REBALANCEADOR DE ESTRATEGIAS

Memoria del Trabajo Fin de Master

28 DE FEBRERO DE 2023

Javier Castro

Bruno Ezcaray

Índice

1.Resumen ejecutivo:

Descripción de los objetivos del proyecto:

Realizar un algoritmo de inversión basado en inversión activa que ajusta las asignaciones de los activos en una cartera de inversión para mantener el nivel deseado de exposición a cada estrategia. En el caso particular de este algoritmo los activos serán estrategias de inversión que a su vez invertirán en los diferentes ETFs disponibles mensualmente, en otras palabras, se realizará un rebalanceador de estrategias basado en redes neuronales.

El rebalanceador de estrategias de inversión tiene como objetivo Estos objetivos más específicos:

-Optimización de rendimiento: El rebalanceador busca maximizar el rendimiento de la cartera ajustando las asignaciones de activos en función de los indicadores de las diferentes estrategias de inversión.

-Reducción del riesgo: El rebalanceador busca reducir el riesgo de la cartera ajustando las asignaciones de activos en función de la volatilidad de las diferentes estrategias de inversión.

-Automatización de la toma de decisiones: El rebalanceador permite una toma de decisiones automatizada, en la que las decisiones se basan en los resultados del aprendizaje supervisado y se ejecutan automáticamente.

Métodos utilizados:

A partir de los ETFs seleccionados, se crean cuatro estrategias a las cuales se les calcula la rentabilidad y la volatilidad diaria, para luego pasarlos a datos mensuales, debido a la característica de nuestro modelo, estos indicadores sumados a datos de inflación (IPC) americana y europea junto con la rentabilidad y volatilidad de una estrategia cash, componen los inputs del modelo.

El rebalanceador basado en aprendizaje supervisado cuenta con doce características de entrada y un output categórico de cinco clases, una clase por cada estrategia.

Por otro lado, se define un modelo con tres capas densas cada una con su función de activación correspondiente y técnicas de regularización para reducir el sobreajuste de los datos.

Los datos de entrada se separan en train, test y validación. Se estandarizan los inputs tanto para train y test, debidamente con la media y desviación estándar del train. Para analizar el modelo se mide el error, el accuracy y se grafica la matriz de confusión.

Para analizar las predicciones del modelo se realiza un backtesting analizando distintos indicadores tales como rentabilidad acumulada, comisiones, sharpe, sortino y máximo drawdown.

Resultados obtenidos:

Los resultados obtenidos del modelo en un periodo de 10 años son:

Rentabilidad acumulada de 71%, un ratio de sharpe del -0.086, ratio de sortino de -1,05 y un máximo drawdown de un 22%.

Analizando todos los resultados obtenidos, tanto los resultados de entrenamiento y prueba, como los del backtesting, determinamos que nuestro modelo no obtiene mejoras significativas, respecto a los otros modelos más simples con los que nos comparamos. Es probable que esto podría deberse a la simpleza de las estrategias, a la pequeña cuantía de datos y a la variedad de las características en los inputs de los modelos.

2.Introducción:

Ideas:

En un principio, cuando tuvimos la primera sesión del TFM allá por marzo de 2022 estábamos llenos de dudas sobre la elección del tema, pero a pesar del desconocimiento a nivel general optamos por la opción de una cartera de inversión con diferentes activos (clásicos y criptomonedas).

Queríamos realizar una estrategia diaria y no contábamos con suficiente histórico de altcoins y no queríamos basarnos únicamente en Bitcoin o Ethereum. La idea no era adecuada para este trabajo porque no sabíamos bien como plantear el universo de activos y por ello decidimos descartar este planteamiento.

Mas adelante, a medida que íbamos avanzando con el temario del master y llegamos al apartado de Machine Learning empezamos a descubrir nuevas funcionalidades, y en consecuencia, planteamos nuevas ideas.

Dentro del ML, nos enfocamos en el Reinforcement Learning, porque en ese momento nos pareció la mejor técnica de las cuatro que lo componen. Poco a poco, empezamos a concretar la idea de realizar un rebalanceador de estrategias de inversión, alternando estrategias clásicas con otras aplicando RL.

Una cosa teníamos clara, habíamos decidido definitivamente que nuestro trabajo iba a consistir en un rebalanceador de estrategias. En ese momento, vimos que la creación del entorno nos iba a suponer grandes problemas de computación y consecuentemente numerosos costes a los que no nos podíamos enfrentar al menos para este trabajo. Por lo tanto, objetamos que debíamos descartar esa opción y volcarnos mas en el rebalanceador.

Una vez concretado eso, decidimos centrarnos en diferentes estrategias más clásicas y funcionales. Además, dijimos que preferíamos centrarnos en crear una buena estructura del rebalanceador que estuviera conectada al cloud de AWS, utilizando un bucket s3.

Finalmente hemos centrado nuestra idea en crear una buena base con una estructura solida que permitiera demostrar la funcionalidad y las necesidades de un rebalanceador y dejar en segundo plano las características de las diferentes estrategias, utilizando un conjunto de estrategias más conocidas.

Pasos realizados:

El primer paso fue crear el repositorio en GitHub, lugar donde íbamos a reunir las modificaciones del código común del grupo. Nuestro primer objetivo fue que cada uno contribuyera descargando datos de yahoo finance. Para no tener problemas con los datos decidimos escoger únicamente aquellos con un mínimo de histórico del año 2013 y que se comercializaran en euros, para que de esta manera evitáramos le problema de cambio de divisa.

A partir de ese momento decidimos que cada uno se centrará en crear una estrategia personal que incluir en el rebalanceador escogiendo al gusto la introducción de ML en la misma. Nos organizamos de esa manera durante los siguientes dos meses. Cada uno ya tenía una idea general de lo que quería hacer, pero no teníamos la certeza de si podíamos realmente concatenar bien los datos y las estrategias.

No habíamos establecido una idea general en cuanto a temporalidad y tipos de activos, por lo que cada uno había escogido diferentes mercados y tipos de activos como derivados acciones y ETFs. Por lo que hacía mas complejo el sistema y decidimos centrarnos únicamente en ETFs basados en euros.

Posteriormente, la siguiente decisión fue determinar el tipo de estrategias. Contemplamos la posibilidad de introducir estrategias novedosas y personales, pero hacia mas complejo la interacción con el rebalanceador, por lo que concretamos hacer un rebalanceador de estrategias pasivas.

El siguiente paso fue definir el tipo de red a utilizar en el modelo, que, en un principio, se pretendía utilizar RL tanto para el modelo como para algunas de las estrategias propuestas. Tras un gran análisis de tiempo, capacidad, disponibilidad de datos, conocimiento y numerosas conversaciones con diferentes profesores, descartamos la posibilidad de emplear RL y seguimos su recomendación sustituyéndolo por un aprendizaje supervisado.

Por último, decidimos crear un modelo de redes densas que intente determinar qué estrategia será la mas conveniente en el siguiente periodo. Primero, decidimos probar el modelo con inputs de rentabilidad y volatilidad de las estrategias. Vimos que a pesar de cambiar el periodo de rebalanceo seguía sin aprender, y por ello, decidimos introducir unos inputs de inflación europea y americana para ver si permitía mejorar el aprendizaje del test del modelo.

Tras la prueba, observamos que pudimos mejorar el accuracy en train, pero no en test, manteniendo aproximadamente los mismos resultados. Aún así, pasamos por el backtesting todos los modelos para poder comparar todas estas pruebas con el benchmark sintético y la prueba de aleatoriedad.

Tras el backtesting determinamos que al agregar los datos de inflación los modelos se desmarcan más del benchmark sintético.

Principales descartes realizados:

-Primer descarte:

Nuestro primer descarte fue la opción de hacer una cartera diversificada con activos alternativos. Decidimos rechazar ese tipo de trabajo cuando vimos que no cumplíamos con los datos históricos mínimos requeridos con criptomonedas. Queríamos utilizar datos diarios y no cumplíamos con el mínimo histórico de 5 años.

-Segundo descarte:

Tras el primer descarte la idea se basó en un rebalanceador con estrategias propias y RL. Decidimos emplear estrategias simples y trasladar la RL al rebalanceador.

-Tercer descarte:

El último cambio en la de nuestro TFM fue descartar la opción de aplicar aprendizaje por refuerzo y sustituirlo por aprendizaje supervisado. Por lo que el resultado definitivo es un rebalanceador de estrategias clásicas y propias basado en aprendizaje supervisado.

-Cuarto descarte:

Agregar otro tipo de redes tales como redes recurrentes o convolucionales al modelo. Se descarto por la naturaleza de los datos de entrada ya que este tipo de redes funcionan mejor clasificando imágenes.

Marco teórico:

En el marco teórico del presente trabajo, se pueden considerar los siguientes aspectos relevantes:

Estrategias de inversión: encontramos cuatro diferentes estrategias basadas en Momentum, mayor volumen negociado, mayor y menor volatilidad, también estaría la estrategia efectivo. Todas estas estrategias invierten en diferentes etfs. .

Rebalanceo de cartera: rebalanceo estático de la cartera mensualmente.

Aprendizaje supervisado: basado en una red neuronal que recibe como input las rentabilidades y volatilidades de las estrategias junto con la inflación

Evaluación del modelo: valoramos diferentes métricas utilizadas para evaluar la eficacia de los modelos, como el categorical crossentropy, la precisión, el recall, entre otros.

3.Estado del arte:

El estado del arte para un rebalanceador de estrategias de inversión basado en aprendizaje supervisado está evolucionando rápidamente a medida que se desarrollan nuevas técnicas y algoritmos de aprendizaje automático. A continuación, se presentan algunas tendencias generales en la investigación y las aplicaciones de estas tecnologías:

Estrategias basadas en índices: La administración de carteras se basa típicamente en estrategias de inversión basadas en índices, en las que se intenta replicar el rendimiento de un índice de referencia en lugar de intentar superarlo. Los rebalanceadores basados en aprendizaje supervisado pueden ayudar a ajustar automáticamente la composición de la cartera para mantenerla en línea con el índice de referencia y minimizar los desvíos.

Modelos predictivos: Los algoritmos de aprendizaje supervisado pueden ser entrenados para predecir el rendimiento futuro de los activos financieros, lo que puede ser utilizado para ayudar a ajustar la composición de la cartera en tiempo real.

Análisis de sentimiento: Algunos algoritmos de aprendizaje supervisado se basan en el análisis de sentimiento para predecir los movimientos del mercado y el rendimiento futuro de los activos financieros. Estos algoritmos pueden ser utilizados para ajustar la cartera en función de las noticias y otros eventos que puedan afectar el mercado.

Principales técnicas de rebalanceo:

Rebalanceo periódico: esta técnica consiste en fijar un periodo de tiempo para revisar y ajustar la cartera. Por ejemplo, se puede establecer un rebalanceo trimestral, semestral o anual. En este caso, se venden los activos que han subido de precio y se compran los que han bajado, para mantener el peso objetivo de cada uno.

Rebalanceo por bandas: se fijan unos límites superior e inferior para el porcentaje de cada activo en la cartera. Cuando el valor de un activo supera el límite superior, se vende una parte para reducir su peso en la cartera. Cuando el valor de un activo cae por debajo del límite inferior, se compra para aumentar su peso en la cartera.

Rebalanceo por flujo de caja: esta técnica se utiliza para mantener la proporción de la cartera cuando se realizan aportaciones o retiradas de dinero. Si se hace una aportación, se invierte en los activos que están por debajo de su peso objetivo. Si se hace una retirada, se venden los activos que están por encima de su peso objetivo.

Rebalanceo dinámico: en este caso, se utiliza un algoritmo que ajusta la cartera en función de las condiciones del mercado y de las previsiones de futuro. Por ejemplo, se pueden utilizar técnicas de análisis técnico o de aprendizaje automático para detectar patrones en el comportamiento de los activos y ajustar la cartera en consecuencia.

4. Metodología:

Una descripción detallada de los métodos y herramientas utilizados en el proyecto, incluyendo la selección de los datos, las técnicas de modelado y los algoritmos de rebalanceo.

Selección de datos:

Previamente a la selección de los datos tuvimos que decidir el tipo de activos en los que nos queríamos centrar. Comenzamos escogiendo cada uno diferentes tipos de activo para cada estrategia, pero por la falta de datos de futuros y por la complejidad de su concatenación e interacción decidimos estandarizar el tipo de activos con los que hemos trabajado. Posteriormente, dudamos entre seleccionar acciones, ETFs o derivados.

Decidimos trabajar únicamente con ETFs, ya que nos permitía abarcar de una mejor manera un universo de activos mas amplio. Nos basamos principalmente en la página de justetf (1). Nos pareció una página que ofrecía un catálogo de etfs por sector, por lo que fuimos seleccionando aquellos valorados en euros y que mínimamente tuvieran datos históricos desde el 1 de enero de 2013, todo esto descargado desde Yahoo Finance(2).

Obtención y limpieza de datos de activo:

Con la librería boto3 nos conectamos al bucket s3 de AWS, y de esta manera se dan las claves de acceso consiguiendo entrar al bucket y obtener la lista de objetos que contiene el histórico de los activos. Después, se realiza un “loop” para leer todos lo csvs, exceptuando los datos de inflación.

De cada archivo csv nos quedamos con los datos de apertura, cierre y volumen negociado entre el rango de fechas del uno de enero de 2013 y el 1 de noviembre de 2022 de los etfs. Se indexa la fecha y se colocan los activos con sus nombres por columnas.

Posteriormente, se realiza un thresold del 2% por columna para hacer una primera limpieza de los NA. Aun así, sigue habiendo un 1% de NAs, por lo que rellenamos un con el método full fill con un máximo de 2 datos, si siguen habiendo NA se descarta el activo. Finalmente nos quedamos con cincuenta y seis activos. Convertimos la fecha a formato datetime para el conjunto de activos.

El proceso anterior se reúne en la función obtener datos, la cual tiene tres outputs: las variables activos_close, activos_volumen y activos_open.

Este procedimiento también se realiza para los datos de inflación europeos y americanos de manera independiente ya que necesita otro tratamiento de los datos por que se tratan de datos mensuales. Lo correspondiente a la inflación se recoge en la función obtener_macros.

Para tener una primera imagen de los datos descargados plasmamos en un gráfico las rentabilidades individuales de los cinco mejores/peores activos.

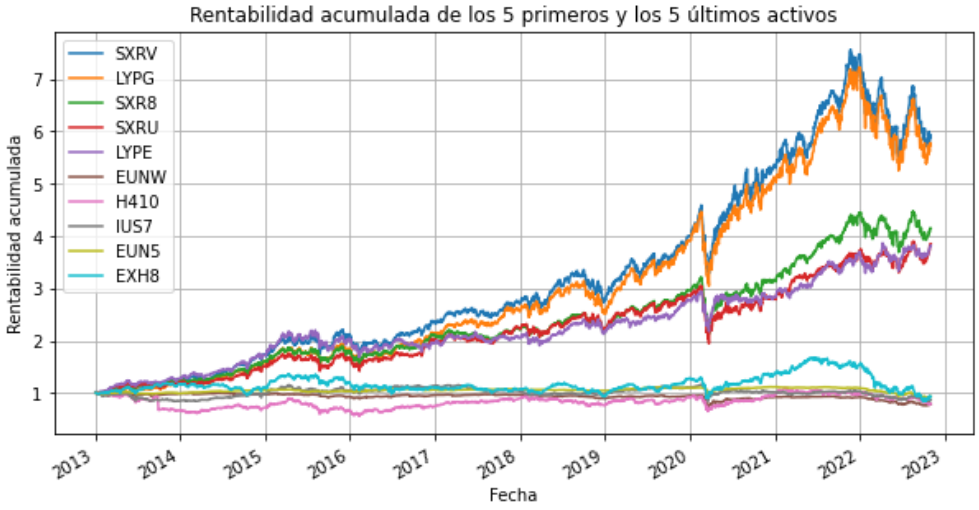


Gráfico 1: Rentabilidad acumulada de los cinco primeros y últimos activos

Podemos ver que las rentabilidades de los activos seleccionados oscilan mayoritariamente entre el 0% y el 200% en un periodo de diez años. Con esto podemos constatar que las rentabilidades de las estrategias rondaran en este mismo rango.

Posteriormente, se definen las variables de capital inicial, numero de activos y la comisión.

La variable capital_inicial sería 10000 euros, y serviría para cada estrategia. En su conjunto habrá un total de 44444 euros con un 10% de capital reservado para el efectivo al inicio del periodo, n_activos corresponde con los activos que se va a trabajar en cada estrategia, en nuestro caso diez.

A posteriori, para poder conseguir los datos de input que requiere nuestro modelo se necesitan las series de precios de cada una de las estrategias y por esto con los datos obtenidos de cada etf se van a calcular las series de precios históricas de cada una de las estrategias. Cada estrategia está definida como una función que recibe como input las variables activos_close, n_activos, capital_inicial y comisión.

Dentro de la función se realiza un loop (bucle) que va iterando el dataframe de los activos_close y mensualmente calcula cuanto capital tiene disponible para invertir. En función de la estrategia calcula en que activos invertir. En el caso de la estrategia momentum, selecciona los 10 activos con mayor rentabilidad del mes anterior; en el caso de EW, calcula los 10 activos con mayor volumen negociado el mes anterior; el de volatilidad máxima, selecciona los 10 activos con mayor volatilidad anterior; y vol_min, los activos menor volatilidad anterior.

Una vez que cada estrategia ha seleccionado sus correspondientes activos, se calcula en función del capital disponible el número de acciones a comprar en base al último close del mes. Ese proceso queda almacenado en la variable capital invertido y calculamos el excedente debido a la imposibilidad de compra fraccional de las acciones. Finalmente, se

obtiene una valorización de la estrategia, es decir, conseguimos una serie histórica de precios reunida en la serie de cada estrategia'.

Graficamos la serie de precios y la volatilidad de cada estrategia.

- Estrategia Momentum:

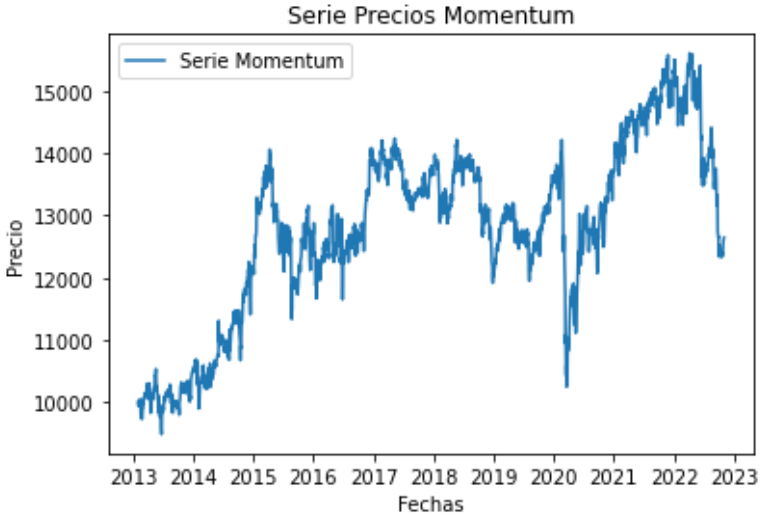


Gráfico 2:Serie de precios estrategia Momentum

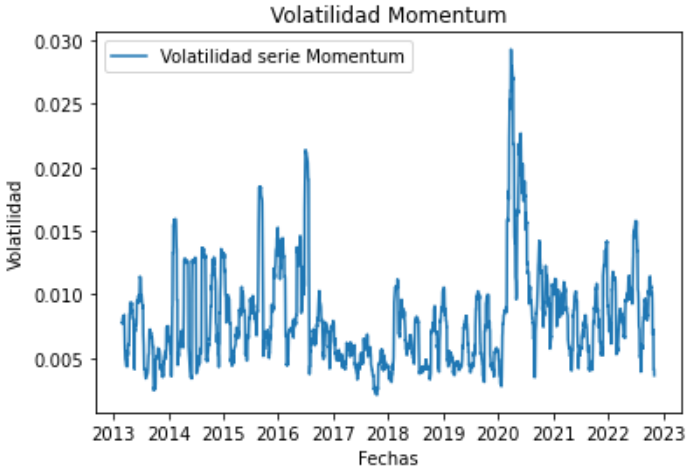


Gráfico 3: Volatilidad serie Momentum

Se puede observar de los gráficos 2 y 3 como que la serie momentum no logra obtener una rentabilidad más allá de 50% a lo largo del periodo, por otro lado suvolatilidad se mantiene relativamente constante a excepción del periodo Covid, donde claramente se visualiza su máximo.

- Estrategia Volumen:

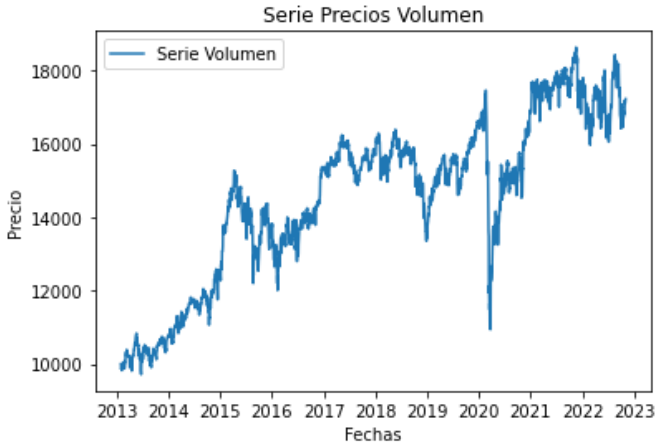


Gráfico 4: Serie de precios estrategia de la estrategia Volumen

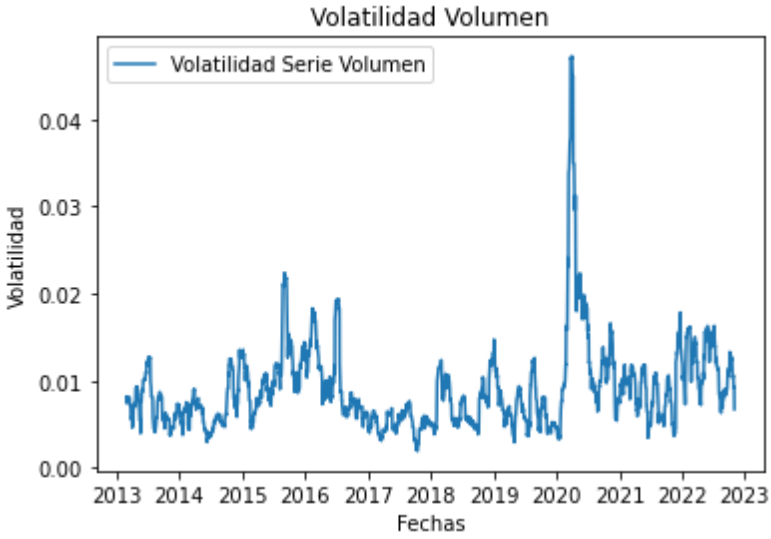


Gráfico 5: Volatilidad de la estrategia Volumen

Los gráficos 4 y 5, correspondiente a la estrategia volumen, nos muestran que logra obtener rentabilidades en casi todos los periodos llegando hasta un 80% en su mejor momento, adaptándose correctamente a la época con mayor volatilidad

- Estrategia máxima volatilidad:

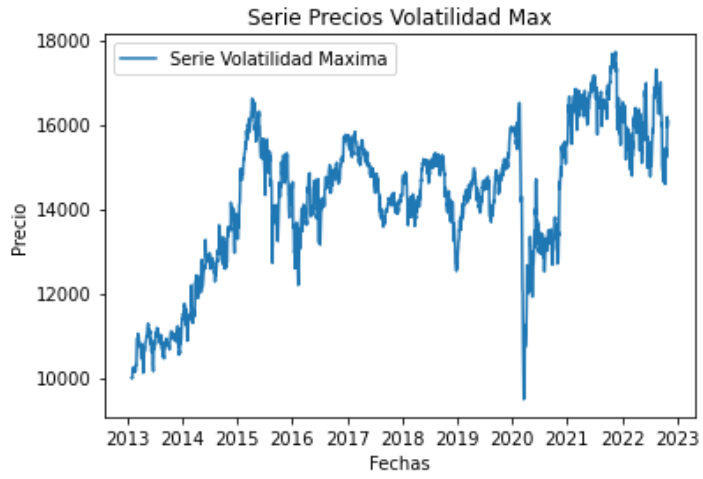


Gráfico 6:Serie de precios estrategia volatilidad máxima

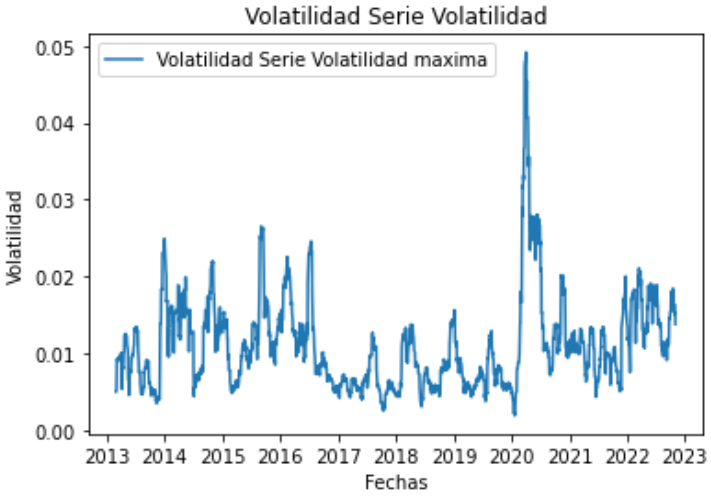


Gráfico 7:Volatilidad de la estrategia de volatilidad máxima.

Los gráficos 6 y 7, correspondientes a la estrategia de Volatilidad máxima, muestran que a pesar de funcionar bien al principio en los primeros periodos donde no hay mayor volatilidad en los mercados, no logra su objetivo en periodos menos estables.

- Estrategia volatilidad mínima:

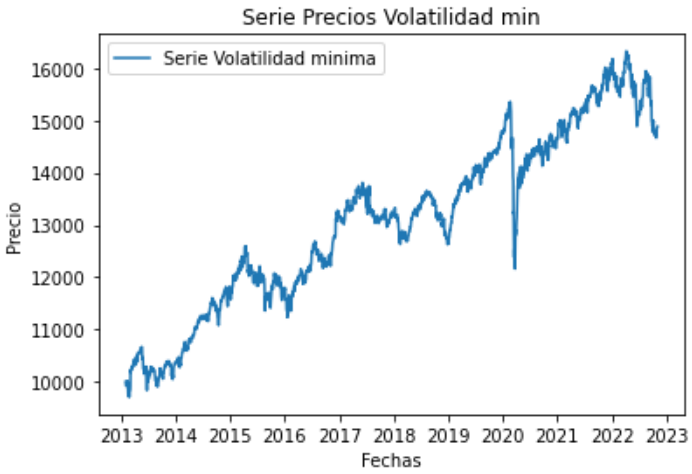


Gráfico 8: Serie de precios estrategia de volatilidad mínima

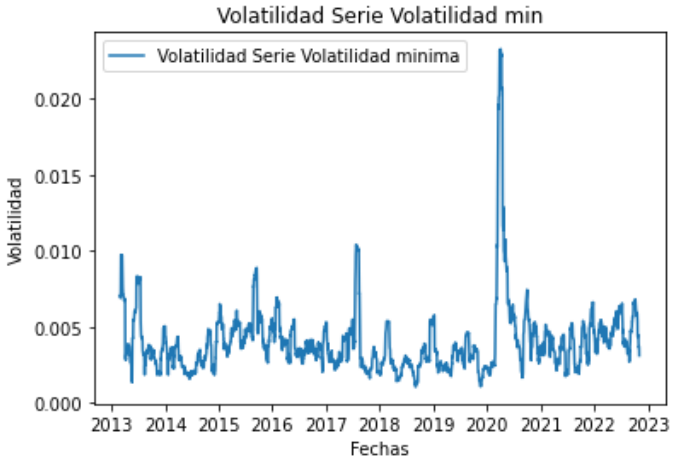


Gráfico 9: Volatilidad de la estrategia de volatilidad mínima

Los gráficos 8 y 9, correspondientes a la serie de volatilidad mínima, funciona de manera mas estable, como era de esperarse por la construcción de la misma.

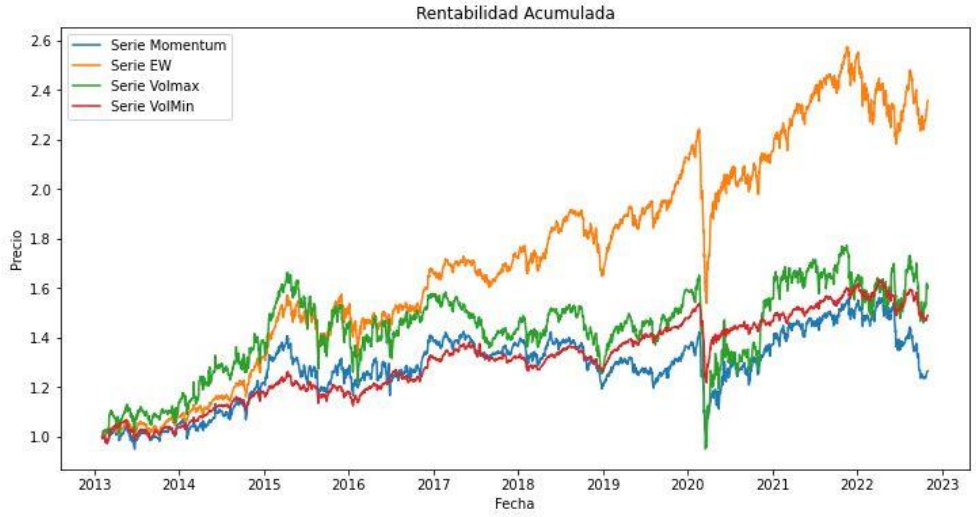


Gráfico 10: Rentabilidades acumuladas del conjunto de estrategias

Se observa en el grafico 10 que la estrategia de volumen es la líder en términos de rentabilidad acumulada, en cambio las estrategias restantes tienen una rentabilidad considerablemente menor, pero similares entre ellas. En cuanto a las volatilidades de las estrategias podemos destacar una observación curiosa en el caso de la estrategia por volumen, ya que no se trata de la estrategia más volátil del conjunto, pero en cambio, es la estrategia más rentable.

La estrategia más volátil como pudiera ser predecible es la de volatilidad máxima y la mínima la estrategia contraria. Otro punto considerable es que la estrategia momentum tiene mayor volatilidad que la estrategia Volatilidad mínima y aun así no logra superarla en términos de rentabilidad.

Explicación y pasos realizados en el Modelo:

Definimos la función de nuestro modelo que recibe como input las series de cada estrategia además de los datos de inflación. Nombramos las columnas con cada serie de estrategias y definimos una serie de efectivo, serie_cash.

Todo se concatena en un mismo dataframe llamado estrategias_close al cual se le calculara la rentabilidad diaria y posteriormente se le aplica un resample para obtener los datos de rentabilidad y volatilidad mensuales.

Mas adelante, los resamples son guardados en nuevos dataframes donde se les cambia el nombre de las columnas a cada una. Todo se concatena en un nuevo dataframe datos_inputs, acto seguido, se borran las filas con NA, más concretamente la primera y última fila.

Después, a este dataframe se le agrega las variables de inflación mensuales. Esto configura el dataframe datos_inputs conformando así el input de nuestro modelo el cual tiene doce características, por lo que el dataframe tiene doce columnas.

En estas doce columnas contemplamos cinco columnas de rentabilidades (cuatro estrategias más la correspondiente a cash) cinco columnas de volatilidades y los datos de inflación americanos y europeos. Teneos un conjunto de ciento dieciséis muestras.

El target de nuestro modelo es la estrategia con mayor rentabilidad en el siguiente periodo. Para calcular el target nos quedamos para cada muestra la estrategia más rentable. Posteriormente se borran, las dos primeras filas, lo que provoca que la estrategia más rentable quede como output de la primera muestra en el periodo siguiente. Finalmente, tenemos el dataframe con los input y output que necesita cada modelo para su entrenamiento.

En el caso específico de nuestro modelo pasamos los datos output a categóricos de cinco clases ya que la última capa del modelo tiene cinco neuronas y separamos los datos en train y test, con un tamaño del test del 30% y una semilla para que siempre sean los mismos datos. A los conjuntos de x_train y x_test se le aplica una estandarización para que todos los datos de input tengan el mismo rango.

Técnicas de modelado:

Nuestro modelo:

En esta tesis se presenta un modelo de redes neuronales utilizando la librería Keras, el cual ha sido diseñado para rebalancear estrategias de inversión. El modelo es un Multilayer Perceptron (MLP) con 3 capas densas, una capa de normalización por lotes y una capa de abandono (dropout) entre cada capa densa para reducir el sobreajuste. El modelo utiliza una estrategia de inversión que divide el capital entre cinco opciones de inversión, y por lo tanto, la capa de salida tiene 5 neuronas con una función de activación softmax para que las salidas sumen 1.

La elección del número de neuronas en cada capa y las tasas de regularización se han ajustado de manera experimental para obtener el mejor rendimiento del modelo. Además, se ha utilizado la regularización L1L2 en los pesos de las capas densas para evitar el sobreajuste.

-Capa de entrada: esta capa define la forma de entrada de los datos y especifica el tamaño de la capa oculta. En este modelo, se usa una capa densa con 128 neuronas y una función de activación ReLU (Rectified Linear Unit).

-Capa de normalización por lotes: esta capa normaliza los datos para que tengan una media cercana a cero y una desviación estándar cercana a uno, lo que ayuda a mejorar el entrenamiento de la red.

-Capa de activación: esta capa utiliza la función de activación sigmoidal para introducir no linealidad en la red.

-Capa de Dropout: esta capa ayuda a prevenir el sobreajuste de la red, eliminando aleatoriamente algunas de las conexiones entre las neuronas durante el entrenamiento.

-Capa oculta: esta capa utiliza otra capa densa con 64 neuronas y una función de activación ReLU.

-Capa de normalización por lotes: como antes, normaliza los datos para mejorar el entrenamiento.

-Capa de activación: otra capa de activación sigmoidal.

-Capa de Dropout: otra capa que ayuda a prevenir el sobreajuste de la red.

-Capa de salida: esta capa utiliza la función de activación softmax para clasificar los datos en una de las cinco categorías posibles.

Después de definir el modelo, se utiliza el optimizador Adam para minimizar la función de pérdida categorical_crossentropy. Se emplea el optimizador Adam y la función de pérdida Categorical Crossentropy juntos en el entrenamiento para la clasificación de múltiples clases porque Adam es un algoritmo de optimización eficiente y adaptable, mientras que la Categorical Crossentropy es una función de pérdida adecuada para medir el rendimiento de la red neuronal en la clasificación de múltiples clases.

Se entrena el modelo durante un máximo de 1000 épocas con una validación del 10% y se utiliza el método EarlyStopping para detener el entrenamiento si la pérdida de validación no mejora después de cinco épocas consecutivas. Finalmente, se guarda el historial del entrenamiento en la variable history.

La función modelo devuelve los datos input, datos output, datos input estandarizados. Con ellos obtenemos una serie histórica de las predicciones cada estrategia de nuestro modelo a lo largo del conjunto de meses de nuestro dataframe.

Para compáralo, vamos a definir dos nuevos modelos, una regresión logística multiclase y un modelo de red neuronal multiclase simple, obtenido del temario de clase de Luis Lago.

Modelo de regresión logística:

Primeramente, los datos se dividen en conjuntos de entrenamiento y prueba utilizando la función train_test_split de scikit-learn. Luego, los datos de entrenamiento y prueba se estandarizan utilizando la clase StandardScaler de scikit-learn.

A continuación, se define una clase para crear el modelo de regresión logística (3). El constructor de la clase inicializa los parámetros del modelo: una matriz de pesos W y un vector de sesgo b. La clase también tiene métodos para predecir las salidas del modelo, calcular la pérdida, ajustar los parámetros del modelo utilizando el algoritmo de descenso de gradiente y calcular la precisión del modelo.

Después, se define otra clase para crear el modelo de regresión logística multiclase. El constructor de esta clase crea múltiples instancias de la clase anterior, una para cada clase en los datos. El método fit de esta clase ajusta cada modelo individual a su clase correspondiente utilizando los datos de entrenamiento. El método predict de esta clase calcula las salidas de todos los modelos y normaliza los resultados para obtener

probabilidades para cada clase. El método accuracy de esta clase calcula la precisión del modelo utilizando las salidas de todos los modelos.

Finalmente, se construye y entrena el modelo multiclase utilizando los datos de entrenamiento estandarizados y la clase MulticlassLogisticRegressionModel. El modelo se entrena utilizando el algoritmo de descenso de gradiente con un learning rate de 0.8 y 200 épocas. El modelo se ajusta a las salidas (y_train) utilizando la función fit.

Modelo de red neuronal:

Primero, se divide el conjunto de datos en dos conjuntos: uno para entrenamiento y otro para pruebas. Esto se hace con la función train_test_split de Scikit-learn.

A continuación, se estandarizan los datos. La estandarización se realiza para que todas las características tengan la misma escala. Esto es importante para que el algoritmo de entrenamiento no sea influenciado por características con valores muy grandes o muy pequeños. La estandarización se realiza mediante la clase StandardScaler de Scikit-learn.

Se define una clase llamada NeuralNetworkModel, que es la implementación de la red neuronal multicapa (3). En el método _init_, se inicializan los parámetros de la red: las matrices de pesos W y los vectores de bias b. Estos parámetros se inicializan aleatoriamente. Después de entrenar el modelo, se grafica el error en cada época.

Se hacen predicciones utilizando el conjunto de entrenamiento y el conjunto de pruebas, y se calcula la precisión (accuracy) en cada conjunto.

Diferencias entre los modelos:

El segundo modelo utiliza una regresión logística para clasificar las observaciones en una de dos clases, mientras que los dos modelos de redes neuronales se utilizan para clasificar observaciones en una de varias clases.

El tercer modelo utiliza una red neuronal con una única capa oculta, mientras que nuestro modelo utiliza una red neuronal con dos capas ocultas. Además, incluye algunas técnicas de regularización, como la regularización L1 y L2 y la deserción, para evitar el sobreajuste del modelo.

En resumen, la principal diferencia conceptual entre los modelos es la complejidad y flexibilidad que cada uno ofrece para ajustarse a diferentes tipos de datos y problemas de clasificación. Mientras que la regresión logística es una técnica simple y lineal que funciona bien para problemas de clasificación binaria, las redes neuronales son modelos más complejos y flexibles que pueden manejar datos más complejos y problemas de clasificación multiclase. Además, las técnicas de regularización son útiles para prevenir el sobreajuste del modelo.

5. Cloud:

En cuanto a la infraestructura Cloud hemos utilizado únicamente el servicio de AWS S3, ya que proporciona un almacenamiento seguro y escalable (5). Además, es fácil de acceder y compartir los archivos con otros servicios de AWS. Al no tener grandes cantidades de datos, ni unas estrategias que necesiten gran capacidad de computación, no vimos la necesidad de realizar una infraestructura tan amplia en el Cloud. Pero en caso de utilizar una mayor cantidad de datos y estrategias o rebalanceador que requiera de mayor capacidad computacional se podría realizar de la siguiente manera.

Para ello se mantendría el uso de un bucket S3 para almacenar los archivos csv, que esté conectado con Amazon Glue para procesar los archivos y almacenarlos en una tabla, posteriormente con el mismo Amazon Glue se puede crear un flujo de trabajo para que ejecute los scripts de Python almacenados también en S3. Para los resultados se puede utilizar Amazon SES, de manera que llegarían por correo electrónico o a su vez almacenarlos en Amazon Athena para que puedan ser consultados.

Amazon Glue es un servicio de procesamiento de datos ETL que puede ayudar a transformar y limpiar los datos almacenados en S3, y luego almacenarlos en una tabla que puede ser utilizada para análisis posteriores.

Crear un flujo de trabajo con Amazon Glue para ejecutar scripts de Python almacenados en S3 es una gran opción para automatizar el proceso de transformación de datos y hacerlo más eficiente.

Para enviar los resultados a los usuarios, Amazon SES es una excelente opción para enviar correos electrónicos automatizados con los resultados. Y si se desea, también se pueden almacenar los resultados en Amazon Athena, lo que permitirá a los usuarios consultar y analizar los datos en tiempo real.

6.Resultados:

- Nuestro modelo:

Se entrena el modelo con los datos x_train y t_train con mil épocas de manera aleatoria con un validation_split del 10% y un callback de un early stoping como se define anteriormente.

Resultados de nuestro modelo:

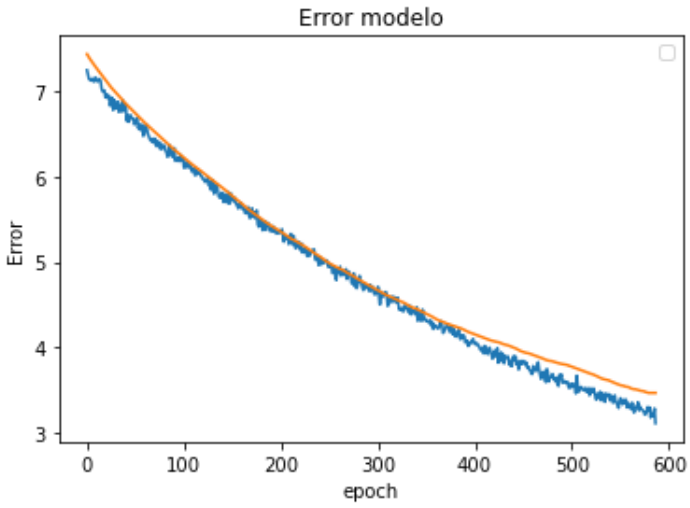


Gráfico 11: Error de nuestro modelo

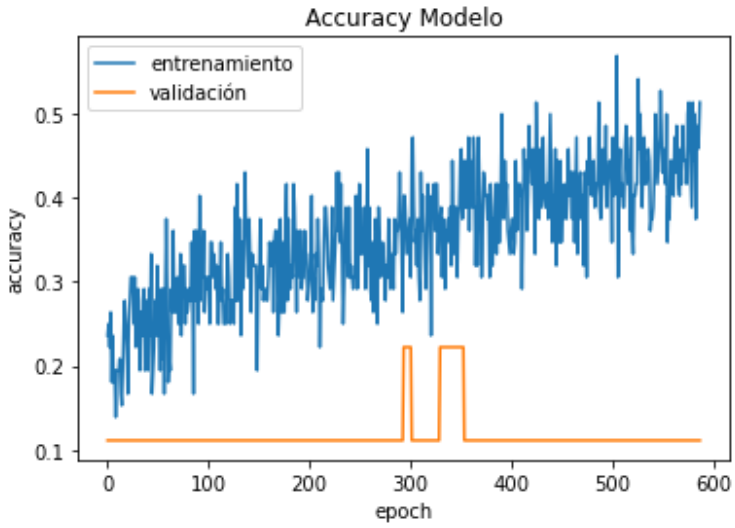


Gráfico 12: Accuracy del modelo

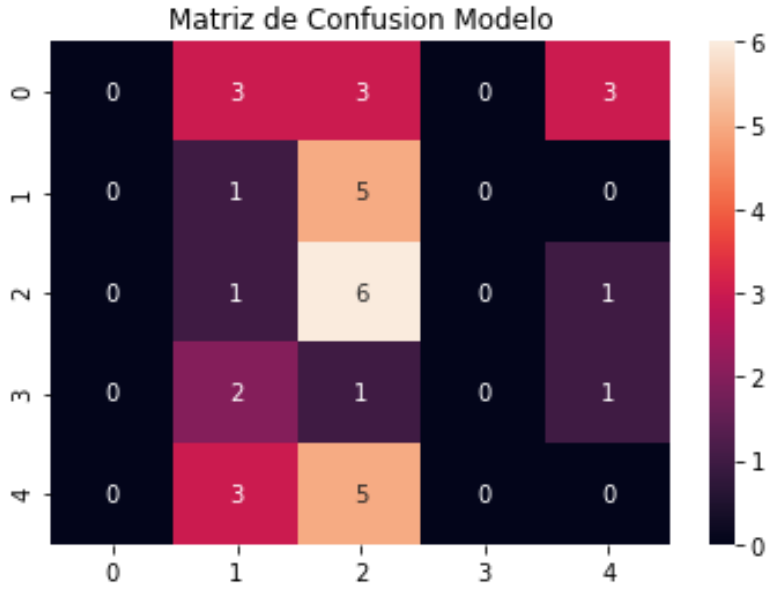


Gráfico 13: Matriz de confusión de nuestro modelo

De los gráficos anteriores se observa que el modelo logra disminuir el error progresivamente ya sea en train como en validación, por otro lado en términos de accuracy se ve claramente que el modelo logra mejorar en train, pero no en validación, obteniendo un resultado de 10% aproximadamente. Esto nos da un primer indicio de overfitting.

Realizando las predicciones en test obtenemos que en train tiene un 51% de accuracy y en test solo llega a un 20%. Esto nos confirma que el modelo no logra aprender de manera adecuada y se sobreajusta a los datos de entrenamiento a pesar de tener técnica de regularización como el Dropout, Batchnormalization e incluso un Early Stopping.

Este problema ya existía en las primera pruebas del modelo, es decir que aun incluyendo las técnicas anteriores, no se consigue mejorar los resultados.

Por otro lado, analizando el grafico 27, de matriz de confusión, podemos determinar que el modelo no predice las estrategias de momentum ni de Volatilidad mínima, centrándose principalmente en la estrategia de volumen y máxima volatilidad.

- Modelo de regresión logística:

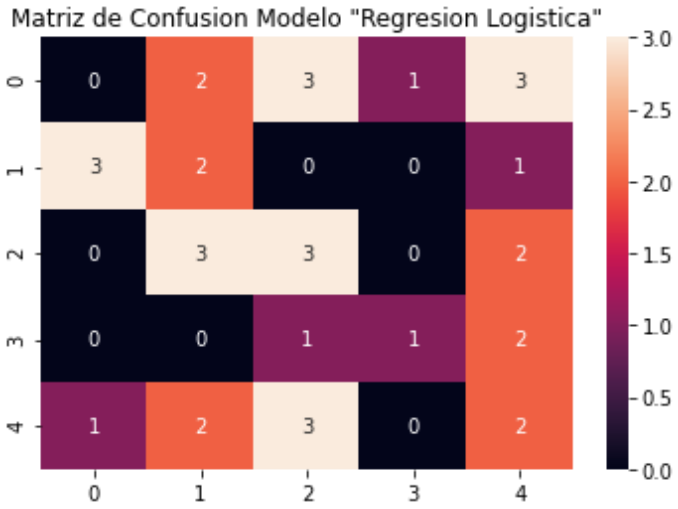


Gráfico 14: Matriz de confusión del modelo de regresión logística

Los resultados del modelo de regresion logística multiclase son de un 46% de accuracy en train y un 21% de accuracy en test. Por otro lado viendo el grafico 30, podemos observar que las estrategias de momentum y Volatilidad mínima no se predicen tanto, pero si se observa mayor peso en la estrategia Cash.

- Modelo de Red neuronal:

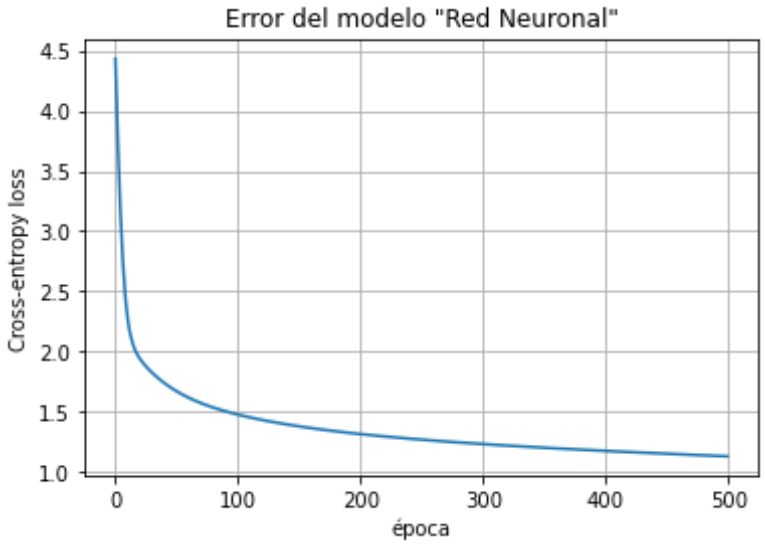


Gráfico 15: Error del modelo de red neuronal

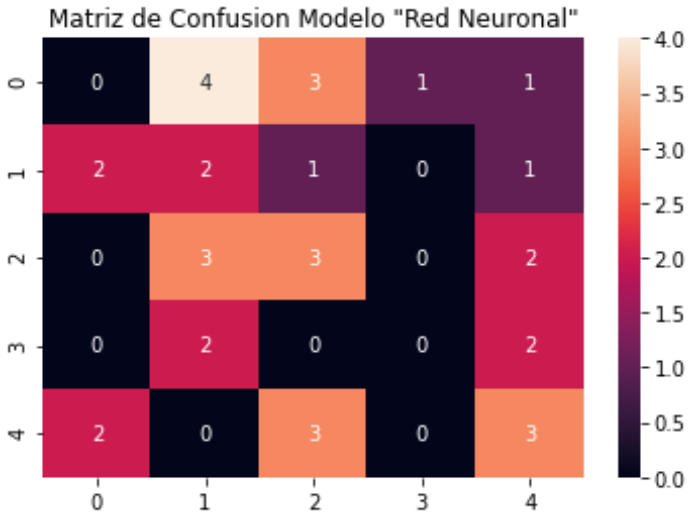


Gráfico 16: Matriz de confusión del modelo de red neuronal

Los resultados del modelo de red neuronal multiclase simple son de un 53% de accuracy en train y un 20% de accuracy en test. Por otro lado viendo el grafico 33, podemos observar que el error logra disminuir en el entrenamiento, pero aun así se puede ver que tiene overfitting ya que el accuracy en test es de solo un 20%.

Del grafico 34, extraemos que las estrategias de momentum y Volatilidad mínima no se predicen tanto, pero si se observa mayor peso en la estrategia Cash.

En general, los resultados de los modelos son muy pobres en términos de precisión, ya que la precisión de test oscila entre 20% y 21%. Esto indica que los modelos no son muy útiles para hacer predicciones precisas.

Todos los modelos tienen resultados similares en términos de precisión de train y test. Sin embargo, es importante tener en cuenta que la regresión logística es un modelo más simple en comparación con una red neuronal, lo que sugiere que el modelo de red neuronal podría estar sobredimensionado o sobreajustado en función de la cantidad de datos de entrenamiento.

También podemos concluir que los dos modelos de red neuronal a pesar de que el nuestro tiene técnicas de regularización, no logra mejorar el modelo simple.

Backtesting:

Posteriormente, definimos la función backtesting la cual recibe como input la variable activos_close, activos_vol, n_activos, capital_inicial, comisiones, activos_open y predicciones de algunos de los modelos (4).

Lo primero que se realiza es la inicialización de los dataframes para guardar la información de que y cuantas acciones comprar. Después con un bucle que itera diariamente, cuando se cumple la condición de rebalanceo de cambio de mes, se rebalancea modificando el capital asignado a cada estrategia según la predicción entregada por el modelo.

Una vez se sabe cuánto capital se asigna para cada estrategia en cada mes, se escogen según los criterios de cada estrategia que activos y cuantas acciones comprar. Luego, se calculan las diferencias con las posiciones del mes anterior y se obtiene una orden de compra/venta por cada estrategia. Se guarda y se suma el capital invertido en cada estrategia. Se consigue el delta en función del capital asignado en un comienzo.

También se actualiza el dataframe de posiciones del mes anterior con las posiciones que quedaron invertidas actuales. Con las órdenes de compra de cada estrategia se calcula una orden de compra venta para el portafolio completo y a eso se le calcula la comisión. Se guarda la valorización día a día del portafolio en una variable que después será la serie histórico de los modelos.

Análisis del backtesting:

Dentro de la función de backtesting se calculan los ratios de sharpe y sortino, el drawdown, y el máximo drawdown. La función devuelve la serie histórica de precios del modelo completo, la comisión total, un dataframe con las ordenes de compraventa, los ratios, drawdons y máximo drawdown.

Después se llama a esa función y se aplica a los distintos modelos. Donde sacamos tres gráficos por cada modelo, los cuales se pueden ver a continuación.

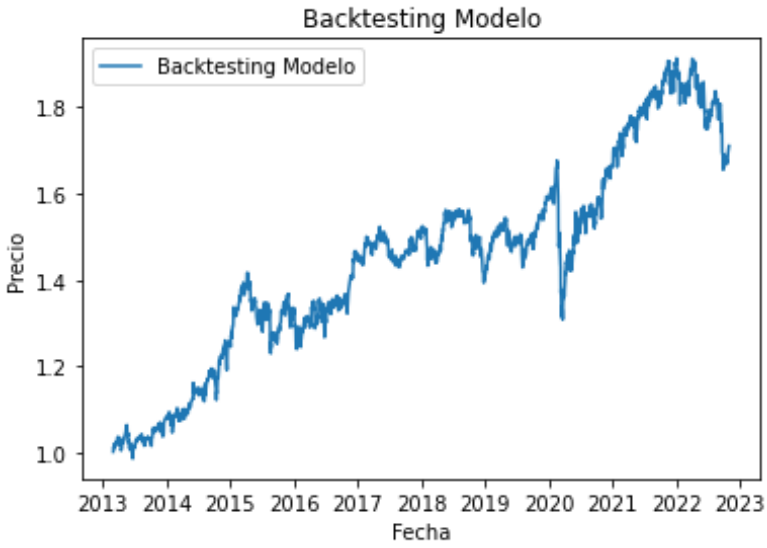


Gráfico 17: Backtesting de nuestro modelo

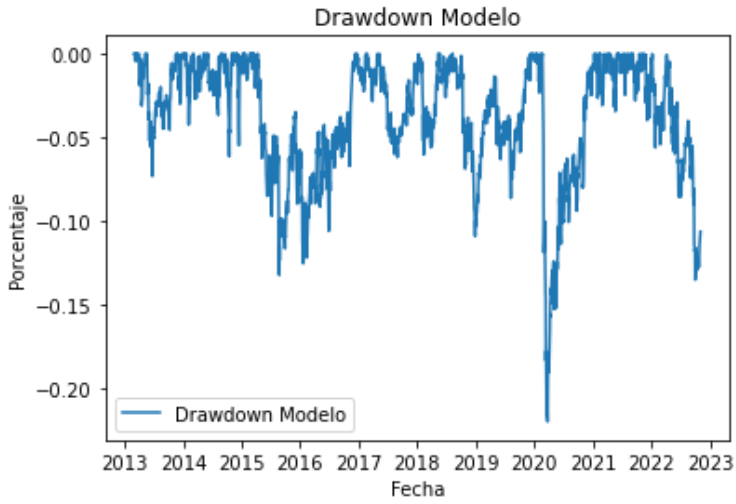


Gráfico 18: Drawdown de nuestro modelo

De los gráficos 17 y 18, podemos observar que nuestro modelo a pesar de no aprender y mostrar indicios de overfitting, logra obtener rentabilidad en el periodo analizado, por otro lado, observamos que el máximo drawdown se obtiene en el peor momento del covid, lo cual tiene sentido por como reaccionaron los mercados.

Se aplica la misma función de backtesting para el modelo de regresión logística y el de red neuronal.

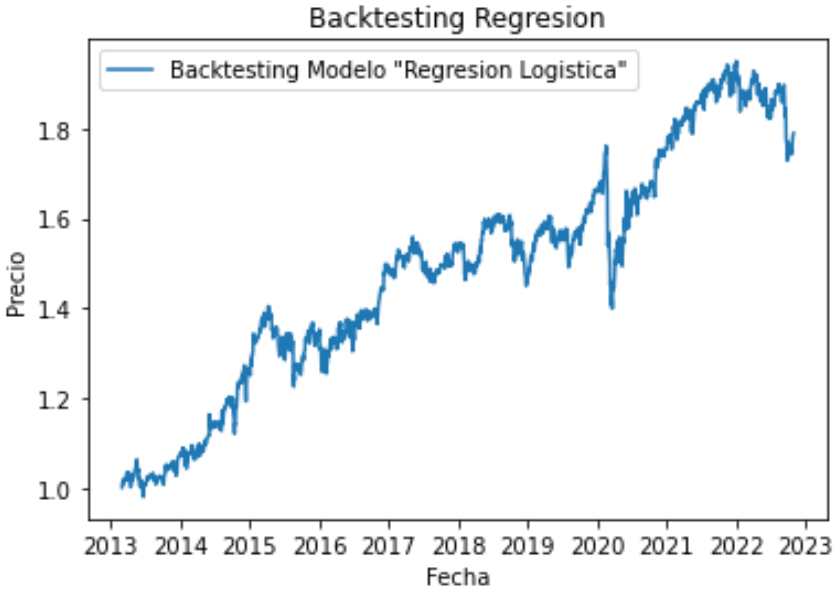


Gráfico 19: Backtesting del modelo de regresión logística

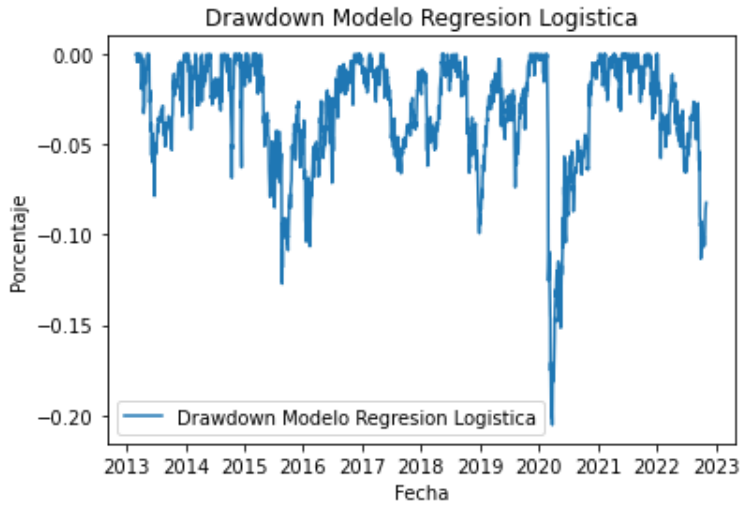


Gráfico 20: Drawdown del modelo de regresión logística

De los gráficos 19 y 20, podemos observar que el modelo de regresion logística, logra obtener rentabilidad semejante a nuestro modelo en el periodo analizado, por otro lado, observamos que el drawdown se comporta de la misma manera que los otros modelos.

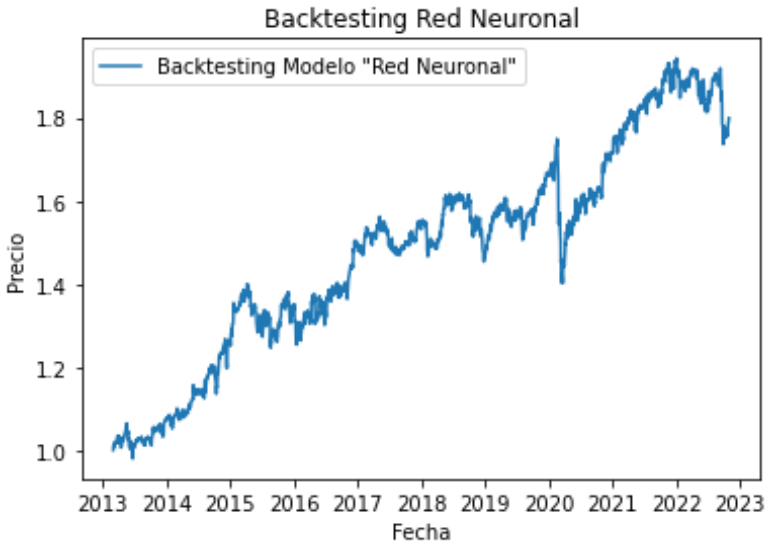


Gráfico 21: Backtesting del modelo de red neuronal

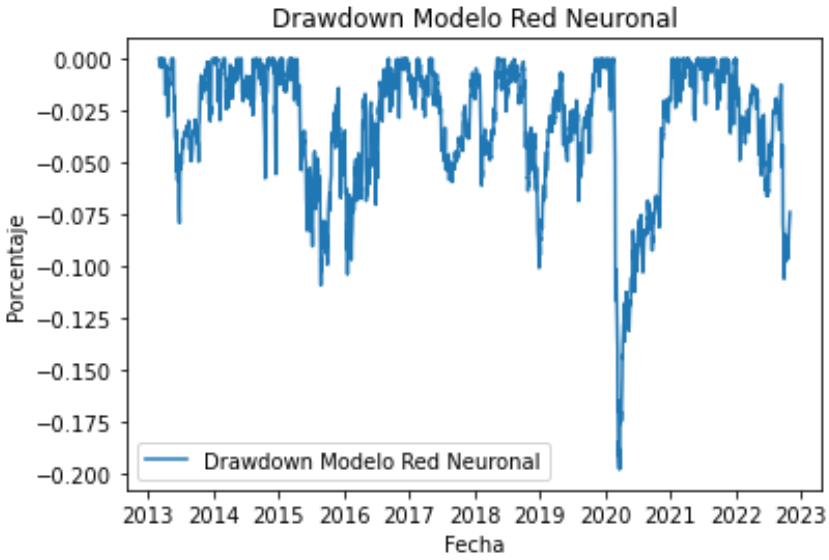


Gráfico 22: Drawdown del modelo de red neuronal

De los gráficos 21 y 22, podemos observar que el modelo de red neuronal, logra obtener rentabilidad semejante a nuestro modelo en el periodo analizado, por otro lado, observamos que el drawdown también se comporta de la misma manera que los otros modelos.

Siguiendo la misma lógica que la función backtesting se crea una función benchmark, la única diferencia es que hace los rebalanceos entre las cinco estrategias con EW.

Obteniendo los siguientes resultados:

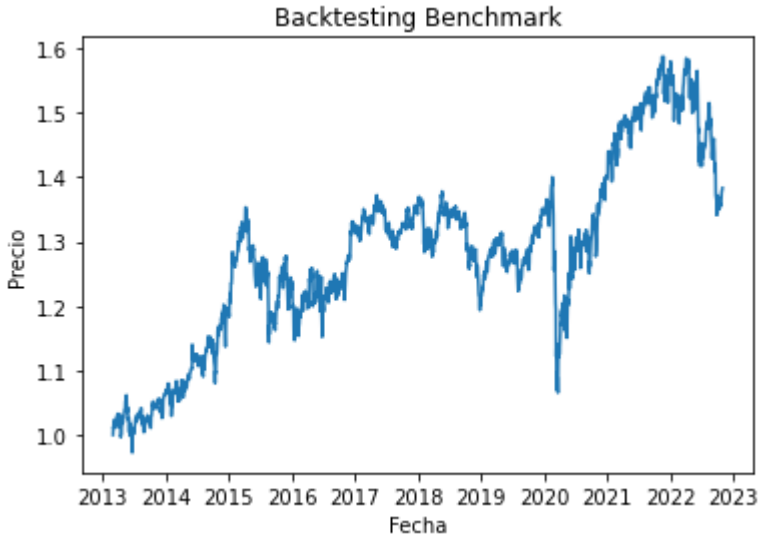


Gráfico 23: Backtesting del Benchmark

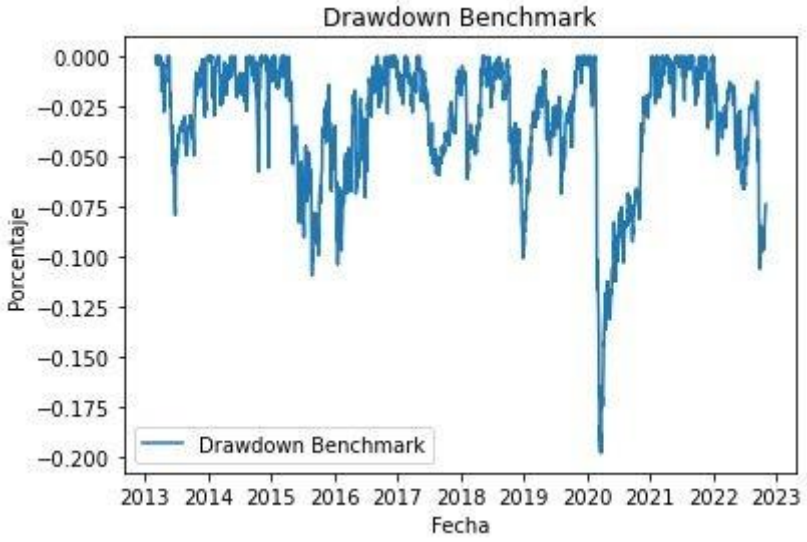


Gráfico 24: Drawdown del Benchmark

Podemos ver en los gráficos 23 y 24 que el benchmark obtiene una rentabilidad promedio de las estrategia. Al haber tres estrategias muy similares en términos de rentabilidad y solo una que es considerablemente más rentable, lógicamente los resultados de rentabilidad no van a ser buenos. Por otro lado, en términos de drawdown, podemos observar que es bastante estable, sin considerar el periodo de Covid.

Plasmamos los resultados en la siguiente tabla:

indicadores	Modelo	Regresión Log	Red Neuronal	Benchmark
Rent. Acumulada	171,06%	178,92%	179,91%	138,3%
Comisiones	3011,47	2918,72	2902,89	1956,28
Sharpe	-0,086	-0,088	-0,092	-0,089
Sortino	-1,05	-1,006	-1,05	-1,00
Max Drawdown	-21,97%	-20,53%	-19,82%	-23,82%

Tabla 1: Comparación resultados Backtesting

Analizando los resultados de cada modelo por separado, podemos observar que los tres modelos han obtenido mejores resultados que el benchmark sintético en términos de rentabilidad acumulada. La red neuronal ha sido la que ha obtenido una mayor rentabilidad acumulada con un 179,91%, seguida de cerca por la Regresión Log con un 178,92%, mientras que el Benchmark ha obtenido un 138,3%.

En cuanto a las comisiones, los tres modelos han incurrido en gastos similares, siendo nuestro modelo el que ha tenido un mayor gasto con 3011,47, seguida de cerca por la Regresión Logística con 2918,72, el Benchmark tiene bastante menos comisión con 1956,28.

En cuanto a las métricas de riesgo, el modelo Benchmark ha obtenido un Sharpe ligeramente peor que los otros modelos, con -0,089. Nuestro modelo con la Regresión

Log han obtenido valores muy similares en el Sharpe, con -0,086 y -0,088, respectivamente. En cuanto al Sortino, el modelo Benchmark también ha obtenido un valor ligeramente mejor que los otros modelos, con un -1,00. Los tres modelos han obtenido sortino muy cercanos de -1,05

En cuanto al máximo drawdown, los tres modelos han obtenido valores similares, siendo el Benchmark el que ha obtenido el peor resultado con un -23,82%, seguido por nuestro modelo con -21,97%. La Red Neuronal un -19,82%.

En general, podemos concluir que los tres modelos han obtenido resultados similares en términos de métricas de riesgo, pero han superado al modelo Benchmark sintético en términos de rentabilidad acumulada.

En conjunto, estos resultados sugieren que cualquier modelo puede ser eficaz para invertir en este universo de activos/estrategias, superando al modelo Benchmark sintético en términos de rentabilidad acumulada. Esto se debe a que las estrategias son rentables en el periodo analizado, por lo que los modelos no van a perder dinero, pero no quiere decir que aprendan y se adapten en cada momento.

Analizando todos los resultados obtenidos por los tres modelos, tanto los resultados de entrenamiento y prueba, como los del backtesting, determinamos que nuestro modelo no obtiene mejoras significativas, respecto a los otros modelos más simples, esto se podría deber a la simpleza de las estrategias, de la pequeña cuantía de datos y de la variedad de las características en los inputs de los modelos.

Prueba de la aleatoriedad:

Siguiendo la misma lógica de backtesting se crea una función monos para realizar la prueba de aleatoriedad donde el rebalanceo se hace con datos aleatorios que sumen 1 cada rebalanceo. Posteriormente, se hacen 2000 iteraciones sobre la función monos, y guardamos los resultados.

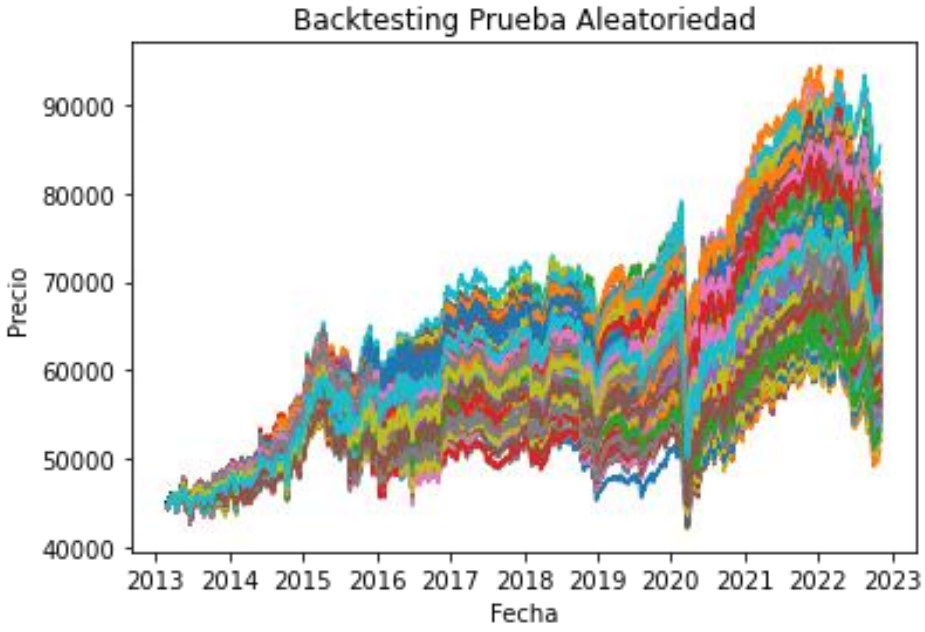


Gráfico 25: Backtesting de la prueba de aleatoriedad

Para efecto de comparación vamos a mostrar los monos del percentil 50 y 100. A estos le calculamos el retorno acumulado y graficamos todas las series de precios de cada modelo, benchmark y monos.

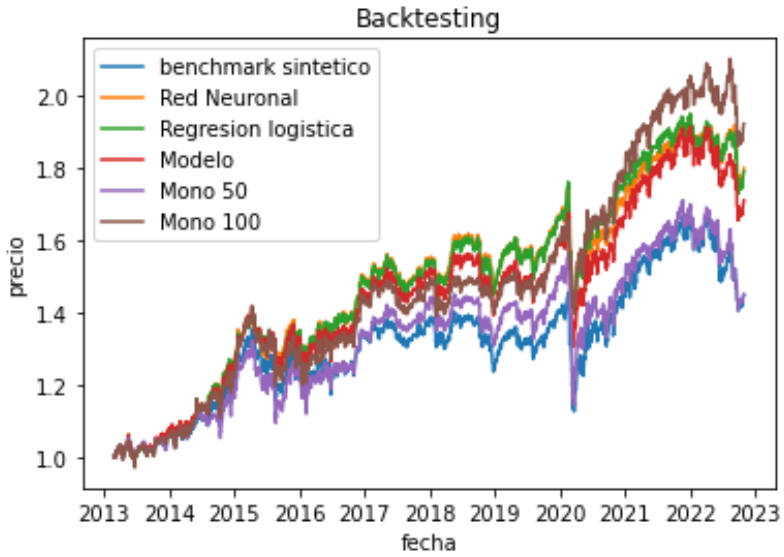


Gráfico 26: Backtesting de la prueba de aleatoriedad

En el grafico anterior, en primer lugar, se puede observar que el benchmark está en el percentil cincuenta de la prueba de aleatoriedad, lo que nos indica unos resultados coherentes. Por otro lado, se puede ver que los tres modelos de aprendizaje logran superar el benchmark sintético y quedan por debajo del percentil cien de la prueba de aleatoriedad.

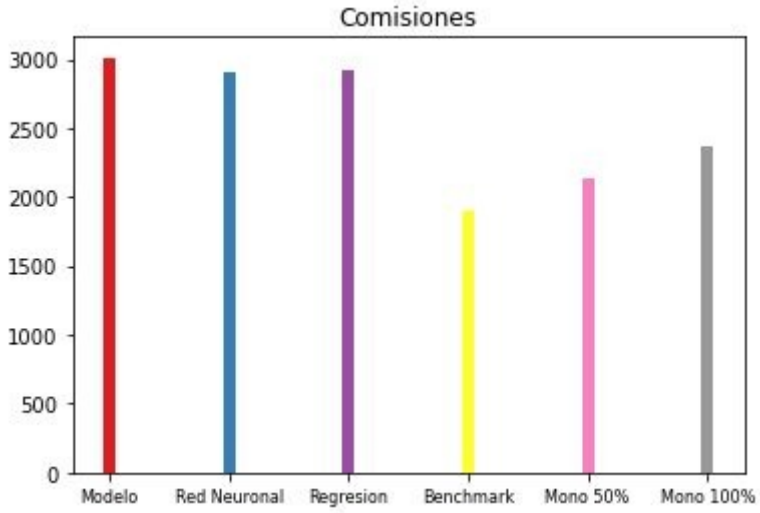


Gráfico 27: Comisiones

Podemos apreciar que los tres modelos han incurrido en comisiones similares, pero son mayores que el benchmark y las de la prueba de aleatoriedad.

7.Discusión:

Hemos podido ver que no hay tanta diferencia entre los resultados obtenidos a pesar de la diferencia en la arquitectura de las redes, como las técnicas de regularización, por ejemplo, dropout, bacthnormalization y los regularizadores L1 y L2. Además, en general los modelos acarrean problemas de overfitting y un rendimiento bajo ya que no hacen predicciones precisas y los resultados en test no son buenos.

También, hemos apreciado una alta correlación entre las diferentes estrategias que impide que el rebalanceador se adapte a los distintos periodos, y por ello, resultaría imposible superar el rendimiento de las estrategias.

Posibles mejoras del proyecto:

Para mejorar nuestro proyecto, se pueden considerar dos posibles mejoras: la primera consiste en utilizar estrategias más descorrelacionadas para reducir el riesgo y aumentar la diversificación de la cartera de inversión. Esto permitiría evitar que el rendimiento de la cartera dependa en gran medida del rendimiento de un solo activo, lo que podría generar una mayor estabilidad minimizando su volatilidad, y en consecuencia mejorar su rentabilidad a largo plazo.

En segundo lugar, se puede implementar un rebalanceador dinámico, donde la red pueda aprender cuándo y cómo rebalancear la cartera, y a su vez reducir los costos de comisión. Con un rebalanceo dinámico, se puede ajustar el peso de los activos en la cartera en tiempo real para maximizar las ganancias y minimizar las pérdidas, lo que resultaría en un mejor rendimiento.

8.Conclusiones:

Concluimos que, finalmente es mejor incorporar los datos de inflación en los inputs del modelo, por lo que el rebalanceador toma decisiones cuando cambia de mes, por otro lado, es un modelo que tiene rendimientos positivos durante el periodo analizado comparándose con el benchmark sintético, pero menor respecto a los dos modelos de comparación. Por lo que, el uso de modelos mas simples como la regresión logística que ofrecen resultados similares en un menor tiempo.

Tras todo el ejercicio de pruebas en nuestro modelo, pensamos que introducir más datos macroeconómicos y estrategias más descorrelacionadas pueda mejorar su aprendizaje. También que el rebalanceador tiene potencial, pero la rentabilidad está prácticamente capada a la de las estrategias, por lo que se debería probar nuevas posibles estrategias e ir analizando su rendimiento, esto también podría sacar más provecho a la red neuronal y desmarcarse del modelo de Regresión Logística.

Por último, pensamos que incorporar aprendizaje por refuerzo en diferentes estrategias o incluso en el mismo rebalanceador podría mejorar aún más los resultados.

9. Bibliografía:

(1) JustETF. (Septiembre, 2022). JustETF: Su fuente independiente para fondos cotizados en bolsa. Recuperado de https://www.justetf.com/es/

(2) Yahoo! Finance. (Octubre, 2022). Recuperado de https://finance.yahoo.com/

(3) Lago, L. (2022). Clase Redes Neuronales.

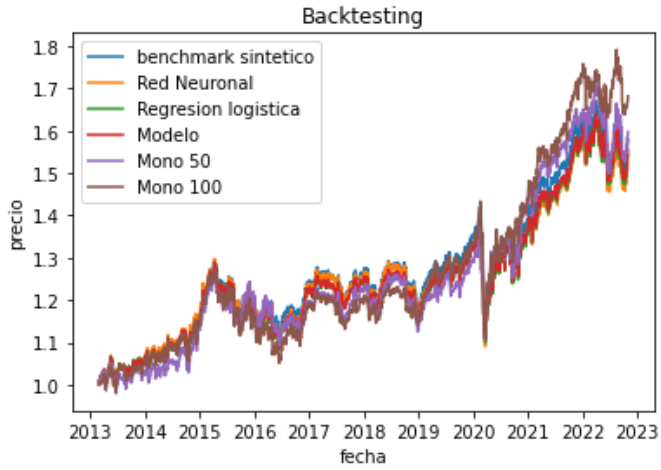
(4) De la Rosa, T. (2022). Backtesting Avanzado.

(5) De la Calle, F. (2022). Clase AWS y servicios Cloud.

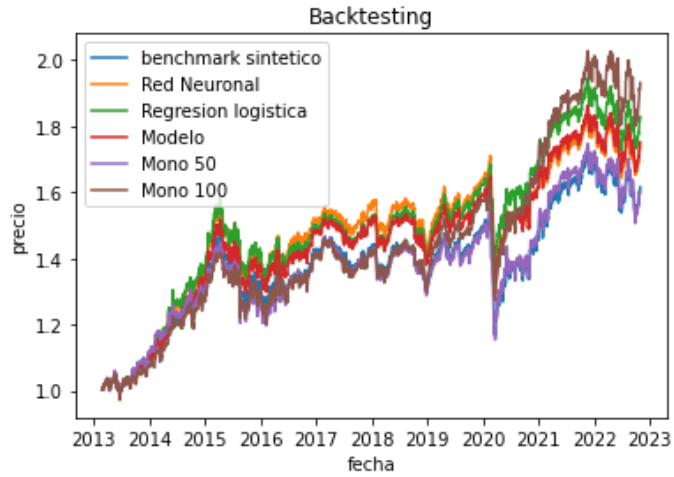
10.Anexos:

Gráficos del backtesting sin datos de inflación y rebalanceo cada X días:

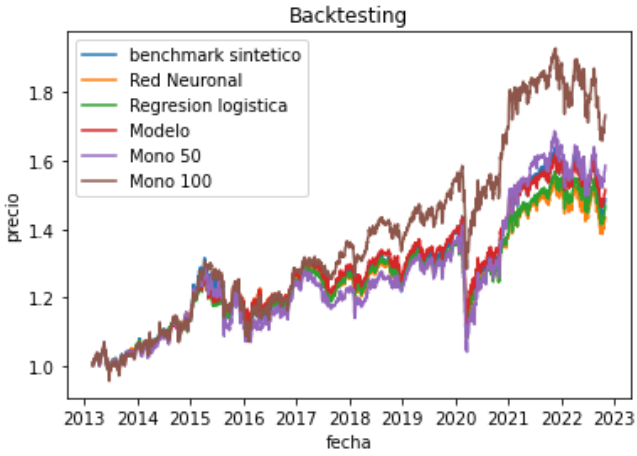
Rebalanceo 7 Días:



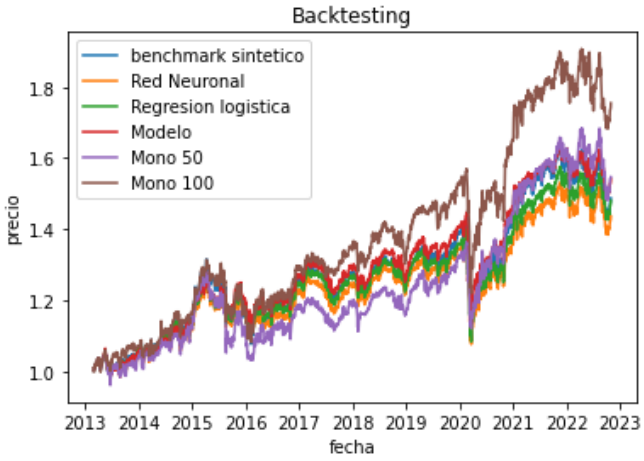
Rebalanceo 15 Días:



Rebalanceo 20 Días:



Rebalanceo 22 Días:



Podemos ver que estos rebalanceos consiguen distintas rentabilidades, y también en general no aprenden mucho si se comparan con el Benchmark o el mono del 50%