Introducción a la negociación algorítmica y a los métodos de aprendizaje automático

David Rafael Torres Daza

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: José Tomás Alcalá Nalvaiz
12 de junio de 2024

Abstract

The purpose of this paper is to explore in depth the impact of machine learning on automated trading, identifying how these algorithms have not only improved traditional trading but also introduced new investment strategies. Through the discussion, we will outline the evolution of Machine Learning (ML), apply the fundamental principles of automated trading and analyze specific ML models applied to this field. This study will not only provide a theoretical understanding of these processes, but will also include a case study to illustrate their application and effectiveness in the real financial market.

The paper is structured in three main chapters. In the first chapter, ML, a branch of artificial intelligence that combines statistical analysis and computer science, is introduced. Three ML methods are explained: supervised learning, which uses labeled data to train models capable of predicting outputs based on the data from a given set of data known; the no supervised learning, that analyzes data without being labeled to find patterns or underlying groupings; and the reinforcement learning which is based in agents that take decisions in a dynamic environment to maximize rewards along the time. Moreover, it is argued several applications of ML in the financial sphere, as the price movements predictions and the investment portfolio optimization. In this chapter, it is also tackled the concept of automatic trading, also the automatic trading or algorithmic trading. It is a methodology in which the purchase and sale decisions in financial markets are carried out by software which uses default algorithms. These algorithms allow to operate with a speed and precision that get over human capabilities, eliminating the emotions of trading decisions and allowing to answer quickly to the market conditions. The automatic trading systems use mathematic models and quantitative analysis to trigger trading signs.

In the second chapter, it is detailed two models of ML which are applicable to automatic trading: the methods based on trees (Random Forests) and the support vector machines (SVM). The methods based on trees divide the characteristics space in simple regions to take decisions based on some data observations, while the SVM classify data through the construction of the best hyperplane that divides the classes in the characteristic space. These methods were selected by their efficiency in complex contexts and no typical linear of financial markets.It explains in detail the construction and training of these models, as well as their advantages and disadvantages.

The third chapter is devoted to a case study whose objective is to apply ML models to predict asset prices in the financial market and to develop an automatic trading system. The investment portfolio used includes shares of the Ibex 35 and the S&P 500. Two main strategies are implemented: one based on the prediction of future returns using Random Forest and another that combines this prediction with the classification of price trends by means of SVM. The results of these strategies are compared with those obtained using a technical analysis strategy based on moving averages.

In the appendices, detailed and additional analyses are included to complement the third chapter, showing the results of the evolution of the capital and the trades made. The process of optimizing hyperparameters for Random Forest models is also described, as well as the importance of the features used in the data set. An outline of the code used in Python is provided for the implementation and evaluation of the trading strategies. These appendices offer a more complete technical overview of the methods and results obtained, underlining the sturdiness and effectiveness of the ML models applied in this study.

To sum up, this project shows that the ML models are effective to develop automatic trading strategies which generate positive and consistent performances. We have tested the automatic learning models applied on financial trading with simple models both in the number of characteristics and the parameter optimization. This allows the study of more complex models to improve the strategies performance and open new odds to the application of machine learning in quantitative finance.

Índice general

Capítulo 1

Introducción

Con el paso del tiempo, el desarrollo de la tecnología ha supuesto un cambio en numerosos sectores industriales y comerciales, mejorando la eficiencia, precisión y la capacidad analítica a niveles inimaginables. Una de estas tecnologías es la inteligencia artificial (IA), la cual está jugando un papel muy importante en varios ámbitos, particularmente a través del aprendizaje automático. Este avance no ha sido relevante únicamente en áreas como la medicina e ingeniería, sino que también ha tenido un importante impacto en el ámbito financiero.

El sector financiero ha adoptado el aprendizaje automático como una herramienta esencial para el análisis de grandes volúmenes de datos y la toma de decisiones en tiempo real a gran velocidad. Esta adaptación ha permitido el desarrollo y perfeccionamiento del trading automático, donde los algoritmos desarrollados por el *machine learning* se encargan de la ejecución y precisión de las operaciones a una velocidad que supera las capacidades de cualquier ser humano.

1.1. Concepto de Aprendizaje Automático

El aprendizaje automático, o más comúnmente conocido como *machine learning* (ML), es una rama de la inteligencia artificial relativamente nueva, que combina el análisis estadístico con las ciencias de la computación. La información sobre el concepto de *machine learning* y sus aplicaciones en las finanzas cuantitativas ha sido obtenida y adaptada de [6].

El *machine learning* se enfoca en obtener sistemas que son capaces de aprender y mejorar, a partir de la experiencia y práctica, sin estar directamente programados para ello. Es decir, este sistema, aprende de los datos sin una especificación previa. Esta capacidad de aprendizaje, se logra mediante algoritmos que construyen modelos matemáticos basados en conjuntos de datos de entrada para hacer predicciones o clasificaciones, sin intervención humana directa.

La relevancia del ML ha crecido exponencialmente en las últimas décadas, debido a su utilidad en una amplia variedad de ámbitos sociales e industriales, incluyendo la salud, la automotriz, la financiera y más. En el contexto financiero, el ML permite analizar grandes volúmenes de datos a una velocidad y precisión que supera las capacidades humanas, lo cual es crítico en mercados que dependen de la rapidez y la exactitud de la información para tomar decisiones de inversión y operaciones en el *trading*.

Los algoritmos de ML son aplicados de varias formas a los problemas de finanzas cuantitativas.

Algunos ejemplos son:

- Predicción del movimiento del precio futuro de un activo;
- Predicción del movimiento de liquidez debido al movimiento de capital en grandes fondos;
- Obtención del sentimiento y pronóstico de analistas de valores;
- Clasificación/reconocimiento de imágenes para uso en señales de oferta/demanda de commodities (materias primas).

El ML se clasifica principalmente en tres categorías, según el método de aprendizaje y las características de los datos disponibles: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Los métodos en cada categoría difieren en cómo el algoritmo de *machine learning* es "recompensado" por realizar predicciones o clasificaciones correctas. A continuación se va a explicar cada uno de estos métodos de aprendizaje automático.

1.1.1. Aprendizaje Supervisado

El aprendizaje supervisado está caracterizado por el uso de algoritmos que involucran datos etiquetados, es decir, los datos están denotados con valores como categorías (clasificación supervisada) o respuestas numéricas (regresión supervisada). En este enfoque, el modelo es entrenado utilizando un conjunto de datos que contiene entradas junto con las salidas correctas. El objetivo principal es que, tras su experiencia de entrenamiento, el modelo sea capaz de predecir la salida correcta para nuevos datos que no haya visto antes. Para ello, aprende cuáles son los factores que influyen en la respuesta de un dato de entrada.

Este tipo de aprendizaje es utilizado en las finanzas cuantitativas. Por ejemplo, se puede usar un algoritmo de regresión supervisada en el que se utilizarían datos históricos de los mercados para predecir movimientos futuros de precios en un activo.

El proceso para aplicar el aprendizaje supervisado es:

1º Obtención y preparación de datos, que deben estar etiquetados para entrenar el modelo de manera efectiva.

2º Seleccionar un algoritmo adecuado y aplicarlo a nuestro conjunto de datos. Algunos ejemplos son la regresión lineal, los árboles de decisión o las redes neuronales para aprender las relaciones entre las variables de entrada y la salida deseada.

3º Evaluar el rendimiento del modelo utilizando un conjunto de datos no vistos por el algoritmo, es decir, que no se han usado durante el entrenamiento, y comprobar que el modelo pueda generalizar bien a nuevos datos.

1.1.2. Aprendizaje No Supervisado

El aprendizaje no supervisado es un tipo de *machine learning* donde los modelos son entrenados mediante datos que no están etiquetados. Es decir, a diferencia del aprendizaje supervisado, este método no utiliza respuestas válidas previamente conocidas. En cambio, el objetivo es explorar la estructura subyacente de los datos para identificar patrones o agrupaciones.

Este método es útil en las situaciones donde la etiquetación de los datos es imposible o se desconocen las categorías posibles. Su método canónico es el agrupamiento no supervisado, o conocido como *unsupervised clustering*, el cual intenta segmentar el conjunto de datos en subconjuntos que están relacionados de alguna forma.

En las finanzas cuantitativas, esta técnica sirve para agrupar activos financieros que tienden a tener comportamientos similares en términos de retorno y volatilidad. Más específicamente, si un activo sube de valor, otro activo del mismo subgrupo en principio subirá de valor también, y esto se obtiene sin necesidad de predefinir las categorías anteriormente.

Aunque el aprendizaje no supervisado puede proporcionar información realmente importante y útil, también presenta varios inconvenientes, principalmente en términos de cómo medir la similitud entre ejemplos en situaciones de mayor complejidad o cómo interpretar los resultados. Las agrupaciones encontradas no siempre son intuitivamente comprensibles, por lo que se tendrá que realizar una investigación más profunda para validar su utilidad.

1.1.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un tipo de *machine learning* que se basa en cómo unos agentes autónomos deben tomar decisiones dentro de un cierto entorno dinámico, con el objetivo de maximizar una cierta noción de recompensa acumulativa a lo largo del tiempo. A diferencia del aprendizaje supervisado y no supervisado, el aprendizaje por refuerzo, no se basa en datos etiquetados, es decir, no hay un conjunto de datos de entrada/salida, ni busca relaciones ocultas en los datos, sino que aprende a través de la interacción continua con el entorno para alcanzar un objetivo específico.

Estos algoritmos tienen diversas aplicaciones y son utilizados en varios sectores pero en las finanzas cuantitativas sirven para optimizar los portafolios de inversión.

A pesar de su potencial, el aprendizaje por refuerzo también presenta inconvenientes, como, por ejemplo, la necesidad de realizar una gran cantidad de interacciones con el entorno dinámico para aprender cuáles son las decisiones efectivas a tomar, además de la dificultad de balancear entre el uso de las estrategias efectivas ya conocidas y la exploración de nuevas estrategias. Otro inconveniente sería la variabilidad de la recompensa y la demora en obtenerla, lo cual complica el proceso de aprendizaje y la evaluación del rendimiento del agente.

Desafortunadamente, el aprendizaje por refuerzo no va a ser considerado y estudiado en más detalle en este trabajo. Además, es un tema que está en plena evolución y en constante avance. Se puede consultar más información en [1].

1.1.4. Aplicación del *machine learning* en las finanzas cuantitativas

Como se ha podido leer, el *machine learning* está muy presente en las finanzas cuantitativas, particularmente en el desarrollo de estrategias cuantitativas. Nosotros lo aplicaremos para los siguientes casos:

Pronóstico y predicción: Las técnicas de ML sirven para predecir el precio de un activo basado en los datos históricos de sus precios. Su precisión depende de la calidad y disponibilidad de los datos históricos, el mercado al que pertenece el activo, el período de los datos y el algoritmo que se use. Estas predicciones pueden ser usadas para obtener el precio en un tiempo posterior o un conjunto de futuros precios. Algunos ejemplos de estas predicciones incluyen la predicción del precio en un día del NASDAQ, predicción del spread en forex y predicción de liquidez en el order-book.

Precisión del modelo: Uno de los aspectos más importantes del *machine learning*, es determinar cuál de los modelos es el mejor para nuestro problema o el conjunto de datos en particular; esto se conoce como *selección de modelo*.

Para el aprendizaje supervisado, el problema aparece cuando se ajusta la complejidad del modelo y esto puede aumentar la probabilidad de estar sobreajustado. En esta situación, se podrá llegar a un gran resultado con los datos entrenados pero no se podrá generalizar a datos no vistos. Esto lleva a buscar un equilibrio entre flexibilidad y resultados efectivos, lo cual es conocido como *equilibrio sesgo-varianza*, que puede ser mitigado mediante un control en el proceso de ajuste y selección del modelo, y entre estos controles, destacaremos la validación cruzada o *cross-validation*.

1.2. Concepto de Trading Automático

El *trade* es un término económico que consiste en la compra y venta de bienes o servicios, es decir, es el intercambio de bienes o servicios entre participantes. El medio de intercambio más común es el dinero, pero existen otros medios como el intercambio de bienes o servicios, denominado trueque, o incluso el pago con una moneda virtual, por ejemplo, el Bitcoin, el cual ha obtenido mucha relevancia en los últimos años.

1.2.1. Mercado y Trading

La red que permite estos intercambios se denomina *mercado*, y se entiende como *trading* a la realización de la actividad de *trade*. Entonces, en los mercados financieros, el *trading* se refiere a la compra y venta de instrumentos financieros, como por ejemplo la compra de acciones de una empresa que cotiza en la bolsa americana.

Con la evolución de la tecnología a lo largo de los últimos años, se han desarrollado varias plataformas que brindan nuevas experiencias al usuario y una de ellas es la posibilidad de realizar operaciones de compra y venta de forma automática.

1.2.2. ¿Concepto de Trading Automático?

El trading automático, también llamado trading algorítmico, funciona de la forma en la que la toma de decisiones de compra y venta en los mercados financieros es realizada por un software que ejecuta varios algoritmos diseñados para identificar oportunidades de trading. Este método evita la intervención humana, lo cual es una ventaja en el tema de las emociones en las decisiones de trading, permitiendo así una respuesta rápida a las condiciones del mercado.

1.2.3. Tipos de Trading Automático

Los sistemas de trading automático utilizan modelos matemáticos y análisis cuantitativo para analizar datos de mercado y generar señales de trading. Estos sistemas pueden operar a diversas escalas de tiempo, desde microsegundos en el caso del *high-frequency trading (HFT)*, pasando por operaciones diarias denominadas *day trading*, hasta períodos más largos como el *swing trading*. El objetivo del uso de un sistema de trading automático, es conseguir un rendimiento mediante operaciones en las que se compra un valor para venderlo a un precio superior o viceversa, venderlo para comprarlo a un precio inferior.

1.2.4. Importancia del Backtesting

La implementación de estos sistemas implica una rigurosa fase de *backtesting*, donde las estrategias son probadas en datos históricos para evaluar su efectividad y robustez antes de ser usadas en un entorno de mercado real. Este proceso ayuda a identificar y ajustar los parámetros del modelo, minimizando así el riesgo y maximizando las probabilidades de éxito.

1.2.5. Ventajas e inconvenientes del *Trading* algorítmico

Como todo sistema, y en especial los automáticos, presenta ventajas e inconvenientes tal y como se comenta en [10] y [5] y de donde resumimos los puntos más importantes.

Respecto a las ventajas, algunas de ellas son que la ejecución de las órdenes son de forma rápida y precisa, mientras que un trader tiene limitaciones en la velocidad y la precisión, lo cual le puede costar una pérdida de oportunidades. También gracias a la inexistencia de las emociones, que pueden jugar una mala pasada a un trader, el algoritmo realiza las operaciones respetando una serie de reglas.

Respecto a las desventajas, es necesario tener un conocimiento básico sobre programación y estadística, además de que hay que ir monitoreando y realizando controles del sistema para adaptarse a las situaciones del mercado y hacer los cambios pertinentes.

1.3. Series de tiempo

El análisis de series de tiempo es fundamental en finanzas, ya que los precios de los activos y otros indicadores financieros son secuenciales. Una serie de tiempo es una secuencia de datos o puntos recogidos en intervalos de tiempo regulares. En el contexto financiero, estas series encierran o contienen la información fundamental para modelar y predecir el comportamiento de los precios de los activos.

1.3.1. Componentes

Desde un punto de vista clásico, las series de tiempo están formadas por tres componentes:

- **Tendencia**: Representa la dirección general del movimiento de los datos a largo plazo. La tendencia puede ser ascendente, descendente o constante, y puede cambiar debido a varios factores.
- **Estacionalidad**: Captura los patrones que se repiten en intervalos regulares, como días, semanas, meses o años. Estos patrones estacionales son importantes en finanzas, ya que muchos mercados tienen comportamientos cíclicos.
- **Ruido**: Variaciones aleatorias o irregulares que no siguen un patrón específico. El ruido es el componente más difícil de modelar ya que representa las fluctuaciones impredecibles en los datos.

1.3.2. Modelos para series de tiempo

Existen varios modelos y métodos para analizar y predecir series de tiempo. Entre los más utilizados están los modelos *ARIMA*, *GARCH* y *ML*, se puede encontrar más información en [7]. Hemos elegido los modelos de machine learning para modelar las series de tiempo, ya que son capaces de capturar patrones complejos y no lineales en los datos.

1.4. Modelos a estudiar

A continuación introducimos los dos modelos de *machine learning* que se han seleccionado para su estudio por su relevancia en el ámbito del trading automático: los métodos basados en árboles y las máquinas de soporte vectorial (SVM).

- **Métodos basados en árboles:** Los métodos basados en árboles, como los bosques aleatorios (*random forests*), funcionan mediante la división del espacio de características en regiones más simples; facilitando así la toma de decisiones basada en la observación de las características de los datos. Para el trading automático, los árboles los utilizamos para predecir los retornos del precio.
- **Máquinas de soporte vectorial (SVM):** Las SVM son otro tipo de modelos que usamos para clasificar los datos mediante la construcción del mejor hiperplano, que separa las clases en el espacio de características. En el contexto del trading, las SVM las utilizamos para clasificar tendencias de mercado.

1.4.1. Motivación

Mi elección de centrarme en los métodos basados en árboles y las SVM está motivada por su rendimiento en contextos complejos y no lineales, como los que se encuentran en los mercados financieros. Estas herramientas estadísticas avanzadas, pueden encontrar patrones ocultos en datos muy diversos y caóticos como es en los mercados financieros, lo que es importante para desarrollar estrategias de trading algorítmico efectivas. Además, mi interés en estas técnicas es importante, ya que combina rigor matemático con una aplicación directa en los mercados financieros, lo cual es un área de gran pasión para mí. Concluiremos este trabajo con un caso práctico donde entrenaremos estos modelos utilizando un conjunto de datos extraídos específicamente de varios activos en un mercado financiero, como los precios históricos del Banco Santander, Amazon, Indra, etc. El objetivo será desarrollar una estrategia de trading automático que nos permitirá realizar pruebas y comparar el rendimiento de nuestra estrategia con otras.

Capítulo 2

Modelos de *Machine Learning* en Trading Automático

En este capítulo se explorarán los modelos de *machine learning* que se aplicarán al trading automático. Se empezará con una breve introducción al aprendizaje supervisado, estableciendo sus bases, ya que los dos modelos que se van a estudiar pertenecen a este tipo. Después, se explicarán con más detalle los dos modelos: los métodos basados en árboles y las máquinas de soporte vectorial (SVM). La mayoría de la información de este capítulo ha sido adaptada de [6].

2.1. Introducción

El aprendizaje supervisado es el tipo de *machine learning* más utilizado, ya que se realizan dos tareas muy generales: clasificación y regresión.

En los problemas de clasificación, se trata de estimar la pertenencia de un conjunto de características a un grupo categórico. Por ejemplo, se predice si el precio de un activo subirá o bajará al día siguiente, basado en el precio de los N días anteriores.

En los problemas de regresión, se estima el valor real de la respuesta en base a un conjunto de características. Por ejemplo, se predice el valor de un activo en el próximo día, basado en el histórico de su precio, no solo si subirá o bajará.

2.1.1. Marco Matemático

En el enfoque de regresión, el problema se formula en términos de una función de estimación. La "verdadera" respuesta de la variable y, que puede ser categórica o de valor real, se modela mediante una función $f = f(\mathbf{x})$. Los predictores o características $\mathbf{x} \in \mathbb{R}^p$ son un vector de p-dimensiones que contiene las mediciones de las características. Además, se añade una componente ε a f que recoge información no disponible en $\mathbf{x}$. El termino ε generalmente se asume que sigue una distribución normal con media cero y varianza σ^2:

$$y = f(\mathbf{x}) + \varepsilon$$

El objetivo del *machine learning* es realizar una estimación de y, denotada como $\hat{y}$, intentando encontrar una función $\hat{f}$ que "mejor" aproxime a f. Una vez encontrada $\hat{f}$, se puede estimar fácilmente el valor de $\hat{y}$ a partir de un vector $\mathbf{x}_{test}$.

El enfoque de clasificación plantea el problema como la estimación de la forma de una distribución de probabilidad, la cual se da como $p(y \mid \mathbf{x})$. Esto se llama *distribución de probabilidad condicional* y representa la probabilidad de que y tome cualquier valor (o categoría) dado los valores de las características $\mathbf{x}$. El beneficio de este enfoque es que se pueden asignar probabilidades a los distintos valores de y, lo cual facilita la elección entre estos valores. Sin embargo, en las finanzas cuantitativas, un error puede

tener grandes consecuencias, por lo que se utilizan valores umbrales para asegurar que la probabilidad asignada a un valor sea muy alta, mostrando así confianza en la elección.

A continuación, definimos las funciones que utilizarán los modelos para encontrar el mejor predictor con un conjunto específico. Definimos una función llamada **función de pérdida** con argumentos el valor de la verdadera respuesta y y su valor estimado del modelo $\hat{y}$, dada por $\frac{1}{N}\sum_{i=1}^{N} L(y,\hat{y})$.

En clasificación, el modelo de *pérdida* más utilizado es el de **0-1 pérdida** y en este caso $L(y,\hat{y}) = I(y \neq \hat{y})$

En regresión, el modelo de *pérdida* más común es $L(y,\hat{y}) = (y_i - \hat{y}_i)^2$ y entonces tendremos lo que se conoce como el error cuadrático medio (ECM):

$$\text{ECM} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Aunque los dos enfoques explicados arriba parezcan distintos, ambos intentan llevar a cabo la misma tarea: estimar el mejor valor de y.

Clasificación En clasificación, y es un valor categórico que puede tomar valores de un conjunto finito de posibles elecciones K. Para estimar el valor de y se utiliza:

$$\hat{y} = \hat{f}(x) = \arg\max_{k \in K} p(y = k \mid x)$$

Regresión En regresión, $y \in \mathbb{R}$, es decir, es un valor real. Para estimar el valor de y se utiliza:

$$\hat{y} = \hat{f}(x) = \widehat{\mathbb{E}[y \mid \mathbf{x}]}$$

2.1.2. Entrenamiento

Por último, definimos como podemos "entrenar" un modelo con un conjunto específico de datos. Para ello el objetivo es minimizar una de las funciones de pérdida que hemos definido anteriormente, segun si estamos en clasificación o regresión.

Un valor mínimo de la *función de pérdida* establece que los errores entre los valores verdaderos y estimados no son muy grandes, lo cual lleva a pensar que el modelo funcionará de forma similar cuando lo expongamos a datos no vistos en el entrenamiento. Aunque aparece un problema conocido como **equilibrio sesgo-varianza**, en el que el problema principal es que si se reduce demasiado la *función de pérdida*, entonces la generalización del funcionamiento del modelo será mucho peor y, por lo tanto, el modelo estará sobreajustado.

2.2. Modelos basados en árboles

En esta sección vamos a tratar una técnica del aprendizaje supervisado llamada **Árboles de Decisión** que aprende a construir reglas de decisión de las características que pretenden predecir el valor de la respuesta.

2.2.1. Introducción

Los modelos de árboles de decisión trabajan mediante reglas de decision fundamentalmente binarias que dividen el espacio de características en un número de regiones simples cuyos lados son paralelos a los ejes. Para la predicción de una observación, se usa la media o moda de la región a la que pertenece la observación en el espacio de las características.

Su principal beneficio es que producen un conjunto de reglas *si-entonces-sino* que son interpretables. En cambio, su principal desventaja es que, a menudo, la técnica de decisión de árboles no es tan efectiva como otras técnicas del aprendizaje supervisado, como las máquinas de vectores de soporte. Sin embargo, se vuelve competitiva cuando se realiza en un entorno de consenso, como con el **Bootstrap Aggregation ("Bagging")**, **Random Forest** o **Boosting**. Desarrollaremos mas adelante estos métodos.

2.2.2. Descripción Matemática

La forma matemática de expresar cómo funcionan los modelos de árboles de decisión de manera probabilística mediante funciones base adaptativas se da como:

$$f(\mathbf{x}) = \mathbb{E}(y \mid \mathbf{x}) = \sum_{m=1}^{M} w_m \phi(\mathbf{x}; \mathbf{v_m})$$

Donde w_m es la respuesta media en una region R_m, $\phi(\mathbf{x}; \mathbf{v_m})$ es la función base parametrizada por v_m, que devuelve un 1 si $\mathbf{x} \in R_m$ y un 0 en el caso contrario y $\mathbf{v_m}$ representa los valores umbrales en los que se divide el espacio de caracteristicas. Estas divisiones definen como el espacio de caracteristicas en $\mathbb{R}^p$ esta dividido en M regiones separadas.

2.2.3. Arboles de Decision para Regresion

Se va a mostrar un ejemplo de un problema de regresion con dos variables caracteristicas (X_1, X_2) y una respuesta numerica y. Como se trabaja en $\mathbb{R}^2$, facilita la visualizacion de la partición llevada a cabo. El árbol resultante es el mostrado en la figura 2.1:

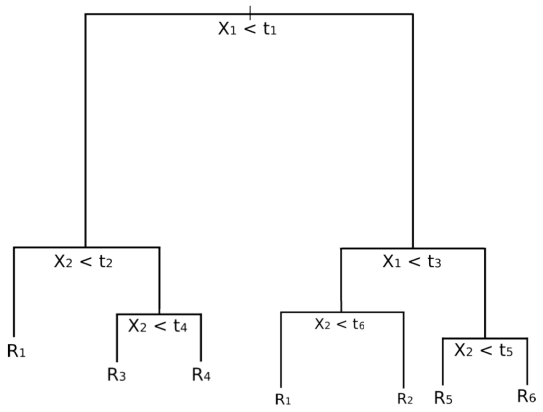


Figura 2.1: Árbol de decisión con seis regiones separadas

Se observa como en la primera "rama" se hace la pregunta de si la caracteristica X_1 es menor que un valor umbral t_1, en caso de afirmativo, se considera la siguiente rama de la izquierda y se pregunta si la caracteristica X_2 es menor que otro valor umbral t_2. Si es así, terminamos en la region R_1; de lo contrario se pregunta si X_2 es menor que otro valor umbral t_4, si se cumple se termina en la region R_3 y sino en la region R_4.

Similarmente, si el X_1 inicial es mayor o igual que el valor t_2, se considera la rama derecha y se produce una subdivisión adicional en las capas subsiguientes. Así de forma recurrente hasta que queden definidas las regiones R_2, R_5 y R_6. Como se puede observar en la imagen, se pueden repetir las regiones en los finales de las "ramas". De esta manera, los valores umbrales t_1 a t_6 son los que determinan cómo se divide el espacio de las características en regiones.

En la Figura 2.2 se muestra la partición de $\mathbb{R}^2$ después de haber sido "entrenado" con el conjunto hipotético de datos del ejemplo. Se puede observar cómo cada división del espacio es paralela a un eje de las características.

La división paralela a los ejes permite la generalización a dimensiones mayores. Así es, que para un subconjunto de $\mathbb{R}^p$, se divide el espacio en M regiones, denotadas por R_m cada una, y todas ellas son p-dimensionales "hiperbloques".

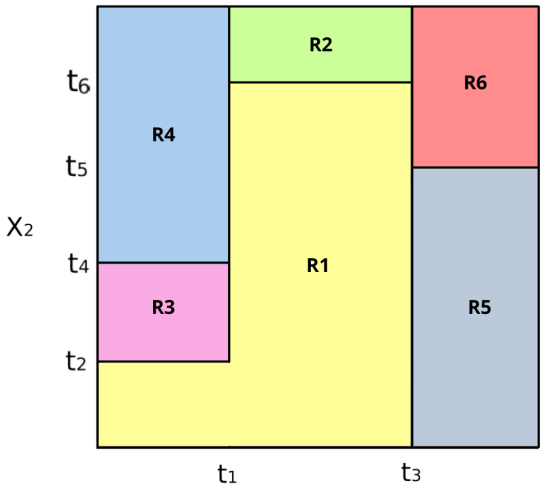


Figura 2.2: Árbol de decisión con seis regiones separadas

Con esto se ha visto cómo un árbol realiza predicciones una vez que ya está creado. Pero no se ha explicado cómo un árbol es creado, es decir, cómo "crece" o "entrena". A continuación se va a describir el algoritmo que se sigue durante el entrenamiento para obtener estos resultados.

Creación de un árbol de regresión

El proceso de creación de un árbol se realiza de forma recursiva:

- Se empieza con un nodo raíz "padre" que se dividirá en dos nodos "hijos". Esta división se produce buscando el valor óptimo de una característica que mayor similitud produzca entre los nodos resultantes. El proceso se repetiría recursivamente para cada uno de los nuevos nodos.
- Un nodo se convierte en hoja, y por lo tanto acaba el proceso recursivo para este nodo, cuando posee una profundidad determinada o cuando no tenga las suficientes muestras para dividirse en otros dos nodos. Ambos criterios se definen antes de comenzar con la iteración.

A continuación, se definirá de forma algorítmica cómo se produce la partición, para ello se usará una técnica conocida como **recursive binary splitting**.

Recursive binary splitting

El objetivo va a ser minimizar algún tipo de criterio de error, en este caso, al tratarse de un modelo predictivo, minimizaremos la suma residual de cuadrados (RSS). El RSS para los árboles de decisión en el que un espacio característico es dividido en M regiones es la siguiente expresión:

$$\text{RSS} = \sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2$$

Donde el primer sumatorio recorre todas las regiones y el segundo sumatorio recorre todas las observaciones de una región, donde realiza la diferencia al cuadrado (para penalizar los errores más grandes) del valor real de la respuesta y_i y el valor medio de la respuesta de la región $\hat{y}_{R_m}$.

Pero este método tiene un inconveniente principal: es demasiado costoso computacionalmente, ya que hay que considerar todas las posibles particiones del espacio característico en M regiones. De aquí es donde aparece el *recursive binary splitting* (RBS).

RBS comienza en la parte superior del árbol y lo divide en dos ramas, es decir, divide el espacio característico en dos regiones. Realiza esta partición del árbol múltiples veces y se queda con la división del espacio de características que minimiza el RSS. En este momento, el árbol crea una nueva rama en

una partición particular y vuelve a llevar a cabo el mismo procedimiento, es decir, elige la partición en la que el RSS tenga un mejor rendimiento.

Ahora se presenta el problema de saber cuándo se debe terminar este método. Hay varias formas de parar, como por ejemplo, incluir una profundidad máxima del árbol, asegurar que haya un mínimo de observaciones en cada región y/o que cada región sea lo suficientemente homogénea hasta que el árbol esté balanceado.

Pero siempre tenemos que tener cierta atención con no sobreajustar el modelo, lo que motiva el concepto de "poda" del árbol, el cual no va a ser desarrollado en este trabajo pero se puede encontrar más información en [8] y [9].

2.2.4. Árboles de decisión para la clasificación

Como se había especificado anteriormente, este modelo nos permite también hacer predicciones de valores categóricos y no únicamente de respuestas de valores reales.

Ahora, para realizar la predicción de un valor categórico, se usa la moda de la región a la que pertenece la observación, en lugar del valor medio de la región como en el caso de regresión. Entonces tomamos el valor de la clase más repetido de la región y se lo asignamos como la respuesta de la observación.

Sin embargo, en este caso, ahora no se puede aplicar el mismo criterio (RSS) anterior, ya que no se puede aplicar en el contexto categórico y por eso tendremos que considerar otros tipos de criterios:

Tasa de aciertos

En vez de calcular qué tan lejos estaba la respuesta del valor medio en una región, se puede definir una Tasa de aciertos como la fracción de observaciones en una región que no pertenecen a la clase más frecuente. El error se define como:

$$E = 1 - \arg\max_{c}(\hat{\pi}_{mc})$$

Donde $\hat{\pi}_{mc}$ representa la fracción de datos de la región R_m que pertenecen a la clase c.

Índice de Gini

El índice de Gini es una alternativa de métrica de error en la que se mide cuán "pura" es una región. La "pureza" se refiere a cuántos de los datos de entrenamiento en una región pertenecen a una sola clase. Se define como:

$$G = \sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c)$$

Entropía cruzada/desviación

Una alternativa muy similar al índice de Gini es la entropía cruzada o desviación:

$$D = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c$$

Cualquiera de los tres métodos es útil, sin embargo, la tasa de aciertos es anecdóticamente menos utilizada que los otros dos métodos para maximizar la efectividad de predicción.

2.2.5. Ventajas e Inconvenientes

Ventajas

- La construcción y evaluación de los modelos de árboles de decisión es rápida, lo que los hace adecuados para aplicaciones en tiempo real.

- Los modelos pueden ser entrenados con un conjunto de datos que tiene características categóricas y continuas.
- Los árboles de decisión pueden ser visualizados de manera sencilla, lo que facilita su interpretación ya que generan reglas de si-entonces-sino.
- Los árboles de decisión no requieren que los datos sigan una distribución específica, lo que los hace flexibles para diferentes tipos de datos.

Inconvenientes

- Los árboles de decisión pueden tender a sobreajustar los datos de entrenamiento, especialmente si no se "podan" adecuadamente.
- Los modelos de árboles de decisión sufren de inestabilidad, lo que significa que son muy sensibles a pequeños cambios en el espacio característico. Hablando de *equilibrio sesgo-varianza*, son estimadores de alta varianza.
- La eficacia de predicción de estos modelos no tiene el mismo rendimiento que otros modelos de *machine learning*.

Respecto a la última desventaja, como habiamos denotado en la introducción de esta sección, también pueden ser muy competitivos a la hora de predecir cuando son usados de una manera combinada entre ellos. A continuación definiremos brevemente varios métodos de consenso como **Bootstrap aggregation ("Bagging")**, **Random Forest** o **Boosting**.

2.2.6. Métodos de consenso

A continuación vamos a explicar los métodos de consenso que se pueden aplicar en los árboles de decisión para mejorar el rendimiento predictivo del modelo combinado. Aunque antes tenemos que definir el *Bootstrap*, una técnica de la estadística frecuentista.

The Bootstrap

El *Bootstrapping*[4] es una técnica de remuestreo, es decir, un muestreo aleatorio con reemplazo de un conjunto de datos. Se utiliza en las finanzas cuantitativas, ya que permite generar nuevas muestras sin tener que obtener nuevos datos para el entrenamiento, lo cual es útil para cuando la información es limitada como el precio de una acción durante la historia.

El proceso es muestrear repetidamente datos del conjunto de datos inicial con reemplazo y así obtener múltiples conjuntos de entrenamiento separados. Esto logrará una disminución de la varianza de las predicciones de los métodos de aprendizaje automático y así logrará una mejora en el rendimiento.

Las dos siguientes técnicas que describiremos a continuación, *Bootstrap Aggregation* y *Random Forest*, utilizan la técnica del *Bootstrapping*.

Bootstrap Aggregation

Como se había nombrado anteriormente, los árboles de decisión son estimadores de alta varianza, lo que implica que un pequeño cambio en el conjunto de datos del entrenamiento, como añadir un número de datos, generará un árbol completamente distinto al anterior, lo que cambiará el rendimiento predictivo de este.

La forma de minimizar este efecto es aplicar la técnica de *Bootstrap Aggregation* o *Bagging*. La idea es combinar varios árboles de decisión que han sido entrenados con distintas muestras *bootstrap* y promediar sus predicciones para así reducir la varianza de estas predicciones.

El motivo de que esto funcione es el hecho de que si tenemos N datos independientes e identicamente distribuidos $X_1, \ldots, X_n$ con varianza σ^2, entonces la varianza de la media $\overline{X}$ es σ^2/N.

El proceso de *Bagging* es crear B muestras *bootstrap* del conjunto de entrenamiento, se entrena cada una independientemente para crear B estimadores separados $\hat{f}^b(x)$, se promedian todos estos estimadores y así obtenemos un nuevo estimador de menor varianza, $\hat{f}_{avg}$:

$$\hat{f}_{\text{avg}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x)$$

Este método se puede aplicar a los árboles de decisión ya que son estimadores de alta varianza. Se crearán cientos de árboles muy profundos (no "podados") en muestras *bootstrap* del conjunto de datos de entrenamiento, se combinarán de la forma descrita anteriormente y así reducirán la varianza del estimador general.

Random Forest

El método *Random Forest*[2] es muy similar al *Bagging*; su diferencia es que utiliza una técnica denominada *bagging* característico. Esta técnica reduce la correlación entre los árboles de decisión y así aumenta su rendimiento predictivo.

El *bagging característico* selecciona aleatoriamente un subconjunto de dimensión p de las características en cada división en el crecimiento de cada árbol. El objetivo de esta técnica es evitar las características predictivas muy fuertes que lleven a divisiones similares en los árboles. Si una característica es fuerte en el valor de la respuesta, se seleccionará en la mayoría de los árboles, así se obtendrán árboles correlacionados. Los *Random Forest* evitan esto al dejar características fuertes sin incluir en varios de los árboles generados.

Cabe destacar que si se seleccionan las p características en *Random Forest*, esto sería tal cual el proceso de *Bagging*. Por eso, una regla que se utiliza en *Random Forest* es usar $\lfloor\sqrt{p}\rfloor$ características en cada división.

En este método tenemos que definir tres hiperparámetros que utilizaremos para entrenar nuestro modelo: el número de árboles N que utilizaremos, la profundidad k de los árboles de decisión y el número mínimo de observaciones M para que un nodo del árbol se pueda dividir en dos nodos "hijos". La elección de estos hiperparámetros se realiza mediante la validación cruzada para saber cómo ajustarlos.

Utilizaremos este método para nuestro análisis predictivo del precio de una acción en el siguiente capítulo.

Boosting

El último método es el *Boosting*, que a diferencia del *Bagging*, no utiliza la técnica de *Bootstrap*. En este caso, los modelos se forman de manera iterativa y secuencial. Se crea el modelo i y luego se creará el modelo $i+1$ dependiendo de los errores del modelo anterior.

La idea básica es iterar modelos de *machine learning* débiles, se entienden débiles como poca capacidad predictiva, en un conjunto de datos de entrenamiento que se actualiza continuamente y unirlos para así formar un modelo fuerte.

El algoritmo básico de *Boosting* comienza con un estimador inicial $\hat{f}(x) = 0$ y los residuos calculados como $r_i = y_i$ para cada observación i. Luego, se inicia un proceso iterativo para $b = 1,\ldots,B$, donde B es el número total de árboles. En cada iteración, se construye un árbol $\hat{f}^b$ con k divisiones y se actualizan los residuos mediante la fórmula $r_i^{b+1} = r_i^b - \lambda_b\hat{f}^b(x_i)$. Finalmente, el modelo resultante se compone de la suma de todos los árboles: $\hat{f}(x) = \sum_{b=1}^{B}\lambda_b\hat{f}^b(x)$. Donde λ_b es la tasa de aprendizaje que controla la contribución de cada modelo base al modelo final, afectando así a la velocidad de aprendizaje.

Es importante notar que cada árbol en el siguiente paso se ajusta a los residuos de los datos. Por lo tanto, en cada iteración, se mejora el modelo al optimizar su rendimiento en las regiones del espacio de características donde el rendimiento es deficiente.

2.3. Máquinas de vectores de soporte

En esta sección se va a tratar sobre las máquinas de vectores de soporte, una técnica muy importante en el aprendizaje supervisado, sobre todo para la clasificación. Se puede consultar más información en [3].

Se introducirán varios conceptos más adelante como clasificadores de vectores de soporte, clasificadores de margen máximo y máquinas de vectores de soporte.

2.3.1. Introducción

El objetivo de este algoritmo es categorizar nuevos datos que no hayan sido vistos previamente por el algoritmo en dos grupos separados basado en sus propiedades y conociendo ya datos que han sido clasificados, esto se conoce como la clasificación supervisada binaria.

Un ejemplo es la clasificación de un nuevo conjunto de reseñas de una película en dos grupos de sentimiento, reseña positiva o reseña negativa. Las máquinas de vectores de soporte son realmente útiles en estas situaciones.

De una forma matemática, las máquinas de vectores de soporte construyen un hiperplano separador lineal en espacios vectoriales de dimensión finita. En el conjunto de datos, cada observación es de la forma $(\mathbf{x}, y)$ con $\mathbf{x} = (x_1, \ldots, x_p)$ donde los x_j son las características e y es la clasificación, que toma los valores +1 o -1.

En muchos ejemplos, el conjunto de datos no es linealmente separable y los datos se superponen entre sí. Para abordar este problema se realiza una transformación del espacio característico original en otro donde la separación de los grupos se vea más clara.

2.3.2. Hiperplanos separadores lineales

Se comenzará introduciendo la idea base que formará los cimientos de las máquinas de vectores de soporte, este es el hiperplano lineal separador. Este hiperplano separador divide el espacio característico en dos regiones. Un hiperplano tiene, en general, dimensión $p-1$ en un espacio característico de dimensión p.

A continuación se va a definir matemáticamente el hiperplano afín. Sea $\mathbf{x} = (x_1, \ldots, x_p) \in \mathbb{R}^p$, entonces el hiperplano queda determinado por la siguiente ecuación:

$$b_0 + b_1 x_1 + \cdots + b_p x_p = 0$$

con $b_i \in \mathbb{R}$ para todo $i > 0$ y $b_0 \neq 0$, así el hiperplano no pasará por el origen. Equivalentemente, con notación de producto interno:

$$\mathbf{b} \cdot \mathbf{x} + b_0 = 0$$

donde $\mathbf{b} = (b_1, b_2, \ldots, b_p)$ es el vector de coeficientes.

Si tenemos un elemento $\mathbf{x} \in \mathbb{R}^p$, para saber en qué parte del espacio se encuentra respecto al hiperplano, se sustituye el punto en la ecuación. Si vemos que se cumple la relación, entonces el punto pertenece al hiperplano. El punto estará en la región superior del hiperplano si:

$$\mathbf{b} \cdot \mathbf{x} + b_0 > 0$$

y en la región inferior del hiperplano si:

$$\mathbf{b} \cdot \mathbf{x} + b_0 < 0$$

Esto es útil para poder saber en qué región pertenecerá un nuevo dato, mediante el cálculo del signo de la ecuación anterior.

2.3.3. Clasificación

Se vuelve a considerar el ejemplo anterior de las reseñas de una película de cine. Si consideramos un conjunto de n observaciones que usaremos para el entrenamiento, $\mathbf{x_i} \in \mathbb{R}^p$, donde es p-dimensional ya que se han seleccionado p palabras que serán las características. Para cada $\mathbf{x_i}$ tiene asociado una categoría $y_i \in \{-1, 1\}$, el -1 podría ser reseña negativa y entonces el 1 reseña positiva, formando el par $(\mathbf{x_i}, y_i)$.

Ahora bien, el objetivo sería encontrar el clasificador perfecto basado en el conjunto de entrenamiento que separe perfectamente las dos clases y así permita clasificar futuras observaciones.

El clasificador podría ser el hiperplano anterior descrito, pero como se ve en la figura 2.3 puede haber varios. Para resolver este inconveniente, más adelante se describirá la condición para elegir el más "óptimo".

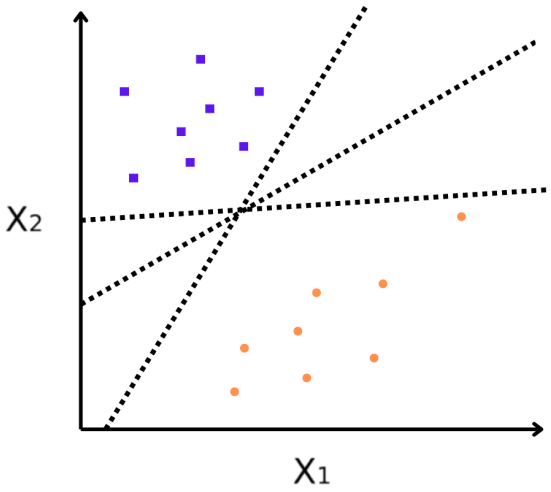


Figura 2.3: Varios posibles hiperplanos separadores

De esta forma, si $y_i = 1$ entonces $\mathbf{b} \cdot \mathbf{x_i} + b_0 > 0$ y si $y_i = -1$ entonces $\mathbf{b} \cdot \mathbf{x_i} + b_0 < 0$. Por lo que, para cada observación del entrenamiento que quede por encima o por debajo del hiperplano quedará etiquetada con +1 o -1.

Podemos formalizarlo considerando la función $f(\mathbf{x})$ que, con una observación $\mathbf{x}^* = (x_1^*, \ldots, x_p^*)$, se define como: $f(\mathbf{x}^*) = \mathbf{b} \cdot \mathbf{x}^* + b_0$. Por lo tanto, como antes, si $f(\mathbf{x}^*) > 0$ entonces $y^* = +1$, y si $f(\mathbf{x}^*) < 0$ entonces $y^* = -1$.

2.3.4. Obtención del clasificador

Hasta ahora, hemos definido el hiperplano separador con su ecuación correspondiente. Pero como hemos podido ver en la figura 2.3, no es único, por eso ahora vamos a ver cómo obtener el hiperplano óptimo, el cual denominamos como el hiperplano de margen duro máximo.

Para lograrlo, primero calculamos las distancias perpendiculares desde cada observación $\mathbf{x}_i$ para un hiperplano separador. La distancia perpendicular más pequeña de todas ellas se llama **margen**.

Entonces, el hiperplano de margen duro máximo es el hiperplano que tiene el margen más grande para garantizar que es la distancia más alejada a un dato de entrenamiento. Además, denominamos a $f(\mathbf{x}^*)$ del correspondiente hiperplano como el clasificador de margen duro máximo.

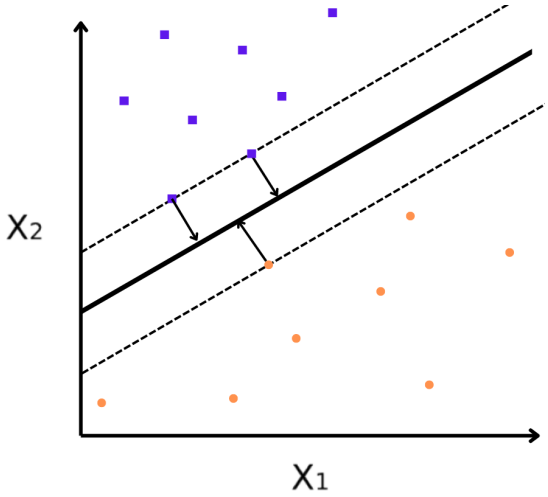


Figura 2.4: Hiperplano de margen duro máximo

Pero todavía nos falta mostrar el algoritmo para encontrar los valores b_j que determinan este clasificador de margen duro máximo para poder usarlo con cualquier observación.

Cabe destacar que la idea clave detrás del clasificador de margen duro máximo es que la ubicación del hiperplano de margen duro máximo depende de lo que se denomina vector de soporte, que son los vectores que van de las observaciones que se encuentran en el margen al hiperplano perpendicularmente.

Un inconveniente del clasificador es que el hiperplano va a ser muy sensible a las ubicaciones de los vectores de soporte, cualquier cambio en estos puede derivar en un hiperplano completamente distinto. Pero también trae una ventaja computacionalmente, y es que solo se tienen que almacenar los vectores de soporte una vez entrenado el modelo.

2.3.5. Construcción del clasificador de margen máximo

El método para determinar el hiperplano de margen duro máximo para el clasificador de margen duro máximo es el siguiente: dado n observaciones que son utilizadas para el entrenamiento $\mathbf{x}_1,\ldots,\mathbf{x}_n \in \mathbb{R}^p$ y sus n etiquetas $y_1,\ldots,y_n \in \{-1,1\}$, el clasificador de margen duro máximo es la solución del siguiente problema de optimización:

Maximizar $M \in \mathbb{R}$, probando valores para $b_1,\ldots,b_n \in \mathbb{R}$ tal que:

$$\sum_{j=1}^{p} b_j^2 = \mathbf{b}\cdot\mathbf{b} = 1$$

$$y_i(\mathbf{b}\cdot\mathbf{x}+b_0) \geq M, \quad \forall i = 1,\ldots,n$$

El valor M representa a qué distancia tendrá que estar cada observación del hiperplano. Este proceso también establece que cada observación este en el lado correcto.

Aunque a veces nos encontramos en situaciones en las que no se puede construir un hiperplano separador de margen duro máximo mediante la optimización del proceso anterior, por eso se utiliza lo que se llama **margen suave**, que motiva el concepto de **clasificador de vectores de soporte**.

2.3.6. Clasificador de vectores de soporte

Como hemos comentado anteriormente, uno de los problemas del clasificador de margen duro máximo es que es muy sensible a cualquier perturbación en el conjunto de datos del entrenamiento. Como se puede observar en la figura 2.5, en la imagen de la izquierda tenemos el hiperplano de margen duro máximo, pero en la imagen de la derecha, al añadir un dato, cambia el hiperplano completamente.

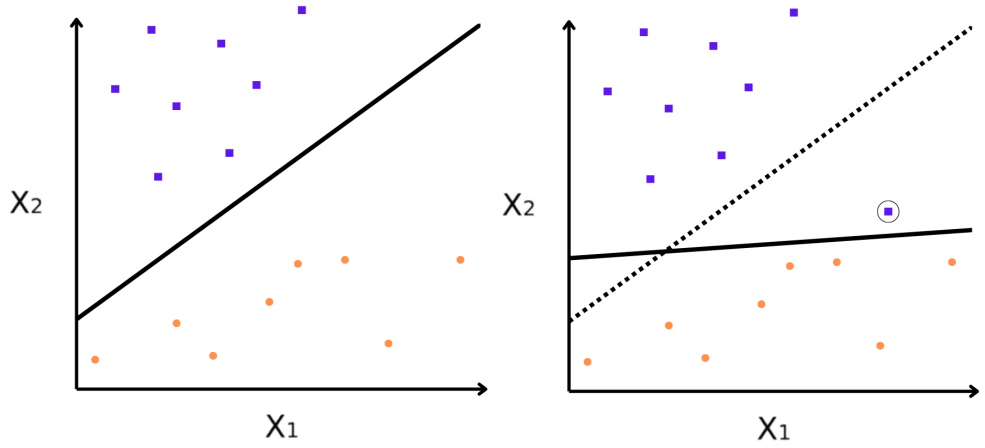


Figura 2.5: Cambio del hiperplano de margen duro máximo al añadir un punto

Para solucionarlo, consideramos un clasificador que no separe perfectamente las dos clases pero que se comporte mejor con la perturbación de los datos, aunque con un coste en la mala clasificación de algunas observaciones del entrenamiento.

Así es como funciona el clasificador de vectores de soporte o clasificador de margen suave, es decir, este clasificador permite que haya observaciones que estén en el lado incorrecto o en el lado correcto pero que no respeten el margen.

A continuación, vamos a formalizar matemáticamente cómo se obtiene este nuevo clasificador. Para ello, primero tenemos que introducir nuevos parámetros: n valores ε_i, conocidos como *variables de holgura*, y un parámetro $C \in \mathbb{R}^+ \cup \{0\}$, llamado *coste*.

Entonces, el problema de optimización ahora sería:

Maximizar $M \in \mathbb{R}$, variando valores para $b_1, \ldots, b_n, \varepsilon_1, \ldots, \varepsilon_n \in \mathbb{R}$ tal que:

$$\sum_{j=1}^{p} b_j^2 = \mathbf{b} \cdot \mathbf{b} = 1$$

$$y_i(\mathbf{b} \cdot \mathbf{x} + b_0) \geq M(1 - \varepsilon_i), \quad \forall i = 1, \ldots, n$$

$$\varepsilon_i \geq 0, \quad \sum_{i=1}^{n} \varepsilon_i \leq C$$

Los ε_i nos dicen dónde está la observación i respecto al margen y al hiperplano. Para $\varepsilon_i = 0$, $\mathbf{x}_i$ está en el lado correcto del margen, mientras que para $\varepsilon_i > 0$, $\mathbf{x}_i$ está en el lado correcto del margen. Por último, si $\varepsilon_i > 1$, tenemos que $\mathbf{x}_i$ está en el lado incorrecto del hiperplano.

El valor de C nos dice cuántos ε_i pueden violar el margen. Así, para $C = 0$ implica que $\varepsilon_i = 0 \quad \forall i$ y estaríamos en el caso del clasificador de margen duro máximo, y para $C > 0$ se tiene que como máximo C observaciones pueden violar el hiperplano.

Tendremos que tener cuidado con la elección del valor C, ya que es el parámetro que controla el equilibrio entre sesgo y varianza. Un valor pequeño de C significará una alta varianza y un sesgo pequeño, lo que provocará un sobreajuste en el modelo y una poca capacidad predictiva para datos no vistos en el entrenamiento. Para un valor grande de C, tendrá un alto sesgo y poca varianza, por lo tanto no se ajustará a la complejidad del modelo.

Uno de los problemas que se nos presenta, es que funciona muy bien para cuando los datos están linealmente o casi linealmente separados, pero cuando los datos no son separables de forma lineal, no funciona. Para eso extendemos el concepto del clasificador de vectores de soporte a máquinas de vectores de soporte.

2.3.7. Máquinas de vectores de soporte

La extensión de los clasificadores de vectores de soporte a las máquinas de vectores de soporte permite calcular los límites de decisión no lineales para poder clasificar un conjunto de datos que no son lineales.

Por ejemplo, en la figura 2.6 se puede observar cómo el clasificador de vectores de soporte realiza una separación lineal que proporcionará un rendimiento malo cuando lo sometamos a un conjunto de datos no vistos.

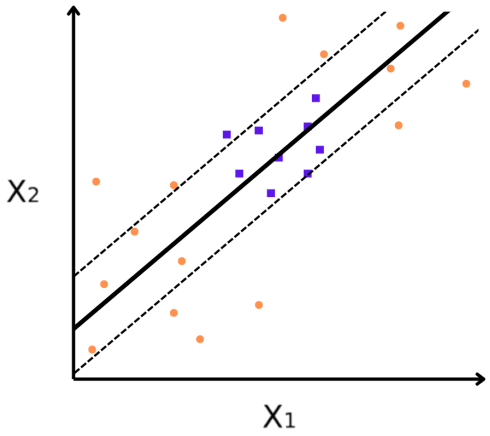


Figura 2.6: Separación lineal del clasificador de los vectores de soporte en un conjunto de datos no lineal

Las máquinas de vectores de soporte funcionan gracias a un truco común cuando se consideran situaciones no lineales. Este truco es aplicar una transformación a un conjunto de p características $x_1,\ldots,x_p$ y convertirlo, por ejemplo, en un conjunto de $2p$ características $x_1,x_1^2,\ldots,x_p,x_p^2$.

Mientras que el límite de decisión es lineal en el nuevo espacio característico de dimensión $2p$, en el anterior espacio característico p-dimensional será no lineal.

Básicamente, las máquinas de vectores de soporte son una extensión de los clasificadores de vectores de soporte que resulta de la expansión del espacio de características mediante funciones conocidas como núcleos, por eso a esta transformación se le llama “kernel tricks”. Pero antes de entender los núcleos, explicaremos aspectos de la solución del problema de optimización anterior.

Para calcular la solución del problema de optimización, el algoritmo solo necesita los productos internos entre las observaciones y no las observaciones en sí mismas.

El producto interno para dos vectores (dos observaciones) de dimensión p está definido como:

$$\langle \mathbf{x}_i, \mathbf{x}_k \rangle = \sum_{j=1}^{p} x_{ij} x_{kj}$$

No lo demostraremos, pero se puede comprobar que el clasificador de vector de soporte puede escribirse como una combinación de productos internos:

$$f(\mathbf{x}) = b_0 + \sum_{i=1}^{n} a_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Con n coeficientes a_i. Para estimar los valores de los coeficientes b_0 y a_i, solo necesitamos calcular los productos internos del subconjunto (S) de las observaciones de entrenamiento formado por los vectores de soporte.

Esto implica que:

$$a_i = 0 \quad \text{si} \quad \mathbf{x}_i \notin S$$

Así que la fórmula anterior la podemos reescribir como:

$$f(\mathbf{x}) = b_0 + \sum_{i \in S} a_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

Todo esto sirve para la expansión en las máquinas de vectores de soporte. Si reemplazamos el producto interno por una función de producto interno más general llamada "kernel"$K = K(\mathbf{x_i}, \mathbf{x_k})$, podemos modificar los clasificadores de vectores de soporte para usar funciones de kernel no lineales.

Además, para volver a los clasificadores de vectores de soporte, solo tenemos que tomar K como:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \sum_{j=1}^{p} x_{ij} x_{kj}$$

Este kernel es conocido como kernel lineal, utilizado en clasificadores de vectores de soporte aplicados en casos lineales.

También hay otros tipos de funciones de kernel, como el kernel polinomial de grado d, que forman unas fronteras de decisión más flexibles y así podemos ajustar el clasificador de vectores de soporte en espacios de características de mayor dimensión:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{kj}\right)^d$$

Y otro tipo de función de kernel muy popular es el kernel radial:

$$K(\mathbf{x}_i, \mathbf{x}_k) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{kj})^2\right), \quad \gamma > 0$$

donde γ es un parámetro que controla el ancho de la función radial.

Este kernel funciona de la forma en la que si utilizamos una nueva observación de prueba $\mathbf{x}^*$, la suma $\sum_{j=1}^{p} (x_j^* - x_{ij})^2$ será grande y tendremos una exponencial de exponente grande y negativo. Entonces el valor de $K(\mathbf{x}^*, \mathbf{x}_i)$ será pequeño y, por consiguiente, $\mathbf{x}_i$ no tendrá apenas efecto sobre la ubicación de $\mathbf{x}^*$ a través de $f(\mathbf{x}^*)$. Así que el comportamiento del kernel radial tendrá un efecto muy localizado y solo las observaciones cercanas a $\mathbf{x}^*$ tendrán efecto en su clasificación.

En la siguiente figura 2.7 podemos ver cómo en un conjunto de datos no lineal se crean las fronteras de decisión:

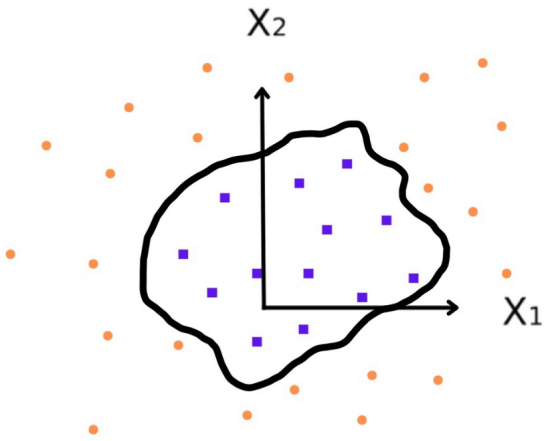


Figura 2.7: Kernel radial

2.3.8. Ventajas e Inconvenientes

Las máquinas de vectores de soporte son una técnica de clasificación realmente buena con varias ventajas, como su eficiencia computacional en grandes conjuntos de datos.

Ventajas

- Son efectivos en altas dimensiones y pueden manejar muchas características.
- Solo es necesario almacenar la información de los vectores de soporte.
- Pueden transformar los datos mediante varias funciones kernel y encontrar un límite de decisión óptimo, lo que les hace ser versátiles.

Inconvenientes

- Son sensibles a datos atípicos, lo que puede llevar a una mala configuración del separador.
- Cuando se usan funciones no lineales son difíciles de interpretar, ya que el límite de decisión se encuentra en el espacio de características transformado.
- El entrenamiento de SVM puede ser computacionalmente costoso.

Capítulo 3

Caso práctico

En este capítulo vamos a aplicar los modelos de machine learning descritos anteriormente para predecir los precios de un activo en un mercado financiero y así poder crear un sistema de trading automático. Formamos una cartera de inversión que está constituida por acciones del Ibex35 y alguna acción de las empresas más grandes del mundo. Utilizamos los datos históricos de cada activo para entrenar y evaluar los modelos.

Aplicamos trading automático con los valores de esta cartera en base a las predicciones del modelo entrenado y comentaremos los rendimientos obtenidos. Además aplicamos una estrategia del análisis técnico para comparar su efectividad con nuestros modelos de machine learning.

Mi objetivo esperado es que se consiga un rendimiento positivo, sobre todo constante a lo largo del tiempo y que supere el rendimiento de la estrategia del análisis técnico.

3.1. Descripción del problema (fuentes y datos seleccionados)

El objetivo es desarrollar una estrategia para realizar trading automático y obtener rendimientos positivos con ella, para lograrlo vamos a intentar utilizar los modelos de árboles de decisión para poder predecir los precios futuros de las acciones de nuestra cartera y las máquinas de vectores de soporte para poder clasificar el precio de mañana en precio más alto o precio más bajo.

Usamos el lenguaje de Python ya que es la mejor opción para desarrollar un sistema de trading gracias a su amplia gama de librerías que nos permiten obtener los datos de los precios, entrenar los modelos de machine learning y realizar la prueba de trading automático.

3.1.1. Fuente de datos

Para poder entrenar estos modelos, los datos utilizados los descargamos desde Python mediante la librería Yfinance [11] que los descarga desde la página de Yahoo! Finance [13]. En este conjunto de datos $(\mathbf{x_1}, \cdots, \mathbf{x_n})$ cada observación está formada por varias características, como el precio de cierre de hoy, el del día anterior, el volumen, etc. Estas características las seleccionamos mediante la validación cruzada. Cada observación está etiquetada con el retorno que tiene el precio en el siguiente día, es decir, tendremos $(\mathbf{y_1}, \ldots, \mathbf{y_n})$.

3.1.2. Período temporal

Hemos obtenido las observaciones de las acciones de nuestra cartera desde el 1 de enero de 2021 al 31 de enero de 2023. El período elegido evita grandes crisis, como la caída de la bolsa en marzo de 2020 debido al Covid-19, que pueden afectar al rendimiento de predicción de los modelos, además de un período corto para evitar otros factores externos que pudieran afectar a las observaciones.

3.1.3. Cartera de inversión

Nuestra cartera de inversión está formada por valores del Ibex35 y por empresas de mayor valor en el mundo. Los valores con sus tickers son Banco Santander (SAN.MC), BBVA (BBVA.MC), Caixabank (CABK.MC), que son tres bancos del Ibex35, Amadeus (AMS.MC), Indra (IDR.MC), que son dos empresas tecnológicas también del Ibex35 y por último Apple (AAPL), Amazon (AMZN) y Tesla (TSLA) que pertenecen al Top 10 de empresas con mayor capitalización de mercado.

El Ibex35 es el principal índice bursátil de la bolsa española formado por las 35 empresas españolas que cotizan en bolsa y que más liquidez tienen. El resto de valores son empresas de Estados Unidos que pertenecen al índice más importante y que refleja la economía del mundo, el S&P500, que son las 500 empresas más grandes que cotizan en bolsa de Estados Unidos.

3.2. Metodología para crear el sistema de trading automático

En esta sección vamos a definir el proceso que hemos llevado a cabo para poder conseguir un sistema de trading automático.

3.2.1. Modelos de machine learning

Tenemos dos algoritmos de machine learning para poder utilizar de alguna manera para crear el sistema de trading automático.

- Árboles de decisión: Usamos el método de consenso de random forest para poder predecir los retornos de un activo el día de mañana para poder tomar una decisión en la operativa.

- Máquinas de vectores de soporte: Las utilizamos para poder clasificar el precio de un activo para el día de mañana en las categorías sube o baja.

Entrenamos nuestros modelos con los datos que hemos determinado anteriormente en 3.1.2 pero solo hasta el mes anterior al que determinamos para realizar la prueba de trading automático y comprobar si el rendimiento es bueno. Realizaremos la operativa en dos meses distintos, en enero de 2023 (mes alcista) y así el período temporal será hasta el 31 de diciembre de 2022, y diciembre de 2022 (mes bajista) y el período temporal será hasta el 30 de noviembre de 2022.

3.2.2. Estrategia

Consideramos dos estrategias distintas pero muy similares para poder hacer trading automático en los valores de nuestra cartera de inversión.

Estrategia RF

La primera estrategia se basa en la implementación del algoritmo de Random forest a nuestro modelo con el conjunto de datos que hemos definido anteriormente para poder predecir los rendimientos futuros de un activo. Obtenemos la función estimadora $\hat{f}(\mathbf{x})$, la evaluamos con el vector de características de hoy, $\mathbf{x_t}$, y así predecimos el retorno de mañana de un activo, $\hat{y}_t$. Entonces comparamos este valor para ver si $\hat{y}_t$ es mayor o menor que 0 y entonces tomamos una operativa:

1. Si $0 < \hat{y}_t$ compramos en el día de hoy y vendemos en el cierre del día de mañana para cerrar la operación.

2. Si $0 > \hat{y}_t$ vendemos en el día de hoy y compramos en el cierre del día de mañana para cerrar la operación.

Estrategia RF + SVM

La segunda estrategia es una implementación de las máquinas de vectores de soporte a la estrategia anterior para comprobar si se incrementa el rendimiento y la efectividad. Con las máquinas de vectores de soporte obtendremos el clasificador $\hat{f}(\mathbf{x})$, la evaluamos con el vector de características de hoy, $\mathbf{x_t}$, y así obtenemos la etiqueta de esta observación, si el precio sube o baja.

La vamos a combinar junto a la primera estrategia de la siguiente forma para decidir cuándo se toma una operación y en qué sentido:

- Si $0 < \hat{y}_t$ y la etiqueta de $\mathbf{x_t}$ es sube, compramos en el día de hoy y vendemos en el cierre del día de mañana para cerrar la operación.
- Si $0 > \hat{y}_t$ y la etiqueta de $\mathbf{x_t}$ es baja, vendemos en el día de hoy y compramos en el cierre del día de mañana para cerrar la operación.
- En los demás casos no se toma ninguna operación, es decir, ese día no se opera.

3.2.3. Estrategia del análisis técnico

El análisis técnico es el estudio de la acción del mercado mediante el uso de las gráficas y varios indicadores. No entramos en más detalle acerca del análisis técnico pero se puede obtener más información en [12].

La estrategia se basa en usar un indicador muy frecuente en los analistas técnicos que son las medias móviles. Estas medias móviles tienen un valor *N* que indica que el precio al que está la media móvil equivale a la media de los últimos *N* precios diarios de una acción.

Utilizamos la media de período 3 días y 5 días, y la operativa es la siguiente:

- Cuando la media de 3 períodos cruza por encima a la media de 5 días, se realiza la operación comprando, cuando la media de 3 períodos cruza por debajo a la media de 5 períodos se vende y así se cierra la operación.

3.2.4. Procedimiento del sistema de trading automático

1. Obtenemos nuestro conjunto de datos de una acción y lo dividimos en dos partes, el conjunto de entrenamiento y el de prueba.
2. Entrenamos los modelos con el conjunto de entrenamiento haciendo una validación cruzada para obtener los mejores hiperparámetros y así obtener las funciones predictoras, en el caso de los Random Forest, y el clasificador, en el caso de la máquina de vectores de soporte.
3. Aplicamos la estrategia desde el primer día del conjunto de prueba, por ejemplo para la primera estrategia, para el primer día hallamos $\hat{f}(\mathbf{x_t})$, realizamos la operativa, la cerramos al día siguiente y en este día volveríamos a calcular $\hat{f}(\mathbf{x_{t+1}})$ para tomar la segunda operativa y así sucesivamente hasta el último día del mes que se cerraría la última operación.

3.3. Resultados

Una vez que ya tenemos todo definido y establecido, explicamos cómo vamos a calcular el rendimiento que tendrá nuestra cartera de inversión y cada acción para cada estrategia que hemos explicado. Esto nos sirve para comparar las estrategias y la efectividad de nuestros modelos para predecir precios.

3.3.1. Capital

Comenzamos con un capital disponible de 100.000€, el cual dividimos para cada acción en partes iguales, es decir, en cada acción tenemos disponibles 12.500€. En cada operativa que realizamos, invertimos todo el capital que tenemos disponible en ese momento en esa acción.

3.3.2. Índices de rendimiento

Para medir el éxito de cada estrategia, vamos a comprobar qué rendimiento nos genera cada estrategia para la cartera de inversión además de la tasa de aciertos, es decir, el porcentaje de aciertos para cada acción. También como habíamos dicho antes, esto lo evaluamos para cada tipo de mes (alcista y bajista) para ver cómo se comportan los modelos y las estrategias según en el momento en el que nos encontremos.

3.3.3. Tablas con los resultados de cada estrategia

Tras realizar el entrenamiento de los modelos y aplicar las tres estrategias, hemos obtenido los siguientes resultados para un mes bajista. Los resultados del mes alcista se encuentran en apéndice I.1:

Acción	Capital Final	Rend. (%)	Nº de Operaciones	Op. Exitosas	Porcentaje
SAN	12280.24	-1.76	4	1	25
BBVA	11946.31	-4.43	4	1	25
CABK	13568.25	8.54	2	1	50
AMS	12033.61	-3.73	3	0	0
IDR	12826.56	2.61	3	2	66.7
AAPL	11258.85	-9.95	3	0	0
AMZN	11422.65	-8.61	3	0	0
TSLA	11535.21	-7.72	2	0	0
Total	**96871.68**	**-3.13**	**24**	**5**	**20.83**

Cuadro 3.1: Resultados del AT para la cartera de inversión en un mes bajista

Acción	Capital Final	Rend. (%)	Nº de Operaciones	Op. Exitosas	Porcentaje
SAN	12125.90	-2.99	20	9	45
BBVA	11950.16	-4.40	20	8	40
CABK	11718.19	-6.26	20	11	55
AMS	12075.70	-3.40	20	7	35
IDR	12559.40	0.48	20	9	45
AAPL	12917.08	3.34	20	12	60
AMZN	14835.35	18.68	20	13	65
TSLA	16797.94	34.38	20	13	65
Total	**105979.72**	**5.98**	**160**	**82**	**51.25**

Cuadro 3.2: Resultados del RF para la cartera de inversión en un mes bajista

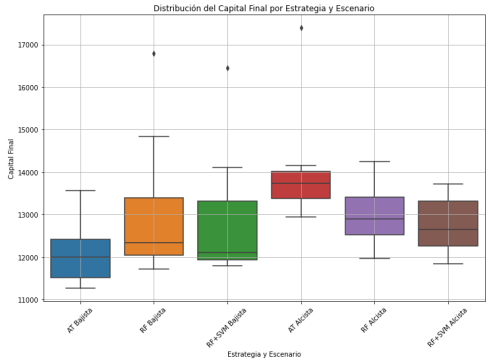
Acción	Capital Final	Rend. (%)	Nº de Operaciones	Op. Exitosas	Porcentaje
SAN	11946.18	-4.43	14	5	35.7
BBVA	11988.58	-4.09	11	3	27.3
CABK	11799.35	-5.60	16	8	50
AMS	11882.89	-4.95	14	5	35.7
IDR	12228.46	-2.18	14	5	35.7
AAPL	13056.14	4.45	16	10	62.5
AMZN	14107.04	12.86	15	9	60
TSLA	16442.06	31.54	13	10	76.9
Total	**103450.70**	**3.45**	**113**	**55**	**48.67**

Cuadro 3.3: Resultados del RF+SVM para la cartera de inversión en un mes bajista

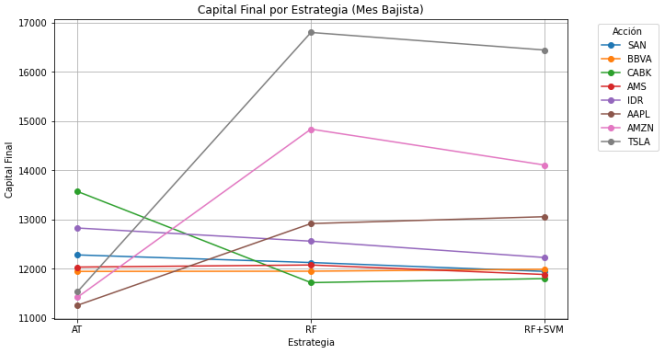
Podemos observar que la estrategia del análisis no ha sido rentable en la cartera de inversión generando una pérdida de 3,13 %, sin embargo, las estrategias que han usado los modelos de Random Forest y SVM han resultado rentables generando un 5,98 % en el caso de solo el Random Forest y un 3,45 % cuando lo combinamos con SVM. Comparando la segunda estrategia con la tercera, con las SVM se realiza un 29,48 % de operaciones menos, tanto positivas como negativas, es decir, es una estrategia más conservadora al tomar menos riesgo aunque a veces se penalice con la no toma de operaciones que resultarían positivas. Esto se puede apreciar para el ejemplo que explicamos en apéndice I.2. Un dato a destacar es que, si nos fijamos en la estrategia RF + SVM, a veces tenemos la idea de que para generar un rendimiento positivo necesitamos acertar más del 50 % de las operaciones, pero podemos apreciar que en este caso se acierta el 48,67 % y se está generando un rendimiento positivo.

3.4. Gráfica de ejes paralelos y diagrama de cajas

A continuación podemos ver la representación de los resultados del capital final mediante una gráfica de ejes paralelos del mes bajista (el mes alcista se encuentra en apéndice I.1) y la representación de los diagramas de cajas para ambos escenarios.



(a) Diagrama de cajas para ambos escenarios



(b) Gráfica de ejes paralelos en mes bajista

La gráfica de ejes paralelos muestra claramente el buen funcionamiento de nuestros modelos, especialmente cuando se aplican a acciones del S&P500 y en meses alcistas, como se puede ver en el Apéndice I.1. El diagrama de cajas revela que la media del capital final de todas las estrategias no baja de 12000€, y en un mes alcista, esta media supera el capital inicial. Los modelos también aportan estabilidad en el rendimiento de la estrategia en ambos escenarios. En contraste, el análisis técnico es más volátil y su rendimiento varía significativamente según el mes, haciéndolo menos constante y regular en comparación con las estrategias basadas en modelos.

3.5. Conclusión

El objetivo de conseguir que nuestros modelos de machine learning funcionen en la predicción de precios para realizar un sistema de trading automático, ha sido logrado con un rendimiento positivo. También se ha logrado el segundo objetivo que consistía en superar a una estrategia de análisis técnico. En el mes alcista no se supera respecto al rendimiento, pero sí que es mejor en términos de volatilidad y constancia en distintos escenarios. El sistema de trading se podría mejorar utilizando otras características, usando otros períodos temporales, otras acciones e incluso utilizando otros métodos de machine learning con el fin de lograr un sistema más robusto en términos de rentabilidad, lo cual queda abierto para una futura investigación y trabajo. Finalmente, hemos realizado un análisis detallado de los resultados en el Apéndice I para comprobar cómo funciona la estrategia con una acción de nuestra cartera de inversión, Tesla, y cómo es el proceso de optimización de los hiperparámetros. De mismo modo, en el Apéndice II, describimos un esquema del código empleado en Python para construir el sistema de trading automático.

Bibliografía

[1] AMAZON WEB SERVICES, *What is Reinforcement Learning?*, https://aws.amazon.com/es/what-is/reinforcement-learning/.

[2] BREIMAN, L., *Random forests*, Machine Learning, 45(1):5–32, 2001.

[3] CARMONA SUÁREZ, E. J., *Tutorial sobre Máquinas de Vectores Soporte (SVM)*, 2013.

[4] EFRON, B., *Bootstrap methods: Another look at the jackknife*, *The Annals of Statistics*, **7**(1), pp. 1–26, 1979.

[5] GUEROLA PÉREZ, L., *Desarrollo de un Sistema de Trading Algorítmico*, E.T.S.I. Industriales (UPM), 2020, https://oa.upm.es/63166/1/TFG_LAURA_GUEROLA_PEREZ.pdf.

[6] HALLS-MOORE, M., *Advanced Algorithmic Trading*, QuantStart.com, London, 2017.

[7] HAMILTON, J. D., *Time Series Analysis*, Princeton University Press, 1994.

[8] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J., *The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd Ed.*, Springer, 2011.

[9] JAMES, G., WITTEN, D., HASTIE, T., AND TIBSHIRANI, R., *An Introduction to Statistical Learning: with applications in R*, Springer, 2013.

[10] MARTÍN HINOJOSA, I., *El Trading Algorítmico en los Mercados Financieros: Estrategia Basada en la Volatilidad de los Precios de las Opciones*, Universidad Pontificia Comillas, 2018, https://repositorio.comillas.edu/jspui/bitstream/11531/23667/1/TFG%20-%20Isabel%20Martin%20Hinojosa.pdf.

[11] SÁNCHEZ, M., *Práctica en el uso de la librería yfinance*, https://github.com/manursanchez/Practicas_Python_DataScience/blob/main/1.%20Pr%C3%A1ctica%20en%20el%20uso%20de%20la%20librer%C3%ADa%20yfinance.ipynb.

[12] WIKIPEDIA, *Análisis técnico*, https://es.wikipedia.org/wiki/An%C3%A1lisis_t%C3%A9cnico.

[13] YAHOO FINANZAS, *Yahoo Finanzas, Noticias y Cotizaciones*, https://es.finance.yahoo.com/.

Apéndice I

Análisis de los resultados

I.1. Resultados de las estrategias en un mes alcista

En las siguientes tablas tenemos un resumen de cómo han funcionado las estrategias en nuestra cartera de inversión para un mes alcista. En este caso, a diferencia del mes bajista, el análisis técnico ha tenido un gran rendimiento y la mayoría de sus operaciones han sido positivas. En las estrategias en las que hemos empleado los algoritmos de Random Forest y las máquinas de vectores de soporte también se ha obtenido un rendimiento positivo, aunque inferior que en el mes bajista. Aunque el número de operaciones positivas ha sido mayor que en el mes bajista.

Acción	Capital Final	Rend. (%)	Nº de Operaciones	Op. Exitosas	Porcentaje
SAN	13724.38	9.80	2	2	100
BBVA	13531.14	8.25	3	1	33.3
CABK	12938.62	3.51	3	3	100
AMS	14157.82	13.26	2	2	100
IDR	12935.27	3.49	3	2	66.7
AAPL	13734.15	9.87	1	1	100
AMZN	13973.37	11.78	3	3	100
TSLA	17393.76	39.15	1	1	100
Total	**109388.51**	**9.39**	**18**	**15**	**83.3**

Cuadro I.1: Resultados del AT para la cartera de inversión en un mes alcista

Acción	Capital Final	Rend. (%)	Nº de Operaciones	Op. Exitosas	Porcentaje
SAN	11958.40	-4.34	20	8	40
BBVA	13920.36	11.36	20	16	80
CABK	13002.63	4.02	20	13	65
AMS	12631.91	1.03	20	12	60
IDR	12177.91	-2.57	20	12	60
AAPL	12793.11	2.34	18	10	55.6
AMZN	13238.61	5.91	18	9	50
TSLA	14248.58	13.99	18	10	55.6
Total	**103971.51**	**3.97**	**154**	**90**	**58.44**

Cuadro I.2: Resultados del RF para la cartera de inversión en un mes alcista

Acción	Capital Final	Rend. (%)	Nº de Operaciones	Op. Exitosas	Porcentaje
SAN	11834.28	-5.32	18	6	33.3
BBVA	13720.84	9.77	16	13	81.3
CABK	12384.41	-0.93	13	8	61.5
AMS	12706.40	2.93	15	8	53.3
IDR	11860.22	-5.12	12	5	41.7
AAPL	12578.82	2.63	14	7	50
AMZN	13474.86	7.80	12	7	58.3
TSLA	13270.75	6.16	14	7	50
Total	**101830.58**	**1.83**	**114**	**61**	**53.51**

Cuadro I.3: Resultados del RF+SVM para la cartera de inversión en un mes alcista

Por último, tenemos la gráfica de ejes paralelos para el mes alcista, en la que difiere respecto al mes bajista debido a la irregularidad que presenta la estrategia del análisis técnico, que en este caso es muy positiva.

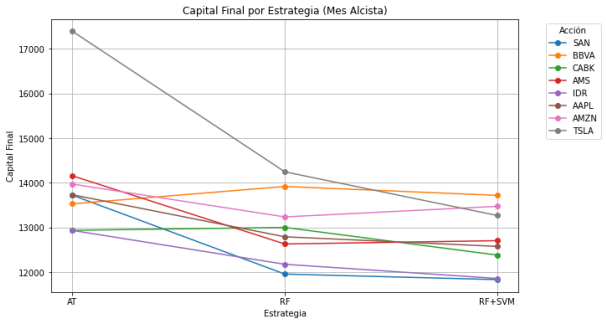


Figura I.1: Gráfica de ejes paralelos en mes alcista

I.2. Funcionamiento de las estrategias en Tesla

A continuación vamos a ver cómo ha sido el funcionamiento de cada estrategia en una acción de nuestra cartera de inversión, Tesla. Veremos una gráfica con la toma de operaciones y otra con la evaluación del capital.

I.2.1. Análisis técnico

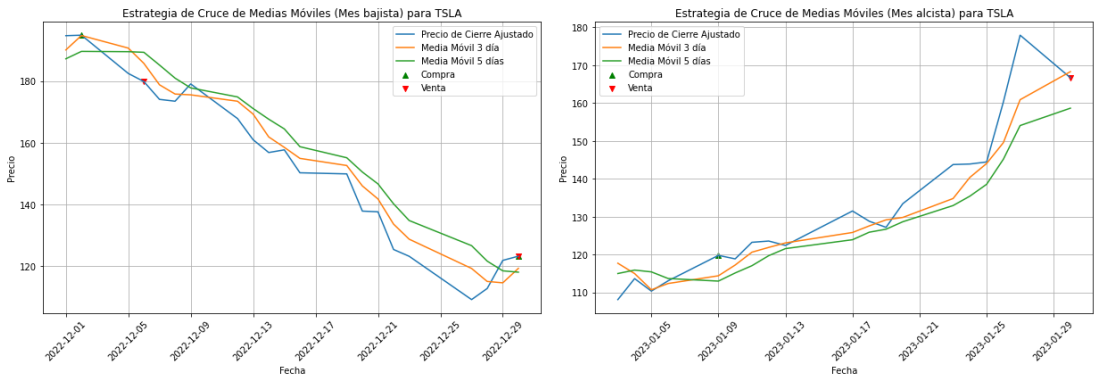


Figura I.2: Gráficas del precio, medias móviles y operaciones en ambos meses

Al utilizar un período temporal de un mes y unas medias móviles de 3 y 5 períodos, no da mucho margen a que se realicen muchos cruces entre ellas. Por lo que el número de operaciones suele ser muy pequeño.

I.2.2. Random Forest

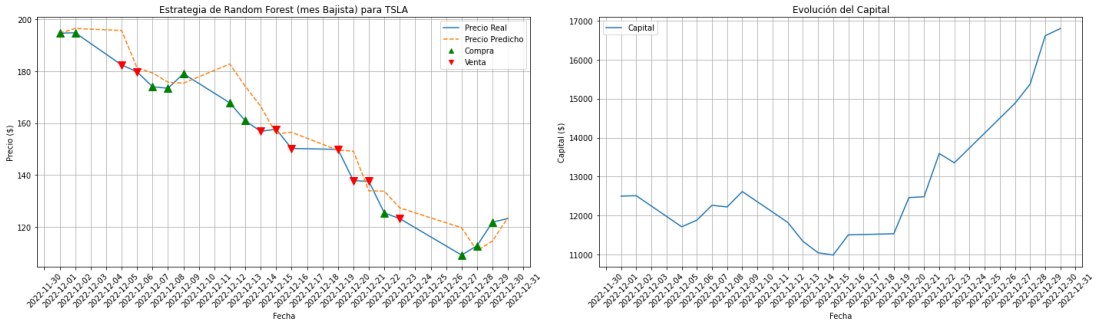


Figura I.3: Gráficas del precio, precio predicho, operaciones y capital en mes bajista

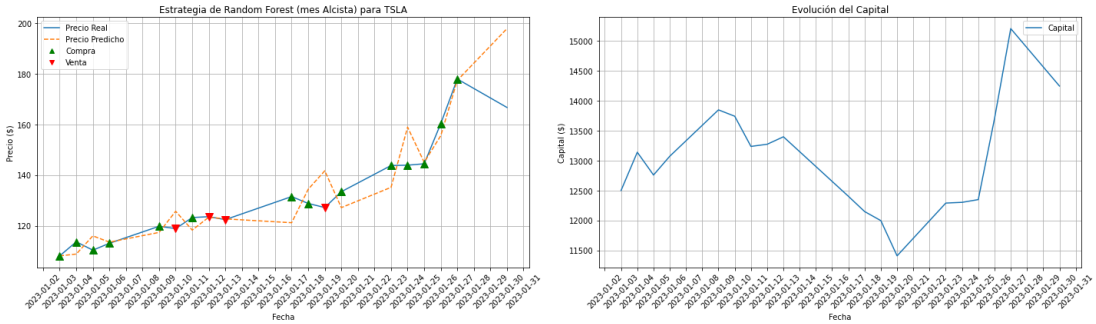


Figura I.4: Gráficas del precio, precio predicho, operaciones y capital en mes alcista

Podemos observar cómo nuestro modelo predice con bastante semejanza el precio real del activo, y gracias a ello, tomar una serie de operaciones que hacen que la evolución del capital sea positiva al final del mes aunque en ocasiones se produzcan caídas debido a operaciones que resultan negativas.

I.2.3. Random Forest + SVM

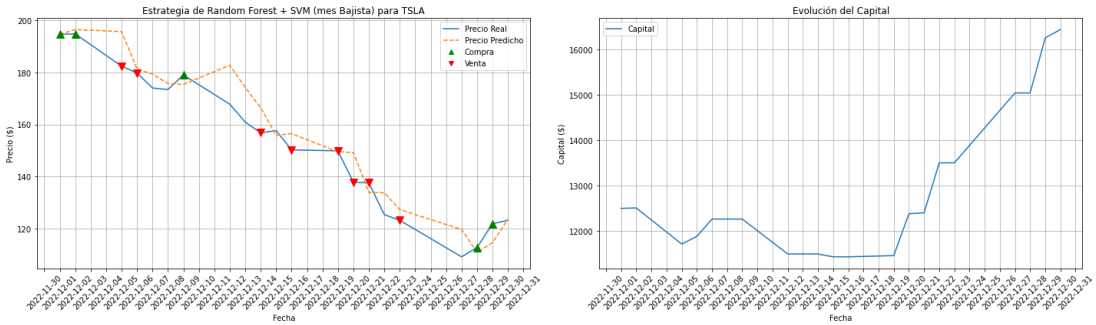


Figura I.5: Graficas del precio, precio predicho, operaciones y capital en mes bajista

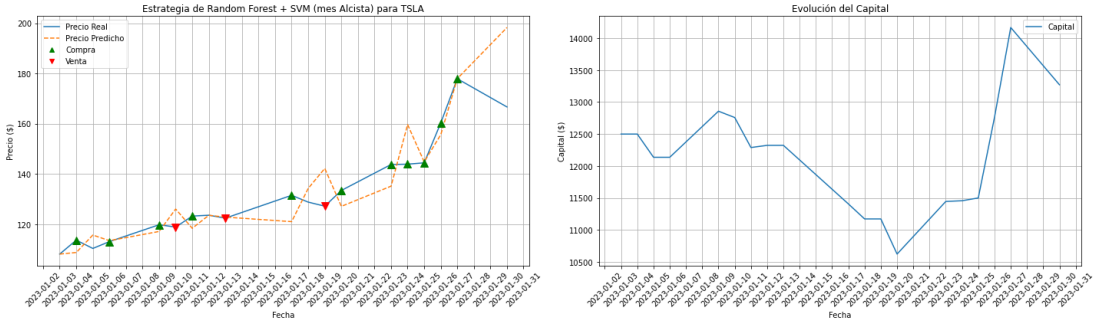


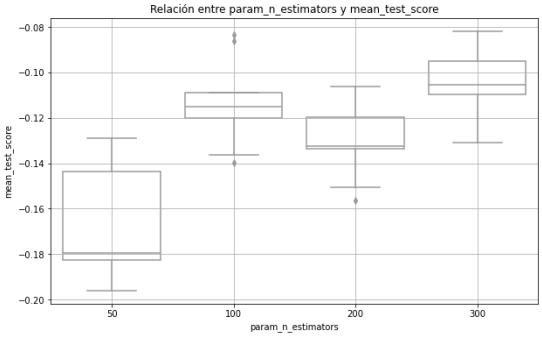
Figura I.6: Graficas del precio, precio predicho, operaciones y capital en mes alcista

Podemos destacar la diferencia con el número de operaciones que se toman en esta estrategia respecto a la anterior, en este caso se toman menos operaciones que resultaban positivas y negativas.

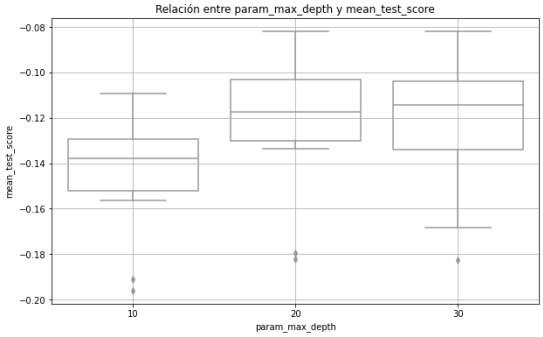
I.3. Optimización de los hiperparámetros

Vamos a desarrollar cómo optimizamos los valores de los hiperparámetros para entrenar los modelos con el algoritmo de Random Forest. Para ello continuamos con el ejemplo de Tesla para ver cómo los hemos optimizado. Para este algoritmo usamos los hiperparámetros del número de estimadores, la profundidad del árbol y el número de muestras para que se divida un nodo.

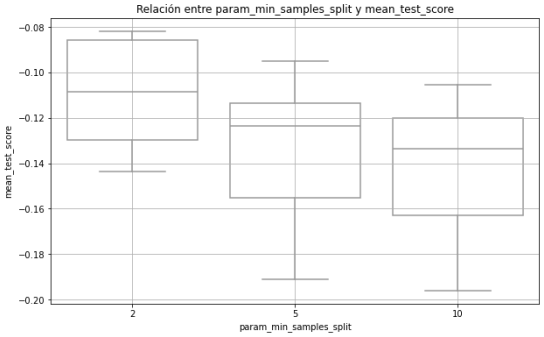
Nosotros establecemos una horquilla de valores predeterminada para cada hiperparámetro y mediante la validación cruzada obtenemos las siguientes gráficas:



(a) Número de estimadores



(b) Profundidad



(c) Muestras en nodo

Figura I.7: Gráficas de los hiperparámetros para cada valor

Para ver qué valor de cada hiperparámetro es más óptimo, en la validación cruzada se combinan todos los valores de todos los hiperparámetros y obtenemos un valor para cada posible combinación que indica el buen rendimiento de cada árbol. El objetivo es que mediante los diagramas de cajas busquemos

para qué valor del hiperparámetro es mayor el rendimiento y además no sufra de una gran variabilidad.

Por ejemplo, si nos fijamos en el número de muestras para que un nodo se divida, vemos que el valor 2 tiene una menor variabilidad y mayor valor en el rendimiento que el resto de candidatos. Por eso tendríamos que ajustar la horquilla de valores y volver a hacer la prueba con valores cercanos a 2. Ahora, si nos fijamos en el número de estimadores, el valor 300 es el que aporta un mayor rendimiento y tampoco sufre mucha variabilidad, por lo que deberíamos ajustar la horquilla a valores próximos a 300 y mayores para saber cuál sería el óptimo.

I.4. Importancia de las características

Para saber si las características que hemos elegido para nuestro conjunto de datos están aportando suficiente valor al modelo, se puede hacer mediante la siguiente gráfica:

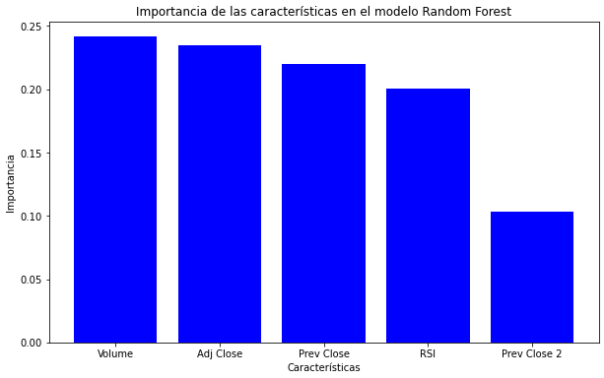


Figura I.8: Importancia de las características

En este caso se puede observar cómo la característica del volumen está siendo bastante importante para nuestro modelo y está aportando más valor que el resto. Todas ellas son bastante importantes excepto el precio de cierre de hace dos días, por lo que nos podríamos plantear quitar esta característica de nuestro conjunto de datos.

Apéndice II

Código de Python

Vamos a hacer un resumen del código que hemos usado en Python para entrenar nuestros modelos y para comprobar los rendimientos que han tenido nuestras estrategias con la capacidad predictiva de los modelos.

II.1. Librerías Importantes

Definimos las librerías más relevantes para realizar el entrenamiento del modelo, la estrategia y la visualización de los resultados.

```
import yfinance as yf # Descargar datos de Yahoo Finance
import pandas as pd # Manipulacion y analisis de datos estructurados
import numpy as np # Computacion numerica
from sklearn.ensemble import RandomForestRegressor # Uso del metodo Random
    Forest
from sklearn.model_selection import GridSearchCV # Uso de la validacion cruzada
from sklearn.svm import SVC # Uso de las maquinas de vectores de soporte
import matplotlib.pyplot as plt #Crear graficos y visualizaciones
```

II.2. Obtención de los datos

A continuación se muestra cómo descargamos los datos de un ticker en concreto, por ejemplo, en este caso se utiliza el de Tesla y para un período temporal que comienza el 2021-01-01 al 2023-01-31.

```
# Descargar los datos de la accion
ticker = 'TSLA'
start_date = '2021-01-01'
end_date = '2023-01-31'
data = yf.download(ticker, start=start_date, end=end_date)
```

II.3. Preparación de los conjuntos de datos

Creamos los conjuntos de entrenamiento y de prueba con las características que queremos probar.

```
# Dividir los datos en entrenamiento y prueba
train_data = data[:'2022-12-31']
test_data = data['2023-01-01':]

# Definir las caracteristicas (X) y el objetivo (y)
features = ['Adj Close', 'Prev Close', 'Prev Close 2', 'Volume', 'RSI']
X_train = train_data[features]
y_train = train_data['Return']
```

II.4. Entrenamiento de los modelos SVM y Random Forest

En el siguiente fragmento de código se muestra cómo se entrenan los modelos y cómo se aplica la validación cruzada.

```
#Entrenar el modelo de SVM
#Definicion de la orquilla de hiperparametros
    param_grid_svm = {
    'svc__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'svc__C': [0.1, 1, 10, 100, 1000, 10000],
    'svc__gamma': [1e-2, 1e-3, 1e-4, 'scale', 'auto'],
    'svc__degree': [2, 3, 4, 5],
    'svc__coef0': [0.0, 0.1, 0.5, 1.0]

#Entrenamiento del modelo mediante validacion cruzada
tscv = TimeSeriesSplit(n_splits=5)
svm_pipeline = make_pipeline(StandardScaler(), SVC(random_state=random_seed))
grid_search_svm = GridSearchCV(svm_pipeline, param_grid_svm, cv=tscv, n_jobs=-1,
    scoring='accuracy')
grid_search_svm.fit(X_train, y_train_class)
best_svm = grid_search_svm.best_estimator_

#impresion de los hiperparametros optimos
print("Mejores hiperparametros para SVM:")
print(grid_search_svm.best_params_)

# Entrenar el modelo de Random Forest
#Definicion de la orquilla de hiperparametros
param_grid_rf = {
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5]
}

#Entrenamiento del modelo mediante validacion cruzada
grid_search_rf = GridSearchCV(RandomForestRegressor(random_state=random_seed),
    param_grid_rf, cv=tscv, n_jobs=-1)
grid_search_rf.fit(X_train, y_train)
best_rf = grid_search_rf.best_estimator_

#Impresion de los hiperparametros optimos
print("Mejores hiperparametros para Random Forest:")
print(grid_search_rf.best_params_)
}
```

II.5. Análisis de las características y de los hiperparámetros

Con el siguiente trozo de código se obtiene una serie de gráficas con las que se pueden interpretar los resultados de las características y de los hiperparámetros elegidos para poder realizar un ajuste en ellos y mejorar los modelos.

```
# Importancia de las caracteristicas
feature_importances = best_rf.feature_importances_
importance_df = pd.DataFrame({'Feature': features, 'Importance':
    feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Graficar la importancia de las caracteristicas
plt.figure(figsize=(10, 6))
plt.bar(importance_df['Feature'], importance_df['Importance'], color='b')
plt.xlabel('Caracteristicas')
plt.ylabel('Importancia')
```

```
plt.title('Importancia de las caracteristicas en el modelo Random Forest')
plt.show()

# Obtener los resultados del GridSearchCV
cv_results_rf = pd.DataFrame(grid_search_rf.cv_results_)

# Visualizar los resultados de los hiperparametros con diagramas de cajas (Boxplots)
param_columns_rf = [col for col in cv_results_rf.columns if col.startswith('param_')]
score_column_rf = 'mean_test_score'

for param in param_columns_rf:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=cv_results_rf[param], y=cv_results_rf[score_column_rf], color='white')
    plt.xlabel(param)
    plt.ylabel(score_column_rf)
    plt.title(f'Relacion entre {param} y {score_column_rf}')
    plt.grid(True)
    plt.show()
```

II.6. Estrategia

Realización de la estrategia para obtener el rendimiento que se tiene con los modelos entrenados. Obtendremos la evolución del capital, el número de operaciones y el desarrollo de cada operación.

```
# Estrategia de trading
capital = 12500
initial_capital = capital
capital_evolution = [capital]
dates = [test_data.index[0]]

# Almacenar precios reales y predichos
actual_prices = [test_data['Adj Close'].iloc[0]]  # Incluir el precio inicial
predicted_prices = [test_data['Adj Close'].iloc[0]]  # Incluir el precio predicho inicial (el mismo)
buy_signals = []
sell_signals = []

# Contadores de operaciones
total_operations = 0
positive_operations = 0
negative_operations = 0

# Definir las caracteristicas para el conjunto de prueba
X_test = test_data[features]
y_test = test_data['Return']

for i in range(len(test_data)-1):
    current_features = pd.DataFrame([X_test.iloc[i].values], columns=features)
    next_day_price = test_data['Adj Close'].iloc[i + 1]
    current_date = test_data.index[i]
    next_date = test_data.index[i + 1]

    # Predecir el retorno del dia siguiente
    predicted_return = best_rf.predict(current_features)[0]
    current_price = test_data['Adj Close'].iloc[i]
    predicted_next_price = current_price * (1 + predicted_return)

    # Almacenar precios reales y predichos
    actual_prices.append(next_day_price)
```

```
    predicted_prices.append(predicted_next_price)

    # Predecir si el precio subira o bajara con SVM
    predicted_label = best_svm.predict(current_features)[0]
    predicted_label_text = 'Sube' if predicted_label == 1 else 'Baja'

    # Estrategia de compra/venta combinada
    if current_price < predicted_next_price and predicted_label == 1:
        operation = 'Compra'
        profit = (next_day_price - current_price) / current_price * capital
        buy_signals.append((current_date, current_price))  # Cambiar a current_date y current_price
    elif current_price < predicted_next_price and predicted_label == 0:
        operation = 'No operacion'
        profit = 0
    elif current_price > predicted_next_price and predicted_label == 1:
        operation = 'No operacion'
        profit = 0
    elif current_price > predicted_next_price and predicted_label == 0:
        operation = 'Venta'
        profit = (current_price - next_day_price) / current_price * capital
        sell_signals.append((current_date, current_price))  # Cambiar a current_date y current_price
    else:
        operation = 'No operacion'
        profit = 0

    capital += profit
    capital_evolution.append(capital)
    dates.append(next_date)

    # Contar operaciones
    if operation != 'No operacion':
        total_operations += 1
        if profit > 0:
            positive_operations += 1
        else:
            negative_operations += 1
```

II.7. Impresión por pantalla de los resultados

Por último mostramos cómo se crean las gráficas y se imprime por pantalla las operaciones.

```
    # Imprimir los resultados de cada operacion
    print("-------------")
    print(f"{operation} al precio del dia {current_date.strftime('%Y-%m-%d')}: {current_price:.2f}")
    print(f"Precio predicho para el dia {next_date.strftime('%Y-%m-%d')}: {predicted_next_price:.2f}")
    print(f"Precio del dia {next_date.strftime('%Y-%m-%d')}: {next_day_price:.2f}")
    print(f"Etiqueta estimada por SVM: {predicted_label_text}")
    print(f"Beneficio/perdida: {profit:.2f}")
    print(f"Capital despurs de la operacion: {capital:.2f}")

# Imprimir resultados de las operaciones
print(f"\nTotal de operaciones: {total_operations}")
print(f"Operaciones positivas: {positive_operations}")
print(f"Operaciones negativas: {negative_operations}")

# Graficar la evolucion del capital
plt.figure(figsize=(10, 6))
```

```
plt.plot(dates, capital_evolution, label='Capital')
plt.xlabel('Fecha')
plt.ylabel('Capital ($)')
plt.title('Evolucion del Capital')
plt.xticks(rotation=45)
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Graficar los precios reales y predichos
plt.figure(figsize=(10, 6))
plt.plot(test_data.index, actual_prices, label='Precio Real')
plt.plot(test_data.index, predicted_prices, label='Precio Predicho', linestyle='
    --')
plt.xlabel('Fecha')
plt.ylabel('Precio ($)')
plt.title(f'Estrategia de Random Forest + SVM (mes Alcista) para {ticker}')
plt.xticks(rotation=45)
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=1))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))

# Agregar los marcadores de compra y venta
plt.plot([], [], '^', color='green', label='Compra')  # Para la leyenda
plt.plot([], [], 'v', color='red', label='Venta')     # Para la leyenda
for signal in buy_signals:
    plt.plot(signal[0], signal[1], '^', color='green', markersize=10)
for signal in sell_signals:
    plt.plot(signal[0], signal[1], 'v', color='red', markersize=10)

plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```