Universidad de Granada

Proyecto fin de Doble Grado en Ingeniería Informática y Matemáticas

Una estrategia para bolsa basada en algoritmos evolutivos y su implementación en una plataforma de trading

---

Miguel Ángel Torres López

supervisado por
Prof. Jose Manuel Zurita López

9 de agosto de 2019

Agradecimientos

En este proyecto se intenta predecir el mutable
valor de las acciones de bolsa.
A pesar de esto, doy mi agradecimiento
a la única acción que no necesita ninguna predicción:
el apoyo de los que siempre están ahí,
bien por afecto, bien por profesionalidad.

Resumen

El mercado de valores ha sido durante muchos años el lugar donde se invierte más dinero. Poseer información privilegiada sobre los activos bursátiles es cada vez más importante, pues dan a empresas e inversores la capacidad de tomar mejores decisiones económicas. Con el objetivo de conseguir estos conocimientos, se ha diseñado e implementado un algoritmo evolutivo basado en árboles de decisión. Estos árboles, construidos a partir de la herramienta genética, se componen de indicadores bursátiles clásicos que conforman las reglas del modelo de inversión.

Esta técnica, a pesar de tener un componente aleatorio, puede reporta unos resultados exitosos, especialmente en el caso de las inversiones a largo plazo. No obstante, en inversiones a corto plazo o tendencias demasiado inestables los beneficios son más modestos e, incluso, se producen pérdidas.

A partir de este proyecto, se propone el uso de múltiples herramientos y modelos de la bolsa de valores para adquirir una mejora de los conocimientos del mercado que, si bien está repercutido por factores ajenos a los indicadores, puede ser perfilado mediante estos.

Palabras clave: predicción de bolsa, minería de datos, algoritmo evolutivo, árbol de decisión, indicadores bursátiles

Abstract

The stock market has been for many years the place where most money is invested on. To own insider information about the stock asset is becoming increasingly important cause of the better decisions that investors and business are able to take. In order to achive that knowledge, an evolutionary algorithm based on decision trees has been designed and developed.

We find researches where the focus is to know the exact price that the asset will reach in some time. Other ones try to classificate the trend where the stock is involved or even the one where it will be. At this end-of-grade project, we work with a different point of view. We are not interested anymore on the stock value. Instead of that, a buy and sell signal model is going to be built. That means that the knowledge we aqcuire is not the stock value but the time where we should buy or sell.

To obtain the decision tree who will send us the signal we need some trading tools.

As Python is the language selected, *backtrader*, a backtesting framework, is going to be used. This framework let us to dislinkage the backtesting process from the main evolutionary algorithm developed. Besides, *backtrader* gives us some plotting amenities, to get a better analysis, and a strong parallel processing, that will be necessary due to time limitations.

After that, we need to get the stock market data we are going to work with. We check multiple ways to take them to *backtrader* but, the best solution seems to be *pandas_datareader*.

Finally, as we need to calculate numerous times the indicator values at different days, we work with an analysis package for that porpouse, *TA-Lib*. The package allows to work out many classic indicators from a period at once, instead of calculate them day per day.

Once we have introduced the tools, we are prepared to build the algorithm. Because the aim is to develop a tree decision model we must first explain the tags or classes.

The model must send two different signals: buy and sell. However, a three tag tree is proposed due to some undeterminated moments at the stock market. So, the three tags are: Buy, if we should send a buy signal, sell, if we should send a sell signal, and stop, if the model can not take any previous tag.

Next, we need to design how conditions on tree nodes are built. The tree is formed with rules that look like

$$indicator(params) <= pivot$$

- *Indicator* is an indicator taken from the list (MACD, ATR, ROC, EMA, SMA, Momentum, HILL, RSI, OBV, AD, TRANGE, Bollinger Band High, Bollinger Band low).
- *Params* is a variable list of parameters that depends on the indicator.
- *Pivot* is a real number that splits the day on two branches, the positive condition one and the negative one.

As expected, we just manage binary trees.

At this point, we define the evolutionary algorithm. Our first approach was building the first population with random procedures. Further experiment with heuristic methods gave better results for the first generation. Despite of the time wasted on it, we choose to warm up the population by building them with randomness for indicators but entropy functions for pivots.

After that, each tree is evaluated with the fitness function, that is, the simulation of a training period where *backtrader* works out the return from invest with each tree based strategy.

The money returned by each tree is used to set the crossover probability, that is, the probability of being selected to generate a new tree. That probability is proportional to the score achieved on the simulation.

In order to get the new population, the crossover is managed as next:

First select a random node from each tree. Note that selected nodes could be roots, the top nodes, but could not be a leaf as this will produce a nonsense decision tree.

Then, the subtrees under the choosen nodes are swapped. That way, we have two new trees with some different conditions.

Note that params and pivots in the condition are not changed with that crossover. So thats the point to introduce the mutation. After each new tree generation, the mutation factor is mandatory. Four mutation methods are devoloped.

- Indicator mutation. Selects a new indicator from the above list. That triggers to change parameters and pivot.
- Pivot mutation. Following a normal distribution, introduce a random noise on pivot.
- Parameters mutation. Change the parameters by taking a new one based on a discrete uniform distribution, if the parameter is

discrete, or on a normal distribution, if the parameter is continuous.

- Leaf mutation. That is a special mutation that change the tag of a leaf. As we cant take leaves for crossover, those will remain down the same indicator at every population. This relation was set up in the first generation, so it should not be correct.

Once the new population is finally generated, the process starts again by evaluating the trees.

Due to time limit, some improvements are made to reduce the execution time. The first one is the *backtrader* parallel execution. After that, a deep analysis revealed that most of the time was wasted on taking data from *pandas dataframe*. So, to reduce that a cache to save data closer was developed.

Once the algorithm is ready for executions, a window of parameters is proposed to check the best ones.

The algorithm seems to learn, reasonably well, the pattern to invest with benefits in the trainning period. Furthermore, buy signals are sent close-fitting to minimums.

However, while investing on test periods, results are not that good. At small periods, minimums are well found, but the model sends sell signals even when commissions are bigger than benefits.

At long investing periods, signals are not sent frecuently. That produce *Buy&Hold* strategies at uptrends and no buys at downtrends.

Despite of the results, the algorithm builds models that learn well the patterns on the training period. To improve results we suggest to use the evolutionary algorithm with a complementary algorithm that could help the first by testing the trend.

Through this project, stock indicator rules are prove to be given useful knowledge to understand the market situation. Nevertheless, a prediction algorithm with them seems not worth at unsettled trend markets.

Keywords: stock prediction, data mining, evolutionary algorithm, decision tree, stock indicator

Índice

Índice de figuras

1 (Introducción) Predicción en el mercado de valores

Una de las formas más sencillas, y a la vez difíciles, de enriquecerse es la compra de bienes y posterior venta a un precio mayor. La ganacia se produce si se consigue que la diferencia entre el precio de compra y el precio de venta sea mayor que los costes producidos a razón de esa transacción. En el mundo de los negocios la compra y la venta de bienes aparecen en casi todos los sectores económicos. Por ejemplo:

- Distribución de mercancias. Hay empresas que se dedican a comprar mercancias en un punto y distribuirlas por otras zonas a un precio mayor. El coste de este ejercicio reside en el transporte. La empresa debe asegurar que los costes de transportes no superan a la diferencia de precio entre compra y venta.

- Especulación de terrenos o edificios. Un poco más cara que la anterior, la compra, y posterior venta, de edificios puede producir beneficios. Las empresas que se dedican a esto tienen en cuenta las posibles construcciones que se van a hacer por la zona que, posiblemente, elevarán el valor del inmueble y harán el negocio posible. Mantener terreno conlleva un pago de impuestos, que tendrá que ser cubierto con el beneficio de la venta. Pero, en este caso, los inmuebles se pueden alquilar para sacar un beneficio continuo.

Del mismo modo, el mercado de valores brinda una posibilidad para extraer beneficio con la compra y venta. Se compran acciones a un precio determinado y se intentan vender a otro más alto. Si se tiene información adicional que nos dé certeza sobre la subida del precio de la acción entonces el negocio es seguro. Si, por el contrario, no se dispone de información, el valor de las acciones podría bajar y provocar una pérdida de dinero.

Por tanto, tener conocimiento sobre el estado del mercado sitúa a empresas y particulares en posiciones ventajosas. Es por esto que, siguiendo distintas estrategias, las personas que invierten en bolsa intentan extraer información sobre la situación actual y futura del mercado.

A lo largo de este proyecto se va a intentar extraer información a partir del histórico de valores del mercado, es decir, vamos a utilizar el estado del mercado en un período pasado para predecir el estado futuro.

1.1 Precedentes en la predicción de bolsa

El primer mercado de valores se sitúa en el siglo XVII. Según afirma Beattie (2017) en la revista digital Investopedia, la primera empresa en ofrecer acciones se situaría en Bélgica, promovida por la incipiente economía generada en las colonias de Asia.

A lo largo de tantos años de vida, los valores bursátiles se han intentado predecir de muchas formas. A medida que la sociedad avanza y se van creando nuevas herramientas, los modelos de predicción van evolucionando y se hacen más complejos.

En la actualidad podemos distinguir varias agrupaciones de análisis de bolsa:

- **Análisis chartista**. Debe su nombre a la palabra inglesa *chart*. Las estrategias de esta agrupación se valen de reglas gráficas aplicadas sobre índices o sobre el propio valor de la acción. Esas reglas gráficas advierten de series temporales, como las ondas de Elliot (Elliot, 1994), o de tendencias y rupturas (Gartley, 1935).

  Esta estrategia se lleva utilizando mucho tiempo por su simplicidad y su fácil comprensión y evaluación. Su éxito se debe probablemente a este hecho, y es que con una sencilla herramienta gráfica se pueden deducir, a gran escala, algunas subidas o bajadas de los valroes de bolsa.

- **Análisis de índices o indicadores**. Esta modalidad de análisis se basa en el uso de estadísticos. En el campo de la estadística, los estadísticos son valores que representan una determinada característica de la variable a la que refiere. En el caso de los indicadores, informan del estado de uno o varios valores distintos de la bolsa. Una de las publicaciones con más renombre es la teoría de Dow[1], cuyo autor, Charles H. Dow, fue el fundador del *Wall Street Journal*, diario en el que publicaba sus principios para invertir en bolsa.

  Según la variable que resume el indicador podemos hablar de varios tipos. El primer tipo son los indicadores que representan la situación general de la bolsa de un país, por ejemplo el IBEX35 (35 empresas con más capital de España). Otro grupo de indicadores son los representan la situación de las empresas que se dedican a un sector particular, como el SX7E (bancos de los paises europeos). Por último hay índicadores que aportan información estadística sobre un solo valor, por ejemplo la EMA (Media Móvil Exponencial).

[1]Hamilton, W. P. (1922). *The Stock Market Barometer; a Study of Its Forecast Value Based on Charles H. Dow's Theory of the Price Movement*, Harper & Bros., Nueva York

Este proyecto se centra en el análisis de indicadores de un solo valor. En concreto, como se verá a lo largo del trabajo, se propone un modelo basado en árboles de decisión construidos a partir de un algoritmo genético. Los árboles de decisión son, a grandes rasgos, estructuras que permiten crear una jerarquía de condiciones que tienen como conclusión de las mismas una clasificación. En el caso que nos compete, la clasificación corresponderá a decidir si es un buen momento para comprar, para vender o para ninguno de los dos. En el apartado 5 se entrará con más profundidad en este modelo.

1.2 Tipos de información extraíble a partir de árboles

Podemos encontrar en la bibliografía varios modelos basados en árboles que, según la información que persiguen conseguir, se pueden dividir en varios grupos. En este punto debemos hacer notar que existen otras formas y herramientas de extracción información o predección, pero por la naturaleza de nuestra propuesta nos centraremos en las estrategias basadas en árboles. Debido al caracter motivacional de la sección, no se entra en profundidad en estos conceptos. No obstante, la definición de árbol puede verse posteriormente en el apartado 4.

1.2.1 Información total

El más ambicioso de los objetivos de un análisis de bolsa es el de encontrar una forma de predecir el valor exacto de un valor bursátil en un instante de tiempo futuro. Encontramos entonces una función $f : \Omega \to \mathbb{R}$ donde Ω es un conjunto de indicadores del valor a predecir en un instante pasado o presente y la imagen, contenida en $\mathbb{R}$ es el valor predicho para un instante en el futuro.

El instante predicho cambia según el autor del estudio (pueden ser horas, días o semanas), normalmente, cuanto más alejado está del instante en el que se evalúa f del instante que se pretende predecir, mayor es el error.

La función f se puede representar como un árbol en el que en los nodos se sitúan operaciones (usualmente unarias o binarias) y las hojas contienen parámetros de la función f o valores constantes. La función se evalúa desde las hojas hasta el nodo raíz.

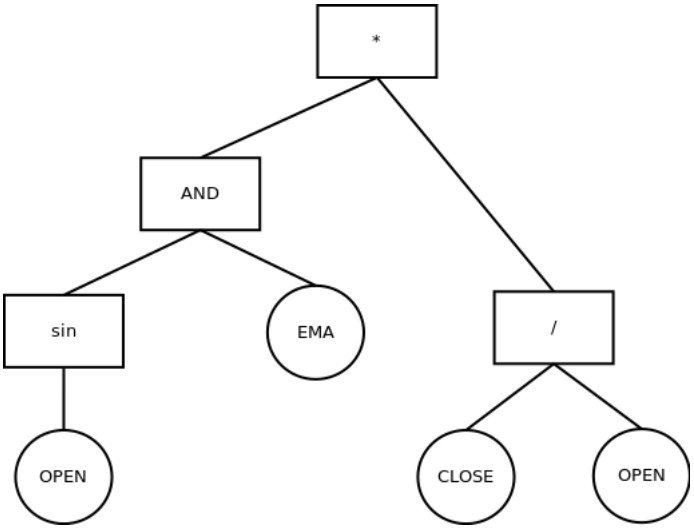

Figura 1: Ejemplo de árbol de programación genética.
Fuente: elaboración propia.

En la figura 1 puede verse un árbol de este tipo. De este se puede extraer la función
$f(\text{OPEN, CLOSE, EMA}) = (sin(\text{OPEN}) \text{ AND EMA}) * (\text{CLOSE/OPEN})$
que, dados unos valores para la apertura, el cierre y la media móvil exponencial, nos devuelve el valor en el instante futuro.

Para ajustar los árboles se usa un algoritmo genético con función *fitness* el error medio cuadrático entre el valor predicho y el error real (Sheta, 2013). Se comentará con profundidad el funcionamiento de los algoritmos genéticos más adelante. También existe otra variante basada en el *Conditional Sharpe Ratio* aplicada en Esfahanipour (2011) y Mousavi (2014).

Por último, una función *fitness* algo diferente es puntuar cada estrategia a partir del beneficio simulado obtenido en un periodo de tiempo (Potvin, 2004).

Si se consiguiese hallar una función f con un error nulo, obtendriamos una información perfecta con la que podríamos conseguir el máximo beneficio en bolsa. No obstante, este tipo de modelos es demasiado ambicioso. Computacionalmente es pesado y la función resultante no es, en casi todos los casos, interpretable.

1.2.2 Información de tendencias

En lugar de buscar una predicción exacta del valor en el futuro, podemos estar interesados en obtener información sobre la tendencia de un valor, es

decir, si está en un periodo ascendente o descendente.

Este método puede ser visto como una relajación del método desarrollado en el apartado anterior. Aunque el paso de una imagen continua ($\mathbb{R}$) a una imagen discreta (sube, baja, se queda igual) nos obliga a realizar otro tipo de árboles. En este caso se suele proponer un árbol de decisión con tres etiquetas. Pero no vamos a entrar en detalle aquí en esta herramienta, ya que será explicada en profundidad en el apartado 4.

Estos modelos de información son bastante más sencillos de interpretar. Pero, por contra, son bastante más imprecisos. El modelo nos dice la tendencia que tendrá el valor. Sin embargo, según los objetivos de inversión, las comisiones u otras situaciones económicas, esta información podría no ser suficiente para sacar beneficios.

Para profundizar en la extracción de este tipo de información se sugiere ver Nair (2010) o Huang (2008).

1.2.3 Información de señales de compra y venta

A medio camino entre los dos tipos de información anteriores tenemos los modelos de señales, mediante los cuales, los más importante no es conocer el valor de la acción, si sube o si baja, sino buscar un modelo que nos diga cuando comprar y cuando vender. A estas advertencias de los modelos de compra y venta se le llaman señales de compra y venta. Cabe decir que se puede hacer una extensión del modelo para que no tenga una decisión binaria, por ejemplo un modelo de cuatro etiquetas: compra discreta, venta discreta, compra masiva y venta masiva.

De este tipo de extracción de información hay menos ejemplos, que suelen estar ejecutados a partir de varios árboles de salida booleana, cada uno de ellos asociado a una señal. En Myszkowski (2010) podemos ver un claro ejemplo con dos árboles, uno que indica la compra y otro que indica la venta.

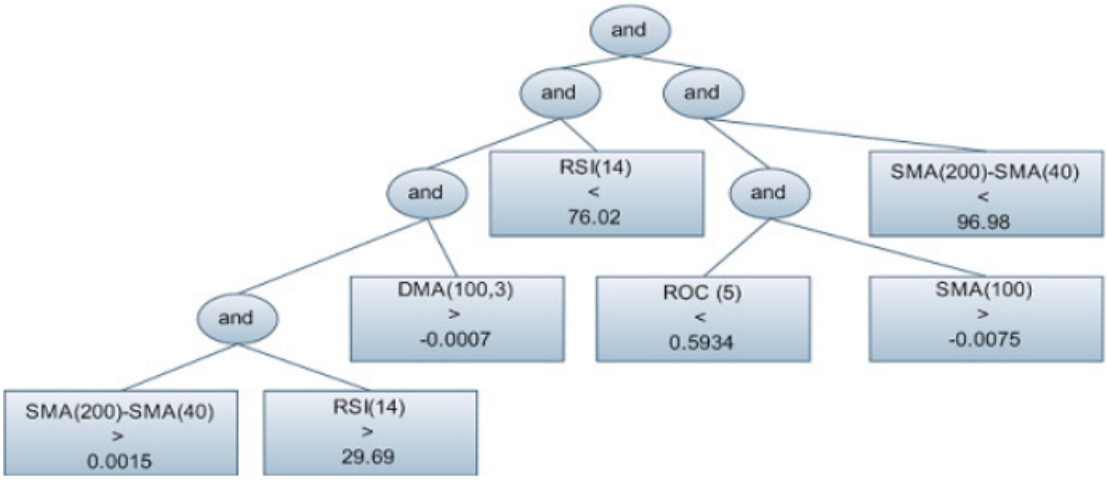

Figura 2: Ejemplo de árbol de señales de compra.
Fuente: Myszkowski (2010)

En la figura 2 se muestra un árbol de señal de compra. Nótese que en los nodos hoja siempre hay una condición booleana que puede contener un solo indicador o, en caso especial, dos.

1.3 Objetivo del proyecto

La importancia de obtener información exclusiva sobre el estado de la bolsa es evidente, ya que esto te permite invertir en bolsa con mayor posibilidad de sacar beneficios que si se realizara por mera intuición.

De este modo, siguiendo el desarrollo realizado en los antecedentes aportados, se propone conseguir los siguientes objetivos principales:

- Diseñar un algoritmo genético basado en árboles que permita extraer información de señales de compra y venta.
- Debido al coste computacional de los algoritmos genéticos, se busca implementar dicho algoritmo de la forma más eficiente posible, en especial la función de evaluación.
- Ejecutar en distintas tendencias bursátiles y comparar los resultados con los aportados en la bibliografía.

A pesar de que el histórico de datos es clave para predecir los valores futuros, el mercado de valores a menudo se ve influenciado por otros factores

no predecibles (noticias, política, guerras o catástrofes ambientales, por ejemplo). Por tanto, aunque el modelo predicho sea muy bueno, siempre habrá un punto de incertidumbre que nunca estaremos en condiciones de analizar.

En consecuencia, partimos de la idea de que la inversión perfecta no se puede conseguir, al menos de forma juiciosa y sistemática. No obstante, añadimos un punto adicional a los objetivos que, a priori, es complicado alcanzar:

- Mejorar los resultados obtenidos en los distintos mercados de valores de la bibliografía.

2 Fundamentos de Bolsa

2.1 Mercado de valores

El mercado de valores o bolsa es un tipo de mercado en el que se compran y venden productos bursátiles. Para este trabajo, prestaremos principal atención a las acciones, el producto bursátil más simple de un mercado de valores.

Una acción equivale a una pequeña participación en la empresa a la que pertenece la acción. De esta forma, inversores privados tienen la posibilidad de comprar porciones de empresas. Por ejemplo, si una empresa está dividida en 200 acciones, tras comprar 50 de estas, seríamos propietarios de una cuarta parte de la empresa. Ser partícipes del accionariado de una empresa puede reportarnos beneficios de distinta índole según las normas de la misma. En ocasiones algunas empresas deciden repartir una determinada cantidad de dinero entre sus accionistas a partir de las ganancias obtenidas en un periodo (dividendo). Otras veces, ser poseedor de una gran parte de la empresa otorgará derechos dentro de la misma como, por ejemplo, participar en las decisiones importantes que se tomen.

Por último, el mercado de valores español abre de 9:00 a 17:30 de lunes a viernes. Pero cabe destacar que hay una aleatoriedad de 30 segundos, es decir, la hora exacta de la apertura y del cierre varía de forma no definida hasta en 30 segundos. Además, después del cierre hay un periodo de subasta de 5 minutos, durante el cual el valor de las acciones está oculto para evitar manipulaciones de los precios en los últimos minutos. No obstante, no es el momento de explicar a fondo el funcionamiento interno de un mercado de valores. Nos será suficiente con los puntos básicos anteriormente explicados.

2.2 El valor de las acciones

El valor de una acción no es fijo, si no que cambia según la oferta y la demanda de las acciones de la empresa. Esto convierte a las acciones en un producto muy conveniente para especular.

El mecanismo de compra y venta es el que define el valor de una acción. Para comprar o vender, un inversor debe enviar a bolsa una orden de compra o venta, respectivamente. Tras enviar la orden, esta queda registrada y esta será ejecutada tan pronto como sea posible. Aclaremos este proceso con un ejemplo sencillo:

Supongamos que existe una empresa que tiene en el mercado cuatro acciones. Cada una de ellas pertenece a un propietario distinto, llamados A, B,

C y D.

El inversor A tiene una orden de venta de su acción a 4 euros y C tiene una orden de compra de una acción a 1 euro. En esta situación ninguno puede vender ni comprar, y el precio queda situado en 4 euros por acción. Ahora bien, el accionista B decide vender su acción por 3 euros. En el momento en el que realiza una orden de venta, el precio de las acciones baja a 3 euros. Esto no quiere decir que el inversor A venda su acción a este precio. Para bajar de 4 a 3 euros el precio de su acción debería cancelar la orden y realizar una nueva. En esta situación no se puede ejecutar ninguna orden en el mercado, como se puede ver en la figura 3.

Órdenes de compra	Precio (euros)	Órdenes de venta
	5	
	4	1
	3	1
	2	
1	1	

Figura 3: Situación de las órdenes.
Fuente: elaboración propia.

De este modo, tras conocer que el precio de la acción ha bajado de 4 a 3 euros, el inversor D decide comprar y sitúa una orden de compra a 3 euros de una acción. Entonces, la orden de venta de C y la orden de compra de D pueden ejecutarse y, por ende, anularse. Ahora el inversor D pasa a tener 2 acciones y el mercado se queda con tan solo las dos órdenes de A y B. A causa de este hecho, el precio de la acción vuelva a subir a 4 euros, la orden de venta más baja.

En el ejemplo anterior se han simplificado las órdenes de compra y venta. En realidad, las ordenes que existen son un poco más complejas:

- **Orden de mercado.** Con este tipo de orden se compran todas las acciones que se especifiquen al mejor precio posible. En caso de que la orden sea de compra, se adquieren acciones a los precios más bajos, en caso de que sea de venta, a los precios más altos. Refiriendo al caso anterior, si B hubiera emitido una orden de mercado de una acción, habría comprado la acción del inversor A, que en este caso tenía un valor de 4 euros.

- **Orden limitada.** En este caso, se compran todas las acciones que se emitan en la orden siempre y cuando su precio esté situado en un límite establecido. Por ejemplo, si D hubiera emitido una orden limitada a 3 euros de dos acciones, hubiera comprado por 3 euros la acción de C pero no hubiera adquirido la de A, que estaba situada a 4 euros. La orden limitada permanece vigente hasta que la suma total de las acciones sea comprada.

- **Orden on Stop.** En esta modalidad, la orden no se hace efectiva hasta que el precio de las acciones llega al precio estipulado en la orden. Una vez llegado ese momento, la orden se convierte en una orden de mercado, es decir, se compran o se venden todas las acciones que se especifiquen en la misma al mejor precio posible.

- **Orden Stop-Limit.** Este tipo de orden no se hace efectiva hasta que el precio de las acciones no llegue al precio de la orden. En ese momento, la orden se convierte en una orden limitada. Es muy frecuente que las plataformas ofrezcan este tipo de orden, pues funciona como seguro para no perder todo el capital cuando las acciones caen de precio rápidamente. Si se sospecha que las acciones pueden bajar su valor, se coloca una orden stop-limit de venta a un precio que se considere bajo. De este modo, si el valor de las acciones decae, estas se venden de forma automática.

2.3 Corredores de bolsa o brokers

Para agilizar las transacciones en bolsa, no se permite que particulares envíen órdenes al mercado. Esta tarea se delega en los corredores de bolsa o *brokers*. La entidad de *broker* se encarga de captar inversores, lanzar órdenes a bolsa y presentar a compradores con vendedores. Para ser broker se debe aprobar un examen y demostrar que se tienen los conocimientos necesarios. A cambio de realizar este trabajo se lleva una comisión que todo inversor debe abonar.

Las comisiones pueden ser de dos tipos:

- **Comisiones fijas.** Son un valor fijo que cada broker cobra. Pueden ser por transacción, según cantidad de acciones y su precio, y por tiempo de servicio, según la cantidad de días que el cliente dispone para realizar transacciones. Las comisiones fijas son beneficiosas para inversores con un gran capital.

- **Comisiones variables.** Son precios variables que dependen del país de inversión, la cantidad invertida e incluso el tiempo que dura una inversión. Las comisiones variables afectan en la misma medida a inversores grandes y pequeños. Por este motivo, un inversor con poco capital pagará menos comisiones con este tipo de cobro.

La mayoría de los corredores de bolsa tienen comisiones mixtas. Algunos cobran distintas comisiones según la bolsa en la que se quiera operar.

Sirva todo lo anteriormente expuesto como introducción a los mecanismos y herramientas que intervienen en el mercado de valores. Solo debemos añadir que en los supuestos que trabajaremos más adelante, utilizaremos una comisiones variable del 1 % de la cantidad invertida. Esto implicará que por cada 100 euros invertidos, se perderá 1 en concepto de comisiones de *broker*.

2.4 Herramientas de simulación

El propósito central de este proyecto es encontrar un modelo con estructuctura de árbol que sea capaz de invertir en bolsa de forma beneficiosa. Puesto que la ejecución del modelo es necesaria para medir su bondad, hay que trabajar en un entorno de pruebas en el que poder hacer ensayos. Este entorno no debe ser la bolsa real, porque simular grandes periodos costaría el mismo tiempo que la longitud del periodo y, además, se estaría haciendo con dinero real, con todas las dificultades que esto puede tener.

En su lugar, las plataformas de *backtesting* ofrecen las simulaciones que se necesitan. El *backtesting* es el proceso que permite simular un periodo de bolsa e invertir con una estrategia particular de forma sistemática. A continuación, se mencionan algunas plataformas y un pequeño análisis de cada una para discernir cuál es mejor para esta tarea.

2.4.1 MetaTrader5

MetaTrader5 es una software, disponible en versión escritorio y en versión web, con el que se pueden comprar y vender acciones en tiempo real pero de forma simulada, es decir, sin participar con dinero real. Es una plataforma ampliamente usada y, por tanto, es fácil encontrar ejemplos iniciales o ayuda técnica. Por comentar una de las facilidades, en la página web de la plataforma[2] se nos ofrece una demo gratuita con acceso a la mayoría de los recursos, aunque hay algunas herramientas de pago.

[2]https://www.metatrader5.com [Última consulta 10 de Julio de 2019]

La programación de las estrategias se organiza en varios archivos con unas especificaciones concretas. Además, existe una tienda empotrada en la plataforma para comprar y vender estrategias, índices o bibliotecas con otros usuarios. Cada producto está puntuado por los usuarios que lo han usado. A priori, esto puede ser bastante útil para comparar la estrategia propuesta en el proyecto con otras estrategias clásicas o con las estrategias mejor puntuadas en la plataforma.

A pesar de todas estas ventajas, no usaramos *MetaTrader5* por ser de carácter privativo. Asimismo, el lenguaje de programación de las estrategias es *MQL5*, un lenguaje propio de la plataforma, que nos impide usar con facilidad bibliotecas externas.

2.4.2 Backtrader

Backtrader es un *framework* de libre licencia desarrollado en el lenguaje *Python*. La página web de la plataforma[3] contiene una amplia documentación en la que se explican todos los componentes de este *framework*. También se encuentran, en el mismo lugar, el método de instalación y un tutorial de iniciación para simulación de bolsa.

La herramienta contiene indicadores ya programados, aunque se permite hacer más de forma manual. Instalando una librería adicional se pueden generar gráficas de ganancias en un periodo de simulación. Este desacoplamiento de la parte gráfica no supone un gran problema porque en la instalación viene integrada la opción para instalar todo, tal y como se verá en el siguiente apartado.

2.4.2.1 Instalación

Pasamos pues, a ilustrar la instalación de *backtrader*. La instalación puede hacerse en cualquier sistema operativo que soporte *Python* pero, en lo que sigue, se supondrá que el sistema es *Ubuntu*.

En primer lugar, es preciso tener la versión 2.7 de *Python*, como se indica en la documentación de *backtrader*[4] que se ubica en su página web. Para comprobar la versión, se puede usar el comando *python -Version*. En caso de no tener instalado *Python* o no tener la versión indicada, se puede instalar usando *sudo apt-get install python2.7*.

[3]https://www.backtrader.com [Última consulta 10 de Julio de 2019]

[4]https://www.backtrader.com/docu/index.html [Última consulta 10 de Julio de 2019]

El *framework backtrader* está disponible desde el código fuente en *Github*. No obstante, es más cómodo usar la versión para el instalador de paquetes de *Python, pip*. Si no se tiene instalado *pip*, la instalación se realiza con el comando *sudo apt-get install python-pip*.

Una vez llegados a este punto, se puede optar por una versión con posibilidad de generar gráficas o una versión más ligera sin esta posibilidad. Para facilitar el análisis de resultados, se recomienda instalar la versión con gráficas. Su instalación se ejecuta con el comando *pip install backtrader[plotting]*, como podemos ver en la figura 4. Para instalar una versión más ligera, se usa el mismo comando quitando la directiva *[plotting]*. En este caso se instala una versión similar pero sin la posibilidad de hacer gráficas.

Figura 4: Resultado de instalar backtrader con plotting.
Fuente: elaboración propia.

La instalación de backtrader conlleva la instalación de otros paquetes adicionales con los que tiene dependencias. Entre ellos hay que destacar el paquete *numpy*, que más tarde nos será útil como herramienta matemática para generar distribuciones estadísticas.

2.4.2.2 Preparación del entorno

Una de las estructuras de datos más comunes en *backtrader* son las líneas. Una línea no es más que un conjunto de datos ordenados por orden cronológico, del más antiguo al más moderno. Por ejemplo, al hablar de un valor de mercado tenemos 5 líneas básicas: valor de apertura, valor de salida, valor máximo, valor mínimo y volumen de compra/venta.

En términos de informática, para acceder a los elementos de cada línea, hay que tener en cuenta que no se puede usar la indexación de un vector como en un lenguaje de programación común. En su lugar, el índice 0 representa el instante actual y, a partir de este, se cuentan con los números enteros las posiciones de los demás. Hay que aclarar que para acceder a instantes anteriores usaremos índices negativos, como por ejemplo:

```
self.sma = SimpleMovingAverage(...)
valor_actual            = self.sma[0]
valor_instante_anterior = self.sma[-1]
```

Por tanto, según el día que se esté simulando, los datos de otro día concreto se acceden con un índice o con otro. De esta forma, suponiendo que se está simulando el lunes y queremos acceder al dato del miércoles el índice es el 2. Si por el contrario se simula el jueves, para acceder al miércoles se usa el índice -1. Nótese que, por motivos académicos, no se deben usar los índices mayores que 0, ya que esto supondría tener información sobre el futuro y la predicción de la bolsa sería un fraude.

Antes de ver cómo crear una estrategia, es preciso ver como se puede simular una prueba introduciendo unos datos y un presupuesto inicial. Para ello, en un archivo con terminación *.py*, para poder ejecutar con *Python*, se insertará el siguiente código:

```
from __future__ import (absolute_import, division, print_function, unicode_literals)

import datetime
import os.path
import sys
import backtrader as bt # Importar todas las herramientas de backtrader

if __name__ == '__main__':
        cerebro = bt.Cerebro()

# Crear un paquete de datos
        data = bt.feeds.YahooFinanceCSVData(
        dataname='YAHOO', # Ruta absoluta donde se encuentran los datos descargados de YAHOO! Finance
# Fecha inicial de los datos
        fromdate=datetime.datetime(2000, 1, 1),
# Fecha final de los datos
        todate=datetime.datetime(2000, 12, 31),
        reverse=False)

# Activar los datos en el cerebro
        cerebro.adddata(data)
# Establecer dinero inicial
        cerebro.broker.setcash(100000.0)
```

```
print('Starting Portfolio Value: %.2f' %cerebro.broker.getvalue())
cerebro.run()   # EJECUTAR BACKTESTING (AHORA MISMO, SIN NINGUNA ESTRATEGIA)
print('Final Portfolio Value: %.2f' %cerebro.broker.getvalue())
```

Nótese que se trata de una archivo *Python* en el que solo hemos tenido que importar el paquete *backtrader*. Por tanto, en el caso de necesitar otros paquetes para realizar nuestra estrategia, solo será necesario incorporarlos de la forma habitual en este lenguaje de programación.

El código es ejecutable y funcional, pero hay un problema, no se le ha especificado a la clase *cerebro* cuál va a ser la estrategia a seguir. Por este motivo, si se haciera la simulación, el presupuesto inicial y final no variarían.

Para ejecutar el *backtesting* simplemente se ejecuta el *script* de *Python* con el comando *python rutadelscript.py* en la terminal. Seguramente dará un fallo porque la ruta donde debería estar el fichero con los datos a simular está vacía. Esto era de esperar pues no hemos descargado ningún archivo de datos. El desarrollo de este tema se abordará en la sección 2.5.

2.4.2.3 La primera estrategia

Antes de programar la primera estrategia de *trading*, vamos a ilustrar con una fútil cómo estructurar una clase para que *backtrader* pueda trabajar con ella como estrategia. La estructura básica es una clase con un método *__init__* y otro método llamado *next*.

El primero puede usarse para instanciar y agrupar los datos que requiera nuestra estrategia. El segundo método, por su parte, es llamado en cada instante simulado por *backtrader*, considerando que un instante es cada una de las marcas temporales que hay registradas en nuestros datos. Por aclarar esto un poco, si aportamos al *cerebro* unos datos tomados de forma diaria, los instantes son cada uno de los días.

Cabe destacar que, en cada llamada al segundo método, solo son accesibles los datos con marcas temporales menores o iguales al instante correspondiente a la llamada. Existe, no obstante, una forma de saltarse esta regla y conseguir información de instantes posteriores. Pero como se especificó en el apartado anterior, no es de interés académico.

Veamos un ejemplo básico en el que, en cada instante, se imprime en pantalla el precio actual del valor:

```
# Crear una estrategia
class TestStrategy(bt.Strategy):

    def log(self, txt, dt=None):
```

```
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
    # Guarda una referencia de la linea de valores de cierre
        self.dataclose = self.datas[0].close

    def next(self):
    # Muestra por pantalla el valor de cierre
        self.log('Close, %.2f' % self.dataclose[0])
```

Al ejecutar el *cerebro* con esta estrategia, veremos el valor de cierre de cada instante y, si los datos lo facilitan, la marca temporal asociada a cada instante.

Para activar la estrategia es necesario indicar al *cerebro* la clase creada con la siguiente línea, que puede ser situada justo después de indicar el presupuesto inicial:

```
# Registrar estrategia
cerebro.addstrategy(TestStrategy)
```

De este modo, la estrategia está completa, pero en ningún instante se realizan compras o ventas. Para nuestro posterior estudio es imprescindible introducir estas acciones. Sin embargo, por motivos de espacio y tiempo, no entraremos en profundidad en todos los aspectos de la compraventa.

Veamos el código de la primera táctica de inversión que nos muestra el funcionamiento de las órdenes bursátiles:

```
class DoubleDownStrategy(bt.Strategy):

    def log(self, txt, dt=None):
        dt = dt or self.datas[0].datetime.date(0)
        print('%s, %s' % (dt.isoformat(), txt))

    def __init__(self):
        self.dataclose = self.datas[0].close

        # Para mantener las ordenes no ejecutadas
        self.order = None

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            return

        if order.status in [order.Completed]:
            if order.isbuy():
                self.log('BUY EXECUTED, %.2f' % order.executed.price)
            elif order.issell():
                self.log('SELL EXECUTED, %.2f' % order.executed.price)

            self.bar_executed = len(self)

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        self.order = None

    def next(self):
        self.log('Close, %.2f' % self.dataclose[0])
        # Si hay una compraventa pendiente no puedo hacer otra
        if self.order:
            return

        # Si no tengo nada adquirido
        if not self.position:
            if self.dataclose[0] < self.dataclose[-1]:
                if self.dataclose[-1] < self.dataclose[-2]:
                    self.log('BUY CREATE, %.2f' % self.dataclose[0])
```

```
                    self.order = self.buy()

        else:
            # Ya hemos adquirido algo
            if len(self) >= (self.bar_executed + 5):
                self.log('SELL CREATE, %.2f' % self.dataclose[0])
                self.order = self.sell()
```

Aquí se introducen varios conceptos nuevos del *framework*. En primer lugar, las acciones *self.buy()* y *self.sell()*, ejecutadas dentro del método *next*, indican que queremos lanzar una orden de compra o de venta, respectivamente. Cuando no se especifica ningún parámetro, *backtrader* compra o vende al precio de cierre del instante actual una sola acción. Esto no quiere decir que la orden se ejecute en ese instante, sino que, a partir de ese momento y si el precio del producto lo permite, se realizará lo antes posible. Nótese que una vez lanzada una orden hay que esperar a que se complete o se cancele antes de lanzar otra.

Según lo anterior, esta situación hace necesario incluir el método *notify_order*. Como su propio nombre sugiere, es llamado cuando una orden cambia de estado o, en otras palabras, es el momento a partir del cual se puede lanzar una nueva orden. En este ejemplo, cuando una orden es completada, mostramos por pantalla si es de compra o venta y el precio de la misma. En ciertas ocasiones, las órdenes pueden terminar sin compra o venta. En ese caso, puede haber ocurrido una de dos: ser canceladas, si la estrategia canceló la orden con el comando *self.cancel()*, o rechazas, si bien el dinero disponible era insuficiente para ejecutar la orden.

Por último, comentaremos la parte lógica de la estrategia presentada en el código anterior. En cada instante, el método *next* evalúa alguno de los siguientes casos:

- **Existe una orden lanzada y no terminada.** En este caso debemos esperar. Téngase en cuenta que *backtrader* permite cancelar una orden, aunque no entraremos en este punto con profundidad.
- **No hay una orden lanzada y tampoco tengo nada comprado.** La cantidad de acciones que se poseen puede comprobarse con *self.position*, que se interpreta como un booleano negativo si no se tiene nada en posesión. En esta situación solo cabe esperar una orden de compra y, para nuestro ejemplo, la lanzaremos si los últimos dos instantes han bajado su precio de cierre de forma consecutiva.
- **No hay una orden lanzada pero tengo algo comprado.** En este caso, se supone que solo se puede vender, aunque si se tuviese dinero no hay ningún problema en comprar más acciones.

Para ilustrar una nueva herramienta, se propone lanzar una orden de venta cinco instantes después de haber comprado. Para ello podemos comprobar la longitud de *self* con la función *len* de *Python*. Esta acción, que puede resultar un tanto extraña, es una forma propia del lenguaje *Python* de comprobar las dimensiones de un objeto, en este caso de la compra.

2.4.3 Otras plataformas

- **Tradestation.** Permite programar estrategias, pero no hacer *backtesting*. Además no es de acceso gratuito y obliga a invertir con dinero real.
- **Cloud9trader.** Tiene una demo que permite programar estrategias y hacer *backtesting*. Está bien para probar estrategias sencillas basadas en indicadores ampliamente conocidos. No obstante, la programación hay que hacerla en la ventana del navegador, con un lenguaje propio y sin posibilidad de usar paquetes externos.
- **Plus500.** Es una plataforma de *trading* con escasas posibilidades de automatización. Tan solo permite algunas sencillas órdenes condicionales preprogramadas.
- **PyAlgoTrade.** Es un proyecto desarrollado en *Python* disponible en *Github*. Tiene posibilidad de *backtesting*, de utilizar paquetes externos y, en el caso de estar haciendo *trading* con Bitcoins, de comprar y vender estos de forma real. Los datos del mercado hay que conseguirlos de forma externa en formato CSV.

2.5 Conseguir datos históricos para realizar *backtesting*

Como vamos a usar *backtrader*, una herramienta de simulación de bolsa en local, vamos a necesitar extraer el histórico de la bolsa. En esta sección veremos cómo conseguir los valores históricos en bolsa de diferentes empresas de forma automática. A parte de conseguir los datos, veremos la forma de incorporarlos a backtrader para su posterior uso.

2.5.1 Quandl

Quandl es una empresa que reúne miles de paquetes de datos financieros de todo el mundo. Para acceder a esta información es necesario registrarse

en su página web. Una vez tengamos acceso, es posible descargar muchos de los archivos en formato CSV. Aunque quizás la opción más cómoda para este proyecto sea utilizar la *API* que ofrece. De esta forma se realiza el acceso a los datos desde *Python*, sin necesidad de hacer una descarga previa a la ejecución del *script*.

Para usar la *API* seguiremos los pasos de instalación que se pueden encontrar en la documentación[5] de *Quandl*. Así, instalamos el paquete *quandl* con el gestor de paquetes *pip*. No obstante, para usar el paquete, debemos indicar la *api_key* que nos dan al inscribirnos en su página, única para cada usuario.

```
# Crear un paquete de datos con QUANDL
        data = bt.feeds.Quandl(
                dataset='WFE',
                fromdate = datetime.datetime(2016,1,1),
                todate = datetime.datetime(2017,1,1),
                dataname='INDEXES_BMESPANISHEXCHANGESMADRID',
                buffered=True,
                apikey='')
```

No obstante, como puede verse en la figura 5, *backtrader* tiene algún error interno al usar esta *API*. Error que, como en un foro de la comunidad se apunta[6], se debe a una mala lectura de los vectores de datos recibidos. Ante la imposibilidad de resolver este problema, y sin información sobre si backtrader lo arreglará, se descarta usar esta forma de adquisición de datos.

```
miguel@miguelpc:/media/miguel/10BC63AFBC638DD2/Universidad/TFG/Backtrader Strate
gies$ python basic.py
Starting Portfolio Value: 100000.00
Traceback (most recent call last):
  File "basic.py", line 35, in <module>
    cerebro.run()   # EJECUTAR BACKTESTING (AHORA MISMO, SIN NINGUNA ESTRATEGIA)
  File "/home/miguel/.local/lib/python2.7/site-packages/backtrader/cerebro.py",
line 1127, in run
    runstrat = self.runstrategies(iterstrat)
  File "/home/miguel/.local/lib/python2.7/site-packages/backtrader/cerebro.py",
line 1212, in runstrategies
    data.preload()
  File "/home/miguel/.local/lib/python2.7/site-packages/backtrader/feed.py", lin
e 685, in preload
    while self.load():
  File "/home/miguel/.local/lib/python2.7/site-packages/backtrader/feed.py", lin
e 476, in load
    _loadret = self._load()
  File "/home/miguel/.local/lib/python2.7/site-packages/backtrader/feed.py", lin
e 707, in _load
    return self._loadline(linetokens)
  File "/home/miguel/.local/lib/python2.7/site-packages/backtrader/feeds/quandl.
py", line 107, in _loadline
    o = float(linetokens[next(i)])
IndexError: list index out of range
```

Figura 5: Ejecución del script recolectando datos de Quandl.
Fuente: elaboración propia.

[5]https://docs.quandl.com/docs/python-installation [Última consulta 10 de Julio de 2019]

[6]https://community.backtrader.com/topic/797/quandl-data-feed-futures-data [Última consulta 10 de Julio de 2019]

2.5.2 YAHOO! Finance

Una de las opciones más completas para conseguir los datos es la página web de *YAHOO! Finance* [7]. Basta con buscar el índice o la empresa sobre la que se desean obtener los datos, indicar las fechas y la frecuencia de muestreo y pinchar en el botón de descarga. A continuación, los datos se descargarán en formato CSV con las columnas de fecha, apertura, clausura, máximo, mínimo y volumen. Este formato es ampliamente utilizado por la comunidad científica para almacenar grandes cantidades de datos.

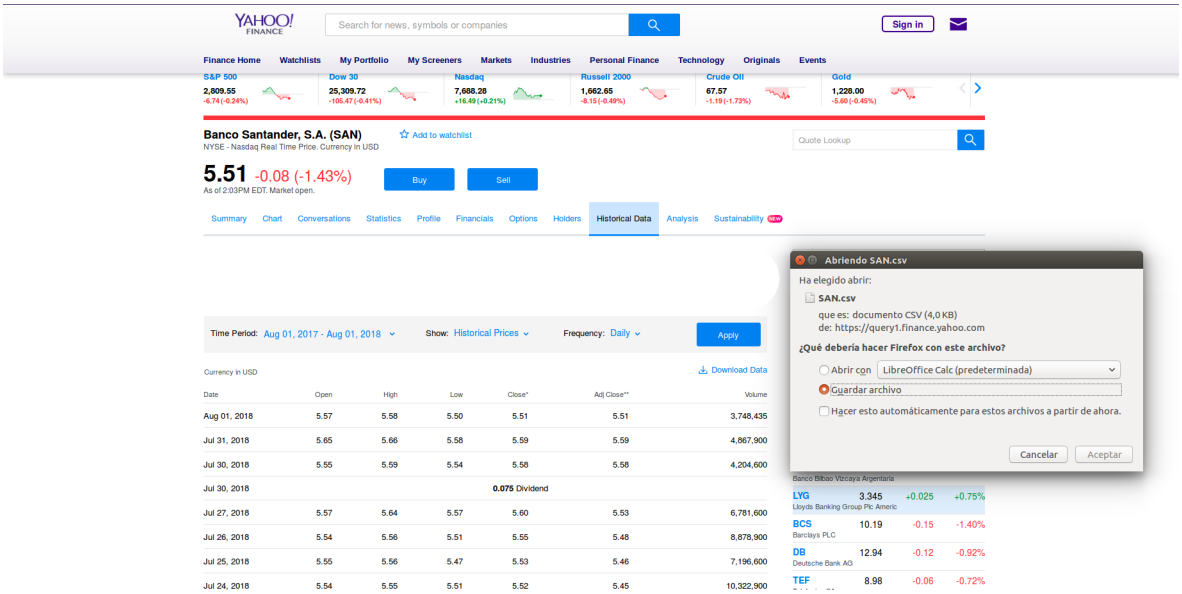

Figura 6: Descarga de los datos históricos de *Banco Santander*. Fuente: elaboración propia.

Por ejemplo, imaginemos que buscamos los valores históricos de *Banco Santander* desde el 1 de agosto de 2017 hasta la misma fecha del año siguiente. Para ello indicamos la empresa, buscamos el apartado de *Historical Data* e indicamos las fechas. El resultado se muestra en la figura 6.

Finalmente, para cargar estos datos en *backtrader* basta con indicar la ruta donde se encuentra el archivo en el ordenador y el rango de fechas que se quiere ejecutar con el siguiente código:

```
# Crear un paquete de datos con YAHOO FINANCE
data = bt.feeds.YahooFinanceCSVData(
```

[7] https://finance.yahoo.com

```
        dataname='Data/SAN.csv',
        fromdate=datetime.datetime(2017, 8, 1),
        todate=datetime.datetime(2018, 8, 1),
        reverse=False)

# Activar los datos en el cerebro
cerebro.adddata(data)
```

Aunque en principio carece de interés, el rango de fechas que se indique debe estar contenido en el archivo, pero no tiene por qué ser el mismo. Este hecho permite descargar y almacenar grandes cantidades de datos, previo a su uso, y almacenarlas para un futuro.

Esta opción para adquirir datos es sencilla al tener interfaz gráfica, pero no se recomienda si se requiere ejecutar muchas veces con distintas empresas. Por tanto, queda descartado su uso por el momento.

2.5.3 Pandas Datareader

De forma alternativa a la descarga de CSV anterior, podemos utilizar el paquete *Pandas-datareader*, que proporciona una función que automatiza la descarga de datos de *YAHOO! Finance*.

Aunque estuvo un tiempo en desuso y en su documentación está marcado así[8], ahora vuelve a estar operativo. La *API* de *YAHOO! Finance* cambió y, durante un tiempo, *Pandas-datareader* no la actualizó. No obstante, el tratamiento de los datos por parte de *Pandas-datareader* ya se ha adaptado, aunque no se vea reflejado aún en su documentación.

Por nuestra parte, porcomodidad, usaremos a partir de ahora la siguiente fórmula para cargar los datos en *Backtrader*:

```
start_date = "2013-08-01"
end_date   = "2014-02-26"
simudatos = pdr.get_data_yahoo("SAN", start=start_date, end=end_date)
df_cerebro = bt.feeds.PandasData(dataname = simudatos)
```

Como puede observarse en el código, basta llamar a la función con el símbolo de la empresa y las fechas de inicio y fin para descargar los datos diarios. Nótese que la descarga se realiza desde el propio script y es necesaria realizarla en cada ejecución, pero se hace de forma automática. Aunque suponga una pequeña carga para el tiempo, lo usaremos por la facilidad de cambiar de fechas y empresa.

[8] https://pandas-datareader.readthedocs.io/en/stable/whatsnew.html#v0-6-0-january-24-2018 [Última consulta 20 de Julio de 2019]

3 Algoritmos genéticos

La evolución es un proceso de optimización cuyo objetivo es la mejora continua de una habilidad o característica. Este proceso se observa en la mayoría de los seres vivos que, en un intento de adaptarse al medio y fortalecerse, cambian progresivamente.

Los algoritmos genéticos tienen como inspiración este comportamiento. Para hallar la solución óptima de un problema, se genera una población de soluciones, no necesariamente buenas. A continuación, se simula una descendencia de estas, es decir, nuevas soluciones del problema que se parecen a las anteriores y, mediante selección natural, se favorece la evolución de las soluciones hacia una mejora. En el caso de los algoritmos genéticos, esta selección natural requiere de una forma de medir la bondad de una solución. Si se reitera este proceso la población de soluciones irá potenciando la característica que consigue mejor puntuación en la manera de medir.

A continuación se ofrece una aproximación a los algortimos genéticos. Un desarrollo más profundo puede encontrarse en otras obras[9].

3.1 Definición

Un algoritmo genético es un conjunto de procedimientos ordenados que, efectuados de forma iterativa y aplicados sobre un conjunto de posibles soluciones, hacen mejorar a la agrupación hacia unas soluciones mejores. Se identifican cinco elementos básicos en un algoritmo genético:

- **Población.** Es un conjunto de soluciones del problema que queremos resolver, cuya representación computacional definirá al resto de los elementos.
- **Función *fitness* o función de evaluación.** Es la función cuyo dominio es la población del problema y que tiene por imagen algún subconjunto de $\mathbb{R}$. Esta función, hablando en términos matemáticos, es una norma que evalúa cómo de buena es una solución.
- **Cruce.** Es un proceso mediante el cual varias soluciones producen una o varias soluciones combinadas nuevas.

[9]Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons.

- **Mutación** Se define como el proceso que actúa sobre un individuo de la población y le produce una transformación signficativa. Este procedimiento no es imprescindible, pero ayuda a que las soluciones sigan variando tras muchas generaciones.
- **Selección.** Es el criterio de eliminación que permite eliminar individuos de la población.

En los siguientes epígrafes se va a profundizar en cada elemento de un algoritmo genético, mostrando especial interés en ver los valores más usuales de estos.

3.1.1 Población

En la naturaleza, cada organismo tiene unas características concretas que influyen en su habilidad para sobrevivir y reproducirse. Estas características, en última instancia, son definidas por los cromosomas de dicho individuo. En un algoritmo genético, se usa tradicionalemente la notación de cromosoma para referirse a la codificación de un individuo.

La representación clásica de un cromosoma es un vector de *bits* de dimensión fija.

En el caso de que el espacio de búsqueda sea discreto, cada individuo de la población se puede representar con un vector de longitud n donde 2^n es el número de valores posibles del espacio.

En el caso continuo, tomaremos $\mathbb{R}^n$ como espacio de búsqueda y será necesario acotarlo, es decir, el dominio adopta la forma

$$[x_{min_1}, x_{max_1}] \times \cdots \times [x_{min_i}, x_{max_i}] \times \cdots \times [x_{min_n}, x_{max_n}]$$

Entonces, mediante una representación binaria, por ejemplo la de coma flotante de longitud l, podemos transformar cada $x = (x_1, \ldots, x_i, \ldots, x_n) \in \mathbb{R}^n$ del dominio en $b = (b_1, \ldots, b_i, \ldots, b_n)$ donde $\forall 0 < i \leq n \ b_i \in \{0, 1\}^l$. Es decir, se traslada el espacio continuo a una aproximación en coma flotante. Nótese que la precisión numérica de un computador siempre es finita y, por tanto, ninguna representación puede cubrir el espacio de búsqueda continuo al completo. No obstante, Esta aproximación nos permite trabajar con el caso continuo igual que con el caso discreto con buenos resultados.

Por supuesto, lo anterior es solo una propuesta clásica que se aplica a un caso genérico. Según el problema a tratar, un individuo puede ser representado de otras formas más convenientes.

Una vez que tenemos definida la estructura de un individuo, la población será un conjunto de estos. La cantidad de individuos de la población se puede variar a lo largo de la ejecución del algoritmo, aunque, cuanto mayor sea el número de individuos, mayor será el espacio de búsqueda inspeccionado y mejores serán los resultados. Eso si, cuanto más exaustiva sea la búsqueda, mayor será el tiempo de ejecución.

Otro factor importante a tratar es la calidad de la población en la primera generación, ya que esta determinará, en gran medida, la calidad de las generaciones posteriores. En el caso de que no se tenga información previa sobre el problema se puede optar por tomar una población inicial aleatoria. No obstante, contruir una población inicial cercana a la solución acelera el proceso de convergencia.

3.1.2 Función *fitness*

En un modelo evolutivo, el individuo con mejores características tiene mayores posibilidades de sobrevivir. Para valorar cuán bueno es un individuo de la población, se necesita una función matemática que cuantifique la bondad de este. Definimos la función *fitness* o función de evaluación como

$$f : \Gamma \to \mathbb{R}$$

donde Γ es el espacio en el que se definen los individuos de la población.

Esta función debe evaluar cada individuo de cada generación. Luego, al tratarse de un algoritmo iterativo, conviene que la función no sea computacionalmente pesada. A menudo se usan aproximaciones de la función de evaluación real con el fin de ahorrar tiempo.

3.1.3 Selección

La selección es el operador básico de la evolución. Su objetivo es dar importancia a los individuos fuertes y aumentar su presencia. Se pueden aplicar operadores de selección en dos momentos a lo largo del algoritmo:

- En la selección de la nueva generación, con el objetivo de filtrar los individuos más fuertes (los mejor valorados hablando en términos de la función *fitness*).
- En el cruce de los individuos, con el fin de producir una nueva generación. Es razonable que los individuos con mejores características tengan más posibilidades de tener descendientes en la próxima generación.

Existen multitud de formas de aplicar la selección, las más comunes son las siguientes:

Selección aleatoria

Cada individuo de la población tiene la misma probabilidad de salir elegido, $\frac{1}{s}$, donde s es el número de individuos. Siguiendo una distribución uniforme en $[0, 1]$ se pueden extraer los individuos.

Selección proporcional

La probabilidad de que un individuo sea seleccionado viene dado por la función $\varphi : \Gamma \to [0, 1]$, definida como

$$\varphi(x_i) = \frac{f(x_i)}{\sum_{j=1}^{s} f(x_j)}$$

Esto es, cada individuo tiene una probabilidad de ser seleccionado proporcional a la valoración de la función *fitness*.

Al igual que en la selección anterior, una distribución uniforme se puede usar para extraer individuos.

Selección por torneo

Se extrae una muestra, con o sin devolución, de la población de tamaño k con $k < s$. De entre estos, se selecciona el individuo con mayor puntuación en la función *fitness*.

El proceso se repite hasta que se hayan extraido todos los individuos requeridos.

Selección basada en posición

En lugar de usar directamente la puntuación de la función *fitness*, se usa la posición respecto al resto de individuos de la población.

Por tanto, se selecciona el elemento x_i donde $i = Entera(Z)$ y $Z \sim U(0, U(0, s))$, siendo U la distribución uniforme.

Selección con elitismo

En ciertos casos conviene mantener a los mejores individuos de una población, lo que permitirá asegurar que la nueva población tendrá algún individuo, al menos, tan bueno como el mejor de la generación anterior.

Seleccionar varios individuos para que permanezcan en la población suele ser una buena práctica.

3.1.4 Cruce

La reproducción es el proceso mediante el cual los individuos de una población dan lugar a la siguiente población. Así pues, un cruce es una operación que, dados uno o varios individuos padres, genera uno o varios individuos hijos que serán parte de la próxima generación.

La forma técnica de hacer un cruce depende directamente de la representación de los individuos. Por tanto, dado que fue el caso descrito anteriormente, vamos a presentar algunas formas de hacer cruces cuando tenemos representación binaria.

Cruce de un punto

Dados dos padres a cruzar, se toma un valor i donde $i = Entera(Z)$ y $Z \sim U(1, l-1)$ con l la longitud del vector de *bits* que representa a un individuo.

En consecuencia se crean dos nuevos individuos. El primero tendrá los i primeros *bits* del primer padre y los $l - i$ últimos del segundo progenitor. El segundo individuo, por su parte, se genera con los bits que no se han tomado en el primero.

Cruce de dos puntos

Este cruce es similar al anterior, con la diferencia de que la parte que se toma del primer padre para el primer hijo viene dada por una subcadena, de los *bits* del padre, que comienza y acaba en dos índices generados con una distribución uniforme, tal y como venimos haciendo.

Nótese que aleatorizar el final de la subcadena es equivalente a aleatorizar la dimensión de la misma. En ocasiones se puede optar por una subcadena de tamaño fijo.

Cruce uniforme

Es un procedimiento similar a los anteriores, solo que el reparto se hace *bit* a *bit*. Se inicializa una máscara de tamaño l con los *bits* a 0. Por cada elemento de la máscara, se toma un valor de $X \sim U(0, 1)$; si $x > p$, entonces el valor de la máscara se actualiza a 1. El primer hijo tendrá los bits del primer padre que tengan 0 en la máscara y el resto del segundo padre. Por su parte, el segundo hijo se construye de forma contraria.

3.1.5 Mutación

El objetivo de las mutaciones es la introducción de nuevo material genético en la población, más allá de las combinaciones entre los individuos. Una mutación demasiado frecuente o demasiado agresiva producirá que se pierda el factor evolutivo, no obstante, en aplicado en determinadas dosis puede llegar a ser efectivo si se desea explorar más en el espacio de búsqueda.

Del mismo modo que en la sección de cruce, tenemos distintas variantes de esta operación basadas en la forma de escoger los bits a mutar.

Una vez más, las distintas variantes están basadas en la representación binaria. Otras representaciones de los individuos necesitan de otras mutaciones adaptadas al caso correspondiente.

Mutación uniforme

En este caso, dado un individuo, se seleccionan, a partir de una distribución uniforme, los *bits* que se van a mutar. La mutación consiste, pues, en alterar cada posición con el valor contrario al original (0 si era 1 y 1 si era 0).

Mutación *inorder*

Es un caso similar al anterior, pero los *bits* mutables son solo un subconjunto de los totales. Dicho subconjunto inmutable puede ser producto de una aleatorización o, simplemente, ser seleccionado manualmente si se quiere proteger una zona de la representación de este operador.

Mutación Gaussiana

Esta es una mutación basada en el método de ruido Gaussiano. Si se ha usado la codificación binaria de los individuos, es necesario volver al valor decimal que este representa, por lo que solo es válida en espacios de búsqueda continuos.

Una vez se tiene el valor decimal, se usa el método de ruido Gaussiano sobre el valor, esto es, $x_j = x_j + N(0, \sigma_j)$, con N la distribución normal y $\sigma_j = x_j * 0{,}1$

Finalmente, para concluir se devuelve la variable a la codificación binaria.

Con este último método se pueden proteger los cambios drásticos en los valores, ya que la distribución normal controla que no se cambien los *bits* más significativos en la codificación binaria.

Entiéndase que en un marco práctico, los valores son decimales pero se

guardan en la máquina con representación binaria. Luego las transformaciones entre binaria y decimal se hacen de forma sistemática. Además, al generar un valor con la distribución normal, éste ya está en representación binaria.

3.2 Ejemplo de algoritmo genético con *Pyvolution*

Pyvolution es un paquete de *Python* que facilita el uso de algoritmos genéticos. Aunque la última versión fue lanzada en 2012, es un paquete completo, con una sintaxis simple y sencillo de utilizar. En apenas 30 lineas y sin necesidad de dar muchas especificaciones, se pueden realizar algoritmos completos. Una ejecución normal consta de varios parámetros, a través de los cuales se pueden controlar las características de las generaciones, los individuos por generación, probabilidad y severidad de las mutaciones, cantidad de individuos elitistas e, incluso, el tiempo máximo de ejecución.

Para ilustrar brevemente su uso, se adjunta uno de los ejemplos que vienen en la página de *Github* del paquete.[10]

```
import math
from pyvolution.EvolutionManager import *
from pyvolution.GeneLibrary import *

"""
Queremos calcular una solucion del siguiente sistema:
a + b + c - 17 = 0
a^2 + b^2 - 5 = 0
"""

def fitnessFunction(chromosome):
        """
        Dado un "cromosoma", esta es la funcion que calcula su puntuacion
        La puntuacion es una float mayor que 0.
        """

        #Accedemos a los cromosomas o valores a ajustar
        a = chromosome["a"]
        b = chromosome["b"]
        c = chromosome["c"]
        d = chromosome["d"]

        #Calculamos los valores que nos gustaria que fueran 0
        val1 = math.fabs(a + b + c - 17)
        val2 = math.fabs(math.pow(a, 2) + math.pow(b, 2) - 5)

        #La funcion distancia agrupa los valores para una mejor puntuacion
        dist = math.sqrt(math.pow(val1, 2) + math.pow(val2, 2)

        if dist != 0:
        return 1 / dist # Cuanto menor sea la distancia mayor sera la puntuacion
        else:
        return None     #Devolver None indica que los cromosomas han sido ajustados


#Configuramos el algoritmo genetico
em = EvolutionManager(
        fitnessFunction,
        individualsPerGeneration=100,
        mutationRate=0.2,       #Probabilidad de mutacion
        maxGenerations=1000,
        stopAfterTime=10,   #Para simulacion tras 10 segundos
        elitism=2,          #Mantener los 2 mejores de cada generacion
)
```

[10]https://github.com/littley/pyvolution [Última consulta 10 de Julio de 2019]

```
#Creamos una funcion de mutacion inversamente proporcional a la bondad del ajuste
mutator = FloatInverseFit("mut", maxVal=0.01, startVal=1)

#Indicamos que los puntos iniciales se toman siguiendo una distribucion normal
#de media 0 y desviacion 100. Ademas marcamos la mutacion como la definida antes.
atype = FloatGeneType("a", generatorAverage=0, generatorSTDEV=100, mutatorGene="mut")
btype = FloatGeneType("b", generatorAverage=0, generatorSTDEV=100, mutatorGene="mut")
ctype = FloatGeneType("c", generatorAverage=0, generatorSTDEV=100, mutatorGene="mut")

#Registramos los parametos y la mutacion
em.addGeneType(mutator)
em.addGeneType(atype)
em.addGeneType(btype)
em.addGeneType(ctype)

#Ejecutamos
result = em.run()
```

A pesar de la sencillez, no es práctico usar este paquete de *Python* para nuestro propósito. Los árboles que van a componer nuestra población tienen muchas variables que deben ser introducidas en la herramienta genética. Además, algunas de las variables son continuas, como los parámetros de algunos indicadores, y otras variables son discretas, como el indicador de cada nodo.

Así mismo, *backtrader* tiene posibilidad de paralelización en la ejecución. Esto permite que la función *fitness*, el factor que más incapacita computacionalmente, se pueda ejecutar mucho más rápido.

No obstante, el problema que descarta definitivamente este paquete es la imposibilidad de hacer cruces entre árboles. Solo está preparado para trabajar con representaciones numéricas.

En lugar de usar *Pyvolution*, realizaremos una estructura de algoritmo genético propio. Esto llevará más trabajo, pero se obtendrá un algoritmo más rápido.

4 Árbol de Decisión

Los árboles se utilizan en diversos campos de la informática. A pesar de que son difíciles de representar y visualizar de forma global, resultan una herramienta muy útil para separar situaciones o datos.

La estructura de los árboles se puede usar con muchos fines: ordenar un vector, resolver problemas de alta complejidad computacional (como encontrar la mejor jugada posible en un tablero de ajedrez), representar redes de routers y switches o expresar la gramática de un lenguaje, entre otros.

En el caso que nos ocupa, estamos interesados en un tipo muy especial de árboles, los árboles de decisión.

4.1 Definición

Un grafo es un conjunto de vértices y aristas. Las aristas tienen dos extremos, en cada uno de ellos se sitúa un vértice.

Un grafo simple es aquel que no tiene lazos (aristas cuyos extremos se conectan con el mismo vértice) ni aristas múltiples entre dos de sus vértices.

Un árbol es, pues, un grafo simple G tal que para cada dos vértices de G existe un único camino simple entre ellos, es decir, hay una única sucesión de aristas que empieza en uno de los vértices y termina en el otro sin repetir arista.

Un árbol de decisión será, por tanto, una especialización de árbol en la que a los vértices se los interpreta de la siguiente forma:

- Un vértice especial, llamado nodo raíz, se considera el inicio del camino de decisión.
- Un conjunto de vértices, llamados nodos, se conectan con, al menos, otros dos vértices. En cada nodo hay una condición. La verficación de dicha condición determina qué nodo es el siguiente en el camino de decisión.
- Un conjunto de vértices, llamados hojas o nodos terminales, se conenctan con un único vértice. En cada hoja hay una clasificación del dato que cumple todas las condiciones que tienen los nodos del camino que empieza e la raíz y termina en la hoja.

El objetivo de un árbol de decisión es, por tanto, clasificar datos, de una determinada naturaleza, según una serie de condiciones. La construcción de dicho árbol y la elección de las condiciones las veremos en los siguientes apartados.

En este punto, se procede a plantear un problema clásico para dar un ejemplo de árbol de decisión. En un campeonato de tenis al aire libre son necesarias unas buenas condiciones metereológicas. Para tomar la decisión de si se puede jugar un partido, o no, se puede el árbol de decisión de la figura 7.

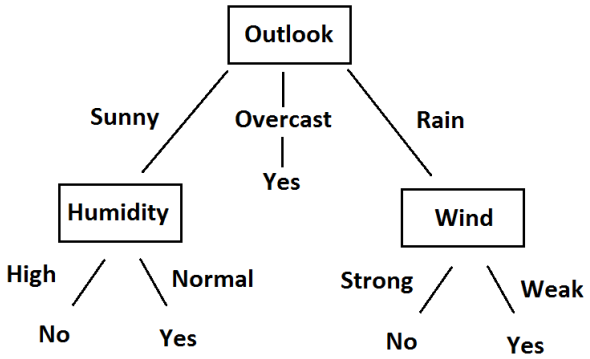

Figura 7: Árbol de decisión para decidir si un partido de tenis se juega.
Fuente: https://sefiks.com [Última consulta 12 de Julio de 2019]

Como se puede ver, el árbol se interpreta del nodo raíz hasta las hojas. Luego la primera condición a comprobar es el estado del cielo: si está nublado, entonces se juega; si por el contrario, está soleado o lloviendo, entonces hay que comprobar otra condición para tomar la decisión.

En este ejemplo es clara la identificación de los elementos del árbol de decisión. La clasificación de los partidos se compone de dos etiquetas: sí se puede jugar y no es factible jugar. Del mismo modo, las condiciones presentes en el árbol son tres:

- **Situación del cielo.** Puede tomar tres valores; soleado, nublado y lluvioso.
- **Humedad.** Condición que puede tomar dos valores; humedad alta y humedad normal.
- **Viento.** Puede tomar dos valores; viento fuerte o viento débil.

En los siguientes epígrafes se desarrollará cómo se pueden construir estos árboles de forma metódica.

4.1.1 Etiquetado

Para poder clasificar un dato, es necesario saber cuáles son las posibles clasificaciones o, dicho de otra manera, cuáles son las clases de los datos. Esta clase viene dada por la etiqueta o *tag*. En el ejemplo anterior del campeonato de tenis las clases serían jugar y no jugar.

Cuando el problema solo tiene dos clases se denomina problema binario. Aunque son los problemas más comunes, en algunas ocasiones puede interesas discutir entre más clasificaciones. Por poner un ejemplo, se puede construir un árbol de decisión que clasifique todos los delitos de código penal atendiendo a la dureza de su pena: baja, media o alta.

En el caso que nos ocupa, el *trading* en bolsa, no se tiene una clasificación natural para los valores de las acciones. Es claro, según el objetivo adelantado en la introducción, que se tiene que diferenciar entre los instantes buenos para comprar y aquellos que son buenos para vender. Pero, la definición de qué es un buen día para comprar no es clara.

En principio, parece buena la idea de clasificar como buenos días de compra aquellos en los que el precio de la acción, tras un periodo razonable, sube.

Crear estas clases de manera que representen un buen momento para comprar o vender es objeto de estudio en nuestro trabajo, pero por motivos de secuenciación, se hablará de ello más tarde.

4.1.2 Conjunto de entrenamiento

Una vez que la estructura del árbol está hecha, es sencillo clasificar un dato. Basta con ir comprobando las condiciones de cada nodo y continuar el recorrido del grafo, según los resultados de estas. Cuando una condición nos dirija a una hoja, encontraremos la clase predicha para ese dato concreto. Pero, ¿cómo podemos crear el árbol?

Aquí entra el juego el conjunto de entrenamiento. Un conjunto de entrenamiento es un compendio de datos de la misma naturaleza que los que queremos clasificar, pero que ya están clasificados. La forma de clasificar estos datos de forma previa a tener el árbol es muy variada. No obstante, la mejor opción suele ser contar con el conocimiento experto de alguien capaz de clasificar los datos.

Por tanto, a partir de este conjunto etiquetado, deberemos inducir un árbol de decisión cuyas hojas clasifiquen bien los datos conocidos. Se han propuesto, en distintos manuales, diferentes formas de construirlos. Si se desea profundizar más en estas construcciones se puede consultar Barros (2015), aquí no se desarrollaran.

De esta forma, si los datos con clase desconocida tienen una procedencia parecida a los datos del conjunto de entrenamiento, serán correctamente clasificados por el árbol construido con los datos de clase conocida. O, al menos, esta es la intención.

Los árboles de decisión también se pueden formar a partir de otros métodos, por ejemplo, a partir de conocimiento de una procedencia segura. En nuestra propuesta, lo haremos a partir de un algoritmo genético.

5 Algoritmo propuesto

En este proyecto de fin de grado, se propone un algoritmo genético basado en árboles de decisión cuyo objetivo es extraer información de señales de compra. Es decir, se pretende diseñar un árbol de decisión que sirva como modelo para saber cuál es el momento idóneo para situar órdenes de compra y de venta en el mercado de valores y extraer beneficio.

5.1 Árboles de decisión

El primer punto a tratar para diseñar el modelo, que se va utilizar para extraer las señales de compra y venta, es la estructura del árbol de decisión. En primer lugar se expone la elección de las etiquetas. A continuación, se desarrollará el contenido de los nodos, en otras palabras, las condiciones que dividen los caminos.

En cuanto a la clasificación de los días de bolsa, se propone un diseño con tres etiquetas:

- Etiqueta *Buy*. Un día clasificado con esta etiqueta, es un día bueno para comprar. Implica que el modelo envía una señal de compra y, consecuentemente, se debe colocar una orden de compra de acciones, tantas como nos sea posible con el capital disponible.

- Etiqueta *Sell*. En el caso de que un día se clasifique con ésta, signfica que es un buen día para vender. Implica que el modelo envía una señal de venta y, por tanto, se ha de colocar una orden de venta de todas las acciones que se tengan.

- Etiqueta *Stop*. Es una etiqueta comodín que se usa cuando las condiciones de un camino no son concluyentes para decidir si hay que comprar o vender. En esta situación el modelo no envía ninguna señal.

Esta elección está motivada por la necesidad de que el modelo sea capaz de transmitir señales de compra y venta. La forma más sencilla de conseguir estos envíos es utilizar dos etiquetas, una de compra y otra de venta. Pero, de ser así, todos los caminos de condiciones llevarían al modelo a una compra o a una venta. En ciertas ocasiones, el estado de la bolsa puede no ser determinante para decidir si es un momento de compra o de venta. Es entonces cuando aparece, de forma lógica, la tercer etiqueta, ni es momento

de comprar ni lo es de vender.

Una vez conocida la posible clasificación, se debe desarrollar cuáles van a ser las características que van a definir la clase de un día. En otras palabras, es necesario conocer las condiciones que van a constituir los nodos. En el árbol se van a sitúar condiciones que dividen a los nodos en dos ramas, la que cumple la condición y la que no. Por tanto, todos los nodos serán de bifurcación binaria. Las reglas que se van a evaluar son de la forma:

$$indicador(parametros) <= pivote$$

.

Donde *pivote* es una un número real e *indicador* es eso, un indicador de bolsa, con parámetros o no, según la naturaleza del indicador. Se ha usado la siguiente lista con indicadores clásicos:

- MACD
- ATR
- ROC
- EMA
- SMA
- Momento
- HILL
- RSI
- OBV
- AD
- TRANGE
- BBANDS_HIGH
- BBANDS_LOW

En el caso de algunos indicadores, como EMA (*Exponential Moving Average*) o SMA (*Simple Moving Average*), se hace necesaria una ligera modificación. Esto es debido a que al entrenar un árbol en un determinado periodo,

en este aparecerán reglas como $EMA(10) <= 500$. La potencia del indicador EMA se encuenta al comparar su valor con el valor actual de la acción y observar si se ha producido una bajada o subida importante respecto a los últimos días.

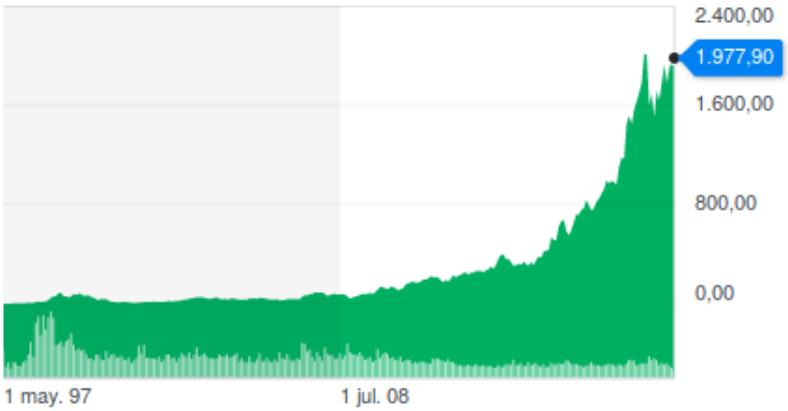

Figura 8: Valor de las acciones de Amazon.
Fuente: https://finance.yahoo.com [Última consulta 19 de Julio de 2019]

En ciertos casos, por ejemplo en tendencias laterales, podría tener sentido usar esta condición. Si el valor de la acción lleva unos días demasiado alto o demasiado bajo puede signficar un cambio de tendencia.

Pero, en otros casos, resulta fatal. Imaginemos que nuestro modelo aprende en un periodo de entrenamiento en que una condición con EMA es muy significante. Al pasar al periodo de prueba, esta condición puede ser absurda, como en el caso que se ve en la figura 8 de Amazon. El valor es solo creciente, no tiene sentido poner un pivote fijo.

En su lugar, se propone modificar este indicador restando o dividiendo a su valor el precio actual de la acción para, de alguna forma, escalar el indicador y hacerlo intemporal. En el caso de la división, se obtiene la proporción entre el valor actual y un valor representante de los últimos días. Por su parte, la resta es un poco menos aconsejable, ya que hay una gran diferencia entre que una acción a 2 euros suba 10, y que una acción de 500 euros suba 10. Ambos casos son positivos, pero el primero es mucho más significativo.

(SE PUEDE AÑADIR AQUÍ UNA SÍNTESIS SOBRE LOS INDICADORES USADOS Y LAS MODIFICACIONES O PARÁMETROS INICIALES (los parámetros cambian con el algoritmo genético))

5.2 Estructura genética

Para construir el árbol modelo, se utilizará un patrón genético como el explicado en el apartado 3, en el que, oportunamente, se hizo una explicación genérica de un algoritmo genético. A continuación se procede a replicar esa construcción adaptándola a nuestra población, los árboles de decisión.

En primer lugar, se crea una generación y, mediante una simulación de inversión en un periodo de prueba, se puntuan los árboles. A continuación se realiza un proceso de selección y de cruce de los árboles. Y por último, se sortea una serie de mutaciones. Una vez terminado el procedimiento, vuelve a comenzar el proceso de simulación para encontrar la próxima generación.

A continuación se va a profundizar en cada uno de los pasos seguidos en este procedimiento.

5.2.1 Primera generación

La primera generación en los algoritmos genéticos suele ser aleatoria y la lógica de los siguientes pasos va perfilando y mejorando la población a largo plazo.

Puesto que nuestra simulación se prevé costosa, se va a proceder a realizar un ligero calentamiento de la población para que se reduzcan las iteraciones necesarias hasta la obtención de una población buena. Con el fin calentar los árboles en la primera población, se propone usar una construcción con algunos elementos aleatorios y otros heurísticos.

En primer lugar, para cada árbol se toman indicadores de forma aleatoria que conforman los nodos. Cada vez que se añade un nuevo nodo con su indicador, el pivote que divide los datos se toma a partir de los mejores resultados de una función de entropía. La función de entropía se aplica sobre los datos del periodo de prueba. A continuación se explicará detalladamente cada paso.

5.2.1.1 Etiquetado de datos de prueba

El mismo periodo que se va a usar para entrenar el algoritmo genético va a ser utilizado para generar esta primera población. Así pues, se toman estos datos y se etiquetan siguiendo estas consideraciones:

Nombremos máxima subida, y lo notamos como M, a la siguiente expresión:

$$M = max\{cierre_i - cierre_j \mid \forall z \, con \, i < z \leq j \; cierre_{z-1} - cierre_z \geq 0\}$$

De forma análoga, se obtiene la máxima bajada, notada como m:

$$m = max\{cierre_i - cierre_j \mid \forall z \, con \, i < z \leq j \; scierre_{z-1} - cierre_z \leq 0\}$$

Seguidamente, se definen dos tipos de tendencias.

Una consecución de días tiene tendencia positiva siempre que en mitad no se halle una bajada del precio superior a la sexta parte de m.

De forma análoga, una consecución de días se dice con tendencia negativa si en mitad se halla una subida superior a la sexta parte de M.

Es decir,

$$\{i, i+1, ..., j\} \text{ tiene tendencia positiva} \iff$$
$$\forall z, k \, con \, i < k < z \leq j \quad cierre_z - cierre_k \geq m/6$$

$$\{i, i+1, ..., j\} \text{ tiene tendencia negativa} \iff$$
$$\forall z, k \, con \, i < k < z \leq j \quad cierre_z - cierre_k \leq M/6$$

En última instancia, marcan los datos con cuatro etiquetas:

- **-2** para el último día de cada tendencia negativa.
- **-1** para el resto de días de las tendencias negativas.
- **2** para el último día de cada tendencia positiva.
- **1** para el resto de días de las tendencias positivas.

Estas etiquetas son un intento de marcar las diferencias de días buenos para comprar y días buenos para vender.

En consecuencia, al sumar las etiquetas de un grupo de días, cuanto más negativo sea el resultado, más influencia tienen los mínimos de las tendencias negativas y, por tanto, los días están orientados a comprar.

En el caso contrario, una suma positiva implica una buena fecha para vender.

En la figura 9 se puede observar un ejemplo simple de etiquetado de un periodo. En ella se ven los tramos clasificados como crecientes en azul y los decrecientes en rojo. Hay que destacar que la transición de D a E es positiva,

pero está en mitad de un tramo decreciente y la diferencia no supera la sexta parte de la subida permitida por lo que, según lo anteriormente expuesto, se marca también como decreciente.

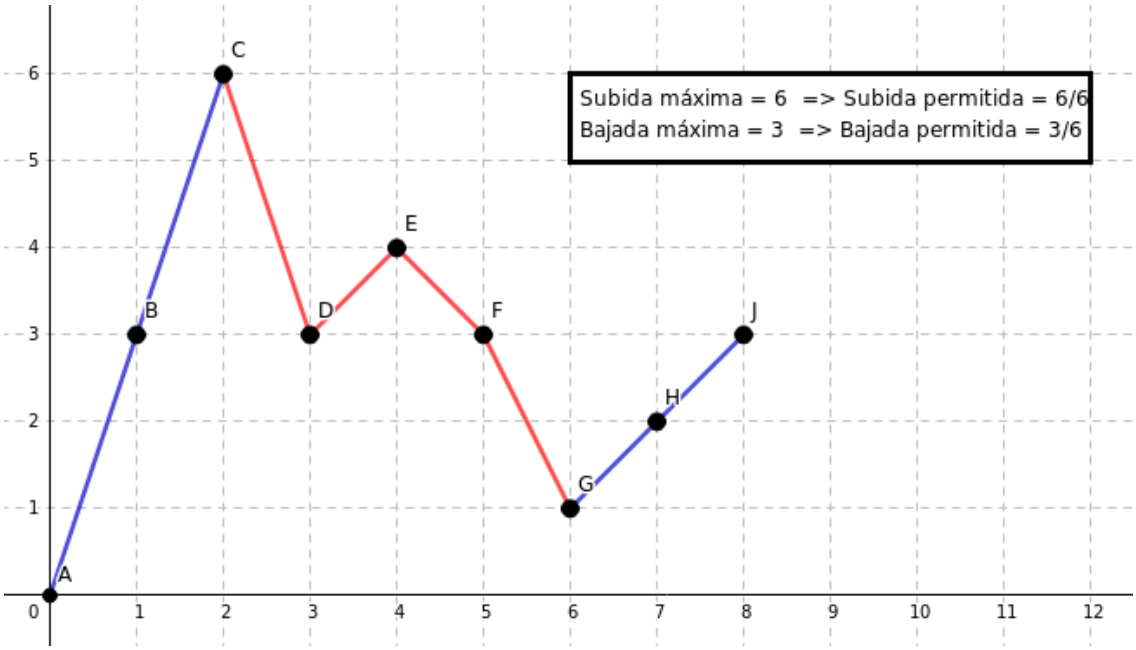

Figura 9: Ejemplo de etiquetado de un periodo.
Fuente: elaboración propia

5.2.1.2 Selección de pivotes

Una vez se ha seleccionado un indicador para el nodo de un árbol de la primera generación, de forma aleatoria, el pivote se selecciona de forma heurística. Hay que destacar que el pivote seleccionado no es el mejor de todos los posibles, simplemente se considera una elección para rectificar la aleatoridad de la selección de indicadores.

El pivote final elegido es un valor de una rejilla conformada por diez valores equidistantes situados entre el máximo y el mínimo valor del parámetro de los datos que cumplen las condiciones para llegar al nodo.

Puesto que tenemos cuatro etiquetas, en primera instancia se debería usar una función de entropía de cuatro clases. Pero las etiquetas 2 y -2 son poco frecuentes. Además, debido a que nuestros nodos dividen los datos en dos ramas, no parece oportuno usar una entropía de cuatro clases. En su lugar proponemos hacer dos clases, una para las etiquetas negativas (-1 y -2), y otra para las positivas (1 y 2).

La función de entropía de dos clases es, por tanto:

$$entropy(x) = x * log_2(x) + (1-x)log_2(1-x) \quad x \in (0,1)$$

Donde x es la proporción de una de las clases. De este modo, en caso de que la proporción de una clase sea 1, se obtiene la información perfecta (la rama solo tiene datos de una clase), pero entonces, la función entropía no puede calcularse. Hay que tener en cuenta este caso especial y evitar realizar el cálculo de la función.

Para calcular el mejor pivote de un nodo, se evalúan las entropías de las dos ramas que forma y se hace la suma proporcional al número de individuos. El pivote cuya función de entropía sea mayor es seleccionado como mejor pivote de la rejilla.

5.2.1.3 Selección de las hojas

Cuando los datos que cumplen las condiciones de una rama son pocos o la rama ya tiene demasiada profundidad en este calentamiento de la primera generación, es necesario dar por terminada la rama y culminar la construcción con una hoja.

Las hojas son, simplemente, una de las tres señales posibles que permite el modelo propuesto: comprar, vender o no hacer nada.

Para escribir la decisión que se debe tomar en esa hoja se va a usar una vez más la etiqueta puesta en los datos de prueba. Notando $x_{tag=a}$ al número de instancias de los datos que cumplen el camino de condiciones y cuya etiqueta es a, definimos la variable que va a determinar la acción de la hoja como:

$$\alpha = \frac{x_{tag>0} - x_{tag<0} + x_{tag=2} - x_{tag=-2}}{x_{tag>0} + x_{tag<0}}$$

Es importante destacar que, para cada hoja, los datos a tratar son aquellos que cumplen las condiciones del camino de nodos hasta la hoja. De este modo, por tanteo se llega a la siguiente adjudicación de la señal de la hoja:

$$\text{señal} = \begin{cases} \text{Comprar} & \text{si} \quad \alpha < -2 \\ \text{Stop} & \text{si} \quad -2 \leq \alpha \leq 2 \\ \text{Vender} & \text{si} \quad 2 < \alpha \end{cases}$$

De forma coherente a la señal que se quiere enviar, se colocan las etiquetas.

5.2.2 Cruce

En cada iteración del algoritmo se evalúa la población con *backtrader* para simular los beneficios que cada individuo tendría en el periodo de prueba. Una vez extraídos los beneficios, se procede a generar la nueva población mediante un cruce que promocione a los árboles que mejores resultados han dado.

De entre las distintas formas que hay para seleccionar los individuos que se van a reproducir, se escoge la selección proporcional. Más tarde, se explicará el cruce natural de dos árboles de decisión[11].

Este cruce natural entre árboles tiene una pequeña desventaja, y es que puede generar peores poblaciones que las progenitoras. Se hace prácticamente indispensable, entonces que activemos los métodos elitistas y eliminatorios. Los mejores árboles pasarán dos a la siguiente generación, una vez mutados y otra vez sin modificar. Por su parte, los peores se descartarán por completo tanto del cruce como de la mutación.

5.2.2.1 Probabilidades de reproducción

En primer lugar, hay que calcular la probabilidad de cruce de cada individuo. Para ello se hace una normalización especial de los beneficios.

Restando el beneficio del peor árbol a las puntuaciones de todos, se consique que estas sean siempre mayores o iguales que 0.

A continuación, con el fin de potenciar las estrategias de compra y venta múltiple, más ventajosas que la estrategia de *buy&hold*[12], se van a modificar ligeramente las ganacias. Por cada venta que produzca un árbol, se suma una cantidad fija de euros a sus beneficios. No obstante, para ciertos valores demasiado altos de esta cantidad con los que se supere las comisiones, las compras y ventas se realizarán de forma compulsiva. Es importante, por consiguiente, ajustar bien este valor para evitar este resultado.

En el último paso de la normalizacion, se dividen todas las puntuaciones entre la suma de las puntuaciones. Esto produce que la suma de las mismas sea 1 y, además, cada valoración es proporcional a la cantidad de beneficios que reporta, a excepción de la cantidad añadida en favor de las transacciones rápidas.

[11] Véase la sección 5.2.2.2

[12] La estrategia *buy&hold* es una estrategia clásica que se basa en la compra inicial de todas las acciones posibles y la venta final de las mismas. Esta estrategia recibe su nombre por la lógica que sigue: comprar y mantener.

Por último, tomando $x \sim U[0,1]$ y notando p_i como la puntuación normalizada del árbol i-ésimo, se pueden sortear los individuos que se reproducen de forma proporcional a su beneficio de la siguiente forma:

$$seleccionado = \begin{cases} \text{árbol 1} & \text{si} \quad 0 \leq x < p_1 \\ \text{árbol 2} & \text{si} \quad p_1 \leq x < p_1 + p_2 \\ ... & \\ \text{árbol n} & \text{si} \quad \sum_{i=1}^{n-1} p_i \leq x \leq \sum_{i=1}^{n} p_i = 1 \end{cases}$$

Son necesarios dos árboles para realizar un cruce, del que, a su vez, resultan otros dos árboles. Se tienen que ejecutar tantas selecciones como sean necesarias para que la siguiente generación mantenga el número de individuos.

Una anotación final es necesaria. En el caso especial de que todos los árboles tengan los mismos beneficios, este procedimiento llevaría a dividir por 0. Para evitar esto, de forma natural se divide el intervalo [0,1] en tantos intervalos de igual dimensión como árboles haya y continuamos con el caso general. A cada árbol se le asigna un intervalo, esta vez, al ser todos iguales, no importa cuál y la función de selección podría ser la siguiente:

$$seleccionado = \begin{cases} \text{árbol 1} & \text{si} \quad 0 \leq x < \frac{1}{n} \\ \text{árbol 2} & \text{si} \quad \frac{1}{n} \leq x < \frac{2}{n} \\ ... & \\ \text{árbol n} & \text{si} \quad \frac{n-1}{n} \leq x \leq 1 \end{cases}$$

5.2.2.2 Cruce natural entre dos árboles

El cruce entre individuos de la población es el mecanismo central de los algoritmos genéticos. En nuestra propuesta, al tener una población de árboles, se va a usar el cruce natural entre estos.

Como ya se ha introducido en el punto anterior, con este cruce propuesto no se asegura que los árboles de una generación sean mejores que los progenitores. No obstante, entendemos que es una buena forma de agrupar condiciones y de crear asociaciones diferentes de estas. En consecuencia, esto permite explorar el espacio de búsqueda de forma que una generación contenga modelos parecidos para la predicción en bolsa.

El cruce parte de dos árboles, llamados padres o progenitores. En cada árbol progenitor se marca, con un patrón aleatorio, un nodo. Es indispensable que dicho nodo no sea una hoja, aunque sí podría ser la raíz.

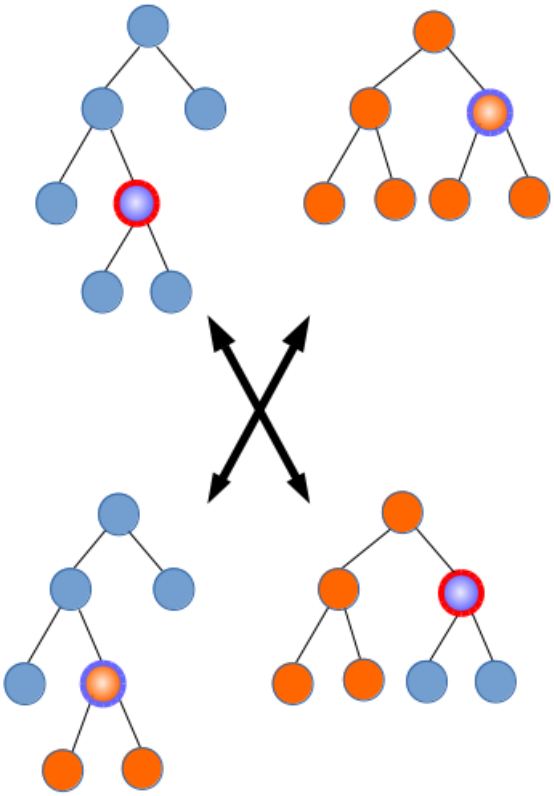

Figura 10: Ejemplo de cruce entre dos árboles.
Fuente: elaboración propia

Para construir a los hijos, se intercambian sendos nodos seleccionados y, con ellos, todos los nodos que penden de él. En la figura 10 puede verse una descripción gráfica del cruce.

Este cruce puede producir varias anomalías al iterar. Una de ellas es la generación de árboles con una profundidad extrema. Es decir, la población tiene una cierta probabilidad de converger a árboles con demasiados nodos o, en su defecto, con insuficientes nodos.

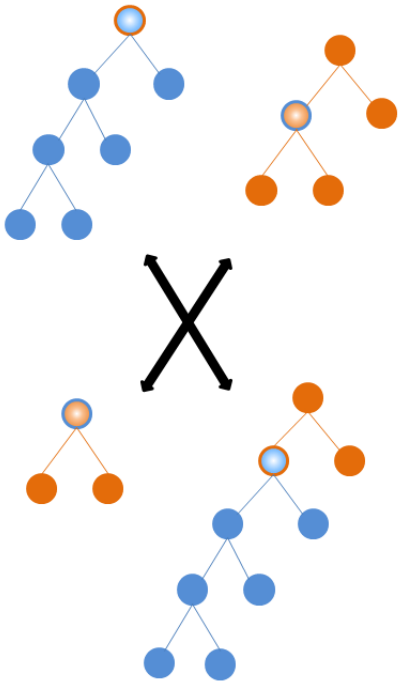

Figura 11: Ejemplo de generación de un árbol pequeño y otro grande.
Fuente: elaboración propia

Por lo general, un exceso de nodos, que puede entenderse como un exceso de condiciones, va a producir modelos muy específicos. Los árboles demasiado específicos suelen aprender muy bien la casuística de los dos datos de prueba. Sin embargo, fallan al intentar extenderse al periodo de prueba. Esto es lo que en el ámbito de la ciencia de datos se denomina sobreaprendizaje.

En contraposición, un árbol con pocas condiciones o nodos, va a tender a generalizar demasiado las situaciones y, a menudo no, son capaces si quiera de obtener un bueno modelo para los datos de entrenamiento.

Para solventar estos dos problemas se actuará de la siguiente forma:

- La selección de nodo a intercambiar en el cruce no se hace de forma uniforme. Empezando desde el nodo raíz y con igual probabilidad, se sortea si el nodo elegido para intercambio es una de tres: el actual, está en la rama izquierda o está en la rama derecha. Por consiguiente, es más probable que se tome un nodo cercano a la raíz para hacer el cambio. Nótese que no es posible intercambiar una hoja, luego si en este procedimiento se llega a una hoja, el nodo a intercambiar es el inmediatamente superior.

En un intento de justificar la anulación de las hojas como punto de mutación se pide reflexionar sobre el siguiente caso extremo. Para generar dos nuevos árboles, los progenitores han marcado como punto de corte una hoja, el primer progenitor, y la raíz, el segundo. Por tanto, al generar a los nuevos hijos, el primer árbol sería el primer progenitor al completo al que se le ha añadido el segundo en una hoja. El segundo hijo, por su parte, remplazaría todo el segundo árbol progenitor por una hoja del primero. Esto es, el segundo hijo no tienes nodos y, se entiende que en este caso especial, clasifica todos los días con la misma etiqueta, la que aparece en su única hoja.

Esta forma condicionada de seleccionar los nodos tiene como objetivo evitar la formación de árboles muy pequeños. Ya que, para que una rama grande se sustituya por una rama pequeña, es necesario que la selección de nodo baje mucho en la jerarquía. Pero la probabilidad de que la selección baje mucho es una probabilidad condicionada múltiples veces.

- Para eliminar los árboles demasiado grandes, simplemente se propone desarrollar un método que cuente los nodos de un árbol y, en cada generación, se eliminen los que superen una cierta cantidad. A pesar de que el método anterior parece efectivo para controlar los árboles pequeños, también se ha añadido el caso contrario: si hay pocos nodos en un árbol, éste se elimina.

  De forma inicial, hemos acotado los nodos de un árbol entre 15 y 45. No obstante estas cifras son orientantivas, tras ver algunos resultados podría ser conveniente cambiarlas.

  Para mantener constante la cantidad de inviduos por población, se tiene que cruzar tantos más árboles como la mitad de eliminaciones se hallan producido.

5.2.3 Mutaciones

Tal y como se han presentado el cruce y la primera generación, hay varios factores de la población que no es posible alterar, estos son:

- **Los parámetros de los indicadores.** Algunos indicadores, como el EMA (Exponential Moving Average), depende de uno o varios parámetros. En primera instancia se han impuesto unos valores, con cierta variabilidad, para ellos. El cruce permite cambiar los nodos de posición y, a la larga, cambiar las combinaciones de indicadores pero, en ningún caso, se produce un cambio en los parámetros de los indicadores. Los

valores impuestos para estos en un inicio no tiene por qué ser óptimos. Además, la idoneidad de un valor depende, en cierta medida, del conjunto de indicadores que precedan a la condición. Se crea, entonces, la necesidad de establecer un mecanismo para cambiarlos.

- **La combinación entre el último nodo y las etiquetas.** El cruce propuesto no permite mover las hojas, salvo en el caso de que lo que se mueva sea un nodo superior. Esto hace que las señales vayan asociadas a unos indicadores elegidos en la primera iteración de forma aleatoria lo que, a poco que se mire, parece una dura restricción. Es necesario crear un método que cambie las etiquetas.

- **Los pivotes de cada nodo.** A partir de la función de entropía y los datos de entrenamiento se eligen los primeros pivotes de la primera generación. Pero una vez que el nodo cambia de posición a causa del cruce, el pivote no tiene por qué estar bien elegido. Es necesario entonces un proceso para cambiar los pivotes.

El mecanismo que desarrollaremos en los sucesivos epígrafes, con el que podemos usar para alterar todos estos factores estáticos, es la mutación.

Las mutaciones suceden con cierta probabilidad sobre distintos lugares de los individuos. Según la naturaleza de estos, se pueden plantear distintos métodos de mutación.

En nuestro caso, se irá recorriendo el árbol nodo por nodo desde la raíz hasta las hojas. En cada nodo se toma un entero aleatorio en un rango dado. Según el valor extraído, se ejecuta una mutación de parámetros, de pivotes, de indicadores de hojas (en caso de que el nodo fuese una hoja) o ninguna. A continuación se desarrolla cada mutación.

5.2.3.1 Mutaciones en indicadores

El cruce permite cambiar la composición de condiciones que conforman el modelo. No obstante, es un cambio muy general. En ocasiones, cambiando un único indicador de un nodo en mitad de una rama se produzca un cambio muy positivo.

Ya sea con el objetivo de provocar estos cambios o simplemente para explorar más árboles en el espacio de búsqueda similares a la generación anterior, se propone que, de forma aleatoria, se tome una nueva función indicadora que suplante a la anterior.

En consecuencia a este cambio, mantener el pivote que se situaba en el mismo nodo carece de sentido, pues cada indicador tiene imagen en una horquilla distinta de valores. Se hace obligado, por tanto, recalcular el pivote del nodo.

5.2.3.2 Mutaciones en los pivotes

Todos los indicadores utilizados tienen su imagen en $\mathbb{R}$, luego los pivotes que condicionan también. Existen varias formas de mutar un valor real. Aquí se propone introducir un ruido basado en la normal.

En cada mutación de pivote, el nuevo pivote es un valor extraido de una distribución normal $N(pivote, |pivote/4|)$, es decir, se saca de una normal centrada en el antiguo valor y con varianza la cuarta parte del antiguo valor.

5.2.3.3 Mutaciones en parámetros de indicadores

La mutación de parámetros es específica, según el indicador al que pertenezca.

Los parámetros que se mueven en $\mathbb{R}$ pueden calcularse, igual que la mutación de pivotes, a partir de una distribución normal. En el caso de que el parámetro se mueva en $\mathbb{R}_0^+$, como en el caso de las *Bollinger Bands*, se debe rectificar con un valor absoluto.

En cuanto a los parámetros que se mueven en $\mathbb{N}$ o subconjuntos del mismo, optamos por hacer una mutación discreta sumando o restando 2.

5.2.3.4 Mutaciones en las hojas

Las mutaciones en una hoja son bastante sencillas. Una hoja puede tener los valores *'Compra'*, *'Vende'* o *'Stop'*. Basta con tomar una nueva etiqueta de forma aleatoria.

6 Detalles de la implementación

En esta sección se va a tratar de aclarar algunos puntos de la implementación del software, como el diagrama de clases y las optimizaciones de código para reducir el tiempo de ejecución.

El código completo realizado para este proyecto no se puede mostrar aquí por motivos de espacio. Se aporta como anexo. Para ejecutar el programa basta con invocar el script *genetreec.py* con python. El script dará error si no se han instalado los paquetes necesarios (véase sección 2.4.2.1).

Todo el código se encuentra comentado con las aclaraciones que se han creído necesarias.

El software se divide en cuatro archivos:

- **genetreec.py**. Es el núcleo del programa. Contiene las clases *Simulate*, *TreeStrategy* y *EndStats*, necesarias para simular el *backtesting*. Tiene una fuerte dependencia con el paquete *backtrader*.
- **indicator.py**. Posee todos los indicadores usados y la herramienta para llamarlos y evaluar las fechas que se requiera.
- **tagger.py**. Etiqueta los datos para el calentamiento.
- **tree.py**. En él está la definición de árbol, así como las hojas y los nodos. Contiene las clases *Leaf*, *Node* y *Tree*

6.1 Diagrama de clases

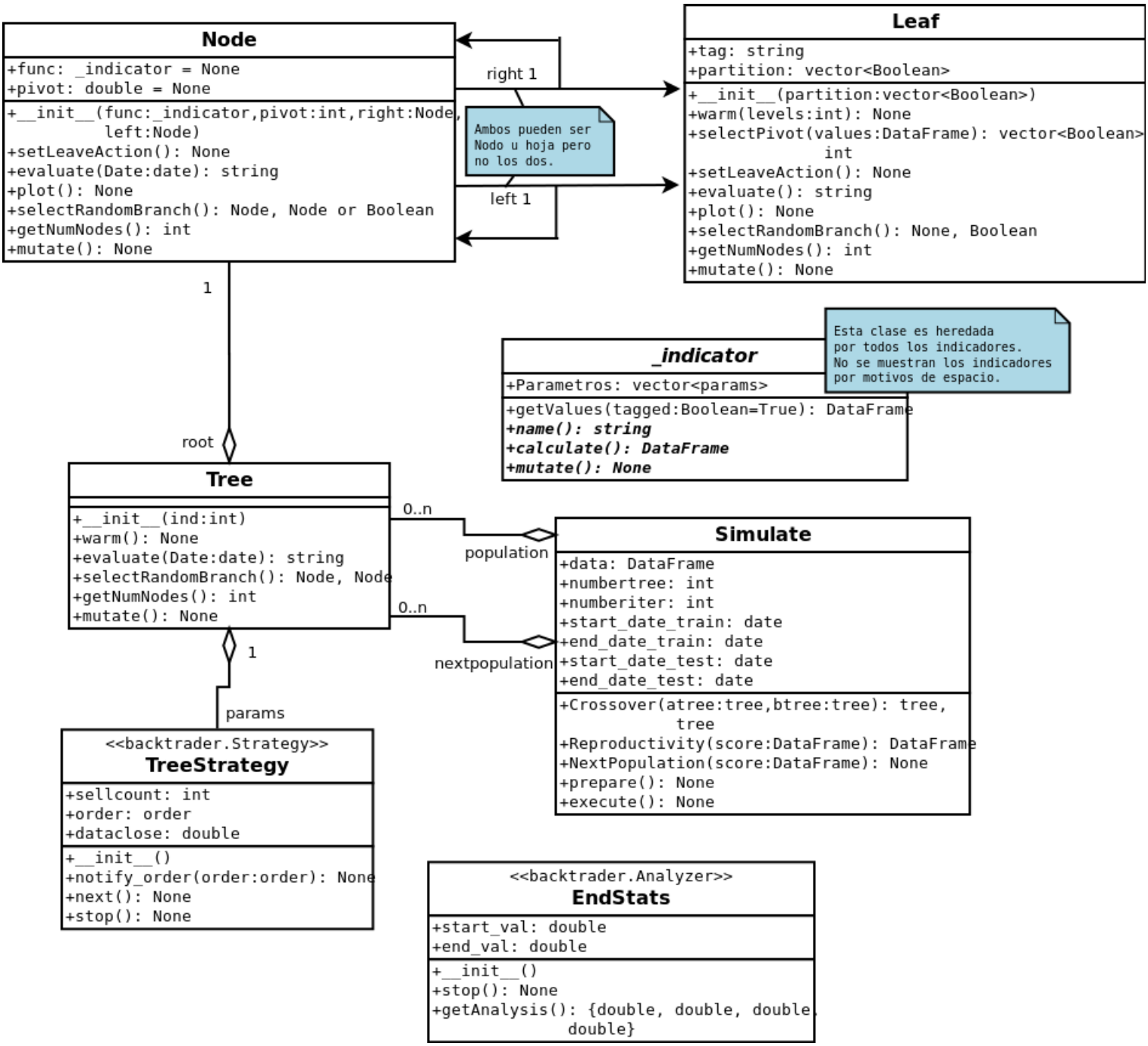

Figura 12: Diagrama de clases del software desarrollado.
Fuente: elaboración propia

6.2 Optimizaciones

Los algoritmos evolutivos suelen tener unos buenos resultados, no solo por su caracter natural, si no por su amplia capacidad para inspeccionar el espacio de búsqueda cercano a las buenas soluciones.

En espacios de búsqueda amplios, como es el caso, las combinaciones son muy grandes y las búsquedas se alargan mucho. A esto hay que añadirle el peso de la función *fitness* que, en este caso, conlleva simular un periodo de bolsa y evaluar muchas veces los indicadores en distintas fechas. Adeás, se dispone de una máquina bastante modesta para su ejecución.

Todos estos inconvenientes hacen necesario poner especial cuidado a la hora de realizar el software e intentar reducir el gasto de tanto tiempo como espacio. Si bien este último, en principio, no nos producirá problema.

6.2.1 Paralelización

Al utilizar *backtrader* en la introducción (véase 2.4.2) se mostraron varios ejemplos sencillos de uso. Pero el paquete dispone de otras opciones que permiten adaptarlo a nuestras necesidades. Se comentará, a continuación, la posibilidad de ejecutar en paralelo.

Lo primero que se debe reformular para poder ejecutar en paralelo es la estrategia. Como se concretó, la clase *cerebro* recibe una estrategia como parámetro, que es quien dá las señales de compra y venta. De esta forma, si se simulase cada uno de los árboles de la población, digamos 50, se tendrían que crear 50 cerebros, añadir 50 veces los datos y simular 50 veces las mismas fechas (cada vez con un individuo distinto).

Para corregir este comportamiento, se sugiere usar la función *optstrategy* de la clase *cerebro* de la siguiente forma:

```
cerebro.optstrategy(TreeStrategy, tree=list(population))
```

En realidad, esta función está pensada para añadir una rejilla de parámetros que matizan la estrategia definida. En lugar de esto, nosotros vamos a pasar la propia población de árboles como rejilla de parámetros.

Una vez realizado este cambio, Backtrader ejecutará la estrategia con los distintos árboles de forma iterativa. No es necesario cargar los datos múltiples veces, basta con una sola copia de estos.

Ahora vamos a eliminar la linealida de ejecución. Backtrader permite ejecutar de forma paralela una estrategia a la que se ha aportado una re-

jilla de parámetros. Con la directiva *cerebro = bt.Cerebro(maxcpus=None)* indicamos que no hay límite de núcleos para la ejecución.

La máquina usada dispone de 4 núcleos. Esto nos da una mejora signficativa de tiempo, ya que nos permite evaluar 4 individuos a la vez. Cuantos más núcleos tenga la máquina, más acentuada será la mejora.

6.2.2 Caché para indicadores de primer orden

La parte computacional más costosa es la evaluación de los indicadores. El cálculo de algunos de ellos depende de los valores de la acción de varios dás atrás. Cargar esos datos y hacer las operaciones repetidas veces es un gasto innecesario de tiempo.

Además los árboles tienen varios niveles de profundidad. Para cada día que se simule la inversión se calculan la misma cantidad de indicadores que de niveles.

En un intento de reducir esto, se va usar el paquete de python TA-Lib[13]. TA-Lib es un paquete de cálculo para indicadores técnicos de bolsa. Este software está pensado para calcular un gran número de indicadores de bolsa a lo largo de un periodo, es decir, aportados los valores de un periodo de tiempo, te devuelve el indicador en el mismo periodo.

Un caso de uso sería el siguiente, en el que se calcula el indicador EMA:

```
data['EMA'] = talib.EMA(df['Close'], period)
```

La fuerza de este paquete reside en la capacidad de calcular indicadores en espacios de tiempo grande. Para sacar partido de este hecho, en lugar de ir calculando los indicadores en los días que nos interesan, se va a optar por calcularlo en todos los días del periodo.

Se realiza entonces un *DataFrame* en el que se van almacenando todos los indicadore calculados. Cada vez que se quiere acceder a un indicador en una fecha, primero se comprueba si está ya calculado y, en caso negativo, se calcula y se guarda. Este almacén puede perdurar incluso entre distintas generaciones, ya que los árboles deberían tener indicadores con parámetros iguales o similares.

De forma sistemática, cuando se requiera un indicador, se accederá la función *getValues* de cada indicador, que mantiene esta estructura de guardado de datos:

[13]https://mrjbq7.github.io/ta-lib/doc_index.html. Última consulta 25 de Julio de 2019

```
def getValues(self):
        if self.name() in df.columns.values:  # df es global
                return df[self.name()]
        else:
                return self.calculate()
```

Para guardar los datos, cada columna se ha nombrado de forma única para cada indicador. El nombre se conforma por el nombre del indicador y los parámetros que hubiere.

6.2.3 Caché para indicadores de segundo orden

La primera caché elimina coste computacional calculando los indicadores en todas las fechas de una sola vez. Ahora surge un problema provocado por este hecho, cada día hay que acceder a los indicadores de ese día en unas 4 o 5 ocasiones (una por nivel de profundidad).

Por consiguiente, se van a producir en la simulación del mismo día varias búsquedas en un *DataFrame* cuyo índice son fechas. A priori, se podría pensar que buscar por fechas es igual que buscar por un entero, pero no es así. Python tiene un tipo especial para tratar con fechas que hace que las búsquedas se realenticen mucho.

Para evitar esta coste adicional de búsqueda, se propone realizar un segundo nivel de caché. En esta ocasión, este almacén va a guardar los indicadores del último día del que se pidió un indicador.

Como en el mismo día se va a requerir el uso de varios indicadores, estamos ahorrando búsquedas en el *DataFrame* que conforma la primera caché.

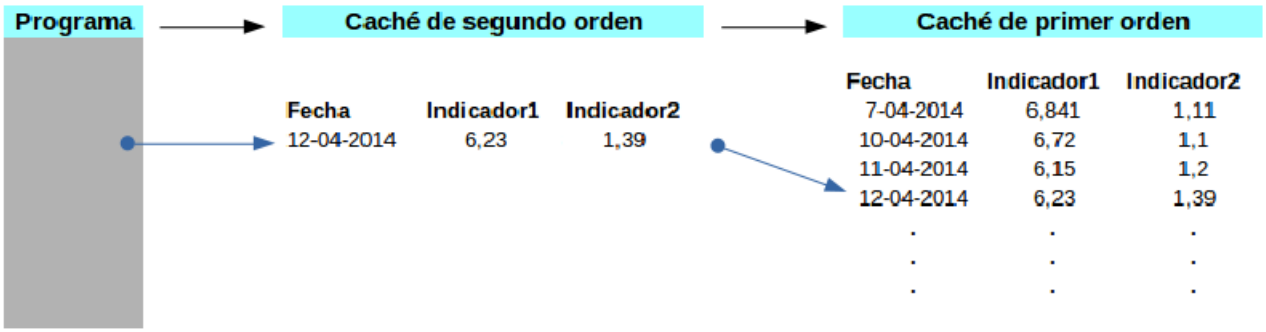

Figura 13: Estructua de las cachés.
Fuente: elaboración propia

El código sigue la misma idea que el desarrollado para la primera caché, solo hay que tener en cuenta que es necesario hacer una nueva variable global

para guardar los datos del día. Además, ahora la caché puede fallar por dos motivos distintos. El primero que el indicador no esté en la primer caché, por tanto es necesario calcular los valores del indicador e incluirlos en el *dataframe*. Y el segundo, que los datos guardados en la segunda caché sean de otro día, en cuyo caso es necesario cargar el día correcto en ésta.

```
def getValueByIndex(index, func):
        global thisday
        if func.name() in thisday.columns.values:
                if thisday.index != index:
                        thisday = df.loc[[index]]
                return thisday[func.name()][0]
        ret = func.getValues(False).loc[index]
        thisday = df.loc[[index]]
        return thisday[func.name()][0]
```

7 Análisis de resultados

Una vez que el algoritmo está totalmente desarrollado y es funcional, se lanzan un compendio de ejecuciones con distintas combinaciones de parámetros con el objetivo de ver cuán bueno es. En un intento de mejorar la comprensión de todas las ejecuciones realizadas, se va a dividir esta sección en dos epígrafes.

En el primero de ellos se hace un análisis de los resultados obtenidos haciendo variaciones en los parámetros del algoritmo genético. Estos son el número de árboles, el número de iteraciones y el tipo de periodo.

Más tarde, en el segundo epígrafe, se van a comparar los resultados obtenidos, a través de los mejores parámetros, con otro algortimo de naturaleza parecida desarrollado en un artículo.

7.1 Análisis de parámetros

Para medir el rendimiento del algoritmo en distintas situaciones, se han propuestos varios periodos de diferente longitud y que presentan distintas tendencias. Por sencillez, todos han sido seleccionados del mismo valor de bolsa, *Banco Santander*.

Para obtener una mejor comprensión, se han definido cinco configuraciones de parámetros. Cada configuración tiene sus parámetros establecidos en la figura 14. Estas configuraciones, dependiendo de la longitud del periodo, pueden tardar desde unos minutos en ejecutar hasta cuatro horas en el caso más grande.

	Número de árboles	Número de iteraciones
Corto	40	60
Medio-Corto	60	80
Medio	60	200
Largo-Medio	100	100
Largo	100	400

Figura 14: Configuraciones de parámetros.
Fuente: elaboración propia

Nótese que estas aproximaciones de tiempo ejecución se aportan a partir

de la máquina disponible, en este caso, de cuatro núcleos y 3.4GHz.

En los casos expuestos a continuación se sigue siempre el mismo procedimiento. En primer lugar, el algoritmom genético efectúa, con los parámetros correspondientes, una serie de ejecuciones que terminan con una población evolucionada. De esta población obtenida, se selecciona el mejor de los individuos y se prueba tanto en el periodo de entrenamiento como en el periodo de prueba.

7.1.1 Perido largo

En la primera de las ejecuciones, el periodo de entrenamiento comprende desde el 1 de abril de 2012 hasta el 1 de enero de 2016, por tanto, corresponde con un periodo bastante largo. De forma coherente, el periodo de prueba empieza el 1 de enero de 2016 y finaliza el 1 de mayo de 2019. Al comparar los perfiles de los precios en sendos periodos en las gráficas, que se muestran a continuación, se puede observar que son parecidos, lo que debería mejorar los resultados.

		Largo	Largo-Medio	Medio	Medio-Corto	Corto	Buy&Hold
Entrenamiento	**01/04/2012 – 01/01/2016**	86.68%	98,01%	105.17%	95,98%	99,69%	-33,29%
Prueba	**01/01/2016 – 01/05/2019**	24,38%	54,17%	74,79%	0,00%	53,73%	1,02%

Figura 15: Ejecución en un periodo largo ascendente-descendente. Las cantidades mostradas son el porcentaje de beneficio respecto al presupuesto inicial.
Fuente: elaboración propia

A priori puede parecer que los resultados, que se pueden ver en la figura 15 son muy buenos. Pero, en realidad, muestran algunas carencias que se hacen obvias cuando se ve el calendario de inversión.

Lo primero que destaca en la figura 16, que representa el calendario de inversión del algoritmo con parámetros de perfil medio, es que la gran mayoría de las ganancias proviene de una única compraventa. El resto de inversiones son de poca duración y apenas si producen ganancias. La causa de este hecho puede ser una falta de especialización. A lo largo de la ejecución, algún árbol ha conseguido hacer esa transacción tan ventajosa y, a partir de esta, no se ha conseguido mejorar prácticamente nada.

Es claro que, una inversión de compra y venta algo más rápida nos habría aportado una mayor ganacia.

Cuando analizamos el periodo de prueba, representado en la figura 17, obtenemos un perfil similar. Se ha ajustado de manera muy eficiente una única compraventa. Si bien es cierto que esa transacción es casi perfecta, hay otras transacciones válidas que no se han ejecutado.

Figura 16: Calendario de inversión del periodo de entrenamiento largo con parámetros de perfil Medio.
Fuente: elaboración propia

Figura 17: Calendario de inversión del periodo de prueba largo con parámetros de perfil Medio.
Fuente: elaboración propia

En resumen, los peridos largos dan muchas posibilidades de compraventa que nuestro algoritmo no sabe aprovechar. La parte positiva es que la mayoría de las veces, cuando invierte, lo hace con beneficio. No obstante, en ocasiones, puede que no invierta y tengamos un modelo fútil, como es el caso de la ejecución de parámetros de perfil Medio-Corto.

7.1.2 Perido medio

En este segundo caso, se han elegido dos periodos distintos. En el primer periodo, claramente alcista, el entrenamiento comprende desde el 29 de septiembre de 2002 hasta el 29 de septiembre de 2003. Por otro lado, el periodo de prueba empieza el 29 de septiembre de 2003 y finaliza el mismo día de 2004.

El segundo periodo, de carácter bajista, esta formulado también de año en año, el entrenamiento se comienza el 1 de noviembre de 2009 y la prueba empieza el 1 de noviembre de 2010.

Una vez más, se han tomado fechas en las que los perfiles de los precios se asemejan.

		Largo	Largo-Medio	Medio	Medio-Corto	Corto	Buy&Hold
Entrenamiento alcista	**29/09/2002 – 29/09/2003**	121,02%	116,67%	110,60%	103,79%	93,37%	64,62%
Prueba alcista	**29/09/2003 –29/09/2004**	-4,54%	-9,47%	18,16%	-12,01%	10,04%	14,26%
Entrenamiento bajista	**01/11/2009 – 01/11/2010**	107.90%	82,87%	70,18%	63,92%	71.86%	-25,00%
Prueba bajista	**01/11/2010 – 01/11/2011**	1,34%	-10,10%	4,48%	-22,35%	-7,76%	-35,87%

Figura 18: Ejecuciones en un periodo medio alcista y en un periodo medio bajista. Las cantidades mostradas son el porcentaje de beneficio respecto al presupuesto inicial. Fuente: elaboración propia

En esta ocasión, los resultados son significativamente peores. Con motivo de ilustrar la adaptación de los modelos obtenidos en el periodo de entrenamiento, se han tomado periodos de prueba ligeramente peores, es decir, el precio tiende, significativamente, a bajar más que en el entrenamiento.

En todos los casos, el modelo obtenido con el algoritmo genético fue mejor que la estrategia *Buy&Hold* evaluados en el periodo de entrenamiento. Esto quiere decir que el algoritmo es capaz de encontrar condiciones de compra y venta que reportan beneficios en el periodo de aprendizaje. Sin embargo, estas condiciones no devuelven buenos resultados en periodos ligeramente distintos. De hecho, incluso en periodos alcistas, se pierde dinero cuando el modelo en el entrenamiento consigue sacar un 121.02 % de beneficios.

Otro punto que se puede destacar es la irregularidad del algoritmo. Habitualmente, otros algoritmos tienden a sobreaprender cuando se insiste demasiado en los datos de entrenamiento. Así pues, es habitual encontrar un punto de entrenamiento a partir del cual los resultados obtenidos en las pruebas empeoran.

En este algoritmo propuesto, los resultados obtenidos para el periodo de prueba parecen variar de forma independiente al periodo anterior. Tras múltiples ejecuciones con los mismos parámetros se puede observar como, en ocasiones, los resultados varían de forma significativa. Este hecho puede ser la causa de una alta aleatoriedad en algoritmo debida a, posiblemente, la gran cantidad de mutaciones necesarias y la aleatoriedad de la primera generación.

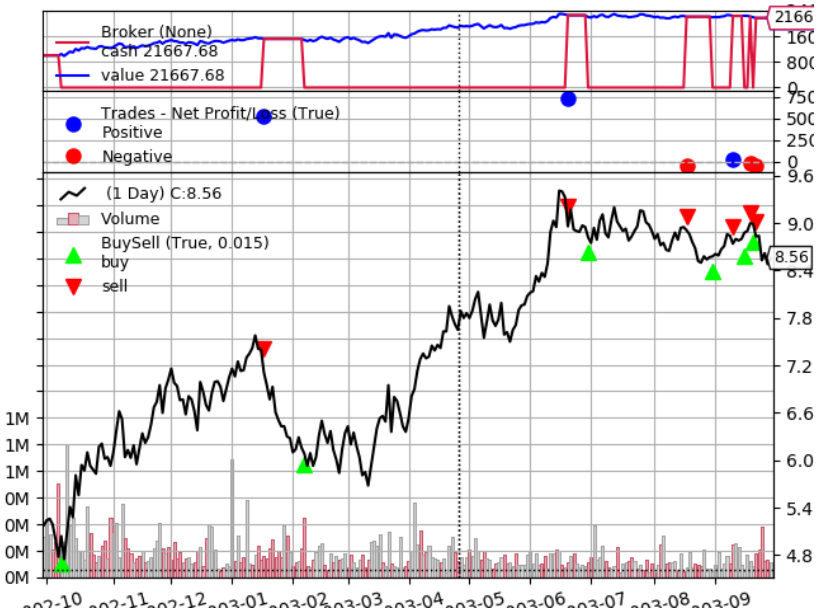

Figura 19: Calendario de inversión del periodo de entrenamiento medio con parámetros de perfil Medio-Corto.
Fuente: elaboración propia

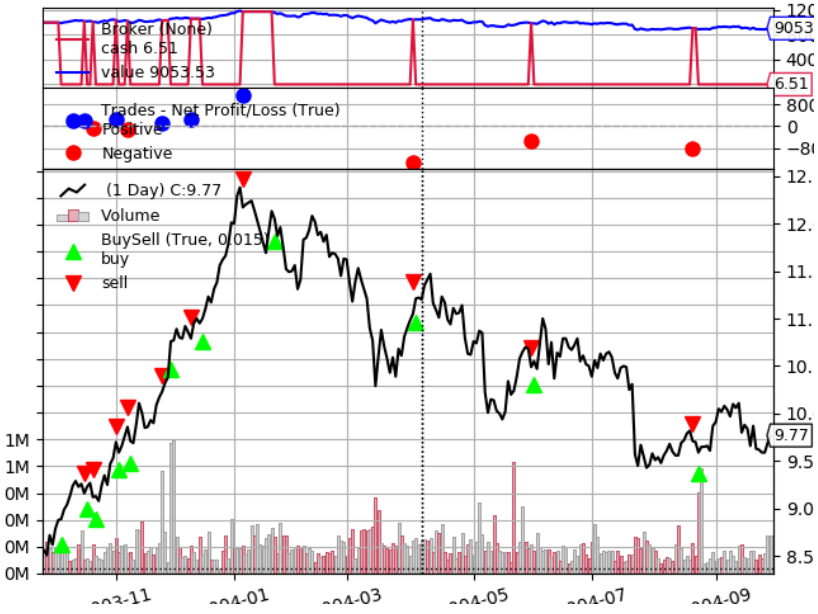

Figura 20: Calendario de inversión del periodo de prueba medio con parámetros de perfil Medio-Corto.
Fuente: elaboración propia

En las figuras 19 y 20 puede observarse bastante bien esta poca adaptación. En el periodo de entrenamiento se hacen varias inversiones de unas semanas, e incluso de varios meses, entre las cuales se deja un margen de tiempo hasta que es un buen momento de compra. Por el contrario, en el periodo de prueba apenas si hay unas semanas que el modelo no tiene nada comprado, es decir, siempre que vende intenta comprar inmediatamente.

Se considera, en consecuencia a este hecho, la hipótesis de que el algoritmo tiene un buen aprendizaje pero una mala adaptación a otros periodos.

7.1.3 Periodo corto

Para la última de las ejecuciones de parámetros, se han escogido también dos periodos pero, esta vez, se ha tomado un periodo alcista y un periodo lateral. Este segundo es algo complicado, pues el periodo de entrenamiento es lateral pero el de prueba tiene tendencia bajista.

En primer lugar, el periodo alcista tiene su entrenamiento desde el 1 de julio de 2016 y hasta el 22 de octubre de 2016. Por otro lado, el periodo de preuba empieza a partir de entonces y hasta el 1 de febrero de 2017.

En segundo lugar, el periodo lateral comprende un poco más de tres meses. El periodo de entrenamiento empieza el 1 de enero de 2018 y, por su parte, el periodo de prueba comienza el 22 de abril de 2018. La simulación termina, finalmente, el 22 de julio de 2018. Como ya se ha adelantado, no se esperan buenos resultados. Los modelos generados tiene poca capacidad de adaptación y, como hemos tomado periodos de entrenamiento y prueba con perfiles muy distintos, la adaptación debe ser bastante peor.

		Largo	Largo-Medio	Medio	Medio-Corto	Corto	Buy&Hold
Entrenamiento alcista	**01/07/2016 – 22/10/2016**	32,46%	34,08%	36,32%	30,45%	28,71%	19,74%
Prueba alcista	**22/10/2016 –01/02/2017**	13,09%	4,51%	14,28%	2,79%	10,78%	16,53%
Entrenamiento bajista	**01/01/2018 – 22/04/2018**	23,73%	22,48%	22,48%	19,80%	20.37%	3,04%
Prueba bajista	**22/04/2018 – 10/08/2018**	-13,01%	-29,34%	-25,67%	-20,96%	-25,78%	-8,70%

Figura 21: Ejecuciones en un periodo corto alcista y en un periodo corto lateral. Las cantidades mostradas son el porcentaje de beneficio respecto al presupuesto inicial. Fuente: elaboración propia

Como se observa en la figura 21, en los periodos cortos no se obtienen buenos resultados. De hecho, en el caso del periodo lateral, y prueba bajista, los resultados son pésimos. Mientras que, en el entrenamiento, la estrategia

propia supera con creces a la estrategia *Buy&Hold*, en la prueba tiene pérdidas estrepitosas al ser un periodo bajista. Esto confirma la hipótesis de la baja adaptación del algoritmo.

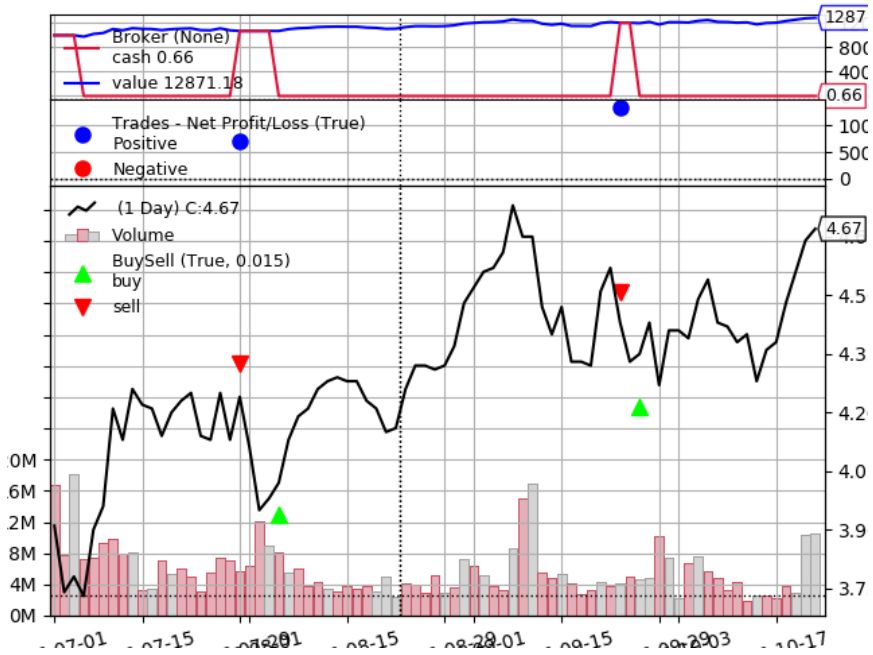

Figura 22: Calendario de inversión del periodo de entrenamiento corto alcista con perfil de parámetros Corto.
Fuente: elaboración propia

En la figura 22, correspondiente al periodo alcista de entrenamiento, se encontra un modelo generado con perfil de parámetros corto que invierte de una forma muy eficiente, esto es, compra en los mínimos y vende en los máximos.

Teniendo esta regla como referencia, se analiza el mismo modelo ejecutado, esta vez, en el periodo de prueba en la figura 23. Este periodo tiene un perfil de precio distinto, los mínimos y máximos no son tan destacables. No obstante, el algoritmo ha generado un modelo que, tal y como se intuye, esta intentando hacer algo similar a lo realizado en el periodo del que aprendió. Las primeras transacciones son torpes y demasiado abundantes, produciendo pérdidas con las comisiones. Por tanto, a pesar de ser un periodo alcista, no tiene tanto éxito como la estrategia de *Buy&Hold*.

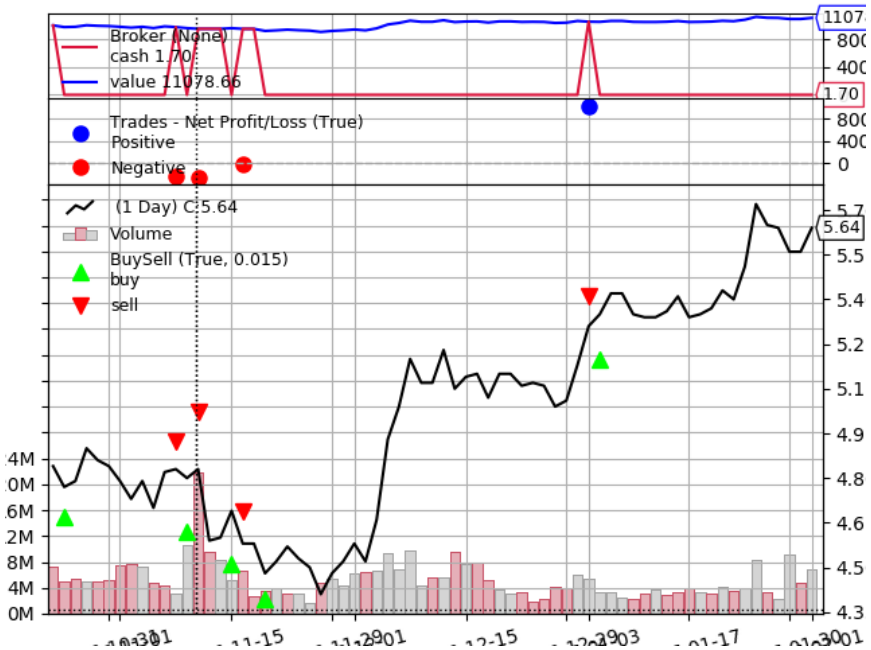

Figura 23: Calendario de inversión del periodo de prueba corto alcista con perfil de parámetros Corto.
Fuente: elaboración propia.

Por último, se propone hacer una análisis del periodo entrenado con perfil lateral pero probado con un perfil bajista. Las figuras 22 y 24 contiene los calendarios de inversión de estas simulaciones realizadas con configuración de parámetros Largo.

En el entrenamiento se tiene una inversión muy ajustada, prácticamente inmejorable. No obstante, esto podría ser fruto de un sobreaprendizaje. Por su parte, en el periodo de prueba, se observan los mismos patrones de compra y venta. Pero al ser las oscilaciones del precio más pequeñas que en el caso anterior, las comisiones se llevan esos escasos beneficios que pudiera tener.

En contraposición, encontramos un punto a favor, la precisión para encontrar los mínimos. Los indicadores parecen ser unas buenas herramientas para detectar estos. Nótese que, tras una acentuada caída durante el primer mes, se compra en el último momento antes de la subida. Por tanto, aunque los resultados no son para nada correctos, las señales de compra y venta no son demasiado erróneas.

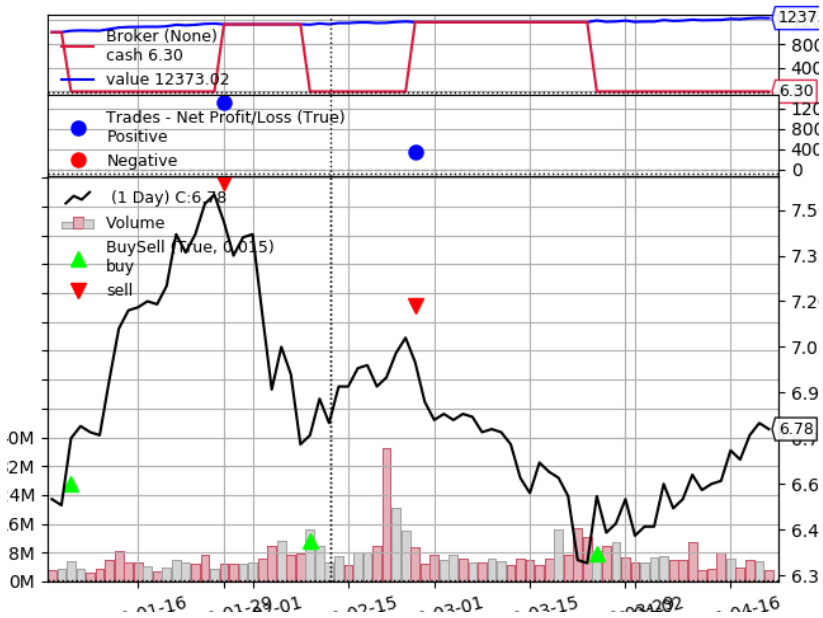

Figura 24: Calendario de inversión del periodo de entrenamiento corto lateral con perfil de parámetros Largo.
Fuente: elaboración propia.

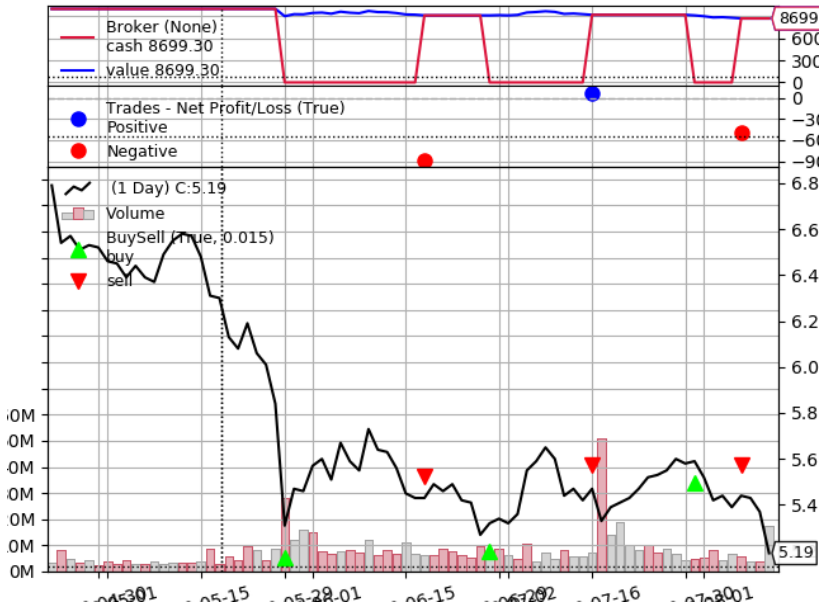

Figura 25: Calendario de inversión del periodo de prueba corto bajista con perfil de parámetros Largo.
Fuente: elaboración propia.

7.2 Contraste con otras estrategias

8 Conclusiones y trabajos futuros

Bibliografía

Barros, R., de Carvalho, A., & Freitas, A. (2015). *Automatic design of decision-tree induction algorithms* (primera edición). Springer International Publishing.

Beattie, A. (2017). *What was the first company to issue stock?* https://www.investopedia.com/ask/answers/08/first-company-issue-stock-dutch-east-india.asp. Investopedia.

Bonde, G. & Khaled, R. (2012). Stock price prediction using genetic algorithms and evolution strategies. *In Proceedings of the International Conference on Genetic and Evolutionary Methods (GEM) (p. 1)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Elliott, R. & Prechter, R. (1994). *R.n. elliott's masterworks: the definitive collection*. New Classics Library.

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction* (segunda edición). Chichester: John Wiley & Sons Ltd.

Esfahanipour, A. & Mousavi, S. (2011). A genetic programming model to generate risk-adjusted technical trading rules in stock markets. *Expert Systems with Applications*, *38*(7), 7911-9052.

Gartley, H. M. (1935). *Profits in the stock market*. Health Research Books.

Huang, C.-J., Yang, D.-X., & Chuang, Y.-T. (2008). Application of wrapper approach and composite classifier to the stock trend prediction. *Expert Systems with Applications*, *34*(4), 2870-2878.

Kampouridis, M. & Otero, F. E. (2017). Evolving trading strategies using directional changes. *Expert Systems with Applications*, *73*, 145-160.

Koza, J. R. & Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*. MIT press.

Luke, S. (2013). *Essentials of metaheuristics* (segunda edición). Disponible en https://cs.gmu.edu/ sean/book/metaheuristics/Essentials.pdf. Lulu.

Mousavi, S., Esfahanipour, A., & Zarandi, M. H. F. (2014). A novel approach to dynamic portfolio trading system using multitree genetic programming. *Knowledge-Based Systems*, *66*, 68-81.

Myszkowski, P. B. & Bicz, A. (2010). Evolutionary algorithm in forex trade strategy generation. *In Proceedings of the International Multiconference on Computer Science and Information Technology. p. 81-86.* IEEE.

Nair, B. B., Mohandas, V., & Sakthivel, N. (2010). A genetic algorithm optimized decision tree-svm based stock market trend prediction system. *International journal on computer science and engineering*, *2*(9), 2981-2988.

Potvin, J.-Y., Soriano, P., & Vallée, M. (2004). Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, *31*(7), 1033-1047.

Rouwhorst, S. E. & Engelbrecht, A. (2000). Searching the forest: using decision trees as building blocks for evolutionary search in classification databases. *1*, 633-638. Proceedings of the 2000 Congress on Evolutionary Computation.

Sheta, A., Faris, H., & Alkasassbeh, M. (2013). A genetic programming model for s&p 500 stock market prediction. *International Journal of Control And Automation*, *6*(5), 303-314.

Zhou, C., Yu, L., Huang, T., Wang, S., & Lai, K. K. (2006). Selecting valuable stock using genetic algorithm, 688-694.