Alumnos: Alejandro López Rodríguez,
Antonio Rodríguez Sánchez.
MIAX 5
Proyecto Fin de Master
Profesor: Tomás de la Rosa
e-mail: alelopezespana@gmail.com
e-mail: arodrisa@gmail.com
github: https://github.com/arodrisa/ProyectoFinal_Miax5
drive: https://drive.google.com/drive/folders/1_SJbBEJNy7jKuys0YkEQ0cCi7h_6sX_A?usp=sharing

Índice

1. Introducción

1.1. Contexto Histórico

Desde los albores de la humanidad, el ser humano ha buscado la seguridad y la prosperidad en la sociedad, donde cada miembro cumple con una función específica en pos del beneficio de la comunidad. La especialización de los trabajadores trajo como consecuencia inevitable la aparición del comercio, conocido como el intercambio de bienes y servicios entre dos o más partes. Según se fueron haciendo más grandes y complejas las sociedades humanas, el comercio tuvo que ir reinventándose y adaptándose. Ya fueron los griegos y los romanos los primeros en crear normativas para el intercambio estructurado de mercancías entre los mercaderes, mas no fue hasta la aparición de las lonjas en la Europa del S.XII donde aparecen los primeros antecedentes de las bolsas contemporáneas. Todo este auge trajo la figura de los prestamistas y de los banqueros, siendo los Caballeros de la Orden del Temple los precursores de la creación de las casas de cambio, en el futuro llamadas bancos, donde guardaban riquezas con el fin de evitar robos de los viajantes.

Finalmente, fue en el año 1602 cuando fue fundada la primera bolsa de valores del mundo, la Bolsa de Valores de Ámsterdam, creada por la Compañía Holandesa de las Indias Orientales, considerada como la primera sociedad anónima de la historia, cuyas acciones solo podían ser negociadas en la bolsa de Ámsterdam, aunque hay fuentes que consideran que la primera bolsa fue fundada en Amberes en el año 1531. El crecimiento del comercio internacional propició la creación de nuevas bolsas, teniendo como base la primera Ley de Bolsas de 1724, destacando la creación de la bolsa de París en 1724, la bolsa de Londres en 1801, la bolsa de Nueva York en 1817 y la bolsa de Tokio en 1878. Actualmente existen más de 100 bolsas en todo el mundo, y se negocian gran cantidad de activos como acciones, forex, renta fija, derivados, incluso se han creado mercados de criptodivisas. El aumento de la complejidad de los mercados unido al vertiginoso desarrollo de las herramientas informáticas, ha propiciado la introducción de nuevas tecnologías a las bolsas tradicionales. Este nuevo concepto ha revolucionado los mercados los últimos años, aumentando la velocidad de negociación de los activos, la disponibilidad de la información y la oportunidad de especulación a mucho menor plazo. Con el auge de la inteligencia artificial, los matemáticos, ingenieros e informáticos están cobrando cada vez más importancia en este sector, creando nuevos y mejorados algoritmos que superan en muchos aspectos las capacidades de análisis humanas, cambiando el status quo vigente desde la Edad Modera en el comercio de activos financieros. Pero, ¿de donde han salido todos estos nuevos conocimientos matemáticos e informáticos?

Podría considerarse que el origen de la informática data de la creación del ábaco en la Antigua China; también podría considerarse la máquina calculadora del S.XVII creada por Pascal, o la creación en el S.XIX de una máquina creada por el matemático Baggage capaz de realizar operaciones matemáticas. Pero si hablamos de ordenadores, tenemos que avanzar hasta la Segunda Guerra Mudial. Uno de los primeros ordenadores fue el *ENIAC*, enorme máquina estadounidense utilizada para lanzar precisos ataques de artillería. También podemos hablar del *Mark I*, desarrollado por *IBM*, o las máquinas Collosus, desarrolladas por Alan Turing y Thomas H Flowers, que junto con el descifrado de la máquina de transmisión de mensajes secretos nazi, *ENIGMA*, permitieron que los aliados tuvieran una gran ventaja estratégica en las operaciones bélicas de la Segunda Guerra Mundial, acortando, según dicen los expertos, dos años la guerra y salvando 20 millones de vidas. Pero el que fue el realmente considerado primer ordenador fue el *Z3*, automático y programable, creado por un ingeniero alemán en 1943. A partir de este momento, los ordenadores fueron evolucionando, haciéndose cada vez más complejos y más pequeños, lo que permitió tener gran cantidad de avances en distintos campos científicos gracias a la capacidad computacional de los ordenadores.

Estas nuevas máquinas también contribuyeron al desarrollo de la Inteligencia Artificial, que se puede definir como el campo científico de la informática que trata de crear programas y mecanismos considerados "inteligentes". Dentro de este campo destaca como primera gran invención la cración del *Perceptrón*, la primera neurona artificial creada por *Rosenblatt* en el año 1957, aunque no debemos olvidar la creación del

algoritmo de *Mínimos Cuadrado*s en el año 1821. En los siguientes años se hicieron considerables avances, como el algoritmo de clustering *Lloyd-Forgy* en 1967, los árboles de decisión por *Quinlan* en 1986 o la creación de la metodología *CRISP-DM* en 1999. En el siguiente milenio continuaron los avances, destacando la creación en 2009 *ImageNet*, base de datos de imágenes clave en el desarrollo de Visión Artificial, o un hito muy importante en el aprendizaje por refuerzo: la victoria de*Google Deep Mind* al juevo Go en el año 2016, sin olvidarnos que ya fue en el siglo pasado cuando los ordenadores empezaron a ganar al ajedrez a los grandes maestros soviéticos. En la actualidad, la Inteligencia Artificial se aplica a una infinidad de distintos ámbitos: medicina, redes sociales, urbanismo... y bolsa. La posibilidad de crear algoritmos de Deep Learning con mayor facilidad y menores tiempos computacionales ha propiciado, como ya dijimos anteriomente, la figura del matemático o informático capaz de crear modelos que mejorar la toma de decisiones humana. Este proceso está en auge, y sin lugar a dudas va a seguir creciendo durante los próximos años, y ello ha propiciado la investigación y desarrollo constante de nuevos y mejorados algoritmos, como el que veremos a continuación.

1.2. Motivación

El trabajo presentado tiene como objetivo poner en práctica los conocimeintos adquiridos en el master. En él se ven todas las partes de la creación de un algoritmo: establecer la idea, capturar los datos, limpiar y organizar los datos, realizar pruebas y modificaciones, y en caso de fracaso, analizar el porqué y explorar alternativas. También se plantean diferentes opciones para la ejecución del mismo, ya que se plantea el realizarlo en local o en cloud.

El algoritmo diseñado realizará operaciones de especulación en el mercado de acciones americanas, con el objetivo de sacar beneficios monetarios a partir de un cierto capital inicial. Este algoritmo considera acciones de empresas que están o que hayan estado en el índice bursátil SP500, sin incluir productos apalancados, productos derivados, futuros, opciones, renta fija ni ningún tipo de activo distinto a las *acciones del SP500* que se detallarán más adelante. El algoritmo ha sido pensado, diseñado, programado, optimizado y testeado por sus dos desarrolladores en el ámbito de datos históricos obtenidos de *AlphaVantage*, y no ha sido puesto en producción a tiempo real en un bróker. El objetivo final ha sido maximizar la rentabilidad obtenida manteniendo siempre un nivel de riesgo en unos parámetros establecidos, y tener un número de operaciones lo suficientemente grande como para que los resultados obtenidos en el periodo de backtest fuesen estadísticamente significativos. Para ello, se ha diseñado una estrategia basada en la reversión a la media de un ratio basado en la media ponderada de tres ratios comúnmente utilizados: el *ratio de Sharpe*, el *ratio de Calmar* y el *Profit Factor*. Las operaciones se realizan mediante un conjunto de reglas cuya base ha sido diseñada por los dos desarrolladores, y que han sido analizadas y mejoradas por un modelo de inteligencia artificial, con el objetivo de mejorar los resultados de rentabilidad y de riesgo. El modelo de inteligencia artificial ha sido obtenido de las librerías de *TensowFlow2.0* y *Keras*, pero la arquitectura final del modelo, optimización de los hiperparámetros, estructura del algoritmo y toda la parte que es ajena al propio funcionamiento interno del algoritmo ha sido desarrollada por los dos autores de este trabajo. A continuación se va a detallar todo el proceso que ha dado lugar al algoritmo final y a sus resultados.

Este trabajo es puramente de índole académica, y en ningún caso representa ninguna recomendación de inversión ni pretende asesorar a posibles inversores para especular en los mercados financieros. La operativa en bolsa conlleva un alto riesgo, sobre todo para aquellos que no tienen la debida cualificación, y por eso se repite que este trabajo no pretende que ningún inversor arriesgue su capital o el de terceros en base a lo aquí expuesto. Por ello, los autores del trabajo declinamos todo tipo de responsabilidad por las pérdidas materiales y morales que cualquier persona pudiese sufrir por realizar cualquier tipo de actividad económica o financiera en base a lo que aparezca en este trabajo de fin de Máster.

2. Obtención de los datos

La obtención de los datos se ha realizado desde la plataforma *AlphaVantage*. Tras capturarlos, se han subido aun storage de Google Cloud y se han cargado en una base de datos. La decisión de tomar los datos de AlphaVantage se tomó tras analizar varias fuentes de datos, y corroborar que las series históricas descargadas tenían en todos los casos analizados dividendos, splits y contrasplits, tal y como anuncian.

Una muestra de los datos descargados corresponde a la imagen 1:

	timestamp	open	high	low	close	adjusted_close	volume	dividend_amount	split_coefficient
0	2021-04-07	133.84	134.94	133.78	134.93	134.930000	2976136	0.0	1.0
1	2021-04-06	135.58	135.64	134.09	134.22	134.220000	3620964	0.0	1.0
2	2021-04-05	133.64	136.69	133.40	135.93	135.930000	5471616	0.0	1.0
3	2021-04-01	133.76	133.93	132.27	133.23	133.230000	4074161	0.0	1.0
4	2021-03-31	134.54	134.71	132.71	133.26	133.260000	4945315	0.0	1.0
...	...	...	...	...	...	...	...	...	...
95	2020-11-17	117.60	118.54	117.07	117.70	116.149438	4134455	0.0	1.0
96	2020-11-16	118.30	118.55	117.12	118.36	116.800744	5293385	0.0	1.0
97	2020-11-13	115.19	117.37	115.01	116.85	115.310636	4683512	0.0	1.0
98	2020-11-12	115.63	116.37	113.48	114.50	112.991595	6500799	0.0	1.0
99	2020-11-11	118.12	118.35	116.22	117.20	115.656025	4289601	0.0	1.0

Figura 1: Muestra de datos de Alphavantage

En el caso de esta plataforma, han incluido los dividendos, splits y contrasplits en la serie "adjusted_close". Para comprobar que tienen los split y contrasplit, se ha comprobado que no haya saltos, esto se mostrará en más detalle en el capítulo siguiente.

Por otro lado, para comprobar que incluyen dividendos, se va a tomar como ejemplo el dividendo cobrado por AAPL el 2020-11-06. En la imagen 2 se puede observar los datos correspondientes a los dias anteriores y posteriores a la fecha de ejecución del dividendo. Y se puede corroborar que la diferencia entre el "close" y "adjusted_close" los días anteriores a la ejecución del dividendo tienen una diferenia del importe del dividendo.

	timestamp	open	high	low	close	adjusted_close	volume	dividend_amount	split_coefficient
45	2020-11-11	117.19	119.63	116.4400	119.49	119.490000	112294954	0.000	1.0
46	2020-11-10	115.55	117.59	114.1300	115.97	115.970000	138023390	0.000	1.0
47	2020-11-09	120.50	121.99	116.0500	116.32	116.320000	154515315	0.000	1.0
48	2020-11-06	118.32	119.20	116.1300	118.69	118.690000	114457922	0.205	1.0
49	2020-11-05	117.95	119.62	116.8686	119.03	118.824767	126387074	0.000	1.0
50	2020-11-04	114.14	115.59	112.3500	114.95	114.751802	138235482	0.000	1.0
51	2020-11-03	109.66	111.49	108.7300	110.44	110.249578	107624448	0.000	1.0

Figura 2: Datos dividendo AAPL

2.1. Descarga de datos

La API de *AlphaVantage* funciona realizando llamadas al endpoint realizadas componiendo URLs dependiendo de lo que se solicite.
El resultado vendrá en formato csv o json dependiendo de la función. En algunos casos la opción de seleccionar un csv o un json será opcional, y en otros vendrá dado.

Se ha creado una librería propia que interactúa con la API. Para ello, se han creado funciones, basándose en la documentación, para posteriormente ejecutar las que fueran necesarias.

Aunque se han utilizado más funciones, se han creado más llamadas al endpoint, para mostrar cómo funciona la API, se va a mostrar el proceso con dos de las funciones utilizadas:

- Listing & Delisting Status
- Intraday (Extended History)

En primer lugar hay que obtener una APIKey en la web, ya sea la versión gratuita o la de pago. En nuestro caso para montar la librería optamos por la libre. Pero para descargar posteriormente la información en un tiempo prudente, decidimos pagar una mensualidad y así poder descargar todos los datos en un tiempo prudencial.

En segundo lugar obtendremos la URL "base", que será la que se utilizará para ir añadiendo los diferentes condicionantes de las peticiones:

https://www.alphavantage.co/query?

Los siguientes pasos serán añadir parámetros según la función que se quiera utilizar.

2.1.1. Listing & Delisting Status

Esta función indica los activos listados y deslistados para una fecha específica. Utilizando esta función podemos descargar todos los activos americanos que han sido listados en la historia, evitando así el sesgo de supervivencia.

Esta función tiene los siguientes parámetros:

- Obligatorios:
  - function: indica la función a utilizar. En este caso sería: function=LISTING_STATUS.
  - apikey: indica la apikey del usuario, siendo: apikey=fool_apikey.
- Opcionales
  - date: se le pasa la fecha que se desee comprobar los activos a partir de 2010-01-01, si se pasa vacío se tomará el último día. Se indicará como: $date = 2020 - 12 - 31$.
  - state: este parámetro establece si se quieren pasar activos listados ("active") o deslistados ("delisted"). Por ejemplo: $state = active$.

Tras indicar los parámetros en la función, se unen utilizando un ampersand ("&").
La respuesta del endpoint es un csv con la información del simbolo, nombre, bolsa, tipo de activo, fecha en la que se deslistó y status (activo o deslistado).

2.1.2. Daily Adjusted

Esta función permite descargar las series históricas ajustadas para un ticker dado. Devuelve los datos de los valores de: open, high, low, close, volumen, close_adjusted, dividendos y factor de ajuste por split.
Los parámetros para llamarla son los siguientes:

- Obligatorios:
  - function: indica la función a utilizar. En este caso sería: function=TIME_SERIES_DAILY_ADJUSTED.
  - symbol: indica el ticker a descargar, siendo: symbol=AAPL.
  - apikey: indica la apikey del usuario, siendo: apikey=fool_apikey.
- Opcionales
  - outputtype: hay dos opciones, "compact" y "full". Compact devuelve los últimios 100 datos y full devuelve todo el histórico para ese ticker. En este caso sería: $outputsize = compact$.
  - datatype: este parámetro establece el formato de retorno de los datos, pudiendo optar entre "json" o "csv", en este caso: $datatype = csv$.

2.2. Almacenamiento y Cloud

Una vez descargados los datos, para almacenarlos, se ha creado el circuito que sigue los siguientes pasos:

1. Descarga listado: En primer lugar se descarga un listado de los tickers de todos los activos, tanto listados como deslistados.
2. Descarga fichero de cada ticker.
3. Se comrpueba que los ficheros no estén vacíos, y en caso de que no lo esté se le añade una columna en la que contenga el nombre del ticker. Si está vacío, se descarta.
4. Una vez descargado y añadida la columna, el fichero se guarda en un bucket.
5. Cuando ha terminado el proceso de descarga se cargan en una tabla de "bigquery" a la que se irá accediendo para ir realizando los test.

Para la realización del test, que se explicará a continuación, se empezó trabajando con tablas de "bigquery", pero al comprobar que las llamadas y descarga de datos tardaba unos 30 segundos, para agilizar el proceso de desarrollo, se optó como solución funcional descargarlos en un fichero. Así que contamos con las dos opciones, descarga de "bigquery" o lectura de fichero de texto plano.

3. Tratamiento de los datos

Una vez descargados, los datos se han analizado y se ha comprobado que no sólo se hayan descargado correctamente, sino que sean correctos, coherentes y carentes de posibles errores en el histórico.
Para ello, en primer lugar vamos a homogeneizar las fechas, controlar splits y outliers, y una posterior batería de checks.

3.1. Homogeinización

En primer lugar, homogeneizamos las fechas, es decir, vamos a hacer que todos los activos contengan las mismas fechas. Para ello existen varias opciones:

1. Coger todos los días laborables entre la primera y última fecha de la serie y quitar los festivos
2. Coger todas las fechas comunes de todos los activos

En nuestro caso la primera opción era inviable porque no disponíamos del calendario de festivos de los años correspondientes a los datos seleccionados.
Por tanto, hemos optado por la segunda opción. Para ello hemos lanzado una query de todas las filas y hemos cogido los campos únicos de fecha y posteriormente hemos incluido en cada uno de los activos las fechas requeridas.

3.2. Splits y Outliers

Tras tener unificados todas las fechas, hemos realizado un test para comprobar que no hubiera Splits sin corregir por la plataforma. Para ello hemos comprobado que no hubiera ningún salto en la serie tal que se duplique o se divida por dos su precio, es decir que haya un split o un contra split; o sea 1:2 o 2:1, en cada caso.
Para esto hemos comprobado que los precios no se hayan multiplicado de un día para otro por 2 o dividido por dos, lo que es lo mismo retornos logarítmicos absolutos sean menores de 0.69; esto viene representado en la ecuación 1

$$\ln\left(\frac{Price_{A_T}}{Price_{A_{T-1}}}\right) \leq 0.69 \tag{1}$$

Tal y cómo se ha planteado no valdría para comprobar si los activos han hecho un split o contrasplit, sino que valdría para comprobar outliers. Ya que si, por ejemplo, comprobamos que sube un día y baja al día siguiente será un outlier.
Por lo tanto, una vez implantado el check de outliers, hay que añadirle la opción de que si se da un salto y los días posteriores se mantiene dicho salto, entonces si habrá un split. En la imagen 3 se puede observar que el ticker MIL tiene varios outliers, y cómo se pueden observar no son split ya que el precio no se mantiene.
El tratamiento de esta casuística consistirá en que, vez detectados, se compararán con otras fuentes de datos (“Investing”,“Yahoo finance”, “Google finance”, “Reuters”) y se comprobará si de verdad ocurrieron o no. En el caso de que ocurrieran será un trabajo de análisis el que determinará si son relevantes o no. Ya que por ejemplo en octubre de 2008 pasó con el ticker VW, pero fue una anomalía que rápidamente se corrigió. En el caso citado de la figura 3 tras comprobarlo, se observó que no era correcto y se corrigió repitiendo los valores anteriores.

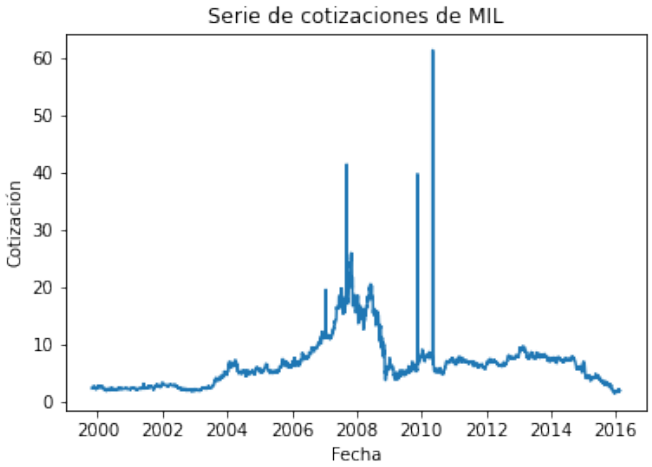


Figura 3: Serie de cotizaciones de MIL

3.3. Resto Checks

Una vez subsanados los errores que pueden darse por posibles fallos en el tratamiento de los datos capturados por mercado y la homogeinización de los mismos, vamos a correr una batería de test que van a tratar diferentes problemas que puedan existir ya no sólo por la falta de datos, sino por posibles errores en las capturas o el mantenimiento de los mismos.

3.3.1. Check horizontales

El siguiente paso es comprobar que no haya acciones que presenten repeticiones de precios constantes, o que la serie parezca "escalonada".
Para ello, comprobaremos que no hay más de 10 repeticiones de precio, es decir que no haya 10 retornos iguales a cero consecutivos.
Aunque en el set de acciones del SP no hemos encontrado casos, si que los hemos visto al comprobar todas las acciones. En este caso, había unas cuantas de test que se notaban que estaban escalonadas y no tenían mucho sentido. Por ejemplo en la figura 4 se puede observar que al final tiene una línea horizontal.

3.3.2. Check zeros

El objetivo de este check es comprobar que no haya acciones con valores cero. En esta ocasión queremos analizar que no exista ningún caso, vamos a querer analizar todos, ya que consideramos que todos los activos que coticen un cero es un error. Así que siempre vamos a querer comprobarlo para descartarlo en caso de que sea correcto y la empresa haya hecho default. En esta ocasión no hemos encontrado ningún valor con cero en los activos del SP500.

3.3.3. Check retorno logaritmico

El objetivos de este check es comprobar que no haya repeticiones de retornos logarítmicos en la serie debidos a posibles errores en la obtención de las series.
En esta ocasión se ha comprobado que no haya más de 10 retornos con el mismo retorno logarítmico. Con esto nos quitamos también las repeticiones.

Figura 4: Serie de cotizaciones de PCS

3.3.4. Check retorno absoluto

Este caso es el mismo que el anterior, con la única diferencia que comprobamos que los retornos absolutos, por si la plataforma utilizara retornos absolutos en vez de logarítmicos.

3.4. Limpieza

Una vez realizadas todas las modificaciones y checkeos se procede a arreglar las series que sean necesarias. En el caso de los outliers se eliminan repitiendo el último valor válido.
En el resto de casos, las funciones nos devuelven los resultados y se analizan visualmente y se interpreta lo que ha podido pasar. Como resumen se ha visto que el final de la serie de 'MHS' presentaba muchas repeticiones porque dejó de cotizar en 2013.
Por otro lado, "DNB" y "NVLS" suspendieron su cotización puntualmente y luego prosiguieron, por lo que se han creado dos series. En la figura 5 se puede observar el caso de DNB, que dejó de cotizar el 2019-02-15 y volvió a cotizar el 2020-07-01 y al dibujar la imagen se puede observar un fuerte salto en la misma representado por una linea diagonal que une ambos precios en ambas fechas. Esto sucede por cómo el paquete de plotear las imágenes de pandas trata las discontinuidades.

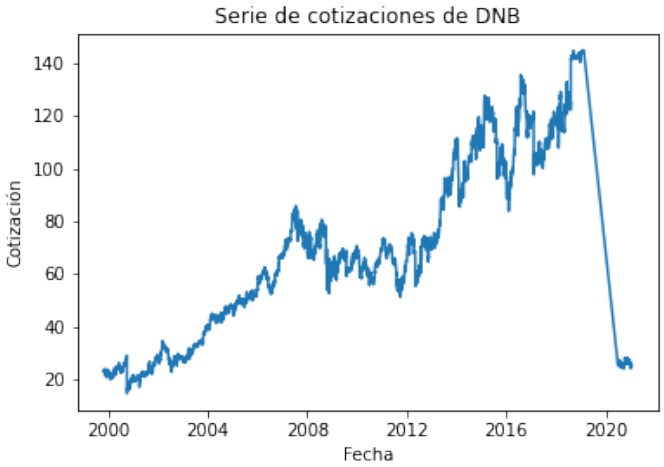


Figura 5: Serie de cotizaciones de DNB

Por último comentar que sólo se han reportado estos fallos de las 699 analizadas. Como se contará más adelante, se intentó hacer un primer algoritmo fallido que implicaba el uso de todas las series, y en ese caso sí que había más problemas ya que había series de test y series con datos muy pobres que requerían mucho más análisis. En total presentaban anomalías unas 500 series históricas.

4. Planteamiento Estrategia

4.1. Introdución

Ahora que ya tenemos la data homogeneizada, checkeada y perfectamente estructurada, podemos empezar a plantear nuestro algoritmo. Como ya mencionamos anteriormente, vamos a implementar un modelo basado en la reversión a la media de ratios aplicados a nuestras empresas. Para la creación de algoritmos de inversión con Intligencia Artificial vamos a procurar tratar con series estacionarias, pues dan mucha más información, y las redes neuronales pueden aprender mucho más de ellas que de series no estacionarias. Además, por su propiedad de estacionariedad, asumimos que presentan una reversión a la media, característica de la que nos vamos a aprovechar para sacar ventaja al mercado. Esta estacionariedad puede ser media por el Test Aumentado de Dickey-Fuller, un modelo autorregresivo que nos indica cuan estacionaria o explosiva es una serie temporal. Sin embargo, tras un estudio de los ratios en gran cantidad de las empresas, no vamos a incluir este modelo en nuestro algoritmo, pues vamos a dejar que las redes neuronales aprendan de otro tipo de variables.

4.2. Ratios

Se han considerado diferentes ratios que se calculan a partir de la serie de cotizaciones de un activo, como por ejemplo el porcentaje de días alcistas respecto al de días bajistas o la esperanza matemática, pero tras un exhaustivo análisis, se han decidido utilizar ratios que se centren más en los movimientos del precio que en la proporción de sesiones alcistas. Los tres ratios que hemos escogido son:

- **Ratio se Sharpe:** ratio que mide el riesgo en relación a la rentabilidad. La rentabilidad libre de riesgo se ha asumido como 0 %. Su ecuación es: 2

$$Sharpe\ Ratio\ = \frac{R\ -\ R_f}{\sigma} \tag{2}$$

- **Ratio de Calmar:** ratio que mide la rentabilidad obtenida en función del mayor DrawDown de la ventana. Su ecuación es: 3

$$Calmar\ Ratio\ = \frac{Annualized(R_p)}{MaxDD_p} \tag{3}$$

- **Profit Factor:** es el cociente entre las ganancias netas de los días positivos entre las pérdidas netas de los días negativos. Su eciación es: 4

$$Profit\ Factor\ = \frac{\sum_{over\ all\ positive\ returns} netreturn}{\sum_{over\ all\ negative\ returns} netreturn} \tag{4}$$

Estos ratios requieren un hiperparámetro: la ventana temporal sobre la que serán calculados. Inicialmente hemos seleccionado una ventana móvil de 120 datos, que la iremos modificando según avancen las pruebas del algoritmo. Consideramos como "dato" un día con cotizaciones de las empresas, es decir, en esta ventana no se incluyen ni fines de semana, festivos, ni cualquier día que fuese eliminado en la parte de homogeinización de la data. Una ventana más corta hará que los ratios reaccionen más rápidamente a los cambios en el precio, mientras que una ventana más larga provocará cambios más lentos. Teóricamente, los valores más altos de la ventana(hasta llegar a cierto umbral) se suelen relacionar con operaciones o sucesos más seguros y con menos ruido, sacrificando velocidad y oportunidades, mientras que las ventanas más cortas se relacionan con una mayor cantidad de oportunidades que se detectan más rápidamente, pero esto trae como consecuencia más falsas señales y ruido. Escoger la ventana óptima es una tarea de suma importancia, pero siempre intentando no caer en la sobreoptimización.

Como ya hemos comentado anteriomente, estos tres ratios son estacionarios. El ratio de Sharpe y el Profit factor son muy parecidos, aunque suelen diferir en la magnitud de sus valores. Sin embargo, el ratio de Calmar presenta picos que lo convierten en una serie heterocedástica; estos picos son debidos precisamente a los DrawDown, cuyas oscilaciones propician que la serie varía en gran medida su valor. Podemos ver que esta serie no oscila tanto respecto a su media, sino que tiene una base y unos picos cada cierto tiempo que responden a los DrawDown. En la figura 6 se pueden observar ejemplos de estos tres ratio.

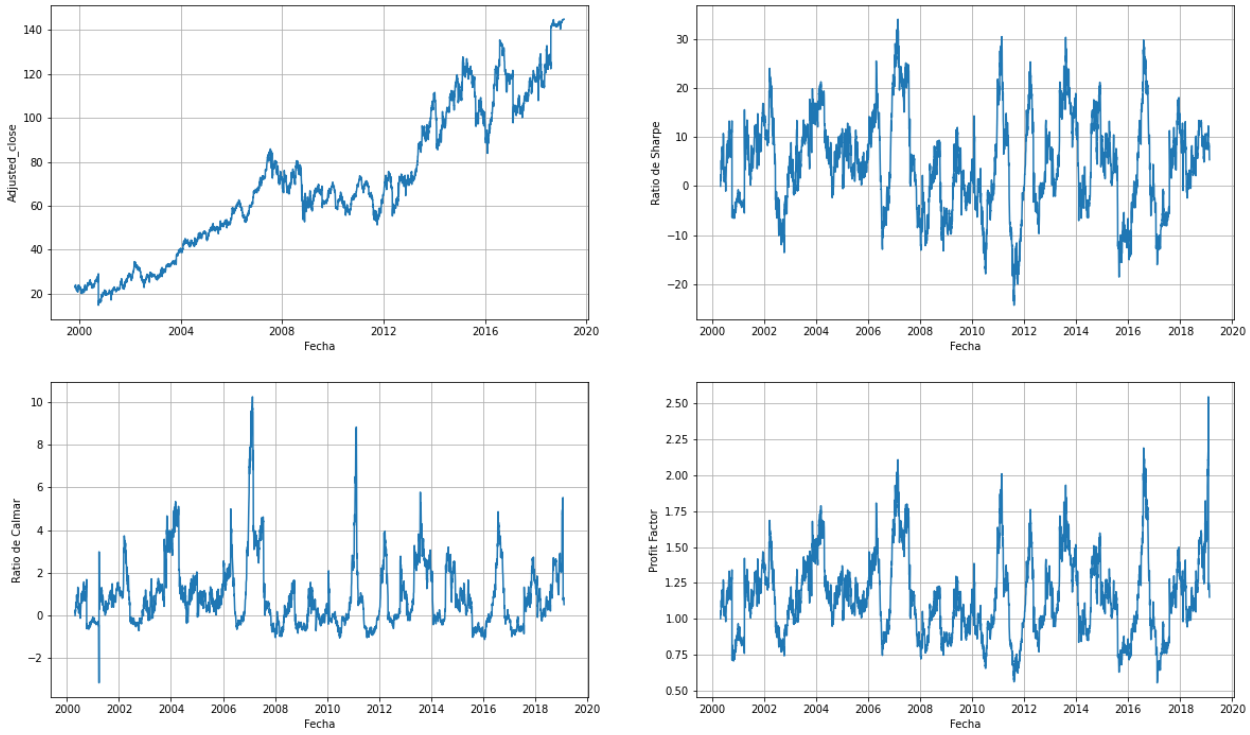


Figura 6: Ejemplo de los ratios

Tras escoger estos tres ratios, vamos a intentar convertirlos en uno solo que sea capaz de aglutinar gran cantidad de información sobre los datos pasados de nuestras series financieras. Para ello primeramente vamos a normalizar los ratios y así igualar sus magnitudes. En este punto debemos mirar un poco hacia el futuro: cuando vayamos a entrenar nuestro modelo de IA, deberemos separar nuestra data en Train y Test, y cuando hagamos las predicciones sobre Test, deben ser lo más realistas posibles. Por ello, no podemos normalizar nuestros ratios con la media y la varianza de todo nuestro dataset, sino que debemos hacerlo solo con el conjunto de entrenamiento. Usamos un objeto de la clase StandardScaler de la librería scikit-learn de Python, el cual lo entrenaremos con las series de los 3 ratios de todas las empresas de nuestro dataset, pero solo con la data hasta el día 31 de Diciembre de 2015, pues vamos a dejarnos los últimos 5 años de data para el conjunto de Test. Una vez entrenado el StandardScaler, vamos a transformar los ratios de nuestras empresas de manera que sus magnitudes sean similares.

Para crear un "ratio definitivo", el cual llamaremos simplemente *Ratio*, vamos a realizar una suma ponerada de nuestros tres ratios, dando lugar a la función objetio, representada por el ratio de la ecuación 5.

$$Ratio = Sharpe\ Ratio \cdot 0.45\ +\ Calmar\ Ratio \cdot 0.1 + Profit\ Factor \cdot 0.45 \tag{5}$$

Se han probado distintos valores para las betas, y al final hemos decidido usar una beta de 0.45 para el ratio de Sharpe, 0.45 para el Profit Factor, y 0.1 para el ratio de Calmar, para así tener en cuenta los picos de este último pero evitar que afecten en demasía a la estacionariedad y homocedasticidad del Ratio. El resultado es el observado en la imagen 7:

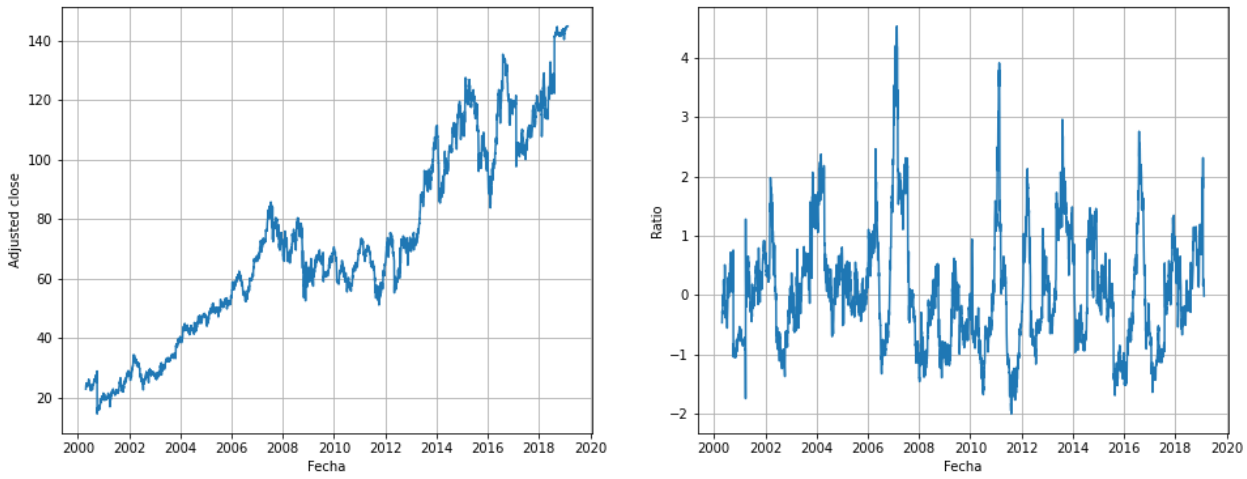


Figura 7: Ejemplo del ratio

Este ratio está altamente correlacionado con el precio, de manera que la primera idea podría ser estar en largo cuando el ratio esté subiendo, y mantenernos fuera de mercado cuando el ratio esté bajando(para empresas individualmente). Sin embargo, hay una gran cantidad de posibles estrategias que se podrán seguir teniendo en cuenta la estacionariedad del ratio y su correlación con el precio. En concreto en este trabajo hemos desarrollado dos: una con éxito y otra sin éxito. Primeramente intentamos seleccionar cada x meses cuales de nuestras empresas estarían en tendencias alcistas más robustas, y para ello intentamos predecir el valor del Ratio para dentro de esos x meses. Tras no conseguirlo, decidimos pasar directamente a una estrategia más especulativa; buscar reversiones a la media desde valores inferiores de nuestro ratio para realizar operaciones de compra, obteniendo un cierto éxito. Estas dos estrategias se van a desarrollar posteriormente, pero ahora vamos a centrarnos en uno de los aspectos clave de cualquier algoritmo de inversión y/o proyecto de Inteligencia Artificial: las características.

4.3. Creación de variables para la modelización del *Ratio*

En problemas estándar de inteligencia artificial sobre variables aleatorias(como el dataset del Titanic o el Boston Housing) las variables que nuestros modelos utilizarán como inputs ya nos vienen dadas. Sin embargo, cuando se trata de bolsa, y más aún cuando trabajamos directamente con series de cotizaciones, esas variables las debe de crear el propio desarrollador. Es un trabajo tanto de entendimiento de la serie como un trabajo creativo, pues no es ni fácil ni automático crear variables que puedan tener influencia sobre el posterior desarrollo de nuestra serie en cuestión. Sin embargo, tenemos algo a nuestro favor: la serie es estacionaria, y, como ya dijimos con anterioridad, podemos usar esto en nuestro favor. A continuación vamos a especificar que variables hemos creado, por qué motivo, y como funcionan. Estas variables se calcucan en base a la serie del *Ratio*, es decir, cuando a partir de este punto nos refiramos a la serie, estaremos referenciando a la serie del *Ratio*.

- **Diferencias absolutas respecto a datos pasados:** vamos a ver cual ha sido la variación de la serie respecto a n datos en el pasado. Esta diferencia no va a ser porcentual, pues al tratarse de una serie estacionaria con varianza bastante constante, no debemos tomar sus variaciones porcentuales, sino en valores estándar. Utilizar diferencias porcentuales sería un craso error, cuyo summum se alcanzaría cuado los datos estuviesen muy cerca del valor cero. Hemos escogido tres valores para n, siendo n = 120, 80, 40. Cuando modifiquemos en el futuro el valor de la ventana, estos tres valores de n matendrán la proporción. Los nombres de estas variables son "diff_120", "diff_80", "diff_40".

- **Desviación típica:** vamos a crear dos variables que nos digan a cuántas desviaciones típicas se encuentra el precio respecto a su media. El periodo usado para calcular la media(en este caso vamos a usar un promedio simple) y la desviación típica van a ser dos: 120 y 60. Igual que antes, estos dos valores mantendrán su propoción en el caso de variar la ventana del *Ratio*. Los nombres de estas variables son "desv_120" y "desv_60".

- **Percentiles:** creamos dos variables que nos indican en qué percentil se encuentra el actual valor de la serie respecto a una ventana. Estas dos ventanas serán de tamaño 120 y 60(igual que con las desviaciones típicas). La desviación típica y el percentil miden la misma característica de la serie, pero vamos a incluir los dos pues su naturaleza es distinta, además de que la serie de percentiles varía con mucha más celeridad que la serie de desviaciones. Los nombres de estas variables son "perc_120" y "perc_60".

- **Diferencias entre promedios exponenciales:** nos calculamos 3 medias móviles exponenciales, cuya fórmula es 6. A continuación, nos creamos dos variables, que serán la diferencia entre la EMA rápida y la media, y la diferencia entre la EMA media y la lenta. Desde el punto de vista del análisis técnico, cuando una media móvil rápida se encuentra por encima de una más lenta, es indicativo de tendencia alcista; este es el motivo por el que creamos estas dos variables, para ver cual es la tendencia actual en dos plazos distintos. Los periodos de las EMAs son 120, 60 y 25, para así ver a medio y corto plazo la fuerza de la tendencia alcista. Los nombres de estas variables son: "diffemas_120_60", "diffemas_60_25".

$$\begin{aligned} Smoothing &= \frac{2}{1+Days} \\ EMA_T &= Ratio_T * Smoothing + EMA_{T-1} * (1 - Smoothing) \end{aligned} \tag{6}$$

- **La serie del Ratio:** al tratarse de una serie estacionaria, la propia serie puede actuar como variable. El nombre de esta variable es: "ratio"

4.4. Análisis Exploratorio de Datos

Ahora vamos a realizar un breve Análisis Exploratorio de Datos(EDA) sobre nuestras variables de entrada, basadas en el *Ratio* con ventana de 120 días. Lo primero es analizar cada una de las variables respecto a su media, desviación típica y percentiles, como se puede observar en la tabla 8.

	ratio	diff_120	diff_80	diff_40	desv_120	desv_60	perc_120	perc_60	diffemas_120_60	diffemas_60_25
mean	-0.009361	0.007682	0.008691	0.004053	0.092516	0.071708	0.510850	0.515866	0.001063	0.001476
std	0.983618	1.379550	1.151283	0.848239	1.402345	1.387681	0.348113	0.346428	0.242630	0.242020
min	-2.941681	-9.609536	-9.618976	-8.948220	-10.223565	-7.309730	0.008333	0.016667	-1.279557	-1.744700
25%	-0.721633	-0.876559	-0.712485	-0.498022	-1.020885	-1.024126	0.166667	0.166667	-0.164137	-0.154702
50%	-0.136811	-0.005216	0.022584	0.016083	-0.059404	-0.002798	0.516667	0.516667	-0.014293	-0.006291
75%	0.559742	0.889091	0.735013	0.507673	1.152836	1.132379	0.858333	0.866667	0.152477	0.145382
max	12.718013	12.364238	11.304510	9.906897	9.373618	7.398733	1.000000	1.000000	2.250778	2.414337

Figura 8: Descripción de las variables

El *Ratio* se distribuye alrededor del valor cero un poco orientado hacia la parte negativa(esto se debe a que se construyó a partir de 3 ratios normalizados). Vemos como el máximo es 12.71 y el mínimo -2.94, esto es consecuencia de que algunas empresas hayan tenido subidas muy acentuadas en sus cotizaciones. Las tres variables ''diferencia" tienen su media mayor que cero, lo que implica una mayor presencia de tendencias alcistas. Las desviaciones típicas y los percentiles también se orientan más hacia valores positivos(en el caso de los percentiles a valores mayores de 0.5) al igual que las diferencias entre las EMAs, denotando una mayor fuerza de los movimientos alcistas respecto a los bajistas. Veamos ahora la matriz de correlaciones.

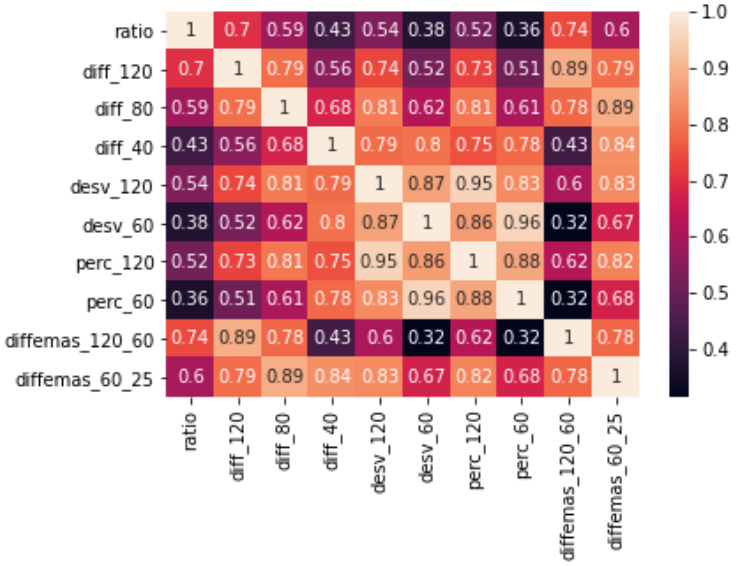


Figura 9: Matriz de correlaciones

Como se puede observar en la image 9, todas las correlaciones son positivas, como cabía esperar. Las variables que están más correladas con el Ratio son las diferencias respecto a sí mismo y las diferencias de las EMAs, siendo mayor aquellas con periodos más largos, puesto que se asemejan más a la ventana del *Ratio* de 120 valores. Cuanto menor sea la ventana de las variables, menor correlación con el *Ratio* tienen. Finalmente, vamos a utilizar el diagrama boxplot (conocido como diagrama de caja y bigotes), figura 10 para ver la distribución de los valores atípicos de las variables.

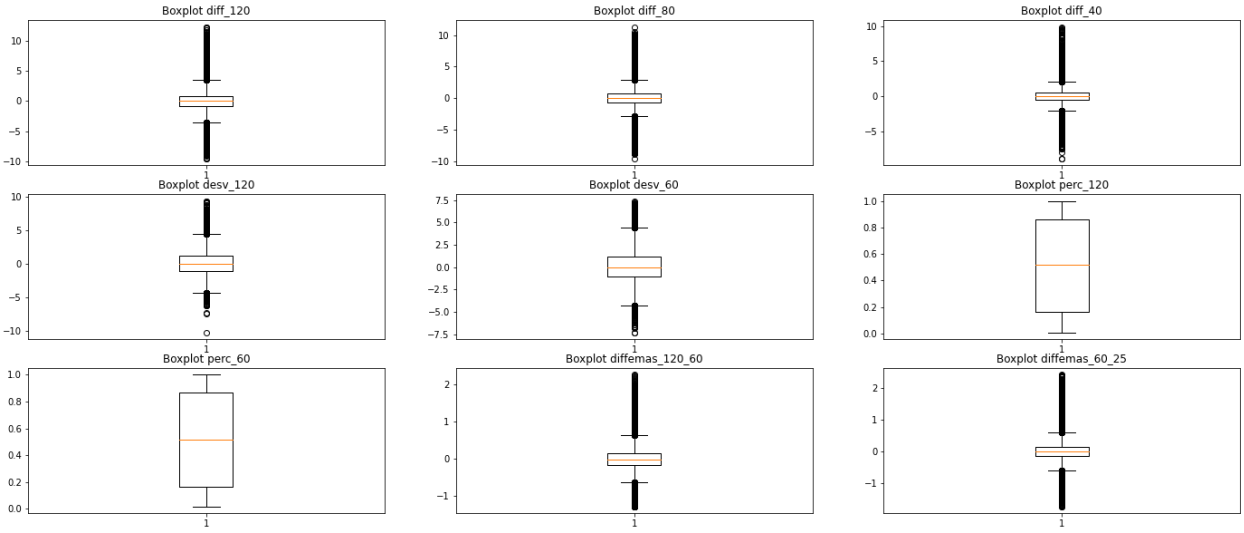


Figura 10: BoxPlots de las variables

Al igual que en la imagen 8, podemos ver como hay datos que se alejan en gran medida de su media. Al tratarse de series bursátiles, es normal la presencia de importantes outliers. En las primeras variables vemos como los valores atípicos se distribuyen de manera equitativa entre la parte superior y la inferior, sin embargo, las diferencias entre EMAs presentan mayor cantidad de outliers en su parte superior, denotando nuevamente una mayor fuerza de las tendencias alcistas. Finalmente veamos el boxplot del ratio en la imagen 11:

Este caso es mucho más extremo, aunque ya nos los esperábamos tras ver la imagen 8. La presencia de tendencias alcistas sumamente acentuadas y estables promueve la aparición de gran cantidad de outliers con valores extremadamente altos, presentes en los picos de volatiliad del Ratio, posiblemente marcados por su componente el Ratio de Calmar.

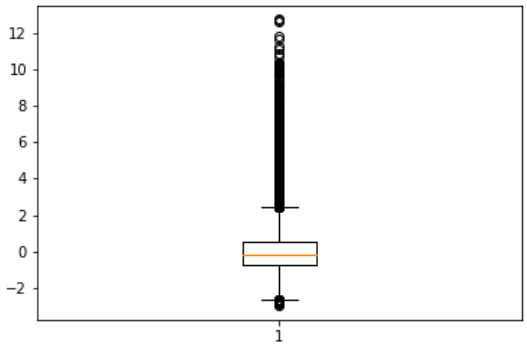


Figura 11: BoxPlots del ratio

Una vez creadas las variables, solo nos quedan por calcular las etiquetas. Puesto que hemos trabajado con dos estrategias distintas, también hemos creado dos etiquetas distintas para cada uno de los algoritmos. Cada algoritmo tiene una forma específica de calcular lo que queremos predecir; el algoritmo de selección de activos pretende ser un problema de clasificación o de regresión en el que queremos saber para cada empresa cual será el valor de su Ratio dentro de 120 días. El algoritmo de reversión a la media intenta predecir cuando subirá el Ratio tras una fuerte caída. El cálculo de las etiquetas es complejo, por ello lo detallaremos en la sección correspondiente de cada uno de los algoritmos.

Definidas las variables, vamos a pasar a explicar los dos modelos diferentes que hemos implementado.

5. Algoritmo 1: Selección de Activos

5.1. Introducción

En este primer modelo pretendíamos hacer una selección de las mejores acciones para los próximos 120 días cotizados, es decir, coger nuestro dominio de varios cientos de acciones(pues este número va variando) y un día determinado seleccionar las 30 que tendrían un "mejor comportamiento" durante los próximos 120 días. Este "mejor comportamiento" lo medimos con nuestro *Ratio*, con el que intentamos ver la tendencia futura de cada uno de los activos. Tras realizar esta selección podríamos crear un segundo modelo para la asignación de capital solo con estas 30 empresas, que irían cambiando cada 120 días. Sin embargo, como ya dijimos anteriomente, el modelo de selección de activos no llegó a funcionar, por lo que no implementamos este segundo modelo de asignación de capital sobre estas 30 empresas seleccionadas.

Este modelo se ha llevado a cabo con dos estrategias distintas: usando un problema de regresión y usando un problema de clasificación.

En el problema de regresión, vamos a crear una red neuronal que nos intente predecir cuál va a ser el valor numérico del *Ratio* de cada una de las empresas activas en el momento actual para dentro de 120 días. No pretendemos acertar de manera exacta el valor del ratio(cosa que es matemáticamente imposible, pues se tratan de valores continuos) pero asumimos que las predicciones tendrán un margen de confianza aceptable, el cual podremos usar para rankear las empresas. De esta manera, cada 120 días ejecutaríamos nuestro modelo, obtendríamos los valores del Ratio predicho, y podríamos rankear las empresas de mejores a peores. Ello implica que vamos a tener que consturirnos varios conjuntos de entrenamiento y de test.

Escogimos como fecha de inicio del primer conjunto de test el 2 de Enero de 2015. Eso signfica que si estuviésemos en producción, al cierre de mercado del último día cotizado entrenaríamos nuestro modelo de Inteligencia Artificial con la data de mercado hasta ese día, y realizaríamos una predicción de los valores del *Ratio* de cada una de las empresas para 120 días después del 2 de Enero. Tras rankearlas, cogemos las 60 mejores y las seleccionaos para nuestra certera de activos teniendo en cuenta sus correlaciones (es decir, no nos vamos a coger las 30 con mayores ratios predichos, sino que también vamos a procurar que las empresas estén descorreladas, aun a pesar de no seleccionar todas las stocks del top 30). Este proceso lo iríamos repitiendo de nuevo cada 120 días, modificando los conjuntos de train y test. Vamos añadiendo la nueva data disponible al conjunto de train, mientras que el de test son siempre los próximos 120 días respecto al día inicial de test. Es importante remarcar que no hemos entrenado con aquellos datos cuya predicción cae en el conjunto de test, es decir, que no entrenamos con los últimos 120 datos del conjunto de entrenamiento, puesto que en el modelo en producción, al no tener los datos de test disponibles, no conoceríamos las etiquetas de esos 120 datos. Esto mismo lo hemos hecho para el resto de algoritmos descritos en este trabajo.

También hemos de destacar que este proceso iterativo influye en la creación del *Ratio*. Recordemos que normalizábamos los 3 ratios iniciales (Sharpe, Calmar y ProfitFactor) con un StandardScaler; puesto que vamos creando nuevos conjuntos de train y test, también hemos modificado nuestro StandarScaler, de manera que nos creamos un nuevo objeto de esta clase con cada nuevo conjunto, y los vamos entrenando con sus correspondientes conjuntos de entrenamiento. De esta manera estamos haciendo los modelos de igual manera que como si estuviésemos en producción, entrenando el StandardScaler solo con los datos de train disponibles en cada momento.

5.2. Implementación del algoritmo

Ahora pasamos a la implementación del algoritmo. Al tratarse de series temporales, nuestros modelos de inteligencia artificial pueden usar capas convolucionales y capas recurrentes, además de densas. Por ello hay un nuevo parámetro a tener en cuenta: el LAG. Llamamos LAG al tamaño de la nueva ventana que utilizaremos para crearnos las series de cada característica que le pasaremos como input a la red neuronal. Este LAG lo usará la red para tener en cuenta no solo los valores actuales de las variables, sino también los pasados. A mayor LAG, más valores pasados de la serie tendrá en cuenta. Hemos probado modelos usando cuatro valores de LAG: 30, 60, 90 y 150. Inicialmente vamos a trabajar con un LAG de 60, aunque todo el desarrollo de a continuación, excepto la selección final de la mejor arquitectura de la red, es igual para los 4 valores.

Inicialmente nos encontramos con nuestro primer conjunto de entrenamiento (con data desde el año 2000 hasta finales del 2014) y con nuestro primer conjunto de test (data desde el primer día cotizado de Enero del 2015 hasta 120 días después). En el conjunto de train hemos separado un 30 % de los datos para validación. Tras realizar la división en estos conjuntos (recordamos que los últimos 120 días de data no los utilizamos para entrenar, puesto que en producción no dispondríamos de sus etiquetas correspondientes) normalizamos las 10 variables de que disponemos con un StandardScaler entrenado con la data de train. Después hacemos el reshape de los datos metiéndole la ventana de tamaño LAG, y ya podemos utilizar esta data como input para nuestra red neuronal.

La arquitectura de la red neuronal ha sido diseñada tras un largo proceso de optimización manual de los hiperparámetros. No han sido usadas herramientas de AutoMachineLearning, puesto que conocemos la naturaleza del problema y tenemos suficiente cualificación como para poder ir retocando los parámetros y añadiendo más o menos objetos (capas, regularización, etc.) a la red con el fin de maximizar el accuracy final. El objetivo que perseguimos no es únicamente maximizar el accuracy, sino que la curva de accuracy en el conjunto de validación sea lo más estable posible, que veamos que realmente está aprendiendo, y evitar que tenga picos o momentos de alta voatilidad. La configuración del mejor modelo resultante se encuentra en la figura 12.

Layer (type)	Output Shape	Param #	Kernel_size	Activation	Padding	Regularizers	Initializers
conv1d_4 (Conv1D)	(None, 24, 256)	18176	7	relu	valid	l2(0.0005)	GlorotNormal
conv1d_5 (Conv1D)	(None, 20, 128)	163968	5	relu	valid	l2(0.0005)	GlorotNormal
conv1d_6 (Conv1D)	(None, 16, 64)	41024	5	relu	valid	l2(0.0005)	GlorotNormal
conv1d_7 (Conv1D)	(None, 14, 32)	6176	3	relu	valid	l2(0.0005)	GlorotNormal
flatten_1 (Flatten)	(None, 448)	0	-	-	-	-	-
dense_2 (Dense)	(None, 32)	14368	-	relu	-	l2(0.0005)	GlorotNormal
dense_3 (Dense)	(None, 1)	33	-	linear	-	-	-

Total params: 243,745
Trainable params: 243,745
Non-trainable params: 0

Optimizer : Adam
Learning Rate : 0.0001
Loss : 'mean_absolute_error'
Batch_size : 512

Figura 12: Arquitectura de la red del modelo fallido

Como podemos ver en la figura 12, no se ha conseguido que el modelo aprenda en el conjunto de validación. Este mismo proceso lo hemos realizado para los otros tres valores de LAG, y no hemos conseguido mejorar los resultados. A priori podemos ver que hay 3 posibles motivos para este fracaso:

1. Las variables no tienen influencia real sobre el valor que queremos predecir.

2. La data es en su mayor parte aleatoria y tiene tanto ruido que el modelo no es capaz de separarlo.

3. No hay ninguna relación entre los patrones de train y los patrones de validación, lo que significa que la data es completamente aleatoria y es imposible aprender de ella, independientemente del modelo que utilicemos.

De momento no podemos saber cuál de estas 3 razones es la real (podrían serlo las 3 al mismo tiempo), pero antes de empezar a divagar, decidimos cambiar ligeramente el funcionamiento del modelo. En vez de hacer un problema de regresión, realizamos un problema de clasificación en el cual no queremos predecir el valor del ratio, sino simplemente saber si se encontrará por encima de un valor predefinido.

En el problema de clasificación vamos a intentar predecir la etiqueta de nuestros datos: esta etiqueta será binaria, y nos dirá si dentro de 120 días el valor del *Ratio* será superior a un umbral, cuyo valor es 0. A primera vista sería más beneficioso poner un valor mayor, por ejemplo en el percentil 0.9, de manera que solo aquellas empresas que lo alcancen se añadan a nuestra cartera. Sin embargo, desde el punto de vista de la IA, es mejor tener las clases balanceadas. Por ello, no usaremos solamente la etiquetas para clasificar, sino que podríamos rankear las empresas usando la probabilidad devuelta por la capa de salida de que un dato sea 1, es decir, en el algoritmo anterior usábamos el valor predicho del ratio para rankear, mientras que en este usamos la probabilidad de que sea de la clase positiva(mayor que 0). El algoritmo funciona de igual manera que en el problema de regresión, nada más que ahora cambiamos la capa de salida sigmoide y la función de coste. Tras realizar un nuevo proceso de optimización, el mejor modelo resultante se puede observar en la figura 13

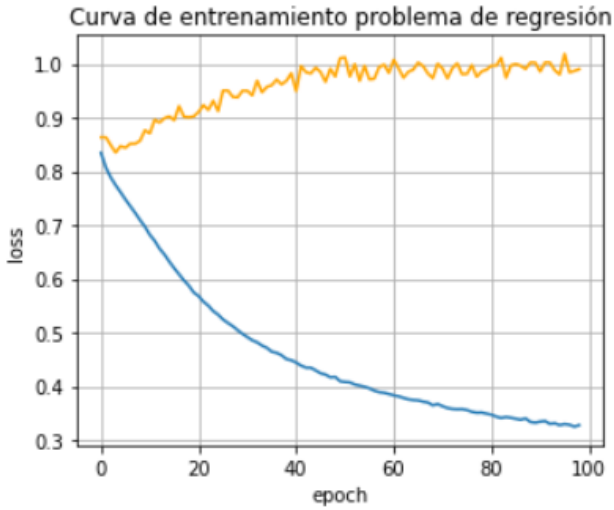


Figura 13: Curva de aprendizaje del modelo fallido

Nuevamente, nuestra red no ha conseguido aprender en el conjunto de validación; nuestro intento de cambiar los resultados utilizando un problema de clasificación no ha dado sus frutos. Entonces puede que sea el momento de cambiar de estrategia, utilizar una estrategia donde no se intenten tratar todos los millones de datos que tenemos, y centrarnos solo en unos sucesos determinados donde podamos minimizar el ruido y la información que necesitamos. Fruto de ello surge el siguiente algoritmo de reversión a la media.

6. Algoritmo de Reversión

6.1. Introducción

Las series de precios de activos bursátiles tienen un altísimo componente entrópico. En el anterior algoritmo lo hemos sufrido, pero nos ha servido para darnos cuenta de que tal vez usar los datos de cada día y de cada empresa puede que no sea lo más eficiente. Tal vez sería más eficiente buscar un suceso en concreto, y aprender únicamente respecto de ese suceso, y no respecto a cualquier día en general. Fruto de ello surge el Algoritmo de Reversión a la Media del *Ratio*.

Este algoritmo es viable gracias a la estacionariedad de nuestro *Ratio*, propiedad que ya hemos mencionado anteriormente en numerosas ocasiones. Este algoritmo pretende aprender únicamente los movimientos que hará el *Ratio* tras un suceso determinado: el corte de la segunda desviación típica por la parte inferior. Como serie estacionaria que es, buscaremos aquellas veces que se aleje en demasía de su media, y partiremos de la hipótesis nula de que en este momento habrá una alta probabilidad de que retorne a su media con cierta celeridad. El modelo solo será entrenado con estos sucesos, y nuestro algoritmo de inversión solo podrá realizar compras cuando se cumpla este suceso. Nuestra etiqueta de clasificación será binaria: 1 o 0. 1 indicará que el *Ratio* ha tocado el Take Profit, situado justo en la media(es decir, en el valor cero de la desviación típica) y 0 indicará que ha tocado el Stop Loss, situado a 2.5 desviaciones típicas bajo la media.

Ya lo explicamos en la descripción de variables, pero como es importante vamos a recalcarlo: No estamos usando directamente el valor real de la desviación típica del *Ratio*, sino que cuando hablamos de desviación típica, o de dos desviaciones típicas, nos referimos a cuántas desviaciones típicas respecto a su media se encuentra actualmente el *Ratio*. Si este valor es por ejemplo 1.5, significa que el valor del *Ratio* es igual a su media más 1.5 multiplicado por el valor de su desviación típica. Al decir que únicamente vamos a utilizar aquellos datos donde el valor del *Ratio* ha cortado hacia abajo la segunda desviación típica inferior, significa que el valor del *Ratio* en el momento actual es inferior a la media actual menos 2 multiplicado por el valor de la desviación típica, y que el día anterior era mayor que la media menos 2 multiplicado por la desviación típica de ayer.

Antes de continuar, debemos definir bien cuales van a ser nuestros conjuntos de train, validación y test. En el modelo anterior, estos conjuntos estaban delimitados por fechas, pero incluían las series de las variables de todos los días con cotizaciones de todas las empresas. Es decir, el conjunto de entrenamiento se componía de todas y cada una de las filas de datos existentes, para cada fecha y para cada empresa con valores en esa fecha. Sin embargo, ahora no vamos a utilizar toda esa data, sino únicamente un pequeño subconjunto de ella. Nuestro nuevo conjunto de entrenamiento (que incluirá también el conjunto de validación) solo se compondrá de aquellas filas de tabla de train original en las que se cumpla el suceso ya definido: que el valor actual de la variable ''desv_120", o sea, a cuantas desviaciones típicas se encuentra el *Ratio* respecto de su media, haya cortado hacia abajo la segunda desviación típica inferior, o sea, la media menos dos desviaciones típicas. Esto nos dará lugar a una tabla con datos ''salteados". En las figuras 14 y 15 se pueden observar una parte de estas dos tablas para poder aclararlo visualmente.

En la 14, vemos como la tabla de train dispone de manera ordenada de las variables de cada empresa cada día, con el formato salido de la parte de preprocesado. Sin embargo, en 15 vemos como nuestra nueva tabla de train para el algoritmo actual solo se ha cogido aquellas filas donde se ha cortado hacia abajo la segunda desviación típica, y serán esos los datos que usaremos para entrenar nuestro modelo. Como es obvio, haremos esto mismo para la tabla de test, de manera que nuestro algoritmo solo dará predicciones para aquellas filas en las que se haya cortado la segunda desviación.

	timestamp	open	high	low	close	adjusted_close	volume	Name	ratio	diff_120	diff_80	diff_40	desv_120	desv_60	perc_120	perc_60	diffemas_120_60	diffemas_60_2
0	2000-10-31	45.63	47.63	45.6300	46.31	30.680541	2799000.0	A	-1.059711	-1.257717	-0.546043	0.051047	-0.530822	1.213705	0.441667	0.866667	-0.264759	-0.09483
1	2000-11-01	46.00	47.31	45.5600	46.81	31.011793	2430700.0	A	-1.262248	-1.761855	-0.752402	-0.086717	-0.902703	-0.376412	0.208333	0.416667	-0.262867	-0.09428
2	2000-11-02	47.13	48.69	46.8800	48.13	31.886297	2455500.0	A	-1.197323	-1.563064	-0.741532	-0.027969	-0.767491	0.129735	0.316667	0.616667	-0.259870	-0.09069
3	2000-11-03	47.63	47.69	45.8100	46.81	31.011793	1700100.0	A	-1.276068	-1.774886	-0.840617	-0.125095	-0.914854	-0.496100	0.183333	0.350000	-0.258158	-0.09073
4	2000-11-06	46.94	46.94	45.8800	46.38	30.726916	1692300.0	A	-1.126547	-1.422096	-0.779912	0.078809	-0.603508	0.683744	0.416667	0.783333	-0.253959	-0.08393
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2168684	2015-12-24	8.63	9.74	8.6300	9.64	8.700332	1004151.0	BTU_1	-1.198337	0.867732	0.058341	0.300788	1.256460	2.240106	0.991667	1.000000	0.088499	0.04757
2168685	2015-12-28	9.57	9.57	8.1100	8.24	7.436798	1135024.0	BTU_1	-1.234993	0.879517	-0.004538	0.408150	1.140335	1.828230	0.958333	0.983333	0.091301	0.05549
2168686	2015-12-29	8.53	8.56	7.9000	8.17	7.373621	725636.0	BTU_1	-1.201849	0.937626	0.018661	0.470681	1.208016	2.014076	0.983333	0.983333	0.094455	0.06388
2168687	2015-12-30	7.85	8.22	7.8201	8.03	7.247268	743598.0	BTU_1	-1.246174	0.836351	-0.020911	0.349145	1.072640	1.569443	0.925000	0.933333	0.096684	0.06926
2168688	2015-12-31	7.88	8.18	7.6200	7.68	6.931385	1059431.0	BTU_1	-1.307726	0.697288	-0.031184	0.253427	0.894383	1.052600	0.833333	0.800000	0.097751	0.07116

2168689 rows × 20 columns

Figura 14: Variables de todos los datos de entrenamiento

	timestamp	open	high	low	close	adjusted_close	volume	Name	ratio	diff_120	diff_80	diff_40	desv_120	desv_60	perc_120	perc_60	diffemas_120_60	diffemas_60_25
173	2001-07-11	28.35	28.75	26.20	26.87	17.801471	3102100.0	A	-1.774335	-2.172682	-0.773787	-1.034121	-2.138051	-2.532604	0.008333	0.016667	-0.130301	-0.173286
841	2004-03-11	31.40	31.99	30.20	30.20	20.007608	6176700.0	A	0.346603	-2.701963	-1.011716	-0.938895	-2.091169	-2.198300	0.008333	0.016667	0.003580	-0.136286
869	2004-04-21	29.05	29.45	29.00	29.04	19.239104	4114700.0	A	0.211239	-0.972111	-0.499212	-0.639981	-2.041208	-1.469398	0.008333	0.016667	-0.153003	-0.225677
871	2004-04-23	29.27	29.75	29.06	29.50	19.543856	2438400.0	A	0.150940	-1.094977	-0.540833	-0.679929	-2.076156	-1.530877	0.008333	0.016667	-0.166908	-0.242186
1116	2005-04-14	20.81	21.01	20.62	20.67	13.693949	2092700.0	A	-1.182696	-0.446295	-0.112795	-1.135119	-2.084333	-2.633401	0.008333	0.016667	0.062822	-0.095502
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2168393	2014-10-29	10.50	10.61	10.15	10.35	138.849831	9263786.0	BTU_1	-2.081162	-1.408978	-1.206668	-1.570334	-2.028030	-1.306183	0.058333	0.116667	-0.295813	-0.380945
2168549	2015-06-15	2.56	2.57	2.35	2.39	32.355487	25587617.0	BTU_1	-2.208762	-0.465617	-0.543696	-0.681734	-2.334681	-2.592581	0.008333	0.016667	-0.062179	-0.058017
2168572	2015-07-17	1.37	1.39	1.27	1.29	17.463840	14357296.0	BTU_1	-2.302660	-0.321999	-0.779951	-0.262599	-2.472237	-2.369173	0.008333	0.016667	-0.097668	-0.100715
2168577	2015-07-24	1.32	1.34	1.23	1.24	16.786947	11421530.0	BTU_1	-2.232871	-0.206373	-0.550625	-0.286219	-2.027827	-1.658179	0.025000	0.050000	-0.106851	-0.107549
2168586	2015-08-06	1.09	1.12	1.03	1.10	14.891647	6249363.0	BTU_1	-2.408310	-0.679271	-0.810257	-0.654609	-2.199710	-1.837420	0.025000	0.050000	-0.137806	-0.150263

21415 rows × 20 columns

Figura 15: Variables de los datos de entrenamiento que han cruzado la segunda desviación típica

Las principales diferencias existentes entre los dos algoritmos son las siguientes:

- Al tratar solo sucesos determinados, el algoritmo partirá de mucha menos casuística y tendrá que discriminar mucha menos información. En el algoritmo anterior las redes neuronales debían poder dar una predicción para cualquier valor del *Ratio* en cualquier día y en cualquier empresa, lo que son infinitamente más de casos posibles que en el algoritmo de reversión, donde las redes solo deberán aprenderse el comportamiento del *Ratio* cuando corte la segunda desviación típica, ignorando todas las demás situaciones. Esto implica que le resultará mucho más fácil aprender durante la fase de entrenamiento. El problema podría venir de no tener suficientes datos de ese determinado suceso para que el algoritmo aprendiese, pues el número de veces que el *Ratio* corta la segunda desviación es muy inferior a la data total de la que disponemos. Por poner un ejemplo con números, uno de los

modelos que veremos después dispone de 1.463.000 datos en el conjunto de entrenamiento con toda la data, pero cuando restringimos esa data solo a los cortes de la segunda desviación, nos quedamos solo con 14.353 datos. El problema aquí está muy claro: si el suceso reduce mucho la cantidad de data para entrenar, dispondremos de menos ejemplos para que la red aprenda, pero al mismo tiempo, teóricamente, no necesitaría tantos ejemplos para poder aprender, pues al ser un suceso muy restringido, con una cantidad pequeña de datos la red debería ser capaz de modelizarlo.

- Ya no le obligamos al modelo a hacer predicciones justo en fechas determinadas (antes obligábamos al modelo a darnos las predicciones de los ratios de las empresas (justo cada 120 días) sino que ahora damos al modelo libertad para decir si debemos abrir una operación o no dentro de nuestro nuevo conjunto de test.

- Las etiquetas que debe aprender son infinitamente más flexibles. Esto no es exclusivo de este algoritmo, sino que se ve en numerosos algoritmos distintos. Como desarrolladores, podemos intentar predecir lo que va a ocurrir dentro de x días, o podemos intentar predecir si va a ocurrir un suceso independientemente de cuando ocurra. En el primero de los casos, estamos ''atando de pies y manos" a nuestro modelo de Inteligencia Artificial, el cual no puede ''opinar" si ese número x de días es adecuado o no. Lo único que podemos hacer es cambiar el valor de x, pero aún así, a veces ese x será positivo y a veces no, pero el algoritmo no puede cambiarlo. La manera de solucionar esto es liberalizar el número de días para realizar la predicción, usando un método de doble barrera: marcamos una barrera por arriba (el TP) y una por abajo (el SL) y de esta manera el algoritmo solo deberá aprender a qué barrera llegará antes, sin sufrir la influencia de que le digan exactamente cuántos días deben de pasar. Tras realizar numerosas pruebas, tanto en este trabajo como en proyectos ajenos a este TFM, se ha llegado a la conclusión, basada en resultados objetivos, de que la segunda vía permite obtener mejores resultados. Sin embargo, hay un inconveniente que no hemos tenido en cuenta: es cierto que el accuracy del modelo será mayor, pero por las condiciones del contexto del problema en cuestión, es posible que si tardamos demasiado tiempo en alcanzar una de las dos barreras, esto vaya en contra de nuestros intereses, que no han sido modelizados en la red neuronal. Tener mucho tiempo una operación abierta nos ''congela" capital que podríamos estar utilizando en otras operaciones, cosa que nos perjudica. Esto se puede solucionar de dos maneras: la primera es crear un modelo de triple barrera, y la segunda es tratar ese punto fuera del modelo de inteligencia artificial, incluyéndolo en la parte de asignación de capital (por ejemplo, cerrando las operaciones obligatoriamente si pasan más de x días). Nosotros hemos aplicado la segunda de las maneras, pues la primera produciría que la red neuronal tuviese mayores dificultades para aprender, pues pasaría a tener 3 clases, y una de ellas (alcanzar la tercera barrera) podría ser difícil de aprender.

- El punto más negativo es la restricción en las posibles operaciones a realizar. Es decir, el número máximo de operaciones que podría abrir nuestro algoritmo de inversión es el número de veces que el *Ratio* corta a la desviación. Tras introducir el modelo de red neuronal, este número se verá reducido, podemos asumir a priori que a la mitad. Si luego cambiamos el threshold (es decir, clasificar como TP solo aquellos datos donde la probabilidad es mayor de, por ejemplo, 0.7) el número de posibles operaciones será aún menor. Y finalmente, con el algoritmo definitivo de gestión de capital, las opciones que tomemos podrán reducir aún más nuestro número de operaciones, si por ejemplo decidimos no abrir más de una operación a la vez en un activo, limitar el número máximo de operaciones activas al mismo tiempo, o no estar invertido simultáneamente en acciones muy correlacionadas. Todos estos filtros reducirán drásticamente el número de operaciones, y puede llevar a tener un número tan pequeño que no nos permita diversificar ni tener un volumen estadísticamente significativo. Es la ''lacra" de poder operar únicamente cuando el *Ratio* corte la segunda desviación típica: hemos visto que casi todo son ventajas, sobre todo desde el punto de vista del aprendizaje de las redes neuronales, pero desde el punto de vista el volumen de operaciones, este algoritmo podría tener problemas.

Ya hemos definido como funciona nuestro algoritmo, por qué lo hemos desarrollado, y cuáles son sus pros y contras. Vamos a tratar solo aquellos datos donde el *Ratio* corta la segunda desviación típica, a operar sobre un subconjunto de ellos ayudándolos de las predicciones de una red neuronal, y a gestionar el capital y las operaciones finales mediante un algoritmo de gestión de capital.

Ahora vamos a pasar al desarrollo del algoritmo. Primero vamos a ver que resultados podríamos obtener con el algoritmo sin inteligencia artificial, y luego veremos como se mejoran los resultados utilizando Inteligencia Artificial. Recordemos que en esta parte nuestro único objetivo es obtener el mayor accuracy posible; de momento, no nos estamos preocupando ni de la rentabilidad, ni del riesgo ni de la asignación del capital, únicamente de poder predecir correctamente una mayor cantidad de datos. La separación en train, validación y test será diferente al algoritmo anterior. Hemos separado la data a partir del inicio del año 2016 para el conjunto de test, y vamos a entrenar con la data desde el 2000 hasta finales del 2015. Para el conjunto de validación, hemos seleccionado el último 30 % de los datos de train, mas debemos tener cuenta dos cosas: en primer lugar, al igual que en el algoritmo anterior, aquellos datos cuya etiqueta se "alcance" dentro del conjunto de test, han sido eliminados. Es decir, que si por ejemplo el *Ratio* de una empresa determinada corta la segunda desviación a finales de 2015 y toca el TP o el SL en el 2016, ese dato ha sido borrado, pues si estuviésemos en producción no dispondríamos de tal etiqueta. En segundo lugar, nuevamente, el StandardScaler utilizado para la normalización tanto de los 3 ratios dis(Sharpe, Calmar y PF) que forman el *Ratio*, como las variables, han sido entrenados únicamente con los datos del conjunto de entrenamiento. En todos los aspectos hemos tenido muy presente en no entrenar con nada que esté relacionado con el conjunto de test, y así evitar esos comunes errores en proyectos de Machine Learning.

Antes de pasar al desarrollo del algoritmo, hay otras dos cosas a tener en cuenta. La primera es que hemos utilizado dos valores de la ventana móvil para el cálculo del *Ratio*: 120 y 365. Ya hablamos anteriormente de cómo influye el hecho de tener una ventana más rápida y una más lenta. Para este modelo de reversión a la media, tanto para el algoritmo básico sin inteligencia artificial como para el algoritmo con inteligencia artificial, hemos hecho todas las pruebas y optimizaciones usando tanto el *Ratio* (y sus variables correspondientes) con la ventana de 120 dias y la ventana de 365. Como ya dijimos en la descripción de las variables, las ventanas de las variables se han movido proporcionalmente(por ejemplo, las dos variables que modelizan la diferencia del *Ratio*: para el *Ratio* con ventana 120, esas dos variables muestra la diferencia respecto a 120 y a 60 datos del pasado, mientras que para el *Ratio* de ventana 365, las diferencias son respecto a 365 y 180 datos del pasado). La segunda es que hay varios hiperparámetros que hasta el momento no hemos comentado: el umbral de corte de las desviaciones típicas y el TP/SL. Establecer que el punto de entrada será el corte por debajo de la segunda desviación típica se ha hecho por dos motivos: el primero es por convención, pues se suele usar casi siempre la segunda desviación, y el segundo motivo es porque la cantidad de datos resultante nos pareció apropiada para el proceso de entrenamiento. La elección de poner el TP en el valor cero de la desviación típica y el SL en el -2.5 fue para que las clases estuviesen más o menos balanceadas, para qué el TP tuviese considerablemente más ganancias en rentabilidad que pérdidas el SL, y para que en el caso de que el *Ratio* se moviese en nuestra contra, poder cortar rápidamente las operaciones antes de que apareciese una divergencia entre el movimiento del *Ratio* y el movimiento de su desviación típica (puede ocurrir que el *Ratio* caiga abruptamente pero que su desviación típica se mantenga estable, en ese caso iríamos acumulando pérdidas sin que saltase el SL). Una vez que hemos aclarado todos los puntos, vamos a pasar al desarrollo del algoritmo. Primero, hemos sacado los resultados del modelo sin usar Inteligencia Artificial (que posteriormente podrá funcionar como benchmark) y luego hemos repetido el proceso utilizando redes neuronales. De esta manera, podremos ver cuantitativamente si el uso de redes neuronales mejora el algoritmo inicial sin inteligencia artificial. En primer lugar, vamos a sacar los resultados de todos los modelos en el conjunto de validación. A continuación, vamos a realizar el backtest mediante la asignación de capital para el modelo sin IA con ventana de 120, con ventana de 365, y vamos a hacer lo mismo para el mejor algoritmo con IA con ventana de 120 y con ventana de 365, todo en el conjunto de validación.

Vamos a tener que tomar numerosas decisiones, como el threshold para clasificar las etiquetas mediante la red neuronal, como queremos asignar cuantitativamente el capital, cuanto riesgo queremos asumir...

Todo esto lo vamos a estudiar con los resultados obtenidos en el conjunto de validación, y una vez definidos todos los hiperparámetros del algoritmo de gestión de capital, usaremos el modelo tal cual lo hemos establecido en el conjunto de test, con el objetivo de obtener un 100 % de realismo. En conclusión, a continuación vamos a realizar dos tareas: implementar los modelos sin IA y con IA, cuyos objetivos respectivamente son establecer un benchmark, y maximizar el accuracy obtenido por la red neuronal. La segunda tarea será crear el algoritmo de gestión de capital basándonos en las predicciones tanto del modelo sin IA que consistirá en entrar cada vez que cortamos la segunda desviación, por eso es el benchmark básico) como del modelo con IA, que usará redes neuronales para decidir en qué situaciones entrar y en que situaciones mantenerse al margen. Y estas dos tareas las realizaremos exclusivamente en el conjunto de validación.

6.2. Algoritmo sin inteligencia artificial

Como ya mencionamos anteriormente, nuestros nuevos conjuntos de train y test disponen únicamente de aquellos días en los que el *Ratio* de una determinada empresa ha cortado hacia abajo la segunda desviación típica inferior. Por ello, el algoritmo sin IA va a asumir que entramos siempre que ocurra este suceso, independientemente de si ya estamos invertidos en dicha empresa, el número de operaciones activas, y demás (recordemos que en esta primera tarea no estamos utilizando la gestión de capital, simplemente estamos viendo los resultados teóricos que obtendríamos si operásemos según lo que nos dice el algoritmo). Las dos métricas que vamos a utilizar serán, primeramente, el accuracy, y en segundo lugar, el porcentaje de variación de aquellos activos en los que hemos estado invertidos. Luego, cuando implementemos el modelo con IA, solamente tomaremos una parte de las entradas que ha realizado el algoritmo sin IA, y veremos cómo ha variado el accuracy obtenido y el porcentaje de variación de la cotización de las empresas invertidas. El conjunto de validación comienza en el día 2011-06-10 y acaba en el 2015-12-31. En este periodo, el *Ratio* de cada una de las empresas en el conjunto de validación ha cortado la segunda desviación típica inferior 6823 veces, lo que implica que tenemos 6823 operaciones. Ahora, vamos a sacar cual ha sido la variación del precio de las empresas correspondientes en los periodos en los que se han abiertos dichas operaciones y vamos a estudiar los resultados. Antes de continuar, debemos recordar que acertar con la etiqueta no nos garantiza que la operación sea positiva, pues recordemos que la etiqueta nos dice si ha saltado el TP o el SL en el *Ratio*, que aunque esté muy correlacionado con el precio, no tiene por qué coincidir siempre. Por ello, no solo vamos a medir el accuracy del modelo sin IA, sino que también vamos a ver si dichas operaciones realizadas sobre la cotización de los activos ha dado retorno positivo o no.

6.2.1. Ventana 120

Vamos a estudiar los resultados obtenidos por el algoritmo sin IA sin realizar la gestión de capital, como acabamos de decir. Por ello, para medir la "rentabilidad'' de las operaciones, vamos a usar como métrica el porcentaje de variación del activo entre su compra y su venta, es decir, que si compramos a 10 y vendemos a 11, vamos a decir que hemos obtenido una rentabilidad del 10 %. Cuando implementemos el algoritmo de gestión de capital esto no será así, sino que a ese 10 % le tendremos que contabilizar las comisiones, el porcentaje de capital introducido en la operación y el capital total del que disponemos para obtener la rentabilidad de la operación, pero por ahora no lo vamos a hacer así, sino que vamos a contabilizar únicamente el porcentaje de variación del activo, así que cuando escribamos "rentabilidad'' o ganancias medias en porcentaje en el siguiente análisis, nos referiremos a la variación del activo durante dicha operación. Como vamos a contabilizar como "operación realizada'' todas los días en los que se corta la segunda desviación típica inferior, el accuracy es simplemente dividir las etiquetas con valor 1 (TP) entre las etiquetas con valor 0

(SL). El accuracy que obtiene el algoritmo es de un 59.84 %; este número será nuestro benchmark en el algoritmo con Inteligencia Artificial.

Ahora queremos estudiar las estadísticas de las operaciones en las que ha saltado el TP y en las que ha saltado el SL, puesto que, como ya hemos dicho, no todas las operaciones que tocan el TP acaban siendo positivas. Si cogemos todas las operaciones en las que ha saltado el Take Profit, vemos que su ganancia media es de un 6.9 %, su desviación típica es de un 12 %, y el porcentaje de operaciones positivas es de un 81.6 %. Esto significa que la probabilidad de que una operación que toca el TP sea rentable es de un 81.6 %. Evidentemente, nos gustaría que ese número fuese mayor, pero recordemos que no estamos colocando el TP y el SL es el "precio'' del activo, sino en su Ratio, pues esto nos facilita la predicción con el modelo de IA, pero eso implica que solo saquemos beneficios un 81.6 % de las veces; aun así, es un número suficientemente alto. Si cogemos ahora las operaciones en la que ha saltado el Stop Loss, vemos que la ganancia media es de un -3.08 %, la desviación típica es 5.49 % y el porcentaje de operaciones que han sido negativas es de un 78.43 %. En este caso esto va a nuestro favor, pues un 21.57 % de las veces que salta el SL no acabamos con pérdidas en la operación. Ya hemos estudiado los resultados de acuerdo a la etiqueta que nos ha dado nuestro modelo de reversión a la media del Ratio; ahora vamos a ver los resultados de las operaciones independientemente de si ha sido TP o SL. La ganancia media es de un 2.8 % y la desviación típica es de un 11.09 %. Vemos que el algoritmo sin IA es rentable por sí solo, pues la ganancia esperada es de un 2.8 % por operación. El porcentaje de operaciones positivas es de un 57.46 % (algo inferior al 59.84 % que era el porcentaje de Take Profits) y la ganancia media de las operaciones positivas es de un 9.4 %.
Tras este análisis, hemos obtenido dos conclusiones.

1. El algoritmo es ya de por sí positivo sin necesidad de introducir Inteligencia Artificial(recordemos que no estamos usando gestión de capital, y que las ganancias medidas en % se refieren al porcentaje de variación del activo entre su compra y su venta).

2. Aproximadamente un 80 % de las operaciones que salen en TP o en SL tienen un resultado monetario acorde a la etiqueta.

Cuando hagamos los backtest para el modelo de IA con la ventana de 120 dias y la de 365, compararemos los resultados. Vayamos ahora por la ventana de 365.

6.2.2. Ventana 365

El accuracy del modelo en función de los TP obtenidos es de un 44 %, bastante inferior a la ventana de 120 días. Respecto a las operaciones en las que ha saltado el TP, la ganancia media es de un 16.44 %, la desviación típica es del 17.3 %, y el porcentaje de operaciones rentables es de un 93.6 %. Respecto a los SL, la ganancia media es de un -5.4 %, la desviación 8.89 % y el porcentaje de operaciones negativas es de un 85.64 %.

Podemos apreciar que el accuracy del TP es considerablemente menor; sin embargo, la ganancia media por operación es mucho mayor que con la ventana de 120 días. Además, el porcentaje de operaciones positivas en TP y las negativas en SL ha subido también considerablemente, lo que significa que hay una mayor correlación entre la serie de la desviación del *Ratio* y el precio de las empresas. De esta manera, es mucho más probable(un 93.6 %) que una operación sea positiva cuando salta el TP y negativa cuando salta el SL, cosa que nos interesa. Ahora veamos los resultados de las operaciones independientemente de si ha saltado TP o SL.

El porcentaje de operaciones rentables es de un 49 % (superior al porcentaje de TP), la ganancia media de las operaciones positivas es de un 15.89 %, la ganancia media de todas las operaciones es de un 4.2 % y la desviación típica es de un 17.16 %. Podemos apreciar que la ganancia media por operación, que al fin y al

cabo es la métrica más importante, ha subido de un 2.8 % a un 4.2 %, lo que es una mejora bastante grande. Podemos decir que, aunque la tasa de acierto es menor con la ventana de 365, tanto en lo referente a TP como a operaciones positivas, luego las ganancias son considerablemente mayores, lo que nos da mejores resultados con esta ventana, y además la correlación entre las etiquetas y los resultados de las operaciones es mayor.

Una vez análizados los resultados del modelo sin Inteligencia Artificial, vamos a desarrollar el algoritmo con Inteligencia Artificial.

6.3. Algoritmo con inteligencia artificial

Ya hemos visto que los resultados teóricos (sin haber incluido gestión de capital ni comisiones, solo mirando las variaciones de los activos) da resultados positivos. Sin embargo, suponemos que estos resultados podrían ser mejorados utilizando técnicas de Machine Learning. El algoritmo sin IA opera siempre que se cumpla la única condición de entrada: que se corte la segunda desviación típica inferior. Sin embargo, podemos introducirle nuevos filtros con el objetivo de restringir algunas de esas operaciones, y que nuestra tasa de acierto sea mayor. Podemos por ejemplo añadirle una nueva regla, que sea operar únicamente si el activo es muy estacionario en el momento actual (que puede ser medido con el Test Aumentado de Dickey-Fuller o el exponente de Hurst), o podemos decidir operar únicamente si el *Ratio* ha caído de manera abrupta hasta la segunda desviación, con la intención de que retorne de manera igual de abrupta a la media. Sin embargo, las redes neuronales pueden crear estos filtros por nosotros de manera mucho más eficiente. Por ello, vamos a usar el Deep Learning con el objetivo de crear una red neuronal que nos diga para cada uno de nuestros datos, es decir, para cada vez que el *Ratio* de una empresa corte su segunda desviación típica inferior, cuál es la probabilidad que tenemos de que el *Ratio* toque antes el TP que el SL. De esta manera, filtraremos muchas operaciones malas (y también algunas operaciones buenas) con el fin de mejorar el accuracy del modelo sin Inteligencia Artificial. Como ya dijimos anteriormente, al tratar con series temporales, podremos utilizar capas convolucionales y capas recurrentes, las cuales requieren de una dimensión extra en la data, el LAG, que es el tamaño de las sub-ventanas que servirán para que las capas puedan tratar no solo el dato del día actual, sino también los datos pasados. Por ello, vamos a probar distintos modelos, tanto para las ventanas de 120 y 365 días, como para el LAG de 30, 60, 90 y 150 valores. El resultado que obtendremos de cada uno de estos, en principio, 8 modelos, será la curva del accuracy obtenido en validación. Nuestro objetivo es para estos 8 conjuntos diferentes, encontrar el modelo que obtiene una mejor curva. A continuación, escogeremos el mejor valor de LAG tanto para la ventana de 120 como para la de 365, obteniendo de esta manera dos modelos y dos resultados diferentes, que podremos comparar con los resultados obtenidos por el algoritmo sin IA para 120 y para 365. A continuación, ajustaremos nuestro algoritmo de gestión de capital para estos dos mejores modelos (también lo utilizaremos con los dos modelos sin IA para ver nuevamente como varían los resultados) y al final nos quedaremos con uno solo de nuestros dos modelos, definido por si usa una ventana de 120 o de 365, por el valor del LAG, por la arquitectura de su red neuronal, y como el terminado algoritmo de gestión de capital que habremos ajustado en el conjunto de validación. Una vez hecho esto, reentrenaremos la red neuronal con todos los datos de train, evaluaremos en test para ver el resultado de su curva, y ejecutaremos el algoritmo de gestión de capital en los datos de test, obteniendo así los resultados definitivos de los algoritmos.

Pero antes de todo eso, debemos encontrar las mejores arquitecturas de redes neuronales y las mejores configuraciones de hiperparámetros para los dos valores de nuestra ventana y los 4 valores de nuestro LAG. Ahora nos vamos a centrar exclusivamente en temática de redes neuronales.

Al tratarse de un problema de clasificación, nuestra capa de salida va a ser siempre una capa densa con una única neurona y función de activación sigmoide. Nuestro optimizador por defecto será un Adam cuyo learning rate variaremos a menudo para encontrar la velocidad de aprendizaje óptima. La función de coste será la binary_crossentropy, pues tenemos únicamente dos clases. Las dos métricas en las que más nos vamos a fijar son el accuracy en train y el accuracy en validación. El primer paso es que el accuracy en train sea

creciente, pues si nuestra red no es capaz de aprender jamás obtendremos buenos resultados. Tras lograr que la red aprenda correctamente en train, deberemos conseguir que aprenda en validación, utilizando distintas técnicas de regularización y optimizando los demás hiperparámetros.

Tenemos muchas opciones distintas para probar: utilizar capas convolucionales 1D y usar cada cada variable como un canal, utilizar capas convolucionales 2D dándole a las variables la ordenación de 2 dimensiones (esta idea no es muy acertada, la comentaremos más adelante), usar redes recurrentes(GRUs y LSTMs), mezclar todas las anteriores y reforzarlas con capas densas ocultas. Vamos a describir el proceso que nos ha llevado hasta las arquitecturas definitivas.

Inicialmente empezamos con LAG 30 probando capas convolucionales 1D. Lo primero que queremos hacer es estimar el número de capas ocultas convolucionales sin ningún tipo de regularización que obtienen un mejor crecimiento del accuracy en train. Para ello, vamos probando con distintas configuraciones del modelo para 1, 2, 3, 4 y 5 capas convolucionales ocultas, variando el número de filtros, el tamaño del kernel, el batch size y el learning rate (esto es una primera aproximación). Vemos que el número óptimo de capas es 3, puesto que con 4 y 5 capas la mejora en la curva de train apenas es significativa. Con esto hemos conseguido ''acotar" nuestro problema respecto al número de capas. Ahora es cuando empezamos el verdadero proceso de optimización. En este trabajo no esperamos que el lector sea un experto en Deep learning, pero sí que se requieren ciertos conocimientos básicos. Vamos a explicar brevemente cada uno de los conceptos, pero vamos a centrarnos sobre todo en cómo influye cada elemento en el desarrollo del algoritmo y cómo podemos utilizarlos para alcanzar mejores resultados, pero lo que no vamos a hacer va a ser convertir este trabajo en una clase teórica sobre el funcionamiento de redes neuronales. Los distintos parámetros que vamos a optimizar son:

- *Número de filtros:* nos indica la cantidad de filtros que ajustará la capa convolucional. El número de filtros de una capa será el número de canales de la siguiente capa. Aumentar este valor aumenta la complejidad de su correspondiente capa. Por ello, hay que tener en cuenta que si es muy bajo, puede que el modelo no aprenda lo suficiente, pero si es muy alto, probablemente tenderá a sobreoptimizar. Generalmente, con capas convolucionales se pretende que la dimensionalidad del problema vaya disminuyendo, por lo que cada capa debería ir disminuyendo su número de filtros; además se suelen usar potencias de 2, puesto que pueden ir dividiéndose entre 2 hasta llegar al valor unitario. Los valores más comunes para este hiperparámetro suelen estar entre 32 y 256; con 3 capas empezaremos probando los valores 128, 64 y 32. Si quisiésemos una mayor complejidad probaríamos con 256, 126 y 64, y se queremos menor complejidad, 64, 32 y 16. También probaremos a repetir valores en el número de filtros, como por ejemplo 128, 128 y 64, pues aunque la norma general es que vayan disminuyendo, no siempre es lo óptimo.
- *Kernel size:* indica el tamaño máximo del filtro de la capa. Cuanto mayor sea este valor, más datos tendrá en cuenta el filtro en cada iteración. Este hiperparámetro puede reducir la dimensionalidad del problema dependiendo del padding que se utilice. Igual que antes, lo comúnmente aceptado es empezar con un valor alto e ir disminuyéndolo, pues cada vez tendríamos series de datos más comprimidas. Sus valores por defecto suelen estar entre 3 y 7, normalmente siendo números impares.
- *Función de activación:* la función de activación modela la componente no lineal de las capas. Se suelen usar la "Relu", "Selu", "Elu"... por defecto usaremos una "Relu".
- *Padding:* nos indica cómo tratar los extremos de nuestra serie de datos. Tiene 3 valores posibles: "valid", "same" o "casual'. Sin pararnos a explicarlos en demasía podemos decir que el padding "same" mantiene la dimensión de los datos añadiendo ceros en los extremos para que el kernel pueda realizar sus cálculos, mientras que el padding "valid" reduce la dimensionalidad, pues en el momento en el que el kernel tiene uno de sus extremos fuera de la serie, deja de trabajar. Este parámetro también puede reducir la dimensionalidad de los datos en cada capa.

- *Kernel Initializers:* esta función nos indica como queremos inicializar los pesos de las neuronas. Las dos maneras que vamos a utilizar van a ser una inicialización aleatoria, y la inicialización "Xavier", que pretende que la varianza de la salida de una capa sea la misma que la de la entrada.

Estos son los hiperparámetro correspondientes a las capas convolucionales que optimizaremos. Ahora pasaríamos a los hiperparámetros de las capas densas, aunque no vamos a desarrollarlos, pues los 3 valores que vamos a optimizar son casi idénticos a los expuestos en las capas convolucionales: el número de neuronas (que equivale al número de filtros de las capas convolucionales), la función de activación y el "kernel initializer". Estos valores se corresponden solo a las capas ocultas, pues como ya dijimos, la capa de salida será una densa sigmoide con una sola neurona.

Ahora pasamos a la regularización: el objetivo de regularizar es poner ''trabas", dificultar el aprendizaje en la fase de entrenamiento, con el fin de que no se sobreajuste demasiado a la data de train y poder generalizar a la data de validación. Nosotros hemos trabajado con dos tipos de regularización:

- *Dropout:* omite ciertas neuronas de manera aleatoria durante el periodo de entrenamiento. Este valor se mide en tanto por uno, y nos indica el porcentaje de neuronas a omitir. Cuando mayor sea este valor, más dificultades tendrá el modelo para aprender en train. Si es muy bajo puede que sobre ajustemos, pero si es muy alto podemos provocar que el modelo no sea capaz de aprender.
- *Regularización l1 y l2:* modifican la función de coste para provocar un decaimiento controlado de los pesos de las neuronas. De igual manera que el dropout, valores bajos pueden provocar que el modelo no generalice en demasía, mientras que valores altos pueden llegar a penalizar demasiado el entrenamiento.

Finalmente, nos quedan dos hiperparámetros sumamente importantes que no se encuentran propiamente dentro de la arquitectura de las capas, sino que pertenecen más al algoritmo de aprendizaje. Estos dos hiperparámetros son:

- *Learning rate:* nos indica la velocidad a la que queremos aprender. Un valor demasiado alto puede provocar saltos muy grandes en el gradiente, lo que nos lleva a no poder aprender correctamente, mientras que valores muy bajos pueden provocar un aprendizaje muy lento, e incluso llegar a no aprender. Este hiperparámetro será sumamente importante, por lo que en vez de tratarlo aquí de forma teórica, lo veremos de forma práctica más adelante.
- *Batch size:* el tamaño de los mini batches que utilizaremos para entrenar. Igual que con el learning rate, veremos más adelante como afecta este hiperparámetro a la curva de aprendizaje.

Ya hemos definido brevemente cada uno de los hiperparámetros que vamos a optimizar, y como podrían afectar a los resultados de nuestro algoritmo. Ahora vamos a pasar directamente a los resultados obtenidos del proceso de optimización para la ventana de 120 días y el LAG de 30 utilizando capas convolucionales y capas densas.

6.3.1. Ventana 120

Antes de optimizar, debemos ver cómo se distribuyen las dos clases en nuestro problema. Para ello vamos a comentar que se utilizará la terminología Take Profit (TP) y Stop Loss (SL); siendo el primero el límite de salida con beneficio y el segundo el límite de salida a pérdidas. Con la ventana de 120 y el LAG de 30, el conjunto de train tiene 14353 datos, donde el 60.48 % se corresponden a Take Profit (cuya etiqueta en el código es 1), validación tiene 6816 datos con un 59.87 % de Take Profit, y test tiene 6383 datos con un 57.5 % de Take Profit. Las clases están bastante balanceadas, y la diferencia entre la proporción de las clases en los 3 conjuntos es bastante similar, lo que nos beneficiará enormemente con los modelos de IA. El conjunto de test no lo vamos a tocar hasta el último momento; la métrica que vamos a utilizar para ver qué modelos son

los mejores será la curva de accuracy en validación. Hay que tener en cuenta que partimos de un 59.87 % de acierto, el cual será nuestro benchmark inicial, lo que significa que nuestra "ganancia" de accuracy respecto a la data inicial se tendrá que calcular sobre ese 59.87 % de acierto.

Se ha realizado un largo proceso de optimización para encontrar las mejores arquitecturas. Como dijimos anteriormente, la optimización ha sido manual, sin usar herramientas auxiliares como "KerasTuner", "automated machine learning" o "Talos". Para la ventana de 120 dias y el LAG de 30 datos, el mejor modelo resultante ha sido el representado por en la figura 16.

Layer (type)	Output Shape	Param #	Kernel_size	Activation	Padding	Regularizers	Initializers
conv1d_12 (Conv1D)	(None, 30, 256)	18176	7	relu	same	l2(0.00075)	GlorotNormal
conv1d_13 (Conv1D)	(None, 30, 128)	163968	5	relu	same	l2(0.00075)	GlorotNormal
conv1d_14 (Conv1D)	(None, 30, 64)	24640	3	relu	same	l2(0.00075)	GlorotNormal
flatten_4 (Flatten)	(None, 1920)	0	-	-	-	-	-
dense_4 (Dense)	(None, 1)	1921	-	sigmoid	-	-	-

Total params: 208,705
Trainable params: 208,705
Non-trainable params: 0

Optimizer : Adam
Learning Rate : 0.0001
Loss : 'binary_crossentropy'
Batch_size : 2048

Figura 16: Arquitectura modelo conv1d, ventana 120, lag 30

Vemos que se han cumplido algunas de nuestras hipótesis de partida, y otras no. Nos hemos quedado con 3 capas convolucionales ocultas cuyo número de filtros es decreciente, empezando por 256 y acabando por 64, y de igual modo hemos sacado que la mejor secuencia de los valores del "kernel size" ha sido nuevamente decreciente, 7, 5, 3, como dijimos anteriormente que era la forma comúnmente aceptada. Sin embargo, durante el proceso de optimización, nos dimos cuenta de que reducir las otras dimensiones del problema no mejoraban el accuracy. El padding que hemos utilizado ha sido "same" que no reduce la dimensionalidad del problema con el kernel size, sino que la mantiene colocando ceros en los extremos. Tampoco hemos usado una funcionalidad no descrita anteriormente, los "strides", que también reducen la dimensionalidad de la data en cada capa. Podemos ver que el resultado ha sido que únicamente variamos el número de canales, establecido por el número de filtros, pero que el tamaño de la ventana móvil, el LAG, no lo vamos reduciendo. Esta no es la forma estándar de trabajar con capas convolucionales, pero sí que es la forma que obtiene unos mejores resultados en este problema.

La regularización ha tenido un papel muy importante en el desarrollo de este problema. Hemos probado tanto a usar dropout como regularización l1 y l2. La regularización l1 ha obtenido malos resultados, mientras que el dropout y la l2 han obtenido resultados idénticos. Para ambos hiperparámetros, hemos ido variando sus valores hasta encontrar los óptimos, y como con dichos óptimos los resultados eran casi idénticos, decidimos incluir la regularización l2 por el simple hecho de que, según algunas breves investigaciones en otros proyectos de IA encontrados por internet, lo normal es usar Dropout en vez de l2, y de esta manera mostramos que la regularización l2 también es una técnica aceptable.

Una vez acabadas las capas convolucionales, antes de pasar a la capa de salida, probamos a introducir capas densas para ir reduciendo la dimensionalidad del problema. Sin embargo, estas capas densas no mejoraban el accuracy del modelo. Nos ocurre lo mismo que antes; por convenio se suelen meter algunas capas densas entre la última convolucional y la capa densa de salida; sin embargo, añadir estas capas no nos mejoró

los resultados del modelo, aunque tampoco los empeoró. Por ello, nos hemos basado en el principio universal de que, si existen dos soluciones para un problema que proporcionan el mismo resultado, debemos quedarnos con la más simple. Hemos decidido basarnos en este principio en vez de en los estándares del machine learning, y por ello no hemos añadido ninguna capa densa oculta. Por último, hemos usado la inicialización Xavier anteriormente descrita para los pesos de las capas ocultas.

En este punto no hemos usado un early stopping, pues queremos llegar hasta un punto en el que se vea muy claramente que la curva en validación ha dejado de mejorar (evidentemente cuando vayamos a crear el modelo que usaremos para realizar el backtest sí que utilizaremos un early stopping, pero de momento solo estamos explorando los resultados de nuestros modelos). Ahora vamos a tratar los dos hiperparámetros que antes dejamos en suspenso: el leraning rate y el batch size. Empezamos usando valores altos del learning rate, lo que provocaba que alcanzásemos muy rápidamente el accuracy máximo en validación. Conforme lo disminuíamos, el accuracy máximo salía casi el mismo, pero la curva llegaba hasta el más lentamente. En cambio, con el batch size ocurre al contrario, valores más altos producen un aprendizaje más rápido.
Veamos ahora la curva de aprendizaje en train y en validación.

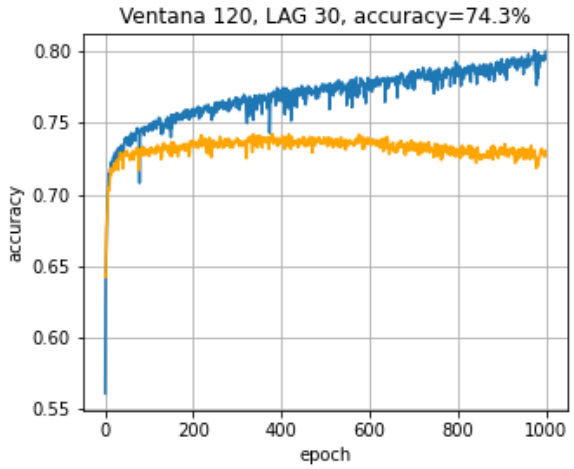


Figura 17: Curva entrenamiento del modelo conv1d, ventana 120, lag 30

Como se puede observar en la figura 17, las curvas presentan unas curvas de aprendizaje bastante estables. Como aprenden lentamente, empiezan cercanas al benchmark del 60 % y suben de manera estable hasta el 74.3 %, donde forma un "codo", momento en el cual empiezan a sobre ajustar los pesos, lo que provoca que la curva de accuracy en validación empiece a ser decreciente. Una curva de accuracy tiene 3 factores que determinan cuan "buena" es: el valor máximo alcanzado, la estabilidad en el crecimiento, y la ausencia de picos. Vemos que esta curva es buena en los tres sentidos, por lo que estamos satisfechos con los resultados. Sin embargo, lo que no sabemos es si arquitecturas de otra naturaleza u otros valores del LAG pueden aumentar el accuracy.

Ahora vamos a realizar el mismo proceso para los otros 3 valores de LAG: 60, 90 y 150. Estos incrementos permiten que el modelo pueda buscar sus patrones en datos más alejados. El proceso de optimización ha sido el mismo, y las arquitecturas óptimas han resultado ser casi idénticas. En todos los casos hemos mantenido la misma arquitectura con 3 capas, los mismos filtros, kernel size y padding. También ha ocurrido lo mismo

que antes con las capas densas ocultas: añadirlas no mejoraba los resultados del modelo, por lo que se ha decidido no añadirlas por el principio de simplicidad. El learning rate y el batch size óptimos han sido los mismos que en primer modelo, lo único que ha variado ligeramente han sido los valores de l2: con LAG 30, el l2 valía 0.00075, con LAG 60 vale 0.001, con LAG 90 vale 0.002, y con LAG 150 su valor es de 0.001. A continuación vamos a ver las curvas de aprendizaje de los 4 modelos (que recordemos son los modelos óptimos para cada valor del LAG usando el Ratio con ventana de 120 días).

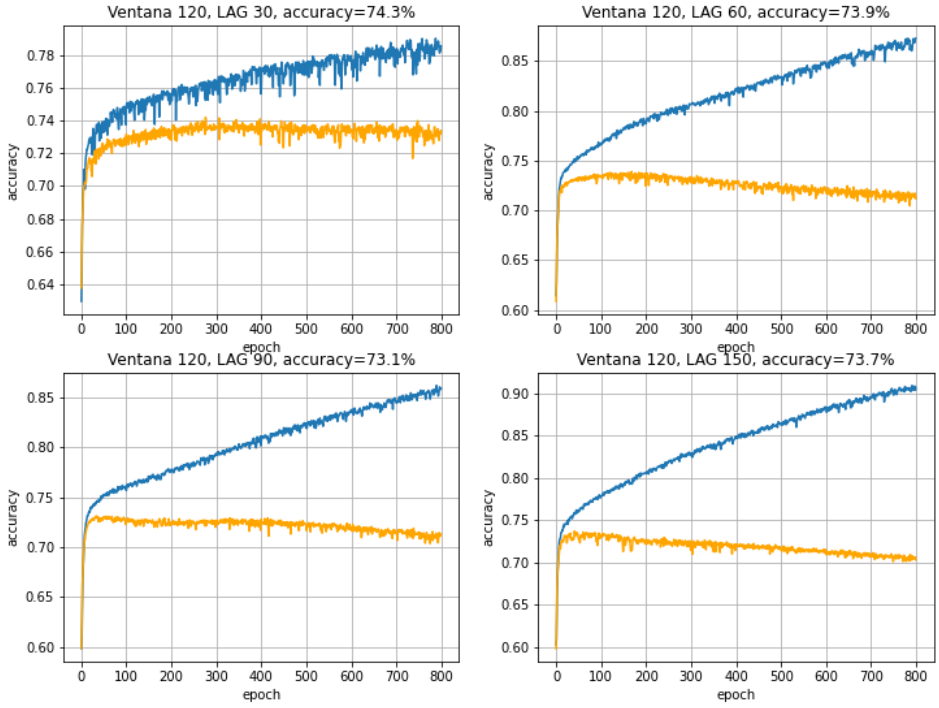


Figura 18: Curvas entrenamiento lags 30, 60, 90, 150

En la figura 18, vemos claramente como el LAG óptimo es 30. Los demás modelos llegan con mucha mayor rapidez a su valor más alto, pero se quedan ligeramente por debajo del 74.3 % de LAG 30. Debemos de tener en cuenta que un mismo entrenamiento de la red con los mismos datos y la misma estructura puede dar resultados ligeramente diferentes, pero hemos ejecutado estos modelos varias veces, y podemos asegurar que LAG 30 obtiene entorno a un 0.2 % más de accuracy que cualquier otro, teniendo además una curva más estable.

En este punto podríamos concluir que ya hemos encontrado el mejor valor para LAG, pero no sería correcto. De los 4 valores que hemos probado, el más pequeño nos ha dado el mejor resultado, por lo que sería lógico pensar que el valor óptimo del LAG puede ser más pequeño que 30. Por ello, repetimos el proceso anterior para LAG 20. El modelo óptimo continúa teniendo la misma arquitectura que los demás, con un l2 de 0.001. Veamos ahora su curva.

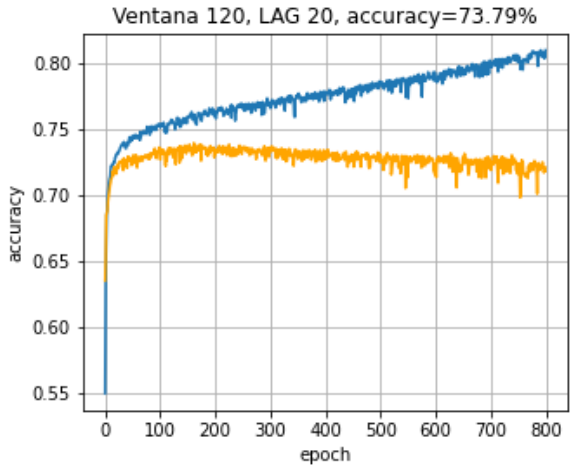


Figura 19: Curva entrenamiento del modelo conv1d, ventana 120, lag 20

En la figura 19, vemos que no superamos el accuracy del modelo con LAG 30, por lo que ahora sí que concluimos que el valor óptimo de LAG es 30. Ese modelo es actualmente el que mejores resultados nos ha dado en el conjunto de validación, mas podemos probar otras técnicas. Como bien dijimos al inicio de este apartado, podemos usar también capas convolucionales 2D o capas recurrentes. Vamos a probar 3 estructuras: "GRUs", convolucionales 2D, y convolucionales 1D jugando con las capas. Hemos optimizando los valores de cada una de estas tres arquitecturas, aunque no de forma tan exhaustiva que con las convolucionales 1D, y únicamente lo hemos hecho para LAG 30 pues, como veremos a continuación, los resultados no se mejoran.

Vamos a optimizar un modelo con LAG 30 en el que utilicemos capas "GRUs" con la posibilidad de incluir densas al final. Tras probar distintas combinaciones, el modelo resultante ha sido el mostrado en la figura 20.

Hemos tenido que aumentar la complejidad del modelo, pues con la arquitectura del modelo anterior, las capas GRUs no conseguían aprender lo suficiente. Hemos añadido una capa con 512 unidades, dos capas densas de 32 y 16, y hemos bajado los l2 a 0.00025, excepto el de la primera capa densa, que se encuentra en 0.0025.

Como se puede observar en la figura 21, la curva de accuracy en validación es bastante similar, y el resultado de accuracy es prácticamente el mismo. Sin embargo, ambas curvas tienen los picos más acentuados, cosa que va en detrimento de la estabilidad de la curva. Aun así los resultados son casi idénticos, pero como no son significativamente mejores, optamos por mantener el modelo anterior con capas Conv1D como mejor modelo.

Layer (type)	Output Shape	Param #	Activation	Regularizers	Initializers
gru (GRU)	(None, 30, 512)	804864	relu	l2(0.00025)	GlorotNormal
gru_1 (GRU)	(None, 30, 256)	591360	relu	l2(0.00025)	GlorotNormal
gru_2 (GRU)	(None, 30, 128)	148224	relu	l2(0.00025)	GlorotNormal
gru_3 (GRU)	(None, 64)	37248	relu	l2(0.00025)	GlorotNormal
dense (Dense)	(None, 32)	2080	relu	l2(0.0025)	GlorotNormal
dense_1 (Dense)	(None, 16)	528	relu	l2(0.00025)	GlorotNormal
dense_2 (Dense)	(None, 1)	17	sigmoid	-	-

Total params: 1,584,321
Trainable params: 1,584,321
Non-trainable params: 0

Optimizer : Adam
Learning Rate : 0.001
Loss : 'binary_crossentropy'
Batch_size : 2048

Figura 20: Arquitectura modelo con capas GRUs para ventana 120, lag 30

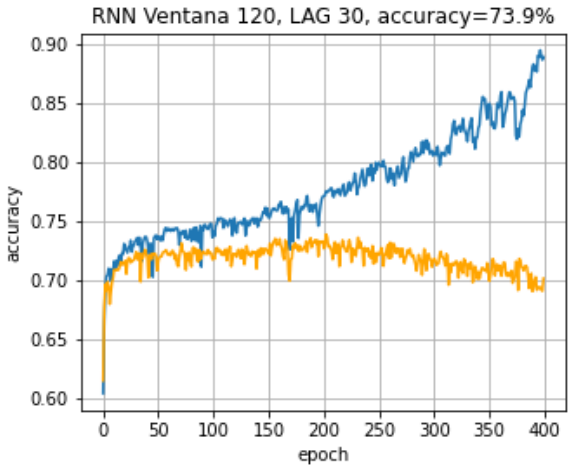


Figura 21: Curva entrenamiento para modelo con capas GRUs para ventana 120, lag 30

Ahora vamos a probar con capas convolucionales 2D. En este caso, en vez de decir que cada variable es un canal, vamos a darles la propiedad de estar en dos dimensiones y a asumir un solo canal. Esto tiene varios problemas que hacen que el modelo no esté bien desde el punto de vista teórico. Lo primero que asumimos es que la relación entre las distintas series existe, aunque eso no nos debería preocupar. El principal problema es que el orden en el que se encuentran las variables importa, pues es como si las estuviésemos poniendo una al lado de otra, lo que significa que, por ejemplo, que "perc_60" se encuentra entre "perc_120" y "diffemas_120_60", y a priori no existe ningún motivo para ese orden. De igual manera los extremos quedan condicionados. Es decir, al poner la data en dos dimensiones, estamos agrupado en esa segunda dimensión únicamente por el orden en el que se encuentran colocadas las variables, hecho que no es correcto. Aun así,

ya hemos visto con los últimos modelos que las generalidades existentes en el mundo del Deep Learning pueden no ser óptimas para nuestro problema, así que vamos a probar este nuevo modelo de todos modos. Realizamos una breve optimización, dando como resultado el modelo de la figura 22.

Layer (type)	Output Shape	Param #	Kernel_size	Activation	Padding	Regularizers	Initializers
conv2d (Conv2D)	(None, 30, 10, 256)	12800	7	relu	same	l2(0.0005)	GlorotNormal
conv2d_1 (Conv2D)	(None, 30, 10, 128)	819328	5	relu	same	l2(0.0005)	GlorotNormal
conv2d_2 (Conv2D)	(None, 30, 10, 64)	73792	3	relu	same	l2(0.0005)	GlorotNormal
flatten_4 (Flatten)	(None, 19200)	0	-	-	-	-	-
dense_4 (Dense)	(None, 1)	19201	-	sigmoid	-	-	-

Total params: 925,121
Trainable params: 925,121
Non-trainable params: 0

Optimizer : Adam
Learning Rate : 0.0001
Loss : 'binary_crossentropy'
Batch_size : 2048

Figura 22: Arquitectura modelo con capas conv2d para ventana 120, lag 30

La arquitectura es prácticamente idéntica a la del modelo con capas 1D. Veamos en la image 23 la curva de aprendizaje.

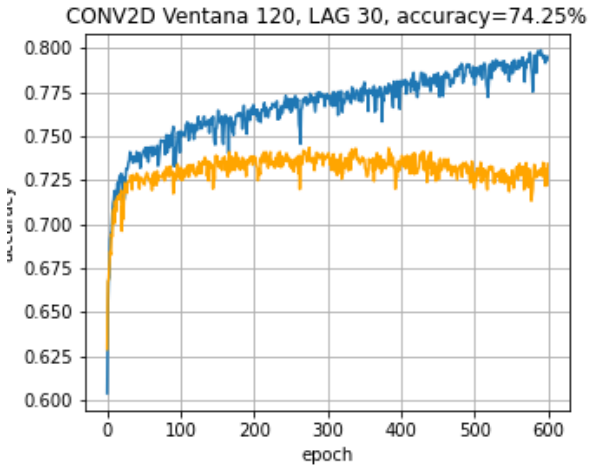


Figura 23: Curva entrenamiento para modelo con capas conv2d para ventana 120, lag 30

La curva tiene una estructura parecida a la del modelo con convolucionales 1D. El resultado es prácticamente el mismo, pero no hemos superado de manera significativa sus resultados, por lo que continuamos asumiendo que el modelo 1D sigue siendo el óptimo. Aunque podríamos plantearnos cómo es posible que este modelo, que teóricamente no es del todo correcto, tenga los mismos resultados que el de 1D. Tal vez esté ignorando la segunda dimensión que le hemos incluido a los datos y solo trabaje con la primera, lo que lo convierte en un problema idéntico al anterior. Es solo una suposición, pero se podría estudiar por qué realmente ha ocurrido que los resultados fuesen tan similares. Por último, hemos probado a "jugar" con las salidas de las

capas usando keras funcional. Estos dos modelos no los vamos a presentar de una manera “formal” puesto que no podemos demostrar matemáticamente que lo que hemos planteado tenga sentido, ni tampoco se lo podemos dar teóricamente desde un punto de vista estricto. Vamos a expresar simplemente por qué motivos creemos que pueden funcionar, y luego vamos a entrenarlos a ver qué resultados nos dan.

El primer modelo que hemos planteado es idéntico al óptimo de convolucionales 1D, pero en cada capa oculta(excepto en la primera), su entrada es la salida de la capa anterior concatenada con los inputs del algoritmo, o sea, las variables. Lo que intentamos con esto es que cada capa ‘’recuerde” las variables de entrada además de la salida de la capa anterior. Veamos ahora su estructura en la figura 24 y la curva de aprendizaje en la figura 25.

Layer (type)	Output Shape	Param #	Connected to	Kernel_size	Activation	Padding	Regularizers	Initializers
input_1 (InputLayer)	[(None, 30, 10)]	0		-	-	-	-	-
conv1d (Conv1D)	(None, 30, 256)	18176	input_1[0][0]	7	relu	same	l2(0.00075)	GlorotNormal
tf.concat (TFOpLambda)	(None, 30, 266)	0	conv1d[0][0] input_1[0][0]	-	-	-	-	-
conv1d_1 (Conv1D)	(None, 30, 128)	170368	tf.concat[0][0]	5	relu	same	l2(0.00075)	GlorotNormal
tf.concat_1 (TFOpLambda)	(None, 30, 138)	0	conv1d_1[0][0] input_1[0][0]	-	-	-	-	-
conv1d_2 (Conv1D)	(None, 30, 64)	26560	tf.concat_1[0][0]	3	relu	same	l2(0.00075)	GlorotNormal
tf.concat_2 (TFOpLambda)	(None, 30, 74)	0	conv1d_2[0][0] input_1[0][0]	-	-	-	-	-
flatten (Flatten)	(None, 2220)	0	tf.concat_2[0][0]	-	-	-	-	-
dense (Dense)	(None, 1)	2221	flatten[0][0]	-	sigmoid	-	-	-

Total params: 217,325
Trainable params: 217,325
Non-trainable params: 0

Optimizer : Adam
Learning Rate : 0.0001
Loss : 'binary_crossentropy'
Batch_size : 2048

Figura 24: Arquitectura primer modelo funcional con ventana 120, lag 30

En la figura 25 se puede ver como la curva es bastante estable, y consigue igualar los resultados del modelo óptimo, aunque nuevamente, como nos los supera significativamente, descartamos este modelo.
Ahora pasemos al segundo modelo. En este, cada capa oculta funciona de manera estándar, pero tras acabar la tercera capa convolucional, la capa de “Flatten” no va solamente a aplanar la salida de la última capa, sino que va a concatenar la salida de las tres capas y de las variables de entrada. De esta manera queremos que la parte final del modelo tenga en cuenta no solo la salida de la última capa, sino de todas las capas y de las variables de entrada, para que las pueda “recordar” y ajustar los pesos para cada uno de los distintos niveles de profundidad de las capas ocultas del modelo. Veamos la arquitectura , figura 26 y los resultados 27.

El resultado es prácticamente idéntico al anterior, así que nuevamente descartamos el modelo pues no mejora los resultados anteriores.
Hemos probado muchas arquitecturas distintas, algunas más lógicas y otras menos lógicas. Todas han dado resultados parecidos, sin destacar ninguna; por ello, vamos a asumir que el mejor modelo para nuestra ventana de 120 días es el que obtuvimos con capas convolucionales 1D y un LAG de 30.
Pasemos ahora a la ventana de 365 dias.

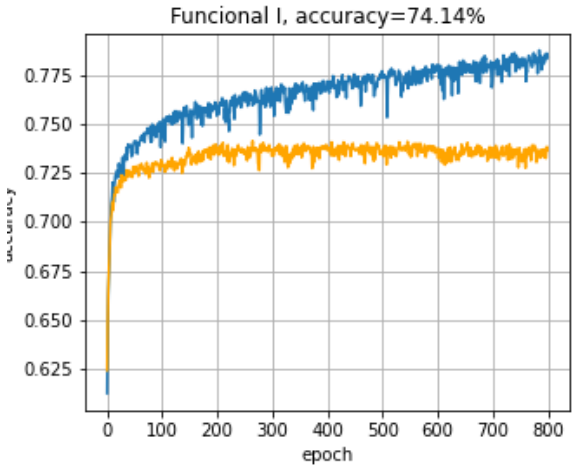


Figura 25: Curva entrenamiento primer modelo funcional

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 30, 10)]	0	
conv1d_3 (Conv1D)	(None, 30, 256)	18176	input_2[0][0]
conv1d_4 (Conv1D)	(None, 30, 128)	163968	conv1d_3[0][0]
conv1d_5 (Conv1D)	(None, 30, 64)	24640	conv1d_4[0][0]
tf.concat_3 (TFOpLambda)	(None, 30, 458)	0	input_2[0][0] conv1d_3[0][0] conv1d_4[0][0] conv1d_5[0][0]
flatten_1 (Flatten)	(None, 13740)	0	tf.concat_3[0][0]
dense_1 (Dense)	(None, 1)	13741	flatten_1[0][0]

Total params: 220,525
Trainable params: 220,525
Non-trainable params: 0

Figura 26: Arquitectura segundo modelo funcional con ventana 120, lag 30

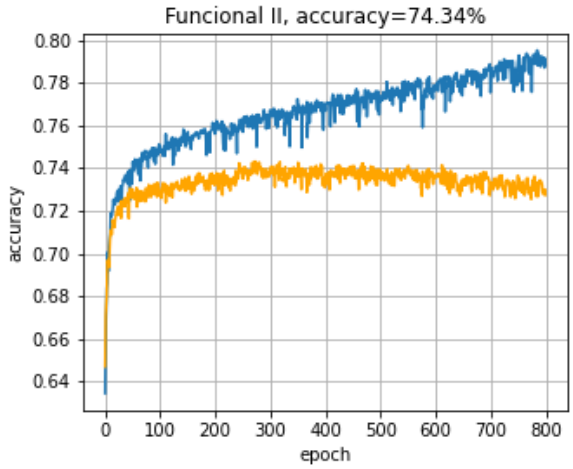


Figura 27: Curva entrenamiento segundo modelo funcional

6.3.2. Ventana 365

Ahora vamos a buscar el mejor modelo para el *Ratio* con la ventana de 365 datos. No vamos a volver a explicar todo el proceso que ya hicimos con la ventana de 120 pues es casi idéntico, vamos a pasar directamente a comparar los resultados. Lo que sí que hemos de tener en cuenta es que ahora nuestras clases van a encontrarse en diferente proporción respecto a la ventana de 120, y también que vamos a disponer de menos datos para entrenar y para validar.

Usando el LAG 30, vemos que disponemos de 7708 datos en train, 2584 en validación, y 4423 en test. Como dijimos anteriormente, subir la ventana normalmente suele traer operaciones más largas y robustas, con menos ruido, pero provoca el disponer de menos datos para entrenar. Veremos si esta disminución en la cantidad de datos afecta al entrenamiento o no. El porcentaje de datos con Take profit y con Stop Loss también habrá variado. El porcentaje de Take Profits que tenemos en cada conjunto es: 53.84 % en train, 44.07 % en validación, y 51.4 % en test. Las clases siguen balanceadas, pero puede que el sesgo existente entre train y validación sea un problema. Teniendo esto en cuenta, debemos ver que nuestro benchmark ha variado: antes partíamos de un 59 % de acierto en validación, mientras que ahora partimos de un 44 %, lo que implica que casi con toda seguridad, el accuracy que obtendremos en validación será considerablemente menor. Pero debemos recordar, como bien vimos en el algoritmo sin IA, que las ganancias por operación con la ventana de 365 eran bastante mayores a las de 120, lo que implica que, aun a pesar de tener una menor tasa de acierto, ganamos más desde el punto de vista del trading, por lo que el hecho de que obtengamos un menor accuracy con 365 no implica que los resultados vayan luego a ser peores. Es más, al tener solo un 44 % de acierto inicial, podría decirse que tenemos ‘’más recorrido” para mejorar esa tasa de acierto, y además estos aciertos nos reportarán en el futuro más ganancias monetarias que la ventana de 120. Pasemos ahora a buscar el mejor modelo.

Primero veamos las 4 curvas de aprendizaje de los modelos con LAG 30,60, 90 y 150. La arquitectura ha sido la misma que la usada con la ventana de 120, excepto porque ahora varían tanto los valores de l2 como los learning rate óptimos para cada LAG.

Como vemos en la figura 28, nuevamente el mejor LAG es 30, aunque los resultados apenas varían. En

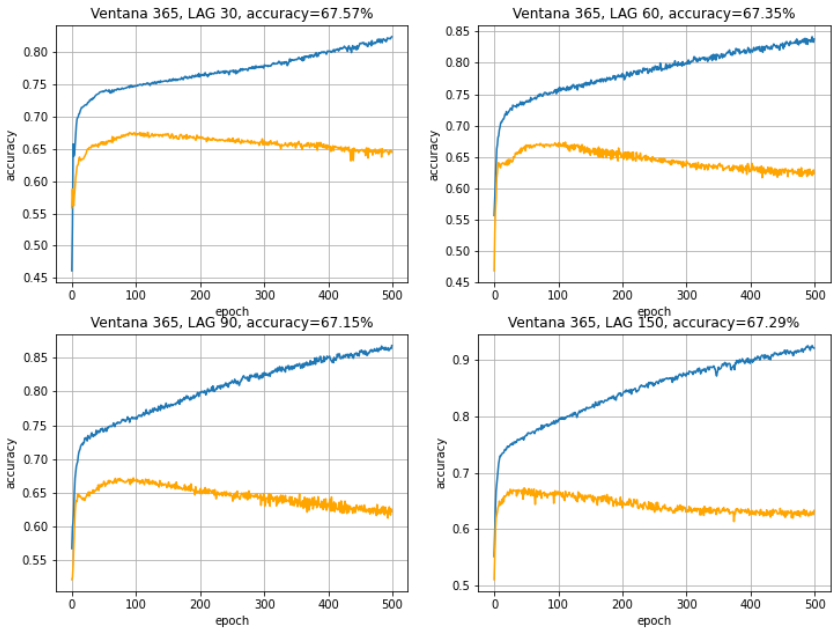


Figura 28: Curva entrenamiento modelo conv1d, ventana 365, lag 30, 60, 90, 150

este caso las cuervas de validación son mucho más parecidas entre los distintos valores del LAG. Tienen un crecimiento estable hasta llegar a su máximo, forman el codo y empiezan a bajar debido a la sobre optimización. Las curvas tienen un crecimiento bastante bueno, por lo que consideramos que las curvas son buenas. Vemos también que, como ya anunciamos anteriormente, el accuracy obtenido en validación es del 67.5 %, 7 puntos menor que con 120; en cambio, con 120 pasamos de un 59 % a un 74 %, mientras que aquí hemos pasado de un 44 % a un 67 %, con lo que hemos conseguido mejorar mucho más los resultados respecto a nuestro benchmark sin IA. Ahora vamos a ver el gráfico de LAG 20, a ver si mejora los resultados de LAG 30.

Vemos en la figura 29 que los resultados no son significativamente mejores, por lo que nos quedamos, nuevamente, con el valor de 30 como LAG óptimo. Ahora vamos a ver en la figura 30 exactamente como es la arquitectura de este modelo óptimo para la ventana de 365 días.

Este modelo tiene tres capas convolucionales 1D, con filtros 256, 128 y 64 en orden decreciente, con kernel sizes 7,5,3, exactamente igual que con la ventana de 120 días. Las funciones de activación son “Relu”, y el padding “same”. Utilizan regularización l2 con un valor de 0.0005. Igual que antes no incluye capas densas ocultas, solo la capa densa de salida sigmoide. Volvemos a usar el optimizador Adam, pero en este caso con un learning rate de 0.0005. El batch size se mantiene en 2048.

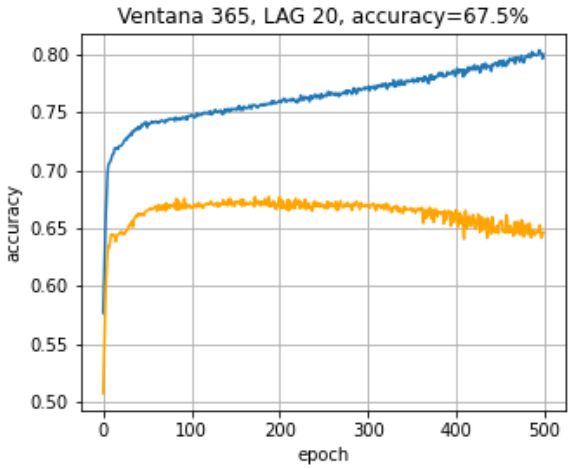


Figura 29: Curva entrenamiento modelo conv1d, ventana 365, lag 20

Layer (type)	Output Shape	Param #	Kernel_size	Activation	Padding	Regularizers	Initializers
conv1d_12 (Conv1D)	(None, 30, 256)	18176	7	relu	same	l2(0.0005)	GlorotNormal
conv1d_13 (Conv1D)	(None, 30, 128)	163968	5	relu	same	l2(0.0005)	GlorotNormal
conv1d_14 (Conv1D)	(None, 30, 64)	24640	3	relu	same	l2(0.0005)	GlorotNormal
flatten_4 (Flatten)	(None, 1920)	0	-	-	-	-	-
dense_4 (Dense)	(None, 1)	1921	-	sigmoid	-	-	-

Total params: 208,705
Trainable params: 208,705
Non-trainable params: 0

Optimizer : Adam
Learning Rate : 0.00005
Loss : 'binary_crossentropy'
Batch_size : 2048

Figura 30: Curva entrenamiento modelo conv1d, ventana 365, lag 30

Podemos observar como el modelo óptimo para la ventana de 120 es idéntico al de 365 excepto por leves variaciones en el l2 y el learning rate, lo que demuestra la robustez del modelo. Para no sobrecargar demasiado el trabajo no vamos a incluir nuevamente los modelos con Conv2D, GRUs ni con keras secuencial, pues ya ha quedado sobradamente claro cuál es la arquitectura óptima para ambos valores de las ventanas.

Ya tenemos los dos modelos óptimos para la ventana de 120 y la de 365. Ahora, vamos a entrenarlos usando un early stopping con el objetivo de que dejen de entrenar en un momento lo más óptimo posible. Estos dos modelos serán los que usaremos más adelante para realizar los backtest sobre el conjunto de validación (pues recordemos que el algoritmo de backtest o de gestión de capital también tiene parámetros que hemos de ajustar, y eso debemos hacerlo en el conjunto de validación, para luego aplicarlo al conjunto de test sin dejarnos influenciar por él).

6.4. Creación y análisis de los modelos

Ahora que ya tenemos las arquitecturas e hiperparámetros óptimos para las ventanas de 120 y 365, vamos a pasar a entrenar esos dos modelos un número de épocas óptimo. Puesto que ya sabemos cuál es aproximadamente su accuracy en validación máximo y donde se encuentra el codo de la curva de accuracy, sabemos de forma aproximada en que momento debería pararse el entrenamiento. Vamos a usar un early stopping con patience $\bar{5}00$ y delta $\bar{0}.005$. En el modelo de 120 el accuracy en validación se para en 73.97 % y en 365 se para en 67.33 %, justo sobre el codo. Estos son los modelos que vamos a utilizar para hacer los dos backtest sobre el conjunto de validación.

Ahora vamos a pasar a estudiar los modelos en mayor profundidad, pues queremos ver como se distribuye la probabilidad de cada clase, como varían las métricas al variar el umbral, o ver que variables son más importantes.

6.4.1. Ventana de 120

Para el modelo de ventana 120, lo primero que vamos a hacer va a ser ver la distribución de las probabilidades de la predicción de cada una de las clases(o sea la salida de la última capa sigmoide de nuestra red, que nos indica la probabilidad de que el dato sea de una clase o de otra).

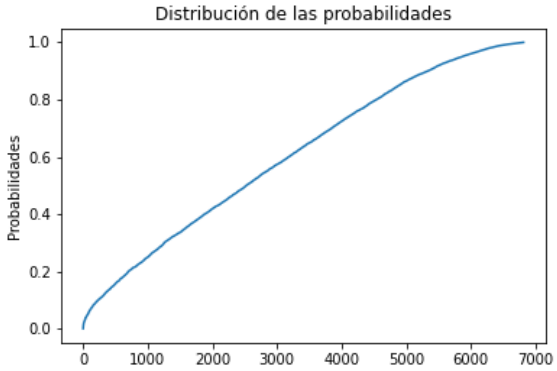


Figura 31: Distribución de probabilidades

Como vemos en la imagen 31, las probabilidades se distribuyen de manera casi uniforme y no sigmoide, como podríamos esperar. Probablemente esto produzca que otras métricas también se distribuyan de manera uniforme. Ahora vamos a ver nuestra matriz de confusión en la imagen 32 y sus métricas en la imagen 33.

Podemos apreciar que la precisión, el recall y el f1 son algo mayores en la clase 1 (recordemos que era TP) que en la clase 0, puede que esto se deba a que tenemos un 60 % de 1s. La métrica que más nos interesa es la precisión de la clase 1.

Hay distinas formas de afrontar este problema: podemos intentar maximizar el recall con el fin de encontrar todos aquellos datos en los que la etiqueta es TP aun a pesar de incluir etiquetas de SL, pero desde el punto de vista de la inversión esto no nos interesa. Maximizar el recall podría ser el objetivo en, por ejemplo, un algoritmo que busque terroristas en un aeropuerto, pues en este problema lo que quieres es

real	0.0	1.0
pred		
0.0	1730	795
1.0	1005	3286

Figura 32: Matriz de confusión

	precision	recall	f1-score	support
0.0	0.69	0.63	0.66	2735
1.0	0.77	0.81	0.78	4081
accuracy			0.74	6816
macro avg	0.73	0.72	0.72	6816
weighted avg	0.73	0.74	0.73	6816

Figura 33: Métricas de la matriz de confusión

que todos los terroristas sean etiquetas como tal, y no te importa etiquetar indebidamente a personas que no son terroristas (pues asumimos que cuando intervengan los "humanos" podrán deshacer el error), y de esta forma no importa la precisión que obtengamos, solo te importa el recall (aunque habría que tener en cuenta otros factores externos como la cantidad máxima de personas que puedes "detener" pero eso sería ya estudiar el contexto de dicho problema). La segunda forma de afrontar nuestro problema es maximizar la precisión. No nos importa perdernos muchos Take Profits, no tenemos que entrar en todas y cada una de las empresas calificadas como 1, lo que queremos es asegurarnos de que aquellas empresas en las que entremos vayan a salir en Take Profit. Queremos maximizar la tasa de acierto únicamente de aquellas empresas que nuestro algoritmo califique como TP, para de esta manera tener más operaciones positivas (recordemos que estamos en el apartado de IA, y nuestro objetivo es simplemente mejorar nuestro acierto independientemente luego de los rendimientos, eso lo ajustaremos más adelante con el algoritmo de asignación de capital).

Una forma posible de maximizar la precisión es subir el umbral de predicción, que por defecto está en 0.5. Si únicamente calificamos como 1 aquellos datos cuya probabilidad sea mayor de, por ejemplo, 0.7, probablemente nuestro accuracy disminuirá, el recall bajará muchísimo, pero aumentaremos la precisión, que es nuestro objetivo. Pero surge un problema: si aumentamos mucho el umbral, es cierto que subirá la precisión, pero también bajará significativamente el número de empresas que nuestro algoritmo diga que compremos, y como ya dijimos anteriormente, hacer pocas operaciones trae muchos problemas: falta de diversificación, aumento del riesgo, ser más susceptible al azar, resultados estadísticamente no significativos... Por eso debemos encontrar un punto medio entre maximizar la precisión y tener un número suficientemente alto de operaciones. Esto lo haremos en el algoritmo de asignación de capital pero podemos ayudarle viendo como varían la precisión y el número de datos con predicción positiva en función del umbral.

En la gráfica 34 vemos como la función es prácticamente lineal, como consecuencia de lo que dijimos al principio: la distribución de las probabilidades de la red es uniforme. Hay algo que debemos tener en cuenta: a veces cuando se hacen proyectos con algoritmos que en realidad no son capaces de aprender sobre sus datos,

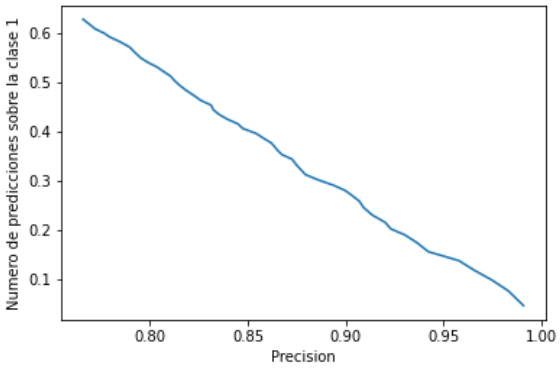


Figura 34: Variación de la precisión en función del número de datos etiquetados como TP

ocurre que subir el umbral no está correlacionado con el aumento de la precisión, es decir, que aumentar el umbral no garantiza que la precisión aumenta, sino que muchas veces sube y baja de manera aleatoria. Esto en principio no tiene sentido, pues si el umbral es más restrictivo, la precisión debería subir, es un concepto básico, pero la razón de que eso no ocurra es que el algoritmo en realidad no está aprendiendo sobre el conjunto de datos. Para nuestro problema hemos visto que el aumento del umbral está directamente relacionado con el aumento de la precisión, lo que es una importante prueba de que nuestro algoritmo realmente ha sido capaz de aprender nuestra data.

Ahora vamos a ver otro concepto importante en problemas de clasificación: la curva ROC, que relaciona la sensibilidad con la especificidad mediante la variación del umbral. La medida de la curva ROC es el AUC (área bajo la curva); cuánto mayor sea el AUC, mejor será el modelo (de acuerdo a dicha métrica). El bechmark de una curva ROC es una línea diagonal, cuyo AUC es de 0.5. Veamos nuestra curva ROC en la figura 35.

Tenemos una ROC bastante estable (en problemas con algunos sesgos a veces hay picos o saltos) con un AUC de un 80 %, lo que son buenos resultados y, sobre todo, robustos.

Ahora vamos a pasar a estudiar la importancia de nuestras variables. Hay muchas formas de estudiar las variables, pero nosotros vamos a usar una. Vamos a entrenar un Random Forest que minimice el Gini, que es un criterio de información de los datos, parecido a la entropía, que es otra medida que se usa en los Random forest. En este caso nuestra data perderá su condición de serie temporal, pues los Random Forest solo funcionan con data en dos dimensiones.

El Gini de la imagen 36 ha dado más importancia a la desviación y a las diferencias entre las "EMAS", y menos peso a los percentiles. Vamos a hacer dos pruebas para ver realmente como afectan estas variables a nuestro modelo. Lo primero que vamos a hacer es volver a entrenarlo quitando primero las dos peores variables, las de los percentiles.

En la figura 37 vemos como la curva y el accuracy no han tenido variaciones significativas habiendo quitado las dos variables con un menor Gini.
Veámoslo ahora quitando la variable más importante, ''desv_120".

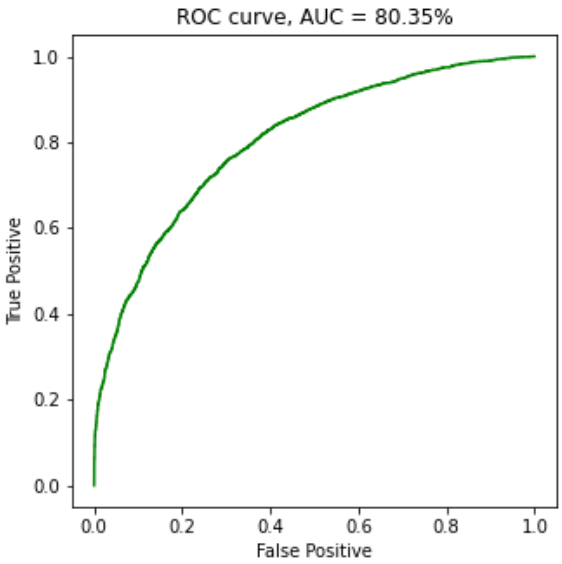


Figura 35: Curva ROC

	gini
desv_120	0.161955
diffemas_60_25	0.145309
diffemas_120_60	0.135080
diff_120	0.130254
ratio	0.125708
diff_80	0.093654
desv_60	0.087630
diff_40	0.086973
perc_120	0.018757
perc_60	0.014681

Figura 36: Coeficiente Gini de las variables

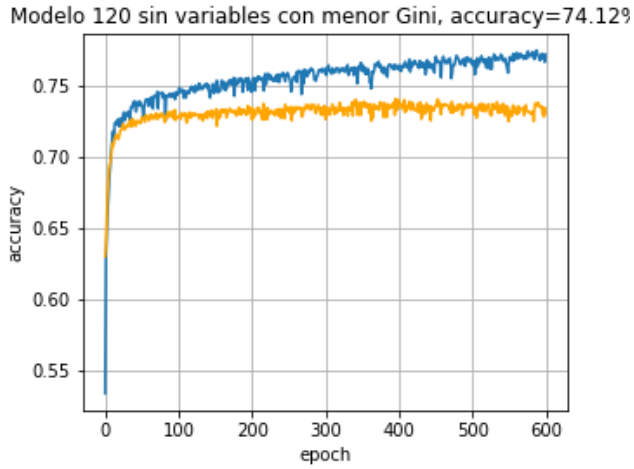


Figura 37: Curva aprendizaje eliminando las variables menos significativas

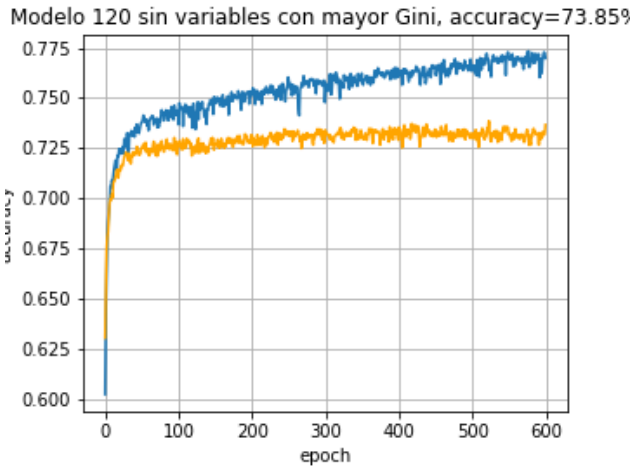


Figura 38: Curva aprendizaje eliminando la variable menos significativa

En la figura 38 se puede observar que el accuracy ha disminuido muy ligeramente; parece que efectivamente quitar esta variable haga que los resultados sean peores, pero la pérdida es tan ínfima que no la consideramos significativa, por lo que vamos a dejar nuestro modelo final para la ventana de 120 con todas las variables.

6.4.2. Ventana de 365

Como ya hemos visto todos los conceptos, vamos a pasar directamente a ver los resultados de cada uno de nuestros elementos. Empezamos por la distribución de las probabilidades.

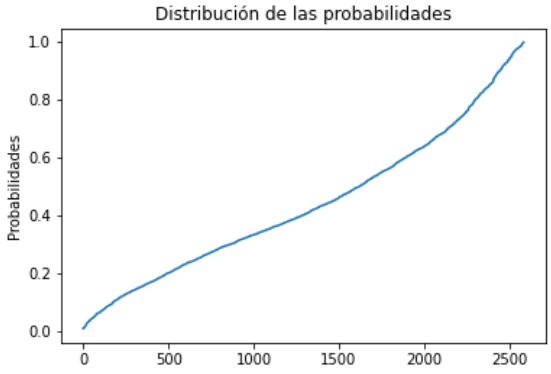


Figura 39: Distribución de probabilidades

En la imagen 39 vemos como sigue pareciéndose mucho a una uniforme, pero en este caso es ligeramente más sinusoidal, donde las probabilidades tienden a irse hacia los extremos. Veamos ahora la matriz de confusión de la figura 40 y explicadas en la figura 41

real	0.0	1.0
pred		
0.0	1102	515
1.0	343	624

Figura 40: Matriz de confusión

	precision	recall	f1-score	support
0.0	0.68	0.76	0.72	1445
1.0	0.65	0.55	0.59	1139
accuracy			0.67	2584
macro avg	0.66	0.66	0.66	2584
weighted avg	0.67	0.67	0.66	2584

Figura 41: Métricas de la matriz de confusión

Tenemos una precisión de un 65 % como cabía esperar, ligeramente inferior al accuracy. En este caso vemos como los resultados son algo mejores para la clase 0, pues recordemos que con 365 solo había un 44 % de 1s. Veamos ahora la gráfica 42 que muestra la precisión y los datos predichos como positivos en función de la variación del umbral de predicción.

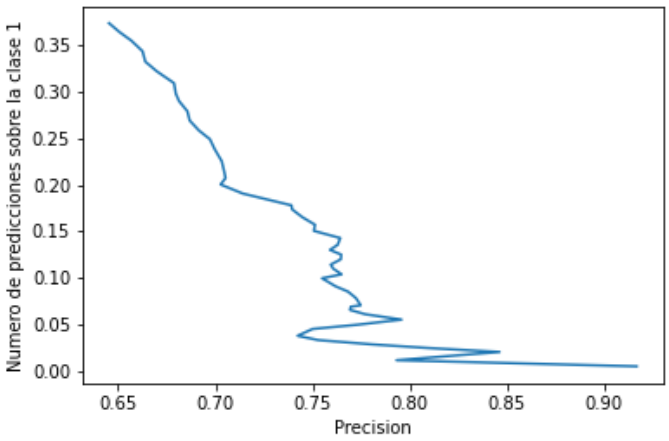


Figura 42: Variación de la precisión en función del número de datos etiquetados como TP

Aquí nos surge el primer problema: la curva deja de ser uniforme para tener importantes retrocesos. Esto podría deberse a que el modelo no ha aprendido correctamente, pero no es así. Si recordamos, en los inicios de este trabajo, cuando tratamos las comparaciones entre la ventana de 120 y de 365 días, una de las cosas que dijimos entonces fue que el usar una ventana más larga traía como consecuencia el tener una menor cantidad de datos, sobre todo en el conjunto de validación. Con la ventana de 120, teníamos 14353 datos en train y 6816 en validación, mientras que con la ventana de 365 tenemos 7708 en train y 2584 en validación. La disminución de los datos en validación puede hacer que nuestros resultados no sean del todo estadísticamente significativos, y eso es lo que ocurre con la gráfica anterior: al llegar a umbrales entre 0.75 y 0.85, la falta de datos provoca esos picos que ensucian nuestra curva.
Podemos ver como la tendencia subyacente es positiva, va cayendo de manera casi uniforme, pero esos umbrales con altas volatilidades pueden llegar a ser un problema, sobre todo porque disminuimos la cantidad de datos etiquetados como positivos, lo que implica menos operaciones, y además estos resultados tienen menos tasa de acierto. Llegados a este punto podemos aventurarnos a decir que los backtest con ventana de 365 y umbral entre 0.7 y 0.85 van a salir considerablemente peores que los de 120 y los de 365 con umbral menor de 0.7.

Este problema podría solucionarse de varias maneras. Una de ellas sería pasar algunos datos de train al conjunto de validación, de esta manera no solo igualaríamos un poco el balanceo de las clases, sino que tendríamos más datos y ello mejoraría la curva. Para probar nuestra teoría, lo que hemos hecho ha sido pasar los últimos 1.250 datos de train al conjunto de validación y reentrenar el modelo. Veamos ahora cual es la nueva curva que relaciona la precisión y la cantidad de datos etiquetas como positivos respecto a los umbrales en la figura 43.

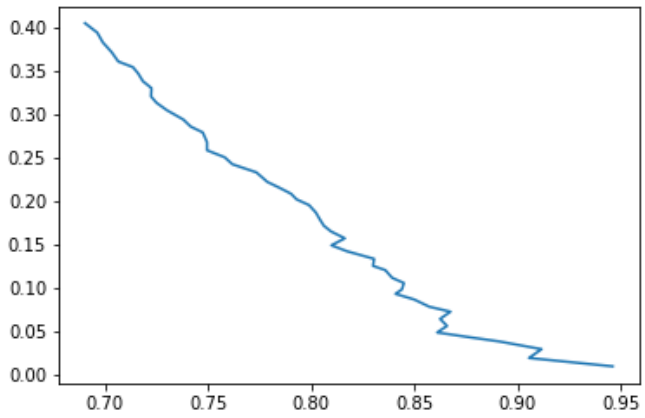


Figura 43: Variación de la precisión transfiriendo datos de train a validación

Vemos que, efectivamente, nuestra hipótesis era cierta, ahora la curva es mucho más estable y apta para nuestro problema. Hemos ido probando pasando datos en grupos de 250, y se ha visto claramente como según aumentábamos los datos, la curva se iba haciendo más estable y presentaba menos picos. Sin embargo, ahora debemos plantearnos si queremos implementar esta nueva solución o no. Está claro que nuestro modelo funcionaría mejor pasando esos 1.250 datos de train a validación. El problema es que estamos modificando nuestro conjunto de validación, y por ende, el conjunto en el que probaremos nuestro backtest. Esto es más bien un problema del contexto, del negocio. Si este fuese nuestro conjunto de test y nos "obligasen" a realizar el backtest sobre dicho conjunto, nosotros como desarrolladores no podríamos decidir "ampliar" el conjunto de test para que los resultados nos salgan mejores, pues nuestros jefes o clientes quieren un backtest en ese periodo en concreto. Por ello, tenemos que asumir nuestras decisiones y dejar el modelo tal y como está, con el conjunto de validación inicial, pues se puede dar el caso en que no tuviésemos la libertad para ampliar el conjunto con el fin de sacar mejores resultados. Por ello, nos quedamos con los conjuntos y el modelo inicial, aunque eso nos traiga luego problemas al hacer el backtest con umbrales altos. Como ya dijimos, es el coste que hay que pagar por utilizar ventanas temporales grandes. Aun así, 365 puede tener buenos resultados con umbrales pequeños, lo veremos más adelante. Continuamos ahora con el análisis del modelo. Una vez estudiada la curva de precisiones y umbrales, pasamos a ver la curva ROC en la figura 44.

Vemos que la curva ROC tiene ahora un AUC del 71 %; este resultado era de esperar, pues ya vimos que el accuracy y las demás métricas eran más bajas en este modelo. Ahora pasamos a estudiar la importancia de las variables Primeramente vamos a ver el Gini de un random forest en la figura 45

Los percentiles siguen siendo las variables con menos importancia, pero en este caso las más importantes han sufrido algún cambio: La diferencia entre las EMAS largas ha bajado varias posiciones, al igual que la desviación típica, pero las diferencias absolutas han sufrido un repunte. Al igual que hicimos con 120, vamos a ver cómo reacciona nuestro modelo si eliminamos por un lado las variables con menor Gini y luego las variables con mayor Gini. Veamos en la figura 46 la curva quitando las dos variables de los percentiles.

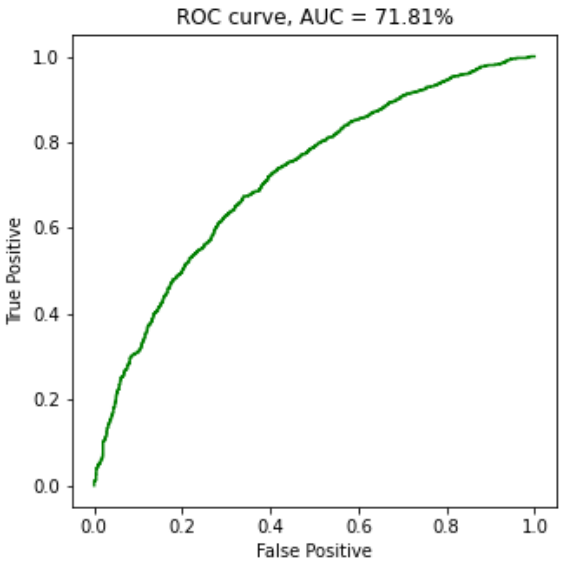


Figura 44: Curva ROC

	gini
diffemas_180_80	0.188390
diff_240	0.126744
diff_365	0.122649
ratio	0.120675
desv_365	0.111737
diffemas_365_180	0.106088
diff_120	0.082663
desv_180	0.078008
perc_365	0.033268
perc_180	0.029779

Figura 45: Coeficiente Gini de las variables

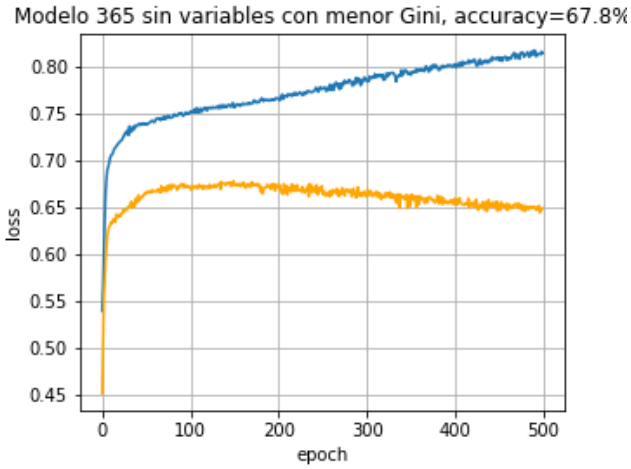


Figura 46: Curva de aprendizaje eliminando las variables menos significativas

Vemos que el accuracy ha subido varias décimas, con lo que evidenciamos que realmente las variables de los percentiles no aportan mucho, pero la ganancia tampoco es tan grande como para rehacer el modelo. Veamos ahora, en la figura 47, los resultados quitando la variable más importante, ''diffemas_180_80".
Vemos como en este caso si que ha disminuido considerablemente el accuracy, por lo que esta variable parece

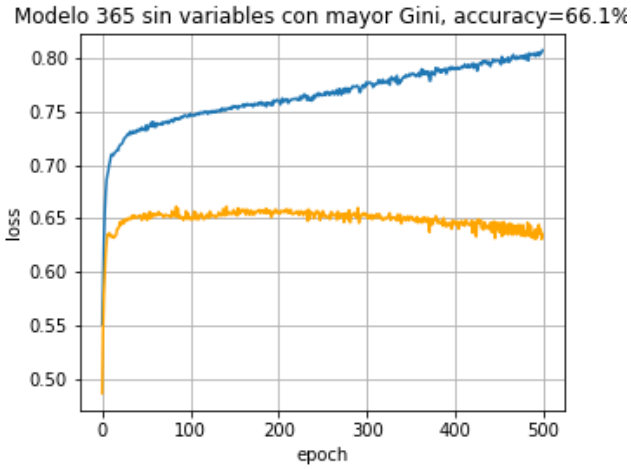


Figura 47: Curva aprendizaje eliminando la variable menos significativa

que tiene más importancia que las demás; además, el accuracy ha disminuido más en este modelo que en el de 120. Aún así, como la eliminación de variables peores no nos ha proporcionado un gran salto en el accuracy, vamos a mantener el modelo como lo teníamos, con todas sus variables.

7. Gestión de la cartera

7.1. Introducción

Una vez finalizada la creación de nuestro mejor modelo de Inteligencia Artificial, vamos a pasar a crear el algoritmo de gestión de capital. A lo largo de este trabajo hemos establecido el dominio de acciones en las que vamos a operar, hemos procesado la data, establecido las reglas básicas de entrada, y creado redes neuronales que nos ayuden a decidir cuándo debemos y cuando no debemos entrar a mercado. Cada vez que el Ratio de una empresa corta la segunda desviación típica, debemos decidir si comprar o abstenernos. Vimos los resultados (teóricos, sin gestión de capital) de operar en todas las ocasiones en la sección "Algoritmo sin IA". A continuación, vimos que podíamos crear y entrenar algoritmos basados en redes neuronales que podían filtrar muchas de estas operaciones, de manera que nuestro porcentaje de acierto aumentase. Con la ventana de 120 llegamos a obtener un 74.2 % de accuracy, y en la de 365 un 67.5 %. Sin embargo, con estos modelos no tenemos en cuenta ni qué empresas estamos prediciendo, ni en qué fecha, ni nada ajeno a solamente esa operación en concreto. Para convertir estas predicciones en un algoritmo de trading viable, lo que necesitamos es añadirle un modelo de gestión de capital, pues las predicciones son solo una parte del algoritmo de inversión. Tenemos que responder a muchas preguntas. ¿Cuánto dinero invierto en cada operación? ¿Podemos comprar dos veces la misma empresa? ¿Cuál es el umbral óptimo para clasificar una operación como TP o SL?

Como ya dijimos anteriormente, queremos maximizar la rentabilidad teniendo un alto control sobre el riesgo y nuestra exposición al mercado. Hemos decretado que no queremos tener invertido simultáneamente más del 80 % del capital del que disponemos pero, sin llegar a superar esa cantidad, queremos quedarnos lo más cerca posible de ella, pues si no invertimos un alto porcentaje del capital, nuestra rentabilidad se verá mermada. Todas las decisiones que vamos a tomar tienen algo positivo y algo negativo.

Hay que destacar, que todas las pruebas y la optimización del algoritmo de gestión de capital lo realizamos en el conjunto de validación, al igual que los modelos de IA. Una vez establecidas las reglas e hiperparámetros, lo ejecutaremos en test, sin poder a posteriori cambiar ningún aspecto del algoritmo tras ver los resultados en test, pues si esto fuese producción no podríamos hacerlo. No se pueden usar los resultados de test para modificar los modelos pasados. Hacerlo implicaría sobreoptimizar y hacer trampas, ya que estaríamos optimizando, viendo lo que pasa en el futuro.

Vamos a establecer algunas de las preguntas más importantes que debemos hacernos para decidir a la hora de crear nuestro algoritmo de gestión de capital. Hay muchas pequeñas cuestiones a las que debemos responder, y las responderemos a continuación, pero estas son las más importantes y generales.

- Cuánto capital asignar por operación: si metemos mucho capital a cada operación, por ejemplo, un 20 %, solamente podremos tener en cartera 4 acciones como mucho al mismo tiempo. Esto hace que baje nuestra diversificación, que aumente el riesgo y que estemos más expuestos a sucesos extraordinarios de acciones individuales. Por ello, es conveniente que esa cantidad sea menor, para así poder diversificar el 80 % del capital entre más empresas, pero hay una pega; si no tenemos suficientes señales de entrada, es posible que el dinero invertido cada día no sea suficientemente alto, y como hemos dicho antes, si tenemos poco capital invertido, nuestros rendimientos serán bajos. Sin embargo, si tenemos una gran cantidad de señales de entrada, podemos permitirnos el invertir cantidades de capital más pequeña, pues si hay muchas operaciones susceptibles de ser cogidas, nos acercaremos con facilidad a ese 80 % de capital invertible. Por ello, debemos ajustar muy cuidadosamente el capital a invertir por operación de acuerdo al número total de operaciones de que dispongamos.
- Comprar dos veces la misma empresa: si optamos por añadir capital a una empresa ya en cartera, estamos aumentando nuestra exposición a dicho activo, cosa que no nos beneficia, ya que reducimos la

diversificación. Pero si no tenemos un numero suficientemente alto de operaciones como para diversificar correctamente ni como para usar un porcentaje de capital pequeño para invertir en cada empresa, esta podría ser una opción.

- Qué umbral de predicción seleccionar: las redes neuronales nos dicen la probabilidad que tiene un dato de acabar en TP o en SL. Si seleccionamos como TP aquellos datos cuya probabilidad esté por encima de un umbral, al que llamaremos trigger, aumentamos la precisión del modelo (esto lo explicamos detalladamente en la sección de Algoritmo con IA) pero disminuye el número de operaciones. Estamos viendo como el número de operaciones realizadas es algo fundamental en cualquier algoritmo de trading.
- Calcular el dinero a asignar a una operación teniendo en cuenta el capital con flotante o el capital sin flotante: esta es una cuestión importante, pues a la hora de calcular el capital a invertir nos surge la duda de qué capital utilizar. Si asumimos que el dinero que estamos ganando o perdiendo con las operaciones activas "es nuestro" podemos usarlo para calcular el capital a invertir, aunque es posible que estemos adelantando acontecimientos. Si usamos el capital sin flotante, solo tenemos en cuenta el capital de las operaciones ya cerradas, pero es posible que el cálculo del capital a meter en una operación vaya con retraso, tanto si estamos ganando como si estamos perdiendo.
- Establecer la prioridad en la asignación de capital cuando hay varias acciones disponibles: si en un mismo día hay varias empresas en las que poder operar, pero no hay suficiente capital para entrar en todas, debemos decidir a cuáles darles prioridad. Hay muchos factores a tener en cuenta a la hora de crear nuestro algoritmo de asignación de capital, como acabamos de ver. Nosotros nos hemos planteado todas estas cuestiones y más, hemos hecho muchas pruebas y optimizado los distintos hiperparámetros, y al final hemos creado una serie de reglas para crear nuestro algoritmo final de inversión.

7.2. Descripción

Para la creación del algoritmo, se ha partido de las siguientes suposiciones:

- Los dividendos se reinvierten de manera implícita ya que la serie está ajustada a dividendos.
- Las ventas y las compras se realizarán en la apertura, nada más abrir el mercado y en ese orden para así poder disponer de liquidez para poder hacer compras que se puedan hacer. Cuando realicemos una venta, veneremos toas las acciones de la acción correspondiente; no usamos ventas partciales de posiciones. No podremos coprar un activo en el que ya estamos invertidos.
- Las comisiones de compra y venta serán un 0.1 % del capital total de compra o venta. Este umbral se ha tenido en cuenta considerando las distintas comisiones de los distintos brokers. En concreto nos hemos centrado en "Darwinex" e "Interactive Brokers".
- El capital libre para las operaciones se asignará por orden alfabético del ticker mientras haya suficiente capital disponible.No se puede operar con apalancamiento ni operaciones en corto.
- Nunca se podrá operar más de un 5 % del volumen diario para así no afectar a la concentración de mercado y poder deshacer fácilmente.
- Vamos a invertir como mucho un 80 % del capital disponible, con lo cual vamos a tener siempre un margen de liquidez.
- Para la asignación de capital vamos a utilizar el criterio de kelly, el cual nos indica según el porcentaje de acierto y las ganancias de las operaciones realizadas anteriormente el porcentaje a asignar en cada operación.

- Como se ha mencionado anteriormente el SL y TP se determinan por las desviaciones típicas del ratio. Entramos cuando la desviación cruce el -2, salimos con SL si tocamos -2.5 y salimos con TP si la desviación cruza el 0.
- En caso de que una empresa deje de cotizar ya sea por una OPA o una quiebra y tengamos posición en la misma siempre vamos a asumir que salimos con pérdidas. Siendo el precio de salida un 10 % del precio de entrada.

Por otro lado, para controlar el capital, se han declarado 3 variables:

- liquidez: indica el capital que se encuentra en efectivo.
- capital sin flotante: indica la liquidez más el dinero que tenemos invertido pero sin tener en cuenta el flotante, es decir, asumiendo que el precio de las acciones en cartera es el de entrada. Esta variable no tiene en cuenta los beneficios corrientes, sino que se actualiza solo cuando se cierra definitivamente una operación.
- capital con flotante: indica la liquidez más el importe de las operaciones abiertas por el último precio de cierre del activo, es decir, que tenemos en cuenta las ganancias que están obteniendo las operaciones activas.

El proceso de asignación de capital se puede ver en el diagrama de la imagen 48
Diariamente, tras el cierre de mercado se recalculan los ratios y se comrpueban todas las operaciones. En primer lugar, se comprueba si hay operaciones abiertas y se comprueba si tienen condición de venta, esto es que la desviación típica del ratio haya tocado uno de los límites de TP o SL. En caso de que haya tocado SL o TP, se coloca para que mañana se venda la operación con el precio de apertura.

Una vez comprobado si hay señales de venta, se comprueban las señales de compra. Para ello se cogen los ratios y se comprueba si hay alguno cuya desviación típica baje de -2 puntos. En caso positivo,se realiza una predicción con la red neuronal seleccionada en los capítulos anteriores. El resutlado será una probabilidad de que el las desviación típica ratio vaya a tocar antes el TP que el SL, si la probabilidad supera un trigger se considerará para compra en la apartura del día siguiente, sino, se descartará.

Una vez determinadas las compras y ventas, a la apertura del día siguiente, se realizan las ventas para adquirir liquidez e inmediatamente después se realiza la compra. Para el proceso de compra seguiremos varios pasos:

1. En primer lugar, miramos el capital líquido que es el porcentaje de liquidez con respecto al capital sin flotante.
2. Calculamos el capital utilizable, que correponde a la liquidez menos el capital minimo que ha de quedar libre. Siendo este último igual a uno menos el máximo capital a invertir (en tanto por ciento), en nuestro caso el máximo capital a invertir será de 80 %.
3. Calculamos el capital de kelly. El capital de kelly corresponde a la ecuación 7. Este capital nos da el capital óptimo a invertir según la precisión (porcentaje de acierto) de nuestras predicciones y el rendimiento medio de las operaciones cerradas, dividido todo ello por un factor (1 ó múltiplo de 2) que permite aumentar el número de operaciones. En nuestro caso, hemos determinado que necesitamos un número mínimo de 40 operaciones, por ello la precisión inicial es de 60 % y rendimiento inicial 12.2 %. Estos datos se han obtenido de analizar las operaciones sin IA.
4. Comprobamos que el capital de kelly sea mayor que el capital utilizable, y que sea mayor que cero. En caso positivo continuamos, en caso negativo el capital a invertir será cero. Para el backtest, una vez

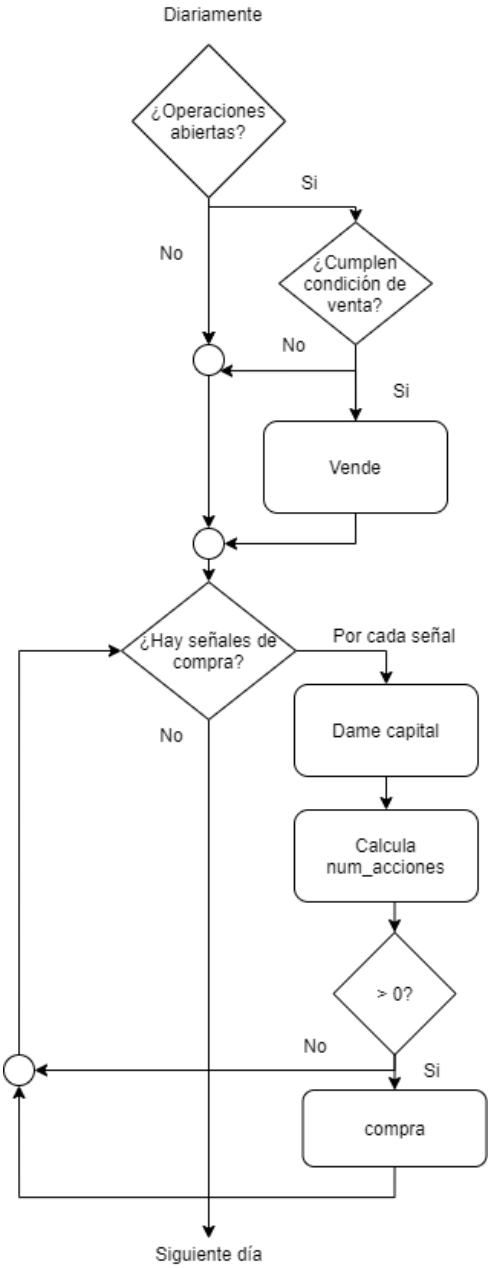


Figura 48: Diagrama algoritmo gestión de capital

obtenido est valor se comparará con el 5 % del volumen del día y se obtendrá el mínimo de ambos. El capital Total será el total de esta cantidad, y sólo se inviertirá si se dispone de capital para pagar todo, nunca fracciones.

5. El capital a invertir se distribuirá entre acciones y comisiones, es decir, en la compra será:

$$\text{num\_acciones} = \frac{\text{Capital Total}}{\text{precio} * (1 + \text{comisiones})}$$

6. Si el cálculo anterior es positivo, establecemos dos posibilidades para determinar el capital a invertir:
   - Agresiva: El capital a invertir será el capital de kelly multiplicado por el capital con flotante.
   - Menos agresiva: El capital a invertir será el capital de kelly multiplicado por el capital sin flotante.
7. Una vez obtenido el capital, obtendremos el número de acciones diviendo el capital por el precio y restando las comisiones y comprobamos que estas
8. Actualizamos la liquidez y capital sin flotante

El capital de kelly

$$\text{Capital Kelly} = \frac{\text{precisión} - \frac{1-\text{precisión}}{1+\text{rendimiento}}}{\text{divisor kelly}} \tag{7}$$

Por último comentar que si el último día tenemos posiciones abiertas se venden inmediatamente. Por otro lado, si un activo deja de cotizar consideramos que se vende a pérdidas siendo el precio de venta un 10 % del precio de compra.

8. Backtest y análisis de resultados

Una vez detalladas las partes del algoritmo que hemos desarrollado, hay que probarlo, para lo que se ha desarrollado un backtest. En primer lugar se ha realizado el backtest para el conjunto de validación para poder optimizar el algoritmo. Una vez optimizado, se ha ejecutado en el conjuto de test.

8.1. Backtest con datos de validación

La base del backtest es el diagrama de la imagen 48 que ya ha sido explicada en el capítulo anterior. Para simplificar su ejecución se han generado varios dataset que contienen los diferentes datos ordenados por fecha y empresa. Así ejecutaremos un código que sigue el diagrama de la imagen 48 para cada día con los siguientes conjuntos de variables:

- triguer: 0.5,0.6,0.7,0.8
- comisiones: 0.01, 0.001
- agresividad kelly: agresivo y menos agresivo

Tras ejecutarlo, se han analizado los resultados, obteniendo diferentes métricas, que junto con las tres variables de capital (liquidez, capital con flotante y capital sin flotante), mencionadas en el capítulo anterior, nos han ayudado a decidir el modelo seleccionado. Dichas variables se han calculado sobre el capital con flotante, que es el que nos da una foto más clara con los últimos valores de mercado de la cartera total. En cuanto a las variables, en la tabla 1 se muestra un resumen de los 10 mejores resultados ordenados (bajo nuestro criterio) de menos a más.

Name	dd_max	sharpe1	sharpe2	calmar	rdto_log	rdto_total	num_operaciones
m120_k2_kgFALSE_c0.001_t0.5	0.242369	0.467653	7.467820	1.796746	0.435475	0.545697	77
m120_k4_kgFALSE_c0.001_t0.7	0.213384	0.505054	8.065064	2.105041	0.449182	0.567029	108
m365_k2_kgFALSE_c0.001_t0.8	0.082959	1.178695	18.822255	5.590363	0.463773	0.590062	62
m365_k4_kgFALSE_c0.01_t0.7	0.121200	0.733888	11.719251	3.859289	0.467747	0.596394	100
m365_k2_kgTRUE_c0.001_t0.8	0.082020	1.188521	18.979164	5.706613	0.468054	0.596884	62
m120_k2_kgTRUE_c0.001_t0.7	0.203422	0.569286	9.090768	2.407750	0.489789	0.631971	57
m120_k1_kgTRUE_c0.001_t0.7	0.272503	0.473735	7.564944	1.811619	0.493671	0.638320	53
m120_k1_kgFALSE_c0.001_t0.7	0.272491	0.474606	7.578857	1.813907	0.494274	0.639307	53
m120_k2_kgFALSE_c0.001_t0.7	0.200841	0.572749	9.146073	2.462870	0.494644	0.639915	57
m365_k4_kgFALSE_c0.001_t0.7	0.109802	0.867014	13.845108	5.214140	0.572525	0.772738	99

Tabla 1: Mejores 10 resultados backtest set validación

Las variables que se han utilizado son las siguientes:

- dd_max: Corresponde al drawdown máximo encontrado en el periodo seleccionado.
- sharpe1: Corresponde al sharpe calculado como el rendimiento anualizado partido de la volatilidad diaria anualizada. En este caso se ha considerado que no hay activo libre de riesgo, es decir que está 0. Para anualizar los resultados se ha considerado que un año cuenta con 255 días laborables.
- sharpe2: Corresponde al sharpe calculado como el rendimiento anualizado partido de la volatilidad diaria. En este caso se ha considerado que no hay activo libre de riesgo, es decir que está 0.Para anualizar los resultados se ha considerado que un año cuenta con 255 días laborables.

- calmar: Corresponde al ratio de calmar calculado como el retorno total dividido por el máximo drawdown.
- rdto_log: Corresponde con el rendimiento contínuo total, es decir la suma de los retornos logarítmicos.
- rdto_total: Corresponde con la ganancia total del algoritmo, se puede calcular como la exponencial del rdto_log, o como el último valor dividido por el valor inicial menos uno.
- num_operaciones: Corresponde con el número total de operaciones realizadas por el algoritmo.

Por otro lado, hemos analizado también diferentes gráficas que nos han ayudado en la decisión. Todas las gráficas que se van a presentar están expresadas en puntos porcencuales sobre el valor inicial de cada una de las series mostradas. También comentar que están ordenadas de peor a mejor, bajo nuestro criterio, pero siempre mostrando las mejores de todas las pruebas. La figura 50 muestra el capital sin flotante. Estas dos primeras figuras muestran que por un lado se va invirtiendo el capital que tenemos disponible, y por otro lado que el capital va subiendo poco a poco con las diferentes inversiones que vamos cerrando. En la figura 51 se puede observar que las inversiones siguen en general un patrón ascendente con bajadas y subidas generalizadas en los mismos puntos. También se observa que las gráficas de "m365", que corresponden a los backtest cuyas ventanas son de 365 días empiezan más tarde, esto es debido a que las ventanas que se utilizan en cada uno de los algoritmos necesitan un número diferente de días, y para no comprometer la integridad de los datos y obviar casos de lookahead bias, no se ha permitido que haya datos de test en validación, ni datos de validación en train, es por ello que empieza más tarde.

En la gráfica49 se puede observar los ratios de liquidez entre capital sin flotante. Estos ratios nos van a indicar cómo de invertido está el algoritmo en cada momento. Es decir, indican cuanta liquidez hay invertida por capital invertido en cada momento; a nosotros nos interesa que este valor sea lo más bajo posible. Por otro lado, nos sirve para comprobar que estemos cumpliendo la condición establecida de que no queremos estar invertidos más de un 80 % del capital. Como se puede observar, en general, interesa que haya muy poca liquidez, es decir que estemos invertidos el mayor tiempo posible. En algunos casos se observa que hay días puntuales que no hay ninguna operación abierta, pero que esta situación dura poco. También se puede observar que el algoritmo que mejor funciona es el que más tiempo está invertido.

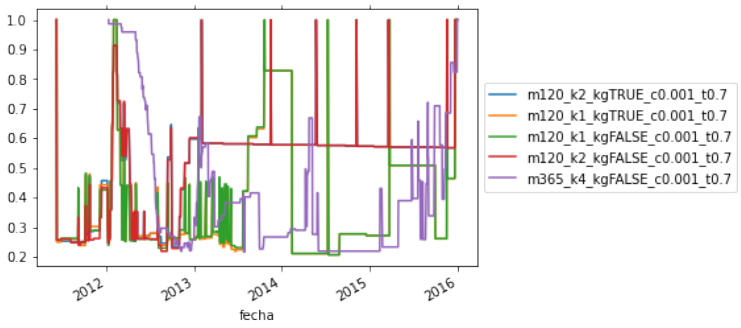


Figura 49: Liquidez entre capital sin flotante para TOP 5

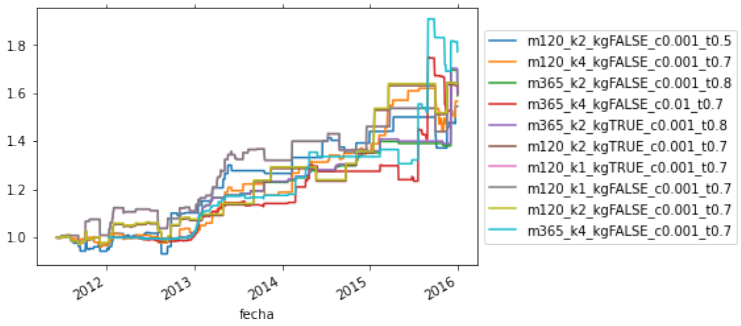


Figura 50: Capital sin flotante

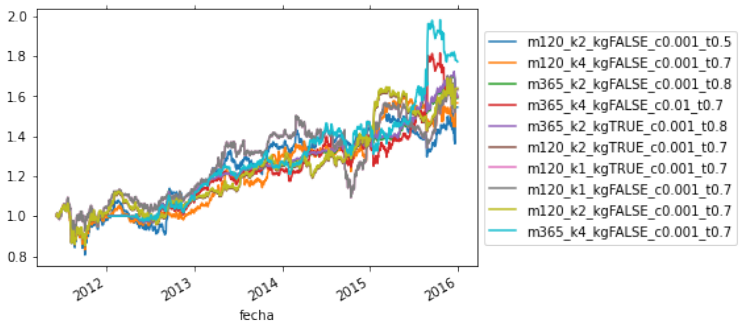


Figura 51: Capital sin flotante

Tras analizar los resultados de las gráficas anteriores y la tabla, se puede observar que el modelo que mejor rentabilidad ofrece es el denominado "m365_k4_kgFALSE_c0.001_t0.7'. No se ha optado por el segundo modelo,"m120_k2_kgFALSE_c0.001_t0.7", ya que ofrece menor rentabilidad y además vemos que entre los 10 mejores modelos hay 4 de 365 días y 6 de 120 días. Las características de este algorimto son:

- ventana de 365 días
- divisor de kelly 4
- kelly no agresivo
- comisiones de 0.1 % por operación
- trigger para discriminar probabilidades de 0.7
- capital a invertir $1,000,000

8.2. Resultados y análisis con los datos de Test

8.2.1. Análisis resultados algoritmo con IA en Test

Ya hemos establecido que el mejor modelo que hemos obtenido en validación ha sido con la ventana de 365 días y LAG 30. Ahora debemos reentrenar el modelo con los datos de train y de validación, y así podremos hacer las predicciones sobre el conjunto de test. La arquitectura y el early stopping es la misma, pues como ya hemos dicho, no podemos modificar ningún parámetro tras ver los resultados en test. Una vez reentrenado el modelo, lo guardamos para realizar las predicciones sobre test. A continuación vamos a analizar esos resultados sobre test como ya hicimos anteriomente con el conjunto de validación. Los resultados deberían ser lo más parecidos posibles a los que ya obtuvimos sobre validación. El accuracy que ha obtenido este último modelo ha sido del 68.1 %. Vemos que es ligeramente superior al obtenido en validación, pero se debe a que en Test tenemos una mayor proporción de datos con etiqueta TP.

Lo primero que vamos a ver en la figura 52 es la distribución de las probabilidades que devuelve la última capa de la red. Como se puede observar, la distribución es idéntica a la obtenida en validación.

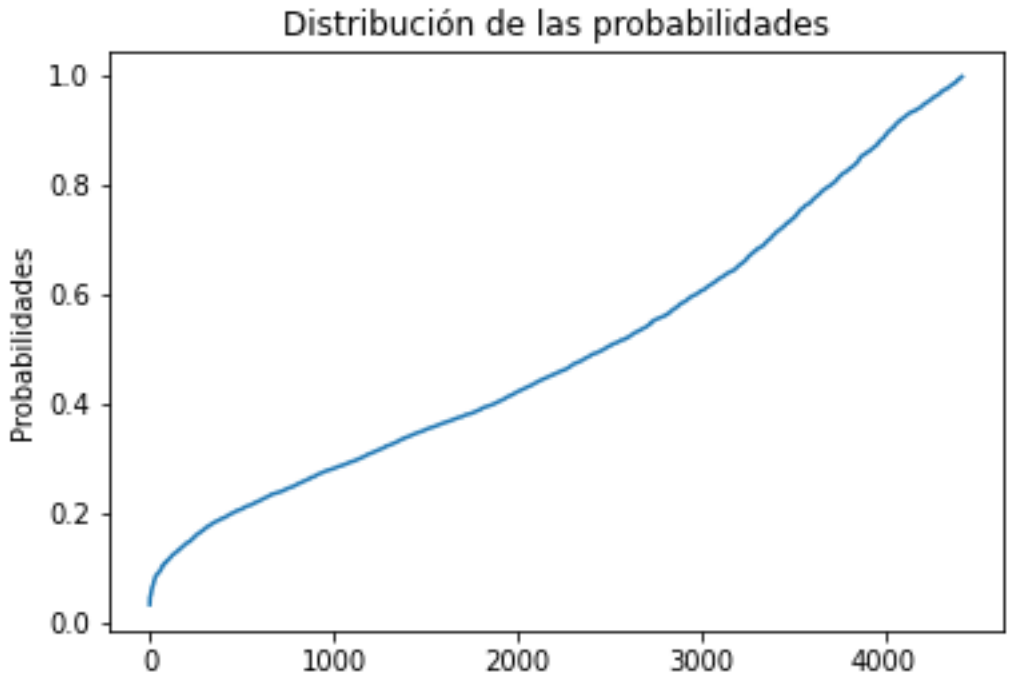


Figura 52: Distribución de probabilidades

Ahora veamos la matriz de confusión en la figura 53 y sus métrica en la figura 54.

Los resultados son algo mejores que los que obtuvimos en validación, por el mismo motivo que antes. Ahora pasamos a variar los umbrales de selección entre 0.5 y 0.99 para ver como varía la precisión y la cantidad de datos etiquetados como TP en función del umbral de predicción.

real	0.0	1.0
pred		
0.0	1606	869
1.0	542	1406

Figura 53: Matriz de confusión

	precision	recall	f1-score	support
0.0	0.65	0.75	0.69	2148
1.0	0.72	0.62	0.67	2275
accuracy			0.68	4423
macro avg	0.69	0.68	0.68	4423
weighted avg	0.69	0.68	0.68	4423

Figura 54: Métricas matriz de confusión

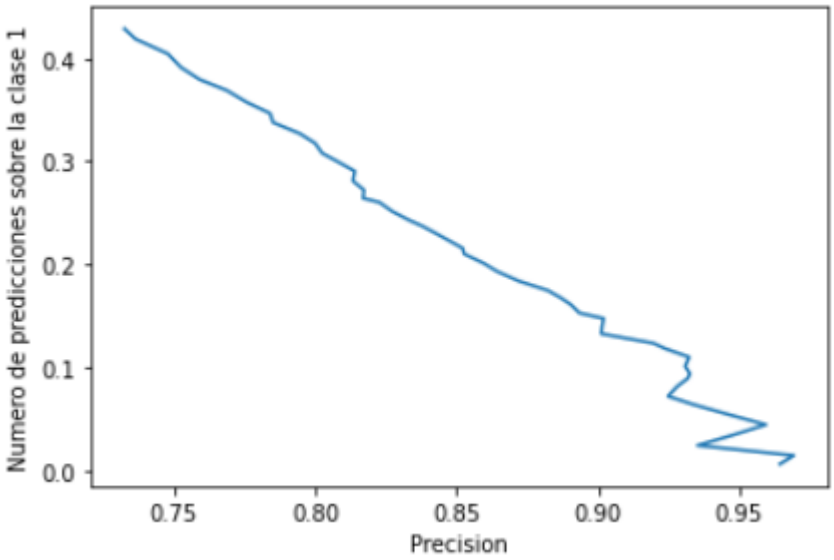


Figura 55: Variación de la precisión en función del número de datos etiquetados como TP

En la figura 55 podemos apreciar que la curva es parecida a la de valdiación, pero en los valores altos de lus umbrales los picos han desaparecido. Como bien dijimos en su momento, en validación eso ocurría por la fatla de datos, pero como en Test disponeos de más datos que en validación, esta curva se ha estabilizado bastante..

Ahora pasamos a estudiar la curva ROC y su AUC.

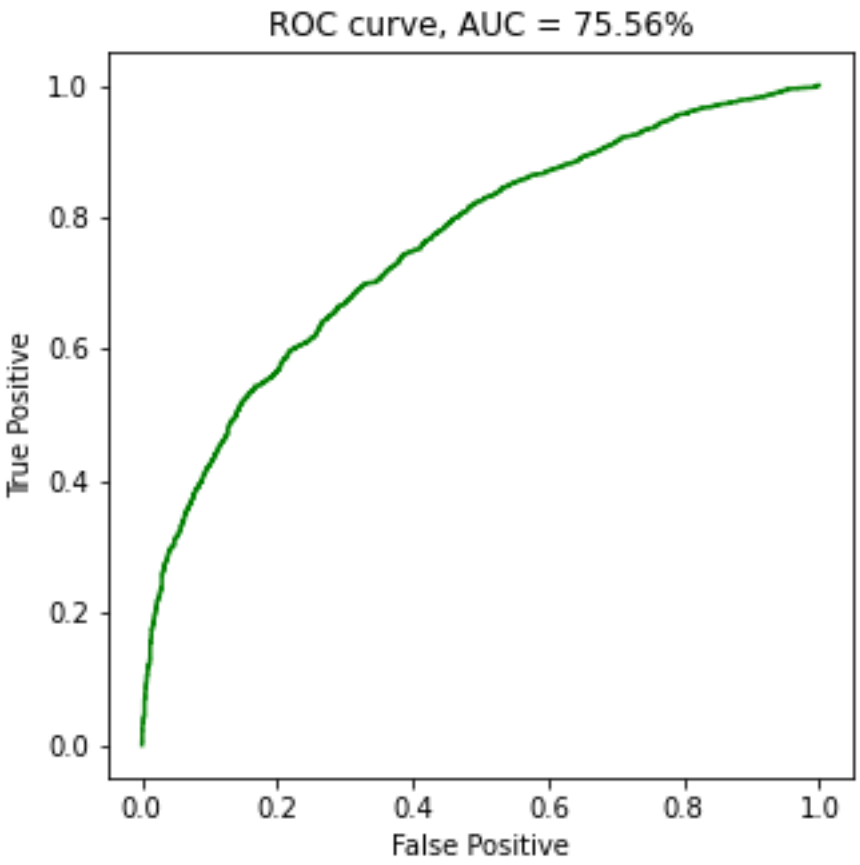


Figura 56: Curva ROC y AUC

La curva ROC y su AUC son varios puntos ejores que la de validación; ha quedado bastante claro que todas las metricas han mejorado respecto al conjunto de validación debido al aumento de la proporción de casos de TP. Aún así, vemos que los resultados son muy parecidos y robustos, lo que significa que nuestro modelo ha sido capaz de generalizar correctamente al conjunto de Test. Veamos ahora la improtancia de las variables con el Gini.

Ha quedado bastante claro que las variables de los percentiles son innecesarias, pues en todos los modelos, y tanto en validación como en test, han sido las que tienen un menor Gini. Respecto a las variables más relevantes, las diferencias entre las emas y e*Ratio* siguen siendo las más relevantes. Hemos visto como los

	gini
diffemas_180_80	0.165583
ratio	0.137043
desv_365	0.122698
diffemas_365_180	0.118119
diff_365	0.116222
diff_240	0.107315
diff_120	0.084161
desv_180	0.083313
perc_365	0.035098
perc_180	0.030448

Figura 57: Coeficientes Gini de las varibles

resultados en test van acordes a los resultados obtenidos en validación, lo que implica una buena generalización. Podemos concluir con que el modelo de Inteligencia Artificial ha sido un éxito en Test, no solo por obtener buenos resultados, sino por haber generalizado perfectamente y no haber sobreajustado al conjunto de entrenamiento.

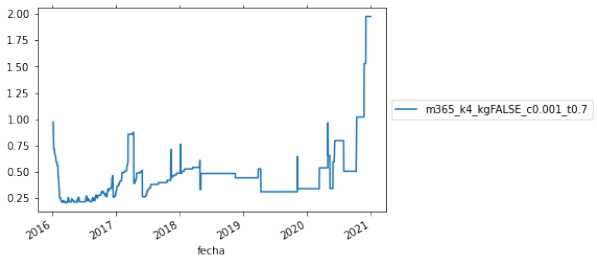
8.2.2. Backtest con datos de Test

Tras relanzar el backtest con datos de test se han obtenido los resultados de la tabla 2. Aqui se puede observar que el algoritmo no ofrece el rendimiento esperado, sino que pasa a dar una rentabilidad de un 30 % menos a lo largo de 5 años. Pero se comprueba que con una elección diferente de capital, el algoritmo tiene un performance mejor que el benchmark. Esto puede ser debido a que al disponer de menos capital, ha tenido menos opciones de fallar, pero por otro lado, ha tenido menos posibilidades de diversificación, por ello, consideramos que ha sido una simple casualidad. Por ello, como fuente de mejora se puede considerar mejorar el criterio de selección de las operaciones.

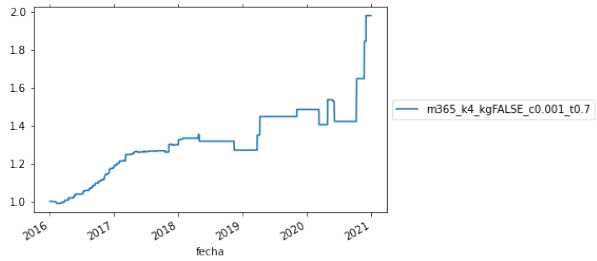
Name	dd_max	sharpe1	sharpe2	calmar	rdto_log	rdto_total	num_operaciones
finalbt_365_10M	0.403284	0.293796	4.691543	0.773108	0.311782	0.365857	111
finalbt_365_100k	0.435153	0.338283	5.401953	0.903428	0.393129	0.481610	115
finalbt_365_1M	0.403193	0.370928	5.923242	1.055115	0.425415	0.530226	111
sp500	0.339250	0.519122	8.289708	1.793647	0.608494	0.837662	0

Tabla 2: Resultados backtest datos de Test

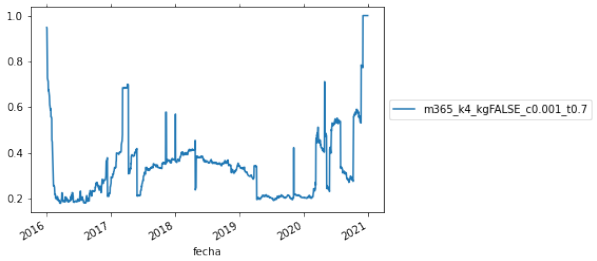
En las figuras 58 se pueden observar las mismas gráficas anteriormente descritas para el conjunto de validación pero con el algoritmo seleccionado y el conjunto de Test.



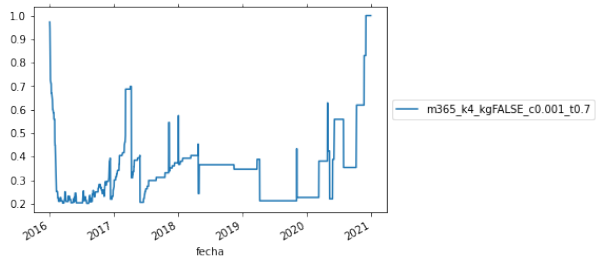
(a) Liquidez



(b) Capital sin flotante



(c) Liquidez entre capital con flotante0



(d) Liquidez entre capital con flotante

Figura 58: Salidas Backtesting set de Test

Aqui se puede destacar que salvo momentos puntuales siempre está invertido más del 40 % del capital. Por otro lado, se aprecia que a finales de 2021 sube mucho la liquidez. Esto es debido a que a principios de 2020 se desarrolló la crisis del *COVID-19*, que aunque provocó un imacto negativo muy grande en los mercados, también hubo unas fuertes subidas. Este impacto se puede apreciar claramente en la figura 59. Por otro lado, se puede observar que hay una altra correlación entre ambos, de hecho, la correlación es de 0.773220. Por otro lado, creemos que el algoritmo se mueve más debido a que tenemos muchas menos empresas invertidas a la vez, de hecho el total son 111 operaciones a lo largo de 5 años; sin embargo el índice contiene 500 empresas, por lo que está mucho más diversificado y es lógico que presente menos movimietos bruscos.

En la tabla 3 se puede observar una salida del registro de operaciones. En el que se puede observar la rentabilidad de cada una de ellas, el status (1 comprado, 2 vendido cuando ha tocado TP, 3 vendido cuando ha tocado SL y 4 cuando ha dejado de cotizar una empresa). También hay que destacar que se puede observar el capital de kelly que se ha asignado a cada una de las operaciones.

Por último, al analizar el backtest también generamos el registro de operaciones en formato “strategy” ya que era un requisito del trabajo. Se ha seleccionado este formato ya que para nosotros no es importante el precio de entrada sino el ratio, y por otro lado, miramos la fecha de entrada y salida. El precio sólo es importante para el cálculo de las acciones con las que vamos a entrar ya que vamos a incluir siempre un porcentaje por operacion. Un ejemplo de salida es el que se puede observar en la tabla 4.

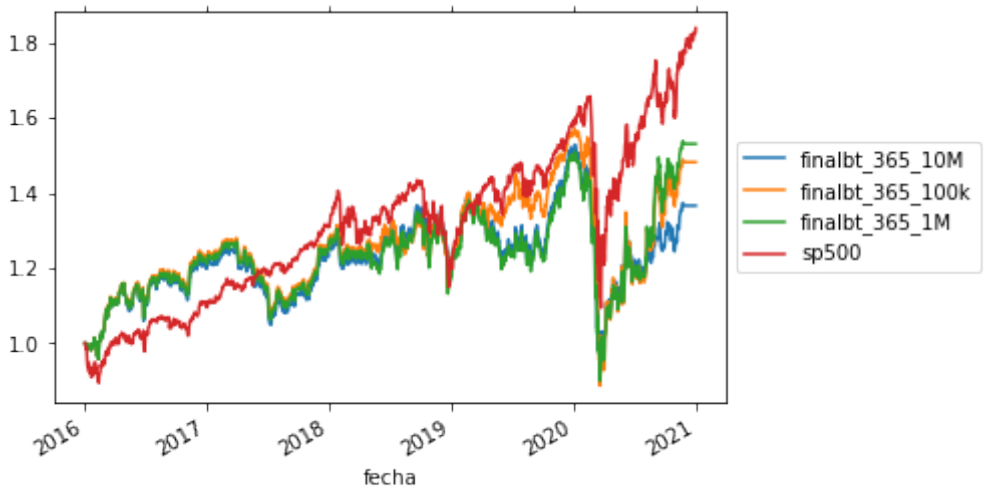


Figura 59: Capital del algoritmo vs benchmark

date_in	date_out	Name	price_in	price_out	num_stocks	ratio_in	ratio_out	status	performance	kelly_capital
2016-01-05	2016-12-08	APD	113.928573	134.225064	120.0	-2.043131	0.308925	2	0.175797	0.013786
2016-01-05	2016-12-30	RHI	41.547633	45.201009	331.0	-2.269885	0.011485	2	0.085759	0.013786
2016-01-06	2016-07-22	FFIV	95.420000	121.460000	144.0	-2.019890	0.060884	2	0.270356	0.013786
2016-01-06	2016-09-06	IP	30.305573	41.166285	454.0	-2.026550	0.004043	2	0.355659	0.013786
2016-01-07	2016-03-28	CXO	80.191932	101.672439	171.0	-2.135155	0.069177	2	0.265330	0.013786
2016-01-07	2016-09-06	LRCX	66.185274	86.838399	208.0	-2.024840	0.251999	2	0.309429	0.013786
2016-01-07	2016-08-24	LUMN	15.163364	18.808184	908.0	-2.010445	0.047606	2	0.237892	0.013786
2016-01-07	2016-03-29	NFX	29.170000	31.460000	472.0	-2.013983	0.106101	2	0.076350	0.013786
2016-01-07	2016-09-08	SLM	5.827577	7.289344	2363.0	-2.069582	0.057916	2	0.248337	0.013786
2016-01-07	2016-12-08	WYN	64.798144	74.673424	212.0	-2.101473	0.136455	2	0.150098	0.013786

Tabla 3: Registro de operaciones del algoritmo

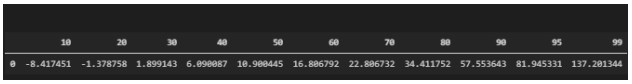
Name	exp_in	exp_out	n_shares	p_in	p_loss	p_out
APD	2016-01-05	2016-12-08	120.0	NaT	NaT	NaT
RHI	2016-01-05	2016-12-30	331.0	NaT	NaT	NaT
FFIV	2016-01-06	2016-07-22	144.0	NaT	NaT	NaT
IP	2016-01-06	2016-09-06	454.0	NaT	NaT	NaT
CXO	2016-01-07	2016-03-28	171.0	NaT	NaT	NaT
LRCX	2016-01-07	2016-09-06	208.0	NaT	NaT	NaT
LUMN	2016-01-07	2016-08-24	908.0	NaT	NaT	NaT
NFX	2016-01-07	2016-03-29	472.0	NaT	NaT	NaT
SLM	2016-01-07	2016-09-08	2363.0	NaT	NaT	NaT
WYN	2016-01-07	2016-12-08	212.0	NaT	NaT	NaT

Tabla 4: Salida en formato “strategy”

8.3. Monos Aleatorios

Ya hemos visto cuales son los resultados de nuestro algoritmo, y los hemos comparado con nuestro benchmark, pero ,además , podemos compararlo con resultados obtenidos de manera aleatoria. A esto se le conoce como "monos aleatorios", donde lo que hacemos es realizar muchas simulaciones donde invertimos de manera pseudoaleatoria, aunque normalmente se suele utilizar algún algoritmo básico; en nuestro caso usaremos itMarkowitz. Vamos a realizar 1.000 simulaciones distintas. Nos cogemos de manera aleatoria la fecha de compra y de venta dentro de nuestro conjunto de test con una distancia mínima de dos días. A continuación, escogemos 5 activos que hayan cotizado durante los últimos 500 días, y utilizaremos esta ventana para calcular Markowitz. Construimos la cartera eficiente, nos sacamos la mejor de todas las carteras y usamos los pesos de los activos para invertir. Compraremos esas 5 acciones en la fecha de compra invirtiendo el porcentaje de capital que nos diga su peso, y los venderemos en la fecha de venta. Si la empresa deja de cotizar antes de la fecha de venta, vendemos en ese último día. De esta forma, cada simulación nos dará una rentabilidad distinta obtenida de manera pseudoaleatoria. Consideramos que nuestro algoritmo ha obtenido buenos resultados si se encuentra por encima del percentil 90 de las rentabilidades de los monos, pero si se encuentra por encima del 99 podemos pensar que los resultados son ''demasiado buenos" y podemos haber incurrido en algún error en el backtest, en la asignación de capital o en el algoritmo de inteligencia artificial. Veamos ahora los percentiles de las rentabilidades de los monos en la figura 60

Nuestro modelo obtenía una rentabilidad de un 53 %, valor que se encuentra en el percentil 88.8 respecto a las rentabilidades de los monos. Es un resultado bastante positivo, pero sin llegar a ser demasiado alto como para inducir a que hay algún error en el algoritmo.



	10	20	30	40	50	60	70	80	90	95	99
0	-8.417451	-1.378758	1.899143	6.090087	10.900445	16.886792	22.886732	34.411752	57.553643	81.945331	137.201344

Figura 60: Rentabilidades monos aleatorios

9. Implementación en Cloud, mejoras y trabajos futuros

9.1. Implementación en Cloud

9.1.1. AWS

Vamos a explicar una posible manera de poner en producción nuestro algoritmo. No hemos llegado a ponerlo físicamente en producción ni nos hemos creado los recursos, pero vamos a explicar paso a paso lo que habría que hacer para poner el modelo en producción de manera segura, eficiente y, sobre todo, económica. Para poner en producción nuestro algoritmo con AWS necesitaremos en principio 4 servicios:

- *Amazon SageMaker*: lo utilizaremos para entrenar el modelo y desplegarlo en la nube para su posterior utilización.
- *S3*: almacenamiento de la data y del estado de las variables del algoritmo.
- *Cloud9*: para programar la descarga de data de AlphaVantage, la lectura y escritura en S3, la predicción del modelo de IA, la utilización del algoritmo de gestión del capital y el lanzamiento de las órdenes a mercado.
- *Lambda*: utilizaremos las funciones lambda para ejecutar el script de Cloud9 los días y horas debidos.

Vamos a partir de que tenemos la data de entrenamiento y la arquitectura del modelo guardadas en un S3. El primer paso es crear un usuario de "SageMaker Studio" y abrir el "Studio". Seleccionamos, por ejemplo, una máquina de características " 2 vCPU + 4 GiB" y luego seleccionamos Python3 (TensorFlow2 GPU Optimized). Después de haberlo configurado, cargamos la data de S3 y entrenamos nuestro modelo con la arquitectura que ya tenemos definida con todos los datos de que disponemos. Al terminar, lo desplegamos usando la función "deploy", y vemos como nuestro modelo aparece en la sección "Modelos". Para llamar al "EndPoint" vamos a usar una función lambda.
Nos creamos un nuevo entorno en "Cloud9", que es donde vamos a programar el resto del algoritmo. Lo primero que queremos es bajarnos los últimos datos guardados de nuestras variables de S3, pues los necesitaremos en formato de 3 dimensiones (de acuerdo a nuestro valor de LAG 30) para que el modelo de IA nos pueda dar una predicción. También nos bajamos las variables que necesitamos del algoritmo, como la lista de operaciones activas, el porcentaje de acierto y de ganancia actual para el criterio de Kelly, y demás. Entonces nos bajamos los últimos datos de precios de AlphaVantage con nuestro algoritmo de descarga de datos, con los que podemos crearnos los valores de las variables para el día de hoy. Una vez que tenemos esto, llamamos al endpoint para cargarnos el modelo, y realizamos las predicciones para el día de hoy sobre aquellas empresas que hayan cortado la segunda desviación típica inferior. Con estos datos podemos pasar directamente al algoritmo de gestión de capital: con las variables y el valor del ratio de hoy, nuestro algoritmo decidirá que operaciones realizar y qué variables modificar. Las operaciones las mandaríamos al bróker correspondiente mediante su API, y los nuevos datos, operaciones y otros elementos nuevos generados los subiríamos a S3 para poder usarlos el día siguiente.
Este script de Cloud9 se ejecutará los días y horas que consideremos mediante una función lambda. De esta manera, hemos puesto en producción nuestro algoritmo minimizando los costes, pues lo único que se debería pagar sería el endpoint del modelo de SageMaker y la ejecución de la función lambda cada día.

9.1.2. Google Cloud

Para la gestión desde google cloud se propone una solución, utilizando diferentes servicios. En este caso se propone partir del modelo ya entrenado y de los datos históricos. Para capturar los datos de cierre se propone descargarlos de AlphaVantage y los de apertura para determinar las acciones a comprar del bróker. Otra opción sería obtener ambos del bróker. Los servicios requeridos serán:

- *Google Cloud Storage*: Lo utilizaremos para guardar el modelo, los diferentes datos que vayamos a descargar diariamente e historificarlos.
- *Bigquery*: Este servicio nos permitirá guardar los datos descargados en tablas SQL que facilitarán su manejo. Una vez descargados los datos se guardará una copia en SQL.
- *Cloud Function*: Aquí crearemos los diferentes servicios que nos permitirán lanzar los procesos de Python necesarios. Esta aplicación permite crear funciones que se llamen con distintos trigger.
- *Cloud Scheduler*: Permite programar el lanzamiento de las tareas del cloud function.
- *Data Studio*:

En primer lugar mencionar que el hecho de guardar los datos en los dos sistemas de almacenamiento (storage y bigquery) es por un tema de recuerrencia, en caso de que falle uno de los sistemas se podría utilizar el otro.

Por otro lado, la arquitectura propuesta es para que se lancen todos los procesos desde Cloud Functions. Aunque se puede hacer utilizando *App Engine* y montando dos servicios, uno de captura de precios de cierre y calculo de ratios; y otro de captura de precios de apertura y operativa de acciones. Volviendo al primer caso, lo que se propone es partir del modelo entrenado (guardado en cloud storage) y los datos historificados (cloud storage y/o bigquery) y generar dos lanzamientos con el cloud scheduler. El primer lanzamiento será a cierre de mercado y capturará los precios de cierre y realizará los cálculos de los ratios. Tras ello, generará una salida en la que indicará las posibles acciones a comprar. Al día siguiente, en la apertura del mercado, el segundo trigger saltará lanzando el proceso de captura de precios de apertura y el algoritmo de gestión de capital. En el que se calculará si es posible realizar las entradas marcadas el día anterior (dependiendo del capital requerido y el capital disponible). Tras ello se realizará una salida con las nuevas posiciones. Como en el caso anterior, todas las salidas se replicarán en bigquery y en cloud storage y serán enviadas por correo electrónico junto con un log para comprobar que el proceso ha corrido correctamente. Por último en data studio crearemos un tablero de mando en el que se puedan ver los logs y todos los parámetros del estado de la cartera, liquidez, capital con y sin flotante, registro de operaciones...

9.2. Mejoras y trabajos futuros

Tras ir realizando el trabajo, se nos han ido ocurriendo muchas ideas que creemos que podrían venir muy bien para poder mejorar el algoritmo. Estas ideas se podrán desarrollar en el futuro, pero requieren de mucho tiempo tanto de realización de pruebas, de programación, y de capacidad computacional para llevarlas a cabo, por eso las más complejas no las hemos llegado a incluir. Entre ellas podemos destacar las siguientes:

- Modificar el criterio de seleccion de activos seleccionando aquellos que presenten una descorrelación más elevada con los ya presentes
- Modificar el criterio de seleccion de activos seleccionando aquellos que presenten una probabilidad de tocar antes el TP que el SL mayor.
- Establecer un TP y SL no solo a valores de desviación típica del ratio, sino también a precios, es decir, contar con ambas posibilidades.
- Tener simultaneamente varios algoritmos con distintas ventanas y arquitecturas dando predicciones, de manera que tengamos muchas más posibles operaciones que realizar y poder tener filtros más restrictivos con el fin de aumentar la casa de acierto.
- El punto de entrada (la segunda desviación típica) y los puntos de SL y TP (-2.5 y 0 respectivamente) se han colocado de forma estática de manera que dividan los datos de manera eficiente, pero podríamos crear un algoritmo que vaya moviendo el punto de entrada y los TP y SL de acuerdo a la volatilidad del activo. Esto se podría hacer con modelos ARCH/GARCH o con redes neuronales.

A. Apendice1: Explicación código

A continuación vamos a dar unas señas para poder ejecutar todo el código de nuestro trabajo de manera adecuada. Tenemos archivos .py y archivos .ipynb, dependiendo de cómo queremos ir mostrando los resultados. El código y los datos están disponibles en el siguiente link de drive: `https://drive.google.com/drive/folders/1_SJbBEJNy7jKuys0YkEQ0cCi7h_6sX_A?usp=sharing`

Lo primero es la descarga de AlphaVantage. Se compone de 4 archivos: alpha_vantage.py es la librería, config_alpha.py proporciona las claves y gcloud.py es la librería de Google Cloud. El único archivo que hay que ejecutar para realizar la descarga es main.py (aunque no es necesario hacerlo, pues ya tenemos la data descargada en Google Cloud y podemos tirar directamente desde ahí). Esto se refiere al Capítulo 2 del TFM.

Una vez tenemos los datos guardados en el storage de Google Cloud, vamos a realizar el preprocesado . El preprocesado se hace en el archivo 01, que se corresponde con el Capítulo 3 del TFM. Ese archivo nos devuelve la tabla ''dataml.csv". A partir de esa tabla, los dos archivos 02 y 03 nos crean las variables para las ventanas de 120 y 365 días, dando lugar a las tablas ''tabla_reversión_m120.pkl" y ''tabla_reversión_m365.pkl"; se corresponde a los 3 primeros puntos del Capítulo 4. Estas tablas serán las que usemos para crear los conjuntos de train, validación y test.

Los siguientes archivos van a ser Notebooks, pues como vamos a ir tratando con la data queremos ir viendo las salidas de la misma. El siguiente archivo es el 04, en el que realizamos el EDA para nuestras variables (solamente para la ventana de 120); se corresponde al Capítulo 4.4. Los siguientes archivos a ejecutar son el 05 y el 06, donde creamos el algoritmo sin IA, hacemos el backtest teórico sin gestión de capital, y exploramos los resultados obtenidos. Se corresponde al capítulo 6.2.

A continuación pasamos a la parte de Inteligencia Artificial. El siguiente archivo que debemos ejecutar es el 07, donde creamos los conjuntos de train, validación y test para la ventana de 120. Este archivo nos crea los tres archivos ''train_reversión_m120_lag30.pkl", ''val_reversión_m120_lag30.pkl", y ''test_reversión_m120_lag30.pkl". Solamente hemos incluido los archivos para LAG 30, no para 20,60, 90 y 150, pero si se quisiesen generar, solo habría que cambiar el valor del LAG en el archivo 07. Estos tres archivos de datos generados los usaremos para el entrenamiento y predicción de nuestras redes. Los siguientes archivos a ejecutar son los 08, 09, 10 y 11, que se corresponden a los distintos modelos de Inteligencia Artificial que hemos descrito antes, el modelo con Conv1D, el modelo con GRUs, el modelo con Conv2D y los dos modelos funcionales. Estos archivos no crean ninguna tabla ni archivos de otro tipo, solo sirven para ver los resultados de las redes. Estos cinco archivos se corresponden al Capítulo 6.3.1.

El siguiente archivo es el 12, que, al igual que el 7, crea los conjuntos de train, validación y test para la ventana de 365. El archivo 13 ejecuta el modelo de IA con capas Conv1d para la ventana de 365. Se corresponde al Capítulo 6.3.2.

A continuación, los archivos 14 y 15 crean, usando early stopping, los dos modelos definitivos de ventana 120 y 365 que utilizaremos para hacer los backtest en el conjunto de validación. Los archivos 16 y 17 analizan los dos mejores modelos definidos anteriormente, el de ventana 120 y el de ventana 365. Se corresponde al Capítulo 6.4.

El archivo 20 es el que ejecuta el backtest, pero para simplificarlo hemos dejado directamente preparado el backtest en test. Así que antes de eso, tenemos el archivo 18 donde nos creamos el modelo con ventana 365 que utilizaremos para realizar las predicciones en el conjunto de test, y el archivo 19, que es donde analizamos nuevamente el modelo de IA respecto a su matriz de confusión, métricas, etc.

Ahora pasamos a ejecutar el backtest con el algoritmo de gestión de capital. Vamos a explicarlo utilizando el mejor modelo obtenido, que es con el que realizaremos el backtest final en el conjunto de test. El archivo 20 genera el listado de operaciones y otro que contiene las series históricas de la liquidez, el capital con flotante y capital sin flotante durante el desarrollo del backtest. Para ello, debemos pasarle los datos requeridos, que son: ''tabla_reversion_m365.pkl" que contiene toda la data histórica con las ohlc, luego ''test_reversión_m365_lag30.pkl" que contiene los datos organizados para introducirsela al modelo y que devuelva las predicciones, y por último necesitamos ''test_data_m365_lag30.pkl" que contiene la data en dos dimensiones correspondiente a las fechas de las operaciones. Finalmente llamamos al modelo ''Test_m365_lag30", que es donde tenemos guardado el modelo ya entrenado y listo para realizar las predicciones en test.

Una vez acabados los backtest, podemos ejecutar el archivo 21 que se corresponde a los monos aleatorios con Markowitz.