Universidad Internacional de La Rioja (UNIR)

Facultad de Economía y Empresa
Grado en Economía

Trabajo de Fin de Grado (TFG)

MACHINE LEARNING FINANCIERO APLICADO A LA PREDICCIÓN DEL EUR/USD

Elaborado por Iñaki Gangutia Arrázola

Dirigido por Javier Martínez Rodríguez

Julio de 2025

Resumen

Este trabajo tiene como objetivo explorar, desde una perspectiva teórica y práctica, diversos métodos de tratamiento de datos aplicados a series temporales financieras, evaluando su impacto sobre modelos de clasificación binaria para generar señales de compra o venta. Entre los métodos analizados, tomados de López de Prado (2018), se incluyen la diferenciación fraccionaria, el Análisis de Componentes Principales (PCA) y la Disminución Media de Impureza (MDI). Los resultados muestran que la aplicación de PCA sobre variables diferenciadas unitariamente mejora el rendimiento de los modelos al reducir efectos de sustitución, especialmente en la regresión logística. En cambio, MDI y principalmente la diferenciación fraccionaria no aportaron mejoras con los modelos utilizados. En definitiva, los resultados obtenidos evidencian la importancia del preprocesamiento con series temporales financieras y orientan futuras aplicaciones hacia métodos que mejoren la calidad y estructura de las variables de entrada.

Palabras Clave: aprendizaje automático, finanzas, predicción, diferenciación fraccionaria.

Abstract

This paper aims to explore, from a theoretical and practical perspective, several data processing methods applied to financial time series, evaluating their impact on binary classification models to generate buy or sell signals. The methods analyzed, taken from López de Prado (2018), include fractional differentiation, Principal Component Analysis (PCA) and Mean Decrease Impurity (MDI). The results show that the application of PCA on unit differenciated variables improves the performance of the models by reducing substitution effects, especially in logistic regression. On the other hand, MDI and mainly fractional differentiation did not improve the performance of the models used. In short, the results obtained show the importance of preprocessing with financial time series and guide future applications towards methods that improve the quality and structure of the input variables.

Keywords: machine learning, finance, prediction, fractional differentiation.

Índice

Capítulo 1

Introducción

La Economía es la ciencia social que investiga cómo los individuos asignan recursos escasos. A su vez, las Finanzas, como parte de la Economía, estudian cómo los individuos eligen entre valores futuros inciertos (Van Der Wijst, 2013). Por lo tanto, la función esencial de los mercados financieros es asignar capital de forma eficiente, siendo un componente clave del crecimiento de una economía. Debido a la incertidumbre intrínseca que poseen, la predicción financiera como proceso de estimar precios futuros de activos financieros es un campo que fascina al ser humano desde hace siglos.

Un origen destacable puede ubicarse en el Japón del siglo XVIII por parte del comerciante (*trader*) de futuros de arroz Munehisa Honma, el cual amasó una gran fortuna y llegó a convertirse en asesor financiero del Shogunato Tokugawa (Hyongdo, 2014). Sus éxitos se basaron en descubrir que los precios del arroz no dependían únicamente de la oferta y la demanda del mercado, sino también de la psicología de sus participantes. Esto lo vio reflejado en los distintos patrones que formaban las velas japonesas, desarrollando así un enfoque propio conocido como Técnica Sakata.

Avanzamos ahora hasta los Estados Unidos de América del siglo XX para presentar a James Harris Simmons (alias: Jim Simmons), matemático y gestor del fondo de cobertura (*hedge fund*) Renaissance Technologies. Este *hedge fund* se caracteriza por la aplicación de métodos cuantitativos y de aprendizaje automático (*machine learning*) para identificar patrones en los mercados financieros. Es importante destacar la utilización de estas técnicas por su aporte de objetividad en las predicciones financieras que realizan, basándose en técnicas de minería de datos (*data mining*) sobre un gran conjunto de datos. Prueba del éxito de este enfoque es el principal fondo de Renaissance Technologies, llamado Medallion Fund, con unas rentabilidades brutas anualizadas en el período de 1998 a 2018 del 66,07 % siendo este el fondo con las mayores rentabilidades registradas de la historia y destacando que obtuvo únicamente rentabilidades anuales positivas durante los 20 años mencionados (Zuckerman, 2019).

La línea de trabajo irá orientada a lo comentado previamente, en la que se utilizarán técnicas de *machine learning* y métodos específicos de tratamiento de datos financieros en Python, para obtener predicciones financieras en base a datos reales sobre el par de

divisas EUR/USD. Concretamente, se han extraído datos de tres fuentes diferentes: Yahoo Finanzas (*Yahoo Finance*), Sistema de la Reserva Federal (*Federal Reserve System* / FED) y Banco Central Europeo (*European Central Bank* / ECB). Estos datos contienen información de diversos tipos de activos como divisas extranjeras (*Foreign Exchange* / FX), materias primas (*commodities*), índices, acciones (*stocks*), bonos y criptomonedas (*cryptocurrencies*), además de tasas de interés del banco central (*Central Bank rates*).

Este enfoque se apoya principalmente en las aportaciones realizadas por López de Prado (2018) en su libro Advances in Financial Machine Learning, siendo este autor uno de los principales referentes a nivel mundial de la utilización de técnicas de *machine learning* en el sector financiero, tanto en el ámbito académico como en el ámbito laboral.

El objetivo, por tanto, en primer lugar, será comprender en profundidad y aplicar de forma práctica los métodos de tratamiento de datos financieros propuestos. Siendo estos principalmente la diferenciación fraccionaria, el análisis de componentes principales (*Principal Component Analysis* / PCA) y la disminución media de impureza (*Mean Decrease Impurity* / MDI).

En segundo lugar, será comparar los resultados obtenidos sobre una variedad de modelos de clasificación binaria, para apreciar cómo sus distintas arquitecturas internas de algoritmos aprenden de forma diferente sobre los datos, arrojando así distintos resultados, ya que se tratará de obtener señales acertadas sobre la dirección futura $(t + 1)$ del activo, en este caso, del par de divisas EUR/USD con una frecuencia diaria. Para comparar los resultados obtenidos se utilizará como métrica de rendimiento el retorno acumulado (*Cumulative Return*), siendo esta una métrica específica para el problema planteado, debido a que se tratará de simular un sistema de *trading* en el que se ejecutan las predicciones realizadas con una compra o una venta en el caso de cambiar la señal de valor $(0, 1)$. Aún siendo esencial en un sistema de *trading* real, no se tendrán en cuenta los costos de transacción, ya que el objetivo de este trabajo es comprender y aplicar las técnicas en sí.

Los modelos de clasificación binaria utilizados poseerán arquitecturas variadas para comprobar las diferencias en su rendimiento ante los distintos datos utilizados como entrada (*input*). Los modelos utilizados han sido los siguientes: regresión logística (*Logistic Regressio*n / LR), *Support Vector Machine* (SVM), *K-Nearest Neighbor* (KNN), *Random Forest* (RF) y *eXtreme Gradient Boosting* (XGBoost).

Debido al valor que tiene el poder aplicar de forma práctica estos conceptos, se compartirá la totalidad del código desarrollado en el lenguaje de programación Python, siendo

esta la parte más retadora y en la que más tiempo se ha invertido para el desarrollo de este trabajo. El hecho de compartir el código es un valor añadido y diferenciador del trabajo, ya que, por lo menos en gran parte de los artículos académicos y trabajos accesibles de forma gratuita, no se comparte o únicamente se incluyen secciones de forma racionada. Por lo tanto, matizar que otro objetivo de este trabajo es facilitar al lector la comprensión de los conceptos incluidos tanto de forma teórica como práctica, eliminando barreras a la democratización del conocimiento. El código completo desarrollado para este trabajo se encuentra disponible en el siguiente repositorio: `https://github.com/Gangutia/TFG` (Gangutia, 2025). Para aquellos lectores que no deseen ejecutar o descargar el código, se ofrece una versión en formato HTML con los datos y resultados ya cargados: `https://gangutia.github.io/TFG/TFG.html`. Destacar también McKinney (2012), recurso que ha sido de gran ayuda para utilizar de forma eficiente librerías como NumPy y Pandas.

Capítulo 2

Marco teórico

En este capítulo serán expuestas las distintas ideas y desarrollos teóricos, de cara a comprender en profundidad las distintas técnicas utilizadas, poniéndolas al mismo tiempo en contexto y presentando las razones teóricas por las que aportan valor a los modelos realizados posteriormente.

2.1. Nivel estadístico de confianza

El nivel estadístico de confianza es la probabilidad de que el método utilizado para construir el intervalo de confianza contenga el valor verdadero del parámetro a evaluar. Por el contrario, el nivel de significatividad será la probabilidad de que se contenga el valor falso del parámetro a evaluar.

Frost (2020) propone el nivel de confianza del 95 % porque ofrece un equilibrio razonable entre el riesgo de cometer un error de tipo I (falso positivo) y el poder estadístico del estudio, que está relacionado con el error de tipo II (falso negativo). Por ello, a modo de unificar un criterio común en este trabajo se utilizará un nivel de confianza del 95 % en los test estadísticos y métodos realizados.

El nivel de confianza se denota como $1 - \alpha$ y el nivel de significatividad como α. Por lo tanto, el nivel de confianza del 95 % implica un $\alpha = 0,05$, lo que significa que en el 5 % de las ocasiones, el intervalo no contendrá el valor verdadero. En el caso de evaluar un conjunto de valores, aproximadamente el 95 % de ese conjunto contendrá el valor verdadero del parámetro a evaluar.

Una vez establecido un nivel de confianza y por tanto α, al realizar un test estadístico será esencial comprobar el p-valor (*p-value*) como medida de evidencia contra la hipótesis nula (H_0), representando la probabilidad de obtener un resultado tan extremo como el observado. Si el p-valor es menor o igual que el p-valor crítico (α) del nivel de significatividad establecido, se rechaza H_0, aceptando así la hipótesis alternativa (H_1) y viceversa.

2.2. Métodos utilizados

2.2.1. Diferenciación fraccionaria

La diferenciación fraccionaria (*fractional differentiation*) es un método de transformación de datos que asegura la estacionariedad de los datos, es decir, que sus propiedades estadísticas no varían con el tiempo, al mismo tiempo que preserva tanta memoria como sea posible (López de Prado, 2018, cap. 5, sec. 5.1, p. 75).

La estacionariedad de las variables utilizadas como entrada (*input*) es necesaria para los algoritmos supervisados de *machine learning*. Esto se debe a que el proceso que siguen estos algoritmos es mapear una observación sin etiquetar (validación / *test*), sobre una colección de observaciones etiquetadas (entrenamiento / *train*) e inferir así la etiqueta (*label*) que debe asignarle a cada nueva observación, como las observaciones *test*. Por ello, si las variables *input* no son estacionarias, no podrá mapearse cada nueva observación sobre un gran conjunto de ejemplos mutuamente comparables (López de Prado, 2018, cap. 5, sec. 5.2, p. 75-76).

Respecto a la memoria, se considera que está presente en una serie temporal cuando los valores futuros están relacionados con observaciones pasadas, es decir, que se puede apreciar una evolución en los valores de la serie a lo largo del tiempo. En este caso, la serie temporal original posee toda la memoria posible y cuanto más se diferencie, menos relación o memoria conservará. Una forma sencilla de cuantificar cuánta memoria se conserva en la serie temporal es mediante la correlación de Pearson entre la serie original y la diferenciada resultante (López de Prado, 2018, cap. 5, sec. 5.6, p. 85).

El método comúnmente utilizado para convertir las series temporales en estacionarias es el de la diferenciación unitaria (*integer differentiation*), el cual tiende a sobrediferenciar las series temporales al ser aplicado arbitrariamente y, por lo tanto, elimina la mayor parte de la memoria original que estas contenían. En (López de Prado, 2018, cap. 5, sec. 5.6, p. 86-87) se muestran los estadísticos del test de Dickey-Fuller Aumentado (*Augmented Dickey-Fuller* / ADF) tras aplicar varios valores d entre 0 y 1, para diferenciar las series de algunos de los contratos de futuros más líquidos mundialmente, demostrando que con un valor $d = 0,6$ todas las series son estacionarias e incluso la mayoría con un valor $d = 0,3$ al presentar valores de ADF menores a $-2,8623$, siendo este el valor crítico del test para el nivel de confianza del 95 %. Otros métodos similares a este en el que se obtienen

los retornos son el del cambio porcentual o el de las diferencias logarítmicas, los cuales presentan problemas similares, aunque a modo de simplificación en este trabajo se utilizará únicamente la diferenciación unitaria simple mencionada.

De forma teórica, el método de la diferenciación fraccionaria suaviza el problema principal que sufren las series temporales al diferenciarlas unitariamente, conservando una mayor parte de su memoria. Esto se debe a que, como su nombre indica, no se diferenciará unitariamente ($d = 1$), sino que se diferenciará fraccionariamente (a modo de ejemplo $d = 1/2$ o $d = 0{,}5$), buscando obtener el mínimo valor d que cumpla la condición de ser significativamente estacionario mediante el test ADF.

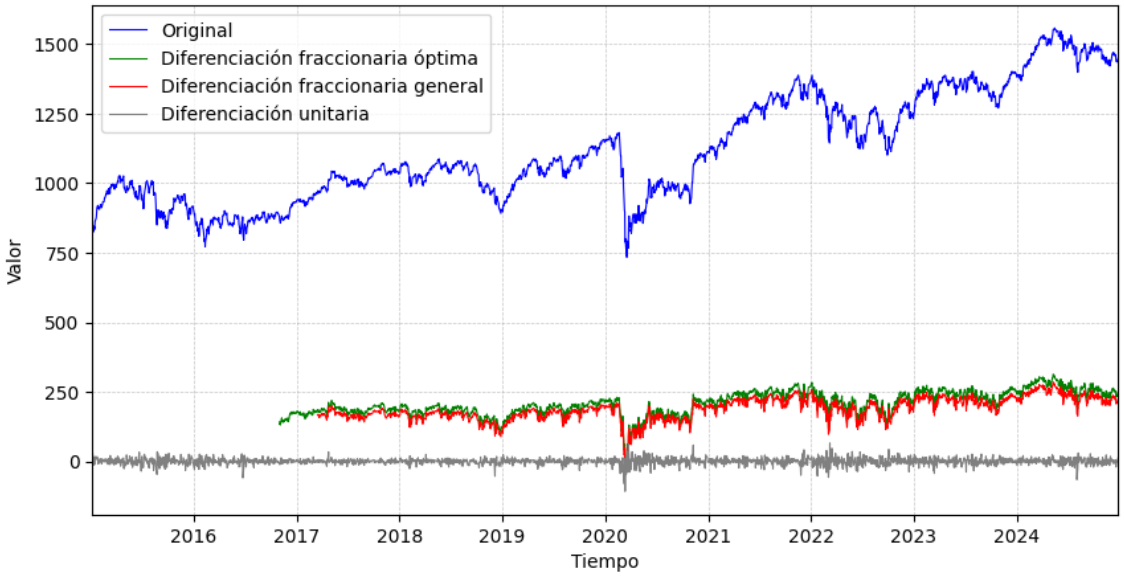

Figura 2.1: Ejemplo de la serie temporal del índice NASDAQ100 según el método de diferenciación utilizado

Como ejemplo visual, la Figura 2.1 muestra el resultado de las distintas diferenciaciones sobre la serie temporal del índice NASDAQ100 (representado en los datos como: ˆN100), plasmando lo comentado previamente sobre el concepto de memoria que está siendo explicado. Se puede apreciar visualmente cómo las series diferenciadas fraccionariamente verde y roja conservan memoria de la serie original azul, al contrario que la serie gris diferenciada unitariamente, que simplemente muestra los retornos. Mencionar que la serie roja con la etiqueta "Diferenciación fraccionaria general" hace referencia a un ajuste realizado, para así obtener el mínimo d sobre todas las variables *input*, al mismo tiempo que se consigue una longitud individual similar, en el Capítulo 3 se desarrollará detalladamente.

2.2.2. Ortogonalización de Variables

La ortogonalización de variables es un método matemático de transformación de variables utilizado para eliminar la correlación entre estas. Esto implica que al ortogonalizar, se obtienen variables que son linealmente independientes en términos de correlación y que no comparten información redundante.

En este trabajo, se utilizará el Análisis de Componentes Principales (*Principal Component Analysis* / PCA) para lograrlo, por lo que se centrarán los esfuerzos en explicar este método en concreto. El PCA es un método estadístico de transformación de variables utilizado para reducir los efectos de sustitución de un conjunto de variables, además, de frecuentemente, reducir la dimensionalidad. Este tipo de práctica, también puede contribuir a facilitar la convergencia del modelo de *machine learning* utilizado, ya que simplifica las relaciones o patrones a identificar.

Según López de Prado (2018, cap. 8, sec. 8.3, p. 114), en la literatura de estadística y econometría, a los efectos de sustitución se les denomina multicolinealidad, estos se dan cuando la importancia estimada de una variable es reducida por la presencia de otras variables relacionadas. Un ejemplo de lo comentado es la dilución de la importancia de variables, cuantificada la importancia en este trabajo mediante la Disminución Media de Impureza (*Mean Decrease Impurity* / MDI), método que será desarrollado en la próxima Sección 2.2.3.

Con la intención de solucionar parcialmente el problema de efectos de sustitución presentado sobre el MDI, primero se debe realizar el PCA, para tras ello, realizar el MDI con los componentes ortogonales generados (López de Prado, 2018, cap. 8, sec. 8.4.2, p. 118).

Tras utilizar el método del PCA, las correlaciones de Pearson de los nuevos componentes creados serán idealmente nulas, es decir, de una suma total de 0 entre las intersecciones de todos los componentes evaluados, al mismo tiempo que se trata de conservar la máxima varianza explicada.

El proceso interno a realizar en el PCA se presenta en López de Prado (2018, cap. 8, sec. 8.4.2, p. 118): se parte de un *input* en forma de matriz de variables estacionarias ($X_{t,n}$) con t observaciones y n variables estacionarias, como por ejemplo las variables diferenciadas fraccionariamente.

1. **Estandarización de variables**: se estandarizan las variables de X para obtener la matríz Z, de forma que queden centradas en media 0 y varianza 1. Esto asegura que

el PCA se enfoque en la correlación entre variables y no en la varianza absoluta.

$$Z_{t,n} = \sigma_n^{-1}(X_{t,n} - \mu_n)$$

donde μ_n es la media de $X_{t,n}$ y σ_n su desviación típica.

2. **Construcción de la matriz de correlación**: se calcula el producto escalar de Z consigo misma, lo cual equivale a una matriz de correlación.

$$P = Z'Z$$

3. **Descomposición espectral**: se computan los autovalores (Λ) y autovectores (W) sobre la matriz P:

$$PW = W\Lambda$$

   - Los Λ representan la varianza explicada por cada componente.
   - Los W indican la dirección de los componentes principales.

4. **Ordenación de componentes**: se ordenan los componentes en orden descendente según la varianza explicada mediante Λ.

5. **Selección de componentes**: se calcula la varianza acumulada para determinar cuántos componentes de los generados conservar, según el valor del parámetro que se utiliza en el proceso.

6. **Proyección sobre componentes principales**: se calcula otro producto escalar entre la matriz estandarizada (Z) y la matriz de autovectores (W) de los componentes seleccionados, proyectando así los nuevos ejes, es decir, los componentes.

$$Componentes = ZW$$

Con intención de ejemplificar el resultado de realizar un PCA, la Figura 2.2 muestra el *ranking* de componentes ordenado de forma descendente, de esta forma, los que explican mayor varianza están en la parte superior. Todos ellos explican de forma conjunta un mínimo del 95 % de la varianza de las variables *input*, por lo que se muestran únicamente los conservados tras la selección de componentes. La línea roja vertical representa la varianza promedio (varianza total dividida por el número de componentes) que debería explicar cada componente con un peso similar, por lo que si un componente la sobrepasa significa

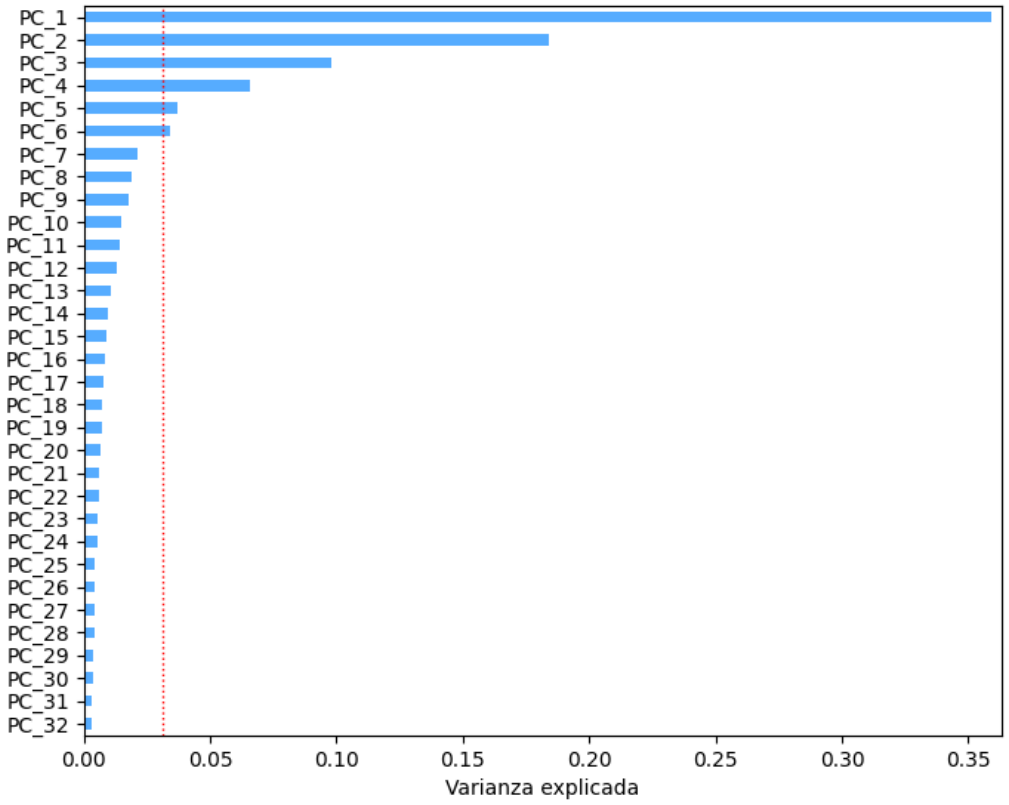

Figura 2.2: Varianza explicada por componente mediante PCA, con componentes ordenados de forma descendente y representando la varianza promedio mediante la línea roja vertical

que aporta más varianza que el promedio y si no la sobrepasa aporta menos varianza que el promedio, apreciando además que la varianza explicada de cada componente nuevo generado decrece respecto a la del anterior con una gran rapidez inicial.

El método PCA utilizado determina una jerarquía entre los componentes, siendo esta que unos componentes son más principales que otros, todo ello sin tener en cuenta las *labels* objetivo, por lo que se trata de un aprendizaje no supervisado (*unsupervised learning*) y sin riesgo de *overfitting* (López de Prado, 2018, cap. 8, sec. 8.4, p. 119-120).

2.2.3. Importancia de Variables

La importancia de variables (*feature importance*) es un concepto en *machine learning* que explica cuánto contribuye cada variable al resultado obtenido por el modelo.

López de Prado (2018, cap. 8, sec. 8.2, p. 114) expone su relevancia para no caer en el mantra de la caja negra (*black box*), en referencia a la utilización de algoritmos de *machine learning* y especialmente con los que poseen una considerable complejidad matemática y no linealidad. Este mantra trata a los algoritmos como una *black box*, en la cual se introduce un *input* y se devuelve un resultado (*output*), pero sin tratar de comprender el proceso interno que el algoritmo ha realizado para aprender sobre los datos *train* y de esta forma inferir un resultado. Esto es sin duda, un gran error, ya que sí se puede conocer qué realiza el algoritmo de forma interna y de esta forma abrir la *black box*. Es natural que el ser humano no sea capaz de comprender relaciones en más de tres dimensiones (sin incluir el tiempo), pero precisamente aquí radica la utilidad de este tipo de algoritmos, desenvolviéndose mejor cuando se utiliza como *input* un conjunto de datos con una gran cantidad de variables y por ello, de dimensiones.

En el caso de este trabajo se utilizará el método de Disminución Media de Impureza (*Mean Decrease Impurity* / MDI) para cuantificar la importancia de las variables. Este método se basa en clasificadores de árboles de decisión (*decision trees*), aunque en este caso se ampliará a un *random forest* para obtener estimaciones más robustas y estables de la importancia de las variables, al promediar los resultados de múltiples árboles y reducir la varianza asociada a modelos individuales. Es importante mencionar que se deben utilizar como *input* los datos de la parte de *train* para evitar fuga de datos (*data leakage*), ya que si no, se obtendrán conclusiones de datos aún no vistos (*test*).

El proceso de funcionamiento de este método se desarrolla en López de Prado (2018, cap. 8, sec. 8.3.1, p. 114-115). En cada nodo de cada *decision tree* se realiza una partición

en base a la variable seleccionada para disminuir la impureza de los datos recibidos. Por lo tanto, se puede cuantificar en cada *decision tree* cuánta reducción de impureza puede asignarse a cada variable. Y al tener un bosque de *decision trees*, es decir, un *random forest*, se promedian los valores para obtener la media global, además de la desviación típica. Por último, se realizará un *ranking* en orden descendente de las variables *input* por importancia para apreciar realmente la importancia de cada variable.

Un ajuste importante realizado sobre el modelo *random forest* es el del hiperparámetro `max_features=1` para evitar efectos de enmascaramiento (*masking effects*), asignando una variable aleatoria por nivel y forzando así al modelo a que explore diferentes combinaciones.

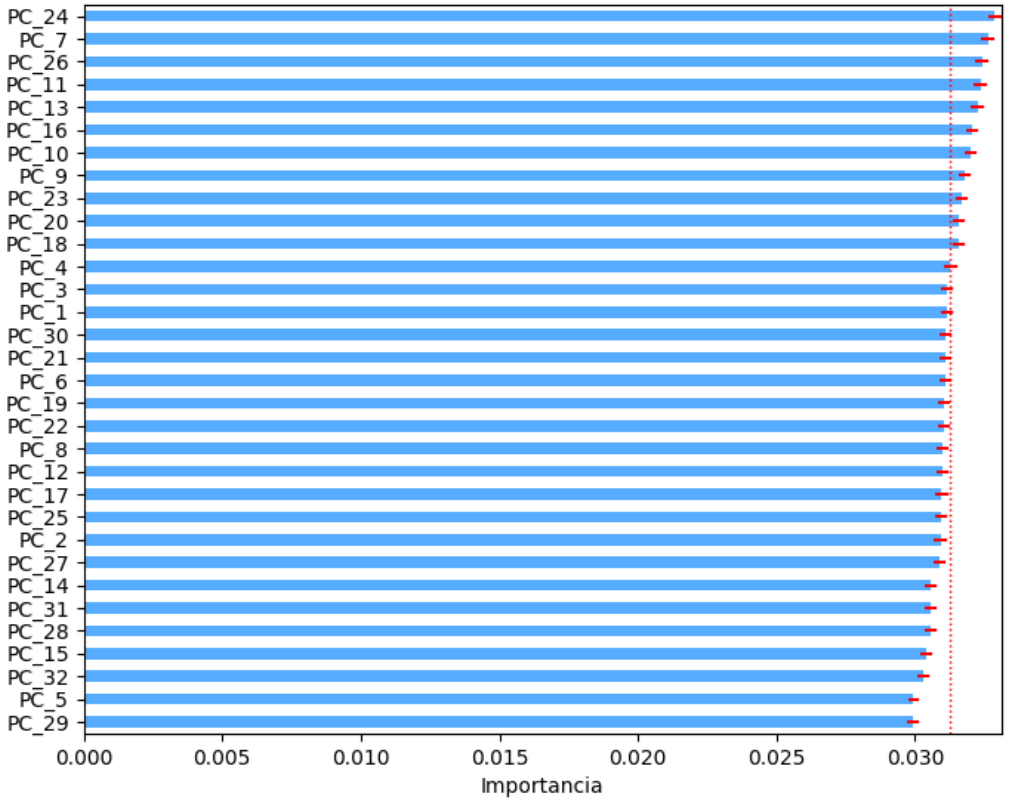

Figura 2.3: Importancia media por componente mediante MDI, con componentes ordenados de forma descendente y representando la importancia promedio mediante la línea roja vertical

A modo de ejemplo ilustrativo, en la Figura 2.3 se puede apreciar la importancia media aportada por cada componente y su desviación típica entre todos los *decision trees* del *random forest*. La línea roja vertical representa la importancia promedio (importancia total dividida entre el número de componentes) que debería poseer cada componente con

un peso similar.

La conclusión extraída es que, únicamente 11 de 31 componentes poseen una importancia media, teniendo en cuenta su desviación típica, ligeramente superior a la importancia promedio que debería poseer cada componente. Esta selección permitirá comprobar si realmente son estos los componentes que aportan más señal y menos ruido, mejorando así el rendimiento del modelo en el que se utilicen como *input*.

López de Prado (2018, cap. 8, sec. 8.2, p. 114) también expone la relevancia real que posee la *feature importance* como herramienta de investigación, en contraposición a la prueba hacia atrás (*backtesting*). La primera trata de mejorar la calidad de la información que se utiliza en el modelo como *input*, tanto con la utilización de información que aporte más señal, como facilitando la captura de esta señal por parte del modelo. Por el contrario, la tendencia o tentación que suele darse con la segunda, es la de repetir el proceso hasta que se obtenga el resultado deseado, es decir, un falso descubrimiento.

Un ejemplo simple de lo comentado y con el que el lector estará familiarizado es con la ejecución de un modelo que dependa del hiperparámetro *random_state*. Si no se establece una semilla aleatoria específica, cada vez que se ejecute el modelo el resultado variará, y por lo tanto existe la tentación de mostrar en el trabajo presentado el mejor de los resultados, pudiendo considerarse incluso un resultado anómalo y, por lo tanto, un falso descubrimiento. López de Prado (2018, cap. 8, sec. 8.1, p. 113) justifica que este tipo de prácticas, son consideradas fraude académico, ya que de forma teórica en 1 de cada 20 iteraciones de un proceso como este, se obtendrá un falso descubrimiento sujeto al nivel estándar de significatividad del 5 %. Para más detalle, American Statistical Association (2022) presenta el tipo de decisiones éticas que deben tomarse al realizar prácticas estadísticas, incluida la mencionada.

López de Prado (2018, cap. 8, sec. 8.4, p. 119-120) presenta una inquietud común en la utilización de algoritmos de *machine learning* financiero, siendo el riesgo de sufrir sobreajuste (*overfitting*) en la generalización a otros datos. Los algoritmos siempre encontrarán un patrón, incluso si este patrón es parte del ruido no convergiendo y no de la señal, que es lo que ayudará a obtener predicciones correctas y no aleatorias. El método MDI utilizado sobre los componentes es parte de esta inquietud, ya que determina una jerarquía, en este caso otorgando una importancia a cada componente y teniendo en cuenta las *labels* objetivo, por lo que se trata de un aprendizaje supervisado (*supervised learning*) y con riesgo de *overfitting*.

2.3. Modelos utilizados

Esta es la selección de modelos orientados a clasificación binaria que se ha realizado en este trabajo, por lo que se procederá a exponer de forma teórica las bases de cada uno, para así poseer un contexto adecuado en su ejecución y evaluación.

2.3.1. Regresión Logística (LR)

Cox (1958) introdujo el modelo de regresión logística (LR) que ajusta un modelo lineal por máxima verosimilitud a partir de un conjunto de variables explicativas, utilizando la función logística para modelar la probabilidad de que ocurra un evento binario (0/1, no/sí, fracaso/éxito).

2.3.2. *Support Vector Machine* (SVM)

Cortes y Vapnik (1995) presentaron el modelo *Support Vector Machine* (SVM) que busca encontrar un hiperplano que separe dos clases en el espacio de datos con el mayor margen posible. Para problemas no lineales, se emplean funciones núcleo (*kernels*) no lineales, que proyectan los datos a espacios de mayor dimensión donde la separación sea linealmente posible.

2.3.3. *K-Nearest Neighbors* (KNN)

Cover y Hart (1967) explicaron el modelo *K-Nearest Neighbors* (KNN) el cual asigna a cada nueva observación la clase más frecuente entre sus k vecinos más cercanos en el conjunto de entrenamiento, definidos según una métrica de distancia en el espacio de variables, típicamente la distancia euclidiana.

2.3.4. *Random Forest* (RF)

Breiman (2001) definieron al modelo *Random Forest* (RF) como una combinación (*ensemble*) de predictores basados en árboles de decisión (*decision trees*), donde cada árbol es entrenado de forma independiente sobre un subconjunto aleatorio de los datos generado mediante muestreo con reemplazo (*bootstrap*), y un subconjunto aleatorio de variables en cada partición del árbol. La predicción se obtiene de la agregación de las salidas individuales de todos los árboles, siendo en problemas de clasificación la clase predicha por la mayoría a modo de votación y en problemas de regresión el promedio

de las predicciones. Este enfoque corresponde a un método de agregación conocido como *bootstrap aggregating* (*bagging*).

2.3.5. *eXtreme Gradient Boosting* (XGBoost)

Chen y Guestrin (2016) consideran al modelo *eXtreme Gradient Boosting* (XGBoost) como una implementación optimizada del algoritmo de *Gradient Boosting* aplicado a *decision trees*. El algoritmo de *Gradient Boosting* construye un *ensemble* de *decision trees* de forma secuencial, donde cada nuevo árbol se entrena para corregir los errores residuales del conjunto anterior, mediante la minimización de una función de pérdida. Este proceso iterativo es característico del enfoque de refuerzo (*boosting*).

2.4. Métricas de evaluación utilizadas

Estas han sido las métricas elegidas para la evaluación de los modelos de clasificación binaria y comparación de métodos utilizados en este trabajo.

2.4.1. *Accuracy*

López de Prado (2018, cap. 14, sec. 14.8, p. 206) la define como la fracción de oportunidades correctamente etiquetadas por el algoritmo. Donde TP (verdaderos positivos / *True Positives*) y TN (verdaderos negativos / *True Negatives*) representan las predicciones correctas, mientras que FP (falsos positivos / *False Positives*) y FN (falsos negativos / *False Negatives*) son los errores del modelo.

$$Accuracy = \frac{\text{Número de predicciones correctas}}{\text{Número total de predicciones}} = \frac{TP + TN}{TP + TN + FP + FN}$$

2.4.2. Retorno Acumulado (*Cumulative Return* / CR)

El CR representa la rentabilidad total obtenida durante un período de tiempo T, asumiendo la reinversión de los beneficios en cada instante (r_i), reflejando así la variación total relativa respecto al valor inicial.

$$\text{CR} = \prod_{i=1}^{T}(1 + r_i) - 1$$

2.4.3. Tau de Kendall ponderado hiperbólicamente

Según López de Prado (2018, cap. 8, sec. 8.4, p. 120), el Tau de Kendall ponderado hiperbólicamente (*hyperbolic-weighted Kendall's tau*) es una medida de correlación entre dos *rankings* que asigna mayor peso a la concordancia entre los elementos más relevantes, reduciendo la influencia de aquellos menos importantes o potencialmente ruidosos. Su aplicación puede ser de utilidad entre el *ranking* de importancia generado por el PCA y por el MDI, para así comparar si sus conclusiones se alinean.

Capítulo 3

Metodología

Este capítulo recoge de forma ordenada y clasificada las distintas etapas seguidas en la programación del código en Python. Se detallarán con un enfoque pragmático los planteamientos realizados, con la intención de mostrar tanto el proceso de pensamiento llevado a cabo para la implementación en código de los distintos métodos, como el ajuste para los requerimientos específicos de este trabajo.

3.1. Recolección de datos

En la recolección de datos (*data collection*) se decide, en primer lugar, qué datos de activos financieros se desean incluir en el trabajo, además de la vía a utilizar para su obtención. Para ello, un enfoque claro es la información del tipo de activo financiero que se desea incluir, para posteriormente seleccionar un conjunto de activos individuales representativos dentro de cada tipo. Como información se seleccionó el precio y los tipos de activos incluidos fueron: FX, *commodities*, índices, *stocks*, bonos y *cryptocurrencies*, además de *Central Bank Rates*.

Como fuentes se utilizaron tres, mediante las APIs gratuitas disponibles para obtener los datos de forma directa en Python, estas fueron: Yahoo Finance, FED, ECB. Es importante destacar las limitaciones que se dan al utilizar herramientas gratuitas para la obtención de datos de los mercados financieros, siendo posiblemente el mayor punto diferencial entre instituciones financieras e individuos, ya que estas poseen acceso a una gran cantidad de datos tanto propios como de la fuente más común, Bloomberg.

En este trabajo se está utilizando como Entorno de Desarrollo Integrado (*Integrated Development Environment* / IDE) Jupyter Lab. Se concreta este detalle debido a errores de funcionamiento al utilizar las APIs mediante otros IDEs como Jupyter Notebook.

Finalmente, se realiza una unión (*merge*) de los datos en base al índice por fecha de las tres fuentes obtenidas, ya que estaban en tablas de datos (*Data Frames*) distintas, aglomerando así todos los datos obtenidos previamente y facilitando su uso en las siguientes etapas.

3.2. Procesado y limpieza de datos

El procesado y limpieza de datos (*data processing & cleaning*) fue la más breve de las etapas. En esta, se buscó la existencia de valores nulos numéricos (*Not a Number* / NaN) y duplicados, para en un segundo paso eliminar las filas que contenían NaN y de esta forma igualar todas las entradas por fecha de datos. Se decidió eliminar las filas que contenían NaN y no otro enfoque alternativo para evitar introducir sesgos artificiales y mantener la precisión temporal, ya que precisamente al tratarse de series temporales financieras, los valores NaN son de días de no cotización, por lo que el interés es adaptar las variables a un calendario común que permita no perder excesivas filas.

En el caso de este estudio se decidió evitar la inclusión de ciertos activos precisamente por una gran diferencia en los días de cotización, como es el caso de los activos chinos con un calendario de cotización considerablemente distinto.

3.3. Análisis Exploratorio de Datos (EDA)

La etapa del Análisis Exploratorio de Datos (*Exploratory Data Anylsis* / EDA) se centra en explorar las características de la serie temporal de la variable a predecir, para poseer un punto de referencia básico. Concretamente, se busca determinar su estacionariedad y su normalidad.

3.3.1. Estacionariedad

Para determinar la estacionariedad se graficó la serie temporal, además de realizar un test ADF. En este test se obtuvo un p-valor de 0,08 aceptando de esta forma la hipótesis nula (H_0) de no estacionariedad al no obtener un p-valor menor a 0,05 asumiendo un 95 % de nivel de confianza.

3.3.2. Normalidad

Respecto a la normalidad, esta se refiere a la distribución normal de sus valores, utilizando para ello los retornos, diferenciando unitariamente la serie original. Sobre la serie diferenciada se realizó un gráfico de probabilidad (*Q-Q plot*), para comparar visualmente si sigue una distribución normal/gaussiana. Además, también se realizó el test Jarque-Bera, obteniendo un p-valor muy próximo a 0 siendo menor que 0,05 por lo que se rechaza H_0

debido al p-valor significativo asumiendo un 95 % de confianza, tratando a la serie como no normal.

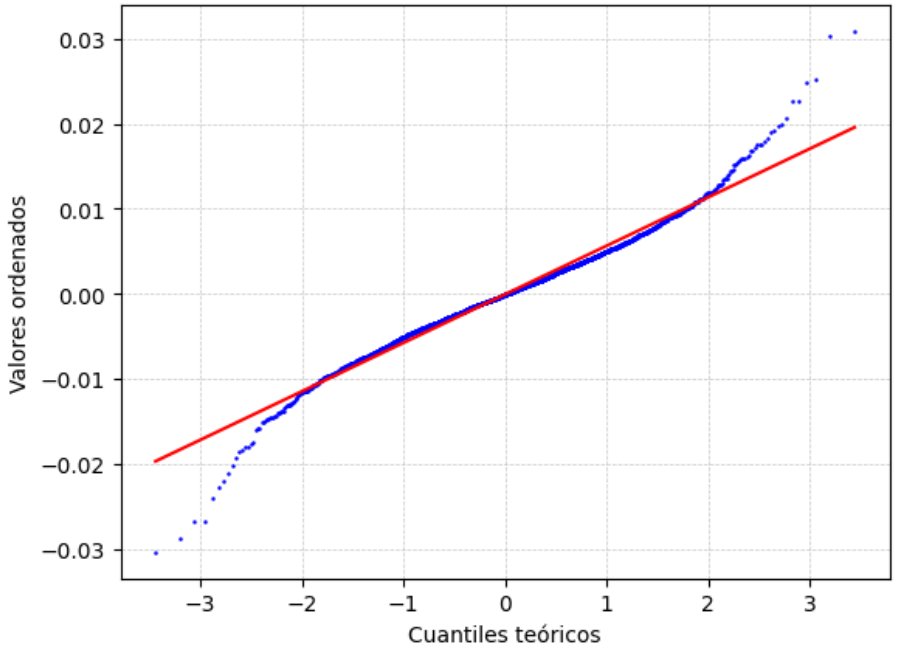

Figura 3.1: *Q-Q plot* sobre los retornos del par de divisas EUR/USD comparados con una distribución normal / gaussiana representada por la línea roja

En la Figura 3.1 se observa lo mencionado, representando la línea roja una distribución normal y los puntos azules los retornos de la serie que está siendo analizada. Se puede apreciar que las colas de la distribución de los retornos de la serie y son considerablemente más alargadas respecto a la distribución normal y, por lo tanto, con mayor probabilidad de retornos más extremos de lo normal, siendo esto algo totalmente común y esperado en series temporales financieras (Mandelbrot, 1997).

3.4. Ingeniería de variables

Esta etapa de ingeniería de variables (*feature engineering*) es la más relevante del trabajo, ya que incluye tanto el proceso de diferenciación fraccionaria como el proceso de PCA, consistiendo en transformar y crear nuevas variables que sean de utilidad como *input* para el modelo.

3.4.1. Variable objetivo como *label* binaria

En primer lugar, se realizó un ajuste en los datos del par de divisas EUR/USD. Ya que se trata de una predicción, el valor a predecir en $t = 0$ será el valor del próximo instante $(t + 1)$, es decir, de $t = 1$. Por ello, se ajustó al día previo la columna del par de divisas EUR/USD mediante un desplazamiento (*shift*), siendo así en cada fecha el valor a predecir el del instante $t + 1$. Recordar que se está resolviendo un problema de clasificación binaria, por lo que se generaron las *labels* para la variable binaria sobre la direccionalidad de los valores ya desplazados. La decisión de etiquetado es fundamental, ya que define la variable objetivo del modelo de clasificación, siendo en este caso concretamente de clasificación binaria. Para ello, se etiquetaron las diferencias calculadas sobre el precio tras el *shift*, con un valor 1 si el cambio entre instantes fue positivo y con un valor 0 si el cambio fue nulo o negativo, simplemente con la intención posterior de obtener una predicción sobre la direccionalidad futura en $t + 1$ y así posicionarse en el mercado a favor de dicha direccionalidad.

También se realiza un conteo de la cantidad de valores generados, obteniendo unas señales objetivo prácticamente balanceadas, *label* 0 con un total de 1211 observaciones y *label* 1 con un total de 1168 observaciones.

3.4.2. Diferenciación fraccionaria

En el contexto de este trabajo, este método se aplica sobre series temporales financieras, concretamente de precios y rendimientos (%), las cuales en su forma original comúnmente no son estacionarias, pero sí poseen memoria. Por lo tanto, el objetivo que se persigue es conservar la máxima memoria, cumpliendo al mismo tiempo la condición de que la serie temporal sea estacionaria. Esto se traduce de forma práctica en diferenciar lo mínimo posible con la condición de obtener un p-valor significativo en el test ADF, siendo menor a $0,05$.

El código del proceso de diferenciación fraccionaria fue desarrollado sobre lo aportado en López de Prado (2018, cap. 5, sec. 5.5.2, p. 83-84). Existen varios métodos para realizar la diferenciación fraccionaria, cada uno con sus problemas propios. En el caso de este trabajo, se ha utilizado el método de diferenciación fraccionaria de ventana de ancho fijo (*Fixed-width window Fractional Differentiation* / FFD), que tiene la ventaja de mantener una considerable parte de la memoria respecto a la serie original, perdiéndose precisamente

en otros métodos relacionados más memoria, aunque no tanta como en una diferenciación entera. Como desventaja, se pierde cierta cantidad de registros iniciales según el valor d de diferenciación utilizado, pero esta contraparte es precisamente la que permite ajustar la longitud de la serie para conservar su memoria.

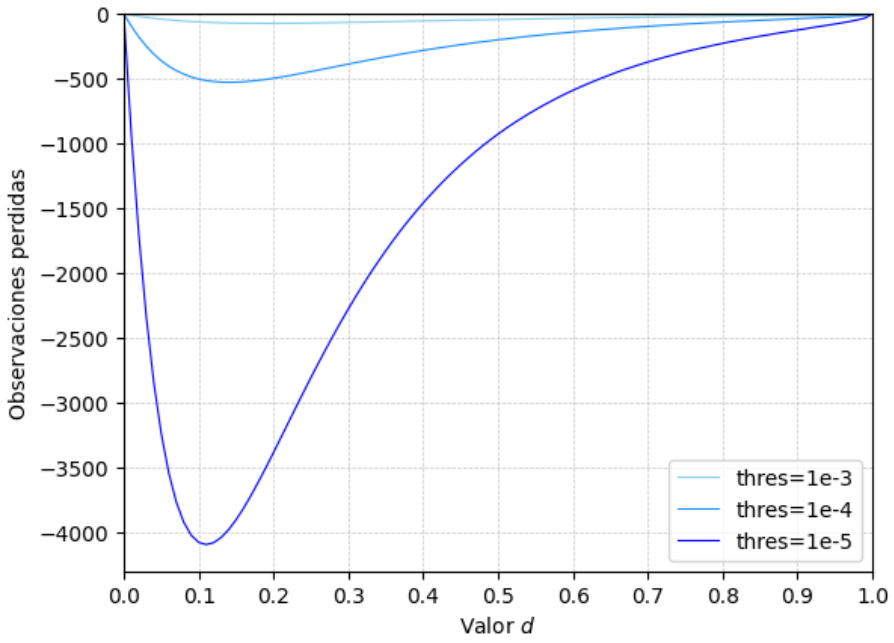

Figura 3.2: Observaciones perdidas según valor de diferenciación / d en el intervalo [0, 1] y precisión decimal / *thres* utilizados en la función de diferenciación fraccionaria

En la Figura 3.2 se ejemplifica lo mencionado, el eje x muestra el valor d utilizado en la diferenciación fraccionaria y el eje y las observaciones iniciales perdidas. Respecto al valor de *thres* (*threshold* / umbral), este hace referencia a la precisión decimal utilizada, suponiendo que a mayor precisión utilizada, mayor cantidad de observaciones perdidas. En este trabajo se ha utilizado una precisión de $1e-4$ o $0,0001$, por lo que la pérdida es de un máximo de 528 observaciones.

En este trabajo, se considerará una serie temporal como estacionaria a toda aquella que obtenga un resultado significativo al aplicar el test ADF utilizando el intervalo de confianza estándar del 95 %, por lo que el p-valor deberá ser igual o menor a 0,05 aceptando así H_1.

Otro aporte relevante realizado ha sido la búsqueda del valor de diferenciación fraccionaria óptimo d^* para cada serie temporal individual y, finalmente, para el conjunto de variables que serán utilizadas como *input*. En este caso, la optimización tendrá en cuenta el valor óptimo d^* para cada serie temporal individual, con el objetivo de conocer cuál es

la serie que sufre una mayor pérdida de observaciones como se ha mostrado en la Figura 3.2. El índice de fechas objetivo será el de la serie con menor longitud, optimizando de esta forma el valor d del resto de series con esta longitud determinada como restricción, al mismo tiempo que se debe obtener un p-valor menor que $0,05$ para considerar la serie como estacionaria con un nivel de confianza del $95\,\%$.

Para hallar el valor óptimo (d^*) de cada serie temporal se ha utilizado el método de optimización de la bisección en una primera capa ($L1$) evaluando el punto medio entre dos valores sucesivamente para acotar de forma eficiente el valor a optimizar, unido en una segunda capa ($L2$) a una búsqueda por fuerza bruta en valores próximos al obtenido previamente para evitar así mínimos locales, ya que el estadístico del test ADF no se comporta de forma totalmente estable en este caso. Las máximas iteraciones en $L1$ están ajustadas a la precisión establecida mediante el parámetro $thres$, ya que $1e-4$ es mayor a $(1/(2^{14}))$ siendo este el espacio resultante en 14 iteraciones o cortes, pero por debajo de 13 iteraciones sería menor, por lo que al añadir la $L2$ es admisible reducir las iteraciones a 12 considerando que el espacio se seguirá incluyendo en la búsqueda. Esto reduce de forma simplificada el costo computacional de n a $log_2(n)$ mediante la búsqueda binaria implementada. En $L2$ la precisión utilizada es similar al $thres$ $1e-4$, ya que es la mínima precisión decimal considerada en el proceso de diferenciación, siendo en este caso el costo computacional similar al original por fuerza bruta de n, pero ayudando a evitar que el estadístico del test ADF quede atrapado en mínimos locales mayores, se evaluarán los 3 próximos puntos en cada dirección al valor heredado de $L1$, siendo un total de 7. Con todo ello, se obtendrá finalmente un *DataFrame* con todas las series temporales de las variables utilizadas como *input* en la función, siendo en el *output* estacionarias con un valor óptimo d^* considerando la restricción de longitud mínima. Recordar que en la Figura 2.1 se muestran las distintas formas de una serie temporal concreta, siendo la obtenida en el *output* la línea roja con etiqueta "Diferenciación fraccionaria general", es decir, con restricción de longitud.

La Figura 3.3 plasma de forma visual la búsqueda del valor de diferenciación fraccionaria óptimo d^*, utilizando como ejemplo la serie temporal del rendimiento del bono europeo a 5 años (EU5Y) y la conservación de memoria a través de la correlación de Pearson que esto supone. El eje x plasma el valor d utilizado en la diferenciación fraccionaria, hasta llegar al $d = 1$ de la propia diferenciación unitaria. Mencionar también, que $d = 0$ equivaldría a la serie temporal original. La línea roja horizontal presenta la frontera de significatividad del

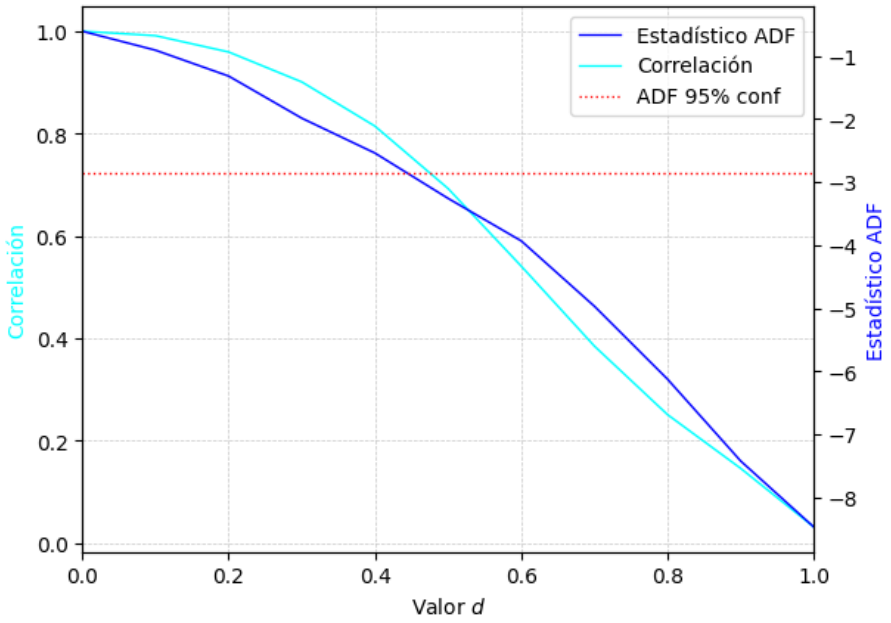

Figura 3.3: Ejemplo de la serie temporal del rendimiento del bono europeo a 5 años representando Memoria / correlación de Pearson respecto a la serie original según el valor d utilizado en la diferenciación fraccionaria en el intervalo [0, 1] y el punto de intersección con el test ADF al 95 % de confianza para considerar la serie como estacionaria

test ADF, por lo que, cuando la línea azul oscura la cruza a su parte inferior, supone que la serie ya es considerada estacionaria al 95 % de confianza y por lo tanto con un p-valor menor a $0,05$. En este caso, el valor de diferenciación óptima es $d^* = 0{,}448$ conservando una memoria / correlación de Pearson respecto a la serie original de 0,76. Por otro lado, la diferenciación unitaria convencional con $d = 1$, únicamente conserva una correlación de 0,03 respecto a la serie original (López de Prado, 2018, cap. 5, sec. 5.6, p. 84-85).

3.4.3. Análisis de Componentes Principales (PCA)

Se prosigue tratando la implementación del PCA, en este caso, realizando únicamente ligeras modificaciones pero sin ajustes significativos sobre el código original aportado en López de Prado (2018, cap. 8, sec. 8.4.2, p. 119). El valor utilizado en *varThres* (*variance threshold* / umbral de varianza) ha sido de 0,95 por lo que los componentes generados explicarán como mínimo un 95 % de la varianza del conjunto de datos *input*, se ha elegido este valor como equilibrio entre la varianza explicada y el número de componentes que selecciona (31) respecto a la cantidad de variables originales (84). Destacar que, a

mayor varianza explicada mínima establecida, mayor será también el número de componentes generados seleccionados, por lo que el valor de *varThres* puede ser ajustado según necesidades particulares. Recordar la Figura 2.2 que mostraba la varianza explicada por componente.

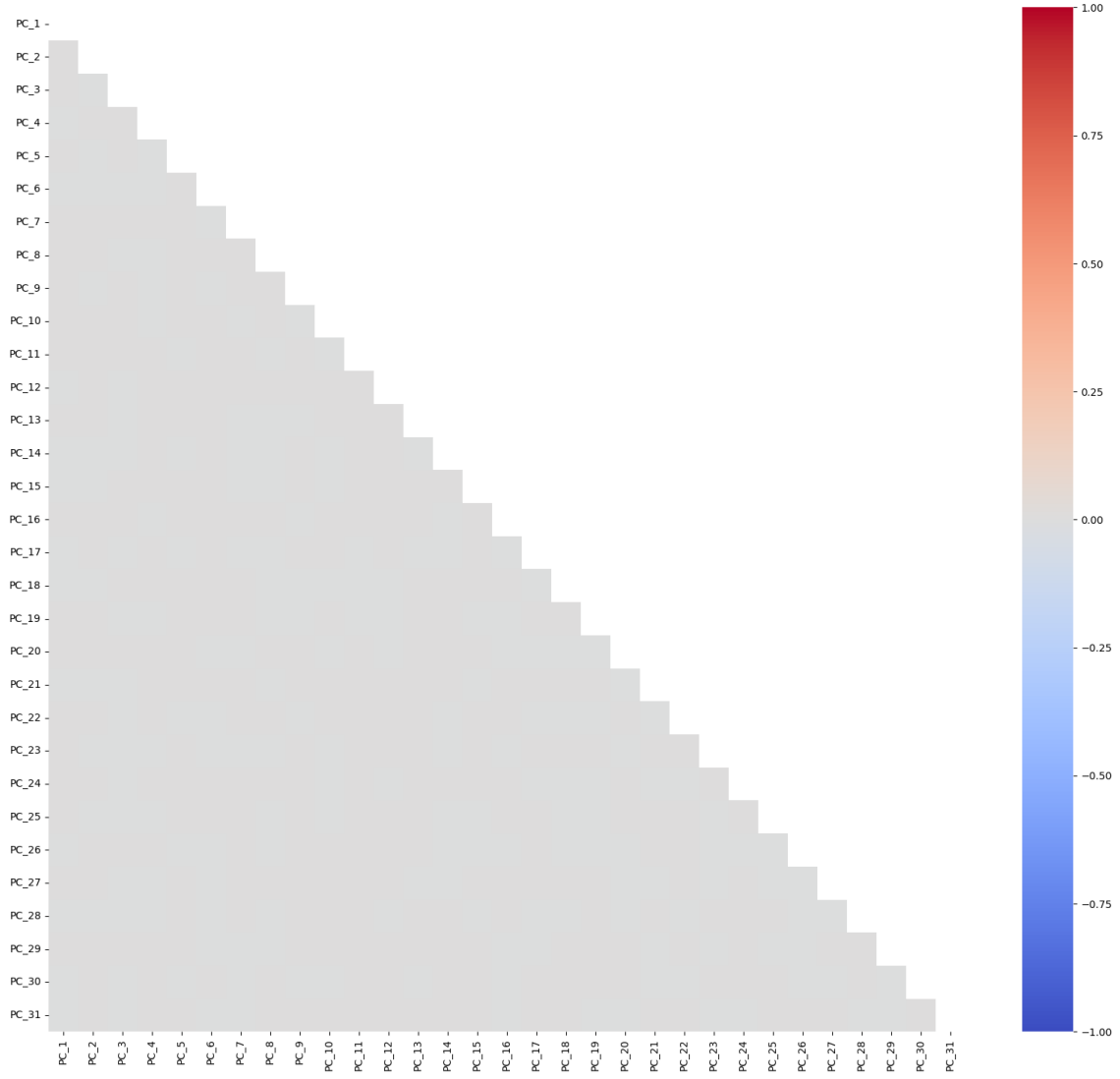

Figura 3.4: *Heatmap* de los componentes generados mediante el PCA, apreciando una correlación nula entre los componentes ortogonales generados por el método

En la Figura 3.4 se demuestra visualmente mediante un *heatmap* que los componentes generados mediante el PCA poseen una correlación de Pearson nula entre sí, siendo de esta forma los componentes totalmente ortogonales. Esto supone que al utilizarlos como *input* en el modelo no generarán efectos de sustitución y, por lo tanto, no reducirán la importancia de otros componentes al ser elegidos.

En este trabajo se ha utilizado el PCA de forma global, es decir, sobre todas las variables, pero esta no es la única forma. También se podría aplicar el PCA de forma local, es decir, por distintas segmentaciones o grupos dependiendo de distintos criterios. Un ejemplo claro sería aplicarlo sobre cada tipo de activo u otro tipo de criterios relacionados. Esto implicaría aplicar el PCA sobre todos los bonos en su conjunto (DGS + EU) o, de otra forma, aplicar el PCA sobre los bonos de cada zona económica, por un lado DGS y por otro EU. Unido a la utilización de un *heatmap*, también podría utilizarse sobre las concentraciones (*clusters*) de correlación de Pearson que se aprecien. De esta forma, teóricamente se simplifica la información que estas series temporales contienen, facilitando al modelo el obtener señal sin necesidad de elegir entre un componente u otro, ya que serían ortogonales entre sí.

3.5. Selección de variables

Selección de variables (*feature selection*) es la etapa que incluye la implementación del MDI, seleccionando las variables que aportan más señal a los resultados y descartando de esta forma variables que se cree que aportan más ruido que señal al modelo. De forma secundaria, también se han utilizado mapas de calor (*heatmaps*) para apreciar las relaciones entre las distintas variables.

3.5.1. *Heatmap* sobre diferenciación fraccionaria

Mediante un *heatmap* en la Figura 3.5, se muestran las correlaciones de Pearson entre las variables que utilizaremos como *input*. Este *heatmap* ha sido realizado sobre las variables diferenciadas fraccionariamente, ya que, como se ha comentado en la Sección 2.2.1, de esta forma se conserva mayor memoria / correlación de Pearson, que es precisamente lo que se utiliza para construir el *heatmap*. De esta forma, se podrán apreciar visualmente de forma simple las relaciones existentes entre las distintas variables y obtener así una percepción inicial más acertada. Unas agrupaciones obvias con altos valores de correlación de Pearson son los rendimientos de los bonos, tanto por la parte estadounidense (DGS) como por la parte europea (EU). Este es un claro ejemplo que sufriría de efectos de sustitución, ya que, se poseen datos de distintos vencimientos en años, por lo que el modelo de *machine learning* podría priorizar un vencimiento sobre otros, estando realmente todos íntimamente relacionados y perdiendo así importancia los no elegidos.

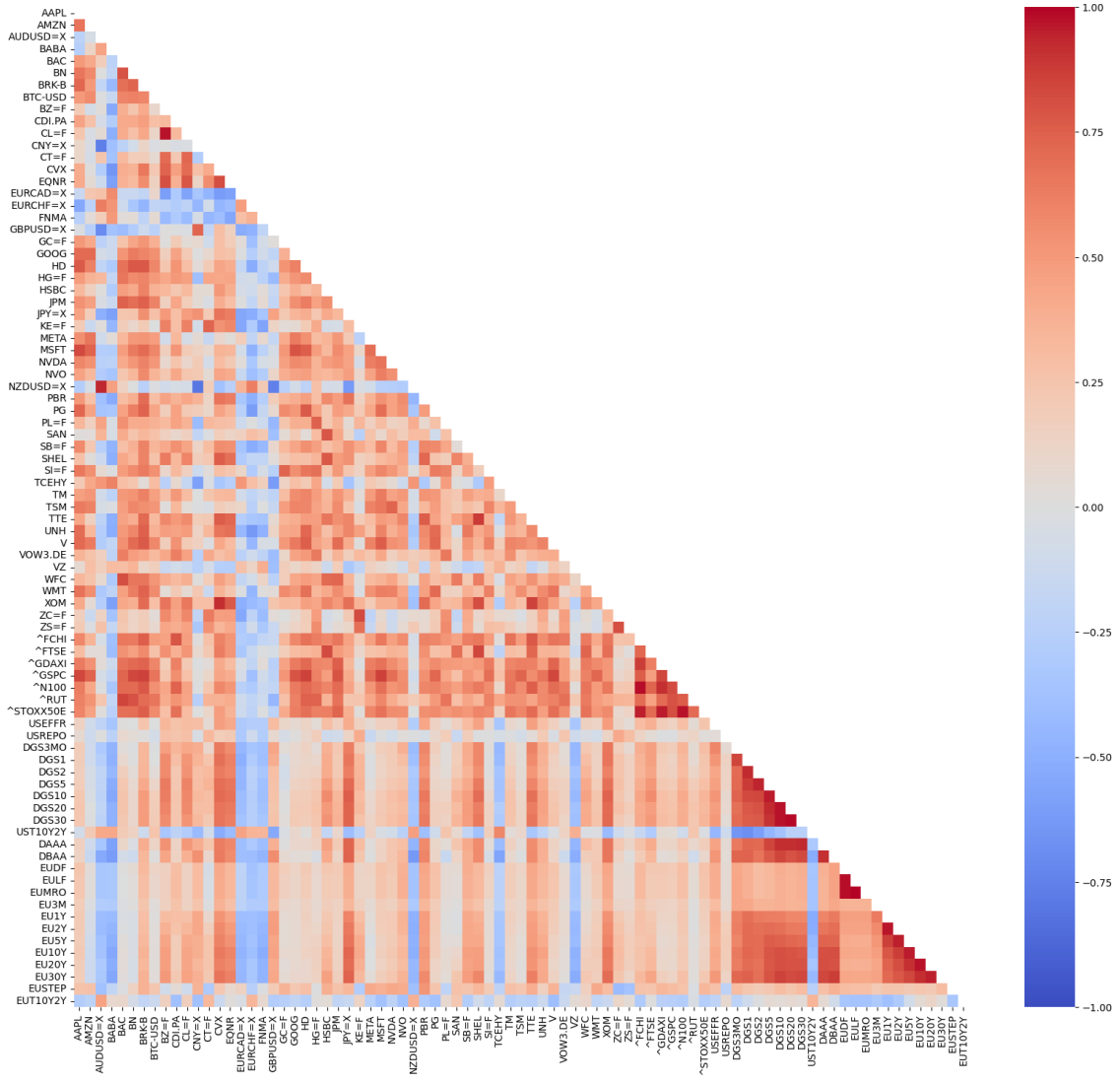

Figura 3.5: *Heatmap* de las variables diferenciadas fraccionariamente

Matizar que se está utilizando la terminología de bonos con el objetivo de simplificar, ya que, técnicamente, dependiendo del país, la terminología cambia ligeramente según la duración de estos.

3.5.2. *Mean Decrease Impurity* (MDI)

El código se ha desarrollado en base a lo aportado en López de Prado (2018, cap. 8, sec. 8.3.1, p. 115-116), realizando únicamente la ligera modificación de incluir un parámetro de estado aleatorio (*random state*), tanto para facilitar la reproducibilidad por parte del lector, como para evitar indicios de fraude científico mediante la repetición del proceso hasta conseguir aleatoriamente el resultado deseado.

Se recuerda que el proceso fue detallado en la Sección 2.2.3 y en la Figura 2.3 se muestra el resultado tras aplicar el método MDI sobre los componentes generados mediante el PCA. El desarrollo en código para la realización de la Figura 2.3 se basó en lo aportado en López de Prado (2018, cap. 8, sec. 8.6, p. 123-124), aunque fue adaptado para utilizar de forma específica el desarrollo sobre el MDI.

Para realizar la selección de variables se filtraron los componentes, conservando únicamente los que sobrepasen la línea roja vertical restando a su media de importancia la desviación típica de importancia, representando la línea roja vertical la importancia promedio (importancia total dividida entre el número de componentes) que debería poseer cada componente con un peso similar, todo ello plasmado en la Figura 2.3. Finalmente, únicamente cumplieron estas condiciones 11 de 31 componentes, los cuales fueron utilizados como *input* para así comprobar si realmente son estos componentes los que aportan más señal y menos ruido, mejorando así el rendimiento del modelo.

3.5.3. Comparación entre PCA y MDI

Por último en esta etapa, para verificar que los resultados obtenidos sobre las variables mediante el PCA y el MDI poseen correlación en cuanto a sus conclusiones, otorgando más relevancia o menos a ciertas variables, se desarrolla lo presentado en López de Prado (2018, cap. 8, sec. 8.5, p. 119-121). Esta comparación surge debido a los distintos enfoques de cada método, obteniendo el PCA sus conclusiones con un enfoque de *unsupervised learning* y el MDI con un enfoque de *supervised learning*. Por ello, idealmente, los resultados de ambos métodos deberían arrojar un resultado similar, confirmando de esta forma el haber identificado una importancia de ciertas variables de forma común mediante enfoques de

aprendizaje distintos.

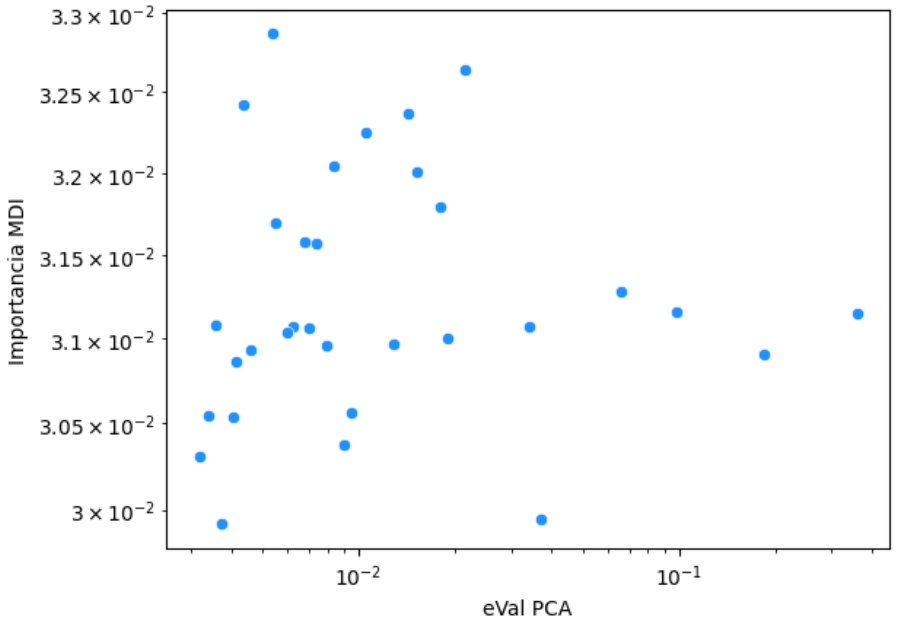

Figura 3.6: Relación entre la varianza explicada por cada variable en el PCA y la importancia obtenida en el MDI mediante diferenciación fraccionaria, todo ello en escala logarítmica

La Figura 3.6 representa mediante un gráfico de dispersión (*scatter plot*) la relación entre la jerarquía determinada mediante PCA y la jerarquía determinada mediante MDI sobre las variables diferenciadas fraccionariamente, todo ello en escala logarítmica. A simple vista se puede apreciar que no existe relación alguna, ya que los bajos valores en PCA poseen altos valores en MDI y viceversa, prueba de ello es el valor de -0,05 obtenido mediante la correlación de Pearson, junto con el valor de 0,02 del Tau de Kendall ponderado hiperbólicamente. Este Tau utilizado se ha realizado sobre la importancia de componentes del MDI y del *ranking* invertido del PCA, por lo que cuanto más cercano a 1 sea este valor, mayor será la consistencia entre ambos.

Si se representa la misma relación pero sobre las variables diferenciadas unitariamente, el resultado parece mejorar, aunque sin estar estos resultados muy correlacionados. En la Figura 3.7 visualmente se aprecia un indicio de relación, confirmándolo mediante el valor 0,29 de la correlación de Pearson y el valor de 0,52 de *weighted Kendall's Tau*.

Por todo ello, existe una sospecha de que el MDI puede sufrir de *overfitting*. Otra posible causa puede ser que se han utilizado como *input* en el PCA variables que poseen

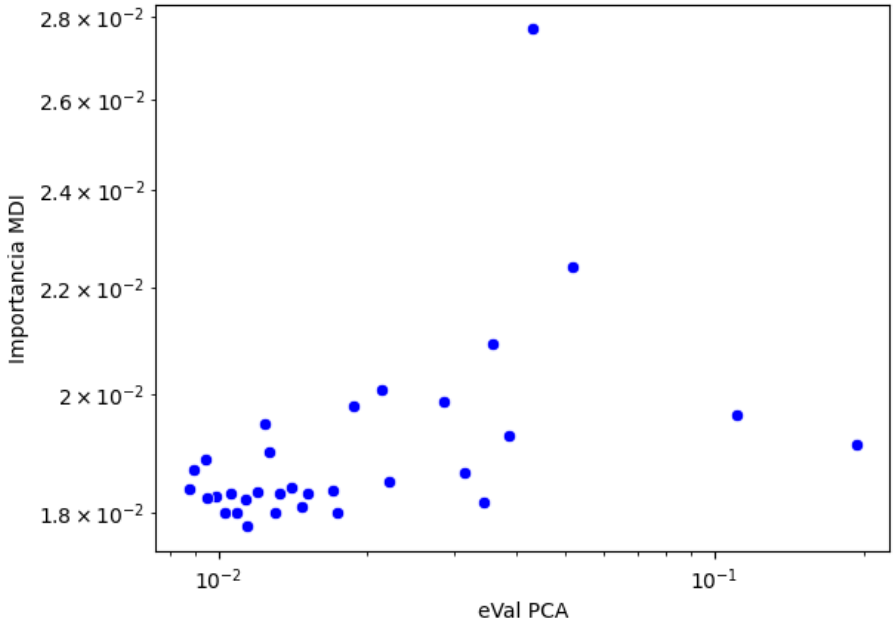

Figura 3.7: Relación entre la varianza explicada por cada variable en el PCA y la importancia obtenida en el MDI mediante diferenciación entera, todo ello en escala logarítmica

gran varianza explicada respecto al conjunto de variables, pero sin ser relevantes para el objetivo del modelo. También puede contribuir el hecho de que al capturar el PCA relaciones lineales, oculte las no lineales.

3.6. Entrenamiento y evaluación de modelos

En esta última etapa, entrenamiento y evaluación de modelos (*model training and evaluation*) se utilizó todo lo aportado en fases anteriores para el entrenamiento de los modelos, ya que realmente, lo que implica una mayor cantidad de esfuerzos no es la construcción del modelo en sí, sino el refinamiento de los datos que van a ser utilizados como *input* en este. De otro modo, sería tan sencillo como aglutinar una gran cantidad de variables y utilizarlas como *input* directamente sin criterio alguno.

Para incluir de forma simultánea en los cinco modelos de clasificación los distintos *inputs* generados y así poder comparar su desempeño, se ha creado un diccionario de Python con las seis formas que se utilizarán como *input*, siendo los datos originales sometidos a los métodos: diferenciación unitaria, PCA sobre diferenciación unitaria, MDI sobre PCA de diferenciación unitaria, diferenciación fraccionaria, PCA sobre diferenciación fraccionaria, MDI sobre PCA de diferenciación fraccionaria.

Respecto al motivo de aplicación de una variedad de modelos y no de uno solo, esto es con el objetivo de comparar cómo los distintos métodos se comportan ante las distintas arquitecturas internas de los algoritmos de cada modelo, ya que cada uno aprende de distintas formas sobre los datos *train*. El LR puede considerarse como el modelo más simple posible, siendo este el base. El SVM se enfoca en la búsqueda de un hiperplano para maximizar el margen entre clases. El KNN se basa en la distancia a los vecinos para asignar una clase por mayoría. El RF combina *decision trees* de forma independiente y paralela para asignar una clase por mayoría. El XGBoost combina *decision trees* de forma secuencial mediante refuerzo para asignar una clase. Se puede apreciar que la forma de aproximarse al problema de estos algoritmos varía en gran medida basándose en conceptos diferentes, por lo que los resultados deberán diferir entre sí al capturar de forma distinta los patrones y ruido. Han sido elegidos estos cinco modelos tanto por sus diferencias en el enfoque de aprendizaje como por la accesibilidad y uso generalizado que poseen, ya que son algunos de los algoritmos de *machine learning* más comunes.

Se comprueba ahora cada combinación posible entre los cinco modelos y los seis métodos de tratamiento de datos utilizados como *input*. El proceso comienza con una separación entrenamiento-validación (*train-test split*) mediante una función propia, la cual tiene en cuenta la secuencialidad al tratarse de series temporales, por lo que se debe respetar el orden de los datos en la partición generada. Tras ello, se estandarizan los datos mediante el escalado estándar (*standard scaler*) para homogeneizar unidades y dispersión de las variables, beneficiándose algunos modelos de las variables transformadas con media 0 y varianza 1, siendo este el motivo por el que ha sido elegido.

Ahora sí, se declara el modelo con sus respectivos hiperparámetros para facilitar el aprendizaje, mencionar que se realizaron algunas pruebas en los hiperparámetros más relevantes para encontrar los valores que mejor funcionasen en estos. Los hiperparámetros seleccionados para cada modelo fueron los siguientes:

LR: se utilizó como método de optimización `solver=lbfgs`, con unas iteraciones máximas `max_iter=10000`, inverso de la regularización L_2 `C=0.01` y semilla aleatoria `random_state=42`.

SVM: se incluyó como función núcleo `kernel=rbf` aportando no linealidad, con un inverso de la regularización L_2 `C=1.0` y semilla aleatoria `random_state=42`.

KNN: como número de vecinos más cercanos a considerar `n_neighbors=75`, con el peso de los vecinos `weights=distance` ponderándolos según distancia.

RF: el número de árboles de decisión `n_estimators=1000`, con una profundidad máxima `max_depth=4`, una función para medir la calidad de la división `criterion=gini`, cantidad de variables consideradas para dividir el nodo `max_features=sqrt` suponiendo la raíz cuadrada del total, muestras con reemplazo `bootstrap=True` y semilla aleatoria `random_state=42`.

XGBoost: se estableció como función objetivo `objective=binary:logistic` la logística binaria, con evaluación del modelo durante el entrenamiento `eval_metric=logloss`, número de árboles `n_estimators=1000`, tasa de aprendizaje `learning_rate=0.01`, profundidad máxima de cada árbol `max_depth=2`, proporción de muestras para cada árbol `subsample=0.8`, proporción de características en cada árbol `colsample_bytree=0.5`, mínima reducción de pérdida para partición en un nodo `gamma=10` y semilla aleatoria `seed=42`.

Tras ello, se predicen los valores que serán utilizados para evaluar el modelo. Se declara un nuevo *DataFrame* que recogerá las *labels* predichas y las de *test*, además de incluir una nueva columna que será representada con un 1 si la predicción ha sido correcta y con un -1 si ha sido errónea, lo cual facilitará el posterior cálculo de los rendimientos obtenidos.

Se calculan ahora las dos métricas de evaluación propuestas, el *accuracy* y el CR. Obteniendo así los resultados de ambas métricas para cada uno de los métodos y modelos, haciendo fácilmente comparables e interpretables los resultados.

Es importante destacar que se considera como métrica más representativa del rendimiento de cada modelo el CR, debido a que no se trata únicamente de predecir correctamente el resultado como hace el *accuracy*, sino de predecir correctamente de forma prioritaria los mayores movimientos diarios, ya que serán los que mayores ganancias o pérdidas generarán. Por ello, predecir erróneamente como venta un movimiento de +0,1 % no será relevante, pero predecir correctamente como venta un movimiento de -1 % sí que tendrá un gran impacto. Por lo que de cara al Capítulo 4 se utilizará únicamente el CR como métrica más representativa, ya que el *accuracy* posee algunas carencias en este contexto, pero ha sido útil para apreciar lo comentado en el desarrollo del código.

Finalmente, con intención de apreciar el CR se realizó una visualización que será empleada en 4 para comparar la estrategia de mantener (*hold*) con la del modelo sobre el que se desee comparar. Esta estrategia *hold* supone simplemente comprar en el instante inicial del período ($t = 0$) y no realizar acciones adicionales. Siendo el caso del CR el seguir las indicaciones de compra o venta sugeridas por el modelo evaluado, pudiendo las posiciones mantenerse o invertirse.

Capítulo 4

Análisis de resultados

En este capítulo se compartirán los resultados obtenidos en cada modelo de clasificación, utilizando el CR como métrica de evaluación entre ellos. Esta comparación entre distintos modelos y métodos de tratamiento de datos permitirá evaluar varios aspectos, como la combinación de qué modelo con qué método obtiene el mejor resultado global, qué modelo funciona mejor con cada método y qué método funciona mejor con cada modelo.

	Modelo de clasificación				
Método	LR	SVM	KNN	RF	XGBoost
Integer diff	151,25	113,46	124,86	131,56	150,23
Integer PCA	**164,64**	133,09	147,86	127,42	136,67
Integer MDI	146,26	82,33	103,85	110,24	129,68
Fractional diff	1,45	9,75	3,36	-0,97	1,73
Fractional PCA	29,69	21,04	17,57	41,42	23,07
Fractional MDI	33,69	21,72	31,06	27,09	23,44

Tabla 4.1: CR (como %) mediante los diferentes modelos de clasificación y métodos de tratamiento de datos

La Tabla 4.1 recoge el CR de cada combinación entre modelo y método posible, siendo los utilizados cinco modelos y seis métodos, por lo que el total de combinaciones a comparar sumará 30. El mejor resultado global fue la combinación del modelo LR con el método *Integer* PCA con un CR de 164,64 %. Además, el total de transacciones realizadas fue de 258 sobre un total de 371 posibles.

La Figura 4.1 muestra la importancia de cada componente utilizado en el modelo LR como *input* en el entrenamiento, es decir, sobre lo que ha aprendido el algoritmo para producir los resultados. Una vez más, la línea roja vertical representa el promedio teórico de importancia de cada componente (importancia total / número de componentes) con un peso similar. En este caso, esto aporta explicabilidad al modelo, es decir, en qué se basa para generar los *output* de indicaciones de compra o venta según la dirección del mercado

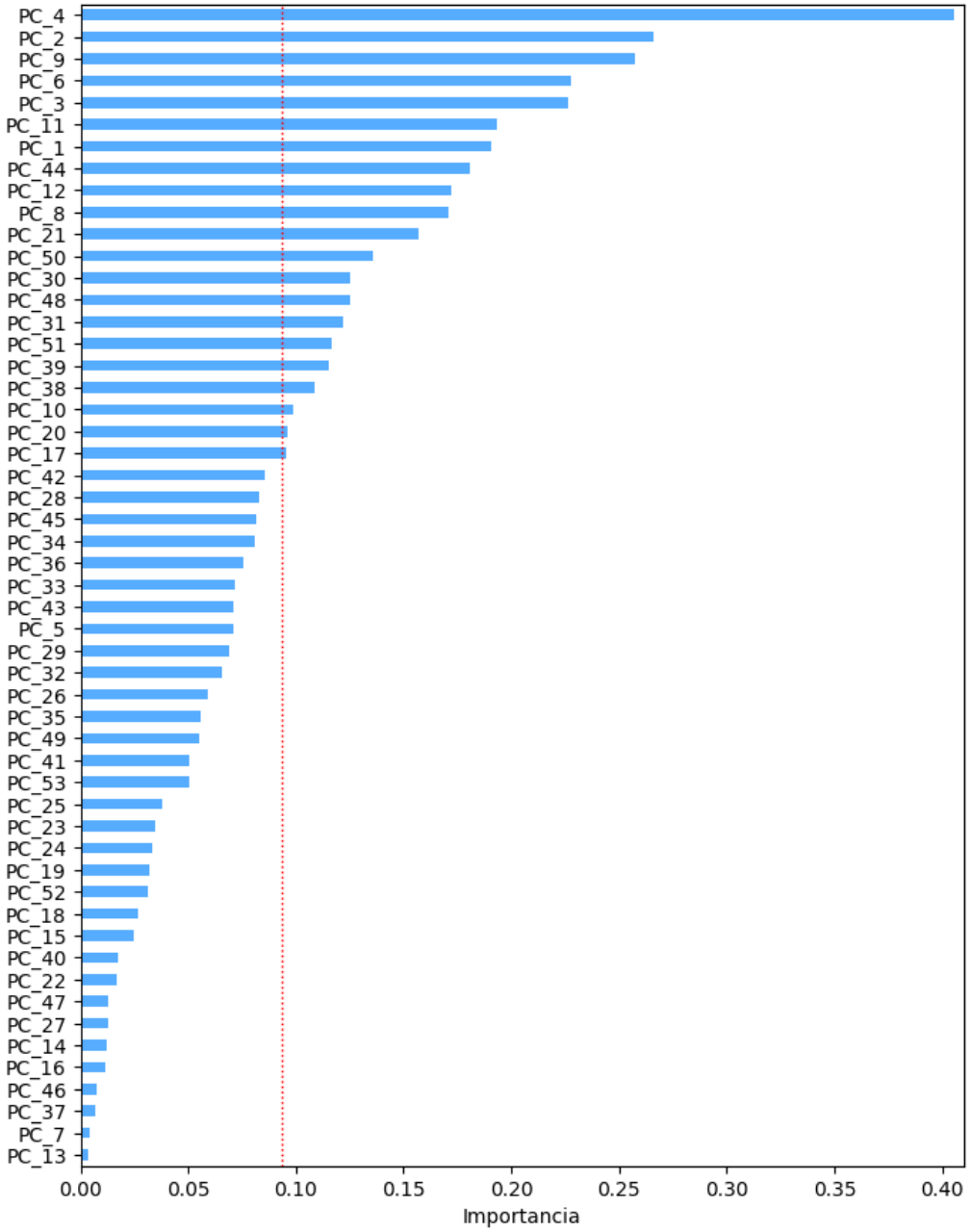

Figura 4.1: Importancia de cada componente en el LR mediante método *Integer* PCA, con variables ordenadas de forma descendente y representando la importancia promedio mediante la línea roja vertical

predicha. Este es el símil a abrir la *black box*, concepto desarrollado en la Sección 2.2.3 en mayor extensión, ya que al saber sobre lo que ha aprendido el modelo se pueden realizar ajustes y experimentar con distintas combinaciones de variables.

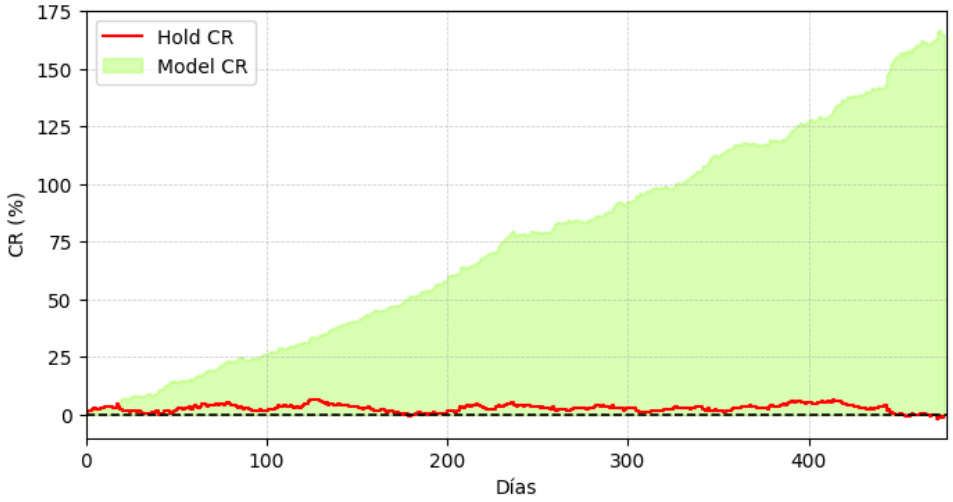

Figura 4.2: Comparación del CR entre estrategia *hold* y el modelo LR mediante el método *Integer* PCA

Otro elemento relevante que mostrar mediante la Figura 4.2 es la comparación del CR de la estrategia *hold* respecto al CR de la estrategia del modelo LR mediante el método *Integer* PCA. Puede apreciarse que la trayectoria del CR de LR es considerablemente estable y con inclinación ascendente. Por el contrario, el CR de *hold* es prácticamente nulo, sin ninguna fluctuación direccional significativa.

Respecto al resto de modelos utilizados, en el SVM y KNN el mejor método de combinación fue también el *Integer* PCA, con un CR de 133,09 % y 147,86 % respectivamente. En el caso del RF y XGBoost el mejor método de combinación fue el *Integer diff*, con un CR de 131,56 % y 150,23 % respectivamente.

Respecto a los métodos utilizados, tanto en *Integer diff* con un 151,25 %, *Integer* PCA con un 164,64 %, *Integer* MDI 146,26 % y *Fractional* MDI con un 33,69 %, el modelo que obtuvo un CR más alto fue el LR. En el resto de casos, mediante *Fractional diff* el mejor modelo fue SVM con un 9,75 % y mediante *Fractional* PCA fue RF con un 41,42 %.

Por último, mencionar que todos los modelos de forma consistente obtienen un peor resultado con los métodos que implican *Fractional* y especialmente con el método *Franctional diff*, el cual posee los CR más bajos. Además, en la parte de métodos *Integer* el *Integer MDI* presenta en todos los modelos un CR inferior al resto de métodos.

Capítulo 5

Conclusiones

Los objetivos de este trabajo son, en una primera fase, comprender en profundidad de forma teórica los métodos de tratamiento de datos financieros propuestos. En una segunda fase, implementarlos de forma práctica mediante código. Y en una tercera y última fase, utilizarlos mediante algoritmos de *machine learning* de clasificación binaria y de esta forma comparar su desempeño. El conocimiento necesario para el desarrollo de los métodos de tratamiento de datos mencionados fue obtenido en López de Prado (2018), siendo estos la diferenciación fraccionaria, el PCA y el MDI.

5.1. Discusión de resultados

Los modelos de clasificación binaria con objetivo de generar señales de compra o venta de frecuencia diaria sobre la dirección futura del par de divisas EUR/USD, se basaron en algoritmos de *machine learning* como LR, SVM, KNN, RF o XGBoost. Estos han mostrado diferencias significativas en sus rendimientos dependiendo del método de tratamiento de datos aplicado, siendo la mejor combinación la del modelo LR junto con el método *Integer* PCA, por lo que en el caso de este método se aprecia que la reducción de los efectos de sustitución mediante la ortogonalización de las variables ayuda a que el modelo obtenga mejores resultados.

También se puede apreciar que los modelos como RF y XGBoost basados en *ensembles* responden mejor ante las variables diferenciadas sin necesidad de ortogonalización previa, por lo que parecen estar menos afectados por los efectos de sustitución, manejando mejor la multicolinealidad de las variables. El motivo de esto puede deberse a que la independencia de variables no es un supuesto en este tipo de modelos basados en *decission trees*, siendo capaces además captar relaciones no lineales, por lo que la linealización realizada en la ortogonalización puede ser contraproducente en este caso.

Por otro lado, en la totalidad de los modelos, el CR obtenido mediante el *Integer* MDI ha sido inferior respecto a los dos métodos comentados previamente, lo que indica que los componentes seleccionados mediante este método sobre los componentes generados mediante el PCA no son suficientes para capturar toda la información relevante y mejorar

el rendimiento, siendo por lo tanto necesaria la información proporcionada por el resto de componentes no incluidos.

Respecto a la *fractional differentiation* aplicada mediante FFD y los métodos que surgen de ella como *Fractional* PCA y *Fractional* MDI, los modelos utilizados parecen no ser capaces de captar patrones que mejoren el rendimiento, obteniendo en estos tres métodos un CR considerablemente menor en comparación con el resto de la parte *Integer*. Esto puede deberse a la incapacidad de los modelos para captar secuencialidad y patrones temporales presentes en este tipo de transformaciones.

5.2. Conclusión general

Este trabajo ha logrado implementar y evaluar diferentes métodos de tratamiento de datos financieros en combinación con algoritmos de *machine learning*, cumpliendo así con el objetivo inicial de analizar su aplicabilidad y eficacia sobre datos reales. La combinación de LR con *Integer* PCA ha resultado ser la más efectiva, mostrando que, bajo ciertos supuestos, la ortogonalización de variables puede ser muy beneficiosa para modelos lineales. En general, se demuestra cómo distintos métodos de tratamiento de datos pueden influir significativamente en el rendimiento de modelos predictivos aplicados a finanzas.

5.3. Limitaciones y mejoras

Una limitación principal del trabajo es la ausencia de modelos que capten explícitamente secuencialidad en los datos, como redes neuronales recurrentes (*Recurrent Neural Networks* / RNN), redes de memoria a largo corto plazo (*Long Short-Term Memory* / LSTM) o arquitecturas basadas en *Transformers* las cuales emplean mecanismos de atención sin necesidad de recurrencia. En futuras investigaciones se podrían implementar estos modelos para evaluar si son capaces de extraer patrones temporales complejos que los modelos actuales no detectan, especialmente con los métodos basados en diferenciación fraccionaria por su conservación de memoria de las series temporales.

Respecto a los métodos, existen varias alternativas que pueden ser implementadas de forma conjunta a las actuales. Referente a la ortogonalización de variables, se podría comenzar utilizando el PCA de forma aislada sobre tipos de activos o grupos, además de la utilización de otros métodos relacionados como mínima redundancia máxima relevancia (*Minimum Redundancy Maximum Relevance* / mRMR), que elige variables con alta corre-

lación con la variable objetivo pero con poca correlación entre ellas. Sobre la importancia de variables López de Prado (2018) también propone otros métodos como disminución media de precisión (*Mean Decrease Accuracy* / MDA) o importancia individual de variable (*Single Feature Importance* / SFI). En Li, Wang, Basu, Kumbier, y Yu (2019) se propone otro método interesante en el que se robustece el MDI usando muestras fuera del conjunto de entrenamiento (*out of bag* / oob), formando así el método denominado como MDI-oob. Finalmente, respecto a la evaluación de los modelos, podrían implementarse funciones que ayuden a elevar la robustez de los resultados, como la validación cruzada (*Cross Validation* / CV), la validación cruzada K-Fold purgada (*Purged K-Fold CV*) o la utilización de varios tamaños en la partición del *train-test split*.

Referencias bibliográficas

American Statistical Association. (2022). *Ethical guidelines for statistical practice.* Descargado de `https://www.amstat.org/docs/default-source/amstat-documents/EthicalGuidelines.pdf?v=0525` (Accedido: 29-05-2025)

Breiman, L. (2001). Random forests. *Machine learning*, *45*, 5–32.

Chen, T., y Guestrin, C. (2016). Xgboost: A scalable tree boosting system. En *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Cortes, C., y Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*, 273–297.

Cover, T., y Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, *13*(1), 21–27.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *20*(2), 215–232.

Frost, J. (2020). Understanding significance levels in statistics. *URL: https://statisticsbyjim.com/hypothesis-testing/significance-levels/*.

Gangutia, I. (2025). *Repositorio del trabajo de fin de grado: Machine learning financiero aplicado a la predicción del eur/usd.* `https://github.com/Gangutia/TFG`. (Accedido: 30-05-2025)

Hyongdo, L. (2014). *The god of trading, honma.* Lee Hyongdo.

Li, X., Wang, Y., Basu, S., Kumbier, K., y Yu, B. (2019). A debiased mdi feature importance measure for random forests. *Advances in Neural Information Processing Systems*, *32*.

López de Prado, M. (2018). *Advances in financial machine learning.* John Wiley & Sons.

Mandelbrot, B. B. (1997). *The variation of certain speculative prices.* Springer.

McKinney, W. (2012). *Python for data analysis: Data wrangling with pandas, numpy, and ipython.* O'Reilly Media, Inc.

Van Der Wijst, N. (2013). *Finance: a quantitative introduction.* Cambridge University Press.

Zuckerman, G. (2019). *The man who solved the market: How jim simons launched the quant revolution.* Penguin.