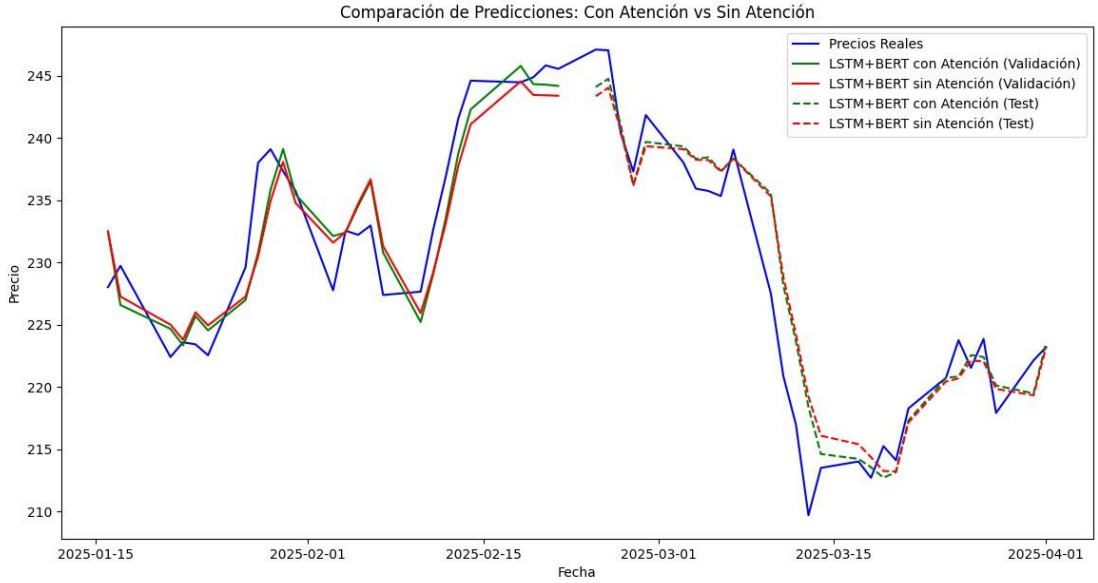

Figura 19: Comparativa desempeño Capa de Atención

Dropout: es utilizado para mitigar el sobreajuste desactivando parte de las neuronas durante el entrenamiento, influyendo en la capacidad del modelo de depender de conexiones específicas. En el estudio se comparan diferentes tasas de *dropout* (0,2 0,3 y 0,4).

La variación de la tasa de *dropout* incluye en la precisión del modelo donde en los resultados expuestos en la *Figura 20* se aprecia como un *dropout* moderado con 0,3 de valor, ofrece un buen rendimiento en términos de RMSE y MAE.

DROPOUT	RMSE	MAE
0,2	3,57	2,61
0,3	3,36	2,44
0,4	3,63	2,67

Tabla 4: Comparativa Dropout

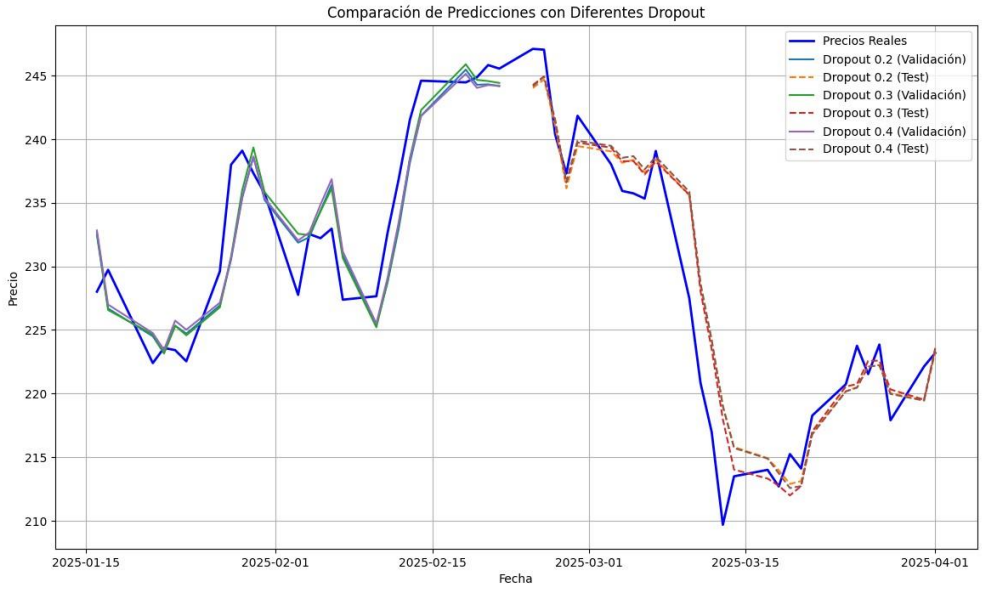

Figura 20: Comparativa desempeño Dropout

6.- Análisis comparativo y resultados

En este capítulo se presentan primero las estadísticas descriptivas de los datos históricos y los valores de sentimiento generados, junto con la distribución del índice NSI. A continuación, se ilustran las curvas de pérdida durante el entrenamiento de los tres modelos. Finalmente, se visualizan las predicciones en los conjuntos de validación y test comparando también sus métricas

6.1.- Visualización de datos

Tras ejecutar el código procedemos a visualizar el contenido de los datos descargados. Por un lado, en la *Figura 21* se observan como los datos de los históricos presentan las columnas iniciales de: Date, Open, High, Low, Close y Volume donde luego se añaden otras, entre ellas *Sentiment_BERT* y *Sentiment_FinBERT* que nos informan del puntaje de sentimiento basados en modelos de lenguaje BERT y FinBERT. También contemplamos la columna de NSI que indica si se han producido cambios significativos en el precio:

Primeras filas de stock_data:

	Date	Open	High	Low	Close	Volume	NSI	Sentiment_BERT	Sentiment_FinBERT
0	2024-03-11	172.127235	173.560470	171.241418	171.938126	60139500	0	-0.266323	-0.412865
1	2024-03-12	172.336237	173.212106	170.206295	172.415863	59825400	-1	-0.438975	0.000000
2	2024-03-13	171.958052	172.376076	169.957488	170.325760	52488700	0	-0.214293	-0.029914
3	2024-03-14	172.097377	173.490792	171.241418	172.186951	72913500	0	-0.614233	-0.035770
4	2024-03-15	170.365564	171.808746	169.489695	171.808746	121664700	1	-0.435276	-0.221057

Figura 21: Datos históricos

También se realiza un resumen estadístico visualizado en la *Figura 22* sobre las características del conjunto de datos donde se destaca:

- ➔ El precio medio de la acción de Apple durante el período analizado es alrededor de 216 – 218$.
- ➔ La desviación estándar presenta un valor próximo a 24,5 indicando que la acción tiene cierta volatilidad.
- ➔ Se registran los valores mínimos y máximos donde la diferencia significativa revela la amplitud del rango de precios durante el periodo estudiado, asociado a eventos relevantes que han ocurrido en el mercado o compañía.

Resumen estadístico de los datos de precios:

	Open	High	Low	Close	Volume	NSI	Sentiment_BERT	Sentiment_FinBERT
count	266.000000	266.000000	266.000000	266.000000	2.660000e+02	266.000000	266.000000	266.000000
mean	216.063743	218.323849	214.043174	216.321785	5.561970e+07	0.093985	-0.392034	-0.055680
std	24.306496	24.535320	23.981100	24.392036	3.005689e+07	0.733692	0.339322	0.193986
min	164.572913	165.617978	163.308889	164.224564	2.323470e+07	-1.000000	-0.999581	-0.921988
25%	205.010971	210.799026	201.722030	206.956947	4.028265e+07	0.000000	-0.602537	-0.149845
50%	223.790695	225.533626	221.339702	223.571404	4.849460e+07	0.000000	-0.393668	-0.039575
75%	231.700250	233.693061	229.125710	232.419907	6.076482e+07	1.000000	-0.171254	0.061659
max	257.906429	259.814335	257.347047	258.735504	3.186799e+08	1.000000	0.978664	0.442744

Figura 22: Estadísticas datos históricos

Se observa en la *Figura 23* un gráfico del índice NSI, el predominio del sentimiento neutro indicando que gran parte de las noticias no expresan un sentimiento claro o han sido balanceadas en su contenido, siendo común en informes financieros técnicos que simplemente reportan los hechos.

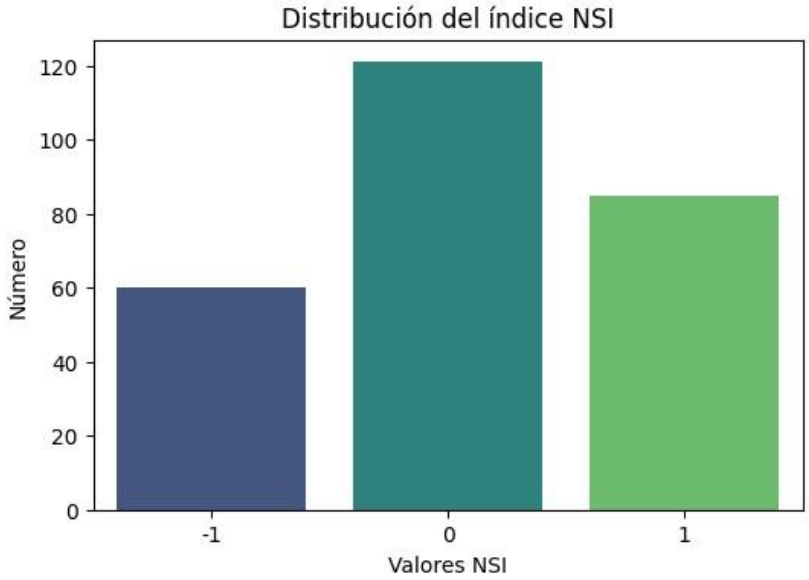

Figura 23: Resultados índice NSI

6.2.- Entrenamiento de los modelos

Modelo LSTM: el entrenamiento de este modelo observado en la *Figura 24* presenta una notable reducción de la pérdida indicando que aprende rápidamente las tendencias generales de los datos. Después de dicha reducción, tanto la pérdida de entrenamiento como la de validación continúan reduciéndose y se estabilizan en valores muy bajos (0.003) en las últimas épocas. La similitud entre el *loss* y el *val_loss* indica que el modelo no está sobreajustándose ya que mantiene una buena capacidad de generalización. El *learning rate* se mantiene constante en 0.001 hasta que rondando la época 50 se reduce, pero las pérdidas decrecen muy ligeramente. La convergencia de las curvas a valores bajos da a entender que el modelo ha captado las características esenciales de los datos históricos.

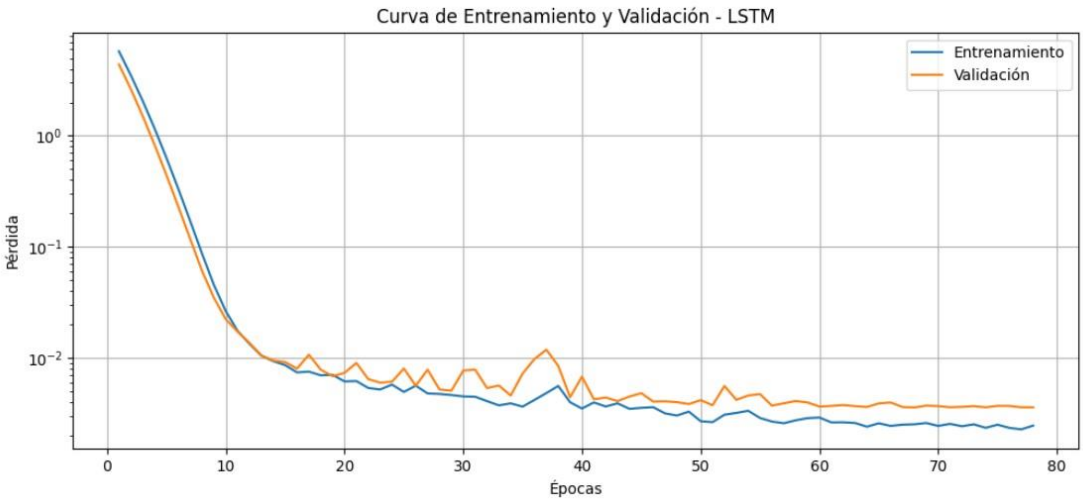

Figura 24: Curvas entrenamiento y validación - Modelo LSTM

Modelo LSTM + BERT y **LSTM + FinBERT:** el entrenamiento de estos modelos apreciados en las *Figuras 25* y *Figura 26* empiezan con pérdidas mayores al modelo LSTM, pero tras las 20 primeras épocas se observa una rápida reducción indicando que aprende de forma efectiva las relaciones entre los datos históricos y los sentimientos devueltos por BERT. Hay que destacar que a medida que pasan las épocas el *learning rate* se va reduciendo progresivamente hasta el 1^{-6} en las últimas épocas. Con este método se consigue refinar el entrenamiento y evitar oscilaciones en la pérdida para acercarse lo máximo posible al óptimo. Desde la época 100 a la 150, las pérdidas se mantienen casi constantes, indicando que los modelos ya han aprendido la mayoría de las características relevantes y que se están realizando un ajuste fino de los pesos sin riesgo de sobreajuste, ya que la pérdida de validación no se dispara, sino que se mantiene alineada con la de entrenamiento.

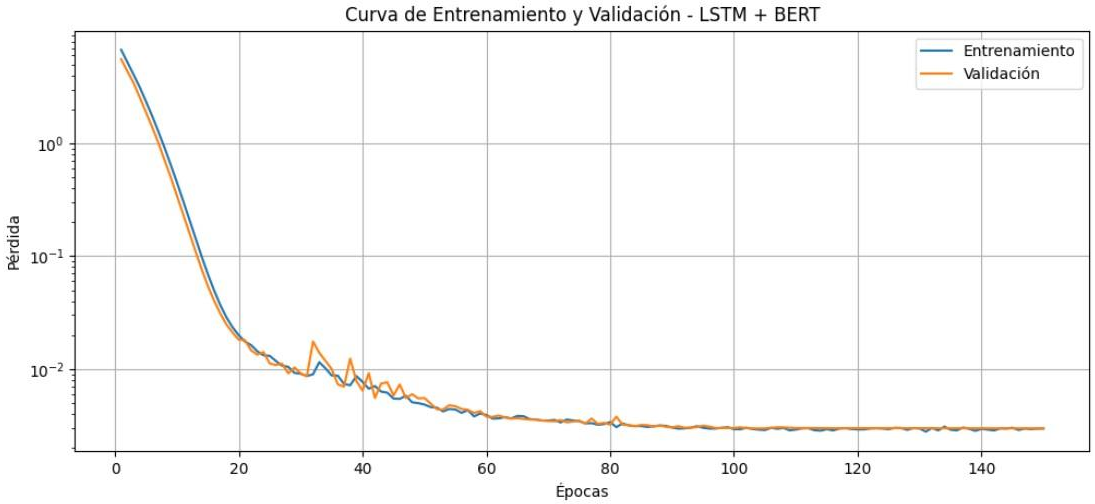

Figura 25: Curvas entrenamiento y validación - Modelo LSTM + BERT

Figura 26: Curvas entrenamiento y validación - Modelo LSTM + FinBERT

6.3.- Resultados de los modelos

Se realizan las predicciones para los tres modelos donde se observa en la *Figura 27* por un lado los trazos continuos que corresponden a la fase de validación y también los trazos discontinuos que reflejan la fase de prueba. Además, en la tabla se incluyen las métricas de evaluación para cada modelo:

Modelo	RMSE	MAE	R^2	MAPE
LSTM	5,10	3,74	0,79	1,67%
LSTM + BERT	3,28	2,38	0,91	1,06%
LSTM + FinBERT	3,00	2,19	0,93	0,97%

Tabla 5: Comparativa desempeño de los modelos

- ➔ **RMSE y MAE:** el modelo LSTM puro que solo emplea precios de históricos obtiene un RMSE de 5,10 y MAE de 3,74, dichos valores se ven reducidos en los otros dos modelos que incorporan las variables de sentimiento con valores de RMSE cercanos a 3,1 y MAE de 2,2.
- ➔ **Coeficiente de Determinación (R^2):** el modelo LSTM presenta un coeficiente de 0.79, es decir, explica aproximadamente el 79% de la variabilidad de los precios en el conjunto de test. Por otro lado, incluir modelos con variables de sentimiento mediante BERT o FinBERT eleva el coeficiente de determinación a valores superiores de 0.9. Este incremento se traduce en que la adición de sentimiento a los modelos logra capturar más del 90% de la variabilidad de los precios.
- ➔ **Error Porcentual Medio Absoluto (MAPE):** el modelo LSTM alcanza un 1,67% mientras que los otros modelos presentan resultados de 1,06% y 0,97%. La reducción de MAPE en los modelos con análisis de sentimientos presentan mejores resultados no solo en términos absolutos sino también en términos relativos.

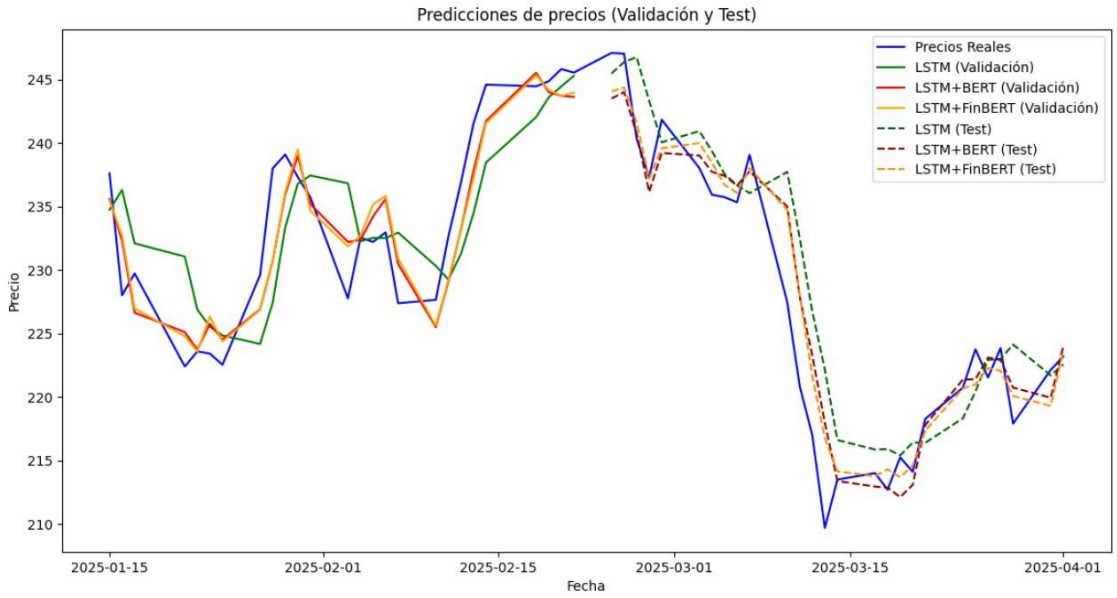

Figura 27: Comparativa predicción de precios modelos

En la figura se aprecia como las líneas de validación se superponen con la serie real de precios, aunque la de LSTM puro tiene un poco de desplazamiento. Por otra parte, en las líneas de test se observa como el modelo LSTM + FinBERT se ajusta con mayor precisión a los picos de la serie temporal. De forma clara se contempla como los modelos que incorporan sentimiento superan en resultados al modelo LSTM, pero fijándonos en las métricas comentadas anteriormente la diferencia entre modelo LSTM + BERT y

LSTM + FinBERT es mínima. Pese a que el modelo FinBERT es un modelo preentrenado específicamente en el dominio financiero, no consigue unos resultados notorios que indiquen un gran incremento en la precisión respecto al modelo genérico de BERT.

7.- Conclusiones y trabajos futuros

capítulo repasa el impacto de incorporar información de sentimiento en los modelos de predicción de series temporales. También se tratan las limitaciones del proyecto y se presentan propuestas de mejora.

7.1.- Interpretación de los resultados

En rasgos generales, la adicción de variables de sentimientos haciendo uso de modelos de lenguaje BERT y FinBERT, ha demostrado mejorar de forma consistente la capacidad de predicción de las series temporales de precios. Aunque el modelo LSTM puro presenta una base sólida, más allá de los datos únicamente números, la integración de información cualitativa aporta nuevas perspectivas que ofrecen una reducción en el error de la predicción y aumenta la explicación de la variabilidad de los datos. En definitiva, los resultados indican que en el campo financiero la combinación de datos cuantitativos e información de texto beneficia los modelos de predicción.

7.2.- Limitaciones del estudio

El análisis realizado se ha centrado exclusivamente en la acción de Apple y en un conjunto de noticias y precios recogidos a través de unas APIs, lo que restringe la capacidad de generalizar los resultados para otros valores o sectores que no sea el tecnológico. También, se podría contemplar un mayor rango de fechas de precios para capturar todavía mejor las tendencias, así como noticias provenientes de otras fuentes como pueden ser las redes sociales o los foros teniendo una mayor información sobre los hechos acontecidos y enfoques de los acontecimientos que han surgido o surgen a lo largo del tiempo.

7.3.- Pasos futuros y mejoras

Teniendo en cuenta lo mencionado en el apartado anterior, se podría añadir al código los *tickers* de otras compañías de distintos sectores o incluso analizar activos alternativos como criptomonedas o ETFs. Además, se podría pagar a APIs las cuales ofrecen precios de históricos y noticias ya correlacionadas durante 2 años de antigüedad

como es el caso de News API entre otras, ya que las versiones gratuitas están muy limitadas.

También sería conveniente desarrollar una interfaz gráfica de usuario que permita de manera automática al usuario, ajustar parámetros clave como el rango de fechas, la selección de la acción o modelo de sentimiento sin tener que editar el código fuente, siendo más sencillo y de fácil adopción para perfiles no técnicos.

8.- Bibliografía

[1] Cloudflare. (s. f.). *¿Qué es una red neuronal?* https://www.cloudflare.com/es-es/learning/ai/what-is-neural-network/

[2] Ichi.Pro. (2021, 5 de enero). *Principales tipos de redes neuronales y sus aplicaciones – Tutorial*. https://ichi.pro/es/principales-tipos-de-redes-neuronales-y-sus-aplicaciones-tutorial-126738056604814

[3] DataScientest. (2024, 20 de mayo). *Memoria a largo plazo a corto plazo (LSTM): ¿Qué es?* https://datascientest.com/es/memoria-a-largo-plazo-a-corto-plazo-lstm

[4] Lov Technology. (s. f.). *Guía completa de redes neuronales LSTM y su aplicación*. https://lovtechnology.com/red-neuronal-lstm/

[5] QuestionPro. (s. f.). *Procesamiento del lenguaje natural (PLN): Qué es, usos y ejemplo*. https://www.questionpro.com/blog/es/procesamiento-del-lenguaje-natural/

[6] IBM. (2024, 11 de agosto). *¿Qué es el PLN (procesamiento de lenguaje natural)?* https://www.ibm.com/es-es/think/topics/natural-language-processing

[7] OpenWebinars. (2024, 2 de julio). *Procesamiento de Lenguaje Natural: Qué es y cómo funciona*. https://openwebinars.net/blog/procesamiento-de-lenguaje-natural/

[8] TestingBaires. (2024, 23 de mayo). *Modelo de Proceso de Lenguaje Natural*, https://testingbaires.com/modelo-de-proceso-de-lenguaje-natural/

[9] Datascientest.com. (2024, 1 de agosto). *NLP Natural Language Processing: Introducción*. https://datascientest.com/es/nlp-introduccion

[10] Delatorre.ai. (s. f.). *Los transformers en la inteligencia artificial: una explicación sencilla*. https://delatorre.ai/los-transformers-en-la-inteligencia-artificial-una-explicacion-sencilla/

[11] Amazon Web Services. (s. f.). *¿Qué son los transformadores en la inteligencia artificial?* https://aws.amazon.com/es/what-is/transformers-in-artificial-intelligence/

[12] DataCamp. (2024, 11 de septiembre). *¿Qué es el BERT? Introducción a los modelos BERT*. https://www.datacamp.com/es/blog/what-is-bert-an-intro-to-bert-models

[13] Prosus AI Tech Blog. (2020, agosto 19). *FinBERT: Financial Sentiment Analysis with BERT*. https://medium.com/prosus-ai-tech-blog/finbert-financial-sentiment-analysis-with-bert-b277a3607101

[14] Jiang, T., & Zeng, Q. (2023, 3 de junio). *Financial sentiment analysis using FinBERT with application in predicting stock movement*. arXiv. https://arxiv.org/abs/2306.02136

[15] Sonkiya, P., Bajpai, V., & Bansal, A. (2021, 18 de julio). *Stock price prediction using BERT and GAN*. arXiv. https://arxiv.org/abs/2107.09055

[16] Halder, S. (2022, 11 de noviembre). *FinBERT-LSTM: Deep Learning based stock price prediction using News Sentiment Analysis*. arXiv. https://arxiv.org/abs/2211.07392

9.- Anexos

Este capítulo final presenta el enlace al directorio del código y ficheros utilizados en el desarrollo del estudio.

9.1.- Código fuente

A continuación, tienes el enlace al repositorio público en GitHub donde se ha volcado el material comentado en este estudio:

https://github.com/N0X-y/stock_prediction

En el enlace se encuentra:

- ➔ *README.md*: descripción general del proyecto y explicación del flujo de trabajo.
- ➔ *AAPL_historical_data.csv*: precios históricos de *Apple* para el periodo analizado.
- ➔ *AAPL_news_data.csv*: titulares relacionados con la compañía para el rango de fechas.
- ➔ *stock_prediction.ipynb*: cuaderno Jupyter que contiene el código fuente con todas las funciones y resultados obtenidos.