



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

25 de julio de 2019

Agradecimientos

A pesar del tema de trabajo,
me gustaría dar mi agradecimiento a la única
acción que no necesita ninguna predicción,
el apoyo de los que siempre estuvieron ahí.

Resumen

Aquí va el resumen del tfg en español y las palabras clave.

Abstract

Here lies the tfg's abstract in english and the key words.

Índice

1. (Introducción) Predicción en el mercado de valores	6
1.1. Precedentes en la predicción de bolsa.	7
1.2. Tipos de información extraíble a partir de árboles	8
1.2.1. Información total	8
1.2.2. Información de tendencias	9
1.2.3. Información de señales de compra y venta	10
1.3. Objetivo del proyecto	10
2. Fundamentos de Bolsa	12
2.1. Mercado de valores	12
2.2. El valor de las acciones	12
2.3. Corredores de bolsa o brokers	14
2.4. Herramientas de simulación	14
2.4.1. MetaTrader5	14
2.4.2. Backtrader	15
2.4.2.1. Instalación	15
2.4.2.2. Preparación del entorno	16
2.4.2.3. La primera estrategia	18
2.4.3. Otras plataformas	20
2.5. Conseguir datos históricos para realizar backtesting	20
2.5.1. Quandl	20
2.5.2. YAHOO! Finance	21
2.5.3. Pandas Datareader	23
3. Algoritmos genéticos	24
3.1. Definición	24
3.1.1. Población	25
3.1.2. Función fitness	25
3.1.3. Selección	26
3.1.4. Cruce	27
3.1.5. Mutación	28
3.2. Ejemplo de algoritmo genético con Pyvolution	29
4. Árbol de Decisión	31
4.1. Definición	31
4.1.1. Etiquetado	32
4.1.2. Conjunto de entrenamiento	32

5. Algoritmo propuesto	34
5.1. Árboles de decisión	34
5.2. Estructura genética	36
5.2.1. Primera generación	36
5.2.1.1. Etiquetado de datos de prueba	36
5.2.1.2. Selección de pivotes	38
5.2.1.3. Selección de las hojas	39
5.2.2. Cruce	40
5.2.2.1. Probabilidades de reproducción	40
5.2.2.2. Cruce natural entre dos árboles	41
5.2.3. Mutaciones	43
5.2.3.1. Mutaciones en indicadores	44
5.2.3.2. Mutaciones en los pivotes	44
5.2.3.3. Mutaciones en parámetros de indicadores	44
5.2.3.4. Mutaciones en las hojas	45
6. Detalles de la implementación	46
6.1. Diagrama de clases	47
6.2. Optimizaciones	48
6.2.1. Paralelización	48
6.2.2. Caché para indicadores de primer orden	49
6.2.3. Caché para indicadores de segundo orden	50
7. Análisis de resultados	51
8. Conclusiones y trabajos futuros	52
Bibliografía	53

Índice de figuras

1.	Árbol de programación genética. Fuente: elaboración propia.	8
2.	Árbol de señal de compra. Fuente: Myszkowski (2010)	10
3.	Situación de las órdenes. Fuente: elaboración propia.	13
4.	Resultado de instalar backtrader con plotting. Fuente: elaboración propia.	16
5.	Ejecución del script recolectando datos de Quandl. Fuente: elaboración propia.	21
6.	Descarga de los datos históricos de Banco Santander. Fuente: elaboración propia.	22
7.	Árbol de decisión. Fuente: https://sefiks.com [Última consulta 12 de Julio de 2019]	32
8.	Valor de las acciones de Amazon. Fuente: https://finance.yahoo.com [Última consulta 19 de Julio de 2019]	35
9.	Ejemplo de etiquetado de un periodo. Fuente: elaboración propia	38
10.	Ejemplo de cruce entre dos árboles. Fuente: elaboración propia	42
11.	Diagrama de clases. Fuente: elaboración propia	47

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 ganancia 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 venta aparece en casi todos los sectores económicos. Por ejemplo:

- Distribución de mercancías. Hay empresas que se dedican a comprar mercancías 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 provocando una pérdida de dinero.

Por tanto, disponer de información sobre el mercado, sitúa a empresas y particulares en posiciones ventajosas. Es por esto que, siguiendo distintas estrategias, muchos intentan extraer información sobre la situación del mercado.

A lo largo de este proyecto, vamos a intentar extraer nuestra propia 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).

Lleva utilizándose 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, algunos valores.

- **Análisis de índices o indicadores.** Los índices son estadísticos que representan una determinada característica del estado de la bolsa de uno o varios valores distintos. Una de las publicaciones con más renombre es la teoría de Dow¹, cuyo autor, Charles H. Dow, fue el fundador del Wall Street Journal. Diario en el que publicaba sus principios para invertir en bolsa.

Algunos indicadores representan la situación general de la bolsa de un país, por ejemplo el IBEX35 (35 empresas con más capital de España). Otros indicadores representan la situación de las empresas que se dedican a un sector particular, como el SX7E (bancos de los países europeos). También hay indicadores que aportan información estadística sobre solo un valor, por ejemplo la EMA (Media Móvil Exponencial).

En este proyecto vamos a centrarnos en el análisis de indicadores de un solo valor. En concreto, como se verá a lo largo del trabajo, vamos a proponer un modelo basado en árboles de decisión formados con un algoritmo genético.

¹Hamilton, 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..

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. Notar que existen otras formas y herramientas de extracción información o predeción, pero por la naturaleza de nuestra propuesta nos centraremos en las estrategias basadas en árboles.

1.2.1. Información total

El más ambicioso de los objetivos propuestos 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 $f : \Omega \rightarrow \mathbb{R}$ 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 y, 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.

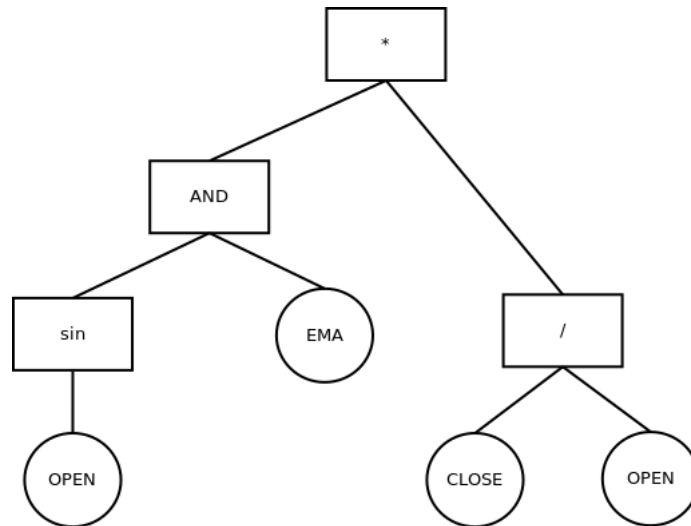


Figura 1: Árbol de programación genética. Fuente: elaboración propia.

En la figura 1 puede verse un árbol de este tipo. De este se puede extraer la función

$f(\text{OPEN}, \text{CLOSE}, \text{EMA}) = (\sin(\text{OPEN}) \text{ AND EMA}) * (\text{CLOSE}/\text{OPEN})$
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). También encontramos otra variante basada en el *Conditional Sharpe Ratio* en Esfahanipour (2011) y Mousavi (2014).

Por último, una función *fitness* algo diferente sería 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, obtendríamos 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 saber si el valor subirá o bajará.

Puede ser visto como una relajación del 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 encontramos un árbol de decisión con tres etiquetas. No vamos a entrar en detalle aquí en esta herramienta, ya que será explicada mejor en el apartado 4.

Esta información es bastante más sencilla de interpretar. Pero, por contra, es bastante más imprecisa. El modelo nos dice la tendencia que tendrá el valor, no obstante, 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 anteriores tenemos los modelos de señales. No estamos interesados en saber el valor de la acción, sino que buscamos 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. Notar que se puede hacer una extensión del modelo para que no sea solo binario, 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.

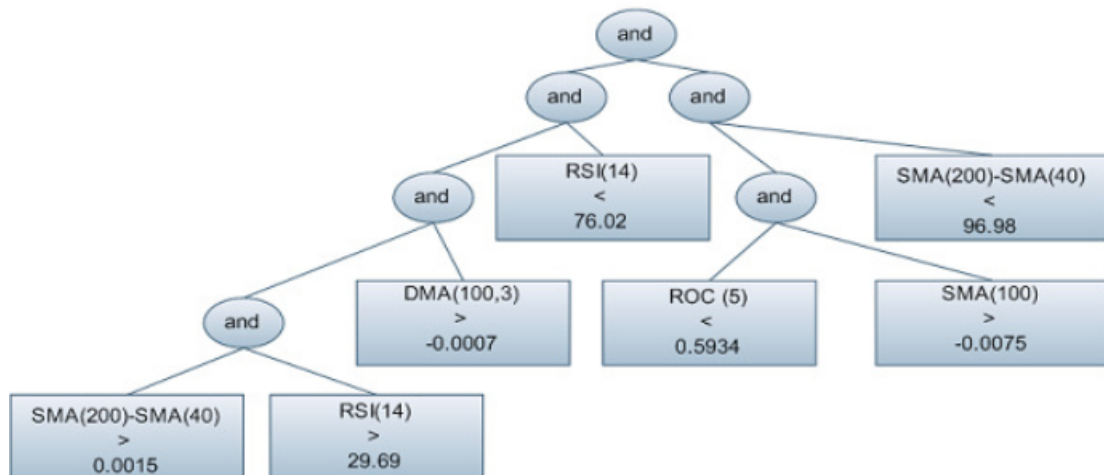


Figura 2: Árbol de señal de compra. Fuente: Myszkowski (2010)

En la figura 2 se muestra un árbol de señal de compra. Notar 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 clara. Te permite invertir en bolsa con mayor posibilidad de sacar beneficios que si se invirtiera por mera intuición.

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 tanto, aunque el modelo predicho sea muy bueno, siempre habrá un punto de incertidumbre que no sabremos analizar.

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 conseguir:

- 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 corresponde con una pequeña participación en la empresa a la que pertenezca la acción. De esta forma, inversores privados tiene 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. Esto puede reportarnos beneficios de distinta índole según las normas de la empresa. En ocasiones algunas empresas deciden repartir una determinada cantidad de dinero entre sus accionistas en relación a las ganancias obtenidas en un periodo (dividendo). Otras veces, ser poseedor de una gran parte de la empresa nos otorgará derechos dentro de la misma, como por ejemplo participar en las decisiones importantes que se tomen.

El mercado de valores español abre de 9:00 a 17:30 de lunes a viernes. Cabe destacar que hay una aleatoriedad de 30 segundos y que, además, después del cierre hay un periodo de subasta de 5 minutos para evitar manipulaciones de los precios. No vamos a entrar en como funciona este sistema.

2.2. El valor de las acciones

El valor de una acción no es fijo, va cambiando 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 será ejecutada tan pronto como sea posible. Aclaremos esto 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, llamémosles A, B, C y D.

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 está situado en 4 euros por acción. Ahora bien, 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 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.

Tras ver que el precio de la acción se bajado de 4 a 3 euros, el inversor D decide comprar y sitúa una orden de compra a 3 euros de una acciones. Entonces, la orden de venta de C y la orden de compra de D pueden ejecutarse. Ahora D pasa a tener 2 acciones y el mercado se queda con las dos órdenes de A y B, provocando que el precio de la acción vuelva a subir a 4 euros.

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.** Se compran todas las acciones que se emitan en la orden al mejor precio posible. En caso de compra a los precios más bajos, en caso de venta a los precios más altos. Si B hubiera emitido una orden de mercado de una acción, habría comprado la acción de A a 4 euros.
- **Orden limitada.** Se compran todas las acciones que se emitan en la orden siempre y cuando su precio esté situado en el límite establecido. Si D hubiera emitido una orden limitada a 3 euros de dos acciones, hubiera comprado por 3 euros la acción de C pero no la de A, que está a 4 euros. La orden permanece vigente hasta que la suma de las acciones sea comprada.
- **Orden on Stop.** No se hace efectiva hasta que el precio de las acciones llega al precio de la orden. En ese momento, la orden se convierte en

una orden de mercado.

- **Orden Stop-Limit.** Este tipo de orden no se hace efectiva hasta que el precio de las acciones llega 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 y el inversor posee acciones de la empresa.

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, colocar órdenes y presentar a compradores con vendedores. Para ser broker se debe aprobar un examen y demostrar que se tienen los conocimientos necesarios. Por 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, 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. Las comisiones variables afectan en 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.

No corresponde a este trabajo discutir más allá sobre este tema. De ahora en adelante asumiremos unas comisiones variables.

2.4. Herramientas de simulación

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 de forma simulada, es

decir, sin participar con dinero real. Podemos observar que es una plataforma ampliamente usada, por tanto, es fácil encontrar ejemplos iniciales o ayuda técnica. En la página web de la plataforma² se nos ofrece una demo gratuita con acceso a casi todos los recursos.

La programación de las estrategias se divide en varios archivos con unas especificaciones concretas. No obstante, 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 la han usado.

A pesar de todas estas ventajas, no usaremos MetaTrader5 por ser de carácter privativo. Además, 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 realizado en el lenguaje Python. La página web de la plataforma³ contiene una amplia documentación en la que se explican todos los componentes del framework. Asimismo encontramos el método de instalación y un tutorial de iniciación.

El framework viene con indicadores ya programados, aunque te permite hacer más de forma manual. También se pueden producir gráficos temporales, pero hay que instalar una librería adicional. Esto no es problema porque viene integrada en la instalación como veremos a continuación.

2.4.2.1 Instalación

Pasamos ahora a su instalación. Necesitamos tener la versión 2.7 de Python, tal y como se indica en la documentación de backtrader⁴ que podemos encontrar en su página web. Para comprobar la versión podemos usar el comando *python -Version*. En caso de no tener instalado Python o no tener la versión indicada, podemos instalarlo usando *sudo apt-get install python2.7*.

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 está instalado pip, podemos hacerlo con el comando

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

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]

`sudo apt-get install python-pip.`

Una vez llegados a este punto, podemos optar por una versión con posibilidad de generar gráficas o no. Para facilitar el análisis de resultados, instalaremos la versión con plotting con el comando `pip install backtrader[plotting]` como podemos ver en la figura 4. El mismo comando quitando la directiva `[plotting]` instala la versión sin posibilidad de hacer gráficas.

```
miguel@miguelpc:~$ pip install backtrader[plotting]
Collecting backtrader[plotting]
  Downloading https://files.pythonhosted.org/packages/.../backtrader-2.4.0-py3-none-any.whl (419kB)
  100% |#####| 419kB 2.4MB/s
Collecting matplotlib; extra == "plotting" (from backtrader[plotting])
  Downloading https://files.pythonhosted.org/packages/.../matplotlib-3.3.0-py3-none-any.whl (12.6MB)
  100% |#####| 12.6MB 14.1MB/s
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../pyparsing-2.4.7-py2.py3-none-any.whl (61kB)
  100% |#####| 61kB 6.6MB/s
Collecting backports.functools-lru-cache (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../backports.functools-lru-cache-1.0.2-py2.py3-none-any.whl (102kB)
  100% |#####| 102kB 8.1MB/s
Collecting pytz (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../pytz-2019.3-py2.py3-none-any.whl (512kB)
  100% |#####| 512kB 2.9MB/s
Collecting six>=1.10 (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../six-1.12.0-py2.py3-none-any.whl (215kB)
  100% |#####| 215kB 4.4MB/s
Collecting python-dateutil>=2.1 (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../python-dateutil-2.6.1-py2.py3-none-any.whl (215kB)
  100% |#####| 215kB 4.4MB/s
Collecting kiwisolver>=1.0.1 (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../kiwisolver-1.0.1-py2.py3-none-any.whl (952kB)
  100% |#####| 952kB 1.7MB/s
Collecting cyclr>=0.10 (from matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../cyclr-0.10-py2.py3-none-any.whl (12.1MB)
  100% |#####| 12.1MB 10.1MB/s
Collecting setuptools (from kiwisolver>=1.0.1->matplotlib; extra == "plotting")
  Downloading https://files.pythonhosted.org/packages/.../setuptools-44.1.1-py2.py3-none-any.whl (573kB)
  100% |#####| 573kB 2.6MB/s
```

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

Observamos que se han instalado otros paquetes adicionales. En concreto se ha instalado el paquete numpy, que más tarde nos será útil como herramienta matemática.

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 pares ordenados por uno de los elementos de menor a mayor. Por ejemplo, al hablar de un valor de mercado tenemos 5 líneas: valor de apertura, valor de salida, valor máximo, valor mínimo y volumen de compra/venta.

Al acceder a cada línea hay que tener en cuenta que no usa la indexación de un vector en un lenguaje común. En su lugar, el índice 0 representa el instante actual y para acceder a instantes anteriores usaremos índices negativos, por ejemplo:

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

Antes de ver como crear una estrategia, vamos a ver como podemos simular una prueba introduciendo unos datos y un presupuesto inicial. Para ello, en un archivo con formato py para que podamos ejecutar con Python escribimos 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())
```

Notemos 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 tenemos que incorporarlos de forma habitual.

En este ejemplo, no le hemos especificado a la clase *cerebro* cuál va a ser la estrategia a seguir. Por este motivo, al hacer la simulación el presupuesto inicial y final no varían. Para ejecutar el backtesting simplemente se ejecuta el script de Python con el comando *python rutadelscript.py* en terminal. Nos dará un fallo, la ruta donde debería estar el fichero con los datos está vacía, pues no hemos descargado ningún archivo de datos. 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 estrategia fútil cómo estructurar una clase para que backtrader pueda trabajar con ella como tal. 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 requiere nuestra estrategia. El segundo método es llamado en cada instante por backtrader. Consideramos que un instante es cada una de las marcas temporales que hay registradas en nuestros datos. Así pues, los datos recogidos en `__init__` son la referencia para llamar al método `next`. 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.

Veamos un ejemplo:

```
# 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 línea 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 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)
```

La estrategia está completa, pero no realizamos ninguna compra ni venta. 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 una nueva 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 órdenes 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 introducimos varios nuevos conceptos 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, backtrader compra o vende al precio de cierre del instante actual una acción. Esto no quiere decir que la orden se ejecute en ese instante, si no que, a partir de ese momento y si el precio del producto lo permite, se realizará. Notar que una vez lanzada una orden hay que esperar a que se complete o se cancele antes de lanzar otra.

Esta situación hace necesario incluir el método *notify_order*. Como su propio nombre sugiere, es llamado cuando una orden cambia de estado. En este ejemplo, cuando una orden es completada, mostramos por pantalla si es era de compra o venta y el precio de la misma. Las ordenes pueden ser canceladas o rechazadas. Un ejemplo de este hecho sería intentar comprar una cantidad de acciones con un presupuesto insuficiente para las mismas.

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

- **Existe una compraventa lanzada y no terminada.** En este caso debemos esperar. Notar que backtrader permite cancelar una orden, aunque no entraremos en este punto.
- **No hay una orden lanzada y tampoco tengo nada comprado.** Esto puede comprobarse con *self.position*, que devuelve negativo si no

se tiene nada en posesión. En esta situación solo cabe esperar una orden de compra y para este ejemplo la lanzaremos si los últimos dos instantes han bajado su precio de cierre.

- **No hay una orden lanzada pero tengo algo comprado.** Para ilustrar una nueva herramienta, vamos a lanzar una orden de venta 5 instantes después de haber comprado. Para ello podemos comprobar la longitud de *self* con la función *len* de Python.

2.4.3. Otras plataformas

- **Tradestation** permite programar estrategias, pero no hacer *backtesting*. Además no es de acceso gratuito.
- **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 órdenes sencillas condicionales preprogramadas.
- **PyAlgoTrade** es un proyecto desarrollado en Python disponible en Github. Tiene posibilidad de *backtesting*, utilizar paquetes externos y, en el caso de estar haciendo *trading* con Bitcoins, 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 como conseguir los valores históricos en bolsa de diferentes empresas de forma automática. A parte de conseguir los datos, vamos a ver como se incorporan a backtrader para su posterior uso.

2.5.1. Quandl

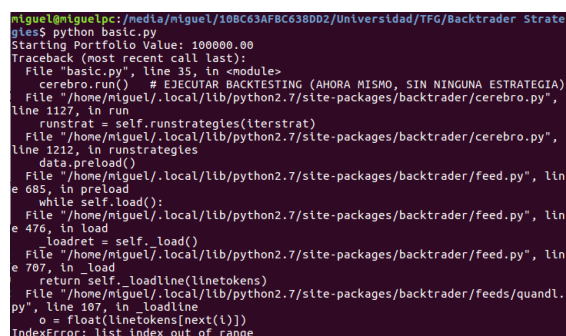
Quandl es una empresa que reúne miles de paquetes de datos financieros de todo el mundo. Para acceder a esta información es necesario registrarse

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 es utilizar la api que ofrece. De esta forma podemos realizar 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 documentación⁵ de Quandl. Instalamos el paquete quandl con el gestor de paquetes pip. Para usar el paquete quandl, debemos indicar a Quandl la *api_key* que nos dan al inscribirnos en su página.

```
# 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. Probablemente, como en este foro de la comunidad se apunta⁶, se debe a una mala lectura de los vectores de datos recibidos.



```
miguel@miguelpc:~/ndia/miguel/108C63AF8C638DD2/Univeridad/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(literstrat)
  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 187, 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.

2.5.2. YAHOO! Finance

Una de las opciones más completas para conseguir los datos es la página web de YAHOO! Finance (<https://finance.yahoo.com>). Basta con buscar el índice o la empresa sobre la que se quieren los datos, indicar las fechas y

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]

la frecuencia de muestreo y pinchar en el botón de descarga. Los datos se descargan en formato CSV. Este formato es ampliamente utilizado por la comunidad científica para almacenar grandes cantidades de datos.

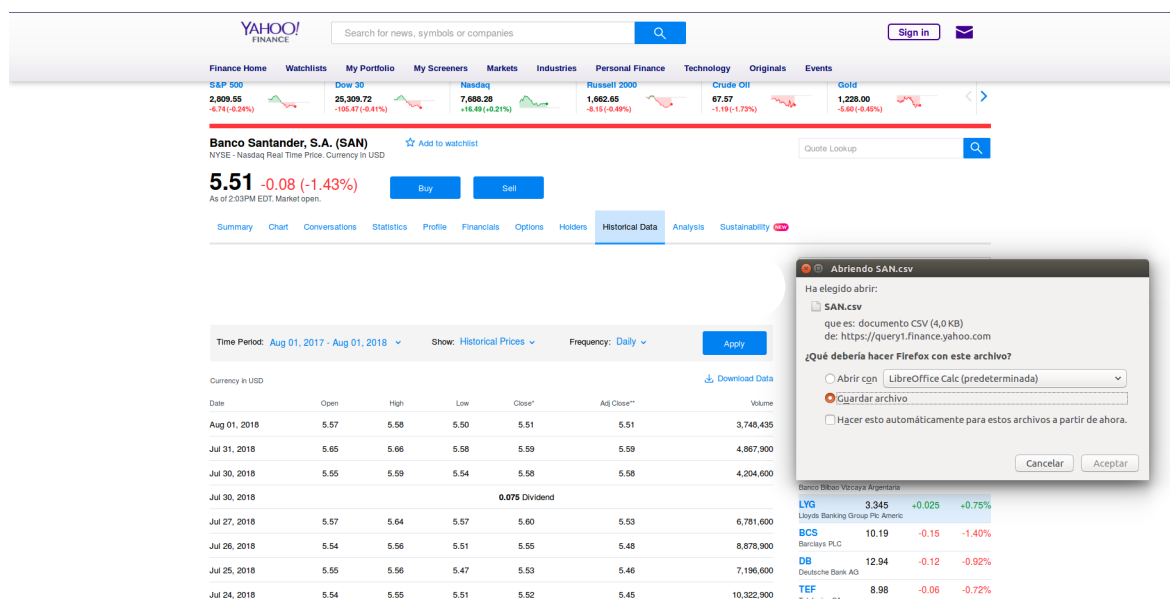


Figura 6: Descarga de los datos históricos de Banco Santander. Fuente: elaboración propia.

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, nos vamos al apartado de *Historical Data* e indicamos las fechas. El resultado se muestra en la figura 6.

Para cargar estos datos en backtrader basta con indicar la ruta y el rango de fechas con el siguiente código:

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


2.5.3. Pandas Datareader

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

Aunque estuvo un tiempo en desuso y en su documentación⁷ está marcado así, ahora vuelve a estar operativo. La API de YAHOO! Finance cambió y, durante un tiempo, *Pandas-datareader* no la actualizó.

Por comodidad, 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 final para descargar los datos diarios la misma.

⁷<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. 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 la solución hacia una mejora.

A continuación se ofrece una aproximación a los algoritmos genéticos. Un desarrollo más profundo puede encontrarse en otras obras⁸.

3.1. Definición

Un algoritmo genético se compone de cinco elementos básicos:

- **Población.** Un conjunto de soluciones del problema que queremos resolver. Su representación computacional definirá al resto de los elementos.
- **Función fitness o función de evaluación.** Una función cuyo dominio sea la población del problema y que tenga por imagen algún subconjunto de \mathbb{R} . Representa como de buena es una solución.
- **Cruce.** Un proceso mediante el cual una o varias soluciones produzcan una o varias soluciones combinadas nuevas.
- **Mutación** Un proceso que actúe sobre un individuo de la población y le produce una transformación significativa. Este procedimiento no es imprescindible, pero ayuda a que las soluciones sigan variando tras muchas generaciones.
- **Selección.** Un criterio de eliminación que permite descartar individuos de la población.

⁸Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons.

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, se pueden representar por los cromosomas de dicho individuo. En un algoritmo genético se usa tradicionalmente 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 es de la forma

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

Mediante una representación binaria, por ejemplo la de coma flotante, de profundidad l podemos transformar cada $x = (x_1, \dots, x_i, \dots, x_n) \in \mathbb{R}$ en $b = (b_1, \dots, b_i, \dots, b_n)$ donde $\forall 0 < i \leq n \ b_i \in \{0, 1\}^l$. Esto nos permite trabajar con el caso continuo igual que con el caso discreto.

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

Una vez que tenemos representado a 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, pagaremos esta mejor búsqueda con un incremento del tiempo de ejecución. Así mismo, la calidad de la población en la primera generación determinará, en gran medida, la calidad de las generaciones posteriores.

3.1.2. Función fitness

En un modelo evolutivo, el individuo con mejores características tiene mayores posibilidades de sobrevivir. Con motivo de valorar como de bueno es un individuo de la población necesitamos una función matemática que

nos cuantifique la bondad de este. Definimos la función *fitness* o función de evaluación como

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

donde Γ es el espacio donde 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.

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, para filtrar los individuos más fuertes (hablando en términos de la función *fitness*).
- En el cruce de los individuos para 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.

Selección proporcional

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

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

Selección por torneo

Se extrae una muestra (con o sin devolución) de la población de tamaño k con $k < s$. Se selecciona entonces el individuo con mayor puntuación en la

función *fitness*.

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.

Se selecciona el elemento x_i donde $i = \text{Entera}(Z)$ 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. Esto asegurará que la nueva población tendrá algún individuo al menos tan bueno como en 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 generan la siguiente población. Un cruce es una operación que, dados uno o varios individuos padres, genera uno o varios individuos hijos que pasarán a la próxima generación.

La forma técnica de hacer un cruce depende directamente de la representación de los individuos. Vamos a presentar algunas formas de hacer cruces cuando tenemos representación binaria pura.

Cruce de un punto

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

Se generan entonces dos individuos. El primero tendrá los i primeros bits del primer padre y los $l - i$ últimos del segundo progenitor. El segundo individuo se genera con los bits que no se han tomado en el primero.

Cruce de dos puntos

Similar al cruce de un punto. La diferencia con este es que la parte que se toma del primer padre para el primer hijo viene dada por una subcadena de los bits del padre de inicio y final dos índices generados con una distribución uniforme, como venimos haciendo.

Notar 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 bit a bit. Se inicializa una máscara de tamaño l 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. 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 común o demasiado agresiva producirá que se pierda el factor evolutivo, no obstante, en determinadas dosis puede llegar a ser efectivo para explorar más en el espacio de búsqueda.

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

Mutación uniforme

Dado un individuo, se seleccionan, a partir de una distribución uniforme, los bits a mutar. La mutación consiste en poner el valor contrario al original (0 si era 1 y 1 si era 0).

Mutación *inorder*

Similar a la anterior, pero los bits mutables son solo un subconjunto de los totales. Dicho subconjunto inmutable puede ser producto de una aleatorización o seleccionado manualmente si se quiere proteger una zona 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.

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

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.

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 líneas y sin necesidad de dar muchas especificaciones se pueden realizar algoritmos completos. Una ejecución normal consta de varios parámetros donde 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, muestro uno de los ejemplos que vienen en la página de Github del paquete⁹.

```
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
)

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

⁹<https://github.com/littlel/pyvolution> [Última consulta 10 de Julio de 2019]

```

#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 parametros 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 sería práctico usar este paquete de Python para nuestro propósito. En su lugar, realizaremos una estructura de algoritmo genético propio.

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), representación de 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 simple es un grafo sin lazos ni aristas múltiples entre sus vértices. Un árbol es un grafo simple G tal que para cada dos vértices de G existe un único camino simple entre ellos.

Un árbol de decisión será, por tanto, un árbol compuesto por:

- Un vértice especial, llamado nodo raíz, que se considera el inicio del camino de decisión.
- Un conjunto de vértices, llamados nodos, conectados con al menos otros 2 vértices. En cada nodo hay una condición. La verificación de dicha condición determina que nodo es el siguiente en el camino de decisión.
- Un conjunto de vértices, llamados hojas o nodos terminales, conectados con un único vértice. En cada hoja hay una clasificación del dato que cumple todas las condiciones que tienen los nodos del camino desde la raíz hasta 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.

Planteamos un problema para dar un ejemplo de árbol de decisión. En un campeonato de tenis al aire libre son necesarias unas buenas condiciones meteorológicas. Para tomar la decisión de si se puede jugar un partido o no podemos plantear el árbol de decisión de la figura 7.

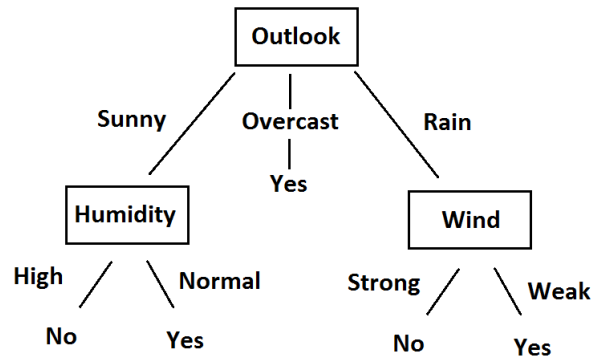


Figura 7: Árbol de decisión. Fuente: <https://sefiks.com> [Última consulta 12 de Julio de 2019]

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.

4.1.1. Etiquetado

Para poder clasificar un dato, es necesario saber cuáles son las posibles clasificaciones o, como se suele nombrar, cuales 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 nombra problema binario.

En el caso que nos ocupa, el *trading* en bolsa, no se tiene una clasificación natural. Crear estas clases de manera que representen un buen momento para comprar o vender será objeto de estudio en nuestro trabajo.

4.1.2. Conjunto de entrenamiento

Una vez que la estructura del árbol esta 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, de los que ya sabemos su clase real. Por tanto, a partir de

este conjunto, deberemos inducir un árbol de decisión cuyas hojas clasifiquen bien los datos conocidos. De esta forma, si los datos con clase desconocida tienen una procedencia parecida a los datos del conjunto de entrenamiento, serán correctamente clasificados. 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 experto 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, queremos un árbol de decisión que nos sirva como modelo para saber cuando situar órdenes de compra y de venta en el mercado de valores.

5.1. Árboles de decisión

El modelo que vamos a utilizar para extraer las señales de compra y venta es un árbol de decisión con tres etiquetas:

- Etiqueta *Buy*. Implica que debemos colocar una orden de compra de acciones, tantas como nos sea posible con el capita disponible.
- Etiqueta *Sell*. Implica que debemos colocar una orden de venta de todas las acciones que se tengan.
- Etiqueta *Stop*. Es una etiqueta comodín. Se usa cuando las condiciones de un camino no son concluyentes para decir si hay que comprar o vender.

En los nodos del árbol se sitúan condiciones que dividen a los nodos en dos ramas, la que cumple la condición y la que no. Las condiciones son de la forma:

$$\text{indicador}(\text{parametros}) \leq \text{pivote}$$

Donde *pivote* es un número real e *indicador* es 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
- OBV

- AD
- TRANGE
- BBANDS_HIGH
- BBANDS_LOW

En el caso de algunos indicadores, como EMA (Exponential Moving Average) o SMA (Simple Moving Average) se han modificado ligeramente por motivos lógicos. Al entrenar un árbol en un determinado periodo, en este aparecerán reglas como $EMA(10) \leq 500$.

Usar esta condición en un periodo posterior no tiene mucho sentido, ya que la potencia del indicador EMA se encuentra al comparar con el valor actual de la acción y ver si se ha producido una bajada o subida importante respecto a los últimos días.

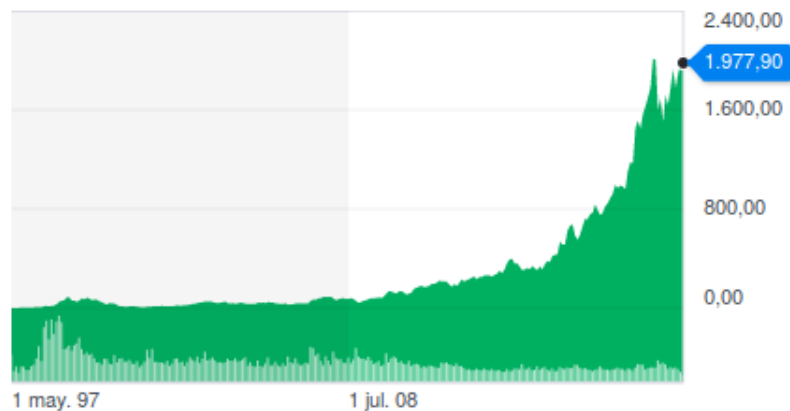


Figura 8: Valor de las acciones de Amazon. Fuente: <https://finance.yahoo.com> [Última consulta 19 de Julio de 2019]

En ciertos casos podría tener sentido usar la condición, por ejemplo, ver si el valor de la acción lleva unos días demasiado alto o demasiado bajo para el periodo de entrenamiento.

Pero 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 una cota fija.

En su lugar, se propone restar o dividir el valor actual al indicador para, de alguna forma, escalar el indicador y hacerlo intemporal.

(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, vamos a usar un patrón genético como el explicado en el apartado 3. En ese apartado se hizo una explicación genérica de un algoritmo genético. Ahora vamos a replicar esa construcción adaptándola a nuestra población, los árboles de decisión.

Primero creamos una generación y mediante una simulación de inversión en un periodo de prueba puntuamos los árboles. Realizamos un proceso de selección y cruce de los árboles. Y por último se sortean una serie de mutaciones. Entonces vuelve a comenzar el proceso de simulación para encontrar la próxima generación.

A continuación se va a profundizar en cada una de las partes de este proceso.

5.2.1. Primera generación

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

Puesto que nuestra simulación se prevé costosa, vamos a realizar un ligero calentamiento de la población para que se reduzcan los pasos necesarios hasta la obtención de una población buena. Para calentar los árboles vamos a usar un proceso 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. Vamos a explicar esto mejor.

5.2.1.1 Etiquetado de datos de prueba

El mismo periodo que vamos a usar para entrenar el algoritmo genético lo vamos a usar para generar esta primera población. Tomamos estos datos y los etiquetamos siguiendo estas consideraciones:

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

$$M = \max\{cierre_i - cierre_j | \forall z \text{ con } i < z \leq j \\ \text{se tiene que } cierre_{z-1} - cierre_z \geq 0\}$$

De forma análoga, tenemos la máxima bajada notada como m :

$$m = \max\{cierre_i - cierre_j | \forall z \text{ con } i < z \leq j \\ \text{se tiene que } cierre_{z-1} - cierre_z \leq 0\}$$

Ahora, definimos dos tipos de tendencias.

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

De forma análoga, una consecución de días se dice con tendencia negativa si en mitad no encontramos una subida superior a la sexta parte de M .

Es decir,

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

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

En última instancia, marcamos 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.

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 podemos observar un ejemplo simple de etiquetado de un periodo. En ella podemos ver 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 tanto, se marca también como decreciente.

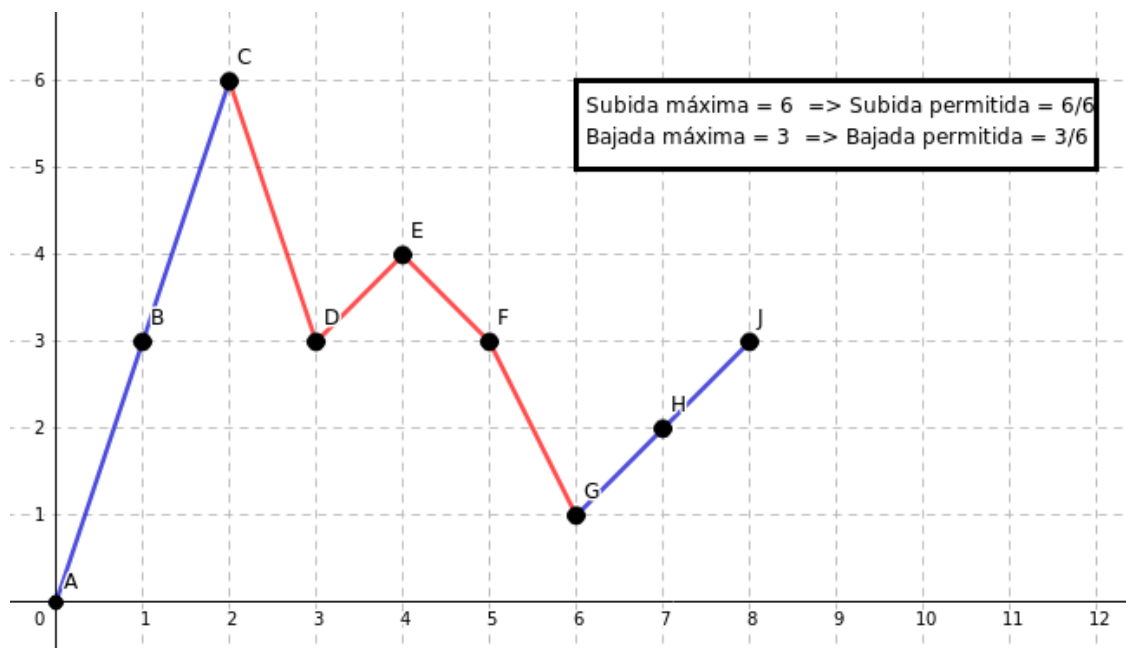


Figura 9: Ejemplo de etiquetado de un periodo. Fuente: elaboración propia

5.2.1.2 Selección de pivotes

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

El pivote final elegido es un valor de una rejilla conformada por 10 valores equidistantes situados entre el máximo y el mínimo valor de los datos que cumplen las condiciones para llegar al nodo.

Puesto que tenemos 4 etiquetas, en primera instancia se debería usar una función de entropía de 4 clases. Pero las etiquetas 2 y -2 son poco frecuentes. Además, debido a que nuestros nodos dividen a los datos en dos ramas, no tiene mucho sentido usar una entropía de 4 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. En el caso de que la proporción de una clase sea 1, se obtiene la información perfecta (la rama solo tiene datos de una clase) y entonces la función entropía no puede calcularse. Hay que tener en cuenta este caso especial y devolver un 1 en lugar de calcular el valor 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 que 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 vamos a usar una vez más la etiqueta puesta en los datos de prueba. Notando $x_{tag=a}$ al número de instancias que 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. Por tanteo se llega a la siguiente adjudicación de la señal de la hoja:

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

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, procedemos a generar la nueva población mediante un cruce que promocióne los árboles que mejores resultados han dado.

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

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 sin modificaciones a la siguiente generación y 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 de los beneficios, restando el beneficio del peor árbol a todos para tener las puntuaciones mayores o iguales que 0.

Con el fin de potenciar las estrategias de compra y venta, más ventajosas que la estrategia de *buy&hold* (comprar al principio y esperar hasta el final para vender) se van a modificar ligeramente las ganancias. Por cada venta que haga un árbol, se suman 10 euros a sus beneficios.

A continuación, se dividen todas las puntuaciones entre la suma de las puntuaciones. Esto produce que la suma sea 1 y, además, cada nueva valoración es proporcional a la cantidad de beneficios que reporta.

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

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

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.

En el caso especial de que todos los árboles tengan los mismos beneficios, este procedimiento nos llevaría a dividir por 0. Para evitar esto, de forma natural dividimos en tantos intervalos de igual dimensión el intervalo $[0,1]$ como árboles haya y continuamos con el caso general.

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 esta propuesta vamos a tratar con el cruce de árboles.

Como ya hemos 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, es una buena forma de agrupar condiciones y crear asociaciones de estas con una señal clara para la 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.

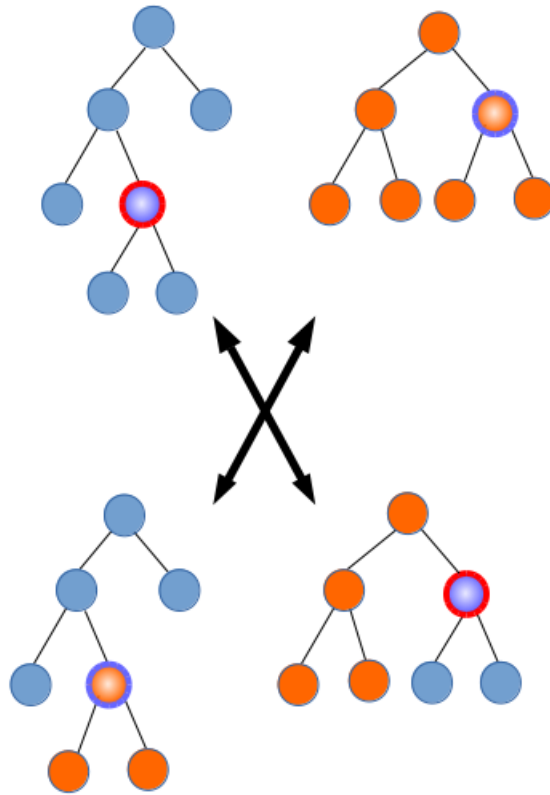


Figura 10: Ejemplo de cruce entre dos árboles. Fuente: elaboración propia

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 en los extremos. Es decir, la población tiene una cierta probabilidad de converger a árboles con demasiados nodos o, en su defecto, con insuficientes nodos.

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 a unos datos con relación a los de entrenamiento pero distintos. Esto es lo que en el ámbito de la ciencia de datos se llama 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 buen modelo para los datos de entrenamiento.

Para solventar estos dos problemas vamos a actuar 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. Notar 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.

Esta preferencia tiene como objetivo evitar la formación de árboles muy pequeños.

- Para eliminar los árboles demasiado grandes, simplemente se propone hacer un método que cuente los nodos de un árbol y, en cada generación, se eliminen los que superen una cierta cantidad. También se ha añadido el caso contrario, si hay pocos nodos en un árbol, éste se elimina.

Para mantener constante la cantidad de individuos 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 pueden cambiar:

- **Los parámetros de los indicadores.** El cruce permite cambiar los nodos de posición, y a la larga cambiar las combinaciones de indicadores. Algunos indicadores, como el EMA (Exponential Moving Average), depende de uno o varios parámetros. En primera instancia se han puesto unos valores lógicos para ellos, pero no tienen por qué ser los mejores.
- **La combinación entre el último nodo y las etiquetas.** El cruce propuesto no permite mover las hojas, salvo si también se mueve un nodo superior. Esto hace que las señales vayan asociadas a unos indicadores elegidos en la primera iteración de forma casi aleatoria.
- **Los pivotes de cada nodo.** A partir de la función de entropía y los datos de entrenamiento se eligen los primeros pivotes. Pero una vez que el nodo cambia de posición a causa de la mutación, el pivote no tiene por qué estar bien elegido.

Es necesario incluir de alguna forma la posibilidad de que estos factores puedan cambiar. El mecanismo, que desarrollaremos en los sucesivos epígrafes, que podemos usar para ello 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, vamos a ir recorriendo el árbol nodo por nodo. 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 o de hojas (en caso de que el nodo fuese una hoja).

5.2.3.1 Mutaciones en indicadores

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

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

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 extraído de una distribución normal $N(pivote, |pivote/8|)$, es decir, se saca de una normal centrada en el antiguo valor y con varianza la octava 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 1.

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 vamos 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. Su uso está restringido a la primera ejecución, luego guarda las etiquetas para ahorrar tiempo.
- **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

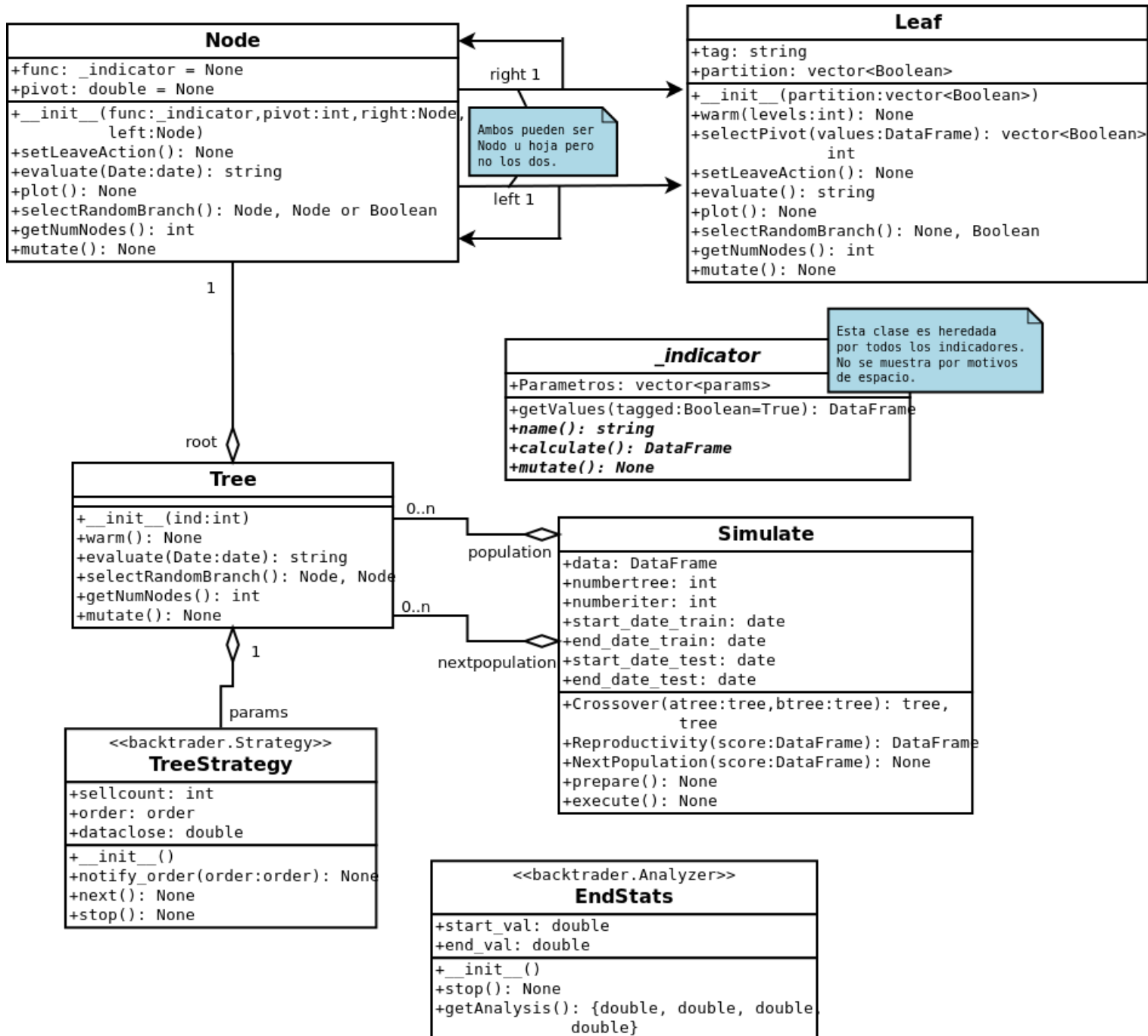


Figura 11: Diagrama de clases. Fuente: elaboración propia

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, disponemos 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. Hablaremos ahora de la posibilidad de ejecutar en paralelo.

Lo primero que tenemos que reformular para poder ejecutar en paralelo es la estrategia. Como vimos, la clase *cerebro* recibe una estrategia como parámetro, que es quien da las señales de compra y venta. De esta forma, si tuviésemos que simular cada uno de los árboles de la población, digamos 50, tendríamos 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, vamos a 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 linealidad 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 significativa 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 días 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-Lib¹⁰. 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 indicadores 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:

¹⁰https://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

7. Análisis de resultados

SE VAN A COMPARAR LOS RESULTADOS CON LOS OBTENIDOS POR MOUSAVI (2014). PERO SON MUCHAS EMPRESAS Y CADA SIMULACIÓN TARDA UN TIEMPO.

8. Conclusiones y trabajos futuros

Bibliografía

- 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.