Universidad de Valladolid

Facultad de Ciencias Económicas y Empresariales

Grado en Economía

Trabajo de Fin de Grado

Inteligencia Artificial para la optimización de carteras:

de Markowitz al Aprendizaje Profundo

Presentado por

Pablo Javier Iglesias Castañón

Tutelado por

María Mercedes Prieto Alaiz

Alfredo Martínez Bobillo

Valladolid, 19 de julio de 2022

RESUMEN

El Deep Learning (DL) es una tecnología llamada a cambiar la forma de identificar, afrontar y resolver los retos presentes y futuros de la humanidad. El objetivo de este proyecto es presentar dicha herramienta a perfiles iniciados o avanzados en la disciplina económica mediante un caso de gestión de carteras, comparando los resultados obtenidos con carteras tradicionales derivadas de la Teoría Moderna.

En concreto, se ha implementado una Red Neuronal de Memoria a Corto y Largo Plazo o *Long-Short Term Memory* (LSTM), capaz de predecir a un año vista el precio de cualquier activo financiero dada su cotización histórica. Con estas predicciones, se ha propuesto una metodología innovadora para formar carteras diversificadas y adaptadas al perfil de riesgo del inversor. De este modo, no solo se ha superado el desempeño de las principales carteras basadas en el Modelo de Markowitz y sus extensiones posteriores, sino que se ha demostrado una nueva forma eficaz, funcional, consistente y flexible de construir carteras de inversión.

Palabras clave: *Aprendizaje Profundo*, *Redes Neuronales LSTM*, *Gestión de carteras*.

ABSTRACT

Deep Learning (DL) is a groundbreaking technology destined to change the way in which humanity faces current and future challenges. The main purpose of this paper is to introduce such an instrument to novice and experienced economists. In doing so, a simple portfolio management case has been set to gain first-hand experience of the prospects that Deep Learning offers to the economy and society as a whole.

Specifically, a LSTM Neural Network designed and implemented to predict one-year forward stock prices will be used to build diversified and profitable portfolios, beating major approaches from Markowitz Model.

Keywords: *Deep Learning*, *LSTM Neural Network*, *Portfolio management*. **JEL:** C22, C45, G11.

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN

La Revolución Digital fue un periodo de transición marcado por la proliferación de los sistemas digitales como reemplazo a la tecnología electromecánica empleada hasta finales del siglo XX. Este proceso, congénito al desarrollo de la computación y las redes de comunicación global, se considera la antesala de una nueva etapa en la historia de la humanidad: la Era de la Información.

La llegada de los datos masivos a las sociedades modernas ha puesto de manifiesto la brecha existente entre la información y el conocimiento. Precisamente, uno de los mayores retos de la era digital consiste en obtener, validar, clasificar y procesar los datos para alcanzar conclusiones sólidas. De esta dificultad, han surgido áreas tan populares como el Big Data (BD) o la Inteligencia Artificial (AI).

El Deep Learning se ha convertido en uno de los segmentos más prometedores y apasionantes de este abanico tecnológico, postulándose como una herramienta capaz de cambiar la forma de afrontar los grandes retos de nuestra era. El objetivo principal de este proyecto consiste en introducir dicha herramienta mediante un caso práctico de gestión de carteras. En concreto, se va a diseñar e implementar una Red Neuronal de Memoria a Corto y Largo Plazo o *Long-Short Term Memory* (LSTM), capaz de predecir el precio cualquier activo en un horizonte anual. A partir de estas predicciones, se ha propuesto una metodología novedosa para construir carteras diversificadas y adaptadas al perfil de riesgo del inversor, comparando su desempeño con carteras derivadas de la Teoría Moderna iniciada por Markowitz en 1952.

Para ello, se han presentado cuidadosamente los fundamentos teóricos de ambas metodologías en el segundo capítulo del documento. En la siguiente sección se ha descrito el caso práctico, los instrumentos y técnicas utilizadas, la metodología propuesta en este proyecto y los resultados obtenidos. Finalmente, en el capítulo cuarto se han discutido dichos resultados y las implicaciones que el Deep Learning puede tener para la economía y la sociedad.

El procedimiento planteado en este proyecto no solo ha exhibido un desempeño superior a las metodologías tradicionales, sino que ha demostrado ser una técnica eficaz, funcional, consistente a largo plazo y flexible para el inversor, permitiendo lidiar con varias restricciones impuestas por las metodologías tradicionales como la prohibición de la venta a corto o la incorporación de efectivo en la cartera, todo ello sin apoyarse en premisas inverosímiles como la consideración de un mercado perfecto, completo y eficiente.

Mas allá del valor que esta propuesta pueda aportar a la disciplina económica y financiera, también es objetivo prioritario del proyecto invitar a la reflexión sobre los cambios venideros, dado el potencial de estas nuevas técnicas, y la necesidad de preparar a las nuevas generaciones de economistas para liderar esta transición tecnológica con responsabilidad, ética y honestidad.

2. FUNDAMENTOS

Antes de afrontar el caso práctico, es imprescindible presentar los fundamentos teóricos de las metodologías empleadas. En primer lugar, se desarrollará la Teoría Moderna de Carteras o *Modern Portfolio Theory* (MPT), haciendo especial hincapié en las aportaciones de Markowitz. A continuación, se introducirá el Deep Learning y los conceptos básicos que subyacen a las Redes LSTM.

2.1. Teoría Moderna de Carteras

La gestión de carteras es la disciplina de la economía financiera que se encarga de la toma de decisiones de inversión sobre un conjunto de activos financieros. Los activos financieros son títulos o anotaciones contables que otorgan al comprador el derecho a percibir un ingreso futuro procedente del vendedor. Presentan dos características principales: riesgo y rentabilidad. El riesgo viene determinado por la solvencia del vendedor y las garantías que ofrece para cumplir con su obligación de pago. La rentabilidad es la plusvalía que obtiene el comprador por exponerse al riesgo. Finalmente, se denomina cartera de valores a cualquier combinación de activos financieros.

Antes de 1952, los gestores de carteras trataban de maximizar la rentabilidad de sus inversiones sin tener en cuenta el riesgo, circunstancia que obligaba a liquidar en pérdidas numerosas operaciones. Aunque los economistas de la época fueron capaces de intuir una relación inversa entre el número de activos en cartera y el riesgo de la inversión, no establecieron ningún criterio que determinase la elección de dichos activos. Fue entonces cuando el economista Harry Markowitz publicó el artículo *Portfolio Selection* (1952) en la prestigiosa revista *The Journal of Finance*. Esta obra se ha considerado el punto de partida de la Teoría Moderna de Carteras y las finanzas cuantitativas gracias al enfoque conocido como media-varianza, que puso a disposición del inversor el aparato matemático y estadístico necesario para seleccionar su cartera óptima. Tobin (1958) y Sharpe (1964) reforzaron el modelo de Markowitz al considerar un activo libre de riesgo. Esto permitió relajar algunas hipótesis de partida que restaban aplicabilidad a la Teoría Moderna de Carteras. (Anexo I).

2.1.1. Conceptos básicos

Los modelos de media-varianza se construyen a partir de la rentabilidad diaria de los activos financieros. Suponiendo una sucesión de precios del activo i, indexados por la variable tiempo $\{P_{it}, t = 1, \dots, T\}$, cada dato de una serie temporal se debe interpretar como una muestra de tamaño uno de la distribución de probabilidad correspondiente a la variable aleatoria de ese instante. La rentabilidad de un activo i en el periodo t se obtiene como:

$$R_{it} = \frac{P_{it} - P_{it-1}}{P_{it-1}} \quad (1)$$

Ciertos supuestos del proceso estocástico $\{R_{it}, t = 1, \dots, T\}$ (Novales, 2017), permiten simplificar su tratamiento empírico. En concreto, se considera que para cualquier t con $t = 1 \dots T$, R_{it} tiene una distribución simétrica, aproximadamente una normal, caracterizada por la esperanza o rentabilidad esperada $E(R_{it})=\mu_i$ y la varianza, volatilidad o riesgo, $Var(R_{it}) = \sigma_i^2 = E(R_i - \mu_i)^2$. Se puede estimar μ_i y σ_i^2 mediante sus homólogos muestrales:

$$E(R_i) = \hat{\mu}_i = \frac{\sum_{t=1}^{T} R_{it}}{T} \quad (2)$$

$$Var(R_i) = \hat{\sigma}_i^2 = \frac{\sum_{t=1}^{T} (R_{it} - \hat{\mu}_i)^2}{T - 1} \quad (3)$$

También es habitual expresar la rentabilidad y el riesgo de los activos financieros de forma anualizada. Además, la volatilidad se suele dar en términos de desviación típica. De este modo, es posible trabajar con cifras en tanto por ciento y fácilmente comparables. Sabiendo que un año equivale a unos 252 periodos, jornadas o días de cotización, la rentabilidad esperada y el riesgo anualizado de un activo son:

$$\hat{\mu}_i^{ann} = \left(\prod_{t=1}^{T} 1 + R_{it} \right)^{\frac{252}{T}} - 1 \quad (4)$$

$$\hat{\sigma}_i^{ann} = \sqrt{\hat{\sigma}_i^2 \cdot 252} \quad (5)$$

En 1952, Markowitz presentó estos fundamentos y extendió su análisis a carteras compuestas por n activos financieros. De este modo, una cartera de inversión p queda definida por un vector $\mathrm{X} = (x_1, \dots, x_n)$, que representa la proporción del capital o el peso x invertido en cada título y un vector $R_p = (R_1, \dots, R_n)$ que recoge las rentabilidades, siendo su distribución conjunta normal multidimensional con esperanza o rentabilidad esperada $E(R_p)$ y matriz de varianzas y covarianzas Σ. La covarianza entre dos activos i y j se expresa como:

$$\sigma_{ij} = E\left[(R_{it} - \mu_i)(R_{jt} - \mu_j)\right] \tag{6}$$

De este modo, $E(R_p)$ y Σ_p adoptan la siguiente forma:

$$E(R_p) = \begin{pmatrix} E(R_1) \\ \vdots \\ E(R_n) \end{pmatrix} = \mu^T = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix} \tag{7}$$

$$\Sigma_p = \begin{bmatrix} \sigma_1^2 & \cdots & \sigma_{1n} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \cdots & \sigma_n^2 \end{bmatrix} \tag{8}$$

Siendo μ el vector de rentabilidades esperadas de los activos, es posible formular la rentabilidad esperada y el riesgo de una cartera p en notación escalar y matricial:

$$\mu_p = \sum_{i=1}^{n} x_i \, \mu_i = \mathrm{X}^T \mu \tag{9}$$

$$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i \, x_j \, \sigma_{ij} = \mathrm{X}^T \, \Sigma \, \mathrm{X} \tag{10}$$

Al cambiar los pesos del vector X es posible formar infinitas carteras, combinando así los n activos en diferentes proporciones. Esto se conoce como asignación de títulos o *asset allocation*. Para dos activos financieros A y B, (9) y (10) equivalen a:

$$\mu_p = x_A \mu_A + x_B \mu_B \tag{11}$$

$$\sigma_p^2 = x_A \sigma_A^2 + x_B \sigma_B^2 + 2 x_A x_B \sigma_{AB} \tag{12}$$

Alternativamente, la volatilidad de la cartera se puede expresar en términos del coeficiente de correlación, medida estadística que cuantifica la dependencia lineal de dos variables aleatorias en un rango (-1, 1). Así, el coeficiente de correlación de dos activos i y j adopta la siguiente forma:

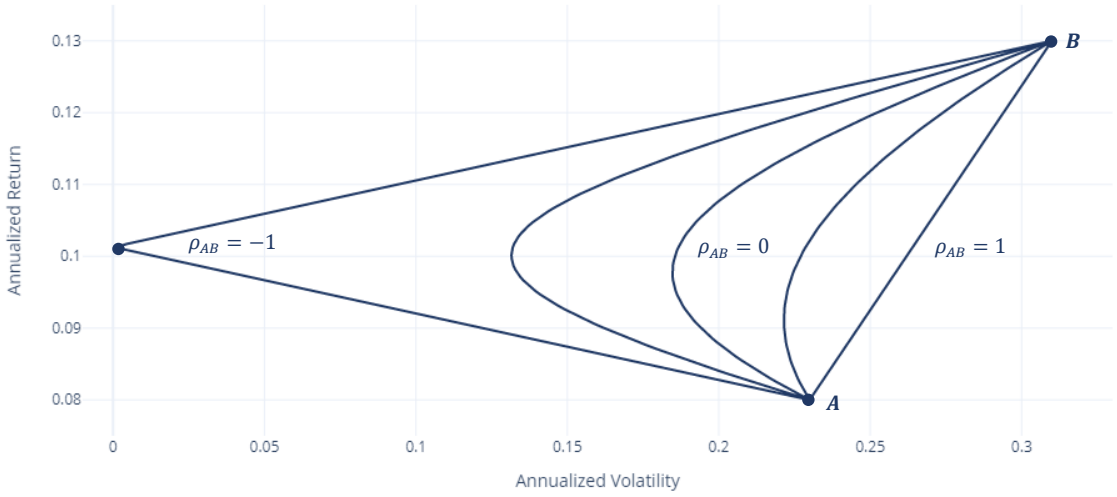
$$\rho_{ij} = \frac{\sigma_{ij}}{\sigma_i \sigma_j} \qquad (13)$$

Finalmente, despejando la covarianza en (13) y sustituyendo en (12) se tiene:

$$\sigma_p^2 = x_A \sigma_A^2 + x_B \sigma_B^2 + 2 x_A x_B \sigma_A \sigma_B \rho_{AB} \qquad (14)$$

En la Figura 2.1. se ha representado el conjunto de posibilidades de inversión de los activos A y B, es decir, todas las posibles carteras que se pueden obtener con dichos títulos combinando diferentes pesos x_A y x_B, de tal forma que siempre se verifique $x_A + x_B = 1$, dados μ_A, μ_B, σ_A^2, σ_B^2 y para diferentes valores de ρ_{AB}. Los puntos A y B representan $\mathrm{X} = (x_A, x_B) = (1, 0)$ y $\mathrm{X} = (0, 1)$, respectivamente.

Figura 2.1. Conjunto de posibilidades de inversión en dos activos A y B según ρ_{AB}.



Fuente: *Elaboración propia. Anexo II, pág.70.*

Como se puede apreciar, cuanto más cercana sea la correlación a -1, se pueden obtener binomios rentabilidad-riesgo mucho mejores para el inversor, alcanzando puntos de mayor rentabilidad a cambio de un menor riesgo y viceversa. Se trata de una de las contribuciones principales de Harry Markowitz, ya que por primera vez estableció un criterio objetivo para la selección de activos. Esta idea es la base de la diversificación eficiente o no ingenua. Así, cualquier inversor racional y averso al riesgo perseguirá reducir la volatilidad de su cartera incluyendo en ella activos con correlación negativa.

2.1.2. Supuestos del modelo de Markowitz

Antes de avanzar con el desarrollo del modelo de Markowitz es necesario conocer todas las hipótesis de partida. Se trata de un conjunto muy amplio y restrictivo de premisas que han sido objeto de crítica desde la publicación del artículo en 1952.

2.1.2.1. Supuestos sobre el inversor

- Los inversores buscan maximizar la utilidad esperada de su riqueza final, de forma que persiguen la cartera que les proporcione un mayor bienestar de acuerdo con sus preferencias subjetivas sobre el riesgo.
- Su horizonte temporal para el análisis es uniperiodo.
- Consideran únicamente la media y la varianza de la rentabilidad histórica para formar carteras, así como el coeficiente de correlación o la covarianza.
- Se basan en el concepto de diversificación eficiente, escogiendo aquellos títulos con menor coeficiente de correlación.
- Aunque los inversores pueden ser también amantes o neutrales al riesgo, Markowitz considera únicamente individuos aversos al riesgo y racionales. Buscan maximizar la rentabilidad y minimizar la volatilidad de su cartera.

2.1.2.2. Supuestos sobre los mercados y los activos financieros

- La información al alcance del inversor es completa y perfecta.
- Los mercados financieros se presuponen perfectos: ausencia de costes de transacción, impuestos, inflación y variabilidad del tipo de interés.
- Existe un numero finito de activos con riesgo.
- La venta a corto (*short-selling*) no está permitida.
- Las proporciones invertidas en cada título suman siempre la unidad.
- Los individuos son precio-aceptantes (*price-takers*).
- Los activos financieros se consideran infinitamente divisibles, por lo que es posible adquirirlos de forma fraccionada.
- Los valores tienen liquidez inmediata al finalizar el periodo de referencia.
- Existen al menos dos títulos con distinta rentabilidad esperada y ningún par de ellos presenta correlación perfecta y negativa.

2.1.3. Etapas del modelo de Markowitz

El modelo de Markowitz se basa en la elección de los títulos y pesos adecuados para obtener la cartera óptima de un inversor cualquiera, dada su actitud subjetiva frente al riesgo. Dicho modelo se desarrolla en dos etapas. La primera etapa es objetiva, ya que conduce a un resultado común para el conjunto de los inversores: la Frontera Eficiente. La segunda etapa es subjetiva, puesto que el resultado varía en función de las preferencias individuales ante el riesgo. Se trata de modelizar la tolerancia o aversión al riesgo del inversor para identificar su cartera óptima.

2.1.3.1. Determinación de la Frontera Eficiente

La primera etapa consiste en encontrar las carteras que maximizan la rentabilidad esperada para un nivel de riesgo dado o viceversa. Para dos carteras P y Q, se dice que P es preferida o dominante sobre Q cuando su rentabilidad esperada es mayor y su riesgo es menor o igual o cuando su rentabilidad esperada es mayor o igual y su riesgo es menor. En términos de lógica proposicional esto equivale a:

$$P \succ Q \;\Leftrightarrow \left(\mu_P > \mu_Q \wedge \; \sigma_P^2 \leq \sigma_Q^2\right) \vee (\mu_P \geq \mu_Q \wedge \; \sigma_P^2 < \sigma_Q^2) \qquad (15)$$

La generalización de esta lógica a los n activos y carteras con riesgo del mercado se traduce en un problema de programación cuadrática con formulación primal:

$$\max_{x_i} \mu_p = \sum_{i=1}^{n} x_i \, \mu_i = \mathrm{X}^T \mu$$

Sujeto a:

- $\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i \, x_j \sigma_{ij} = \mathrm{X}^T \, \Sigma \, \mathrm{X} \leq \sigma_0^2$
- $\sum_{i=1}^{n} x_i = \mathrm{X}^T \cdot \bar{1} = 1; \; \bar{1} = (1, \dots, 1)^T$
- $x_i \geq 0; \; i = 1, \dots, n$

(16)

Como se puede ver, se trata de obtener los pesos que maximizan la rentabilidad esperada de una cartera p, sujeto a un riesgo menor o igual que un nivel objetivo σ_0^2, siendo todos los pesos positivos, dada la prohibición de la venta a corto, y con suma la unidad. Alternativamente, el problema dual queda definido como:

$$\min_{x_i} \sigma_p^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i = \mathrm{X}^T \, \Sigma \, \mathrm{X}$$

Sujeto a:

- $\mu_p = \sum_{i=1}^{n} x_i \mu_i = \mathrm{X}^T \mu \geq \mu_0$
- $\sum_{i=1}^{n} x_i = \mathrm{X}^T \cdot \bar{1} = 1; \; \bar{1} = (1, \ldots, 1)^T$
- $x_i \geq 0; \; i = 1, \ldots, n$

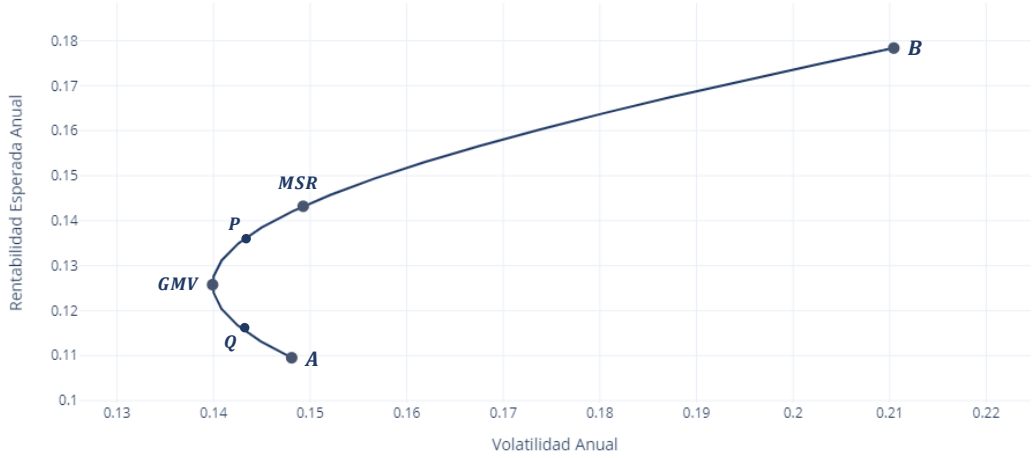
(17)

En este caso, se busca minimizar el riesgo de la cartera, sujeto a una rentabilidad esperada mayor o igual que un mínimo exigido μ_0. Igualmente, los pesos deben sumar la unidad y ser positivos.

Resolviendo el primal o el dual para cada nivel de riesgo o rentabilidad objetivo se tiene una frontera de carteras matemáticamente óptimas. Sin embargo, no todas serán eficientes en el sentido de Markowitz, puesto que algunas de ellas estarán dominadas en términos de media-varianza.

Regresando a un universo sencillo de dos activos A y B, con rentabilidad esperada, volatilidad y covarianza dada, se ha representado gráficamente en la Figura 2.2. el resultado de la optimización cuadrática de Markowitz (curva azul) que, en el caso de dos activos, coincide con el conjunto de posibilidades de inversión:

Figura 2.2. Frontera Eficiente de dos activos A y B



Fuente: *Elaboración propia. Anexo II, pág.64.*

Se puede comprobar como la cartera Q, pese a pertenecer a la curva resultante de la optimización, está dominada en términos de media-varianza por la cartera P, de mayor rentabilidad esperada e igual riesgo. Así, la Frontera Eficiente es el tramo de la curva que recoge todas las carteras dominantes en media-varianza, es decir, las carteras eficientes en el sentido de Markowitz; de tal forma que a mayor riesgo se alcance siempre una mayor rentabilidad esperada. El punto que determina el inicio de la Frontera Eficiente se corresponde con la Cartera de Mínima Varianza o *Global Minimum Variance* (GMV). Esta cartera es la que proporciona un menor riesgo de todo el conjunto de posibilidades de inversión. Se obtiene como:

$$\min_{\mathrm{X}} \sigma_p^2 = \mathrm{X}_{GMV}^T \, \Sigma \, \mathrm{X}_{GMV}$$

Sujeto a: *(18)*

- $\mathrm{X}_{GMV}^T \cdot \bar{1} = 1; \ \bar{1} = (1, \dots, 1)^T$
- $x_i \geq 0; \ i = 1, \dots, n$

Otra cartera interesante que conviene conocer, aunque posterior a Markowitz, es la Cartera de Máximo Ratio de Sharpe o *Maximum Sharpe Ratio* (MSR). Esta cartera se basa en la Ratio de Sharpe (1962), que expresa la rentabilidad esperada por unidad de riesgo de un determinado activo, una medida muy útil para evaluar la calidad de un título o cartera con relación a otras potenciales inversiones.

$$S_p = \frac{\mu_p - R_f}{\sigma_p} \qquad (19)$$

El parámetro R_f representa la rentabilidad del activo libre de riesgo, introducido por Tobin en 1958 (Anexo I). La cartera MSR es la que maximiza la rentabilidad esperada por unidad de riesgo de todo el conjunto de posibilidades de inversión:

$$\min_{\mathrm{X}} -\frac{\mu_p}{\sigma_p} = \frac{\mathrm{X}_{MSR}^T \, \mu}{\sqrt{\mathrm{X}_{MSR}^T \, \Sigma \, \mathrm{X}_{MSR}}}$$

Sujeto a: *(20)*

- $\mathrm{X}_{MSR}^T \cdot \bar{1} = 1; \ \bar{1} = (1, \dots, 1)^T$
- $x_i \geq 0; \ i = 1, \dots, n$

Finalmente, es relevante señalar que el conjunto de posibilidades de inversión con más de dos activos financieros no es la curva de la optimización sino una superficie o nube de puntos situada entre los conjuntos de posibilidades de inversión, dos a dos, de los n activos ordenados de mayor a menor rentabilidad esperada, y la curva resultante de la optimización cuadrática.

2.1.3.2. Etapa subjetiva: Actitud frente al riesgo y cartera óptima

Una vez definida la Frontera Eficiente, se trata de encontrar la cartera óptima del inversor. Para ello, se parte de una función de utilidad esperada que toma como variables la rentabilidad esperada y el riesgo en varianza de la cartera:

$$U = f\left(\mu_p, \sigma_p^2\right) \qquad (21)$$

A partir de esta función de utilidad es posible obtener una curva de indiferencia para cada nivel de satisfacción. Cada curva de indiferencia representa binomios rentabilidad-riesgo que proporcionan un mismo nivel de bienestar. Sin embargo, el modelo de Markowitz no propone ninguna herramienta al inversor para obtener su función de utilidad, sino que se limita a definir las propiedades de las curvas de indiferencia resultantes de acuerdo con la Teoría de Expectativas de Utilidad de Neumann y Morgenstern (1944):

- Las curvas de indiferencia tienen pendiente positiva conforme al principio de aversión al riesgo, de forma que un incremento del riesgo debe saldarse con un incremento de la rentabilidad esperada.
- Son cóncavas respecto al eje de ordenadas, de modo que hay una relación marginal de sustitución decreciente entre el rendimiento y el riesgo.
- Las curvas con una ordenada en el origen más elevada proveen una mayor utilidad al inversor dado el principio de racionalidad. A mayor rentabilidad para un mismo nivel de riesgo se obtiene mayor bienestar y a mayor riesgo para una misma rentabilidad esperada se obtiene menor bienestar.
- De todos los axiomas anteriores se deduce que las curvas de indiferencia son diferenciables.

Se verifican entonces las siguientes condiciones:

$$\frac{\partial U(\mu_p, \sigma_p^2)}{\partial \mu_p} > 0 \qquad (22)$$

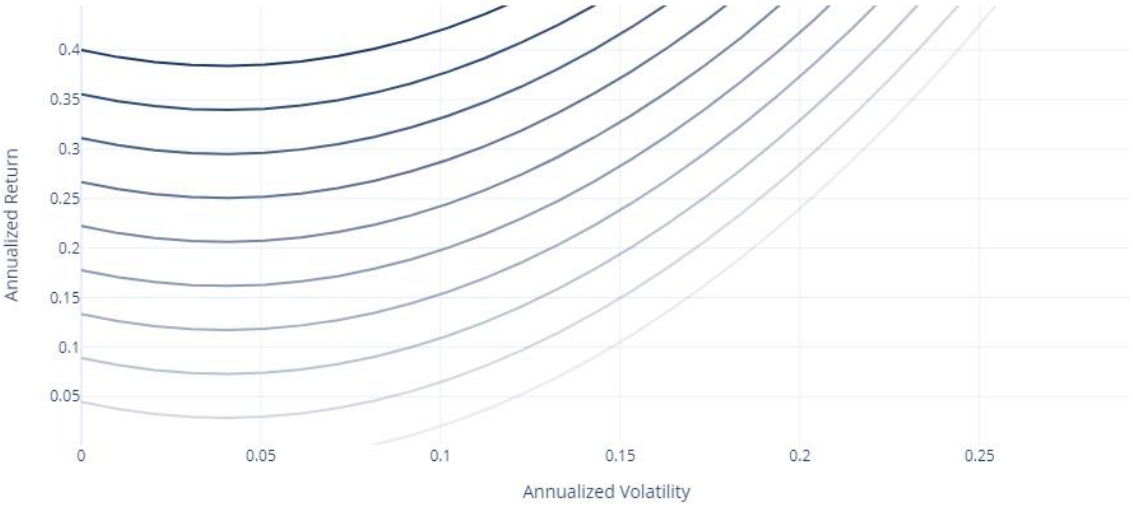
$$\frac{\partial U(\mu_p, \sigma_p^2)}{\partial \sigma_p} < 0 \qquad (23)$$

La función de utilidad más sencilla que se puede obtener en consistencia con los axiomas anteriores presenta la siguiente forma:

$$U = f\left(\mu_p, \sigma_p^2\right) = \mu_p - \frac{1}{\gamma}\sigma_p^2 \; ; \;\; \gamma > 0 \qquad (24)$$

La constante γ representa la tolerancia al riesgo del inversor. En otras palabras, mide la cantidad de riesgo que está dispuesto a asumir por una unidad adicional de rentabilidad. Gráficamente, este parámetro determina la inclinación de la curva en el espacio cartesiano. De esta forma, un valor de γ más elevado se traduce en una mayor tolerancia al riesgo y una menor pendiente de la curva. Análogamente, si el valor de γ disminuye la tolerancia al riesgo es menor y la pendiente aumenta.

Figura 2.3. Mapa de curvas de isoutilidad de U con $\gamma = 0.2$.



Fuente: *Elaboración propia. Anexo II, pág.64.*

Una vez determinado el mapa de curvas de indiferencia del inversor, la cartera óptima se obtiene en el punto de tangencia con la Frontera Eficiente:

$$\max_{x_i} U = u(\mu_p, \sigma_p^2)$$

Sujeto a: *(25)*

- $\sum_{i=1}^{n} x_i = 1$
- $x_i \geq 0;\ i = 1, \dots, n$

Para la función de utilidad (25), el problema primal adopta la siguiente forma:

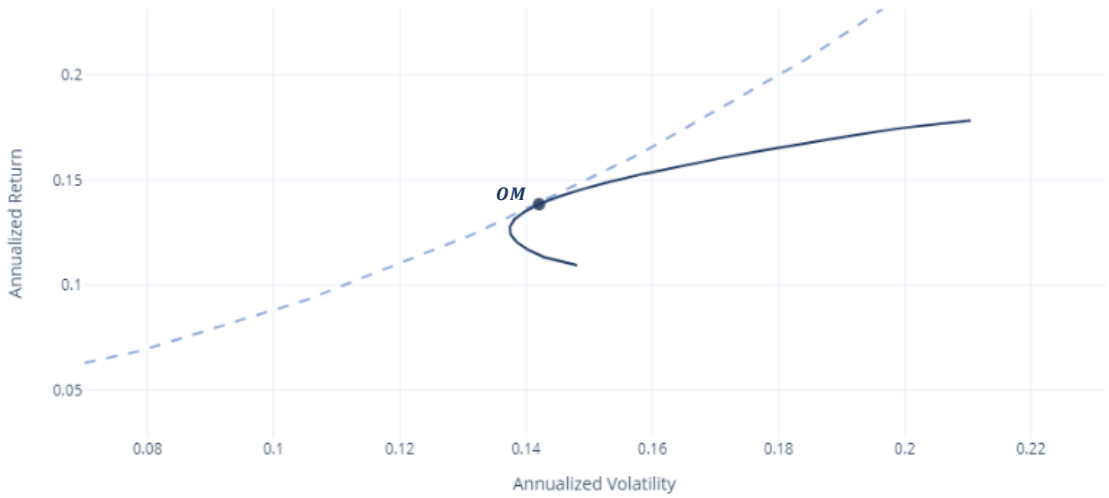
$$\max_{x_i} U = \mu_p - \frac{\sigma_p^2}{\gamma}$$

Sujeto a: *(26)*

- $\sum_{i=1}^{n} x_i = 1$
- $x_i \geq 0;\ i = 1, \dots, n$

En la Figura 2.4. se ha representado la solución al modelo de Markowitz. El trazo discontinuo representa la curva de indiferencia tangente a la Frontera Eficiente, obtenida al resolver (26). El punto de tangencia representa la cartera óptima (OM) de un inversor con función de utilidad (24) y tolerancia $\gamma = 0.2$ para los dos títulos A y B.

Figura 2.4. Cartera Óptima de Markowitz.



Fuente: *Elaboración propia. Anexo II, pág.65.*

2.1.4. Críticas al modelo de Markowitz

El modelo de Markowitz, pese a su enorme trascendencia y significación, ha sido objeto de cuantiosas críticas desde su publicación en 1952. Las más extendidas se pueden resumir en los siguientes puntos:

- En primer lugar, las premisas de un mercado perfecto, completo y eficiente son irrealizables en la práctica (Statman, 1985). En los mercados existen numerosas restricciones a la hora de configurar una cartera: comisiones de compraventa, comisiones de custodia o cambios de divisa. Estos costes de transacción son significativos y alteran las decisiones de inversión.
- Utilizar series históricas para estimar los parámetros esperados produce sesgos importantes (Michaud, 1989). Además, la varianza no siempre se considera una buena medida del riesgo para el inversor.
- Las carteras eficientes derivadas del modelo suelen estar formadas en la práctica por pocos activos de elevada rentabilidad y baja correlación, lo que implica asumir carteras de escasa diversificación y excesivo riesgo.
- Se trata de un modelo estático que no se ajusta a problemas multiperiodo. Teniendo en cuenta la elevada sensibilidad de la mayoría de los parámetros al horizonte temporal, los resultados que se obtienen al evaluar el modelo en cada periodo suelen ser inconsistentes, lo que dificulta la deducción de estrategias de inversión sólidas a medio y largo plazo. Por ello, el modelo de Markowitz no se considera una herramienta de gestión o monitoreo de carteras, sino un instrumento de selección de activos o *stock picking*.
- La segunda etapa del modelo es impracticable por la falta de herramientas para deducir una función de utilidad coherente con el inversor. En la praxis, lo más habitual es recurrir a carteras predefinidas como la GMV o la MSR, así como calcular la cartera de menor riesgo para una rentabilidad objetivo.

Aunque algunas de estas debilidades se han intentado atajar dentro y fuera de la Teoría Moderna de Carteras (MPT), ninguna aproximación ha logrado ser aún lo suficientemente integral en el tratamiento del problema como para resolver todos los dilemas planteados.

2.2. Deep Learning

El Deep Learning (DL) es un segmento del Machine Learning (ML) y una rama de la Inteligencia Artificial (IA). La IA es un concepto amplio sobre la capacidad de un sistema para imitar funciones cognitivas del ser humano como percibir, razonar, aprender o resolver problemas (Kaplan y Haenlein, 2009). Aunque habitualmente se presenta como un campo disruptivo y contemporáneo, lo cierto es que se trata de una tecnología anterior incluso al propio Modelo de Markowitz. En 1943, los investigadores Warren McCulloch y Walter Pitt presentaron la primera neurona artificial, inspirada en los planteamientos lógico-matemáticos de Alan Turing. Esta innovación se considera el comienzo de la IA. Sin embargo, no fue hasta 1956 cuando John McCarthy, Marvin Minsky y Claude Shannon acuñaron oficialmente el término en la conferencia de Dartmouth para referirse a la ingeniería de elaborar máquinas inteligentes. La efervescencia actual en torno a esta disciplina se explica principalmente por la rápida evolución del hardware a partir de los años 90, crucial para trasladar a la praxis los importantes avances teóricos del siglo XX.

Mientras la IA es un campo extenso que abarca todo tipo de funciones cognitivas, el ML es el área particular se encarga de estudiar y replicar el aprendizaje humano. Se dice que una máquina aprende cuando es capaz de mejorar su desempeño en una tarea sin haber sido programada específicamente para ello. Por ejemplo, un brazo robótico diseñado para levantar y trasladar una carga siempre que se den una serie de prerrequisitos podría considerarse una IA. No obstante, si esa misma máquina fuese capaz de aprender por sí sola cuándo transportar la carga mediante estímulos percibidos del entorno, pertenecería al ámbito específico del ML.

El primer algoritmo de ML reconocido se remonta a 1952, cuando el informático estadounidense Arthur Samuel creó un programa capaz de jugar a las damas y mejorar el resultado tras cada partida. El mismo año, Claude Shannon presentó un ratón artificial conocido como *Theseus* que logró atravesar un laberinto imantado recordando la ruta más eficiente mediante prueba y error. Estas dos innovaciones supusieron el punto de partida para la creación de algoritmos más sofisticados, capaces de hacer frente a problemas de mayor complejidad.

Así, en enero de 2013, los investigadores de *DeepMind* crearon un sistema capaz de aprender desde cero cualquier juego de Atari sin tener ningún conocimiento previo sobre sus reglas y superar el desempeño humano. Más tarde, en 2017, repitieron su hazaña con el lanzamiento de *AlphaGo*, un agente capaz de batir al dieciocho veces campeón del mundo de Go, Lee Sedol. El Go es considerado uno de los juegos de mesa más complejos del mundo, hasta cincuenta y dos órdenes de magnitud por encima del ajedrez. Estos experimentos han sentado las bases de una nueva rama del ML que ha permitido elevar su potencial exponencialmente: el Deep Learning. El DL abarca una serie de algoritmos para modelar abstracciones de alto nivel con arquitecturas computacionales que admiten transformaciones no lineales múltiples e iterativas sobre los datos. Estas arquitecturas se denominan Redes Neuronales Profundas o *Deep Neural Networks* (DNN) y funcionan de forma similar al cerebro humano.

2.2.1. Conceptos básicos

En este epígrafe se van a introducir los aspectos fundamentales que subyacen a una arquitectura de DL, especialmente los más relevantes para las Redes LSTM.

2.2.1.1. Neurona artificial

La neurona artificial es la unidad mínima de cómputo de una Red Neuronal. Las neuronas actuales se basan en el Perceptrón Simple de Frank Rosenblatt (1958), una sofisticación de la neurona McCulloch-Pitts (1943). En la práctica, se trata de dos funciones matemáticas que proporcionan una salida y a partir de un vector de datos de entrada $\mathrm{Z} = (z_1, \dots\ z_n)$ procedente del entorno u otra neurona.

La neurona artificial está compuesta por cuatro elementos principales:

- Los pesos sinápticos, $\mathrm{W} = (w_1, \dots, w_n)$: representan cuánto afecta cada dato de entrada a la salida de la neurona.
- Sesgo o *bias*, b: Es una constante que no depende del vector de entrada. Los pesos sinápticos y el sesgo son los parámetros que las neuronas ajustan para generar la salida deseada en función de los datos de entrada. Suponen la base del aprendizaje neuronal.

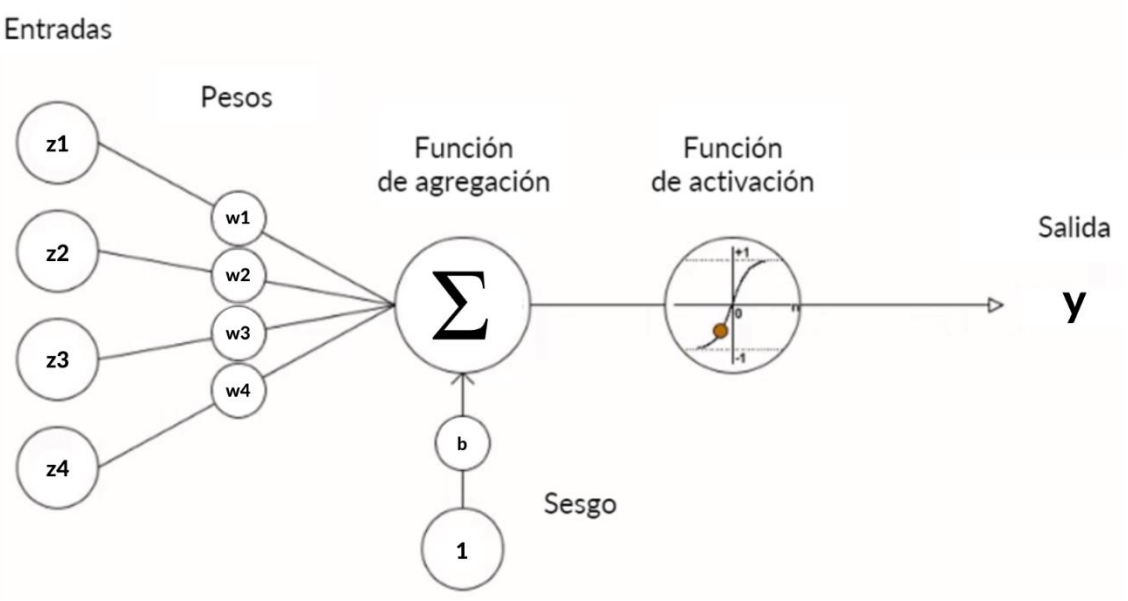
- Función de agregación: Determina la forma en la que los pesos sinápticos afectan a las variables de entrada y cómo se introduce el sesgo. En general, se utiliza la suma ponderada. El resultado de la función de agregación se suele denominar señal, s. Hasta ahora, la neurona se comporta de forma muy similar a un modelo clásico de regresión lineal múltiple:

$$s = b + w_1 z_1 + w_2 z_2 + \cdots + w_n z_n = b + \sum_{i=1}^{n} w_i \, z_i = b + \mathrm{W}^T \mathrm{Z} \qquad (27)$$

- Función de activación, $f(s)$: Es el elemento más relevante de las neuronas artificiales. Introduce deformaciones no lineales en el modelo para que sea posible aproximar patrones complejos y funciones que no son linealmente separables. El resultado de esta función es la salida de la neurona, de modo que $f(s) = y$.

En la Figura 2.5. se ha representado la estructura de la neurona. Es interesante ver como los datos de entrada se transforman siguiendo un flujo unidireccional hacia la salida. En Deep Learning, este tipo de algoritmo se conoce como propagación hacia delante o *forward propagation*.

Figura 2.5. Esquema del Perceptrón Simple.



Fuente: *Elaboración propia.*

2.2.1.2. Función de activación

Las propiedades más deseadas en una función de activación son las siguientes:

- No lineal: Con la no-linealidad se introducen irregularidades en el modelo que permiten aproximar con precisión una gran variedad de funciones.
- Diferenciable: Es necesario que la función de activación sea diferenciable en su dominio para emplear métodos de optimización basados en derivadas como el descenso del gradiente.
- Acotada: Es deseable que la función de activación acote los valores de salida o los transforme en un determinado rango.
- Simplicidad: Se buscan funciones de activación sencillas para reducir el coste computacional de la red.
- Por último, es preferible que se aproximen a la identidad en el origen para reducir el impacto de la aleatoriedad en los pesos iniciales.

Como ejemplo subóptimo de una función de activación es habitual presentar la función identidad. Se trata de una función lineal que devuelve el mismo valor que su entrada:

$$f(s) = s \tag{28}$$

Esta función no es adecuada para las redes neuronales puesto que no introduce no-linealidades. Una red neuronal compleja con muchas capas y neuronas cuya función de activación es la identidad equivale a un Perceptrón Simple.

Otro ejemplo es la función signo, utilizada por McCulloch-Pitts y Frank Rosenblatt en sus neuronas originales. Esta función proporciona una salida binaria (0, 1) que permite ajustar la activación de la neurona en función del sesgo, es decir, para qué valores de $\mathrm{W}^T\mathrm{Z}$ la salida tomará un valor 1. En este caso, se dice que el sesgo actúa como umbral o grado de inhibición de la neurona:

$$f(s) = sgn(s) = \begin{cases} 1, & \mathrm{W}^T\mathrm{Z} - b > 0 \\ 0, & \mathrm{W}^T\mathrm{Z} - b \leq 0 \end{cases} \tag{29}$$

Figura 2.6. Función signo.

Fuente: *Elaboración propia. Anexo II, pág.65.*

El mayor problema de la función signo y, en general, de las salidas binarias, es la necesidad de emplear más de una neurona para aprender funciones que no sean linealmente separables. Por esta razón, en las últimas décadas se han propuesto funciones de activación alternativas que superan dicha dificultad.

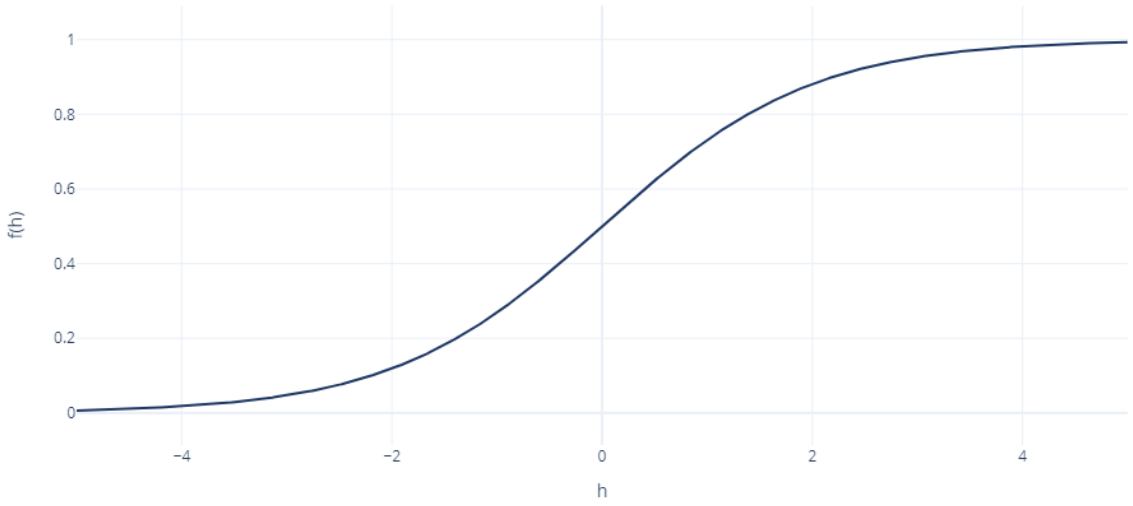
Dos ejemplos son las funciones de activación empleadas en las Redes Neuronales LSTM: la función tangente hiperbólica y la función sigmoide o logística. Ambas son muy similares, estrictamente crecientes y derivables, de elevada interpretabilidad y adecuadas para datos de entrada basados en series temporales.

En las Figuras 2.7. y 2.8. se puede comprobar como a partir de un determinado valor de la señal, tanto positivo como negativo, ambas funciones se saturan, de forma que su salida converge siempre al máximo o al mínimo de su rango. Para lidiar con este problema es fundamental normalizar el vector de datos de entrada en un rango apropiado, normalmente, entre cero y uno.

En particular, la función sigmoide o logística está acotada en el rango (0, 1). Esta propiedad la hace idónea para problemas de clasificación:

$$f(s) = \text{sigmoid}\, s = \frac{1}{1 + e^{-x}} \tag{30}$$

Figura 2.7. Función sigmoide o logística.

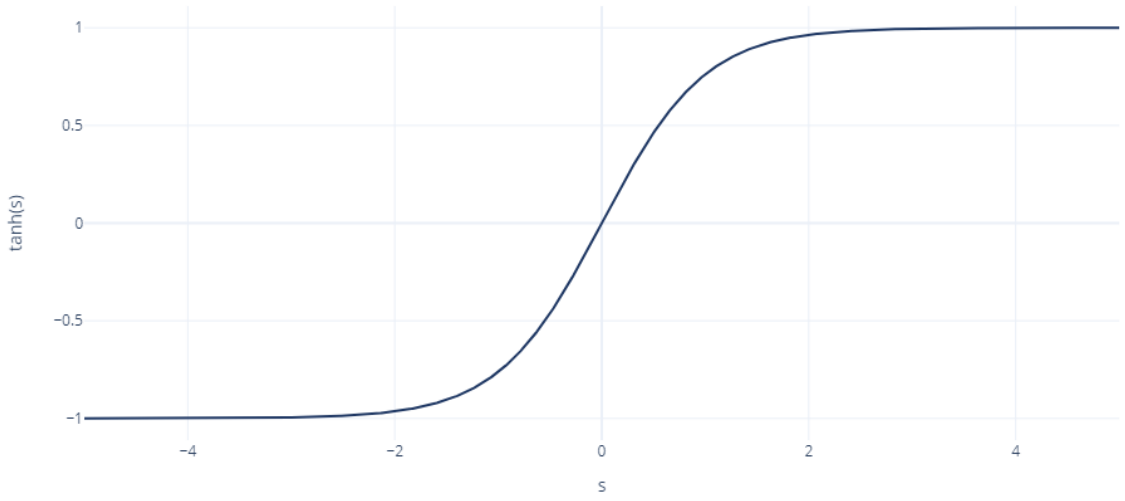


Fuente: *Elaboración propia. Anexo II, pág.66.*

La tangente hiperbólica está acotada en el rango (-1, 1) y, por tanto, centrada en cero. Esta simetría hace que sea más eficiente que la función sigmoide, pero a cambio se satura con mayor facilidad.

$$f(s) = \tanh s = \frac{\sinh s}{\cosh s} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{31}$$

Figura 2.8. Función tangente hiperbólica.



Fuente: *Elaboración propia. Anexo II, pág.66.*

2.2.1.3. Red Neuronal

Una Red Neuronal es simplemente una combinación de neuronas. La topología o arquitectura de la red hace referencia a cómo se organizan las neuronas dentro de la misma. Lo más habitual es trabajar con redes de propagación hacia delante y densamente conectadas. Esto significa que las neuronas están unidas por capas, de forma que toman como vector de entrada las salidas de las neuronas situadas en la capa inmediatamente anterior y propagan su salida hacia todas las neuronas de la siguiente capa.

Existen tres tipos de capa según su ubicación en la Red Neuronal:

- *Input layer* o capa de entrada: Es la capa que conecta los datos de entrada con la Red Neuronal. Cada una de las neuronas que la integra posee tantas entradas como variables tiene el modelo.
- *Output layer* o capa de salida: Recibe de entrada las salidas de las neuronas situadas en la capa inmediatamente anterior y proporciona la salida final de la red. Es crucial que el número de neuronas de esta capa y el tipo de salida que ofrecen se adapten al problema tratado.
- *Hidden layers* o capas ocultas: Son todas las capas situadas entre la capa de entrada y la de salida. El número de capas ocultas y neuronas que puede tener una Red Neuronal es arbitrario e ilimitado, lo que permite al modelo ajustarse a cada problema concreto y aproximar funciones complejas. De esta propiedad surge el término Deep Learning.

No obstante, en este punto es importante introducir el Teorema de Aproximación Universal de Hornik. Este teorema establece que una Red Neuronal con una sola capa oculta puede aproximar bien cualquier función continua si se le proporciona un número suficiente de neuronas. Conforme aumenta el número de capas ocultas también crece la complejidad y el coste computacional de la Red Neuronal, lo que puede derivar en problemas de aprendizaje y resultados imprecisos. Por ello, es importante ser meticuloso a la hora de diseñar la red y no proporcionar más capas y neuronas de las realmente necesarias para hacer frente al problema que se busca resolver.

2.2.1.4. Entrenamiento: función de coste y optimizador

El entrenamiento de una Red Neuronal hace referencia al aprendizaje por parte de todas sus neuronas de los pesos sinápticos más adecuados para aproximar la función objetivo. Esta etapa se suele traducir en un problema de minimización del error entre el valor predicho y el valor buscado. Si el valor buscado es conocido se habla de Aprendizaje Supervisado o *Supervised Learning* (SL). En caso contrario, se estaría ante un escenario de Aprendizaje No Supervisado o *Unsupervised Learning* (UL). En este proyecto, el aprendizaje será de tipo supervisado.

Para calcular el error se emplea una función continua conocida como función de coste o pérdida. Su objetivo es medir cómo se comporta la Red Neuronal frente al problema que se busca resolver. Cuando la salida esperada de la red es un escalar, se suele escoger como función de coste el Error Cuadrático Medio o *Mean Square Error* (MSE). Siendo $\hat{y}$ el valor estimado e y el valor real, se tiene:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (32)$$

Como $\hat{y}$ depende de todos los pesos sinápticos y transformaciones no lineales de la Red Neuronal, la función de coste será no convexa y no será posible encontrar mínimos de forma analítica. Por tanto, es necesario establecer un algoritmo que determine cómo se va a realizar la optimización. Dicho algoritmo recibe el nombre de optimizador o método de optimización.

Las primeras Redes Neuronales empleaban un enfoque de fuerza bruta para llevar a cabo esta tarea, de forma que se asignaban los pesos sinápticos de cada neurona aleatoriamente y, tras una gran cantidad de pruebas, se escogía la configuración más eficaz de acuerdo a la función de pérdida. Como cabe esperar, se trata de un procedimiento subóptimo y muy costoso computacionalmente.

En la actualidad, casi todos los métodos de optimización se basan en el descenso del gradiente. Se llama gradiente al vector de derivadas parciales de la función de coste con respecto a las m neuronas de la red y a sus n pesos sinápticos, recogidos en el vector $\mathrm{W} = (b_1, w_{11}, \dots, b_m, w_{mn})$. Para una función de coste g:

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_0} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix} \tag{33}$$

En esencia, el gradiente representa la dirección en la que asciende la pendiente en un punto de la función de coste tomando pequeñas variaciones de los pesos. Dado que el objetivo no es ascender sino descender por dicha función hasta llegar a un mínimo, lo que interesa es dar pequeños pasos en dirección opuesta al gradiente. La longitud de los pasos viene determinada por un parámetro α conocido como tasa de aprendizaje. De este modo, los pesos se irán actualizando iterativamente mediante la siguiente expresión, basada en el Teorema de Taylor:

$$\mathrm{W}' = \mathrm{W} - \alpha \, \nabla g \tag{34}$$

Este proceso se lleva a cabo con un algoritmo conocido como retropropagación del error o *backpropagation*, repartiendo el error entre las neuronas que integran la Red Neuronal según su grado de implicación, comenzando por las últimas capas y actualizando los pesos sinápticos a medida que se retrocede hacia las primeras. Cuando el gradiente se hace cero o alcanza un valor cercano a cero conocido como criterio de parada, se habrá llegado a una zona de mínimo coste.

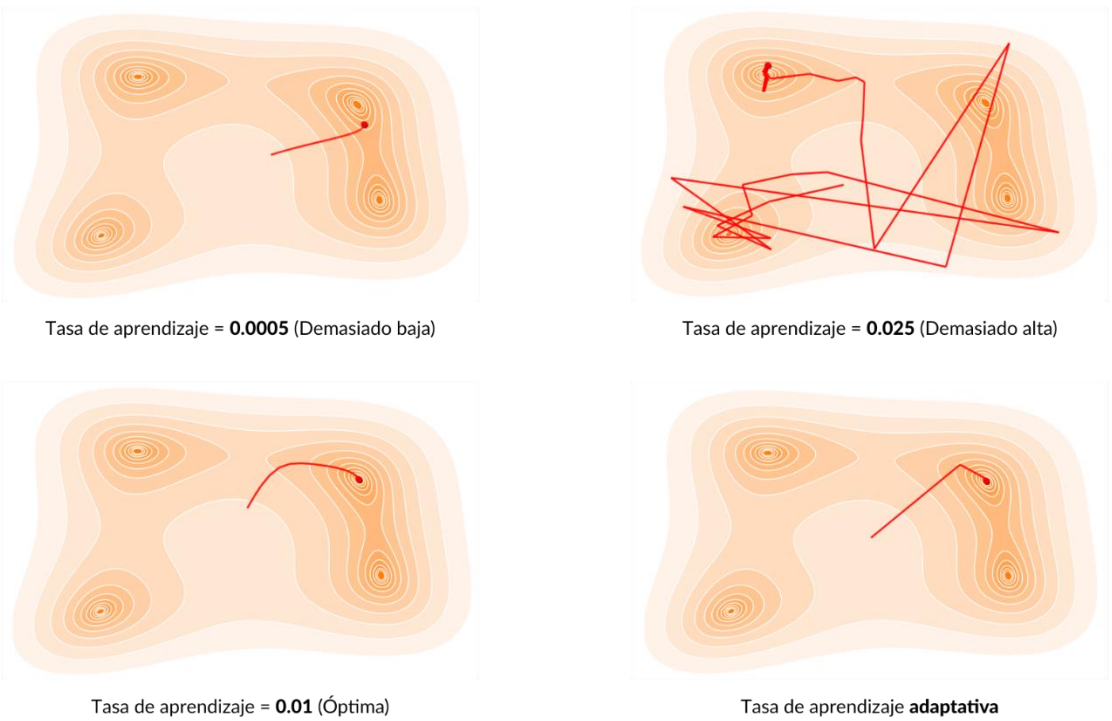
No obstante, uno de los principales problemas del descenso del gradiente es la existencia de mínimos locales que impiden la convergencia al mínimo global de la función de pérdida. Esto hace que la optimización sea muy sensible al valor inicial de los pesos sinápticos, que normalmente se escoge aleatoriamente para inicializar el algoritmo.

La tasa de aprendizaje α es un elemento clave para alcanzar con éxito el mínimo global de la función de coste. Una tasa de aprendizaje demasiado baja hará que el descenso se detenga en puntos subóptimos. Además, el coste computacional será muy elevado ya que harán falta muchas actualizaciones de los pesos sinápticos para alcanzar un mínimo. En cambio, una tasa de aprendizaje demasiado alta hará que la trayectoria del descenso sea irregular, con rebotes zigzagueantes que no permitirán al algoritmo converger a una zona de mínimo coste.

En la práctica, no es recomendable mantener la tasa de aprendizaje constante. En su lugar, se utilizan sofisticaciones del descenso del gradiente que permiten un ajuste dinámico de la tasa de aprendizaje conforme se aproxima al óptimo. Entre las más implementadas destacan el optimizador SGD (*Stochastic Gradient Descent*), el optimizador ADAM (*Adaptative Moment Estimation*) y el optimizador RMSprop, basado en las medias móviles del gradiente; las cuales permiten alcanzar el óptimo con un menor número de actualizaciones y abandonar mínimos locales en busca de zonas con un coste más bajo.

En la Figura 2.6. se ha representado el efecto de la tasa de aprendizaje sobre el descenso del gradiente. La superficie naranja representa las curvas de nivel de la función de coste, de forma que las zonas más oscuras simbolizan los puntos más bajos de la función. Como se puede apreciar, la convergencia al óptimo es muy sensible a la tasa de aprendizaje escogida.

Figura 2.9. Efecto de la tasa de aprendizaje sobre el descenso del gradiente.



Fuente: *Elaboración propia con ilustraciones de Gradient Descent Simulator (Ben Frederickson).*

Finalmente, es necesario introducir los conceptos de iteración, época y lote. A lo largo de este epígrafe, se ha repetido la idea de actualizar los pesos sinápticos de la Red Neuronal. Cada una de estas actualizaciones recibe el nombre de iteración, puesto que implica una ejecución completa del algoritmo de optimización. Para poner en marcha el algoritmo es necesario suministrar a la red un conjunto de datos de entrada conocido como set de entrenamiento. Cada vez que el set de entrenamiento completo se propaga hacia delante por todas las capas de la red y se actualizan los pesos sinápticos se dice que ha transcurrido una época.

En este punto, se podría pensar que una época es equivalente a una iteración. No obstante, los sets de entrenamiento suelen contener una gran cantidad de datos, haciendo que sea muy lento y computacionalmente costoso actualizar los pesos sinápticos una única vez por cada época. Por ello, lo más habitual es dividir el set de entrenamiento en lotes. Un lote no es más que una submuestra de los datos de entrada, de modo que el número de iteraciones es equivalente al número de lotes necesarios para completar una época.

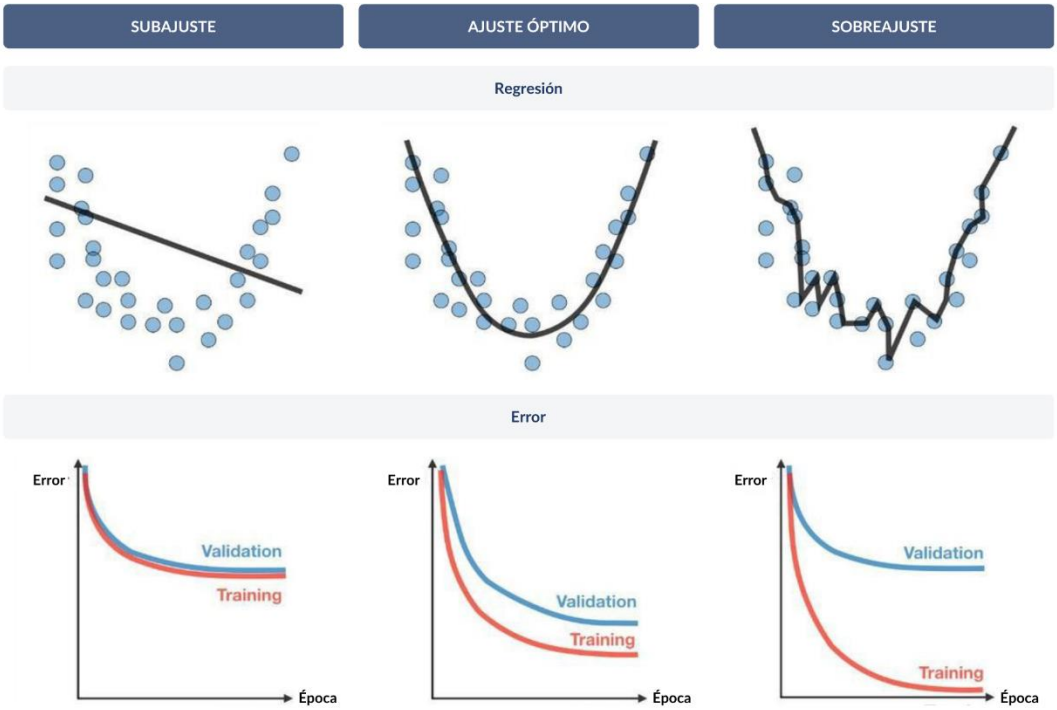
2.2.1.5. Validación: sobreajuste y subajuste

La validación hace referencia a la etapa de testeo de la Red Neuronal. En esencia, consiste en proporcionar a la red datos de entrada que no ha procesado durante el aprendizaje, el set de validación; para poder analizar su comportamiento sobre valores desconocidos. Habitualmente, lo que se trata de evitar es el sobreajuste u *overfitting* y el subajuste o *underfitting*.

Se dice que una Red Neuronal está sobreajustada cuando es capaz de modelar el set de entrenamiento con mucha precisión, pero su comportamiento empeora de forma notable en el set de validación. Normalmente, el sobreajuste está causado por una sobreexposición al ruido en los datos de entrenamiento. El ruido es una irregularidad aleatoria exclusiva de cada conjunto de datos que puede desviar la atención de la red. Esto significa que, en cierto punto del entrenamiento, la Red Neuronal puede dejar de mejorar su capacidad general de resolver el problema y comenzar a aprender patrones o propiedades demasiado específicos.

Por otro lado, la red también puede incurrir en un subajuste. Esto sucede cuando el modelo no ha aprendido correctamente con el set de entrenamiento, debido principalmente al uso de topologías erróneas o demasiado sencillas para resolver problemas complejos, la insuficiencia o inadecuación de las variables de entrada para explicar el comportamiento general de la salida o a una duración deficiente del entrenamiento. En la Figura 2.7. se han representado estas ideas gráficamente:

Figura 2.10. Sobreajuste y subajuste en problemas de regresión.



Fuente: *Elaboración propia con ilustraciones de VIU.*

Para corregir estos problemas se pueden llevar a cabo dos tipos de acciones. La más habitual es modificar los hiperparámetros del modelo. Los hiperparámetros son las variables que deben ajustarse durante el entrenamiento para optimizar el desempeño de la red y lograr que su configuración se adapte lo máximo posible al problema planteado. Los más relevantes se han ido introduciendo en los epígrafes anteriores: el número de capas y neuronas, la función de activación, el número de épocas y lotes de entrenamiento, la función de coste, el optimizador y la tasa de aprendizaje. Otra opción para corregir el sobreajuste del modelo son las técnicas de regularización. Entre las más sencillas y populares se encuentran el abandono o *dropout* y la parada temprana o *early stop*.

2.2.2. Tipos de Redes Neuronales

En la actualidad, existen varias arquitecturas que se han demostrado muy eficaces y robustas para resolver una amplia gama de problemas. En concreto, para lidiar con series de datos temporales como los precios diarios de los activos financieros, se emplean las Redes Neuronales Recurrentes o *Recurrent Neural Networks* (RNN). Las Redes de Memoria a Corto y Largo Plazo o *Long-Short Term Memory* (LSTM) que se van a implementar en este proyecto son una sofisticación de las anteriores.

2.2.2.1. RNN

Las RNN son un tipo de Red Neuronal específicamente diseñadas para tratar con series temporales o datos secuenciales. Estrictamente, no tienen una arquitectura por capas, aunque se puede pensar en ellas de forma intuitiva como una sola capa oculta que recoge los datos de la capa de entrada y vuelca el resultado en la capa de salida. Sin embargo, esta capa oculta no estaría formada por neuronas sino por celdas, en concreto, celdas recurrentes.

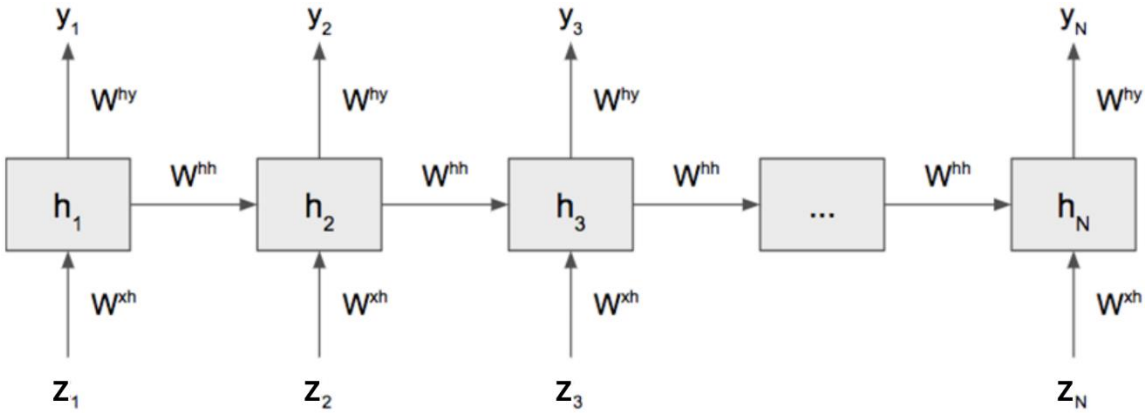
Las celdas recurrentes funcionan de forma similar a un Perceptrón Simple al que se le añade un nuevo elemento: el estado oculto h. Para cada periodo de tiempo o posición en la secuencia t, existe una celda recurrente que recibe la variable de entrada z_t, propaga un estado oculto h_t a la siguiente celda y genera una salida y_t:

$$h_t = F(b_h + w_{hh}h_{t-1} + w_{hx}z_t) \quad (35)$$

$$y_t = G(b_y + w_{hy}h_t) \quad (36)$$

En la Figura 2.11. se ha representado la topología de una RNN. Esta configuración permite establecer conexiones arbitrarias entre las neuronas de la red que facilitan la detección de ciclos y la consciencia de temporalidad. Por ello, se dice que tienen memoria. Sin embargo, uno de los inconvenientes de las RNN es la amnesia a largo plazo. A medida que los datos se propagan por las celdas recurrentes, la salida proporcionada por las celdas más lejanas va perdiendo peso o importancia en el estado oculto.

Figura 2.18. RNN.

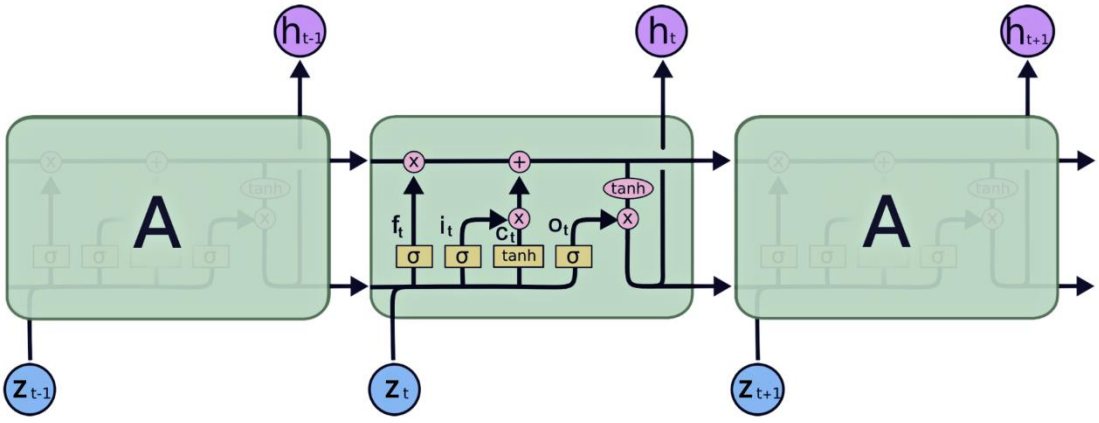


Fuente: *Medium.*

2.2.2.2. LSTM

Las redes LSTM presentan una topología más compleja, diseñada para resolver el problema de amnesia a medio plazo. Cada celda LSTM está compuesta por un circuito de cuatro Redes Neuronales de propagación hacia delante y densamente conectadas que regulan el flujo de información desde la entrada de la celda hasta un nuevo elemento: la cinta de estado o memoria de la red. En la Figura 2.19. se ha esquematizado el funcionamiento de una red LSTM. Como se puede apreciar, cada celda tiene dos flujos: una cinta inferior que recoge la variable de entrada z_t y el estado oculto de la celda anterior h_{t-1}, y una cinta superior, la cinta de estado.

Figura 2.19. Red LSTM.



Fuente: *ResearchGate.*

Las entradas z_t y h_{t-1} se procesan simultáneamente dentro de cada celda LSTM de la siguiente forma:

- Compuerta de olvido o *forget gate*: Esta primera Red Neuronal tiene como objetivo determinar si la información recibida es relevante para el modelo o si debe ser olvidada. Para ello, emplea una función de activación sigmoidal que actúa como una válvula. Una salida cercana a cero impide el flujo de la información procedente de la celda anterior a la memoria de la red.

$$f_t = \text{sigmoid}(b_f + w_{fh}h_{t-1} + w_{fz}z_t) \quad (37)$$

- Compuerta de actualización o *update gate:* Su misión es incorporar nueva información a la memoria de la red. Para ello, emplea dos Redes Neuronales con función de activación sigmoidal y tangente hiperbólica:

$$i_t = \text{sigmoid}(b_i + w_{ih}h_{t-1} + w_{ix}z_t) \quad (38)$$

$$c_t = \tanh(b_c + w_{ch}h_{t-1} + w_{cx}z_t) \quad (39)$$

Siendo e_{t-1} el vector de información contenido en la memoria de la red a la entrada de la celda LSTM del periodo t, su vector de salida e_t será:

$$e_t = f_t e_{t-1} + i_t c_{t-1} \quad (40)$$

- Compuerta de salida u *output gate:* Su objetivo es calcular el nuevo estado oculto h_t. Por un lado, se aplica la función tangente hiperbólica al vector e_t para normalizar sus valores en el rango (-1, 1). A continuación, z_t y h_{t-1} se trasladan a otra función de activación sigmoidal:

$$e'_t = \tanh(e_t) \quad (41)$$

$$o_t = \text{sigmoid}(b_o + w_{oh}h_{t-1} + w_{ox}x_t) \quad (42)$$

Finalmente, el valor del nuevo estado oculto será:

$$h_t = o_t e'_t \quad (43)$$

Los parámetros b_f, w_{fh}, w_{fz}, b_i, w_{ih}, w_{iz}, b_c, w_{ch}, w_{cz}, b_o, w_{oh} y w_{oz} son, en este caso, el objetivo de aprendizaje de cada celda LSTM durante el entrenamiento.

3. PROCEDIMIENTO Y RESULTADOS

En esta sección se va a plantear y resolver el caso práctico de gestión de carteras.

3.1. Descripción del problema

Suponiendo un mercado integrado únicamente por los treinta activos con riesgo del *Dow Jones Industrial Average* (DJIA), se pide construir a 1 de enero de 2021 la mejor cartera de inversión posible a partir de una submuestra aleatoria de cinco títulos. Para ello, solo es posible obtener el precio de cierre diario de los activos desde el año 2000. Además, se sigue una política de rebalanceo anual, de tal forma que no es posible reajustar el peso asignado a cada uno de los títulos hasta el 1 de enero de 2022. Las operaciones de cobertura y especulación mediante la venta a corto no están permitidas y los criterios que determinarán la calidad de la cartera al finalizar el año son: la rentabilidad anual (4), el riesgo en desviación típica (5), la Ratio de Sharpe (19) y el número de activos con peso no nulo.

3.2. Instrumentos utilizados

El problema descrito anteriormente se ha resuelto en *Python*, uno de los lenguajes de programación más populares por su carácter flexible, multiparadigma y de alto nivel. Cuenta además con una amplia variedad de librerías en código abierto que permiten el diseño, implementación y entrenamiento de Redes Neuronales. Para ello, se ha recurrido a *Scipy*, *Keras* y *Tensorflow*. El código fuente se encuentra en el Anexo II del documento.

3.3. Obtención de los datos

En primer lugar, se pide seleccionar al azar cinco títulos del DJIA. Para asegurar la aleatoriedad de dicha submuestra, se ha empleado la librería *Random*, obteniendo:

- *Cisco Systems, Inc.* (NASDAQ: CSCO),
- *The Walt Disney, Co.* (NYSE: DIS),
- *Intel, Corp.* (NASDAQ: INTC),
- *Minnesota Mining and Manufacturing, Co.* (NYSE: MMM)
- *Nike, Inc.* (NYSE: NKE).

De los activos escogidos, CSCO e INTC pertenecen al sector tecnológico, DIS al entretenimiento, MMM al sector industrial y NKE al consumo cíclico. Como cabía esperar de un índice heterogéneo como el DJIA, los títulos seleccionados no están concentrados en una industria, por lo que su evolución histórica en el mercado ha sido bastante dispar. Tradicionalmente, las acciones tecnológicas proporcionan mayor rentabilidad a cambio de mayor volatilidad, mientras que las industriales acostumbran a ser más estables y predecibles.

Una vez seleccionados los cinco activos, se ha empleado la API pública de *Yahoo Finance* para obtener los precios de cierre históricos. En la Figura 3.1. se muestra el resultado de dicha consulta en formato tabular:

Figura 3.1. Precio de cierre de CSCO, DIS, INTC, MMM y NKE (2000-2021).

Date	CSCO	DIS	INTC	MMM	NKE
2000-01-03	39.074158	23.115252	26.234377	26.434858	4.743142
2000-01-04	36.882019	24.469286	25.009356	25.384460	4.484426
2000-01-05	36.769024	25.484808	25.216660	26.119749	4.743142
2000-01-06	36.158848	24.469286	23.746632	28.220520	4.718502
2000-01-07	38.283176	24.082420	24.726662	28.780737	4.718502
...	...	...	...	...	...
2021-12-23	61.494038	153.630005	50.522724	171.561081	164.884323
2021-12-27	62.619656	152.800003	51.143059	173.257355	166.785248
2021-12-28	62.728268	155.199997	50.965820	174.179047	165.630753
2021-12-29	63.152836	154.869995	51.034748	174.934052	167.979568
2021-12-30	62.817131	155.929993	50.946129	174.179047	166.695694

5535 rows × 5 columns

Fuente: *Elaboración propia con datos de Yahoo Finance.*

La tabla anterior es en realidad una matriz de precios 5535×5, donde cada fila representa un periodo t y cada columna un activo i. De acuerdo con el enunciado del caso, la serie histórica comienza en el año 2000. Los datos de 2021 no son conocidos al formarse las carteras a 1 de enero. Sin embargo, se han incluido en la consulta para poder evaluar las carteras al concluir el ejercicio.

Por esta razón, se ha dividido la matriz precios en:

- Set de entrenamiento (2000-2019): empleado para obtener las carteras basadas en la MPT y para entrenar a la Red LSTM, Es una matriz 5031×5.
- Set de validación conocido (2020): usado para ajustar los hiperparámetros de la Red LSTM. Es una matriz 253×5.
- Set de validación desconocido (2021): utilizado para evaluar y comparar las carteras y estrategias de inversión propuestas. Es una matriz 251×5.

3.4. Carteras basadas en la MPT

En esta sección se van a obtener tres carteras basadas en la MPT. Como se ha mencionado en el capítulo anterior, la segunda etapa del Modelo de Markowitz es difícil de implementar en la práctica, ya que la función de utilidad de cada inversor es desconocida y bastante complicada de formular. Por tanto, se van a obtener las siguientes carteras de referencia:

- Cartera de Mínima Varianza o *Global Minimum Variance* (GMV): ofrece la menor volatilidad entre todas las carteras situadas en la Frontera Eficiente de Markowitz. No tiene en cuenta la rentabilidad.
- Cartera de Máximo Ratio de Sharpe o *Maximum Sharpe Ratio* (MSR): ofrece la mayor rentabilidad esperada por unidad de riesgo entre todas las carteras situadas en la Frontera Eficiente de Markowitz.
- Cartera de Máxima Rentabilidad o *Maximum Return* (MR): proporciona la mayor rentabilidad esperada de todas las carteras situadas en la Frontera Eficiente de Markowitz. Equivale a invertir el 100% del capital en el título de mayor rentabilidad esperada.

3.4.1. Transformación de los datos

La metodología de Markowitz requiere convertir cada serie histórica de precios en rentabilidades diarias. Para ello, se ha aplicado (1) en el set de entrenamiento para cada título y periodo, comenzando por el segundo.

3.4.2. Obtención de las carteras

El vector de pesos de todas las carteras que se van a construir adopta la siguiente forma: $\mathrm{X} = (x_{CSCO}, x_{DIS}, x_{INTC}, x_{MMM}, x_{NKE})$. A partir de las rentabilidades diarias de cada activo, se ha calculado la volatilidad esperada diaria (3) y se ha anualizado (5). La rentabilidad esperada anualizada se ha obtenido directamente mediante (4).

En la Figura 3.2 se muestran los resultados alcanzados. NKE es el activo con mayor rentabilidad esperada. Como cabía esperar, los activos de mayor variabilidad son los tecnológicos, CSCO e INTC, aunque en este caso son también los de menor rentabilidad esperada, principalmente debido al estallido de la burbuja *puntocom* entre el año 2000 y 2001, lo que pone de manifiesto la alta sensibilidad de estas medidas al horizonte temporal que se haya considerado. El activo de menor riesgo es el perteneciente al sector industrial, MMM, según lo

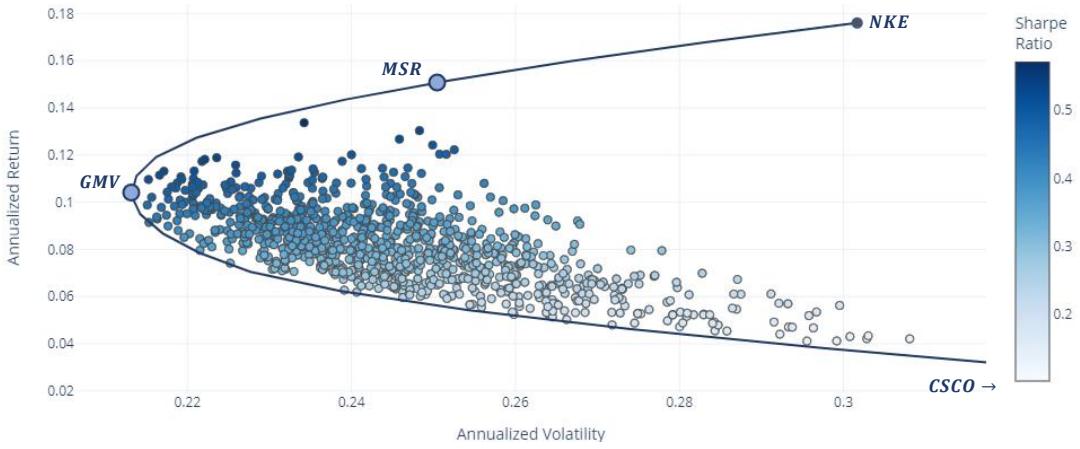
Figura 3.2. Rentabilidades y desviaciones esperadas anualizadas (2000-2019).

	CSCO	DIS	INTC	MMM	NKE
Rentabilidad Esperada (%)	0.65	9.62	3.88	9.49	16.45
Volatilidad Esperada (%)	38.47	29.68	36.75	23.05	29.59

Fuente: *Elaboración propia.*

Por tanto, la Frontera Eficiente quedará definida entre la Cartera GMV y NKE, tal y como se muestra en la Figura 3.3.

Figura 3.3. Frontera eficiente de CSCO, DIS, INTC, MMM y NKE.



Fuente: *Elaboración propia.*

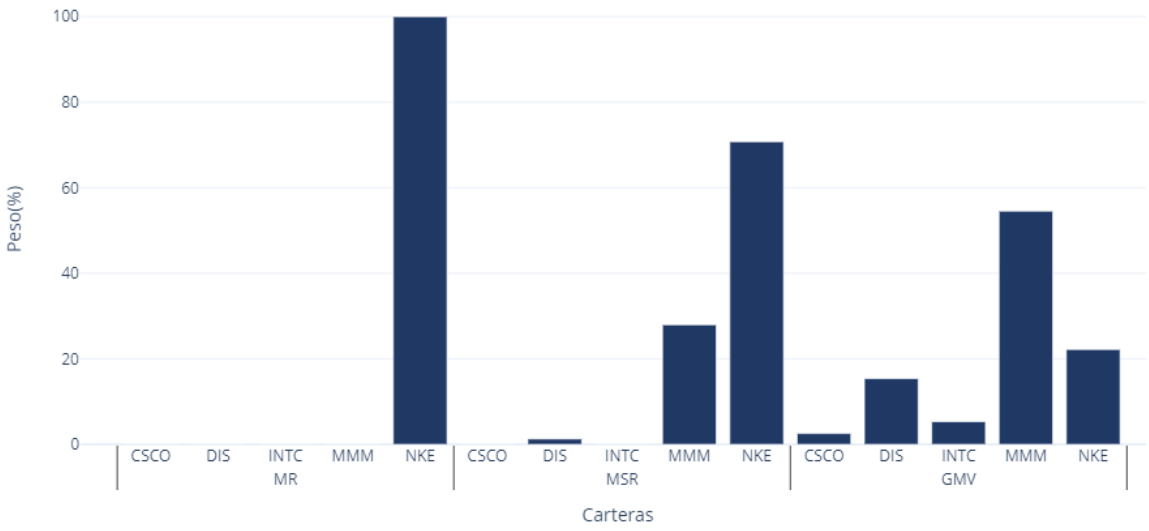
Finalmente, se han obtenido los pesos de las carteras GMV, MSR y MR:

- GMV: Se ha calculado la matriz de varianzas y covarianzas aplicando (3), (6) y (8) a partir de las rentabilidades diarias. A continuación, se han calculado los pesos con (18).
- MSR: Se ha aplicado (20) partiendo del vector de rentabilidades esperadas y la matriz de varianzas y covarianzas.
- MR: NKE, al ser el activo de mayor rentabilidad esperada, concentrará el 100% del capital en esta cartera.

La asignación de activos resultante, así como la rentabilidad y volatilidad esperada de las carteras GMV, MSR y MR considerando el periodo 2000-2019 se muestra en la Figura 3.4.

Figura 3.4. Distribución de pesos y rentabilidad esperada de MR, MSR y GMV.

	Rentabilidad Esperada	Volatilidad Esperada	CSCO(%)	DIS(%)	INTC(%)	MMM(%)	NKE(%)
MR	16.45	29.59	0.00	0.00	0.00	0.00	100.00
MSR	15.42	24.44	0.00	1.28	0.00	27.97	70.75
GMV	12.16	20.64	2.55	15.39	5.32	54.54	22.20



Fuente: *Elaboración propia.*

3.5. Carteras basadas en las Redes LSTM

La metodología propuesta en este proyecto para construir carteras de inversión consta de tres fases:

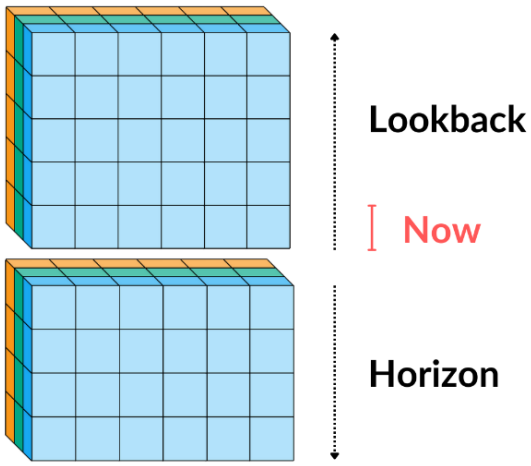
1. Se escogen los activos financieros que van a integrar la cartera. Para ello, se recomienda realizar un análisis fundamental exhaustivo de un conjunto amplio de subyacentes.
2. Se realiza una predicción a un año del precio de cierre diario de los títulos seleccionados implementando y entrenando una Red Neuronal LSTM para cada uno de ellos.
3. Se asigna el peso de los activos en función de los resultados obtenidos en la etapa anterior mediante un procedimiento de normalización ponderada. Para adaptar la cartera al perfil de riesgo del inversor se ha incluido en el modelo un parámetro denominado tolerancia.

3.5.1. Transformación de los datos y configuración de la red

En primer lugar, se han normalizado todos los sets de datos en un intervalo (0, 1). Esto es fundamental para el correcto funcionamiento de las Redes LSTM debido al problema de saturación de sus funciones de activación, la sigmoide y la tangente hiperbólica.

Tal y como se ha explicado en el capítulo anterior, el entrenamiento de modelos de Aprendizaje Supervisado como las Redes LSTM requiere una entrada de datos pareados. Por un lado, hay que proporcionar a la red una matriz tridimensional de tamaño $b \times l \times n$, siendo b el número de submuestras del set de entrenamiento que se pueden tomar para generar los lotes, l el número de periodos de cada lote y n el número de variables de entrada. Esta matriz servirá para calcular las salidas. Por otro lado, también se debe suministrar a la red una matriz $b \times h \times 1$, siendo h el número de periodos de la salida, que se utilizará para calcular el error cometido en cada salida. Estas matrices reciben el nombre de tensores y se han generado siguiendo una estrategia conocida como *Rolling Window* que se ha representado gráficamente en la Figura 3.5.

Figura 3.5. Rolling Window.



Fuente: *Elaboración propia con ilustración de DeepDow.*

Siendo T el número de periodos del set de entrenamiento, una *Rolling Window* consiste en realizar dos particiones en cada periodo t, comenzando por $t = l$ y concluyendo en $t = T - h$, añadiendo submuestras desde el periodo $t - l$ hasta el periodo t (*lookback*) al tensor de entrada y submuestras desde el periodo $t + 1$ hasta el periodo $t + h$ (*horizon*) al tensor de salida. Siguiendo esta estrategia, se obtendrán finalmente $b = T - l - h$ submuestras en cada tensor.

Aunque el objetivo es entrenar a la red para predecir 251 periodos por cada paso a partir de los l periodos anteriores, es habitual definir primero una *Rolling Window* menos exigente computacionalmente para ajustar los hiperparámetros con mayor rapidez. En este caso, se ha definido un *lookback* de $l = 60$ y un *horizon* $h = 1$, de forma que la Red LSTM va a realizar predicciones uniperiodo del precio de cierre de cada activo tomando los 60 precios anteriores. Con esta configuración se han obtenido dos tensores de tamaño 4970×60×1 y 4970×1×1 para cada uno de los activos.

Para validar el modelo y ajustar progresivamente los hiperparámetros se ha usado el set de validación conocido (2020). En la Figura 3.6. se muestra, como ejemplo, el ajuste del modelo LSTM para el activo NKE. No obstante, hay que remarcar que esta configuración no es posible en el escenario planteado, puesto que los precios del 2021 son desconocidos en el momento de la predicción.

Figura 3.6. Validación de la Red LSTM uniperiodo (2020, NKE).

Fuente: *Elaboración propia. Anexo II, pág.72.*

Los hiperparámetros escogidos tras sucesivas validaciones han sido:

- Número de épocas: 25
- Número de neuronas por celda LSTM: 50
- Tamaño del lote: 32
- Función de pérdida: Error Cuadrático Medio (MSE)
- Optimizador: RMSprop (Tasa de aprendizaje adaptativa)
- Función de activación: No aplica (celdas LSTM)

A continuación, se ha adaptado la *Rolling Window* al escenario planteado. El *horizon* debe ser igual al número de periodos del set de validación desconocido, mientras que se ha optado por establecer un *lookback* proporcional: $h = 251$ y $l = 2510$. Así, se han obtenido dos tensores de tamaño 2270×2510×1 y 2270×251×1.

3.5.2. Obtención de las carteras

Tras entrenar de nuevo a la Red LSTM, se ha obtenido la predicción multiperiodo del precio de cierre diario de cada activo para todo el año 2021. Mediante estas predicciones, se han construido cuatro carteras de inversión siguiendo el proceso que se describe a continuación:

1. Se calcula el factor de rentabilidad F de cada activo financiero. Esta medida es más que el cociente entre el último precio estimado por la Red LSTM y el último precio conocido. Se trata de una forma alternativa de expresar la rentabilidad esperada, obteniendo por cuanto se espera que se multiplique el precio de un activo en un año de acuerdo con la predicción realizada.
2. El vector de pesos de la cartera LSTM se determina aplicando la siguiente expresión, siendo $\sum F = F_{CSCO} + F_{DIS} + F_{INTC} + F_{MMM} + F_{NKE}$:

$$\mathrm{X}^{LSTM} = \left(\frac{F_{CSCO}}{\sum F}, \frac{F_{DIS}}{\sum F}, \frac{F_{INTC}}{\sum F}, \frac{F_{MMM}}{\sum F}, \frac{F_{NKE}}{\sum F}\right) \quad (44)$$

3. Puesto que no es habitual que un activo doble su precio o lo reduzca a la mitad en un solo año, los valores que cabe esperar de F se encuentran en el intervalo (0.5, 2). Esto hace que la Cartera LSTM se acerque a una cartera equiponderada o de pesos iguales. Así, se han propuesto cuatro variantes sencillas de la Cartera LSTM que ofrecerán una gran flexibilidad al inversor. Dichas variantes se calculan a partir de un parámetro tol o tolerancia. Así, siendo $\sum F = (F_{CSCO})^{tol} + (F_{DIS})^{tol} + (F_{INTC})^{tol} + (F_{MMM})^{tol} + (F_{NKE})^{tol}$:

$$\mathrm{X}^{LSTMtol} = \left(\frac{(F_{CSCO})^{tol}}{\sum F}, \frac{(F_{DIS})^{tol}}{\sum F}, \frac{(F_{INTC})^{tol}}{\sum F}, \frac{(F_{MMM})^{tol}}{\sum F}, \frac{(F_{NKE})^{tol}}{\sum F}\right) \quad (45)$$

De la expresión (45) se puede deducir que la tolerancia amplifica el peso de los activos con mayor factor de rentabilidad (F), al tiempo que se reduce el peso de aquellos con menor rentabilidad esperada.

La tolerancia debe su nombre al concepto de tolerancia al riesgo. Cuanto mayor sea el valor de este parámetro, más agresiva y menos diversificada estará la cartera resultante, incrementándose el riesgo de la inversión. La tolerancia se puede ver también como un parámetro de confianza en el modelo LSTM. Si el agente está seguro de la capacidad predictiva del modelo, los valores más altos de la tolerancia proporcionarán carteras compuestas mayoritariamente por un único activo: el de mayor rentabilidad esperada. Para el caso propuesto se he definido una tolerancia de 1, 2, 5, 10 y 15.

Los factores de rentabilidad obtenidos una vez entrenada la Red LSTM y realizada la predicción para cada activo se han resumido en la Figura 3.7.

Figura 3.7. Validación de la Red LSTM uniperiodo (2020, NKE).

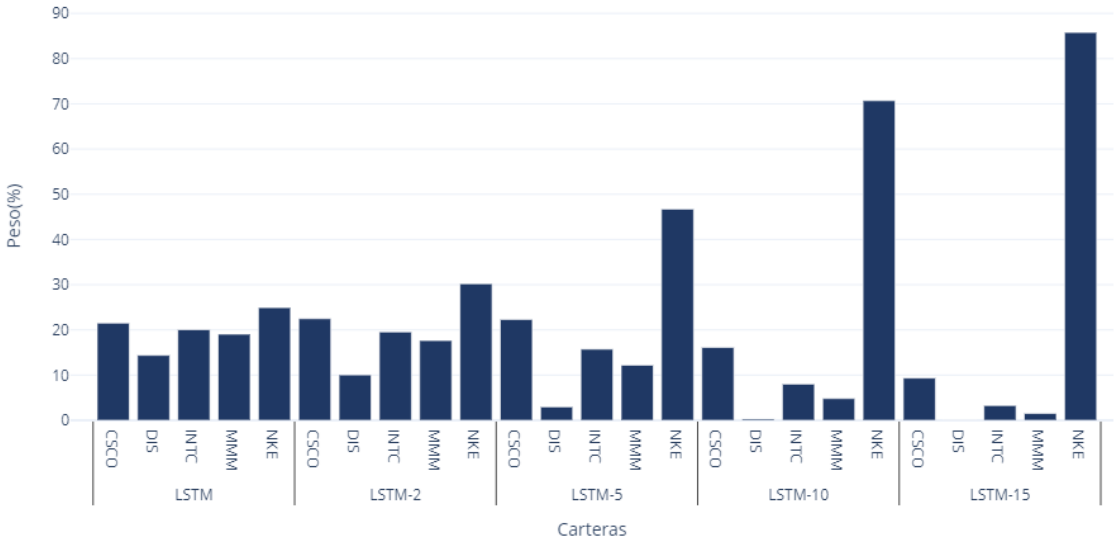
	F	F^2	F^5	F^10	F^15
CSCO	1.1194	1.2532	1.7580	3.0906	5.4332
DIS	0.7488	0.5607	0.2354	0.0554	0.0130
INTC	1.0440	1.0899	1.2402	1.5380	1.9074
MMM	0.9921	0.9843	0.9612	0.9238	0.8879
NKE	1.2976	1.6838	3.6792	13.5367	49.8045

Fuente: *Elaboración propia.*

Finalmente, en la Figura 3.8. se muestra la asignación de activos para las carteras LSTM, así como su rentabilidad-riesgo esperado según el Modelo de Markowitz:

Figura 3.8. Validación de la Red LSTM uniperiodo (2020, NKE).

	Rentabilidad Esperada	Volatilidad Esperada	CSCO(%)	DIS(%)	INTC(%)	MMM(%)	NKE(%)
LSTM	10.73	23.40	21.52	14.39	20.07	19.07	24.94
LSTM-2	11.02	23.59	22.49	10.06	19.56	17.66	30.22
LSTM-5	12.21	24.24	22.33	2.99	15.75	12.21	46.73
LSTM-10	14.21	25.81	16.14	0.29	8.03	4.83	70.71
LSTM-15	15.40	27.48	9.36	0.02	3.29	1.53	85.80



Fuente: *Elaboración propia.*

3.6. Evaluación y análisis comparativo de las carteras

Una vez calculados los pesos de las diferentes carteras, se ha procedido a evaluar y comparar su desempeño durante el 2021. Para ello, se ha calculado el riesgo en desviación típica y la rentabilidad anual de cada activo a partir del segundo set de validación (2021) con (3), (4) y (5). A continuación, se ha aplicado (9) y (10) a los vectores de pesos obtenidos en el epígrafe anterior para obtener la rentabilidad anualizada y el riesgo en desviación típica de cada cartera en 2021. Por último, se ha dividido rentabilidad entre riesgo para calcular las Ratios de Sharpe.

En la Figura 3.9. se muestra un resumen de las métricas evaluadas para todas las carteras construidas en el epígrafe anterior, además de la Cartera Equidistribuida (EW) y la Cartera Aleatoria (RW). También se ha incluido el peso de cada activo en las carteras.

Figura 3.9. Desempeño de las carteras.

	Rentabilidad	Volatilidad	Ratio Sharpe	CSCO(%)	DIS(%)	INTC(%)	MMM(%)	NKE(%)
LSTM	16.98	15.67	1.08	21.52	14.39	20.07	19.07	24.94
LSTM(2)	19.09	16.04	1.19	22.49	10.06	19.56	17.66	30.22
LSTM(5)	22.92	17.87	1.28	22.33	2.99	15.75	12.21	46.73
LSTM(10)	23.99	21.84	1.10	16.14	0.29	8.03	4.83	70.71
LSTM(15)	22.91	24.92	0.92	9.36	0.02	3.29	1.53	85.80
MR	20.54	28.07	0.73	0.00	0.00	0.00	0.00	100.00
MSR	17.22	21.12	0.82	0.00	1.28	0.00	27.97	70.75
GMV	8.86	14.73	0.60	2.55	15.39	5.32	54.54	22.20
EW	14.43	15.48	0.93	20.00	20.00	20.00	20.00	20.00
RW	14.60	16.13	0.91	27.62	24.42	24.55	14.62	8.79

Fuente: *Elaboración propia. Anexo II, pág.85.*

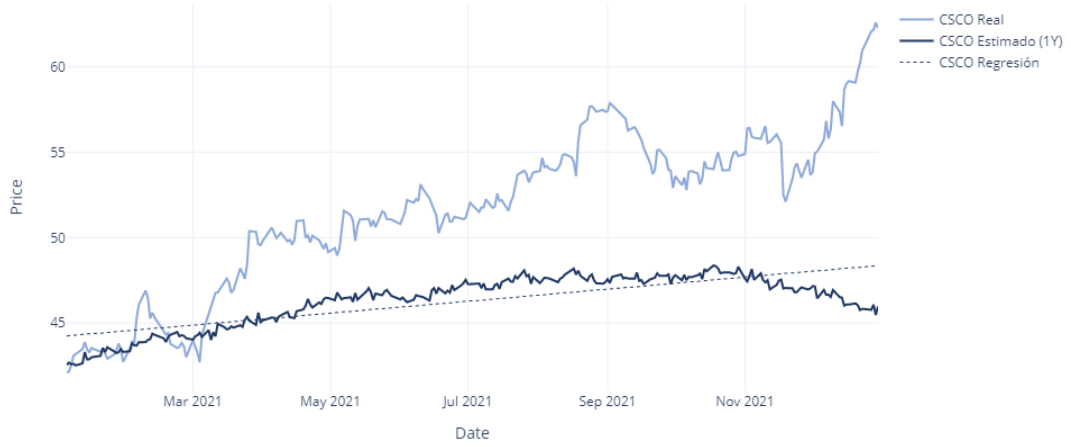
Se puede apreciar como las carteras basadas en la MPT son las que muestran un desempeño más pobre, inferior incluso a las carteras de *benchmarking*, las cuales no requieren de un análisis previo. La cartera MSR refuerza una de las grandes críticas a la teoría de Markowitz. Se trata de una cartera muy poco diversificada, concentrada dos títulos y que comporta un riesgo excesivo.

Por otro lado, puede decirse que la GMV ha cumplido su función, siendo la cartera de menor volatilidad. No obstante, su rentabilidad es muy pobre en comparación con el resto de carteras. Finalmente, la MR evidencia que el Modelo de Markowitz no ha predicho correctamente cuál iba a ser el activo de mayor rentabilidad. En este caso, la rentabilidad de CSCO durante el 2021 fue superior a la de NKE. Por tanto, de haberse elegido esta cartera se habría asumido un riesgo demasiado alto, al invertir en un solo título a cambio de una menor rentabilidad que otras carteras.

Analizando ahora las Carteras LSTM, se comprueba como todas ellas, a excepción de la más arriesgada, proporcionan para el año 2021 una Ratio de Sharpe superior a uno, de modo que su rentabilidad ha superado al riesgo. Tal y como cabía esperar dada su configuración, la volatilidad sube conforme aumenta la tolerancia, aunque también crece la rentabilidad y la Ratio de Sharpe hasta llegar a un cierto umbral de tolerancia. Esto pone de manifiesto que una cartera poco diversificada implica habitualmente asumir riesgos excesivos a cambio de rentabilidades subóptimas. La mejor cartera de acuerdo a los criterios fijados en el enunciado del caso es sin lugar a dudas la LSTM-5, la cual proporciona la Ratio de Sharpe más elevada, con una rentabilidad cercana a la más alta y una volatilidad relativamente próxima a la más baja. Además, la asignación de activos está suficientemente bien equilibrada, de forma que no se incurre en un riesgo desmedido. Ningún título concentra más del 50% del capital invertido y se ha depositado capital en todos ellos. Además, comparando con la Figura 2.8. puede apreciarse cómo la rentabilidad que se puede esperar de acuerdo con el Modelo de Markowitz para estas carteras está muy lejos de la que finalmente han reportado. Esto manifiesta que carteras no eficientes en el sentido de Markowitz pueden comportarse en la praxis sustancialmente mejor que las situadas sobre la Frontera Eficiente.

Finalmente, se va a analizar qué tan bien ha funcionado la Red Neuronal LSTM a la hora de predecir el precio de los activos. Para ello, se va a tomar como ejemplo el caso de CSCO, al haber sido el título de mayor rentabilidad entre la submuestra escogida y haber experimentado una evolución especialmente interesante. En cualquier caso, se puede acceder a la validación y evaluación gráfica de cada activo individual en el Anexo II del documento.

Figura 3.10. Validación de la Red LSTM Multiperiodo (2021, CSCO).



Fuente: *Elaboración propia. Anexo II, pág.74.*

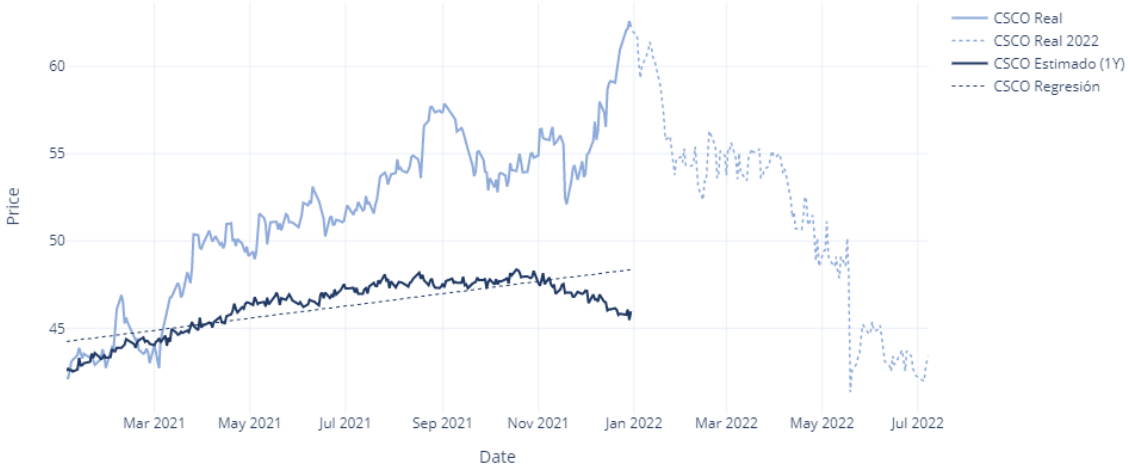
En la Figura 3.10 se ha representado la estimación del precio de CSCO para todo el año 2021, así como su recta de regresión lineal y el precio real del activo. Se aprecia con claridad un problema de subajuste, puesto que el modelo no alcanza a recoger la variabilidad propia de un activo financiero. Además, la predicción del precio tampoco es excesivamente buena, llegando a desviarse más de un 30% al finalizar el año, tal y como se aprecia en la Figura 3.11:

Figura 3.11. Desviación en escala logarítmica de la predicción (2021, CSCO).

Fuente: *Elaboración propia. Anexo II, pág.74.*

No obstante, si se amplía el espacio temporal incorporando la evolución del precio de CSCO hasta la actualidad y se prologa la recta de regresión, se aprecia como el modelo sí realiza una buena aproximación tendencial del precio a largo plazo:

Figura 3.12. Validación de la Red LSTM Multiperiodo (2021-2022, CSCO).



Fuente: *Elaboración propia. Anexo II, pág.75.*

Además, cabe señalar que el caso de CSCO es un tanto especial. 2021 fue un año particularmente bueno para las empresas tecnológicas y su acción se vio muy favorecida por la inercia general del mercado. Esto hace muy complicado explicar la evolución del precio de CSCO en 2021 atendiendo solo a su cotización histórica. Sin embargo, la abrupta caída que ha experimentado la acción desde el comienzo de 2022 conduce a otro punto interesante, y es que las predicciones realizadas por los modelos LSTM, aún con problemas de subajuste, pueden ser una buena herramienta para determinar si un activo está sobrevalorado o infravalorado en cada instante, facilitando las decisiones de compra y venta.

Aunque se trata de un modelo preliminar y muy mejorable, tal y como se mostrará a continuación, lo cierto es que una buena estimación tendencial del precio de los activos ha sido suficiente para el afrontar el problema planteado. Esto se debe a que las carteras LSTM se han construido en términos comparativos, dependiendo el peso de cada activo de la rentabilidad estimada de todos ellos. De este modo, basta con adivinar la tendencia a futuro de cada activo con una precisión razonable para que los resultados de la cartera sean potencialmente buenos.

En cuanto a las causas del subajuste, todo parece apuntar a una insuficiencia en los parámetros de entrada para explicar la salida. Proporcionar a la Red Neuronal una única variable para una tarea tan compleja como explicar el precio futuro de un activo financiero parece insuficiente, sobre todo con un horizonte temporal tan ambicioso. Bajo esta premisa, se ha implementado un modelo LSTM Multivariante para tratar de mejorar la predicción del precio de CSCO, de tal forma que la Red Neuronal recibe un total de once variables de entrada por cada periodo:

- El precio de cierre, apertura, máximo y mínimo de cada jornada cotizada.
- El volumen de negociación.
- Tres indicadores técnicos de la volatilidad: las Bandas de Bollinger.
- Un indicador técnico de *momentum* o impulso: el Índice de Fuerza Relativa.
- Un indicador técnico de tendencia: la Media Móvil de Convergencia.
- El mes de cotización.

La Figura 3.13. recoge la tabla o matriz de entrenamiento de la Red Multivariante. Los únicos hiperparámetros que se han modificado respecto al modelo univariante son el número de épocas y el tamaño del lote, pasando a 50 y 24, respectivamente.

Figura 3.13. Set de entrenamiento de la Red LSTM Multivariante (2000-2019).

Date	Close	Open	High	Low	Volume	BB_AVG	BB_LOW	BB_HIGH	RSI	MACD	Month
2000-01-03	38.725597	39.397528	39.509516	37.112964	53076000	36.602304	34.114060	39.090547	73.983863	1.744719	1
2000-01-04	36.553020	37.807291	38.344835	36.463430	50805600	36.669497	34.264167	39.074826	55.907526	1.550279	1
2000-01-05	36.441036	35.858696	37.448932	34.850800	68524000	36.672856	34.269000	39.076712	55.159426	1.371340	1
2000-01-06	35.836315	36.127485	36.508246	35.343566	48242600	36.673976	34.271703	39.076250	51.177006	1.167277	1
2000-01-07	37.941669	35.612311	37.986465	35.612311	62260600	36.803882	34.422532	39.185231	61.577759	1.162046	1
...	...	...	...	...	...	...	...	...	...	...	...
2019-12-24	43.915020	44.264282	44.347002	43.795538	7044700	41.801532	39.155366	44.447698	63.296262	0.416811	12
2019-12-26	43.979355	43.915018	44.117223	43.795536	11091700	41.918259	39.109102	44.727416	63.702886	0.479566	12
2019-12-27	43.905827	44.108032	44.135604	43.823107	11056800	42.034526	39.101173	44.967879	62.845941	0.517402	12
2019-12-30	43.740398	43.887456	44.006941	43.464666	12128600	42.139305	39.120651	45.157959	60.862217	0.527952	12
2019-12-31	44.080460	43.694436	44.098842	43.574950	14892600	42.289120	39.198255	45.379984	63.418439	0.557329	12

5031 rows × 11 columns

Fuente: *Elaboración propia. Anexo II, pág.90.*

El resultado sobre el set de validación (2021) se muestra en la Figura 3.14. Se puede apreciar una enorme mejoría en la capacidad predictiva del modelo. No solo la estimación del precio futuro es mucho más precisa, sino que, gracias a los indicadores técnicos propagados por las cintas de estado de la Red LSTM, la variabilidad de la predicción se acerca mucho más a la evolución de un activo financiero real, con tendencias a corto plazo y puntos de inflexión evidentes, así como figuras técnicas reconocibles como el doble techo de mayo, seguido de una clara ruptura en el mes de junio.

Figura 3.14. Validación de la Red LSTM Multivariante (2021, CSCO).

Fuente: *Elaboración propia. Anexo II, pág.91.*

3.7. Consideraciones finales sobre las carteras LSTM

Algunas consideraciones sobre la metodología propuesta en este proyecto son:

- Sigue un enfoque de inversión a largo plazo. Una vez construida la cartera, se recomienda mantener la asignación de activos durante al menos un año y repetir el análisis para actualizar los pesos a las nuevas predicciones. Se aconseja también no modificar los títulos de la cartera sin causa justificada.
- Se recomienda la gestión activa, reajustando periódicamente la cartera para devolver la proporción invertida en cada título a la asignación original.

- Se recomienda elegir una tolerancia entre 1 y 10, siendo 1 el valor indicado para los perfiles de inversión más aversos al riesgo y 10 para los amantes. Valores más altos de la tolerancia proporcionan carteras muy concentradas en pocos activos, de escasa diversificación y de riesgo injustificable.
- El modelo, tal y como se ha definido con (44) y (45), asigna el 100% del capital disponible a los activos seleccionados. Se ha tratado de asemejar así a las hipótesis del Modelo de Markowitz para poder evaluar y comparar las carteras de forma justa y equilibrada. Sin embargo, es posible ajustar el modelo para considerar el dinero en efectivo como un activo más de la cartera. Para una cartera de n títulos, siendo $0 \leq L \leq 2$ el factor de liquidez y con $\sum F = (F_1)^{tol} + (F_2)^{tol} + \cdots + (F_n)^{tol} + (L)^{tol}$, el vector de pesos de la cartera LSTM propuesto sería:

$$\mathrm{X}^{LSTMtol} = \left(\frac{(F_1)^{tol}}{\sum F}, \frac{(F_2)^{tol}}{\sum F}, \ldots, \frac{(F_n)^{tol}}{\sum F}, \frac{(L)^{tol}}{\sum F} \right) \qquad (46)$$

  De este modo, el último elemento del vector de pesos (46) es la proporción del capital disponible para la inversión que se mantendrá en efectivo. Esta formulación tiende a asignar mayores pesos al efectivo cuanto menor sea el número de activos en cartera y menor rentabilidad se espere de ellos, lo que resulta especialmente interesante al permitir operaciones de ajuste en épocas de recesión o periodos bajistas modificando el factor de liquidez de la cartera.
- Asimismo, se puede adaptar el modelo para considerar la venta a corto con un factor de liquidez $1 \leq L \leq 2$. Para una cartera de n activos con dinero en efectivo, con $\sum F = (F_1)^{tol} + (F_2)^{tol} + \cdots + (F_n)^{tol} + (L)^{tol} - 1(n+1)$ y venta a corto se tiene:

$$\mathrm{X}^{LSTMtol} = \left(\frac{(F_1)^{tol} - 1}{\sum F}, \frac{(F_2)^{tol} - 1}{\sum F}, \ldots, \frac{(F_n)^{tol} - 1}{\sum F}, \frac{(L)^{tol} - 1}{\sum F} \right) \qquad (47)$$

  Mediante la expresión anterior, se asignará un peso menor que cero a los activos con rentabilidad esperada negativa de acuerdo con la predicción de la Red LSTM, lo que se traduce en una posición corta.

4. CONCLUSIONES

El objetivo principal de este proyecto era presentar un procedimiento innovador para construir carteras de inversión mediante proyecciones de precios realizadas con Redes Neuronales LSTM, tratando de superar el desempeño de los métodos tradicionales basados en el Modelo de Markowitz y sus extensiones posteriores. Para ello, se ha planteado un escenario sencillo y adaptado a las hipótesis de la Teoría Moderna, a fin de contrastar ambas metodologías con imparcialidad.

Aunque es cierto, y debe ser remarcado, que el procedimiento propuesto depende en gran medida de la correcta definición e implementación de las Redes LSTM, así como de su capacidad predictiva y de generalización, es irrebatible a la luz de los resultados obtenidos que permite construir carteras más diversificadas y rentables que las principales aproximaciones de la Teoría Moderna, presentándose como una herramienta eficaz, pragmática, consistente y flexible para el inversor.

- Eficacia: Las Ratios de Sharpe obtenidas muestran como las Carteras LSTM han proporcionado la mayor rentabilidad por unidad de riesgo en todos los casos. También se ha podido comprobar la brecha existente entre el marco analítico de la Teoría Moderna de Carteras y la realidad observada en los mercados, lo que refuerza algunos argumentos de sus principales críticos.
- Pragmatismo: La metodología propuesta es funcional y escalable gracias a la amplia variedad de librerías en *Python* que permiten implementar Redes Neuronales adaptadas a las necesidades concretas de cada problemática.
- Consistencia: El funcionamiento interno de las celdas LSTM hace que las estrategias de inversión derivadas de este método sean consistentes en el tiempo. Al contrario que los enfoques de media-varianza, las Redes LSTM no presentan tanta sensibilidad al horizonte temporal considerado, puesto que son capaces de filtrar, actualizar y descartar los datos temporales según la relevancia que adquieren en cada predicción.
- Flexibilidad: Los parámetros introducidos en el modelo, tolerancia y factor de liquidez, proporcionan una gran flexibilidad y adaptabilidad al perfil de riesgo de cada inversor y a las condiciones transitorias del mercado.

Además, los riesgos derivados del posible error en las proyecciones se suavizan gracias al enfoque ponderado del vector de pesos propuesto, haciendo que una aproximación tendencial razonablemente precisa de la evolución futura de los activos sea suficiente para obtener buenas carteras con un buen desempeño.

Así, este trabajo y su extensión al ámbito económico abren la puerta a una nueva forma de hacer economía, más abierta y heterodoxa, libre de prejuicios y de sesgos ideológicos. Es nuestro deber como economistas presentes y futuros abandonar las premisas erróneas o simplistas que nos preceden para explicar fenómenos que escapan a nuestra racionalidad y mantener los ojos y la mente abierta a las nuevas visiones y perspectivas que tecnologías como el Deep Learning nos ofrecen de la realidad, soltar la pesada losa del *deber ser* y centrar nuestros esfuerzos en guiar a la sociedad a un nuevo y mejorado *ser*.

ANEXO I. APORTACIONES POSTERIORES AL MODELO DE MARKOWITZ

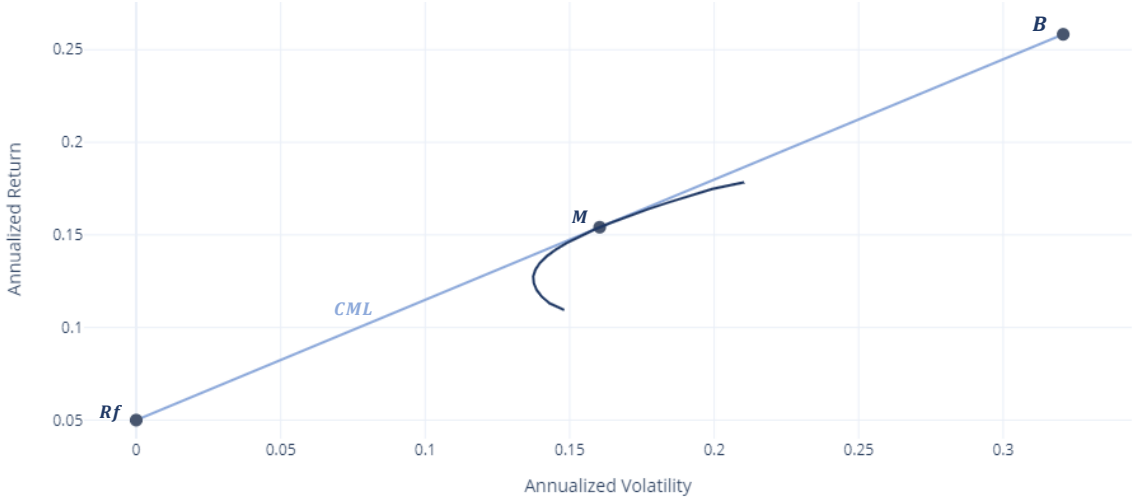
I.I. Modelo de Tobin

James Tobin (1918-2002), economista keynesiano ganador del Nobel en 1981, advirtió una gran limitación de partida en el modelo de Markowitz: al no considerar los activos sin riesgo o el apalancamiento financiero presupone que los inversores disponen únicamente del capital inicial para invertir, lo cual resulta una premisa bastante alejada de la realidad. Para superar este inconveniente, Tobin incorporó al modelo un *Activo Libre de Riesgo* (R_f).

Bajo esta nueva perspectiva, Tobin concluyó que la cartera óptima no depende de la función de utilidad del inversor. Al igual que la Frontera Eficiente, la cartera óptima de riesgo puro es única y común para el conjunto de los individuos y recibe el nombre de *Cartera de Mercado (M)*. Para obtenerla, basta con calcular la recta que corta al eje vertical en el activo libre de riesgo y es tangente a la Frontera Eficiente. Esta recta se denomina *Línea del Mercado de Capitales (CML)* y representa el conjunto de combinaciones óptimas rentabilidad-riesgo entre R_f y *M*.

$$\mu_p = R_f + \sigma_p \left(\frac{\mu_m - R_f}{\sigma_m} \right) \tag{48}$$

Figura A1. Línea del Mercado de Capitales (*DIS*, *INTC* y *PG*), con $R_f = 0.05$.

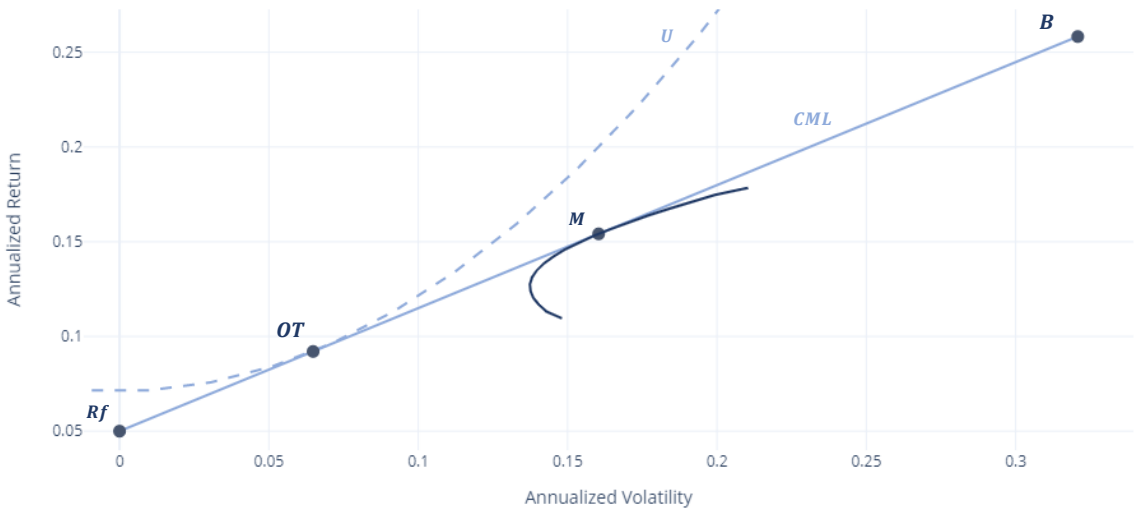


Fuente: *Elaboración propia. Anexo II, pág.65.*

En la Figura A1 se ha representado la *CML* para el universo de activos con riesgo formado por *DIS*, *INTC* y *PG*. Para ello, se ha considerado un activo libre de riesgo con rentabilidad anual del 5% representado por el punto R_f. Por otro lado, el punto *M* simboliza la *Cartera de Mercado* o *Cartera de Riesgo Puro*, resultado de invertir el 100% del montante inicial en los tres títulos con riesgo considerados. Finalmente, el punto B es el binomio rentabilidad-riesgo que se obtendría al pedir prestado la totalidad del capital disponible al tipo de interés R_f para invertirlo por completo en la cartera de mercado.

Cualquier otro punto sobre la *CML* representa la cartera resultante de prestar o pedir prestado una cantidad determinada del capital inicial e invertir el resto en *M*. Estas carteras se denominan mixtas y permiten al inversor alcanzar binomios de rentabilidad y riesgo más favorables en función de su actitud frente al riesgo. Las combinaciones situadas sobre la *CML* y por debajo de *M* representan carteras mixtas con préstamo, resultado de prestar una porción del capital inicial al tipo de interés R_f e invertir el resto en la cartera de mercado; mientras que los puntos por encima de *M* representan carteras mixtas con endeudamiento, producto de pedir prestado una parte del montante inicial al tipo de interés R_f para invertir en *M* con apalancamiento financiero, es decir, con más capital del disponible antes de acometer la inversión.

Figura A2. Cartera óptima de Tobin



Fuente: *Elaboración propia. Anexo II, pág.65.*

En la Figura A2. se ha representado la cartera óptima *(OT)* del inversor con función de utilidad (26) y tolerancia al riesgo $\gamma = 0.2$ según el modelo de Tobin. Se puede observar con claridad como la curva de indiferencia que alcanza con el óptimo de Tobin tiene una ordenada en el origen superior a la que permitía obtener el óptimo de Markowitz, por lo que el modelo de Tobin proporciona una bienestar mayor. Esta situación se repite para cualquier individuo y función de utilidad, salvo que el óptimo de Markowitz coincida con la cartera de mercado *(OM=M)*, en cuyo caso se obtendría la misma cartera óptima en ambos modelos.

Estas ideas se concretan en el *teorema de separación*. Análogamente a Markowitz, Tobin divide su modelo en una etapa objetiva y otra subjetiva. Según el autor, la elección individual de la cartera de activos con riesgo es independiente (está separada) de la actitud del inversor (mayor o menor aversión) frente al riesgo y de su riqueza inicial *(Tobin, 1958)*. Por ende, una vez identificada la Frontera Eficiente, los inversores determinan su cartera óptima en dos pasos diferenciados. En primer lugar, obtienen la cartera óptima de riesgo puro en el punto de tangencia entre la *CML* y la Frontera Eficiente. Para ello, entre todas las carteras situadas sobre la Frontera Eficiente, se identifica aquella que hace máxima la pendiente de la recta con origen en R_f (54). Finalmente, identifican su cartera óptima pura dado su mapa de curvas de isoutilidad:

$$\max \ \frac{\mu_Q - R_f}{\sigma_Q} \qquad (49)$$

Aunque Tobin logró mejorar la *MPT* al incorporar el apalancamiento financiero al modelo de Markowitz, su *teorema de separación* también presenta debilidades significativas que contrastan con la realidad económica y financiera. En la praxis, el mercado de capitales es manifiestamente imperfecto y reporta tasas de préstamo y endeudamiento diferentes, $R_f \neq R_f(L) \neq R_f(B)$, lo cual implica el incumplimiento práctico del *teorema de separación*. En consecuencia, la cartera de riesgo puro deja de ser común para el conjunto del mercado y pasa a depender de la actitud individual frente al riesgo de los inversores. Además, Tobin tampoco consiguió ofrecer herramientas apropiadas para estimar la función de utilidad ni reducir la complejidad matemática del modelo.

I.II. Modelo de Sharpe

Por su parte, William Forsyth Sharpe (1934), laureado con el Premio Nobel de Economía junto al propio Harry Markowitz en 1990, centró su investigación en la simplificación matemática del modelo. Como premisa inicial, Sharpe consideró que la relación entre los rendimientos de los activos no es directa, sino que está ligada a la rentabilidad conjunta del mercado, representada por un índice de referencia.

De esta forma, la rentabilidad esperada de una cartera (55) se obtiene mediante un modelo econométrico clásico de regresión lineal simple. La ordenada en el origen o término independiente (α_{pt}) es el rendimiento autónomo de la cartera en el instante t, mientras que el término dependiente es la rentabilidad esperada del índice de referencia ponderada por *beta* (β_{pt}) que representa la intensidad con la que el índice afecta a la cartera. Los parámetros α_{pt} y β_{pt} se estiman mediante *Mínimos Cuadrados Ordinarios (MCO)*.

$$\mu_{pt} = \alpha_{pt} + \beta_{pt}\mu_{mt} + \varepsilon_{pt} \tag{50}$$

Además, Sharpe introduce uno de los conceptos fundamentales de la *MPT* al dividir el riesgo de la cartera (56) en sistemático (σ_{st}^2) y específico (σ_{et}^2). El primero es aquel que no se puede eliminar por ser intrínseco al mercado y depender de aspectos generales de la economía como los tipos de interés, la inflación global o la situación geopolítica de los países. El segundo, viene determinado por factores propios de cada activo y puede eliminarse mediante diversificación ingenua.

$$\sigma_{pt}^2 = \sigma_{st}^2 + \sigma_{et}^2 = \beta_{pt}^2\sigma_{mt}^2 + \sum_{i=1}^{n} x_{it}^2\,\sigma_{it}^2 \tag{51}$$

Finalmente, la *CML* de Tobin (30) puede reformularse en términos de Sharpe (57) con facilidad, ya que $\hat{\beta}_p = \sigma_{pm}/\sigma_m^2$ resulta un buen estimador de β_p.

$$\mu_{pt} = R_{ft} + (\mu_{mt} - R_{ft})\,\beta_{pt} \tag{52}$$

Si se reescribe la expresión anterior para activos individuales se obtiene lo que Sharpe denominó *Línea del Mercado de Títulos (SML)* (58), expresión fundamental de su *Modelo de Valoración de Activos Financieros (CAPM)*. Esta fórmula determina el valor intrínseco de todos los títulos del mercado. Las combinaciones por encima de la *SML* representan activos infravalorados y por debajo activos sobrevalorados. En ambos casos, se dice que el mercado no está en equilibrio.

$$\mu_{it} = R_{ft} + (\mu_{it} - R_{ft})\,\beta_{it} \qquad (53)$$

Como se puede apreciar, el *CAPM* no tiene en cuenta el riesgo específico, puesto que puede eliminarse mediante la diversificación. Por su parte, β_i es una medida del riesgo sistemático que sirve a su vez para clasificar los activos. Así, una $\beta_i < 1$ indica que el activo i es defensivo, ya que presenta una menor variabilidad que el conjunto del mercado. Análogamente, $\beta_i > 1$ señala que el activo i es agresivo, mientras que para $\beta_i = 1$ se dice que i es neutral. Además, $\beta_i < 0$ indica que i es contrario al mercado. Este tipo de activos son relativamente escasos, pero resultan interesantes en épocas de recesión económica.

Aunque Sharpe consiguió reducir sustancialmente la complejidad matemática del modelo de Markowitz y aportar herramientas de fácil aplicabilidad a la *MPT*, lo cierto es que su modelo sigue asentándose bajo unas hipótesis iniciales demasiado rígidas. Por ejemplo, el *CAPM* considera insignificantes los costes de transacción, perfectamente divisibles los activos individuales y homogéneas las expectativas de los inversores. Además, la tasa de préstamo y endeudamiento continúa siendo única y común para el conjunto de los inversores y el modelo es estático, debiendo evaluarse por separado en cada periodo de decisión. Por si fuera poco, se ha demostrado que el parámetro *beta* es inestable en el tiempo y muy sensible al intervalo temporal del análisis. Finalmente, al estimarse mediante un modelo de regresión lineal simple, su consistencia queda a merced de los problemas clásicos de heteroscedasticidad y autocorrelación de los errores.

ANEXO II. CÓDIGO FUENTE

II.I. Definición de librerías y funciones básicas

```
import random
import pandas as pd
import numpy as np

import time
from dateutil.relativedelta import relativedelta
from datetime import datetime

import plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots

import torch
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.preprocessing.sequence import TimeseriesGenerator
from scipy.optimize import minimize
from scipy.stats import linregress

import yfinance as yf
import ta

import os


def generate_portfolio(tickers, init_date, end_date, indicators=None,
rl_mode=False):

    '''
    Descarga desde la API de Yahoo Finance la serie histórica de los activos
seleccionados
    -- init_date >> Fecha de Inicio
    -- end_date >> Fecha de Fin
    -- indicators >> Indicadores: Open, Close, High, Low, Volume
    '''

    stocks = yf.Tickers(tickers)

    if rl_mode:

        data = stocks.history(start=init_date, end=end_date, actions=False,
group_by='ticker').dropna()
        data = data.reindex(columns=tickers, level=0)
        data = data.loc[:, (tickers, indicators)]
        data.columns = data.columns.remove_unused_levels()
        data.columns.set_names(['Asset', 'Channel'], inplace=True)

    else:

        data = stocks.history(start=init_date,
end=end_date).loc[:,('Close')].dropna()

    return data


def random_returns(mean, std, n=2520):

    return np.random.normal(loc=mean, scale=std, size=n)
```

```python
def annualize_returns(returns, periods_per_year=252):

    return ((1+returns).prod())**(periods_per_year/returns.shape[0])-1


def annualize_volatility(returns, periods_per_year=252):

    return returns.std()*(periods_per_year**0.5)


def portfolio_return(weights, returns):

    return weights.T @ returns


def portfolio_volatility(weights, covariance_matrix, periods_per_year=252):

    return (weights.T @ covariance_matrix*periods_per_year @ weights)**(0.5)


def markowitz_optimization(target_return, expected_returns, covariance_matrix,
weight_bounds=(0.0, 1.0)):

    '''
    Optimización dual de Markowitz. Devuelve el vector de pesos (w)
    que minimiza la volatilidad para alcanzar una rentabilidad objetivo
    '''

    n_assets = expected_returns.shape[0]
    init_weights = np.repeat(1/n_assets, n_assets)

    # Restricciones

    target = {'type': 'eq',
              'args': (expected_returns,),
              'fun': lambda returns, weights: target_return -
portfolio_return(returns, weights)}

    budget = {'type': 'eq',
              'fun': lambda weights: np.sum(weights) - 1}

    # Optimizacion

    output = minimize(portfolio_volatility, init_weights,
                      args=(covariance_matrix,), method='SLSQP',
                      constraints=(target, budget),
                      bounds=(weight_bounds,)*n_assets)

    return output.x


def optimal_weights(expected_returns, covariance_matrix, n=20,
weight_bounds=(0.0, 1.0)):

    '''
    Ejecuta la optimización dual de Markowitz 'n' veces
    generando una matriz de pesos (W)
    '''
    target_returns = np.linspace(expected_returns.min(),
expected_returns.max(), n)

    return [markowitz_optimization(target_return, expected_returns,
covariance_matrix,
                                   weight_bounds) for target_return in
target_returns]
```

```
def msr(expected_returns, covariance_matrix, riskfree_rate=0,
weight_bounds=(0.0, 1.0)):

    '''
    Devuelve el vector de pesos (w) que proporciona la máxima rentabilidad
    por unidad de riesgo (Ratio de Sharpe).
    '''
    n_assets = expected_returns.shape[0]
    init_weights = np.repeat(1/n_assets, n_assets)

    # Restricciones

    budget = {'type': 'eq',
              'fun': lambda weights: np.sum(weights) - 1}

    def nsr(weights, expected_returns, covariance_matrix, riskfree_rate):

        '''
        Devuelve el Ratio de Sharpe negativo para utilizar el programa dual
        '''
        ret = portfolio_return(weights, expected_returns)
        vol = portfolio_volatility(weights, covariance_matrix)

        return -(ret-riskfree_rate)/vol

    # Optimizacion

    output = minimize(nsr, init_weights,
                      args=(expected_returns, covariance_matrix,
riskfree_rate,),
                      method='SLSQP',
                      options={'disp': False},
                      constraints = (budget),
                      bounds=(weight_bounds,)*n_assets)

    return output.x


def gmv(covariance_matrix):

    '''
    Devuelve el vector de pesos (w) que proporciona la
    cartera de mínima varianza
    '''

    return msr(np.repeat(1, covariance_matrix.shape[0]), covariance_matrix)


def ef(expected_returns, covariance_matrix, n=20):

    """
    Devuelve la rentabilidad y volatilidad esperada de 'n'
    puntos de la Frontera Eficiente
    """
    weights = optimal_weights(expected_returns, covariance_matrix, n)
    rets = [portfolio_return(w, expected_returns) for w in weights]
    vols = [portfolio_volatility(w, covariance_matrix) for w in weights]
    ef = pd.DataFrame({"Returns": rets, "Volatility": vols})

    return ef



def plot_ef(returns, show_points=False, show_cml=False,
inner_portfolios=False, utility=False, tol=0.2, rf=0.05):

    efficient_frontier = ef(annualize_returns(returns), returns.cov())
    fig = go.Figure()
```

```
    X, Y = efficient_frontier.iloc[:,1].tolist(),
efficient_frontier.iloc[:,0].tolist()
    Xlim = [min(X)-0.01, max(X)+0.01]
    Ylim = [min(Y)-0.01, max(Y)+0.01]

    if show_cml:

        Xlim = [0, max(X)+0.01]
        Ylim = [rf-0.01, max(Y)+0.01]
        weights = msr(annualize_returns(returns), returns.cov(),
riskfree_rate=rf)
        ret = portfolio_return(weights, annualize_returns(returns))
        vol = portfolio_volatility(weights, returns.cov())
        max_x = 2*vol
        cml_x = np.array([0, max_x])
        cml_y = rf + cml_x*((ret-rf)/vol) #CML Equation
        max_y = cml_y[-1]

        fig.add_trace(go.Scatter(x=cml_x, y=cml_y,
line=dict(color='#8eaadb')))
        fig.add_trace(go.Scatter(x=[vol], y=[ret],
                                 marker=dict(size=10,
                                             color='#47556e',
                                             symbol=0), mode='markers'))
        fig.add_trace(go.Scatter(x=[0], y=[rf],
                                 marker=dict(size=10,
                                             color='#47556e',
                                             symbol=0), mode='markers'))
        fig.add_trace(go.Scatter(x=[max_x], y=[max_y],
                                 marker=dict(size=10,
                                             color='#47556e',
                                             symbol=0), mode='markers'))
    if utility:

        if show_cml:

            X_cml = np.linspace(0, max_x, 100)
            Y_cml = rf + X_cml*((ret-rf)/vol)

            util=[]

            for x, y in zip(X_cml, Y_cml):

                util.append(y-(1/tol)*x**2)

                if y-(1/tol)*x**2 > util[0]:

                    xopt, yopt = x, y

                util = [max(util)]

            x_u = np.linspace(-0.5, 0.5, 50)
            y_u = []

            for x in x_u:

                y_u.append(util[0]+(1/tol)*x**2)

            fig.add_trace(go.Scatter(x=x_u, y=y_u, line=dict(color='#8eaadb',
dash='dash')))

            fig.add_trace(go.Scatter(x=[xopt], y=[yopt],
                                     marker=dict(size=10,
                                                 color='#47556e',
                                                 symbol=0), mode='markers'))

        else:
```

```
            util=[]

            for x, y in zip(X, Y):

                util.append(y-(1/tol)*x**2)

                if y-(1/tol)*x**2 > util[0]:

                    xopt, yopt = x, y

                util = [max(util)]

            x_u = np.linspace(0, 0.5, 20)
            y_u = []

            for x in x_u:

                y_u.append(util[0]+(1/tol)*x**2)

            fig.add_trace(go.Scatter(x=x_u, y=y_u, line=dict(color='#8eaadb',
dash='dash')))

            fig.add_trace(go.Scatter(x=[xopt], y=[yopt],
                                     marker=dict(size=10,
                                                 color='#47556e',
                                                 symbol=0), mode='markers'))



    if inner_portfolios:

        np.random.seed(75)

        weights = [np.random.rand(returns.shape[1]) for i in range(1000)]
        weights = [(w/sum(w)).tolist() for w in weights]

        pairs = [[portfolio_return(w, annualize_returns(returns)),
portfolio_volatility(w, returns.cov())] for w in np.array(weights)]
        pairs = pd.DataFrame(pairs)

        fig.add_trace(go.Scatter(x=pairs.iloc[:,1], y=pairs.iloc[:,0],

marker=dict(color=(pairs.iloc[:,0])/(pairs.iloc[:,1]),
                                                showscale=True,
                                                size=7,
                                                line=dict(width=1),
                                                colorscale='Blues',

colorbar=dict(title="Sharpe<br>Ratio")), mode='markers'))



    fig.add_trace(go.Scatter(x=X, y=Y, line=dict(color='#1F3864')))


    if show_points:

        a, b =
portfolio_return(gmv(returns.cov()),annualize_returns(returns)),
portfolio_volatility(gmv(returns.cov()),returns.cov())

        fig.add_trace(go.Scatter(x=[b], y=[a],
                                 marker=dict(size=14,
                                             color='#8eaadb',
                                             line=dict(width=2,
color='#1F3864'),
                                             symbol=0), mode='markers'))
```

```
        a, b = portfolio_return(msr(annualize_returns(returns),
returns.cov()),annualize_returns(returns)),
portfolio_volatility(msr(annualize_returns(returns),
returns.cov()),returns.cov())

        fig.add_trace(go.Scatter(x=[b], y=[a],
                                 marker=dict(size=14,
                                             color='#8eaadb',
                                             line=dict(width=2,
color='#1F3864'),
                                             symbol=0), mode='markers'))


        Xmax =
annualize_returns(returns).loc[annualize_returns(returns).sort_values().index[
-1]]
        Ymax =
annualize_volatility(returns).loc[annualize_returns(returns).sort_values().ind
ex[-1]]

        Xmin =
annualize_returns(returns).loc[annualize_returns(returns).sort_values().index[
0]]
        Ymin =
annualize_volatility(returns).loc[annualize_returns(returns).sort_values().ind
ex[0]]

        fig.add_trace(go.Scatter(x=[Ymax], y=[Xmax],
                                     marker=dict(size=10,
                                                 color='#47556e',
                                                 symbol=0), mode='markers'))

        fig.add_trace(go.Scatter(x=[Ymin], y=[Xmin],
                                     marker=dict(size=10,
                                                 color='#47556e',
                                                 symbol=0), mode='markers'))


    fig.update_layout(template='plotly_white',
                      xaxis=dict(title='Annualized Volatility', range=Xlim),
                      yaxis=dict(title='Annualized Return', range=Ylim),
                      coloraxis_colorbar=dict(title="Sharpe Ratio"),
                      showlegend=False)

    return fig
```

II.II. Figuras gráficas

Figura 2.1. Conjunto de posibilidades de inversión en dos activos A y B según ρ_{AB}.

```
annret_a, annvol_a = 0.13, 0.31
annret_b, annvol_b = 0.08, 0.23

target_corr = np.linspace(-1, 1, 5)

covs_ab = [corr*((annvol_a)/(252**0.5))*((annvol_b)/(252**0.5)) for corr in
target_corr]

covs_mat = [[[((annvol_a)/(252**0.5))**2, cov_ab],[cov_ab,
((annvol_b)/(252**0.5))**2]] for cov_ab in covs_ab]

efficient_fronts = [ef(pd.Series([annret_a, annret_b]), cov_mat, 100) for
cov_mat in covs_mat]

fig = go.Figure()

fig.add_trace(go.Scatter(x=pd.concat(efficient_fronts).iloc[:,1].tolist(),
y=pd.concat(efficient_fronts).iloc[:,0].tolist(),
                         line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Annualized Volatility', range=[-
0.01,0.33]),
                  yaxis=dict(title='Annualized Return', range=[0.075,0.135]),
                  showlegend=False)
```

Figura 2.2. Frontera eficiente de dos activos A y B.

```
ticker_sample = ['DIS', 'INTC', 'PG']

ticker_yf = ' '.join(ticker_sample)

returns = generate_portfolio(ticker_yf,init_date='2010-01-01',end_date='2020-
01-01')

plot_ef(returns.pct_change().dropna(), show_points=True,
inner_portfolios=True, utility=False)
```

Figura 2.3. Mapa de curvas de isoutilidad para $U = E_p - \frac{1}{\gamma}\sigma_p^2$; $\gamma = 0.2$.

```
fig = go.Figure()
X = np.linspace(0, 0.5, 50)
Y = np.linspace(0.1, 0.5, 50)
U = np.linspace(0.1, 0.5, 10)
alpha = np.linspace(0.1, 1, 10)

tol=0.1

for u, a in zip(U, alpha):
    util=[]

    for x, y in zip(X, Y):

        util.append(u-y+(1/tol)*x**2)
```

```
    fig.add_trace(go.Scatter(x=X, y=util,
line=dict(color='rgba(31,56,100,'+str(a)+')')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Annualized Volatility', range=[-0.005,
0.255]),
                  yaxis=dict(title='Annualized Return', range=[0, 0.425]),
                  showlegend=False)
```

Figura 2.4. Cartera Óptima de Markowitz.

```
plot_ef(returns.pct_change().dropna(), show_points=False,
inner_portfolios=False, utility=True)
```

Figura A1. Línea del Mercado de Capitales (*DIS*, *INTC* y *PG*), con $R_f = 0.05$.

```
plot_ef(returns.pct_change().dropna(), show_points=False, show_cml=True,
inner_portfolios=False, utility=False)
```

Figura A2. Cartera óptima de Tobin

```
plot_ef(returns.pct_change().dropna(), show_points=False,
show_cml=True, inner_portfolios=False, utility=True)
```

Figura 2.6. Función signo.

```
fig = go.Figure()
x = np.linspace(-5, 5, 1000)
y = np.sign(x)

fig.add_trace(go.Scatter(x=x, y=y, line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='h'),
                  yaxis=dict(title='f(h)'),
                  showlegend=False)
```

Figura 2.7. Función tangente hiperbólica.

```
fig = go.Figure()

x = np.linspace(-5, 5, 1000)
y = np.tanh(x)

fig.add_trace(go.Scatter(x=x, y=y, line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='h'),
                  yaxis=dict(title='f(h)'),
                  showlegend=False)
```

Figura 2.8. Función sigmoide o logística.

```
fig = go.Figure()

x = np.linspace(-5, 5, 1000)
y = 1/(1 + np.exp(-x))

fig.add_trace(go.Scatter(x=x, y=y, line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='h'),
                  yaxis=dict(title='f(h)'),
                  showlegend=False)
```

II.III. Caso práctico

I.III.I. Entrenamiento de la Red LSTM

```
ticker_list =
['AAPL','AXP','BA','CAT','CSCO','CVX','DD','DIS','GS','HD','IBM','INTC','JNJ',
'JPM','KO','MCD','MMM','MRK','MSFT','NKE','PFE','PG','RTX','TRV','UNH','V','VZ
','WBA','WMT','XOM']

assets = sorted(random.sample(ticker_list, k=5))

################################################################################

# Dataset

################################################################################

init_date = '2000-01-02'
end_date = '2021-12-31'

portfolio = generate_portfolio(assets, init_date=init_date, end_date=end_date)

################################################################################

# Neural Network

################################################################################

def lstm_training(data, test_period, lookback, horizon, neurons=50, epochs=25,
batch_size=32, optimizer='rmsprop', loss='mse'):

    scaler = MinMaxScaler(feature_range=(0,1))
    assets = data.columns.tolist()

    training_losses = []
    price_predictions = []
    regression_over_predictions = []

    for asset in assets:

        train = data.loc[:str(int(test_period)-1), asset].to_frame()
        test = pd.concat([train.loc[:, asset].tail(lookback),
data.loc[test_period, asset]]).to_frame()


################################################################################

        # Training

################################################################################

        train = scaler.fit_transform(train)

        X_train = []
        y_train = []

        for i in range(lookback, len(train)-horizon):

            X_train.append(train[i-lookback:i,0])
            y_train.append(train[i:i+horizon,0])

        X_train, y_train = np.array(X_train), np.array(y_train)

        X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
        y_train = np.reshape(y_train, (y_train.shape[0], horizon, 1))
```

```
        model = Sequential()
        model.add(LSTM(units=neurons, input_shape=(X_train.shape[1],1)))
        model.add(Dense(units=horizon))
        model.compile(optimizer=optimizer, loss=loss)
        model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size)

        training_losses.append(model.history.history['loss'])


################################################################################

        # Testing

################################################################################

        x_test = test.values
        x_test = scaler.transform(x_test)

        X_test = []

        for i in range(lookback, len(x_test)):

            X_test.append(x_test[i-lookback:i,0])

        X_test = np.array(X_test)
        X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))

        prediction = model.predict(X_test)
        prediction = scaler.inverse_transform(prediction)

        if prediction.shape[1]!=1:

            prediction = prediction[0]
            name = 'forecast'

        else:

            name = 'validation'

        prediction = pd.DataFrame(prediction)
        prediction.index = data.loc[test_period].index
        prediction.columns = [asset]

        price_predictions.append(prediction)

        regression = linregress(range(len(prediction)), prediction.loc[:,
asset].tolist())
        regression = regression.intercept + range(len(prediction)) *
regression.slope
        regression = pd.DataFrame(regression)
        regression.index = data.loc[test_period].index
        regression.columns = [asset]

        regression_over_predictions.append(regression)


################################################################################

        # Save Data

################################################################################

        prediction.to_excel(r'_data/'+asset+'_prediction_'+name+'.xlsx')
        regression.to_excel(r'_data/'+asset+'_regression_'+name+'.xlsx')

    return training_losses, price_predictions, regression_over_predictions
```

```
################################################################################

# Retrieve Training Data

################################################################################

def retrieve_data(data_folder):

    validation_predictions = []
    forecast_predictions = []
    validation_regressions = []
    forecast_regressions = []

    [validation_predictions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'prediction_validation' in file]
    [forecast_predictions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'prediction_forecast' in file]
    [validation_regressions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'prediction_validation' in file]
    [forecast_regressions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'regression_forecast' in file]

    return validation_predictions, validation_regressions,
forecast_predictions, forecast_regressions
```

I.III.II. Validación de la Red LSTM (2020)

```
validation_comparison = [pd.concat([portfolio.loc['2020'].iloc[:, i],
validation_predictions[i]], axis=1) for i in range(5)]

for i in range(len(assets)):

    validation_comparison[i].columns = [validation_comparison[i].columns[0]+'
Real',
                                        validation_comparison[i].columns[1]+'
Estimado (1D)']

################################################################################

# Cisco Systems - NASDAQ: CSCO

################################################################################

i=assets.index('CSCO')

fig = go.Figure()

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=validation_comparison[i].columns[0]))
fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=validation_comparison[i].columns[1]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
################################################################################

# The Walt Disney Company - NYSE: DIS

################################################################################

i=assets.index('DIS')

fig = go.Figure()

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=validation_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=validation_comparison[i].columns[1]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

```
################################################################################

# Intel Corporation - NASDAQ: INTC

################################################################################

i=assets.index('INTC')

fig = go.Figure()

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=validation_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=validation_comparison[i].columns[1]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
################################################################################

# Minnesota Mining and Manufacturing Company - NYSE: MMM

################################################################################

i=assets.index('MMM')

fig = go.Figure()

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=validation_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=validation_comparison[i].columns[1]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
################################################################################

# Nike - NYSE: NKE

################################################################################

i=assets.index('NKE')

fig = go.Figure()

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=validation_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=validation_comparison[i].index,
y=validation_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=validation_comparison[i].columns[1]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

I.III.III. Predicción de la Red LSTM (2021)

```
forecast_comparison = [pd.concat([portfolio.loc['2021'].iloc[:, i],
                                  forecast_predictions[i],
                                  forecast_regressions[i]], axis=1) for i in
range(5)]

for i in range(len(assets)):

    forecast_comparison[i].columns = [forecast_comparison[i].columns[0]+'
Real',
                                      forecast_comparison[i].columns[1]+'
Estimado (1Y)',
                                      forecast_comparison[i].columns[2]+'
Regresión']

################################################################################

# Price Prediction: Accuracy Analysis

################################################################################

daily_deviations =
[((np.log(forecast_comparison[i].iloc[:,1]/forecast_comparison[i].iloc[:,0]))*
100).round(2) for i in range(5)]

mean_deviation=[round(((forecast_comparison[i].iloc[:,1]/forecast_comparison[i
].iloc[:,0]).mean()-1)*100,2) for i in range(5)]
final_deviation=[round(((forecast_comparison[i].iloc[-
1,1]/forecast_comparison[i].iloc[-1,0])-1)*100,2) for i in range(5)]
returns_real=[round(((forecast_comparison[i].iloc[-
1,0]/forecast_comparison[i].iloc[0,0])-1)*100,2) for i in range(5)]
returns_forecast=[round(((forecast_comparison[i].iloc[-
1,1]/forecast_comparison[i].iloc[0,1])-1)*100,2) for i in range(5)]
returns_forecast_adj=[round(((forecast_comparison[i].iloc[-
1,1]/forecast_comparison[i].iloc[0,0])-1)*100,2) for i in range(5)]

accuracy_forecast = pd.DataFrame(data=[mean_deviation, final_deviation,
returns_real, returns_forecast, returns_forecast_adj],
                                 columns=assets,
                                 index=['Desviación Media', 'Desviación
Final', 'Rentabilidad Real',
                                        'Rentabilidad Estimada', 'Rentabilidad
Estimada Ajustada'])

accuracy_forecast
```

	CSCO	DIS	INTC	MMM	NKE
Desviación Media	-9.73	-17.35	-10.05	-6.27	-14.04
Desviación Final	-26.16	-5.21	-3.15	-7.26	15.36
Rentabilidad Real	47.77	-12.24	6.87	6.64	20.45
Rentabilidad Estimada	8.03	-17.64	3.44	-9.56	52.12
Rentabilidad Estimada Ajustada	9.11	-16.81	3.51	-1.10	38.95

```
################################################################################

# Cisco Systems - NASDAQ: CSCO

################################################################################

i=assets.index('CSCO')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

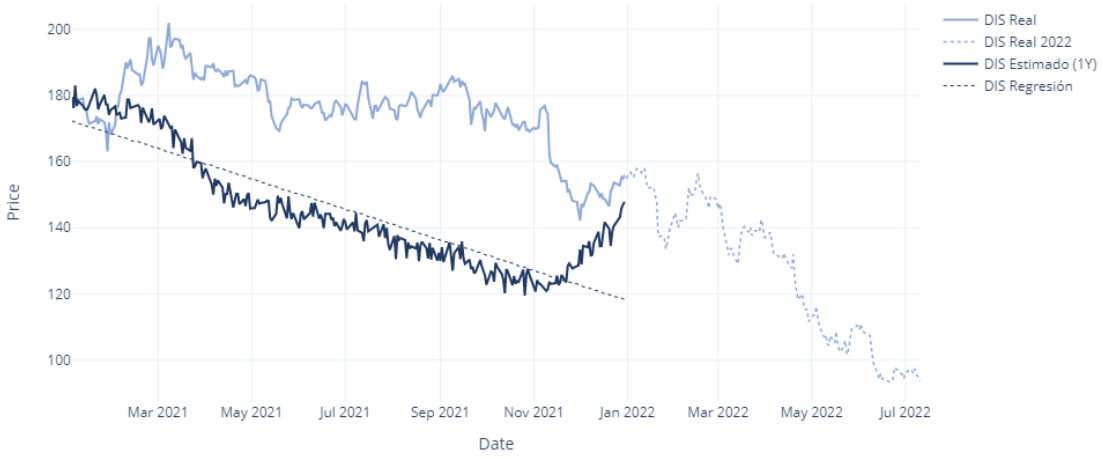

```
i=assets.index('CSCO')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=daily_deviations[i],
                         line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Logarithmic Deviation to Real Price (%)'),
                  barmode='stack')
```


```
i=assets.index('CSCO')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=yf.Ticker(assets[i]).history(start='2022-01-
01').index,
                         y=yf.Ticker(assets[i]).history(start='2022-01-
01').Close,
                         line=dict(color='#8eaadb', dash='dot', width=1.5),
                         name=forecast_comparison[i].columns[0]+' 2022'))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

```
###############################################################################

# The Walt Disney Company - NYSE: DIS

###############################################################################

i=assets.index('DIS')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
i=assets.index('DIS')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=daily_deviations[i],
                         line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Logarithmic Deviation to Real Price (%)'),
                  barmode='stack')
```

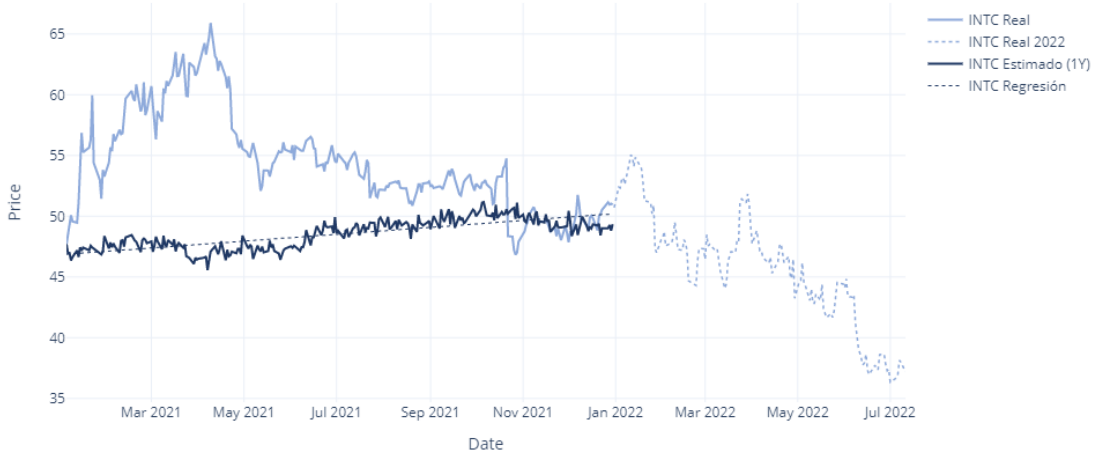

```
i=assets.index('DIS')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=yf.Ticker(assets[i]).history(start='2022-01-
01').index,
                         y=yf.Ticker(assets[i]).history(start='2022-01-
01').Close,
                         line=dict(color='#8eaadb', dash='dot', width=1.5),
                         name=forecast_comparison[i].columns[0]+' 2022'))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```



```
###############################################################################

# Intel Corporation - NASDAQ: INTC

###############################################################################

i=assets.index('INTC')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
i=assets.index('INTC')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=daily_deviations[i],
                         line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Logarithmic Deviation to Real Price (%)'),
                  barmode='stack')
```


```
i=assets.index('INTC')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=yf.Ticker(assets[i]).history(start='2022-01-
01').index,
                         y=yf.Ticker(assets[i]).history(start='2022-01-
01').Close,
                         line=dict(color='#8eaadb', dash='dot', width=1.5),
                         name=forecast_comparison[i].columns[0]+' 2022'))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```



```
################################################################################

# Minnesota Mining and Manufacturing Company - NYSE: MMM

################################################################################

i=assets.index('MMM')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
i=assets.index('MMM')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=daily_deviations[i],
                         line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Logarithmic Deviation to Real Price (%)'),
                  barmode='stack')
```


```
i=assets.index('MMM')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=yf.Ticker(assets[i]).history(start='2022-01-
01').index,
                         y=yf.Ticker(assets[i]).history(start='2022-01-
01').Close,
                         line=dict(color='#8eaadb', dash='dot', width=1.5),
                         name=forecast_comparison[i].columns[0]+' 2022'))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

```
###############################################################################

# Nike - NYSE: NKE

###############################################################################

i=assets.index('NKE')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```


```
i=assets.index('NKE')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=daily_deviations[i],
                         line=dict(color='#1F3864')))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Logarithmic Deviation to Real Price (%)'),
                  barmode='stack')
```


```
i=assets.index('NKE')

fig = go.Figure()

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb'),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=yf.Ticker(assets[i]).history(start='2022-01-
01').index,
                         y=yf.Ticker(assets[i]).history(start='2022-01-
01').Close,
                         line=dict(color='#8eaadb', dash='dot', width=1.5),
                         name=forecast_comparison[i].columns[0]+' 2022'))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 1],
                         line=dict(color='#1F3864'),
                         name=forecast_comparison[i].columns[1]))

fig.add_trace(go.Scatter(x=forecast_comparison[i].index,
y=forecast_comparison[i].iloc[:, 2],
                         line=dict(color='#1F3864', dash='dot', width=1),
                         name=forecast_comparison[i].columns[2]))

fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

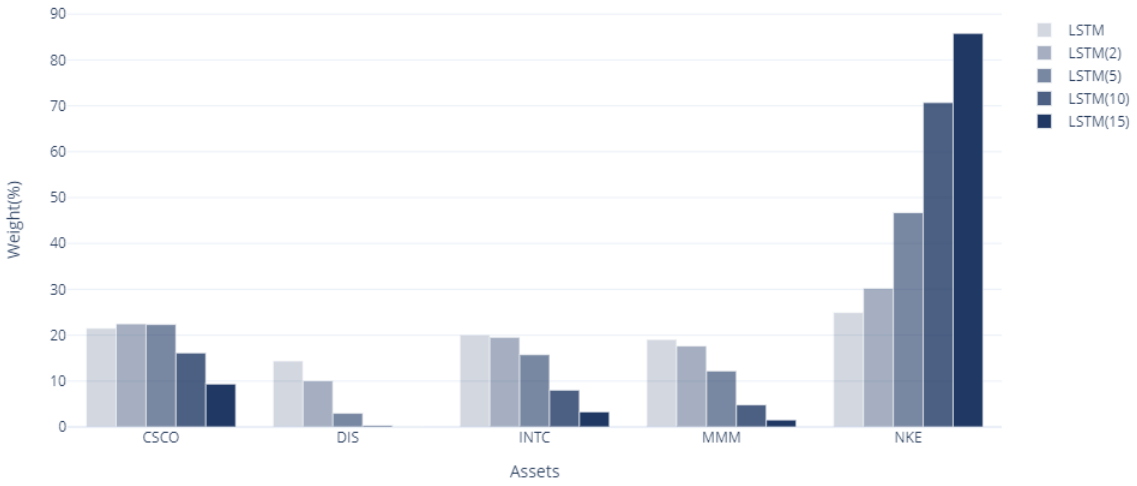
I.III.IV. Análisis comparativo de las carteras

```
return_factors = [forecast_comparison[i].iloc[-
1,1:].mean()/forecast_comparison[i].iloc[0,0] for i in range(5)]

lstm_w = np.array([x/sum(np.array(return_factors)) for x in
np.array(return_factors)])
lstm_ret = annualize_returns(portfolio_return(lstm_w,
portfolio.loc['2021'].pct_change().T))*100
lstm_vol = annualize_volatility(portfolio_return(lstm_w,
portfolio.loc['2021'].pct_change().T))*100

lstm_parameter = 2
lstm2_w = np.array([x/sum(np.array(return_factors)**lstm_parameter) for x in
np.array(return_factors)**lstm_parameter])
lstm2_ret = annualize_returns(portfolio_return(lstm2_w,
portfolio.loc['2021'].pct_change().T))*100
lstm2_vol = annualize_volatility(portfolio_return(lstm2_w,
portfolio.loc['2021'].pct_change().T))*100

lstm_parameter = 5
lstm5_w = np.array([x/sum(np.array(return_factors)**lstm_parameter) for x in
np.array(return_factors)**lstm_parameter])
lstm5_ret = annualize_returns(portfolio_return(lstm5_w,
portfolio.loc['2021'].pct_change().T))*100
lstm5_vol = annualize_volatility(portfolio_return(lstm5_w,
portfolio.loc['2021'].pct_change().T))*100

lstm_parameter = 10
lstm10_w = np.array([x/sum(np.array(return_factors)**lstm_parameter) for x in
np.array(return_factors)**lstm_parameter])
lstm10_ret = annualize_returns(portfolio_return(lstm10_w,
portfolio.loc['2021'].pct_change().T))*100
lstm10_vol = annualize_volatility(portfolio_return(lstm10_w,
portfolio.loc['2021'].pct_change().T))*100

lstm_parameter = 15
lstm15_w = np.array([x/sum(np.array(return_factors)**lstm_parameter) for x in
np.array(return_factors)**lstm_parameter])
lstm15_ret = annualize_returns(portfolio_return(lstm15_w,
portfolio.loc['2021'].pct_change().T))*100
lstm15_vol = annualize_volatility(portfolio_return(lstm15_w,
portfolio.loc['2021'].pct_change().T))*100


#Markowitz-Based Benchmarks

mark_max = np.argmax(annualize_returns(portfolio.pct_change()).tolist())
mr_w = np.zeros(len(assets))
mr_w[mark_max] = 1
mr_ret = annualize_returns(portfolio_return(mr_w,
portfolio.loc['2021'].pct_change().T))*100
mr_vol = annualize_volatility(portfolio_return(mr_w,
portfolio.loc['2021'].pct_change().T))*100

msr_w = msr(annualize_returns(portfolio.pct_change().dropna()),
portfolio.pct_change().cov())
msr_ret = annualize_returns(portfolio_return(msr_w,
portfolio.loc['2021'].pct_change().T))*100
msr_vol = annualize_volatility(portfolio_return(msr_w,
portfolio.loc['2021'].pct_change().T))*100

gmv_w = gmv(portfolio.pct_change().cov())
gmv_ret = annualize_returns(portfolio_return(gmv_w,
portfolio.loc['2021'].pct_change().T))*100
gmv_vol = annualize_volatility(portfolio_return(gmv_w,
portfolio.loc['2021'].pct_change().T))*100
```

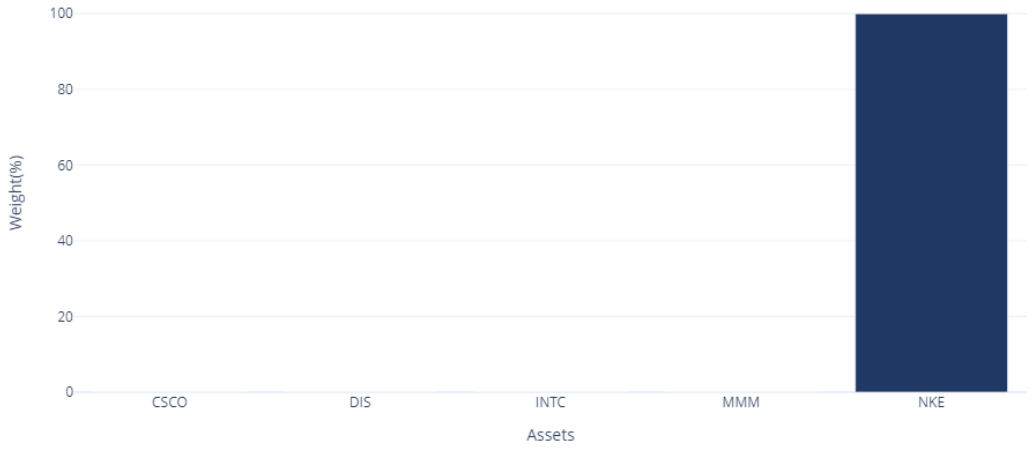
```
#Naive Equally-Weighted and Random-Weighted Benchmarks

ew_w = np.repeat(1/len(assets), len(assets))
ew_ret = annualize_returns(portfolio_return(ew_w,
portfolio.loc['2021'].pct_change().T))*100
ew_vol = annualize_volatility(portfolio_return(ew_w,
portfolio.loc['2021'].pct_change().T))*100

rand = np.random.rand(5)
rw_w = np.array([x/rand.sum() for x in rand])
rw_ret = annualize_returns(portfolio_return(rw_w,
portfolio.loc['2021'].pct_change().T))*100
rw_vol = annualize_volatility(portfolio_return(rw_w,
portfolio.loc['2021'].pct_change().T))*100


backtest = pd.DataFrame([[lstm_ret, lstm_vol, (lstm_w*100).round(2)],
                         [lstm2_ret, lstm2_vol, (lstm2_w*100).round(2)],
                         [lstm5_ret, lstm5_vol, (lstm5_w*100).round(2)],
                         [lstm10_ret, lstm10_vol, (lstm10_w*100).round(2)],
                         [lstm15_ret, lstm15_vol, (lstm15_w*100).round(2)],
                         [mr_ret, mr_vol, (mr_w*100).round(2)],
                         [msr_ret, msr_vol, (msr_w*100).round(2)],
                         [gmv_ret, gmv_vol, (gmv_w*100).round(2)],
                         [ew_ret, ew_vol, (ew_w*100).round(2)],
                         [rw_ret, rw_vol, (rw_w*100).round(2)]],
                        index = ['LSTM', 'LSTM(2)', 'LSTM(5)', 'LSTM(10)',
'LSTM(15)',
                                 'MR', 'MSR', 'GMV', 'EW', 'RW'],
                        columns = ['Rentabilidad', 'Volatilidad', 'weights'])

backtest.insert(backtest.columns.get_loc('weights'), 'Ratio Sharpe',
backtest.Rentabilidad/backtest.Volatilidad)

backtest['CSCO(%)'] = [w[0] for w in backtest.weights]
backtest['DIS(%)'] = [w[1] for w in backtest.weights]
backtest['INTC(%)'] = [w[2] for w in backtest.weights]
backtest['MMM(%)'] = [w[3] for w in backtest.weights]
backtest['NKE(%)'] = [w[4] for w in backtest.weights]

#backtest.drop(columns='weights', inplace=True)

backtest = backtest.round(2)
```

	Rentabilidad	Volatilidad	Ratio Sharpe	CSCO(%)	DIS(%)	INTC(%)	MMM(%)	NKE(%)
LSTM	16.98	15.67	1.08	21.52	14.39	20.07	19.07	24.94
LSTM(2)	19.09	16.04	1.19	22.49	10.06	19.56	17.66	30.22
LSTM(5)	22.92	17.87	1.28	22.33	2.99	15.75	12.21	46.73
LSTM(10)	23.99	21.84	1.10	16.14	0.29	8.03	4.83	70.71
LSTM(15)	22.91	24.92	0.92	9.36	0.02	3.29	1.53	85.80
MR	20.54	28.07	0.73	0.00	0.00	0.00	0.00	100.00
MSR	17.22	21.12	0.82	0.00	1.28	0.00	27.97	70.75
GMV	8.86	14.73	0.60	2.55	15.39	5.32	54.54	22.20
EW	14.43	15.48	0.93	20.00	20.00	20.00	20.00	20.00
RW	14.60	16.13	0.91	27.62	24.42	24.55	14.62	8.79

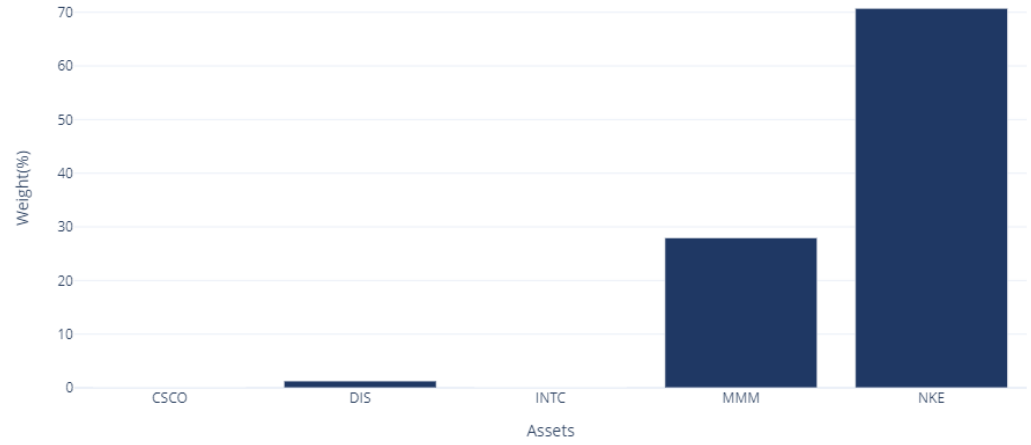
```
###############################################################################

# LSTM y LSTM-tol

###############################################################################


fig = go.Figure()

fig.add_trace(go.Bar(y=backtest.weights['LSTM'], x=assets,
                     marker_color='#1F3864', opacity=0.2, name='LSTM'))

fig.add_trace(go.Bar(y=backtest.weights['LSTM(2)'], x=assets,
                     marker_color='#1F3864', opacity=0.4, name='LSTM(2)'))

fig.add_trace(go.Bar(y=backtest.weights['LSTM(5)'], x=assets,
                     marker_color='#1F3864', opacity=0.6, name='LSTM(5)'))

fig.add_trace(go.Bar(y=backtest.weights['LSTM(10)'], x=assets,
                     marker_color='#1F3864', opacity=0.8, name='LSTM(10)'))

fig.add_trace(go.Bar(y=backtest.weights['LSTM(15)'], x=assets,
                     marker_color='#1F3864', opacity=1, name='LSTM(15)'))

fig.update_layout(template='plotly_white', xaxis=dict(title='Assets'),
yaxis=dict(title='Weight(%)'))
```



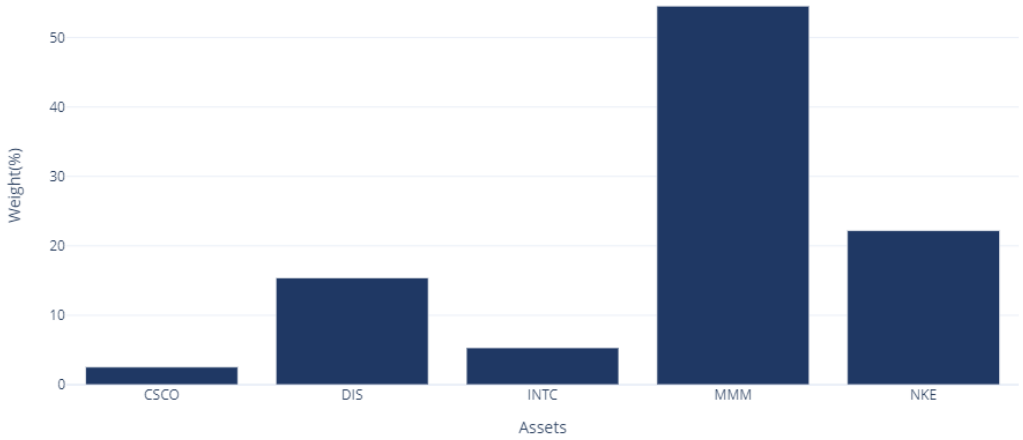

```
###############################################################################

# MR

###############################################################################


fig = go.Figure()

fig.add_trace(go.Bar(y=backtest.weights['MR'], x=assets,
                     marker_color='#1F3864'))

fig.update_layout(template='plotly_white', xaxis=dict(title='Assets'),
yaxis=dict(title='Weight(%)'))
```




```
################################################################################

# MSR

################################################################################


fig = go.Figure()

fig.add_trace(go.Bar(y=backtest.weights['MSR'], x=assets,
                     marker_color='#1F3864'))

fig.update_layout(template='plotly_white', xaxis=dict(title='Assets'),
yaxis=dict(title='Weight(%)'))
```




```
################################################################################

# GMV

################################################################################


fig = go.Figure()

fig.add_trace(go.Bar(y=backtest.weights['GMV'], x=assets,
                     marker_color='#1F3864'))

fig.update_layout(template='plotly_white', xaxis=dict(title='Assets'),
yaxis=dict(title='Weight(%)'))
```



I.III.V. Mejora de la capacidad predictiva (LSTM Multivariante)

```
################################################################################

# Dataset

################################################################################

init_date = '2000-01-02'
end_date = '2021-12-31'

portfolio = generate_portfolio(assets, init_date=init_date, end_date=end_date)

################################################################################

# Neural Network

################################################################################

def lstm_multivariate_training(data, test_period, lookback, horizon,
neurons=50, epochs=50, batch_size=24, optimizer='rmsprop', loss='mse'):

    scaler = MinMaxScaler(feature_range=(0,1))
    assets = data.columns.tolist()

    training_losses = []
    price_predictions = []
    regression_over_predictions = []

    for asset in assets:

        train = data.loc[:str(int(test_period)-1), asset].to_frame()
        test = pd.concat([train.loc[:, asset].tail(lookback),
data.loc[test_period, asset]]).to_frame()


################################################################################

        # Training

################################################################################

        train = scaler.fit_transform(train)

        X_train = []
        y_train = []
```

```
        for i in range(lookback, len(train)-horizon):

            X_train.append(train[i-lookback:i,0])
            y_train.append(train[i:i+horizon,0])

        X_train, y_train = np.array(X_train), np.array(y_train)

        X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
        y_train = np.reshape(y_train, (y_train.shape[0], horizon, 1))

        model = Sequential()
        model.add(LSTM(units=neurons, input_shape=(X_train.shape[1],1)))
        model.add(Dense(units=horizon))
        model.compile(optimizer=optimizer, loss=loss)
        model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size)

        training_losses.append(model.history.history['loss'])


################################################################################

        # Testing

################################################################################

        x_test = test.values
        x_test = scaler.transform(x_test)

        X_test = []

        for i in range(lookback, len(x_test)):

            X_test.append(x_test[i-lookback:i,0])

        X_test = np.array(X_test)
        X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))

        prediction = model.predict(X_test)
        prediction = scaler.inverse_transform(prediction)

        if prediction.shape[1]!=1:

            prediction = prediction[0]
            name = 'forecast'

        else:

            name = 'validation'

        prediction = pd.DataFrame(prediction)
        prediction.index = data.loc[test_period].index
        prediction.columns = [asset]

        price_predictions.append(prediction)

        regression = linregress(range(len(prediction)), prediction.loc[:,
asset].tolist())
        regression = regression.intercept + range(len(prediction)) *
regression.slope
        regression = pd.DataFrame(regression)
        regression.index = data.loc[test_period].index
        regression.columns = [asset]

        regression_over_predictions.append(regression)
```

```
################################################################################

        # Save Data

################################################################################


        prediction.to_excel(r'_data/'+asset+'_prediction_'+name+'.xlsx')
        regression.to_excel(r'_data/'+asset+'_regression_'+name+'.xlsx')

    return training_losses, price_predictions, regression_over_predictions


################################################################################

# Retrieve Training Data

################################################################################


def retrieve_data(data_folder):

    validation_predictions = []
    forecast_predictions = []
    validation_regressions = []
    forecast_regressions = []

    [validation_predictions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'prediction_validation' in file]
    [forecast_predictions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'prediction_forecast' in file]
    [validation_regressions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'prediction_validation' in file]
    [forecast_regressions.append(pd.read_excel(data_folder+'/'+file,
index_col=0, header=0)) for file in os.listdir(data_folder) if
'regression_forecast' in file]

    return validation_predictions, validation_regressions,
forecast_predictions, forecast_regressions


################################################################################

# Collecting Data

################################################################################


csco = yf.Ticker('CSCO').history(start='1999-01-02', end='2021-12-31').loc[:,
['Close', 'Open', 'High', 'Low', 'Volume']]

csco['BB_AVG'] =
ta.volatility.BollingerBands(close=csco["Close"]).bollinger_mavg()
csco['BB_LOW'] =
ta.volatility.BollingerBands(close=csco["Close"]).bollinger_lband()
csco['BB_HIGH'] =
ta.volatility.BollingerBands(close=csco["Close"]).bollinger_hband()
csco['RSI'] =  ta.momentum.RSIIndicator(close=csco["Close"]).rsi()
csco['MACD'] =  ta.trend.MACD(close=csco["Close"]).macd()
csco['Month'] = csco.index.month

csco = csco.loc['2000':]
```

Date	Close	Open	High	Low	Volume	BB_AVG	BB_LOW	BB_HIGH	RSI	MACD	Month
2000-01-03	38.725597	39.397528	39.509516	37.112964	53076000	36.602304	34.114060	39.090547	73.983863	1.744719	1
2000-01-04	36.553020	37.807291	38.344835	36.463430	50805600	36.669497	34.264167	39.074826	55.907526	1.550279	1
2000-01-05	36.441036	35.858696	37.448932	34.850800	68524000	36.672856	34.269000	39.076712	55.159426	1.371340	1
2000-01-06	35.836315	36.127485	36.508246	35.343566	48242600	36.673976	34.271703	39.076250	51.177006	1.167277	1
2000-01-07	37.941669	35.612311	37.986465	35.612311	62260600	36.803882	34.422532	39.185231	61.577759	1.162046	1
...	...	...	...	...	...	...	...	...	...	...	...
2019-12-24	43.915020	44.264282	44.347002	43.795538	7044700	41.801532	39.155366	44.447698	63.296262	0.416811	12
2019-12-26	43.979355	43.915018	44.117223	43.795536	11091700	41.918259	39.109102	44.727416	63.702886	0.479566	12
2019-12-27	43.905827	44.108032	44.135604	43.823107	11056800	42.034526	39.101173	44.967879	62.845941	0.517402	12
2019-12-30	43.740398	43.887456	44.006941	43.464666	12128600	42.139305	39.120651	45.157959	60.862217	0.527952	12
2019-12-31	44.080460	43.694436	44.098842	43.574950	14892600	42.289120	39.198255	45.379984	63.418439	0.557329	12

5031 rows × 11 columns

```
i=0

fig = go.Figure()

fig.add_trace(go.Scatter(x=csco.loc['2021':].index,
y=forecast_comparison[i].iloc[:, 0],
                         line=dict(color='#8eaadb', width=1.5),
                         name=forecast_comparison[i].columns[0]))

fig.add_trace(go.Scatter(x=_prediction.index, y=_prediction.CSCO,
                         line=dict(color='#1F3864', width=2),
                         name='CSCO Estimado (1Y Multivariante)'))

fig.add_trace(go.Scatter(x=_prediction.index, y=forecast_comparison[i].iloc[:,
1],
                         line=dict(color='#1F3864', width=1),
                         name='CSCO Estimado (1Y)'))


fig.update_layout(template='plotly_white',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),
                  barmode='stack')
```

BIBLIOGRAFÍA

AI Wiki. (2020). *Epochs, Batch Size, & Iterations*. Recuperado el 16-05-2022 de: https://machine-learning.paperspace.com/wiki/epoch#:~:text=Batch%20size%20is%20the%20total,needed%20to%20complete%20one%20epoch.

Browlee, J. (2017). *Multivariate Time Series Forecasting with LSTMs in Keras*. Recuperado el 28-05-2022 de: https://machinelearningmastery.com/category/deep-learning-time-series/

Calvo, D. (2017). *Definición de red neuronal artificial.* Recuperado el 09-05-2022 de: https://www.diegocalvo.es/definicion-de-red-neuronal/#:~:text=Las%20redes%20redes%20neuronales%20no,hace%20el%20sistema%20nervioso%20biol%C3%B3gico.

Colah's blog. (2015). *Understanding LSTM Networks*. GitHub. Recuperado el 09-05-2022 de: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Dobilas, S. (2022). *LSTM Recurrent Neural Networks. How to Teach a Network to Remember the Past*. Towards Data Science. Recuperado el 09-05-2022 de: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

Dueñas, A., Prieto, K. y Sánchez, J. (2017). *Análisis de rentabilidad y riesgo de un portafolio de inversión, aplicando el modelo de Harry Markowitz.* UCC.

Franco, L., Avendaño, C. y Barbutín, H. (2011). *Modelo de Markowitz y Modelo de Black-Litterman en la Optimización de Portafolios de Inversión.* Revistas Tecnológicas, No. 26, pp. 71-88.

Ganagedara, T. (2020). *Stock Market Predictions with LSTM in Python*. Datacamp. Recuperado el 28-05-2022 de: https://www.datacamp.com/tutorial/lstm-python-stock-market

Garzón, J. (2018). *Cómo usar redes neuronales (LSTM) en la predicción de averías en las máquinas*. GFT. Recuperado el 27-05-2022 de: https://blog.gft.com/es/2018/11/06/como-usar-redes-neuronales-lstm-en-la-prediccion-de-averias-en-las-maquinas/

Gastón, E. (2016). *Evolución de las funciones de utilidad para la toma de decisiones*. Escritos Contables y de Administración, Vol. 6, No. 1, pp. 15-43.

Grajales, D. *Gestión de portafolios.* (2009). *Una mirada crítica más allá de Markowitz*. AD-minister, pp.154-162.

Hernández, C. (2021). *Predicción y Clasificación de series temporales bursátiles mediante Redes Neuronales Recurrentes*. Universidad de Valladolid.

Hernández, L. (2022) *¿Qué es y cómo funciona el modelo de Markowitz? Teoría de la cartera y frontera eficiente*. Rankia. Recuperado el 02-04-2022 de: https://www.rankia.com/blog/bolsa-desde-cero/3479118-que-como-funciona-modelo-markowitz-teoria-cartera-frontera-eficiente

Hochreiter, S. y Schmidhuber, J. (1997). *Long Short-Term Memory*. Neural Computation, Vol. 9, No. 8, pp. 1735–1780

López, C. (2014). *Mercado de capitales y gestión de cartera*. UADE.

López, J. (2017). *Modelo de Markowitz*. Economipedia.com. Recuperado el 02-04-2022 de: https://economipedia.com/definiciones/modelo-de-markowitz.html

López, M. (2018). *Advances in Financial Machine Learning*. John Wiley & Sons, Inc.

Markowitz, H. (1959). *Portfolio Selection: Efficient Diversification of Investment*. New York Wiley.

Markowitz, H. *Portfolio Selection*. (1952). The Journal of Finance, Vol. 7, No. 1, pp. 77-91.

Mendizábal, A., Miera, L. y Zubia, M. (2002). *El modelo de Markowitz en la gestión de carteras*. Cuadernos de Gestión, Vol. 2, pp. 36-46.

Michaud, R.O. (1989) *The Markowitz optimization enigma: Is 'optimized' optimal?*. Financial Analysts Journal, Vol. 45, pp. 31-42.

Neumann, J. y Morgenstern, O. (1953): *Theory of games and economic behavior*. Princeton University Press, Vol. 3.

Rodríguez, V. (2018). *Conceptos básicos sobre redes neuronales*. Recuperado el 09-05-2022 de: https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales

Ruiz, D. (2021). *Gestión de cartera inteligente mediante técnica Deep Learning*. Universidad de Málaga.

Santana, C. (2021) *¿Qué es el Descenso del Gradiente? Algoritmo de Inteligencia Artificial | DotCSV*. Scenio. Recuperado el 01-05-2022 de: https://www.youtube.com/watch?v=A6FiCDoz8_4

Serafeim, L. (2020). *Time-Series Forecasting: Predicting Stock Prices Using An LSTM Model*. Towards Data Science. Recuperado el 01-06-2022 de: https://towardsdatascience.com/lstm-time-series-forecasting-predicting-stock-prices-using-an-lstm-model-6223e9644a2f

Sharpe, W. (1964). *Capital Asset Prices: A theory of market equilibrium under conditions of risk*. The Journal of Finance, Vol. 19, No. 3, pp 425-442.

Shefrin, Hersh & Meir Statman. (1985). *The disposition to sell winners too early and ride losers too long: theory and evidence*. The Journal of Finance, Vol. 40 No. 3, pp. 777-790.

Tobin, J. (1958). *Liquidity preference as behaviour towards risk*. The review of economic studies, pp. 65-86.

Toquero, M. (2021). *Clasificación de fallos en motores en estado transitorio mediante redes neuronales.* Universidad de Valladolid.

Weiming, J. (2019). *Mastering Python for Finance*. Packt, No. 2.

Yahoo Finance.