



UNIVERSIDAD DE GRANADA

Facultad de Ciencias

E.T.S. de Ingenierías Informática y de Telecomunicación

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y
MATEMÁTICAS

TRABAJO DE FIN DE GRADO

Uso de IA en la selección de activos financieros: Aplicaciones del Deep Learning en el diseño de estrategias de inversión basadas en momentos

Presentado por:
Manuel Díaz-Meco Terrés

Curso académico 2025-2026

Uso de IA en la selección de activos financieros: Aplicaciones del Deep Learning en el diseño de estrategias de inversión basadas en momentos

Manuel Díaz-Meco Terrés

Manuel Díaz-Meco Terrés *Uso de IA en la selección de activos financieros: Aplicaciones del Deep Learning en el diseño de estrategias de inversión basadas en momentos* .

Trabajo de fin de Grado. Curso académico 2025-2026.

Responsable de tutorización

Fernando Berzal Galiano
Departamento de Ciencias de la Computación e Inteligencia Artificial

Doble Grado en Ingeniería Informática y Matemáticas
Facultad de Ciencias

E.T.S. de Ingenierías Informática y de Telecomunicación

Universidad de Granada

DECLARACIÓN DE ORIGINALIDAD

D. Manuel Díaz-Meco Terrés

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2025-2026, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 6 de noviembre de 2025

Fdo: Manuel Díaz-Meco Terrés

No te dejes zarandear; por el contrario, en todo impulso, corresponde con lo justo, y en toda fantasía, conserva la facultad de comprender.

Marco Aurelio
Meditaciones

Agradecimientos

Este trabajo marca el final de cinco intensos años en el doble grado de Ingeniería Informática y Matemáticas. Supone la culminación de una etapa clave en mi vida, llena de retos, aprendizajes y experiencias que me han ayudado a crecer tanto personal como profesionalmente. Sin embargo, este camino no lo he recorrido solo, y quiero expresar mi más sincero agradecimiento a todas las personas que, de una forma u otra, me han acompañado y apoyado para llegar hasta aquí.

A mi madre, por haberme apoyado siempre y ser un pilar esencial durante estos cinco años, no lo podría haber conseguido sin ti. A mi padre, por darme la calma y la serenidad en los momentos más difíciles, por escucharme con paciencia y por recordarme siempre lo que de verdad importa. A mi hermana, por ser la alegría de la casa y transmitirme siempre tu buen humor y alegría.

A Lucía, por ser una inspiración constante y una persona a la que admiro profundamente. Gracias por alegrarme los días, por estar en los malos momentos y por hacer que todo tenga más sentido. Nos queda mucho por recorrer juntos.

A los jugones: Ventura, Mata, Juan, Ismael, Roberto, Avilés, Chicho, Reyes, Rojas, Víctor, Carrillo, Valeria, Lydia y Callejas. Gracias por haber aparecido en mi vida. Sois un apoyo fundamental y con vosotros todo ha sido más fácil.

A la gente que he conocido en la carrera, tanto de mi año como de un año menos. Isa y Kessler, aunque nos hayamos conocido en mi segunda mitad de la carrera, sois de lo mejor que me llevo. Ana, dúo Serrallo, por tu corazón tan puro y tu infinita empatía. Javi, por tu vitalidad y energía. Aarón, por tu sabiduría y tu calma. Clara, por tu alegría y sensatez. Soledad, por haber sido uno de mis pilares durante estos años, te admiro muchísimo.

A Elytras: Ramón, Sánchez y Sergio. En especial a Ramón, por haberme acogido en su casa durante varios años y compartir contigo un ratillo de paz cada semana. Paula, por tu bondad y tu dulzura. Sara, por todos esos cafés a las seis de la tarde en el Triunfo.

A mi tutor, Fernando Berzal. Por tu paciencia y todo lo que me has enseñado. Me siento muy afortunado por el tutor que he escogido. Gracias por las tutorías en las que acabábamos hablando de libros y curiosidades.

A Inversión Racional, en especial a Luis Miguel, por haberme hecho caer por la madriguera. Afortunadamente, no creo que salga nunca más.

Y, por último, a todos aquellos que han puesto su conocimiento a disposición de los demás —a través de apuntes, vídeos, pódcasts o libros—, gracias por permitirme aprender de vosotros, por dejarme subir a los hombros de tantos gigantes.

Resumen

Los mercados financieros constituyen una de las piezas más importantes de la economía moderna. Cada día, millones de inversores compran y venden activos movidos por sus expectativas, temores y estrategias, haciendo que los precios fluctúen sin descanso. Una de las estrategias más populares de las últimas décadas es la conocida como "inversión basada en momentos" —conocida internacionalmente como *momentum investing*—.

El fundamento de esta estrategia es sencillo: comprar los activos que mejor se han comportado en el pasado reciente y vender aquellos con peor desempeño. Aunque simple, la inversión basada en momentos ha demostrado resultados empíricamente positivos, superando en muchos períodos a la media del mercado. Sin embargo, esta estrategia no está exenta de críticas. Algunos reputados inversores, como Benjamin Graham, advertían ya que «*el inversor inteligente nunca debe prever el futuro exclusivamente mediante una extrapolación del pasado*». Intuitivamente, es lógico pensar que, en un entorno financiero caracterizado por la alta volatilidad y la complejidad estructural de los mercados, la inversión basada en momentos debería presentar ciertas debilidades.

El presente Trabajo Fin de Grado surge precisamente de esa reflexión: ¿es posible mejorar la estrategia tradicional de inversión por momentos incorporando técnicas de aprendizaje automático y modelos predictivos avanzados? Para responder a esta pregunta, se ha desarrollado un proyecto que combina los campos de las matemáticas, la informática y las finanzas, integrando modelos clásicos y modernos de *machine learning* aplicados a datos históricos de mercado. El objetivo ha sido comparar el rendimiento de carteras basadas en la estrategia de Andreas F. Clenow —descrita en su libro *Stocks on the Move* y fundamentada en la inversión basada en momentos— con las obtenidas mediante modelos predictivos, analizando sus resultados en términos de rentabilidad y riesgo.

Desde un punto de vista teórico, correspondiente a la primera parte del trabajo, se trataron los fundamentos matemáticos de la regresión lineal y exponencial, extendiéndolos hacia modelos más sofisticados como las máquinas de soporte vectorial (*support vector regressors*), los *random forest* y distintas arquitecturas de redes neuronales —recurrentes, convolutivas y *transformers*. Cada modelo se ha estudiado en profundidad, tanto en su formulación matemática como en su procedimiento de entrenamiento, analizando también los métodos de optimización de hiperparámetros, las estrategias de validación durante el entrenamiento y las principales métricas utilizadas para evaluar su rendimiento. Todo ello se ha enfocado en su aplicación al contexto de las series temporales financieras.

La segunda parte del trabajo se ha centrado en transformar la teoría en práctica mediante la creación de un entorno de experimentación en Python. Dicho entorno permite definir, entrenar y evaluar modelos predictivos de forma sistemática, garantizando la coherencia de los resultados y la reproducibilidad de los experimentos. El sistema implementa una arquitectura modular que separa la gestión de datos, la configuración de hiperparámetros y el registro de métricas, lo que facilita la replicabilidad de los experimentos y la incorporación

Resumen

de nuevos modelos en el futuro. Además, todos los modelos se han entrenado diferenciando claramente los conjuntos de entrenamiento, validación y prueba, empleando técnicas de validación temporal como la *walk-forward validation*.

Junto a este entorno, se ha desarrollado una aplicación web que muestra de manera visual los resultados de los modelos y permite simular carteras personalizadas utilizando sus predicciones. Gracias a su diseño interactivo, el usuario puede seleccionar activos, modelos y períodos de tiempo para analizar el comportamiento de cada estrategia.

La metodología seguida en este trabajo se desarrolló en distintas fases. En primer lugar, se estudiaron los fundamentos de la estrategia basada en momentos, tomando como referencia la formulación propuesta por Andreas F. Clenow. A continuación, se revisaron los principales modelos de machine learning empleados en la literatura —como las regresiones lineales y exponenciales, las máquinas de soporte vectorial, los *random forest* y diversas arquitecturas de redes neuronales—, analizando sus fundamentos teóricos y matemáticos. Posteriormente, se utilizó el índice S&P 500 como entorno de referencia para realizar la optimización de hiperparámetros y la ingeniería de características, con el objetivo de obtener un modelo “general del mercado” que reflejara el comportamiento medio de los activos. Una vez identificado el modelo con mejor rendimiento, éste se aplicó al resto de universos de acciones con el fin de evaluar su capacidad de generalización y comparar los resultados con las carteras basadas en la estrategia *momentum investing*.

Los resultados obtenidos muestran que los modelos predictivos son capaces de superar, en la mayoría de los casos analizados, a la estrategia basada en momentos. En particular, las carteras construidas a partir de las predicciones de los modelos con arquitectura *transformer* alcanzaron una rentabilidad acumulada superior a la lograda por la estrategia de Clenow bajo el mismo universo de activos. De hecho, superaron en algunos casos al índice estadounidense S&P 500.

Desde el punto de vista personal y académico, el desarrollo de este proyecto ha permitido profundizar en los fundamentos teóricos del aprendizaje automático y en su aplicación práctica al ámbito financiero, un área de especial interés para el autor. A lo largo del trabajo, ha aprendido a implementar modelos complejos en Python, entrenarlos adecuadamente sobre series temporales y diseñar experimentos reproducibles siguiendo buenas prácticas de programación científica. Además, el proyecto ha supuesto el reto de integrarse en una base de código preexistente desarrollada por el tutor, lo que exigió comprender su arquitectura interna y ampliarla mediante nuevas clases y métodos coherentes con el sistema original.

En suma, este trabajo constituye una síntesis entre teoría, práctica e innovación, combinando los conocimientos adquiridos en Matemáticas e Ingeniería Informática con un enfoque aplicado al análisis de los mercados financieros. Los resultados obtenidos demuestran el potencial de los modelos predictivos para mejorar estrategias de inversión tradicionales, aunque también ponen de relieve los límites inherentes a la predicción de mercados complejos y no estacionarios. Este proyecto abre así la puerta a futuras líneas de investigación orientadas a ampliar el conjunto de variables macroeconómicas y fundamentales consideradas —incorporando indicadores que reflejen mejor los ciclos económicos y las condiciones del mercado—, así como a explorar métodos de inteligencia artificial más avanzados que acerquen la predicción financiera a una disciplina plenamente científica.

Summary

Financial markets are one of the key pillars of the modern economy. Every day, millions of investors buy and sell assets driven by their expectations, fears, and strategies, causing prices to fluctuate constantly. One of the most popular strategies of recent decades is the so-called "momentum investing".

The principle behind this strategy is simple: buying the assets that have performed best in the recent past and selling those with the worst performance. Although straightforward, momentum investing has shown empirically positive results, outperforming the market average in many periods. However, this strategy is not exempt from criticism. Some renowned investors, such as Benjamin Graham, already warned that «*the intelligent investor should never forecast the future solely by extrapolating the past*». Intuitively, it is reasonable to assume that in a financial environment characterized by high volatility and structural complexity, momentum investing should present certain weaknesses.

This Final Degree Project arises precisely from that reflection: is it possible to improve the traditional momentum-based investment strategy by incorporating machine learning techniques and advanced predictive models? To address this question, a project has been developed that combines mathematics, computer science, and finance, integrating both classical and modern *machine learning* models applied to historical market data. The objective has been to compare the performance of portfolios based on the strategy proposed by Andreas F. Clenow —described in his book *Stocks on the Move* and rooted in momentum investing— with those obtained through predictive models, analyzing their results in terms of profitability.

From a theoretical perspective, corresponding to the first part of the project, the mathematical foundations of linear and exponential regression were studied and extended towards more sophisticated models such as support vector regressors, random forests, and different neural network architectures —recurrent, convolutional, and *transformers*. Each model was examined in depth, both in its mathematical formulation and in its training process, also analyzing methods for hyperparameter optimization, validation strategies during training, and the main metrics used to evaluate performance. All these aspects were approached with a focus on their application to financial time series.

The second part of the project focused on transforming theory into practice through the creation of an experimentation framework in Python. This environment allows models to be defined, trained, and evaluated systematically, ensuring consistency of results and experiment reproducibility. The system implements a modular architecture that separates data management, hyperparameter configuration, and metric logging, thereby facilitating the replicability of experiments and the incorporation of new models in the future. In addition, all models were trained with a clear distinction between training, validation, and test sets, using temporal validation techniques such as *walk-forward validation*.

Alongside this environment, a web application was developed to visually present the

Summary

results of the models and to simulate customized portfolios using their predictions. Thanks to its interactive design, users can select assets, models, and time periods to analyze the behavior of each strategy.

The methodology followed in this project was structured in several phases. First, the foundations of the momentum-based investment strategy were studied, taking as reference the formulation proposed by Andreas F. Clenow. Next, the main *machine learning* models used in the literature —such as linear and exponential regressions, support vector machines, random forests, and various neural network architectures— were reviewed, analyzing their theoretical and mathematical principles. Subsequently, the S&P 500 index was used as a reference environment to perform hyperparameter optimization and feature engineering, with the aim of obtaining a “general market” model that would reflect the average behavior of assets. Once the best-performing model was identified, it was applied to other universes of stocks in order to assess its generalization capacity and compare the results with portfolios based on the *momentum investing* strategy.

The results show that predictive models are able to outperform, in most of the cases analyzed, the traditional momentum-based strategy. In particular, the portfolios built from the predictions of *transformer*-based models achieved higher cumulative returns than those obtained by Clenow’s strategy under the same universe of assets. In fact, in some cases, they even outperformed the S&P 500 index itself.

From both a personal and academic standpoint, the development of this project has allowed the author to deepen his understanding of the theoretical foundations of machine learning and its practical application to the financial domain —an area of particular interest to him. Throughout the project, he has learned to implement complex models in Python, to train them properly when dealing with time series, and to design reproducible experiments that follow good practices in scientific programming. The project has also presented the challenge of integrating into a pre-existing codebase developed by the project supervisor, which required understanding its internal architecture and extending it with new classes and methods consistent with the original system.

In conclusion, this work represents a synthesis of theory, practice, and innovation, combining the knowledge acquired in Mathematics and Computer Engineering with an applied approach to financial market analysis. The results demonstrate the potential of predictive models to enhance traditional investment strategies, while also highlighting the inherent limitations of forecasting complex and non-stationary markets. This project therefore opens the door to future research aimed at expanding the range of macroeconomic and fundamental variables considered —incorporating indicators that better capture business cycles and market conditions— as well as exploring more advanced artificial intelligence methods that bring financial prediction closer to a truly scientific discipline.

Índice general

| | |
|--|-----------|
| Agradecimientos | v |
| Resumen | VII |
| Summary | IX |
| Introducción | xv |
| | |
| I. Fundamentos teóricos | 1 |
| 1. Preliminares | 3 |
| 1.1. Fundamentos de probabilidad | 3 |
| 1.2. Muestreos y estadísticos | 6 |
| 1.3. Fundamentos de análisis | 7 |
| 1.3.1. Funciones continuas y lineales | 8 |
| 1.3.2. Convexidad | 9 |
| 1.3.3. Funciones diferenciables | 9 |
| 1.3.4. Caracterización de convexidad | 11 |
| | |
| 2. Modelos de clásicos de aprendizaje supervisado | 13 |
| 2.1. ¿Qué es el aprendizaje automático? | 13 |
| 2.2. Regresión exponencial | 13 |
| 2.2.1. Regresión lineal | 14 |
| 2.2.2. Regresión exponencial | 16 |
| 2.2.3. Observaciones y limitaciones | 17 |
| 2.3. Support Vector Regression | 18 |
| 2.3.1. Support Vector Machines | 18 |
| 2.3.2. Planitud y pérdida ϵ -insensible | 19 |
| 2.3.3. Formulación de SVR | 20 |
| 2.3.4. Extensión de SVR a problemas no lineales mediante el truco del kernel | 22 |
| 2.3.5. Otras funciones de coste | 24 |
| 2.3.6. Observaciones y limitaciones | 24 |
| 2.4. Random Forest | 25 |
| 2.4.1. Árboles de Regresión y Clasificación – CART | 25 |
| 2.4.2. Conceptos Fundamentales de random forest | 26 |
| 2.4.3. Margen y generalización | 28 |
| 2.4.4. Observaciones y limitaciones | 29 |

Índice general

| | |
|--|-----------|
| 3. Redes Neuronales | 31 |
| 3.1. ¿Qué es una red neuronal? | 31 |
| 3.1.1. Funciones de activación | 33 |
| 3.1.2. La red neuronal como aproximador universal | 36 |
| 3.2. Entrenamiento de redes neuronales | 41 |
| 3.2.1. Gradiente descendente | 41 |
| 3.2.2. Problemas frecuentes | 44 |
| 3.2.3. Regularización | 45 |
| 3.3. Redes recurrentes | 46 |
| 3.3.1. Estructura de una red neuonal recurrente simple | 47 |
| 3.3.2. Problemas de RNN | 48 |
| 3.3.3. Redes neuronales recurrentes con puertas | 49 |
| 3.4. Redes convolucionales | 52 |
| 3.4.1. Convolución | 53 |
| 3.4.2. Capa convolucional | 54 |
| 3.4.3. Capa de pooling | 55 |
| 3.5. Transformers | 56 |
| 3.5.1. Estructura encoder-decoder | 57 |
| 3.5.2. Codificación posicional | 58 |
| 3.5.3. Mecanismos de atención | 59 |
| 4. Métricas y metodología | 63 |
| 4.1. Métricas estadísticas | 63 |
| 4.2. Métricas bursátiles | 64 |
| 4.3. Validación de modelos | 66 |
| 4.4. Selección de hiperparámetros | 67 |
| 4.4.1. Optimización bayesiana | 68 |
| II. Desarrollo del proyecto | 71 |
| 5. Diseño e implementación del sistema | 73 |
| 5.1. Objetivos | 73 |
| 5.1.1. ¿Cómo se construye la cartera? | 74 |
| 5.1.2. Estrategia de Clenow | 75 |
| 5.2. Requisitos del sistema | 77 |
| 5.3. Diseño del sistema | 77 |
| 5.3.1. Modelos | 78 |
| 5.3.2. Experimentos | 79 |
| 5.3.3. Simulaciones | 80 |
| 5.3.4. Técnicas de optimización de hiperparámetros | 81 |
| 5.4. Despliegue del sistema | 81 |
| 5.5. Interfaz de usuario | 83 |
| 5.6. Manual de usuario | 84 |
| 6. Resultados experimentales | 91 |
| 6.1. Entrenamiento de modelos | 91 |
| 6.1.1. Modelos clásicos | 91 |

Índice general

| | | |
|---------------------|---|------------|
| 6.1.2. | Modelos simples basados en redes neuronales | 94 |
| 6.1.3. | Tuning de hiperparámetros | 98 |
| 6.1.4. | Ingeniería de características: ¿importa la situación macro? | 99 |
| 6.2. | Simulación de carteras | 102 |
| 6.2.1. | Clásicos contra Clenow | 103 |
| 6.2.2. | Redes neuronales contra Clenow | 105 |
| 7. | Conclusiones y líneas de trabajo futuras | 111 |
| A. | Mercados financieros | 113 |
| A.1. | ¿Qué es la bolsa? | 113 |
| A.1.1. | Acciones | 114 |
| A.1.2. | Índices, fondos y ETFs | 115 |
| B. | Resultados del entrenamiento de modelos | 117 |
| B.1. | Hiperparámetros | 117 |
| B.2. | Ingeniería de Características | 122 |
| Bibliografía | | 127 |

Introducción

Gracias al crecimiento, la productividad, la inventiva y la innovación de nuestras corporaciones, el capitalismo crea riqueza, un juego de suma positiva para sus propietarios. Invertir en acciones es una estrategia ganadora.

John C.Bogle

El pequeño libro para invertir con sentido común

Contexto y Antecedentes

En 2024, la media diaria de acciones compradas y vendidas sólo en el mercado estadounidense fue de 12 200 millones títulos [1]. Esta cifra da una idea de la magnitud y relevancia del mercado bursátil, donde cada jornada se producen millones de transacciones que reflejan las expectativas, miedos y estrategias de inversores de todo el mundo. Ante un entorno tan volátil, en el que se buscan las mayores rentabilidades posibles, la elección de una buena estrategia de inversión es crucial.

Si uno se deja llevar por el sentido común a la hora de invertir en los mercados bursátiles (o por el Sistema 1 como diría Daniel Kahneman¹) lo más probable es que invirtiera en aquellos activos que mejores rentabilidades han conseguido en el pasado reciente. Esta simple metodología es la base de una famosa estrategia de inversión, llamada "inversión basada en momentos" (o momentum, si uno quiere ser más técnico).

La idea, sin embargo, está lejos de ser meramente intuitiva. De hecho, ya en los años sesenta [3] se mostró que seleccionar acciones según su fuerza relativa —es decir, aquellas que habían tenido un mejor comportamiento en el pasado reciente— generaba rendimientos superiores al mercado. Décadas más tarde, Jegadeesh y Titman [4] demostraron que comprar acciones ganadoras y vender perdedoras en horizontes de 3 a 12 meses producía retornos anómalos, contradiciendo la teoría del *random walk*². Desde entonces, la inversión basada en momentum se ha consolidado como una de las anomalías más robustas y estudiadas en las finanzas modernas. Conocidos divulgadores y economistas españoles, como es el caso de Hugo Ferrer, han escrito libros en los que se trata el *momentum investing* [5].

Esta simple, pero eficaz, técnica rompe con lo que dijo uno de los mejores inversores del siglo pasado y fundador de una de las mayores gestoras del mundo³: «*Las rentabilidades*

¹El Sistema 1 y Sistema 2 son dos alegorías creadas por el psicólogo ganador del premio Nobel de economía en su libro "Pensar rápido, pensar despacio" [2]. El 1 se refiere a los procesos mentales rápidos, automáticos e intuitivos, mientras que el 2 está asociado con el pensamiento más lento, deliberativo y racional.

²La teoría del paseo aleatorio plantea que el valor de las acciones en el tiempo es aleatorio e impredecible.

³John C. Bogle fundó en 1975 la gestora Vanguard, ofreciendo por primera vez a sus clientes la posibilidad de invertir en fondos indexados como el S&P500 o el Nasdaq.

Introducción

pasadas no garantizan las rentabilidades futuras» [6]. Siguiendo esta importante cita, en cuanto uno empieza a razonar acerca de este método (es decir, utiliza su Sistema 2) se da cuenta de que esta técnica presenta debilidades claras al basarse únicamente en cuánto haya subido o bajado el precio de la acción en las últimas semanas. Lo que uno probablemente piense tras este razonamiento es que sería mucho mejor usar algún tipo de información extra o métodos alternativos que impulsen la, aparentemente trivial, inversión basada en momentos.

Es en este punto donde entran en juego los algoritmos y técnicas de inteligencia artificial, capaces de procesar e incorporar grandes volúmenes de información adicional y aplicarlos en la toma de decisiones de inversión. Su uso, sin embargo, no es algo nuevo: se dice que la empresa pionera fue el fondo de inversión Bridgewater Associates, fundado por Ray Dalio en 1975, la cual cuenta con más de 154 mil millones de dólares en activos a día de hoy. Otro pionero fue Jim Simons, con su fondo Renaissance Technologies, el cual maneja también una cuantiosa suma de dinero. Actualmente, muchos fondos de inversión utilizan el poder de la computación para obtener mayores rentabilidades.

En este trabajo se compararán precisamente las rentabilidades obtenidas por las carteras respaldadas por diferentes modelos predictivos y las basadas en momentos. Se revisarán las bases matemáticas de todos los modelos utilizados y de las métricas que se usen para la comparación de resultados.

Motivación

La estrategia de inversión basada en momentos ha demostrado históricamente una rentabilidad mayor que la media del mercado. En ocasiones, una demostración empírica es más que suficiente para probar que una estrategia es buena – siguiendo la filosofía de Nassim Taleb. Sin embargo, la dependencia únicamente de la evolución pasada de los precios en un marco temporal limitado la hace vulnerable a posibles cambios abruptos de mercado, conocidos en el mundo financiero como "cisnes negros" [7].

Por otro lado, los avances agigantados en los campos del aprendizaje automático y la inteligencia artificial permiten desarrollar herramientas que mejoran la toma de decisiones en entornos de alta incertidumbre, como es el caso de los mercados financieros. El uso de modelos predictivos es un gran complemento que permite detectar patrones complejos, no lineales y dinámicos.

La motivación principal de este trabajo consiste en comprobar si el uso de modelos predictivos como las máquinas de soporte vectorial (SVR), random forests o distintas arquitecturas de redes neuronales son capaces de mejorar las rentabilidades de la inversión basada en momentos. Estos modelos permiten la incorporación de más información a la hora de hacer las predicciones, aumentando así las capacidades de adaptación del mercado.

Este proyecto permite combinar los conocimientos adquiridos en ambas titulaciones – Matemáticas e Informática – con uno de mis grandes intereses personales: la economía y los mercados financieros.

Objetivos

Partiendo de la motivación, se establecen los siguientes objetivos:

- Estudiar y describir los fundamentos teóricos de los distintos modelos de predicción utilizados: regresión exponencial, máquinas de soporte vectorial, Random Forest y redes neuronales.
- Analizar el soporte matemático de cada modelo, incluyendo su formulación, métodos de optimización y fundamentos estadísticos.
- Implementar estos modelos en Python y aplicarlos a datos reales de mercado desde 1990, empleando técnicas de preprocesamiento y entrenamiento.
- Comparar el rendimiento de los modelos utilizando métricas de evaluación como MSE, MAPE o R^2 y analizar los rendimientos de las carteras usando la inversión basada en momentos frente a las que usan estos modelos.
- Desarrollar una aplicación web que permita al usuario observar las predicciones de distintos modelos para diferentes activos financieros.

Estructura de la memoria

Se pasa a describir la estructura del presente Trabajo Final de Grado:

- **Parte 1.** En esta parte se explican los conceptos técnicos y teóricos detrás de los modelos matemáticos necesarios, así como de las métricas utilizadas para la comparación de resultados.
 1. Preliminares: Conceptos matemáticos necesarios para las explicaciones en profundidad de los distintos tipos de modelos predictivos.
 2. Modelos clásicos de aprendizaje supervisado: Explicación de los modelos englobados en el conocido marco *Machine Learning* que no pertenecen a las redes neuronales. Estos modelos son: regresiones lineales y exponenciales, *Support Vector Regressors* (SVR) y *Random Forest*.
 3. Redes Neuronales: Explicación de las redes neuronales, además de las variantes de las mismas utilizadas en el proyecto: redes recurrentes, convolutivas o los famosos *transformers*.
 4. Métricas y metodología: Aclaración de las métricas utilizadas para la evaluación de los modelos, tanto estadísticas como financieras, y la metodología utilizada para su creación.
- **Parte 2.** Esta segunda parte se centra en el desarrollo, desde el punto de vista informático, del proyecto.
 1. Desarrollo de los modelos predictivos: Explicación y exposición del procedimiento para el entrenamiento de los modelos y los resultados obtenidos.

2. Aplicación web: Presentación de la aplicación web creada para mostrar los distintos modelos y resultados al usuario.
3. Conclusiones y líneas de trabajo futuro: Conclusiones del proyecto y posibles trabajos futuros a partir de él.

Planificación

La planificación del presente Trabajo Fin de Grado ha sido la siguiente:

1. **Documentación y análisis del problema:** Comprensión del problema planteado y lectura del libro sobre el cual se fundamenta este trabajo: el libro de Andreas Clenow [8].
2. **Familiarización con el proyecto previamente creado y el nuevo lenguaje:** El código de este TFG se ha desarrollado sobre un proyecto creado por el tutor del presente TFG: Fernando Berzal. La familiarización con las clases y estructura de este proyecto es necesaria, así como el aprendizaje de Python.
3. **Investigación sobre modelos de Machine Learning y Deep Learning:** Lectura de artículos y libros con el objetivo de comprender adecuadamente el funcionamiento y las bases de los modelos que se quieren utilizar en la práctica.
4. **Creación de modelos:** Diseño, desarrollo e implementación de diversos tipos de modelos y arquitecturas de redes neuronales.
5. **Simulaciones:** Creación del entorno de simulación para comparar los resultados de las carteras guiadas por modelos frente a las guiadas por momentum o Clenow.
6. **Mejora de modelos:** Mejora de los modelos creados para obtener mejores resultados frente a la estrategia de inversión basada en momentos.
7. **Creación aplicación web:** Desarrollo y despliegue de una aplicación web.
8. **Redacción de la memoria:** Redacción de la memoria conforme se vaya avanzando en el desarrollo de la parte software y se haya leído acerca de los fundamentos matemáticos y técnicos de los modelos escogidos.

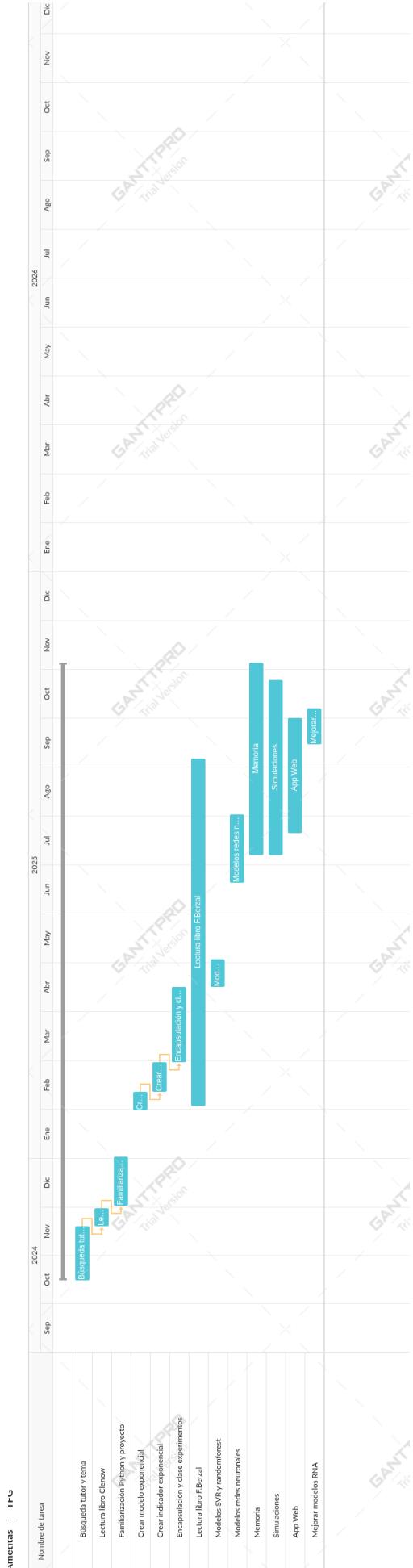


Figura 1.: Diagrama de Gantt

Parte I.

Fundamentos teóricos

1. Preliminares

En esta primera parte de la memoria se presentan todos los fundamentos matemáticos detrás de los modelos y herramientas usadas, desde los modelos hasta métricas usadas para la comparación de las predicciones. Para la parte de probabilidad e inferencia se, ha seguido a [9] y [10]. Para la parte de análisis matemático, se ha seguido a [11] y [12].

Para lo que sigue n es un número natural (así como k , y m), es decir $n \in \mathbb{N}$, y se notarán los vectores de \mathbb{R}^n en negrita y los valores escalares (los pertenecientes a \mathbb{R}) sin la negrita.

1.1. Fundamentos de probabilidad

Definición 1.1 (Espacio de probabilidad). Un espacio de probabilidad es una terna (Ω, \mathcal{A}, P) donde:

1. Ω es un conjunto arbitrario.
2. $\mathcal{A} \subset \mathcal{P}(\Omega)$ tiene estructura de σ -álgebra, verificando:
 - $\mathcal{A} \neq \emptyset$.
 - Cerrada para las uniones numerables.
 - Cerrada para la formación del complementario.
3. $P : \mathcal{A} \rightarrow [0, 1]$ es una función de probabilidad que satisface los tres axiomas siguientes:
 - $P(A) \geq 0 \quad \forall A \in \mathcal{A}$.
 - $P(\Omega) = 1$.
 - Para cualquier sucesión $\{A_n\}_{n \in \mathbb{N}} \subseteq \mathcal{A}$ de sucesos disjuntos, se verifica

$$P\left(\bigcup_{n \in \mathbb{N}} A_n\right) = \sum_{n \in \mathbb{N}} P(A_n).$$

Definición 1.2 (Variable aleatoria). Una variable aleatoria sobre un espacio de probabilidad (Ω, \mathcal{A}, P) es una función medible $X : \Omega \rightarrow \mathbb{R}$, i.e.

$$X^{-1}(B) \in \mathcal{A}, \quad B \in \mathcal{B}$$

O equivalentemente,

$$X^{-1}((-\infty, x]) \in \mathcal{A} \quad \forall x \in \mathbb{R}$$

1. Preliminares

Observación 1.3. ■ \mathcal{B} es la σ -álgebra de Borel, es decir, la mínima σ -álgebra sobre \mathbb{R} que contiene a todos los intervalos.

- $X^{-1}(B) = \{\omega \in \Omega : X(\omega) \in B\}$. En este caso se denota $\{X \in B\}$.
- $X^{-1}((-\infty, x]) = \{\omega \in \Omega : X(\omega) \leq x\}$.

Definición 1.4 (Función de distribución). Una función de distribución de X , $F_X : \mathbb{R} \rightarrow [0, 1]$ dada por

$$F_X(x) = P_X[(-\infty, x]] = P[X^{-1}((-\infty, x])] = P[X \leq x]$$

Cumple las siguientes propiedades:

- Es no decreciente.
- Es continua a la derecha.
- $F_X(-\infty) = \lim_{x \rightarrow -\infty} F_X(x) = 0$ y $F_X(+\infty) = \lim_{x \rightarrow +\infty} F_X(x) = 1$.

Teorema 1.5 (Teorema de correspondencia). *Toda función $F : \mathbb{R} \rightarrow [0, 1]$ que cumpla las tres propiedades anteriores es la función de distribución de alguna variable aleatoria.*

Definición 1.6 (Variable aleatoria discreta). Una variable aleatoria (v.a.) $X : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}, \mathcal{B}, P_X)$ se dice discreta si existe un conjunto numerable $E_X \subset \mathbb{R}$, tal que $P_X(X \in E_X) = 1$.

La función masa de probabilidad de una v.a. discreta se define como:

$$p_X : E_X \rightarrow [0, 1], \quad p_X(x) = P(X = x), \quad \forall x \in E_X.$$

Se tiene entonces $\sum_{x \in E_X} p_X(x) = 1$.

Definición 1.7 (Variable aleatoria continua). Una variable aleatoria (v.a.) $X : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}, \mathcal{B}, P_X)$, se dice continua si existe una función $f_X : \mathbb{R} \rightarrow \mathbb{R}$ tal que

$$F_X(x) = \int_{-\infty}^x f_X(y) dy, \quad \forall x \in \mathbb{R}.$$

- (i) Es no negativa.
- (ii) Es integrable con $\int_{\mathbb{R}} f_X(x) dx = 1$.

Definición 1.8. Sea X una variable aleatoria discreta. Si $\exists \sum_{x \in E_X} |x| p_X(x) < \infty$, entonces la esperanza matemática de X se define como:

$$\mathbb{E}[X] = \sum_{x \in E_X} x p_X(x) = \sum_{x \in E_X} x P(X = x)$$

Definición 1.9 (Esperanza matemática). Sea X una variable aleatoria continua. Si $\exists \int_{\mathbb{R}} |x| f_X(x) dx <$

∞ , la esperanza matemática de X se define como:

$$\mathbb{E}[X] = \int_{\mathbb{R}} xf_X(x) dx$$

Definición 1.10. [Varianza] Sea X una variable aleatoria tal que $\exists \mathbb{E}[X] < \infty$. Entonces se define la varianza de X , denotada por $\text{Var}(X)$ o σ_X^2 , como:

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

O, equivalentemente

$$\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

Definición 1.11 (Vector aleatorio). Un vector aleatorio $X = (X_1, \dots, X_n)$ sobre un espacio probabilístico base (Ω, \mathcal{A}, P) se define como una función $X : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}^n, \mathcal{B}^n)$ de modo que:

$$X^{-1}(B) = \{\omega \in \Omega ; X(\omega) \in B\} \in \mathcal{A}, \quad \forall B \in \mathcal{B}^n,$$

donde $X_i : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}, \mathcal{B})$, $\forall i = 1, \dots, n$.

Teorema 1.12 (Teorema de medibilidad). $X = (X_1, \dots, X_n)$ es un vector aleatorio sobre (Ω, \mathcal{A}, P) si y sólo si X_i es una variable aleatoria sobre (Ω, \mathcal{A}, P) , para $i = 1, \dots, n$.

Definición 1.13 (Esperanza matemática para vectores aleatorios). Sea $X = (X_1, \dots, X_n) : (\Omega, \mathcal{A}, P) \rightarrow (\mathbb{R}^n, \mathcal{B}^n, P_X)$ un vector aleatorio n -dimensional. La esperanza matemática, $\mathbb{E}[X]$, de X , si existe, se define como un vector determinístico cuyas componentes son las medias o esperanzas matemáticas de sus componentes aleatorias, es decir:

$$\mathbb{E}[X] = (\mathbb{E}[X_1], \dots, \mathbb{E}[X_n]) = (\mu_1, \dots, \mu_n) \in \mathbb{R}^n.$$

Algunas propiedades importantes de la esperanza matemática son:

- **Linealidad.** Para cualesquiera $(a_1, \dots, a_n), (b_1, \dots, b_n) \in \mathbb{R}^n$,

$$\exists \mathbb{E}[X_i] \Rightarrow \exists \mathbb{E}[a_i X_i + b_i], \quad i = 1, \dots, n.$$

$$\exists \mathbb{E} \left[\sum_{i=1}^n a_i X_i + b_i \right] \Rightarrow \sum_{i=1}^n a_i \mathbb{E}[X_i] + b_i.$$

- **Monotonía.** Sean X_1 y X_2 variables aleatorias unidimensionales, tales que $\exists \mathbb{E}[X_1], \exists \mathbb{E}[X_2]$, entonces

$$X_1 \leq X_2 \Rightarrow \mathbb{E}[X_1] \leq \mathbb{E}[X_2]$$

Definición 1.14 (Independencia). Sean X_1, \dots, X_n v.a. definidas sobre (Ω, \mathcal{A}, P) . Son independientes si y sólo si

$$F_{(X_1, \dots, X_n)}(x_1, \dots, x_n) = F_{X_1}(x_1)F_{X_2}(x_2) \dots F_{X_n}(x_n)$$

1. Preliminares

Definición 1.15 (Distribución normal). Se dice que una variable aleatoria continua X se distribuye según una normal de parámetros $\mu \in \mathbb{R}$ (media) y $\sigma^2 > 0$ (varianza) si su función de densidad f_X viene dada por la siguiente expresión:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}.$$

Mediante el procedimiento de tipificación se tiene que,

$$X \sim \mathcal{N}(\mu, \sigma^2) \iff \frac{X - \mu}{\sigma} \sim \mathcal{N}(0, 1).$$

1.2. Muestreos y estadísticos

Una vez establecidas las definiciones básicas de la teoría de la probabilidad, se introducen los conceptos elementales de la inferencia estadística necesarios para la comprensión de la base matemática del proyecto.

Dada la naturaleza estocástica de los precios de los activos financieros – el lector probablemente haya escuchado a gente decir que invertir es como apostar¹–, estos se consideran variables aleatorias. En este contexto, el problema de inferencia estadística presente es uno de tipo no paramétrico, ya que la función de distribución de estas variables es desconocida.

Las definiciones que se presentan a continuación son básicas para la comprensión de los modelos utilizados como primera aproximación, la regresión exponencial y la lineal, así como los modelos más complejos presentados posteriormente.

Definición 1.16 (Muestra aleatoria simple). Una Muestra Aleatoria Simple (MAS) de una población con distribución F_θ es un conjunto de n variables aleatorias X_1, X_2, \dots, X_n que son independientes e idénticamente distribuidas (i.i.d.) según F_θ .

Observación 1.17. Nótese que, en el ámbito de los mercados financieros, no se cumple que las variables (las series temporales de los precios de los activos) sean independientes e idénticamente distribuidas. Esto es debido fundamentalmente a:

- La autocorrelación inherente que tienen las series temporales.
- La heterocedasticidad (i.e varianza no es constante a lo largo del tiempo) de los mercados financieros, donde hay ciclos y numerosos valores extremos causados por la irracionalidad de Mr. Market².

Se comprobará posteriormente que la suposición de lo contrario para el uso de modelos basados en regresiones lineales y exponenciales resulta en unas predicciones completamente erróneas. Sin embargo, estos modelos serán un buen punto de partida.

¹Peter Lynch, uno de los mejores inversores del siglo pasado, dijo en su conocido libro "Un paso por delante de Wall Street" [13]: « Desde mi punto de vista, una inversión no es más que una apuesta en la que has logrado inclinar las probabilidades a tu favor. »

²En el famoso libro "El inversor inteligente" [14], Benjamin Graham utiliza la personificación de "Mr. Market" para describir la naturaleza a menudo irracional del mercado de valores y los riesgos de seguir ciegamente la opinión de la multitud.

Definición 1.18 (Estadístico muestral). Un estadístico muestral (o simplemente estadístico) es cualquier función real y medible de las observaciones de una muestra aleatoria $T(X_1, X_2, \dots, X_n)$ que no depende de ningún parámetro desconocido. Un estadístico utilizado para estimar un parámetro desconocido de la población se denomina estimador.

Algunos ejemplos importantes de estadísticos muestrales son:

1. Media muestral: $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$
2. Varianza muestral: $\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2$
3. Cuasivarianza muestral: $S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$

Notar que, en la práctica, la varianza muestral no se utiliza. En su lugar, se hace uso de la cuasivarianza muestral S^2 . De hecho, algunos entienden por varianza muestral a lo que se ha definido por cuasivarianza muestral. Esto es debido a que la esperanza de la varianza muestral no coincide con la varianza (véase definición (1.10)) (o varianza poblacional) y la esperanza de la cuasivarianza muestral sí que coincide. Para comprobarlo, denótese por B a la varianza muestral: $\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2$. Entonces:

$$\begin{aligned} \sum_{i=1}^n (X_i - \bar{X})^2 &= \sum_{i=1}^n (X_i^2 - 2X_i\bar{X} + \bar{X}^2) = \sum_{i=1}^n X_i^2 - 2n\bar{X}^2 + n\bar{X}^2 = \sum_{i=1}^n X_i^2 - n\bar{X}^2. \\ B &= \frac{1}{n} \sum_{i=1}^n X_i^2 - \bar{X}^2. \end{aligned}$$

Dado que X_i son i.i.d., siguiendo la definición (1.10) y que $\mu = \mathbb{E}[X]$ y $\sigma^2 = \text{Var}[X]$:

$$\mathbb{E}[X_i^2] = \sigma^2 + \mu^2, \quad \mathbb{E}[\bar{X}^2] = \frac{\sigma^2}{n} + \mu^2,$$

y por tanto

$$\mathbb{E}[B] = (\sigma^2 + \mu^2) - \left(\frac{\sigma^2}{n} + \mu^2 \right) = \frac{n-1}{n} \sigma^2.$$

Ahora, sustituyendo:

$$\mathbb{E}[S^2] = \mathbb{E}\left[\frac{n}{n-1} B\right] = \frac{n}{n-1} \frac{(n-1)\sigma^2}{n} = \sigma^2$$

1.3. Fundamentos de análisis

Tras la introducción de conceptos ligados a la rama de la estadística y la probabilidad, se introducen conceptos fundamentales del análisis matemático.

1.3.1. Funciones continuas y lineales

Definición 1.19 (Continuidad en un punto). Una función $f : X \rightarrow \mathbb{R}$ se dice que es continua en un punto $a \in X$ si, para cada número $\varepsilon > 0$, se puede encontrar un número $\delta > 0$ (que, en general, dependerá de ε y de a) tal que para todo $x \in X$ con $|x - a| < \delta$ se verifica que $|f(x) - f(a)| < \varepsilon$.

La definición anterior suele escribirse, de la siguiente forma:

$$\forall \varepsilon \in \mathbb{R}^+ \exists \delta \in \mathbb{R}^+ : \left\{ \begin{array}{l} |x - a| < \delta \\ x \in X \end{array} \right. \implies |f(x) - f(a)| < \varepsilon$$

Definición 1.20 (Continuidad en un conjunto). Se dice que $f : X \rightarrow \mathbb{R}$ es continua en un conjunto $C \subset X$, si f es continua en todo punto de C .

Definición 1.21 (Función lineal). Sean X e Y dos subespacios vectoriales de \mathbb{R}^n . Una aplicación $f : X \rightarrow Y$ es lineal si $\forall x_1, x_2$ se tiene:

1. $f(x_1 + x_2) = f(x_1) + f(x_2)$
2. $f(\lambda x_1) = \lambda f(x_1)$

Espacios normados y duales

Definición 1.22. Un espacio normado es un par $(X, \|\cdot\|)$ donde X es un espacio vectorial y $\|\cdot\| : X \rightarrow \mathbb{R}$ verifica:

1. $\|x\| > 0 \quad \forall x \in X$
2. $\|\lambda x\| = |\lambda| \|x\| \quad \forall x \in X, \forall \lambda \in \mathbb{R}$
3. $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in X$
4. $\|x\| = 0 \iff x = 0$

Ejemplos comunes de espacios normados son:

- \mathbb{R} con el valor absoluto, $(\mathbb{R}, |\cdot|)$.
- \mathbb{R}^n con la norma euclídea, $(\mathbb{R}^n, \|\cdot\|_2)$.
- \mathbb{R}^n con la norma del máximo, $(\mathbb{R}^n, \|\cdot\|_\infty)$.

Una vez visto el concepto de espacio normado, pasa a definirse el espacio dual:

Definición 1.23. Sea X un espacio normado, se define su espacio dual X^* como:

$$X^* = \{\varphi : X \rightarrow \mathbb{R} \text{ tal que } \varphi \text{ es lineal y continua}\}$$

1.3.2. Convexidad

Definición 1.24 (Conjunto convexo). Un subconjunto X de \mathbb{R}^n es convexo si, para cada par de puntos $\mathbf{x}, \mathbf{y} \in X$, el intervalo $[\mathbf{x}, \mathbf{y}]$ está contenido en X . Simbólicamente,

$$X \text{ convexo} \iff \{\forall \mathbf{x}, \mathbf{y} \in X \implies \lambda \mathbf{x} + (1 - \lambda) \mathbf{y} \in X \quad \forall \lambda \in [0, 1]\}$$



Figura 1.1.: Conjunto de \mathbb{R}^2 convexo (izquierda) y no convexo (derecha). [Wikipedia](#)

Definición 1.25 (Funciones convexas y cóncavas). Sea $N X \subseteq \mathbb{R}^n$ un subconjunto convexo de \mathbb{R}^n y $f : X \rightarrow \mathbb{R}$ una función real definida en X , entonces

- f es convexa si $f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y})$ para todo $\mathbf{x}, \mathbf{y} \in X$ y $\lambda \in [0, 1]$.
- f es cóncava si $f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \geq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y})$ para todo $\mathbf{x}, \mathbf{y} \in X$ y $\lambda \in [0, 1]$.
- f es estrictamente convexa si $f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) < \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y})$ para todo par de puntos $\mathbf{x}, \mathbf{y} \in X$ distintos y $\lambda \in (0, 1)$.
- f es estrictamente cóncava si $f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) > \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y})$ para todo par de puntos $\mathbf{x}, \mathbf{y} \in X$ distintos y $\lambda \in (0, 1)$.

1.3.3. Funciones diferenciables

Definición 1.26 (Diferenciabilidad). Sea $f : X \rightarrow \mathbb{R}^m$ donde $X \subseteq \mathbb{R}^n$ es un abierto. La función f se dice diferenciable en $\mathbf{x} \in X$ si existen una matriz A (de tamaño $m \times n$) y una función de error $r : \mathbb{R}^n \rightarrow \mathbb{R}^m$ definida alrededor de $0 \in \mathbb{R}^n$ tal que para todo \mathbf{y} en un entorno de \mathbf{x} se tiene:

$$f(\mathbf{y}) = f(\mathbf{x}) + A(\mathbf{y} - \mathbf{x}) + r(\mathbf{y} - \mathbf{x}),$$

donde

$$\lim_{\mathbf{v} \rightarrow 0} \frac{\|r(\mathbf{v})\|}{\|\mathbf{v}\|} = 0.$$

La matriz A es única, siempre y cuando esto se cumpla. Se llamará matriz Jacobiana de f en \mathbf{x} y se denotará por $Df(\mathbf{x})$. La función f se dirá *diferenciable* si es diferenciable en todo punto $\mathbf{x} \in X$.

Para el caso particular $m = 1$, es decir, cuando la función $f : X \rightarrow \mathbb{R}$ tiene valores escalares, la matriz Jacobiana $Df(\mathbf{x})$ se reduce a un vector fila denotado típicamente como $\nabla f(\mathbf{x})^T$. $\nabla f(\mathbf{x})$ es el vector columna llamado *gradiente* de f en \mathbf{x} .

1. Preliminares

Lema 1.27 (Regla de la cadena). *Sea $f : X \rightarrow \mathbb{R}^m$ donde $X \subseteq \mathbb{R}^n$ y $g : \text{dom}(g) \rightarrow \mathbb{R}^n$. Se supone que g es diferenciable en $\mathbf{x} \in \text{dom}(g)$ y que f es diferenciable en $g(\mathbf{x}) \in X$. Entonces la composición de f y g , $f \circ g$, es diferenciable en \mathbf{x} , y su matriz Jacobiana es la dada por:*

$$D(f \circ g)(\mathbf{x}) = Df(g(\mathbf{x}))Dg(\mathbf{x}).$$

Donde $\text{dom}(g)$ denota el dominio de la función g . Dada la definición de diferenciabilidad y función diferenciable.

Definición 1.28 (Función de clase C^1). *Sea $f : X \rightarrow \mathbb{R}^m$, donde $X \subseteq \mathbb{R}^n$ es un abierto. Se dice que f es de clase C^1 si es diferenciable en todo punto de X y todas sus derivadas parciales son funciones continuas en X . Es decir,*

$$f \in C^1(X) \iff f \text{ es diferenciable en } X \text{ y } Df \text{ es continua en } X.$$

Definición 1.29 (Mínimo global). *Sea $X \subseteq \mathbb{R}^n$ un subconjunto no vacío de \mathbb{R}^n . Sea $f : X \rightarrow \mathbb{R}$ una función real definida en X , entonces*

- f tiene un mínimo global en X si $f(\mathbf{x}) \geq f(\mathbf{a})$ para todo $\mathbf{x} \in X$.
- f tiene un mínimo global estricto en X si $f(\mathbf{x}) > f(\mathbf{a})$ para todo $\mathbf{x} \in X \setminus \{\mathbf{a}\}$, es decir, que \mathbf{a} es el único punto de X donde f tiene un mínimo global.

Definición 1.30 (Mínimo local). *Sea $X \subseteq \mathbb{R}^n$ un subconjunto no vacío de \mathbb{R}^n y $\mathbf{a} \in X$ un punto de X . Sea $f : X \rightarrow \mathbb{R}$ una función real definida en X , entonces*

- f tiene un mínimo local en \mathbf{a} si existe un entorno abierto $N(\mathbf{a}) \subseteq X$ de \mathbf{a} en X tal que $f(\mathbf{x}) \geq f(\mathbf{a})$ para todo $\mathbf{x} \in N(\mathbf{a})$.
- f tiene un mínimo local estricto en \mathbf{a} si existe un entorno abierto $N(\mathbf{a}) \subseteq X$ de \mathbf{a} en X tal que $f(\mathbf{x}) > f(\mathbf{a})$ para todo $\mathbf{x} \in N(\mathbf{a}) \setminus \{\mathbf{a}\}$, es decir, que \mathbf{a} es el único punto de $N(\mathbf{a})$ donde f tiene un mínimo local.

Definición 1.31 (Punto crítico). *Sea $X \subseteq \mathbb{R}^n$ un subconjunto no vacío de \mathbb{R}^n y $\mathbf{a} \in X$ un punto de X . Sea $f : X \rightarrow \mathbb{R}$ una función real definida en X , entonces un punto crítico de f es un punto $\mathbf{a} \in X$ donde f es diferenciable con $\nabla f(\mathbf{a}) = 0$.*

Definición 1.32 (Punto de silla). *Sea $n \in \mathbb{N}$ un número natural, $X \subseteq \mathbb{R}^n$ un subconjunto no vacío de \mathbb{R}^n y $f : X \rightarrow \mathbb{R}$ una función real definida en X , entonces un punto de silla de f es un punto crítico de f donde f no tiene ningún valor extremo (mínimo o máximo).*

Lema 1.33. *Sea $\mathbf{a} \in X$ un mínimo local de una función convexa $f : X \rightarrow \mathbb{R}$. Entonces \mathbf{a} es un mínimo global.*

Lema 1.34. *Supongamos que $f : X \rightarrow \mathbb{R}$ es una función convexa y diferenciable, con $X \subseteq \mathbb{R}^d$ abierto. Sea $\mathbf{x} \in X$, si $\nabla f(\mathbf{x}) = 0$, entonces \mathbf{x} es un mínimo global.*

Además, el recíproco también es cierto.

Lema 1.35. Supongamos que $f : X \rightarrow \mathbb{R}$ es una función convexa y diferenciable, con $X \subseteq \mathbb{R}^d$ abierto. Sea $\mathbf{x} \in X$, si \mathbf{x} es un mínimo global, entonces $\nabla f(\mathbf{x}) = 0$.

Nótese que una función convexa tenga un mínimo global no implica la unicidad del mínimo (basta tomar una función constante). Esto únicamente ocurre cuando la función es estrictamente convexa.

Lema 1.36. Sean $X \subseteq \mathbb{R}^n$ un subconjunto no vacío de \mathbb{R}^n y $f : X \rightarrow \mathbb{R}$. Entonces f tendrá como mucho un mínimo global (si existe el mínimo, será único).

Teorema 1.37 (Teorema de Weierstrass). Sea $f : X \rightarrow \mathbb{R}$ una función convexa. Supongamos que existe un conjunto no vacío y acotado $A \subseteq \mathbb{R}^n$ tal que f está definida sobre A y es continua. Entonces, f alcanza un mínimo global en A ; es decir, existe $\mathbf{a} \in A$ tal que

$$f(\mathbf{a}) \leq f(\mathbf{x}), \quad \text{para todo } \mathbf{x} \in A.$$

1.3.4. Caracterización de convexidad

Dada la definición de diferenciabilidad y función diferenciable, la convexidad de una función $f : X \rightarrow \mathbb{R}$ puede ser caracterizada por una inecuación en la que interviene el gradiente.

Lema 1.38 (Caracterización de convexidad de primer orden). Suponiendo que X es abierto y que f es diferenciable; en particular, el gradiente existe en todo punto $\mathbf{x} \in X$ y viene dado por

$$\nabla f(\mathbf{x}) := \left(\frac{df}{dx_1}(\mathbf{x}), \dots, \frac{df}{dx_d}(\mathbf{x}) \right)^T$$

Entonces f es convexa si y solo si X es convexo y

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}), \quad \forall \mathbf{x}, \mathbf{y} \in X.$$

Si la función es dos veces diferenciable, podremos caracterizar su convexidad mediante su matriz Hessiana:

$$\nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{d^2 f}{dx_1 dx_1}(\mathbf{x}) & \frac{d^2 f}{dx_1 dx_2}(\mathbf{x}) & \cdots & \frac{d^2 f}{dx_1 dx_d}(\mathbf{x}) \\ \frac{d^2 f}{dx_2 dx_1}(\mathbf{x}) & \frac{d^2 f}{dx_2 dx_2}(\mathbf{x}) & \cdots & \frac{d^2 f}{dx_2 dx_d}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{d^2 f}{dx_d dx_1}(\mathbf{x}) & \frac{d^2 f}{dx_d dx_2}(\mathbf{x}) & \cdots & \frac{d^2 f}{dx_d dx_d}(\mathbf{x}) \end{pmatrix}$$

Lema 1.39 (Caracterización de convexidad de segundo orden). Sea una función $f : X \rightarrow \mathbb{R}$, con X abierto y tal que f es dos veces diferenciable; en particular, la matriz Hessiana existe en todo punto $\mathbf{x} \in X$ y es simétrica. Entonces f es convexa si y solo si X es convexo y para todo $\mathbf{x} \in X$ es semidefinida positiva.

1. Preliminares

Se recuerda que matriz simétrica M , de tamaño $(n \times n)$, es semidefinida positiva si $\forall \mathbf{x} \in \mathbb{R}^n$ se tiene $\mathbf{x}^T M \mathbf{x} \geq 0$. De hecho, si la matriz es semidefinida positiva, entonces todos sus valores propios son positivos. Siguiendo esto y vista la matriz Hessiana, conviene diferenciar los tipos de puntos críticos en función de la naturaleza de la misma:

- Si todos los valores propios de $\nabla^2 f(\mathbf{x})$ son positivos, el punto \mathbf{x} es un mínimo local.
- Si todos los valores propios de $\nabla^2 f(\mathbf{x})$ son negativos, el punto \mathbf{x} es un máximo local.
- Si $\nabla^2 f(\mathbf{x})$ presenta valores propios de distinto signo, el punto \mathbf{x} es un punto de silla.

En consecuencia, los puntos de silla se distinguen de los óptimos locales por el carácter indefinido de la matriz Hessiana, que implica que la función presenta curvatura positiva en algunas direcciones y negativa en otras.

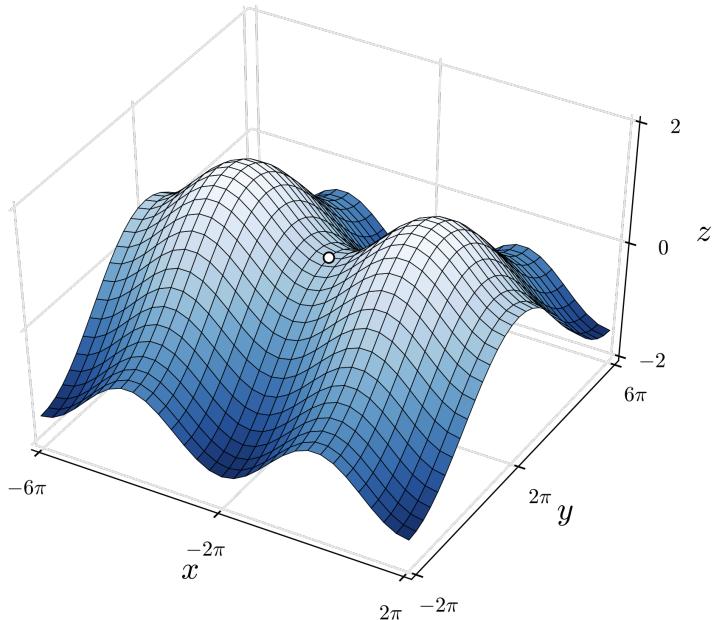


Figura 1.2.: Punto de silla. [Wikipedia](#)

Una vez presentados los conocimientos preliminares necesarios para la comprensión de la parte matemática del proyecto, se procede a describir los fundamentos de cada uno de los modelos empleados en la práctica. Se comenzará con la regresión exponencial, dentro del bloque de los algoritmos de aprendizaje supervisados tradicionales.

2. Modelos de clásicos de aprendizaje supervisado

2.1. ¿Qué es el aprendizaje automático?

El aprendizaje automático, también conocido como *machine learning*, es una rama de la inteligencia artificial que tiene como objetivo desarrollar algoritmos y modelos capaces de aprender automáticamente a partir de datos y poder hacer predicciones, tomar decisiones o encontrar patrones en nuevos datos similares.

Dentro del aprendizaje automático existen varios tipos: aprendizaje supervisado, no supervisado y por refuerzo. En este trabajo sólo se trabajará con modelos de aprendizaje supervisado para problemas de regresión.

En los modelos de regresión se tiene un conjunto de $N \in \mathbb{N}$ datos $\mathcal{D} = \{(x_n, y_n); n \in \mathbb{N}, n \leq N\}$, donde x_n pertenece al conjunto de posibles entradas X , e y_n es la salida asociada a esa entrada y perteneciente al conjunto de posibles salidas Y . El objetivo es encontrar una función $\varphi : X \rightarrow Y$ que consiga generalizar la relación entre las variables de entrada y las de salida. Tanto x_n como y_n pueden ser vectores, \mathbf{x}_n e \mathbf{y}_n , aunque lo más usual es que sólo las entradas lo sean. En un problema de regresión, las salidas son valores reales y se busca que $\varphi(x_n) \approx y_n$.

En la notación empleada, se denota con mayúscula X a la variable aleatoria (siguiendo así con la notación usada en los preliminares) y con minúscula x un valor específico tomado por dicha variable.

2.2. Regresión exponencial

Para poder comprender las matemáticas que hay detrás de los modelos basados en regresiones exponenciales, es necesario conocer los fundamentos de los modelos basados en regresiones lineales.

Los modelos lineales, junto con sus variantes más sofisticadas, proporcionan unas bases sólidas para el estudio el comportamiento de una variable aleatoria en términos de otras variables.

En esta sección, se verán los fundamentos matemáticos de la regresiones exponenciales y lineales. En esta primera parte, donde se trata la regresión lineal, se sigue a [15] y [16].

2. Modelos de clásicos de aprendizaje supervisado

2.2.1. Regresión lineal

Al trabajar con series temporales de tipo financiero, la predicción del comportamiento de la variable aleatoria objetivo se puede hacer en función de sus propias observaciones anteriores. Este tipo de modelos son conocidos como modelos autorregresivos y son los más utilizados cuando se trabaja con series temporales.

Definición 2.1 (Modelo lineal). Un modelo lineal es una expresión de la forma $Y = X\beta + \varepsilon$, donde $Y = (Y_1, Y_2, \dots, Y_n)^T$ es un vector (columna) aleatorio n-dimensional, observable, X es una matriz de orden $n \times k$ con $k < n$, a la que se denota por matriz de diseño, $\beta = (\beta_1 \dots \beta_k)^T$ es un vector desconocido, no observable, al que se conoce como vector de efectos y $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)^T$ es un vector aleatorio, no observable que se denomina vector de errores.

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nk} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

Para estimar el vector de efectos se hace uso del método de mínimos cuadrados. Para ello, hemos de suponer que el vector de errores esté compuesto por variables aleatorias de segundo orden, centradas, incorreladas y homocedásticas. Es decir, $\forall i = 1, \dots, n$ se ha de verificar: $E[\varepsilon_i] = 0$, $E[\varepsilon_i^2] = \sigma^2$ y $E[\varepsilon_i \varepsilon_j] = 0, i \neq j$.

Más concretamente, para que la regresión lineal pueda modelar correctamente se han de cumplir ciertas características:

- Independencia de los errores: Los errores no están correlacionados entre sí. Formalmente $E[\varepsilon_i \varepsilon_j] = 0, i \neq j$.
- Homocedasticidad: La varianza de los errores es constante en todos los niveles de las variables predictoras. Es decir, $\text{Var}(\varepsilon_i) = \sigma^2$. Dado que se ha de suponer que los errores son variables aleatorias centradas, $E[\varepsilon_i] = 0$, $\text{Var}(\varepsilon_i) = E[\varepsilon_i^2] = \sigma^2$
- No multicolinealidad: Baja correlación entre las variables independientes.

Sea $Y = X\beta + \varepsilon$ un modelo lineal de Gauss-Markov, es decir verificando $E[\varepsilon] = 0$ y $E[\varepsilon \varepsilon^T] = \sigma^2 I_n$. Cada componente del vector Y verifica

$$Y_i = \sum_{j=1}^k x_{ij}\beta_j + \varepsilon_i, \quad i = 1, \dots, n \quad \text{con } E[\varepsilon_i] = 0, \text{Var}[\varepsilon_i] = \sigma^2 \text{ y } \text{Cov}[\varepsilon_i, \varepsilon_j] = 0, i \neq j.$$

Para estimar el modelo hay que estimar el vector de efectos del mismo y la varianza. Esta estimación se realiza, en muchos casos, mediante el método de mínimos cuadrados, el cual consiste en minimizar la suma de los cuadrados de los errores cometidos al aproximar Y por $X\beta$, esto es, se trata de minimizar la función

$$S^2(\beta) = \sum_{i=1}^n \varepsilon_i^2 = \sum_{i=1}^n \left(Y_i - \sum_{j=1}^k x_{ij}\beta_j \right)^2 = \|Y - X\beta\|^2$$

Cualquier solución de las ecuaciones normales que resulte en un mínimo absoluto de $S^2(\beta)$ se denomina un estimador de mínimos cuadrados de β y se denota por $\hat{\beta}(Y)$, o simplemente $\hat{\beta}$.

En el caso de la regresión lineal se asume que $X_1 \dots X_n$ influyen de forma lineal. Es decir, la función φ es lineal, donde $Y = \varphi(X_1 \dots X_n) + E$ (E debe ser una variable aleatoria que expresa el error cometido al predecir Y por $\varphi(X_1 \dots X_n)$).

Formalizando el planteamiento, se tiene que sean X e Y dos variables aleatorias tales que $E[Y^2] < +\infty$ y, fijado un valor arbitrario $X = x$, se cumple:

- $\mathbb{E}[Y | X = x] = \beta_0 + \beta_1 x,$
- $\text{Var}[Y | X = x] = \sigma^2.$

Sean x_1, \dots, x_n un conjunto de valores de X (se exigen al menos dos valores distintos). Supongamos que para cada $i = 1, \dots, n$, Y_i es la observación aleatoria de la variable Y supuesto que $X = x_i$. Entonces, $\forall i = 1, \dots, n$ se verifica

- $\mathbb{E}[Y_i] = \mathbb{E}[Y | X = x_i] = \beta_0 + \beta_1 x_i,$
- $\text{Var}[Y_i] = \text{Var}[Y | X = x_i] = \sigma^2.$

Por tanto, $Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$ con $\mathbb{E}[\varepsilon_i] = 0$ y $\text{Var}[\varepsilon_i] = \sigma^2$, $i = 1, \dots, n$. Es decir, se tiene

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

Este resulta ser un modelo lineal de rango máximo, al exigir que $\exists i, j : x_i \neq x_j$.

Coefficiente de determinación

Notando por \hat{Y} a la estimación del modelo $Y = X\beta$, es decir $\hat{Y} = X\hat{\beta}$ se definen los siguientes conceptos:

- La variabilidad total (VT) de las observaciones aleatorias Y_1, \dots, Y_n viene determinada por:

$$\sum_{i=1}^n (Y_i - \bar{Y})^2$$

- La variabilidad explicada (VE), que es la medida de la variabilidad de las estimaciones $\hat{Y}_1, \dots, \hat{Y}_n$:

$$\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2$$

2. Modelos de clásicos de aprendizaje supervisado

- La variabilidad no explicada (VNE), que es la variabilidad de la parte residual:

$$\sum_{i=1}^n R_i^2 = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Se tiene que la variabilidad total puede descomponerse en función de la variabilidad explicada y la no explicada, es decir, $VT = VE + VNE$.

Definición 2.2 (Coeficiente de determinación). El coeficiente de determinación se define como la proporción de la variabilidad total explicada por el modelo de regresión:

$$R^2 = \frac{VE}{VT} = \frac{\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}$$

Propiedades:

1. R^2 es adimensional, por lo que es una medida de bondad adecuada para hacer comparaciones.
2. $0 \leq R^2 \leq 1$.
3. Cuanto mayor sea R^2 , mejor será el modelo.

Obsérvese que el coeficiente de determinación mide la bondad del modelo correctamente bajo los supuestos que se han planteado anteriormente para el uso adecuado de la regresión lineal. Para el caso práctico que se trata en este proyecto, el coeficiente de determinación no es aplicable, pues los modelos empleados no se ajustan mediante mínimos cuadrados ordinarios y, por tanto, no garantizan la descomposición $VT=VE+VNE$. Además, la variable objetivo (retornos de activos) presenta una media próxima a cero, lo que hace que la variabilidad total sea muy reducida y que esta métrica pierda su capacidad interpretativa.

2.2.2. Regresión exponencial

Para esta sección se ha seguido a [17]. En primer lugar, dos breves apuntes necesarios para la definición de la regresión exponencial.

Definición 2.3 (Función exponencial). Sea $a \in \mathbb{R}$, una función exponencial $f : \mathbb{R} \rightarrow \mathbb{R}$ es aquella definida como:

$$f(x) = a^x$$

Observación 2.4. Un caso particular de la función exponencial es la que tiene como base el número e , cuya inversa es la función logaritmo natural $\ln :]0, \infty[\rightarrow \mathbb{R}$.

Se tiene que $\forall u, v \in \mathbb{R}^+$:

- $\ln(uv) = \ln(u) + \ln(v)$
- $\ln(\frac{u}{v}) = \ln(u) - \ln(v)$

- $\ln(u^k) = k \ln(u) \quad \forall k \in \mathbb{R}$

La regresión exponencial es un tipo de modelo no lineal que asume una relación exponencial entre las variables. Este modelo tiene la forma:

$$Y_i = \beta_0 \cdot \beta_1^{x_i} \cdot \varepsilon_i, \quad i = 1, \dots, n,$$

donde ε_i son variables aleatorias positivas que representan el error multiplicativo, con $\mathbb{E}[\ln(\varepsilon_i)] = 0$.

Observamos que este modelo es linealizable, ya que podemos aplicar el logaritmo natural a ambos lados de la expresión y obtener un modelo lineal, explicado anteriormente:

$$\ln(Y_i) = \ln(\beta_0) + x_i \ln(\beta_1) + \ln(\varepsilon_i)$$

Renombrando $Y'_i = \ln(Y_i)$, $\beta'_0 = \ln(\beta_0)$, $\beta'_1 = \ln(\beta_1)$ y $\varepsilon'_i = \ln(\varepsilon_i)$ se tiene el modelo lineal:

$$Y'_i = \beta'_0 + \beta'_1 x_i + \varepsilon'_i$$

Matricialmente, queda expresado como:

$$\mathbf{Y}' = \begin{pmatrix} \ln(Y_1) \\ \vdots \\ \ln(Y_n) \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, \quad \boldsymbol{\beta}' = \begin{pmatrix} \ln(\beta_0) \\ \ln(\beta_1) \end{pmatrix}, \quad \boldsymbol{\varepsilon}' = \begin{pmatrix} \ln(\varepsilon_1) \\ \vdots \\ \ln(\varepsilon_n) \end{pmatrix}$$

De esta forma, podemos estimar β'_0 y β'_1 mediante el método de mínimos cuadrados y luego transformarlos para obtener β_0, β_1 .

2.2.3. Observaciones y limitaciones

Como se ha explicado en la sección anterior, la aplicación de modelos lineales y exponenciales a series temporales financieras presenta limitaciones significativas. La hipótesis de independencia e idéntica distribución en las observaciones rara vez, por no decir nunca, se cumple. Esto es debido, principalmente, a que las series temporales de activos financieros presentan propiedades que violan las suposiciones de los modelos anteriores:

- Autocorrelación: Es claro que los precios de un día dependen de los precios de los días anteriores. Esto hace que la suposición de independencia sea errónea.
- Heterocedasticidad: La varianza de los precios de los activos no es constante a lo largo del tiempo, hay períodos de mayor y menor volatilidad. Esto contradice la suposición de homocedasticidad.
- Eventos extremos: Los mercados financieros están influenciados por factores muy complejos, lo que lleva a relaciones no lineales y la ocurrencia de valores extremos. Los modelos lineales y exponenciales simples no tienen la capacidad de capturar estas complejidades.

Aunque el uso de estos modelos haya servido como base, las predicciones obtenidas dejan

2. Modelos de clásicos de aprendizaje supervisado

mucho que desear, como ya se verá en la parte de resultados.

No se utilizan en este proyecto, pero existen versiones de los modelos lineales clásicos que han sido desarrolladas precisamente para abordar estas limitaciones [16]. Los modelos de mínimos cuadrados balanceados (*Weighted Least Squares*) permiten tratar con los casos en los que la varianza de los errores fluctúa, es decir, heterocedasticidad. Esto se consigue introduciendo una matriz diagonal formada por las varianzas y covarianzas de los errores. Para tratar la autocorrelación se usan los modelos de mínimos cuadrados generalizados (*Generalized Least Squares*).

Por otro lado, existe una limitación que no se ha mencionado explícitamente en ningún momento: el coste computacional. El uso del método de mínimos cuadrados tiene un coste computacional que lo hace no escalable, pues requiere invertir una matriz (o calcular su pseudoinversa), lo que puede resultar prohibitivo en conjuntos de datos enormes. Es por ello por lo que, en la práctica, realmente se hace uso de la técnica de gradiente descendente, que se explicará posteriormente.

Aunque haya versiones más versátiles de los modelos lineales que podrían utilizarse en este proyecto, se ha optado por hacer uso de modelos más potentes. A continuación se presentan dos modelos de aprendizaje automático ampliamente conocidos: regresores de vectores de soporte (SVR) y bosques aleatorios (*RandomForest*, emsembles de árboles de regresión).

2.3. Support Vector Regression

En esta sección, se explican los fundamentos matemáticos detrás de SVR. La Regresión con Vectores de Soporte, *Support Vector Regression* (SVR) en inglés, es una extensión de las Máquinas de Vectores de Soporte (SVM), que se emplean para resolver problemas de clasificación, a problemas de regresión de valores reales. Para este capítulo se ha seguido a [18] y [19].

El objetivo principal de SVR, al igual que el resto de modelos de aprendizaje automático vistos, es determinar una función de regresión $\varphi(x)$ que prediga un valor continuo y con la mayor precisión posible, pero permitiendo un margen de tolerancia arbitrario ϵ en el error. Se quiere también que esta función no "sobreajuste", es decir, que permita generalizar correctamente para datos no vistos. Cuando se quiere que la función $\varphi(x)$ varíe lo menos posible con respecto a los cambios de x se dice, en este contexto, que la función ha de ser lo más "plana" posible.

2.3.1. Support Vector Machines

Las Máquinas de Vectores de Soporte son métodos de aprendizaje supervisado utilizados en problemas de clasificación binaria en las que se formula el problema como uno de optimización. El objetivo es encontrar el hiperplano que maximice la distancia con respecto a los puntos de datos más cercanos de cada clase, conocidos como vectores soporte. A partir de este hiperplano, SVM asigna las nuevas observaciones a uno de los dos lados del hiperplano, donde cada lado determina una clase.

Las SVM están cimentadas en la teoría de Vapnik-Chervonenkis, desarrollada a finales del siglo XX, popularizándose su uso a finales del siglo XX y consolidándose como, posiblemente, el método de clasificación más utilizado durante la primera década del siglo XXI [20].

La principal diferencia entre SVM y su generalización, SVR, es que este último permite su uso para problemas de regresión. Esto se consigue mediante el uso de una región ϵ -insensible alrededor de la función, llamada ϵ -tubo.

2.3.2. Planitud y pérdida ϵ -insensible

Como se ha dicho anteriormente, el objetivo de SVR es encontrar una función $\varphi(x)$ que se desvíe como máximo ϵ de los valores objetivo y_i para todos los datos de entrenamiento y que, al mismo tiempo, sea lo más "plana" posible. Esta "planitud" se logra minimizando la norma del vector de pesos $\|w\|^2$. Con esto se consigue una función más simple y se evita el sobreajuste.

La función de pérdida ϵ -insensible, denotada como L_ϵ , mide el error de predicción ignorando las desviaciones pequeñas hasta un umbral ϵ . Formalmente se tiene:

Definición 2.5 (Función ϵ -sensible). Dada una predicción $\varphi(x)$ y el valor real observado y , la función ϵ -insensible se define como:

$$L_\epsilon(y, \varphi(x)) = \max\{0, |y - \varphi(x)| - \epsilon\}$$

Esta función permite que el modelo sea robusto frente al ruido ya que las pequeñas fluctuaciones de los datos no afecten a la función de coste. Sin embargo, se ha de tener en cuenta que, a mayor ϵ , mayor tolerancia y el modelo dará estimaciones más generales y, a menor ϵ el modelo tenderá al sobreajuste.

La elección de la función de pérdida se ve influida por la distribución del ruido que afecta a las muestras de datos, la dispersión del modelo buscada y la complejidad computacional del entrenamiento.

La mayoría de las veces, las funciones de pérdida seleccionadas son simétricas y convexas. Aunque pueden escogerse funciones de pérdida asimétricas para limitar la subestimación o la sobreestimación, las funciones de pérdida deben ser convexas para garantizar que el problema de optimización tenga una solución única que pueda encontrarse en un número finito de pasos.

Hay casos en los que no es posible que todos los puntos del modelo caigan dentro del ϵ -tubo, para lidiar con esto se introducen las variables de holgura: $\xi_i > 0, \xi_i^* > 0$

- ξ_i^* mide la desviación de los puntos donde el valor predicho es menor que el real por más de ϵ . Es decir, el punto está por encima del límite superior del ϵ -tubo.
- ξ_i mide justo lo contrario, la desviación de los puntos donde el valor predicho es mayor que el real por más de ϵ . Cuando el punto está por debajo del límite inferior del ϵ -tubo.

2. Modelos de clásicos de aprendizaje supervisado

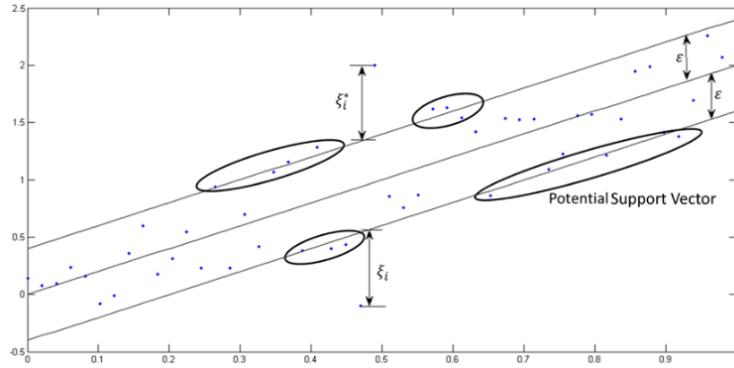


Figura 2.1.: SVR unidimensional lineal [18]

2.3.3. Formulación de SVR

Formulación primal de SVR

Sean $\mathcal{D} = \{(x_n, y_n); n \in \mathbf{N}, n \leq N\} \subset X \times \mathbb{R}$ los datos de entrenamiento y $X \subset \mathbb{R}^N$ el espacio de los datos de entrada. Dada la función

$$\varphi(x) = \langle w, x \rangle + b \quad w \in \mathbb{R}^n, b \in \mathbb{R} \quad (2.1)$$

siendo w el vector de pesos y b el término independiente y $\langle \cdot, \cdot \rangle$ es el producto escalar en X . El objetivo es encontrar w y b que hagan que $\varphi(x)$ sea lo más plana posible. Esto se puede reescribir como el siguiente problema de optimización convexa:

$$\begin{aligned} & \text{minimizar} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \\ & \text{condicionado a} \quad \begin{cases} y_i - \langle w, x_i \rangle - b \leq \epsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \end{aligned} \quad (2.2)$$

La constante $C > 0$ determina el equilibrio entre la planitud de f , representado por $\|w\|^2$, y la cantidad de desviaciones mayores que ϵ toleradas, $\sum_{i=1}^N (\xi_i + \xi_i^*)$. Un C alto implica que el modelo está fuertemente penalizado por los errores, causando que el modelo sobreajuste. Por el contrario, un C bajo permite que haya más errores, haciendo que el modelo generalice mucho y no dé buenos resultados. La selección de valor de C adecuado es crucial.

Formulación dual de SVR

El problema de optimización primal de SVR es del tipo cuadrático convexo, lo que garantiza una solución única y global. La resolución de este problema se resuelve más fácilmente a través de su formulación dual. Esto se consigue mediante el uso de multiplicadores de Lagrange.

Teorema 2.6 (Multiplicadores de Lagrange). Sean $f : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ y $g : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ funciones C^1 con valores reales dados. Sea $\mathbf{x}_0 \in X$ tal que $g(\mathbf{x}_0) = c$, y sea $S = \{\mathbf{x} \in \mathbb{R}^n \mid g(\mathbf{x}) = c\}$ la restricción de igualdad dada por la función g igual a la constante c . Se supone $\nabla g(\mathbf{x}_0) \neq \mathbf{0}$.

Si $f|_S$ (f restringida a S) tiene un máximo o un mínimo local en S , en \mathbf{x}_0 , entonces existe un número real λ tal que $\nabla f(\mathbf{x}_0) = \lambda \nabla g(\mathbf{x}_0)$.

(véase [21])

La idea clave es construir una función de Lagrange a partir de (2.2). Se procede como sigue:

$$\begin{aligned} L := & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) - \sum_{i=1}^N (\eta_i \xi_i + \eta_i^* \xi_i^*) \\ & - \sum_{i=1}^N \alpha_i (\varepsilon + \xi_i - y_i + \langle w, x_i \rangle + b) \\ & - \sum_{i=1}^N \alpha_i^* (\varepsilon + \xi_i^* + y_i - \langle w, x_i \rangle - b) \end{aligned} \quad (2.3)$$

Donde L es el Lagrangiano y $\eta_i, \eta_i^*, \alpha_i, \alpha_i^*$ son los multiplicadores de Lagrange, todos no negativos. Por la condición de punto de silla, las derivadas de L respecto a las variables primarias $(\omega, b, \xi_i, \xi_i^*)$ han de anularse, es decir:

$$\partial_b L = \sum_{i=1}^N (\alpha_i^* - \alpha_i) = 0 \quad (2.4)$$

$$\partial_w L = w - \sum_{i=1}^N (\alpha_i - \alpha_i^*) x_i = 0 \quad (2.5)$$

$$\partial_{\xi_i^*} L = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \quad (2.6)$$

Sustituyendo (2.4), (2.5) y (2.6) en (2.3) da como resultado el problema dual de optimización:

$$\begin{array}{ll} \text{maximizar} & \left\{ \begin{array}{l} -\frac{1}{2} \sum_{i,j=1}^N (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle x_i, x_j \rangle \\ -\varepsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) + \sum_{i=1}^N y_i (\alpha_i - \alpha_i^*) \end{array} \right. \\ \text{condicionado a} & \sum_{i=1}^N (\alpha_i - \alpha_i^*) = 0 \quad \text{and} \quad \alpha_i, \alpha_i^* \in [0, C] \end{array}$$

De (2.5) se tiene $\omega = \sum_{i=1}^N (\alpha_i - \alpha_i^*)$, llamada expansión de vector de soporte. El vector de pesos, ω , se expresa como una combinación lineal de los patrones de entrenamiento, por lo que para calcular $\varphi(x)$ no necesitamos saber ω explícitamente, pues sustituyendo en (2.1):

$$\varphi(x) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) \langle x_i, x \rangle + b \quad (2.7)$$

2. Modelos de clásicos de aprendizaje supervisado

Cálculo del término independiente

Para calcular el término independiente b se hace uso de las condiciones de Karush-Kuhn-Trucker, concretamente de la condición estacionaria. Ésta sostiene que los términos que acompañan a los multiplicadores de Lagrange han de ser 0. Para profundizar más en el teorema de Karush-Kuhn-Trucker y sus condiciones, véase [22]. Aplicando KKT a (2.3) se tiene:

$$\begin{aligned}\alpha_i(\epsilon + \xi_i - y_i + \langle w, x_i \rangle + b) &= 0 \\ \alpha_i^*(\epsilon + \xi_i^* + y_i - \langle w, x_i \rangle - b) &= 0\end{aligned}$$

Usando también (2.5):

$$\begin{aligned}(C - \alpha_i)\xi_i &= 0 \\ (C - \alpha_i^*)\xi_i^* &= 0\end{aligned} \tag{2.8}$$

A partir de estas 4 igualdades se pueden sacar varias condiciones. En primer lugar, la definición de vector de soporte.

Definición 2.7 (Vector de soporte). Las muestras de entrenamiento (x_i, y_i) cuyos multiplicadores de Lagrange correspondientes α_i o α_i^* son distintos de cero se consideran vectores de soporte.

Los vectores de soporte son los puntos de entrenamiento más influyentes, pues definen la forma del ϵ -tubo y la función de regresión.

En segundo lugar, el término b se calcula a partir de los Vectores de Soporte que estén en el límite del ϵ -tubo; es decir, los que cumplen que $0 < \alpha_i < C$ o $0 < \alpha_i^* < C$. Consecuentemente, por (2.8), se tiene que ξ_i y ξ_i^* son 0 y las restricciones se convierten en igualdades:

- Si $0 < \alpha_i < C$, entonces $\epsilon - y_i + \langle w, x_i \rangle + b = 0 \implies b = y_i - \epsilon - \langle w, x_i \rangle$.
- Si $0 < \alpha_i^* < C$, entonces $\epsilon + y_i - \langle w, x_i \rangle - b = 0 \implies b = \epsilon + \langle w, x_i \rangle - y_i$.

2.3.4. Extensión de SVR a problemas no lineales mediante el truco del kernel

Para tratar con los problemas no lineales (la inmensa mayoría de los problemas reales), la idea es aplicar un preprocesamiento a los datos de entrada x_i mediante una función de mapeo $\Phi : X \rightarrow \mathcal{F}$ a un espacio de características \mathcal{F} de mayor dimensión, en el que poder aplicar SVR de forma lineal como se ha explicado antes.

Este mapeo se realiza de forma implícita, no hay que calcular las coordenadas, pues el algoritmo SVR solo depende del producto escalar entre los datos de entrada. En lugar de calcular $\langle \Phi(x_i), \Phi(x_j) \rangle$, lo cual es computacionalmente inviable en muchos casos, se hace uso de una función kernel que se define como $k(x, x') := \langle \Phi(x), \Phi(x') \rangle$. Reformulando (2.7):

$$\begin{aligned}\omega &= \sum_{i=1}^N (\alpha_i - \alpha_i^*) \Phi(x_i) \\ \varphi(x) &= \sum_{i=1}^N (\alpha_i - \alpha_i^*) k(x_i, x) + b\end{aligned}$$

Condiciones para los kernels

La pregunta que uno puede hacerse es: ¿qué funciones kernel se corresponden con el producto vectorial en algún espacio de características \mathcal{F} ? El criterio fundamental para determinar los kernels válidos viene de la mano del siguiente teorema:

Teorema 2.8 (Teorema de Mercer 1909). *Supóngase que:*

- $k \in L_\infty(X^2)$ y el operador integral

$$T_k f(\cdot) := \int_X k(\cdot, x) f(x) d\mu(x)$$

es positivo, donde μ es una medida sobre X con $\mu(X) < \infty$ y $\text{supp}(\mu) = X$;

- $\psi_j \in L_2(X)$ es una función propia de T_k asociada al valor propio $\lambda_j \neq 0$, normalizada con $\|\psi_j\|_{L_2} = 1$, y $\overline{\psi_j}$ denota su conjugado complejo.

Entonces:

1. $(\lambda_j(T))_{j \in \mathbb{N}} \in N_1$.
2. $k(x, x') = \sum_{j \in \mathbb{N}} \lambda_j \psi_j(x) \overline{\psi_j(x')}$ para casi todo (x, x') , con convergencia absoluta y uniforme.

Lo que quiere decir este teorema es que si se cumple:

$$\int_{X \times X} k(x, x') f(x) f(x') dx dx' \geq 0 \quad \forall f \in L_2(X)$$

entonces la función kernel $k(x, x')$ corresponde con el producto escalar en algún espacio de características. El truco del kernel es válido si la función kernel realmente representa un producto escalar en algún espacio de características.

Ejemplos de kernels válidos

Dado $p \in \mathbb{N}$, se tiene que una función kernel completamente válida es la dada por

$$k(x, x') = \langle x, x' \rangle^p$$

Donde para $p = 1$ se tiene la función kernel lineal. Consecuentemente, se tiene también que funciones kernel válidas son las del tipo polimomial, las cuales modelan relaciones no lineales.

$$k(x, x') = (\langle x, x' \rangle + c)^p$$

Existen también funciones kernel menos simples. Un buen ejemplo es la función sigmoide, aunque ésta no siempre cumple el Teorema de Mercer. Sin embargo, en la práctica, ha sido usada con éxito.

$$k(x, x') = \tanh(\vartheta + \kappa \langle x, x' \rangle)$$

2. Modelos de clásicos de aprendizaje supervisado

2.3.5. Otras funciones de coste

Además de la función ϵ -insensible definida anteriormente (2.5), existen otras funciones de coste con distintas implicaciones para la robustez y la dispersión de la solución. Denotamos por $c(x, y, \varphi(x))$ a la función de coste, es decir, la función que determina cómo se penalizan los errores de estimación.

Si se conoce a priori la distribución del ruido $p(\xi)$, se puede conseguir el equilibrio perfecto entre la complejidad de la función de coste y el rendimiento del modelo mediante el uso de máxima verosimilitud. Algunos casos comunes son los que aparecen en la siguiente tabla:

| Función de pérdida | Expresión $c(\xi)$ | Distribución $p(\xi)$ |
|--------------------------|--|--|
| ϵ -insensible | $ \xi _\epsilon$ | $\frac{1}{2(1+\epsilon)} \exp(- \xi _\epsilon)$ |
| Laplaciana | $ \xi $ | $\frac{1}{2} \exp(- \xi)$ |
| Gaussiana | $\frac{1}{2}\xi^2$ | $\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\xi^2}{2}\right)$ |
| Pérdida robusta de Huber | $\begin{cases} \frac{1}{2\sigma}\xi^2 & \text{si } \xi \leq \sigma \\ \xi - \frac{\sigma}{2} & \text{en otro caso} \end{cases}$ | $\begin{cases} \exp\left(-\frac{\xi^2}{2\sigma}\right) & \text{si } \xi \leq \sigma \\ \exp\left(\frac{\sigma}{2} - \xi \right) & \text{en otro caso} \end{cases}$ |
| Polinómica | $\frac{1}{p} \xi ^p$ | $\frac{p}{2\Gamma(1/p)} \exp(- \xi ^p)$ |
| Polinómica a trozos | $\begin{cases} \frac{1}{p\sigma^{p-1}}\xi^p & \text{si } \xi \leq \sigma \\ \xi - \sigma\frac{p-1}{p} & \text{en otro caso} \end{cases}$ | $\begin{cases} \exp\left(-\frac{\xi^p}{p\sigma^{p-1}}\right) & \text{si } \xi \leq \sigma \\ \exp\left(\sigma\frac{p-1}{p} - \xi \right) & \text{en otro caso} \end{cases}$ |

Tabla 2.1.: Funciones de pérdida comunes y sus modelos de densidad correspondientes.

2.3.6. Observaciones y limitaciones

Al igual que sucede con las regresiones exponenciales y lineales, SVR también presenta ciertas limitaciones con el uso de series temporales financieras.

- Heterocedasticidad: SVR también asume que la varianza de los errores es constante, lo que lleva a subestimar o sobreestimar el riesgo en contextos de alta o baja volatilidad.
- Eventos extremos: SVR es muy sensible a los sucesos poco frecuentes que tienen alto impacto, ya que no se adapta bien.
- Elección del kernel: Se ha visto que hay bastantes tipos de kernels, pero no existe uno que funcione en todos los escenarios distintos del mercado.
- No estacionariedad: SVR asume un comportamiento global con un ϵ -tubo fijo, esto lo hace poco flexible a los cambios locales del mercado.

2.4. Random Forest

En este capítulo, se abordarán los fundamentos matemáticos detrás de los bosques aleatorios, *random forests* en inglés. Aparecieron por primera vez hace relativamente poco, cuando en 2001 fueron propuestos en un artículo que servirá de base en este capítulo [23]. Además de éste, se utilizarán los artículos [24], [25] y [26].

En el caso de un problema de regresión, como es el tratado en este proyecto, un random forest promedia las predicciones de varios árboles regresores, consiguiendo mejorar la predicción que consigue un único árbol individual.

Este método de aprendizaje automático ha sido utilizado exitosamente en muchos ámbitos, concretamente en economía. El economista jefe de Google, Hal Varian, destacó el buen desempeño de los random forests en econometría [27]. En este otro artículo [28], los random forests autoregresivos (AR-RF) consiguen muy buenos resultados prediciendo valores de acciones antes y durante la época del COVID-19. Se verá que en este proyecto estos modelos han conseguido predicciones muy acertadas.

2.4.1. Árboles de Regresión y Clasificación – CART

Antes de profundizar en los bosques aleatorios es conveniente definir y explicar los componentes bases de un bosque, los árboles, en particular los árboles de regresión y clasificación.

Un árbol de decisión, o árbol regresor en este caso, es un estimador no paramétrico que partitiona recursivamente el espacio de predictores $X \subset \mathbb{R}^n$ en regiones disjuntas, asociando a cada región (hoja) un valor constante de predicción. Formalmente, un árbol define una partición $\{R_1, R_2, \dots, R_K\}$ de X y estima la respuesta Y mediante la función constante por trozos:

$$T(x) = \sum_{k=1}^K c_k \mathbf{1}\{x \in R_k\}$$

Donde c_k es típicamente el promedio de las respuestas de entrenamiento y_i cuyos predictores x_i caen en la región R_k . Cada región R_k corresponde a una hoja del árbol y se define por un conjunto de pruebas lógicas sobre las variables (p.ej $x_j < t$ para alguna característica j y umbral t).

Los modelos CART (*Classification And Regression Tree*) basados en árboles de decisión pueden aplicarse tanto a resultados categóricos, mediante el uso de árboles de clasificación, como a resultados continuos, mediante el uso de árboles de regresión.

Estos modelos hacen divisiones binarias recursivas partiendo del nodo raíz en función del índice Gini, para el caso de problemas de clasificación, o del ya conocido error cuadrático medio, para problemas de regresión. El objetivo de estas particiones es crear subconjuntos de la muestra lo más homogéneos posibles. Este proceso de división continúa hasta que cada hoja tenga un número mínimo predefinido de muestras, llamado *nodesize*, o hasta que la decisión no aporte una mejora significativa en la homogeneidad.

Una vez que se ha construido el árbol entero las predicciones se realizan de distinta manera

2. Modelos de clásicos de aprendizaje supervisado

según el tipo de problema:

- Árbol de clasificación: Para resultados categóricos, el valor predicho es el valor más común en el nodo final.
- Árbol de regresión: Para resultados continuos, el valor predicho es la media de los valores del nodo final.

Cabe destacar que en los árboles CART típicos hay un parámetro de complejidad c_p que determina qué nodos han de podarse para evitar el sobreajuste. Sin embargo, los árboles que se usan en random forest no suelen podarse, de forma que estos árboles crecen hasta que se alcanza el *nodesize*.

2.4.2. Conceptos Fundamentales de random forest

Al igual que con SVR, se busca una estimación regresiva no paramétrica, con $X \subset \mathbb{R}^N$ el vector aleatorio observado e $Y \in \mathbb{R}$ la variable respuesta.

Un random forest es un modelo predictor que genera las predicciones combinando el resultado de un ensemble de árboles de clasificación o regresión. Sean M el número de árboles y $\mathcal{D} = \{(x_n, y_n); n \in \mathbf{N}, n \leq N\} \subset X \times \mathbb{R}$ los datos de entrenamiento:

Definición 2.9 (Valor de un árbol). Para el j -ésimo árbol, el valor predicho para un valor x se denota como $m(x; \Theta_j, \mathcal{D})$, donde $\Theta_1, \dots, \Theta_M$ son variables aleatorias independientes e idénticamente distribuidas, independientes del conjunto de entrenamiento \mathcal{D} .

El punto fuerte de los random forests respecto a los árboles individuales es su naturaleza de ensemble. Gracias a la combinación de todos los árboles de los que está compuesto el bosque, se reduce el sobreajuste o el error que podría cometer un único árbol. En el contexto de los mercados financieros, esta reducción que consigue un random forest es muy positiva debido a la alta volatilidad y el ruido existente, como ya sabrá el lector.

Una vez visto la definición del valor predicho por el j -ésimo árbol, se tiene que el valor predicho por el random forest es la media de las predicciones de los M árboles individuales, al estar ante un problema de regresión. Es decir:

$$m(x; \Theta_1 \dots \Theta_M, \mathcal{D}) = \frac{1}{M} \sum_{j=1}^M m(x; \Theta_j, \mathcal{D})$$

Parámetros

Hay ciertos parámetros que se pueden cambiar en función de las circunstancias y que tendrán repercusiones en el coste computacional, el sobreajuste o los errores del modelo final.

En primer lugar, antes de la construcción de cada árbol, se extraen a muestras de forma aleatoria, con (o sin) reemplazo, del conjunto de entrenamiento original. Sólo estas observaciones se tienen en cuenta a la hora de la construcción del árbol.

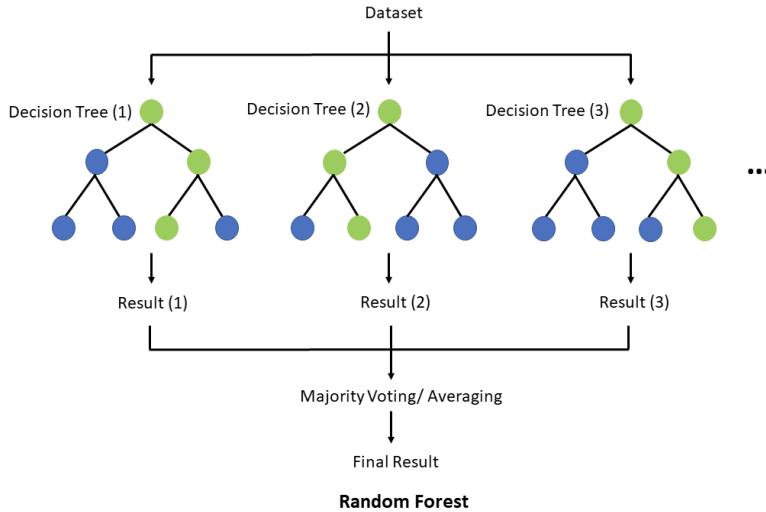


Figura 2.2.: Funcionamiento random forest (TseKiChun, CC BY-SA 4.0 [Wikimedia](#)).

Este procedimiento se conoce como "bagging" (bootstrap-aggregating). Consiste en seleccionar una muestra aleatoria de datos del conjunto de entrenamiento con remplazo, generar un modelo para cada muestra escogida y hacer el promedio de los modelos.

Gracias al bagging, se consigue que los errores cometidos por los árboles individuales estén menos correlacionados, al entrenar cada árbol con un conjunto de datos diferente.

En segundo lugar, a la hora de maximizar el criterio de CART (el cual se define más adelante), se divide el árbol en $M_{try} \in \{1, \dots, N\}$ direcciones elegidas uniformemente al azar. Un valor alto de M_{try} resulta en árboles más correlacionados, lo que hace que el random forest sobreajuste más. Sin embargo, se ha visto que este parámetro no tiene un impacto muy significativo en el modelo [29]. El valor usual para M_{try} es $\frac{N}{3}$.

En tercer lugar, el valor de *nodesize*, del que ya se ha hablado. La construcción de los árboles individuales se detiene cuando cada nodo contiene menos de *nodesize* muestras. Cuanto más bajo sea el *nodesize*, más carga computacional habrá.

Por último, el número de árboles *M*. Un mayor número de árboles en el ensemble hace que el valor predicho sea más preciso y más estable, pero conlleva un mayor coste computacional, como es lógico.

Criterio CART

El criterio de división CART es el encargado de determinar el "mejor" corte para cada nodo (o celda). Sean *A* un nodo cualquiera y $N(A)$ el número de datos del conjunto de entrenamiento \mathcal{D} que "caen" en ese nodo, se define:

Definición 2.10 (Corte). Un corte en *A* es un par (j, z) con $j \in \{1, \dots, N\}$ y *z* la posición del

2. Modelos de clásicos de aprendizaje supervisado

corte para la j -ésima cordenada dentro de los límites de A . \mathcal{C}_A es el conjunto de todos los posibles cortes en A

Definición 2.11 (Criterio CART). Notando por $\mathbf{x}_i = (x_i^{(1)}, \dots, x_i^{(p)})$, para cualquier corte $(j, z) \in \mathcal{C}_A$, el criterio CART se define como:

$$L_{\text{reg},N}(j, z) = \frac{1}{N(A)} \sum_{i=1}^N (Y_i - \bar{Y}_A)^2 \mathbf{1}\{\mathbf{x}_i \in A\}$$

$$- \frac{1}{N(A)} \sum_{i=1}^N \left(Y_i - \bar{Y}_{A_L} \mathbf{1}\{\mathbf{x}_i^{(j)} < z\} - \bar{Y}_{A_R} \mathbf{1}\{\mathbf{x}_i^{(j)} \geq z\} \right)^2 \mathbf{1}\{\mathbf{x}_i \in A\},$$

donde $A_L = \{\mathbf{x} \in A : \mathbf{x}^{(j)} < z\}$, $A_R = \{\mathbf{x} \in A : \mathbf{x}^{(j)} \geq z\}$, y \bar{Y}_A (resp., $\bar{Y}_{A_L}, \bar{Y}_{A_R}$) es la media de los Y_i que pertenecen a A (resp., A_L, A_R), considerando que la media se toma igual a 0 cuando ningún punto \mathbf{x}_i pertenece a A (resp., A_L, A_R). Para cada nodo A , el mejor corte (j^*, z^*) se selecciona maximizando $L_{\text{reg},n}(j, z)$ sobre \mathcal{M}_{try} y \mathcal{C}_A ; es decir,

$$(j^*, z^*) \in \arg \max_{\substack{j \in \mathcal{M}_{\text{try}} \\ (j, z) \in \mathcal{C}_A}} L_{\text{reg},n}(j, z).$$

2.4.3. Margen y generalización

Dado un ensemble (conjunto) de clasificadores $h_1(x), h_2(x), \dots, h_M(x)$ y el conjunto de datos de entrenamiento dado del vector aleatorio (X, Y) , se define la función de margen como:

$$\text{mg}(X, Y) = \frac{1}{M} \sum_{k=1}^M \mathbf{1}\{h_k(X) = Y\} - \max_{j \neq Y} \left(\frac{1}{M} \sum_{k=1}^M \mathbf{1}\{h_k(X) = j\} \right)$$

Lo que mide esta función es la diferencia entre el promedio de votos de la clase verdadera Y y el promedio de la clase con más votos de entre las erróneas j . Cuanto menor sea el valor de la función margen, más probable es que haya un error de clasificación.

Definición 2.12 (Error de generalización). El error de generalización (o probabilidad de error) del bosque es la probabilidad de que el margen sea negativo.

$$PE^* = P_{X,Y}(\text{mg}(X, Y) < 0)$$

En bosques aleatorios, se tiene que $h_k = h_k(X, \Theta_k)$. Para un número alto de árboles, la ley fuerte de los grandes números sostiene el siguiente teorema enunciado por Breiman:

Teorema 2.13 (Convergencia del error). *A medida que el número de árboles crece, PE^* converge casi seguramente a*

$$P_{X,Y}P_\Theta(h(X, \Theta) = Y) - \max_{j \neq Y} P_\Theta(h(X, \Theta) = j) < 0$$

Este resultado explica por qué los bosques aleatorios no sobreajustan conforme más árboles tienen, sino que consigue limitar la probabilidad de error.

Definición 2.14 (Fuerza). La fuerza s de un conjunto de árboles se define como la esperanza del margen:

$$s := \mathbb{E}_{X,Y}[\text{mg}(X, Y)]$$

La fuerza mide la ventaja, de media, que tiene la clase verdadera frente a la clase incorrecta más votada. Cuanto mayor sea la fuerza, más preciso es el resultado dado.

Teorema 2.15 (Acotación del error). *Sea s la fuerza de un conjunto de árboles y ρ la correlación media entre ellos. Si $s > 0$ entonces existe una cota superior para el error de generalización dada por:*

$$PE^* < \frac{\rho(1 - s^2)}{s^2}$$

Aunque este resultado sea más teórico que práctico (en la práctica esta cota no se puede ajustar), recalca los dos valores claves para limitar la probabilidad de error: la fuerza de cada árbol (cada árbol debe tener un error de clasificación bajo para obtener buenos resultados) y la correlación entre árboles (los árboles deben cometer errores diferentes para que el ensemble obtenga buenos resultados).

Para el caso de árboles de regresión, hay pequeños matices. Se ha explicado ya que, para árboles de regresión, se utiliza el error cuadrático medio y que el valor predicho es la media de las salidas $h(x, \Theta_k)$ de cada árbol. El [Teorema 2.13](#) en el caso de regresión:

Teorema 2.16 (Convergencia del error en regresión). *A medida que el número de árboles M en el bosque tiende a infinito, casi seguramente:*

$$\mathbb{E}_{X,Y} \left[\left(Y - \frac{1}{M} \sum_{k=1}^M h(X, \Theta_k) \right)^2 \right] \longrightarrow \mathbb{E}_{X,Y} \left[(Y - \mathbb{E}_\Theta[h(X, \Theta)])^2 \right]$$

Para los árboles de regresión también se puede conseguir una cota superior para el error de generalización.

Teorema 2.17 (Acotación del error en regresión). *Si $\forall \Theta$ cada árbol individual cumple $\mathbb{E}_Y[h(X, \Theta)] = \mathbb{E}_X[Y]$ entonces el error de generalización satisface:*

$$PE^*(\text{forest}) < \rho PE^*(\text{tree})$$

2.4.4. Observaciones y limitaciones

Pese a que los Random Forest han demostrado empíricamente muy buenos resultados predec当地 series temporales, en especial las de carácter económico (como se mencionaba al principio del capítulo), cabe mencionar ciertas limitaciones técnicas, algunas ya familiares.

2. Modelos de clásicos de aprendizaje supervisado

- Heterocedasticidad: Para ciertas pruebas de consistencia teórica para los bosques aleatorios, se requiere cierta homogeneidad en los datos. Se sabe ya que este no es el caso.
- Eventos extremos: Aunque el bagging ayude a reducir el efecto de los valores atípicos, éstos siguen pudiendo afectar a los árboles individuales (CART).
- Selección de parámetros: Al igual que para los SVR se ha dicho que la elección del kernel influye en el rendimiento del modelo y que es complicado que un único kernel funcione bien para todos escenarios en el mercado, un valor constante para cada uno de los parámetros de un Random Forest no dará un resultado óptimo.
- Caja negra: Tanto los SVR, como los Random Forest (como las redes neuronales) tienen una naturaleza intrínseca que hace muy difícil saber cómo está funcionando realmente el modelo.

3. Redes Neuronales

En este capítulo, se explicarán los fundamentos matemáticos de las redes neuronales, así como de los distintos tipos de arquitecturas de redes neuronales que se han utilizado en el proyecto. Aunque las redes neuronales han ganado popularidad en los últimos años, la primera red neuronal fue concebida años antes de la creación y el planteamiento de los dos modelos explicados en el capítulo anterior: SVR y random forest. Para este capítulo se ha seguido a [20].

3.1. ¿Qué es una red neuronal?

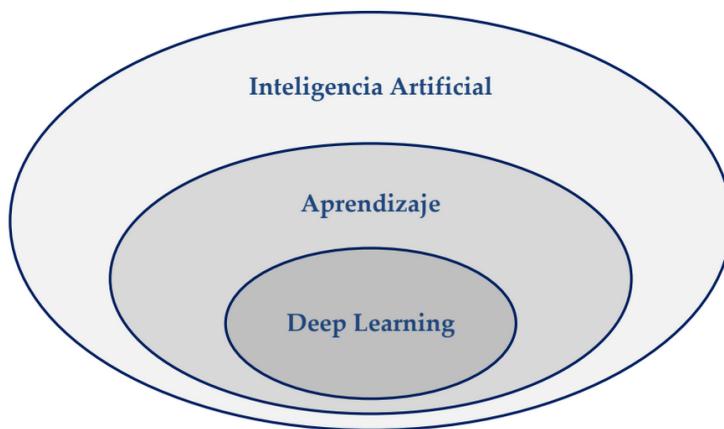


Figura 3.1.: El Deep Learning puede considerarse un subconjunto de las técnicas de aprendizaje, que a su vez son un subconjunto dentro de las técnicas de inteligencia artificial [20].

Las redes neuronales artificiales, conocidas hoy en día bajo el omnipresente nombre *Deep Learning*, son modelos computacionales inspirados en las redes neuronales biológicas. Éstas consisten en un conjunto de unidades (o neuronas) conectadas entre sí, a través de las cuales pasan los datos de entrada, sometidos a múltiples operaciones conforme atraviesan la red, hasta producir los valores de salida.

Lo que diferencia a todas las técnicas basadas en redes neuronales de los modelos explicados en el capítulo anterior es que éstas son capaces de automatizar el proceso de identificar características automáticamente a partir de los datos, con el objetivo de conservar sus rasgos más relevantes y prescindir de los que no lo son.

Las redes neuronales artificiales están formadas por conjuntos de neuronas agrupadas en

3. Redes Neuronales

capas, de tal forma que las neuronas de una misma capa comparten ciertas características. Se denota por x_i a cada una de las n entradas recibidas por una capa formada por m neuronas. La salida de la j -ésima neurona, y_j viene dada por:

$$y_j = f(z_j) = f\left(\sum_i w_{ij}x_i\right) \quad (3.1)$$

Donde w_{ij} son los pesos sinápticos utilizados para modelar las conexiones de entrada a las neuronas, concretamente de la i -ésima entrada con la j -ésima neurona. Los pesos tienen valores reales, positivos para modelar conexiones excitatorias y negativos para conexiones inhibitorias. Por otro lado, z_j es la entrada neta de la j -ésima neurona y f es la función de activación, de la que se hablará en profundidad posteriormente.

Agrupando las entradas en un vector \mathbf{x} y las salidas en \mathbf{y} , la expresión anterior puede escribirse como:

$$\mathbf{y} = f(W\mathbf{x})$$

Donde W es la matriz de pesos, de tamaño $m \times n$. En ocasiones se utiliza un valor fijo b , denominado sesgo o *bias*, que determina en algunos modelos el umbral de activación de la neurona, esto es, el punto a partir del cual la neurona activa su salida.

Se puede o bien añadir los segos de las neuronas a la expresión anterior $f(W\mathbf{x} + b)$, o bien asumir que hay una entrada fija con valor constante 1, $x_0 = 1$, y los sesgos van representados en la matriz de pesos W , que pasa a tener tamaño $m \times (n + 1)$.

Una red con varias capas aplica sucesivamente transformaciones lineales seguidas de funciones de activación, lo que puede verse como una composición de funciones. Denotando por $\mathbf{y}_0 = \mathbf{x}$ la entrada de la red, la salida de la k -ésima capa se obtiene como:

$$\mathbf{y}_k = f(W_k \mathbf{y}_{k-1})$$

Repitiendo este proceso para L capas, la salida final de la red es:

$$\mathbf{y}_L = f(W_L f(W_{L-1} \cdots f(W_1 \mathbf{x})))$$

Esta composición de funciones lineales y no lineales es lo que permite a las redes neuronales modelar relaciones complejas.

Como se ha dicho, las redes neuronales se agrupan en capas, las cuales se suelen dividir en tres tipos:

- Capa de entrada: Es la primera capa de la red y es la que recibe los datos "brutos" del problema.
- Capa oculta: En estas capas es donde ocurre "la magia". Son las que realizan la mayor parte del procesamiento complejo, una capa oculta permite procesar patrones complicados.
- Capa de salida: Es la capa final de la red, encargada de producir el resultado final del problema.

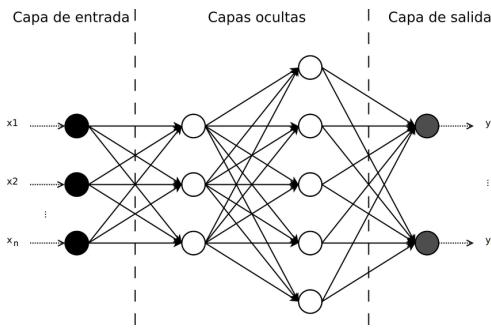


Figura 3.2.: Red neuronal artificial multicapa de tipo *feed-forward*. [ResearchGate](#).

Redes feed-forward

El modelo más clásico de red neuronal que utiliza esta estructura por capas es la red *feed-forward*. En este tipo de redes, los datos fluyen únicamente en una dirección: desde la capa de entrada, pasando por las capas ocultas, hasta la capa de salida. Cada capa procesa la información recibida y la transmite a la siguiente, sin que exista retroalimentación desde capas posteriores hacia anteriores.

Se denominan redes neuronales *feed-forward* precisamente porque carecen de mecanismos de *feedback*. Dentro de ellas, pueden distinguirse distintos niveles de complejidad:

- Redes simples, con una única capa: son las redes neuronales más simples, como su propio nombre indica. Únicamente cuentan con una capa de entrada y otra de salida, que es la única que hace algo. Se dice que tienen una porque sólo tienen una capa con parámetros ajustables.
- Redes multicapa, con una capa oculta: si a una red simple se le añade una única capa oculta, las capacidades de aprendizaje de la red aumentan significativamente. De hecho, una sencilla red neuronal multicapa ya se comporta como un aproximador universal. Dado que las capas ocultas no son visibles al exterior, será necesario el uso de *back-propagation* para ajustar sus parámetros internos.
- Redes profundas, con varias capas ocultas: las redes neuronales actuales suelen incorporar múltiples capas ocultas para poder aprovechar el potencial de las técnicas de *deep learning*.

3.1.1. Funciones de activación

Los modelos de redes neuronales artificiales utilizan la función de activación para determinar cuál es la salida de la neurona. La función de activación recibe como parámetro la entrada neta de la neurona, es decir, la suma ponderada de las entradas y los pesos correspondientes.

El papel de función de activación lo pueden desempeñar muchas funciones. Se pueden tomar tanto funciones de activaciones discretas, como la función escalón; como funciones continuas lineales o no lineales.

3. Redes Neuronales

Función escalón

La función escalón, también conocida como función umbral, fue la primera función no lineal que se utilizó para modelar neuronas artificiales.

$$y = f_{\text{tlu}}(z) = u(z) = \mathbf{1}_{z \geq 0} = \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

Sin embargo, la derivada de esta función es 0, por lo que no se usa en redes actuales, entrenadas con el gradiente descendente. Del mismo modo, la función signo:

$$y = f_{\text{sgn}}(z) = \text{sgn}(z) = \begin{cases} 1 & \text{si } z \geq 0 \\ -1 & \text{si } z < 0 \end{cases}$$

cuya principal diferencia entre ambas es que la segunda es simétrica respecto al origen, tampoco es usada en la actualidad.

Función lineal

La función lineal es posiblemente la función de activación más sencilla que hay. De todas las posibles, la más simple es la función identidad:

$$y = f_{\text{lin}}(z) = z$$

Si se conectan varias capas de neuronas lineales en serie, el resultado final será siempre equivalente a una única capa de funciones lineales. No se diferenciaría de aplicar el primer modelo que se ha explicado en el capítulo anterior: un modelo que utilice una regresión lineal.

Función sigmoidal

Usualmente, interesarán que la función de activación de una neurona sea, además de no lineal, estrictamente creciente, continua y derivable. Las funciones sigmoides satisfacen todos estos requisitos, en concreto la función logística:

$$y = f_{\text{logistic}}(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Es trivial observar que la imagen de esta función es el intervalo (0, 1). Además, la derivada se puede calcular fácilmente ya que cumple:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

El problema de esta función es que cuando los datos tienen valores muy grandes o muy pequeños la función logística tiende a hacerse plana, lo que lleva a que la derivada tienda a 0. A esto se le llama "saturación".

Otra función sigmoidal muy utilizada es la tangente hiperbólica:

$$y = f_{\tanh}(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$

Al igual que la logística, la tangente hiperbólica también puede calcularse fácilmente usando la propia función:

$$\frac{d\tanh(z)}{dz} = (1 + \tanh(z))(1 - \tanh(z)) = (1 - \tanh^2(z))$$

Hay autores que afirman que la tangente hiperbólica funciona mejor que la función logística porque se parece más a la función identidad para valores cercanos a 0.

El interés por que la función sea derivable es debido a que algunas redes neuronales utilizan el algoritmo *back-propagation* (propagación hacia atrás). Éste es un algoritmo que propaga los errores de la desde la capa de salida hacia las capas anteriores en la red. A partir de los errores observados en la salida, ajusta los parámetros internos para mejorar el rendimiento del modelo. La forma en que se ajustan estos parámetros depende de la derivada de la función de activación de las neuronas.

Función lineal rectificada

Las unidades ReLU (*REctified Linear Units*) son muy habituales en *deep learning*: Éstas utilizan una función de activación lineal rectificada, definida como:

$$y = f_{\text{relu}}(z) = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

Cuya derivada coincide con la función escalón ya definida.

$$\frac{df_{\text{relu}}(z)}{z} = u(z)$$

En general, las unidades ReLU tienen una ventaja frente a las unidades sigmoidales cuando se utilizan en redes neuronales entrenadas mediante algoritmos basados en *back-propagation*. Como ya se ha dicho, las unidades sigmoidales tienden a saturarse a partir de cierto valor. El algoritmo de gradiente descendente con *back-propagation* utiliza esta derivada para calcular el gradiente del error. Este gradiente es el que determina cómo modificar los pesos de la red. Una derivada casi nula implica actualizaciones de los pesos muy pequeñas y un entrenamiento lento (incluso puede llegar a estancarse).

La saturación de las neuronas sigmoidales puede dificultar seriamente el entrenamiento de una red neuronal. No obstante, si se utiliza una función de pérdida adecuada, es posible emplear neuronas sigmoidales en la capa de salida, como ocurre en problemas de clasificación. En cambio, cuando se usan neuronas sigmoidales en las capas ocultas, es necesario aplicar técnicas adicionales como la normalización por lotes o por capas para mitigar los efectos de

3. Redes Neuronales

la saturación.

Por otro lado, dado que el uso de unidades ReLU no requiere de funciones trascendentales¹, su evaluación es mucho más eficiente y el entrenamiento de redes neuronales con este tipo de unidades suele ser mucho más rápido que con neuronas sigmoidales.

Aunque también hay casos en los que una función lineal rectificada no es preferible a una sigmoidal al no estar acotada. Esto sucede en las redes neuronales recurrentes. Además, una de las principales limitaciones, si no la principal, de las unidades ReLU es que son incapaces de aprender cuando su nivel de activación es nulo ($z < 0$), ya que su función de activación es completamente plana. Por tanto, suele ser conveniente inicializar los sesgos de las neuronas ReLU con un valor positivo pequeño para que haya más posibilidades de que las neuronas se activen durante las fases iniciales del entrenamiento de la red.

Además, existen las denominadas como *leaky ReLU*, que tienen una pequeña pendiente cuando $z > 0$ para que el gradiente no sea nulo.

3.1.2. La red neuronal como aproximador universal

Una de las características más importantes, si no la que más, de las redes neuronales es su propiedad de aproximador universal, siempre que éstas cuenten con, al menos, una capa oculta con función de activación no lineal. Dicho de forma más coloquial: una red neuronal, con suficientes neuronas y entrenamiento, es capaz de imitar casi cualquier función.

Esta definición que se acaba de dar presenta algunos matices que se abordarán en esta sección, con la presentación de diferentes resultados y teoremas. Para ello, se utilizará como referencia el famoso artículo de Cybenko [30].

En primer lugar, dada la definición de la salida de una neurona (3.1), una red neuronal con n entradas, una capa oculta con k neuronas y una única salida quedaría representada como:

$$G(\mathbf{x}) = \sum_{i=1}^k \alpha_i s \left(\sum_{j=1}^n w_{ji} x_j + b_i \right) = \sum_{i=1}^k \alpha_i s \left(\mathbf{w}_i^\top \mathbf{x} + b_i \right), \quad (3.2)$$

El nuevo término es α_i , que representa el peso que modela la conexión entre la i -ésima neurona y la neurona de salida. Nótese que todas las neuronas en la capa oculta aplican una función sigmoidal s , mientras que la neurona de salida es lineal.

Pese a que ya se han descrito las funciones sigmoidales en Subsección 3.1.1, se presenta una definición más rigurosa.

Definición 3.1 (Función sigmoidal). Una función $s : \mathbb{R} \rightarrow \mathbb{R}$ es sigmoidal si:

$$\lim_{t \rightarrow +\infty} s(t) = 1, \quad \lim_{t \rightarrow -\infty} s(t) = 0.$$

Sea $I_n = [0, 1]^n$ el cubo n -dimensional. Se denota por $C(I_n)$ al espacio de funciones conti-

¹Las funciones trascendentales son aquéllas que no se pueden conseguir a partir de operaciones aritméticas tales como la suma, resta, multiplicación, división y extracción de raíces. La exponencial es un ejemplo.

nusas en I_n y por $M(I_n)$ al espacio de las medidas de Borel regulares, finitas y con signo en I_n .

De manera resumida, una medida puede entenderse como una generalización de la función de probabilidad definida en [Teorema 1.1](#), salvo que no requiere la normalización $\mu(I_n) = 1$. Que sea de Borel significa que está definida sobre la σ -álgebra de Borel (véase [\(1.3\)](#)). Que sea con signo indica que la medida puede tomar valores tanto positivos como negativos. La condición de ser finita implica que $\mu(I_n) < \infty$. Finalmente, la regularidad garantiza que la medida de un conjunto puede aproximarse, en sentido adecuado, mediante abiertos y compactos.

Tanto en [\[31\]](#) (citado en el artículo de Cybenko) como en [\[32\]](#) (usado en la asignatura "Análisis Funcional" del doble grado en la Universidad de Granada) se pueden encontrar estas definiciones.

Definición 3.2 (Función discriminatoria). Una función $s : \mathbb{R} \rightarrow \mathbb{R}$ es discriminatoria si para cualquier medida $\mu \in M(I_n)$ se tiene que

$$\int_{I_n} s(\mathbf{x}^\top \mathbf{w} + b) d\mu(\mathbf{x}) = 0$$

para todo $\mathbf{w} \in \mathbb{R}^n$ y $b \in \mathbb{R}$ implica que $\mu = 0$.

De forma informal, una función s es discriminatoria cuando es capaz de captar cualquier patrón o distribución presente en los datos. En concreto, al considerar todas las combinaciones lineales posibles $\mathbf{x}^\top \mathbf{w} + b$ y aplicar s , se obtiene una familia de funciones lo suficientemente rica como para detectar cualquier variación en la medida μ . Por tanto, si todas las integrales resultan ser cero, la única posibilidad es que la medida no contenga ninguna información que detectar (es decir, que sea la medida nula).

Antes de enunciar el teorema de aproximación universal, se enuncian dos resultados claves para su demostración vistos en la asignatura "Análisis Funcional": el teorema de Hahn-Banach y el teorema de representación de Riesz.

Teorema 3.3 (Hahn–Banach, forma geométrica). *Sea X un espacio normado real, $Y \subset X$ un subespacio cerrado propio, y $x_0 \in X \setminus Y$. Entonces existe un funcional lineal continuo $L : X \rightarrow \mathbb{R}$ tal que*

$$L(y) = 0 \quad \forall y \in Y,$$

$$y \ L(x_0) \neq 0.$$

Teorema 3.4 (Teorema de representación de Riesz–Markov). *Sea K un espacio métrico compacto y $C(K)$ el espacio de funciones continuas reales en K . Entonces, para todo funcional lineal continuo $L \in C(K)^*$ existe una medida de Borel regular finita μ sobre K tal que*

$$L(f) = \int_K f(x) d\mu(x) \quad \forall f \in C(K).$$

Teorema 3.5 (Teorema de aproximación universal). *Sea $s : \mathbb{R} \rightarrow \mathbb{R}$ una función continua y*

3. Redes Neuronales

discriminatoria. Entonces el conjunto de sumas finitas de la forma

$$G(\mathbf{x}) = \sum_{j=1}^k \alpha_j s(\mathbf{x}^\top \mathbf{w}_j + b_j) \quad (3.3)$$

es denso en $C(I_n)$. Es decir, dada $h \in C(I_n)$ y $\varepsilon > 0$ (tan pequeño como se deseé), existe una suma $G(\mathbf{x})$ de la forma descrita tal que

$$|G(\mathbf{x}) - h(\mathbf{x})| < \varepsilon \quad \forall \mathbf{x} \in I_n.$$

Demostración. Sea S el conjunto de todas las funciones de la forma (3.3). El conjunto S es un subespacio lineal de $C(I_n)$. Se desea probar que la clausura de S coincide con $C(I_n)$. Supóngase, en busca de contradicción, que la clausura de S , denotada \overline{S} , no es todo $C(I_n)$. Entonces existe un subespacio cerrado propio $R \subset C(I_n)$ que contiene a \overline{S} . Por el teorema de Hahn–Banach, existe un funcional lineal continuo $L : C(I_n) \rightarrow \mathbb{R}$, con $L \neq 0$, tal que

$$L(h) = 0 \quad \forall h \in R,$$

y en particular $L(h) = 0$ para todo $h \in S$. Por el teorema de representación de Riesz–Markov, dicho funcional L se puede expresar como

$$L(h) = \int_{I_n} h(\mathbf{x}) d\mu(\mathbf{x}),$$

para alguna medida con signo μ definida sobre I_n . Dado que L se anula sobre S , en particular se tiene

$$\int_{I_n} s(\mathbf{x}^\top \mathbf{w} + b) d\mu(\mathbf{x}) = 0 \quad \forall \mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}.$$

Como s es una función discriminatoria, la condición anterior implica necesariamente que

$$\mu = 0.$$

De este modo, el funcional L resulta ser el funcional nulo, lo que contradice la hipótesis de que $L \neq 0$.

Por consiguiente, la suposición inicial es falsa y debe cumplirse que

$$\overline{S} = C(I_n).$$

En conclusión, el conjunto de sumas finitas de la forma (3.3) es denso en $C(I_n)$. \square

De hecho, el teorema sigue siendo válido sustituyendo el cubo I_n por cualquier subconjunto compacto de \mathbb{R}^n .

La principal diferencia entre el teorema y la fórmula dada en (3.2) es el uso de una función sigmoidal en la definición de red neuronal y el uso de una función discriminatoria en el teorema. Sin embargo, con el siguiente lema quedan ambos conceptos casados:

Lema 3.6. *Cualquier función medible, acotada y sigmoidal, s , es discriminatoria. En particular, cualquier función continua sigmoidal es discriminatoria.*

Consecuentemente, se tiene:

Teorema 3.7. *Sea $s : \mathbb{R} \rightarrow \mathbb{R}$ continua sigmoidal. Entonces el conjunto de sumas finitas de la forma*

$$G(\mathbf{x}) = \sum_{j=1}^N \alpha_j s(\mathbf{x}^\top \mathbf{w}_j + b_j)$$

es denso en $C(I_n)$. Es decir, dada $h \in C(I_n)$ y $\varepsilon > 0$ (tan pequeño como se desee), existe una suma $G(\mathbf{x})$ de la forma descrita tal que

$$|G(\mathbf{x}) - h(\mathbf{x})| < \varepsilon \quad \forall \mathbf{x} \in I_n.$$

El resultado de Cybenko asegura que cualquier función continua en un compacto de \mathbb{R}^n podría ser aproximada por una red neuronal con una sola capa oculta, otorgándole así su carácter de aproximador universal.

Sin embargo, Cybenko sólo demuestra el carácter de aproximador universal con funciones sigmoidales, ¿qué pasa entonces con el resto de funciones de activación que se han definido en Subsección 3.1.1?

Este problema fue resuelto posteriormente en [33], en el que se demuestra que una red neuronal es un aproximador universal si y sólo si su función de activación no es un polinomio. Concretamente, se generaliza el segundo de los dos siguientes resultados [34]:

Teorema 3.8 (Primer Teorema de Hornik). *Sea $s : \mathbb{R} \rightarrow \mathbb{R}$ una función de activación acotada y no constante. Entonces, para cualquier medida finita μ , las redes neuronales feedforward estándar con una sola capa oculta pueden aproximar arbitrariamente bien cualquier función en $L^p(\mu)$, (el espacio de todas las funciones f en \mathbb{R}^n tales que $\int_{\mathbb{R}^n} |f(x)|^p d\mu(x) < \infty$), siempre que se disponga de un número suficientemente grande de neuronas ocultas.*

Teorema 3.9 (Segundo Teorema de Hornik). *Sea $s : \mathbb{R} \rightarrow \mathbb{R}$ una función de activación continua, acotada y no constante. Entonces, para cualquier subconjunto compacto $X \subseteq \mathbb{R}^n$, las redes neuronales feedforward estándar con una sola capa oculta pueden aproximar arbitrariamente bien cualquier función continua en X , con respecto a la distancia uniforme, siempre que se disponga de un número suficientemente grande de neuronas ocultas.*

Denótese por $\text{span}\{v_1, v_2, \dots\}$ al conjunto de todas las combinaciones lineales finitas de los vectores $\{v_1, v_2, \dots\}$. Una función no es un polinomio algebraico (casi en todas partes) cuando no coincide con un polinomio salvo en un conjunto de medida nula. Una vez aclarados ambos términos, se pasa a enunciar el teorema principal:

Teorema 3.10 (Teorema de Leshno). *Sea $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ una función de activación localmente acotada y por partes continua. Se define*

$$\Sigma_n = \text{span}\{\sigma(\mathbf{w} \cdot \mathbf{x} + \theta) : \mathbf{w} \in \mathbb{R}^n, \theta \in \mathbb{R}\}.$$

Entonces, Σ_n es denso en $C(\mathbb{R}^n)$ si y sólo si σ no es un polinomio algebraico (casi en todas partes).

3. Redes Neuronales

En particular, el conjunto de sumas definido en (3.3) es denso en $C(\mathbb{R}^n)$. Esto justifica entonces el uso de funciones de activación distintas a las sigmoidales. En particular, funciones no polinómicas como la ReLU, ampliamente utilizadas en la práctica, cumplen las condiciones del teorema de Leshno y, por tanto, poseen el carácter de aproximador universal.

Profundizando más en las funciones rectificadoras, o ReLU, Montúfar [35], demuestra resultados interesantes acerca de la profundidad de las redes neuronales y su capacidad de comportarse como un aproximador universal.

Conviene definir previamente el concepto de "región lineal", pues Montúfar demuestra que una red neuronal poco profunda requiere exponencialmente más unidades que una red neuronal profunda para representar el mismo número de regiones lineales.

Definición 3.11. Sea $G : \mathbb{R}^D \rightarrow \mathbb{R}^m$ una función lineal a trozos. Una región lineal de G es un subconjunto conexo A maximal del espacio \mathbb{R}^D en el que G es lineal. (Maximal significa que no existe un subconjunto conexo $A' \subseteq \mathbb{R}^D$ que contenga estrictamente a A y en el que G siga siendo lineal).

Si se recuerda la definición de la función ReLU en Subsección 3.1.1, se observa que la función no es lineal, pero que, sin embargo, sí es lineal a trozos. Una red neuronal que utilice este tipo de función de activación va a dar lugar a funciones lineales a trozos, pues habrá combinaciones lineales y composiciones de funciones lineales a trozos, dando lugar a regiones lineales.

Teorema 3.12. *El número máximo de regiones lineales que puede representar una red neuronal con D unidades de entrada y L capas ocultas, con $K_i \geq D$ unidades lineales rectificadas en la i -ésima capa oculta ($i \in \{1, \dots, L\}$), está acotado inferiormente por*

$$\left(\prod_{i=1}^{L-1} \left\lfloor \frac{K_i}{D} \right\rfloor^D \right) \sum_{j=0}^D \binom{K_L}{j}.$$

Como consecuencia de este teorema se tiene un corolario que muestra cómo el número de regiones lineales crece de forma exponencial con el número de capas ocultas L : es decir, la profundidad de la red aumenta el número de regiones.

Corolario 3.13. *Una red neuronal que use como función de activación la función rectificador, con D unidades de entrada y L capas ocultas con $K \geq D$ unidades cada una, puede calcular funciones que tienen $\Omega\left(\left(\frac{K}{D}\right)^{(L-1)D} K^D\right)$ regiones lineales.*

Este resultado demuestra que es preferible, en términos de cantidad de regiones lineales, tener una red neuronal más profunda que ancha. Dicho de otro modo, las redes profundas son capaces de producir sustancialmente más regiones lineales que las redes poco profundas.

3.2. Entrenamiento de redes neuronales

Una red neuronal presenta una gran cantidad de posibles configuraciones debido a que tanto los pesos w_{ij} , como los sesgos b_j pueden tomar infinidad de valores. El entrenamiento de la red neuronal se basa en minimizar una determinada función de coste o de pérdida que depende directamente de los pesos y sesgos de la red.

Dado un conjunto de entrenamiento $\mathcal{D} = \{(\mathbf{x}_n, y_n); n \in \mathbb{N}, n \leq N\}$ ² y sea $\varphi(\mathbf{x}_n)$ la salida de la red neuronal con y_n el correspondiente valor real. Una función de coste típica que podría usarse es la media de los errores que comete la red neuronal al cuadrado. Esta función recibe el nombre de "error cuadrático medio" o MSE, por sus siglas en inglés (*Mean Squared Error*):

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N (y_n - \varphi(\mathbf{x}_n))^2$$

Nótese que, aunque en la parte de las funciones en las que se trataban funciones de coste para SVR (Subsección 2.3.5) se usaba la notación $c(x, y, \varphi(x))$, sin embargo, se usará para las funciones de coste de las redes neuronales la notación:

$$\mathcal{L}(w_{1,1}^{(1)}, b_1^{(1)} \dots) := \mathcal{L}$$

ya que los argumentos de la función coste son distintos. Además, se utilizará únicamente \mathcal{L} para no sobrecargar notación.

Nótese que, en la práctica, también se usará la función de pérdida de Huber, la cual combina el error cuadrático medio y el error absoluto medio (MAE, el cual se explicará en el capítulo siguiente). La función de pérdida de Huber se define como:

$$\mathcal{L}_\delta = \frac{1}{N} \sum_{n=1}^N \begin{cases} \frac{1}{2}(y_n - \varphi(\mathbf{x}_n))^2, & \text{si } |y_n - \varphi(\mathbf{x}_n)| \leq \delta \\ \delta \cdot \left(|y_n - \varphi(\mathbf{x}_n)| - \frac{1}{2}\delta\right), & \text{en otro caso} \end{cases}$$

donde δ es un hiperparámetro que determina cuándo cambiar de cuadrático a lineal.

¿Cómo se minimiza la función de coste? Se puede conseguir mediante el uso del gradiente descendente, el cual ya se ha mencionado en alguna ocasión. Se necesitará el gradiente de la función de coste, $\nabla \mathcal{L}$, y el algoritmo de propagación de errores o *back-propagation*.

3.2.1. Gradiente descendente

El gradiente descendente es un algoritmo iterativo usado para encontrar mínimos locales en funciones diferenciables, preferiblemente convexas, en las que el único mínimo local es el mínimo global de la función Teorema 1.33. En el marco de las redes neuronales, sirve para minimizar la ya conocida función de coste \mathcal{L} , la cual, normalmente, no será convexa. Sin embargo, el gradiente descendente es un algoritmo robusto que suele funcionar incluso para funciones no convexas.

²En el capítulo anterior los valores de entrada eran escalares, en este capítulo, por el contrario, serán vectores de tamaño D .

3. Redes Neuronales

Sea, en primer lugar, $f : \mathbb{R}^d \rightarrow \mathbb{R}$ una función convexa y diferenciable. El algoritmo de optimización basado en el gradiente descendente consiste en ir calculando una sucesión de vectores $\mathbf{x}_0, \mathbf{x}_1, \dots$ con \mathbf{x}_0 arbitrario. En cada iteración $t \geq 0$, el término \mathbf{x}_{t+1} se calcula sumándole al término anterior un valor $\mathbf{p}_t \in \mathbb{R}^d$, llamado paso:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{p}_t$$

Estos pasos deben de tomarse de forma que el valor de la función f vaya decreciendo en cada iteración, es decir, $f(\mathbf{x}_{t+1}) < f(\mathbf{x}_t)$ para cada $t \geq 0$. Para ello, al ser f diferenciable, de la definición de diferenciabilidad (1.26), se sigue:

$$\lim_{\|\mathbf{v}_t\| \rightarrow 0} f(\mathbf{x}_{t+1}) = \lim_{\|\mathbf{p}_t\| \rightarrow 0} f(\mathbf{x}_t + \mathbf{p}_t) = \lim_{\|\mathbf{v}_t\| \rightarrow 0} \left(f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^\top \mathbf{v}_t + r(\mathbf{p}_t) \right) = f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^\top \mathbf{p}_t.$$

Si se toma un paso \mathbf{p}_t de forma que $\nabla f(\mathbf{x}_t)^\top \mathbf{p}_t < 0$, f decrece en cada iteración. Se debería elegir \mathbf{p}_t de forma que

$$\nabla f(\mathbf{x}_t)^\top \mathbf{p}_t = \left(\frac{df}{dx_1}(\mathbf{x}_t) \cdot p_1 + \cdots + \frac{df}{dx_d}(\mathbf{x}_t) \cdot p_d \right)$$

tenga el valor más negativo posible (sin aumentar la norma del paso $\|\mathbf{p}_t\|$, pues esto rompería con la definición de diferenciabilidad). Esto se conseguirá cuando \mathbf{p}_t apunte en la misma dirección que $-\nabla f(\mathbf{x}_t)$. El paso, en la versión más sencilla del gradiente descendente, se toma como una constante para todas las iteraciones, notada por η y llamada *tamaño de paso* o *tasa de aprendizaje* (*stepsize* o *learning rate*, en inglés). Consecuentemente, una iteración del gradiente descendente queda definida como:

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t)$$

La tasa de aprendizaje es otro hiperparámetro más que ha de ser escogido cuidadosamente. Una tasa de aprendizaje pequeña deriva en una convergencia del algoritmo innecesariamente lenta. Por otro lado, una tasa de aprendizaje elevada puede desencadenar oscilaciones o la divergencia del algoritmo, impidiendo llegar al mínimo.

Sin embargo, en la práctica suele ser difícil encontrar una tasa de aprendizaje buena para el gradiente descendente. En la mayoría de los casos se recurre a una estrategia de prueba y error, la cual, sin embargo, no suele constituir la solución más adecuada. Existen, sin embargo, múltiples técnicas heurísticas que se pueden usar como criterio para ir cambiando la tasa de aprendizaje (adaptativa) conforme se ejecuta el algoritmo. La tasa de aprendizaje deja de ser una constante η y pasa a depender de la iteración t , $\eta(t)$.

- *AdaGrad*: Ajusta la tasa de aprendizaje de forma individual para cada parámetro, reduciéndola más en aquellos con gradientes grandes y frecuentes.
- *AdaDelta*: Variante de AdaGrad que evita que la tasa de aprendizaje disminuya demasiado, usando medias móviles de gradientes e incrementos.
- *Rprop*: Ignora la magnitud de los gradientes y adapta el tamaño de paso basándose solo en el signo, acelerando la convergencia.
- *RMSprop*: Variante adaptativa que combina la robustez de Rprop con medias móviles

de gradientes al cuadrado para evitar oscilaciones.

- *Adam*: Combina RMSprop con el uso de la media y la varianza, corrigiendo sesgos y mejorando la adaptación.

Además, en el proyecto también se ha utilizado *ReduceLROnPlateau*. Esta técnica consiste en reducir la tasa de aprendizaje (*learning rate*) cuando una métrica monitorizada (por ejemplo, la función de pérdida o la precisión) deja de mejorar durante un número determinado de épocas, conocido como *patience*. Para ello, se ajusta la tasa de aprendizaje multiplicándola por un factor menor que 1, lo que permite refinar la convergencia del modelo y escapar de posibles mesetas en la función de error. La actualización se realiza de la siguiente forma:

$$\eta \leftarrow \eta \cdot \text{factor}$$

cuando la métrica no mejora tras el número de épocas definido por la paciencia.

Por otro lado, además del gradiente descendente clásico que se ha explicado, existen otras variantes muy populares. La más popular de ellas probablemente sea el algoritmo del *gradiente descendente estocástico* (*Stochastic Gradient Descent*, *SGD*, en inglés)

Gradiente descendente estocástico

En muchos problemas de aprendizaje automático, la función que se desea optimizar f se puede expresar como la suma de funciones de coste f_i asociadas al i -ésimo ejemplo del total de n que hay en el conjunto de entrenamiento utilizado:

$$f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n f_i(\mathbf{x})$$

Se ha supuesto que se dispone del gradiente exacto de la función f , pero, ¿y cuándo esto no es así? De hecho, cuando se entrena un modelo en aprendizaje automático, no suele ser el caso. Si sólo se dispone de una muestra de datos (el conjunto de entrenamiento) y ésta no es representativa de la distribución real de los datos del problema modelado, la estimación estará sesgada.

Si se utilizan gradientes con ruido (estimaciones del valor real del gradiente que difieren en un valor ξ , conocido como error o ruido) en los algoritmos de optimización, como por ejemplo el gradiente descendente clásico, no se puede asegurar alcanzar el mínimo local de la función f en un tiempo razonable.

Aunque, a priori, pueda parecer un problema, es lo que justifica el uso del gradiente descendente estocástico. El funcionamiento del gradiente descendente estocástico es muy similar al del gradiente descendente clásico. Mientras que el clásico utiliza el conjunto de entrenamiento completo en cada iteración (aprendizaje por lotes), el estocástico utiliza muestras del conjunto de entrenamiento (aprendizaje por minilotes) o, incluso, una única muestra (aprendizaje online).

Además de reducir el coste computacional calculando una aproximación del valor real del gradiente en vez de estimarlo lo más exactamente posible, también se reduce la eficiencia por

3. Redes Neuronales

iteración al usar minilotes, de hecho, en un factor de $\frac{n}{m}$, donde m es el tamaño de la muestra. Sin embargo, hay que tener en cuenta que cuanto menor sea la muestra escogida para el entrenamiento, mayor será el error cometido en la estimación del gradiente.

En ocasiones, el uso del gradiente descendente estocástico es prácticamente obligatorio, pues el coste computacional del gradiente descendente clásico es prohibitivo. Imagínese un conjunto de entrenamiento con millones de datos. Calcular el gradiente para todos ellos en cada una de las iteraciones es, la mayoría de las veces, inviable.

Dado un $i \in \{1, 2, \dots, n\}$ fijo, escogido de forma aleatoria, el término \mathbf{x}_{t+1} se calcula:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \eta \nabla f_i(\mathbf{x}_t)$$

El gradiente estocástico $\mathbf{g}_t := \nabla f_i(\mathbf{x}_t)$ es un vector aleatorio de dimensión d . La clave es que, aunque el gradiente calculado a partir de una única muestra o un minilote sea una estimación imprecisa respecto al valor verdadero, su valor esperado coincide con éste.

$$\mathbb{E}_i[\mathbf{g}_t] = \mathbb{E}_i[\nabla f_i(\mathbf{x}_t)] = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\mathbf{x}) = \nabla f(\mathbf{x}_t) \quad \forall \mathbf{x}_t \in \mathbb{R}^d$$

3.2.2. Problemas frecuentes

Algunos de los problemas que presentan las redes neuronales han sido mencionados durante el capítulo y se procede ahora a enunciarlos más detalladamente. Nótese, en primer lugar, que, igual que sucedía con algunos de los modelos presentados en el capítulo anterior, las redes neuronales son, en gran medida, cajas negras. Se puede observar tanto las entradas como las salidas. Sin embargo, su funcionamiento interno no puede describirse explícitamente.

Funciones no convexas

Se ha dicho que, tanto en el algoritmo del gradiente descendente clásico como el estocástico, se prefiere que la función f sea convexa. Esto no solo se debe a los resultados vistos en la sección de preliminares, que garantizan que un mínimo local sea también global (lema (1.33)) y que, si f es diferenciable, alcanza un mínimo global (teorema (1.37)), sino también a que existen resultados que aseguran la convergencia del gradiente.

Entonces, ¿qué pasa si la función no es convexa? Éste suele ser el caso de la mayoría de situaciones en las que se utiliza el gradiente descendente en redes neuronales. El problema es que puede que este algoritmo se quede "atrapado" en mínimos locales (definición (1.30)) o en puntos de silla (definición (1.32)). Sin embargo, se ha comprobado empíricamente que los resultados obtenidos por el gradiente descendente, tanto en su versión clásica como en la estocástica, al trabajar con funciones no convexas, suelen ser lo suficientemente buenos.

Desvanecimiento y explosión del gradiente

El problema del gradiente evanescente (*Vanishing Gradient Problem*) ocurre cuando los gradientes se vuelven prácticamente nulos a medida que se propagan hacia atrás a través de las

capas de la red. Esto lleva a que la red sea incapaz de aprender. Como se ha visto, las funciones sigmoidales, que se saturan con valores muy grandes o muy pequeños, contribuyen a este problema.

También puede darse el caso opuesto, que los gradientes sea vuelvan excesivamente grandes. Esto provoca actualizaciones de pesos muy grandes y erráticas, causando inestabilidad en el entrenamiento y haciendo que el algoritmo diverja. Aunque sea menos frecuente que el gradiente evanescente, puede suceder, por ejemplo, si la tasa de aprendizaje escogida es muy grande.

Sobreaprendizaje

El sobreaprendizaje ocurre cuando el modelo se ajusta demasiado bien al conjunto de entrenamiento pero no generaliza bien. Su rendimiento se degrada en los casos realmente interesantes: los distintos a los que hay en el conjunto de entrenamiento. Una señal inequívoca de sobreaprendizaje es que su error sobre el conjunto de entrenamiento es mucho menor que sobre un conjunto de prueba (obviamente distinto al de entrenamiento).

En la parte de desarrollo del proyecto se verá cómo se ha lidiado con este problema y cómo se han dividido los datos en conjunto de entrenamiento y conjunto de prueba para poder realizar un seguimiento del posible sobreaprendizaje de los modelos utilizados.

3.2.3. Regularización

Para mitigar el sobreaprendizaje, se han de utilizar técnicas de *regularización*. Se entiende aquí por regularización a cualquier modificación realizada sobre un algoritmo de aprendizaje con el objetivo de reducir su sobreaprendizaje. Las técnicas de regularización más habituales son:

- Dropout: Es probablemente la técnica más conocida y es una de las que ha sido utilizada en el proyecto. Esta técnica de regularización consiste en desactivar aleatoriamente un porcentaje de neuronas durante cada iteración del entrenamiento, forzando a la red a no depender excesivamente de ninguna neurona en particular.
- Regularización L₁/L₂: Consiste en agregar un término de penalización arbitrario a la función coste. El nombre de estas técnicas de regularización vienen de la nomenclatura de los espacios L_p , donde la norma para dichos espacios se define como:

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}}, \quad p \geq 1$$

De tal forma que, para L_1 y L_2 , las normas correspondientes son:

$$\|x\|_1 = \sum_{i=1}^n |x_i|, \quad \|x\|_2 = \left(\sum_{i=1}^n x_i^2 \right)^{\frac{1}{2}}$$

3. Redes Neuronales

La diferencia entre la regularización L_2 (también conocida como *weight decay*) y L_1 no reside en el signo de los términos añadidos —ambas penalizaciones son siempre no negativas—, sino en el efecto que producen sobre los pesos. La regularización L_2 tiende a reducir los pesos de forma suave, pero raramente los hace exactamente cero, mientras que L_1 favorece que muchos de ellos se anulen por completo, dando lugar a modelos dispersos (*sparse models*).

- *Early Stopping*: La parada temprana consiste en detener el entrenamiento del modelo antes de completar todas las épocas planificadas cuando se detecta que el error deja de mejorar de forma significativa durante un número prolongado de iteraciones sobre un conjunto de validación no utilizado para entrenar los pesos de la red. Una época es un recorrido completo del algoritmo de aprendizaje por todo el conjunto de entrenamiento. En cada época, el modelo procesa todos los datos de entrenamiento (normalmente dividido en lotes (*batches*) y ajusta sus parámetros en función del error calculado.
- Normalización: La normalización de las entradas no se considera una técnica de regularización, ya que su objetivo principal es mejorar la eficacia del descenso por gradiente al homogeneizar la escala de las características. No obstante, en redes profundas también puede influir indirectamente en la estabilidad del entrenamiento. Por otro lado, la normalización interna en la red, conocida como *Batch Normalization* (normalización por lotes), tampoco fue diseñada originalmente como un método de regularización, sino para acelerar la convergencia y estabilizar el aprendizaje; sin embargo, en la práctica ejerce un efecto regularizador y puede mejorar la capacidad de generalización del modelo.

Otra forma de mitigar el sobreajuste es aumentar el número de datos de entrenamiento. Esto proporciona una mayor variedad de ejemplos y mejora la capacidad de generalización del modelo. En la mayoría de casos, empero, no es posible conseguir más datos y el uso de técnicas de regularización como las mencionadas se vuelven indispensables.

3.3. Redes recurrentes

Para esta sección, además de [20], se ha utilizado [36] y [37]. Las redes neuronales recurrentes (RNN) son redes neuronales con memoria. De hecho, para problemas en los que se tratan series temporales, como es el de este proyecto, donde el orden y dependencia temporal de los datos es crucial, suelen funcionar muy bien.

En una red *feed-forward* convencional, cada capa opera de manera independiente y únicamente procesa la información hacia adelante, sin retener ningún contexto previo. En contraste, una red recurrente introduce conexiones que introducen caminos de realimentación en sus salidas, de forma que las salidas de una capa pueden convertirse en entradas para capas anteriores o incluso para sí misma, dotando así al modelo de la capacidad de "recordar".

No obstante, las redes neuronales de tipo *feed-forward* presentan varias limitaciones que las redes neuronales recurrentes son capaces de suplir, concretamente dos.

La primera es la dificultad para tratar secuencias de longitud variable en los datos de entrada. Por ejemplo, al usar una red neuronal para traducir una frase de un idioma a otro,

la frase original (por ejemplo, "hace mucho calor en Granada") y su traducción pueden tener diferente número de palabras. Una red neuronal recurrente sí podría emplearse en este caso. Asimismo, estas redes presentan otra dificultad importante: el manejo de secuencias muy largas, ya que la información lejana en el tiempo puede perderse durante el entrenamiento debido al problema del desvanecimiento del gradiente. En este proyecto, la primera limitación resulta irrelevante, ya que las entradas corresponden a series de precios de ciertos activos cuya longitud es fija³. No obstante, se ha considerado oportuno mencionarla.

La segunda limitación es la no compartición de características. Una red tradicional no comparte las características aprendidas a lo largo de diferentes posiciones de la secuencia. Por ejemplo, si se ha aprendido cierta característica en la primera posición de la secuencia, este aprendizaje no se aplica en el resto de la secuencia.

Las redes neuronales recurrentes consiguen superar estos dos problemas porque comparten los mismos parámetros a lo largo de la secuencia de datos de entrada. Permiten, tanto manejar longitudes de secuencias variables, como aplicar el conocimiento aprendido en una parte del modelo a otras partes.

3.3.1. Estructura de una red neuronal recurrente simple

En primer lugar varios apuntes sobre la notación. Se seguirán utilizando vectores x para mantener la coherencia con la notación utilizada en las redes neuronales tradicionales. Por otro lado, a diferencia de la sección anterior, el instante t se notará en el superíndice $x^{<t>}$, ya que ésta es la notación habitual para las redes neuronales recurrentes.

Para las redes neuronales recurrentes hay diversos tipos de arquitecturas dependiendo de si se el resultado buscado es un único valor o es una secuencia de valores y de si los datos de entrada son un único valor o una secuencia de ellos. En este proyecto lo que se busca es predecir el valor de un activo en el futuro a partir de valores en el pasado, por lo que la arquitectura será una del tipo *many – to – one* (muchos a uno). Se puede intuir cuales son las posibles arquitecturas: uno a uno, uno a muchos, muchos a muchos...). En la figura pueden

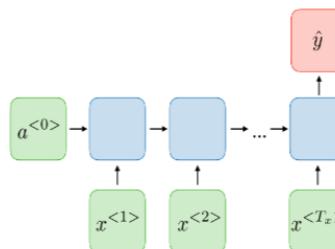


Figura 3.3.: Arquitectura *many – to – one*, donde T_x denota la longitud de la secuencia. [38]

observarse "cuadrados" de diferentes colores: verdes para las entradas y llamados *estados ocultos*, rojo para la salida y azul para las unidades recurrentes, las cuales son encargadas

³El hecho de que el precio pase de 100\$ el día 1 a 1000\$ el día 90 no se considera una "variación" en la longitud de la secuencia.

3. Redes Neuronales

de realizar los cálculos correspondientes en función de las entradas actuales, las entradas anteriores y sus parámetros. Estas unidades recurrentes pueden ser de varios tipos, la más sencilla es la *unidad de red recurrente simple*, que es la que se explicará a continuación. Otro de los tipos posibles se explicará posteriormente.

El estado oculto $\mathbf{a}^{<t>}$ en un instante t depende de la entrada actual $\mathbf{x}^{<t>}$ y del estado oculto anterior $\mathbf{a}^{<t-1>}$. Este valor servirá para calcular la salida en cada instante $\mathbf{y}^{<t>}$. Dado que la arquitectura *many-to-one* cuenta con una única salida, únicamente se calcula la correspondiente al último paso temporal.

$$\mathbf{a}^{<t>} = g_1(W_{aa}\mathbf{a}^{<t-1>} + W_{ax}\mathbf{x}^{<t>} + \mathbf{b}_a)$$

donde W_{aa} es la matriz de pesos que conecta el estado oculto en el instante anterior con el estado oculto actual, y W_{ax} es la matriz de pesos que conecta la entrada actual con el estado oculto. Ambos conjuntos de pesos se comparten a lo largo de todos los pasos temporales de la secuencia. La función g_1 es una función de activación como las presentadas en la Subsección 3.1.1. Normalmente suele escogerse una función ReLU o la tangente hiperbólica.

$$\hat{\mathbf{y}} = g_2(W_{ya}\mathbf{a}^{<t>} + b_y)$$

Donde W_{ya} es otra matriz de pesos, compartida por todos los instantes, y g_2 es otra función de activación que no tiene por qué ser igual a la primera. Para entender mejor la expresión anterior conviene observar cómo se expresa una salida en una arquitectura *many-to-many* (muchos a muchos):

$$\mathbf{y}^{<t>} = g_2(W_{ya}\mathbf{a}^{<t>} + b_y)$$

La "memoria" de la que se hablaba se consigue gracias a que cada estado oculto $\mathbf{a}^{<t>}$ depende en gran medida del anterior $\mathbf{a}^{<t-1>}$, además de la entrada correspondiente. Por último, en el caso en el que no haya información previa, $\mathbf{a}^{<0>}$ se inicializa como un vector nulo.

Cuando las unidades de las redes neuronales recurrentes sean simples, se dirá que la red es una red neuronal recurrente simple.

3.3.2. Problemas de RNN

Pese a las ya explicadas mejoras de las redes neuronales simples respecto a las redes neuronales tradicionales, éstas siguen teniendo problemas. De hecho, los principales problemas de las RNN simples también los tienen las redes tradicionales, pero éstos se agravan enormemente en el momento en el que se introducen conexiones recurrentes. Estos problemas son los del desvanecimiento y la explosión del gradiente. Esto es debido fundamentalmente al uso del algoritmo *back-propagation a través del tiempo*.

Al igual que con las redes neuronales tradicionales, las recurrentes recurrentes simples también tienen una función de coste \mathcal{L} conjunta, definida basándose en el coste en cada instante de tiempo:

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{T_y} \mathcal{L}(\hat{\mathbf{y}}^{<i>}, \mathbf{y}^{<i>})$$

Donde $\hat{\mathbf{y}}$ es la secuencia de salida obtenida por la red, \mathbf{y} es la secuencia de salida real y T_y es

la longitud de la secuencia de salida.

Back-propagation hace referencia al proceso por el cual se van calculando sucesivamente todas las derivadas parciales de la función coste con respecto a los distintos parámetros. Consecuentemente, *back-propagation a través del tiempo* hace referencia al proceso análogo, pero para redes neuronales recurrentes, ya que, al calcular las derivadas, se ha de hacer de forma sucesiva desde los últimos instantes hasta los primeros.

En las redes recurrentes se suele realizar el back-propagation en cada instante de tiempo $T < T_y$, considerando sólo parte de la función de coste de la secuencia entera:

$$\mathcal{L}^{(T)}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^T \mathcal{L}(\hat{\mathbf{y}}^{}, \mathbf{y}^{})$$

Haciendo uso de la regla de la cadena (1.27) a través de la secuencia, y recordando que los parámetros W son compartidos por todos los instantes de la red recurrente, la derivada total de la función de coste $\mathcal{L}^{(T)}$ respecto a cualquier matriz o vector de parámetros W se obtiene sumando las derivadas parciales calculadas en cada instante de tiempo. Así, tenemos:

$$\frac{d\mathcal{L}^{(T)}}{dW} = \sum_{t=1}^T \left. \frac{d\mathcal{L}^{(T)}}{dW} \right|_t$$

Al utilizar la regla de la cadena repetidamente a lo largo de los pasos de tiempo, los gradientes pueden volverse exponencialmente pequeños. Del mismo modo, los gradientes pueden volverse exponencialmente grandes. Para que una RNN pueda "memorizar" de manera correcta y robusta, los parámetros deben situarse en una región del espacio en la que el gradiente ni se desvanezca ni explote, lo cual es francamente complicado.

Además, incluso asumiendo que los gradientes no explotan ni se desvanecen, las dificultades para captar dependencias a largo plazo siguen siendo significativas. Una dependencia a largo plazo es la relación entre elementos de la secuencia que se encuentran en instantes de tiempo bastante alejados. Esta dificultad es debida a que los valores asociados, con el paso del tiempo, a miembros de la secuencia lejanos van a ser exponencialmente pequeños en comparación con los pesos de los cercanos.

A continuación, se presenta una variante de las redes neuronales simples que consigue evitar parcialmente algunos de los problemas con el gradiente y conseguir así memoriar algo mejor las dependencias de largo plazo.

3.3.3. Redes neuronales recurrentes con puertas

Tras revisar con profundidad el funcionamiento de las redes neuronales recurrentes, se explican a continuación las dos redes neuronales recurrentes con puertas (*gated RNN*) más utilizadas: *long short-term memory* y *gated recurrent units*. Este tipo de redes recurrentes se basan en la idea de crear caminos a lo largo del tiempo que tengan gradientes que ni desvanezcan ni explotan. A través de estos caminos, las RNN con puertas son capaces de acumular información de instantes lejanos, incluyendo mecanismos para olvidar la información menos

3. Redes Neuronales

relevante.

LSTM

Las conocidas como unidades LSTM (*Long Short-Term Memory*) fueron presentadas en [39], trabajo que servirá de base para esta sección.

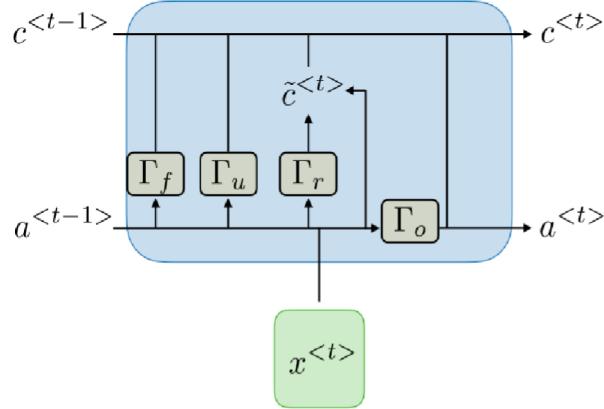


Figura 3.4.: Estructura de una red LSTM. [38].

Se puede observar en la imagen que hay muchos valores que no han sido vistos antes. De hecho, los únicos mencionados han sido la entrada y el estado oculto en el instante t ($x^{<t>}$ y $a^{<t>}$ respectivamente). En las unidades LSTM, se incorpora también una celda de memoria $c^{<t>}$ que actúa como un camino adicional para transportar datos a lo largo del tiempo. Estos caminos adicionales dependen de una serie de puertas que controlan el flujo de información transportada:

- Puerta de relevancia (*relevance gate*, Γ_r):

$$\Gamma_r = \sigma(W_r[a^{<t-1>}, x^{<t>}] + b_r)$$

- Puerta de actualización (*update gate*, Γ_u):

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

- Puerta de olvido (*forget gate*, Γ_f):

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

- Puerta de salida (*output gate*, Γ_o):

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

Se recuerda que σ corresponde con la función logística, vista en [Subsección 3.1.1](#). Ésta es aplicada componente a componente. Las tres puertas son calculadas de forma similar, pero cada una de ellas desempeñará un papel distinto.

En primer lugar, el candidato a la celda de memoria se calcula usando la puerta de relevancia:

$$\tilde{c}^{(t)} = \tanh \left(W_c \left[\Gamma_r \odot a^{(t-1)}, x^{(t)} \right] + b_c \right)$$

donde \tanh denota la tangente hiperbólica (también vista en [Subsección 3.1.1](#)), que se aplica componente a componente. \odot denota el producto componente a componente. La puerta de relevancia indica, como dice su propio nombre, cómo de relevante es la información del estado oculto previo $a^{(t-1)}$.

En segundo lugar, el valor de la celda de memoria en el instante t es calculado mediante la puerta de actualización y la puerta de olvido:

$$c^{(t)} = \Gamma_u \odot \tilde{c}^{(t)} + \Gamma_f \odot c^{(t-1)}$$

Si se observa la expresión, se aprecia papel que desempeñan las puertas de actualización y la de olvido. La de olvido indica cómo de importante es el valor de la memoria en el instante anterior y, la de actualización, cómo de relevante es la información del valor candidato $\tilde{c}^{(t)}$.

Por último, el estado oculto $a^{(t)}$ se calcula haciendo uso de la puerta que queda, la de salida:

$$a^{(t)} = \Gamma_o \odot \tanh(c^{(t)})$$

El estado oculto dependerá del estado oculto anterior y de la celda de memoria actual. La salida del modelo se calcula de forma análoga a la de las RNNs simples, salvo que el estado oculto utilizado es el calculado usando las puertas explicadas.

GRU

Las GRU son una modificación de las redes LSTM en las que se trabaja con un solo estado que actúa de estado oculto y celda de memoria a la vez: $a^{(t)} = c^{(t)}$. De este modo, el anterior esquema presentado en [Figura 3.4](#) queda ahora de la siguiente forma: Las dos puertas que aparecen, ya familiares, se calculan igual que en las LSTM:

- Puerta de relevancia (*relevance gate*, Γ_r):

$$\Gamma_r = \sigma(W_r[a^{(t-1)}, x^{(t)}] + b_r)$$

- Puerta de actualización (*update gate*, Γ_u):

$$\Gamma_u = \sigma(W_u[a^{(t-1)}, x^{(t)}] + b_u)$$

El candidato a celda de memoria también se calcula igual que en LSTM. Sin embargo, el valor de la celda de memoria es cambia, al no tener puerta de olvido:

$$c^{(t)} = \Gamma_u \odot \tilde{c}^{(t)} + (1 - \Gamma_u) \odot c^{(t-1)}$$

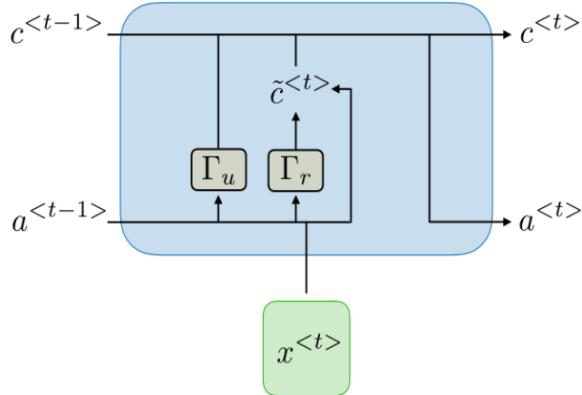


Figura 3.5.: Estructura de una red GRU. [38].

$$c^{<t>} = \Gamma_u \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>}$$

En este caso $(1 - \Gamma_u)$ actúa como valor de puerta de olvido, luego la puerta de actualización tiene más peso en la salida final de GRU. Por último, y como ya se ha comentado antes,

$$a^{<t>} = c^{<t>} ,$$

lo que evita el uso de una puerta de olvido. De esta forma, la implementación de redes GRU es más simple (y computacionalmente más eficiente) que la de redes LSTM.

Obsérvese que, además de estos dos tipos de redes neuronales recurrentes con puertas, existen otras variantes menos conocidas como puede ser la *Minimal Gated Unit*[40].

la capacidad que tienen para conseguir equivarianza

3.4. Redes convolucionales

Las redes neuronales convolucionales, también conocidas como redes neuronales convolutivas, son un tipo de red neuronal en el que todas las neuronas de una capa no están necesariamente conectadas con todas las neuronas en la siguiente capa. Para ello, utiliza de capa específica, las capas convolucionales, en el que sólo un grupo de neuronas que están "cerca" tiene repercusión en la siguiente capa.

Las redes neuronales convolucionales o CNN (*Convolutional Neuronal Network*) son utilizadas frecuentemente cuando se trata con imágenes. Esto es debido al alto número de neuronas que se utilizarían utilizando las capas tradicionales. Poniendo un ejemplo, con una imagen de 128×128 RGB⁴ se utilizaría una capa de entrada con $128 \cdot 128 \cdot 3 = 49152$ neuronas. Una imagen de ese tamaño, en los tiempos que corren, tiene una calidad realmente pobre.

Aunque la verdadera fortaleza de las CNN reside en su capacidad para explotar la estruc-

⁴RGB es un modelo de color que combina los colores rojo (Red), green (Verde), azul (Blue) para crear una amplia gama de colores. Se utiliza principalmente para representar imágenes digitales.

tura espacial de las imágenes mediante el uso de filtros compartidos, lo que proporciona *equivarianza traslacional*: una vez que la red aprende a detectar un patrón en una región de la imagen, puede reconocerlo automáticamente en cualquier otra posición. Gracias a esta propiedad, las CNN son especialmente adecuadas para tareas como la detección de bordes, formas u objetos en imágenes, incluso cuando estos aparecen en distintas ubicaciones.

Sin embargo, el uso de redes convolucionales no sólo se limita a imágenes (o vídeos, que esencialmente son secuencias temporales de imágenes), sino que también puede aplicarse a series temporales.

Para comprender qué es una red neuronal convolucional, es necesario entender qué es una capa convolucional y la esencia de la misma, la operación de la convolución.

3.4.1. Convolución

La convolución es una operación binaria (es decir, toma dos operandos, en este caso funciones) realizada sobre dos funciones para producir una tercera función que suele interpretarse como una versión modificada de una de las funciones originales. Formalmente la convolución se define como:

$$\star : \mathcal{F} \times \mathcal{F} \longrightarrow \mathcal{F},$$

donde \mathcal{F} es el espacio de funciones en el cual se trabaje. Para valores reales, la convolución de dos funciones f y g es dada por:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau = \int_{-\infty}^{\infty} f(t - \tau)g(\tau) d\tau$$

Para el caso del procesamiento digital de señales o la predicción de valores en de series temporales, la operación de convolución es discreta. Esto implica que la integral pasa a ser sumatoria:

$$(f \star g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n - m] = \sum_{m=-\infty}^{\infty} f[n - m] g[m]$$

Lo habitual es que uno de los operandos sea la señal que se desea procesar, notado por $x[n]$, y el otro el filtro (también llamado máscara) con el que se procesa la señal, notado por $h[n]$. Si el filtro es de longitud finita y está definido únicamente en el dominio $\{0, 1, \dots, K - 1\}$, la operación de convolución, para cada posición de la señal, implica efectuar K multiplicaciones y $K - 1$ sumas:

$$(x \star h)[n] = \sum_{k=0}^{K-1} h[k] x[n - k]$$

Las convoluciones y los correspondientes filtros también pueden extenderse para el caso multidimensional. Para el caso, por ejemplo, de procesamiento de imágenes digitales, se aplicaría un filtro bidimensional (una matriz de pesos) $K_1 \times K_2$ para una convolución bidimensional.

$$(x \star h)[n_1, n_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} h[k_1, k_2] x[n_1 - k_1, n_2 - k_2]$$

3. Redes Neuronales

Donde $[n_1, n_2]$ corresponderían con las coordenadas $[x, y]$ de los píxeles de una imagen. Nótese la posible confusión con la notación. La primera x corresponde con la señal que se quiere procesar, la segunda corresponde con la primera coordenada del respectivo píxel de la imagen.

3.4.2. Capa convolucional

El elemento fundamental de las redes convolutivas es el uso de la operación de convolución, que reemplaza a la clásica multiplicación de pesos por las entradas. En esencia, una capa convolutoria aplica métodos específicos que permiten integrar conocimiento previo sobre el problema directamente en la arquitectura de la red neuronal.

La señal que recibe la red es procesada por una convolución que usa un filtro que no está previamente establecido, sino que se aprende. Este filtro es llamado *kernel*. Consecuentemente, los pesos correspondientes a la máscara de convolución serán los parámetros de la capa convolutoria que se quiere entrenar.

A diferencia de las redes *feed-forward*, donde cada neurona está conectada con todas las neuronas de la capa anterior, en una capa convolutoria, cada neurona sólo está conectada a un número concreto de neuronas de la capa anterior. En una imagen, por ejemplo, en vez de usar todas las neuronas de la capa de entrada (recuérdese el ejemplo de la imagen 128×128) se tomaría únicamente una ventana arbitraria de $A \times B$ píxeles. Esta conectividad local, siguiendo el ejemplo de las imágenes, tiene sentido, pues aprovecha que los píxeles más cercanos están más relacionados que los lejanos.

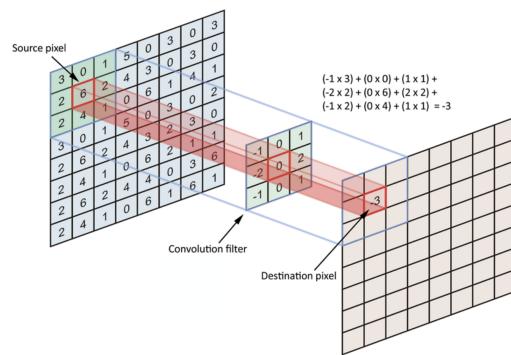


Figura 3.6.: Operación de convolución para el caso bidimensional. [Medium](#)

Por otro lado, el kernel es compartido por varias neuronas. Esto consigue reducir considerablemente el número de parámetros (pesos) que deben ajustarse, facilitando el entrenamiento de la red neuronal. Ésto es conocido como *weight sharing*.

Cada kernel en una capa convolutoria tiene como objetivo aprender a detectar una característica concreta (en el caso de imágenes bordes, texturas o patrones). La salida de la capa convolutoria para cada kernel se denomina *mapa de características (feature map)*. El número de mapas de características vendrá determinando por el número de kernels que tiene la capa convolutoria, de tal manera que una capa convolutoria con K kernels producirá K mapas

de características.

Además del número de filtros K y del tamaño de estos F , al diseñar una capa convolucional han de ajustarse dos hiperparámetros más:

- **Paso, salto, zancada o *stride*:** Este hiperparámetro, notado como S , indica cada cuántas muestras de la entrada se obtiene una salida en la operación de convolución. Esto permite reducir el coste computacional pues se reduce tanto el número de operaciones aritméticas que se ha de realizar, como el tamaño de la salida obtenida. Sin embargo, esto trae consigo que se reduzca también el grado de detalle con el que se extraen las características de la entrada.
- **Rellenado con ceros o *zero padding*:** La cantidad de relleno, notada como Z , permite conseguir salidas del tamaño que se quiera llenando la entrada de ceros a su alrededor. Normalmente, se utiliza para preservar las dimensiones de la entrada en la capa de salida.

Pese a todo este despliegue de hiperparámetros y configuraciones, el coste computacional de las capas convolutivas es realmente significativo. La única forma de reducirlo es, *a priori*, utilizando un valor para la zancada mayor que 1. Sin embargo, la reducción que supone este pequeño cambio no suele ser suficiente. Es aquí donde entran en juego las capas de submuestreo, también llamadas capas de *pooling*.

3.4.3. Capa de pooling

Este tipo de capas es utilizadas frecuentemente en las CNN para reducir, básicamente, el tamaño de su entrada, de forma que las capas posteriores se beneficien al trabajar con datos de menos dimensionalidad.

La versión tradicional de una capa pooling se limita a combinar un conjunto de valores de entrada adyacentes utilizando una función máximo (*max pooling*) o el promedio (*average pooling*), siendo el primero de los mencionados el más utilizado.

A modo de ejemplo, para reducir la dimensionalidad de una imagen, un max pooling 2×2 divide la imagen en bloques 2×2 y conserva sólo el valor máximo de cada bloque:

$$\begin{bmatrix} 3 & 12 & 17 & 5 \\ 10 & 8 & 7 & 15 \\ 6 & 14 & 9 & 11 \\ 18 & 4 & 2 & 16 \end{bmatrix} \Rightarrow \begin{bmatrix} 12 & 17 \\ 18 & 16 \end{bmatrix}$$

Sin embargo, surge un problema evidente: al aplicar varios niveles de pooling, se pierde la información sobre la posición exacta de los elementos presentes en la capa anterior. No obstante, este problema tiene una cara "positiva": las redes convolucionales que usan capas pooling se vuelven más robustas frente a pequeñas traslaciones. Esto es debido a que las respuestas de las capas posteriores tienden a ser aproximadamente invariantes a las traslaciones, es decir, que cambios pequeños en la posición de los valores de entrada apenas alteran la salida de la red. Esta propiedad ha demostrado ser muy positiva en la práctica, especialmente en problemas donde importa más la detección un patrón que su localización.

3. Redes Neuronales

Por otro lado, las capas pooling, a diferencia de las convolucionales, no tienen parámetros entrenables. En su lugar, la configuración de estas capas viene dada mediante el uso de dos hiperparámetros (sí, más hiperparámetros):

- Ventana o *window*: Como su propio nombre indica, este hiperparámetro regula el tamaño de la entrada sobre la que se va a aplicar la operación correspondiente (el máximo en el caso de max pooling o el promedio en el caso de average pooling). Remitiéndose al ejemplo anterior, la ventana usada ha sido de tamaño 2×2 .
- Paso, salto, zancada o *stride*: Al igual que en las capas convolucionales, se puede especificar el tamaño S del salto dado sobre la entrada. Lo normal es utilizar un tamaño de paso igual al de la ventana. Remitiéndose nuevamente al ejemplo anterior, el tamaño de paso usado ha sido 2.

Las redes neuronales convolucionales suelen alternar capas convolucionales, para extraer características, capas de pooling ,para la reducción de dimensionalidad. Tras ellas, una o más capas completamente conectadas para hacer la regresión final (o clasificación).

3.5. Transformers

Los *transformers* son el tipo de red neuronal más popular en la actualidad, presentes en la mayoría de chatbots (tales como ChatGPT, Gemini o Grok) que han irrumpido con fuerza en la vida de millones de personas. Fueron introducidos por primera vez en el influyente artículo: "Attention is All You Need"[\[41\]](#), el cual servirá de base para esta sección.

Constituyen una arquitectura de red neuronal basada exclusivamente en mecanismos de atención, eliminando por completo la recurrencia y la convolución.

Los mecanismos de atención permiten superar las limitaciones de memoria de las RNN simples cuando se trabajaba con secuencias largas. Además, los transformers suplen otra limitación clara de las redes neuronales recurrentes (no sólo las simples): la secuencialidad. Los transformers tienen la capacidad de operar en paralelo, al igual que las redes convolucionales, consiguiendo unos tiempos de entrenamiento menores a las RNN/LSTM.

Aunque nacieran en el contexto del Procesamiento del Lenguaje Natural (PLN), los transformers han demostrado su potencial en otros ámbitos como la visión por computador, el audio, o las series temporales. Si bien es cierto que para el caso de las series temporales hay numerosas variaciones de los transformers creadas con el objetivo de conseguir mejores resultados, un estudio reciente afirma que la versión tradicional de los transformers consigue, en la práctica, mejores resultados con series temporales [\[42\]](#). En este proyecto, se ha utilizado la versión clásica de los transformers, la cual se explicará a continuación. Además de los 2 artículos ya citados, se usará también [\[43\]](#).

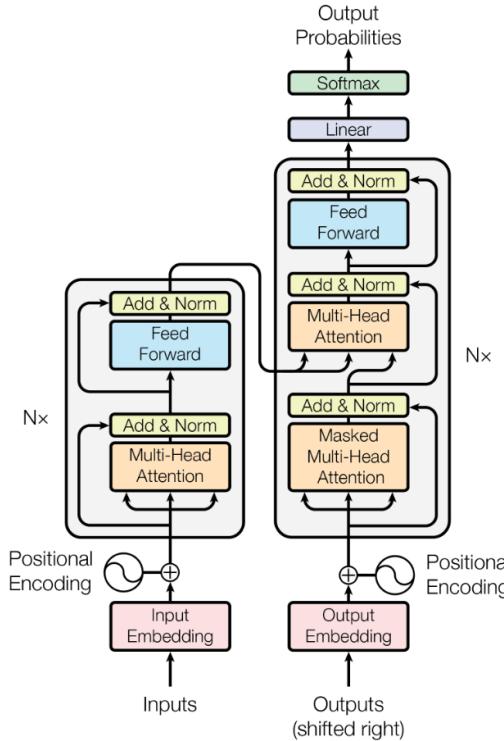


Figura 3.7.: Arquitectura original de un transformer. [28]

3.5.1. Estructura encoder-decoder

Dado un conjunto de valores de entrada $\{x^{<1>}, \dots, x^{<T>}\}$ (se mantiene la notación de Sección 3.3) el *encoder* (codificador) genera una secuencia de igual longitud $\{z^{<1>}, \dots, z^{<T>}\}$ que servirá de entrada para el *decoder* (decodificador). Este último es el encargado de generar la salida final del modelo.

Observando la Figura 3.7, se distinguen nítidamente dos bloques: a la izquierda el encoder, y a la derecha el decoder. Por otro lado, un *embedding* (los bloques rosas iniciales) es la representación numérica de la secuencia original con la que trabaja realmente el transformer, la dimensión del embedding resultante se notará como d_{model} .

El *encoder* (o codificador) está formado por $N = 6$ capas idénticas. Es importante señalar que la figura no muestra explícitamente las seis capas, sino que utiliza la anotación “ $N\times$ ” junto a cada bloque para indicar que dicho bloque se repite N veces. En el trabajo original, el valor utilizado es $N = 6$ tanto para el codificador como para el decodificador. Cada capa cuenta con dos subcapas. La primera de ellas es un mecanismo de auto-atención múltiple (llamado comúnmente *multi-head self-attention*) y una capa feed-forward ligeramente modificada para aplicarse de manera *position-wise*, es decir, de forma independiente a cada posición de la secuencia pero compartiendo los mismos parámetros en todas ellas. Normalmente,

3. Redes Neuronales

consta de dos transformaciones lineales separadas por una activación ReLU:

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2,$$

donde W_1, b_1, W_2, b_2 son valores fijos para todos los elementos de la secuencia (también llamados comúnmente *tokens* en este contexto).

A cada salida de las subcapas se le aplica una *conexión residual*. Consiste en sumar la entrada original de la subcapa con su salida, seguida de una normalización por capas (*layer normalization*). Formalmente:

$$\text{LayerNorm}(\mathbf{x} + \text{Sublayer}(\mathbf{x})),$$

donde x es la entrada a la subcapa y $\text{Sublayer}(x)$ es la función implementada por dicha subcapa (ya sea la red feed-forward o el mecanismo de auto-atención).

El decoder (decodificador) se compone igualmente de $N = 6$ capas idénticas. La estructura es similar a la del encoder, salvo por algunos matices. Se incluye una subcapa extra para procesar la salida del encoder con un mecanismo multi-head self-attention. Además, se modifica la subcapa inicial de auto-atención para conseguir que las predicciones para la posición t sólo puedan depender de las salidas conocidas en posiciones anteriores a t .

3.5.2. Codificación posicional

Dado que el modelo no tiene ni recurrencias ni convoluciones, ha de existir un método para mantener la información acerca del lugar de cada elemento en la secuencia. Esto se consigue mediante el uso de codificadores posicionales (*positional encodings*). Los codificadores posicionales aportan la información necesaria a los embeddings de forma que el modelo pueda distinguir correctamente la secuencia ordenada de una ordenada aleatoriamente.

En la arquitectura propuesta en el paper original, las codificaciones posicionales son deterministas y fijas:

$$PE(t)_i = \begin{cases} \sin(\omega_i t), & i \bmod 2 = 0 \\ \cos(\omega_i t), & i \bmod 2 = 1 \end{cases}$$

donde cada ω_i es la frecuencia predefinida para cada dimensión. En la práctica, cada PE se suma a los embeddings originales de la secuencia, de modo que cada vector de entrada \mathbf{x}_t queda enriquecido con información sobre si corresponde al inicio, mitad o final de la secuencia. El modelo aprende así a discriminar posiciones relativas porque cualquier desplazamiento k entre dos posiciones puede inferirse a partir de sus codificaciones posicionales.

Posteriormente, han aparecido nuevas codificaciones posicionales más dinámicas y adaptativas. Un ejemplo son las codificaciones posicionales aprendibles, que optimizan los vectores de posición junto con el resto de parámetros del modelo. Otro ejemplo es la codificación de marcas temporales (*timestep encoding*), que incorpora información temporal explícita (fecha u hora) mediante capas de embeddings que se ajustan durante el entrenamiento.

Se pasa ahora a explicar más en detalle lo que hace tan novedoso⁵ y potente a los transfor-

⁵En realidad, los mecanismos de atención ya existía, lo novedoso fue descubrir que se podía prescindir completa-

mers: los mecanismos de atención.

3.5.3. Mecanismos de atención

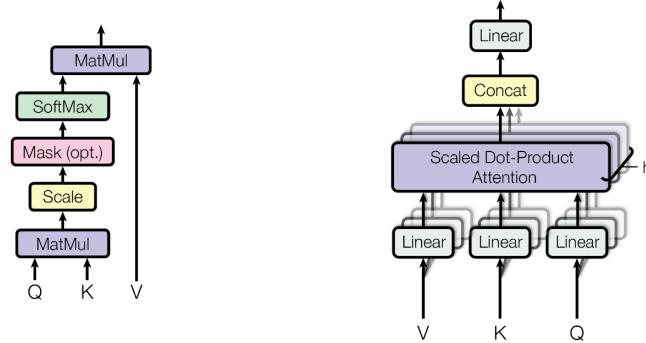


Figura 3.8.: Izquierda: Atención por producto escalar escalado. Derecha: Multi-head attention consistente de múltiples capas de atención en paralelo. [28].

La atención, más concretamente, la auto-atención, permite que una secuencia de entrada se represente a sí misma "prestando atención" a sus diferentes posiciones, ponderando la importancia relativa de unos elementos de la secuencia sobre otros.

Una función de atención puede describirse como el mapeo de una consulta y un conjunto de pares clave-valor a una salida, donde la consulta (*query*), las claves (*keys*) y los valores (*values*) son vectores.

Concretamente, cada vector de la entrada $x^{<t>}$, es proyectado linealmente en estos tres vectores: una consulta $q^{<t>} = W^Q x^{<t>}$, una clave $k^{<t>} = W^K x^{<t>}$ y un valor $v^{<t>} = W^V x^{<t>}$. Las proyecciones W^Q, W^K, W^V son matrices entrenables de dimensiones $d_{\text{model}} \times d_k$, $d_{\text{model}} \times d_k$ y $d_{\text{model}} \times d_v$ respectivamente, donde d_k y d_v corresponden con las dimensiones escogidas para las consultas y claves, y valores.

Para cada posición t , se calcula un peso de atención respecto a cada posición j de la secuencia evaluando la relación entre la consulta $q^{<t>}$ y la clave $k^{<j>}$

Atención por producto escalar escalado

La entrada de un mecanismo de "atención por producto escalar escalado" (*Scaled Dot-Product Attention*) consta de consultas y claves de dimensión d_k , y valores de dimensión d_v .

La idea es calcular el producto escalar de todas las consultas con todas las claves, dividir cada uno entre $\sqrt{d_k}$ y aplicarle una función *softmax* para obtener los pesos de los valores.

Una función softmax es, en suma, una función que transforma un vector numérico en un vector de probabilidades. Asegura que los pesos asignados a cada valor formen una

mente de las recurrencias.

3. Redes Neuronales

distribución de probabilidad en el sentido de [Teorema 1.1](#)(toman valores entre 0 y 1 y la suma de todos es 1).

En la práctica, la función de atención se calcula de forma simultánea sobre un conjunto de consultas de tamaño T , agrupadas en una matriz Q de tamaño $T \times d_k$. El conjunto de claves y valores también se agrupan en matrices K y V , de tamaño $T \times d_k$ y $T \times d_v$ respectivamente. El valor resultante se calcula como:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V$$

Intuitivamente, si la consulta $q^{<t>}$ tiene un alto producto escalar con la clave $k^{<j>}$ (comparado con las demás), la atención otorga un peso mayor al valor $v^{<j>}$ correspondiente en la combinación, indicando que la posición j de la secuencia es muy relevante para construir la representación de la posición t . En cambio, si dos vectores son ortogonales (producto escalar cercano a 0), la contribución será despreciable. La división por $\sqrt{d_k}$ es un factor de normalización que evita valores excesivamente grandes en el softmax cuando d_k es alto, estabilizando así el entrenamiento.

Atención multicapa

En lugar de utilizar una única función de atención (una única "cabeza"), el transformer utiliza múltiples mecanismos de atención ejecutados en paralelo (múltiples "cabezas"), cada uno con sus propias proyecciones de dimensiones d_k y d_v , para posteriormente combinar los resultados. Esto se conoce como atención multi-cabeza (*multi-head attention*).

Denotando por h al número de cabezas, para cada cabeza i se tienen matrices W_i^Q, W_i^K, W_i^V independientes. Cada cabeza i produce una salida de atención:

$$\text{head}_i = \text{Attention} \left(QW_i^Q, KW_i^K, VW_i^V \right),$$

a continuación las h salidas se concatenan y se proyectan por medio de otra matriz entrenable W^O para obtener el resultado final de la capa:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O.$$

Al tener múltiples mecanismos de atención concatenados, cada cabeza puede especializarse en "prestar atención" a distintos patrones o aspectos de la secuencia.

¿Cómo se utiliza esto en el modelo?

En las capas de atención del encoder–decoder (en la [Figura 3.7](#), la capa multi-head attention que recibe como entradas las salidas del encoder y las de la capa previa del decoder), las consultas provienen de las capas previas del decoder, mientras que las claves y los valores provienen de la salida del encoder. Esto permite que todas las posiciones del decoder atiendan a todas las posiciones de la secuencia de entrada.

Por otro lado, el encoder contiene capas de self-attention, en las que las consultas, claves y valores son las salidas de la capa anterior. En este caso, cada posición del encoder atiende a todas las posiciones de la capa anterior del propio encoder. Lo mismo sucede con las capas self-attention del decoder.

Para problemas de clasificación, como el del artículo original, tras las $N = 6$ capas del decoder se aplicaría una capa lineal y una softmax. Sin embargo, para problemas de regresión, el uso de la función softmax no sería necesario, por lo que se usaría una única capa lineal.

En el contexto de las series temporales financieras, los mecanismos de atención permiten que el modelo pueda centrarse en analizar las fluctuaciones a largo plazo o a corto, o en los valores anómalos de la secuencia dada. Al combinar varios mecanismos de atención, los transformers pueden centrarse en todos estos factores a la vez.

Tras la explicación de los modelos basados en redes neuronales utilizados en el proyecto, se concluye la parte teórica del proyecto a falta de un breve capítulo en el que se definirán las métricas utilizadas para comparar las predicciones de cada modelo en la práctica, así como una descripción teórica de la metodología utilizada para el entrenamiento y elección de los distintos hiperparámetros ya explicados.

4. Métricas y metodología

Una vez desarrollada la parte más teórica de la memoria, se definirán en este capítulo las métricas utilizadas en la realización de este proyecto para comparar los resultados de los modelos, así como la metodología usada para la validación de los resultados experimentales.

4.1. Métricas estadísticas

Dentro del ámbito puramente estadístico, algunas de ellas ya han sido mencionadas en la memoria, como por ejemplo el error cuadrático medio, que ha servido de función de coste para las redes neuronales.

MSE y RMSE

El error cuadrático medio (MSE, *Mean Squared Error*) es la medida de error más utilizada para estimar la calidad de un modelo de predicción cuantitativa:

$$MSE = \frac{1}{N} \sum_{n=1}^N (\varphi(x_n) - y_n)^2$$

Aplicando la raíz al MSE se obtiene la medida del error denominada RMSE (*Root Mean Squared Error*):

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N} \sum_{n=1}^N (\varphi(x_n) - y_n)^2}$$

RMSE representa la desviación de las diferencias entre los valores de las predicciones $\varphi(x)$ y los valores observados y .

MAE y MAPE

El problema de estas métricas es que usan los errores al cuadrado. Esto provoca que sean sensibles a la presencia de anomalías en los datos (también llamadas *outliers*). Los errores muy grandes tienen efectos mucho más grandes, concretamente cuadráticos, sobre las medidas de error como RMSE y MSE. Es por esto que se suelen utilizar otras métricas como el error absoluto medio (MAE, *Mean Squared Error*), definido como:

$$MAE = \frac{1}{N} \sum_{n=1}^N |\varphi(x_n) - y_n|$$

4. Métricas y metodología

El MAE es ,sencillamente, la media de las diferencias, en valor absoluto, entre las predicciones $\varphi(x)$ y las observaciones y .

Igual que con el error cuadrático medio, también existen versiones normalizadas del MAE. La más popular de ellas quizás sea el error absoluto medio porcentual (MAPE, *Mean Absolute Percentage Error*), definido como:

$$MAPE = \frac{100}{N} \sum_{n=1}^N \frac{|\varphi(x_n) - y_n|}{|y_n|}$$

El MAPE expresa el error medio de estimación como porcentaje respecto al valor observado de y . Esta medida presenta dos problemas: no puede calcularse cuando $y = 0$ y es asimétrico, ya que el error por subestimación está limitado al 100 %, mientras que el error por sobreestimación no tiene límite. Esto provoca un sesgo a favor de modelos que tienden a subestimar el valor real.

4.2. Métricas bursátiles

Tras este breve repaso de las métricas estadísticas, se presentan las quizás menos conocidas –por lo menos en un ámbito matemático-informático como es el de este proyecto– métricas bursátiles utilizadas para medir la calidad de una cartera o estrategia. Para esta parte se han utilizado [44] y [45].

Rentabilidad

En primer lugar, nótese que la rentabilidad de un activo o cartera en el período t se define como una variable aleatoria real R_t , que viene dada como:

$$1 + R_t = \frac{V_{final}}{V_{initial}}$$

Esta definición varía ligeramente si hay dividendos o si se realizan aportaciones extra dentro del período.

Volatilidad

La volatilidad de un retorno R_t es su desviación típica, es decir:

$$\sigma = \sqrt{Var(R_t)}$$

Bajo independencia y varianza estable, la varianza de un horizonte de T períodos es T veces la varianza de un período y, por tanto, la desviación típica escala como \sqrt{T} . Consecuentemente, la volatilidad anual se puede obtener mediante:

$$\sigma_{anual} = \sqrt{D}\sigma_{diario}$$

Donde D denota el número de días hábiles del año (en la práctica se usa $D = 252$).

Hay mucha gente que piensa que el riesgo y la volatilidad son lo mismo, esto es lo que dicen Warren Buffet, considerado mejor inversor de la historia, y su mano derecha, Charlie Munger, sobre la volatilidad y el riesgo [46]:

Warren: Usar la volatilidad para medir el riesgo no tiene sentido. Se usa porque los profesores universitarios quieren medir el riesgo y no saben cómo, así que dicen que la volatilidad mide el riesgo. Para nosotros el riesgo es la probabilidad de pérdida permanente de capital, no tiene nada que ver con la volatilidad. La probabilidad de pérdida permanente de capital viene determinada por el riesgo del negocio que a su vez viene determinado por excesiva deuda (si hay un bache, los bancos te anulan los créditos), la naturaleza del negocio (algunos negocios son de mayor riesgo) o de no ser el productor de bajo coste (el productor de bajo coste puede vender más barato y llevarte tu cuota de mercado). Más allá de eso, el único riesgo es que pagues demasiado, aunque ese problema es más de tiempo que de pérdida de dinero. Entonces el riesgo se convierte en el riesgo de uno mismo, es decir, si vas a ser capaz de mantener la confianza en el negocio y no vender en las bajadas. La volatilidad nos da igual, ganamos más cuando hay mucha volatilidad, ya que hay más oportunidades. Nosotros no renunciamos a rentabilidad a cambio de menor volatilidad en los resultados. Si nos dan a elegir, preferimos un 15 % con sustos a un 12 % tranquilo. Los académicos miden el riesgo usando la volatilidad histórica de la acción respecto a la del índice, al resultado lo llaman beta. Bajo este modelo, una acción tiene más riesgo después de bajadas importantes (sin importar el negocio que hay detrás). Es absurdo. Si alguien te habla de la beta, sal corriendo. A un purista de la beta le da igual lo que produce la compañía, sus competidores o la deuda que tiene. Lo único que quiere saber para medir el riesgo es el histórico de precios de la acción. Para nosotros es al revés, nos da igual el histórico de precios y nos centramos en entender el negocio, predecir su evolución a largo plazo, evaluar a la directiva y ver si lo podemos comprar con un descuento significativo con respecto a su valor intrínseco. El problema es que estos factores no se pueden cuantificar con fórmulas matemáticas y, por eso, a muchos académicos no les gustan, pero que no se puedan cuantificar no quiere decir que no sean importantes para invertir con éxito.

Charlie: Nos parece inútil este gran énfasis que se da a la volatilidad. Si las probabilidades están a nuestro favor, no nos importa la volatilidad.

Ratio de Sharpe

El ratio de Sharpe fue desarrollado originalmente por William F. Sharpe en 1966, quien lo denominó inicialmente *reward-to-variability ratio* o ratio recompensa-variabilidad.

Sea b_t la tasa base prefijada para el período t . Se define el "exceso" de rentabilidad como $X_t := R_t - b_t$. El ratio de Sharpe mide el exceso de rendimiento por unidad de riesgo total

4. Métricas y metodología

asumido. Matemáticamente se define como:

$$\frac{\mathbb{E}[X_t]}{\sqrt{Var(X_t)}}$$

Una limitación importante del ratio de Sharpe es que considera toda la volatilidad como riesgo, sin distinguir entre desviaciones positivas (rentabilidades superiores a la base) y negativas (rentabilidades inferiores). Sin embargo, para la mayoría de los inversores, únicamente las segundas resultan indeseables.

Ratio de Sortino

El ratio de Sortino sólo considera los excesos de rentabilidad a la baja como penalización. De esta forma, el ratio de Sortino, desarrollado por Frank A. Sortino en 1991, se define como:

$$\frac{\mathbb{E}[X_t]}{\sqrt{\mathbb{E}[\min(X_t, 0)]}}$$

El denominador se corresponde con la llamada *downside deviation* (desviación a la baja), que sólo tiene en cuenta las desviaciones negativas de los rendimientos frente a la tasa base b_t .

Obsérvese que, en el caso "perfecto" en el que nunca haya retornos por debajo de la tasa base, la desviación a la baja sería 0. En este escenario, el ratio de Sortino tiende a infinito, lo cual tiene sentido: la inversión ofrece rentabilidad sin riesgo a la baja.

4.3. Validación de modelos

Al igual que en secciones y capítulos anteriores, se utilizará [20] como base para esta parte de la memoria.

Tras observar las métricas que se usarán para evaluar las predicciones generadas por el modelo se ha de explicar una parte fundamental en el proceso de construcción de modelos predictivos: la validación. La validación consiste en comprobar la capacidad de generalización del modelo en datos no vistos, permitiendo evitar el conocido sobreajuste y observar cuáles son los mejores modelos entre varias alternativas.

Dado un conjunto de datos, se ha de poder utilizar dicho conjunto tanto como para entrenar el modelo como para estimar la calidad del modelo. Para ello, este conjunto de datos se divide entre "conjunto de entrenamiento", el cuál será el utilizado para entrenar el modelo, y "conjunto de prueba", el cuál será sobre el que se mida la calidad del modelo.

La metodología a seguir es: dividir el conjunto de datos entre entrenamiento y prueba, construir el modelo a partir del conjunto de entrenamiento y usar dicho modelo sobre los datos del conjunto de prueba, que hasta ese momento nunca los habrá visto. Es muy importante que la evaluación se haga sobre el conjunto de prueba y no sobre el conjunto de entrenamiento, pues en otro caso las métricas no tienen validez.

Para evaluar la calidad de un modelo se ha de recurrir a un conjunto de prueba independiente del conjunto de entrenamiento utilizado en la construcción. La cuestión que puede surgir ahora es: ¿cómo dividir los datos entre conjunto de entrenamiento y conjunto de prueba?

En este proyecto se ha utilizado una variación de la conocida "validación cruzada". Esta metodología consiste en crear y evaluar múltiples modelos de forma sistemática, cada uno de ellos sobre diferentes subconjuntos del conjunto de datos disponible. Concretamente, la validación cruzada de k iteraciones, conocida como k -CV (*k-fold Cross-Validation*), se realiza de la siguiente manera:

- Primero, se divide aleatoriamente el conjunto de datos en k subconjuntos disjuntos del mismo tamaño.
- Seguidamente, para cada uno de los k conjuntos se construye un modelo utilizando dicho subconjunto como conjunto de prueba y empleando los $k - 1$ para entrenarlo.

Validación en series temporales

En series temporales el orden cronológico de las observaciones es crucial y no puede romperse aleatoriamente. Consecuentemente, la validación descrita anteriormente no puede ser utilizada ya que ha de respetar la secuencia temporal. No tiene sentido "predecir el pasado" utilizando observaciones del futuro.

El método apropiado para este caso es la validación *walk-forward*, también conocida como *walk-forward cross-validation*. Consiste en entrenar el modelo con una ventana de datos históricos y validar en el siguiente bloque temporal. Después, la ventana se desplaza hacia adelante en el tiempo y se repite el proceso. El procedimiento es el siguiente:

- En primer lugar, se divide el conjunto de entrenamiento en diversas ventanas temporales de la misma longitud.
- A continuación, se entrena con los datos $[t_0, t_k]$ y se valida en $[t_{(k+1)}, t_{(k+h)}]$. Luego se hace lo mismo con la siguiente ventana y así sucesivamente hasta agotar la serie.

Esta metodología refleja mucho mejor el escenario real de las series temporales, donde se utiliza el pasado para predecir el futuro.

4.4. Selección de hiperparámetros

Se ha visto la gran cantidad de parámetros que hay, tanto para los modelos de aprendizaje automático más simples como para las redes neuronales más sofisticadas. La pregunta que puede surgir llegado este punto es: ¿cuál es la combinación de hiperparámetros que da mejores resultados? La respuesta: no se sabe.

Todos los modelos vistos (obviando los basados en regresiones lineales y/o exponenciales) actúan como cajas negras. Por ello, resulta difícil estimar si el uso de un valor concreto para un hiperparámetro dará mejores resultados que la elección de otro valor. Esto sucede para un

4. Métricas y metodología

único hiperparámetro. Se intuye que cuando existen múltiples hiperparámetros, se produce una explosión combinatoria.

En este proyecto se ha abordado el problema usando dos metodologías: una simple y otra un poco más compleja. La primera consiste, simplemente, en realizar experimentos con hiperparámetros seleccionados al azar y analizar algunas de las métricas estadísticas explicadas antes para seleccionar la mejor combinación. La segunda consiste en el uso de optimización bayesiana usando procesos gaussianos, la cual será explicada a continuación con la ayuda de [47] [48] y [49].

4.4.1. Optimización bayesiana

La optimización bayesiana es un enfoque que suele ser aplicado en problemas con carácter de "caja negra", como es el caso de la búsqueda de hiperparámetros de redes neuronales. El objetivo es buscar los hiperparámetros que optimicen una función objetivo $f(\mathbf{x})$ (p.ej el MSE). La optimización bayesiana construye un modelo probabilístico "sustituto" de la función objetivo y utiliza dicho modelo para guiar de manera inteligente la búsqueda de hiperparámetros. Esto permite obtener resultados de mayor calidad realizando un número significativamente menor de evaluaciones que al emplear combinaciones aleatorias de hiperparámetros.

La idea principal es tratar la función objetivo desconocida $f(\mathbf{x})$ como una realización de un proceso estocástico, incorporando en cada paso toda la información recopilada de evaluaciones anteriores. A diferencia de métodos de optimización locales (que utilizan gradientes o aproximaciones locales de la función), la optimización bayesiana aprovecha todas las evaluaciones previas para inferir globalmente dónde podría hallarse el óptimo. Por ello es conveniente, como se ha hecho en este proyecto, hacer algunas pruebas aleatorias con las que inicializar el modelo sustituto que guiará el proceso de búsqueda.

En cada iteración, se ajusta o actualiza un modelo probabilístico de $f(\mathbf{x})$ a partir de los datos observados hasta el momento. A continuación, una función de utilidad o adquisición deriva de este modelo probabilístico un criterio que permite escoger el siguiente punto \mathbf{x} que se evaluará. Esto equilibra el proceso de exploración (examinar regiones aún no evaluadas del espacio de búsqueda) y explotación (refinar lo ya evaluado que sea considerado como prometedor). Este proceso iterativo continúa hasta alcanzar un número predefinido de evaluaciones o hasta converger hacia una solución que se considere suficientemente buena.

Procesos gaussianos

La optimización gaussiana usualmente utiliza un proceso gaussiano (GP, *Gaussian Process*) como modelo sustituto de la función objetivo $f(\mathbf{x})$. Formalmente, la definición de un proceso gaussiano es la siguiente:

Definición 4.1. Un proceso gaussiano es una colección de variables aleatorias, de las cuales cualquier número finito tiene una distribución gaussiana conjunta.

Un proceso gaussiano queda completamente especificado al definir su función de media

$m(\mathbf{x})$ y su función de covarianza (también llamada kernel) $k(\mathbf{x}, \mathbf{x}')$:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x})), (f(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned}$$

El proceso gaussiano se denota como:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')).$$

En la práctica, la función media $m(\mathbf{x})$ suele tomarse con valor constante 0 y toda la complejidad del modelo recae sobre el kernel k . Dentro del marco de la memoria, esto no es nuevo, se ha visto ya en capítulos anteriores (2.3) la importancia del kernel en el rendimiento de distintas funciones y métodos. Una función de covarianza comúnmente utilizada es la exponencial cuadrática (SE, *squared exponential*):

$$\text{cov}(f(\mathbf{x}_p), f(\mathbf{x}_q)) = k(\mathbf{x}_p, \mathbf{x}_q) = \exp\left(-\frac{1}{2}|\mathbf{x}_p - \mathbf{x}_q|^2\right)$$

Utilizar un GP como modelo implica que, antes de observar datos, se tiene una distribución previa sobre la función objetivo. Cada nueva observación obtenida (por ejemplo, de cada nueva combinación de hiperparámetros \mathbf{x}) se obtiene un error $y = f(\mathbf{x})$) proporciona información que "actualiza" la distribución. Una vez observado un conjunto de datos $D_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, y considerando el modelo

$$y = f(\mathbf{x}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2),$$

la predicción de un nuevo punto \mathbf{x}_* sigue una distribución normal cuya media y varianza son:

$$\begin{aligned} \mu(\mathbf{x}_*) &= \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{y}, \\ \sigma^2(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top (K + \sigma_n^2 I)^{-1} \mathbf{k}_*. \end{aligned}$$

Donde K es la matriz de covarianzas $k(\mathbf{x}_i, \mathbf{x}_j)_{i,j=1\dots n}$ y $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}_1), \dots, k(\mathbf{x}_*, \mathbf{x}_n)]^\top$.

Obsérvese que la media de la distribución es una combinación lineal de las observaciones \mathbf{y} . De hecho, puede verse como:

$$\mu(\mathbf{x}_*) = \sum_{i=1}^n \alpha_i k(\mathbf{x}_*, \mathbf{x}_i)$$

con $\alpha = (K + \sigma_n^2 I)^{-1} \mathbf{y}$.

Función de adquisición

Una vez se tiene el modelo probabilístico, GP, se necesita una función para decidir donde evaluar a continuación la función real. Ésta es la llamada función de adquisición. La función de adquisición $a(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}$ toma como entrada el modelo estadístico actualizado (es decir, la distribución sobre $f(\mathbf{x})$) y asigna a cada candidato \mathbf{x} una puntuación que indica cómo de

4. Métricas y metodología

"prometedor" sería evaluar en \mathbf{x} . La elección del siguiente punto que se va a probar se realiza entonces maximizando esta función de adquisición:

$$\mathbf{x}_{n+1} = \arg \max_{\mathbf{x} \in \mathcal{X}} a(\mathbf{x}, D_n),$$

dónde D_n son los datos observados hasta la iteración n . Las funciones de adquisición más utilizadas son las siguientes:

- Probabilidad de mejora (PI, *Probability of Improvement*): Probablemente sea la más sencilla. Para problemas de minimización, PI selecciona puntos que maximizan la probabilidad $P[f(x) < f(x_{\text{best}})]$ de mejorar el mejor valor observado hasta el momento $f(\mathbf{x}_{\text{best}})$. Bajo el modelo GP, esta probabilidad tiene forma analítica. Si $\mu(x)$ y $\sigma(x)$ son la media y desviación estándar predichas en \mathbf{x} , y f_{\min} es el mínimo observado, entonces

$$a_{PI}(\mathbf{x}) = \Phi(\gamma(\mathbf{x})) \quad \gamma(\mathbf{x}) = \frac{f_{\min} - \mu(\mathbf{x})}{\sigma(\mathbf{x})},$$

donde Φ es la función de distribución acumulada de la normal estándar. PI es fácil de calcular pero tiende a ser miope, ya que no considera cuánto podría mejorar un punto, sólo si es probable que lo mejore.

- Mejora esperada (EI, *Expected Improvement*): No se considera la probabilidad de mejora, sino la magnitud de la misma. Formalmente, la mejora se define como $l(\mathbf{x}) = \max\{0, f_{\min} - f(\mathbf{x})\}$ (para el caso de minimización).

$$a_{EI}(\mathbf{x}) = \sigma(\mathbf{x})(\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x}); 0, 1))$$

EI suele ser una de las funciones de adquisición más utilizadas, ya que equilibra de forma automática exploración y explotación: si $\mu(x)$ es mucho mayor que f_{\min} (es decir, peor que el mejor actual), EI puede ser pequeño a menos que $\sigma(x)$ sea grande; por otro lado, si $\mu(x)$ es muy prometedora (baja) o la incertidumbre $\sigma(x)$ es alta, EI será alto, incentivando ese punto.

- Cota superior de confianza (UCB, *Upper Confidence Bound*): Una idea más reciente consiste en seleccionar el próximo punto basándose en una combinación lineal de la predicción media y la incertidumbre. Para minimización, suele usarse la cota inferior:

$$a_{LCB}(\mathbf{x}) = \mu(\mathbf{x}) - \kappa\sigma(\mathbf{x}),$$

donde $\kappa > 0$ es un parámetro regulable. Equivalente, en maximización sería $a_{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x})$. κ controla explícitamente el balance entre exploración y explotación: un κ grande da más peso a la incertidumbre, explorando zonas menos conocidas, mientras que un κ pequeño prioriza las predicciones bajas de $\mu(x)$ explotando la información actual.

Parte II.

Desarrollo del proyecto

5. Diseño e implementación del sistema

5.1. Objetivos

El objetivo principal del proyecto es conseguir, a partir de las predicciones que hacen los distintos modelos que se han desarrollado, comparar (e incluso batir) las rentabilidades alcanzadas por la inversión basada en momentos (*momentum investing*). La inversión basada en momentos consiste, como se ha explicado al inicio de la presente memoria, en invertir en las acciones con mejor desempeño en el pasado reciente. Para afinar este "pasado reciente", existen diversos tipos de momentos, en función del marco temporal que abarquen, para decidir qué activos han conseguido las mejores rentabilidades.

Por ejemplo, el momentum a 1 mes escogerá una ventana temporal de un mes para decidir cuál ha sido la acción que mejor lo ha hecho dentro del universo de activos considerados. De forma análoga, el momentum a 2, 3 ó 9 meses funcionará igual, pero tomando ventanas del período correspondiente. Supóngase el siguiente escenario:

| | 1 mes | 3 meses | 9 meses |
|-----------|-------------|-------------|-------------|
| Microsoft | 6 % | 8 % | 18 % |
| Apple | 7 % | 13 % | 9 % |
| Nvidia | 24 % | 11 % | 8 % |

Tabla 5.1.: Rendimiento pasado de distintas cotizaciones en bolsa de empresas tecnológicas.

Si se escogiera el momentum a 1 mes, la empresa en la que se invertiría sería Nvidia. Si, por otro lado, se tomara el momentum a 3 meses, la ganadora sería Apple. Microsoft sería la seleccionada por su momentum a 9 meses.

Como se ha mencionado en los preliminares de esta memoria, la estrategia basada en momentos se consolidó a partir del artículo publicado por Narasimhan Jegadeesh y Sheridan Titman [4], quienes demostraron que invertir en valores con un mejor comportamiento reciente y desinvertir en aquellos con peores resultados generaba rentabilidades significativas en horizontes de 3 a 12 meses. Desde entonces, el momentum a 12 meses se ha consolidado como la referencia más utilizada. De hecho, existen diversos vehículos de inversión, como índices o fondos cotizados (*Exchange Traded Fund, ETF*), que replican esta estrategia. Algunos ejemplos pueden ser [S&P500 Momentum](#) o [MSCI USA Momentum](#).

5. Diseño e implementación del sistema

5.1.1. ¿Cómo se construye la cartera?

En primer lugar, ha de definirse brevemente lo que es una cartera de activos o, en el contexto de los mercados financieros, sencillamente *cartera*. La cartera de un inversor es, simplemente, su conjunto de activos de inversión. Una vez establecida, la cartera se actualiza o “rebalancea” mediante la venta de valores existentes y la compra de nuevos activos. Los activos de una cartera aumentan mediante la aportación de fondos adicionales para incrementar su tamaño total y se reducen mediante la venta de valores [45].

Criterio de selección de activos

La composición de las carteras en este proyecto queda determinada por un criterio de selección de activos. En el caso de la estrategia de la inversión basada en momentum, dicho criterio se basa en el comportamiento pasado: se asume que los valores que han obtenido mejores rentabilidades recientes continuarán haciéndolo en el futuro.

Es aquí donde entran en juego los modelos desarrollados en este proyecto. En lugar de utilizar únicamente el momentum histórico como criterio, se emplearán las predicciones generadas por distintos modelos desarrollados. Serán dichos modelos los que determinen qué acciones tienen mayor probabilidad de obtener buenos resultados en el futuro y, por tanto, cuáles formarán parte de la cartera.

Supóngase que se quiere construir una cartera formada por únicamente dos activos y se presenta una situación similar a la de la tabla anterior (5.1.1), con más empresas y rentabilidades diferentes:

| | 3 mes | 6 meses | 12 meses | Predicción modelo 1 mes |
|-----------|-------|---------|----------|-------------------------|
| Microsoft | 6 % | 8 % | 18 % | 7 % |
| Apple | 7 % | 13 % | 9 % | 3 % |
| Nvidia | 24 % | 11 % | 8 % | 9 % |
| Google | 2 % | 21 % | 10 % | 12 % |
| Amazon | 12 % | 9 % | 11 % | 14 % |

Tabla 5.2.: Rendimiento pasado de distintas empresas y predicción a un mes vista.

Con el criterio el momentum a 3 meses, las dos acciones escogidas para formar la cartera serían Nvidia y Apple. Si se escogiera, por otro lado, el momentum a un año, las ganadoras serían Microsoft y Amazon. Si el criterio fuera la predicción generada por el modelo, serían Amazon y Google. El objetivo es conseguir que la rentabilidad de la cartera construida a partir de las predicciones del modelo sea superior a la de la cartera basada en momentum (de cualquier horizonte).

El objetivo de los modelos es predecir, en términos relativos, la evolución esperada de los precios de los activos, de modo que sea posible establecer un ranking de rentabilidades y seleccionar aquellos con mejores perspectivas. Así, las predicciones del modelo pueden aportar una ventaja frente a las estrategias basadas solo en el momentum histórico.

Rotación de activos

La pregunta que uno puede hacerse es: ¿el activo seleccionado formará parte de la cartera para siempre? La respuesta es, casi con total seguridad, no¹. Entonces, ¿cuándo cambia la cartera y cómo? En este proyecto se ha establecido un rebalanceo mensual, lo que significa que cada mes se revisa la composición de la cartera y se decide, en función del criterio elegido, si mantener o sustituir los activos actuales.

Retomando el ejemplo anterior, supóngase que los activos escogidos han sido Google y Amazon. Tras un mes, llega el momento del posible rebalanceo. El modelo predice entonces que las acciones que mejores rentabilidades esperadas serán Google y Apple. Se procederá, pues, a rebalancear la cartera, sustituyendo Amazon por Apple, de modo que la nueva cartera queda compuesta por Google y Apple.

En los experimentos, se asumirá que la cartera se construye equiponderando todos los activos seleccionados, sin asignar mayor ponderación a aquellos que presenten un mejor desempeño según el criterio utilizado. Es decir, si n es el número de activos que componen la cartera, el peso para el activo i es $w_i = \frac{1}{n}$.

5.1.2. Estrategia de Clenow

La estrategia a batir no será la de momentum, sino una más refinada. Esta estrategia es la que plantea Andreas Clenow en su libro *Stocks on the Move: Beating the Market with Hedge Fund Momentum Strategies* [8].

Esta estrategia será utilizada como *benchmark*. Un benchmark es un índice de referencia o estándar que se utiliza para medir y comparar el rendimiento de una inversión, un fondo o una cartera. En palabras del propio Clenow:

Escoger un índice es más importante de lo que podría parecer. El índice es tu referencia. No significa que tengas que replicar el índice, pero es una vara de medir para evaluar tu rendimiento. Si no estás superando al índice, no estás haciendo un buen trabajo.

Pese a que la estrategia de Clenow no es, como tal, un índice (como el S&P500 o el MSCI World), es la que se usará para medir el rendimiento de las distintas carteras "basadas en modelos" presentadas en este proyecto.

¿En qué consiste, pues, la estrategia de Clenow? Básicamente, esta estrategia utiliza como criterio un momentum *mejorado*, calculado como la pendiente de una regresión exponencial multiplicada por el coeficiente de determinación (2.2) de dicha regresión.

Además de este criterio, Clenow utiliza otras reglas que no han sido aplicadas en el proyecto. Algunas de sus reglas detalladas incluyen operar sólo los miércoles, sólo comprar si el índice S&P 500 está por encima de su media móvil de 200 días o vender una acción si ya no está en el 20 % superior de los activos incluidos por dicho índice ordenados por momentum.

¹En el hipotético caso en el que una empresa tuviera una rentabilidad fija superior al resto de las del mercado para toda la eternidad, esa empresa sería escogida y no se sustituiría jamás. No obstante, se trata de un escenario prácticamente imposible en la realidad de los mercados financieros.

5. Diseño e implementación del sistema

Por otro lado, Clenow no asigna la misma ponderación a los activos que componen su cartera, calibra el tamaño de cada una de las posiciones para ajustar su riesgo. Esto último tampoco se ha tenido en cuenta en la experimentación, para limitar el número de grados de libertad y centrarse en la evaluación de la calidad de las recomendaciones realizadas por los modelos entrenados.

En definitiva, la referencia utilizada en este proyecto es un criterio de inversión basada en momentos que utiliza regresión exponencial ponderada por el coeficiente de determinación. En primer lugar, ¿por qué regresión exponencial?

Para medir el momentum, uso la regresión exponencial. Esto plantea dos preguntas obvias: ¿qué es la regresión y por qué es exponencial? Antes de abordar la parte exponencial, necesitas entender el concepto de regresión lineal. [...]

La regresión lineal es un método para ajustar una línea sobre una serie de valores. Es una forma de encontrar la línea que mejor se ajusta, en este caso a una serie temporal de precios. La pendiente te dice cuánto debe subir o bajar la línea por cada punto de datos sucesivo. La línea resultante es el mejor ajuste lineal a los datos de precios, o mejor dicho, la que tiene el menor error. La pendiente es lo que realmente nos interesa, ya que eso nos dice la dirección del precio de la acción. [...]

La pendiente de regresión lineal es, por lo tanto, una medida de la velocidad o del momentum de la acción. Sin embargo, el problema es que la pendiente se expresa en dólares y centavos. Si una acción que tiene un precio de \$10 sube \$2 por día, eso es mucho más significativo que si una acción con un precio de \$100 avanza los mismos \$2 cada día.

Por eso se utiliza la regresión exponencial. Mientras que la pendiente de regresión lineal se expresa en unidades monetarias, la pendiente exponencial se expresa en porcentaje.

Y, en segundo lugar, ¿qué papel desempeña el coeficiente de determinación en el criterio de selección de activos?

La acción con la pendiente más alta estará en la parte superior de la lista. Cuanto más fuerte esté subiendo algo, más arriba estará en la lista. Este es un ranking de momentum puro. Todavía hay un pequeño problema con nuestro enfoque de ranking. Usar solo la regresión exponencial anualizada para el ranking significa que no nos importa el ajuste. [...]

Existe un método perfectamente válido para medir qué tan bien nuestros datos de precios se ajustan a la línea de regresión. Se llama coeficiente de determinación, generalmente designado como R^2 . [...] El R^2 te dice qué tan bien la serie de precios se ajusta a la línea de regresión, cero es el valor mínimo para el R^2 , mientras que uno es el máximo. [...]

De esta manera, nuestros rankings de momentum serán una combinación de momentum, en forma de la pendiente de regresión, y también una medida de calidad en términos de R^2 . Multiplica la pendiente de regresión exponencial por el coeficiente de determinación (R^2) y tendrás una base bastante buena para

clasificar acciones entre sí.

Una vez aclarado el origen del criterio de selección de activos queda por determinar un parámetro: ¿qué valores se utilizan para calcular la regresión? Clenow opta por escoger los 90 días anteriores para calcular la regresión. Este valor será también el escogido para calcular los modelos desarrollados en este proyecto.

5.2. Requisitos del sistema

Una vez explicado el objetivo principal del proyecto, los principales requisitos del back-end del sistema serían los siguientes:

1. Funcionales:

- Entrenar distintos modelos predictivos con un mismo conjunto de datos y bajo un mismo procedimiento de entrenamiento.
- Almacenar predicciones, métricas y metadatos en formatos estándar (JSON, CSV, XML) que permitan su posterior análisis e integración.
- Comparar los modelos mediante métricas comunes: MAE, MAPE, MSE, RMSE.
- Usar las predicciones de los modelos como base para la construcción y simulación de carteras de inversión.
- Integración sencilla con la aplicación web posterior.
- El sistema debe ser capaz de entrenar y evaluar modelos en un tiempo razonable con los recursos disponibles.

2. No funcionales:

- El código ha de ser modular y extensible, de forma que permita la incorporación de nuevos modelos sin modificar la estructura base.
- Los experimentos y simulaciones deben poder replicarse bajo las mismas condiciones iniciales.
- El código ha de mantener un nivel adecuado de legibilidad y mantenibilidad, apoyado en pruebas unitarias y de integración.

Una vez establecidos los principales requisitos del sistema, se describe la organización del proyecto, concretamente la estructura de directorios y archivos.

5.3. Diseño del sistema

El diseño del sistema está compuesto por varias familias de componentes: modelos (encargados de realizar predicciones), experimentos (utilizados en la evaluación y creación de

5. Diseño e implementación del sistema

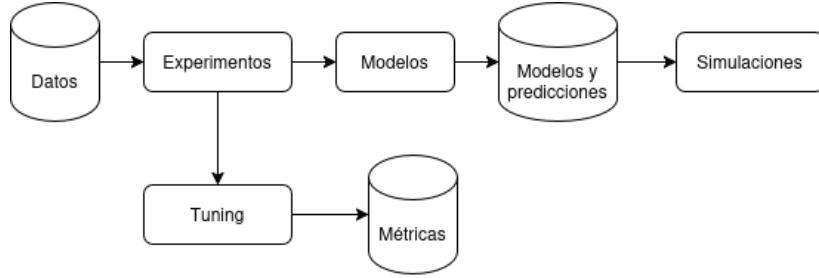


Figura 5.1.: Esquema general del sistema.

modelos), simulaciones (de carteras de inversión) y algoritmos de optimización de hiperparámetros (tuning).

Los experimentos constituyen el núcleo del sistema, ya que se encargan de gestionar el proceso completo de entrenamiento, validación y evaluación de los modelos. Para ello, automatizan la creación de los modelos, aplican técnicas de cross-validation, explicadas en [Sección 4.3](#), y registran las métricas de rendimiento obtenidas.

Los modelos, por su parte, son los responsables de generar predicciones sobre los activos financieros a partir de los datos procesados. En función de la arquitectura utilizada, redes neuronales o modelos de aprendizaje clásicos, se usarán diferentes clases.

Los algoritmos de optimización de hiperparámetros permiten aplicar los procesos explicados en [Sección 4.4](#) para mejorar las capacidades predictivas de los modelos.

Por último, las simulaciones utilizan las predicciones generadas por los modelos construidos para crear y evaluar diferentes carteras de inversión, calculando métricas como las explicadas en [Sección 4.2](#).

A continuación, se describen en detalle cada una de las familias de componentes.

5.3.1. Modelos

Esta familia está compuesta, a su vez, de dos subfamilias de clases conectadas entre sí:

- Modelos: Esta clase base es la encargada de construir y entrenar los modelos, los cuales generarán las predicciones para los distintos activos. De ella heredan dos familias:
 - Modelos Keras: Se han creado 3 clases para cada uno de los diferentes tipos de redes neuronales explicadas en el [Capítulo 3](#). RecurrentModel para las redes recurrentes, ConvolutionalModel para las redes convolutivas y TransformerModel para los transformers. Todas ellas heredan de la clase KerasModel.
 - Modelos Scikit-Learn: Se han creado 3 clases para cada uno de los diferentes tipos de modelos de aprendizaje automático clásicos explicados en el [Capítulo 2](#). La clase ExponentialScikitLearnModel para el modelo basado en regresión exponencial que utiliza Clenow, RandomForestModel para los modelos basados en Random Forest y SVRModel para los modelos basados en SVR. Todas ellas heredan de la

clase ScikitLearnModel.

- Fábricas de Modelos: Para cada uno de los modelos creados hay un patrón de diseño "factory" (fábrica). Estas clases sirven para crear y entrenar los modelos cómodamente y serán las utilizadas por los experimentos.

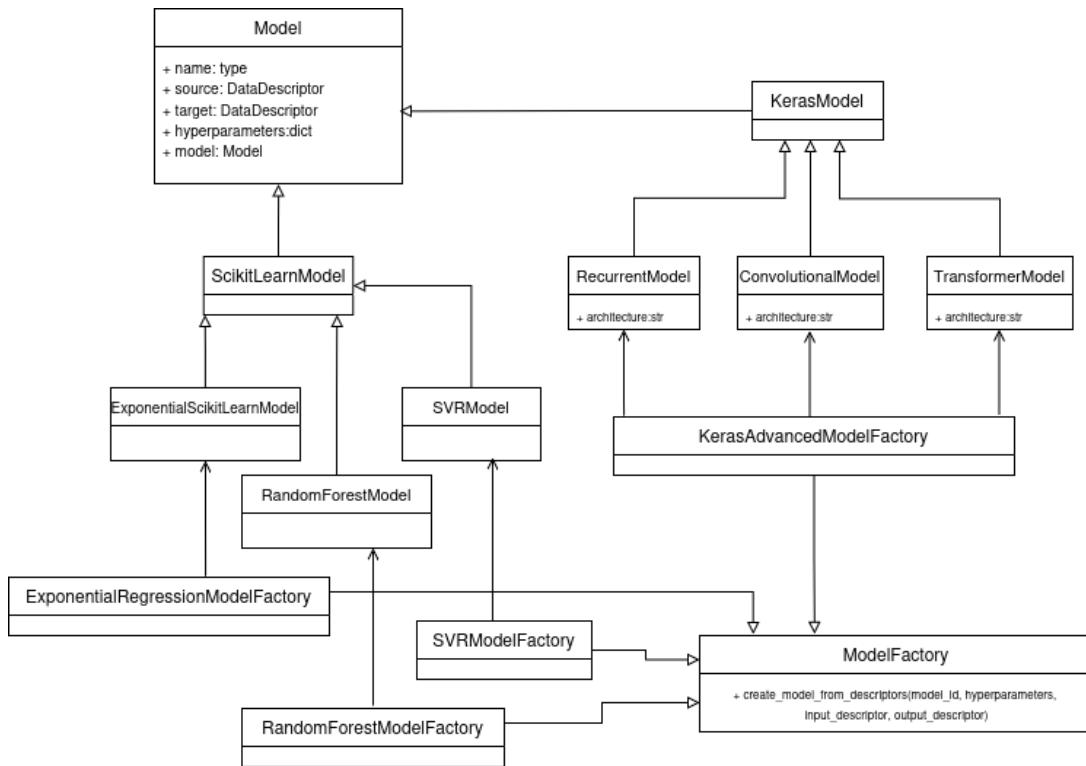


Figura 5.2.: Diagrama de clases para los modelos y sus fábricas.

5.3.2. Experimentos

Como se ha mencionado, los experimentos son los que gestionan el proceso de entrenamiento, validación y evaluación de los modelos. Los experimentos, a su vez, se dividen en dos según cómo se haya entrenado el modelo:

- Experimentos globales: Estos modelos se entrena n utilizando cross-forward validation. Se explicará con más detalle en [Sección 6.1](#).
- Experimentos locales: Estos modelos reciben como entrada los últimos *horizon* días y predicen a *lookahead* días vista. Esta clase ha sido únicamente utilizada para generar las predicciones de la regresión exponencial, utilizadas para la estrategia de Clenow.

5. Diseño e implementación del sistema

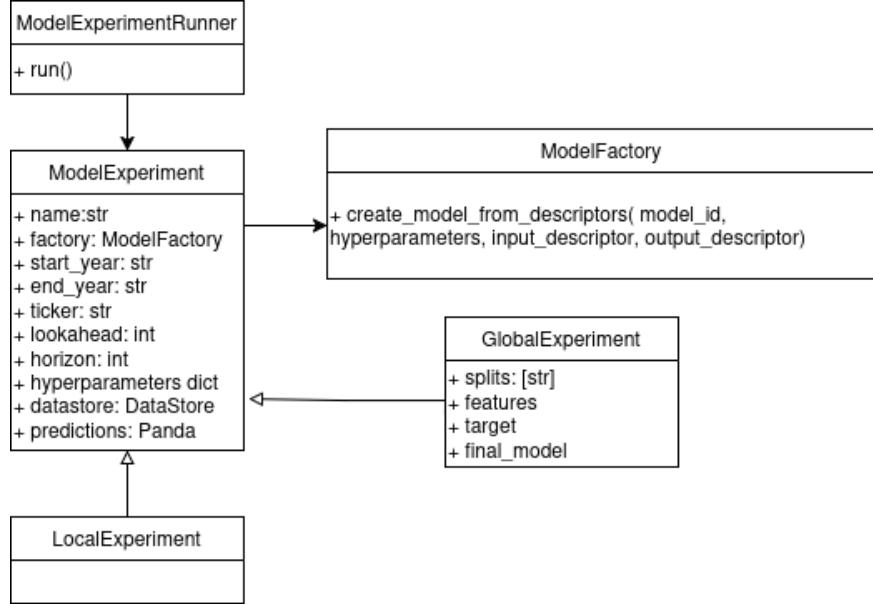


Figura 5.3.: Diagrama de clases de los experimentos.

5.3.3. Simulaciones

Las simulaciones utilizan los indicadores (criterios) para construir las carteras a partir de una fecha de comienzo y de final. Estos indicadores son construidos a partir de las predicciones generadas por los modelos creados a partir de los experimentos. Además, la clase `SimulationFactory` será la que conecta con toda la aplicación web que será explicada posteriormente.

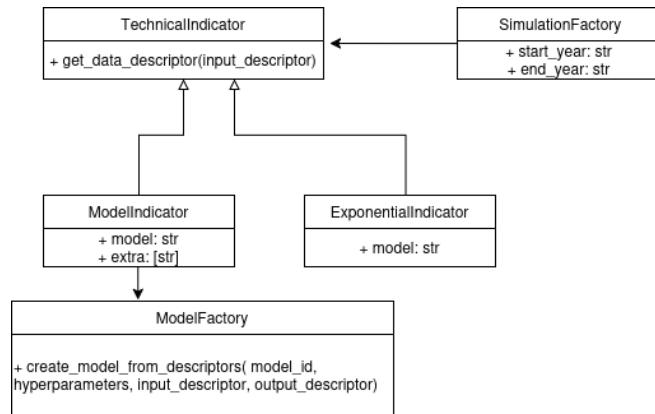


Figura 5.4.: Diagrama de clases de las simulaciones

5.3.4. Técnicas de optimización de hiperparámetros

Esta familia implementa diferentes clases orientadas a la optimización de hiperparámetros y a la automatización del proceso de búsqueda de configuraciones óptimas para cada modelo. Las dos metodologías utilizadas han sido las explicadas en [Sección 4.4](#): optimización de bayes y búsqueda aleatoria.

Además, se han creado dos clases complementarias a las que implementan estas técnicas para poder continuar la ejecución desde el "último punto guardado". Esta funcionalidad resulta especialmente útil, dado que algunos experimentos requieren varias horas de cómputo y pueden verse interrumpidos por limitaciones de hardware o caídas del sistema.

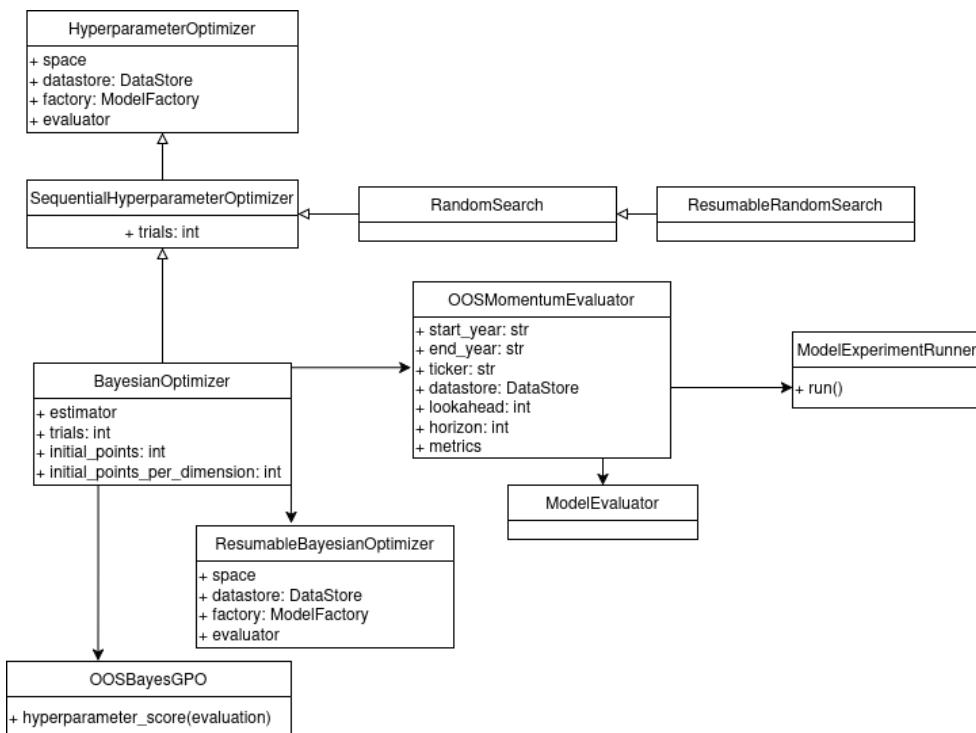


Figura 5.5.: Diagrama clases para tuning

5.4. Despliegue del sistema

En esta sección se describe tanto la metodología seguida a la hora de desarrollar la parte informática del proyecto, como el diseño del sistema implementado.

En primer lugar, todo el proyecto ha utilizado como base un proyecto previamente desarrollado por el tutor de este Trabajo Final de Grado: Fernando Berzal. En segundo lugar, el código escrito para este trabajo puede encontrarse en [mi repositorio de GitHub](#). Y, en tercer lugar, el código ha sido desarrollado en Python y se han utilizado las bibliotecas Scikit-learn

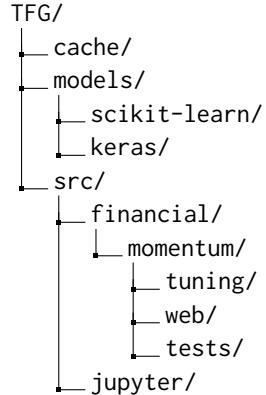
5. Diseño e implementación del sistema

y Keras para desarrollar los modelos.

La metodología de desarrollo seguida ha sido la del "ciclo de vida clásico", también llamada "en cascada". Los objetivos del TFG estaban claros desde el inicio: desarrollar distintos modelos predictivos para superar los rendimientos de las estrategias basadas en momentum propuestas por Clenow.

La parte informática de este proyecto consta de dos partes bien definidas: la interfaz de usuario(front-end del sistema) y el entrenamiento de modelos a partir de los datos históricos disponibles (back-end del sistema). Esta sección se centrará en la segunda parte.

La estructura de directorios que ha sido utilizada ha sido la siguiente:



Donde cada una de las carpetas tiene la siguiente función:

- cache: En esta carpeta se guardan los datos de los distintos activos y acciones con los que se entrena a los modelos. Estos datos han sido extraídos por el tutor de distintas fuentes: [Yahoo Finance](#), [Investing](#) y [FRED](#).
- models: En esta carpeta se guardan los modelos entrenados y todos los datos relacionados con ellos (métricas, hiperparámetros, predicciones). A su vez, esta carpeta se divide en 2: scikit-learn y keras. Los modelos construidos con las librerías *Scikit-Learn* son los mencionados en el [Capítulo 2](#), los construidos con la librería *Keras* son los basados en redes neuronales, mencionados en el [Capítulo 3](#).
- src: En esta carpeta se encuentra todo el desarrollo del proyecto, concretamente en el directorio financial/momentum. En este directorio se encuentran todas las clases y ficheros creados en el TFG, las cuales serán explicadas posteriormente. Por otro lado, en el directorio jupyter se han creado algunos *notebooks* que se han utilizado para hacer pruebas durante el desarrollo de las distintas clases.

El flujo de datos del sistema resulta en:

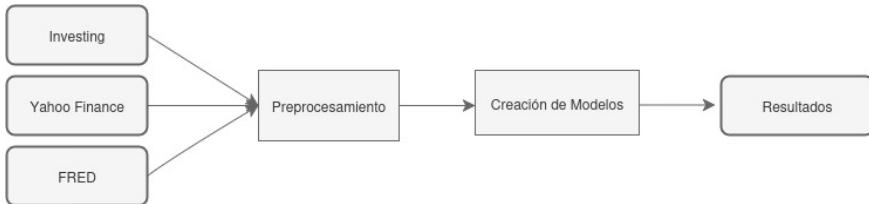


Figura 5.6.: Flujo de Datos

5.5. Interfaz de usuario

En este capítulo se presentará la aplicación web desarrollada complementaria al proyecto principal. La aplicación web permite poder visualizar las predicciones de los modelos, sus métricas y ejecutar simulaciones de carteras.

Para construir la aplicación se han utilizado las librerías de python: Flask, Jinja y la librería [plotly](#) para las gráficas, además de los correspondientes archivos HTML y CSS. En primer lugar, se enumeran los requisitos de usuario que ha de cumplir la aplicación web:

- Visualización de acciones: El sistema deberá permitir que el usuario observe todas las acciones que cuentan con algún modelo entrenado.
- Modelos por acción: El sistema deberá permitir que el usuario observe los distintos modelos entrenados asociados a una acción.
- Predicciones y métricas: El sistema deberá permitir que el usuario consulte las predicciones de cada modelo, así como sus métricas e hiperparámetros.
- Configuración de simulaciones: El sistema deberá permitir que el usuario configure simulaciones de carteras mediante parámetros como universo de acciones, número de acciones y modelo a utilizar.
- Comparación de carteras: El sistema deberá permitir que el usuario observe los rendimientos de la cartera simulada frente al índice S&P500 y la estrategia de Clenow con el mismo universo y número de acciones.

La aplicación se divide, esencialmente, en dos partes: la consulta de los modelos predictivos y la ejecución de simulaciones de carteras.

- Modelos: En la aplicación web se pueden ver todos los modelos para cada una de las acciones. Se pueden ver las predicciones del modelo, los hiperparámetros y las métricas. Para ello, la aplicación lee los archivos que han sido almacenados cuando se han entrenado los correspondientes modelos.
- Simulaciones: La aplicación web permite ejecutar las simulaciones de carteras que usen los modelos entrenados. Para ello se hace uso de la clase `SimulationFactory`.

Véase la conexión de cada una de las páginas principales de la aplicación con el back-end:

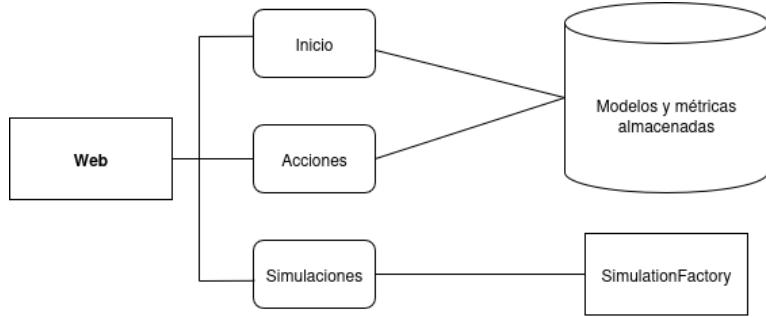


Figura 5.7.: Arquitectura general de la aplicación web.

5.6. Manual de usuario

El usuario no requiere tener un conocimiento técnico para poder usar la aplicación. De hecho, todas las páginas de la aplicación cuentan con un apartado de ayuda que explica el contenido y la finalidad de dicha página.

Nótese que todas las páginas de la aplicación cuentan con el mismo encabezado, donde se puede consultar las tres secciones de la aplicación: Inicio, Acciones, Simulaciones; y pie, donde se puede acceder al pdf de esta memoria y una página de contacto acerca del autor.

En la página de inicio, `index.html`, aparecen los últimos modelos entrenados, divididos entre los modelos explicados en el [Capítulo 2](#) y los modelos explicados en [Capítulo 3](#). Obsérvese [Figura 5.8](#)

En la página de acciones, se muestran todas las acciones e índices que cuentan con, al menos, algún modelo entrenado. Nótese que los activos se muestran por su *ticker*, no por su nombre. Se solventa el primer requisito de usuario. Obsérvese [Figura 5.9](#).

Cada acción cuenta con su correspondiente página, donde puede verse el nombre de la acción y los modelos entrenados para dicha acción. Se puede observar el tipo de modelo, los datos con los que ha sido entrenado y la arquitectura del mismo. El segundo requisito, queda solventado. Obsérvese [Figura 5.10](#). Al acceder a un modelo concreto, se pueden observar: las predicciones de dicho modelo frente al valor real, las métricas para el conjunto de entrenamiento y el de prueba – las cuales ya han sido explicadas y se ha hecho hincapié en su diferencia –, y los hiperparámetros del modelo. El tercer requisito se cumple. Obsérvese [Figura 5.11](#)

Nótese que la gráfica es dinámica, es decir, se puede modificar la ventana temporal para ver con más detalle la diferencia entre las predicciones y los valores reales. Para ello, el usuario puede tanto pinchar en la barra que aparece justo debajo de la gráfica, como pulsar alguno de los botones que aparecen en la esquina superior derecha. Véase [Figura 5.12](#).

En la página de configuración de la simulación, el usuario dispone de numerosas variables para configurar la simulación: número de activos que habrá en la cartera, activos que pueden ser elegidos en la simulación, la ventana temporal, el modelo utilizado como criterio y el

5.6. Manual de usuario

The screenshot shows a dark-themed web interface. At the top left is a circular logo with 'vs.' and 'momentum'. To its right are navigation links: 'Inicio', '|', 'Acciones', '|', 'Simulaciones'. On the far right is a help icon (a question mark). Below the header, the title 'Últimos modelos entrenados' is centered. Underneath, there are two columns: 'Machine Learning' and 'Deep Learning', each listing several trained model entries. A footer bar at the bottom contains 'Autor' and 'Repo'.

| Machine Learning | Deep Learning |
|----------------------------|--------------------------------------|
| randomforest - EWJ - 2025 | transformer - EWJ - 2025 - multiple |
| svr - EWJ - 2025 | transformer - FEZ - 2025 - multiple |
| clenow - EWJ - 2025 | transformer - VYM - 2025 - multiple |
| randomforest - FEZ - 2025 | transformer - USMV - 2025 - multiple |
| svr - FEZ - 2025 | transformer - AGG - 2025 - multiple |
| clenow - FEZ - 2025 | transformer - TIP - 2025 - multiple |
| randomforest - VYM - 2025 | transformer - IEF - 2025 - multiple |
| svr - VYM - 2025 | transformer - SHY - 2025 - multiple |
| clenow - VYM - 2025 | transformer - DBC - 2025 - multiple |
| randomforest - USMV - 2025 | transformer - GLD - 2025 - multiple |
| svr - USMV - 2025 | transformer - EEM - 2025 - multiple |
| clenow - USMV - 2025 | transformer - CSPX - 2025 - multiple |
| randomforest - AGG - 2025 | transformer - XLI - 2025 - multiple |
| svr - AGG - 2025 | transformer - XLP - 2025 - multiple |
| clenow - AGG - 2025 | transformer - XLV - 2025 - multiple |

Figura 5.8.: Página principal.

activo refugio². El cuarto requisito queda solventado, obsérvese la [Figura 5.13](#).

Tras presionar el botón azul "Ejecutar simulación" y esperar a que se complete la simulación, el usuario puede observar el rendimiento de su cartera frente a la estrategia de Clenow ([5.1.2](#)) y el S&P500.

Respecto a la gráfica, el funcionamiento es el mismo que el explicado para las predicciones de los modelos ya que se usa la librería [plotly](#).

Encima de la gráfica puede observarse una tabla comparativa con las distintas métricas bursátiles explicadas en [Capítulo 4](#) entre las tres carteras. Si el usuario quiere, puede observar únicamente las métricas de una determinada cartera mediante el uso de los botones situados encima de la tabla. Véase [Figura 5.14](#).

²Si el criterio seleccionado predice que todos los activos del universo van a dar rendimientos negativos el dinero ha de colocarse en un activo, denominado "refugio", que protegerá la cartera de rendimientos negativos.

5. Diseño e implementación del sistema

The screenshot shows a user interface for a financial application. At the top, there is a dark blue header bar with a white circular logo containing a stylized 'M' and the text 'vs. momentum'. To the right of the logo are three navigation links: 'Inicio', 'Acciones', and 'Simulaciones'. On the far right of the header is a small blue circular icon with a white question mark. Below the header is a light gray content area with a title 'Acciones con modelos disponibles' centered above a grid of stock tickers. The grid is organized into five rows and ten columns. The first row contains: AAPL, ACN, AGG, AMZN, ASML, BAC, BAM, BRK-B, CNDX, and CSPX. The second row contains: CVX, DBC, DIS, EEM, EIMI, EUNL, EWJ, F, FEZ, and GLD. The third row contains: GM, GOOG, IEF, INTC, IWB, IWDA, IYY, JNJ, KO, and MCD. The fourth row contains: META, MSFT, MXUS, NVDA, NVS, ORCL, PFE, QCOM, QQQ, and RSP. The fifth row contains: SHY, SPY, TIP, TSLA, UNH, URTH, USMV, VONE, VYM, and WBD. Below the grid, there are two more rows of single-ticker boxes: WFC, XLC, XLE, XLF, XLI, XLP, XLV, XOM, and ^GSPC. At the bottom of the page is a dark blue footer bar with the text 'Autor' and 'Repo'.

Figura 5.9.: Página con las acciones con modelos disponibles.

5.6. Manual de usuario

The screenshot shows a web application interface. At the top, there is a dark blue header bar with the logo 'VS. momentum' on the left and navigation links 'Inicio | Acciones | Simulaciones' on the right. Below the header is a light gray content area with a title 'Modelos disponibles para Apple Inc.' followed by a table. The table has four columns: 'Arquitectura', 'Entrenado hasta', 'Tipo', and 'Información Extra'. The rows list various model architectures and their details:

| Arquitectura | Entrenado hasta | Tipo | Información Extra |
|--------------|-----------------|--------------|-----------------------------|
| clenow | 2025 | scikit-learn | |
| cnn | 2025 | keras | M2 Money Stock |
| cnn | 2025 | keras | |
| cnn2d | 2025 | keras | |
| lstm | 2025 | keras | M2 Money Stock |
| lstm | 2025 | keras | |
| randomforest | 2025 | scikit-learn | |
| rnn | 2025 | keras | M2 Money Stock |
| rnn | 2025 | keras | |
| svr | 2025 | scikit-learn | |
| transformer | 2025 | keras | |
| transformer | 2025 | keras | M2 Money Stock |
| transformer | 2025 | keras | VIX (CBOE Volatility Index) |

At the bottom of the content area, there is a dark blue footer bar with the text 'Autor | Repo'.

Figura 5.10.: Página con los modelos disponibles para una acción.

5. Diseño e implementación del sistema



Figura 5.11.: Página de resultados de un modelo concreto.



Figura 5.12.: Ampliación de la gráfica.

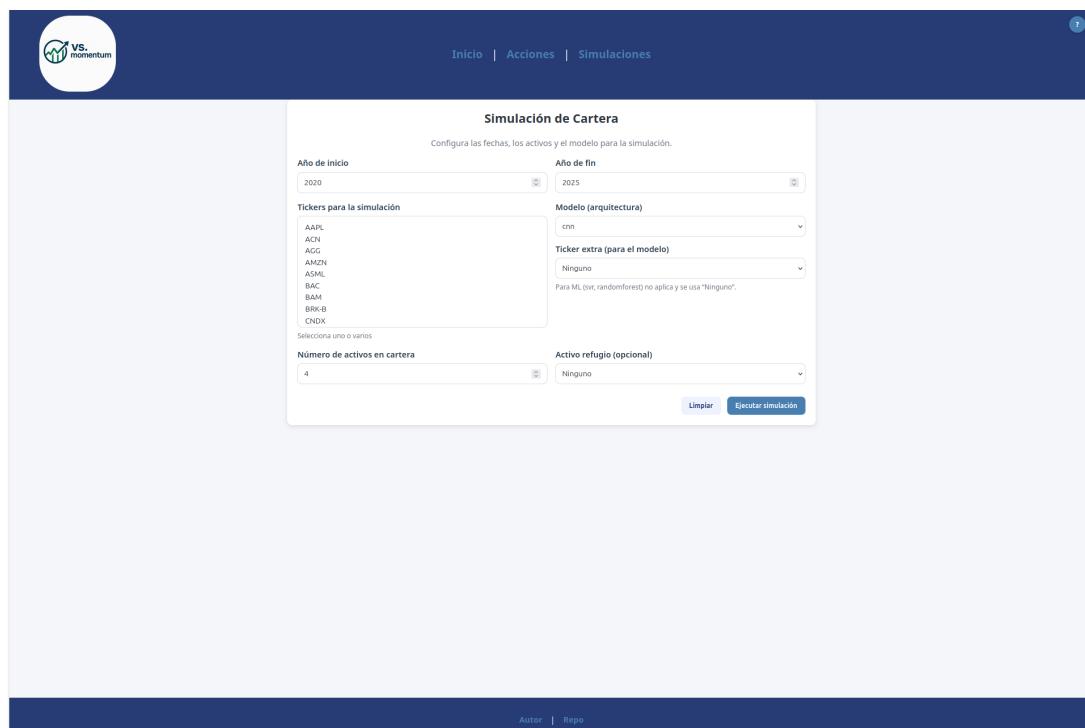


Figura 5.13.: Página de configuración de la simulación.

5. Diseño e implementación del sistema

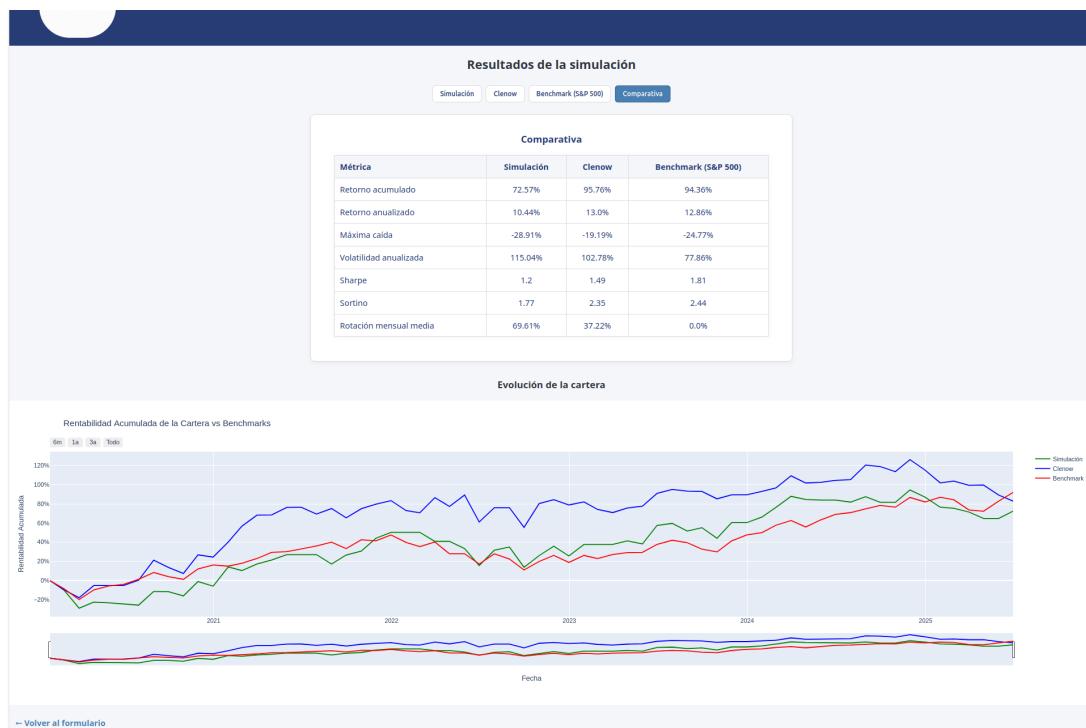


Figura 5.14.: Página de resultados de la simulación.

6. Resultados experimentales

6.1. Entrenamiento de modelos

El objetivo de aprendizaje para cada activo i consiste en predecir un retorno relativo a L días vista, con $L = 20$ (*lookahead*), empleando como información de entrada la historia reciente de ese mismo activo (y, opcionalmente, de otros) durante $H = 90$ días (*horizon*). En concreto, la variable "diana" en el instante t se define como

$$y_t = \frac{P_{t+L} - P_t}{P_t},$$

donde P_t es el precio del activo en el instante t . Al trabajar con retornos relativos, el problema es independiente de la escala del precio y comparable entre activos.

Las entradas para el modelo son los cambios porcentuales del precio del activo de los H días anteriores, $[t - H + 1, t]$. Se ha cogido 90 como valor para *horizon* porque es el valor usado por Clenow en su estrategia.

Para que el modelo no "viera el futuro" se ha utilizado una validación cruzada de tipo *walk-forward* (explicada en [Sección 4.3](#)), la cual consiste en dividir el conjunto de datos en cortes temporales crecientes $\tau_1, \tau_2, \dots, \tau_K$, en este caso anuales. Para cada corte τ_i , el modelo sólo ha sido entrenado con los datos anteriores a τ_i (datos de entrenamiento) y genera las predicciones para para $[\tau_i, \tau_{i+1})$ (datos de prueba o *test*). Todas las predicciones generadas se concatenan en una única serie ordenada por fecha, la cual será la utilizada para construir las carteras.

Al representar las predicciones gráficamente, éstas se alinean con los valores reales del activo de tal forma que pueda comprobarse visualmente la diferencia entre el valor predicho y el valor real.

Toda la creación de los distintos modelos ha sido automatizada mediante el uso de clases genéricas como `ModelFactory`.

6.1.1. Modelos clásicos

Todas las imágenes muestran las predicciones desde el año 2016 hasta 2025, acompañadas de las correspondientes métricas de rendimiento. Para ilustrar el rendimiento de los distintos tipos de modelos, de observarán únicamente los modelos entrenados para predecir los valores del índice estadounidense S&P 500 al ser la composición de las 500 empresas con mayor capitalización bursátil de Estados Unidos. Del mismo modo, los experimentos realizados para mejorar los modelos se basarán en el uso de este índice.

6. Resultados experimentales

6.1.1.1. Regresión Exponencial

La regresión exponencial, como se ya se ha explicado, es la utilizada en la metodología de Clenow. Como se ha mencionado anteriormente, los modelos basados en regresión exponencial han sido entrenadas de una forma distinta al resto de modelos. Esto es debido a que se ha querido seguir lo más fielmente la estrategia expuesta por Clenow.

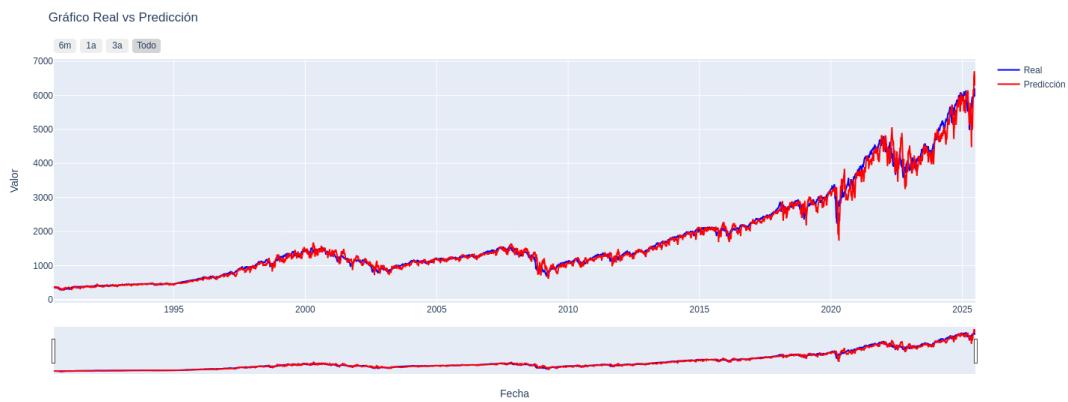


Figura 6.1.: Modelo basado en regresión exponencial para el S&P 500.

Pueden observarse ciertos picos más agudos, pero el resultado es relativamente bueno. Recuérdense las limitaciones de estos modelos en (2.1).

| Set | N | MSE | RMSE | MAE | MAPE |
|--------|------|------------|----------|---------|----------|
| Global | 8830 | 20516.1348 | 143.2345 | 86.0702 | 0.0484 % |

Tabla 6.1.: Métricas de rendimiento para el modelo basado en regresión exponencial para el S&P 500.

Ha de tenerse en cuenta, para comparar con los posteriores modelos, que este modelo tiene predicciones desde 1990 porque no requiere de un entrenamiento previo como sí lo requieren los modelos entrenados globalmente. A medida que se incrementa el volumen de datos, es esperable que el error acumulado también crezca.

Para poder comparar mejor los resultados, véanse las predicciones del modelo desde el año 2016:

6.1. Entrenamiento de modelos

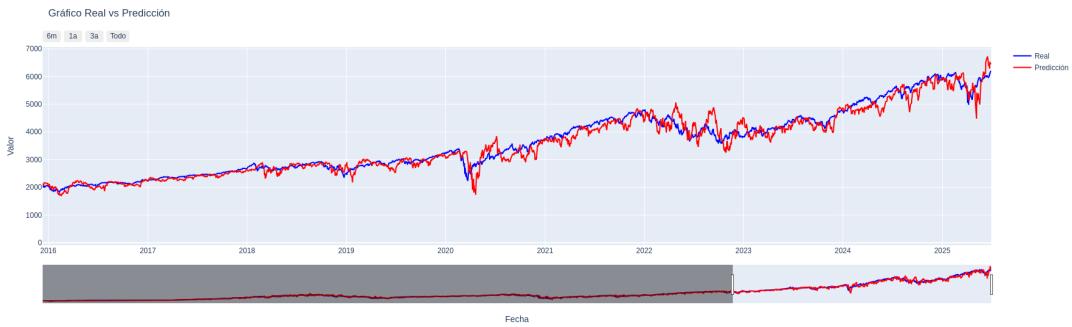


Figura 6.2.: Modelo basado en regresión exponencial para el S&P 500 desde el año 2016.

6.1.1.2. SVR

A continuación, se presenta el desempeño de los modelos globales con el índice estadounidense. En primer lugar, SVR.



Figura 6.3.: Modelo SVR S&P 500.

Puede observarse que el modelo tiende a replicar la serie temporal con un ligero desfase temporal, lo que limita su capacidad real de predicción. Dicho de forma simple: la línea roja (predicciones) es prácticamente un calco de la línea azul (valores reales), solo que desplazada un poco a la derecha. Esto será común a todos los modelos siguientes. Recuérdese las limitaciones y observaciones de este modelo en la sección (2.3).

| Set | N | MSE | RMSE | MAE | MAPE |
|---------------|-------|--------|--------|--------|----------|
| Entrenamiento | 73437 | 0.0025 | 0.0496 | 0.0413 | 5.2129 % |
| Test | 15063 | 0.0037 | 0.0605 | 0.0485 | 4.0792 % |

Tabla 6.2.: Resultados de las métricas para el modelo SVR para S&P 500.

6. Resultados experimentales

Las métricas son mejores que las conseguidas por las regresiones exponenciales. Sin embargo, dichas métricas no son comparables entre sí pues el número de datos es distinto. Esto es debido a cómo se han entrenado los modelos en cada caso.

A pesar de esta mejoría, dichas métricas no recogen el desfase observado. Es crucial, pues, observar las predicciones reales antes que las métricas para poder interpretar correctamente los resultados. De hecho, las métricas deben entenderse como un complemento a la evaluación visual, y no como una medida suficiente por sí mismas.

6.1.1.3. RandomForest

Se observan a continuación los resultados obtenidos por el modelo random forest.

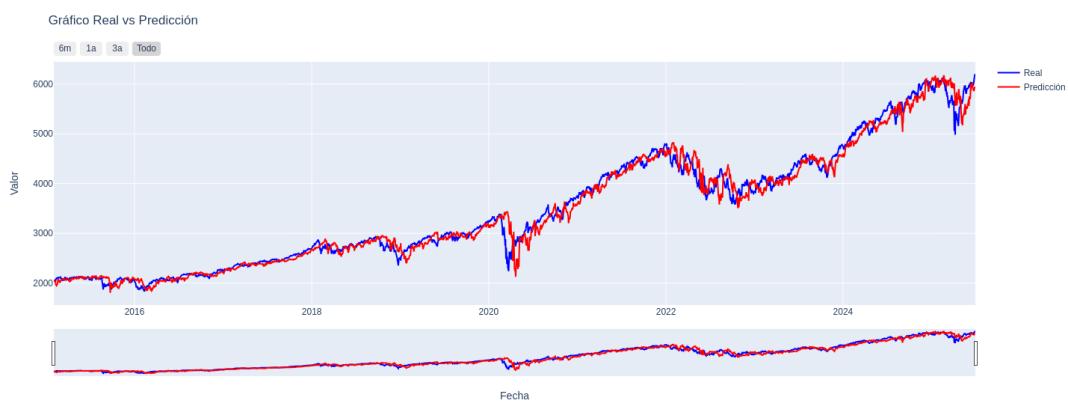


Figura 6.4.: Modelo RandomForest S&P 500.

Igual que sucedía con SVR, las predicciones son un desplazamiento de la serie original hacia el futuro. Podría decirse que el modelo se limita a replicar los valores reales anteriores, no a predecir. Recuérdense las limitaciones y observaciones de RandomForest (2.4).

| Set | N | MSE | RMSE | MAE | MAPE |
|---------------|-------|--------|--------|--------|----------|
| Entrenamiento | 73437 | 0.0002 | 0.0136 | 0.0093 | 0.6596 % |
| Test | 15063 | 0.0027 | 0.0511 | 0.0371 | 1.9758 % |

Tabla 6.3.: Resultados de las métricas para el modelo RandomForest para S&P 500.

6.1.2. Modelos simples basados en redes neuronales

Seguidamente, se analizarán los resultados de los distintos tipos de redes neuronales para el S&P 500. Todas las redes neuronales han contado con los mismos hiperparámetros:

- 2 capas ocultas con 64 y 32 neuronas respectivamente.

- El tamaño del lote (*batch_size*) de 16 muestras.
- El número de épocas ha sido de 10.
- Se ha empleado la función de activación ReLU en las capas ocultas y una función lineal en la capa de salida.
- La función de pérdida usada ha sido MSE.

6.1.2.1. Redes recurrentes y LSTM

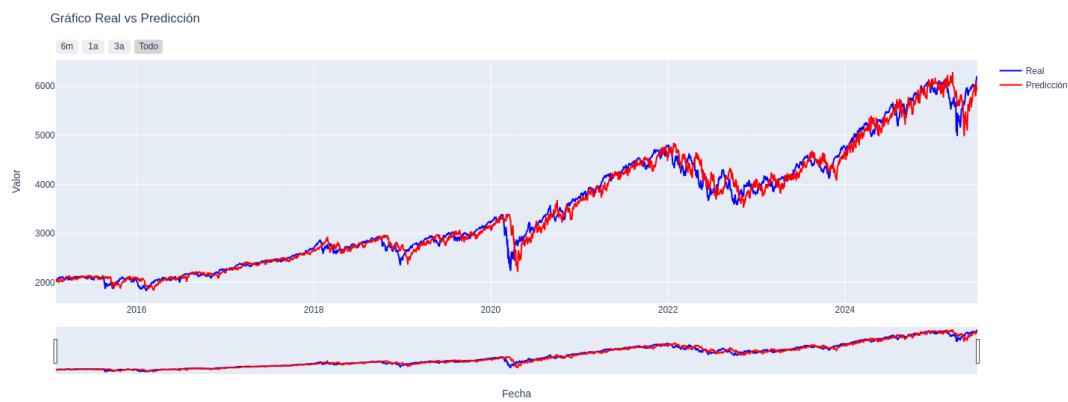


Figura 6.5.: Modelo RNN S&P 500.

Parece que a la red neuronal recurrente le sucede lo mismo que con las anteriores: la serie predicha es un ligero desplazamiento hacia el futuro de la serie real.

| Set | N | MSE | RMSE | MAE | MAPE |
|---------------|----------|------------|-------------|------------|-------------|
| Entrenamiento | 73437 | 0.0021 | 0.0462 | 0.0328 | 2.1982 % |
| Test | 15063 | 0.0025 | 0.0494 | 0.0361 | 2.0666 % |

Tabla 6.4.: Resultados de las métricas para el modelo RNN S&P 500.

Las métricas son parecidas a las conseguidas por SVR o la regresión exponencial: buenas, pero engañosas.

6. Resultados experimentales

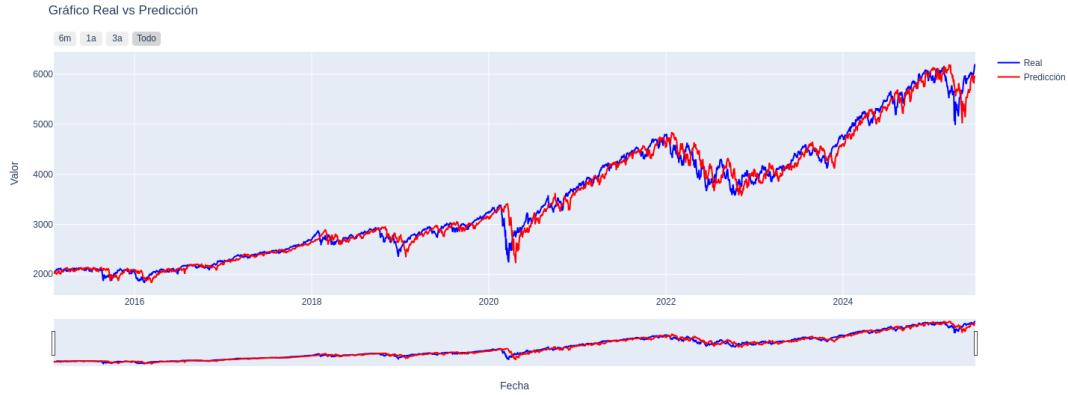


Figura 6.6.: Modelo LSTM Apple.

Como puede observarse, LSTM genera unas oscilaciones bruscas, incluso más violentas que las producidas por el modelo que utilizaba la regresión exponencial.

| Set | N | MSE | RMSE | MAE | MAPE |
|---------------|----------|------------|-------------|------------|-------------|
| Entrenamiento | 73437 | 0.0020 | 0.0451 | 0.0316 | 1.6408 % |
| Test | 15063 | 0.0024 | 0.0484 | 0.0353 | 1.7601 % |

Tabla 6.5.: Resultados de las métricas para el modelo LSTM para S&P 500.

6.1.2.2. Redes convolutivas

Véanse los resultados de las redes neuronales convolutivas.

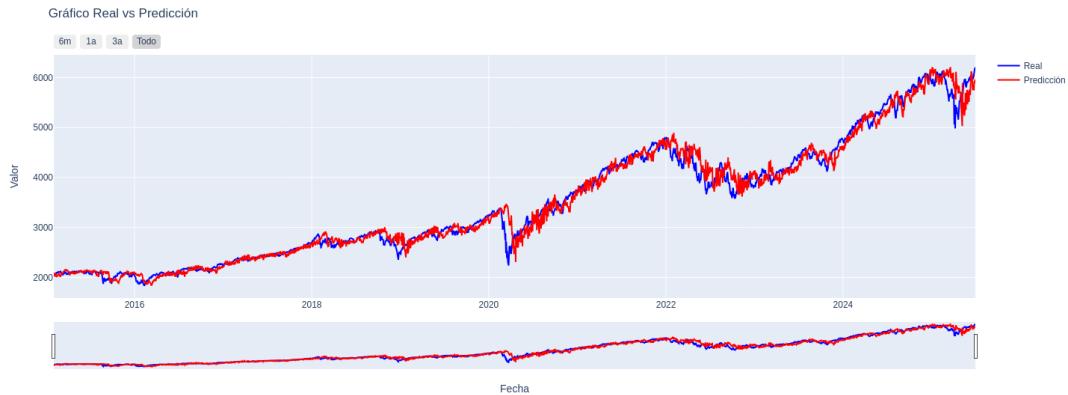


Figura 6.7.: Modelo CNN S&P 500.

6.1. Entrenamiento de modelos

Las redes convolutivas siguen siendo, a grandes rasgos, desplazamientos de la serie real al futuro.

| Set | N | MSE | RMSE | MAE | MAPE |
|---------------|----------|------------|-------------|------------|-------------|
| Entrenamiento | 73437 | 0.0021 | 0.0458 | 0.0320 | 2.3919 % |
| Test | 15063 | 0.0025 | 0.0498 | 0.0360 | 2.7884 % |

Tabla 6.6.: Resultados de las métricas para el modelo CNN para S&P 500.

6.1.2.3. Transformers

Por último, véanse los resultados de los recientes transformers.

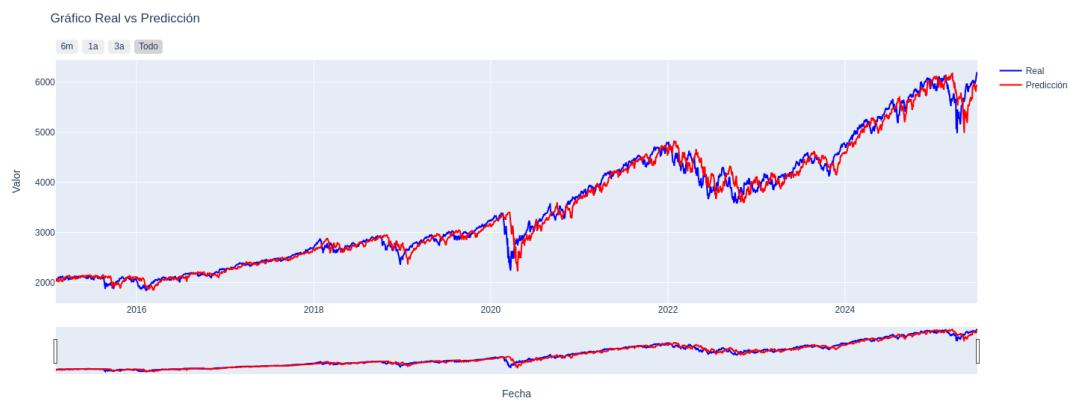


Figura 6.8.: Modelo Transformer S&P 500.

Como con los modelos anteriores, los modelos basados en transformers tampoco consigue realmente "predecir" la evolución futura del activo, sino que tienden a reproducir la dinámica pasada de la serie con cierto desfase temporal. Esta limitación está estrechamente ligada a su naturaleza autorregresiva: el modelo aprende a estimar el próximo valor únicamente a partir de observaciones pasadas, por lo que termina aproximando la continuidad del propio proceso temporal en lugar de anticipar cambios estructurales.

| Set | N | MSE | RMSE | MAE | MAPE |
|---------------|----------|------------|-------------|------------|-------------|
| Entrenamiento | 73437 | 0.0020 | 0.0452 | 0.0317 | 1.4968 % |
| Test | 15063 | 0.0024 | 0.0484 | 0.0354 | 1.5837 % |

Tabla 6.7.: Resultados de las métricas para el modelo Transformer para S&P 500.

6. Resultados experimentales

6.1.3. Tuning de hiperparámetros

Como se ha podido comprobar, los resultados obtenidos por los modelos distan de ser lo suficientemente buenos. Si el modelo lo que hace es, a grandes rasgos, replicar el comportamiento de la serie orginal en el futuro, la rotación de los activos dse va a limitar a replicar una estrategia convencional basada en momentos.

Se han observado los resultados de las distintas arquitecturas de redes neuronales, pero con los mismos hiperparámetros para todas ellas. Esto no es lo correcto. Cada arquitectura ha de tener unos hiperparámetros concretos con el fin de obtener los mejores resultados posibles. Sin embargo, ya se ha visto que las redes neuronales son, en esencia, cajas negras. Es por esto que la optimización de hiperparámetros no es una ciencia exacta.

Lo que se ha hecho es seguir lo visto en [Sección 4.4](#): ejecutar una batería de pruebas aleatorias para cada arquitectura y hacer una optimización bayesiana posteriormente. Los resultados pueden verse en [Apéndice B](#).

Se han añadido dos métricas más para todos estos experimentos: la correlación y el *hit rate*:

- Correlación: La correlación entre las predicciones y los valores reales mide cómo de bien el modelo capta la dirección de los movimientos. Si las predicciones del modelo mantienen una correlación alta (cercana a 1) indican que el modelo reproduce correctamente las tendencias, aunque no acierte en valores absolutos.
- *Hit rate*: La tasa de acierto indica qué porcentaje de veces el modelo predice correctamente el signo del movimiento (positivo o negativo). No se busca que el modelo acierte a la perfección el valor real, sino que sirva para establecer un ranking con el que rebalancear carteras. Para ello, el modelo ha de ser capaz de anticipar si el mercado va a subir o a bajar.

$$\text{hit_rate} = \frac{\sum_{i=1}^N \mathbf{1}\{\text{sgn}(\hat{y}_i) = \text{sgn}(y_i)\}}{N}$$

El modelo escogido como ganador ha sido el que tiene el nombre en las tablas:

h4_f64_d01.adam_hubert_e80_b32.t64x32.

Este modelo tiene los siguientes hiperparámetros:

- 2 capas ocultas con 64 y 32 neuronas respectivamente.
- El tamaño del lote (*batch_size*) de 32 muestras.
- El número de épocas ha sido de 80.
- ReLU ha sido la función de activación usada para las capas ocultas y lineal para la capa de salida.
- La función de pérdida usada ha sido Huber.
- El optimizador empleado ha sido Adam.
- Se ha usado una partición de validación del 10 %.

- Se han aplicado las siguientes técnicas de regularización:
 - *Dropout* con probabilidad 0.1.
 - *Early stopping* con paciencia de 8 épocas.
 - *Reduce on plateau* con factor 0.5 y paciencia de 4 épocas.
- El número de cabezas de atención ha sido 4 y la dimensión de la red feed-forward ha sido 64.

Los resultados (sobre datos de prueba) de este modelo son:

| N | MAE | RMSE | Corr | Hit Rate |
|------|--------|--------|--------|----------|
| 2748 | 0.0330 | 0.0441 | 0.1046 | 0.4014 |

Tabla 6.8.: Resultados de evaluación del modelo escogido.

Se ha escogido este modelo porque es el que tiene las métricas más balanceadas entre todos los entrenados. Si bien es cierto que algunos modelos LSTM presentan resultados similares, se ha optado por escoger un Transformer debido, principalmente, a la diferencia entre tiempos de entrenamiento (18 horas vs. 1 hora)¹.

6.1.4. Ingeniería de características: ¿importa la situación macro?

Los modelos entrenados han utilizado exclusivamente los valores históricos del propio activo como datos de entrada. Ahora bien, ¿qué sucede si se incorporan variables adicionales que puedan contener información relevante sobre el comportamiento futuro del activo?

Este proceso se conoce como ingeniería de características (*feature engineering*) y consiste en transformar, combinar o generar nuevas variables a partir de los datos originales con el objetivo de mejorar la capacidad predictiva del modelo.

En el contexto financiero, esto se traduce en incluir indicadores técnicos, variables que reflejen, por ejemplo, la volatilidad del mercado, o factores macroeconómicos (tipos de interés, datos de paro o liquidez global).

El propósito de esta práctica es dotar al modelo de una visión "más general" del entorno en el que se encuentra el activo a predecir. Aunque parezca que tiene sentido, puede que dar más información al modelo sea contraproducente, dando lugar a peores resultados que los obtenidos por el modelo que cuenta únicamente con los valores del propio activo. De hecho, esto es lo que dice Warren Buffet acerca de los aspectos macro (macroeconómicos) [46]:

La situación macro no importa. De hecho, es peligroso pensar en los aspectos macro ya que puede hacer que perdamos el foco en lo importante, la productividad futura del activo. Los tipos de interés, si la bolsa estaba cara o barata o cualquier

¹18 horas ha llegado a tardar un modelo LSTM con unos hiperparámetros concretos, no todos los modelos LSTM creados para este proyecto.

6. Resultados experimentales

aspecto político no harían que dejara de crecer amíz en Nebraska o que los estudiantes dejaran de ir a NYU. [...] Ignora los aspectos macro, son una distracción muy cara para los inversores. He visto mucha gente desperdiciar oportunidades de inversión por preocuparse demasiado de aspectos económicos o problemas que puede tener el país. Se olvidan de los aspectos positivos.

Las variables adicionales que se han considerado han sido:

- U.S. Dollar Index (DX-Y.NYB): Índice del Dólar estadounidense (US Dollar Index), que mide el valor del dólar frente a una cesta de divisas extranjeras. Es un indicador clave de la fortaleza del dólar y afecta a los precios de activos globales, especialmente materias primas y acciones de empresas exportadoras.
- Gold Futures (GC=F): Precio del oro (Gold Futures). El oro suele actuar como activo refugio, por lo que su evolución puede reflejar el sentimiento de riesgo de los mercados financieros.
- M₂ Money Stock (M₂NS): Agregado monetario M₂ de la Reserva Federal de Estados Unidos. Representa la cantidad total de dinero en circulación (efectivo, depósitos a corto plazo y fondos del mercado monetario), y es un indicador de la política monetaria y la liquidez del sistema financiero.
- CBOE 10-Year Treasury Note Yield Index (TNX): Rendimiento del bono del Tesoro estadounidense a 10 años (10-Year Treasury Yield). Es una referencia clave para los tipos de interés a largo plazo y las expectativas sobre crecimiento e inflación.
- CBOE Volatility Index (VIX): Índice de volatilidad del S&P 500 (Volatility Index). Refleja la volatilidad implícita en las opciones del S&P 500 y es conocido como el “índice del miedo” del mercado.
- Merrill Lynch Option Volatility Estimate (MOVE): Índice de volatilidad de los bonos del Tesoro estadounidense (ICE BofAML MOVE Index). Mide la volatilidad implícita en el mercado de renta fija y complementa la información del VIX sobre la percepción del riesgo.
- Total Nonfarm Payrolls (PAYEMS): Empleo total en el sector no agrícola de Estados Unidos (Nonfarm Payrolls). Es uno de los indicadores más importantes del mercado laboral y del ciclo económico.
- Bank Prime Loan Rate (DPRIME): Tipo de interés preferencial de los bancos estadounidenses (Prime Rate). Refleja las condiciones crediticias y el coste de financiación en la economía.

Los resultados de los experimentos realizados pueden hallarse, nuevamente, en el [Apéndice B](#). El mejor modelo ha sido el siguiente:

S&P 500 + VIX

Los resultados (sobre datos de prueba) de este modelo son:

| N | MAE | RMSE | Corr | Hit Rate |
|------|---------|---------|--------|----------|
| 2619 | 0.03549 | 0.04612 | 0.1585 | 0.3555 |

Tabla 6.9.: Resultados de evaluación del modelo escogido.

La correlación obtenida por este nuevo modelo es más de un 50 % mayor que la obtenida por el modelo base y únicamente se aumenta el error un 7.5 % para el MAE y un 4.5 % para el RMSE. Si bien la tasa de acierto direccional ha disminuido, el aumento de correlación compensa esta subida.

Ahora bien, estas métricas, ¿cómo de buenas son en comparación con otros modelos desarrollados en trabajos externos? En [50] y [51] se pueden ver correlaciones de 0.05 o MSE de 0.15. Aunque los experimentos realizados en estos trabajos son diferentes, pues no se entran los modelos de la misma forma (entre otros factores), permiten comparar los resultados obtenidos en este proyecto frente a otros trabajos que desarrollan modelos de inteligencia artificial en el ámbito bursátil.

Dejando atrás las comparaciones con otros trabajos, que pueden no resultar de interés por las diferencias en metodología y objetivo, véanse las mismas métricas para todos los modelos vistos hasta ahora para el S&P 500:

| Modelo | N | MAE | RMSE | Corr | Hit Rate |
|------------------------|------|---------|---------|----------|----------|
| Exponencial (Clenow) | 2619 | 0.04481 | 0.06318 | 0.01642 | 0.53769 |
| SVR | 2619 | 0.03558 | 0.04987 | 0.02489 | 0.61550 |
| Random Forest | 2619 | 0.03538 | 0.04751 | 0.07240 | 0.48377 |
| CNN | 2619 | 0.03786 | 0.05120 | -0.06434 | 0.44903 |
| LSTM | 2619 | 0.03520 | 0.04616 | 0.13952 | 0.37304 |
| RNN | 2619 | 0.03560 | 0.04672 | 0.06915 | 0.42574 |
| Transformer | 2619 | 0.03542 | 0.04641 | 0.10624 | 0.34135 |
| Transformer optimizado | 2748 | 0.0330 | 0.0441 | 0.1046 | 0.4014 |
| Transformer+VIX | 2619 | 0.03549 | 0.04612 | 0.1585 | 0.3555 |

Tabla 6.10.: Resultados de todos los modelos presentados.

Viendo los resultados, uno podría preguntarse: ¿ha merecido la pena la optimización y la ingeniería de características? Para poder responder a esa cuestión ha sido necesario llevar a cabo todo el proceso, lo cual ya constituye, en cierto modo, una respuesta. Además, la mejora tanto en el tiempo de entrenamiento (50 minutos frente a 14) como en la calidad del modelo es notable respecto al segundo mejor modelo, el LSTM sin ajuste de hiperparámetros. Puede que, en este proyecto concreto —en el que se han considerado poco menos de treinta activos—, el esfuerzo invertido para encontrar el "mejor modelo" no parezca compensar (aunque esta valoración se hace, como dice el refrán, a toro pasado). Sin embargo, en un proyecto profesional en el que se analicen miles de activos, una diferencia de 35 minutos en el entrenamiento puede resultar crucial.

Tras optimizar los hiperparámetros y realizar un proceso de ingeniería de características pa-

6. Resultados experimentales

ra el índice S&P 500, se procede a entrenar modelos con las mismas características para cada uno de los activos que se usarán para las simulaciones, los cuales se verán a continuación.

6.2. Simulación de carteras

Tras ver las predicciones de los distintos modelos para algunos activos, se muestran los resultados realmente importantes para el proyecto: el desempeño de las carteras que utilizan estas predicciones frente a las que usan la estrategia de Clenow.

En primer lugar, el universo de activos seleccionado para las simulaciones es el siguiente:

| Sector | Empresas |
|-----------------|--|
| Tech | Apple (AAPL), Microsoft (MSFT) |
| Bancos | Bank of America (BAC), Wells Fargo (WFC) |
| Aseguradoras | Berkshire Hathaway (BRK-B), UnitedHealth (UNH) |
| Entretenimiento | Disney (DIS), Warner Bros (WBD) |
| Petróleo y gas | Exxon (XOM), Chevron (CVX) |
| Farmacéuticas | Pfizer (PFE), Johnson & Johnson (JNJ) |
| Alimentación | Coca-Cola (KO), McDonald's (MCD) |
| Coches | General Motors (GM), Ford (F) |

Tabla 6.11.: Principales empresas por sector.

La estrategia que se ha seguido consiste en tomar dos acciones representativas de cada sector para que las simulaciones sean "más realistas". Sería beneficioso para Clenow tomar únicamente un universo de empresas tecnológicas puesto que la inmensa mayoría de ellas han tenido un crecimiento relativamente continuo en los últimos 10 años.

Como se ha mencionado antes, las simulaciones se realizan desde el año 2020 hasta mediados de 2025. Se ha tomado el marco temporal más amplio para intentar no ser ventajista tomando intervalos concretos en los que una estrategia puede ser más beneficiosa que otra.

Además de las acciones, se han hecho simulaciones utilizando fondos sectoriales (de los sectores de las empresas anteriores):

| Sector | ETF (Ticker) | Nombre completo |
|-----------------|--------------|--|
| Tech | QQQ | Invesco QQQ Trust |
| Bancos | XLF | Financial Select Sector SPDR Fund |
| Seguros | XLF | Financial Select Sector SPDR Fund |
| Entretenimiento | XLC | Communication Services Select Sector SPDR Fund |
| Petróleo y gas | XLE | Energy Select Sector SPDR Fund |
| Farmacéuticas | XLV | Health Care Select Sector SPDR Fund |
| Alimentación | XLP | Consumer Staples Select Sector SPDR Fund |
| Coches | XLI | Industrial Select Sector SPDR Fund |

Tabla 6.12.: ETFs representativos por sector

El último escenario es similar al anterior, pero incluye un conjunto más amplio de activos, incorporando tanto bonos como materias primas. De este modo, se dispone de un mayor número de instrumentos entre los que seleccionar al construir la cartera, lo que permite reducir la concentración del riesgo y mejorar la diversificación².

| ETF (Ticker) | Nombre completo |
|---------------------|--|
| CSPX | iShares Core S&P 500 UCITS ETF |
| IWDA | iShares MSCI World UCITS ETF |
| EEM | iShares MSCI Emerging Markets ETF |
| QQQ | Invesco QQQ Trust |
| XLF | Financial Select Sector SPDR Fund |
| XLE | Energy Select Sector SPDR Fund |
| XLV | Health Care Select Sector SPDR Fund |
| XLP | Consumer Staples Select Sector SPDR Fund |
| XLI | Industrial Select Sector SPDR Fund |
| GLD | SPDR Gold Shares |
| DBC | Invesco DB Commodity Index Tracking Fund |
| SHY | iShares 1–3 Year Treasury Bond ETF |
| IEF | iShares 7–10 Year Treasury Bond ETF |
| TIP | iShares TIPS Bond ETF |
| AGG | iShares Core U.S. Aggregate Bond ETF |
| RSP | Invesco S&P 500 Equal Weight ETF |
| USMV | iShares MSCI USA Min Vol Factor ETF |
| VYM | Vanguard High Dividend Yield ETF |
| FEZ | SPDR EURO STOXX 50 ETF |
| EWJ | iShares MSCI Japan ETF |

Tabla 6.13.: ETFs representativos de sectores y activos.

Por último, y como bien señalaba Clenow, en todos los experimentos se escogerá como índice de referencia el conocido S&P 500.

6.2.1. Clásicos contra Clenow

Para las simulaciones en las que se trabaje con acciones, la cartera se compondrá únicamente de cuatro acciones, aproximadamente el 20 % del número de acciones del universo considerado en [Tabla 6.11](#). En primer lugar, obsérvense los resultados de la simulación utilizando el primer modelo clásico de aprendizaje supervisado explicado: SVR

²En el ámbito de los mercados financieros, este proceso se conoce como *diversificación*.

6. Resultados experimentales

| Métrica | Simulación | Clenow | Benchmark (S&P 500) |
|------------------------|------------|----------|---------------------|
| Retorno acumulado | 27.33 % | 42.28 % | 94.36 % |
| Retorno anualizado | 4.64 % | 6.84 % | 13.28 % |
| Máxima caída | -25.02 % | -28.96 % | -24.77 % |
| Volatilidad anualizada | 95.24 % | 96.82 % | 77.86 % |
| Sharpe | 0.73 | 0.93 | 1.81 |
| Sortino | 1.00 | 1.54 | 2.44 |
| Rotación mensual media | 73.18 % | 44.13 % | — |

Tabla 6.14.: Métricas cartera acciones de simulación con SVR como modelo guía.



Figura 6.9.: Simulación cartera acciones con SVR como modelo guía.

Puede verse que, tanto Clenow como la cartera guiada por SVR no consiguen batir a su índice referencia: el S&P 500. Entre ambos, la rentabilidad obtenida por Clenow es mayor. Ambas simulaciones comienzan con pérdidas, esto es debido al COVID-19 en marzo de 2020. Ambas simulaciones comienzan con pérdidas, debido al impacto de la crisis del COVID-19 en marzo de 2020. No obstante, Clenow alcanza resultados muy favorables hasta 2023, superando al S&P 500. Esto se explica por el marcado período alcista entre 2020 y 2023, que resultó especialmente beneficioso para las estrategias basadas en momento.

| Métrica | Simulación | Clenow | Benchmark (S&P 500) |
|------------------------|------------|----------|---------------------|
| Retorno acumulado | 20.20 % | 42.28 % | 94.36 % |
| Retorno anualizado | 3.41 % | 6.63 % | 12.86 % |
| Máxima caída | -33.19 % | -28.96 % | -24.77 % |
| Volatilidad anualizada | 109.68 % | 96.82 % | 77.86 % |
| Sharpe | 0.61 | 0.93 | 1.81 |
| Sortino | 0.86 | 1.54 | 2.44 |
| Rotación mensual media | 63.85 % | 44.13 % | 0.00 % |

Tabla 6.15.: Métricas de simulación cartera acciones con random forest como modelo guía.

6.2. Simulación de carteras



Figura 6.10.: Simulación cartera acciones con random forest como modelo guía.

Los resultados obtenidos por random forest son incluso peores que SVR. Tras ver los resultados de los modelos clásicos de aprendizaje supervisado, ¿habrá merecido la pena el tuning de hiperparámetros y la ingeniería de características?

6.2.2. Redes neuronales contra Clenow

Acciones

En primer lugar, veáñense los resultados con el universo de activos correspondiente a la [Tabla 6.11](#), el cual está compuesto únicamente por acciones.

| Métrica | Simulación | Clenow | Benchmark (S&P 500) |
|------------------------|------------|----------|---------------------|
| Retorno acumulado | 93.98 % | 42.28 % | 94.36 % |
| Retorno anualizado | 13.24 % | 6.84 % | 13.28 % |
| Máxima caída | -34.64 % | -28.96 % | -24.77 % |
| Volatilidad anualizada | 120.83 % | 96.82 % | 77.86 % |
| Sharpe | 1.36 | 0.93 | 1.81 |
| Sortino | 1.90 | 1.54 | 2.44 |
| Rotación mensual media | 63.51 % | 44.13 % | 0.00 % |

Tabla 6.16.: Métricas de simulación cartera acciones con transformer+VIX como modelo guía.

6. Resultados experimentales

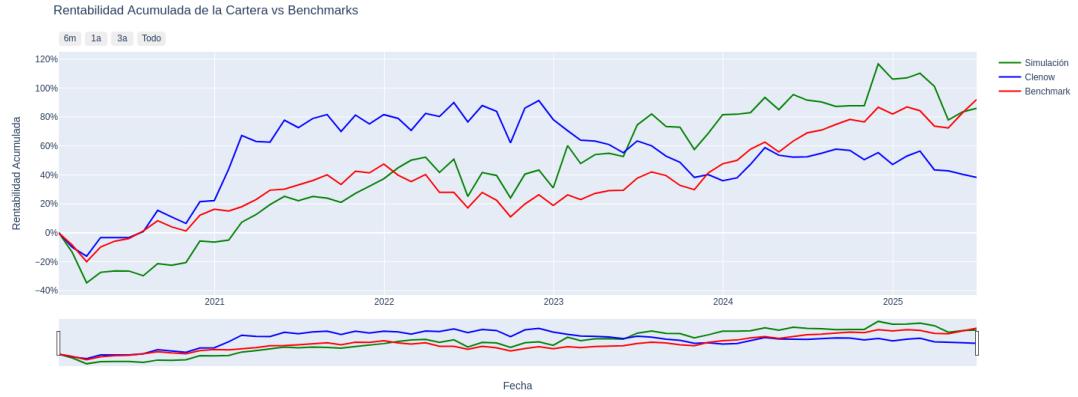


Figura 6.11.: Simulacion cartera acciones con modelos transformer refinados como guía.

La cartera guiada por el modelo transformer junto con el índice de volatilidad VIX ha conseguido unos resultados significativamente mejores que los obtenidos por SVR y random forest. Además, ha batido a la estrategia de Clenow y ha conseguido una rentabilidad prácticamente igual a la obtenida por el S&P 500.

De hecho, al considerar cinco activos en lugar de cuatro y optar por mantener el dinero en efectivo cuando las predicciones para todos los activos del universo seleccionado son negativas, tanto los resultados obtenidos por Clenow como los del modelo transformer mejoran:

| Métrica | Simulación | Clenow | Benchmark (S&P 500) |
|------------------------|------------|----------|---------------------|
| Retorno acumulado | 100.89 % | 64.09 % | 94.36 % |
| Retorno anualizado | 13.54 % | 9.43 % | 12.86 % |
| Máxima caída | -33.51 % | -22.57 % | -24.77 % |
| Volatilidad anualizada | 112.45 % | 89.68 % | 77.86 % |
| Sharpe | 1.47 | 1.27 | 1.81 |
| Sortino | 2.05 | 2.08 | 2.44 |
| Rotación mensual media | 62.04 % | 40.56 % | 0.00 % |

Tabla 6.17.: Comparativa de métricas entre la simulación, el método de Clenow y el índice S&P 500.

6.2. Simulación de carteras

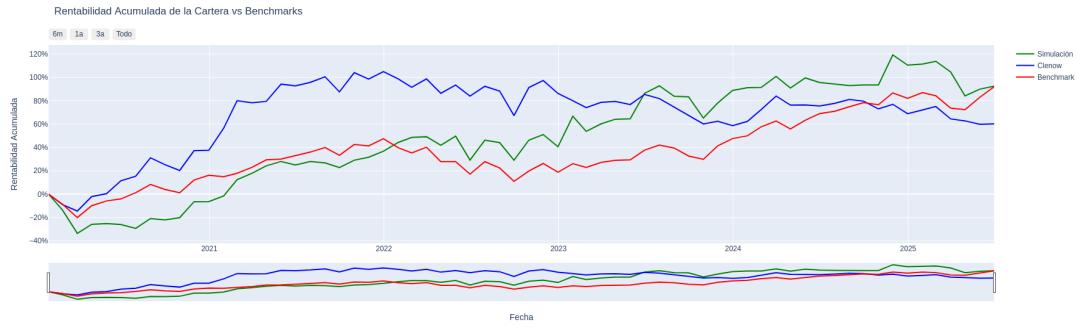


Figura 6.12.: Simulación con modelos transformer con 5 activos y efectivo como refugio.

Índices y ETFs

A continuación, se presentan los resultados obtenidos para ambos universos de simulación: por un lado, las carteras compuestas por dos activos seleccionados de entre los índices sectoriales de la [Tabla 6.12](#), y por otro, las carteras formadas por siete activos, escogidos del universo más amplio que incluye tanto bonos como materias primas, [Tabla 6.13](#). En ambos casos, el activo refugio será el dinero en efectivo. En primer lugar, se presentan los resultados del segundo universo:

| Métrica | Simulación | Clenow | Benchmark (S&P 500) |
|------------------------|------------|----------|---------------------|
| Retorno acumulado | 51.09 % | 65.70 % | 94.36 % |
| Retorno anualizado | 7.80 % | 9.63 % | 12.86 % |
| Máxima caída | -22.36 % | -16.28 % | -24.77 % |
| Volatilidad anualizada | 57.86 % | 55.98 % | 77.86 % |
| Sharpe | 1.49 | 1.84 | 1.81 |
| Sortino | 1.98 | 2.57 | 2.44 |
| Rotación mensual media | 69.63 % | 37.38 % | 0.00 % |

Tabla 6.18.: Comparativa de métricas de rendimiento entre la simulación, Clenow y el S&P 500 como benchmark.

6. Resultados experimentales



Figura 6.13.: Simulación cartera índices mundiales con modelos transformers como guía.

Los resultados han resultado ser peores que los obtenidos por la cartera formada únicamente por acciones, tanto Clenow como la que utiliza los modelos transformers. De hecho, ninguna de las dos estrategias bate al S&P 500. Véanse ahora los resultados con un universo más concentrado, tomando únicamente los ETFs sectoriales.

| Métrica | Simulación | Clenow | Benchmark (S&P 500) |
|------------------------|------------|----------|---------------------|
| Retorno acumulado | 141.26 % | 59.47 % | 94.36 % |
| Retorno anualizado | 17.38 % | 8.86 % | 12.86 % |
| Máxima caída | -27.51 % | -15.76 % | -24.77 % |
| Volatilidad anualizada | 89.08 % | 75.9 % | 77.86 % |
| Sharpe | 2.09 | 1.35 | 1.81 |
| Sortino | 3.13 | 2.08 | 2.44 |
| Rotación mensual media | 73.33 % | 45.2 % | 0.0 % |

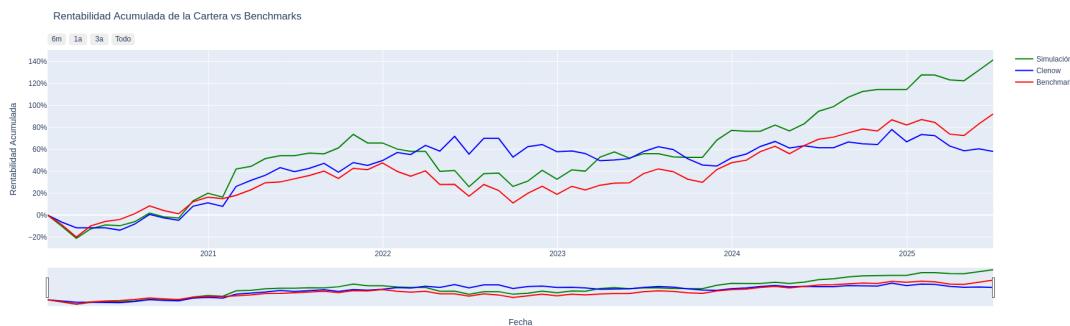


Figura 6.14.: Simulación cartera índices sectoriales con modelos transformers como guía.

6.2. Simulación de carteras

En este caso, los resultados son claramente mejores para la cartera que utiliza el modelo transformer como guía frente a la estrategia de Clenow. Al emplear únicamente índices sectoriales, la estrategia alcanza un equilibrio más eficiente entre diversificación y concentración: la exposición está repartida entre varios sectores relevantes, pero sin diluir en exceso el impacto de cada activo. En cambio, el universo más amplio que incluía bonos y materias primas introducía activos con dinámicas menos correlacionadas con las acciones, lo que reducía la sensibilidad al ciclo bursátil pero también limitaba el potencial de rentabilidad.

7. Conclusiones y líneas de trabajo futuras

Conclusiones

En este Trabajo Final de Grado se ha podido profundizar en las matemáticas detrás de muchos de los principales modelos de machine learning: SVR, randomforest y distintos tipos de arquitecturas de redes neuronales. Dichos modelos, han sido aplicados al mundo de los mercados bursátiles, un campo que muchos tachan de impredecible¹. Entre ellos, el padre del análisis fundamental y mentor de Warren Buffett, Benjamin Graham: «*El futuro de los precios de los valores nunca es predecible»*[14].

Los modelos han sido utilizados para construir carteras y comparar sus resultados con los obtenidos por las carteras basadas en la inversión por momentos. Se ha podido comprobar que los rendimientos de nuestras carteras han sido mejores, en la mayoría de los casos, que los obtenidos por las creadas por la estrategia de Andreas F. Clenow.

Además, se ha creado una aplicación web que permite al usuario poder consultar los resultados (predicciones y métricas) de todos los modelos entrenados y construir simulaciones usando estos modelos, configurando dicha simulación a su gusto.

En este trabajo he podido aprender los fundamentos de numerosos modelos de aprendizaje automático, los cuales no he visto en la carrera. He podido indagar en cómo han de entrenarse los modelos correctamente cuando se usan series temporales y cómo mejorar los modelos mediante distintas técnicas de optimización.

Además, toda mi aplicación ha sido desarrollada sobre un proyecto maduro ya creado, lo que ha supuesto un desafío ya que he tenido que comprender adecuadamente el código ya escrito para poder añadir las clases y sistemas requeridos por mi proyecto.

He podido combinar las matemáticas y la informática con la bolsa y los mercados financieros, una de mis pasiones. De hecho, este trabajo me ha ayudado a asimilar muchos conceptos financieros.

Líneas de trabajo futuras

El principal objetivo de este trabajo fue comparar las estrategias basadas en momentum con aquellas que emplean modelos de aprendizaje automático. En concreto, se analizó una estrategia fundamentada en el momentum investing frente a diversas estrategias construidas mediante distintos tipos de modelos de machine learning. No solo se llevó a cabo una

¹Siguiendo este pensamiento, en este trabajo se ha intentado "predecir lo impredecible", lo cual es un oxímoron.

7. Conclusiones y líneas de trabajo futuras

comparación entre ambas aproximaciones, sino que los resultados mostraron que la estrategia desarrollada en este proyecto superó a la basada en momentos.

No obstante, este trabajo presenta varias posibles líneas de mejora:

- Modelos basados en datos fundamentales. En lugar de predecir únicamente en función del precio de cotización del activo, podrían desarrollarse modelos que se basen en sus datos fundamentales. Por ejemplo, en el caso de las acciones, el modelo podría incorporar como variables de entrada los estados financieros de la empresa o sus informes de resultados.
- Optimización específica por activo. Todo el proceso para encontrar el mejor modelo se ha realizado sobre el índice S&P 500, sin embargo, la mejor práctica sería hacer este mismo proceso para cada uno de los activos del universo.
- Ampliación del universo de activos. Los universos empleados en este trabajo han sido relativamente reducidos. Para mejorar la robustez de los resultados, sería conveniente ampliar el conjunto de activos a cientos o incluso miles, entrenando los modelos correspondientes para cada caso.
- Incorporación de factores macroeconómicos. Podrían utilizarse modelos capaces de identificar el contexto macroeconómico con el fin de mejorar las predicciones. Por ejemplo, la curva de tipos podría emplearse para anticipar posibles recesiones económicas². Asimismo, algunos economistas sostienen que los mercados son cíclicos; entre los enfoques más conocidos destacan los ciclos de Juan Ramón Rallo, las ondas de Kondratieff o el reloj de inversión de Merrill Lynch. Se podría, por tanto, desarrollar un modelo que determine en qué fase del ciclo económico se encuentra el mercado.
- Despliegue de la aplicación en un entorno accesible. En este proyecto, la aplicación se ha desarrollado y ejecutado localmente en el ordenador personal del autor. Una posible mejora consistiría en contenerizar la aplicación y desplegarla en un dominio web, de modo que pueda ser accesible públicamente y utilizada por cualquier usuario.

²En los últimos años, sin embargo, la curva de tipos ha perdido capacidad predictiva debido a la manipulación de su tramo corto.

A. Mercados financieros

Este trabajo se ha desarrollado en el marco de los mercados financieros y el mundo de la bolsa. Sin embargo, no todas las personas están familiarizadas con los términos bursátiles que se han utilizado en la memoria. En este apéndice se pretenden explicar los conceptos básicos para todos aquellos que se pierdan al escuchar palabras como 'bolsa', 'acciones' o 'carteras'.

A.1. ¿Qué es la bolsa?

Según la [RAE](#) (Real Academia Española), la bolsa es una "institución económica donde se efectúan transacciones públicas de compra y venta de valores, y otras operaciones análogas". Dicho de otra forma, la bolsa es un mercado donde se compran y venden activos financieros. Los más comunes son las acciones y los bonos.

Los emisores habituales de este tipo de activos son las empresas y Estados. Los que compran son los conocidos como «inversores». Además, existen intermediarios encargados de conectar compradores y vendedores para que puedan efectuar las operaciones. Estos intermediarios son los bancos y *brokers*.

Gracias a la bolsa los vendedores consiguen financiación y los compradores *intentan* ganar dinero. El matiz es importante, ya que invirtiendo en bolsa no siempre se gana dinero.

Esto es debido a que los activos tienen un precio determinado por la ley de la oferta y la demanda. Esto trae consigo que los precios fluctúen conforme se compre más o menos. Cuanto más se compre, más sube el precio debido a que la demanda aumenta y, por el contrario, cuanto menos se compre más baja el precio porque la demanda disminuye.

En la bolsa se gana dinero de dos formas: plusvalías y dividendos.

- Plusvalía: Cuando se compra un activo a un precio y lo vende posteriormente a un precio mayor. Por ejemplo, si uno compra una acción a 100 euros y la vende luego a 120, la plusvalía es de 20 euros.
- Dividendos: Son los beneficios que algunas empresas reparten periódicamente a los que tienen acciones de dichas empresas, llamados accionistas.

El problema llega cuando se compra un activo cuyo valor empieza a caer. En ese momento, el inversor particular (o *retail*) tiene dos opciones: o mantener su activo esperando que el precio vuelva a subir (lo que se llama *holdear*), o darse por vencido y vender, perdiendo dinero.

Puede que el lector haya oído hablar de casos en los que personas corrientes comienzan a

A. Mercados financieros

invertir y, casi de la noche a la mañana, parecen volverse millonarias. Sin embargo, este es un ejemplo clásico del problema de las pruebas silenciosas, formulado por Nassim Taleb en *Antifrágil* [52].

Este problema se entiende mucho mejor con la historia que contó el romano Cicerón hace más de 2 mil años:

A un tal Diágoras, que no creía en los dioses, le presentaron unas tablillas de unos fieles que sobrevivieron a un naufragio rezando.

Lo que se deducía de estas tablillas es que los dioses te salvarían de los males si uno tenía fe y oraba. Al leer las tablillas Diágoras preguntó: «¿Y dónde están las tablillas de quienes oraron y se ahogaron?». Supongo que nunca lo sabremos pues los muertos no hablan...

Lo mismo sucede con la inversión en bolsa: se habla de quienes amasan fortunas, pero raramente de sus errores —y de las pérdidas que los acompañaron—, ni mucho menos de la multitud de inversores que depositaron sus esperanzas en los mercados financieros y fracasaron estrepitosamente.

Consecuentemente, uno debe ser consciente de en qué invierte y qué hace con su dinero. Como dijo Benjamin Graham, padre del *value investing* y mentor del considerado mejor inversor de la historia, Warren Buffett: «invertir sin saber en qué se está invirtiendo no es invertir, es especular».

A.1.1. Acciones

Haciendo alusión nuevamente a la RAE, una acción es un "título valor que representa una parte alícuota en el capital de una sociedad mercantil y que da derecho a una parte proporcional en el reparto de beneficios y a la cuota patrimonial correspondiente en la disolución de la sociedad." Una acción es, básicamente, una parte de la propiedad de la empresa.

Surgen tres preguntas a la hora de embarcarse en el mundo bursátil:

- ¿Qué empresa comprar?
- ¿Cuándo comprar una acción?
- ¿Cuándo vender una acción?

La respuesta a estas preguntas ha llevado numerosos libros (a algunos de los cuales se ha hecho referencia en este proyecto), cursos de formación y empresas dedicadas a conseguir las mayores plusvalías posibles.

En este proyecto se han visto algunas estrategias que permiten contestar a estas preguntas a partir de ciertos valores y parámetros. Sin embargo, ha de mencionarse que hay numerosas personas cuya profesión es el estudio de compañías y del mercado. Estos inversores profesionales no se limitan a observar unos pocos indicadores, sino que estudian en detalle aspectos como la contabilidad de la empresa, su situación competitiva, la calidad de su equipo directivo, su sector y otras variables relevantes antes de tomar decisiones de inversión.

Esto último es conocido como "análisis fundamental", el cual se centra en el valor intrínseco de la empresa. Por otro lado el "análisis técnico" únicamente se basa en el estudio del precio y del volumen negociado, con la finalidad de identificar patrones y tendencias en los mercados.

Dado que, como se ha visto, los modelos desarrollados en este proyecto trabajan con las series temporales del valor de distintos activos y acciones, podría decirse que se enmarcan dentro del análisis técnico.

A.1.2. Índices, fondos y ETFs

Hasta ahora se ha hablado de invertir en acciones de empresas concretas. Sin embargo, no es la única forma de participar en los mercados financieros. Existen instrumentos diseñados para invertir de forma diversificada, reduciendo el riesgo que conlleva apostar por una sola compañía. Entre ellos destacan los índices bursátiles, los fondos de inversión y los llamados *ETFs*.

Índices bursátiles

Un índice bursátil es un indicador que refleja la evolución conjunta de un conjunto de empresas seleccionadas según ciertos criterios. Se trata, por tanto, de una medida que permite conocer cómo se comporta un mercado o un sector en su conjunto.

Algunos ejemplos conocidos son:

- S&P 500: recoge las 500 mayores empresas de Estados Unidos por capitalización bursátil.
- IBEX 35: agrupa a las 35 empresas más relevantes de España.
- NASDAQ-100: centrado principalmente en empresas tecnológicas estadounidenses.

Cuando en los medios se afirma que "el mercado sube un 2%", normalmente se hace referencia a que un índice de referencia como el S&P 500 ha aumentado ese porcentaje. De hecho, en este proyecto se ha trabajado con este índice americano tanto para desarrollar los modelos generales, como benchmark.

Fondos de inversión

Un fondo de inversión es un vehículo que agrupa el dinero de muchos inversores con el objetivo de que un equipo profesional lo gestione comprando distintos activos. Dicho de otra forma, muchos pequeños inversores ponen su dinero en común para tener acceso a una cartera diversificada gestionada por expertos.

Existen principalmente dos tipos de fondos:

- Fondos de gestión activa: el equipo gestor selecciona los activos intentando superar a un índice de referencia (*benchmark*). Es decir, tratan de obtener una rentabilidad mayor

A. Mercados financieros

que el mercado mediante análisis y toma de decisiones. Por este motivo, suelen tener comisiones más elevadas.

- Fondos de gestión pasiva: buscan replicar el comportamiento de un índice concreto, como el S&P 500 o el IBEX 35. No intentan batir al mercado, sino igualarlo. Al requerir menos intervención humana, sus comisiones suelen ser más bajas. Como se ha indicado al principio de la memoria, estos fondos fueron popularizados por el fundador de *Vanguard*, John Bogle. Además, en su libro *el pequeño libro para invertir con sentido común*[6] habla sobre las ventajas de invertir en fondos de gestión pasiva frente a los de gestión activa.

ETFs (*Exchange-Traded Funds*)

Los *Exchange-Traded Funds*, más conocidos como ETFs, son fondos de inversión que cotizan en bolsa igual que una acción. Es decir, pueden comprarse y venderse durante el horario bursátil, con un precio que varía en tiempo real.

Lo habitual es que un ETF replique un índice bursátil. Por ejemplo, en lugar de comprar las 500 empresas que componen el S&P 500 (tarea que sería compleja y costosa), un inversor puede comprar un único ETF que imite dicho índice y obtendrá prácticamente el mismo rendimiento.

Los ETFs suelen tener dos ventajas principales:

- Diversificación instantánea: con una sola compra se obtiene exposición a decenas o cientos de empresas.
- Bajas comisiones: suelen ser inferiores a las de los fondos tradicionales, especialmente en los llamados ETFs indexados.

Conviene mencionar una diferencia relevante en el contexto español: la traspasabilidad fiscal de los fondos de inversión. En España, un inversor puede trasladar su dinero de un fondo de inversión a otro sin tener que tributar por las plusvalías obtenidas en el proceso, siempre que no se produzca un reembolso a su cuenta bancaria. Esto permite ir ajustando la cartera a lo largo del tiempo sin generar un coste fiscal inmediato.

Es importante destacar que, en general, los ETFs no disfrutan de esta ventaja en España.¹ En la práctica, esto hace que, desde un punto de vista fiscal, los fondos de inversión resulten más eficientes para el inversor español a largo plazo, mientras que los ETFs suelen ser preferidos por su liquidez y menores comisiones.

En resumen, mientras que invertir en acciones individuales implica elegir empresas concretas –con el riesgo asociado a equivocarse–, los índices, fondos y ETFs permiten acceder al mercado de forma más diversificada. Cada inversor puede elegir el instrumento que mejor se adapte a sus objetivos, conocimientos y tolerancia al riesgo.

¹Existen algunas excepciones específicas recogidas por la normativa, pero no son la norma general.

B. Resultados del entrenamiento de modelos

En este apéndice pueden encontrarse los resultados numéricos de todas las pruebas que se han hecho para la selección de los hiperparámetros de los modelos y también para la incorporación de variables adicionales para mejorar el rendimiento de los modelos autorregresivos (la ingeniería de características descrita en [Subsección 6.1.4](#)).

B.1. Hiperparámetros

A continuación, se exponen los resultados de las distintas pruebas que se han hecho para encontrar la mejor combinación de hiperparámetros, así como las métricas resultantes de cada experimento.

La primera página corresponde a los resultados de la arquitectura LSTM. Nótese que hay menos experimentos para esta arquitectura por limitaciones de hardware y tiempo ya que cada experimento ha llegado a tardar más de 18 horas.

La segunda página corresponde a los resultados de la arquitectura CNN y la tercera a la arquitectura Transformer. Hay más experimentos para estas arquitecturas pues sus tiempos de entrenamiento son considerablemente inferiores.

Los primeros experimentos se han logrado mediante la técnica de búsqueda aleatoria, los finales mediante optimización bayesiana. Tanto para LSTM como para Transformer se distinguen claramente al tener un número de datos distinto. Para transformer aproximadamente la mitad es aleatoria y la segunda mitad es bayes.

Nótese que, al no haberse realizado un número mayor de pruebas, los resultados de la optimización bayesiana no son lo buenos que podrían ser, mejorando ligeramente los resultados de método aleatorio en algunos casos.

Por último, la métrica R^2 que aparece en las tablas no es la que se ha definido al principio de la memoria ([2.2](#)). Este coeficiente de correlación se define de la siguiente forma:

$$R_{\text{oos}}^2 = 1 - \frac{\sum_{i=1}^n (y_i^{\text{true}} - \hat{y}_i)^2}{\sum_{i=1}^n (y_i^{\text{true}} - \bar{y}^{\text{true}})^2},$$

donde y_i^{true} son los rendimientos reales fuera de muestra, \hat{y}_i las predicciones del modelo y \bar{y}^{true} la media de los valores reales.

B. Resultados del entrenamiento de modelos

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|--|------|--------|--------|----------------|---------|----------|
| t128x64.adam_hubert_e120_b32_dooo.bn1.bi1.ldo2.h32 | 2496 | 0.0344 | 0.0458 | -0.0977 | 0.1085 | 0.344 |
| t64.adam_hubert_e120_b16_doo2.bn1.bi0.ldo2.ho | 2496 | 0.0337 | 0.0451 | -0.0674 | 0.1011 | 0.382 |
| lstm.t64.adam_hubert_e120_b16_doo1.rdo0.bn0.bi0.ldo2.ho | 2496 | 0.0339 | 0.0463 | -0.1239 | -0.1331 | 0.371 |
| lstm.t64.adam_hubert_e120_b16_doo1.rdo0.bn0.bi0.ldo2.ho | 2496 | 0.0332 | 0.0448 | -0.0536 | 0.1147 | 0.424 |
| t64.adam_hubert_e120_b32_doo2.bn1.bi1.ldo2.ho | 2496 | 0.0383 | 0.0508 | -0.3541 | 0.0094 | 0.473 |
| t64x32.adam_hubert_e120_b16_dooo.bn0.bi1.ldo0.h32 | 2496 | 0.0359 | 0.0475 | -0.1803 | 0.0142 | 0.474 |
| t128x64.adam_hubert_e120_b16_dooo.bn0.bi0.ldo1.bn0.bi0.ldo2.ho | 2496 | 0.0343 | 0.0456 | -0.0906 | 0.0792 | 0.372 |
| t128x64.adam_hubert_e120_b32_doo2.bn1.bi0.ldoo.bn1.bi0.h32 | 2748 | 0.0430 | 0.0580 | -0.8626 | 0.0049 | 0.482 |
| t64.adam_hubert_e120_b32_doo2.bn0.bi1.ldo2.h64_32 | 2748 | 0.0337 | 0.0452 | -0.1337 | -0.1363 | 0.332 |
| t64.adam_hubert_e120_b32_doo2.rdo0.bn0.bi0.ldo2.h32 | 2748 | 0.0343 | 0.0451 | -0.1274 | 0.0341 | 0.331 |

Tabla B.1.: Resultados de pruebas LSTM sobre el S&P 500 (configuraciones y métricas)

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|--|------|--------|--------|----------------|---------|----------|
| t64x64x32.adam_hubert_e120_b32.dilated | 2496 | 0.0351 | 0.0473 | -0.1698 | -0.0202 | 0.416 |
| tr28x64.adam_hubert_e120_b16.wide | 2496 | 0.0336 | 0.0453 | -0.0770 | -0.0001 | 0.406 |
| t64x32.adam_mse_e120_b32.deep | 2496 | 0.0341 | 0.0457 | -0.0947 | -0.0023 | 0.343 |
| tr28x64.adam_hubert_e120_b16.lite | 2496 | 0.0341 | 0.0457 | -0.0943 | -0.0125 | 0.343 |
| t64x32.adam_hubert_e120_b32.wide | 2496 | 0.0349 | 0.0469 | -0.1523 | -0.0989 | 0.380 |
| t64x32.adam_hubert_e120_b32.dilated | 2496 | 0.0355 | 0.0478 | -0.1960 | -0.0137 | 0.430 |
| t64x64x32.adam_hubert_e120_b16.deep | 2496 | 0.0343 | 0.0458 | -0.1000 | -0.0322 | 0.343 |
| t64x64x32.adam_mse_e120_b32.wide | 2496 | 0.0349 | 0.0477 | -0.1936 | -0.0710 | 0.397 |
| t64x32.adam_mse_e120_b32.lite | 2496 | 0.0351 | 0.0477 | -0.1924 | -0.0899 | 0.401 |
| tr28x64.adam_hubert_e80_b32.dilated | 2496 | 0.0352 | 0.0477 | -0.1901 | -0.1089 | 0.372 |
| t64x32.adam_hubert_e120_b16.wide | 2496 | 0.0340 | 0.0461 | -0.1113 | -0.0952 | 0.414 |
| t64x64x32.adam_hubert_e80_b32.lite | 2496 | 0.0334 | 0.0453 | -0.0732 | -0.0002 | 0.446 |
| t64x64x32.adam_hubert_e120_b32.dilated | 2496 | 0.0355 | 0.0486 | -0.2372 | -0.0275 | 0.449 |
| t64x64x32.adam_hubert_e120_b16.lite | 2496 | 0.0345 | 0.0461 | -0.1112 | -0.0141 | 0.346 |
| tr28x64.adam_hubert_e120_b32.dilated | 2496 | 0.0354 | 0.0484 | -0.2261 | 0.0255 | 0.432 |
| t64x32.adam_hubert_e80_b32.lite | 2496 | 0.0337 | 0.0456 | -0.0892 | -0.0455 | 0.414 |
| tr28x64.adam_hubert_e120_b32.lite | 2496 | 0.0344 | 0.0466 | -0.1357 | -0.0762 | 0.405 |
| t64x64.adam_hubert_e80_b32.dilated | 2496 | 0.0354 | 0.0480 | -0.2075 | -0.0244 | 0.428 |
| tr28x64.adam_hubert_e120_b32.deep | 2496 | 0.0342 | 0.0457 | -0.0960 | 0.0009 | 0.343 |
| t64x32.adam_hubert_e80_b32.deep | 2496 | 0.0337 | 0.0454 | -0.0807 | -0.0051 | 0.343 |
| t64x64x32.adam_mse_e120_b32.dilated | 2748 | 0.0350 | 0.0469 | -0.2172 | 0.0517 | 0.396 |
| t64x32.adam_hubert_e120_b32.wide | 2748 | 0.0340 | 0.0455 | -0.1488 | -0.0498 | 0.349 |
| t64x64x32.adam_mse_e120_b32.wide | 2748 | 0.0359 | 0.0474 | -0.2460 | -0.0429 | 0.354 |
| t64x32.adam_hubert_e120_b32.lite | 2748 | 0.0342 | 0.0458 | -0.1645 | -0.0781 | 0.376 |
| t64x32.adam_mse_e120_b32.dilated | 2748 | 0.0364 | 0.0488 | -0.3200 | 0.0063 | 0.405 |
| tr28x64.adam_hubert_e120_b16.dilated | 2748 | 0.0333 | 0.0444 | -0.0919 | 0.0355 | 0.401 |
| t64x32.adam_hubert_e120_b32.deep | 2748 | 0.0338 | 0.0448 | -0.1134 | 0.0034 | 0.347 |
| t64x32.adam_hubert_e120_b32.dilated | 2748 | 0.0350 | 0.0471 | -0.2296 | -0.0793 | 0.382 |
| t64x64x32.adam_mse_e120_b32.dilated | 2748 | 0.0345 | 0.0465 | -0.1987 | 0.0095 | 0.425 |
| t64x32.adam_hubert_e120_b32.wide | 2748 | 0.0340 | 0.0454 | -0.1425 | -0.0437 | 0.410 |
| t64x64x32.adam_mse_e120_b32.wide | 2748 | 0.0353 | 0.0474 | -0.2447 | -0.0342 | 0.386 |

Tabla B.2.: Resultados de pruebas de CNN sobre el S&P 500 (configuraciones y métricas)

B. Resultados del entrenamiento de modelos

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|---|------|--------|--------|----------------|--------|----------|
| h2_f64_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0338 | 0.0446 | -0.1039 | 0.0748 | 0.342 |
| h2_f128_d02.adam_hubert_e120_b32.t64x32 | 2748 | 0.0340 | 0.0447 | -0.1064 | 0.1413 | 0.332 |
| h4_f128_d02.adam_hubert_e120_b32.t64x32 | 2748 | 0.0336 | 0.0444 | -0.0943 | 0.1051 | 0.345 |
| h2_f128_d01.adam_mse_e120_b32.t64x32 | 2748 | 0.0340 | 0.0448 | -0.1096 | 0.1106 | 0.333 |
| h2_f128_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0340 | 0.0447 | -0.1070 | 0.1875 | 0.331 |
| h4_f128_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0340 | 0.0447 | -0.1094 | 0.2047 | 0.330 |
| h8_f256_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0336 | 0.0445 | -0.0955 | 0.1511 | 0.334 |
| h4_f128_d01.adam_hubert_e120_b32.t64x32 | 2748 | 0.0339 | 0.0447 | -0.1058 | 0.1122 | 0.333 |
| h4_f128_d02.adam_mse_e120_b32.t64x32 | 2748 | 0.0334 | 0.0443 | -0.0866 | 0.1021 | 0.358 |
| h4_f64_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0330 | 0.0441 | -0.0761 | 0.1046 | 0.401 |
| h2_f64_d02.adam_hubert_e120_b32.t64x32 | 2748 | 0.0339 | 0.0447 | -0.1052 | 0.1471 | 0.331 |
| h8_f128_d02.adam_hubert_e120_b32.t64x32 | 2748 | 0.0340 | 0.0447 | -0.1063 | 0.1244 | 0.342 |
| h4_f64_d01.adam_hubert_e120_b16.t64x32 | 2748 | 0.0332 | 0.0442 | -0.0822 | 0.1025 | 0.368 |
| h8_f128_d01.adam_mse_e120_b32.t64x32 | 2748 | 0.0343 | 0.0449 | -0.1183 | 0.1079 | 0.334 |
| h2_f128_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0337 | 0.0445 | -0.0969 | 0.1213 | 0.345 |
| h2_f64_d01.adam_mse_e120_b32.t64x32 | 2748 | 0.0333 | 0.0443 | -0.0864 | 0.1318 | 0.338 |
| h2_f64_d01.adam_hubert_e120_b16.t64x32 | 2748 | 0.0336 | 0.0445 | -0.0957 | 0.1139 | 0.339 |
| h4_f64_d02.adam_hubert_e120_b32.t64x32 | 2748 | 0.0342 | 0.0449 | -0.1175 | 0.1591 | 0.329 |

Tabla B.3.: Resultados de pruebas Transformer sobre el S&P 500 (configuraciones y métricas) – Parte 1 de 2

| Nombre | n | MAE | RMSE | R^2 | corr | hit_rate |
|---|------|--------|--------|---------|--------|----------|
| h8_f128_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0331 | 0.0441 | -0.0795 | 0.1144 | 0.352 |
| h4_f128_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0335 | 0.0444 | -0.0917 | 0.1482 | 0.341 |
| h8_f256_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0338 | 0.0446 | -0.1022 | 0.1069 | 0.343 |
| h2_f128_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0336 | 0.0445 | -0.0958 | 0.1379 | 0.340 |
| h2_f128_d01.adam_hubert_e120_b32.t64x32 | 2748 | 0.0338 | 0.0447 | -0.1048 | 0.0853 | 0.336 |
| h4_f128_d01.adam_mse_e120_b32.t64x32 | 2748 | 0.0335 | 0.0444 | -0.0928 | 0.0772 | 0.400 |
| h2_f64_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0336 | 0.0445 | -0.0950 | 0.1040 | 0.346 |
| h2_f128_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0338 | 0.0446 | -0.1009 | 0.1101 | 0.336 |
| h8_f256_d03.adam_hubert_e120_b16.t64x32 | 2748 | 0.0338 | 0.0445 | -0.0986 | 0.1457 | 0.338 |
| h2_f64_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0338 | 0.0446 | -0.1019 | 0.1283 | 0.337 |
| h8_f256_d02.adam_hubert_e120_b16.t64x32 | 2748 | 0.0343 | 0.0450 | -0.1215 | 0.0827 | 0.332 |
| h2_f128_d02.adam_hubert_e80_b32.t64x32 | 2748 | 0.0330 | 0.0440 | -0.0726 | 0.1626 | 0.351 |
| h2_f64_d02.adam_hubert_e120_b16.t64x32 | 2748 | 0.0335 | 0.0444 | -0.0934 | 0.0936 | 0.338 |
| h4_f128_d01.adam_mse_e120_b32.t64x32 | 2748 | 0.0334 | 0.0443 | -0.0875 | 0.1269 | 0.344 |
| h2_f64_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0338 | 0.0446 | -0.1020 | 0.1446 | 0.332 |
| h2_f128_d01.adam_hubert_e80_b32.t64x32 | 2748 | 0.0331 | 0.0441 | -0.0786 | 0.1037 | 0.352 |
| h8_f128_d01.adam_hubert_e120_b32.t64x32 | 2748 | 0.0337 | 0.0445 | -0.0954 | 0.1230 | 0.345 |
| h2_f128_d01.adam_mse_e120_b32.t64x32 | 2748 | 0.0335 | 0.0444 | -0.0946 | 0.0931 | 0.345 |

Tabla B.4.: Resultados de pruebas Transformer sobre el S&P 500 (configuraciones y métricas) – Parte 2 de 2

B. Resultados del entrenamiento de modelos

B.2. Ingeniería de Características

Después de la optimización de hiperparámetros y escoger el "mejor" modelo, se han realizado más de 100 experimentos en los que se han combinado distintos de los indicadores ya explicados.

Se han hecho combinaciones de 2, 3 y hasta 4 indicadores a la vez. Sin embargo, a veces menos es más. El modelo escogido como ganador ha sido el que únicamente cuenta con el índice de volatilidad VIX como información extra.

Se ha de recalcar que algunos de los datos que se tienen sólo reflejan la información de países concretos y no de la situación global. Este es el caso del indicador de la masa monetaria M₂, que únicamente refleja la situación de la liquidez en Estados Unidos, actualizada mensualmente. Se menciona explícitamente el caso de la M₂ puesto que en los últimos meses ha habido ciertos economistas, entre ellos Michael Howell, que han hablado de una posible correlación entre la liquidez y los mercados bursátiles [53].

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|-------------------|------|--------|--------|----------------|---------|----------|
| TNX | 2619 | 0.0400 | 0.0533 | -0.4417 | 0.0199 | 0.430 |
| DX-Y.NYB | 2619 | 0.0354 | 0.0469 | -0.1140 | 0.0082 | 0.422 |
| GC=F | 2619 | 0.0354 | 0.0468 | -0.1116 | -0.0152 | 0.436 |
| M2NS | 2619 | 0.0342 | 0.0466 | -0.1026 | -0.0422 | 0.486 |
| VIX | 2619 | 0.0355 | 0.0461 | -0.0795 | 0.1585 | 0.355 |
| MOVE | 2619 | 0.0375 | 0.0487 | -0.2040 | 0.0742 | 0.443 |
| PAYEMS | 2619 | 0.0355 | 0.0477 | -0.1566 | -0.1038 | 0.467 |
| DPRIME | 2619 | 0.0354 | 0.0467 | -0.1053 | 0.1058 | 0.378 |
| DX-Y.NYB + GC=F | 2619 | 0.0370 | 0.0515 | -0.3440 | 0.0017 | 0.512 |
| DX-Y.NYB + M2NS | 2619 | 0.0365 | 0.0503 | -0.2826 | -0.0380 | 0.541 |
| DX-Y.NYB + TNX | 2619 | 0.0377 | 0.0506 | -0.2975 | 0.0216 | 0.448 |
| DX-Y.NYB + VIX | 2619 | 0.0358 | 0.0526 | -0.4019 | -0.0915 | 0.499 |
| DX-Y.NYB + MOVE | 2619 | 0.0351 | 0.0472 | -0.1287 | 0.1380 | 0.486 |
| DX-Y.NYB + PAYEMS | 2619 | 0.0350 | 0.0494 | -0.2406 | -0.0697 | 0.578 |
| DX-Y.NYB + DPRIME | 2619 | 0.0396 | 0.0528 | -0.4145 | -0.1031 | 0.409 |
| GC=F + M2NS | 2619 | 0.0402 | 0.0592 | -0.7807 | 0.0460 | 0.475 |
| GC=F + TNX | 2619 | 0.0389 | 0.0528 | -0.4123 | 0.0208 | 0.471 |
| GC=F + VIX | 2619 | 0.0404 | 0.0622 | -0.9605 | -0.0779 | 0.464 |
| GC=F + MOVE | 2619 | 0.0366 | 0.0493 | -0.2329 | 0.0630 | 0.491 |
| GC=F + PAYEMS | 2619 | 0.0385 | 0.0517 | -0.3574 | -0.0185 | 0.476 |
| GC=F + DPRIME | 2619 | 0.0375 | 0.0518 | -0.3631 | 0.0001 | 0.480 |
| M2NS + TNX | 2619 | 0.0435 | 0.0581 | -0.7114 | 0.0187 | 0.422 |
| M2NS + VIX | 2619 | 0.0376 | 0.0577 | -0.6910 | 0.0649 | 0.537 |
| M2NS + MOVE | 2619 | 0.0436 | 0.0580 | -0.7053 | 0.0264 | 0.468 |
| M2NS + PAYEMS | 2619 | 0.0478 | 0.0651 | -1.1537 | -0.0570 | 0.509 |
| M2NS + DPRIME | 2619 | 0.0363 | 0.0491 | -0.2251 | 0.0310 | 0.503 |

Tabla B.5.: Resultados de pruebas Transformer multi-ticker para S&P 500 (configuraciones y métricas) — Parte 1 de 4

B. Resultados del entrenamiento de modelos

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|----------------------------|------|--------|--------|----------------|---------|----------|
| TNX + VIX | 2619 | 0.0379 | 0.0568 | -0.6371 | -0.0645 | 0.473 |
| TNX + MOVE | 2619 | 0.0389 | 0.0515 | -0.3472 | 0.0492 | 0.454 |
| TNX + PAYEMS | 2619 | 0.0384 | 0.0522 | -0.3850 | 0.0218 | 0.485 |
| TNX + DPRIME | 2619 | 0.0388 | 0.0517 | -0.3586 | 0.0243 | 0.440 |
| VIX + MOVE | 2619 | 0.0407 | 0.0633 | -1.0319 | -0.1377 | 0.475 |
| VIX + PAYEMS | 2619 | 0.0371 | 0.0519 | -0.3645 | 0.0773 | 0.502 |
| VIX + DPRIME | 2619 | 0.0403 | 0.0575 | -0.6783 | -0.0760 | 0.383 |
| MOVE + PAYEMS | 2619 | 0.0399 | 0.0541 | -0.4844 | 0.0651 | 0.470 |
| MOVE + DPRIME | 2619 | 0.0379 | 0.0520 | -0.3747 | 0.0219 | 0.491 |
| PAYEMS + DPRIME | 2619 | 0.0406 | 0.0558 | -0.5788 | 0.1021 | 0.506 |
| DX-Y.NYB + GC=F + M2NS | 2619 | 0.0416 | 0.0579 | -0.7001 | -0.0570 | 0.531 |
| DX-Y.NYB + GC=F + TNX | 2619 | 0.0410 | 0.0550 | -0.5338 | 0.0323 | 0.480 |
| DX-Y.NYB + GC=F + VIX | 2619 | 0.0379 | 0.0522 | -0.3819 | 0.0773 | 0.511 |
| DX-Y.NYB + GC=F + MOVE | 2619 | 0.0402 | 0.0538 | -0.4674 | -0.0446 | 0.439 |
| DX-Y.NYB + GC=F + PAYEMS | 2619 | 0.0407 | 0.0569 | -0.6414 | -0.0752 | 0.503 |
| DX-Y.NYB + GC=F + DPRIME | 2619 | 0.0438 | 0.0587 | -0.7481 | -0.0454 | 0.423 |
| DX-Y.NYB + M2NS + TNX | 2619 | 0.0413 | 0.0556 | -0.5691 | 0.0850 | 0.493 |
| DX-Y.NYB + M2NS + VIX | 2619 | 0.0402 | 0.0555 | -0.5611 | 0.0383 | 0.535 |
| DX-Y.NYB + M2NS + MOVE | 2619 | 0.0485 | 0.0660 | -1.2094 | -0.0337 | 0.450 |
| DX-Y.NYB + M2NS + PAYEMS | 2619 | 0.0522 | 0.0706 | -1.5258 | -0.0551 | 0.500 |
| DX-Y.NYB + M2NS + DPRIME | 2619 | 0.0421 | 0.0596 | -0.8051 | -0.0163 | 0.504 |
| DX-Y.NYB + TNX + VIX | 2619 | 0.0401 | 0.0590 | -0.7656 | -0.0355 | 0.476 |
| DX-Y.NYB + TNX + MOVE | 2619 | 0.0414 | 0.0555 | -0.5655 | 0.0923 | 0.494 |
| DX-Y.NYB + TNX + PAYEMS | 2619 | 0.0439 | 0.0586 | -0.7421 | 0.0757 | 0.449 |
| DX-Y.NYB + TNX + DPRIME | 2619 | 0.0417 | 0.0551 | -0.5415 | 0.0658 | 0.433 |
| DX-Y.NYB + VIX + MOVE | 2619 | 0.0418 | 0.0649 | -1.1350 | -0.1197 | 0.477 |
| DX-Y.NYB + VIX + PAYEMS | 2619 | 0.0445 | 0.0747 | -1.8323 | -0.0025 | 0.558 |
| DX-Y.NYB + VIX + DPRIME | 2619 | 0.0392 | 0.0559 | -0.5876 | -0.0609 | 0.475 |
| DX-Y.NYB + MOVE + PAYEMS | 2619 | 0.0398 | 0.0539 | -0.4737 | 0.0699 | 0.483 |
| DX-Y.NYB + MOVE + DPRIME | 2619 | 0.0407 | 0.0556 | -0.5712 | -0.0173 | 0.448 |
| DX-Y.NYB + PAYEMS + DPRIME | 2619 | 0.0441 | 0.0646 | -1.1180 | 0.0634 | 0.539 |
| GC=F + M2NS + TNX | 2619 | 0.0426 | 0.0598 | -0.8147 | 0.1053 | 0.521 |

Tabla B.6: Resultados de pruebas Transformer multi-ticker para S&P 500 (configuraciones y métricas) — Parte 2 de 4

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|------------------------|------|--------|--------|----------------|---------|----------|
| GC=F + M2NS + VIX | 2619 | 0.0438 | 0.0617 | -0.9291 | 0.0661 | 0.521 |
| GC=F + M2NS + MOVE | 2619 | 0.0461 | 0.0626 | -0.9887 | 0.0538 | 0.506 |
| GC=F + M2NS + PAYEMS | 2619 | 0.0517 | 0.0727 | -1.6820 | -0.0136 | 0.540 |
| GC=F + M2NS + DPRIME | 2619 | 0.0445 | 0.0614 | -0.9158 | 0.0644 | 0.510 |
| GC=F + TNX + VIX | 2619 | 0.0403 | 0.0562 | -0.6014 | -0.0604 | 0.467 |
| GC=F + TNX + MOVE | 2619 | 0.0408 | 0.0536 | -0.4593 | 0.0080 | 0.443 |
| GC=F + TNX + PAYEMS | 2619 | 0.0437 | 0.0594 | -0.7881 | -0.0846 | 0.464 |
| GC=F + TNX + DPRIME | 2619 | 0.0439 | 0.0584 | -0.7327 | 0.0076 | 0.442 |
| GC=F + VIX + MOVE | 2619 | 0.0432 | 0.0748 | -1.6168 | -0.1168 | 0.457 |
| GC=F + VIX + PAYEMS | 2619 | 0.0464 | 0.0678 | -1.3341 | -0.1423 | 0.502 |
| GC=F + VIX + DPRIME | 2619 | 0.0418 | 0.0671 | -1.2866 | -0.0677 | 0.488 |
| GC=F + MOVE + PAYEMS | 2619 | 0.0442 | 0.0607 | -0.8706 | -0.0317 | 0.491 |
| GC=F + MOVE + DPRIME | 2619 | 0.0433 | 0.0599 | -0.8176 | -0.0130 | 0.460 |
| GC=F + PAYEMS + DPRIME | 2619 | 0.0476 | 0.0650 | -1.1440 | 0.0033 | 0.487 |
| M2NS + TNX + VIX | 2619 | 0.0422 | 0.0635 | -1.0475 | -0.0587 | 0.494 |
| M2NS + TNX + MOVE | 2619 | 0.0483 | 0.0692 | -1.4294 | 0.0264 | 0.482 |
| M2NS + TNX + PAYEMS | 2619 | 0.0555 | 0.0742 | -1.7951 | 0.0289 | 0.486 |
| M2NS + TNX + DPRIME | 2619 | 0.0449 | 0.0607 | -0.8703 | -0.0145 | 0.485 |
| M2NS + VIX + MOVE | 2619 | 0.0460 | 0.0693 | -1.4357 | -0.0131 | 0.501 |
| M2NS + VIX + PAYEMS | 2619 | 0.0433 | 0.0605 | -0.8572 | -0.0479 | 0.468 |
| M2NS + VIX + DPRIME | 2619 | 0.0428 | 0.0612 | -0.8991 | -0.0430 | 0.483 |
| M2NS + MOVE + PAYEMS | 2619 | 0.0528 | 0.0714 | -1.5840 | 0.0505 | 0.464 |
| M2NS + MOVE + DPRIME | 2619 | 0.0447 | 0.0602 | -0.8381 | 0.0556 | 0.472 |
| M2NS + PAYEMS + DPRIME | 2619 | 0.0623 | 0.0903 | -3.1403 | 0.0249 | 0.525 |
| TNX + VIX + MOVE | 2619 | 0.0459 | 0.0695 | -1.4506 | -0.0140 | 0.448 |
| TNX + VIX + PAYEMS | 2619 | 0.0418 | 0.0612 | -0.9028 | 0.0619 | 0.488 |
| TNX + VIX + DPRIME | 2619 | 0.0412 | 0.0575 | -0.6792 | 0.0609 | 0.443 |
| TNX + MOVE + PAYEMS | 2619 | 0.0419 | 0.0592 | -0.7797 | 0.0351 | 0.485 |
| TNX + MOVE + DPRIME | 2619 | 0.0436 | 0.0660 | -1.2088 | 0.0441 | 0.500 |
| TNX + PAYEMS + DPRIME | 2619 | 0.0440 | 0.0609 | -0.8840 | 0.0401 | 0.487 |

Tabla B.7: Resultados de pruebas Transformer multi-ticker para S&P 500 (configuraciones y métricas) — Parte 3 de 4

B. Resultados del entrenamiento de modelos

| Nombre | n | MAE | RMSE | R ² | corr | hit_rate |
|---------------------------------|------|--------|--------|----------------|---------|----------|
| VIX + MOVE + PAYEMS | 2619 | 0.0423 | 0.0638 | -1.0682 | -0.0209 | 0.489 |
| VIX + MOVE + DPRIME | 2619 | 0.0409 | 0.0610 | -0.8898 | 0.0140 | 0.478 |
| VIX + PAYEMS + DPRIME | 2619 | 0.0448 | 0.0654 | -1.1708 | 0.0736 | 0.486 |
| MOVE + PAYEMS + DPRIME | 2619 | 0.0415 | 0.0589 | -0.7614 | -0.0010 | 0.554 |
| DX-Y.NYB + GC=F + M2NS + TNX | 2619 | 0.0459 | 0.0616 | -0.9246 | 0.0512 | 0.488 |
| DX-Y.NYB + GC=F + M2NS + VIX | 2619 | 0.0446 | 0.0649 | -1.1374 | 0.1093 | 0.464 |
| DX-Y.NYB + GC=F + M2NS + MOVE | 2619 | 0.0466 | 0.0630 | -1.0133 | 0.0137 | 0.481 |
| DX-Y.NYB + GC=F + M2NS + PAYEMS | 2619 | 0.0655 | 0.0947 | -3.5491 | -0.0224 | 0.543 |
| DX-Y.NYB + GC=F + M2NS + DPRIME | 2619 | 0.0428 | 0.0599 | -0.8219 | -0.0345 | 0.498 |
| DX-Y.NYB + GC=F + TNX + VIX | 2619 | 0.0426 | 0.0658 | -1.1997 | -0.0234 | 0.516 |
| DX-Y.NYB + GC=F + TNX + MOVE | 2619 | 0.0436 | 0.0578 | -0.6957 | -0.0206 | 0.462 |
| DX-Y.NYB + GC=F + TNX + PAYEMS | 2619 | 0.0469 | 0.0618 | -0.9387 | 0.1084 | 0.457 |
| DX-Y.NYB + GC=F + TNX + DPRIME | 2619 | 0.0442 | 0.0580 | -0.7087 | 0.0835 | 0.471 |
| DX-Y.NYB + GC=F + VIX + MOVE | 2619 | 0.0419 | 0.0608 | -0.8784 | 0.0212 | 0.471 |
| DX-Y.NYB + GC=F + VIX + PAYEMS | 2619 | 0.0442 | 0.0608 | -0.8756 | 0.0238 | 0.505 |
| DX-Y.NYB + GC=F + VIX + DPRIME | 2619 | 0.0454 | 0.0740 | -1.7823 | -0.0911 | 0.480 |
| DX-Y.NYB + GC=F + MOVE + PAYEMS | 2619 | 0.0434 | 0.0590 | -0.7670 | -0.0529 | 0.471 |
| DX-Y.NYB + GC=F + MOVE + DPRIME | 2619 | 0.0431 | 0.0572 | -0.6577 | 0.0623 | 0.497 |

Tabla B.8.: Resultados de pruebas Transformer multi-ticker para S&P 500 (configuraciones y métricas) — Parte 4 de 4

Bibliografía

- [1] Securities Industry and Financial Markets Association. *Capital Markets Fact Book 2025*. SIFMA, New York, NY, 2025.
- [2] Daniel Kahneman. *Pensar rápido, pensar despacio*. Psicología (Debate). Debate, Barcelona, 20a ed., 10a reimp. edition, 2024.
- [3] Robert A. Levy. Relative strength as a criterion for investment selection. *The Journal of Finance*, 22(4):595–610, December 1967.
- [4] Narasimhan Jegadeesh and Sheridan Titman. Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, 48(1):65–91, March 1993.
- [5] Hugo Ferrer. *Inversor global: inversión y especulación siguiendo las fuerzas económicas que dirigen el mercado de acciones*. Eisenbrauns, S.l., 2014.
- [6] John C. Bogle. *El pequeño libro para invertir con sentido común: el mejor método para garantizar la rentabilidad en bolsa*. Deusto, Barcelona, 1^a ed. edition, 2016.
- [7] Nassim Nicholas Taleb. *El cisne negro: el impacto de lo altamente improbable*. Booket (Espasa). Divulgación. Actualidad. Booket, Barcelona, 1^a ed. en colección booket edition, 2012.
- [8] Andreas F. Clenow. *Stocks on the Move: Beating the Market with Hedge Fund Momentum Strategies*. CreateSpace Independent Publishing Platform, 2015.
- [9] Robert B. Ash. *Basic Probability Theory*. Dover Publications, Mineola, N.Y., dover ed. edition, 2008.
- [10] Vijay K. Rohatgi and A. K. Md Ehsanes Saleh. *An Introduction to Probability and Statistics*. Wiley Series in Probability and Statistics. Wiley, Hoboken, New Jersey, 3rd ed., 7. nachdruck edition, 2016.
- [11] Tom M. Apostol. *Análisis matemático: introducción moderna al cálculo superior*. Reverté, Barcelona, 1974.
- [12] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge; New York; Melbourne; New Delhi; Singapore, version 29 edition, 2023.
- [13] Peter Lynch. *Un paso por delante de Wall Street*. Deusto, Barcelona, 17^a ed. edition, 2021.
- [14] Benjamin Graham. *El inversor inteligente: un libro de asesoramiento práctico*. HarperEnfoque, Nashville, Tennessee, edición revisada edition, 2021.
- [15] Carles M. Cuadras. *Nuevos métodos de análisis multivariante*. CMC Editions, Barcelona, España, 5^a edición revisada edition, 2014.
- [16] Michael H. Kutner, Christopher J. Nachtsheim, John Neter, and William Li. *Applied Linear Statistical Models*. The McGraw-Hill/Irwin Series in Operations and Decision Sciences. McGraw-Hill Irwin, Boston, 5th ed. edition, 2005.
- [17] E. S. Keeping. A significance test for exponential regression. *The Annals of Mathematical Statistics*,

Bibliografía

- 22(2):180–198, 1951.
- [18] Mariette Awad. *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Apress L. P., Berkeley, CA, 2015.
- [19] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.
- [20] Fernando Berzal. *Redes neuronales & deep learning. Volumen I*. Independently published, Granada, 2019.
- [21] Esteban Rubén Hurtado. Derivadas parciales y direccionales – multiplicadores de Lagrange. https://sistemas.fciencias.unam.mx/~erhc/calculo3_20171/derivadas_parciales_direccionales_2016_11.pdf. Apuntes Cálculo Diferencial e Integral II, UNAM.
- [22] Jesús Muñoz López. Métodos de optimización en máquinas de soporte vectorial de vecindad cercana. <https://www.ugr.es/~acanada/docencia/matematicas/TFG-definitivo-2julio2018.pdf>, 2018.
- [23] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001.
- [24] Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. Consistency of random forests. (arXiv:1405.2881), August 2015.
- [25] Gérard Biau and Erwan Scornet. A random forest guided tour. (arXiv:1511.05741), November 2015.
- [26] Trent D. Buskirk. Surveying the forests and sampling the trees: An overview of classification and regression trees and random forests with applications in survey research. *Survey Practice*, 11(1), January 2018.
- [27] Hal R. Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, May 2014.
- [28] Abdullah Bin Omar, Shuai Huang, Anas A. Salameh, Haris Khurram, and Muhammad Fareed. Stock market forecasting using the random forest and deep neural network models before and during the COVID-19 period. *Frontiers in Environmental Science*, 10, July 2022.
- [29] Ramón Díaz-Uriarte and Sara Alvarez De Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1):3, December 2006.
- [30] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989.
- [31] Walter Rudin. *Functional Analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill, Boston, Mass., 2nd ed. edition, 1996.
- [32] Haïm Brézis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Universitext. Springer Science+Business Media, LLC, New York, NY, 2011.
- [33] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, January 1993.
- [34] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

- [35] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Neural Networks*, 2014.
- [36] Sébastien Bubeck. Convex optimization: Algorithms and complexity. (arXiv:1405.4980), November 2015.
- [37] Ian Goodfellow, Aaron Courville, and Yoshua Bengio. *Deep Learning*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, 2016.
- [38] Afshine Amidi. Stanford cs-230 deep learning cheatsheet: Recurrent neural networks. <https://github.com/afshinea/stanford-cs-230-deep-learning/blob/master/en/cheatsheet-recurrent-neural-networks.pdf>, 2018. CS-230 cheatsheet.
- [39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, pages 1735–1780, November 1997.
- [40] Guo-Bing Zhou, Jianxin Wu, Chen-Lin Zhang, and Zhi-Hua Zhou. Minimal gated unit for recurrent neural networks. (arXiv:1603.09420), March 2016.
- [41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.
- [42] Yu Chen, Nathalia Céspedes, and Payam Barnaghi. A closer look at transformers for time series forecasting: Understanding why they work and where they struggle. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.
- [43] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. (arXiv:2202.07125), May 2023.
- [44] Carl R. Bacon. *Practical Portfolio Performance: Measurement and Attribution*. Wiley Finance Series. Wiley, Chichester; Hoboken, NJ, 2004.
- [45] Zvi Bodie, Alex Kane, and Alan J. Marcus. *Investments*. The McGraw-Hill/Irwin Series in Finance, Insurance and Real Estate. McGraw-Hill Education, New York, NY, 10th ed. edition, 2014.
- [46] Javier Caballero. *Warren y Charlie: Lecciones sobre la inversión, los negocios y la vida*. 2020.
- [47] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass., 3rd print. edition, 2008.
- [48] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.
- [49] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. (arXiv:1206.2944), August 2012.
- [50] Masaya Abe and Hideki Nakayama. Deep learning for forecasting stock returns in the cross-section. (arXiv:1801.01777), June 2018.
- [51] Sang Il Lee. Hyperparameter optimization for forecasting stock returns. (arXiv:2001.10278), January 2020.
- [52] Nassim Nicholas Taleb. *Antifrágil: las cosas que se benefician del desorden*. Paidós transiciones. Paidós, Barcelona, 1a ed. edition, 2013.

Bibliografía

- [53] Michael J. Howell. *Capital Wars: The Rise of Global Liquidity*. Springer International Publishing AG, Cham, 2020.