



ugr | Universidad
de **Granada**

Análisis e implementación en Python de técnicas de aprendizaje automático aplicadas a la Bolsa

TRABAJO FIN DE GRADO
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Autor
Francisco Solano López Rodríguez

Director
José Manuel Zurita López

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Septiembre de 2019

Análisis e implementación en Python de técnicas de aprendizaje automático aplicadas a la Bolsa

Francisco Solano López Rodríguez

Palabras clave: bolsa, trading, predicción, red neuronal, aprendizaje automático

Resumen

Predecir los movimientos de los precios en la bolsa de valores ha sido un tema de gran relevancia, que ha atraído la atención de muchos inversionistas e investigadores durante años. Una de las formas tradicionales de predecir futuras tendencias en el precio consiste en recurrir al análisis técnico para determinar el mejor momento de comprar o vender acciones en el mercado. Sin embargo en los últimos años el avance de la tecnología, ha llevado a utilizar nuevas formas de predicción basadas en técnicas de aprendizaje automático, es aquí donde llegamos al objetivo de nuestro proyecto, en el cual se pretende utilizar dichas técnicas para predecir las tendencias en el precio de las acciones. En este trabajo vamos a recurrir al uso de las redes neuronales, un modelo de predicción muy potente, el cual tiene la capacidad de aprender de los datos y aprovechar dicho aprendizaje para tomar decisiones.

En primer lugar introduciremos conceptos de bolsa, fundamentales para la comprensión del problema abordado. Posteriormente se introducirán teóricamente las redes neuronales, lo cual nos ayudará a comprender el funcionamiento del algoritmo implementado. Una vez introducidos los conceptos teóricos necesarios, pasaremos a documentar el software de la estrategia implementada, mediante redes neuronales. Por último se analizarán diferentes casos de estudio, con los cuales podremos obtener conclusiones sobre el uso de las redes neuronales en la predicción de tendencias en el mercado.

Analysis and implementation in Python of machine learning techniques applied to the Stock Exchange

Francisco Solano López Rodríguez

Keywords: stock exchange, trading, prediction, neural network, machine learning

Abstract

Predicting price movements on the stock market has been a major issue, attracting the attention of many investors and researchers over the years. One of the traditional ways to predict future price trends is to use stock market analysis to determine the best time to buy or sell stock exchange shares in the market. However, in recent years the advance of technology has led to the use of new forms of prediction based on machine learning techniques. This is where we reach the objective of our project, in which we intend to use these techniques to predict trends in the price of shares. In this work we will resort to the use of neural networks, a very powerful prediction model, which has the ability to learn from data and take advantage of that learning to make predictions and help decision making.

Numerous studies on the use of neural networks in the stock market can be found in the literature, most of them focus on the prediction of stock price. However, in this study the problem will be approached in a different way, instead of trying to predict the future value of the price of the shares, we will try to predict the market trends, obtaining in this way the best moments of purchase and sale. We are faced with a binary classification problem, whose labels will be 0 for the bearish trend and +1 for the bullish trend. The decision will be made to buy when the exit from the network of a value is close to 1 and to sell if the exit is close to 0, in case the exit is close to 0.5 no action will be taken.

First, basic stock market concepts will be explained, as these concepts will be fundamental to the understanding of this study. We will look at the two main branches of stock market analysis, which are technical analysis and fundamental analysis, used to obtain advance information on the evolution of stock prices. Special emphasis will be placed on technical analysis, as this will be an important piece of this work, thanks to which through the technical indicators we will be able to feed the neural network with information to carry out the training. All the concepts of the stock market explained in this chapter will be necessary, because they will allow us to understand better the problem addressed and in addition everything learned will be very useful for Backtrader, a Python framework to simulate in a very realistic way, the application of strategies of purchase-sale on historical data and with it we will be able to validate implemented strategy.

Next we will introduce theoretical concepts about neural networks, so we will have a better understanding about the algorithm that we will implement to invest in the stock market. We will begin by describing the simple perceptron, which is the simplest artifi-

cial neuron model, capable of correctly classifying into classes on linearly separable sets. Then we will describe the multilayer perceptron, a neuronal model much more complex than the simple perceptron and capable of solving non-linear classification problems. It will describe the backpropagation algorithm used in multilayer perceptron training and the well-known minimization algorithm, stochastic gradient descent, used to minimize network error on training data. One of the problems faced by neural networks is overfitting, in which the network adjusts so well to the training data that it ends up losing its capacity for generalization, losing its predictive power over data outside the training. Due to this, regularization methods will also be explained to avoid overfitting of the neural network and thus avoid losing the capacity for generalization of the network. This chapter contains a strong mathematical content, so at the end of the project, a mathematical appendix has been included with some necessary definitions such as the definition of linearly separable sets or the definition of the gradient of a function.

Once the necessary theoretical concepts have been introduced, we will now describe the details of the implemented software, which has been developed in the Python programming language. This programming language is easy to learn and generally requires less lines of code compared to other languages. In addition Python has many libraries that will facilitate the development of the program and allow us to focus better on the study itself, without wasting time with irrelevant tasks to this study, because the really important thing is to see the applications of neural networks in the stock exchange and not other tasks such as how to store data (Pandas will solve this task) or design a stock exchange simulator (Backtrader contains the simulator we need). We will see how we have proceeded to obtain the stock exchange data, how these data have been processed and what information has been finally obtained to train the neural network. It will describe the implementation of the design of the neural network and the implementation of the buying and selling strategy that makes its decisions automatically through the neural network implemented. For the design of the neural network the Keras library has been used, a library for neural networks, which allows to design the architecture of these in very few lines of code. For the implementation of the strategy I have made use of the Backtrader library, with which we can define strategies for buying and selling and simulate its use on historical data.

Finally, we will present the study cases carried out on different markets, to see the behavior of the strategy with neural networks, in different situations. We will compare the results obtained with a classic strategy and with the simplest strategy buy and hold. In addition, a comparison will be made with the results of a study that uses a strategy based on fuzzy logic.

Yo, **Francisco Solano López Rodríguez**, alumno de la titulación Doble Grado en Ingeniería Informática y Matemáticas de la **Facultad de Ciencias** y de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** de la **Universidad de Granada**, con DNI 20100444P, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco Solano López Rodríguez

Granada a 21 de agosto de 2019 .

D. José Manuel Zurita López, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Análisis e implementación en Python de técnicas de aprendizaje automático aplicadas a la Bolsa*, ha sido realizado bajo su supervisión por **Francisco Solano López Rodríguez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 21 de agosto de 2019 .

El director:

José Manuel Zurita López

AGRADECIMIENTOS

Agradecer en primer lugar a mis padres por haberme apoyado siempre, y gracias a los cuales hoy soy quien soy. Agradecer también a todos los que han estado siempre a mi lado y los que siempre me han apoyado. Y por último agradecer a mi tutor José Manuel Zurita por haber estado siempre disponible cuando lo he necesitado y gracias al cual este proyecto ha sido posible.

ÍNDICE GENERAL

Abstract	1
Introducción	19
Motivación	20
Objetivos	20
Estructura del trabajo	21
1. Estado del arte	23
2. Mercado de valores	25
2.1. Introducción	25
2.2. Análisis técnico	27
2.2.1. Análisis gráfico	27
2.2.2. Indicadores técnicos	29
3. Redes Neuronales	35
3.1. Perceptrón	35
3.1.1. Descripción del modelo	36
3.1.2. Algoritmo del Perceptrón	37
3.2. Perceptrón multicapa	39
3.2.1. Arquitectura del perceptrón multicapa	39
3.2.2. Propagación de los datos de entrada	40
3.2.3. Algoritmo de retropropagación	42
3.3. Sobreajuste y regularización	46
3.3.1. Métodos de regularización	48
4. Algoritmo propuesto	51
4.1. Herramientas utilizadas	52
4.2. Dataset	53
4.2.1. Cargar el conjunto de datos	53
4.2.2. Características adicionales obtenidas mediante indicadores técnicos	54

4.2.3. Añadiendo las etiquetas al Dataset	57
4.3. Diseño de la Red Neuronal	59
4.4. Estrategía	63
4.5. Ejecutando la simulación de la estrategia	64
5. Casos de estudio	67
5.1. Indicadores	67
5.2. Resultados	68
5.2.1. Nasdaq-100 y Eurostoxx	68
5.2.2. Momentos bajistas	85
6. Conclusiones y trabajo futuro	89
6.1. Análisis de los resultados	89
6.2. Conclusiones	91
6.3. Trabajo futuro	93
A. Anexo Informática	95
A.1. Instalación	95
A.1.1. Requisitos	95
A.1.2. Instalación	95
A.1.3. Uso del programa	96
A.2. Backtrader	99
A.2.1. La clase Cerebro	99
A.2.2. La clase Strategy	102
A.2.3. La clase Analyzers	103
B. Anexo Matemáticas	105
Bibliografía	108

ÍNDICE DE FIGURAS

2.1.	Gráfico de velas japonesas	28
2.2.	Ejemplos de tendencias alcista y bajista	28
2.3.	Comparación de media móvil simple y exponencial	30
2.4.	Indicador estocástico	31
2.5.	Gráfico ROC	32
2.6.	Gráfico MACD	33
3.1.	Ejemplo de conjunto no separable en el espacio original, pero separable al aumentar de dimensión	37
3.2.	Funciones de activación	42
3.3.	Subajuste y sobreajuste	47
3.4.	Error de predicción en el conjunto de entrenamiento y en el de test	47
4.1.	Ejemplos simulaciones	58
4.2.	Asignación de etiquetas SAN	59
6.1.	Beneficios mercados bajistas	91
A.1.	Interfaz gráfica implementada	98
A.2.	Simulación de la estrategia en la interfaz gráfica	99
A.3.	cerebro.plot()	101

INTRODUCCIÓN

En el mundo de los mercados financieros y bursátiles, predecir los movimientos de los precios en la bolsa de valores ha sido un tema de gran importancia que ha sido investigado por muchos inversionistas. Este problema de predicción es de extrema complejidad debido a la gigantesca cantidad de variables que influye en los valores de las acciones. El precio de un determinado valor se puede ver influido por factores económicos, factores políticos, factores sociales, determinados sucesos o noticias, datos de empleo, las tasas de interés, etc. Todo esto hace que predecir con exactitud los movimientos de la bolsa sea una tarea imposible de resolver.

Para determinar cuando es el mejor momento de comprar o vender, muchos inversionistas hacen uso del análisis fundamental, el cual toma sus decisiones en función de la estructura económica de la compañía. En cambio otros economistas financieros siguen las premisas del análisis técnico, el cual supone que toda la información relevante, respecto a los beneficios y expectativas de una determinada compañía está reflejada en el precio del título, por lo que es el mercado el que proporciona la mejor información sobre la evolución futura del mismo.

En los últimos años se ha popularizado el término Machine Learning, también conocido en español como aprendizaje automático. Según Arthur Samuel, este área de la inteligencia artificial pretende que las computadoras tengan la habilidad de aprender sin ser explícitamente programadas. Para ello busca construir algoritmos que puedan aprender de los datos y construyan modelos que permitan tomar decisiones.

Uno de los modelos computacionales más conocido del aprendizaje automático, es el de la Red Neuronal, la cual se trata de un modelo matemático que toma su inspiración en la estructura y organización de las neuronas del cerebro.

Los primeros modelos de redes neuronales datan de los años 50, estas han ido evolucionando con el paso de los años, creándose nuevos modelos hasta llegar a lo que conocemos hoy como aprendizaje profundo o su término en inglés deep learning. En sus principios estas no tuvieron mucho éxito debido a que la tecnología no estaba lo suficientemente avanzada para poder satisfacer la cantidad de recursos computacionales que estas requerían. Sin embargo en la actualidad su uso es de gran importancia, debido al avance tecnológico y a los nuevos problemas que se plantean actualmente con la

enorme cantidad de datos que producimos diariamente.

Las redes neuronales han sido ampliamente utilizadas en la predicción de series temporales, es por ello que estas han llamado la atención de inversionistas, con el objetivo de predecir tendencias en el mercado. En este contexto las redes neuronales tienen el objetivo de tomar decisiones de forma dinámica y en tiempo real.

Motivación

El motivo de este trabajo es el de aplicar nuevas formas de inversión en bolsa, mediante técnicas de aprendizaje automático, en este caso mediante el uso de redes neuronales. Las redes neuronales han demostrado ser eficaces en numerosos campos como puede ser el de la meteorología o detección de enfermedades. En este estudio queremos comprobar también su utilidad en la predicción en el mercado de valores.

Este proyecto también pretende contribuir a la investigación ya existente en el área de la predicción en bolsa con sistemas inteligentes, construyendo para ello un programa y aportando los resultados obtenidos por este. Por ello además de mostrar los resultados, también compararemos estos con el de otra investigación anterior.

Otro motivo de este trabajo es el de servir como guía para poder iniciarse en el uso de las redes neuronales con Python y aplicar estas a la bolsa, además de utilizar un framework de simulación de estrategias para poder validar la red neuronal.

Objetivos

En este trabajo vamos a suponer ciertas las premisas del análisis técnico y vamos a utilizarlas para hacer predicciones sobre el precio de un determinado título, determinando el momento ideal de compra o venta del valor correspondiente, con la intención de obtener un beneficio. Para ello vamos a recurrir a técnicas de aprendizaje automático, para realizar nuestras predicciones, en concreto vamos a estudiar el modelo de las redes neuronales artificiales.

El objetivo principal de este trabajo va a consistir en desarrollar un programa, que mediante el uso de redes neuronales sea capaz de predecir futuras tendencias en el valor de las acciones y nos ayude en la toma de decisiones para saber en qué momento es mejor comprar o vender nuestras acciones. Para el desarrollo del trabajo haremos uso del lenguaje de programación Python. Existen muchos motivos por los que utilizar Python para tareas de aprendizaje automático, uno de los principales es la facilidad de implementación, además de requerir en general mucho menos código que la mayoría de lenguajes de programación. Otro motivo es la gran cantidad de recursos de los que disponemos en este lenguaje para tareas de aprendizaje automático, con bibliotecas como scikit-learn o keras.

La lista de objetivos que se persiguen podría resumirse en las siguientes:

- Desarrollar un algoritmo basado en redes neuronales para predecir tendencias en el mercado.
- Realizar diferentes casos de estudio, en los que llevaremos a cabo simulaciones con el algoritmo implementado, sobre distintos valores de mercado y de esta forma validar el uso de la red neuronal para invertir.
- Comparar los resultados obtenidos con otras estrategias.
- Que el presente trabajo muestre de forma detallada el procedimiento seguido para implementar la red neuronal y validarla en un simulador, de forma que cualquiera que lo desee pueda utilizar este trabajo como guía para realizar experimentos similares.

Estructura del trabajo

Este proyecto se estructura de la siguiente forma:

- **Capítulo 1: Estado del arte.** En este capítulo se describe el estado de conocimiento actual respecto al campo de investigación al que este proyecto pretende realizar su aportación. Veremos algunos de los últimos estudios realizados, de los cuales partiremos para empezar la investigación.
- **Capítulo 2: Mercado de valores.** En este capítulo se introducen conceptos básicos de bolsa, centrándonos principalmente en el análisis técnico. En él, veremos los principales indicadores técnicos que se utilizan para la elaboración de técnicas de compraventa en el mercado de valores.
- **Capítulo 3: Redes neuronales.** En este capítulo se describe el modelo del perceptrón multicapa. Veremos tanto la arquitectura de dicho modelo, como el desarrollo matemático del algoritmo de retropropagación utilizado para entrenar las redes neuronales.
- **Capítulo 4: Modelo y estrategia propuesta.** En este capítulo se describe el modelo de perceptrón multicapa implementado y las características utilizadas para el entrenamiento de dicho modelo. Además, se describe la estrategia de compraventa implementada, esta estrategia utiliza la red neuronal para emitir señales de compraventa.
- **Capítulo 5: Casos de estudio.** En este capítulo se muestran los resultados de simular la estrategia implementada sobre numerosos mercados. Se muestra también la comparación con otras estrategias y además con los resultados de un artículo de investigación.
- **Capítulo 6: Conclusiones y trabajos futuros.** En este capítulo se realizará un análisis de los resultados obtenidos para obtener conclusiones sobre el uso de las

redes neuronales en el mercado de valores y se comentarán trabajos futuros sobre este tema, con los cuales poder avanzar más en este campo de investigación.

- **Anexo Informática.** En este anexo se describe la instalación del programa implementado. También se describe un pequeño tutorial de Backtrader, el simulador de bolsa que utilizaremos.
- **Anexo Matemáticas** En este anexo se introducirán definiciones matemáticas necesarias para entender el capítulo de redes neuronales, en el cual se manejan conceptos matemáticos que pueden no resultar del todo familiares, debido a esto se ha visto necesario incluir este anexo.

Capítulo

1

ESTADO DEL ARTE

Burton Gordon Malkiel, conocido por su famoso libro *A random walk down Wall Street*, defiende la hipótesis del mercado eficiente, la cual expresa que el mercado refleja toda la información actual disponible. Esto nos dice que el mercado sigue un paseo aleatorio y que por tanto es imposible predecir los movimientos de este. Por otro lado tenemos la opinión contraria defendiendo que el mercado no es eficiente. Una de las personas a favor de esta teoría, es el gran inversionista Warren Buffett, el cual defiende la hipótesis del mercado no eficiente, considerándose a si mismo como una demostración de esta teoría, debido a los resultados extraordinarios que ha obtenido.

A pesar de la hipótesis del mercado eficiente, intentar predecir las tendencias del mercado ha sido durante muchos años atrás un importante campo de estudio. Los métodos de predicción utilizados abarcan desde técnicas clásicas del análisis bursátil (dentro del cual podemos distinguir entre el análisis técnico y el análisis fundamental), hasta técnicas basadas en inteligencia artificial. Dentro de las técnicas basadas en inteligencia artificial, nos encontramos por ejemplo algoritmos genéticos o modelos de aprendizaje automático entre los que podemos distinguir modelos lineales como el de regresión lineal y modelos no lineales como puede ser Random Forest. Respecto a los modelos no lineales destacamos las redes neuronales ya que son la base de este trabajo y serán estas sobre las cuales centraremos la atención en este capítulo, para enunciar algunas de las últimas investigaciones de su uso en la bolsa.

La predicción en el mercado de valores mediante las aplicación de redes neuronales ha crecido mucho en los últimos años y se han realizado multitud de estudios sobre sus aplicaciones en el mercado, por lo que la literatura disponible al respecto es bastante extensa. Entre los últimos estudios sobre el uso de redes neuronales en bolsa destacamos los siguientes:

- I. El trabajo realizado por Eunsuk Chong, Chulwoo Han y Frank C. Park en su investigación *Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies*[1], en la cual utilizan el aprendizaje

profundo sobre datos de alta frecuencia del mercado koreano.

- II. También podemos encontrarnos el trabajo de Feng Zhou, Hao-min Zhou, Zhihua Yang y Lihua Yang *A strategy combining empirical mode decomposition and factorization machine based neural network for stock market trend prediction*[2], en el cual utilizan máquinas de factorización basadas en redes neuronales sobre los datos del índice compuesto de la Bolsa de Valores de Shanghai (SSEC), el índice de cotizaciones automatizadas de la National Association of Securities Dealers (NASDAQ) y el índice compuesto de precios de acciones Standard & Poor's 500 (S&P 500).
- III. Otro estudio se trata del realizado por Yang Liu (2019) *Novel volatility forecasting using deep learning–Long Short Term Memory Recurrent Neural Networks*[3] en el cual hace uso de redes neuronales recurrentes con retroalimentación para predecir la volatilidad sobre los índices de S&P500 y AAPL.
- IV. Por último mencionar el trabajo de Jing Zhang, Shicheng Cui, Yan Xu, Qianmu Li y Tao Li en su estudio *A novel data-driven stock price trend prediction system*[3], en el cual tratan de predecir las tendencias en el mercado de China.

De los estudios anteriores los 3 primeros utilizan redes neuronales para predecir el valor de las acciones (o la volatilidad como en el caso del estudio III), luego se tratan de redes neuronales que tratan de resolver un problema de regresión, en el cual intentarán minimizar la diferencia entre los valores reales y los obtenidos por la red. En nuestro trabajo utilizaremos otro enfoque en el cual no trataremos de predecir el valor exacto de los precios, en su lugar intentaremos predecir la dirección de la tendencia al igual que se hace en el último estudio mencionado.

Por último mencionar el trabajo de Rodrigo Naranjo, Javier Arroyo, Matilde Santos en su trabajo *Fuzzy modeling of stock trading with fuzzy candlesticks*[4], en el cual proponen una metodología para detectar patrones de velas, mediante el uso de lógica difusa. No imitaremos los pasos de este trabajo, pero si que compararemos los resultados obtenidos en este estudio, con nuestros resultados. Los datos que utilizan para la validación se corresponden con las acciones que componen el índice Nasdaq-100, en los que podemos encontrarnos por ejemplo las acciones de Apple, Adobe o Amazon. También utilizan las acciones del Eurostoxx entre las que podemos encontrar por ejemplo las acciones del Banco Santander.

Capítulo

2

MERCADO DE VALORES

En este capítulo vamos a introducir conceptos básicos sobre el trading, también llamado en español como negociación bursátil. Casi todo lo explicado aquí, será necesario para entender bien Backtrader, el framework de Python, el cual se explica en el anexo de informática, y gracias al cual podremos realizar simulaciones sobre datos históricos de cotizaciones. Con este framework podremos probar nuestros modelos de predicción sobre datos reales, para determinar la validez de nuestros modelos. Este framework de simulación contiene un broker virtual, que nos permitirá crear órdenes de compra y venta y nos permitirá definir estrategias haciendo uso del análisis técnico, es por eso que en este capítulo haremos un breve repaso de todos estos conceptos.

2.1 Introducción

El trading consiste en la compra venta de un determinado valor como puede ser un índice, acciones, divisas o materias primas y cuya finalidad es la obtención de un beneficio. Este beneficio puede obtenerse comprando cuando se piense que va a subir el precio o vendiendo cuando se crea que el precio bajará. Así podría decirse que el trading es un negocio puramente especulativo.

Las personas que se dedican a comprar y vender valores para obtener beneficio de ello, son los denominados traders. La compraventa de acciones se debe hacer por medio de un broker, el cual es una persona física o empresa con licencia para poder realizar operaciones de corretaje financiero y se encargan de actuar como intermediarios entre compradores y vendedores, llevando a cabo las órdenes bursátiles que estos dan. El beneficio de los brokers se encuentra en las comisiones que cobran por realizar las compras y ventas de las acciones.

Una orden bursátil es aquella que el inversor da a un intermediario para que este la realice en el mercado. Las órdenes deben quedar perfectamente identificadas, conte-

niendo si se trata de compra o venta, los valores de la orden, volumen de participaciones dispuesto a negociar, precio al que se quiere comprar o vender, ejecución de la orden y el plazo de validez.

Las ordenes tienen un orden de prioridad, ordenando de mayor a menor precio si se trata de una orden de compra y al revés si se trata de una orden de venta. Si coincidieran las ordenes, estas se ordenarán por el momento en el que se emitieron.

Tipos de ordenes:

- **Orden de apertura**, ejecución en el momento de apertura.
- **Orden de mercado**, se ejecuta al precio actual del mercado.
- **Orden limitada**, limitada al precio que indique el inversor. En el caso de las compras un límite superior y en el caso de las ventas un límite inferior.
- **Orden por lo mejor**, toma el mejor precio que ofrece el mercado en el momento de su emisión.
- **Orden On Stop**, la orden se emite cuando la cotización llega al precio que se ha determinado, convirtiéndose en dicho momento en una orden de mercado.
- **Orden Stop-Limitada**, la orden se emite cuando la cotización llega al precio que se ha determinado, convirtiéndose en dicho momento en una orden limitada.
- **El Stop Dinámico**, el precio de Stop se mueve de forma paralela al precio de la cotización si el movimiento es favorable y se mantiene en caso desfavorable.

Análisis bursátil

Una forma de obtener beneficios mediante la compra y venta en bolsa, consiste en estudiar los mercados, haciendo uso del análisis bursátil, el cual estudia los activos del mercado financiero para obtener información anticipada sobre la evolución de sus cotizaciones. Este análisis se puede dividir en análisis fundamental y análisis técnico.

- **Análisis fundamental:** este consiste en, a partir de la estructura económica de la compañía, determinar la situación de la misma así como las variables que la afectan y en su caso, realizar predicciones sobre la evolución de la compañía en el futuro. Este requiere un estudio detallado de los estados contables de la empresa, planes de expansión, expectativas futuras y factores del entorno socio económico que puedan afectar a la marcha de la empresa.
- **Análisis técnico:** se concentra en el estudio del comportamiento del mercado en sí mismo, pues supone que toda la información relevante, en cuanto a los beneficios y expectativas de una determinada compañía, al sector al que pertenece, a la economía en general, al entorno socio político,... está ya reflejada en el precio del título y, por tanto, es el propio mercado el que proporciona la mejor información sobre la evolución futura del mismo. Para ello se basa en tres premisas:

1. Todo lo que puede afectar al precio de cualquier valor está totalmente descontado.
2. Los precios se mueven en tendencias.
3. La historia se repite.

Para nuestro estudio vamos a hacer uso de este último, el análisis técnico. Para ello utilizaremos datos de cotización de una determinada empresa y haremos uso de los llamados indicadores técnicos, estos son fórmulas matemáticas y estadísticas que se aplican a las series de precios y volúmenes para ayudar a tomar decisiones.

En lo que queda de capítulo explicaremos de forma más detallada el análisis técnico, ya que este nos será necesario posteriormente, cuando implementemos las redes neuronales.

2.2 Análisis técnico

Como ya hemos dicho, el análisis técnico se basa en el estudio del comportamiento del mercado. En general el análisis técnico suele usarse para referirse a todo el análisis relacionado con la utilización de gráficos. Este usa como base las cotizaciones y suele utilizar fórmulas matemáticas y estadísticas para eliminar la subjetividad derivada del análisis gráfico.

2.2.1 Análisis gráfico

El análisis gráfico, también llamado análisis chartista, es la ciencia que se dedica a la observación de la evolución de los mercados a partir de su representación gráfica. Estos gráficos muestran la evolución de cualquier valor cotizado, junto con el volumen de las negociaciones. Existe muchos tipos de gráficos, pero en este trabajo nos bastará con conocer los gráficos de velas japonesas.

En los gráficos de velas japonesas en el eje de abscisas se representa el tiempo y en el eje de ordenadas el precio. Cada día cotizado se representa mediante una barra con forma de vela, donde el cuerpo principal representa el rango entre el precio de apertura y el de cierre del período, si el precio de cierre es superior al de apertura entonces el cuerpo principal se representa de color blanco, en caso contrario de color negro (también pueden representarse con otros colores, como en la gráfica mostrada continuación verde y rojo). Si el precio de apertura y cierre coinciden, este se representa con una raya horizontal llamada doji. Por encima y debajo del cuerpo principal surgen unas líneas llamadas sombras, que representan los precios máximos y mínimos del período.



Figura 2.1: Gráfico de velas japonesas

Tendencias

El objetivo del análisis gráfico es el de identificar las posibles tendencias en los precios, para operar a favor de ellas y además intentar determinar el momento en el que la tendencia empieza a cambiar de dirección, pasando de alcista a bajista o viceversa.

La tendencia es la dirección en la que se mueve el mercado formando ondas sucesivas. Cuando los senos y las crestas son cada vez más altos, se dice que la tendencia es alcista, mientras que si van en descenso se dice que la tendencia es bajista. Si el precio oscila en un movimiento lateral se define una tendencia lateral. Dentro de las tendencias podemos diferenciar entre tendencias a corto y tendencias a largo, dependiendo de si el período de tiempo considerado es de menor o mayor intervalo. En la siguiente gráfica podemos ver un ejemplo de tendencia alcista (línea verde) y bajista (línea roja).



Figura 2.2: Ejemplos de tendencias alcista y bajista

En este trabajo no vamos a entrar en más detalles sobre el análisis gráfico, pues no va a ser utilizado para la construcción de nuestro modelo de predicción, es por eso que no

vamos a explicar estrategias basadas en este, en cambio vamos hacer bastante hincapié en los indicadores técnicos, ya que su uso va a ser fundamental.

2.2.2 Indicadores técnicos

Los indicadores técnicos son fórmulas matemáticas y estadísticas que se aplican a una serie de precios y volúmenes con la intención de ayudar a tomar decisiones de inversión. Los indicadores técnicos nos van a servir para eliminar la subjetividad del análisis gráfico. Dentro de los indicadores técnicos podemos distinguir entre seguidores de tendencia y osciladores.

- **Los seguidores de tendencia** muestran cuando la evolución del precio de un activo sigue una tendencia alcista, bajista o lateral. Son útiles para ver cuando esta perdiendo fuerza una tendencia o cuando comienza una nueva.
- **Los osciladores** nos permiten detectar zonas de sobrecompra o sobreventa, divergencias y convergencias que nos ayudarán a anticipar cambios de tendencia.

A continuación pasamos a explicar algunos de los indicadores técnicos más conocidos, viendo como se calculan y mostrando algunos ejemplos gráficos.

Media móvil simple

Una media móvil consiste en realizar un promedio de un bloque de información a lo largo del tiempo. Sirve como mecanismo para suavizar los gráficos, facilitando de esta forma la visión de una tendencia. En bolsa se suelen utilizar las medias móviles sobre el precio de cierre, aunque ciertas estrategias también utilizan el precio de apertura, el máximo o mínimo del periodo seleccionado.

Dada una secuencia $\{T_i\}_{i=1}^N$ una n media móvil es una secuencia $\{S_i\}_{i=1}^{N-n+1}$, donde cada S_i viene dado por:

$$S_i = \frac{1}{n} \sum_{j=i}^{i+n-1} T_{i-j}$$

Basicamente la media móvil simple es la media aritmética de los n datos anteriores a un dato dado. La media móvil simple da la misma importancia a todas las cotizaciones tomadas en cuenta, a la hora de realizar los cálculos.

Media móvil exponencial

En lugar de ponderar todos los días con el mismo valor como en el caso de la media móvil simple, podría ser preferible dar mayor importancia a las cotizaciones más re-

cientes. La media móvil exponencial hace justo esto, mediante ponderaciones da mayor importancia a las cotizaciones, cuanto más recientes sean estas.

Dada una secuencia $\{T_i\}_{i=1}^N$ una n media móvil exponencial es una secuencia $\{S_i\}_{i=1}^{N-n+1}$, donde cada S_i se calcula de forma recursiva como sigue:

$$S_i = \begin{cases} T_1, & i = 1 \\ \alpha T_i + (1 - \alpha)S_{t-1}, & i > 1 \end{cases}$$

La media móvil exponencial puede verse como una media móvil ponderada exponencialmente. Al ponderar más los últimos precios, es más sensible a cambios de tendencia, pudiéndose anticipar a la media móvil simple.

En la imagen siguiente se muestra un ejemplo de medias móviles de 14 días. Podemos apreciar como se ha suavizado la gráfica, gracias a las medias móviles. También podemos observar como la media móvil exponencial al ponderar con más importancia las cotizaciones más recientes, consigue adelantarse a la media móvil simple, en la detección de las tendencias.



Figura 2.3: Comparación de media móvil simple y exponencial

Indicador de momento

El indicador de momento se trata de un indicador de tipo oscilador que principalmente se usa junto con otros indicadores o como indicador de confirmación.

Se calcula como la diferencia entre el precio de un día y el de un periodo anterior. Este oscilador no siempre se mueve dentro del mismo rango, por ello es necesario tener en cuenta su dirección y valor relativo, en lugar de su valor. Se suele utilizar como parámetro el periodo de 12 sesiones.

Desviación típica

La desviación típica es una medida de dispersión. Este valor expresa en cierto modo como de alejados están los valores respecto de su media. Se calcula de la siguiente forma.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}, \text{ donde } \mu = \frac{1}{N} \sum_{i=1}^N x_i$$

Este indicador nos da una medida de la volatilidad del mercado y con ello de cierta forma una medida del riesgo de inversión en un momento dado. Si el valor de este indicador es muy alto significará que los precios fluctúan demasiado y se puede considerar en tal caso que el mercado es volátil.

Oscilador estocástico

Es un indicador de tipo oscilador, que mide la posición relativa del precio de cierre de un día, respecto de los precios de cierre de los días anteriores, en un intervalo de tiempo considerado. Su cálculo se obtiene con las siguientes fórmulas:

$$\%K = 100 \times \frac{\text{cierre}_{\text{hoy}} - \min\{\text{período analizado}\}}{\max\{\text{período analizado}\} - \min\{\text{período analizado}\}}$$

$\%D$ = Media Móvil Simple de 3 períodos de $\%K$



Figura 2.4: Indicador estocástico

Rate of change

El indicador Rate of Change es un indicador de tipo oscilador, que mide la variación de los precios entre dos períodos.

$$ROC = \frac{\text{cierre}_{\text{hoy}} - \text{cierre}_{\text{hoy-n}}}{\text{cierre}_{\text{hoy-n}}}$$

Es similar al indicador de momento, pero este nos da un valor relativo, a diferencia del indicador de momento que calculaba la diferencia absoluta.

Este indicador sirve para mostrar señales de compra y venta. También puede utilizarse para detectar divergencia, en las que las tendencias están perdiendo fuerza y es posible un cambio de tendencia.

Este indicador se interpreta identificando zonas de compra cuando se sobrepasa la línea del cero de manera ascendente y de venta en caso contrario. En la gráfica siguiente puede apreciarse claramente como este indicador tiene un valor positivo en las zonas de tendencia alcista, mientras que tiene valor negativo en zonas de tendencia bajista.



Figura 2.5: Gráfico ROC

Índice de fuerza relativa (RSI)

El índice de fuerza relativa es un indicador de tipo oscilador. Este muestra la fuerza del precio mediante la comparación de los movimientos individuales al alza o a la baja de los sucesivos precios de cierre. Nos indica zonas de sobreventa y sobrecompra, y además proporciona señales de compra/venta.

Este indicador se mueve en una escala de 0 a 100, y presenta dos áreas que marcan los niveles de sobreventa [0 – 20 ~ 30] y de sobrecompra [70 ~ 80 – 100].

Se calcula como: $RSI = 100 - \frac{100}{1 + RS}$, donde $RS = \frac{MME(U)}{MME(D)}$

RS se corresponde con la fuerza relativa y MME significa media móvil exponencial. U y D se calculan como:

Si el día es de subida

- $U = cierre_{hoy} - cierre_{ayer}$
- $D = 0$

Si el día es de bajada

- $U = 0$
- $D = cierre_{ayer} - cierre_{hoy}$

MACD

El indicador MACD, cuyas siglas se deben al inglés (Moving Average Convergence Divergence), conocido en español como media móvil de convergencia/divergencia, es un indicador que tiene como objetivo detectar tendencias. Presenta tres componentes, la primera componente es la denominada MACD, la segunda es la señal y la tercera el histograma.

- **MACD**, viene dado por la diferencia de dos medias móviles exponenciales. La primera media se corresponde con el promedio rápido y por lo general se utiliza un promedio de 12. La segunda la lenta y suele utilizarse un valor de 26.
- **Señal**, se calcula como la media móvil exponencial del MACD. Se suele utilizar un promedio de 9 períodos.
- **Histograma**, se obtiene como la diferencia entre el MACD y la señal.



Figura 2.6: Gráfico MACD

Capítulo

3

REDES NEURONALES

En este capítulo estudiaremos las redes neuronales, analizando la parte teórica que hay tras este modelo de aprendizaje y entendiendo poco a poco como funcionan. Las redes neuronales artificiales son modelos simplificados de las redes neuronales biológicas. Estas tratan de resolver complejos problemas computacionales como puede ser el reconocimiento de patrones, predicción meteorológica o reconocimiento de objetos. No existe una definición exacta de que es una red neuronal, en este trabajo vamos a exponer la definición de red neuronal propuesta en el libro *Fundamentals of Artificial Neural Networks* de Muhamad H. Haussoun:

Una red neuronal es un modelo computacional, paralelo, compuesto de unidades procesadoras adaptativas con una alta interconexión entre ella.

Sin conocimiento previo sobre que es una red neuronal, esta definición puede parecer difícil de entender, pero a lo largo de este capítulo iremos aumentando nuestra compresión sobre las redes neuronales y entendiendo cada vez mejor la definición anterior. Comenzaremos viendo el modelo del perceptrón simple y acabaremos viendo el modelo del perceptrón multicapa y algunas variantes de este. Veremos también métodos de regularización para evitar el llamado sobreajuste, en el cual nuestro modelo se ajusta extremadamente bien a los datos del entrenamiento, pero cuando le proporcionamos datos nuevos no es capaz de clasificarlos bien.

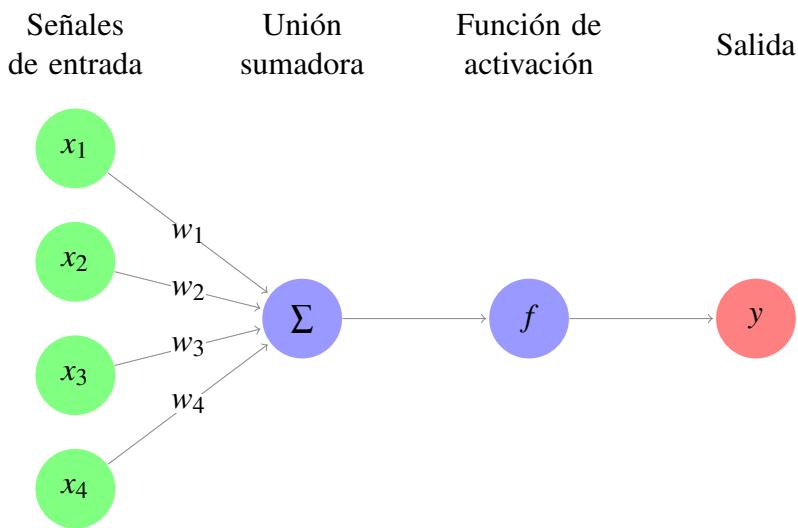
3.1 Perceptrón

El modelo del perceptrón se trata del primer modelo desarrollado de una neurona artificial. El objetivo de este es el de realizar tareas de clasificación de forma automática. Este modelo toma un conjunto de ejemplos de entrenamiento, donde cada ejemplo viene acompañado de la clase a la que pertenece. El perceptrón se encarga de usar dichos datos

de entrenamiento para obtener un modelo capaz de clasificar ejemplos nuevos, dando como salida la clase a la que pertenecen dichos ejemplos.

3.1.1 Descripción del modelo

La arquitectura de este modelo tiene una estructura monocapa, en la que un conjunto de entradas se conectan a la neurona y cuyas conexiones tienen asociados unos valores denominados pesos sinápticos, a los que nos referiremos simplemente como pesos. La siguiente ilustración muestra un ejemplo de perceptrón en la que se puede ver la arquitectura descrita:



El perceptrón toma una lista de entradas x_1, x_2, \dots, x_n , asociadas a unos pesos w_1, w_2, \dots, w_n , llamados pesos sinápticos y devuelve como salida el valor obtenido mediante la siguiente expresión:

$$\text{salida} = \begin{cases} -1 & \text{Si } \sum_{j=1}^n w_j x_j < \text{umbral} \\ +1 & \text{Si } \sum_{j=1}^n w_j x_j \geq \text{umbral} \end{cases} \quad (3.1)$$

Como podemos ver solo hay dos salidas posibles, de donde podemos deducir que el perceptrón es utilizado para problemas de clasificación binaria. En realidad los valores -1 y +1 de salida no son relevantes, lo importante es que hay dos valores posibles, es por esto que en otros textos podremos ver otras salidas distintas como por ejemplo 0 y 1.

Si $x = (x_1, x_2, \dots, x_n)^T$ y $w = (w_1, w_2, \dots, w_n)^T$ entonces la sumatoria que aparece en la expresión 3.1.1 puede escribirse como $w \cdot x$, donde se entiende al operador \cdot como el producto escalar euclídeo entre dos vectores B.2.

Definimos ahora $\theta = -\text{umbral}$, donde umbral es el valor que aparece en la ecuación 3.1.1. Además fijémonos en que podemos expresar dicha ecuación como el resultado de

pasar el valor $w \cdot x + \theta$, a la función $signo()$, donde la función signo viene dada por:

$$signo(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

Luego tenemos que la ecuación 3.1.1 se puede expresar como:

$$salida = signo(w \cdot x + \theta) \quad (3.2)$$

La función utilizada para calcular la salida del perceptrón, es llamada función de activación o función de transferencia. En este caso la función de transferencia utilizada se trata de la función signo.

En el problema de clasificación binaria, al que se enfrenta el perceptrón, tenemos un conjunto de datos de entrenamiento, donde cada dato viene con una etiqueta asociada. Esta etiqueta nos define dos clases de conjuntos, un conjunto de datos con etiqueta -1 y otro conjunto de datos con etiqueta +1. El objetivo del perceptrón sera encontrar el hiperplano que separe estos dos conjuntos.

Entonces tenemos que para el perceptrón existirán unos pesos w_i y un valor de θ de forma que siempre clasifique bien los datos de entrenamiento, solamente si dichos datos son linealmente separables B.4.

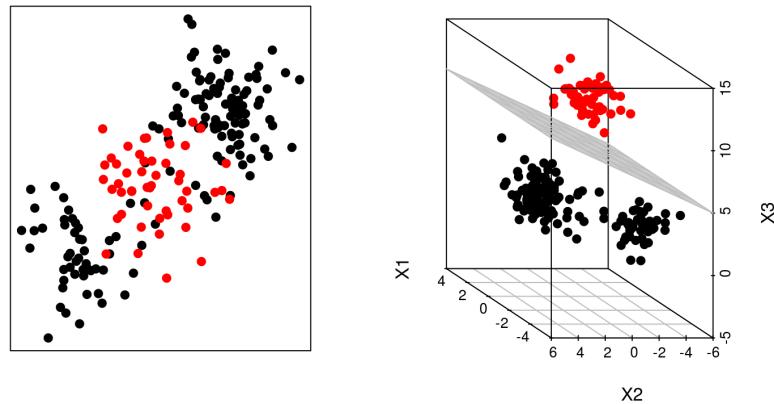


Figura 3.1: Ejemplo de conjunto no separable en el espacio original, pero separable al aumentar de dimensión

3.1.2 Algoritmo del Perceptrón

Veamos ahora el algoritmo del perceptrón, pero antes de ello vamos a cambiar un poco la notación. La salida del perceptron para un dato $x = (x_1, \dots, x_n)$, viene dada por la expresión $signo(w \cdot x + \theta)$. Llamemos x' al vector dado por $(x_1, \dots, x_n, 1)$, donde lo único que hemos hecho es añadirle un 1 al final y llamemos w' al vector $(w_1, \dots, w_n, \theta)$. Entonces tenemos que $signo(w \cdot x + \theta) = signo(w' \cdot x')$. Esta nueva notación nos ahorra

tener que sumar θ y además nos va a ser útil para el algoritmo del perceptrón. Por ello vamos a suponer ahora que los datos de entrenamiento, que utilizaremos en el algoritmo de aprendizaje, van a tener el valor 1 al final.

El algoritmo del perceptron consiste en ir modificando los valores de los pesos, hasta encontrar un hiperplano que separe las clases en dos conjuntos.

En el siguiente pseudocódigo podemos ver dicho algoritmo.

Algoritmo 1: Algoritmo del Perceptrón

Entrada: $(x_i, y_i), i = 1, \dots, n$
 Inicializar w aleatoriamente
mientras !Convergencia hacer
 para $i = 1, \dots, n$ **hacer**
 si $y_i = -1$ y $signo(w \cdot x_i) = +1$ **entonces**
 $w = w - \eta x_i$
 fin
 si $y_i = +1$ y $signo(w \cdot x_i) = -1$ **entonces**
 $w = w + \eta x_i$
 fin
 fin
fin

Podemos ver que la entrada se corresponde con un conjunto de datos (x_1, \dots, x_n) , junto con las etiquetas correspondientes (y_1, \dots, y_n) , que podrán tener el valor -1 ó +1. Después inicializamos los pesos de forma aleatoria.

A continuación entramos en el bucle principal del algoritmo, que se repetirá mientras no se cumpla el criterio de convergencia. Dentro de dicho bucle iteraremos sobre todos los datos de entrenamiento. Los pesos se modificarán solamente cuando la etiqueta original no coincida con la salida del perceptrón, esto es cuando $y_i \neq signo(w \cdot x_i + \theta)$.

Nota: el valor de η es positivo y se llama razón de aprendizaje.

- Si la salida para el dato x_i es positiva pero la etiqueta original es negativa, entonces substraemos a w el vector ηx_i . Explicación:

$$(w - \eta x_i) \cdot x_i = w \cdot x_i - \eta \|x_i\|^2 \text{ con } \eta > 0, \|x_i\|^2 \geq 0$$

Entonces se tiene que $(w - \eta x_i) \cdot x_i \leq w \cdot x_i$

Luego deducimos que restar ηx_i al vector w , hace que nos acerquemos a nuestro objetivo, que es conseguir que $signo(w \cdot x_i + \theta)$ sea igual a -1.

- Si la salida para el dato x_i es negativa pero la etiqueta original es positiva, entonces sumamos a w el vector ηx_i . Explicación:

$$(w + \eta x_i) \cdot x_i = w \cdot x_i + \eta \|x_i\|^2 \text{ con } \eta > 0, \|x_i\|^2 \geq 0$$

Entonces se tiene que $(w + \eta x_i) \cdot x_i \geq w \cdot x_i$

Luego deducimos que sumar ηx_i al vector w , hace que nos acerquemos a nuestro objetivo, que ahora es conseguir que $signo(w \cdot x_i + \theta)$ sea igual a +1.

Se puede demostrar que el algoritmo del perceptrón converge cuando los datos son linealmente separables.

3.2 Perceptrón multicapa

El perceptrón multicapa se trata de una generalización del perceptrón simple, que nació debido a la incapacidad del perceptrón para resolver problemas no lineales. Con el perceptrón multicapa se elimina dicha incapacidad, pudiéndose demostrar que es un aproximador universal, en el sentido de que cualquier función continua sobre un compacto de \mathbb{R}^n puede aproximarse con un perceptrón multicapa, con al menos una capa oculta. Esto sitúa al perceptrón multicapa como una nueva clase de funciones para aproximar o interpolar relaciones no lineales entre datos de entrada y salida.

El perceptrón multicapa tiene la capacidad de aproximar relaciones no lineales y filtrar el ruido que pueda haber en los datos, esto hace que sea un modelo adecuado para abordar problemas reales. A pesar de esto presenta algunas limitaciones como el largo proceso de aprendizaje en problemas complejos dependientes de un gran número de variables, la dificultad que existe en ocasiones de codificar problemas reales mediante valores numéricos, etc...

3.2.1 Arquitectura del perceptrón multicapa

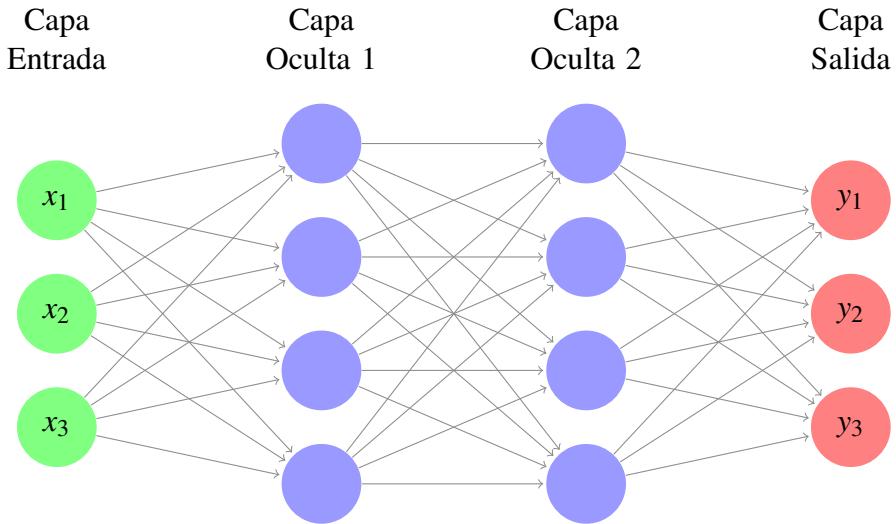
Las neuronas del perceptrón multicapa están organizadas en capas de diferentes niveles. Se pueden distinguir tres tipos de capas: la capa de entrada, las capas ocultas y la capa de salida.

Las neuronas de la capa de entrada solamente se encargan de recibir las señales del exterior y propagarlas a todas las neuronas de la siguiente capa. La última capa se trata de la capa de salida, la cual proporciona al exterior la respuesta de la red neuronal para cada uno de los patrones que ha recibido la red como entrada. Las neuronas de las capas ocultas realizan un tratamiento no lineal de los datos de entrada.

En el perceptrón multicapa las conexiones están dirigidas hacia delante, de forma que las neuronas de una capa se conectan con las neuronas de la capa siguiente. Al igual que en el perceptrón simple las conexiones entre neuronas tienen un peso asociado y las neuronas tienen un umbral asociado. El umbral suele tratarse como si fuera una conexión más con entrada constante e igual a 1.

Se dice que la red tiene una conectividad total debido a que las neuronas de una capa están conectadas con todas las neuronas de la siguiente capa. Lo mismo ocurre con las entradas que se conectan con todas las neuronas de la primera capa oculta.

La siguiente ilustración muestra un ejemplo de perceptrón multicapa con dos capas ocultas, 3 neuronas en la capa de entrada y 3 en la capa de salida.



3.2.2 Propagación de los datos de entrada

El perceptrón multicapa obtiene las salidas para cada dato de entrenamiento propagando los valores de entrada de una capa a la siguiente hasta llegar a la capa de salida. Para ello cada neurona procesa los datos que recibe y produce una salida que propaga hacia la siguiente capa. En esta sección vamos a ver como realiza este proceso el perceptrón multicapa, y que funciones de activación suele utilizarse.

Vamos a suponer que nuestro perceptrón tiene C capas, esto es una capa de entrada, $C - 2$ capas ocultas y una capa de salida.

Vamos a denotar por $w^c = (w_{ji}^c)$ la matriz de pesos asociada a las conexiones de la capa c a la capa $c + 1$, y donde w_{ji}^c representa el peso de la neurona i de la capa c a la neurona j de la capa $c + 1$. La notación de w_{ji}^c puede parecer un poco extraña, ya que parece más intuitivo que el primer subíndice sea el correspondiente a la capa c y el segundo subíndice a la capa $c + 1$, en lugar de al revés. Más adelante veremos que esta forma de expresarlo nos será de utilidad para simplificar la notación.

El umbral de la neurona j de la capa c , lo notaremos como θ_j^c .

La activación de la neurona j de la capa c , la notaremos como a_j^c . Las activaciones se calculan del siguiente modo:

- Activación de las neuronas de la capa de entrada de la red.

$$a_j^1 = x_j \quad \forall j = 1, 2, \dots, n$$

donde $X = (x_1, x_2, \dots, x_n)$ representa el vector de entrada de la red. Luego las activaciones de la capa de entrada simplemente se corresponde con las entradas de la red.

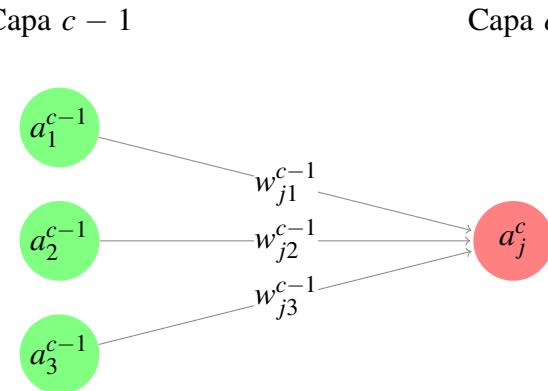
- Activación de las neuronas de la capa c , con $c > 1$, (capas ocultas y capa de salida).

$$a_j^c = f \left(\sum_i w_{ji}^{c-1} a_i^{c-1} + \theta_j^c \right) = f \left(w_j^{c-1} \cdot a^{c-1} + \theta_j^c \right)$$

Con la expresión anterior obtenemos la activación de la neurona j de la capa c , teniendo en cuenta la definición que hicimos de la matriz de pesos $w^c = (w_{ji}^c)$, podemos obtener el vector de activación $a^c = (a_1^c, a_2^c, \dots, a_{n_c}^c)$ mediante la expresión siguiente:

$$a^c = f(w^{c-1}a^{c-1} + \theta^c)$$

Donde θ^c representa al vector $(\theta_1^c, \theta_2^c, \dots, \theta_{n_c}^c)$. Suponemos también que f actúa sobre el vector elemento a elemento.



La salida de la red viene dada por la activación de la capa de salida. Así tenemos que las salidas y_i será igual al valor de la activación a_i^C la cual se corresponde con la activación de la neurona i de la capa de salida.

La función f es denominada, al igual que en el perceptrón simple, función de activación. Las funciones de activación más conocidas son las siguientes.

- #### ■ Función sigmoidal:

$$\sigma : \mathbb{R} \rightarrow [0, 1], \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

- #### ■ Función tangente hiperbólica:

$$\tanh : \mathbb{R} \rightarrow [-1, 1], \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- ### ■ Función ReLU:

$$ReLU : \mathbb{R} \rightarrow [0, +\infty), \quad ReLU(x) = \max(0, x)$$

En la siguiente ilustración podemos ver una representación gráfica de estas funciones y abajo de ellas su derivada. La derivada de la función de activación jugará un papel importante como podremos ver en el siguiente apartado, donde explicaremos el algoritmo de retropropagación.

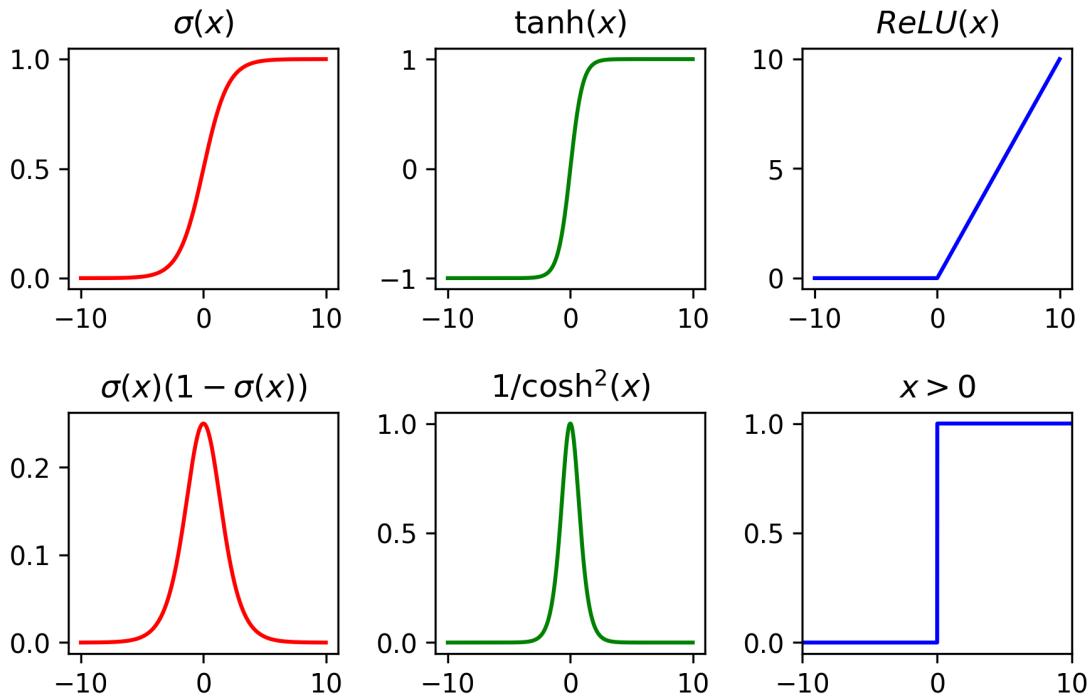


Figura 3.2: Funciones de activación

3.2.3 Algoritmo de retropropagación

El algoritmo de retropropagación se trata de un algoritmo de aprendizaje supervisado y su objetivo es que la salida de la red sea lo más parecida a los datos de entrenamiento. Este va realizando sucesivas modificaciones de los pesos y de los umbrales de la red, de forma que se minimice el error global de las salidas con las etiquetas originales.

El problema se puede formular matemáticamente como un problema de minimización, donde tenemos que encontrar los pesos y umbrales que minimicen el error.

Notación:

- N : número de muestras de entrenamiento.
- N_i : número de neuronas en la capa i .
- $s_i(n)$: salida deseada en la neurona i , para el dato de entrenamiento con índice n .
- $y_i(n, w, \theta)$: salida de la red en la neurona i , para el dato de entrenamiento con índice n , donde w el conjunto de pesos y θ el conjunto de umbrales.

- $E(w, \theta)$: función error de la red para los pesos w y umbrales θ .
- $e(n, w, \theta)$: error cometido por el dato de entrenamiento con índice n.
- C : número de capas de la red. Así tenemos que la capa C es justo la capa de salida.

Por simplificar la notación escribiremos $y_i(n)$ en lugar de $y_i(n, w, \theta)$, E en lugar de $E(w, \theta)$ y $e(n)$ en lugar de $e(n, w, \theta)$, teniendo en cuenta que son funciones de w y de θ .

Se define el error promedio de la red como:

$$E = \frac{1}{N} \sum_{n=1}^N e(n), \text{ donde } e(n) = \frac{1}{2} \sum_{i=1}^{N_C} (s_i(n) - y_i(n))^2$$

Así nos queda que el objetivo es minimizar el valor de E, teniendo que cuando el valor de este sea cercano a 0, las salidas de la red serán próximas a las salidas deseadas. De esta forma acabamos de transformar el problema de aprendizaje del perceptrón en un problema de minimizar la función error. Como las funciones de activación no son lineales, tenemos que la salida de la red es no lineal, y como consecuencia para resolver el problema de minimización tendremos que emplear técnicas no lineales.

Uno de los algoritmos para obtener mínimos de una función es el gradiente descendente en el cual dada una función F de varias variables, se parte de un punto inicial y se va avanzando en la dirección contraria del gradiente de la función para llegar hasta el mínimo. El gradiente de una función nos proporciona la dirección de máxima pendiente, entonces si vamos en la dirección contraria iremos encontrando puntos donde la función F cada vez es menor. La regla para encontrar el siguiente punto es la siguiente:

$$x_{n+1} = x_n - \eta \nabla F(x_n)$$

Si $\eta \in \mathbb{R}^+$ es suficientemente pequeño, entonces $F(x_n) \geq F(x_{n+1})$, y de esta sucesión $\{x_n\}$ acaba convergiendo a hacia un mínimo local de la función.

La misma idea del gradiente descendente es la que vamos a usar para obtener el mínimo de la función de error de la red neuronal, solo que en lugar de usar el gradiente descendente clásico, vamos a utilizar el gradiente descendente estocástico, en el cual en lugar de minimizar el error total E, se realizan sucesivas minimizaciones de los errores $e(n)$ para cada dato de entrenamiento. La regla para actualizar los pesos es la siguiente:

$$w \rightarrow w - \eta \frac{\partial e(n)}{\partial w}$$

Donde $\frac{\partial e(n)}{\partial w}$ es la derivada parcial de $e(n)$ con respecto a w B.5.

Pesos de la capa oculta $C - 1$ a la capa de salida y umbrales de la capa de salida

El peso w_{ji}^{C-1} se modifica siguiendo la regla anterior:

$$w_{ji}^{C-1} \rightarrow w_{ji}^{C-1} - \eta \frac{\partial e(n)}{\partial w_{ji}^{C-1}}$$

Evaluemos las derivada de $e(n)$ respecto de w_{ji}^{C-1} . Como el peso w_{ji}^{C-1} solo afecta a la neurona de salida j obtenemos:

$$\frac{\partial e(n)}{\partial w_{ji}^{C-1}} = -(s_j(n) - y_j(n)) \frac{\partial y_j(n)}{\partial w_{ji}^{C-1}}$$

Ahora tenemos que evaluar la derivada de $y_j(n)$ respecto de w_{ji}^{C-1} . Antes de calcular la derivada recordemos como se calcula $y_j(n)$:

$$y_j(n) = a_j^C(n) = f \left(\sum_k w_{jk}^{C-1} a_k^{C-1}(n) + \theta_j^C \right) = f(w_j^{C-1} a^{C-1}(n) + \theta_j^C)$$

En la expresión anterior podemos ver que el único lugar en donde interviene el w_{ji}^{C-1} es en $w_{ji}^{C-1} a_i^{C-1}$, de donde se deduce que la derivada se expresa como:

$$\frac{\partial y_j(n)}{\partial w_{ji}^{C-1}} = f' \left(w_j^{C-1} a^{C-1}(n) + \theta_j^C \right) a_i^{C-1}(n)$$

Se define $\delta_j^C(n)$ de la siguiente forma:

$$\delta_j^C(n) = -(s_j(n) - y_j(n)) f' \left(w_j^{C-1} a^{C-1}(n) + \theta_j^C \right)$$

De esta forma acabamos obteniendo la siguiente expresión:

$$w_{ji}^{C-1} \rightarrow w_{ji}^{C-1} - \eta \delta_j^C(n) a_i^{C-1}(n)$$

Esta regla se puede generalizar para los umbrales obteniéndose la siguiente regla:

$$\theta_j^C \rightarrow \theta_j^C - \eta \delta_j^C(n)$$

Pesos de la capa c a la capa $c+1$ y umbrales de las neuronas de la capa $c+1$, para $c = 1, 2, \dots, C-2$

De nuevo aplicamos la regla del gradiente descendente estocástico, pero empecemos eligiendo un peso de la capa $C-2$ a la capa $C-1$.

$$w_{kj}^{C-2} \rightarrow w_{kj}^{C-2} - \eta \frac{\partial e(n)}{w_{kj}^{C-2}}$$

Calculamos la derivada de $e(n)$ respecto de w_{kj}^{C-2} , teniendo ahora en cuenta, que a diferencia del caso anterior, ahora el peso w_{kj}^{C-2} afecta a todas las salidas.

$$\frac{\partial e(n)}{\partial w_{kj}^{C-2}} = - \sum_{i=1}^{N_c} (s_i(n) - y_i(n)) \frac{\partial y_i(n)}{\partial w_{kj}^{C-2}}$$

Calculamos la derivada de $y_i(n)$ con respecto de w_{kj}^{C-2} , en donde vemos que el peso afecta en la activación de la neurona $a_k^{C-1}(n)$, en la cual deberemos volver a derivar y donde el calculo de la derivada de $a_k^{C-1}(n)$ respecto de w_{kj}^{C-2} es el único que interviene en el peso es $w_{kj}^{C-2} a_j^{C-2}$.

$$\begin{aligned} \frac{\partial y_i(n)}{\partial w_{kj}^{C-2}} &= \left(f'(w_i^{C-1} a^{C-1} + \theta_i^C) w_{ik}^{C-1} \right) \frac{\partial a_k^{C-1}(n)}{\partial w_{kj}^{C-1}(n)} = \\ &= \left(f'(w_i^{C-1} a^{C-1} + \theta_i^C) w_{ik}^{C-1} \right) \left(f'(w_k^{C-2} a^{C-2} + \theta_k^{C-1}) a_j^{C-2} \right) \end{aligned}$$

Definimos ahora $\delta_k^{C-1}(n)$ de la siguiente forma:

$$\delta_k^{C-1}(n) = \sum_{i=1}^{N_c} \delta_i^C(n) w_{ik}^{C-1} f'(w_k^{C-2} a^{C-2} + \theta_k^{C-1})$$

Usando la definición de $\delta_k^{C-1}(n)$ anterior nos queda

$$w_{kj}^{C-2} \rightarrow w_{kj}^{C-2} - \eta \delta_k^{C-1}(n) a_j^{C-2}$$

Aplicando los mismos cálculos para los umbrales se llega a:

$$u_k^{C-1} \rightarrow u_k^{C-1} - \eta \delta_k^{C-1}(n)$$

Todo el proceso anterior puede repetirse en las siguientes capas, en donde cada neurona propaga hacia atrás su error o valor de δ a las neuronas ocultas de las capas anteriores. Finalmente la regla obtenida para las capas $c = 1, 2, \dots, C-2$ es la siguiente:

$$\begin{aligned}
 w_{kj}^c &\rightarrow w_{kj}^c - \eta \delta_k^{c+1}(n) a_j^c \\
 u_k^{c+1} &\rightarrow u_k^{c+1} - \eta \delta_k^{c+1}(n) \\
 \delta_k^{c+1}(n) &= f'(w_k^c a^c + \theta_k^{c+1}) \sum_{i=1}^{N_c} \delta_i^{c+2}(n) w_{ik}^{c+1}
 \end{aligned}$$

Como ya hemos dicho los errores de las neuronas de la red se retropropagan hacia todas las neuronas de la capa anterior, de ahí viene el nombre de este algoritmo.

El algoritmo de retropropagación funciona de la siguiente forma:

Algoritmo 2: Algoritmo de Retropropagación

Entrada: $(x_i, y_i), i = 1, \dots, n$

Inicializar \mathbf{w} aleatoriamente

mientras $!minimo$ **hacer**

para $i = 1, \dots, N$ **hacer**

 Obtener salida $Y(n)$ para la entrada $X(n)$

 Modificar pesos y umbrales utilizando las reglas anteriores

fin

 Evaluar error total E

fin

Como podemos ver el proceso se repite hasta que no se haya encontrado un mínimo, lo cual sucede cuando la derivada de E respecto de w es cercana a 0. También se puede fijar un número de ciclos como condición de parada.

3.3 Sobreajuste y regularización

Se conoce como subajuste en aprendizaje automático al hecho de dar un modelo demasiado simple para un problema complejo. Por otro lado tenemos el problema totalmente opuesto, en el que obtenemos un modelo que se ajusta demasiado bien a los datos con los que hemos entrenado y aunque esto puede parecer bueno, en general no lo es, a esto se le conoce como sobreajuste. El problema que ocurre con el sobreajuste es que hemos obtenido un modelo que se ajusta demasiado bien a los datos de entrenamiento, a la vez que se pierde la capacidad de generalización, esto es la capacidad de obtener un modelo que funcione bien con datos fuera del conjunto de entrenamiento. Por lo general los datos de los que disponemos no siempre son perfectos, ya que pueden presentar ruido, al intentar afinar al máximo nuestro modelo acabamos ajustándolo a estas imperfecciones.

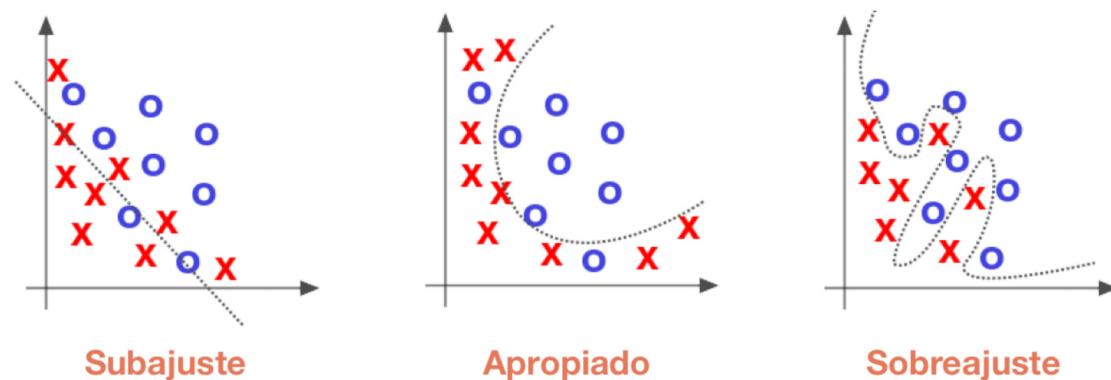


Figura 3.3: Subajuste y sobreajuste

Por lo general el problema del sobreajuste ocurre cuando el modelo es demasiado complejo. Cuando intentamos ajustar un modelo intentamos aumentar el éxito sobre las muestras de entrenamiento, y con ello aumentamos a la vez el éxito en datos fuera de la muestra de entrenamiento. Pero llega un momento en el que esto deja de ocurrir, en este momento empezamos a sobreajustar demasiado la muestra de entrenamiento y mientras que en dicha muestra el éxito sigue mejorando, vemos que en cambio el éxito fuera de la muestra comienza a empeorar. La siguiente gráfica muestra lo que acabamos de explicar.

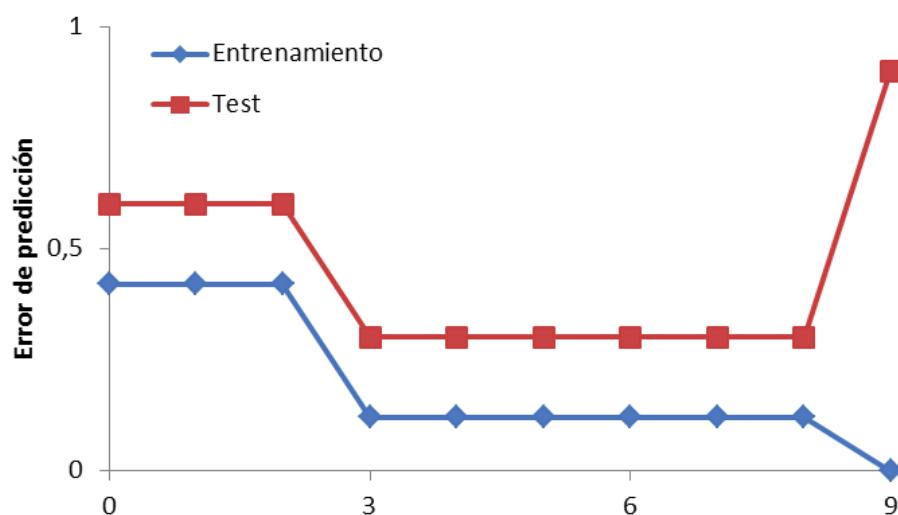


Figura 3.4: Error de predicción en el conjunto de entrenamiento y en el de test

Para remediar el problema del sobreajuste haremos caso del conocido principio de *La navaja de Ockham*, el cual nos dice que la explicación más sencilla suele ser la más probable. Para ello modificaremos la función objetivo, y ahora no solo intentaremos minimizar la pérdida, sino también la complejidad. Este principio, en el que se penalizan los modelos complejos, es denominado regularización. Ahora nuestra función objetivo será la suma de la función de perdida más la función de complejidad, quedando de la siguiente forma:

$$\text{Coste}(\text{Modelo}) = \text{Error}(\text{Modelo}) + \lambda \text{Complejidad}(\text{Modelo})$$

Donde λ es una constante que regula la importancia que se le da a la complejidad en la función de coste.

La regularización resulta en modelos más simples que tienden a generalizar mejor, normalmente a costa de un peor ajuste en los datos de entrenamiento. Para el caso de las redes neuronales reducir la complejidad del modelo es equivalente a reducir el valor de los pesos de la red.

3.3.1 Métodos de regularización

Regularización Lasso

La regularización Lasso o L1, se calcula como la media de los valores absolutos de los pesos. La función que se pretende minimizar sería la siguiente:

$$C = \frac{1}{N} \sum_{i=1}^N |w_i|$$

La regularización L1 favorece a que algunos de los coeficientes tiendan a cero, por lo que puede ser útil si alguno de los atributos de entrada de la red no es relevante, ya que al hacer cero el valor de su conexión hace que este no se acabe teniendo en cuenta en el cálculo de la salida de la red. Por lo tanto este método de regularización funciona mejor cuando utilizamos atributos que no son relevantes para el modelo.

Regularización Ridge

La regularización Ridge o L2, se calcula como la media de los cuadrados de los pesos. La función a minimizar adopta la siguiente forma:

$$C = \frac{1}{2N} \sum_{i=1}^N w_i^2$$

Este método de regularización tiende a hacer que los pesos disminuyan y con ello se consigue reducir la complejidad del modelo.

Regularización ElasticNet

La regularización ElasticNet combina las regularizaciones L1 y L2 y se calcula como una combinación convexa de ambas funciones de regularización es decir ponderándolas

con α y β , cumpliendo que $\alpha \geq 0$, $\beta \geq 0$ y $\alpha + \beta = 1$. La expresión resultante queda como:

$$C = \alpha \cdot \frac{1}{N} \sum_{i=1}^N |w_i| + \beta \cdot \frac{1}{2N} \sum_{i=1}^N w_i^2$$

Este método es efectivo cuando tenemos una gran cantidad de atributos entre los cuales puede haber algunos que no sean relevantes y variables que estén correladas entre ellas.

Capítulo

4 ALGORITMO PROPUESTO

En este capítulo procederé a explicar en detalle la propuesta que he llevado a cabo, para poder disponer de un programa capaz de realizar predicciones en el mercado de valores. Se mostrarán fragmentos de código ya que uno de los objetivos que se persigue, es que el presente trabajo pueda servir como manual para gente que quiera repetir los experimentos, o en su caso crear su propia red neuronal y validarla con Backtrader.

El algoritmo implementado se trata de una red neuronal, para un problema de clasificación binaria, la cual recibirá una entrada y devolverá como salida un valor que nos indicará la dirección de la tendencia en el precio del mercado. Para entrenar la red se necesitan datos de bolsa sobre los días cotizados de un valor concreto. Además de los datos de entrada de la red, deberemos de definir unas etiquetas sobre el conjunto de entrenamiento, con la dirección que seguirá el precio del mercado, para cada día cotizado en la bolsa. Con todo ello podremos entrenar la red neuronal, para que pueda ser utilizada para realizar predicciones sobre datos futuros.

Una vez entrenada la red neuronal, se definirá una estrategia que haga uso de esta red para tomar decisiones. Para probar el funcionamiento de la estrategia se realizará una simulación sobre un conjunto de datos perteneciente a un momento del tiempo posterior al conjunto de entrenamiento, de esta forma el conjunto test no tendrá ninguna información acerca del futuro y la simulación será más realista. Además de ir tomando decisiones durante la simulación, conforme se vaya avanzando en el tiempo se introducirán en la red los nuevos datos generados, para que esta se vaya realimentando y reentrenando con la información actual.

Antes de ver en detalle la propuesta y su implementación vamos a comentar las herramientas software que se han utilizado para el desarrollo del proyecto.

4.1 Herramientas utilizadas

El proyecto será realizado en el lenguaje de programación Python. Existen muchos motivos por los que utilizar Python para tareas de aprendizaje automático, uno de los principales es la facilidad de implementación, además de requerir en general mucho menos código que la mayoría de lenguajes de programación.

Otro punto a favor es la gran cantidad de bibliotecas de las que podemos disponer como por ejemplo Numpy o Scipy para computación científica, Scikit-learn para aprendizaje automático, Pandas para análisis de datos, etc... Por todo esto considero una buena elección el uso de Python para este trabajo.

A continuación se muestran las librerías más relevantes en el desarrollo del programa:

- **Pandas:** utilizada para análisis de datos. Esta biblioteca nos facilita el tratamiento de datos mediante los DataFrame, los cuales son estructuras similares a una tabla SQL y nos permitirá poder almacenar los datos con los que trabajaremos.
- **Numpy:** biblioteca con funciones matemáticas de alto nivel para vectores y matrices. Uno de los problemas de Python es que no es un lenguaje en el que se pueda programar de forma tan eficiente como podría hacerse en lenguajes como C++. Gracias a esta librería los cálculos matemáticos mas pesados, pueden ejecutarse de forma más eficiente y contrarrestar esta desventaja de Python.
- **Matplotlib:** biblioteca para la generación de gráficos. Nos será útil para poder representar gráficamente los resultados de las simulaciones sobre un histórico de datos.
- **Scikit-learn:** biblioteca de aprendizaje automático para Python. En nuestro caso no la emplearemos para definir redes neuronales, ya que las aportadas por esta librería son muy básicas. El motivo de utilizar esta biblioteca es que aporta funciones que serán de gran utilidad, como por ejemplo para normalizar los datos o para utilizar métricas para la validación del modelo.
- **Backtrader:** es una biblioteca creada para realizar pruebas de estrategias de trading. Gracias a esta biblioteca podremos validar nuestro algoritmo, realizando simulaciones sobre los datos y obteniendo gráficas con los resultados.
- **fix_yahoo_finance:** con esta biblioteca podremos descargar los datos diarios de un valor concreto como pueden ser por ejemplo las acciones de Apple. Los datos obtenidos contendrán información sobre el precio de apertura, de cierre, mínimo, máximo y volumen de cada día cotizado.
- **Talib:** esta biblioteca contiene funciones para el cálculo de indicadores técnicos, los cuales nos serán necesarios para aumentar el número de características de nuestro conjunto de datos y nos ayudará a aumentar la precisión de nuestra red neuronal.

- **Keras:** biblioteca que facilita la implementación de redes neuronales. Es la biblioteca principal de nuestro proyecto ya que gracias a ella podremos definir modelos neuronales de forma simple, pero a la vez muy completa ya que nos permite diseñar la estructura de nuestra red, elegir las funciones de activación de cada capa, elegir el algoritmo de actualización de los pesos, elegir funciones de optimización y otras muchas funcionalidades.
- **Tkinter:** biblioteca utilizada para la creación de una interfaz gráfica, que permita facilitar el uso del programa a los usuarios, en lugar de tener que ejecutar en consola. Con esta librería podremos crear una interfaz con campos de texto y botones.

4.2 Dataset

En este apartado pasamos a hablar sobre el conjunto de datos que utilizaremos, tanto de la obtención del mismo, como del tratamiento que realizaremos sobre los datos para utilizarlos en la red.

4.2.1 Cargar el conjunto de datos

Los datos los obtenemos gracias a la librería `fix_yahoo_finance`, mediante la cual podemos descargar datos de un determinado mercado haciendo uso de la función `fix_yahoo_finance.download_data(desde, hasta)`. Para poder descargar los datos es necesario conocer el nombre de estos. Por ejemplo para obtener las acciones del Santander, el nombre que tenemos que aportar es la abreviatura SAN.

Para facilitar la tarea de obtener los datos se ha implementado la siguiente función en Python:

Código 4.1: Obtener los datos

```
1 def getData(data_name):
2     path_data = 'data/' + data_name + '.csv'
3
4     # Comprobamos si los datos ya existen
5     if path.exists(path_data):
6         df = pd.read_csv(path_data, index_col = "Date", parse_dates = True)
7         # Si no existen los descargamos y los guardamos en la carpeta data
8     else:
9         from_date = '2000-01-01'
10        today = datetime.datetime.now()
11        today = today.strftime('%Y-%m-%d')
12        # Descargamos los datos de yahoo finance
13        df = yf.download(data_name, from_date, today)
14        df = df[['Open', 'High', 'Low', 'Close', 'Volume']]
15        df.to_csv(path_data)
16
```

```
17 return df
```

Como podemos ver en la función anterior, antes de cargar los datos, comprobamos previamente si estos ya existen en la carpeta ./data. En caso de que dichos datos no existan, pasarán a descargarse de yahoo finance para posteriormente almacenarlos en la carpeta ./data y así no tener que descargarlos de nuevo si en el futuro deseamos utilizarlos de nuevo. En el caso de la función anterior los datos se descargan desde la fecha 2000-01-01 hasta el día actual.

Una vez terminada la ejecución de la función loadData, esta nos devolverá los datos en un DataFrame, estructura similar a una tabla. Cada fila de la tabla se corresponde con un elemento del conjunto de datos, es decir de un día cotizado en el mercado de valores. La primera columna de esta tabla tiene de nombre Date, es la que contiene las fechas de los datos y gracias a la cual podremos identificar a cada elemento de la base de datos. Las columnas restantes se corresponden con la apertura, el máximo, el mínimo, el cierre y el volumen. Un ejemplo de esta tabla podría ser el siguiente, en el que podemos ver todo lo que acabamos de describir.

Date	Open	High	Low	Close	Volume
2017-12-21	1075.39	1077.52	1069.00	1070.85	1211012.0
2017-12-22	1070.00	1071.72	1067.64	1068.86	860800.0
2017-12-26	1068.64	1068.86	1058.64	1065.85	914574.0
2017-12-27	1066.60	1068.27	1058.38	1060.20	1027634.0
2017-12-28	1062.25	1064.84	1053.38	1055.95	982285.0
2017-12-29	1055.49	1058.05	1052.70	1053.40	1156357.0

4.2.2 Características adicionales obtenidas mediante indicadores técnicos

Además de los datos obtenidos con yahoo, se han añadido más características mediante el uso de indicadores técnicos.

Entre los nuevos atributos se han incluido medias móviles ya que permiten suavizar las fluctuaciones en los precios y permiten detectar mejor las tendencias tanto a corto como a largo plazo según el período de tiempo tomado. Se ha añadido la media móvil exponencial la cual responde de forma más rápida al movimiento de los precios y la media móvil simple que al igual que la exponencial suaviza pero de forma más lenta lo que puede salvarnos de falsas señales.

Se ha añadido el indicador de momento, un indicador que aporta una información muy básica pero a la vez muy importante. Este indicador es útil para ver de forma general la tendencia de los precios. Además puede servir para detectar los suelos y los techos del mercado.

Otro importante indicador utilizado para ampliar nuestra cantidad de atributos ha sido el indicador Rate of Change, gracias al cual podemos ver el cambio entre dos períodos de forma relativa, por tanto dicho indicador nos va a aportar una nueva señal de compra o venta según si su valor este por encima o por debajo de 0.

Se han añadido osciladores estocásticos, estos indicadores nos van a servir para tener información sobre el impulso del precio. Se tiene como regla general que el impulso suele cambiar de dirección antes de que lo haga el precio, así este indicador nos puede servir para anticipar posibles cambios en la tendencia del precio. Además también es capaz de identificar zonas de sobreventa y sobrecompra. Por todos estos motivos se ha considerado como un buen atributo a incluir.

El índice de fuerza relativa también se ha tenido en cuenta como atributo para nuestro dataset y al igual que los osciladores estocásticos permite encontrar zonas de sobreventa y sobrecompra y con ello determinar si la tendencia va a cambiar de dirección.

El indicador MACD al igual que los anteriores detecta zonas de sobreventa y sobrecompra y además también es capaz de identificar tendencias, por lo que es un indicador bastante completo que nos será de gran utilidad como atributo a añadir en la base de datos.

Se ha utilizado también la desviación típica, indicador que nos da una medida de la volatilidad, es decir una medida de que tan dispersos están los datos. Con este indicador se puede tener una medida del riesgo de inversión, siendo el riesgo alto cuando la volatilidad es elevada.

Para todos los indicadores anteriores se han fijado como parámetros aquellos valores más utilizados en el trading.

En total se han añadido 31 nuevos atributos. Para ello hemos utilizados los siguientes indicadores técnicos.

- Momento de 5, 10 y 15 días.
- Media móvil simple de 7, 14 y 21 días.
- Media móvil exponencial de 7, 14 y 21 días.
- Rate of change de 13 y 21 días.
- Oscilador estocástico (%K, %D) de 7, 14 y 21 días.
- Oscilador estocástico rápido (%K, %D) de 7, 14 y 21 días.
- MACD e histograma, con diferencia de medias móviles de 12 y 26 períodos.
- Índice de fuerza relativa de 9, 14 y 21 días.
- Desviación típica de 7, 14 y 21 días.

Estos 31 atributos más los que ya teníamos (apertura, cierre, mínimo, máximo y volumen), nos dan un total de 36 características, las cuáles se corresponden con las

entradas que tendrá nuestra red neuronal. Por lo tanto la capa de entrada de nuestra red tendrá un total de 36 nodos.

Veamos un ejemplo de como añadir las características correspondientes a la media móvil simple y la media móvil exponencial, haciendo uso de la biblioteca `talib`. Para facilitar el uso, creamos dos funciones:

Código 4.2: Ejemplo medias móviles

```

1 import talib as ta
2
3 # Media móvil
4 def moving_average(df, n):
5     df['MA_'] + str(n)] = ta.MA(df['Close'], timeperiod=n)
6     return df
7
8 # Media exponencial
9 def exponential_moving_average(df, n):
10    df['EMA_'] + str(n)] = ta.EMA(df['Close'], timeperiod=n)
11    return df

```

Ahora para añadir las características solo tendríamos que hacer uso de las funciones anteriores a las que debemos de pasar como parámetros el dataframe y el número de días correspondiente al período de tiempo que se va a considerar para hacer la media.

Código 4.3: Medias móviles

```

1 # Media Móvil
2 df = moving_average(df, 3)
3 df = moving_average(df, 13)
4 df = moving_average(df, 21)
5
6 # Media Exponencial
7 df = exponential_moving_average(df, 5)
8 df = exponential_moving_average(df, 13)
9 df = exponential_moving_average(df, 21)

```

Ahora ya tenemos todos los atributos de nuestro dataset, pero nos falta algo muy importante y es normalizar los datos. En las redes neuronales es fundamental la normalización de los datos, pues cada atributo tendrá su propio rango de valores, lo cual hace que sea imposible comparar valores de diferentes órdenes de magnitud. Si no se normalizan los datos, la modificación de los pesos de la red durante el entrenamiento, tendrá una mayor influencia sobre aquellas variables con magnitudes más grandes, haciendo que estas dominen sobre la función objetivo y haciendo que no se pueda aprender de otras características correctamente.

Notar que es necesario que se separen los datos en `train` y `test` antes de realizar la normalización, ya que si normalizamos todos los datos juntos, los datos del `train` tomarían cierta información del `test`, es decir tomarían información del futuro y estaríamos haciendo trampas durante el entrenamiento.

Se ha usado en este trabajo la normalización Z-score, también llamada estandariza-

ción, la cual consigue que los datos tengan media 0 y desviación típica de 1. Se calcula de la siguiente forma:

$$x' = \frac{x - \bar{x}}{\sigma}$$

Este cálculo se puede conseguir fácilmente mediante la clase StandardScaler de sklearn. En el siguiente código se muestra su uso:

Código 4.4: Estandarizar los datos

```

1 from sklearn.preprocessing import StandardScaler
2
3 sc = StandardScaler()
4 X_train = sc.fit_transform(X_train)
5 X_test = sc.fit_transform(X_test)
```

4.2.3 Añadiendo las etiquetas al Dataset

El objetivo de nuestra red es que dada una entrada, esta nos devuelva una salida. Para poder hacer esto, deberá entrenar con unos datos que dispongan de la salida deseada, de cada elemento del conjunto de entrenamiento, para de esta forma poder modificar los pesos de la red de tal manera que las salidas de la red, para los datos de entrenamiento, sean similares a las salidas deseadas. Para ello deberemos añadir a los datos, además de los atributos que hemos añadido anteriormente, una etiqueta con dicha salida deseada.

En nuestro caso la red neuronal no tendrá como objetivo predecir el valor de la acción, sino determinar cuál es la mejor acción a realizar en cada momento, considerando 2 acciones posibles: vender y comprar, que identificaremos como 0 y +1.

Ahora el problema al que nos enfrentamos es el de elegir una forma objetiva de asignar estos valores. ¿Cómo podemos determinar cuál es la mejor acción a tomar en un momento dado? Para tomar la decisión de que acción asignar como etiqueta a cada día del conjunto de datos disponibles, se realizará una simulación durante un período de d días y se asignará la acción que resulte ser más beneficiosa.

Para realizar la simulación consideraremos los siguientes parámetros: g (gain) que se corresponde con la máxima ganancia permitida, l (loss) es el límite de pérdida que se puede tolerar, c (comission x 2) es la comisión de cada operación multiplicada por 2, d es el número máximo de días de duración de la simulación. Notar que en este trabajo consideraremos un porcentaje de comisión fijo, es decir el porcentaje de comisión no será variable y no dependerá del importe de la operación.

Teniendo en cuenta los parámetros mencionados, la forma de asignar la acción será la siguiente.

- Se considerará que es un buen momento de compra si se supera el parámetro fijado $g(gain)$ antes de transcurrir d días desde el día en el que comenzó la simulación

o si tras pasar el período máximo de días de simulación la ganancia se encuentra por encima de c .

- Se considera que es un momento de venta si antes de transcurrir d días se supera el límite de pérdida l o si tras pasar el período máximo de días de simulación la ganancia se encuentra por debajo de c .

A continuación se muestra un ejemplo gráfico de lo explicado, en el que se asigna una acción a un día t . En la primera ilustración se asignará la etiqueta +1 (comprar), ya que llega a g antes de transcurrir d días. En cambio, en la segunda ilustración se asignará la etiqueta 0 (vender), porque durante los d días transcurridos no se ha superado en ningún momento ninguno de los dos límites marcados g y l , y tampoco se encuentra la ganancia por encima de c tras finalizar el período de tiempo de la simulación.

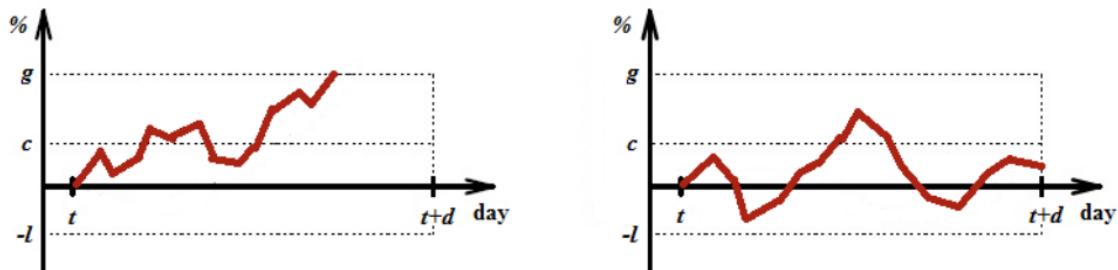


Figura 4.1: Ejemplos simulaciones

Como podemos ver la asignación de la etiqueta depende de 4 parámetros, que son la ganancia, la pérdida, el costo de cada operación y el período de tiempo fijado para hacer la simulación. El coste de la operación en el caso de este trabajo vamos a suponerlo fijo, con una comisión del 1 %. En el caso de los parámetros restantes se realizarán varias simulaciones con diferentes parámetros, para determinar en cada caso cuales son los valores óptimos.

En el siguiente código se muestra como se ha implementado este procedimiento para asignar una acción. La forma de hacerlo es recorrer todos los días de la base de datos y realizar la simulación correspondiente por cada uno de estos días. Tener en cuenta que si la simulación es de por ejemplo 10 días, los últimos 10 días de la base de datos no podrán disponer de estos 10 días posteriores para simular, debido a ello se ha optado por asignar la etiqueta 0 (Vender), para estos datos.

Código 4.5: Función para definir la clase

```

1 def add_label(df, gain, loss ,n_day, commission):
2     df_closes = [df.iloc[i]['Close'] for i in range(len(df.index))]
3     labels = []
4
5     # Recorremos cada día del histórico de datos
6     for i in range(len(df.index)-n_day):
7         close_price = df_closes[i]
8         # Por cada día simulamos durante n_day días
9         for j in range(i+1, i+1+n_day):

```

```

10     dif = (df_closes[j] - close_price)/close_price
11     # Terminar simulación si se cruza alguno de los límites
12     if dif > gain or dif < -loss:
13         break
14
15     action = 1 if dif > commission*2 else 0
16     labels.append(action)
17
18     labels.extend([0 for x in range(n_day)])
19     df['label'] = labels
20
21     return df

```

Con el código anterior ya tendríamos asignada una acción a cada día concreto. En este caso nuestra red neuronal se tendrá que enfrentar a un problema de clasificación binaria, en el que los posibles valores son el 0 y el 1.



Figura 4.2: Asignación de etiquetas SAN

La gráfica anterior muestra las asignaciones de las etiquetas para las acciones del Banco Santander donde se han puesto como parámetros $gain = 0.05$, $loss = 0.03$, $d = 10$ y una comisión del 1 %. Las líneas verdes son las correspondientes a la etiqueta +1 (Comprar) y las líneas rojas a la etiqueta 0 (Vender).

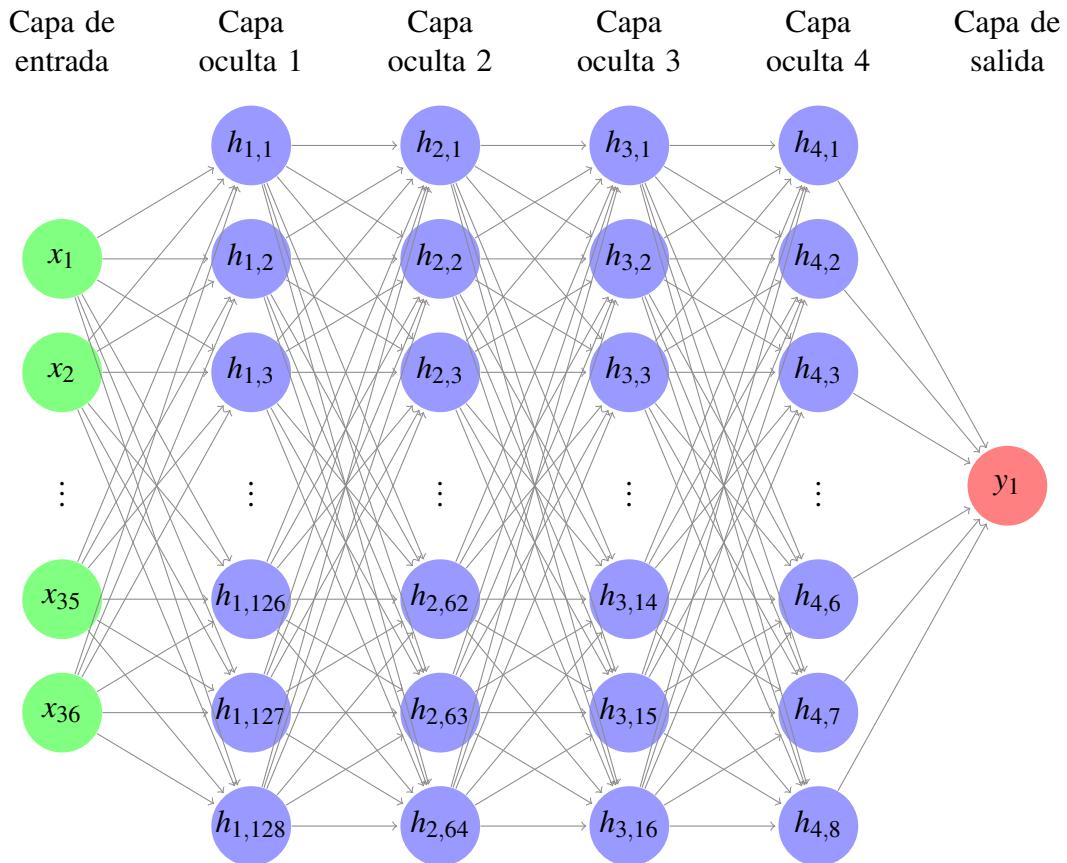
4.3 Diseño de la Red Neuronal

En este apartado pasamos a describir en detalle la red neuronal implementada, la cual usaremos para realizar predicciones y será la pieza principal para ejecutar nuestra

estrategia de compra-venta.

La red neuronal implementada se compone de 6 capas. La primera capa se corresponde con la capa de entrada, esta tiene un total de 36 nodos, uno por cada característica de cada elemento del dataset. Estas características se corresponden con las explicadas anteriormente, es decir los atributos obtenidos con yahoo finance, más los indicadores técnicos incluidos. La última capa es la capa de salida y como ya se ha comentado anteriormente tiene un único nodo. Las capas internas se corresponden con las capas ocultas de la red y tiene 128, 64, 16 y 8 nodos respectivamente.

En la siguiente ilustración podemos ver graficamente la arquitectura descrita.



La función de activación por la que se ha optado en los nodos de las capas ocultas, es la función $ReLU(x) = \max(0, x)$, debido a que es computacionalmente más eficiente de calcular que las funciones sigmoidales, por lo que tanto el tiempo de entrenamiento, como el tiempo en realizar las predicciones será bastante menor que si se usará una función sigmoidal. La capa de salida tiene como función de activación la función tanh. El algoritmo de optimización utilizado para encontrar los pesos que minimicen el error de salida en el conjunto de entrenamiento, es el algoritmo del gradiente descendente estocástico, conocido por sus siglas en inglés como SGD, con una razón de aprendizaje

$$\eta = 0.001.$$

La función objetivo que vamos a minimizar es la función de error cuadrático medio, que se define como la media del cuadrado de los errores.

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

Además se ha aplicado el método de regularización L2, con el cual se reducen los valores de los pesos de la red, reduciendo la complejidad del modelo y de esta forma evitar perder la capacidad de generalización.

Para este estudio se ha implementado una clase llamada NeuralNetwork, con la cual se facilitará la construcción del modelo y el uso de este al implementar algunas funciones que nos serán útiles.

Código 4.6: Clase NeuralNetwork

```

1 class NeuralNetwork():
2
3     def __init__(self):
4         self.model = Sequential()
5         self.memory_x = []
6         self.memory_y = []

```

En el código anterior podemos ver el constructor de la clase. La variable `model` va a contener el modelo de la red neuronal, esta se ha definido como `Sequential()`, esto indica que se van a añadir las capas de la red de forma secuencial. Además se han definido dos vectores que utilizaremos como colas, y en ellos almacenaremos una memoria que nos servirá para reentrenar la red neuronal durante la simulación.

La siguiente función será la que utilicemos para construir el modelo.

Código 4.7: Construcción del modelo con keras

```

1 def build_model(self, input_shape):
2     # Capa oculta 1
3     self.model.add(Dense(128, activation='relu', input_shape = input_shape, kernel_regularizer=←
4                         ↪ =l2(0.001), bias_regularizer=l2(0.001)))
5     # Capa oculta 2
6     self.model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.001), bias_regularizer=←
7                         ↪ l2(0.001)))
8     # Capa oculta 3
9     self.model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.001), bias_regularizer=←
10                        ↪ l2(0.001)))
11    # Capa de salida
12    self.model.add(Dense(1, activation='tanh', kernel_regularizer=l2(0.001), bias_regularizer=←
13                        ↪ l2(0.001)))

```

```

13
14     sgd = SGD(lr=0.001, decay=1e-6, momentum=0.5, nesterov=True)
15     self.model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['accuracy'])

```

Se comienza añadiendo al modelo las capas de la red neuronal una por una, donde Dense significa que los nodos de dicha capa van a ser nodos que implementan la operación $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$. Por último se utiliza la función `compile`, con la cual configuramos el modelo de entrenamiento, en este caso se ha elegido como optimizador el gradiente descendente estocástico (`sgd`) y la función de pérdida `mean_squared_error`.

Entrenar el modelo es tan sencillo como utilizar la función `fit` de keras, con una sola línea de código nuestra red aprenderá de los datos de entrenamiento, utilizando los parámetros especificados en `compile`. En nuestra clase `NeuralNetwork` se ha definido la siguiente función para el entrenamiento:

Código 4.8: Entrenar la red neuronal

```

1 def train(self, x, y, epochs, verbose=0):
2     self.model.fit(x, y, epochs=epochs, verbose=verbose)

```

Para inicializar y actualizar la memoria para el reentrenamiento se han implementado las siguientes funciones:

Código 4.9: Memoria de la red

```

1 def init_memory(self, X, y):
2     for i in range(len(X)):
3         self.memory_x.append(X[i])
4         self.memory_y.append(y[i])
5
6 def update_memory(self, data, target):
7     self.memory_x.insert(0, data)
8     self.memory_x.pop()
9     self.memory_y.insert(0, target)
10    self.memory_y.pop()

```

La memoria funciona como una cola, para actualizarla sacamos de la memoria el dato más antiguo e insertamos el nuevo en ella.

Para el reentrenamiento tenemos la siguiente función, que entrena la red con los datos de la memoria durante 5 iteraciones.

Código 4.10: Función para realimentar la red neuronal

```

1 def reTrain(self, batch_size=None):
2     if batch_size == None:
3         batch_size = len(np.array(self.memory_x))
4
5     X = np.array(self.memory_x)
6     y = np.array(self.memory_y)
7
8     self.model.fit(X, y, epochs=5, batch_size=batch_size, verbose=0)

```

4.4 Estrategía

En esta sección se describe la estrategia implementada mediante señales de compra-venta obtenidas por la red neuronal.

La estrategia seguida es muy simple, y se basa en ejecutar ordenes de compra y venta, en las que compramos todas la acciones que nuestro capital nos permita y para el caso de la venta vendemos todas las acciones que teníamos compradas. Las acciones se compran y venden al precio de cierre de cada día.

Las señales de compraventa son las siguientes:

- **Compra.** Se comprarán todas la acciones que se puedan con el capital del que se dispone, solo si no se tiene ninguna acción ya comprada y si además la salida de la red neuronal devuelve un valor superior a 0.6.
- **Venta.** Se venderán todas las acciones de las que se dispone, cuando la salida de la red devuelva un valor inferior a 0.4.

Cada día se introduce un dato nuevo en la memoria de la red neuronal. Dicho dato se corresponde con el dato más reciente del cual se puede obtener el valor de la etiqueta mediante la función `add_label` definida anteriormente. A la vez que entra un nuevo dato en la memoria de la red, sale el dato más antiguo. Tras la actualización de la memoria de la red, se reentrena y de esta forma nuestra red neuronal siempre estará lo más actualizada posible con respecto a los nuevos datos que van surgiendo.

Código 4.11: Estrategia con la red neuronal

```
1 class NeuralNetworkStrategy(bt.Strategy):
2     X_test = None; model = None; n_day = None
3
4     def __init__(self):
5         self.dataclose = self.datas[0].close
6
7     def next(self):
8         # Realizamos la predicción
9         p = self.model.predict(self.X_test[len(self)-1])[0][0]
10
11     if not self.position: # Si no tenemos acciones compradas
12         if p > 0.6:
13             buy_size = self.broker.get_cash() / self.dataclose
14             self.buy(size = buy_size)
15     else: # En caso de tener acciones compradas
16         if p < 0.4:
17             sell_size = self.broker.getposition(data = self.datas[0]).size
18             self.sell(size = sell_size)
19
20     # Realimentar red neuronal
```

```

21     if len(self)-1 >= self.n_day:
22         features = self.X_test[len(self)-self.n_day-1]
23         target = self.y_test[len(self)-self.n_day-1]
24         self.model.update_memory(features, target) # Actualizamos la memoria
25         self.model.reTrain() # Reentrenamos el modelo

```

4.5 Ejecutando la simulación de la estrategia

En este apartado vemos como se procede a realizar la simulación haciendo uso de todo lo explicado en las secciones anteriores.

Lo primero es la obtención de los datos, añadir la características que se deseen y añadir las etiquetas. En mi caso he creado un fichero llamado `func_utils.py`, para recoger todas estas funciones.

Código 4.12: Obtención de datos y adición de características

```

1 # Obtenemos los datos
2 df = func_utils.getData(data_name)
3
4 # Atributos y etiquetas
5 df = func_utils.add_features(df)
6 df = func_utils.add_label(df, gain = gain, loss = loss, n_day = n_day, commission = commission)

```

Lo siguiente que hacemos es separar los datos en train y test para poder entrenar la red neuronal con el conjunto de entrenamiento y después poder validar la estrategia con el de test. Para ello he creado la siguiente función para separar dichos conjuntos mediante fechas dadas.

Código 4.13: Separar en train y test

```

1 def split_df_date(df, start_train_date, end_train_date, start_test_date, end_test_date):
2     start_date = datetime.datetime.strptime(str(df.index.date[0]), '%Y-%m-%d')
3     end_date = datetime.datetime.strptime(str(df.index.date[len(df.index)-1]), '%Y-%m-%d')
4
5     df = df[start_train_date:end_test_date]
6     df_train, df_test = df[start_train_date:end_train_date], df[start_test_date:end_test_date]
7
8     train_size = df_train.shape[0]
9     test_size = df_test.shape[0]
10
11    # Definimos la matriz de características excluyendo la etiqueta y el vector de etiquetas
12    X = np.array(df.drop(['label'], 1))
13    y = np.array(df['label'])
14
15    # Separamos en train y test
16    X_train, X_test = X[0:train_size], X[len(X)-test_size:len(X)]

```

```

17     y_train, y_test = y[0:train_size], y[len(X)-test_size:len(X)]
18
19     return df_train, df_test, X_train, X_test, y_train, y_test

```

Una vez hemos separado los conjuntos en train y test, podemos pasar a normalizarlos de forma separada, utilizando por ejemplo la normalización Z-Score. Tras normalizarlos se ponen los datos en un formato correcto para la entrada de la red en Keras.

Código 4.14: Normalizar los datos

```

1 sc = StandardScaler()
2 X_train = sc.fit_transform(X_train)
3 X_test = sc.fit_transform (X_test)
4
5 # Ponemos los datos en formato correcto para usarlos en keras
6 X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
7 X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

```

Teniendo los datos separados en train y test y además normalizados, ya podemos pasar a crear una instancia de la red neuronal, construir el modelo y entrenarlo con los datos de entrenamiento. Además podemos inicializar la memoria de la red para el reentrenamiento durante la estrategia.

Código 4.15: Crear el modelo y entrenar la red

```

1 neural_network = model.NeuralNetwork()
2 neural_network.build_model(input_shape = (X_train.shape[1], 1))
3 neural_network.train(X_train, y_train, epochs = epochs)
4
5 # Inicializamos la memoria de la red neuronal
6 neural_network.init_memory(X_train[len(X_train)-15:len(X_train)], y_train[len(y_train)-15:len(y_train)])

```

Ya entrenado el modelo de la red neuronal, podemos crear una instancia de la estrategia NeuralNetworkStrategy que se definió en el apartado anterior. En nuestro se ha creado un fichero llamado strategies.py en el cual se encuentran todas las estrategias definidas, debido a ello, en el código mostrado a continuación accedemos a la estrategia a partir de dicho fichero. Una vez creada la instancia de la estrategia debemos añadirle el modelo entrenado. Además es necesario aportar los datos del test, los cuales se utilizarán para el reentrenamiento.

Código 4.16: Inicializar la estrategia

```

1 # Creamos una instancia de la clase NeuralNetworkStrategy
2 NN_Strategy = strategies.NeuralNetworkStrategy
3
4 # Añadimos los datos del test
5 NN_Strategy.X_test = X_test
6 NN_Strategy.y_test = y_test
7 NN_Strategy.n_day = n_day
8
9 # Añadimos el modelo entrenado

```

```
10 NN_Strategy.model = neural_network
```

Por último podemos crear la instancia cerebro, a la cual le debemos de pasar la estrategia, los datos para la simulación, el dinero inicial y el coste de la comisión de las operaciones. Por último se ejecuta la estrategia con el comando `cerebro.run()` tras lo cual podemos mostrar la gráfica con los resultados con la función `plot()`.

Código 4.17: Crear el cerebro y ejecutar la estrategia

```
1 # Creamos la instancia cerebro
2 cerebro = myCerebro.MyCerebro()
3
4 # Añadimos la estrategia al cerebro
5 cerebro.addstrategy(NN_Strategy)
6
7 # Añadimos los datos al cerebro
8 data = bt.feeds.PandasData(dataname = df_test)
9 cerebro.adddata(data)
10
11 # Fijamos el dinero inicial y la comisión
12 cerebro.broker.setcash(6000.0)
13 cerebro.broker.setcommission(commission=0.01)
14
15 # Ejecutamos la estrategia sobre los datos del test
16 cerebro.run()
17
18 # Mostramos los resultados
19 cerebro.plot()
```

Capítulo

5

CASOS DE ESTUDIO

5.1 Indicadores

Para los estudios llevados a cabo en este capítulo se han tenido en cuenta los siguientes indicadores para el análisis de las simulaciones realizadas sobre los datos de validación.

- **Beneficio Neto** (Beneficio): diferencia entre el dinero final y el dinero inicial.
- **Máximo DrawDown (%)** (Max DD (%)): es un indicador de riesgo. El Drawdown mide la reducción del capital tras una serie de operaciones perdedoras y se calcula como la diferencia entre un máximo relativo de capital menos un mínimo relativo. El máximo DrawDown es el máximo de los DrawDown calculados.
- **Operaciones totales** (Oper. total): número de operaciones llevadas a cabo, donde se entiende por operación la compra de un valor junto a la posterior venta de este.
- **Operaciones positivas** (Oper. +): número de operaciones que han resultado beneficiosas, es decir aquellas en las que hemos obtenido un beneficio tras la venta de las acciones.
- **Operaciones negativas** (Oper. -): número de operaciones que han supuesto una perdida.
- **Promedio de cada operación** (P. Oper.): promedio de beneficio de todas las operaciones, se calcula como la suma del porcentaje de beneficio obtenido en cada operación dividido entre el número total de operaciones.
- **Promedio de las operaciones positivas** (P. Oper. +): promedio de beneficio de las operaciones con resultado positivo, se calcula como la suma del porcentaje de beneficio obtenido en las operaciones positivas dividido entre el número de operaciones positivas.

- **Promedio de las operaciones negativas** (P. Oper. -): análogo al promedio de operaciones positivas.
- **Promedio positivas / Promedio negativas (+/-)**: este estadístico compara el promedio de operaciones positivas y negativas. Si por ejemplo el resultado es un valor menor que 1, significa que para obtener ganancias se necesitan hacer más operaciones positivas que negativas.

5.2 Resultados

5.2.1 Nasdaq-100 y Eurostoxx

Los períodos de tiempo seleccionados para el entrenamiento y la validación son los mismos que los considerados en el artículo *Fuzzy modeling of stock trading with fuzzy candlesticks*. El período de entrenamiento comprende el intervalo de tiempo desde el 22-Dic-2009 hasta el 21-Dic-2011. El periodo de validación con el cual realizaremos la simulación con Backtrader comprende el rango de tiempo desde 22-Dic-2011 hasta el 22-Dic-2013. La cartera de inversión para cada simulación, ha tenido un valor inicial de 6.000 dolares, y el coste de cada operación realizada se ha fijado a 1 %.

Los datos sobre los que se realizarán las validaciones se corresponden con valores pertenecientes al Nasdaq-100 y al Eurostoxx

Se mostrarán los resultados obtenidos mediante 4 estrategias diferentes.

- **Comprar y mantener.** La primera estrategia y la más básica se corresponde con comprar y mantener. Esta estrategia puede resultar la mejor cuando el mercado esta alcista, pero dará perdidas en épocas bajistas.
- **Estrategia clásica.** Esta estrategia está basada en señales de compra y venta mediante indicadores técnicos (se puede consultar en el anexo).
- **Redes neuronales.** Estrategia ya explicada en el capítulo anterior. Se basa en comprar y vender según las señales obtenidas por la red neuronal que se ha implementado.
- **Lógica difusa.** Esta estrategia se trata de la estrategia seguida en el artículo *Fuzzy modeling of stock trading with fuzzy candlesticks*, la cual realiza previsiones de patrones de velas mediante lógica difusa.

La estrategia clásica se ha tomado consiste en las dos siguientes señales de compra y venta:

- **Señal de compra:** cuando la media móvil exponencial de 9 días supere a la media móvil exponencial de 14 días y además el valor del RSI cruce el nivel 70 hacia abajo.

- **Señal de venta:** cuando la media móvil exponencial de 14 días supere a la media móvil exponencial de 9 días y además el valor del RSI cruce el nivel de 30 hacia arriba.

Código 5.1: Estrategia clásica

```

1 class ClassicStrategy(bt.Strategy):
2
3     def __init__(self):
4         self.dataclose = self.datas[0].close
5         self.ema_9 = bt.indicators.EMA(self.datas[0], period=9)
6         self.ema_14 = bt.indicators.EMA(self.datas[0], period=14)
7         self.rsi_14 = bt.indicators.RSIEMA(self.datas[0], period = 14)
8
9     def next(self):
10        if not self.position:
11            if self.ema_9[0] > self.ema_14[0] and self.rsi_14[-1] > 70 and self.rsi_14[0] <= 70:
12                buy_size = self.broker.get_cash() / self.datas[0].open
13                self.buy(size = buy_size)
14        else:
15            if self.ema_9[0] < self.ema_14[0] and self.rsi_14[-1] < 30 and self.rsi_14[0] >= 30:
16                sell_size = self.broker.getposition(data = self.datas[0]).size
17                self.sell(size = sell_size)

```

Los parámetros fijados para la asignación de las etiquetas al conjunto de entrenamiento son los siguientes:

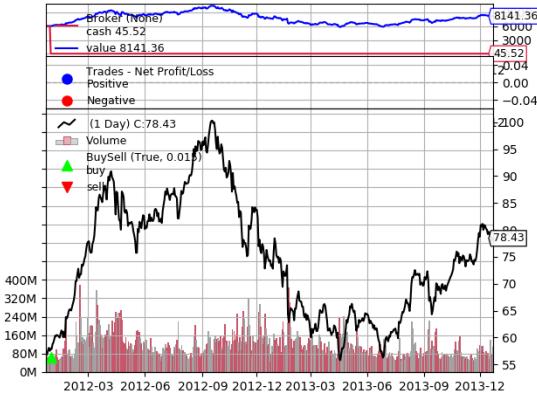
- Ganancia (g): 7 %
- Pérdida (l): 3 %
- Comisión x 2 (c): 1 %
- Días de simulación (d): 10

El número de épocas que se ha fijado para el entrenamiento de la red es de 300.

Nasdaq-100

Los 9 valores considerados para el Nasdaq-100 son: Apple (AAPL), Adobe (ADBE), ADP (ADP), Autodesk (ADSK), Akamai Technologies (AKAM), Alexion Pharmaceuticals (ALXN), Applied Materials (AMAT), Amgen (AMGN) y Amazon (AMZN).

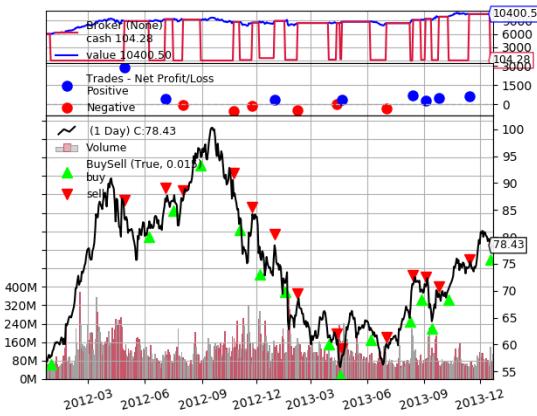
Las siguientes gráficas muestran el resultado de la simulación. La gráfica principal se corresponde con el precio de cierre de cada día cotizado en la bolsa. Los triángulos verdes y rojos señalan respectivamente las compras y ventas llevadas a cabo. Los círculos azules y rojos que hay sobre cada venta indica si la operación fue positiva o negativa. Las gráficas de la parte superior de la ilustración se corresponden con el dinero en efectivo en cartera y el valor del capital. Se muestra también una gráfica donde se comparan las ganancias de comprar y mantener, la estrategia clásica y la estrategia con redes neuronales.

AAPL

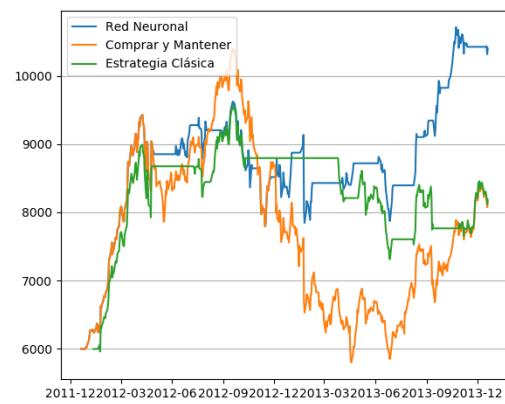
AAPL, estrategia: comprar y mantener



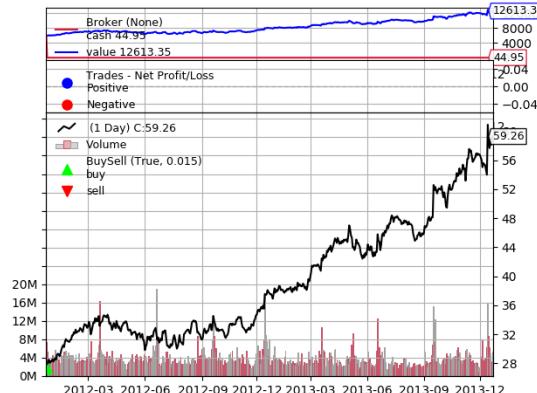
AAPL, estrategia: clásica



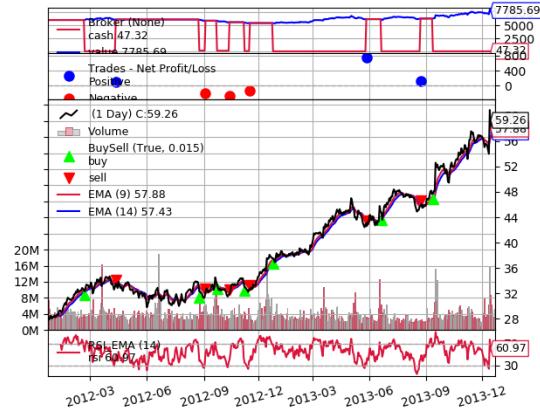
AAPL, estrategia: red neuronal



AAPL ganancias

ADBE

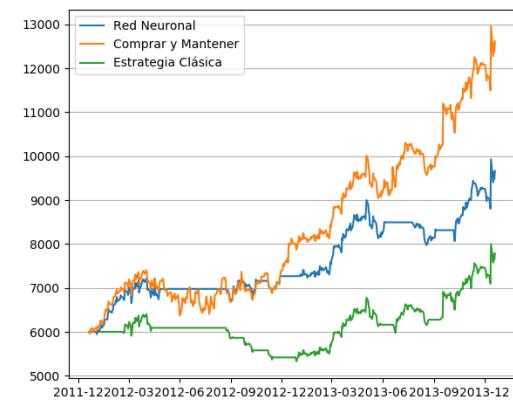
ADBE, estrategia: comprar y mantener



ADBE, estrategia: clásica

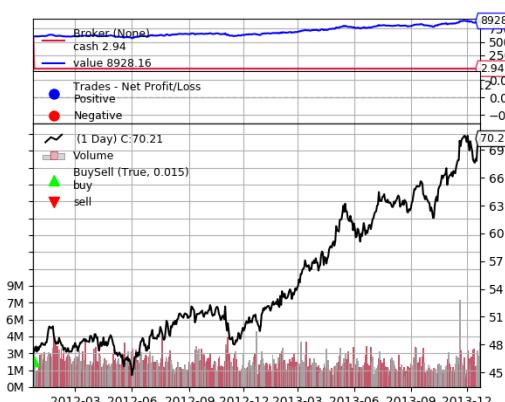


ADBE, estrategia: red neuronal



ADBE ganancias

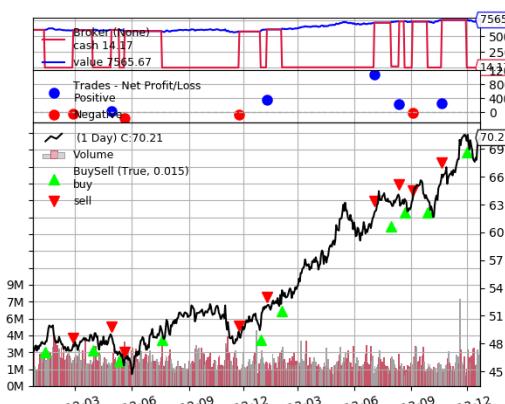
ADP



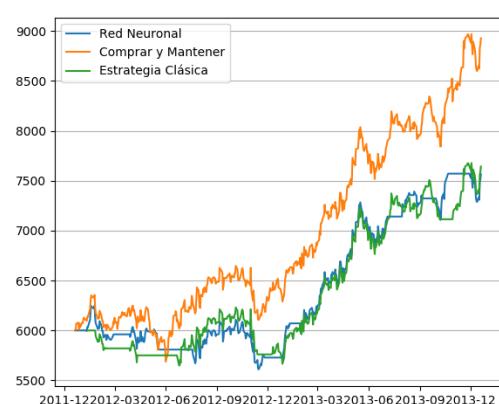
ADP, estrategia: comprar y mantener



ADP, estrategia: clásica



ADP, estrategia: red neuronal



ADP ganancias

5.2. Resultados

ADSK



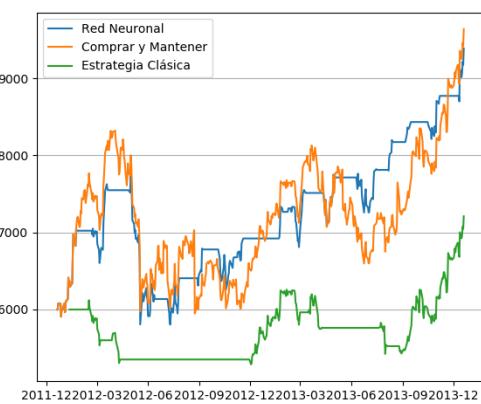
ADSK, estrategia: comprar y mantener



ADSK, estrategia: clásica



ADSK, estrategia: red neuronal



ADSK ganancias

AKAM



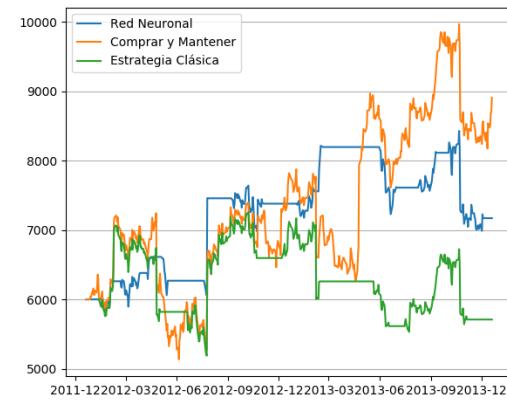
AKAM, estrategia: comprar y mantener



AKAM, estrategia: clásica



AKAM, estrategia: red neuronal



AKAM ganancias

ALXN

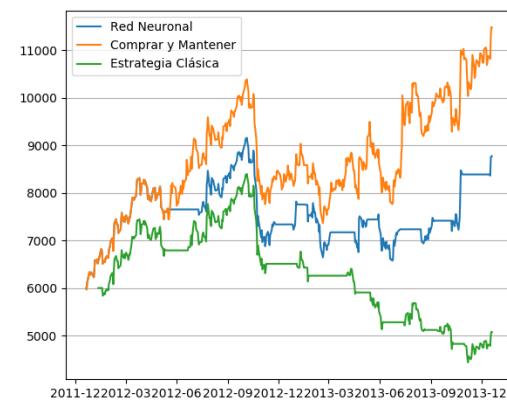
ALXN, estrategia: comprar y mantener



ALXN, estrategia: clásica



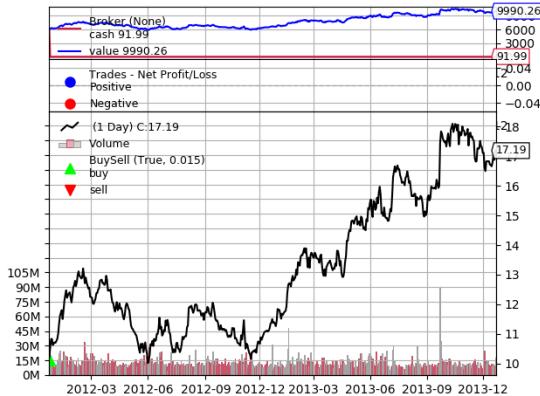
ALXN, estrategia: red neuronal



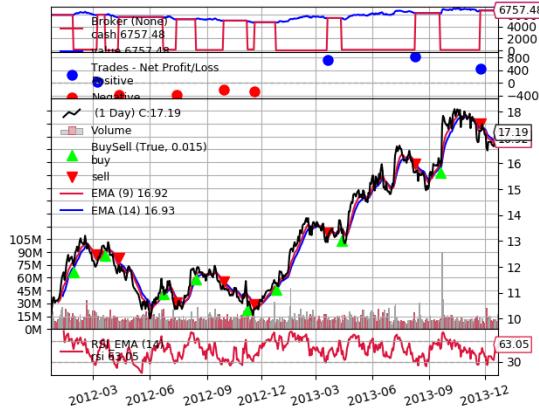
ALXN ganancias

5.2. Resultados

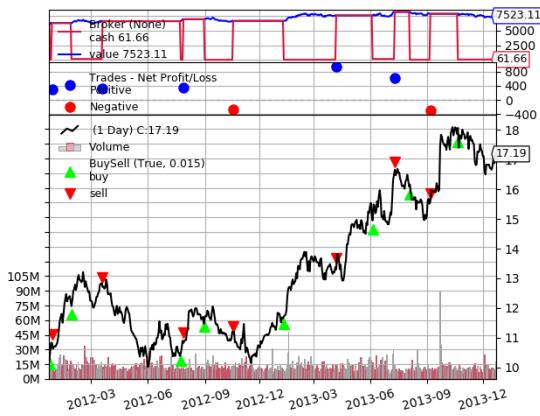
AMAT



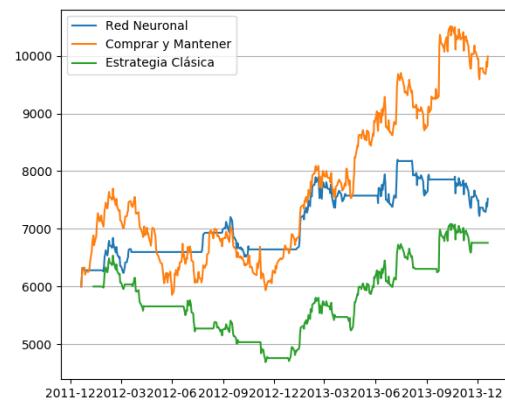
AMAT, estrategia: comprar y mantener



AMAT, estrategia: clásica

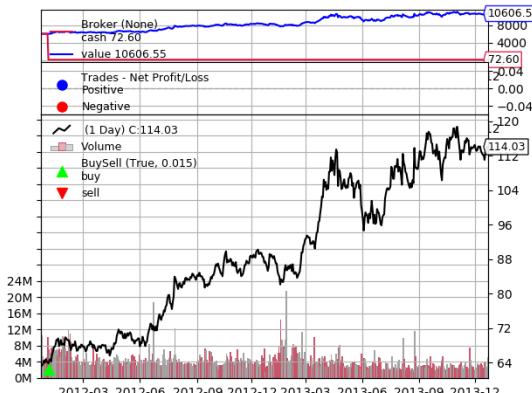


AMAT, estrategia: red neuronal



AMAT ganancias

AMGN



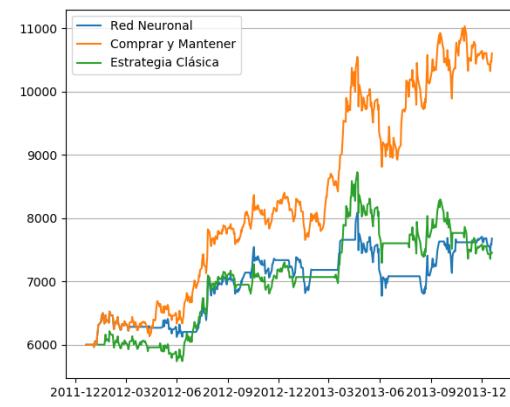
AMGN, estrategia: comprar y mantener



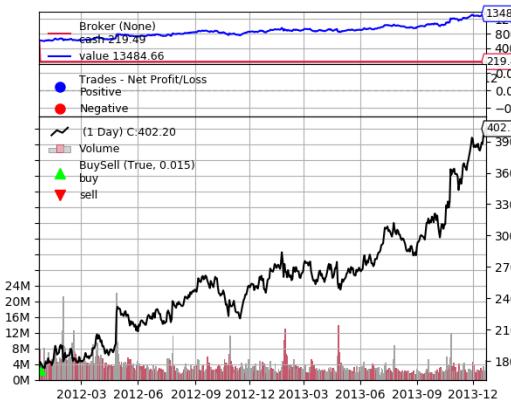
AMGN, estrategia: clásica



AMGN, estrategia: red neuronal



AMGN ganancias

AMZN

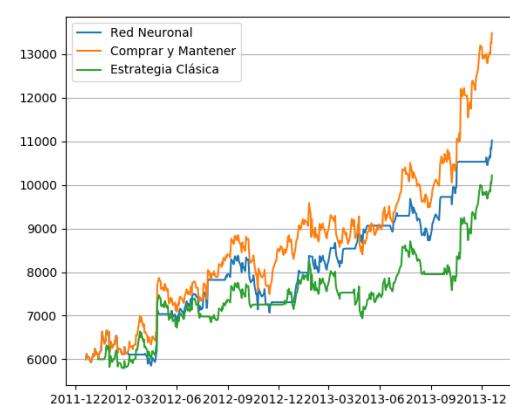
AMZN, estrategia: comprar y mantener



AMZN, estrategia: clásica



AMZN, estrategia: red neuronal



AMZN ganancias

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
AAPL	2141.36	-44.18	0	0	0	0.00	0.00	0.00	Nan
ADBE	6613.35	-13.98	0	0	0	0.00	0.00	0.00	Nan
ADP	2928.16	-10.54	0	0	0	0.00	0.00	0.00	Nan
ADSK	3637.47	-28.51	0	0	0	0.00	0.00	0.00	Nan
AKAM	2908.37	-29.08	0	0	0	0.00	0.00	0.00	Nan
ALXN	5477.06	-29.12	0	0	0	0.00	0.00	0.00	Nan
AMAT	3990.26	-23.93	0	0	0	0.00	0.00	0.00	Nan
AMGN	4606.55	-16.48	0	0	0	0.00	0.00	0.00	Nan
AMZN	7484.66	-15.31	0	0	0	0.00	0.00	0.00	Nan
Total	39787,24	–	0	0	0				

Cuadro 5.1: Resultados comprar y mantener

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
AAPL	2184.60	-23.47	6	3	3	0.06	0.17	-0.06	2.83
ADBE	1785.69	-16.81	6	3	3	0.01	0.06	-0.04	1.50
ADP	1642.49	-9.07	4	2	2	0.05	0.12	-0.02	6.00
ADSK	1209.49	-13.59	5	1	4	-0.01	0.11	-0.05	2.20
AKAM	-290.31	-26.59	6	3	3	-0.00	0.08	-0.09	0.89
ALXN	-925.81	-47.19	7	1	6	-0.03	0.13	-0.06	2.17
AMAT	757.48	-28.29	8	4	4	0.02	0.10	-0.06	1.67
AMGN	1452.08	-16.45	6	4	2	0.04	0.07	-0.02	3.50
AMZN	4217.51	-16.08	4	4	0	0.07	0.07	0.00	NaN
Total	12033.22	–	52	25	27				

Cuadro 5.2: Resultados estrategia clásica

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
AAPL	4400.50	-18.98	14	8	6	0.05	0.10	-0.03	3.33
ADBE	3663.88	-11.41	9	8	1	0.05	0.06	-0.02	3.00
ADP	1565.67	-10.21	9	5	4	0.03	0.06	-0.01	6.00
ADSK	3387.24	-25.33	15	13	2	0.03	0.05	-0.10	0.50
AKAM	1170.53	-17.09	11	7	4	0.02	0.07	-0.06	1.17
ALXN	2771.73	-28.24	9	6	3	0.05	0.10	-0.05	2.00
AMAT	1523.11	-11.89	7	5	2	0.04	0.07	-0.04	1.75
AMGN	1678.82	-16.26	11	5	6	0.02	0.07	-0.02	3.50
AMZN	5023.14	-15.60	15	13	2	0.04	0.05	-0.04	1.25
Total	25184,62	–	100	70	30				

Cuadro 5.3: Resultados redes neuronales

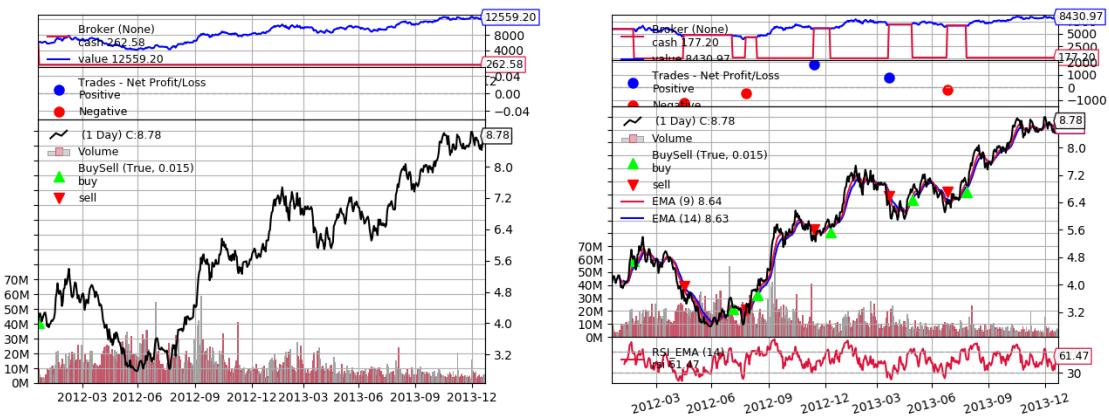
Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
AAPL	-60.77	-8.29	30	16	14	0.00	0.17	-0.20	0.83
ADBE	-198.57	-7.93	22	9	13	-0.13	1.69	-1.38	1.22
ADP	217.84	-3.16	38	22	16	0.07	0.63	-0.68	0.92
ADSK	586.59	-3.50	23	14	9	0.74	2.41	-1.79	1.35
AKAM	-1739.01	-2.85	24	16	8	1.73	3.49	-1.36	2.57
ALXN	235.62	-8.23	22	13	9	0.08	1.00	-1.24	0.81
AMAT	-3.26	-7.86	23	13	10	-0.09	3.88	-5.25	0.74
AMGN	1083.28	-6.60	26	16	10	0.42	1.12	-0.71	1.58
AMZN	753.35	-3.89	28	16	12	0.10	0.32	-0.20	1.58
Total	4184.58	-1.56	264	148	116				

Cuadro 5.4: Fuzzy modeling of stock trading with fuzzy candlesticks

Eurostoxx

Los 9 valores que pertenecen a Eurostoxx son: Aegon (AGN), Allianz (ALV), Deutsche Boerse (DB1), Banco de Sabadell (SAB), Santander (SAN), BME (BME), Credit Agricole (ACA), SAMPO (SAMAS) y British Land Company (BLND).

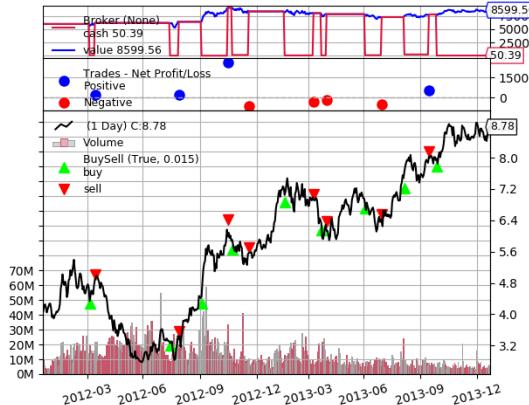
ACA



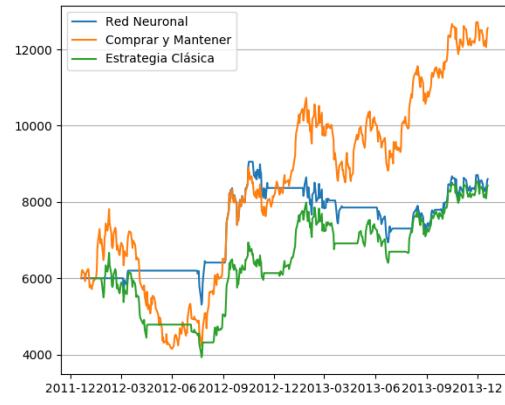
ACA, estrategia: comprar y mantener

ACA, estrategia: clásica

5.2. Resultados

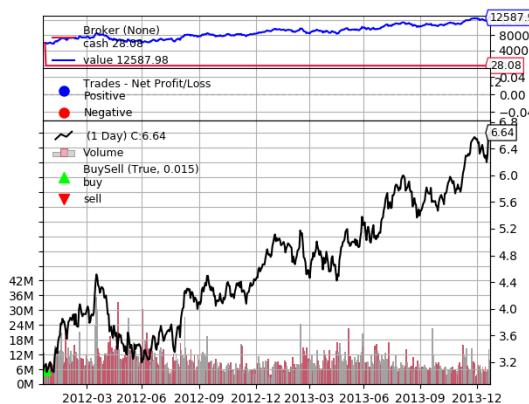


AGN, estrategia: red neuronal



AGN ganancias

AGN



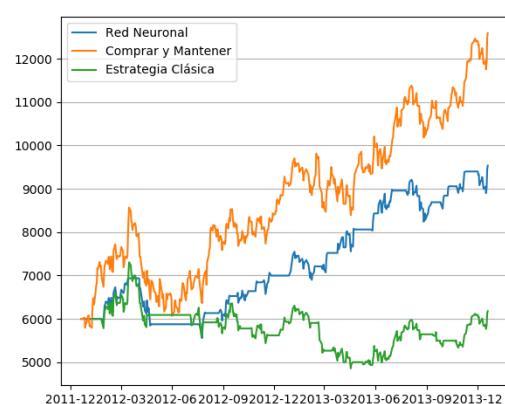
AGN, estrategia: comprar y mantener



AGN, estrategia: clásica

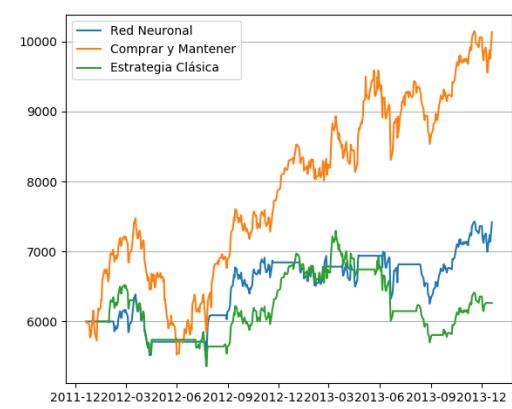
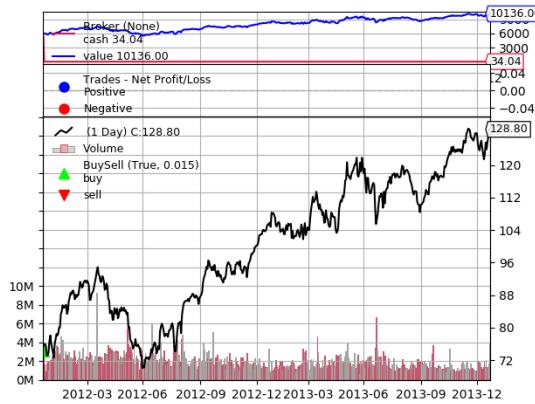


ALV, estrategia: red neuronal

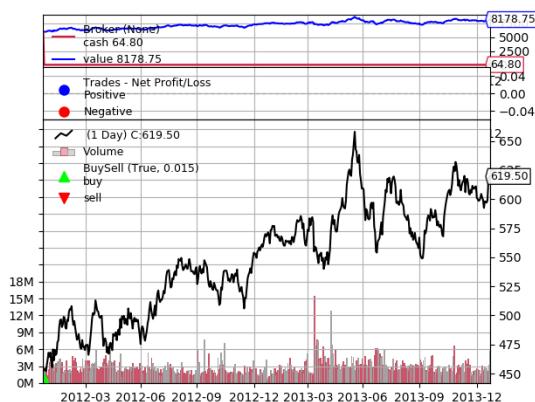


ALV ganancias

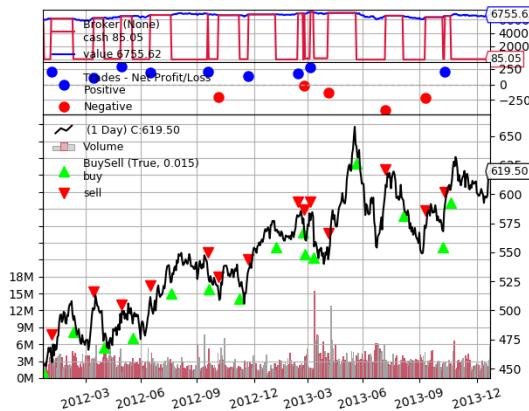
ALV



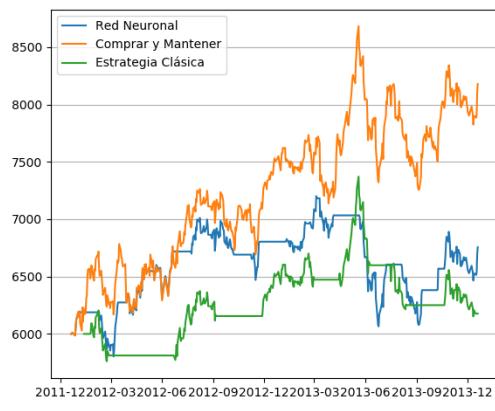
BLND



5.2. Resultados

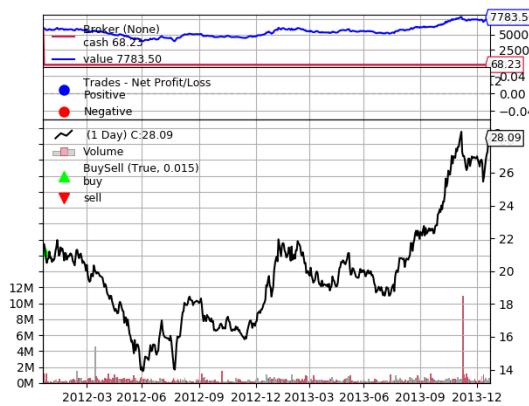


BLND, estrategia: red neuronal



BLND ganancias

BME



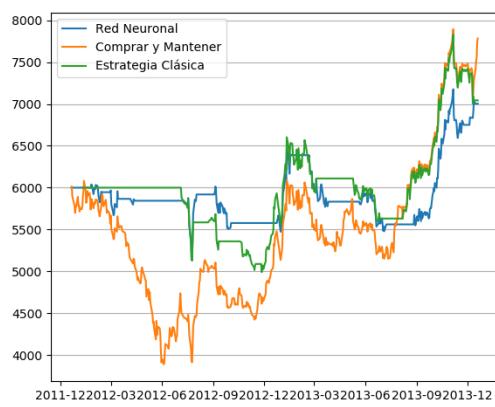
BME, estrategia: comprar y mantener



BME, estrategia: clásica

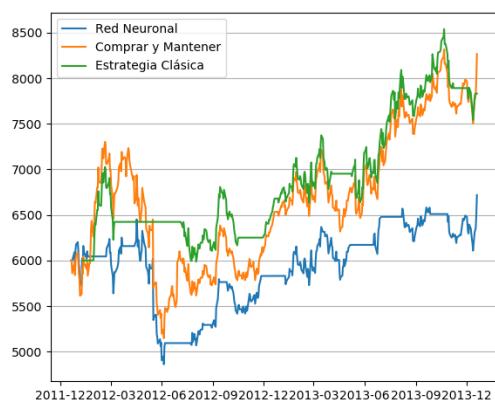


BME, estrategia: red neuronal

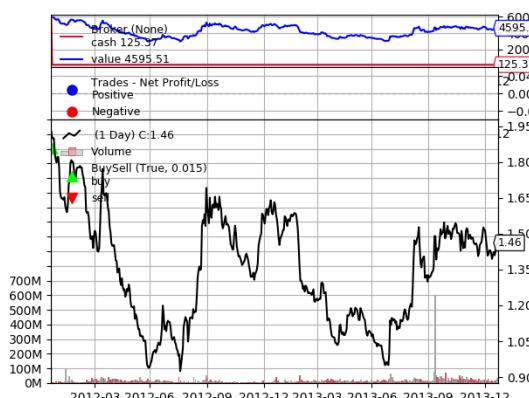


BME ganancias

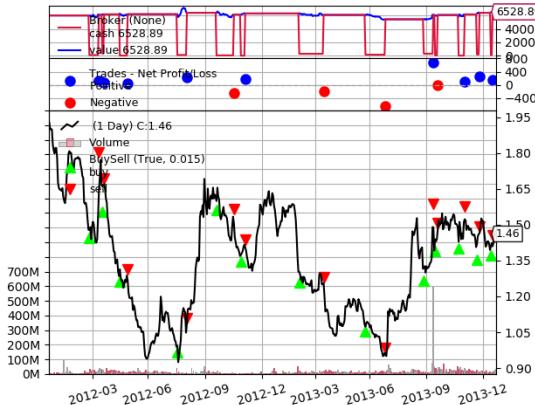
DB1



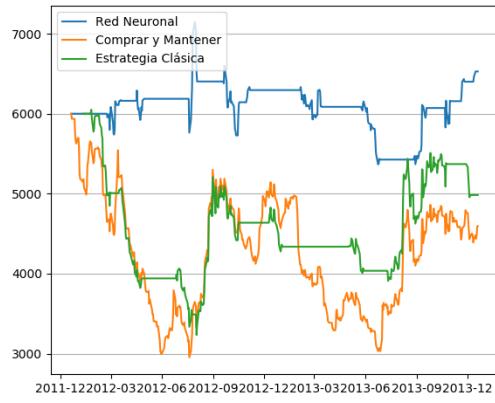
SAB



5.2. Resultados

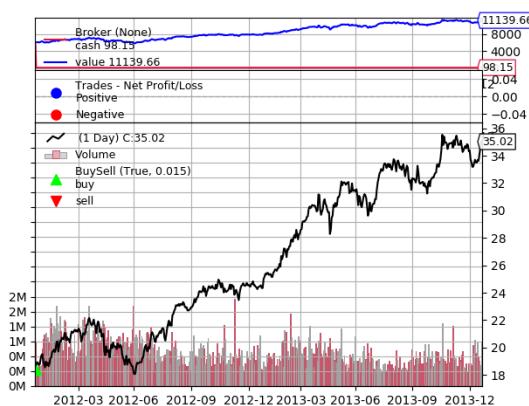


SAB, estrategia: red neuronal

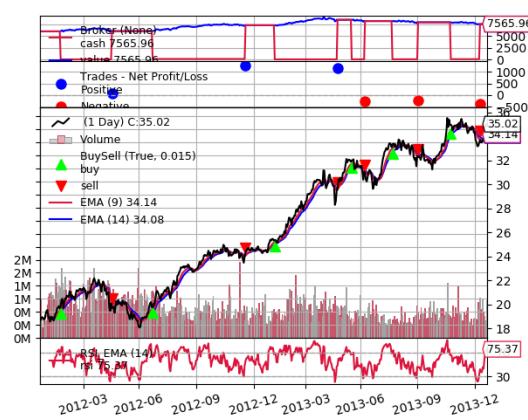


SAB ganancias

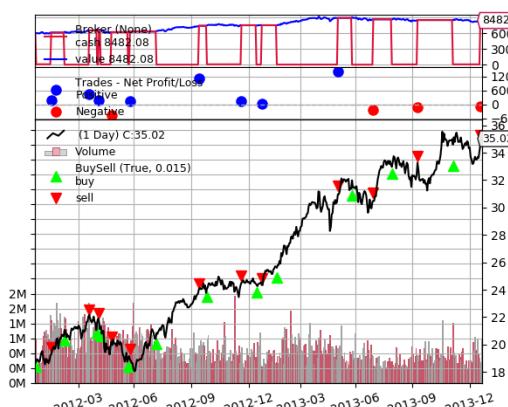
SAMAS



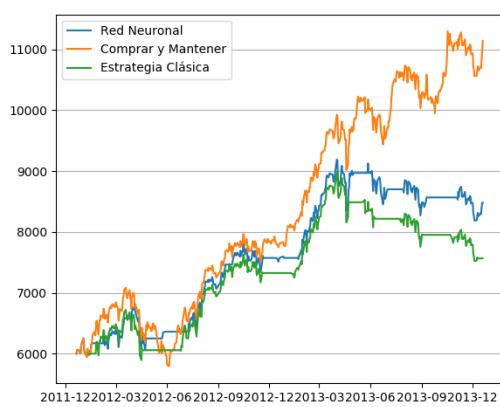
SAMAS, estrategia: comprar y mantener



SAMAS, estrategia: clásica



SAMAS, estrategia: red neuronal



SAMAS ganancias

SAN



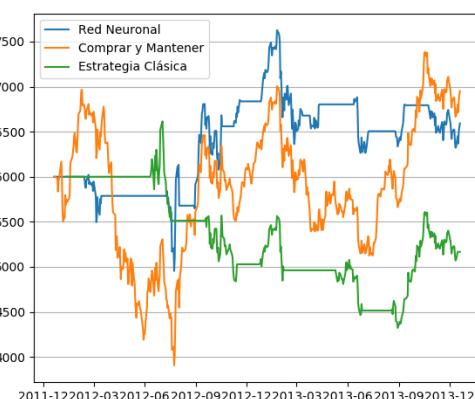
SAN, estrategia: comprar y mantener



SAN, estrategia: clásica



SAN, estrategia: red neuronal



SAN ganancias

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
ACA	6559.20	-46.89	0	0	0	0.00	0.00	0.00	NaN
AGN	6587.98	-29.16	0	0	0	0.00	0.00	0.00	NaN
ALV	4136.00	-26.08	0	0	0	0.00	0.00	0.00	NaN
BLND	2178.75	-16.44	0	0	0	0.00	0.00	0.00	NaN
BME	1783.50	-36.06	0	0	0	0.00	0.00	0.00	NaN
DB1	2266.92	-29.53	0	0	0	0.00	0.00	0.00	NaN
SAB	-1404.49	-50.77	0	0	0	0.00	0.00	0.00	NaN
SAMPO	5139.66	-18.25	0	0	0	0.00	0.00	0.00	NaN
SAN	950.57	-43.92	0	0	0	0.00	0.00	0.00	NaN
Total	28198,09	–	0	0	0				

Cuadro 5.5: Resultados comprar y mantener

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
ACA	2430.97	-41.12	5	2	3	0.04	0.27	-0.11	2.45
AGN	179.44	-33.46	9	3	6	-0.01	0.06	-0.04	1.50
ALV	264.57	-21.88	7	3	4	0.01	0.11	-0.06	1.83
BLND	176.87	-16.54	6	3	3	0.01	0.04	-0.03	1.33
BME	1043.43	-16.82	6	2	4	0.04	0.23	-0.06	3.83
DB1	1832.79	-14.80	5	3	2	0.06	0.11	-0.02	5.50
SAB	-1016.44	-46.57	8	2	6	-0.00	0.33	-0.12	2.75
SAMPO	1565.96	-16.28	6	3	3	0.04	0.13	-0.04	3.25
SAN	-833.47	-34.66	5	1	4	-0.03	0.14	-0.07	2.00
Total	5644,12	-	57	22	35				

Cuadro 5.6: Resultados estrategia clásica

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
ACA	2599.56	-23.41	8	4	4	0.04	0.14	-0.05	2.80
AGN	3533.11	-19.88	20	18	2	0.02	0.04	-0.09	0.44
ALV	1418.58	-13.64	8	5	3	0.02	0.04	-0.03	1.33
BLND	755.62	-15.77	14	9	5	0.01	0.03	-0.03	1.00
BME	1003.99	-15.19	12	6	6	0.02	0.07	-0.04	1.75
DB1	718.05	-24.67	11	8	3	0.01	0.04	-0.07	0.57
SAB	528.89	-24.90	13	9	4	0.01	0.03	-0.05	0.60
SAMPO	2482.08	-10.91	12	8	4	0.03	0.06	-0.03	2.00
SAN	592.01	-17.86	8	4	4	0.02	0.07	-0.03	2.33
Total	13631,89	-	106	71	35				

Cuadro 5.7: Resultados redes neuronales

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
ACA	697.11	-3.59	17	10	7	5.65	18.66	-12.95	1.44
AGN	1880.05	-0.81	15	13	2	26.20	33.50	-8.16	4.11
ALV	715.29	-4.25	29	16	13	0.29	0.86	-0.38	2.23
BLND	483.30	-2.73	23	15	8	0.03	0.09	-0.06	1.45
BME	47.73	-4.76	18	8	10	0.08	3.52	-2.67	1.32
DB1	723.31	-5.52	31	18	13	0.40	1.53	-1.07	1.43
SAB	-1470.02	-25.24	35	11	24	-24.91	56.11	-66.19	0.85
SAMPO	743.65	-0.62	12	10	2	2.81	3.71	-1.69	2.20
SAN	835.86	-3.49	23	12	11	6.88	20.22	-7.67	2.64
Total	4656.28	-0.96	203	113	90				

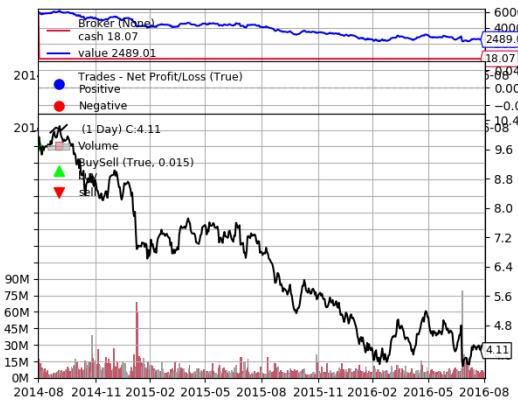
Cuadro 5.8: Fuzzy modeling of stock trading with fuzzy candlesticks

5.2.2 Momentos bajistas

Se ha añadido este caso de estudio, debido a que la mayoría de los mercados que se estudiaron en el artículo [4] (con el cual hemos comparado en la sección anterior), se encontraban en un periodo alcista, por lo que no se pudo comprobar el funcionamiento de la red en periodos bajistas. En esta sección se comparará con la estrategia de comprar y mantener y con la estrategia clásica explicada anteriormente.

Los parámetros considerados en el entrenamiento para esta sección son los mismos que los de la sección anterior.

SAN



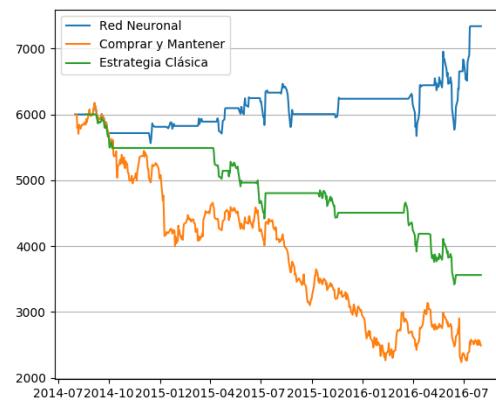
SAN, estrategia: comprar y mantener



SAN, estrategia: clásica



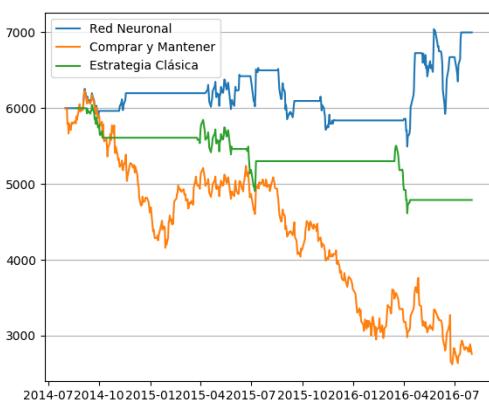
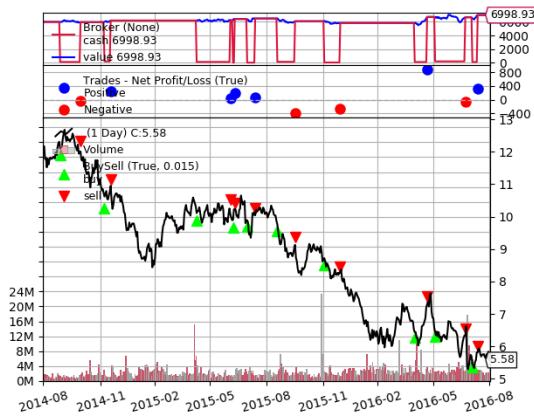
SAN, estrategia: red neuronal



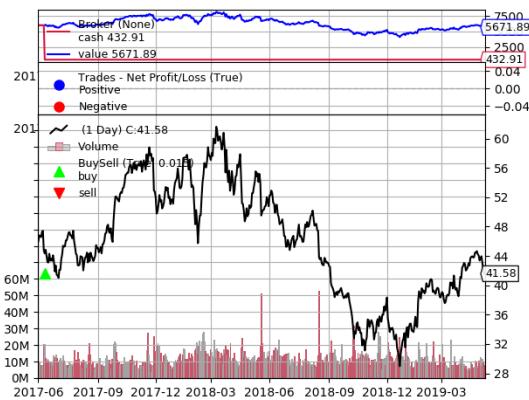
SAN ganancias

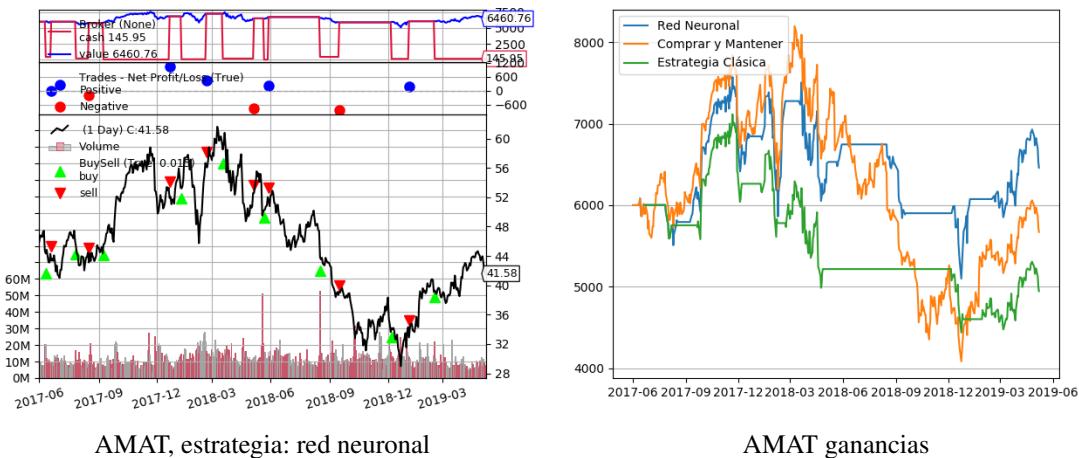
BBVA

5.2. Resultados



AMAT





Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
SAN	-3510.99	-63.78	0	0	0	0.00	0.00	0.00	NaN
BBVA	-3240.88	-57.94	0	0	0	0.00	0.00	0.00	NaN
AMAT	-328.11	-50.15	0	0	0	0.00	0.00	0.00	NaN
Total	-7079.98	–	0	0	0				

Cuadro 5.9: Resultados comprar y mantener

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
SAN	-2439.12	-43.04	7	0	7	-0.07	0.00	-0.07	-0.00
BBVA	-1210.28	-23.62	4	0	4	-0.05	0.00	-0.05	-0.00
AMAT	-1053.61	-37.62	5	1	4	-0.05	0.09	-0.08	1.12
Total	-4703.01	–	16	1	15				

Cuadro 5.10: Resultados estrategia clásica

Compañía	Beneficio	Max DD (%)	Oper. total	Oper.+	Oper.-	P. Oper.	P. Oper. +	P. Oper. -	+/-
SAN	1340.22	-17.08	13	11	2	0.02	0.03	-0.06	0.50
BBVA	998.93	-15.88	10	6	4	0.02	0.05	-0.03	1.67
AMAT	460.76	-32.65	8	5	3	0.01	0.06	-0.09	0.67
Total	2799.91	–	31	22	9				

Cuadro 5.11: Resultados red neuronal

Capítulo

6

CONCLUSIONES Y TRABAJOS FUTUROS

6.1 Análisis de los resultados

A continuación se muestran las tablas con los resultados globales de los casos de estudio.

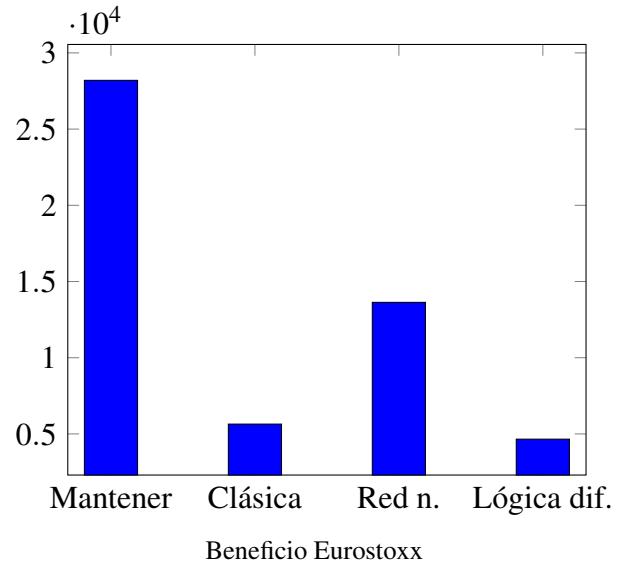
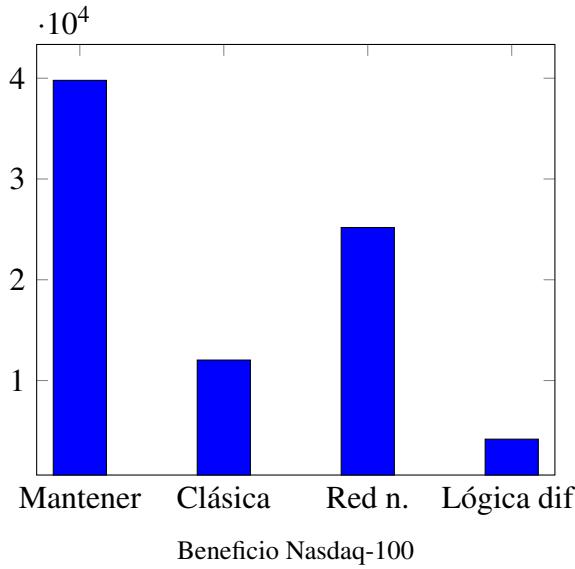
Estrategia	Beneficio	Oper. total	Oper.+	Oper.-
Comprar y Mantener	39787.24	0	0	0
Clásica	12033.22	52	25	27
Redes Neuronales	25184.62	100	70	30
Lógica Difusa	4184.58	264	148	116

Cuadro 6.1: Resultados globales Nasdaq-100

Estrategia	Beneficio	Oper. total	Oper.+	Oper.-
Comprar y Mantener	28198.09	0	0	0
Clásica	5644.12	57	22	35
Redes Neuronales	13631.89	106	71	35
Lógica Difusa	4656.28	203	113	90

Cuadro 6.2: Resultados globales Eurostoxx

Para facilitar la visualización del beneficio se muestra a continuación las gráficas de los beneficios obtenidos por las diferentes estrategia en Nasdaq-100 y Eurostoxx.



A la vista de los resultados, puede parecer que la mejor estrategia es la estrategia de comprar y mantener. Esto en realidad no es así en general y lo que aquí ocurre es que la mayoría de los mercados se encontraban en un período alcista, en el período utilizado para realizar la validación, debido a ello se decidió realizar un experimento adicional sobre mercados bajistas, el cual analizaremos después.

Comparando la estrategia clásica, con la red neuronal, podemos apreciar una diferencia bastante grande sobre el beneficio obtenido, donde el beneficio de la red duplica al de la estrategia clásica. Además podemos apreciar que la red neuronal ejecuta más operaciones, y no solo esto, sino que además aumenta la proporción de operaciones positivas llevadas a cabo. Mientras que el porcentaje de operaciones positivas de la estrategia clásica, no llega si quiera al 50 %, podemos ver que el de la red neuronal llega al 70 %.

Comparando la red neuronal con la estrategia por medio de lógica difusa, podemos ver una diferencia enorme respecto al beneficio, donde en los resultados del Nasdaq-100, los beneficios de la red llegan a cuadruplicar los de lógica difusa. Además podemos ver una diferencia bastante notable entre el número de operaciones llevadas a cabo por la red neuronal y por lógica difusa, siendo mucho más elevado en el caso de la lógica difusa, a pesar de ello la proporción de operaciones positivas, es mayor en el caso de las redes neuronales lo cual acaba reflejándose en el beneficio obtenido.

Mercados bajistas

Veamos ahora los resultados globales del experimento en mercados bajistas. En este caso no se muestran los resultados con lógica difusa, ya que el artículo del que se obtuvieron los resultados empleando dicha técnica, no realizó estos experimentos.

Estrategia	Beneficio	Oper. total	Oper.+	Oper.-
Comprar y Mantener	-7079.98	0	0	0
Clásica	-4703.01	16	1	15
Redes Neuronales	2496.91	35	25	10

Cuadro 6.3: Resultados globales mercados bajistas

En este caso al tratarse de mercados bajistas, evidentemente la simple estrategia de comprar y mantener ya no arroja beneficios, y como podemos ver ni si quiera la estrategia clásica lo hace. En cambio vemos que la red neuronal si que llega a obtener algo de beneficio. Veamos la siguiente ilustración para ver este hecho de forma gráfica.

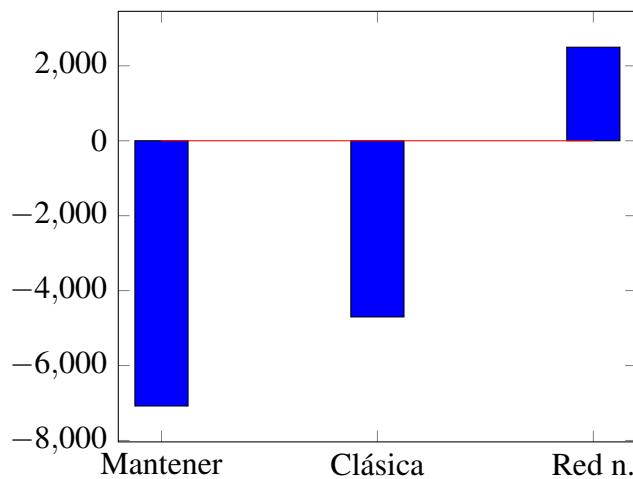


Figura 6.1: Beneficios mercados bajistas

Comparando la estrategia de la red neuronal, con la estrategia clásica, vemos que los resultados hablan por si solos, mientras que la clásica ha dado pérdidas considerables, la red neuronal ha conseguido obtener beneficios. Si comparamos el número de operaciones, vemos que ha sido mayor en el caso de las redes, pero a pesar de ello ha mantenido una proporción de operaciones positivas de mas de un 60 %.

6.2 Conclusiones

En este trabajo hemos visto como aplicar las redes neuronales, para predecir los mejores momentos de compra y venta. Los resultados obtenidos han sido bastante satisfactorios, pues hemos visto como la estrategia definida, tomando decisiones solamente por medio de las predicciones de la red, ha sido capaz de obtener beneficio, incluso en periodos bajistas.

Desde el punto de vista de los inversores, es muy interesante, pues muestra una forma automática de invertir, por medio de señales de compra y venta diferente de las estrategias clásicas, en las cuales se utilizan unos indicadores técnicos y unas reglas sobre estos, para emitir señales de compra y venta. En cambio la red neuronal a partir de los indicadores técnicos aportados a dicha red, es capaz, por decirlo de alguna forma, de establecer mediante el entrenamiento, sus propias reglas para estos indicadores, para emitir señales de compra y venta. Además es capaz de ir modificando las reglas, mediante el transcurso del tiempo, al ir realimentando la red neuronal con los nuevos datos.

Su capacidad de aprender de los datos, hacen de las redes una gran herramienta para invertir. Una de sus capacidades más poderosas es la de poder realimentarse con el transcurso del tiempo, con la nueva información que va surgiendo, de esta forma cada vez aprenden más de los datos, se mantienen actualizadas y perfeccionan su capacidad de predicción, pudiéndose anticipar en ocasiones a cambios en la tendencia del mercado. Hay que tener en cuenta que las redes neuronales puedan caer fácilmente en el sobreaprendizaje y perder su capacidad de generalización, por ello es importante usar siempre métodos de regularización, para que a pesar de realimentar la red, esta no llegue a sobreentrenarse.

El principal problema que presentan las redes neuronales, es el ser un sistema de caja negra, en el cual no sabemos lo que pasa dentro, ni como llega a tomar las decisiones, tan solo conocemos los datos que le introducimos y la salida que obtenemos. Esto puede llegar a ser un problema a la hora de invertir, pues si la red neuronal nos dice que es un buen momento de compra, no podemos saber el porqué de esta respuesta, ni podemos dar una justificación clara al respecto. Otro problema de nuestra red es que tan solo tiene como entrada, datos sobre indicadores técnicos, con lo cual no tiene ninguna información sobre noticias de actualidad o información típica del análisis fundamental.

A pesar de ello, a la vista de los resultados obtenidos en este trabajo, podemos ver que su uso puede ser realmente positivo para el inversor ayudando a incrementar los beneficios. La red neuronal puede usarse como información adicional a la hora de tomar una decisión. Así puede ser un gran complemento para el conocimiento experto del inversor.

Podemos concluir que se ha cumplido el principal objetivo de este proyecto, el cual se trata desarrollar un algoritmo, capaz de realizar predicciones en el mercado, mediante el uso de redes neuronales. La implementación de este algoritmo, ha quedado descrita en el capítulo 4.

Además se ha cumplido el objetivo de realizar diferentes casos de estudio, simulando una estrategia que hiciera uso de la red neuronal implementada y obteniendo los resultados de dicha simulación, para posteriormente comparar los resultados obtenidos con otras estrategias, a la vez que se cumple el tercer objetivo.

Por último, se puede decir también que se ha cumplido el objetivo de mostrar detalladamente todo el proceso seguido, para que este trabajo pueda servir de manual para invertir en bolsa con técnicas de aprendizaje automático basadas en redes neuronales.

6.3 Trabajos futuros

En este trabajo se ha implementado un modelo de perceptrón multicapa, pero también se podrían haber probado otros modelos de redes neuronales más modernas como pueden ser las redes LSTM por su término en inglés Long short-term memory. Estas redes neuronales tienen una arquitectura recurrente y se ha probado que se comportan bien en predicción de series temporales.

Otro trabajo pendiente sería el de diseñar la arquitectura, ya que en este trabajo se ha cogido una arquitectura estándar. Para ello se deberían de probar diferentes modelos, cambiando el número de capas ocultas y también modificando el número de nodos en cada capa oculta hasta dar con un diseño óptimo. En algunos estudios se han utilizado algoritmos genéticos para encontrar este diseño óptimo de la arquitectura.

En este trabajo para cada simulación sobre un mercado, se ha entrenado previamente la red neuronal, únicamente con los datos de dicho mercado, pero podría ser interesante entrenar la red además con datos de otros mercados relacionados. Por ejemplo si se pretende obtener un modelo para invertir en Amazon, podrían utilizarse también para entrenar la red, datos de mercados pertenecientes al Nasdaq-100, como por ejemplo Apple, Facebook o Intel.

Apéndice

A

ANEXO INFORMÁTICA

A.1 Instalación

A.1.1 Requisitos

La instalación de este programa se ha llevado a cabo en el sistema operativo Ubuntu 18.04.

- Como requisito previo es necesario tener instalado Python3. En caso de no disponer de Python3 instalado se puede instalar fácilmente desde la terminal utilizando el siguiente comando:

```
1 sudo apt-get install python3
```

Es recomendable ejecutar el comando `sudo apt-get update` antes de la instalación de Python3.

- Otro requisito necesario es disponer del gestor de paquetes de python3, llamado pip3, el cual debería de haberse instalado junto con python3, pero en caso de no disponer de este se puede instalar con el siguiente comando:

```
1 sudo apt-get -y install python3-pip
```

A.1.2 Instalación

El proyecto puede ser obtenido mediante git, clonando el repositorio mediante el siguiente comando:

```
1 git clone https://github.com/Solano96/TFG.git
```

El proyecto utiliza algunos paquetes cuyas versiones no están actualizadas, por lo que instalar los paquetes necesarios podría contaminar nuestro sistema instalando versiones desactualizadas. Debido a esto se ha creado un entorno virtual en el proyecto que puede ser activado, estando en la carpeta del repositorio, con el siguiente comando:

```
1 source env/bin/activate
```

Una vez ejecutado el comando anterior estaremos dentro del entorno virtual, el cual no tiene nada instalado en un principio, por lo que deberemos ejecutar el siguiente comando para instalar los paquetes necesarios para el proyecto, dentro del entorno virtual:

```
1 pip3 install -r requirements.txt
```

El comando anterior descargará e instalará todos los paquetes especificados en el fichero `requirements.txt`.

Con todo esto ya tendremos el entorno virtual preparado para ejecutar el programa de terminal. Para salir del entorno virtual podemos ejecutar el comando `deactivate`.

Para poder ejecutar la interfaz gráfica debemos instalar el paquete `tkinter`, el cual debe ser instalado por medio del comando `apt-get install`, por lo que deberá ser instalado fuera del entorno virtual. Para instalar `tkinter` ejecutaremos el siguiente comando:

```
1 sudo apt-get install python3-tk
```

Una vez instalado podremos entrar de nuevo al entorno virtual y ejecutar el programa con interfaz.

A.1.3 Uso del programa

Hay dos versiones del programa, una de ellas para realizar pruebas y experimentos desde la terminal y la otra versión con interfaz gráfica. Recordar que si se han instalado todos los paquetes en el entorno virtual, será necesario tener este activo para poder ejecutar los programas.

La versión de pruebas tiene las fechas fijas en el código y habría que modificar el código para poder cambiarlas, esta versión ha sido utilizada principalmente para realizar pruebas y experimentos para obtener resultados. Además de simular la estrategia de la red neuronal, también simula la estrategia de comprar y mantener y la estrategia clásica ya comentada en otro capítulo.

Adicionalmente se ha creado una versión con interfaz, para que cualquier usuario

que disponga del programa, pueda probarlo facilmente y ver los resultados de aplicar las redes neuronales de forma interactiva, ya que durante la simulación se van mostrando los resultados en la gráfica continuamente.

Versión de pruebas

Para utilizar la versión de terminal debemos ejecutar el siguiente comando:

```
1 python3 main.py <data-name> <gain> <loss> <simulation-days> <-->
   ↳ training-epochs>
```

Donde en <data-name> se especificará la abreviatura correspondiente al mercado que se quiera utilizar para realizar la prueba. Los nombres aceptados son aquellos correspondientes a las abreviaturas de los mercados reconocidos por yahoo finance, por lo que si se duda de alguna abreviatura se puede consultar en la página de yahoo. Por ejemplo las acciones de Amazon tiene como abreviatura AMZN o las de Banco Santander la abreviatura SAN.

Los parámetros <gain>, <loss> y <simulation-days> se corresponden con los parámetros asociados a la simulación para asignar las etiquetas a los datos del conjunto de entrenamiento. El parámetro <training-epochs> sirve para definir el número de iteraciones de la red neuronal en el entrenamiento.

A continuación se muestra un ejemplo de ejecución:

```
1 $ python3 main.py SAN 0.07 0.05 10 300
2 Using TensorFlow backend.
3 SAN
4
5 Cargando datos...
6 Datos existentes en /data.
7 data/SAN.csv cargado con éxito.
8
9 Estrategia: red neuronal
10
11 Añadiendo características...
12 Añadiendo etiquetas...
13 Normalizando datos...
14 Entrenando red neuronal...
15
16 Valor inicial de la cartera: 6000.00
17 Valor final de la cartera : 7340.22
18
19 Estrategia: comprar y mantener
20
21 Valor inicial de la cartera: 6000.00
```

```

22 Valor final de la cartera : 2489.01
23
24 Estrategia: clásica
25
26 Valor inicial de la cartera: 6000.00
27 Valor final de la cartera : 3560.88

```

Que además de mostrar los resultados anteriores, almacenará en la carpeta img las gráficas resultantes de la simulación de cada una de las estrategias y además en la carpeta resultados un fichero con los resultados obtenidos por los analizadores técnicos.

Versión con interfaz gráfica

Para ejecutar esta versión simplemente deberemos de escribir en la terminal lo siguiente:

```
1 python3 interface.py
```

Tras lo cual se mostrará la siguiente ventana:

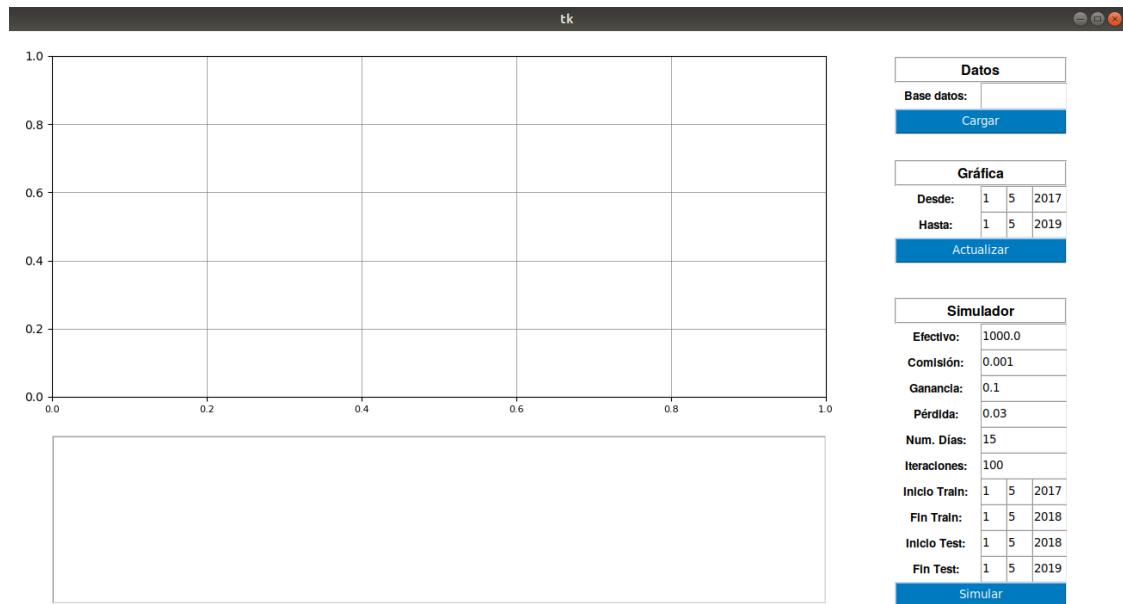


Figura A.1: Interfaz gráfica implementada

En dicha interfaz podremos cargar la base de datos que deseemos, mostrar la gráfica comprendida en un período de tiempo previamente definido y además elegir todos los parámetros deseados para el entrenamiento de la red.

Durante la simulación de la estrategia con la red neuronal, veremos como va actuando la gráfica de la simulación durante el transcurso de esta y además en el cuadro

de texto de la parte inferior, se mostraran las ordenes ejecutadas o que hayan podido quedar canceladas por algún motivo.

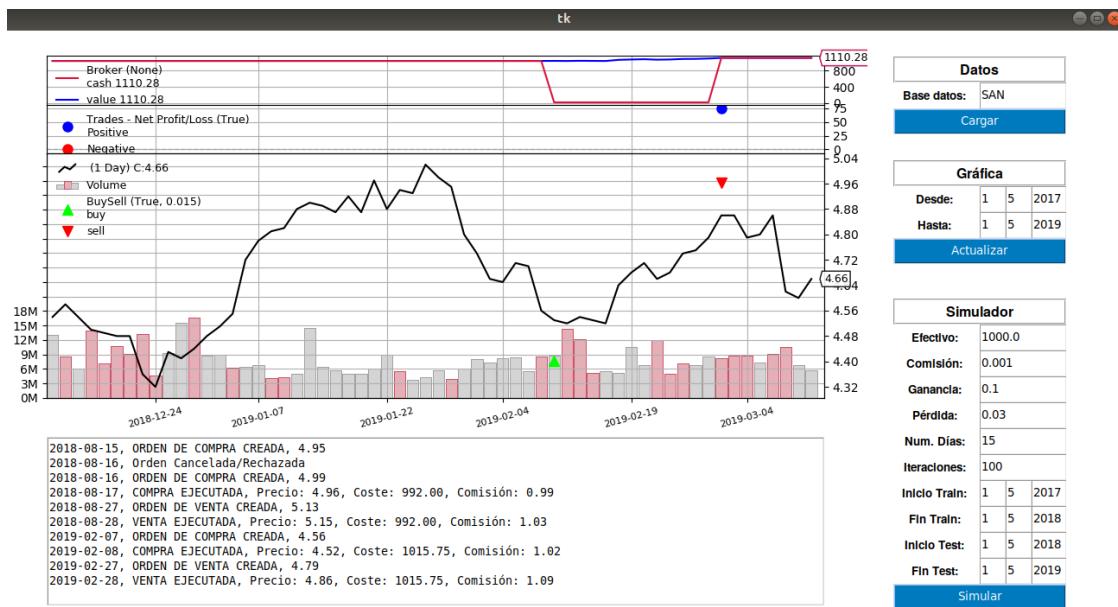


Figura A.2: Simulación de la estrategia en la interfaz gráfica

A.2 Backtrader

En esta sección introduciremos el framework backtrader, el cual utilizaremos para realizar una simulación sobre un histórico de cotización sobre un determinado valor. De esta forma podremos validar la estrategia obtenida mediante el modelo neuronal.

No vamos a entrar en mucho detalle sobre backtrader, pero si lo suficiente como para aprender a utilizarlo de forma básica.

A.2.1 La clase Cerebro

Vamos a comenzar con un ejemplo sencillo que nos servirá como introducción para ir explicando este framework.

Código A.1: Simulación con Backtrader

```

1 import backtrader as bt
2
3 if __name__ == '__main__':
4     # Creamos una entidad cerebro
5     cerebro = bt.Cerebro()
6
7     # Añadimos una estrategia

```

```

8 cerebro.addstrategy(TestStrategy)
9
10 # Obtenemos la base de datos del banco Santander
11 name = 'SAN'
12 data = yf.download(name, '2017-01-01', '2019-01-01')
13 data = data[['Open', 'High', 'Low', 'Close', 'Volume']]
14
15 # Añadimos los datos al cerebro
16 cerebro.adddata(bt.feeds.PandasData(dataname = data))
17
18 # Definimos el saldo inicial y el coste de la comisión
19 cerebro.broker.setcash(100000.0)
20 cerebro.broker.setcommission(commission=0.001)
21
22 initial_value = cerebro.broker.getvalue()
23 print('\nValor inicial de la cartera: %.2f' % initial_value)
24
25 # Ejecutamos la estrategia sobre los datos añadidos
26 strats = cerebro.run()
27
28 final_value = cerebro.broker.getvalue()
29 print('Valor final de la cartera : %.2f' % final_value)
30
31 # Mostramos el resultado de la simulación
32 cerebro.plot()

```

Como podemos ver en el código anterior lo primero que hacemos es crear una instancia de la clase cerebro, esta clase es clave para backtrader, ya que esta se trata de un motor que nos permitirá recopilar los datos de entrada, ejecutar la simulación para validar la estrategia, obtener los resultados de dicha simulación y además poder representar la simulación en un gráfico.

Tras crear la instancia cerebro añadimos como estrategia clase TestStrategy, la cual se trata de una clase en la que se ha definido previamente estrategia a seguir. La clase TestStrategy será explicada más adelante.

A continuación obtenemos los datos del banco Santander, los cuales han sido descargados de yahoo finances con el comando:

```
yf.download(name, '2017-01-01', '2019-01-01')
```

Este comando descarga los datos del nombre proporcionado, entre la fechas 2017-01-01 y 2019-01-01. Tras obtener los datos se los proporcionamos al cerebro.

Después almacenamos en el cerebro el dinero inicial en este caso 100000.0 unidades monetarias y fijamos una comisión por cada operación de compra o venta del 0.1 %, el cual está dividido por 100 para eliminar el porcentaje.

Por último ejecutamos con `cerebro.run()` y mostramos los resultados en una gráfica con `cerebro.plot()`.

Al script del ejemplo se le ha llamado tutorial-backtrader.py, para ejecutarlo basta con abrir la terminal, situarse en la carpeta en la que se encuentra el script y usar el

comando python3 para su ejecución. Los resultados de la ejecución de este ejemplo han sido los siguientes:

```

1 $ python3 tutorial-backtrader.py
2
3 Base Datos: SAN
4 [*****100%*****] 1 of 1 ↵
    ↵ downloaded
5
6 Valor inicial de la cartera: 100000.00
7 Valor final de la cartera : 99998.78

```

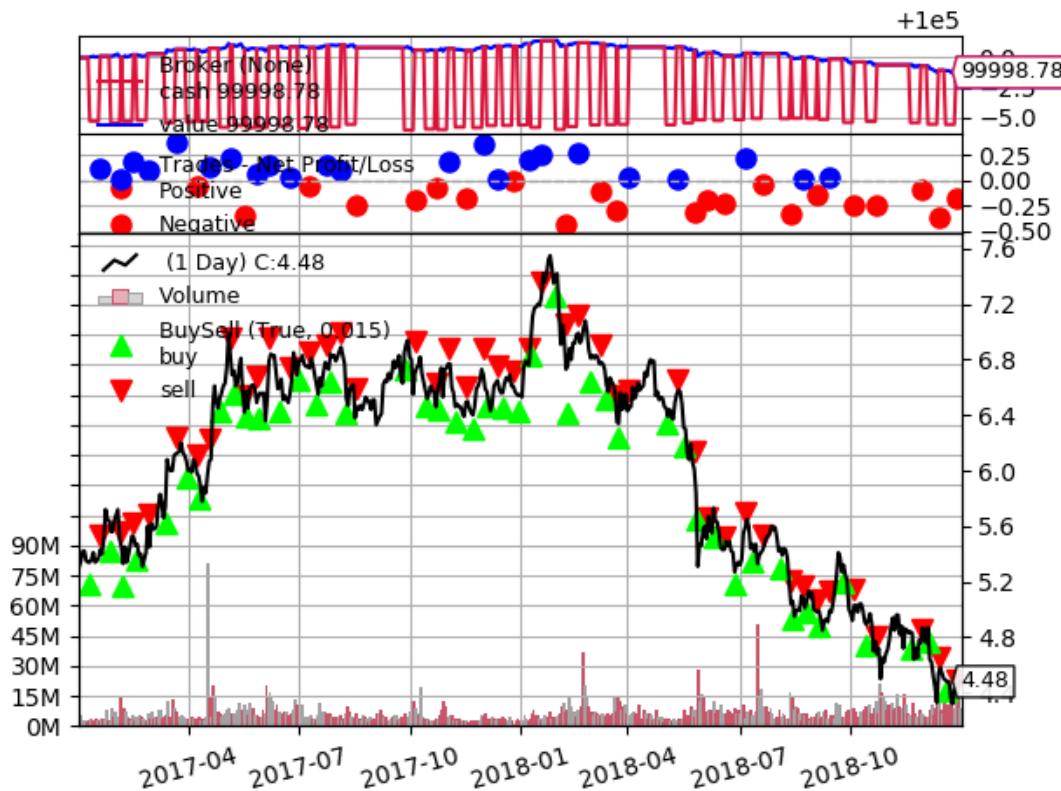


Figura A.3: cerebro.plot()

Las flechas verdes se corresponde con el momento de compra, mientras que las flechas rojas son el momento de venta. Los gráficos de barras en la parte inferior de la figura se corresponden con el volumen. Los puntos rojos y azules tienen como objetivo indicar si la compra-venta ha sido positiva o negativa, es decir si ha resultado en beneficio o no.

A.2.2 La clase Strategy

Para poder definir nuestras propias estrategias, debemos de crear una clase que extienda a la clase `bt.Strategy` de Backtrader. La estrategia definida, para este ejemplo inicial, en la clase `TestStrategy` ha sido la siguiente.

Código A.2: Ejemplo de estrategia simple

```

1  class TestStrategy(bt.Strategy):
2
3      def __init__(self):
4          self.dataclose = self.datas[0].close
5          self.order = None
6
7      def notify_order(self, order):
8          if order.status in [order.Submitted, order.Accepted]:
9              return
10
11     if order.status in [order.Completed]:
12         self.bar_executed = len(self)
13
14     self.order = None
15
16     def next(self):
17         if self.order:
18             return
19
20         if not self.position:
21             if self.dataclose[0] < self.dataclose[-1]:
22                 if self.dataclose[-1] < self.dataclose[-2]:
23                     self.order = self.buy()
24
25         else:
26             if len(self) >= (self.bar_executed + 5):
27                 self.order = self.sell()
```

En el método `next`, es en el que se define la estrategia, en este se programa la lógica de la estrategia, y en el cual se toman las decisiones para cada día de cotización. La variable de clase `self.position` contiene un valor booleano, que nos indica si estamos en el mercado o no, es decir si se tiene acciones compradas o no se tienen aún. El array `dataclose` tiene siempre en la posición 0 el precio de cierre del día correspondiente. Para acceder al precio de n días anteriores al actual se puede usar `self.dataclose[-n]`. Para comprar y vender simplemente se usan los métodos `self.buy()` y `self.sell()`, a los cuales se les puede indicar como parámetro el volumen de la compra o de la venta, que por defecto es una sola acción.

En esta simple estrategia de ejemplo se compra cuando el precio ha bajado durante 3 días seguidos y se vende después de transcurrir 5 días desde que se compro la acción.

Backtrader también permite utilizar indicadores técnicos para definir la estrategia. Para utilizar los indicadores técnicos que Backtrader proporciona, tenemos que escribir

bt.indicators.NOMBRE_INDICADOR, por ejemplo el indicador de la media móvil exponencial sería bt.indicators.EMA(datos, period=n), al cual le pasamos los datos y el periodo de días para el cálculo de la media móvil.

Veamos un ejemplo de estrategia clásica utilizando indicadores técnicos. Comentar que esta estrategia ha sido la empleada en el capítulo de casos de estudio, para comparar con la estrategia de la red neuronal.

Código A.3: Ejemplo de estrategia clásica

```

1  class ClassicStrategy(bt.Strategy):
2
3      def __init__(self):
4          """ Inicializador de la clase ClassicStrategy """
5          self.dataclose = self.datas[0].close
6
7          self.ema_9 = bt.indicators.EMA(self.datas[0], period=9)
8          self.ema_14 = bt.indicators.EMA(self.datas[0], period=14)
9          self.rsi_14 = bt.indicators.RSI_EMA(self.datas[0], period = 14)
10
11     def next(self):
12
13         # Realizar operacion
14         if not self.position:
15             if self.ema_9[0] > self.ema_14[0] and self.rsi_14[-1] > 70 and self.rsi_14[0] <= ↵
16                 ↵ 70:
17                 buy_size = self.broker.get_cash() / self.datas[0].open
18                 self.buy(size = buy_size)
19         else:
20             if self.ema_9[0] < self.ema_14[0] and self.rsi_14[-1] < 30 and self.rsi_14[0] >= ↵
21                 ↵ 30:
22                 sell_size = self.broker.getposition(data = self.datas[0]).size
23                 self.sell(size = sell_size)
```

A.2.3 La clase Analyzers

En Backtrader también podemos añadir al cerebro analizadores, que van realizando cálculos durante la simulación y nos devuelven al final indicadores sobre el resultado de la simulación con los cuales podemos validar la estrategia. Por ejemplo uno de los analizadores que podemos utilizar es el DrawDown. Para añadirlo solamente debemos de añadir la siguiente línea antes de la ejecución del cerebro:

```
cerebro.addanalyzer(bt.analyzers.DrawDown, _name="drawDown")
```

Tras la ejecución del cerebro podemos acceder a los resultados obtenidos por el analizador de la siguiente forma:

```
dd = strats[0].analyzers.drawDown.get_analysis()
```

Al igual que podíamos hacer con las estrategias, también podemos definir nuestros propios analizadores, creando para ello una clase que extienda a la clase bt.Analyzer.

Nota: La variable strats contiene el resultado de ejecutar cerebro.run(), podemos verlo en el código A.1 línea 26.

Código A.4: Ejemplo de analizador simple

```

1  class MyAnalyzer(Analyzer):
2
3      def create_analysis(self):
4          self.rets = AutoOrderedDict()
5          self.rets.trades.total = 0
6          self.rets.trades.positives = 0
7          self.rets.trades.negatives = 0
8
9      def stop(self):
10         super(MyAnalyzer, self).stop()
11
12     def notify_trade(self, trade):
13         if not trade.isclosed:
14             return
15
16         self.rets.trades.total+=1
17
18         if trade.pnlcomm > 0:
19             self.rets.trades.positives+=1
20         elif trade.pnlcomm < 0:
21             self.rets.trades.negatives+=1

```

El analizador anterior lleva un seguimiento del número total de operaciones realizadas, el número total de operaciones positivas y el número total de operaciones negativas. Para ello cada vez que se produce una venta incrementa en una unidad la variable de clase `self.rets.trades.total` y después comprueba mediante la variable `trade.pnlcomm` si el resultado de la operación ha sido positivo o negativo para incrementar el número de operaciones positivas o el número de operaciones negativas.

Para añadir el analizar creado al cerebro, procedemos igual que antes:

```
cerebro.addanalyzer(myAnalyzer.MyAnalyzer, _name = "myAnalyzer")
```

Y tras la ejecución del cerebro para simular la estrategia sobre los datos para la simulación, procedemos de nuevo a obtener los resultados del analizador:

```
ma = strats[0].analyzers.myAnalyzer.get_analysis()
```

Apéndice

B

ANEXO MATEMÁTICAS

Definición B.1 *Dados $X, Y \in \mathbb{R}^N$, donde $X = (x_1, x_2, \dots, x_n)$ e $Y = (y_1, y_2, \dots, y_n)$, se define el producto escalar euclídeo como sigue:*

$$X \cdot Y = \langle X, Y \rangle = \sum_{i=1}^N x_i \cdot y_i$$

Definición B.2 *Sea $X \in \mathbb{R}^N$, con $X = (x_1, x_2, \dots, x_n)$ se define la norma euclídea de X como:*

$$\|X\| = \sqrt{\langle X, X \rangle} = \sqrt{\sum_{i=1}^N x_i^2}$$

Definición B.3 *Un hiperplano de \mathbb{R}^N se define como el conjunto de soluciones de una ecuación lineal, es decir un conjunto de la forma $\{X = (x_1, \dots, x_N) \in \mathbb{R}^N : W \cdot X = c\}$, donde $W \in \mathbb{R}^N$ y es no nulo, y $c \in \mathbb{R}$.*

Ejemplos de hiperplanos:

- En el plano se tiene que los hiperplanos son las rectas de dicho plano.
- En el espacio los hiperplanos son los planos de dicho espacio.

Definición B.4 *Dado dos conjuntos $X = (\vec{x}_1, \dots, \vec{x}_{n_x})$ e $Y = (\vec{y}_1, \dots, \vec{y}_{n_y})$ de puntos en \mathbb{R}^n , se dice que son linealmente separables si existe un conjunto $\{w_1, \dots, w_n, \theta\}$ con $w_i, \theta \in \mathbb{R}$, $i = 1, 2, \dots, n$ tal que:*

$$\begin{cases} \sum_{i=1}^n w_i x_i + \theta > 0 & \forall \vec{x} \in X \\ \sum_{i=1}^n w_i y_i + \theta < 0 & \forall \vec{y} \in Y \end{cases}$$

Básicamente esta definición nos dice que dos conjuntos son linealmente separables si existe un hiperplano, de tal forma que todos los elementos de un conjunto queden a un lado del hiperplano y todos los elementos del otro conjunto queden en el lado opuesto.

Definición B.5 Sea $f : \mathbb{R}^N \rightarrow \mathbb{R}$, con $N > 1$, y sea $e_i = (0, \dots, 0, 1, 0, \dots, 0)$, donde el 1 se encuentra en la componente i -ésima del vector. Se define la derivada parcial de f con respecto a la i -ésima variable en el punto a como la derivada direccional de f en a en la dirección e_i , es decir:

$$\frac{\partial f}{\partial x_i}(a) = \lim_{t \rightarrow 0} \frac{f(a + te_i) - f(a)}{t}$$

Se dice que f es parcialmente derivable en a , si existe la derivada parcial con respecto a la i -ésima variable para todo $i \in \{1, \dots, N\}$.

Definición B.6 Sea $f : \mathbb{R}^N \rightarrow \mathbb{R}$, cuando f es parcialmente derivable en a , se define el gradiente de f en a como el vector dado por

$$\nabla f(a) = \left(\frac{\partial f}{\partial x_1}(a), \frac{\partial f}{\partial x_2}(a), \dots, \frac{\partial f}{\partial x_N}(a) \right)$$

BIBLIOGRAFÍA

- [1] Chong, Eunsuk and Han, Chulwoo and C. Park, Frank, “Deep Learning Networks for Stock Market Analysis and Prediction: Methodology, Data Representations, and Case Studies”, *Expert Systems with Applications*, vol. 83, abr. de 2017. DOI: 10.1016/j.eswa.2017.04.030.
- [2] Zhou, Feng and Zhou, Hao-min and Yang, Zhihua and Yang, Lihua, “EMD2FNN: A strategy combining empirical mode decomposition and factorization machine based neural network for stock market trend prediction”, *Expert Systems with Applications*, vol. 115, jul. de 2018. DOI: 10.1016/j.eswa.2018.07.065.
- [3] Liu, Yang, “Novel Volatility Forecasting Using Deep Learning - Long Short Term Memory Recurrent Neural Networks”, *Expert Systems with Applications*, vol. 132, abr. de 2019. DOI: 10.1016/j.eswa.2019.04.038.
- [4] Naranjo, Rodrigo and Arroyo, Javier and Santos Peñas, Matilde, “Fuzzy modeling of stock trading with fuzzy candlesticks”, *Expert Systems with Applications*, vol. 93, oct. de 2017. DOI: 10.1016/j.eswa.2017.10.002.
- [5] Edgar Nelson Camperos y Alma Yolanda Alanís García, *Redes Neuronales, automática y robótica*. 2000.
- [6] Pedro Isasi Viñuelae Inés M. Galván León, *Redes de Neuronas Artificiales Un Enfoque Práctico*. 2004.
- [7] Zhang, Jing and Cui, Shicheng and Xu, Yan and Li, Qianmu and Li, Tao, “A Novel Data-Driven Stock Price Trend Prediction System”, *Expert Systems with Applications*, vol. 97, dic. de 2017. DOI: 10.1016/j.eswa.2017.12.026.
- [8] Malkiel, Burton Gordon, *A Random Walk Down Wall Street*. W. W. Norton Company, 2003 (original 1975).
- [9] John J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York Institute of Finance S., 31 dic 1998.
- [10] *Backtrader*. dirección: <https://www.backtrader.com>.
- [11] *Keras*. dirección: <https://keras.io>.
- [12] *Pandas*. dirección: <https://pandas.pydata.org>.

- [13] *Ta-Lib*. dirección: <https://mrjbq7.github.io/ta-lib/>.
- [14] *Tkinter*. dirección: <https://docs.python.org/2/library/tkinter.html>.

