UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER EN INTELIGENCIA ARTIFICIAL AVANZADA:
FUNDAMENTOS, MÉTODOS Y APLICACIONES

Trabajo de fin de máster

Aplicación de Aprendizaje Automático en Trading Algorítmico

18 de septiembre de 2020

Miguel Sánchez Mendoza

Dirigido por: Prof. Dr. Francisco Javier Díez Vegas

Curso: 2019-2020: 2ª Convocatoria

Agradecimientos

Mi más sincero agradecimiento a Francisco Javier Díez, tutor de este proyecto, por su tiempo y dedicación. También me gustaría agradecer a todos los docentes que me han instruido, orientado y aconsejado a lo largo de mis años de estudio. Su esfuerzo y vocación han sido y serán siempre un referente. Por último, pero no menos importante, agradecer el apoyo de mi familia y amigos.

Resumen

Una estrategia de inversión es un plan fijo diseñado para obtener un rendimiento rentable al operar en el mercado. En los mercados financieros existen diferentes estrategias que permiten obtener beneficios a largo y corto plazo. Cada una de estas estrategias tiene ventajas e inconvenientes, pero todas pueden ser analizadas y optimizadas para que su rendimiento sea el más adecuado en cada momento.

Nuestro estudio se centrará en el desarrollo, prueba, optimización y evaluación de dos estrategias concretas. Estas dos estrategias se diferencian principalmente en el marco temporal en el que esperan obtener beneficios. La primera estrategia, denominada *comprar y mantener*, se basa en la idea de que todos los mercados evolucionan favorablemente en el largo plazo, por lo que establece pautas para adquirir activos en momentos en los que éstos están infravalorados con la esperanza de que, transcurrido un largo periodo de tiempo (entre 2 y 15 años), aumenten su valor. Por otro lado, analizaremos una estrategia de inversión a muy corto plazo que utiliza indicadores técnicos que pretenden determinar y aprovechar momentos en los que el precio de un activo va a realizar un determinado movimiento (aumentar o disminuir) en los siguientes minutos.

Para el desarrollo de las pruebas se crearán algoritmos que operen el mercado de forma automática utilizando datos históricos y herramientas de simulación creadas para tal fin, utilizando el lenguaje de programación Python. En ambas estrategias se partirá de un algoritmo de aplicación de estrategia básico, para luego optimizarlo mediante el uso de herramientas de aprendizaje automático. Concretamente, para la estrategia de largo plazo utilizaremos clasificación mediante bosques aleatorios (*random forests*) para determinar el mejor momento para invertir, mientras que para la estrategia a corto plazo utilizaremos máquinas de vectores de soporte (*support vector machines*, SVMs) como método de regresión para estimar la variación de precio de un activo en los próximos minutos y calcular la pérdida media que podemos esperar y así determinar si debemos comprar o vender en cada momento.

Abstract

An investment strategy is a fixed plan designed to obtain a profitable return when trading in the market. In the financial markets there are different strategies that allow obtaining benefits in the long and short term. Each of these strategies has advantages and disadvantages, but all can be analyzed and optimized so that their performance is the most appropriate at all times.

Our study will focus on the development, testing, optimization and evaluation of two specific strategies. These two strategies differ mainly in the time frame in which they expect to make a profit. The first strategy, called buy and hold, is based on the idea that all markets evolve favorably in the long term, so it establishes guidelines for acquiring assets at times when they are undervalued in the hope that, after a period of time, long period of time (between 2 and 15 years), increase their value. On the other hand, we will analyze a very short-term investment strategy that uses technical indicators that aim to determine and take advantage of moments in which the price of an asset is going to make a certain movement (increase or decrease) in the following minutes.

For the development of the tests, algorithms will be created that operate the market automatically using historical data and simulation tools created for this purpose, using the Python programming language. In both strategies, we will start from a basic strategy application algorithm, and then optimize it through the use of machine learning tools. Specifically, for the long-term strategy we will use classification through random forests to determine the best time to invest, while for the short-term strategy we will use support vector machines (SVMs) as a regression method. to estimate the price variation of an asset in the next few minutes and calculate the average loss that we can expect and thus determine whether we should buy or sell at all times.

Índice general

1. Introducción general y objetivos **1**

1.1. Motivación 1

1.2. Objetivos 3

1.3. Metodología 3

1.4. Estructura de la memoria 5

2. Estado del Arte **7**

2.1. Trading Algorítmico 7

2.1.1. Análisis y Predicción de Valores de Mercados 8

2.1.1.1. Análisis de movimientos del mercado 9

2.1.1.2. Análisis de influencias 11

2.1.2. Conclusiones 12

2.2. Conceptos Básicos de Trading 12

2.2.1. Econometría 12

2.2.2. Mercados Financieros 13

2.2.3. Índice bursátil 15

2.2.4. High-Frequency Trading 16

2.2.5. Estrategia de trading 17

2.2.5.1. Estrategias según tendencia 17

2.2.5.2. Estrategias según el marco temporal 19

2.2.5.3. Tipos de Inversor 21

2.2.5.4. Datos Financieros 22

2.2.6. Valor en riesgo 23

2.3. Conceptos de Aprendizaje Automático 26

2.3.1. Bosques Aleatorios 26

2.3.2. SVMs 27

2.3.2.1. Kernel Trick 29

2.3.2.2. SVR 30

2.3.2.3. GridSearchCV 31

2.4. Conclusiones 33

3. Estrategia *comprar y mantener* **35**
3.1. Diseño de los experimentos . . . 35
3.1.1. Estrategia *comprar y mantener* simple . . . 36
3.1.2. Estrategia *comprar y mantener* con bosques aleatorios . . . 37
3.2. Implementación . . . 38
3.2.1. Preparación del entorno . . . 39
3.2.2. Implementación de *comprar y mantener* simple . . . 42
3.2.3. Implementación de *comprar y mantener* con bosques aleatorios . . . 42
3.3. Evaluación y resultados . . . 45
3.4. Discusión . . . 48

4. Estrategia de *scalping* **51**
4.1. Diseño de los experimentos . . . 52
4.1.1. Estrategia de *scalping* simple . . . 52
4.1.2. Estrategia de *scalping* con VaR . . . 57
4.1.3. Estrategia de *scalping* con VaR optimizado con SVR . . . 59
4.2. Implementación . . . 60
4.2.1. Preparación del entorno . . . 60
4.2.2. Implementación de la estrategia de *scalping* simple . . . 60
4.2.3. Implementación de la estrategia de *scalping* con VaR . . . 64
4.2.4. Implementación de la estrategia de *scalping* con VaR optimizado con SVR . . . 65
4.3. Evaluación y resultados . . . 67
4.4. Discusión . . . 70

5. Conclusiones y trabajos futuros **73**
5.1. Conclusiones . . . 73
5.2. Trabajos futuros . . . 75

A. Código utilizado en los experimentos **77**
A.1. Preparación del Entorno . . . 78
A.2. Estrategia *comprar y mantener* . . . 79
A.3. Estrategia de *scalping* . . . 82

Índice de figuras

2.1. Algoritmo de búsqueda K2 (Zuo y Kita, 2012). 9
2.2. Orden total de relaciones entre tasas de subidas y bajadas (Zuo y Kita, 2012). . . . 9
2.3. Red bayesiana para la predicción de las subidas y bajadas de FTSE100 (Zuo y Kita, 2012). 10
2.4. Red bayesiana aprendida (Khorram, Ph.D. y Hui, 2011). 11
2.5. Identificación de cambios de tendencia mediante bandas de Bollinger. 18
2.6. Valor de la cotización de Facebook con una media móvil. 20
2.7. Valor de la cotización de Facebook con dos medias móviles y bandas de Bollinger. . 21
2.8. Operación de compra realizada siguiendo una estrategia basada en bandas de Bollinger. 22
2.9. Distribución normal de la pérdida y la ganancia (Yamai y Yoshiba, 2002). 25
2.10. Hiperplanos en un espacio bidimensional (Weinberg, 2012). 28
2.11. Kérnel Polinómico (Elisfm, 2009a). 30
2.12. Kérnel gaussiano en función de σ (Elisfm, 2009b). 30
2.13. Hiperplano en SVR (Sayad, 2010). 31
2.14. Aplicando la función de kérnel en SVR (Sayad, 2010). 32
2.15. Entrenando modelos con validación cruzada (Rodriguez, 2018). 32

3.1. Diagrama para estrategia *comprar y mantener* simple. 36
3.2. Diagrama de Estrategia *comprar y mantener* con *bosques aleatorios*. 38
3.3. Configuración de Intérprete Python en PyDev 41
3.4. Bosque aleatorio: Evolución del precio de MSFT. 44
3.5. Evolución del portafolio en estrategia *comprar y mantener*. 46
3.6. Bosque Aleatorio: predicciones obtenidas. 47
3.7. Evolución del portafolio en estrategia *comprar y mantener* con *bosques aleatorios*. . 48

4.1. Diagrama de estrategia de *scalping*. 52
4.2. Diagrama de estrategia de *scalping* con uso de VaR. 58
4.3. Resultados de la estrategia de scalping. 67
4.4. Resultados de *scalping* con VaR histórico. 68
4.5. Resultados de *scalping* utilizando VaR calculado con SVR. 69
4.6. Comparación de las gráficas del valor acumulado obtenido en los experimentos de scalping. 71

Índice de tablas

2.1. Comparación de resultados, según Zuo y Kita (2012). 10

3.1. Bosque aleatorio: Valores de cierre de MSFT. 44
3.2. Bosque Aleatorio: Serie Temporal 44
3.3. Operaciones realizadas con la estrategia *comprar y mantener*. 46
3.5. Comparación de resultados entre los experimentos de *comprar y mantener*. . . 48
3.4. Operaciones realizadas en estrategia *comprar y mantener* con bosques aleatorios. 49

4.1. Primeras filas de DAX-201312.csv. 65
4.2. Matriz de datos. 66
4.3. Estadísticas de las operaciones realizadas mediante estrategia de scalping. . . 67
4.4. Estadísticas de las operaciones realizadas mediante la estrategia de *scalping* con VaR. 68
4.5. Estadísticas de las operaciones realizadas durante la ejecución de la estrategia de scalping con VaR calculado con SVR. 69
4.6. Comparación de resultados entre los experimentos de *scalping*. 70

Capítulo 1

Introducción general y objetivos

En este primer capítulo se abordarán las características del experimento, es decir, las motivaciones que han impulsado su desarrollo, la definición de objetivos que se pretenden alcanzar —lo cuales nos servirán como meta final— y para finalizar, se verá la estructura en que se dispone este documento. Todo ello servirá, por un lado, como introducción general para una mejor compresión de los experimentos, y por otro, para justificar todo el trabajo que se llevará a cabo.

1.1. Motivación

En 1537 en los Países Bajos, durante el gobierno de Carlos V, se puso en marcha un marco legislativo que dio apoyo a las transacciones financieras y comerciales. Cien años más tarde, en 1637 en Holanda, el mercado de tulipanes se transformó de un mercado estacional sobre algunos bulbos a una rueda de contratos de futuros que dieron lugar a la primera burbuja especulativa, conocida como Tulipomanía o Crisis de los Tulipanes (Mackay, 1841). Desde entonces, los mercados financieros han ido evolucionando hasta como los conocemos hoy, siempre con el objetivo de facilitar el flujo del dinero.

A diario se producen movimientos en los mercados financieros en los que el dinero de los inversores cambia de manos continuamente. Cuando un inversor cree que el precio de un activo va a aumentar, utiliza su dinero para comprar parte de ese activo, con la esperanza de poderlo vender cuando su precio haya aumentado. No obstante, en ese mismo momento, otro inversor puede creer que el precio de ese activo va a disminuir, por lo que realiza una acción de venta con la esperanza de comprar cuando el precio haya disminuido y embolsarse la diferencia. Uno de los dos inversores ganará dinero, mientras que el otro lo perderá irremediablemente. El dinero del que no haya acertado con su previsión del mercado pasará de su bolsillo al del bolsillo del que ha sido más acertado en su previsión. No existe la generación espontánea de dinero en los mercados. Si alguien gana, otro tiene que perder.

Cuando un inversor toma una decisión, debe hacerlo siempre en base a un conocimiento y

estudio previo del mercado, y de todo lo que pueda influir en él. Si, por el contrario, operara guiándose por cualquier criterio que no tuviera una base estudiada, lo mismo le daría utilizar su dinero en un casino que en un mercado concreto. Por fortuna o por desgracia, el dinero de estos inversores desinformados suele servir para enriquecer a los que sí poseen información sobre los movimientos de los precios de los activos disponibles para operar.

Existen numerosas formas de estudiar el vaivén de los precios de los mercados, pero ninguna ha demostrado ser eficaz en el cien por cien de los casos. No obstante, con las medidas de precaución adecuadas, se pueden limitar las pérdidas en los casos en los que la previsión no haya sido acertada, del mismo modo que existen técnicas para aumentar los beneficios en los casos en los que sí se haya acertado. Gracias a la información financiera de la que disponemos en nuestros días, así como de los precios históricos de los mercados, podemos realizar simulaciones con el fin de estudiar estrategias de inversión que nos permitan determinar si podemos conseguir beneficios mediante su uso, o por el contrario perderemos dinero. También podemos hacernos una idea del riesgo que tenemos que estar dispuestos a asumir y de las pérdidas máximas que podemos esperar.

Las técnicas de previsión de datos temporales que nos aporta el mundo del aprendizaje automático son una herramienta muy utilizada en los estudios de mercados actuales. Son la base de la mayoría de los productos financieros que pretenden atraer el dinero de los inversores que esperan aumentar su patrimonio de forma sencilla. Estos productos analizan los mercados de forma automática y generan previsiones mediante la aplicación de diferentes técnicas, algunas conocidas ampliamente —como el uso de medias móviles, cambios porcentuales de precios, etc.— y otras conseguidas mediante la inversión en investigaciones privadas. El rendimiento de estos procesos automáticos es el que diferencia unos productos financieros de otros. No obstante, no podemos olvidar que, por muy inteligente que sea un algoritmo, ninguno podrá anticiparse a las acciones humanas inesperadas que provocan grandes cambios en los mercados, previstos únicamente por aquellos que lo han provocado.

Debido al gran acercamiento del aprendizaje automático al análisis de datos financieros, no podemos dejar de intentar adentrarnos en este mundo con el objetivo de comprobar si efectivamente resultan de utilidad para aumentar el patrimonio, o por el contrario no es más que una estrategia de marketing rimbombante para que las entidades financieras absorban el dinero de inversores poco informados.

Antes de determinar si este estudio era viable se realizó una investigación inicial acerca de las posibilidades de utilizar algoritmos como método de inversión. Concluímos que podríamos realizar una aproximación siguiendo estrategias de inversión, es decir, reglas que establecen criterios concretos a tener en cuenta a la hora de manejar capital. Una vez conocidas y estudiadas se realizaron varios intentos de conseguir ejecutar esas estrategias de forma adecuada. Inicialmente se desarrollaron los experimentos con código en bruto, leyendo información de ficheros externos, e intentando procesar la información como si de un entorno real se tratara. No obstante, esta forma de trabajar era totalmente ineficiente y carecía de recursos o control

básico. No se podría haber justificado ningún resultado utilizando algo que podía ser manipulado facilmente para que devolviera lo que más nos podría haber convenido. Tras buscar otras posibles formas de analizar una estrategia de inversión, así como de determinar qué experimentos eran los más adecuados, se llegó a la conclusión de que la mejor forma de proceder era utilizar alguna herramienta destinada a tal fin.

Después de descubrir la existencia de herramientas que facilitan el diseño y ejecución de estrategias de inversión permitiendo obtener y manejar los datos de forma adecuada, ya era posible desarrollar experimentos que pudieran determinar si el uso de algoritmos de aprendizaje automático puede mejorar los resultados de estrategias de inversión. De esta forma no solo podríamos conseguir resultados de un experimento concreto, si no que podríamos sentar las bases para otros experimentos más avanzados, y facilitar la tarea de otros investigadores que quieran adentrarse en este tipo de estudios.

1.2. Objetivos

En este apartado se encuentran los objetivos establecidos en este proyecto que, en conjunto, pretenden marcar las pautas para el desarrollo que se abordará en los siguientes capítulos, y de esta forma conducirnos a la meta final que se desea alcanzar.

Más concretamente, los objetivos que se pretende conseguir son los siguientes:

- El objetivo principal de este Trabajo de Fin de Máster es realizar un estudio sobre técnicas de análisis de estrategias de inversión haciendo uso de algoritmos computacionales, así como de la aplicación de técnicas de aprendizaje automático para su posterior optimización.

- Como objetivo secundario pretendemos introducirnos en la generación de algoritmos capaces de tomar decisiones por sí mismos en un entorno de inversión financiera, que en un futuro podrá servirnos para desarrollar estrategias eficientes.

- Además, se pretenden sentar las bases para trabajos futuros propios que nos permitan obtener mejores resultados, así como allanar el camino para que otros investigadores se adentren en el mundo de la inversión algorítimca.

1.3. Metodología

Para la consecución de los objetivos anteriores, se deberá seguir un método de trabajo concreto. Éste comenzará reuniendo conocimientos suficientes sobre economía con el fin de hacer entender lo que estamos haciendo, ya que aunque el mundo de la economía y el de la informática se apoyan ambos en la misma base, que son las matemáticas, los conocimientos para moverse en cada campo son totalmente ajenos entre ellos. Es decir, primero reuniremos

los conocimientos básicos sobre el mundo de los mercados financieros, para posteriormente poder realizar algoritmos que nos permitan simular las decisiones que tomaría un inversor en tiempo real con la información de la que dispondría en cada momento. Una vez conocidas y escogidas las estrategias a implementar, deberemos utilizar las herramientas adecuadas que nos permitan realizar los experimentos eficientemente.

En general utilizamos *zipline*[1] por contar con las funciones necesarias para nuestros experimentos, permitiéndonos desarrollar los algoritmos de forma sencilla, centrándonos en el manejo de datos para probar las estrategias, cargando los datos de forma ágil y sabiendo que contamos con información suficiente para solucionar cualquier problema que podamos tener.

Además de *zipline*, utilizaremos las técnicas de aprendizaje automático adecuadas para la optimización de cada estrategia, generando la información suficiente como para comparar los diferentes resultados. Utilizaremos *ScikitLearn*[2] para el estudio de los datos y la obtención de estadísticas, así como para la ejecución de los algoritmos de aprendizaje automático.

Una vez conocidas las herramientas podemos hacernos una idea de cómo debemos diseñar los experimentos, así como de los pasos a seguir para realizar cada uno de ellos. No obstante, puesto que no se ha partido de ningún estudio que pudiéramos emular, debemos añadir que la metodología se ha definido en base a la propia experimentación, es decir, durante todo el estudio esta metodología ha sufrido cambios drásticos para adaptarse a las diferentes dificultades y necesidades que han surgido.

En definitiva, la metodología que mejor se adapta a los experimentos y a las herramientas seleccionadas consta de los siguientes pasos:

1. Desarrollo de la estrategia de inversión, organizando el código para que realice las operaciones acordes con las pautas establecidas para cada una.

2. Ejecución de la estrategia para obtener los resultados que se podrían conseguir en caso de no utilizar ningún algoritmo de aprendizaje automático auxiliar.

3. Generación de modelos predictivos a partir de análisis de datos históricos.

4. Optimización de la estrategia con los modelos desarrollados en el paso anterior, utilizando sus predicciones como eje de toma de decisiones.

5. Ejecución de la estrategia optimizada.

6. Comparación y análisis de resultados.

[1] https://www.zipline.io/
[2] *https://scikit-learn.org/stable/*

1.4. Estructura de la memoria

La memoria de este proyecto se estructura en los cinco capítulos. El primero expone lo necesario para entender el resto del documento, es decir, el contexto en el cual comienza este TFM, los motivos que lo impulsan y los objetivos que se adoptarán como meta para la correcta realización del proyecto.

El segundo reúne toda la información relevante relacionada con el mundo de los mercados financieros y el aprendizaje automático necesaria para entender los siguientes capítulos. Debido a la tangencialidad del mundo de la economía con el de la informática se intentará no profundizar demasiado en conceptos no necesarios, intentando explicar de forma clara lo que significa cada concepto, y para qué sirve cada uno.

En el capítulo tercero se experimentará con una estrategia de inversión a largo plazo sencilla para posteriormente optimizarla mediante un algoritmo de aprendizaje automático. En el primer apartado se diseñaran estas estrategias, para posteriormente implementarlas en un entorno adecuado para su ejecución. Finalmente se evaluarán los resultados y se compararan para comprobar si se ha conseguido mejorar la estrategia inicial.

En el capítulo cuarto se experimentará con tres desarrollos progresivos de una estrategia de inversión a corto plazo denominada *scalping* (reventa en castellano). El primer paso consistirá en la aplicación simple de la estrategia. Seguidamente se modificará la estrategia anterior utilizando un gestor de riesgo para ayudar en la toma de decisiones. Finalmente, se entrenará un modelo predictivo que nos permita optimizar el gestor de riesgo utilizado anteriormente, para tenerlo de nuevo en cuenta en una ejecución final de la estrategia. Al igual que el capítulo anterior, tras el diseño e implementación de las estrategias se procederá a la evaluación de los resultados y a comprobar si se ha conseguido optimizar realmente la estrategia mediante la utilización de aprendizaje automático.

Para terminar, en el último capítulo sintetizaremos el trabajo realizado a lo largo de todo el estudio, prestando especial atención a los objetivos descritos inicialmente y las expectativas planteadas y, por supuesto, en los resultados obtenidos, haciendo un balance de las tareas llevadas a cabo. Además, se hablará de posibles líneas de trabajo futuro describiendo posibles optimizaciones.

Capítulo 2

Estado del Arte

Después de la introducción al contenido que se ha realizado en el capítulo anterior, comenzamos este segundo capítulo con la intención de crear una visión global sobre el ámbito en el que los experimentos se han desarrollado.

2.1. Trading Algorítmico

El Trading Algorítmico se refiere al uso de algoritmos, reglas y procedimientos —automatizados en diferentes grados— para ejecutar operaciones de compra o venta de instrumentos financieros. Usualmente, los algoritmos utilizados en trading algorítmico pueden utilizar una o varias fuentes de datos o entradas. Por ejemplo, es común que una entrada, al menos, sea el precio actual del activo financiero sobre el cual se pretende hacer trading algorítmico. Además, pueden ser entradas del algoritmo las noticias publicadas en sistemas electrónicos en tiempo real. También es usual que el trading algorítmico utilice información respecto del estado o microestructura del mercado, tal como precios de apertura o cierre, índices de volatilidad, montos de variación mínima del precio y otros valores derivados del estado de los mercados. Toda esta información se utiliza con el fin de evitar incurrir en operaciones fallidas o errores en la ejecución (Halls-Moore, 2017).

La mayor dificultad de este campo es intentar predecir algo aparentemente impredecible, ya que en la mayoría de las ocasiones los movimientos más destacados se deben a operaciones realizadas por grandes inversores que se pueden permitir mover grandes cantidades de dinero con la intención de especular sobre estos mismos mercados, provocando reacciones emocionales en los inversores menos poderosos que les llevan a perder su dinero, para el beneficio de estas grandes manos. Sin embargo, muchos de estos movimientos pueden ser estadísticamente predecibles, y se puede estimar, dependiendo de cómo se ha movido el mercado reciente, la probabilidad de que un mercado suba o baje en un momento determinado, así como determinar las mejores posiciones de compra y venta para obtener el mayor beneficio posible, siempre de forma probabilística, es decir, con la menor posibilidad de fallo.

El uso de los llamados *stoploss* en las operaciones de mercadeo nos permite limitar la pérdida en caso de que una operación estadísticamente positiva se vuelva negativa, vendiendo o comprando si las pérdidas llegan hasta cierto nivel. De esta forma, en cada momento y para cada decisión de compra o venta, el algoritmo estimará cual puede ser la mejor opción de salida en caso de que el mercado no se comporte de la forma esperada. De igual modo, una vez determinado el momento de entrada, el algoritmo debe estimar el mejor momento de salida para que la diferencia de valor sea máxima.

Una de las principales ventajas del trading algorítmico es que, al estar guiado por un conjunto de reglas o procedimientos que no involucran la emocionalidad humana, permiten evitar los sesgos conductuales más comunes entre los inversores, como son el exceso de confianza, los sesgos heurísticos, la aversión a la incertidumbre y la aversión al riesgo, entre otros, identificados por el campo de las finanzas conductuales (Hens y Meier, 2016).

2.1.1. Análisis y Predicción de Valores de Mercados

Si realizamos la búsqueda "stock market research" en Google, obtendremos aproximadamente 1,300 millones de resultados. Esto significa que, sin duda, es un tema demandado por los investigadores, ya que un algoritmo correctamente entrenado en este campo, capaz de analizar datos históricos y predecir valores futuros, podría generar ganancias económicas importantes. Sin embargo, como veremos a continuación, la mayoría de los estudios se centran en la predicción de los comportamientos de los mercados, que en la práctica tienden a ser impredecibles, produciéndose subidas y bajadas muy repentinas en momentos inesperados, los cuales pueden provocar los cierres de cuentas y la bancarrota de los usuarios. Una teoría que contradice la predicción de valores en los mercados es la *Hipótesis del Mercado Eficiente* (Fama, 1970), que establece que los precios de los activos reflejan toda la información disponible y que, por tanto, es imposible vencer al mercado de forma consistente, ya que los precios solo deberían reaccionar a nueva información. De esta forma, si alguien consiguiera alguna ventaja tras analizar el histórico de información de un valor, el mercado reaccionaría a esta ventaja, y como resultado, el precio de ese mercado se corregiría. No obstante, aunque sea una teoría generalmente aceptada, muchos investigadores continúan intentando generar modelos cada vez más complejos del sistema financiero.

Por suerte, todos estos modelos pueden probarse en modo simulado con datos reales, por lo que se han podido comprobar los resultados estadísticos de los diferentes métodos y aproximaciones. La mayoría de estos modelos utilizan algoritmos de aprendizaje automático como SVMs, árboles aleatorios, redes neuronales, kNN o redes bayesianas. A continuación presentamos algunos ejemplos que han desarrollado experimentos similares a los que pretendemos realizar a lo largo de este documento.

2.1.1.1. Análisis de movimientos del mercado

En la investigación llevada a cabo por Zuo y Kita (2012), se utiliza el algoritmo K2 (Cooper y Herskovits, 1992) —diseñado para la construcción de redes bayesianas— con el fin de predecir las subidas y bajadas del análisis diario de mercados de valores, comparando los resultados con otros algoritmos. Concretamente, el estudio se realizó durante los resultados obtenidos para el índice *Financial Times Stock Exchange 100* (FTSE100). Se utilizaron los datos desde enero de 2005 hasta diciembre de 2006 para la generación de una red bayesiana, con la intención de predecir los resultados del periodo entre enero de 2007 y diciembre de 2007.

Para ello, se generó un grafo determinado por el algoritmo ilustrado en la figura 2.1.

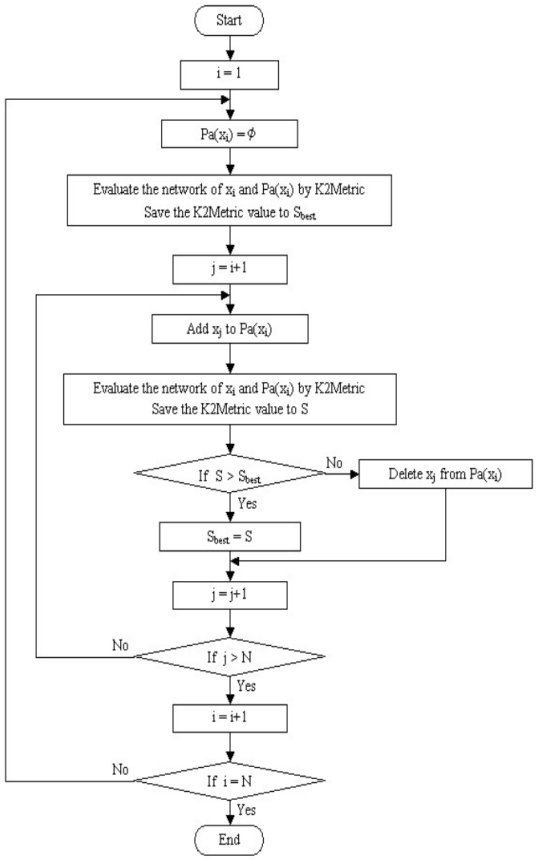

Figura 2.1: Algoritmo de búsqueda K2 (Zuo y Kita, 2012).

La red bayesiana está determinada, según el algoritmo K2, por el conjunto de medidas de subidas y bajadas de índices. El orden total de las relaciones entre las variables aleatorias se define mediante un orden temporal como el que podemos ver en la figura 2.2.

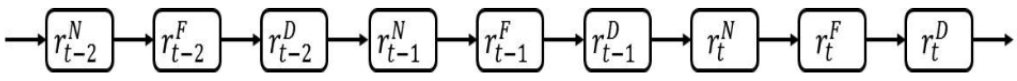

Figura 2.2: Orden total de relaciones entre tasas de subidas y bajadas (Zuo y Kita, 2012).

Una vez generada la red bayesiana, la probabilidad de la tasa de subidas y bajadas del FTSE100 en el día siguiente se define como $P(r^F_{t+1} \mid B)$. Por lo tanto, la decisión de mejora del FTSE100 en el día siguiente se decide de la siguiente forma:

$$r^F_{t+1} = \begin{cases} 0 & \text{si } P(r^F_{t+1} \mid B) < 0{,}5 \\ 1 & \text{si } P(r^F_{t+1} \mid B) \geq 0{,}5 \end{cases}$$

Finalmente, se obtiene la red bayesiana de la figura 2.3.

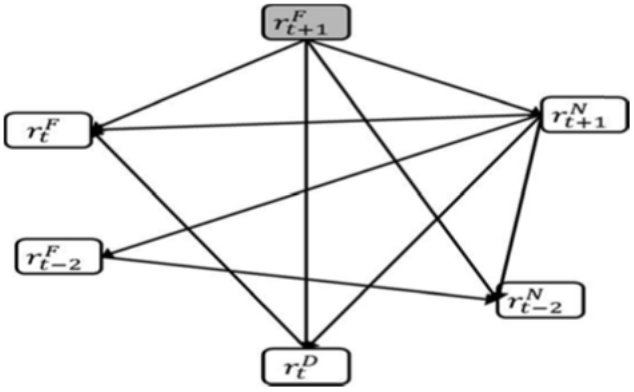

Figura 2.3: Red bayesiana para la predicción de las subidas y bajadas de FTSE100 (Zuo y Kita, 2012).

Una vez generada, se utiliza para predecir el comportamiento del mercado de FTSE100 durante el año 2007, comparando los resultados con otros algoritmos de metodologías distintas como la línea psicológica, o la estimación de tendencias, obteniéndose el resultado de la tabla 2.1.

		Number of tradings	Correct answer rate	Total profit
Psychological line	Case P1	54	51.85 %	282.3 GB £
	Case P2	22	59.09 %	534.1 GB £
Trend estimation	Case T1	253	51.38 %	19.9 GB £
	Case T2	253	50.59 %	-91.7 GB £
Bayesian network		249	61.44 %	5139.7 GB £

Cuadro 2.1: Comparación de resultados, según Zuo y Kita (2012).

Cómo podemos observar, el algoritmo basado en redes bayesianas supera considerablemente en porcentaje de acierto a los demás algoritmos con los que se ha comparado. A pesar de que en algunos casos los resultados son similares, debemos fijarnos en la columna del beneficio total para comprobar que puesto que, se han realizado más operaciones con mayor acierto, el beneficio ha sido muy superior.

2.1.1.2. Análisis de influencias

En otros estudios más complejos se busca generar modelos que pretendan predecir los movimientos del mercado en base a índices externos y en la influencia que estos pueden influir en los valores a estudiar. Por ejemplo, en el estudio de Khorram, Ph.D. y Hui (2011) se introduce el aprendizaje de redes bayesianas a partir de datos como un modelo aplicable para representar cambios en los mercados de valores. Para ello utiliza un extracto de los movimientos registrados en el índice FBM100. Concretamente, se utiliza un modelo bayesiano para el diagnóstico de las diferentes variables escogidas, que posteriormente generará la red bayesiana que se podrá utilizar para predecir los probables precios de los valores en el día.

En este caso, la metodología utilizada se compone de los siguientes cuatro pasos:

1. Preprocesado de los datos. Se categorizó cada mercado dependiendo de su dominio, utilizando como categorías los sectores formales existentes en el mercado de valores de Malasia.

2. Descubrimiento de relaciones entre variables utilizando la red bayesiana. Este paso se compone de dos partes. Una primera para aprendizaje estructural, que identifica la estructura de la red casual, y una segunda que obtiene los parámetros de una tabla de probabilidad condicional para cada nodo. Estas tablas serán las que se utilizan como criterio para juzgar la posibilidad de la influencia de un nodo sobre otro.

3. Diagnosis del modelo de valores.

4. Comprobación de la precisión del modelo frente a datos reales.

Como resultado del paso 2, se obtiene la red bayesiana de la figura 2.4.

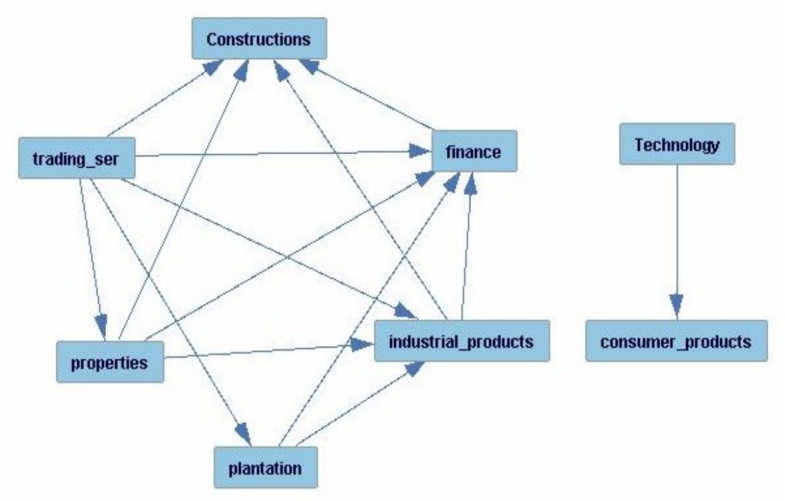

Figura 2.4: Red bayesiana aprendida (Khorram, Ph.D. y Hui, 2011).

Puesto que las relaciones entre las variables son muy complejas, no se pudo conseguir una estructura de grafo acíclico dirigido en el modelo aprendido, por lo que se intentó utilizar el *manto de Markov* (en inglés *Markov blanket*) (Pearl, 1988) para la predicción de cada nodo de forma separada. Sin embargo, no se consiguieron resultados precisos debido a las limitaciones existentes en la computación de los algoritmos basados en restricciones (Bartàk, 1999). La predicción tuvo un 52 % de media de precisión.

Como resultado, los autores concluyeron que este enfoque puede ser mucho más eficiente si la parte del aprendizaje de la estructura se mejora, aumentando el tamaño de la información y nutriendo a la red de nuevos datos sobre el comportamiento de los precios para incrementar la exactitud del modelo.

2.1.2. Conclusiones

Al igual que en los estudios analizados, nuestra idea es desarrollar algoritmos que puedan operar directamente en el mercado. También realizaremos un estudio previo de los datos históricos con la intención de encontrar un modelo predictivo que nos permita determinar cuál es la mejor decisión a tomar en cada momento. En nuestro caso no utilizaremos redes bayesianas, sino que utilizaremos algoritmos más sencillos de manejar en tiempo real, como son los bosques aleatorios y las máquinas de vectores de soporte. Podríamos haber optado por utilizar un mismo algoritmo para todos los experimentos, y de esa forma poder comparar los resultados del mismo en las diferentes pruebas. Sin embargo, uno de los objetivos de este análisis es utilizar diferentes formas de generar algoritmos predictivos, por lo que consideramos importante utilizar diferentes formas de predecir valores.

2.2. Conceptos Básicos de Trading

En este apartado vamos a describir algunos conceptos indispensables para entender el resto del documento y los experimentos. Los diferentes apartados y conceptos están estructurados de forma secuencial, de modo que para entender cada uno es necesario haber comprendido los anteriores.

2.2.1. Econometría

La econometría es la rama de la economía que hace un uso extensivo de modelos matemáticos y estadísticos, así como de la programación lineal y la teoría de juegos, para analizar, interpretar y hacer estimaciones sobre sistemas económicos, prediciendo variables como el precio de bienes y servicios, las tasas de interés, los tipos de cambio, las reacciones del mercado, el coste de producción, la tendencia de los negocios y las consecuencias de la política económica. En la elaboración de la econometría se unen la matemática, la estadística, la investigación

social y la teoría económica. El mayor problema con el que se enfrentan los económetras en su investigación es la escasez de datos, los sesgos que pueden presentar los datos existentes, los sesgos del propio investigador y la ausencia o insuficiencia de una teoría económica adecuada. Aun así, la econometría es la única aproximación científica al entendimiento de los fenómenos económicos (Durlauf y col., 2007).

2.2.2. Mercados Financieros

Un mercado financiero es el lugar, mecanismo o sistema en el cual se compra y vende cualquier activo financiero. La finalidad de un mercado financiero es poner en contacto a compradores y vendedores para que, mediante la realización de acciones de compra-venta, generen una oferta y una demanda de los diferentes activos financieros. De esta forma, si la oferta de un valor es mayor que la demanda su precio disminuirá y, por el contrario, si la demanda es superior a la oferta el precio aumentará. Por tanto, el precio vendrá determinado por la oferta y la demanda (López, 2005). Además del precio, los mercados financieros se caracterizan por su liquidez y la volatilidad media.

La liquidez asegura la capacidad de entrar y salir rápidamente del mercado. Una forma de comparar la liquidez de un conjunto de valores dado es utilizar el volumen promedio de negociación de cada valor. Un mercado con liquidez perfecta es aquel que tiene suficientes compradores y vendedores para negociar en cualquier momento del día de negociación.

La volatilidad a menudo es asociada a grandes oscilaciones en las tendencias del mercado. La volatilidad de un mercado se puede ver a través de su índice de volatilidad o VIX (CBOE, 2019), el cual proporciona al inversor una idea de como de lejos se puede desviar un precio de su valor medio. Se necesita una alta volatilidad para asegurar que los cambios en los precios exceden los costes de transacción. La volatilidad está altamente correlacionada con las noticias de tipo macroeconómico (es decir, las que afectan a indicadores globales de la economía como el desempleo, la economía internacional, la inflación, el crecimiento económico general, etc.), por lo que puede influir enormemente en los beneficios que se pueden obtener en un mercado, así como en las posibles pérdidas.

A continuación describiremos los mercados financieros más conocidos: renta fija, bonos, divisas, valores, materias primas y futuros (García, 2018).

Mercados de Renta fija

Estos mercados se refieren a cualquier tipo de inversión en la que el prestatario o el emisor está obligado a realizar pagos de una cantidad específica en un plazo determinado [1]; por ejemplo, algunas cuentas de ahorro o los fondos a plazo fijo ofertados por la mayoría de los bancos. Se caracterizan por una alta liquidez y una baja volatilidad.

[1] https://www.cnmv.es/Portal/Inversor/RentaFija-Riesgos.aspx

Mercados de Bonos o Valores de Deuda

Son mercados financieros donde los participantes pueden emitir nueva deuda, conocida como mercado primario, o comprar y vender valores de deuda, conocida como mercado secundario. Estos mercados forman parte de los mercados de créditos, los cuales también se componen de los préstamos bancarios.

Entre los tipos de bonos que se pueden negociar se encuentran las Letras del Tesoro, los Bonos del Estado y las Obligaciones del Estado, los cuales se caracterizan por su gran tamaño, liquidez y falta de riesgo financiero. Estos bonos facilitan la transmisión de capital por parte de los ahorradores a las organizaciones que requieren capital para proyectos gubernamentales o expansiones de negocio.

Mercados de Valores

Mercados en los que se emiten y negocian acciones, ya sea a través de bolsas o mercados bursátiles. Las acciones de una empresa son porcentajes de la misma; es decir, cuando compramos acciones de una empresa estamos comprando nuestra participación en la misma. Esta es una de las áreas vitales del mercado económico, porque permite a las compañías acceder a capital y a los inversores formar parte de la compañía, con la posibilidad de obtener ganancias si la empresa aumenta su valor.

Mercado de Divisas

El *Foreign Exchange Market* (Forex, FX, o mercado de divisas) es un mercado global descentralizado para el mercadeo de monedas. Un mercado descentralizado es aquel en el que existen varios precios para el mismo activo financiero. Determina el tipo de cambio, que incluye todos los aspectos de compra, venta e intercambio de divisas a precios actuales o determinados. En este caso existe la figura del creador de mercado, que se dedica a dar precios de compra y de venta, permitiendo comprar o vender en momentos determinados al precio del mercado (p. ej., comprar a 190 y vender a 200). Es con diferencia el mayor mercado del mundo, seguido del mercado de valores.

Mercados de Materias Primas

Mercados que negocian en productos de economía primaria en lugar de en productos manufacturados. Conviene distinguir entre las materias primas agrícolas —entre las que se encuentran el trigo, el café, el cacao, la fruta, el azúcar, etc.— y las materias primas duras, como el oro y el petróleo.

Mercados de Futuros

Los mercados de futuros nacieron originalmente en el siglo XIX con la denominación de "forwards" o "mercados a futuro", con el objetivo de proteger a los productores de materias primas en un mercado caracterizado por épocas de concentración de la oferta (de cosecha) y por precios muy variables a lo largo del año, que restaban atractivo a la labor.

Las consecuencias de estos contratos fueron evidentes. Por ejemplo, supongamos un pacto de compra de un kilo de maíz a 50 céntimos con fecha de vencimiento del 31 de marzo de 2012. Se supone que al llegar a esta fecha se deberá pagar lo acordado, pero pueden pasar varias cosas:

1. El precio es muy similar al pactado, en cuyo caso no debería haber mayores problemas.
2. El precio real a esa fecha es inferior al pactado, por lo que el compromiso del pacto obligará a pagar el kilo a un importe mayor del real.
3. El precio es superior al pactado y por tanto, una vez comprado, se puede vender el kilo de maíz en el presente obteniendo un beneficio de la operación.

El vendedor también corre riesgos, evidentemente, aunque en cualquier caso este riesgo es compartido por ambos y los incumplimientos pueden venir de ambos lados.

Se necesita una gran confianza entre las partes, pues si antes del vencimiento se ve una evolución de precios poco favorable, es necesaria la cooperación de ambas partes para adelantar la entrega, o para tomar algún tipo de medida para reducir el impacto de la variación de precios, antes de que se rompa el acuerdo.

Todo esto ha hecho que los "mercados a futuro" pasen a ser "mercados de futuro", es decir, que se introduzcan en mercados organizados que traten de garantizar las condiciones de la negociación y el cumplimiento de los contratos. Esto se realiza en gran parte gracias a las denominadas "cámaras de compensación".

Los mercados a futuro no solo se realizan en torno a las materias agrícolas sino también para activos financieros, minerales, divisas, índices bursátiles, etc., y la liquidación no tiene por qué realizarse al vencimiento. Antes de dicha fecha, el inversor puede hacer una liquidación adelantada realizando operaciones a futuro inversas, es decir, vendiendo cuando se es comprador o viceversa. De esta manera se intenta minimizar el impacto de las fluctuaciones de precio, algo que puede requerir una atención continua a la evolución de los mismos. Esto es lo que se denomina "realizar operaciones en corto" (BBVA, 2015).

2.2.3. Índice bursátil

Un índice bursátil es un índice de referencia que se forma con un conjunto de valores cotizados en una bolsa de valores. Sirve para representar la evolución de las empresas de un país, un determinado sector de la economía o un tipo de activo financiero (Cardoso y Covarsí,

2002). Los índices bursátiles que aglutinan las principales empresas de un país son un excelente indicador de la economía. Por ejemplo, cuando se dice que la bolsa española baja un cierto porcentaje, se suelen referir a que está bajando el índice bursátil español, es decir, el IBEX 35, formado por las 35 empresas españolas más grandes.

Entre los principales índices bursátiles del mundo podemos destacar:

- IBEX 35: Índice ponderado que incluye a las 35 empresas españolas con más liquidez que cotizan en el Sistema de Interconexión Burstátil Español.
- Promedio Industrial Dow Jones: También denominado *Dow Jones Industrial Average* (abreviado DJIA o Dow-30, informalmente Dow Jones o Dow) que mide el desempeño de las 30 mayores sociedades anónimas que cotizan en el mercado bursátil de Estados Unidos.
- Nasdaq 100: Recoge a los 100 valores de las compañías más importantes del sector de la industria de la tecnología —incluyendo empresas de hardware y de software, las telecomunicaciones, venta al por menor/por mayor y biotecnología— cotizadas en el *Nasdaq Stock Market*. En el índice pueden estar tanto empresas estadounidenses como de otros países.
- Standard&Poor's 500: Se basa en la capitalización bursátil de 500 grandes empresas que poseen acciones que cotizan en las principales bolsas de Wall Street, capturando aproximadamente el 80 % de toda la capitalización de mercado en Estados Unidos. Se considera el índice más representativo de la situación real del mercado (Kenton, 2020).
- DAX 30: Índice ponderado con las 30 compañías más grandes de Alemania que cotizan en la Bolsa de Fráncfort.

2.2.4. High-Frequency Trading

El mercadeo de alta frecuencia (*High Frequency Trading, HFT*) es un método para realizar trading que utiliza algoritmos informáticos para analizar varios mercados al mismo tiempo y realizar transacciones de compra-venta en base a los resultados obtenidos. Tiene lugar desde 1999 cuando la *Securities and Exchange Commission de Estados Unidos* (*SEC*) autorizó los intercambios electrónicos. Se popularizó en el año 2010 cuando diferentes estudios (CFTC y SEC, 2010) concluyeron que algoritmos ejecutando estas prácticas contribuyeron a aumentar la volatilidad en la jornada conocida como *Flash Crash*, durante la cual el índice *Dow Jones Industrial Average* sufrió su mayor caída intradía de la historia, para inmediatamente recuperar la mayor parte de las pérdidas en cuestión de minutos (Grant, 2010).

Mientras que el 70 % de los inversores de baja frecuencia suelen perder dinero, la mayoría de los algoritmos HFT obtienen retornos positivos. Incluso en los peores meses de la crisis

de 2008, el 50 % de las posiciones abiertas estaban relacionadas con este tipo de métodos (Duhigg, 2009).

Los mercados adecuados para el HFT deben cumplir ciertas características como alta liquidez y volatilidad.

2.2.5. Estrategia de trading

Una estrategia de trading es un plan fijado diseñado para obtener retornos positivos en la compra venta de activos en los mercados financieros. Existen dos tipos diferentes de estrategias: las que se basan en tendencias y las basadas en el marco temporal.

2.2.5.1. Estrategias según tendencia

En estados de normalidad los precios se suelen mantener dentro de un rango concreto. Cuando los precios rompen estos rangos, podemos hablar de un cambio de tendencia, donde los precios alcanzan nuevos mínimos y máximos. Las estrategias de inversión según tendencia buscan distinguir cuando la tendencia ha comenzado para entrar en el mercado y salir cuando se cree que se han alcanzado esos nuevos valores. Para conseguir discernir entre un movimiento normal del mercado y un cambio de tendencia los inversores utilizan diferentes especulaciones:

Ruptura de rangos establecidos

En esta estrategia los rangos establecidos son escogidos por el propio analista, utilizando para ello criterios propios o generalizados, como el valor máximo en un determinado número de días, los valores de apertura y cierre en un momento concreto en el que el volumen de operaciones ha sido alto, etc. Por ejemplo, cuando el precio rompe por encima del máximo de 20 días podríamos estar hablando de un inicio de tendencia alcista, y por tanto sería un indicio de compra en caso de que en nuestro planteamiento lo hayamos establecido así bajo nuestro propio criterio.

Cruce de medias móviles[2]

Para utilizar este tipo de señales se crean dos medias, una con periodo de tiempo corto y otra con un periodo más largo, de tal forma que cuando los valores de estas medias se cruzan podríamos hablar de un cambio de tendencia. Por ejemplo, si la media móvil corta se mueve por encima de la media móvil larga, significa que los precios más recientes son mas altos que los antiguos y por tanto podría sugerir que estamos ante una tendencia alcista.

[2]Una media móvil es un valor calculado de manera continua, como la media aritmética del precio de un valor durante un periodo de tiempo específico.

Bandas de Bollinger

Estrategia similar a la anterior, pero utilizando unos indicadores diferentes. Las *bandas de Bollinger* consisten en un conjunto de tres curvas dibujadas en relación con los precios de los activos. La banda del medio es una medida de la tendencia de plazo intermedio, normalmente un promedio móvil simple, que sirve como base para las bandas superior e inferior. El intervalo entre las bandas superior e inferior y la banda intermedia está determinada por la volatilidad, típicamente la desviación estándar de los mismos datos que se usaron para el promedio. Los parámetros t períodos y r desviaciones estándar pueden ajustarse según los propósitos. Como las bandas representan la volatilidad del mercado, cuanto mayor sea la brecha entre las bandas, más volátil será el mercado.

$$BB_m = \frac{1}{t}\sum_{i=1}^{t} precio_i \tag{2.1}$$

$$BB_{sup} = BB_m + r * \sqrt{\frac{\sum_{i=1}^{t}(precio_i - BB_m)^2}{t-1}} \tag{2.2}$$

$$BB_{inf} = BB_m - r * \sqrt{\frac{\sum_{i=1}^{t}(precio_i - BB_m)^2}{t-1}} \tag{2.3}$$

Las fases de tendencia se inician principalmente por la ruptura del rango anterior, materializado por un precio de cierre fuera de una de las dos bandas externas, muy por encima o por debajo de los movimientos recientes. En la Figura 2.5 podemos ver una gráfica en que las bandas de Bollinger señalan cambios de tendencia (Bollinger, 2002).

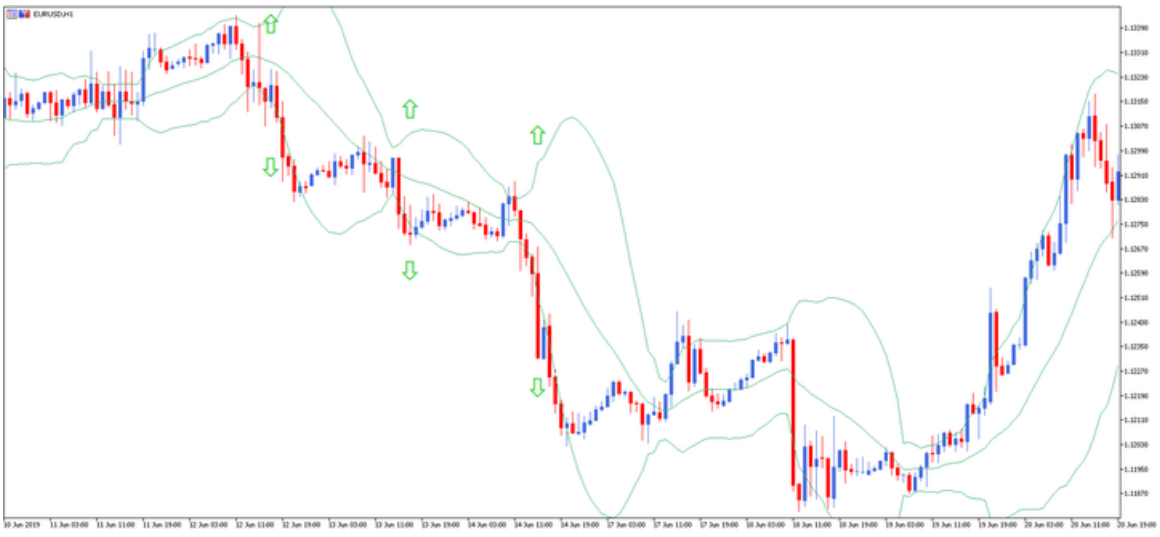

Figura 2.5: Identificación de cambios de tendencia mediante bandas de Bollinger. Fuente: EURUSD, Gráfico D1, MT5 Admiral Markets. Rango de datos: del 10 de junio de 2019 a 20 de junio de 2019. Realizado el 29 de agosto de 2019.

2.2.5.2. Estrategias según el marco temporal

Scalping

En una estrategia de *scalping* ("reventa", en castellano) se realizan un gran número de operaciones generando pequeños beneficios de entre 5 y 10 puntos. Los inversores que utilizan esta estrategia mantienen las posiciones de uno a cinco minutos. Un inversor famoso que utiliza esta estrategia es Paul Rotter, el cual realizaba entre 200.000 y 300.000 operaciones diarias. Antes de que Eurex[3] tomara medidas para penalizar las cancelaciones masivas, este inversor introducía órdenes de elevado volumen, con la intención de que otros inversores las vieran y respondieran, y las cancelaba inmediatamente. Actualmente Rotter lleva las cuentas de clientes con un mínimo de un millón de euros (Kimura, 2020).

Esta estrategia es la más utilizada por inversores pequeños, ya que es menos propensa al riesgo. Además, aplica el concepto del HFT (*high-frequency trading*) realizando varias operaciones al día. Entre sus principales ventajas podríamos destacar que permite establecer objetivos aceptables y alcanzables a corto plazo.

Existen diferentes formas de estrategias de *scalping*:

- Aplicando solo un indicador, como la media móvil, la desviación estándar, etc. Entre sus desventajas más importantes destaca la generación de excesivas falsas señales, por centrarse únicamente en un indicador. Se pierde la vista futura de la tendencia y trabaja con una ventana temporal reducida. En la figura 2.6 podemos observar en color azul el valor de Facebook y en color rojo el valor de una media móvil.

- Utilizando medias móviles y otro indicador como las bandas de Bollinger operando según la propia teoría de las bandas o los mencionados en el punto anterior: En la figura 2.7 hemos utilizado dos medias móviles de distinto tamaño y las bandas de Bollinger. Utilizando estos indicadores, podríamos haber realizado una operación como la que podemos observar en la figura 2.8, ya que una vez se han cruzado las medias móviles, y según nuestra estrategia, podemos esperar a que el valor vuelva a las bandas, y una vez supere el valor de la banda inferior, esperar que al menos retome el valor medio calculado en ese momento. Como se puede observar, se abre la posición cuando el precio vuelve a subir por encima de la banda de Bollinger inferior, y se establece como objetivo de venta la media móvil de ese mismo instante. También se establece un stop o pérdida aceptada en caso de que la operación no se comporte como se espera.

- Operando con soportes y resistencias: Un *soporte* es un punto donde el precio deja de disminuir, mientras que una *resistencia* es un punto donde el precio deja de aumentar. En caso de que el precio llegara a un soporte, abriríamos una operación de compra, mientras que si tocara una resistencia abriríamos una operación de venta.

[3] Una de las principales cámaras de liquidación de derivados financieros futuros, así como el mayor mercado europeo de derivados financieros, con sede el Alemania.

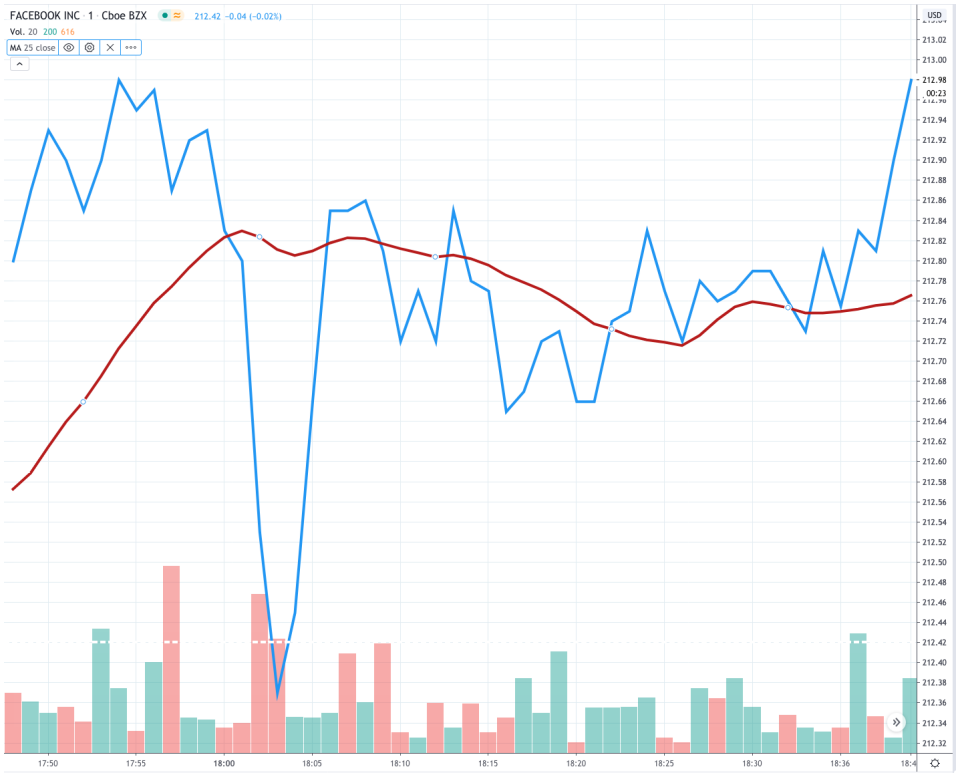

Figura 2.6: Valor de la cotización de Facebook con una media móvil.

Trading Intradía

En el trading intradía se compran y venden múltiples activos dentro de un mismo día de negociación, intentando salir de cualquier operación pendiente al final del día. Los marcos temporales de operación para este tipo de inversores suelen ir entre los 15 minutos y las 4 horas.

Swing Trading

En este caso la estrategia se centra en comprar y vender valores con el propósito de mantenerlos durante varios días y/o semanas. Este tipo de estrategias se valen de análisis de tendencias con datos a nivel diario.

Comprar y mantener (Buy & Hold)

Se compran acciones y se mantienen durante un largo período, independientemente de las fluctuaciones en el mercado. Un inversor que utiliza esta estrategia selecciona activamente las inversiones, pero no le preocupan los movimientos de precios a corto plazo ni los indicadores técnicos. Muchos inversores legendarios, como Warren Buffett y Jack Bogle, elogian este enfoque como ideal para las personas que buscan retornos saludables a largo plazo.

Esta estrategia se basa en la opinión de que a largo plazo los mercados financieros dan una buena tasa de rendimiento incluso teniendo en cuenta un cierto grado de volatilidad. *Comprar*

Figura 2.7: Valor de la cotización de Facebook con dos medias móviles y bandas de Bollinger.

y mantener establece que los inversores nunca verán tales retornos si venden sus acciones después de una caída. Este punto de vista sostiene que sincronizarse con el mercado (es decir, el concepto de que uno puede ingresar al mercado en los mínimos y vender en los máximos) no funciona; intentar esa sincronía da resultados negativos, al menos para inversores pequeños o poco sofisticados, por lo que es mejor para ellos simplemente *comprar y mantener* (Malkiel, 1996).

2.2.5.3. Tipos de Inversor

En los mercados podemos encontrar tantos tipos de inversores como diferentes personalidades se pueden encontrar manejando carteras de inversión. No obstante, existen tres tipos de categorías de inversor generales en las que se puede englobar a cualquiera de ellos: inversores informados, inversores de valor e inversores desinformados.

Inversores informados

Obtienen información privada que les permite predecir futuros cambios en un activo. Suelen ser impacientes y entran al mercado con órdenes limitadas cercanas al precio de mercado. Tienen un alto impacto en el mercado.

Figura 2.8: Operación de compra realizada siguiendo una estrategia basada en bandas de Bollinger.

Inversores de valor Son aquellos que utilizan modelos predictivos y los afinan mediante el uso de indicadores para intentar determinar el verdadero valor de un activo.

Inversores desinformados

No tienen información privilegiada ni predicciones sobre el verdadero valor de un activo. Se especializan en elaborar estrategias que siguen el impulso de los precios a muy corto plazo para intentar capturar los mejores precios.

2.2.5.4. Datos Financieros

La información financiera se presenta en diferentes formas. Existen cuatro tipos esenciales de información financiera:

Datos fundamentales

Incluye información que se puede encontrar en presentaciones regulares e informes comerciales. Son principalmente datos contables informados trimestralmente. Está extremadamente regularizada y tiene baja frecuencia. Al ser tan accesible para el mercado, es poco probable que haya algún valor por explotar, pero en general es muy útil en conjunción con otros tipos de datos.

Datos de Mercado

Toda la información referente a una operación de compra-venta. Cada participante del mercado deja una huella característica en los registros comerciales, y con suficiente paciencia, se puede encontrar la manera de anticipar el próximo movimiento de los competidores.

Datos Analíticos

Son datos derivados basados en una fuente original que ha sido procesada de una manera particular. Los bancos de inversión y las firmas de investigación venden información de resultados de análisis internos de compañías, modelos de negocio, etc.

Datos Alternativos

Pueden ser, por ejemplo, imágenes de satélite o fuentes de vídeo que muestran monitorización de petroleras, actividad de tráfico, ocupación de aparcamientos, etc. Lo que caracteriza a este tipo de información es que no ha sido procesada en otro tipo de datos. Ofrece la oportunidad de trabajar con datos únicos y con conjuntos de datos difíciles de procesar, convirtiéndolo en el tipo de datos más prometedor.

Estructuras de Datos Financieros

Las oportunidades de compra-venta suelen ser una función de los datos que los identifica. Cuanto mayor sea la frecuencia de los datos, mayor será la posibilidad de encontrar oportunidades. Para poder utilizar información desestructurada en mecanismos de aprendizaje automático deberemos saber cómo convertirlos, extraer información de valor y almacenar los resultados en una forma estandarizada.

La mayoría de los algoritmos de aprendizaje automático asumen una representación tabular de los datos. Los profesionales de las finanzas a menudo se refieren a las filas de esas tablas como barras. Algunos métodos de construcción de barras son muy populares en la industria financiera, en la medida en que la mayoría de las APIs de proveedores de datos ofrecen varios de ellos.

2.2.6. Valor en riesgo

El manejo del riesgo es tan importante como las señales de los indicadores financieros para manejar las operaciones. Los principales objetivos del manejo del riesgo son:

- Encontrar la fuente del riesgo. Este puede provenir de la propia definición del mercado, de su volatilidad, etc.
- Establecer un riesgo aceptable, así como un retorno apropiado para el riesgo asumido.

- Encontrar el riesgo mínimo para una determinada cantidad de retorno esperado.
- Establecer el plan de acción, es decir, qué debemos hacer en caso de que el riesgo disminuya o aumente y existan oportunidades de entrada. Estas acciones pueden estar relacionadas con el valor del *stop loss* de una operación o con el nivel de apalancamiento (endeudamiento para realizar operaciones de mayor importe) que podemos asumir.

El Comité de Supervisión Bancaria de Basilea (2004) categoriza el riesgo de diferentes formas, y cada una de ellas requiere de un procedimiento de medida del riesgo distinto. Algunos ejemplos de medidas de riesgo son:

- Riesgo de mercado.
- Riesgo de crédito.
- Riesgo de liquidez.
- Riesgo operacional.
- Riesgo legal.

El riesgo de mercado tiene un efecto directo en el funcionamiento de los algoritmos de compra-venta. Existen diferentes técnicas para medir el riesgo según los datos que utilicemos. De esta forma podríamos calcular el riesgo mediante modelos estadísticos, análisis de escenarios concretos, modelos causales, etc. Cada una de estas técnicas tiene sus propias ventajas e inconvenientes.

El valor en riesgo (*Value at Risk, VaR*), es una técnica estadística para medir el riesgo. Se define generalmente como la pérdida máxima posible de un determinado activo en un periodo concreto con un nivel de confianza establecido, normalmente 95 % o 99 % (Yamai y Yoshiba, 2002).

Existen tres formas de calcular el valor en riesgo:

- VaR paramétrico: utiliza datos de rentabilidad estimados y asume una distribución normal de la rentabilidad.
- VaR histórico: utiliza datos históricos.
- VaR por Monte Carlo: Mediante un algoritmo se generaran cientos o miles de posibles resultados según los datos iniciales introducidos.

Para nuestro estudio utilizaremos el VaR histórico por ser la forma más sencilla para calcularlo. Se calcula distribuyendo de forma normal las pérdidas y las ganancias obtenidas. De esta forma el VaR se definiría como el percentil 100α, tomando como nivel de confianza $100(1-\alpha)$, como podemos ver en la figura 2.9.

Algunos autores aseguran que el VaR por si solo tiene algunos problemas conceptuales. Artzner y col. (1997) enumera las siguientes deficiencias:

- El VaR mide solo los percentiles de las distribuciones de pérdidas y ganancias y, por lo tanto, ignora cualquier pérdida más allá del nivel de VaR.

- No es coherente porque no es subaditivo. [4]

Para solventar estos problemas, el mismo Artzner y col. (1997) propone utilizar lo que denomina la pérdida esperada (del inglés *expected shortfall, ES*) que se define con la siguiente ecuación 2.4, y podemos apreciar en la figura 2.9.

$$ES_\alpha(X) = E[X \mid X \leq VaR_\alpha(X)] \tag{2.4}$$

Siendo X una variable aleatoria que denota las pérdidas y las ganancias, $VaR_\alpha(X)$ el VaR con un nivel de confianza de $100(1-\alpha)$ y $E[X \mid B]$ la expectativa condicional de la variable aleatoria X dado un evento B.

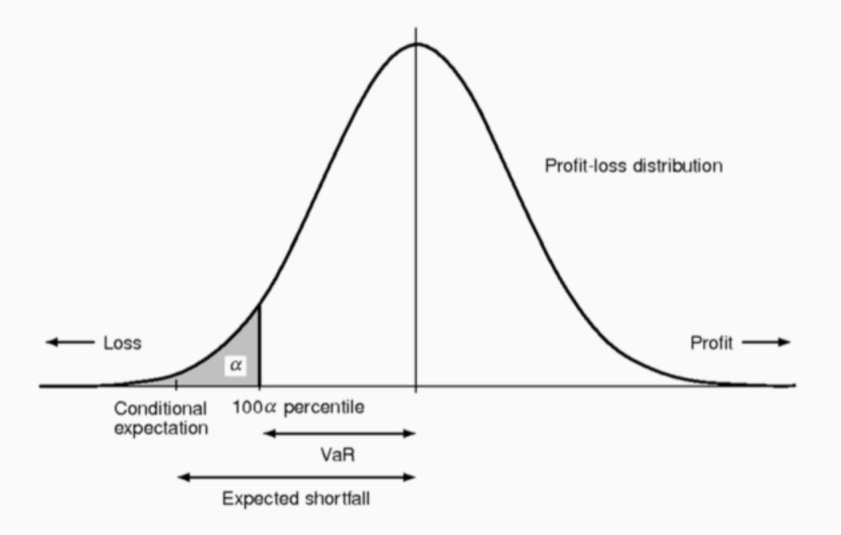

Figura 2.9: Distribución normal de la pérdida y la ganancia (Yamai y Yoshiba, 2002).

Debemos tener en cuenta que las fórmulas que hemos visto hasta ahora están orientadas a un uso en portafolios de activos tomando como pérdidas y ganancias los cambios de valor del portafolio entre días.

[4] Una medida de riesgo ρ es subaditiva cuando el riesgo de la posición total es menor o igual a la suma del riesgo de las carteras individuales.

La subaditividad se puede definir de la siguiente manera. Sean X e Y variables aleatorias que denoten las pérdidas de dos posiciones individuales. Una medida de riesgo ρ es subaditiva si se satisface la siguiente ecuación:

$$\rho(X+Y) \leq p(X) + p(Y)$$

2.3. Conceptos de Aprendizaje Automático

2.3.1. Bosques Aleatorios

El algoritmo de clasificación denominado *bosque aleatorio* utiliza muchos árboles de clasificación. Para clasificar un nuevo objeto a partir de un vector de entrada, se coloca el vector de entrada debajo de cada uno de los árboles en el bosque. Cada árbol da una clasificación, y cada árbol "vota" para esa clase. El bosque elige la clasificación que tiene más votos sobre todos los árboles en el bosque.

Según Breiman (2001), los bosques aleatorios son una combinación de predictores de árboles, de modo que cada árbol depende de los valores de un vector aleatorio muestreado de forma independiente y con la misma distribución para todos los árboles en el bosque. El error de generalización para los bosques converge hasta un límite a medida que aumenta el número de árboles en el bosque. Este error depende de la fuerza de los árboles individuales en el bosque y la correlación entre ellos. Las estimaciones internas supervisan el error, la fuerza y la correlación, y se utilizan para mostrar la respuesta al aumento del número de características utilizadas en la división. Las estimaciones internas también se utilizan para medir la importancia de cada variable (Freund y Schapire, 1996).

El bosque aleatorio no solo testea subconjuntos de datos, sino que también testea subconjuntos de características y deja que cada modelo entrene en un conjunto diferente de características. Puede usarse como una forma de medir la importancia de las características ordenandolas por su importancia y luego utilizar el percentil superior de la característica para entrenar otro modelo de bosque aleatorio que esté especializado en inferencia.

Sin embargo, para el contexto que nos ocupa, aplicaremos estas ideas para analisis de regresión, es decir, predicción de valores futuros (Segal, 2004). Por ejemplo, puesto que en los conjuntos de datos tenemos los valores de apertura, cierre, máximo y mínimo de un determinado activo, podemos utilizar estos datos para predecir su valor futuro. Puesto que la meta de la regresión es obtener la recta o el hiperplano que mejor describa un patrón de datos concreto, consideramos, a priori, que podrá sernos de utilidad.

La regresión lineal parte de dos variables, una independiente x, que en nuestro caso sería el tiempo, y otra dependiente y, que para nosotros sería el precio del activo. La idea entonces es tratar de encontrar una función lineal de x2.5 que nos permita aproximar y mediante una recta o hiperplano.

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \tag{2.5}$$

Para ello se hace uso también de la función 2.6que calcula la suma residual de cuadrados para obtener los valores de los coeficientes.

$$SSE = \sum_{i}^{n} (y_i - f(x_i))^2 \tag{2.6}$$

En el caso de bosques aleatorios esta función se entiende mejor en 2.7, ya que se calcula teniendo en cuenta el error de la rama izquierda y derecha de un nodo dado.

$$SSE = \sum_{izq}(y_i - f_{izq}(x_i))^2 + \sum_{der}(y_i - f_{der}(x_i))^2 \tag{2.7}$$

Gracias a la clase *RandomForestRegressor* [5] del paquete *Scikit-learn* podremos hacer uso de este regresor de forma sencilla con sus valores por defecto.

2.3.2. SVMs

Las máquinas de vector de soporte pertenecen a la familia de los clasificadores lineales puesto que inducen separadores lineales o hiperplanos en espacios de características de muy alta dimensionalidad. Estos hiperplanos sirven para separar las clases en dos espacios lo más amplios posibles.Llamamos vector de soporte al vector formado por los puntos más cercanos al hiperplano. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, pueden ser clasificadas en una clase o la otra. Una buena separación entre las clases permitirá una clasificación correcta (Cristianini y Shawe-Taylor, 2000).

La idea básica en la que se basa su funcionamiento es que dado un conjunto de puntos —subconjunto de un conjunto mayor o espacio—, en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo —cuya categoría desconocemos— pertenece a una categoría o a la otra. Por tanto, la máquina de vector de soporte busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior. En resumen, se busca el hiperplano que tenga la máxima distancia con los puntos que estén más cerca de él mismo, intentando que los puntos del vector etiquetados con una clase estén a un lado del hiperplano, y los de la otra clase en el otro lado. (Castro, 2013)

Puesto que existe un número infinito de hiperplanos que realicen la clasificación, intentaremos obtener el que permita un margen máximo entre los elementos de las clases. Por ejemplo, en la figura 2.10 podemos observar tres hiperplanos en un espacio bidimensional. H_1 no separa las clases, pero H_2 y H_3 si. Sin embargo, H_3 las separa con un margen mayor.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

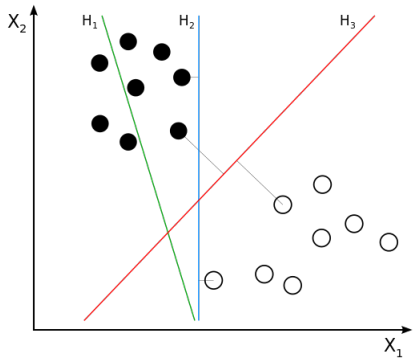

Figura 2.10: Hiperplanos en un espacio bidimensional (Weinberg, 2012).

Existen diferentes formas de entrenar un clasificador SVM. Un ejemplo famoso sería el algoritmo de Platt (1998) de optimización mínima secuencial (del inglés *sequential minimal optimization*, SMO). Este algoritmo toma como entradas un conjunto de datos $\{(x_1, y_1)$, ..., $(x_n, y_n)\}$ dónde x_i es un vector de entrada y y_i es la etiqueta binaria correspondiente. La SVM es entrenada entonces mediante un problema de programación cuadrátia expresado de la siguiente forma:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j K(x_i, x_j) \alpha_i \alpha_j$$

sujeto a:

$$0 \leq \alpha_i \leq C \; para \; i = 1, 2, ..., n$$
$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

donde C es el parámetro de complejidad o hiperparámetro de la SVM y $K(x_i, x_j)$ es la función de kernel. C controla el número y severidad de las violaciones del margen (y del hiperplano) que se toleran en el proceso de ajuste. Si $C = +\infty$, no se permite ninguna violación del margen y, por lo tanto, el resultado es equivalente al llamado clasificador de margen máximo (del inglés *maximal margin classifier*)(Schapire y col., 1998), teniendo en cuenta que esta solución solo es posible si las clases son perfectamente separables. Cuando más se aproxima C a cero, menos se penalizan los errores y más observaciones pueden estar en el lado incorrecto del margen o incluso del hiperplano. C es a fin de cuentas el hiperparámetro encargado de controlar el balance entre sesgo (bias) y varianza del modelo. En la práctica, su valor óptimo se identifica mediante validación cruzada (del inglés *cross-validation*)(Rodrigo, 2017).

2.3.2.1. Kernel Trick

Como hemos visto anteriormente, resulta ideal generar una separación mediante una línea recta, un plano o un hiperplano n-dimensional. Sin embargo, los universos a estudiar no siempre son de dos dimensiones, como el ejemplo, así que por lo general un SVM suele tratar con más de dos variables predictoras, curvas no lineales de separación, casos dónde los conjuntos de datos no pueden ser completamente separados y, por supuesto, más de dos clases en las que clasificar. Puesto que la capacidad computacional necesaria para las SVM en la mayoría de los casos del mundo real se vuelve excesiva, se utiliza como solución una representación por medio de funciones de kérnel que permiten proyectar la información a un espacio de características de mayor dimensión, aumentando así la capacidad computacional de las SVMs.

Este concepto parte de la idea de que existe una versión no lineal de cualquier algoritmo lineal basado en datos. Si encontramos una transformación no lineal $\varphi(x)$ a un espacio de mayor dimensionalidad provisto de un producto escalar que puede ser expresado como un kérnel $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$, entonces podremos construir una versión no lineal del mismo algoritmo donde la transformación no lineal es φ. Este teorema es popularmente conocido como el truco del kérnel (Bhattacharyya, 2018).

Un kérnel es toda aquella función $K(u, v)$ que verifican el teorema de Mercer (1909), es decir, para la cual

$$\int_{u,v} K(\mathbf{u}, \mathbf{v}) g(\mathbf{u}) g(\mathbf{v}) d\mathbf{u} d\mathbf{v} > 0$$

para toda función $g()$ de cuadrado integrable.

Los kérnels más comunes son:

- Lineal: $K(x_i, x_j) = x_i^T x_j$
- Polinómico: $K(x_i, x_j) = \left(x_i^T x_j + 1\right)^n$
- Gaussiano: $K(x_i, x_j) = \exp\left(\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$

El kérnel lineal es el que hemos visto hasta ahora. En la figura 2.11 podemos ver un ejemplo de transformación mediante función de kérnel polinómica, mientras que en la figura 2.12 podemos ver el resultado de aplicar una función de kérnel gaussiana con distintos valores de σ. El kernel gaussiano es también llamado kérnel de función de base radial (del inglés *radial basis function*, RBF) (Zoltan, 2018).

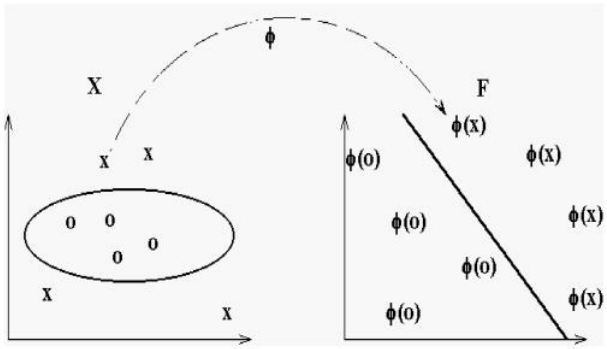

Figura 2.11: Kérnel Polinómico (Elisfm, 2009a).

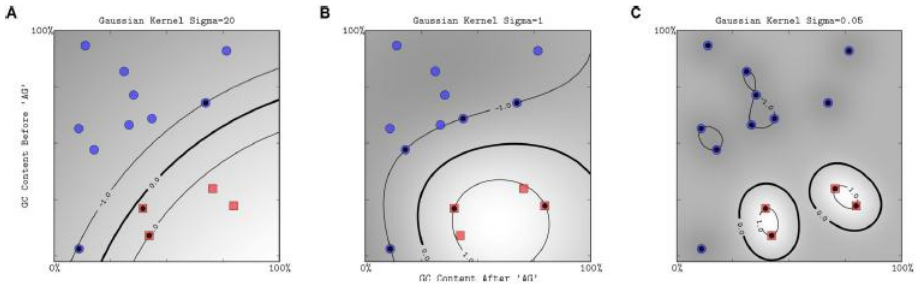

Figura 2.12: Kérnel gaussiano en función de σ (Elisfm, 2009b).

2.3.2.2. SVR

Las máquinas de vectores de soporte puede aplicarse no solo a problemas de clasificación sino también a casos de regresión, manteniendo todas las características principales que caracterizan el algoritmo.

La regresión mediante vectores de soporte (del inglés *support vector regression*, SVR) utiliza los mismos principios que los de clasificación, con solo algunas diferencias menores. En principio, dado que la salida es un número real, se vuelve difícil predecir la información a mano, dado que existen posibilidades infinitas. Para el caso de la regresión, entonces, se establece un margen de tolerancia cerca del vector con el fin de minimizar el error, teniendo en cuenta que parte de ese error es tolerado (Sayad, 2010).

Por ejemplo, para el conjunto de datos de la figura 2.13, el hiperplano que mejor representa el comportamiento de los datos, y puesto que los datos son de carácter lineal, es la recta $y = wx + b$. Una vez obtenido, se construyen las bandas paralelas al hiperplano que cubren la mayor cantidad de datos. Estas bandas son las que se conocen como vectores de apoyo o soporte y vienen determinadas por $+\varepsilon$ y $-\varepsilon$. Sin embargo, como se puede observar, no todos los datos han sido cubiertos por estas bandas, por lo que encontraríamos errores, que se deberán tener en cuenta para la fórmula del algoritmo. A la distancia entre las bandas y cada punto que no se encuentra cubierto se le conoce como épsilon ε. De esta forma la fórmula completa para el cálculo del algoritmo sería la siguiente:

$$min\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}(\xi + \xi^*)$$

donde

- w es la magnitud del hiperplano.
- C es el parámetro de complejidad o hiperparámetro de la SVM.
- ξ y ξ^* son las variables que controlan el error cometido por la función de regresión al aproximar las bandas.

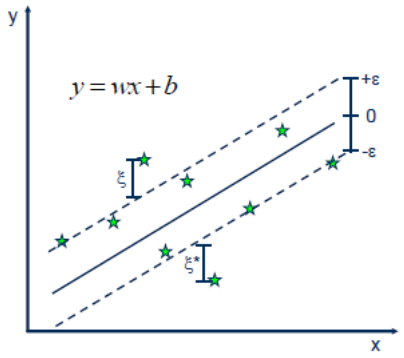

Figura 2.13: Hiperplano en SVR (Sayad, 2010).

Para el caso de un problema lineal, el SVR viene dado por:

$$y = \sum_{i=1}^{N}(\xi_i - \xi_i^*) \cdot < x_i, x > +b$$

Sin embargo, como hemos visto anteriormente, para datos no lineales se debe aplicar una función de kérnel para convertir los datos en un espacio de características dimensionales superiores, para que sea posible la separación lineal, como podemos ver en la figura 2.14. Así, la fórmula quedarían de la siguiente forma:

$$y = \sum_{i=1}^{N}(\xi_i - \xi_i^*) \cdot K(x_i, x) + b$$

En donde K sería la función de kérnel a aplicar.

En *Scikit-learn* encontramos la clase *SVR* [6], que nos permitirá hacer uso de este modelo de forma sencilla.

2.3.2.3. GridSearchCV

Como hemos visto hasta ahora el rendimiento del algoritmo de SVR depende de una buena configuración del parámetro C y los del kérnel. El problema de la selección óptima de parámetros

[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

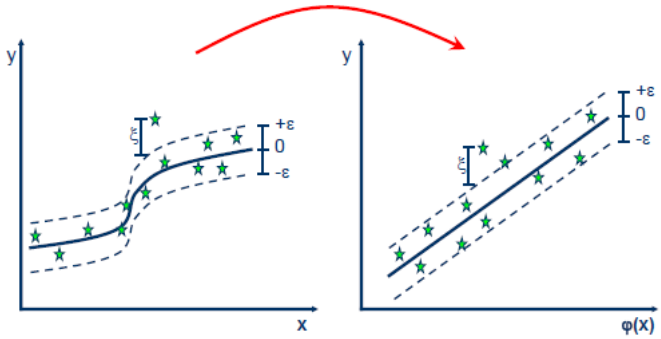

Figura 2.14: Aplicando la función de kérnel en SVR (Sayad, 2010).

se complica aún más por el hecho de que la complejidad del modelo de SVR depende de estos parámetros. Por tanto, resulta imprescindible obtener su valor óptimo mediante técnicas de validación cruzada (Bergmeir, 2012).

La validación cruzada es una técnica con la que se puede identificar la existencia de diferentes problemas durante el entrenamiento de los modelos, como la aparición de sobreajuste, permitiendo así obtener modelos más estables.

En la validación cruzada, el conjunto de datos de entrenamiento se divide en grupos de igual tamaño. Una vez realizada la partición, se procede a entrenar el modelo una vez por cada grupo. Se utiliza el grupo de la iteración para validar los resultados y, el resto, para entrenar. Una diferencia importante con otro tipo de validaciones es que en esta ocasión se entrena y valida con todos los datos, cambiando el conjunto utilizado para la validación en cada iteración. Así es posible identificar si los modelos son inestables o estables, es decir, si el resultado depende de los datos utilizados o no. En caso de que los resultados dependan de los datos utilizados, el modelo posiblemente estará siendo sobreajustado (Rodriguez, 2018).

Este proceso se muestra de forma esquemática en la figura 2.15. En ella se han representado un conjunto de datos que se ha dividido en tres. Posteriormente se entrenan tres modelos con los datos en azul y se valida con los datos en color verde.

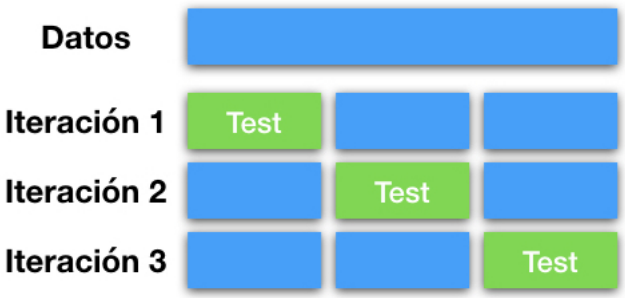

Figura 2.15: Entrenando modelos con validación cruzada (Rodriguez, 2018).

Scikit-learn dispone de varias clases que implementan la metodología de la validación cruzada. En el caso de que se desee utilizar para seleccionar los parámetros de entrenamiento de

un modelo, una de las opciones es *GridSearchCV* [7], siendo uno de los más simples y fáciles de utilizar. El constructor de esta clase se ha de llamar indicándole la instancia de un modelo, los valores a probar y el número de conjuntos en el que se dividen los datos. Esto se realiza mediante los siguientes parámetros:

- *estimator*: el modelo que se ha de evaluar.
- *param_grid*: un diccionario en que se indicar los parámetros a evaluar como clave y el conjunto elementos como valor.
- *cv*: el número de conjuntos en los que se divide los datos para la validación cruzada.

El resultado será la configuración óptima del modelo para la correcta utilización del estimador.

2.4. Conclusiones

Tras exponer las motivaciones de este trabajo, así como algunas investigaciones similares, creemos que hemos justificado los objetivos y la intención de los experimentos que se van a realizar. Con la recopilación de los conocimientos y conceptos necesarios hemos contextualizado todas las acciones que se llevarán a cabo durante el desarrolo e implementación de los algoritmos. Desde este punto podemos comenzar con el diseño de los experimentos, la implementación de las estrategias, y la obtención de resultados para finalmente saber discernir si los resultados han sido positivos, o por el contrario no hemos conseguido mejorar los beneficios haciendo uso de algoritmos de aprendizaje automático.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Capítulo 3

Estrategia *comprar y mantener*

Tras haber expuesto el área de conocimiento en el que está basado este documento, en este capítulo se procederá al desarrollo del primer experimento propuesto. Nos basaremos en la estrategia *comprar y mantener*, que consiste en invertir en momentos oportunos y mantener los valores en cartera, esperando que su valor aumente con el tiempo.

Se realizarán pruebas para las siguientes estrategias de inversión:

1. Estrategia *comprar y mantener* simple.
2. Estrategia *comprar y mantener* optimizada con bosques aleatorios.

Posteriormente se evaluarán los resultados y se compararán objetivamente.

Cabe señalar que la estrategia se va a diseñar e implementar de la forma más sencilla posible, ya que servirá como introducción al desarrollo de estrategias de inversión y dará paso a un experimento más complejo en el siguiente capítulo.

3.1. Diseño de los experimentos

En este apartado diseñaremos los experimentos que se llevarán a cabo. Se comenzará diseñando una estrategia de ejecución sencilla que posteriormente se optimizará mediante la aplicación de un algoritmo de aprendizaje automático.

Después de investigar acerca de diferentes algoritmos determinamos que tanto bosques aleatorios como las máquinas de vector soporte (SVMs) eran los que mejor se adaptaban a los datos que se querían predecir y con los que obteníamos los resultados más adecuados para las tareas a realizar, en los periodos de tiempo seleccionados. Tras varios ciclos de diseño-ejecución-resultados en todos los experimentos llevados a cabo observamos que los resultados más reseñables se daban combinando *comprar y mantener* con bosques aleatorios y *scalping* con máquinas de vector soporte (SVMs). Además, la forma de organizar los datos para entrenar ambos algoritmos era similar, por lo que la decisión fue clara. Cabe señalar que, de haber escogido otros periodos de tiempo, es probable que los algoritmos que mejor resultado hubieran

dado hubieran sido otros, pero puesto que lo importante de estos experimentos es su propio desarrollo, creímos conveniente dejar para otros estudios la comparación cualitativa de los diferentes algoritmos de aprendizaje automático en su aplicación a la optimización de estrategias de inversión.

3.1.1. Estrategia *comprar y mantener* simple

Como hemos dicho, la estrategia *comprar y mantener* se basa en comprar acciones en un momento concreto y mantenerlas en el tiempo con la esperanza de que aumenten su valor. Para ello podemos implementar diferentes formas de escoger ese momento de compra, como simplemente comprar a determinada hora cada día, comprar solo cuando algún indicador financiero nos recomiende hacerlo, o siguiendo cualquier criterio que nos parezca oportuno. En nuestro caso, para la estrategia básica, seguiremos un criterio sencillo basado en comprobar si el precio actual del valor a adquirir es inferior al precio medio de los últimos 33 días. En caso de que se cumpla la condición compraremos 10 acciones del valor hasta que nos quedemos sin capital. Cabe señalar que en esta estrategia solo se realizan operaciones de compra durante un corto periodo de tiempo y se observa el rendimiento de esas acciones a largo plazo.

En la figura 3.1 podemos ver un diagrama simple que resume la estrategia que se utilizará.

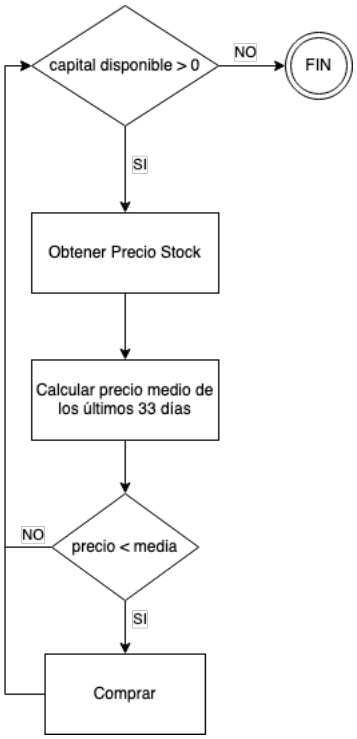

Figura 3.1: Diagrama para estrategia *comprar y mantener* simple.

La ejecución de la estrategia —así como el seguimiento del valor del portafolio— se realizará durante el periodo de tiempo comprendido entre enero de 2008 y diciembre de 2018 para las acciones de las empresas Apple y Microsoft con un capital inicial de 10.000$.

En el algoritmo 3.1 podemos ver la definición en pseudocódigo de la clase *ComprarYMantener*. En esta definición encontramos la estrategia de comprar y mantener que utilizaremos como base.

Algoritmo 3.1 Estrategia *comprar y mantener* simple.

```
clase ComprarYMantener

  establecer Stocks a [Apple, Microsoft]

  Subproceso procesar_informacion_diaria
    Si dinero_disponible > 0 entonces
      Para Cada stock en Stocks Hacer
        precio = obtener_precio( valor = stock )
        datos_historicos = obtener_datos_historicos( valor = stock, dias = 33 )
        media_historica = media( datos_historicos )
        Si precio < media_historica Entonces
          comprar_acciones( valor = stock, cantidad = 10 )
        Fin Si
      Fin Para
    Fin Si
  Fin Subproceso
```

3.1.2. Estrategia *comprar y mantener* con bosques aleatorios

Una vez hemos diseñado un algoritmo sencillo basado en la estrategia *comprar y mantener*, podemos intentar optimizarla haciendo uso de algoritmos de aprendizaje automático con el fin de determinar de una manera más eficiente el momento de compra adecuado. En este caso, primero generaremos un modelo predictivo basado en datos históricos que nos ayuden a predecir en tiempo real el precio futuro de una acción. Para ello realizaremos tareas de tratamiento de datos para ajustar los valores de entrada del algoritmo que generará el modelo predictivo. Seguidamente, utilizaremos la herramienta de predicción recién creada en la ejecución de la estrategia, utilizándolo como criterio para determinar si es un momento oportuno para comprar una determinada acción. En esta ocasión compararemos el precio medio de los últimos 33 días con el precio máximo de los valores predichos por el modelo para los siguientes 7 días, siguiendo el proceso descrito en el diagrama de la figura 3.2.

Con el modelo entrenado podemos ajustar la estrategia *comprar y mantener* y de esa manera esperar obtener mejores resultados. En el algoritmo 3.2 podemos ver el proceso que se ha de seguir en cada lectura de datos.

La estrategia realizará una operación de compra en caso de que el máximo valor de los valores predichos para los siguientes 7 días sea mayor que la media de los últimos 33 días, siempre que exista dinero disponible para comprar.

Para la generación del modelo predictivo utilizaremos los valores de la empresa Microsoft desde el 19 de febrero de 1999 al 30 de octubre de 2009. Por otro lado, y al igual que en el experimento anterior, la ejecución de la estrategia se realizará durante el periodo de tiempo

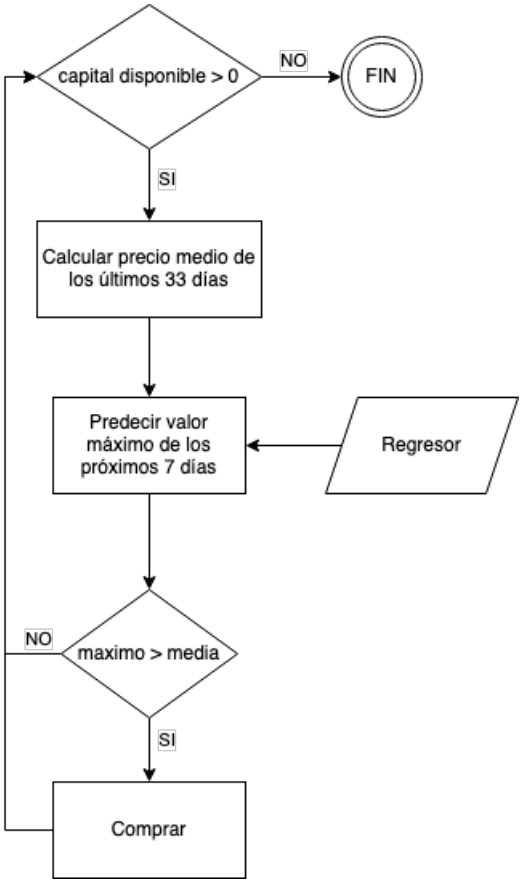

Figura 3.2: Diagrama de Estrategia *comprar y mantener* con *bosques aleatorios*.

comprendido entre enero de 2008 y diciembre de 2018 para las acciones de las empresas Apple y Microsoft.

3.2. Implementación

Los experimentos se llevarán a cabo haciendo uso de una biblioteca adaptada para emular sistemas financieros y, de esa forma, poder obtener resultados cercanos a los que obtendríamos en caso de ejecutarlos en entornos reales. La ventaja principal de hacer uso de esta biblioteca —en lugar de trabajar sobre datos directamente— es que podemos utilizar una extensa lista de funcionalidades que tendríamos que desarrollar por nuestra cuenta en caso de no contar con ella. Por ello, antes de comenzar, se describirá una forma óptima de configurar el entorno de experimentación, ya que servirá como base para entender la implementación y ejecución de los experimentos.

Seguidamente, se procederá a la implementación de las estrategias propuestas y a su posterior ejecución que generarán resultados que podremos describir y comparar en el siguiente apartado.

Algoritmo 3.2 Estrategia *comprar y mantener* con *bosques aleatorios*.

```
clase ComprarYMantener
  establecer Regresor a RandomForestRegressor
  establecer Stocks a [Apple, Microsoft]

  Subproceso procesar_informacion_diaria
    Si dinero_disponible > 0 entonces
      Para Cada stock en Stocks Hacer
        datos_historicos = obtener_datos_historicos( valor = stock, dias = 33 )
        media_historica = media( datos_historicos )
        prediccion = Regresor->predecir_proximos_7_dias( datos_historicos )
        maximo = maximo( prediccion )
        Si maximo > media_historica Entonces
          comprar_acciones( valor = stock, cantidad = 10 )
        Fin Si
      Fin Para
    Fin Si
  Fin Subproceso
```

3.2.1. Preparación del entorno

Para el desarrollo de todas las pruebas se utilizará Python como lenguaje principal, haciendo uso de las siguientes utilidades:

Anaconda[1]: Distribución gratuita y de código abierto de los lenguajes Python y R. Creado específicamente para computación científica como ciencia de datos, aprendizaje automático, análisis, etc. Su principal utilidad es la de gestionar paquetes y entornos de virtualización, permitiéndonos utilizar las versiones correctas para cada situación sin necesidad de estar instalando y desinstalando versiones concretas de otras utilidades que utilizaremos.

Scikit-learn [2]: Biblioteca de código abierto que dispone de una amplia variedad de implementaciones de diferentes algoritmos de aprendizaje automático. También dispone de utilidades para manejar y preparar datos.

Zipline[3]: Biblioteca de código abierto con códigos específicos de trading. Facilita enormemente la obtención de datos históricos así como la aplicación de algoritmos estándar y el estudio de los resultados obtenidos en cada caso. Al no ser compatible con las últimas versiones de Python requiere que se hagan unos pequeños ajustes haciendo uso de las características de Anaconda. En el código 3.2.1 podemos ver la lista de comandos a utilizar para crear un entorno de desarrollo virtual sobre el que configuraremos las versiones adecuadas y que utilizaremos para la ejecución del resto del código. Entre las alternativas a zipline encontramos:

- Ultra Finance[4]: Biblioteca en Python con herramientas para obtener información financiera en tiempo real, analizarla y realizar acciones de simulación de implementación de estrategias. Es una potente herramienta, pero no cuenta con la documentación, como los recursos y los ejemplos de los que sí dispone *zipline*.

[1] https://www.anaconda.com/
[2] https://scikit-learn.org/stable/index.html
[3] https://www.zipline.io/
[4] https://code.google.com/archive/p/ultra-finance/

- PyAlgoTrade[5]: Biblioteca de desarrollo de trading algorítmico bastante completa, con herramientas específicas para probar estrategias basadas en criptomonedas. Permite realizar de forma gratuita algoritmos que operen con información actualizada al minuto. Su última versión se publicó en agosto de 2018. Sin duda es la herramienta que hubiéramos utilizado de no ser por que *zipline* cuenta con mayor soporte de desarrollo y sigue publicando actualizaciones.
- Backtrader[6]: Potente alternativa a zipline, con características muy superiores en muchos aspectos. No obstante, al igual que sucedía con Ultra Finance, no encontramos suficientes recursos externos ni una comunidad que la soporte, en comparación con la de *zipline*. No descartamos utilizar esta herramienta en el futuro, pero para evitar posibles problemas durante los experimentos o atascos en el desarrollo, preferimos escoger una herramienta que nos permita encontrar soluciones de forma más sencilla.

```
conda config --set allow_conda_downgrades true
conda install conda=4.6.11
conda create -n env_zipline python=3.5
conda activate env_zipline
conda install -c Quantopian zipline pyfolio
```

Código 3.2.1: Instalación de *zipline* en entorno virtual

Quandl[7]: Fuente de datos financieros, económicos y alternativos, que atiende a profesionales de la inversión. La plataforma de Quandl es utilizada por más de 400.000 usuarios, incluidos analistas de los principales fondos de cobertura del mundo, gestores de activos y bancos de inversión. Una vez obtenido el registro podemos conseguir una clave de acceso o *API KEY* que nos permitirá leer la información financiera desde las utilidades de *zipline*. En el código 3.2.2 podemos ver la forma correcta de configurar Quandl con *zipline*.

```
QUANDL_API_KEY=API_KEY
zipline ingest -b quantopian-quandl
```

Código 3.2.2: Integración de Quandl en *zipline*

IEX Cloud[8]: Plataforma de información financiera. Provee información de diferentes tipos desde valores institucionales, propiedades, acciones internacionales, fondos, opciones, etc. Requiere suscripción para obtener una clave de acceso o *token* el cual deberemos añadir en el código fuente de *zipline*. Además deberemos modificar el código existente ya que la versión de *zipline* que utilizaremos todavía no está adaptada a IEX Cloud y utiliza fuentes de datos de IEX obsoletas. En el código 3.2.3 podemos ver el código que hay que reemplazar en el fichero *~/anaconda3/envs/env_zipline/lib/python3.5/site-packages/zipline/data/benchmarks.py.*

[5] http://gbeced.github.io/pyalgotrade/
[6] https://www.backtrader.com/
[7] https://www.quandl.com/
[8] https://iexcloud.io/

```
#r = requests.get('https://api.iextrading.com/1.0/stock/{}/chart/5y'.format(symbol))
r = requests.get(
    "https://cloud.iexapis.com/stable/stock/{}/chart/5y?chartCloseOnly=True&token={}".
        format( symbol, IEX_TOKEN )
)
```

Código 3.2.3: Configuración de IEX Cloud

Eclipse[9]: Entorno de desarrollo de código abierto con multitud de opciones de personalización y configuración, lo que lo hacen ideal para trabajar con Python y entornos virtualizados de Anaconda. Entre las diferentes extensiones que podemos instalar, la más importante para nuestro objetivo es la denominada PyDev, que puede encontrarse en el catálogo de extensiones de Eclipse denominado *MarketPlace*.

Una vez instalado *PyDev* deberemos configurar el entorno para que ejecute las versiones correctas de los paquetes haciendo uso del entorno virtualizado creado anteriormente. En la Figura 3.3 podemos ver la ventana de configuración *Preferencias->PyDev->Interpreters->Python Interpreter* una vez seleccionado el interprete en la ruta *~/anaconda3/envs/env_zipline/bin/python3.5*.

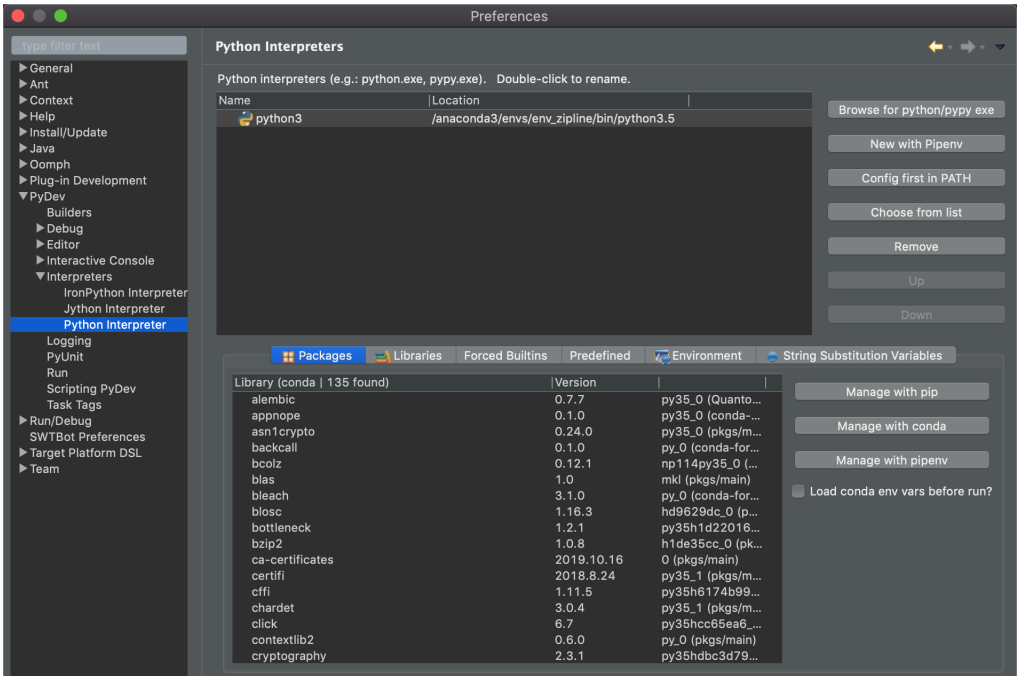

Figura 3.3: Configuración de Intérprete Python en PyDev

Para la correcta ejecución de los distintos algoritmos estableceremos una estructura basada en programación orientada a objetos que nos permita ejecutar las diferentes estrategias de forma similar y así poder evaluarlas de forma equitativa. Utilizaremos algunas funciones globales de *zipline* como *run_algorithm* [10] y *register_calendar* [11] así como los diferentes valores globales que nos suministra el framework. En el código A.1.1 podemos ver el código correspondiente

[9] https://www.eclipse.org/
[10] https://www.zipline.io/appendix.html#running-a-backtest
[11] https://www.zipline.io/appendix.html#zipline.utils.calendars.register_calendar

a la función *run_strategy* que nos permitirá ejecutar una estrategia concreta definida en una clase denominada *strategy_name*.

Como podemos observar en la llamada a la función *run_algorithm* en la línea 43 del código A.1.1, las clases que ejecuten cada estrategia deberán definir las funciones *initialize*, *handle_data*, *before_trading_star*, *analyze* y *_test_args*. Estas funciones serán las encargadas de definir la estrategia como tal.

En este primer experimento hemos definido como datos de entrada los almacenados en el paquete de datos *quandl*, pero más adelante podremos ver como los datos se pueden personalizar siempre que se mantenga una estructura concreta compatible con *zipline*.

Para la obtención de resultados utilizaremos el fichero de salida que genera *zipline* de forma automática con todos los detalles sobre las operaciones realizadas.

3.2.2. Implementación de *comprar y mantener* simple

Comenzamos implementando la clase *BuyAndHold* —descrita en el algoritmo 3.1— en el código A.2.1 con las funciones descritas en el código A.1.1.

Como podemos ver en A.2.1, en la función *initialize* introducimos en la variable *context* los valores *stocks* con los códigos de las empresas seleccionadas para que, de esta forma, estén disponibles durante todos los estados de la ejecución de *zipline*.

Por otro lado, en la función *handle_data* establecemos la estrategia de inversión. Para cada lectura de precios, si tenemos dinero suficiente, recorremos cada uno de los valores de la variable *stocks* y compramos 10 acciones de cada *stock* si el precio del mismo está por debajo de la media de los últimos 33 días. Para el número de días escogido se ha tenido en cuenta el experimento posterior con bosques aleatorios, en el que se utilizan 40 valores, 33 de los cuales son para el grupo de entrenamiento, y 7 para el de test. De esta forma intentamos acercar lo más posible ambos experimentos para que las comparaciones sean más justas.

En la función *_test_args* establecemos las fechas de inicio y finalización para ejecutar la estrategia, así como el valor del capital inicial que establecemos en 10.000$.

Finalmente, en la función *analyze* generamos una gráfica con el valor del portafolio a lo largo de la ejecución de la estrategia.

3.2.3. Implementación de *comprar y mantener* con bosques aleatorios

En primer lugar, deberemos generar el modelo predictivo que nos permitirá optimizar la estrategia. Para ello comenzaremos importando todas las funciones y paquetes necesarios para entrenar el regresor. Encontramos el listado de dependencias en el código A.2.2. Como podemos

observar, utilizamos *pandas*[12] y *numpy*[13] para el tratamiento de los datos en general. De *zipline* utilizaremos:

- *DataPortal*[14] como conector para obtener el histórico de valores de un activo concreto.
- *bundles*[15] para seleccionar el conjunto de datos.
- *get_calendar*[16] para obtener el calendario de operaciones correspondiente.

Utilizaremos el paquete *sklearn*[17] para utilizar los datos obtenidos anteriormente como fuente de entrada para el entrenamiento del bosque aleatorio.

Finalmente, almacenaremos el resultado en un fichero mediante el módulo *dump*[18] del paquete *joblib*.[19]

En el código A.2.3 podemos ver la secuencia completa para obtener el histórico de valores del activo MSFT (Microsoft) desde el 19 de febrero de 1999 al 30 de octubre de 2009, que en total serían cinco mil días de mercados abiertos. Cargamos el paquete de datos *quandle* y establecemos la fecha final *end_date*. Seguidamente, inicializamos el conector *DataPortal* con los datos almacenados en el conjunto de datos de *quandl*, utilizando su buscador de activos, estableciendo como inicio de datos (*first_trading_day*) el 19 de febrero de 1999 ya almacenado de forma estática en *bundle_data.equity_daily_bar_reader.first_trading_day*, y utilizamos también el lector de equidad del mismo conjunto de datos.

Utilizamos el conector para obtener los datos correspondientes al activo MSFT mediante la función *lookup_symbol* del buscador de activos. Finalmente, utilizamos *get_history_window* para obtener el marco de datos que podremos tratar con *pandas*, estableciendo una frecuencia de datos diaria, y obteniendo el valor *close* de cada día, que será el que utilicemos para entrenar el algoritmo. En las últimas cuatro líneas se eliminan valores nulos, se establecen el índice de cada fila como la columna de la fecha, se crea la columna 'close' con los datos obtenidos y se eliminan las filas sobrantes para obtener un conjunto de datos similar al de la tabla 3.1, que en conjunto mostrarían una gráfica como la de la figura 3.4.

Para poder entrenar el algoritmo de bosque aleatorio deberemos modificar la tabla de datos para obtener una tabla numérica. Tras realizar diferentes pruebas en la generación del modelo determinamos que con una ventana temporal de 40 días obteníamos los resultados más precisos. Para ello, añadiremos para cada día 40 columnas adicionales con los precios de cierre pasados del activo. Es decir, en la siguiente columna a 'close' obtendremos el precio del activo del día siguiente, en la segunda columna el de dos días siguientes, y así hasta completar una ventana

[12] https://pandas.pydata.org/
[13] https://numpy.org/
[14] https://www.zipline.io/_modules/zipline/data/data_portal.html
[15] https://www.zipline.io/bundles.html
[16] https://www.zipline.io/_modules/trading_calendars/calendar_utils.html
[17] https://scikit-learn.org/stable/
[18] https://joblib.readthedocs.io/en/latest/generated/joblib.dump.html
[19] https://joblib.readthedocs.io/en/latest/

	close
1999-02-19 00:00:00	147.75
1999-02-22 00:00:00	148.81
1999-02-23 00:00:00	155.44
1999-02-24 00:00:00	152.88
1999-02-25 00:00:00	153.50

Cuadro 3.1: Bosque aleatorio: Valores de cierre de MSFT.

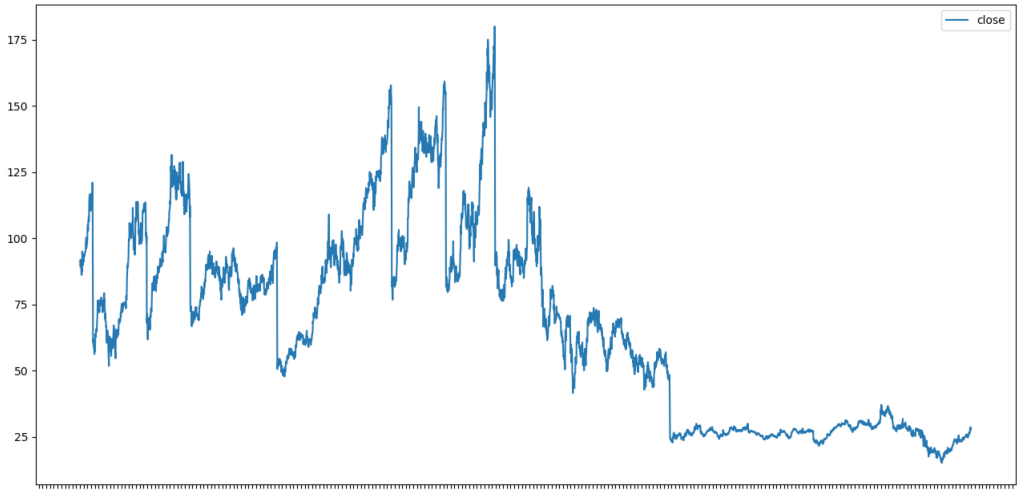

Figura 3.4: Bosque aleatorio: Evolución del precio de MSFT.

temporal de 40 días. Para ello hacemos uso de la función *shift*[20] de *pandas*, como podemos ver en el código A.2.4. Obtenemos así un conjunto de datos similar a los de la tabla 3.2.

	close	1d	2d	...	39d	40d
1999-02-19 00:00:00	147.75	148.81	155.44	...	86.62	81.00
1999-02-22 00:00:00	148.81	155.44	152.88	...	81.00	83.12
1999-02-23 00:00:00	155.44	152.88	153.50	...	83.12	82.00
1999-02-24 00:00:00	152.88	153.50	150.13	...	82.00	84.94
1999-02-25 00:00:00	153.50	150.13	151.75	...	84.94	86.00

Cuadro 3.2: Bosque Aleatorio: Serie Temporal

A continuación separamos los datos en dos conjuntos X e Y.

- En X almacenaremos las primeras 32 columnas con los valores diarios y los precios futuros. Será el conjunto que se utilizará para entrenar el modelo.
- En Y almacenaremos las últimas 8 columnas y se utilizarán como los valores esperados que el algoritmo tendrá que ser capaz de predecir.

De esta manera tendremos en X los valores desde 'close' hasta '32d', y en Y desde '33d' hasta '40d'. Para ello hacemos uso de la función *iloc*[21] de *pandas* como podemos ver en el código

[20] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.shift.html
[21] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.iloc.html

A.2.5. Seguidamente hacemos uso de la función *train_test_split*[22] que separa cada conjunto de datos en otros dos de entrenamiento y pruebas seleccionando los datos de forma aleatoria. En este caso establecemos que se utilice el 40 % de los datos para test. De esta forma conseguimos los subconjuntos *X_train*, *X_test*, *Y_train* e *Y_test* que utilizaremos en el modelo mediante el módulo *RandomForestRegressor*.[23] Utilizamos valores por defecto de entrada y entrenaremos con *X_test* e *Y_test*. Ahora podemos obtener el coeficiente de determinación con la función *score* y los conjuntos *X_test* e *Y_test*, obteniendo como valor de retorno 0.99, por lo que podemos esperar un ajuste significativo de los datos y así obtener un conjunto de valores predichos en *Y_predicted* apropiados. Finalmente, exportamos el modelo entrenado para de esa forma no tener que rehacer los cálculos en predicciones futuras y poder utilizar el regresor recién obtenido en escenarios diferentes.

Para finalizar, ajustamos las funciones *initialize* y *handle_data* del código A.2.1 y las dejamos como en el código A.2.6. Haciendo uso de *data.history* obtenemos los precios de cierre diarios de los últimos 33 días del símbolo correspondiente. Puesto que *data.history* devuelve una fila por cada día solicitado, utilizamos la función *reshape*[24] para convertir la columna de los precios en una sola fila. Utilizando el objeto generado anteriormente, *rf_regressor*, obtenemos una predicción de los precios de los siguientes 7 días y calculamos el máximo. Generamos también la media de los últimos 33 días y de esa manera podemos determinar si esta media es menor que el máximo predicho, y así realizar una operación de compra.

Al igual que en el experimento anterior, en la función *_test_args* establecemos las fechas de inicio y finalización para ejecutar la estrategia, el valor del capital inicial y en la función *analyze* generamos una gráfica con el valor del portafolio a lo largo de la ejecución de la estrategia.

3.3. Evaluación y resultados

En la gráfica de la Figura 3.5 podemos observar los resultados obtenidos con la ejecución de la estrategia *comprar y mantener* simple. Puesto que escogimos una fecha inicial caracterizada por una fuerte recesión debido a la crisis que comenzó en 2008 —y a pesar de que nuestra cartera comienza perdiendo valor—, podemos observar como los resultados finales han sido bastante satisfactorios, multiplicando por 6 aproximadamente la cantidad invertida inicialmente.

En la tabla 3.3 podemos observar las operaciones de compra realizadas durante el proceso de inversión, así como el valor de los activos en el momento de la compra. Se realizan un total de 10 operaciones de compra dando como resultado una cartera de 50 acciones de Apple y 50 acciones de Microsoft. En 2014 Apple realizó una operación conocida como "split" o desdoblamiento de acciones con el objetivo de aumentar el número total de acciones, haciendo que cada acción costara menos y aumentando por tanto el número de acciones de cada inversor

[22] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[23] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[24] https://numpy.org/doc/stable/reference/generated/numpy.ndarray.reshape.html#numpy.ndarray.reshape

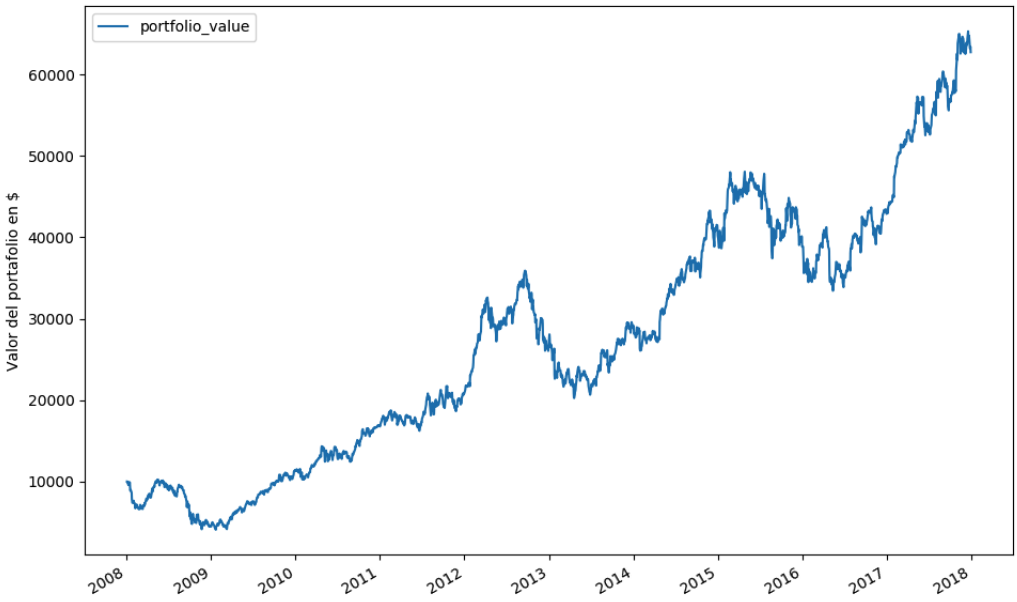

Figura 3.5: Evolución del portafolio en estrategia *comprar y mantener*.

[25]. De esta forma en 2014 nuestro portafolio pasa de 50 acciones de Apple a 350.
En resumen, los datos más importantes del resultado del experimento son los siguientes:

- Presupuesto Inicial: 10.000 $
- Operaciones: 10
- Valor mínimo del Portafolio: 4.331,25 $
- Valor Máximo del Portafolio: 65.563,25 $
- Valor Final del Portafolio: 63.004,75 $

Fecha	Stock	Precio Compra	Cantidad
2008-01-04	AAPL	180.05	10
2008-01-04	MSFT	34.38	10
2008-01-07	AAPL	177.64	10
2008-01-07	MSFT	34.61	10
2008-01-08	AAPL	171.25	10
2008-01-08	MSFT	33.45	10
2008-01-09	AAPL	179.4	10
2008-01-09	MSFT	34.44	10
2008-01-10	AAPL	178.02	10
2008-01-10	MSFT	34.33	10

Cuadro 3.3: Operaciones realizadas con la estrategia *comprar y mantener*.

[25] https://www.ennaranja.com/es-noticia/cada-accion-de-apple-vale-hoy-siete-veces-menos-que-ayer-y-sus-inversores-estan-tranquilos-por-que/

Tras calibrar el regresor podemos superponer los valores generados (azul) sobre los valores reales (rojo) de la figura 3.4, obteniendo como resultado la gráfica de la figura 3.6. Como podemos observar, los datos se ajustan bastante bien, teniendo en cuenta la precisión de 0.99 obtenida anteriormente.

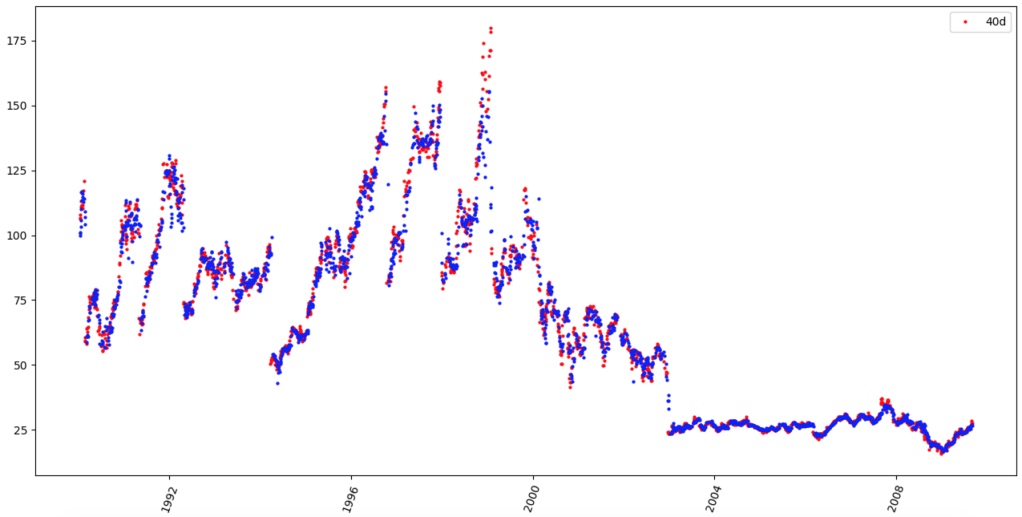

Figura 3.6: Bosque Aleatorio: predicciones obtenidas.

Una vez optimizado y ejecutado el algoritmo obtenemos los resultados de la figura 3.7, que reflejan el valor de nuestro portafolio a lo largo del tiempo. Como hemos comentado anteriormente, a pesar de que en los primeros años el valor de nuestras acciones disminuye, cuando el mercado volvió a ser favorable los beneficios crecen a un ritmo similar al que lo hacían con la estrategia simple, pero con unos resultados finales visiblemente superiores, llegando a multiplicar por 7 el valor de la cartera.

En la tabla 3.4 tenemos el listado de las operaciones realizadas. Al final del experimento terminamos con un total de 50 acciones de Apple y 110 acciones de Microsoft. Las operaciones realizadas difieren de las obtenidas en la tabla 3.3. En este caso podemos comprobar como en esta ocasión se retrasa la compra de acciones de Apple y durante unos días solo se compran acciones de Microsoft. De esta forma hemos podido terminar con más del doble de acciones de Microsoft que en el experimento anterior, y las mismas 50 de Apple. Del mismo modo que sucedía en el experimento anterior, esas 50 acciones de Apple se convierten en 350 en 2014.

En resumen, los datos más importantes del resultado del experimento son los siguientes:

- Presupuesto Inicial: 10.000 $
- Operaciones: 16
- Valor mínimo del Portafolio: 4.575,81 $
- Valor Máximo del Portafolio: 70.002,61 $
- Valor Final del Portafolio: 67.393,71 $

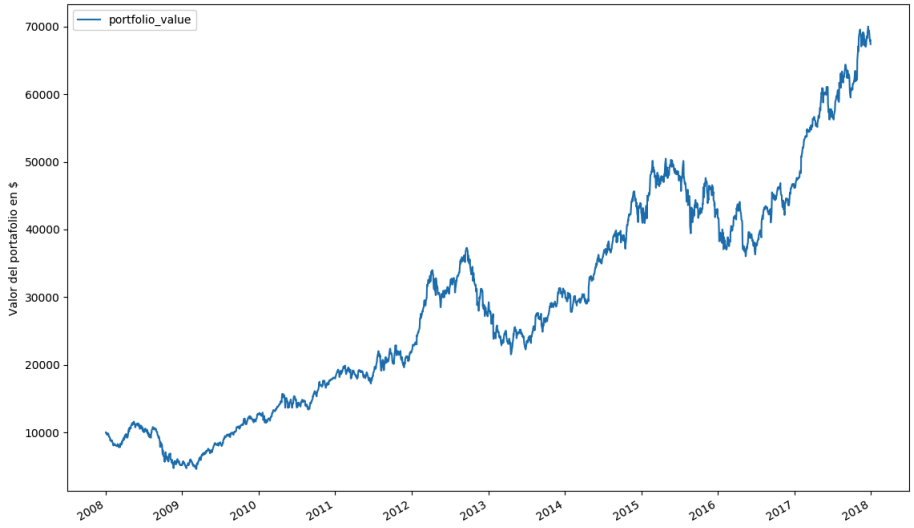

Figura 3.7: Evolución del portafolio en estrategia *comprar y mantener* con *bosques aleatorios*.

3.4. Discusión

Hemos podido comprobar como el uso de un algoritmo de bosques aleatorios nos ha permitido comprar acciones en momentos más oportunos para de esa forma aumentar la cantidad de nuestros activos finales con la misma inversión. De esta forma, al finalizar el periodo teníamos más acciones de cada compañía, y por tanto aumentaba el valor de nuestra cartera. Concretamente, casi un 7 % más de beneficio final. En la tabla 3.5 podemos comparar los resultados de forma más específica.

	Comprar y mantener	CyM con bosques aleatorios	Mejora
Valor Inicial	10.000 $	10.000 $	+ 0 %
Valor Final	63.004,75 $	67.393,71 $	+ 6,9 %
Operaciones Realizadas	10	16	+ 60 %
Acciones Apple	50	50	+ 0 %
Acciones Microsoft	50	110	+ 120 %

Cuadro 3.5: Comparación de resultados entre los experimentos de *comprar y mantener*.

Cabe señalar que la estrategia ha sido implementada de forma sencilla sin añadir inversión durante el periodo de prueba. Si durante los 10 años que duraba el experimento se hubiera invertido más dinero, el algoritmo hubiera podido determinar más momentos específicos de compra de acciones, y los resultados finales probablemente hubieran sido mucho más satisfactorios, es decir, hubiéramos terminado con mucho más dinero.

Fecha	Stock	Precio Inicial	Cantidad
2008-01-02	AAPL	194.84	10
2008-01-02	MSFT	35.22	10
2008-01-03	AAPL	194.93	10
2008-01-03	MSFT	35.37	10
2008-01-07	MSFT	34.61	10
2008-01-09	MSFT	34.44	10
2008-01-10	MSFT	34.33	10
2008-01-11	MSFT	33.91	10
2008-01-14	MSFT	34.39	10
2008-01-15	MSFT	34.0	10
2008-01-16	MSFT	33.23	10
2008-01-17	MSFT	33.11	10
2008-01-18	MSFT	33.01	10
2008-01-26	AAPL	119.15	10
2008-01-27	AAPL	122.96	10
2008-01-28	AAPL	129.91	10

Cuadro 3.4: Operaciones realizadas en estrategia *comprar y mantener* con bosques aleatorios.

Capítulo 4

Estrategia de *scalping*

Después de haber realizado el experimento sobre una estrategia de inversión a largo plazo habiendo obtenido resultados positivos, intentaremos optimizar de un modo similar una estrategia a corto plazo. En este caso nos basaremos en una estrategia de *scalping*, que consiste en abrir y cerrar posiciones varias veces al día con la intención de obtener pequeños beneficios en cada operación.

Se operará sobre el activo futuro DAX —que representa el valor esperado del índice DAX—, con una frecuencia de lectura por minutos en el periodo comprendido entre el 18 de septiembre de 2017 a las 9 horas y 16 minutos, y el 13 de octubre del mismo año a las 7 horas y 56 minutos, es decir, casi un mes. Este periodo se corresponde con un conjunto de datos de 15.000 minutos obtenidos de forma aleatoria de un conjunto de datos mayor. Se escogió este tamaño con la intención de aproximar a un mes el tiempo de ejecución, teniendo en cuenta la falta de lecturas en algunos minutos en el conjunto de datos inicial, y que solo se obtienen valores para los periodos en los que los mercados han estado abiertos (descontando noches, fines de semana y posibles festivos).

Al igual que hicimos en el experimento anterior, comenzaremos definiendo una estrategia de inversión sencilla, para posteriormente optimizarla mediante el uso de aprendizaje automático. En este caso realizaremos el experimento diseñando un paso intermedio en el que se utilizará una técnica de gestión de riesgo. Utilizaremos la técnica conocida como VaR (del inglés *Value At Risk*) por ser una técnica sencilla de aplicar y de entender. Además, al calcularse utilizando datos históricos se presta a una optimización sin complicaciones.

Como ya explicamos en el experimento anterior utilizaremos máquinas de vector soporte (SVMs) para optimizar el cálculo del VaR tras realizar diferentes pruebas con otros algoritmos y determinar que era, en principio, la opción más adecuada.

Se realizarán tres pruebas para las siguientes estrategias de inversión:

1. Ejecución de estrategia de *scalping* simple

2. Ejecución de estrategia de *scalping* con gestión de riesgo (VaR)

3. Ejecución de estrategia de *scalping* con VaR optimizada con SVR.

Posteriormente se evaluarán los resultados y se compararán objetivamente.

4.1. Diseño de los experimentos

En este apartado diseñaremos los experimentos que se llevarán a cabo. Se comenzará diseñando una estrategia de ejecución sencilla que posteriormente se optimizará mediante la aplicación de una técnica de gestión de riesgo, y finalmente se optimizará esta última mediante técnicas de aprendizaje automático.

4.1.1. Estrategia de *scalping* simple

En el diagrama de la figura 4.1 se puede entender de una forma general el proceso de la estrategia. En la implementación final se tomarán en cuenta más situaciones posibles que no se contemplan en el diagrama, como los cambios de señal o el tiempo de espera hasta que se cumplan algunas condiciones.

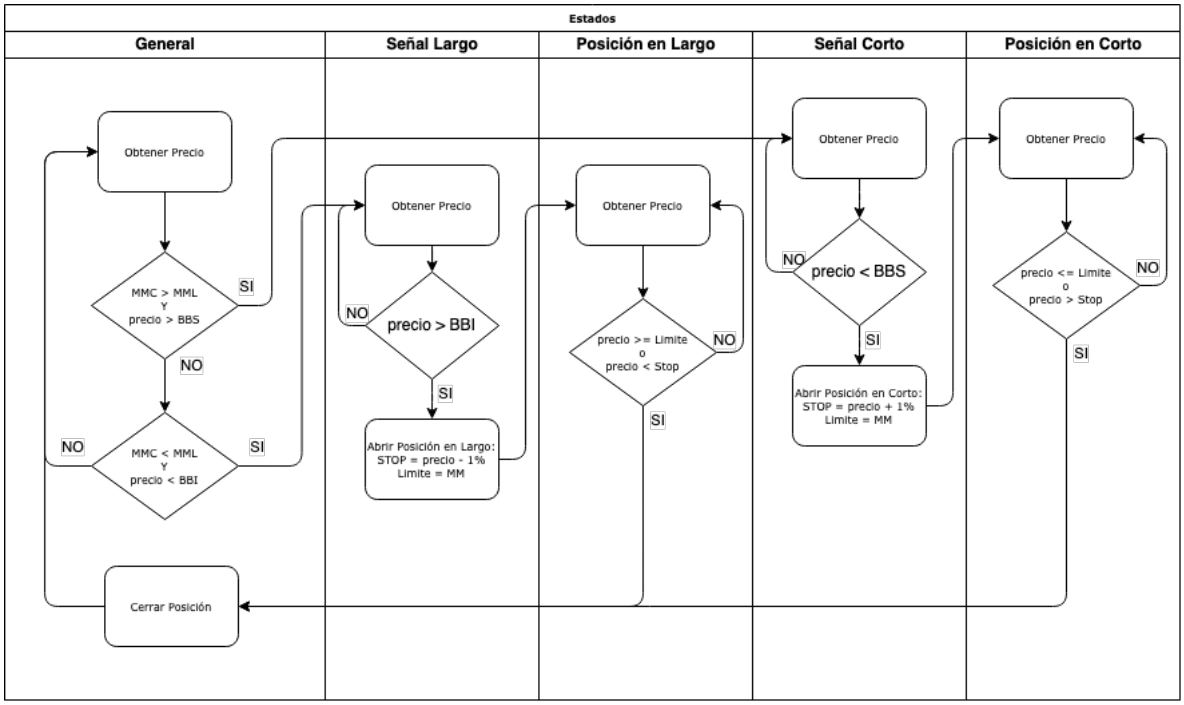

Figura 4.1: Diagrama de estrategia de *scalping*.

En la figura4.1 se utilizan algunas siglas cuya leyenda podemos encontrar a continuación:

- *MM*: Media móvil, utilizada para calcular las bandas de Bollinger y el valor esperado; en el experimento hemos tomado una ventana de 20 minutos.
- *BBS*: Banda de Bollinger superior.

- *BBI*: Banda de Bollinger inferior.
- *MMC*: Media móvil corta; en el experimiento hemos tomado un valor de 100 minutos.
- *MML*: Media móvil larga; en el experimiento hemos tomado un valor de 200 minutos.

Como podemos observar, se distinguen diferentes estados dependiendo de los valores que el precio tenga en cada momento. Una vez obtenido el precio, éste se compara con las bandas de Bollinger y a su vez se comparan las medias moviles para de esa forma determinar si estamos ante una previsión de posición corta o larga, es decir, ante una señal de corto o de largo. Dependiendo de la señal detectada las comparaciones con el precio se harán con respecto a la banda de Bollinger superior o inferior, y una vez cumplida la expectativa, se abrirá una posición en corto o en largo. Una vez abierta la posición nos encontraremos en un estado de posición abierta, en la que en cada lectura de precio compararemos con los valores del límite y del stop establecidos en la apertura de la posición, para así determinar el momento adecuado de cierre, ya sea con beneficios o con pérdidas.

A continuación describiremos mediante pseudocódigo el funcionamiento de la estrategia. Como podemos ver en el algoritmo 4.1, comenzamos declarando la clase 'Scalping' y estableciendo algunos valores que se utilizarán de forma global durante todo el proceso.

Algoritmo 4.1 Estrategia de *scalping*: Valores globales y subprocesos secundarios.

```
clase Scalping
        establecer Stock a DAX
        establecer Stop a 0
        establecer Límite a 0
        establecer Estado a Neutral
        establecer Precio a 0
        establecer Media_Movil a 0
        establecer Precio_Apertura a 0
        establecer Tipo a Neutral
        establecer Valor_Acumulado a 0

        Subproceso reiniciar_valores
                establecer Límite a 0
                establecer Stop a 0
                establecer Estado a Neutral
                establecer Tipo a Neutral

        Subproceso abrir_posicion ( STOP, TIPO )
                establecer Stop a STOP
                establecer Tipo a TIPO
                establecer Precio_Apertura a Precio
                establecer Límite a Media_Movil
                establecer Estado a Abierta

        ...
```

A continuación describiremos las diferentes variables que se utilizan:

- *Stock*: El valor de mercado a manejar. En este caso DAX.
- *Stop*: Precio que, en caso de alcanzarse, indicaría que habría que cerrar la operación (con pérdidas). Será 0 mientras no abramos una operación.

- *Límite*: Precio, en caso de alcanzarse, indicaría que habría que cerrar la operación (con ganancias). Al igual que Stop, será 0 mientras no abramos una operación.
- *Estado*: Utilizaremos esta variable para determinar el estado en el que nos encontramos. Podrá tener los siguientes valores:
  - Largo: Se han cumplido las condiciones de la estrategia que nos hacen esperar que el valor va a aumentar.
  - Corto: Se han cumplido las condiciones de la estrategia que nos hacen esperar que el valor va a disminuir.
  - Abierta: Se ha abierto una posición y quedamos a la espera de cerrarla. No diferenciamos entre si hemos abierto una posición en corto o en largo, para no complicar el algoritmo, ni repetir código de forma innecesaria.
  - Neutral: No nos encontrarnos en ninguna situación anterior.
- *Precio*: Valor del activo en cada momento.
- *Media_Movil*: En esta variable se almacenará el valor de la media móvil en cada lectura de precios.
- *Precio_Apertura*: Precio que tiene el valor en el momento de abrir una operación, lo utilizaremos para calcular el beneficio o la pérdida al cerrar la posición.
- *Tipo*: Al abrir una posición almacenaremos en esta variable el valor *Estado* —que se corresponderá con el tipo de señal actual— antes de que cambie a 'Abierta', para así determinar cuándo cerrar la operación en los siguientes minutos.
- *Valor_Acumulado*: Beneficio en puntos acumulado. Se modificará en el momento del cierre de cada operación de compra-venta.

En los subprocesos *reiniciar_valores* y *abrir_posicion* se establecen valores para las variables anteriores en diferentes momentos de la ejecución. Éstos serán tenidos en cuenta para la posterior toma de decisiones.

En el subproceso principal *procesar_informacion_por_minuto*, que podemos ver en el algoritmo 4.2, se ejecuta la estrategia haciendo uso de las variables definidas anteriormente, y los subprocesos correspondientes.

Comenzamos obteniendo el precio actual del activo. Seguidamente se obtiene el histórico de valores y se calcula con él la media móvil con 20 pasos o minutos. Para calcular las bandas de Bollinger hacemos uso de la ecuación 2.2 para la banda Bollinger superior y la 2.3 para la banda Bollinger inferior, utilizando la media móvil recién calculada.

Seguidamente, obtenemos las medias móviles a corto y largo plazo, para utilizarlas como indicadores de cambios de tendencia. Para calcular la media móvil a corto plazo se utilizarán

los últimos 100 minutos, mientras que para la media móvil a largo plazo se utilizarán los últimos 200 minutos. Estos valores se han escogido después de realizar diferentes pruebas con distintos valores, obteniendo con éstos los mejores resultados. Usaremos estas medias para poder determinar el tipo de posición que podremos esperar con el siguiente criterio:

- Si la media móvil a corto plazo es superior a la media móvil a largo plazo y el valor actual es mayor que la banda de Bollinger superior, suponemos que el valor bajará. La tendencia en este caso es bajista y por tanto nos encontramos con una señal de corto.
- Si la media móvil a corto plazo es inferior a la media móvil a largo plazo y el valor actual es menor que la banda de Bollinger inferior, suponemos que el valor subirá. La tendencia en este caso es alcista y por tanto nos encontramos con una señal de largo.

En los siguientes minutos, y dependiendo del estado en el que nos encontremos, se podrán dar los siguientes escenarios:

- Señal de largo: Si el precio actual está por debajo de la banda de Bollinger superior entonces podemos abrir una posición en corto, esperando que el valor descienda al menos hasta la media móvil actual, es decir, hasta la banda de Bollinger media.
- Señal de corto: En caso de que el precio actual sea mayor que la banda de Bollinger inferior, podemos abrir una posición en largo esperando que el valor ascienda al menos hasta la media móvil actual.

Cuando abramos una operación estableceremos un valor en *Límite*, que será el que pronosticaremos que va a tener; como venimos diciendo, se corresponderá con el valor de *Media_Movil*. Por otro lado, obtendremos el valor a utilizar en caso de que el precio se haya desviado lo suficiente como para que perdamos la confianza en que la operación se realice con éxito. Calcularemos este valor de *Stop* de la siguiente forma:

$$stop = precio \pm (precio\,\%stoploss) \tag{4.1}$$

Donde $stoploss$ sería un porcentaje que estableceremos a mano. La operación a realizar dependerá del tipo de operación que se vaya a abrir. Tras realizar diferentes pruebas consideramos que un 1 % es un valor suficiente para obtener resultados adecuados.

Los criterios para cerrar la operación dependerán del tipo de posición que hayamos abierto:

- Posición en largo:
  - El precio actual es igual o mayor que el límite establecido y podemos cerrar la posición con ganancias.
  - El precio actual es inferior al valor de *stop* establecido y debemos cerrar (con pérdidas).

- Posición en corto:
  - El precio actual es igual o inferior al valor del *límite* establecido y podemos cerrar la posición con ganancias.
  - El precio actual es superior al establecido en stop y debemos cerrar (con pérdidas).

Si se da alguna de estas situaciones, cerramos la posición y actualizamos el valor acumulado con el resultado de la operación, ya sea positivo o negativo.

Algoritmo 4.2 Estrategia de *scalping*: Subproceso principal.

```
Subproceso procesar_informacion_por_minuto
  Precio = obtener_precio( valor = Stock )
  valores_historicos = obtener_datos_historicos( valor = Stock, minutos = 20 )

  Media_Movil = obtener_media_movil( valores_historicos )
  banda_bollinger_superior = Media_Movil + ( desviacion_standard( valores_historicos ) * 2 )
  banda_bollinger_inferior = Media_Movil - ( desviacion_standard( valores_historicos ) * 2 )

  media_movil_corta = obtener_media_movil( minutos = 100 )
  media_movil_larga = obttener_media_movil( minutos = 200 )

  Si Estado != Abierta Entonces
    Si media_movil_corta > media_movil_larga Y Precio > banda_bollinger_superior Entonces
      establecer Posicion a CORTO
    Sino Si media_movil_corta < media_movil_larga Y Precio < banda_bollinger_inferior Entonces
      establecer Posicion a LARGO
    Fin Si
  Fin Si

  Si precio <= banda_movil_superior Y Estado = CORTO Entonces
    vender( valor = Stock )
    abrir_posicion( STOP = Precio + 1%, TIPO = CORTO )
  Sino Si precio >= banda_movil_inferior Y Estado = LARGO Entonces
    comprar( valor = Stock )
    abrir_posicion( STOP = Precio - 1%, TIPO = LARGO )
  Fin Si

  Si  Estado = Abierta Entonces
    Si Tipo = CORTO Entonces
      Si Precio <= Limite Entonces
        cerrar_posicion( valor = Stock )
        establecer Valor_Acumulado a Valor_Acumulado + ( Precio_Apertura - Precio )
        reiniciar_valores
      Sino Si Precio > Stop Entonces
        cerrar_posicion( valor = Stock )
        establecer Valor_Acumulado a Valor_Acumulado - ( Precio - Precio_Apertura )
        reiniciar_valores
      Fin Si
    Fin Si
    Si Tipo = LARGO Entonces
      Si Precio >= Límite Entonces
        cerrar_posicion( valor = Stock )
        establecer Valor_Acumulado a Valor_Acumulado + ( Precio - Precio_Apertura )
        reiniciar_valores
      Sino Si Precio < Stop Entonces
        cerrar_posicion( valor = Stock )
        establecer Valor_Acumulado a Valor_Acumulado - ( Precio_Apertura - Precio )
        reiniciar_valores
      Fin Si
    Fin Si
  Fin Si
Fin Subproceso
```

4.1.2. Estrategia de *scalping* con VaR

En el segundo experimento vamos a implementar una comprobación sencilla del valor en riesgo (VaR) para determinar si debemos salir de una operación abierta, eliminando de esa forma la comprobación del *stop loss*. Una vez hayamos comprobado que el VaR nos puede ser útil, intentaremos mejorar los resultados entrenando (mediante una SVM) una función de regresión que nos permita predecir valores y obtener un VaR más eficiente.

Para determinar el VaR lo haremos teniendo en cuenta la variación de precio en cada paso, normalizando los datos, y obteniendo el cuantil que utilizaremos para calcular la pérdida esperada, como vimos en el apartado 2.2.6. Sin embargo, puesto que en nuestro estudio utilizaremos este concepto tomando como pérdidas y ganancias los cambios de valor de un activo entre minutos, simplificaremos el cálculo de la pérdida esperada utilizando la media aritmética de los valores de la distribución menores que el VaR, es decir, para todo $x < VaR_\alpha(X)$ con un nivel de confianza de $100(1-\alpha)$:

$$ES_\alpha(X) = \frac{\sum_{i=1}^{N} x_i}{N}\ ,\ x < VaR_\alpha(X) \tag{4.2}$$

La decisión de utilizar esta forma de calcular el ES se ha tomado tras realizar diferentes pruebas y comprobar que de esta forma se conseguían los mejores resultados.

Una vez calculada la pérdida esperada, la compararemos con el cambio porcentual de una operación abierta y, dependiendo del tipo de operación, podremos determinar si el riesgo es asumible o no y, por tanto, cerrar la operación o mantenerla abierta. Utilizaremos este valor para determinar cuándo la diferencia entre el valor de apertura de una posición y el valor actual han superado el umbral de pérdidas, y de esta forma cerrar la posición como si de un *stop loss* se tratara.

Puesto que vamos a realizar operaciones en corto y en largo, el cálculo del VaR habrá que realizarlo en ambos sentidos, es decir, el VaR normal se considerará negativo para las posiciones en largo en las que un retroceso del precio suponen pérdidas, pero para las posiciones en corto debemos utilizar este mismo valor en positivo, ya que en esta situación las pérdidas vienen con el incremento del precio.

En la figura 4.2 podemos ver cómo afectaría el uso del VaR al proceso de abrir y cerrar operaciones.

Como podemos observar, sustituimos el uso del valor del *stop* por la comparación del cambio porcentual del precio —con respecto al valor que tenía en el momento de abrir la operación— con el valor de la pérdida esperada, es decir, mientras que en la estrategia anterior cerraríamos en pérdidas si el precio hubiera alcanzado un valor determinado, en este caso cerraremos si el cambio porcentual del precio es mayor o menor que la pérdida esperada dependiendo del estado en el que nos encontremos. Como veremos más adelante, cerrar una operación de esta forma no implica que cerremos en pérdidas necesariamente.

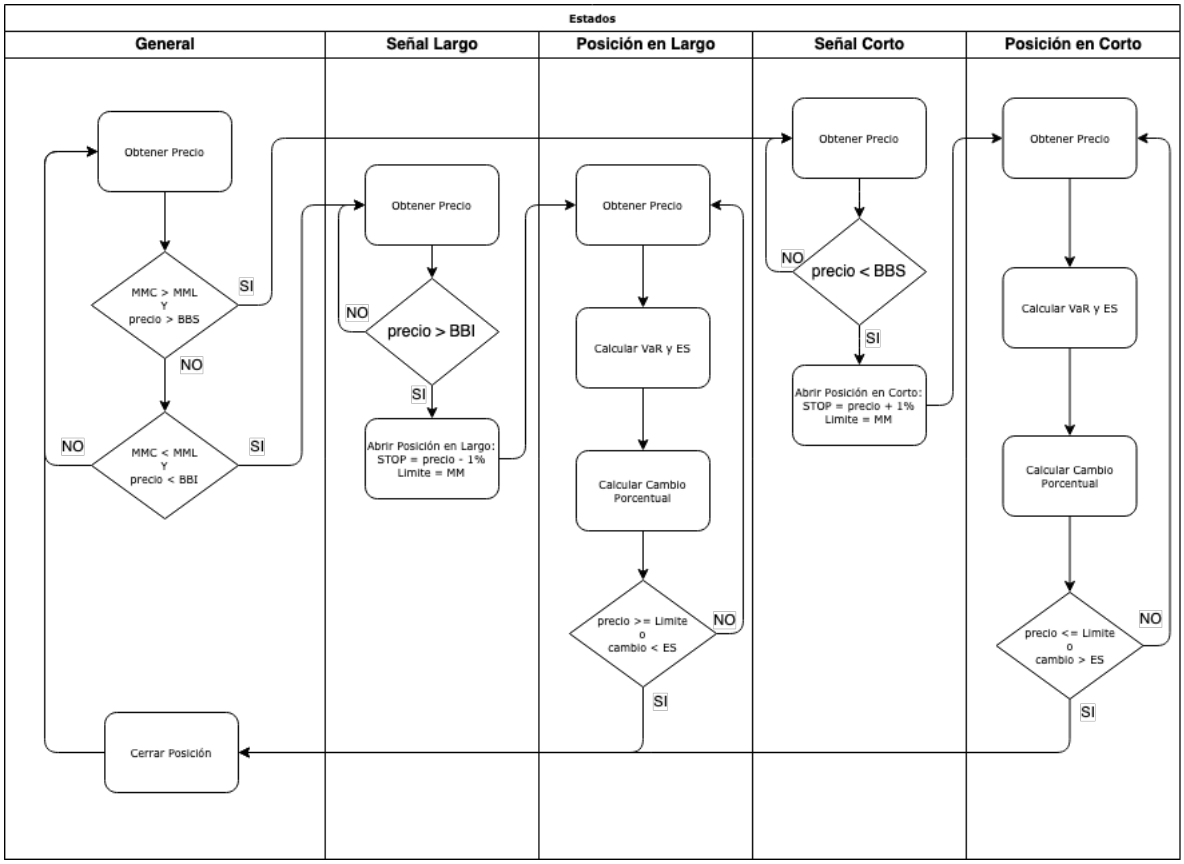

Figura 4.2: Diagrama de estrategia de *scalping* con uso de VaR.

En el algoritmo 4.3 podemos ver en pseudocódigo como se utilizaría el VaR para salir de una operación ya abierta, que se implementaría al principio de la función *procesar_informacion_por_minuto* que ya vimos en el algoritmo 4.2. Además, se eliminaría la comparación del *Precio* con *Stop*.

Algoritmo 4.3 Utilizando VaR en Estrategia de *scalping*.

```
clase Scalping
  ...

  Subproceso procesar_informacion_por_minuto
    ...
    Si Posicion = Abierta Entonces
      cambios_historicos = obtener_cambios_historicos ( minutos = 1440 )
      datos_normalizados = normalizar( cambios_historicos )
      VaR_95 = obtener_percentil( datos = datos_normalizados, percentil = 0.95 )
      cambios_percentil_95 = cambios_historicos < VaR_95
      perdida_esperada = media( cambios_percentil_95 )
      cambio_actual = ( Precio - Precio_Apertura ) / Precio_Apertura * 100

      Si cambio_actual < perdida_esperada Y Tipo = LARGO Entonces
        cerrar_posicion( valor = Stock )
        reiniciar_valores
      Sino Si cambio_actual > perdida_esperada Y Tipo = CORTO Entonces
        cerrar_posicion( valor = Stock )
        reiniciar_valores
      Fin Si
    Fin Si
    ...
  Fin Subproceso
```

Como podemos comprobar, hacemos el cálculo teniendo en cuenta los cambios porcentuales de los últimos 1.440 minutos, es decir, las últimas 24 horas. Una vez normalizados, podemos calcular el VaR que se corresponderá con el cuantil 0,95 de la función. Seguidamente calculamos el ES obteniendo la media de todos los valores históricos que son menores que el cuantil recién calculado. Utilizamos entonces este ES para comparar con el cambio porcentual entre el valor de apertura de la operación y el precio actual.

Cerraremos la operación si se da alguna de las siguientes condiciones:

- Si la operación es en largo y el cambio porcentual actual es menor que el ES.
- Si la operación es en corto y el cambio porcentual actual es mayor que el ES.

Calculamos el ambos casos el resultado de la operación para aplicarlo al valor acumulado y reiniciamos los valores.

4.1.3. Estrategia de *scalping* con VaR optimizado con SVR

Como experimento final, implementaremos una solución para calcular el VaR haciendo uso de aprendizaje automático. En este caso utilizaremos un algoritmo de regresión SVR que entrenaremos con los resultados obtenidos de aplicar el algoritmo de *scalping* durante el periodo comprendido entre el 6 y el 10 de diciembre de 2013. En el algoritmo 4.4 podemos observar las modificaciones realizadas al algoritmo 4.3 y cómo utilizaremos este regresor para predecir valores de cambios porcentuales futuros y añadirlos a los datos que posteriormente normalizaremos para calcular la pérdida esperada y tenerla en cuenta a la hora de salir o no de una operación abierta.

Algoritmo 4.4 Utilizando VaR con SVR en Estrategia de Scalping.

```
clase Scalping
  ...
  establecer Regresor a SVR

  Subproceso procesar_informacion_por_minuto
    ...
    Si Posicion = Abierta Entonces
      cambios_historicos = obtener_cambios_historicos ( minutos = 1440 )
      cambios_predichos = Regresor->predecir( cambios_historicos )
      Añade cambios_predichos A cambios_historicos
      datos_normalizados = normalizar( cambios_historicos )
      VaR_95 = obtener_percentil( datos = datos_normalizados, percentil = 0.95 )
      cambios_percentil_95 = cambios_historicos < VaR_95
      perdida_esperada = media( cambios_percentil_95 )
      cambio_actual = ( Precio - Precio_Apertura ) / Precio_Apertura * 100

      Si cambio_actual < perdida_esperada Y Tipo = LARGO Entonces
        cerrar_posicion( valor = Stock )
      Sino Si cambio_actual > perdida_esperada Y Tipo = CORTO Entonces
        cerrar_posicion( valor = Stock )
      Fin Si
    Fin Si
    ...
  Fin Subproceso
```

4.2. Implementación

A continuación se procederá a la implementación de las estrategias en *zipline* como se procedió en los experimentos realizados para la estrategia *comprar y mantener*. Tras la implementación se ejecutarán las diferentes estrategias y se obtendrán resultados que se estudiarán y compararán en el apartado siguiente.

4.2.1. Preparación del entorno

Al igual que hicimos para la estrategia *comprar y mantener* (cf. sección 3.2.1), preparamos el entorno para la implementación utilizando *zipline*. Seguiremos el mismo patrón utilizado en la estrategia *comprar y mantener*, implementando las funciones *initialize* y *handle_data* con el fin de establecer los parámetros iniciales y ejecutar cada estrategia.

En este caso no podemos leer los datos del DAX directamente desde *Quandl*, ya que este valor no se encuentra entre los disponibles en su versión gratuita. Utilizaremos como fuente de datos un fichero CSV con los datos correspondientes a un periodo de tiempo escogido al azar. Para utilizarlo implementaremos la función *prepareCSV* como en el código A.3.1 y la utilizaremos como fuente de datos en la función *run_algorithm* mediante el parámetro *data.*

Una vez disponemos del entorno adecuado y de toda la información necesaria, podemos proceder con la implementación de las estrategias.

4.2.2. Implementación de la estrategia de *scalping* simple

Como puede verse en el código A.3.2, comenzamos estableciendo algunos valores estándar para el cálculo de los diferentes indicadores. Concretamente, *ma1* y *ma2* son los minutos a tener en cuenta para calcular las dos medias móviles que vamos a utilizar, es decir, la media móvil 1 será la media de los últimos 10 minutos, mientras que la media móvil 2 será la media de los últimos 25 minutos. Por otro lado, *steps* será la cantidad de minutos para calcular las bandas de Bollinger, así como la media y la desviación estándar. Establecemos también el porcentaje de *stop_loss*, es decir, el porcentaje de pérdida que estamos dispuestos a asumir y sobre el que se cerrarán las posiciones para evitar más pérdidas. Por último, establecemos en 2 el número de desviaciones estándar alejadas de la media utilizado para calcular las bandas de Bollingersegún las ecuaciones 2.2 y 2.3. Este valor lo escogemos tras realizar diferentes ejecuciones del experimento y obtener los mejores resultados.

En la función *initialize* establecemos los valores iniciales de las variables relacionadas con el contexto, como el *stock* a manejar. Inicializamos un contador *burndown* para no comenzar a operar hasta que no tengamos los suficientes datos históricos necesarios para calcular los diferentes indicadores. De esta forma, hasta que *burndown* no supere al número estableciendo en *steps*, no se realizarán cálculos ni se operará de ninguna forma.

Para la implementación de la estrategia necesitaremos ir almacenando el estado de ciertas situaciones —como ya vimos en la etapa de diseño (4.1.1)— ya que dependiendo de las diferentes situaciones que pueden ocurrir, deberemos realizar una acción u otra. Al igual que hacíamos con la variable *Estado* en la fase de diseño, en este caso utilizaremos la variable *position* para almacenar el tipo de situación en la que nos encontramos en cada momento, es decir:

- *long*: Se han cumplido las condiciones de la estrategia que nos hacen esperar que el valor va a aumentar.
- *short*: Se han cumplido las condiciones de la estrategia que nos hacen esperar que el valor va a disminuir.
- *trade*: Se ha abierto una posición y quedamos a la espera de cerrarla.
- *None*: Ninguna de las anteriores.

De esta forma podremos tomar decisiones en caso de que hayamos detectado una señal de cualquier tipo, así como salir de posiciones abiertas, tanto en ganancia como en pérdida por *stop loss*.

Para poder calcular el valor conseguido o perdido en cada operación antes de cerrarla, almacenaremos en la variable *open* el valor del precio actual en el momento de abrir una operación para de esa manera poder compararla con el valor del cierre. Por otro lado, cuando abramos la operación estableceremos en *limit* la media móvil, que será el precio que pronosticaremos que va a tener. Una vez abierta la operación, si se llegara a ese valor se cerraría la posición.

Puesto que el valor actual del precio puede pasar a ser superior o inferior a este límite establecido, deberemos saber si la posición que se ha abierto es de tipo largo o corto, para determinar cuando cerrar la posición. Para ello almacenaremos en la variable *kind* el tipo de operación en el momento de abrirla, y la iremos comprobando a cada paso mientras la posición permanezca abierta.

Haciendo uso del porcentaje de *stop_loss* establecido con anterioridad, calcularemos el valor de *stop* al abrir la posición. Este valor lo utilizaremos más adelante para determinar si hemos llegado al punto en el que tenemos que cerrar con pérdidas.

En caso de que el precio no varíe demasiado es posible que pasemos muchos minutos en la posición abierta y no se alcance ningún valor de salida establecido. Para evitarlo, controlaremos el tiempo de duración de una posición almacenando el valor de la variable *burndown* en la variable *time* en el momento en el que se abre una posición. Podremos utilizar este valor para cerrar la posición cuando hayan transcurrido unos determinados minutos. Después de realizar diferentes pruebas establecemos en 600 minutos la duración máxima que una operación puede permanecer abierta. En otro posible estudio se podría determinar este valor de una forma más eficiente calculando el tiempo que genere mejores resultados, pero no es el objeto de este trabajo y solo lo utilizaremos para evitar posibles "atascos".

Por último, en la variable *value* iremos almacenando el valor acumulado de puntos en el momento del cierre. Este valor también nos servirá para dibujar la gráfica de resultados, junto con la variación del precio.

En nuestra función *handle_data*, que podemos ver en el código A.3.3, comenzamos comprobando si tenemos la cantidad suficiente de información como para obtener los valores que necesitamos para calcular las medias móviles y las bandas de Bollinger. Una vez tengamos suficientes datos, podemos comenzar a calcular valores y tomar decisiones.

Para comenzar, obtenemos el precio actual en caso de que exista. En algunos conjuntos de datos existen minutos en los que no se establece ningún valor, por lo que en ese momento no podríamos tomar ninguna decisión con respecto al precio. Por ello, en caso de que en el momento que se está teniendo en consideración no tuviéramos datos, pasaríamos al siguiente, hasta obtener un valor adecuado.

Una vez tenemos un precio comenzamos obteniendo el histórico de valores y calculamos con él la media móvil (con 20 pasos o minutos) y la almacenamos en la variable *ewm.*

Para calcular las bandas de Bollinger hacemos uso de las ecuaciones 2.2 y 2.3, utilizando la media móvil recién calculada. Seguidamente calculamos las medias móviles a corto y largo plazo para utilizarlas como indicadores de cambios de tendencia. Utilizaremos estos valores para poder determinar el tipo de posición que podremos esperar con el siguiente criterio:

- Si $short\_term > long\_term$ y $current\_price > bhi \Rightarrow$ tendencia bajista y por tanto señal de posible entrada en corto.
- Si $short\_term < long\_term$ y $current\_price < blw \Rightarrow$ tendencia bajista y por tanto señal de posible entrada en corto.

En estas expresiones, *short_term* es la media móvil a corto plazo, *long_term* la media móvil a largo plazo, *bhi* la banda Bollinger superior, *blw* la banda Bollinger inferior y *current_price* el precio actual.

Una vez hemos calculado los valores iniciales necesarios para tomar decisiones, comenzamos a determinar si es el momento de abrir posiciones (mediante las comprobaciones del código A.3.4), dependiendo del tipo de señal que hayamos detectado.

Señal de corto

Si el precio actual está por debajo de la banda de Bollinger superior y anteriormente hemos detectado una tendencia bajista, entonces podemos abrir una posición en corto esperando que el valor descienda —al menos— hasta la media móvil actual, es decir, hasta la banda de Bollinger media.

Para no invertir todo el dinero de la cuenta en una misma operación, antes de realizar cualquier acción de compra calculamos el total de acciones que nos podemos permitir, y la dividimos por tres para invertir solo un tercio de nuestras posibilidades. Esto lo hacemos porque

que *zipline* no dispone de mecanismos que nos impidan invertir dinero una vez que nuestro dinero disponible haya llegado a cero.

Calculamos el valor del *stop loss* que introduciremos en la operación, sumando en la ecuación 4.1. En caso de alcanzarse este valor cerraríamos la operación con pérdidas, pues el precio se ha desviado lo suficiente como para que perdamos la confianza en que la operación se realice con éxito.

Haciendo uso de la función *order*[1] abrimos la operación en DAX con la cantidad de acciones calculada anteriormente (en negativo, por ser una operación de venta).

Establecemos el punto de venta o límite de la operación en la media móvil actual y lo almacenamos junto con el resto de información global para tenerla en consideración en las siguientes lecturas.

Señal de largo

En caso de que el precio actual sea mayor que la banda de Bollinger inferior y hayamos determinado una tendencia alcista anteriormente, podemos abrir una posición en largo esperando que el valor ascienda al menos hasta la media móvil actual.

En este caso realizaremos las mismas acciones que para una operación en corto, con la diferencia de que esta vez el *stoploss* lo calcularemos restando en la ecuación 4.1, y en la función de creación de operación el número de acciones a comprar serán de tipo positivo.

Cierre de posición

Una vez hemos lanzado una operación, podemos comprobar si ha llegado el momento de cerrarla (mediante las comprobaciones del código A.3.5). En general cerraremos la posición si se cumple alguna de las siguientes condiciones:

- El precio actual es igual o mayor que el valor establecido en la variable de contexto fijada para el limite, y tenemos una operación abierta en largo.
- El precio es igual o inferior al valor del límite y tenemos una operación abierta en corto.

Para cerrar una operación utilizamos la función *order* con 0 como número de acciones a comprar. Esta acción cerraría todas las posiciones abiertas para el activo (en inglés *asset*) establecido.

Para determinar el valor de la operación comprobaremos la diferencia entre el valor de apertura de la operación y el actual dependiendo de si la operación se ha realizado en corto o en largo.

Nótese que mientras que el resto de valores contextuales se reinician a sus valores iniciales, el valor del tipo de posición lo establecemos en el tipo de posición actual. Esto lo hacemos

[1] *https://www.zipline.io/appendix.html#zipline.api.order*

así porque, aunque hayamos llegado a la media móvil exponencial establecida como límite, la tendencia en principio es la misma, por lo que el precio podría seguir bajando o subiendo y en el siguiente paso se podría determinar abrir otra posición en la misma dirección que la que acabamos de cerrar.

En caso de que ninguna de las condiciones anteriores se cumpla, comprobamos si nuestra operación ha llegado al *stoploss* y por tanto debemos cerrarla. Las condiciones a tener en cuenta son las siguientes:

- El precio es inferior al valor de *stop* y estamos en una operación en largo.
- El precio es superior al establecido en *stop* y tenemos una posición en corto.

Si se da alguna de estas situaciones, cerramos la posición y calculamos el valor de la pérdida, restándosela a la variable contextual del valor y reiniciando el resto de variables a sus valores iniciales.

Para finalizar, dibujamos una gráfica comparativa con el precio que ha ido tomando el activo y el valor de la cantidad de puntos obtenidos, es decir, la variable *value*. Hacemos uso de la función *analyze* del código A.3.6 para dibujar las gráficas, así como de los valores que hemos ido almacenando mediante la función *record* [2] en RETURNS y PRICE.

4.2.3. Implementación de la estrategia de *scalping* con VaR

Para implementar el uso del VaR en nuestra estrategia de *scalping* (segundo experimento) comenzaremos eliminando la comprobación de *stop loss* de nuestro algoritmo anterior. Podemos encontrar esta comprobación entre las líneas 17 y 28 del código A.3.5. Seguidamente añadiremos una variable de contexto a nuestra función *initialize* en la que almacenaremos el valor anterior del precio, para así determinar la variación actual y poder mantener un histórico de cambios porcentuales que utilizaremos para calcular el VaR. Todo este pequeño proceso lo realizaremos añadiendo el código A.3.7 al final de las funciones *initialize* y *handle_data*.

Una vez mantenemos el histórico de cambios, cuando hayamos alcanzado suficiente cantidad de datos podemos comenzar a calcular el VaR. El código A.3.8 muestra cómo calculamos el VaR y tomamos las decisiones oportunas en caso de que tengamos suficientes datos y exista una operación abierta.

Como vimos en el diseño de la estrategia, hacemos el cálculo teniendo en cuenta los cambios porcentuales almacenados en la variable contextual *historical_changes* de los últimos 1440 minutos. Mediante las funciones *mean*[3] y *std*[4] calculamos la media y la desviación estandar que utilizaremos para obtener un conjunto normalizado con *random.normal*[5] y aplicar la función

[2] *https://www.zipline.io/appendix.html#zipline.api.record*
[3] *https://numpy.org/doc/stable/reference/generated/numpy.mean.html*
[4] *https://numpy.org/doc/stable/reference/generated/numpy.std.html*
[5] https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html

percentile [6], para obtener el cuantil con $\alpha = 0{,}95$ que almacenaremos en *value_at_risk*. Establecemos *expected_shortfall* como la media de todos los valores históricos que son menores que *value_at_risk*. Calculamos el cambio porcentual actual en *change* y lo comparamos con *expected_shortfall:*

Cerraremos la operación si se da alguna de las siguientes condiciones:

- Si la operación es en largo y *change* $<$ *expected_shortfall*.
- Si la operación es en corto y *change* $>$ *expected_shortfall*.

Una vez cerrada la posición, calculamos el resultado de la operación para aplicarlo al valor acumulado y reiniciamos los valores.

4.2.4. Implementación de la estrategia de *scalping* con VaR optimizado con SVR

Al igual que hicimos con el regresor de bosques aleatorios en la sección 3.2.3, en esta ocasión vamos a entrenar un regresor de tipo SVR, por las razones expuestas en la sección 3.1. En este caso no nos detendremos a analizar los pormenores de la obtención de datos y la creación del estimador pues, como podemos observar en el código A.3.9, el proceso es similar al que realizamos para bosques aleatorios, con la salvedad de que en este caso obtendremos los valores históricos de un fichero CSV en lugar de obtenerlos de la base de datos *Quandl*.

Comenzamos importando el fichero *DAX-201312.csv* que contiene los valores que tuvo el DAX de apertura, cierre, máximo, mínimo y volumen por cada minuto entre el 6 y el 10 de diciembre de 2013. Al igual que el periodo de ejecución de la estrategia, este periodo se ha seleccionado al azar, procurando que fuera anterior al que se usará durante el experimento. En la tabla 4.1 podemos observar las cinco primeras filas de este fichero.

date	open	high	low	close	volume
2013-12-06 09:30:00	9174.5	9175.0	9170.5	9172.5	166
2013-12-06 09:31:00	9172.0	9172.5	9168.0	9171.5	162
2013-12-06 09:32:00	9171.0	9171.0	9168.5	9169.5	113
2013-12-06 09:33:00	9170.0	9173.0	9169.0	9170.5	159
2013-12-06 09:34:00	9170.5	9174.5	9170.5	9174.0	102

Cuadro 4.1: Primeras filas de DAX-201312.csv.

Tras establecer como índice de datos la columna 'date', nos quedamos con las columnas 'open' y 'close', y con las primeras 1.440 filas, que serían las equivalentes 24 horas de datos. Creamos una columna nueva 'lag_0', en la que introduciremos el cambio porcentual de cada minuto. Una vez eliminados los elementos no numéricos, eliminamos las columnas 'open' y

[6] https://numpy.org/doc/stable/reference/generated/numpy.percentile.html

'close', que ya no nos hacen falta. Así podemos generar una tabla en la que cada fila contendrá el valor del cambio porcentual del minuto concreto en la primera columna, y en las 19 siguientes los valores correspondientes a los 19 minutos posteriores. En la tabla 4.2 podemos ver un ejemplo de las últimas filas de la matriz de datos una vez eliminados los valores no numéricos.

Con la matriz de datos así preparada podemos obtener los conjuntos de entrenamiento y test que utilizaremos para entrenar el algoritmo de regresión. De esta forma dejaríamos X con un total de 15 características, e Y con 5. Inicializamos el regresor *MultiOutputRegressor*[7] con la clase SVR[8] y establecemos los parámetros del kérnel (de la SVM) a testear. Probaremos con lineal, polinómico y de base radial (del inglés *radial basis function kernel, RBF*), y cada uno con un valor de C entre 1 y 10, que será el número de conjuntos en los que se dividirán los datos. Escogemos estos valores tras realizar diferentes pruebas con otros rangos y obtener con éstos los resultados más óptimos. Seguidamente, para evaluar la calidad de las predicciones, creamos un validador cruzado con *GridSearchCV*[9] con series temporales mediante *TimeSeriesSplit* [10], tomando 5 divisiones, para de esa forma poder inicializar una búsqueda del mejor estimador conforme a los parámetros de núcleo introducidos.

Almacenamos el estimador generado en un fichero para posteriormente poder utilizarlo durante la ejecución del algoritmo de *scalping*.

date	lag_0	lag_1	...	lag_18	lag_19
2013-12-10 05:16:00	-0.016280	-0.016283	...	-0.010854	0.010855
2013-12-10 05:17:00	-0.016283	0.005429	...	0.010855	-0.021707
2013-12-10 05:18:00	0.005429	0.010856	...	-0.021707	0.005427
2013-12-10 05:19:00	0.010856	0.000000	...	0.005427	-0.005427
2013-12-10 05:20:00	0.000000	0.010856	...	-0.005427	0.000000

Cuadro 4.2: Matriz de datos.

Para aplicar las predicciones del regresor entrenado en el apartado anterior, cargaremos el fichero *joblib* final en una variable contextual en la función *initialize*, que llamaremos *regressor*. En el código A.3.10 podemos ver los cambios introducidos para calcular el VaR. En este caso utilizamos el regresor para predecir los próximos 5 minutos a partir de los 15 minutos anteriores. Utilizando la función *concatenate* [11], añadiremos estos 5 minutos al conjunto de valores históricos de las últimas 24 horas. De esta forma calcularemos el VaR con la media y la desviación estándar de esta nueva colección de datos y seguiremos la misma estructura de ejecución que en el experimento anterior (cf. sección 4.2.2).

[7] *https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html*
[8] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
[9] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[10] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
[11] https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html

4.3. Evaluación y resultados

En el gráfico de la figura 4.3 podemos observar los resultados obtenidos con la ejecución de la estrategia de *scalping* simple.

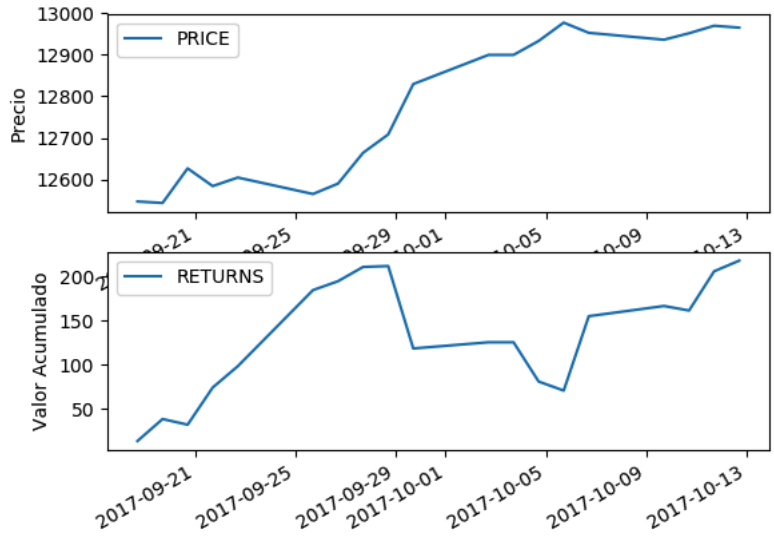

Figura 4.3: Resultados de la estrategia de scalping.

Como podemos observar, hemos obtenido unos resultados positivos a lo largo del tiempo, a pesar de que entre el 29 de septiembre y el 5 de octubre de 2017 se generaron unas pérdidas de más de 100 puntos. En la tabla 4.3 podemos comprobar cómo la mayor parte de las operaciones negativas han sido operaciones en corto, mientras que un alto porcentaje de las operaciones en largo han dado resultados positivos, compensando suficientemente las operaciones negativas. Además, las operaciones en largo han durado menos de la mitad que las operaciones en corto.

	Total	Positivas	Negativas	Duración $\overline{x}$	Rendimiento $\overline{x}$	Beneficios	Pérdidas	Resultado
Largo	90	77	13	46.55 min	3.72	500	-165.71	334.29
Corto	142	87	55	115.37 min	-0.82	585.5	-701.40	-115.89
Total	232	164	68	88.68 min	1.875	1085.5	-867,10	218.40

Cuadro 4.3: Estadísticas de las operaciones realizadas mediante estrategia de scalping.

Una vez implementada una gestión del riesgo mediante VaR y ejecutada la estrategia obtenemos los resultados de la gráfica de la figura 4.4. Como podemos observar, al compararla con la figura 4.3 comprobamos cómo, aunque no alcanzamos los valores que obtendríamos sin aplicar el VaR, no se generan unas pérdidas tan significativas como si no se aplicara, por lo que permitiría obtener beneficios más discretos pero con algo de seguridad. Esto es debido principalmente a que, mientras que la estrategia simple esperaba a llegar al valor de *stop* para cerrar las operaciones fallidas, con la gestión del riesgo hemos podido cerrarlas mucho antes.

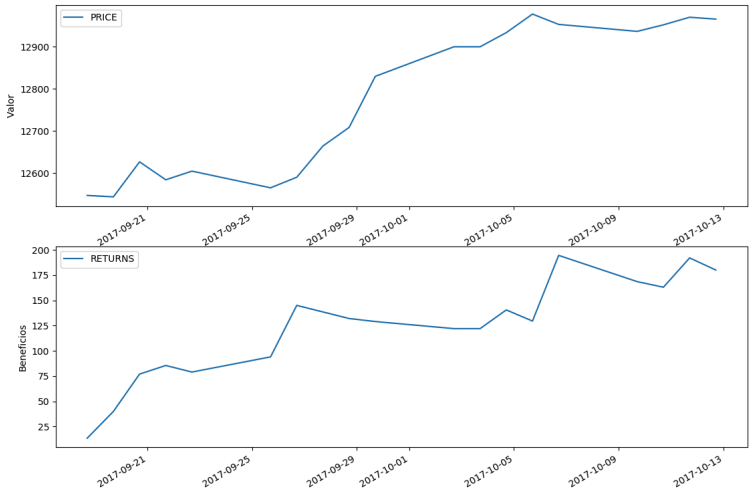

Figura 4.4: Resultados de *scalping* con VaR histórico.

En la tabla 4.4 tenemos algunas estadísticas de las operaciones realizadas durante el experimento. Las operaciones en corto aumentan con respecto a las obtenidas en la tabla 4.3, pero su duración disminuye notablemente, siendo en su inmensa mayoría cerradas por el criterio del VaR al minuto de abrirse. Por otro lado, las operaciones en largo también aumentan, pero disminuye el resultado final casi a la mitad, así como el beneficio medio por operación.

Obtenemos como resultado 38 puntos menos que los obtenidos con la estrategia de *scalping* sin VaR. Sin embargo se puede apreciar cómo los resultados negativos han sido paliados en su mayoría por las salidas rápidas provocadas por el cálculo del VaR, haciendo que tanto el beneficio medio como los resultados finales sean positivos de forma casi constante.

En este caso las operaciones negativas no implican un resultado negativo (como los obtenidos en la aplicación de la estrategia sin VaR), ya que las operaciones negativas se corresponden con aquellas que no han llegado a valor esperado en su ejecución y se han cerrado por comparación con el VaR, lo que implica que aún al cerrar antes de lo esperado el resultado pueda haber llegado a ser positivo.

	Total	Positivas	Negativas	Duración $\overline{x}$	Rendimiento $\overline{x}$	Beneficios	Pérdidas	Resultado
Largo	105	73	32	32.82 min	1.6	460	-292.5	167.5
Corto	253	4	249	6.6 min	0.05	32	-19.5	12.5
Total	359	77	281	14.3 min	0.5	492	-312	180

Cuadro 4.4: Estadísticas de las operaciones realizadas mediante la estrategia de *scalping* con VaR.

Finalmente, en la figura 4.5 podemos ver el resultado de la aplicación del VaR utilizando aprendizaje automático, es decir, estimando el VaR mediante SVR. Concretamente, tras introducir los conjuntos de datos de entrenamiento y test comprobamos como la salida de la función *fit* de *GridSearchCV* nos indica que el mejor estimador —que se utiliza en la ejecución de la

estrategia– es el siguiente:

- *MultiOutputRegressor(estimator=SVR(C=1, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale', kernel='poly', max_iter=-1, shrinking=True, tol=0.001, verbose=False), n_jobs=None)*

Como podemos observar, los beneficios aumentan con respecto al uso del VaR histórico, y superan discretamente los de la estrategia de *scalping* sin uso de VaR.

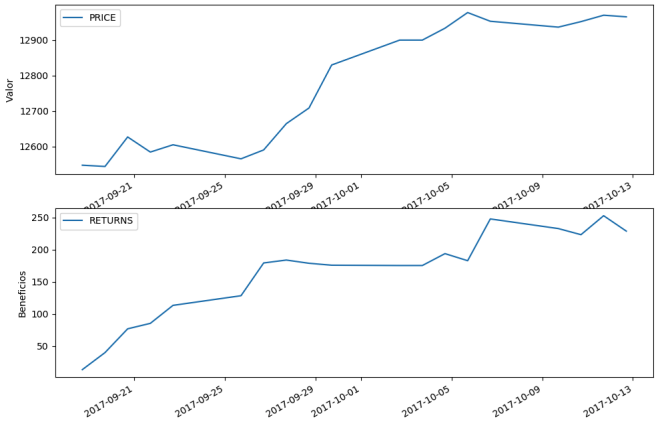

Figura 4.5: Resultados de *scalping* utilizando VaR calculado con SVR.

En la tabla 4.5 podemos comprobar cómo los resultados han mejorado con respecto a los obtenidos en la tabla 4.4. El número de operaciones es prácticamente el mismo en ambas ejecuciones, diferenciándose únicamente en una operación corta. Los resultados de las operaciones en corto no mejoran significativamente, por lo que no podemos esperar que la aplicación de esta técnica nos prevenga de este tipo de operaciones. No obstante, las operaciones en largo obtienen 42 puntos más de beneficio disminuyendo también la duración media de las mismas, por lo que podemos esperar que la aplicación de un estimador SVR en el cálculo del VaR como estrategia de salida en las operaciones nos ayuda a optimizar el cierre de las operaciones. Además, —como podemos comprobar al comparar los resultados con los de la tabla 4.3—, al final del periodo obtenemos mejor resultado sin caer en pérdidas significativas durante el proceso.

	Total	Positivas	Negativas	Duración $\overline{x}$	Rendimiento $\overline{x}$	Beneficios	Pérdidas	Resultado
Largo	105	76	29	32.03 min	1.99	481	-271.5	209.5
Corto	254	4	250	6.6 min	0.05	32	-19	13
Total	360	80	279	14.3 min	0.62	513	-290.5	222.5

Cuadro 4.5: Estadísticas de las operaciones realizadas durante la ejecución de la estrategia de scalping con VaR calculado con SVR.

4.4. Discusión

En la tabla 4.6 podemos comparar los resultados obtenidos en los tres experimentos con estrategia de *scalping*. Entre los cambios más significativos destaca el rendimiento, que es el promedio de los resultados obtenidos en cada operación abierta. La estrategia de *scalping* por sí sola consigue unos resultados finales positivos. No obstante, el riesgo asumido es alto, ya que aunque las operaciones positivas producen más del doble de beneficios que los obtenidos por el resto de estrategias, las operaciones negativas —aunque limitadas por *stoploss*— generan unas pérdidas bastante altas con una media de casi 13 puntos de pérdida por cada operación negativa. Por ello, aunque la estrategia parece funcionar, no debemos olvidar que no es perfecta y podemos generar importantes pérdidas ya que al no contar con ningún sistema de control de riesgo no podemos iniciar esta estrategia con un mínimo de seguridad.

La aplicación del VaR histórico no consigue aumentar los beneficios finales, pero sí disminuye notablemente las pérdidas, haciendo que los resultados sean más progresivos. El cálculo del VaR mediante SVR mejora los beneficios del VaR, y también disminuye el valor medio de las pérdidas en las operaciones negativas dando como resultado unos beneficios similares a la estrategia de *scalping* sin obtener pérdidas tan significativas.

En la figura 4.6 podemos ver una comparación de las gráficas obtenidas tras la ejecución de cada un de las estrategias. Como podemos comprobar la evolución del valor acumulado se suaviza un poco más en cada experimento.

Tras la realización de los experimentos podemos concluir que la utilización de un regresor SVR para el cálculo del VaR como estrategia para disminuir el riesgo en la ejecución de una estrategia de *scalping* a corto plazo puede mejorar notablemente los resultados. No obstante, cabe señalar que todos los algoritmos están sujetos a mejoras que podrían producir mayores ganancias, pero no hemos querido alterar demasiado la ejecución de los mismos para no dificultar su comprensión. Un ejemplo claro sería haber utilizado el VaR para decidir si entrar o no en una operación, en lugar de esperar a abrirla para comprobar si ha sido buena idea o no. Esto nos hubiera ahorrado entrar en muchas operaciones que se han cerrado al minuto siguiente de ser abiertas, pero para su funcionamiento se hubiera tenido que modificar notablemente el algoritmo perdiendo el sentido de los experimentos.

	Total	Positivas	Negativas	Duración $\overline{x}$	Rendimiento $\overline{x}$	Beneficios	Pérdidas	Pérdida $\overline{x}$	Resultado
Scalping	232	164	68	88.68 min	0.94	1085.5	-867.101	12.75	218.40
VaR	359	77	281	14.3 min	0.50	492	-312	1.11	180
SVR	360	80	279	14.3 min	0.62	513	-290.5	1.04	222.5

Cuadro 4.6: Comparación de resultados entre los experimentos de *scalping*.

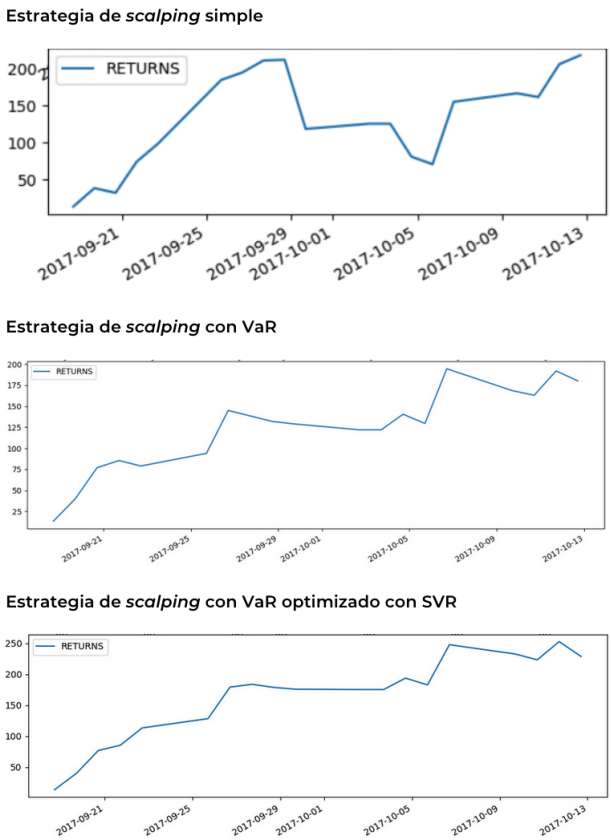

Figura 4.6: Comparación de las gráficas del valor acumulado obtenido en los experimentos de scalping.

Capítulo 5

Conclusiones y trabajos futuros

En este último capítulo presentamos, en primer lugar, las conclusiones obtenidas después de todos los pasos que se han llevado a cabo en la realización de este estudio, y posteriormente mostramos varios posibles trabajos futuros, algunos de los cuales ya se encuentran en pleno desarrollo en el momento de escribir este documento.

5.1. Conclusiones

A lo largo de este trabajo se han seguido los pasos necesarios para alcanzar una serie de objetivos, los cuales se marcaban concretamente en el primer capítulo del presente documento.

El principal objetivo de esta investigación era realizar un estudio de varias estrategias de inversión basadas en algoritmos computacionales, para posteriormente intentar optimizarlas mediante aprendizaje automático. Siguiendo —y cumpliendo— este objetivo hemos conseguido evaluar dos estrategias en diferentes marcos temporales. Por un lado hemos diseñado, implementado y ejecutado una estrategia de inversión a largo plazo, *comprar y mantener*, que posteriormente hemos podido optimizar mediante bosques aleatorios, consiguiendo aumentar el beneficio casi un 7 %. A continuación, hemos realizado el mismo proceso con una estrategia de inversión a corto plazo (*scalping*). En este caso partíamos de una ejecución de estrategia simple a la que posteriormente le hemos añadido gestión de riesgo utilizando la técnica conocida como valor en riesgo (VaR). Esta forma de gestionar el riesgo la hemos podido optimizar con regresión mediante máquinas de vectores de soporte (SVMs), reduciendo así la pérdida en las operaciones fallidas y logrando un aumento progresivo de los beneficios.

Como hemos podido comprobar, la ejecución de estrategias de inversión en entornos computacionales preparados para tal fin, como *zipline*, nos permite analizar si los criterios escogidos para la toma de decisiones que forman nuestra estrategia de inversión nos hubieran reportado beneficios en caso de haberlos ejecutado realmente, al menos, durante el periodo de tiempo escogido para las pruebas. Además, con el fin de mejorar los resultados, estos entornos nos facilitan el realizar cambios y optimizaciones hasta que nuestro conocimiento o capacidad

computacional nos lo permita. No obstante, no podemos olvidar que los resultados pasados no garantizan beneficios futuros, por lo que aunque en nuestras pruebas obtengamos ganancias muy relevantes, es muy probable que al ejecutar esas estrategias en entornos reales no consigamos los mismos beneficios, o incluso que generemos pérdidas importantes en poco tiempo.

La inversión en mercados de valores puede ser una actividad muy lucrativa, pero no está libre de riesgos. Del mismo modo que un inversor puede ganar mucho dinero, también puede perderlo. Intentar predecir de forma efectiva el movimiento de los precios es una tarea casi imposible, como hemos podido comprobar. Sin embargo, hemos demostrado que siguiendo una estrategia simple de inversión, e intentando seguir normas concretas a la hora de comprar o vender, se pueden conseguir rendimientos positivos en mayor o menor medida.

Todas estas acciones que hemos realizado haciendo uso de algoritmos se podrían realizar de forma manual, es decir, utilizando herramientas de inversión sin necesidad de utilizar un algoritmo que las haga por nosotros. No obstante, como ya hemos comentado en apartados anteriores, exponerse a la inversión manual saca a la luz los sesgos conductuales del inversor, que pueden provocar pérdidas irrecuperables, además de provocar un gran estrés y ansiedad. Entre estos sesgos, los más comunes son el exceso de confianza —que nos hace tomar decisiones precipitadas— y la aversión al riesgo —que nos hace cerrar operaciones antes de tiempo—. Antes de escoger las estrategias adecuadas para realizar los experimentos, se realizaron diferentes pruebas en entornos de inversión reales con cuentas de demostración en las que pudimos experimentar todos esos sesgos, entre otras sensaciones desagradables no recomendables.

Una vez hechos los experimentos y obtenidos los resultados, hemos podido comprobar cómo dejando a un algoritmo tomar decisiones por si mismo en base a datos comprobables, los resultados pueden ser positivos. De esta forma hemos alcanzado otro de los objetivos, que era introducirnos en la generación de algoritmos capaces de tomar decisiones por sí mismos en un entorno de inversión financiera, que en un futuro podría servirnos para desarrollar estrategias más eficientes.

Finalmente, tras conocer el mundo de los mercados financieros, hemos podido especular sobre el verdadero flujo del dinero y en qué manos está realmente. Al observar los grandes cambios que se producen de forma violenta en los precios de algunos activos —sobre todo en el mercado de futuros—, y los movimientos que hemos podido comprobar de forma recurrente —haciendo que se rompieran todas las estrategias—, hemos descubierto que algunos grandes inversores son capaces de influir significativamente en los mercados. Esta influencia ha provocado grandes cambios que, evidentemente, solo han podido beneficiar a quienes los han provocado, a los sabían que se iba a producir, o a otros grandes inversores que han entrado en el juego de fuerzas y han conseguido influir en mayor medida, haciendo que los valores se doblegaran a sus propósitos.

Aunque algunas grandes operaciones con mucho volumen puedan provocar cambios significativos, también hemos podido observar cómo esos cambios no han llegado a concluir como deberían, de tal forma que podríamos pensar que ni los grandes inversores son capaces de

manipular los mercados a su voluntad. Sin embargo, un inversor con mucho capital se puede permitir realizar intentos de manipulación fallidos, aunque pierda grandes sumas de dinero, si con los intentos acertados consigue compensar.

Ante los resultados positivos de la estrategia de *scalping*, hemos investigado sobre la mejor forma de aplicar estos algoritmos en entornos reales. Existen numerosas opciones de inversión de forma automática, ya que la mayoría de los entornos de inversión utilizan las mismas herramientas para operar los mercados. La herramienta más común suele ser *MetaTrader*[1], en alguna de sus versiones, la cual opera directamente en los mercados y permite a muchas empresas revender sus servicios. Cuenta con un lenguaje de programación propio, así como con diferentes interfaces de interconexión entre sus sistemas y otras herramientas o lenguajes. Nuestros algoritmos podrían adaptarse de forma sencilla si sustituyéramos el uso de *zipline* por alguna de estas bibliotecas intermedias, o creáramos las nuestras propias.

De este modo hemos logrado nuestro tercer objetivo, que era sentar las bases para trabajos futuros propios que nos permitan obtener mejores resultados. Además, en este mismo objetivo pretendíamos que este estudio sirviera como base para que otros investigadores pudieran adentrarse en la creación de algoritmos de inversión automática aprovechando todos los conocimientos y el tiempo invertido que ha dado como resultado todo este trabajo, ya que sin duda el mayor esfuerzo en toda esta tarea ha sido determinar como conseguir hacer lo que pretendíamos hacer. Puesto que no depende directamente de nosotros que nuestros experimentos puedan servir realmente a alguien en el futuro, consideraremos por logrado el objetivo simulando comprar acciones imaginarias de este documento, confiando en que aumenten de valor con el tiempo.

5.2. Trabajos futuros

Una vez realizados los experimentos con estrategias simples y obtenido unos resultados positivos, el siguiente paso sería seguir investigando estrategias más avanzadas que hagan uso de más información a la hora de tomar decisiones y otro tipo de indicadores que nos permitan obtener mejores resultados. Como ya comentamos en los resultados de los experimentos, hemos querido dejar los algoritmos de las estrategias de la forma más sencilla posible para no complicar su entendimiento, pero existen mejoras en los mismos que permiten tomar decisiones en base a más criterios pero que de implementarlos aumentarían notablemente su complejidad.

Durante todos los experimentos hemos trabajado con entornos simulados, obteniendo información de los mercados de conjuntos de datos pasados y tratándolos como si fueran en tiempo real. Para aplicar las estrategias en entornos reales deberemos utilizar otras herramientas y tecnologías o adaptar las que ya utilizamos para que puedan hacer uso de las plataformas de inversión que nos van a permitir realizar operaciones reales. Entre las tecnologías y entornos

[1] https://www.metatrader5.com/es

que deberemos investigar con el fin de aumentar nuestros conocimientos para poder operar, destacamos las siguientes:

- MQL4/MQL5[2]: Son lenguajes de programación integrados, diseñados para el desarrollo de algoritmos de inversión, obtención de indicadores técnicos y creación de bibliotecas funcionales dentro del software de inversión MetaTrader, que es la plataforma de inversión más utilizada para realizar operaciones en tiempo real. Con MQL4 se podrían aplicar directamente las estrategias, pero probablemente no dispongamos de las herramientas como *scikit learn* ni podríamos realizar predicciones utilizando algoritmos de aprendizaje automático. No obstante, existen agentes intermedios que actúan de comunicador entre este lenguaje y las herramientas que hemos utilizado en nuestros experimentos.
- Oanda[3]: Es una plataforma de inversión que dispone de API para el desarrollo de aplicaciones. De esta forma podríamos desarrollar un algoritmo de inversión que haga uso de esta plataforma para invertir, sin tener que depender de un lenguaje especializado como MQL4.

Debemos, por tanto, evaluar los pros y los contras de las diferentes formas de desarrollar algoritmos funcionales que inviertan en entornos reales, ya sea utilizando lenguajes y tecnologías especificas desarrolladas para tal fin, y seguir utilizando las que hemos podido probar a lo largo de esta investigación.

[2]https://book.mql4.com/content
[3]https://developer.oanda.com/

Apéndice A

Código utilizado en los experimentos

A.1. Preparación del Entorno

```
1  from toolz import merge
2  from zipline import run_algorithm
3  from zipline.utils.calendars import register_calendar, get_calendar
4  from strategies.buy_and_hold import BuyAndHold
5  from os import environ
6  _cols_to_check = [
7    'algo_volatility',
8    'algorithm_period_return',
9    'alpha',
10   'benchmark_period_return',
11   'benchmark_volatility',
12   'beta',
13   'capital_used',
14   'ending_cash',
15   'ending_value',
16   'excess_return',
17   'gross_leverage',
18   'long_value',
19   'longs_count',
20   'max_drawdown',
21   'max_leverage',
22   'net_leverage',
23   'period_close',
24   'period_label',
25   'period_open',
26   'pnl',
27   'portfolio_value',
28   'positions',
29   'returns',
30   'short_value',
31   'shorts_count',
32   'sortino',
33   'starting_cash',
34   'starting_value',
35   'trading_days',
36   'treasury_period_return'
37 ]
38 def run_strategy(strategy_name):
39   mod = None
40     if strategy_name == "buy_and_hold":
41     mod = BuyAndHold()
42     register_calendar("YAHOO", get_calendar("NYSE"), force=True)
43   return run_algorithm(
44     initialize=getattr(mod, 'initialize', None),
45     handle_data=getattr(mod, 'handle_data', None),
46     before_trading_start=getattr(mod, 'before_trading_start', None),
47     analyze=getattr(mod, 'analyze', None),
48     bundle='quandl',
49     environ=environ,
50     # Definimos un capital inicial que puede ser sobreescrito por el test
51     **merge({'capital_base': 1e7}, mod._test_args())     )
```

Código A.1.1: run_strategy

A.2. Estrategia *comprar y mantener*

```
from zipline.api import order, symbol, record
from matplotlib import pyplot as plt
import pandas as pd

class BuyAndHold:

  stocks = ['AAPL', 'MSFT']

  def initialize(self, context):
    context.stocks = self.stocks

  def handle_data(self, context, data):
    if context.portfolio.cash > 0:
      for stock in context.stocks:
        price = data.current(symbol(stock), 'price')
        timeseries = data.history(
                    symbol(stock),
                    'price',
                    bar_count=33,
                    frequency='1d')
                historical_mean = np.mean(timeseries)

                if price < historical_mean:
          order(symbol(stock), 10)

  def _test_args(self):
    return {
      'start': pd.Timestamp('2008', tz='utc'),
      'end': pd.Timestamp('2018', tz='utc'),
      'capital_base': 1e4
    }

  def analyze(self, context, perf):
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    perf.portfolio_value.plot(ax=ax1)
    ax1.set_ylabel('Valor del portafolio en $')

    plt.legend(loc=0)
    plt.show()
```

Código A.2.1: Implementación de Estrategia BunAndHold Simple

```
import pandas as pd
import numpy as np

from zipline.data.data_portal import DataPortal
from zipline.data import bundles
from zipline.utils.calendars import get_calendar

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

from joblib import dump
```

Código A.2.2: Bosque Aleatorio: Paquetes necesarios

```
bundle_data = bundles.load("quandl")

end_date = pd.Timestamp("2009-10-30", tz="utc")

data_por = DataPortal(
    asset_finder=bundle_data.asset_finder,
    trading_calendar=get_calendar("NYSE"),
    first_trading_day=bundle_data.equity_daily_bar_reader.first_trading_day,
    equity_daily_reader=bundle_data.equity_daily_bar_reader
)

MSFT = data_por.asset_finder.lookup_symbol(
    "MSFT",
    as_of_date=None
)

df = data_por.get_history_window(
    assets=[MSFT],
    end_dt=end_date,
    bar_count=5000,
    frequency='1d',
    data_frequency='daily',
    field="close"
)

df = df.dropna()
df.index = pd.DatetimeIndex(df.index)
df['close'] = df[list(df.columns)[0]]
df = df.drop(columns=[list(df.columns)[0]])
```

Código A.2.3: Bosque Aleatorio: Obtención de datos

```
df['1d'] = df.shift(-1)

for d in range(2, 41):
    col = "%dd" %d
    df[col] = df['close'].shift(-1 * d)

df = df.dropna()
```

Código A.2.4: Bosque Aleatorio: Generación de serie temporal

```
X = df.iloc[:, :33]
Y = df.iloc[:, 33:]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4)

regressor = RandomForestRegressor(
    n_estimators=100,
    criterion='mse',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features='auto',
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=True,
    n_jobs=16,
    random_state=None,
    verbose=1,
    warm_start=False)

regressor.fit(X_train, Y_train)

regressor.score(X_test, Y_test) # 0.9901137276141183

Y_predicted = regressor.predict(X_test)

fig, ax = plt.subplots()
fig.subplots_adjust(bottom=0.3)
fig.set_figwidth(16)
fig.set_figheight(8)

ax.plot(Y_test.index, Y_test['40d'], 'ro', markersize=2)
ax.plot(Y_test.index, Y_predicted[:, 7], 'bo',  markersize=2)

ax.legend()

plt.xticks(rotation=70)
plt.tight_layout()
plt.show();

dump(regressor, 'rf_regressor.joblib')
```

Código A.2.5: Bosque Aleatorio: Entrenando el modelo con RandomForestRegressor

```
    def initialize(self, context):
    context.stocks = ['AAPL', 'MSFT']
        context.regressor = load('./strategies/models/rf_regressor.joblib')

  def handle_data(self, context, data):
        if context.portfolio.cash > 0:
            for stock in context.stocks:
                timeseries = data.history(
                    symbol(stock),
                    'price',
                    bar_count=33,
                    frequency='1d')
                np_timeseries = np.array(timeseries.values).reshape(1, -1)
                preds = context.regressor.predict(np_timeseries)
                max_price = np.max(preds)
                historical_mean = np.mean(np_timeseries)

                if max_price > historical_mean:
                    order(symbol(stock), 10)
```

Código A.2.6: Bosques Aleatorios: Implementación en estrategia *comprar y mantener*.

A.3. Estrategia de *scalping*

```
def prepareCSV(csv_pth):
    files = os.listdir(csv_pth)
    start = end = None

    dd = OrderedDict()
    for f in files:
        fp = os.path.join(csv_pth, f)
        n1 = os.path.splitext(fp)[0]
        key = n1.split('/')[1]
        df = pd.read_csv(fp, sep=';', header=None, names=['date','open','high','low','close','volume'])
        df.date = pd.to_datetime(df.date, format='%d/%m/%Y␣%H:%M:%S')
        df.index = pd.DatetimeIndex(df.date)
        df = df.sort_index()
        dd[key] = df.drop(columns=['date'])
        start = df.index.values[0]
        end = df.index.values[10*24*60]


    panel = pd.Panel(dd)

    panel.major_axis = panel.major_axis.tz_localize(pytz.utc)

    return panel, pd.to_datetime(start).tz_localize(pytz.utc), pd.to_datetime(end).tz_localize(pytz.utc)
```

Código A.3.1: Función para utilizar un fichero CSV como fuente de datos.

```
class ScalpBollingerBand:

    stock = 'DAX'
    ma1 = 10
    ma2 = 25
    steps = 20
    stop_loss = 0.001
    stdv = 2

    def initialize(self, context):
        context.stock = self.stock
        context.burndown = 0
        context.position = None
        context.open = 0
        context.limit = 0
        context.kind = None
        context.stop = 0
        context.time = None
        context.value = 0
```

Código A.3.2: ScalpingBollingerBand: Inicializando valores

```
def handle_data(self, context, data):
  # Contamos cada paso y esperamos a llegar al mínimo para comenzar
  context.burndown += 1

  if context.burndown > self.steps:

  # Obtenemos el precio actual (puede que para el minuto actual no esté disponible)
  try:
   current_price = data.current(symbol(context.stock), 'price')
  except:
   return

  # Historial de valores
  hist = data.history(
    symbol(context.stock),
    'close',
    bar_count=20,
    frequency='1m')

  ewm = hist.ewm(span=20, ignore_na=True ).mean()[1]

  if ewm != ewm:
    return

  # Bollinger Bands
  blw = ewm - self.stdv * hist.std()
  bhi = ewm + self.stdv * hist.std()

  if ewm != ewm or bhi != bhi or blw != blw:
    return

  # Medias Móviles
  short_term = data.history(
    symbol(context.stock),
    'price',
    bar_count=self.ma1,
    frequency='1m').mean()

  long_term = data.history(
    symbol(context.stock),
    'price',
    bar_count=self.ma2,
    frequency='1m').mean()

  # Comprobamos indicadores
  if short_term > long_term and current_price > bhi and context.position != 'trade':
    context.position = 'short'
  elif short_term < long_term and current_price < blw and context.position != 'trade':
    context.position = 'long'
```

Código A.3.3: Cálculo de Valores de Estrategia

```
# Comprobamos Bollinger Bands
if current_price <= bhi and context.position == 'short':
  # Cálculamos la cantidad de acciones que nos podemos permitir
  num_shares = (context.portfolio.cash // current_price) // 3
  # Cálculamos el valor de stop loss
  stop = current_price + (current_price * self.stop_loss)
  # Abrimos una operación en CORTO
  order(asset=symbol(context.stock), amount=num_shares * -1)
  # Establecemos los valores contextuales de la operación necesarios para cerrarla
  context.position = 'trade'
  context.kind = 'short'
  context.number_shorts += 1
  context.stop = stop
  context.open = current_price
  context.limit = ewm
  context.time = context.burndown
  context.number_shorts = context.number_shorts + 1
  print('Enter Short: ', current_price, 'limit', context.limit, 'stop', stop)
elif current_price >= blw and context.position == 'long':
  # Cálculamos la cantidad de acciones que nos podemos permitir
  num_shares = (context.portfolio.cash // current_price) // 3
  # Cálculamos el valor de stop loss
  stop = current_price - (current_price * self.stop_loss)
  # Abrimos una operación en LARGO
  order(asset=symbol(context.stock), amount=num_shares)
  # Establecemos los valores contextuales de la operación necesarios para cerrarla
  context.position = 'trade'
  context.kind = 'long'
  context.number_longs += 1
  context.stop = stop
  context.open = current_price
  context.limit = ewm
  context.time = context.burndown
  context.number_longs = context.number_longs + 1
  print('Enter Long: ', current_price, 'limit', context.limit, 'stop', stop)
```

Código A.3.4: Toma de decisiones.

```
if (current_price >= context.limit and context.position == 'trade' and context.kind == 'long') \
or (current_price <= context.limit and context.position == 'trade' and context.kind == 'short') \
or (context.position == 'trade' and (context.burndown - context.time) > 600):
    order(symbol(context.stock), 0)
    if context.kind == 'long':
        context.value += current_price - context.open
    elif context.kind == 'short':
        context.value += context.open - current_price
    context.position = context.kind
    context.limit = 0
    context.kind = None
    context.stop = 0
    context.open = 0
    context.time = 0
    record(RETURNS= context.value )
    print('Exit Trade: ', current_price, context.value, context.portfolio.cash )
if (current_price < context.stop and context.position == 'trade' and context.kind == 'long') \
or (current_price > context.stop and context.position == 'trade' and context.kind == 'short'):
    order(symbol(context.stock), 0)
    context.value -= abs(context.open-context.stop)
    context.position = context.kind
    context.limit = 0
    context.kind = None
    context.stop = 0
    context.open = 0
    context.time = 0
    record(RETURNS= context.value )
    print('Stop: ', current_price, context.value, context.portfolio.cash )
```

Código A.3.5: Determinación de cierre de operación.

```
def analyze(self, context, perf):
  fig = plt.figure()
  ax1 = fig.add_subplot(211)
  ax2 = fig.add_subplot(212)
  perf.plot(y=[
      'PRICE'
      ], ax=ax1)
  ax1.set_ylabel('Bandas')
  perf.plot(y=['RETURNS'],ax=ax2)
  ax2.set_ylabel('Valor del portafolio en $')

  plt.legend(loc=0)
  plt.show()
```

Código A.3.6: Análisis de los resultados.

```
def initialize(self, context):
  ...
  context.last = 0
  context.historical_changes = []

def handle_data(self, context, data):
  ...
  change = ( current_price - context.last ) / context.last * 100
  context.historical_changes.append( change );
  context.last = current_price
```

Código A.3.7: Calculando variación diaria.

```
if len(context.historical_changes) > 1440 and context.position == 'trade':

  # Obtenemos los cambios porcentuales del último día
  historical_changes = np.array(context.historical_changes[-1440:])
  # Calculamos la media y la desviación estándar de los datos obtenidos
  mean = np.mean(historical_changes)
  std_dev = np.std(historical_changes)
  # Obtenemos el cuantil con un alfa de 0.95
  normal_returns = np.random.normal(mean, std_dev, 1440)
  value_at_risk = np.percentile( normal_returns, 1-0.95 )
  # Calculamos el ES y lo utilizamos como VaR
  expected_shortfall = np.mean(historical_changes[historical_changes<=value_at_risk])

  # Calculamos la variación porcentual entre el precio actual y el de apertura
  change = ( current_price - context.open ) / context.open * 100

  if  ( context.kind == 'long' and change < expected_shortfall) \
    or ( context.kind == 'short' and change > expected_shortfall ) :
    # Cerramos la operación
    order(symbol(context.stock), 0)
    # Calculamos el resultado de la operación
    returns = abs( context.open-current_price )
    if (current_price < context.open and context.kind == 'long') \
    or (current_price > context.open and context.kind == 'short'):
      context.value = context.value - returns
    else:
      context.value = context.value + returns
    # Reiniciamos valores contextuales
    context.position = None
    context.limit = 0
    context.kind = None
    context.stop = 0
    context.open = 0
    context.time = 0
    # Imprimimos la información correspondiente
    print('Stop Loss: {} price {} @ VaR {} for {} at {}'.format(
      context.value,
      current_price,
      value_at_risk,
      context.stock,
      get_datetime()
      )
    )
```

Código A.3.8: Cálculo de VaR histórico.

```
1  import pandas as pd
2
3  from sklearn.svm import SVR
4  from sklearn.multioutput import MultiOutputRegressor
5  from sklearn.model_selection import TimeSeriesSplit
6  from sklearn.model_selection import GridSearchCV
7
8  import joblib
9
10 df = pd.read_csv(DAX-201312.csv', sep=";")
11 df.date = pd.to_datetime(df.date, format='%d/%m/%Y %H:%M:%S')
12 df.index = pd.DatetimeIndex(df.date)
13 df = df.drop(columns=['date'])
14
15 df = df[['open', 'close']]
16 df = df.head(1440)
17
18 df['lag_0'] = ( (df.close - df.open) / df.open ) * 100
19 df = df.dropna()
20 df = df.drop(columns=['open', 'close'])
21
22 for i in range(1, 20):
23   col = "lag_{}".format(i)
24   df[col] = df.lag_0.shift(-1 * i).values
25
26 dataset = df.as_matrix()
27 X = dataset[:, 0:15]
28 y = dataset[:, 15:20]
29
30 estimator = MultiOutputRegressor(SVR(gamma='scale'))
31 estimator.get_params().keys()
32 parameters = {
33   'estimator__kernel': ('linear', 'poly', 'rbf'),
34   'estimator__C': [1, 10]
35 }
36 cv = TimeSeriesSplit(n_splits=5)
37 clf = GridSearchCV(estimator, parameters, cv=cv, return_train_score=True)
38 clf.fit(X, y)
39
40 joblib.dump(clf.best_estimator_, 'estimator.joblib')
```

Código A.3.9: Entrenando SVR.

```
1  def initialize(self, context):
2    ...
3    context.regressor = load('./strategies/models/estimator.joblib')
4
5  def handle_data(self, context, data):
6    ...
7    # Obtenemos los cambios porcentuales del último día
8    historical_changes = np.array(context.historical_changes[-1440:])
9    # Predecimos valores futuros tomando los últimos 15 minutos del histórico
10   forecast_changes =
11     context.regressor.predict(
12       historical_changes[-15:].reshape(1, -1)
13     )
14   # Añadimos los nuevos valores al conjunto de datos
15   historical_changes =
16     np.concatenate(
17       (
18         historical_changes,
19         forecast_changes[0]
20       ), axis=0
21   )
22   ...
```

Código A.3.10: Calculando VaR con SVR.

Bibliografía

Artzner, P. y col. (1997). "Thinking Coherently". En: *Risk* 10.11, págs. 68-71.

Bartàk, Roman (1999). "Constraint Programming: In Pursuit of the Holy Grail". En: *Proceedingsof WDS99*.

BBVA (2015). *Â¿Qué son los mercados de futuros?* URL: https://www.bbva.com/es/que-son-los-mercados-de-futuros/.

Bergmeir, Christoph (2012). "On the use of cross-validation for time series predictor evaluation". En: *Information Sciences* 191, págs. 192-213.

Bhattacharyya, Saptashwa (2018). *Support Vector Machine: Kernel Trick; Mercer's Theorem*. URL: https://towardsdatascience.com/understanding-support-vector-machine-part-2-kernel-trick-mercers-theorem-e1e6848c6c4d.

Bollinger, John (2002). *Bollinger on Bollinger Bands*. 978-0-07-137368-5. McGraw Hill.

Breiman, Leo (2001). "Random Forests". En: *Machine Learning*.

Cardoso, Sergio Manuel Jiménez y Manuel M. García-Ayuso Covarsí (2002). *Análisis Financiero*. Ediciones Pirámide.

Castro, José Luis Alba (2013). "Máquinas de Vectores Soporte (SVM)". En: *Universidad de Vigo*.

CBOE (2019). *Cboe Volatility Index*. Inf. téc. Cboe Exchange.

CFTC y SEC (2010). *Findings Regarding The Market Events of May 6, 2010*. Inf. téc. U.S. Commodity Futures Trading Commission, the U.S. Securities y Exchange Commission.

Comité de Supervisión Bancaria de Basilea (2004). *Convergencia Internacional de Medidas y Normas de Capital*. Banco de Pagos Internacionales.

Cooper, Gregory F. y Edward Herskovits (1992). "A Bayesian method for the induction of probabilistic networks from data". En: *Machine Learning* 9.4, págs. 309-347. URL: https://doi.org/10.1007/BF00994110.

Cristianini, N. y J. Shawe-Taylor (2000). *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

Duhigg, Chaerles (2009). "Stock Traders Find Speed Pays, in Milliseconds". En: *New York Times*.

Durlauf, Steven y col., eds. (2007). *Handbook of Econometrics*. Elsevier.

Elisfm (2009a). *SVM 8 polinomial*. URL: https://commons.wikimedia.org/wiki/File:Svm_8_polinomial.JPG.

Elisfm (2009b). *SVM 9 Polinomial*. URL: https://commons.wikimedia.org/wiki/File:Svm_8_polinomial.JPG.

Fama, Eugene (1970). "Efficient Capital Markets: A Review of Theory and Empirical Work". En: *Journal of Finance* 2.25, págs. 383-417.

Freund, Yoav y Robert E. Schapire (1996). "Experiments with a new boosting algorithm". En: *ICML'96: Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, págs. 148-156.

García, Fernando (2018). *Técnica Contable y Financiera*. Wolters Kluwer.

Grant, Jeremy (2010). "High-frequency trading: Up against a bandsaw". En: *Financial Times*.

Halls-Moore, Michael L. (2017). *Successful Algorithmic Trading*.

Hens, Profesor Dr. Thorsten y MSc. BA Anna Meier (2016). "Finanzas Conductuales: La Psicología de la Inversión". En: *Behavioral Finance Solutions GmbH*.

Kenton, Will (2020). *Standard and Poor's 500 Index*. URL: https://www.investopedia.com/terms/s/sp500.asp.

Khorram, Alireza, Cheah Wooi Ping Ph.D. y Liew Tze Hui (2011). *Causal Knowledge-Driven Approach For Stock Analysis*. Inf. téc. International Conference on Business y Economics.

Kimura, Hugh (2020). *Paul Rotter - The Legend, The Myth, The Man*. URL: https://www.tradingheroes.com/paul-rotter-the-legend-the-myth-the-man/.

López, Roberto Gómez (2005). *Mercados Financieros: Futuros y Opciones*.

Mackay, Charles (1841). *Memoirs of Extraordinary Popular Delusions and the Madness of Crowds*. Library of Economics y Liberty.

Malkiel, Burton G. (1996). *A Random Walk Down Wall Street*. W. W. Norton.

Mercer, James (1909). "Functions of positive and negative type and their connection with the theory of integral equations". En: *Philosophical Transactions of the Royal Society* A.209, págs. 441-458.

Pearl, Judea (1988). "Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference". En: *Representation and Reasoning Series*.

Platt, John C. (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Inf. téc. Microsoft Reserach.

Rodrigo, Joaquín Amat (2017). "Máquinas de Vector Soporte". En:

Rodriguez, Daniel (2018). *GridSearchCV*. URL: https://www.analyticslane.com/2018/07/02/gridsearchcv/.

Sayad, Saed (2010). *An Introduction to Data Science*. URL: https://www.saedsayad.com/support_vector_machine_reg.htm.

Schapire, Robert E. y col. (1998). "Boosting the Margin: A new explanation for the effectiveness of voting methods". En: *The Annals of Statistics*.

Segal, Mark R (2004). "Machine Learning Benchmarks and Random Forest Regression". En: *CSF: Center for Bioinformatics and Molecular Biostatistics*.

Weinberg, Zack (2012). *Svm separating hyperplanes*. URL: https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes_(SVG).svg.

Yamai, Yasuhiro y Toshinao Yoshiba (2002). "On the Validity of Value-at-Risk: Comparative Analyses with Expected Shortfall". En: *Monetary and economic studies* 20, págs. 57-85.

Zoltan, Czako (2018). *SVM and Kernel SVM*. URL: https://towardsdatascience.com/svm-and-kernel-svm-fed02bef1200.

Zuo, Yi y Eisuke Kita (2012). *Up/Down Analysis of Stock Index by Using Bayesian Network*. Inf. téc. Canadian Center of Science y Education.