TRABAJO DE FIN DE GRADO

Grado en Estadística

Título: Redes Neuronales Recurrentes: Una aplicación para los mercados bursátiles

Autor: Pau Agustín Granell

Director: Salvador Torra Porras

Departamento: Econometría y Estadística

Convocatoria: Junio 2017

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Resumen:

Las redes neuronales recurrentes (*LSTM*) son una técnica muy popular para aprendizaje en secuencia. Sus características la convierten en una buena candidata para resolver uno de los problemas más difíciles de resolver en finanzas: La predicción de movimientos en los precios de *stocks*. Nuestro objetivo es poder predecir movimientos a 15 días de una compañía del índice *S&P* 500, EBAY, aplicando redes neuronales y *LSTM* a partir de únicamente los históricos de los precios de cierre de estas 500 compañías. Los movimientos se han clasificado en tres clases siendo 1 comprar, 0 mantenerse y -1 vender. Esta clasificación se ha hecho mediante la metodología de las Bandas de Bollinger, que dibuja unos límites para los cuales el precio se considera que está demasiado alto o bajo. Como *features* se han usado variables creadas a partir de los precios de cierre ajustados de las compañías del índice basándonos en los componentes de un posible sistema de *trading* para generar señales de compra y venta.

Palabras clave:

Redes neuronales, *LSTM*, Finanzas, *Machine Learning*, *Deep Learning*, Bandas de Bollinger, *trading*.

Abstract:

Comparison between neural networks and recurrent neural networks for stock movement prices using Bollinger bands as a classifier. Recurrent neural networks such as LSTM are a very powerful technique for sequence learning. Its characteristics make of it a very good candidate to solve one of the hardest problems in finance: Stock movements prediction. Our goal was to predict out of sample movements, 15 days forward, from a company of the S&P 500 index, EBAY, deploying LSTM and neural networks and feeding it only the information of the 500 historic adjusted closing prices. The movements have been classified as 1 buy, 0 hold and -1 sell following the Bollinger Bands methodology, which draws limits for the price to be considered too low or high. As features some variables have been created from the adjusted closing prices of the companies, those variables are based around possible components of a trading system to generate buy and sell signals.

Key words:

Neural Networks, LSTM, Finance, Machine Learning, Deep Learning, Bollinger Bands, Trading.

Clasificación AMS (*American Mathematical Society*)

62-04: *Explicit Machine computation programs (not the theory of computation or programming)*

62-07: *Data analysis*

91G99: *Mathematical finance*

ÍNDICE

1. INTRODUCCIÓN

1.1. Justificación

La realización de este trabajo viene motivada por el interés en conocer más sobre las técnicas de *Machine* y *Deep Learning* y su aplicación en series temporales. Específicamente técnicas como son las redes neuronales y las redes neuronales recurrentes, las cuales son cada día más populares y el conocimiento sobre estas puede resultar de mucha utilidad en el futuro profesional de cualquier estadístico.

Desde que se abrieron los mercados de valores, han aparecido infinidad de *gurús* que afirmaban haber encontrado la clave para predecir el futuro de los precios de los *stocks* a partir de únicamente los históricos de estos mismos. Aun así, si alguien ha sido capaz de encontrar un método infalible, no lo ha difundido seguramente para ser el único enriqueciéndose. Es por ello, que des del punto de vista que puede tener un estudiante de Estadística curioso, me pareció interesante aplicar estas metodologías para ver si se podía realmente predecir los movimientos del precio.

Este proyecto se realiza con la finalidad de aprender tanto como sea posible sobre estas metodologías y su aplicación en *Python* a series temporales. En ningún momento se espera encontrar unos modelos que me permita enriquecerme, ya que eso sería arrogante habiendo tanta gente mucho más preparada que yo intentando lo mismo sin éxito. Se intentará, pero, ver si alguno de estos dos algoritmos permite clasificar mejor las observaciones y si es posible únicamente con la información de los precios predecir algo con un mínimo de seguridad.

1.2. Objetivos

Como se ha comentado, el principal objetivo que persigue este trabajo es aprender tanto como sea posible sobre estas metodologías y su aplicación en *Python*, unas metodologías y un *Software* que no se imparten en el grado pero que pueden ser de mucha utilidad para un estadístico.

A continuación, se listan los objetivos específicos que se persiguen en este trabajo:

- Aprender a programar en *Python* y utilizar sus librerías, más específicamente las usadas para *Machine Learning*; librería Scikit-Learn, y *Deep Learning*; librería Keras.
- Analizar la utilidad de las metodologías propuestas; *multilayer perceptron neural network* y *recurrent neural network*, para predecir movimientos en precios de *stock* de las compañías que forman parte del índice S&P 500 únicamente a partir

de los históricos de estos y comparar el funcionamiento de estos algoritmos si es posible.

- A partir de los resultados obtenidos, ya sean positivos o negativos, analizar cuáles podrían ser las siguientes vías de actuación.

1.3. Metodología

La principal característica de este proyecto es el alto nivel de aprendizaje sobre nuevos conceptos que implica. Las técnicas aplicadas, el entorno usado, los datos o los campos de estas disciplinas eran conceptos que antes de empezar a realizar el trabajo me eran totalmente desconocidos. Es por esta razón que la primera fase (y realmente durante toda la duración del proyecto) del trabajo ha sido la búsqueda de recursos, sobre todo *online*, ya fuera desde tutoriales en *Python* a aplicación y funcionamiento de las metodologías de *Machine Learning* y *Deep Learning* propuestas. Todos los recursos empleados, aunque su aplicación no aparezca directamente en el proyecto, se encuentran en la bibliografía al final de la memoria.

A medida que se iban asimilando los conceptos, la segunda fase del trabajo ha sido la aplicación de los diferentes métodos aprendidos a conjuntos de datos reales. El propósito del trabajo ha sido que la aplicación pudiera ser tan automática como fuera posible, para ello, se ha automatizado la descarga de los datos también para no tener que descargar los 500 archivos *csv* uno por uno. Todo ello ha sido aplicado como se ha comentado en *Python* haciendo uso de sus librerías disponibles. En la memoria se han incluido únicamente los resultados finales encontrados. No se han incluido ninguno de los procedimientos no definitivos realizados durante las distintas iteraciones por las que ha pasado este.

Paralelamente a estas fases, se ha ido realizando la redacción de la presente memoria.

2. STATE OF THE ART

En el presente apartado detallamos por su importancia tres artículos recientes sobre que abarcan temas similares al tratado en este trabajo cuyos aspectos más relevantes son los siguientes:

- Las redes neuronales profundas parecen funcionar mejor que las redes neuronales poco profundas y las metodologías de *Machine Learning* para predecir retornos a un mes vista de los *stocks* del mercado japonés[1].

- Predecir movimientos en los retornos (clasificados como 1 sube y -1 baja) con un horizonte temporal corto como pueden ser días es muy complicado confirmando así la teoría del camino aleatorio[2].

- Las redes neuronales profundas parecen funcionar mejor a horizontes temporales entre uno y tres meses[2].

- Se necesitan bastantes *features* para que los modelos funcionen bien, ya que parece ser que la información que aportan los históricos de los precios es poca. Utilizar volúmenes y variables *dummies* para los días y meses ayuda a que los modelos funcionen mejor[2].

- Las redes neuronales *LSTM* parecen funcionar mejor que los algoritmos sin memoria como son las redes neuronales profundas, *random forest* o regresiones logísticas, aunque la aportación de información de los históricos de los precios a los modelos parece ser limitada como se observaba en los otros estudios[3].

[1] *Deep Learning for Forecasting Stock Returns in the Cross-Section, Masaya Abe and Hideki Nakayama, The University of Tokyo, Tokyo, Japan*, 2017.
[2] *Forecasting ETFs with Machine Learning Algorithms, Jim Kyung-Soo Liew and Boris Mayster, Johns Hopkins University,* 14 de Enero 2017.
[3] *Deep learning with short-term memory networks for financial market predictions, Thomas Fischer, Christopher Krauss, University of Erlangen-Nürnberg, Nürnberg, Germany,* 10 Mayo 2017.

3. METODOLOGÍAS EMPLEADAS

En el presente apartado, se presenta brevemente el funcionamiento de los algoritmos empleados en este proyecto

3.1. Machine learning

Antes de nada, tenemos que saber a qué se refiere todo el mundo cuando habla de *Machine Learning*, para ello, me parece muy comprensible la definición de Tom Mitchell, *Data scientist* y profesor en Carnegie Mellon University.

"Un programa de ordenador aprende de la experiencia E con respeto a unas clases de tareas T y medida de su *performance* P si su *performance* en la tarea T, medido por P mejora con la experiencia E."

Un ejemplo podría ser por ejemplo jugar al ajedrez, dónde:

E = La experiencia de jugar múltiples partidas de ajedrez

T = La tarea de jugar al ajedrez

P = La probabilidad de que el programa gane la próxima partida

Una de las aplicaciones del *Machine learning* son los modelos predictivos. A continuación, se resume brevemente en qué consisten los métodos usados en el trabajo y se describen algunos de los parámetros definibles para estos en las librerías de *Python* empleadas.

3.2. Deep learning

Para que un algoritmo de aprendizaje automático sea considerado *Deep Learning* debe tener alguna de las características siguientes:

- Usar múltiples capas con unidades de procesamiento no lineal para extraer y transformar variables. Cada capa usa la salida de la capa anterior como entrada. Las aplicaciones incluyen modelización de datos y reconocimiento de patrones.

- Estar basados en el aprendizaje de múltiples niveles de características o representaciones de datos. Las características de más alto nivel se derivan de las características de nivel inferior para formar una representación jerárquica.

- Aprender múltiples niveles de representación que corresponden con diferentes niveles de abstracción. Estos niveles forman una jerarquía de conceptos.

Todas estas características tienen en común referirse a múltiples capas de procesamiento no lineal. Donde éstas forman una jerarquía de características desde un nivel de abstracción más bajo a uno más alto. No existe un estándar para el número de capas que convierte un algoritmo en *deep*, pero la mayoría de investigadores en el campo considera que *deep learning* implica más de dos transformaciones intermedias.

A continuación, se definen brevemente los dos algoritmos usados; *Neural network* y *Recurrent Neural Network*.

3.2.1. Neural Network

Las *Artificial Neural Network* (Red Neuronal Artificial o *ANN*) son sistemas computaciones inspirados vagamente por las redes neuronales biológicas que constituyen los cerebros animales. Estos sistemas "aprenden" tareas considerando ejemplos, generalmente sin estar programados específicamente para esa tarea. Por ejemplo, en reconocimiento de imágenes, pueden aprender a identificar ejemplos manualmente etiquetados como "gato" o "no gato" y usar los resultados para identificar gatos en otras imágenes. Hacen esto sin ningún conocimiento a priori sobre gatos o sus características; pelo, colas, bigotes, etc. En vez de eso, evolucionan su propio conjunto relevante de características a partir del material de aprendizaje que procesan.

Para esto, al igual que un cerebro animal, una red neuronal está constituida por unidades o nodos conectados (neuronas artificiales). Cada conexión entre neuronas puede transmitir una señal de una a otra. La neurona que recibe la señal puede procesarla y entonces transmitir una señal a las neuronas conectadas a ella. Normalmente la señal de conexión entre neuronas artificiales es un número real, y el *output* de cada neurona es calculado como una función no lineal de la suma de los *inputs*. Las neuronas y conexiones tienen un *weight* (peso) asignado que se ajusta a medida que el aprendizaje avanza. El *weight* aumenta o disminuye la fuerza de la señal en una conexión.

Las redes neuronales están organizadas en capas, donde cada capa realiza diferentes transformaciones de su *input*. La señal viaja des de la primera capa (*input*),

hasta la última (*output*), pasando por todas las capas intermedias. A continuación, se puede ver un ejemplo de cómo podría ser una red neuronal con 4 *inputs* y un solo *output*, además de solo una capa intermedia con 5 neuronas.

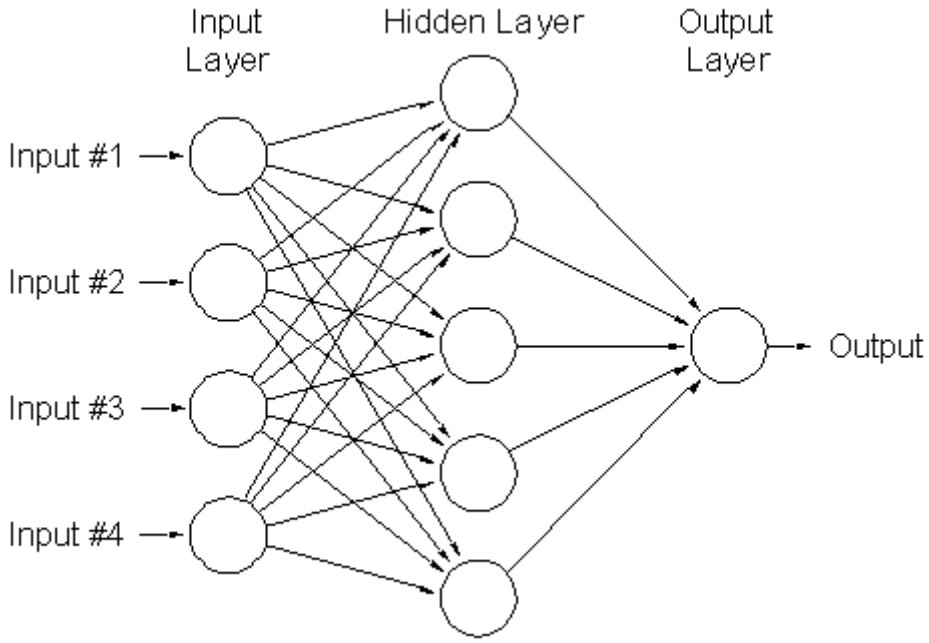


Figura 1: Ejemplo de red neuronal con 4 features y 1 output

Fuente: http://www.astroml.org/book_figures/appendix/fig_neural_network.html

Hay muchos tipos de redes neuronales artificiales, para este trabajo se ha decidido utilizar una red neuronal recursiva (explicada más adelante) y una red neuronal llamada perceptrón multicapa (*multilayer perceptron*).

3.2.2. Perceptrón multicapa

El perceptrón multicapa es una red neuronal artificial formada por múltiples capas. Las capas pueden ser de tres tipos:

- **Capa de entrada:**

Constituida por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento.

- **Capas ocultas:**

Formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.

- **Capa de salida:**

Neuronas cuyos valores de salida se corresponden con las salidas de toda la red.

Parámetros:

- **Número de capas:**

Para crear una red neuronal, primero se crea una red vacía y se le van añadiendo capas. La primera capa tiene que tener en cuenta cuantas dimensiones tiene el *input* (cuantas variables tenemos) y la última tiene siempre el mismo número de neuronas que nuestro *output* deseado (por ejemplo si el resultado es binario, tendrá dos neuronas). Es una práctica habitual hacer que la red se ensanche primero para luego estrecharse (creando una forma parecida a un rombo).

- **Neuronas por capa:**

A continuación, se deben definir también cuantas neuronas tiene cada capa. Como se ha comentado, es costumbre hacer que la red se ensanche para luego estrecharse y las neuronas de la primera y última capa vienen definidas por nuestros *features* y *labels*.

- **Epochs:**

Epochs hace referencia a cuantas veces se va a mirar el *training set*. Cuanto mayor sea, más tenderemos a tener *overfitting*.

- **Batch_size:**

Este parámetro refiere al tamaño de las sub-muestras generadas para calcular el vector gradiente y optimizar el modelo.

3.2.3. Recurrent Neural Networks[4]

Las Redes neuronales recurrentes (*RNN*) son un tipo de redes neuronales diseñada para reconocer patrones en secuencias de datos como pueden ser textos, genomas, escritura, palabra hablada o series temporales numéricas. Son una de las más potentes redes neuronales, siendo aplicables también en imágenes, descomponiéndoles en trozos y tratándolos como secuencias.

En las redes neuronales recurrentes, el *input* no es solamente el ejemplo de *input* que ve en ese momento, sino que también lo es lo que ha percibido anteriormente en el tiempo, es decir, como se puede ver en el diagrama, *BTSXVPE* es el *input* en ese momento y *CONTEXT UNITS* representa el *output* de la iteración anterior.

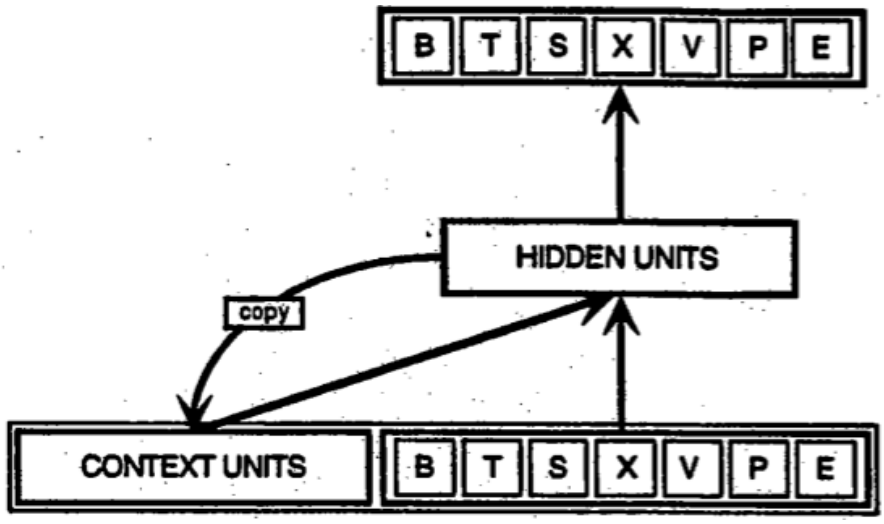


Figura 2: Diagrama de una Red recurrente sin desenvolver

Fuente: https://deeplearning4j.org/lstm.html

Por lo tanto, la decisión del paso temporal *t-1* afecta a la decisión tomada en el paso temporal *t*, es por eso por lo que se dice que las redes neuronales recurrentes tienen memoria.

Por ejemplo, si se tuviera que clasificar que tipo de evento está pasando en cada momento de una película, no está claro como una red neuronal tradicional podría usar su razonamiento sobre eventos anteriores en la película para informar eventos

4 Este apartado, así como el siguiente se inspiran en la información del paper *Conditional Image Synthesis With Auxiliary Classifier GANs*, Augustus Odena, Cristopher Olah, Jonathon Shlens, 20 de Julio de 2017, así como la información del mismo Cristopher Olah en su blog sobre redes neuronales recurrentes.

posteriores. Mientras que las redes neuronales recurrentes pueden hacerlo gracias a que tienen bucles; permitiendo que la información persista.

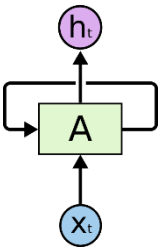


Figura 3: Red neuronal recurrente sin desenvolver

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

La forma de bucle de la red neuronal es totalmente equivalente a tener distintas redes neuronales, una para cada paso temporal *t* donde se permite que la información del instante *t-1* sea transmitida a la red neuronal que computa el instante *t*. El diagrama inferior ilustra cómo se representaría la red neuronal recurrente si se deshiciera el bucle.

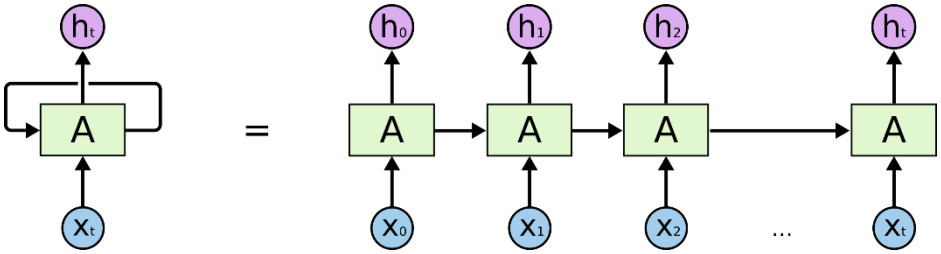


Figura 4: Red neuronal recurrente expandida

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

En los últimos años, la popularidad de las redes neuronales recurrentes ha ido en aumento sobretodo en aplicaciones como son *speech recognition*, *language modeling* o *image captioning*. Ante esta creciente popularidad, también se han aplicado en otros ámbitos como pueden son las series temporales. Una de las redes neuronales recurrentes más populares; y la usada en este trabajo, es la *Long Short-Term Memory* (*LSTM*). Se trata de una de las más populares porque es capaz de asimilar dependencias *Long-Term* que redes neuronales recurrentes de otros tipos no son capaces de asimilar.

Para entender que es una dependencia *Long-Term* primero veamos que es una dependencia *Short-Term*. Consideremos que tenemos un modelo que intenta predecir cuál va a ser la siguiente palabra de una frase y estamos intentando predecir la palabra *francés* de la frase "En Francia la lengua oficial es el *francés*". En este caso, no se necesita mucho contexto para saber que la palabra va a ser *francés*, se dice entonces que el *gap* entre la información relevante (Francia, lengua oficial) y el lugar donde se necesita es

pequeño. Una red neuronal recurrente cualquiera es capaz de aprender estas relaciones donde el *gap* es pequeño utilizando la información pasada.

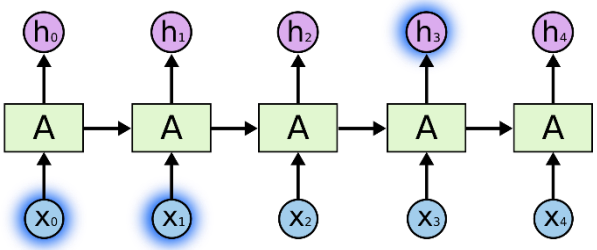


Figura 5: Ejemplo de gap pequeño

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Veamos ahora el caso donde la dependencia es *Long-Term*, es decir, un caso en que el *gap* entre la información relevante y el lugar donde se necesita es grande. Si de nuevo tenemos el modelo que predice palabras, pero ahora el texto es: "Crecí en Francia… Hablo *francés*", donde los puntos suspensivos reflejan otro texto entremedio, se puede ver como la distancia (*gap*) es mayor. Cuanto mayor es esta distancia, más dificultades tienen las redes neuronales recurrentes para conectar la información. Aun así, en teoría las redes neuronales recurrentes son capaces de aprender estas dependencias *Long-Term*, aunque a veces no sea así. Ante este problema, aparecieron las *LSTM*, que no tienen este problema ya que están diseñadas específicamente para poder aprender de estas relaciones *Long-Term*.

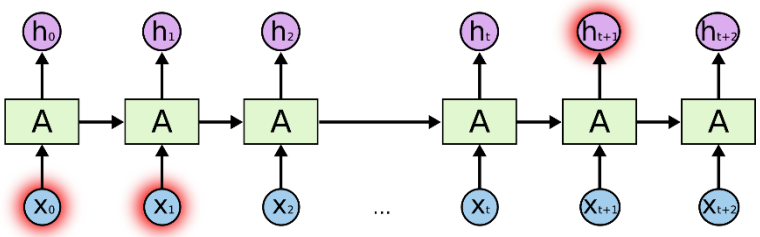


Figura 6: Ejemplo de gap grande

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

3.2.4. Long Short-Term Memory Recurrent Neural Networks

Para entender bien el funcionamiento de las *LSTM*, es importante analizar su estructura, en el diagrama presentado a continuación, se representa la forma de cadena de una *LSTM* y los vectores y operaciones que contiene, donde cada línea representa un

vector que conecta el *output* del instante *t-1* con el *input* del instante *t*, cada circulo rosa representa una operación entre vectores y cada caja amarilla una capa de la red neuronal donde la información pasa por una función.

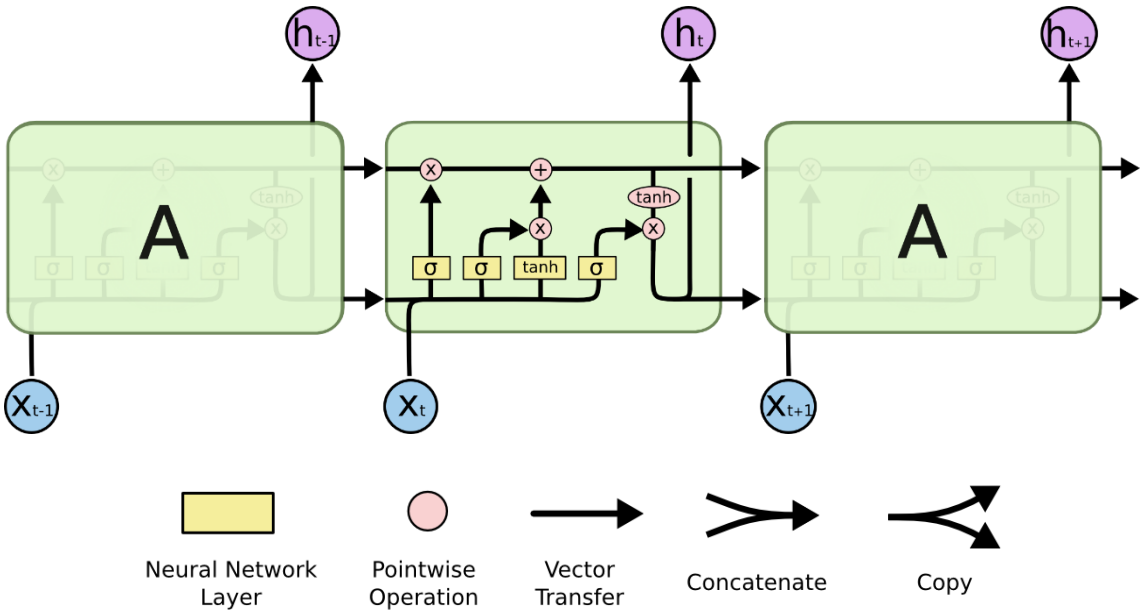


Figura 7: Módulo de una LSTM con sus cuatro capas

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Hay dos componentes de gran importancia dentro de la estructura de la *LSTM*, estas son el *cell state* y las *gates*. El *cell state* es el vector que recorre la parte superior de la *LSTM* en el diagrama y su importancia viene dada porque transporta la mayoría de la información entre iteraciones. La información del *cell state* es alterada únicamente por interacciones lineares, por lo que al no sufrir ninguna transformación mayor la información nunca es alterada de gran manera en una sola iteración, permitiéndole recordar información del pasado con más facilidad.

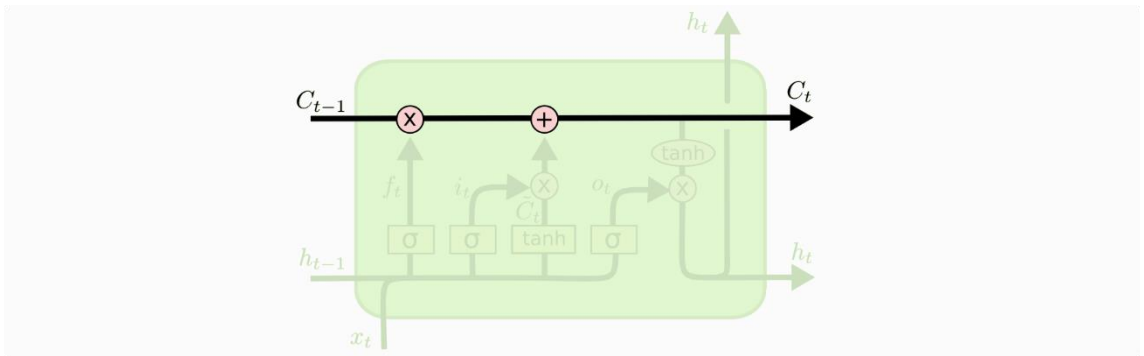


Figura 8: Cell state de una LSTM

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

El otro componente de vital importancia en el funcionamiento de la *LSTM* son las *gates* (puertas). Las *gates* están compuestas de una función (normalmente una sigmoide) y una operación puntual entre vectores. Como bien dice su nombre, funcionan como puertas de entrada para la información, donde la información de un vector pasaría por la capa sigmoidea produciendo un número entre 0 y 1, siendo 0 igual a "toda esta información es irrelevante" o 1 "toda esta información es relevante". En función entonces de la importancia que esta capa sigmoidea da a la información del vector esta pasa después al *cell state* a través de una operación entre vectores.

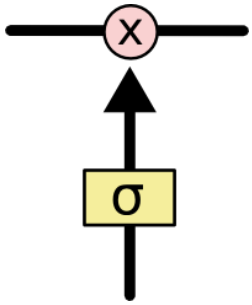
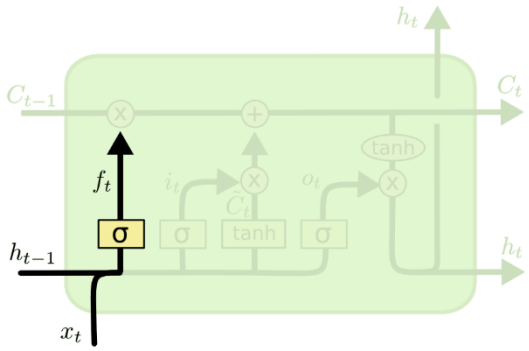


Figura 9: Representación de una gate

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Hay distintos tipos de *gate* en una *LSTM*, el primer tipo es la llamada *forget gate layer*. Como bien indica su nombre, esta es la encargada de decidir qué información se olvida de la iteración anterior, es decir, se trata de una capa sigmoidea que en la iteración t, tiene en cuenta la información h_{t-1} (que se corresponde al *output* de la iteración anterior) y x_t (*input* de la presente iteración t) y decide qué información es relevante. Eso lo hace pasando esta información por una sigmoide que genera un valor entre 0 y 1 para cada elemento del vector de información del *cell state* C_{t-1}. Como se puede ver en el diagrama inferior, el vector que sale de la sigmoide pasa entonces a multiplicar el *cell state*, haciendo que olvide aquella información que se ha multiplicado por un 0 y que recuerde la otra en menor o mayor medida. En el diagrama inferior se puede observar también la función de la *forget layer*. Dónde σ representa la función sigmoidea, W_f son los *weights* (pesos) y $b_f$el *bias* (sesgo).

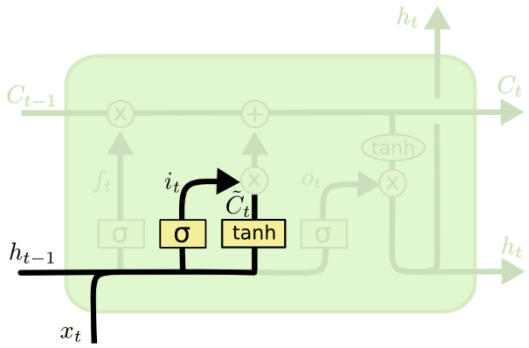


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 10: Representación de la forget gate layer

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Otros dos tipos de las *gates* que hay dentro de una *LSTM* son la *input gate layer* y la *tanh layer*. Estas dos son los componentes de la *LSTM* encargados de decidir qué información nueva es guardada en el *cell state*. La *input gate layer* (i_t en el diagrama) es una capa sigmoidea encargada de decidir que valores van a ser actualizados, mientras que la capa tanh (una tangente hiperbòlica) crea un vector $\tilde{C}_t$ que contiene los candidatos a ser añadidos al *cell State*.

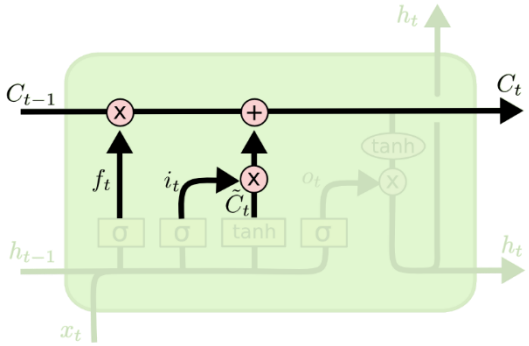


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 11: Input gate layer y tanh layer en una LSTM

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Con la información resultante i_t y $\tilde{C}_t$ se actualiza el *cell State* de la iteración anterior al de esta. Recordemos que hasta ahora al *cell State* únicamente se le había aplicado el producto resultado de la *forget gate layer*, por lo que seguía teniendo únicamente la información de *t-1* habiendo olvidado aquella que era considerada irrelevante. A este *cell state* C_{t-1} se le suma ahora el producto $i_t * \tilde{C}_t$ como puede verse a continuación en el diagrama. Tras esta transformación, ya se tiene el *cell state* correspondiente a la iteración *t*, C_t , que es el que se transmitirá a la siguiente iteración.

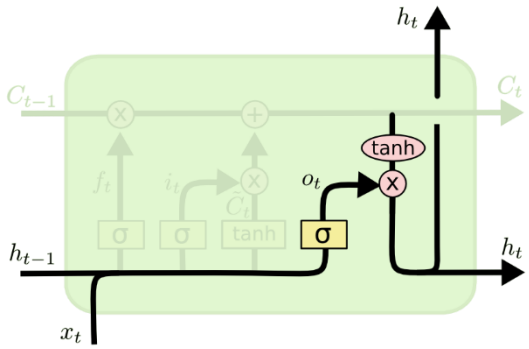


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 12: Actualización del cell state

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Para acabar, solo falta saber cómo se obtiene el *output* de esta iteración, es decir, h_t. Éste es el resultado de parte de la información deseada del *cell state* y parte de la información resultante del *input* de la iteración presente y el *output* de la anterior. Como se puede ver en el diagrama, la información del *cell state* pasa por una hipertangente transformando todos los valores de este en valores entre -1 y 1, y lo multiplica por el resultado de la sigmoide con información del *output* pasado y el *input* presente. Esta información pasará a ser el *output* de la iteración *t*, que será luego usado como *input* en la iteración *t+1*.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 13: Cálculo del output de una LSTM

Fuente: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Parámetros:

- **Hidden_nodes:**

Refiere al número de neuronas de la LSTM, a más neuronas más potencia tiene la red neuronal, aunque también aumenta el número de parámetros a aprender y el tiempo para entrenar la red.

- **Timesteps:**

Número de pasos temporales que se quieren considerar. Por ejemplo, si se quisiera clasificar una frase sería el número de palabras en esta.

- **Input_dim:**

Las dimensiones de los *features*, número de variables a considerar.

4. DESCRIPCIÓN DEL PROGRAMA Y LAS LIBRERÍAS EMPLEADAS

Para empezar con el cuerpo del trabajo, primero se va a explicar que es *Python* y que lo hace tan útil para aplicaciones como pueden ser entrenar modelos predictivos. También se describirán las librerías empleadas para este proyecto.

4.1. Python

Python es un lenguaje de programación orientado a objetos (que permite la definición de tipos de datos, de operaciones nuevas sobre esos tipos de datos y de instanciar el tipo de datos), cuya filosofía hace hincapié en una sintaxis que favorezca un código legible, lo cual ha ayudado a que pudiera aprenderlo rápidamente habiendo trabajado únicamente con R hasta entonces.

Python fue creado a finales de los ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática en los Países Bajos. En 1991, van Rossum publicó el código de la versión 0.9.0 que ya incluía entre otras funciones y tipos modulares como vectores, listas o diccionarios entre otros.

A partir de 1995, van Rossum lanzó la iniciativa *Computer Programming for Everybody* (CP4E), con el fin de hacer la programación más accesible a más gente, con un nivel de "alfabetización" básico en lenguajes de programación. *Python* tuvo un papel crucial en este proceso debido a su orientación hacia una sintaxis limpia. Des de 2007 el proyecto CP4E está inactivo, aunque *Python* sigue intentando ser fácil de aprender y no muy arcano en su sintaxis y semántica para alcanzar a no programadores. A día de hoy *Python* ya va por su versión 3.6.5.

Python es un lenguaje de programación que permite a los programadores adoptar varios estilos: programación orientada a objetos, programación imperativa o programación funcional, lo que se conoce como paradigma múltiple.

Una característica importante de *Python* es la resolución dinámica de nombres; es decir, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado enlace dinámico de métodos). Otra de sus características más importantes es que se trata de un *open-source code* (código abierto), es decir, se trata de un modelo de *software* basado en la colaboración abierta. Esto permite entre otras cosas que terceros puedan crear librerías con funciones implementadas que nos permiten por ejemplo aplicar algoritmos de *Machine* o *Deep Learning* sin tener que programar el algoritmo en sí. A continuación, se van a describir las librerías usadas durante el proyecto.

4.2. Librerías empleadas

4.2.1. Pandas

Pandas es una librería de *software* escrita para el lenguaje *Python* para manipulación y análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. *Pandas* permite que *Python,* que ya era un lenguaje muy válido para preparación de datos, pase a también ser muy válido para el análisis, permite realizar todo el flujo de trabajo en *Python* sin tener que cambiar a un lenguaje como R.

Entre sus características se encuentran:

- Objetos *Dataframe* para manipulación de datos con índices integrados.
- Herramientas para leer y escribir datos de diferentes tipos de archivos.
- Tratamiento de *missings*.
- Inserción y eliminación de columnas de datos.
- Funcionalidades para series temporales: generación de rangos de fechas, estadísticas *moving Windows*, *lagging* y *shifting* de fechas.
- *Merge* y *join* de bases de datos.

4.2.2. Numpy

Numpy es una extensión de *Python* que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas para operar con esos vectores o matrices.

Numpy le da a *Python* unas funcionalidades comparables a MATLAB al poder trabajar con operaciones de vectores o matrices. En nuestro caso, *Numpy* también nos sirve para guardar objetos en forma de vector y poder cargarlos después en otros programas.

4.2.3. Beautiful Soup

Beautiful Soup es una biblioteca para analizar documentos HTML. Esta crea un árbol con todos los elementos del documento y puede ser utilizado para extraer información, convirtiéndola en muy útil para hacer *web scraping* (de lo que hablaremos más adelante)

4.2.4. Scikit-Learn

Scikit-Learn es una librería de *Python* que incluye varios algoritmos (de *Machine Learning*) de clasificación, regresión y *clustering* como son por ejemplo *support vector machines*, *random forests* o *gradient boosting*.

Todos los algoritmos disponibles contienen una documentación con todos los parámetros y opciones para optimizar el funcionamiento de este. *Scikit-Learn* funciona como una comunidad, cuando algún usuario descubre un error o alguna parte a corregir puede colgarlo en sus foros para que ellos lo revisen y lo corrijan. Al igual que si alguien diseña un nuevo código que desearía incluir en la librería, ellos lo revisan y deciden si introducirlo. A cambio de usar sus librerías solo piden ser citados en el trabajo.

4.2.5. Keras

De similar manera a como *Scikit-Learn* proporciona algoritmos de *Machine Learning*, *Keras* ofrece algoritmos para poder emplear redes neuronales. *Keras* no funciona de manera autónoma, sino que corre encima de *Tensorflow* de Google (des de 2017), convirtiéndose en una especie de interfaz.

Tensorflow es una biblioteca de código abierto para *Machine Learning* desarrollada por Google para construir y entrenar redes neuronales.

Keras está construido sobre cuatro principios:

- *User friendliness*:

Keras es una API que pone por delante la facilidad de uso para el usuario. Minimiza el número de acciones requeridas por el usuario y le proporciona *feedback* cuando se produce un error.

- Modularidad:

Los modelos son comprendidos como secuencias de módulos totalmente configurables que se pueden conectar juntos con las mínimas restricciones posibles. En concreto, capas de neuronas, funciones de coste, optimizadores y funciones de activación entre otros son módulos que cuando se combinan crean nuevos modelos.

- Fácil extensión:

Se pueden añadir muy fácilmente nuevos módulos y sobre los ya existentes hay muchos ejemplos. La capacidad de añadir nuevos módulos hace que *Keras* sea también apto para la investigación avanzada.

- Funciona en *Python*:

Todos los modelos están descritos en código de *Python*, lo que hace que sean compactos, fáciles de corregir y permite fácilmente la extensión de estos.

4.3. Pycharm

Para poder programar usando código *Python* se necesita un entorno. Uno de los más usados y el que se ha empleado en este trabajo es *Pycharm*. Entre sus características se encuentran:

- Asistencia y análisis de código:

Con herramientas como sugerencias para complementación de código o destacando errores de sintaxis. Esto permite que aunque el usuario sea relativamente nuevo programando en ese lenguaje no tenga que estar constantemente buscando sus propios errores.

- Navegación de proyectos y código:

Vistas de proyecto especializadas, vistas de estructuras de archivos o facilidad para saltar entre archivos, clases y métodos.

- *Python refactoring*:

Incluyendo renombramiento, métodos de extracción, introducir variables y constantes entre otros.

5. DATOS

La intención del estudio era ver si a partir únicamente de las series temporales de los precios de cierre de los stocks de las empresas que forman parte del índice S&P 500 se podía predecir el movimiento del precio. Se han usado los 500 históricos para predecir los movimientos en uno de ellos. La teoría tras esto, era que en base a lo encontrado en los estudios mencionados en el apartado de *State of the art*, los resultados mejoran cuantos más históricos se usen, ya que cada uno aporta poca información. La empresa escogida para ser predicha ha sido EBAY, aunque podría haber sido cualquier otra del índice que estuviera presente en este des del año 2000, de hecho se probó la metodología en varias aunque no se observó variación en el resultado (más sobre esto en el apartado de conclusiones).

5.1. Standard & Poor's

Standard & Poor's Financial Services LLC (S&P) es una empresa estadounidense de servicios financieros que publican informes sobre investigación financiera y análisis de acciones y bonos. S&P también publica un gran número de índices bursátiles que abarca todas las regiones del mundo, nivel de capitalización de mercado y tipo de inversión. Entre estos índices está el que nos interesa; el S&P500.

5.1.1. S&P 500

El S&P 500 es uno de los índices bursátiles más importantes de los Estados Unidos, siendo incluso considerado el índice más representativo de la situación real del mercado.

El índice se basa en la capitalización bursátil de 500 grandes empresas que poseen acciones que cotizan en las bolsas NYSE o NASDAQ. Las empresas son seleccionadas por un comité de forma que sean representativas de las industrias que operan en la economía de los Estados Unidos. Para ser agregada al índice, una empresa debe satisfacer los siguientes requerimientos de liquidez y tamaño:

- La capitalización bursátil debe ser igual o mayor a 4000 millones de dólares norteamericanos.
- La relación entre el monto anual en dólares negociado a la capitalización bursátil ajustada debe ser superior a 1.0.
- El volumen de acciones negociado mensualmente debe por lo menos ser de 250000 acciones en cada uno de los seis meses previos a la fecha de evaluación.

Entre las empresas que forman parte del índice se encuentras por ejemplo American Express Co, Amazon.com Inc., The Coca-Cola Company o eBay Inc.

5.2. Web scraping

Al necesitar 500 archivos *csv*, uno para cada empresa del índice, descargarlos uno a uno hubiera sido una tarea muy larga y pesada, por lo que me pareció un buen momento para aprovechar y aprender sobre *web scraping* en *Python*. El *web scraping* es el proceso de recopilar información de forma automática de la Web, en otras palabras, nos permite automatizar la extracción de datos de internet. A través de pocas líneas de código, podemos realizar una búsqueda y extraer la información deseada sin hacer nada manualmente. El proceso de *web scraping* de una página web incluye dos subprocesos; búsqueda y extracción. La búsqueda es la descarga de una página (lo que hace por ejemplo Google Chrome) para ser procesada luego. Cuando se ha completado este paso, comienza la extracción. La extracción es el análisis, búsqueda y reformateo del contenido de la página, para luego ser copiado en una hoja de cálculo.

5.3. Obtención de datos

Una vez explicado el concepto de *web scraping* y qué datos se querían conseguir, solo hacía falta poner las dos ideas en común. A través de una sub librería de *Pandas*, llamada *Pandas datareader*, somos capaces de extraer de https://www.quandl.com/ los históricos de los precios en un archivo *csv* mediante solo una línea de código. Ahora bien, hace falta darle los *tickers* (código bursátil que identifica una empresa que cotiza en un mercado bursátil) de las 500 empresas que deseamos. Si tenemos que escribir los 500 *tickers*, resulta poco práctico, por lo que decidí mediante *web scraping* obtener estos.

Utilizando la librería descrita con anterioridad; *Beautiful Soup*, nos dirigimos a la web de Wikipedia con la lista de los *tickers*, en formato tabla como se puede ver a continuación:

S&P 500 Component Stocks [edit]

Ticker symbol	Security	SEC filings	GICS Sector	GICS Sub Industry	Location	Date first added[3][4]	CIK	Founded
MMM	3M Company	reports	Industrials	Industrial Conglomerates	St. Paul, Minnesota		0000066740	1902
ABT	Abbott Laboratories	reports	Health Care	Health Care Equipment	North Chicago, Illinois	1964-03-31	0000001800	1888
ABBV	AbbVie Inc.	reports	Health Care	Pharmaceuticals	North Chicago, Illinois	2012-12-31	0001551152	2013 (1888)
ACN	Accenture plc	reports	Information Technology	IT Consulting & Other Services	Dublin, Ireland	2011-07-06	0001467373	1989

Figura 14: Muestra de la tabla que contiene los tickers de S&P 500

Fuente: https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

A continuación abrimos el origen de datos de la página y buscamos donde se encuentra el código que define esta tabla. Dentro podemos encontrar como se definen los *tickers* como se puede ver:

```
<tr>
<td><a rel="nofollow" class="external text" href="https://www.nyse.com/quote/XNYS:ABBV">ABBV</a></td>
<td><a href="/wiki/AbbVie_Inc." title="AbbVie Inc.">AbbVie Inc.</a></td>
<td><a rel="nofollow" class="external text" href="https://www.sec.gov/cgi-bin/browse-edgar?CIK=ABBV&amp;action=getcompany">reports</a></td>
<td>Health Care</td>
<td>Pharmaceuticals</td>
<td><a href="/wiki/North_Chicago,_Illinois" title="North Chicago, Illinois">North Chicago, Illinois</a></td>
<td>2012-12-31</td>
<td>0001551152</td>
<td>2013 (1888)</td>
</tr>
<tr>
<td><a rel="nofollow" class="external text" href="https://www.nyse.com/quote/XNYS:ACN">ACN</a></td>
<td><a href="/wiki/Accenture_plc" class="mw-redirect" title="Accenture plc">Accenture plc</a></td>
<td><a rel="nofollow" class="external text" href="https://www.sec.gov/cgi-bin/browse-edgar?CIK=ACN&amp;action=getcompany">reports</a></td>
<td>Information Technology</td>
<td>IT Consulting &amp;amp; Other Services</td>
<td><a href="/wiki/Dublin,_Ireland" class="mw-redirect" title="Dublin, Ireland">Dublin, Ireland</a></td>
<td>2011-07-06</td>
<td>0001467373</td>
<td>1989</td>
</tr>
```

Figura 15: Muestra del código que define la tabla que contiene los tickers de S&P 500

Fuente: https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

Descargamos el código y entonces podemos buscar en este mediante un bucle todos los *tickers*. Guardamos estos en un vector que guardamos como un archivo en nuestro ordenador. Esto permite que obtengamos de manera cómoda y rápida los 500 *tickers* para luego guardarlos y ya no tener que preocuparnos más de esto.

Una vez tenemos los *tickers*, mediante una función de *Pandas* podemos descargar de https://www.quandl.com/ los históricos de los precios. Para este trabajo se han descargado primero los datos des del 1 de enero de 2000 hasta el 31 de diciembre de 2016 para usar para entrenar los modelos y validarlos. Después se han descargado los datos des del 1 de Enero de 2017 hasta el 31 de diciembre de 2017 para hacer el *test* de estos.

Los archivos *csv* descargados contienen los precios de apertura, volúmenes intercambiados ese día, precios de cierre, máximos y mínimos del día, etc. Para este trabajo se ha usado solo el precio ajustado de cierre de cada una de las empresas y se

ha juntado todo en un solo archivo *csv* que se ha guardado para no tener que repetir todo este proceso.

Una vez se tienen todos los datos, se ha hecho un gráfico con las correlaciones entre los precios de cierre ajustados de las distintas empresas. Aunque el eje con las empresas no se puede ver (porque hay demasiadas), está claro que hay una mayoría de correlaciones positivas y fuertes entre estas. Es importante recordar que correlación no implica causalidad, por lo que la existencia de correlación no implica tampoco nada más. Por ejemplo, si miráramos los precios el 11 de Setiembre de 2001 (el día del atentado del *World trade center*), todos los precios bajan, pero es a causa de un factor externo a los precios.

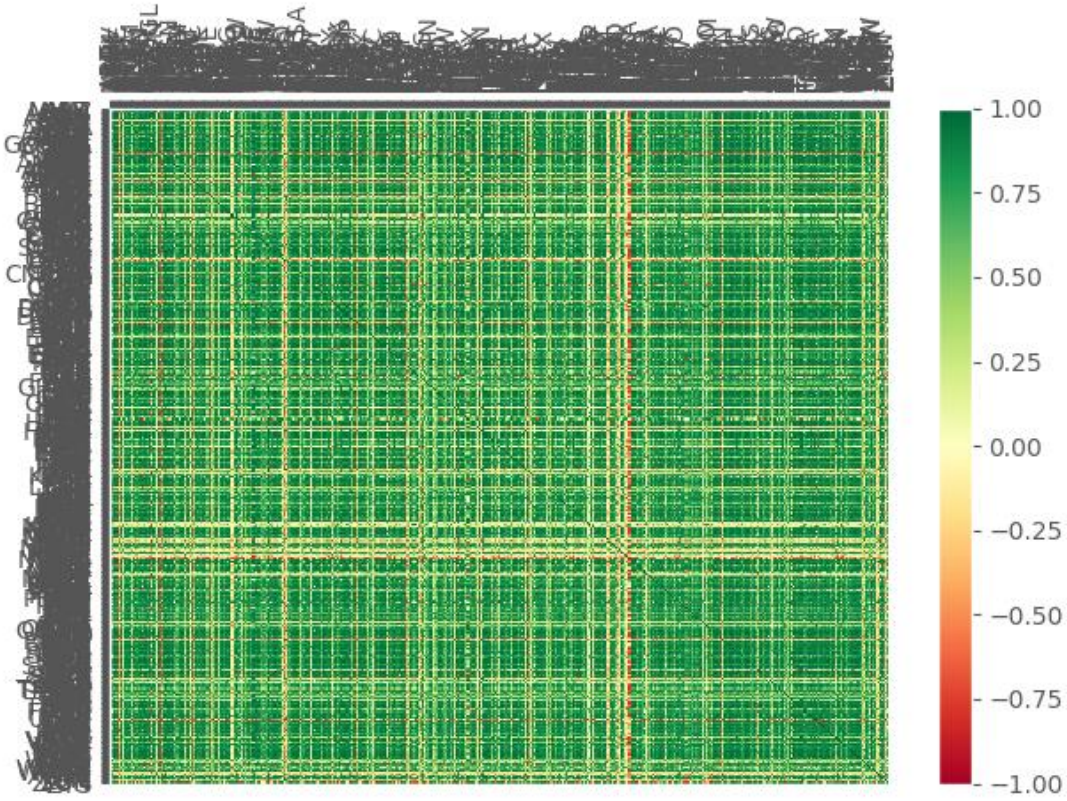


Figura 16: Gráfico de correlaciones entre las 500 empresas del índice S&P 500

6. DEFINICIÓN DEL *LABEL* (VARIABLE DEPENDIENTE)

Para que pudiera asimilarse el funcionamiento de los modelos a un sistema de compra y venta, se decidió discretizar la variable objetivo (el *label*), en otras palabras, si el precio subirá se compra (codificado como un 1), si el precio bajará se vende (un -1) y si se considera que el precio no se mueve lo suficiente se mantiene la posición actual (un 0), de tal manera se evita hacer demasiadas operaciones, pero las que se hacen suelen ser más "seguras".

6.1. Franja temporal

Para decidir cuánto tiempo a futura se desearía mirar, se estudió que habían hecho otros casos similares publicados. En ellos se comentaba como escoger una franja temporal demasiado grande acababa siendo un error, ya que el precio se ve afectado por factores muy ajenos a los precios, mientras que cerrar una ventana temporal demasiada pequeña (1 día por ejemplo) podía dificultar también mucho la predicción por la teoría del camino aleatorio.

A partir de esta información, se decidió entonces que la franja empleada sería de 15 días. Es necesario saber que aunque no se comenten en el trabajo más resultados que los obtenidos durante la última ejecución, se han obtenido modelos para distintas franjas para ver si los resultados mejoraban a medida que el trabajo ha evolucionado[5].

6.2. Bandas de Bollinger

Las bandas de Bollinger son unos indicadores utilizados en el análisis técnico de los mercados financieros introducidos por John Bollinger en los años 80.

La representación gráfica de las bandas de Bollinger son dos curvas que envuelven el gráfico de precios. Estas curvas se calculan a partir de una media móvil (normalmente exponencial aunque a veces simple) sobre el precio de cierre y las desviaciones estándar de este. La práctica común es dibujar las bandas a 2 desviaciones estándar. A continuación se puede ver un ejemplo de las bandas de Bollinger a medida que el precio evoluciona en el tiempo:

[5] Durante la realización del Trabajo se emplearon las metodologías en distintos horizontes temporales; 1 día, 7 días y 30 días, obteniendo peores resultados.

Figura 17: Ejemplo de bandas de Bollinger

Fuente: https://vladimirribakov.com/bollinger-bands-ultimate-guide-part1/

Los valores por defecto que suelen utilizarse para su cálculo son, como se ha comentado, 2 desviaciones estándar y 20 días. Si se reduce o incrementa de forma significativa el valor de la media, hay que ajustar de similar manera el de las desviaciones estándar. Para valores de la media por encima de 50 (largo plazo) suele usarse 2.5 desviaciones, mientras que para valores de la media cercanos a 10 (corto plazo) suele usarse 1.5 desviaciones.

Cuando los precios están por encima de la media y cercanos a la banda superior están relativamente altos, se considera que puede haber sobrecompra. Si están por debajo de la media y cercanos a la banda inferior los precios están relativamente bajos y se considera que puede haber sobreventa.

6.3. Creación de la variable *label*

Una vez se ha entendido como se construyen las bandas de Bollinger, para construir la variable *label* se ha hecho lo siguiente:

1) Definir la media móvil a 15 días sobre el precio de cierre ajustado.
2) Definir las desviaciones estándar sobre el precio de cierre ajustado.
3) Desplazar las bandas 15 días atrás, ya que tenemos que comparar el instante t donde estamos con la banda en t+15.

4) Hacer un recorrido por todo el vector de precios, marcando como un -1 cuando el precio está por encima y un 1 cuando está por debajo, ya que significa que en ese momento el precio está más alto (o bajo si es 1) de lo que debería.

Una vez aplicada esta metodología, se obtienen los *spreads* en la muestra. El *spread* es sencillamente cuantas observaciones de cada clase tenemos. En un mundo ideal, cuando se entrena un modelo de clasificación, se desea tener un *spread* balanceado, lo que viene a querer decir que si se tienen 3 clases, como en este caso, se desearía tener un tercio de la muestra en cada clase.

Uno de los primeros problemas encontrados al utilizar esta metodología para definir el *label* (problema en el que se profundizará más adelante) es que las Bandas de Bollinger dejan una minoría de observaciones por encima y por debajo de las colas, en principio para minimizar el número de acciones (de comprar o vender), lo que ha supuesto que la muestra siempre estaba poco balanceada. Es por eso que se calculó para distintos valores de las diferencias de la desviación estándar como quedaba el *spread*, a continuación los resultados obtenidos:

- Para 2 desviaciones estándar por encima y debajo de la media móvil:

Vender (-1)	Mantener (0)	Comprar (1)
453	3456	339

Tabla 1: Spread para 2 desviaciones típicas

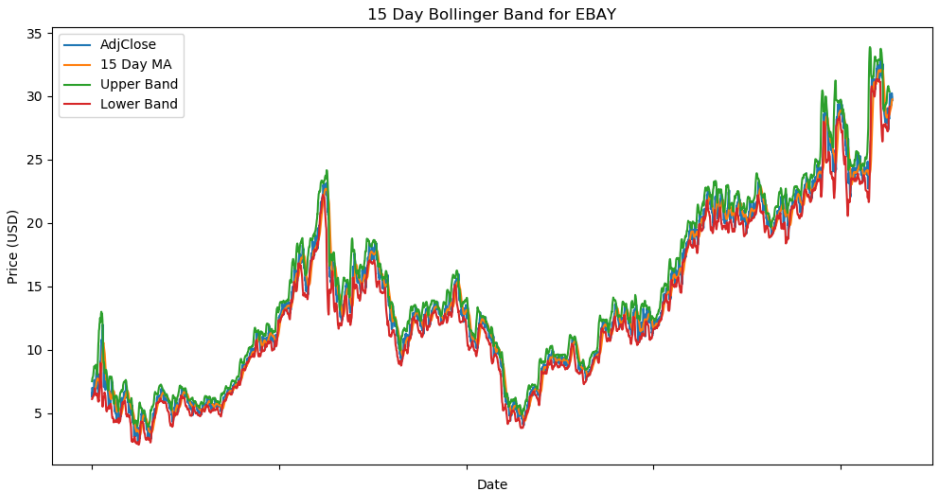


Figura 18: Evolución de las Bandas de Bollinger del precio del stock de EBAY para 2 desviaciones típicas

A continuación se puede ver una sub-muestra de este gráfico donde se pueden ver las bandas de Bollinger claramente.

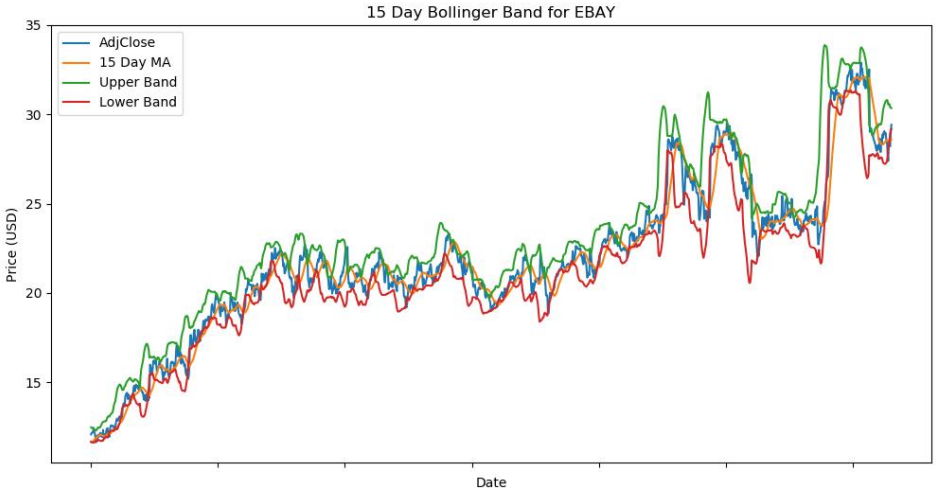


Figura 19: Bandas de Bollinger para 2 desviaciones típicas y una muestra reducida de los datos

- Para 1.5 desviaciones estándar por encima y debajo de la media móvil:

Vender (-1)	Mantener (0)	Comprar (1)
799	2795	654

Tabla 2: Spread para 1.5 desviaciones típicas

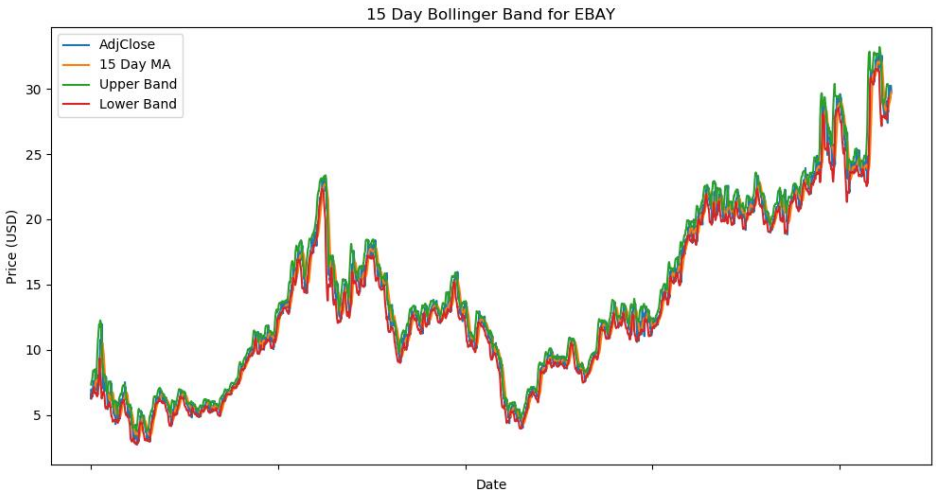


Figura 20: Evolución de las Bandas de Bollinger del precio del stock de EBAY para 1.5 desviaciones típicas

- Para 1.25 desviación estándar por encima y debajo de la media móvil:

Vender (-1)	Mantener (0)	Comprar (1)
1055	2337	856

Tabla 3: Spread para 1.25 desviaciones típicas

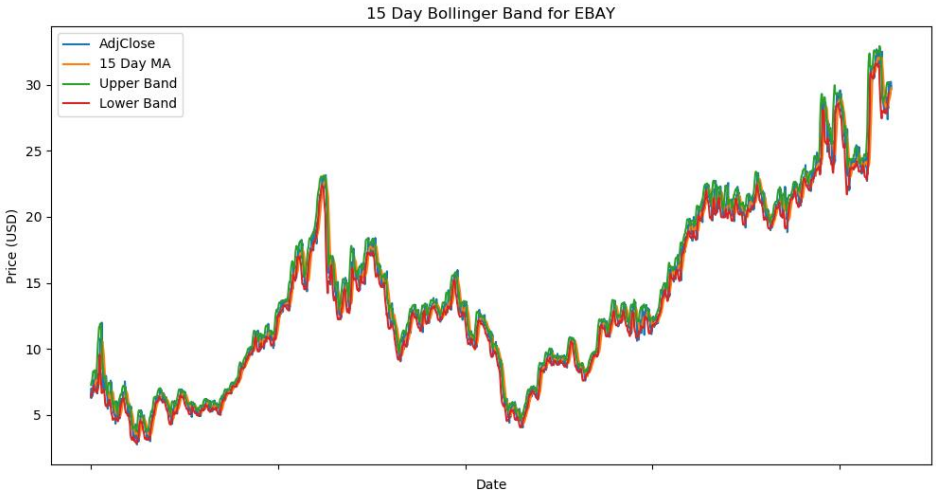


Figura 21: Evolución de las Bandas de Bollinger del precio del stock de EBAY para 1.25 desviaciones típicas

- Para 1 desviación estándar por encima y debajo de la media móvil:

Vender (-1)	Mantener (0)	Comprar (1)
1308	1837	1103

Tabla 4: Spread para 1 desviación típica

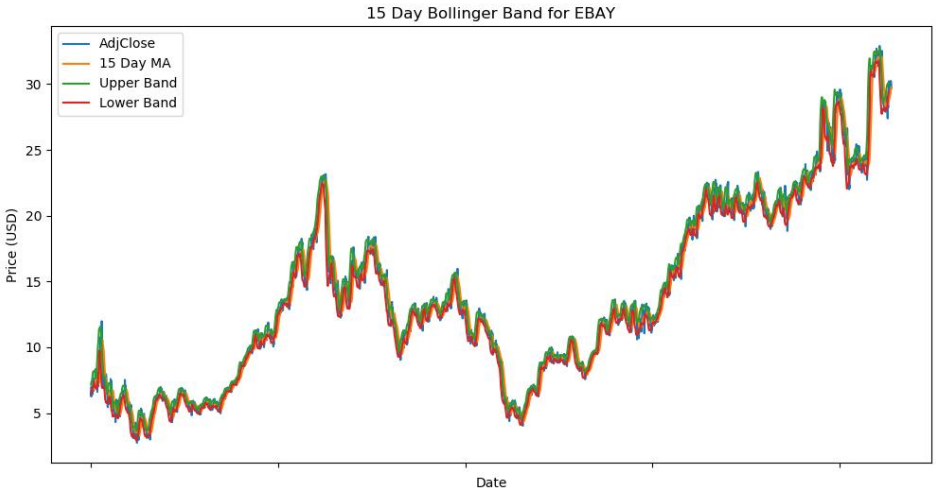


Figura 22: Evolución de las Bandas de Bollinger del precio del stock de EBAY para 1 desviación típica

- Para 0.5 desviación estándar por encima y debajo de la media móvil:

Vender (-1)	Mantener (0)	Comprar (1)
1820	846	1582

Tabla 5: Spread para 0.5 desviaciones típicas

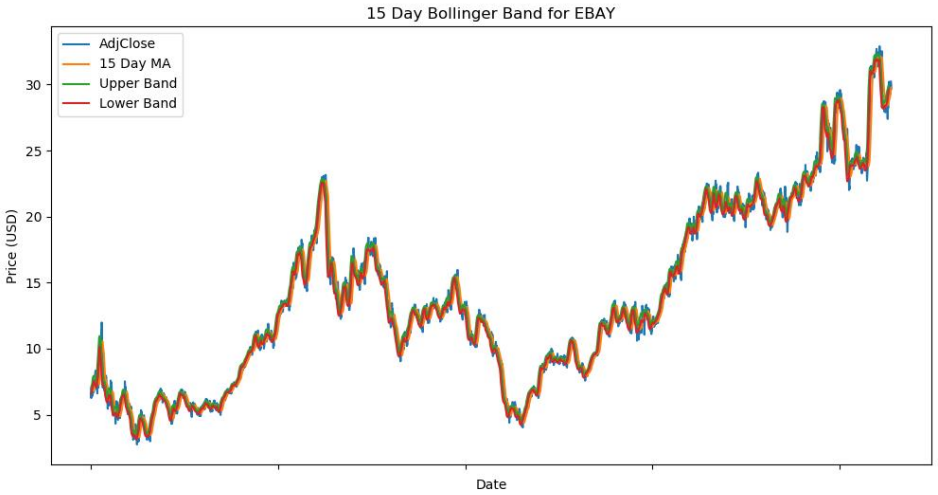


Figura 23: Evolución de las Bandas de Bollinger del precio del stock de EBAY para 0.5 desviaciones típicas

A partir de los resultados encontrados, el mejor resultado para tener una muestra bien balanceada, sería escoger 1 desviación típica, ahora bien, en la metodología de las bandas de Bollinger se intenta en cierta manera no hacer un exceso de operaciones, por lo que usar esta opción no sería habitual. Por ello aunque puede afectar luego el resultado obtenido en los modelos, se ha usado 1.25 veces la desviación estándar para definir el *label*.

6.4. Creación de los *features*

Así como el *label* es como se llama a la variable a predecir, los *features* son los *inputs*, en otras palabras, son lo que en un modelo lineal tradicional llamaríamos las variables independientes. Aunque en un principio se alimentaba a los modelos con las series temporales estandarizadas a partir de cambios porcentuales, esto cambio después a ser volatilidad para acabar como está ahora y se explicará a continuación.

Como se ha comentado, se usan las 500 series temporales para predecir una. A partir de cada una de estas variables, se han creado 6 variables, por lo que se ha pasado de 500 *features* a 3000. Estas variables han sido creadas a partir de los componentes de un posible sistema de *trading* para generar señales de compra y venta. Las variables creadas son las siguientes:

1) Media móvil a 15 días, vamos a llamarla $MM15$
2) Media móvil a 5 días, vamos a llamarla $MM5$
3) Variación a cinco días de la media móvil de 15 días, es decir

$$V1_{15} = MM15(t) - MM15(t-5)$$

4) Variación a cinco días de la media móvil de 5 días, es decir

$$V1_{5} = MM5(t) - MM5(t-5)$$

5) Variación a un día de V1 para la media móvil de 15 días

$$V2_{15} = V1_{15}(t) - V1_{15}(t-1)$$

6) Variación a un día de V1 para la media móvil de 5 días

$$V2_{5} = V1_{5}(t) - V1_{5}(t-1)$$

Aunque en un principio se alimentaban los modelos simplemente a partir de las series temporales, cuando los modelos parecía que necesitaban más información, y solo se disponía de las series temporales, se decidió intentar crear nuevas variables a partir de estas. Se escogieron estas transformaciones en particular porque al crear los *labels* a partir de un método derivado de las medias móviles, parecía tener sentido usar unos *features* que se derivaran de estas. Se escogieron 15 días para que coincidiera con la medida temporal del *label*, mientras que los cinco días resumen la actividad de la última semana de aquel *stock*. Las variaciones empleadas se comportan un poco como si fueran la primera y la segunda derivada de esas medias móviles, siendo metodologías bastante habituales en *trading*, por lo que también podían aportar cierta información.

Una vez se han calculado los *labels* y los *features* ha hecho falta realizar un último ajuste. Las primeras 15 observaciones de la base de datos no tenían *features*, ya que no se puede calcular la media móvil antes de tener 15 observaciones y el programa devuelve *missings*, por esta razón, una vez calculados los *features* se han eliminado las primeras 15 observaciones.

De similar manera, se han eliminado las 15 últimas observaciones de la base de datos, en este caso pero, es debido a que para las últimas 15 no conocemos cuál será su posición en 15 días, ya que es el futuro.

Al acabar todas las transformaciones y arreglos, tenemos una base de datos de $4248x3018$ observaciones. Son 3018 y no 3000 porque el índice S&P500 tiene técnica 505 empresas y dos de estas fueron eliminadas porque no se podía conseguir su información en la web *quandl* a partir del *web scraper* empleado.

6.5. Transformaciones específicas para la red neuronal recursiva

Aunque la teoría tras la red neuronal recursiva, en especial la *long short-term memory*, se explica más adelante, es necesario explicar las transformaciones que se han tenido que hacer en la base de datos en este apartado.

De la manera que funciona la *rnn*, se tienen que tener en cuenta dos aspectos al preparar los *features* para el modelo. Por un lado, el número de variables a usar, en este caso ya hemos determinado que serán 3018. Por otro lado los *timesteps* que se quieren tener en cuenta, donde para que se entienda, un *timestep* (que sería lo mínimo) es lo correspondiente a darle la información de t-15 para predecir t. En este caso se han usado 15 *timesteps*, por lo que se consigue en un solo vector de información, darle a la red la información correspondiente a 15 valores de la serie temporal al mismo tiempo. En la imagen a continuación se puede observar cómo quedaría cada uno de los vectores de aplicársele 3 *timesteps*.

Original data	shift of -1	shift of -2	shift of -3
1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7
5	6	7	8
6	7	8	9
7	8	9	10
8	9	10	Nan
9	10	Nan	Nan
10	Nan	Nan	Nan

Figura 24: Ilustración de un vector temporal con tres timesteps.

Fuente: https://towardsdatascience.com/using-lstms-to-forecast-time-series-4ab688386b1f

No hay ninguna norma escrita sobre cuantos timesteps se deben pasar a la red, idealmente si supiéramos cuantos *timesteps* aportan información usaríamos ese número exacto, pero obviamente no lo sabemos. Usar demasiados *timesteps* puede

añadir mucho ruido al modelo, al mismo tiempo que dificulta mucha más su entrenamiento y requerirá de una estructura mucho más compleja. Por otro lado, quedarse cortos de *timesteps* puede hacer que perdamos información por tratar de simplificar el problema.

Para este ejercicio, se ha escogido usar 16 *timesteps*, eso implica que cada fila de observaciones de la base de datos aporta al mismo tiempo información de 15 pasos temporales al mismo tiempo. Una vez acabado esto entonces, tenemos el mismo número de filas pero ahora tenemos 48288 *features*.

6.6. *Split* de la muestra

La práctica habitual cuando se formulan modelos de predicción es repartir la muestra en tres. El primer bloque es el la muestra de entrenamiento. Esta suele ser la muestra más grande de las tres ya que es donde se entrenara el modelo, como mayor sea, mejor. En esta muestra se formula como se ha comentado el modelo a partir de métricas como pueden ser la función pérdida o el *accuracy*. Dada la complejidad de estas metodologías, se pueden conseguir funciones (si se desea) que expliquen los datos a la perfección, por lo que se tendría lo que se llama *overfitting*, consiguiendo una función que explica todos las observaciones a la perfección pero que si se aplica a otra base de datos no va a funcionar nada bien, ya que solo explica esos primeros datos.

Por ello, se define otro bloque de observaciones, el de validación. La muestra de validación nos sirve para poder hacer (como el nombre indica) una validación del modelo que obtenemos entrenando. Una vez se tiene el modelo (normalmente automáticamente) se calcula sobre la muestra de observaciones de validación y se comparan las estimaciones obtenidas con los resultados reales observados. Al tratarse de un problema de clasificación se compara la clase predicha por el modelo con la clase observada. De esta manera se puede ver si se está obteniendo un modelo con *overfitting* o no, ya que se obtiene un resultado en una muestra de observaciones con la que no se ha calculado este.

El problema viene cuando se calculan muchos modelos, como se intenta en cierta manera maximizar el resultado en validación, lo que estamos es indirectamente crear *overfitting* sobre esta segunda muestra, ya que escogeremos los parámetros que maximicen nuestra *accuracy* en validación, cuando no es lo que deberíamos hacer. Por este motivo se tiene una tercera muestra; test. La muestra test se mantiene oculta durante todo el experimento, no se utiliza en ningún momento previo al final del estudio. Cuando este termina se estima la muestra de test y se observan los resultados. Esto se realiza únicamente cuando el modelo ya está totalmente terminado y los resultados obtenidos en test son los que se deben presentar al final del estudio (habitualmente).

Para este trabajo, se ha dividido la muestra original, des del 1 de enero del año 2000 hasta el 31 de diciembre de 2016 en la muestra de entrenamiento (hasta 2012, un 75% de la muestra) y validación (el 25% restante). Como muestra de test se ha escogido la información del año 2017 (del 1 de enero al 31 de diciembre).

7. RESULTADOS

En este apartado se encuentran los resultados obtenidos, así como la metodología empleada, los problemas encontrados y las posibles direcciones a tomar si se dispusiera de más tiempo y conocimiento sobre el tema.

7.1. Metodología empleada

Como se comenta al principio de este proyecto, hace apenas unos meses mi conocimiento sobre metodologías de *Machine* y *Deep learning* era nulo, al igual que el de *Python*, por lo que este trabajo ha servido sobre todo para poder empezar a coger práctica con estas herramientas de cara al futuro.

A partir de lo aprendido de forma autónoma, así como de lo que mucha gente con conocimiento (amigos y conocidos del entorno de trabajo) me ha comentado, he desarrollado una metodología para intentar definir los parámetros de mis modelos.

Entre los objetivos del trabajo, estaba poder comparar el funcionamiento de la red neuronal *Multilayer Perceptron* con la red neuronal recurrente *Long Short-Term Memory*, aunque por el camino también se han empleado las técnicas de *k-neighbours*, *Random Forest* y *Support vector Machine* solo para ir aprendiendo sobre el funcionamiento de estas técnicas y *Python*, por lo que estas no están incluidas en el trabajo. Haber empezado por las redes neuronales sin hacer primero un paso por los algoritmos de *Machine Learning* hubiera sido como empezar la casa por el tejado.

La metodología empleada en las dos redes ha sido la misma:

1) Una vez se tiene la base de datos transformada en *features* y *labels*, lo primero que se hace es crear una sub-muestra de por ejemplo 100 observaciones con la que trabajar. En esta sub-muestra se irán probando los distintos parámetros para después cuando estos se hayan escogido entrenar el modelo con toda la base de datos. De no trabajar de esta manera, el proceso se ralentizaría mucho.

2) A continuación se construye una red neuronal inicial con tres capas de neuronas, es decir, 1 *input layer*, 1 *hidden layer* y 1 *output layer*. Por lo aprendido, siempre son necesarias al menos tres capas a no ser que se trate de un problema linealmente separable y más de tres capas no tienen por qué mejorar el funcionamiento.

3) Aparte de las neuronas en la capa de entrada y salida, que deben corresponderse con el número de *features* y *labels* no hay ninguna norma escrita sobre cuantas tiene que haber en las *hidden layers*. Por ello, y a partir de lo que gente entendida me ha enseñado, primero se pone una cantidad de neuronas elevada. Sobre que es o que no es muchas neuronas no hay una norma escrita, hay que ir probando.

4) A continuación, cuando se tiene la estructura de la red creada, se entrena con los parámetros por defecto, buscando minimizar la función pérdida. En este punto lo que se busca es mirar si la pérdida va bajando en cada iteración del modelo. Es decir, intentamos ver si parece que nos quedamos en un mínimo local o no, eso querría decir que tenemos que ajustar el *learning rate* o el *decay learning rate*. Para que se entienda mejor, debemos imaginar la función pérdida como dos valles entre unas montañas, una por debajo de la otra y separadas por una valla. Si el *learning rate* fuera muy alto, sería lo correspondiente a saltar de un pico de la montaña al pico opuesto de esta, siempre nos perderíamos el mínimo de la función porque damos pasos demasiado grandes. Por otro lado si el *learning rate* fuera demasiado pequeño, cuando llegáramos a la primera valle (la más elevada) no podríamos saltar la valla porque nuestros pasos son demasiado pequeños, quedándonos siempre en el mínimo local (el primer valle). Una herramienta para evitar poner un *learning rate* demasiado pequeño, es el *decay learning rate*, que hace que a medida que se itera, el *learning rate* se vaya reduciendo.

5) Una vez parece que se ha encontrado el *learning rate* (o *learning rate* y *decay learning rate*) apropiados, toca ajustar el número de neuronas. Como ya se ha comentado no hay un número en particular de neuronas que resuelva el problema, simplemente se trata de ir ajustando de manera que el modelo no tenga demasiadas y tarde mucho a entrenar además de hacer *overfitting* en los datos ni de que tenga demasiado pocas y sea tan simple que no llegue a poder explicar los datos.

6) Otro factor a ir ajustando son las *epochs* y el *batch size*, de la misma manera, unos valores muy grandes en estos parámetros nos pueden llevar a hacer un *overfitting*, mientras que demasiado pequeños nos pueden llevar a simplificar demasiado el problema. De nuevo no hay normas escritas al respeto, se trata de ir jugando con los valores y a partir de prueba y error ajustarlos.

7) Una vez se han determinado los parámetros de la red, habitualmente se entrenaría el modelo escogido con todas las observaciones y cambiando la

métrica a optimizar, siendo antes minimizar la función pérdida y ahora maximizar el *accuracy*.

8) Una vez se obtiene el modelo deseado, se calculan los resultados obtenidos en la muestra de test y se presentan.

7.2. Multilayer Perceptron

Primero se presentarán los resultados obtenidos para la red neuronal del tipo *multilayer perceptron*, así como la configuración final del modelo. La estructura del modelo ha sido de tres capas; un *input layer* con unas dimensiones de *input* de 3018 (número de *features*) y 128 neuronas, una *hidden layer* con 1500 neuronas y una *output layer* con 3 salidas. Las funciones de activación de cada capa han sido *Rectifier Linear Unit* ($f(x) = \max(0, x)$) para las dos primeras y función exponencial normalizada *softmax* (una generalización de la función logística $f(z)_k = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ para j = 1, …, K).

Los parámetros utilizados han sido los siguientes:

- Optimizador: Adam[6] con un *learning rate* de 10^{-5} y un *decay learning rate* 10^{-6}.
- Función pérdida: *Categorical crossentropy*.
- *Epochs*: 1000.
- *Batch size*: 72.

Los resultados observados para las tres muestras son los siguientes:

- *Accuracy* en muestra de entrenamiento: 0.8572
- *Accuracy* en muestra de validación: 0.5198
- Clasificación de los datos en validación:

Vender (-1)	Mantener (0)	Comprar (1)
33	877	152

Tabla 6: Resultados en validación MP

[6] El optimizador Adam funciona igual que el optimizador *gradient descent* solo que se puede ajustar su *decay learning rate.*

- *Accuracy* en muestra test: 0.5135
- Clasificación de los datos en test:

Vender (-1)	Mantener (0)	Comprar (1)
3	219	0

Tabla 7: Resultados en test MP

- Observaciones reales en test:

Vender (-1)	Mantener (0)	Comprar (1)
57	117	48

Tabla 8: Observaciones en test

7.2.1. Observaciones

Partiendo de las ideas iniciales sobre los trabajos comentados en la introducción y los objetivos, en ningún momento se esperaba encontrar una *accuracy* muy elevada en estos modelos. Durante todas las iteraciones que ha ido teniendo el trabajo (que han sido muchas) el objetivo ha sido ver si era posible encontrar algún modelo con un *accuracy* superior a escoger aleatoriamente, lo que significaría que las series temporales de los precios aportan información sobre el futuro de este.

Aunque las *accuracy* encontradas tanto en validación como en test son de hecho superiores a escoger aleatoriamente (ya que al tener tres clases eso sería un 0.33), es importante ver que como más nos alejamos de los datos donde hemos entrenado (es decir a medida que avanzamos en el tiempo), le cuesta más al modelo clasificar como algo que no sea mantenerse (0).

Esta *accuracy* obtenida está entonces en cierta manera inflada porque nuestro modelo tiene problemas para marcar las observaciones como algo que no sean 0. Esto esencialmente es debido a que no tiene suficiente información en los *features* para clasificar, pero también a que es posible que el comportamiento de la serie sea distinto en test. En los siguientes apartados se comentarán más los resultados.

7.3. Long Short-Term Memory neural network

A continuación, se encuentran las características de la red neuronal recurrente empleada. La estructura de esta es dos capas (aunque cuentan por tres); un *input layer* con unas dimensiones de *input* de 48288 (número de *features*) y 512 neuronas (que cuentan como a una *hidden layer)* y una *output layer* con 3 salidas. La función de activación de la capa *output* es de nuevo *softmax*.

Los parámetros utilizados han sido los siguientes:

- Optimizador: Adam con un *learning rate* de 10^{-5} y un *decay learning rate* 10^{-6}.
- Función pérdida: *Categorical crossentropy*.
- *Epochs*: 500.
- *Batch size*: 72.

Los resultados observados para las tres muestras son los siguientes:

- *Accuracy* en muestra de entrenamiento: 0.7966
- *Accuracy* en muestra de validación: 0.5499
- Clasificación de los datos en validación:

Vender (-1)	Mantener (0)	Comprar (1)
85	976	1

Tabla 9: Resultados en validación LSTM

- *Accuracy* en muestra test: 0.5135
- Clasificación de los datos en test:

Vender (-1)	Mantener (0)	Comprar (1)
3	219	0

Tabla 10: Resultados en test LSTM

- Observaciones reales en test:

Vender (-1)	Mantener (0)	Comprar (1)
57	117	48

Tabla 11: Observaciones en test

7.3.1. Observaciones

Al igual que en el modelo estimado con el algoritmo del perceptrón multicapa, los resultados encontrados usando la red neuronal recurrente no son demasiado buenos, y fallan de modo parecido. El *accuracy* en entrenamiento es 0.7966 y en validación 0.5499, lo que a principio podría indicar un mejor funcionamiento que en el otro modelo. El problema está en que cuando se comprueba el funcionamiento de este en test, se obtiene exactamente el mismo resultado, con casi todas las observaciones siendo clasificadas como 0 y solo 3 clasificadas como -1. Las posibles causas son seguramente las mismas y se comentarán más adelante en detalle

7.4. Comparación entre los modelos obtenidos

Una vez se tienen los dos modelos estimados, la técnica más habitual para comparar su funcionamiento, es comparar los resultados obtenidos en test. En este caso en particular, hemos obtenido exactamente los mismos resultados en test para el perceptrón multicapa y la *Long Short-Term Memory*. Basándonos únicamente en estos resultados, no hay indicios que hagan pensar que una funcione mejor que la otra.

Aunque si nos fijáramos en validación, el *accuracy* obtenido por la red recurrente es mayor que el de la red neuronal no recurrente, este resultado podría haber sido manipulado (incluso inconscientemente) por lo que no es una buena medida para comparar dos modelos distintos.

Así como en los estudios previos a este parece ser que los resultados mostraban un mejor funcionamiento de las redes recurrentes respecto las no recurrentes, en este caso no hay pruebas para respaldar esta teoría. Ahora bien, esto no quiere decir que no sea cierto, ya que en este caso podría haber algún tipo de problema en los datos o el planteamiento que no deje realmente ver la mejoría en el funcionamiento de los modelos usando estos algoritmos. Por ello a continuación, se presentan problemas y posibles mejoras que se han encontrado durante la realización del trabajo, y que podrían explorarse en el futuro de tener más tiempo y conocimiento.

7.5. Problemas y posibles mejoras

El principal problema encontrado durante todo el trabajo han sido siempre los datos, más en concreto la posible falta de información de los *features* para predecir el futuro valor del *label*. Este temor encontraba su origen cuando leía documentos de estudios realizados al respeto, donde se comentaba habitualmente que los históricos de los precios no parecían aportar demasiada información sobre el futuro de este. Este hecho se hizo evidente nada más empezar y encontrar que todos los resultados que se iban obteniendo eran malos (peores que escoger aleatoriamente). Como ya se ha comentado, el ánimo del trabajo no era en si encontrar un buen modelo, sino aprender sobre estos y ver si se podían encontrar indicios que hicieran pensar que con los históricos únicamente se podía predecir el futuro valor de este.

Ante el temor de que los precios no fueran suficiente, se crearon como se explica en el apartado sobre los *features* distintas variables de cada serie temporal, en un intento de obtener más información de estos datos. Ahora bien, por el camino también es posible que se haya introducido más ruido, haciendo que este empeorara.

Otro de los problemas que ha supuesto muchas distintas iteraciones al trabajo ha sido como plantear los tres niveles del *label*. En un principio, se definían los dos puntos de corte de manera subjetiva para que quedaran muestras balanceadas (con aproximadamente un tercio de las observaciones en cada clase). Esto pero plateaba varios problemas, como por ejemplo no estar basado en ningún principio de *trading* y principalmente que significaba realizar muchas operaciones. Es decir, normalmente se intenta reducir el número de operaciones, asegurándose que esas que se realizan van a ser lo más seguras que se pueda. Consiguiendo una muestra balanceada se perdía esta característica, y dado que se pretendía modelar las acciones de compra y venta de un *trader* se decidió trabajar con la muestra no balanceada.

Trabajar con la muestra no balanceada provoca dos problemas. Por un lado, como es evidente por los resultados, parece que la muestra de compras y ventas puede ser pequeña y por ello el modelo tiene problemas después para clasificar las observaciones como tal. Por otro lado, el problema para mí más importante es que al tener un modelo que predice mayoritariamente 0, sea por el motivo que sea, y una muestra donde la mayoría de observaciones son 0, cuesta saber si realmente predecimos los 0 bien o simplemente estamos prediciendo todo 0. Es decir, aunque obtenemos un *accuracy* que para mí es bastante bueno (en otro tipo de problema no lo sería pero aquí sí), no podemos saber si realmente es así porque el modelo clasifica casi todas las observaciones como 0, o si realmente clasifica bien los 0 pero clasifica mal las compras y ventas.

Uno de los grandes problemas de intentar predecir series temporales, es que la observación que siempre importa más es siempre la más reciente, por lo que a medida que nos alejamos de donde hemos entrenado el modelo es muy fácil que este no sepa predecir las observaciones, por ejemplo si la serie pasa a tener un comportamiento distinto que el visto con anterioridad. Esto plantea otro problema que es que la red *Long Short-Term Memory* destaca por poder recordar largos periodos de tiempo, pero puede ser que esto no sea lo más adecuado en estos casos, ya que realmente la importancia del evento más reciente es mucho mayor a la del resto de eventos.

Otro de los factores a tener en cuenta es que los mercados presentan épocas con tendencia y épocas sin, siendo las segundas mucho más comunes que las primeras, pero siendo las primeras las épocas en las que se debe abusar de que el mercado tenga tendencia. Es posible entonces que como se observa por el *spread* de los datos, la época de test (e incluso la de validación) puedan encontrarse en momentos en que el mercado no tiene tendencia, dificultando que haya operaciones de compra o venta.

Para terminar con este apartado, algunas mejoras que se me ocurren para poder mejorar los modelos si dispusiera de más tiempo y en muchos casos más conocimiento son las siguientes:

1) Ampliar el modelo con nuevas variables que no sean únicamente las series temporales ya que parece que estas aportan poca información. Las posibilidades más interesantes aunque también son las más complejas sería poder realizar dentro del modelo análisis de noticias diario sobre las empresas o análisis de sentimiento sobre estas en páginas como por ejemplo *twitter*. También se podrían añadir más *features* como por ejemplo los volúmenes diarios.

2) Jugar más con el intervalo temporal para ver si pueden mejorar las predicciones. Aunque a lo largo del trabajo se han probado diferentes horizontes temporales y en ningún caso los resultados mejoraban, es muy difícil encontrar el punto óptimo que predecir en el futuro (días), y me gustaría poder hacer más pruebas al respeto.

3) Probar otras metodologías que no sean redes neuronales, ya que aunque al principio como se ha comentado para aprender sobre los algoritmos hice pruebas con metodologías distintas de *machine learning*, realmente no profundicé en ellas y sería posible que a lo mejor alguna funcionara mejor que las redes neuronales para series temporales.

4) Hacer pruebas para la predicción del precio y no de una clasificación de este, ya que aunque es más complicado de realizar y requiere más conocimiento, es posible que de mejores resultados.

5) Ante el temor de tener demasiado ruido y demasiadas variables, creo que podría ser interesante a lo mejor realizar un estudio de componentes principales a priori para reducir las dimensiones de los *features* y luego aplicar los algoritmos de aprendizaje.

6) Investigar si con metodologías más clásicas como puede ser ARIMA se puede conseguir determinar mejor el horizonte temporal ideal

7) Si en algún momento se encontrara un modelo que funciona se tendría aun que tener en cuenta factores como son por ejemplo los costes de transacción entre otros.

8. CONCLUSIONES

Cuando empecé con este trabajo, mi conocimiento sobre *Python*, *Machine Learning* y *Deep Learning* era nulo. Es por ello que intentar entrar en este mundo de golpe y sobretodo en una aplicación práctica en mente tan complicada y problemática como son las series temporales y aún más los precios de las acciones de empresas en bolsa fue en un principio abrumador. Los materiales a estudiar aunque abundantes eran complicados y se me acumulaba el trabajo sin poder avanzar en la parte práctica porque aún me faltaba el conocimiento. Aun así, creo que eso me ha permitido avanzar a pasos de gigante respeto si hubiera escogido algo más sencillo, y también por ello uno de los principales objetivos del trabajo era aprender lo máximo posible de cara a que esta experiencia pueda ayudarme en un futuro.

Uno de los factores que me ha permitido realizar este trabajo ha sido la gran cantidad de materiales gratuitos, tutoriales, ayudas y *blogs* con información que se pueden encontrar fácilmente por internet. Esto permite que aunque en realidad se estén utilizando algoritmos complejos, se pueda empezar autónomamente a aprender y practicar gracias por ejemplo a librerías como *Keras* o *Scikit-Learn*.

Al principio del trabajo se planteaban entre otros los siguientes objetivos (o hipótesis), si es posible predecir el futuro valor (en este caso un estado) de una empresa en bolsa a partir de la información de este y las otras empresas de un índice en común y si las redes neuronales recurrentes funcionan o no mejores que las redes neuronales estándar para esta aplicación. Sobre la primera hipótesis, no hay pruebas suficientes que indiquen únicamente con lo realizado en este trabajo que se puede predecir este. Aun así, con más tiempo y conocimiento aplicando alguna de las posibles mejoras mencionadas no se descarta que se pudiese probar lo contrario. La segunda hipótesis queda un poco en el aire, al haber obtenido unos resultados idénticos en ambos modelos. Sería necesario primero responder a la primera con total seguridad para pasar a la segunda, ya que en el caso que la respuesta a esta primera sea negativa, no tiene sentido plantear la segunda.

En resumen, considero que aunque los resultados visibles de este trabajo no permiten responder de manera clara a las preguntas formuladas, estas eran des de un principio difíciles de resolver. Por otro lado, la experiencia ganada utilizando estas técnicas por primera vez y aprendiendo *Python* es de aun más valor para mí. Si me arrepiento de alguna cosa es que creo que escogí una aplicación muy complicada para empezar con temas de *Machine Learning*, ya que empecé directamente planteando el problema sobre redes neuronales sin haber hecho nunca ninguno de los algoritmos más sencillos, y además escogiendo una aplicación problemática como son las series temporales y los índices bursátiles. Si hubiera escogido una temática más sencilla a lo mejor además de haber obtenido unos resultados finales más satisfactorios podría haber aprendido más y más rápido.

Ya para concluir, con este trabajo he podido aprender sobre todo como no plantear el problema, es decir, mis primeros instintos eran intentar abarcar todo y solucionar cosas que aún no daban problemas, en vez de ir un poco más despacio y concentrándome en problemas más pequeños. Creo que si volviese a empezar el trabajo lo plantearía de un modo distinto si hubiese sabido todo lo que se ahora, aun así no me arrepiento de todo lo que gracias a este he podido aprender.

9. BIBLIOGRAFÍA

(1) Argimiro Arratial, 8 de mayo de 2014. *Computational Finance: An Introductory Course with R*

(2) *Masaya Abe and Hideki Nakayama, The University of Tokyo, Tokyo, Japan*, 2017. *Deep Learning for Forecasting Stock Returns in the Cross-Section.*

(3) *Jim Kyung-Soo Liew and Boris Mayster, Johns Hopkins University,* 14 de Enero 2017. *Forecasting ETFs with Machine Learning Algorithms.*

(4) *Thomas Fischer, Christopher Krauss, University of Erlangen-Nürnberg, Nürnberg, Germany,* 10 Mayo 2017. *Deep learning with short-term memory networks for financial market predictions.*

(5) Conditional Image Synthesis With Auxiliary Classifier GANs, Augustus Odena, Cristopher Olah, Jonathon Shlens, 20 de Julio de 2017.

(6) Tensorflow: Large-scale machine learning on heterogeneous distributed systems, Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, 14 de Marzo de 2016

10. WEBGRAFÍA

(1) 1 de Febrero 2010. http://scikit-learn.org/

(2) Daphne Koller y Andre Ng, 2012. https://www.coursera.org/

(3) Skymind, 2017. https://deeplearning4j.org/

(4) Masaya Abe y Hideki Nakayama , 2017. https://arxiv.org/ftp/arxiv/papers/1801/1801.01777.pdf

(5) Jim Kyung-Soo Liew y Boris Mayster , 14 Enero de 2017. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2899520

(6) Thomas Fischer, Christopher Krauss, 10 de Mayo 2017. https://www.econstor.eu/bitstream/10419/157808/1/886576210.pdf

(7) Blog de Cristopher Olah, 27 de Agosto de 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

(8) Jason Brownlee, 21 de Setiembre de 2016. https://machinelearningmastery.com/improve-deep-learning-performance/

(9) Stack Exchange, 20 de Julio de 2010. https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netwo?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

(10) Stack Exchange, 10 de Agosto de 2011. https://stats.stackexchange.com/questions/14099/using-k-fold-cross-validation-for-time-series-model-selection?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

(11) Stack Exchange, 25 de Mayo de 2017. https://datascience.stackexchange.com/questions/19196/forget-layer-in-a-recurrent-neural-network-rnn?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

(12) Felix Gers, 2011. http://www.felixgers.de/papers/phd.pdf

(13) Wikipedia. https://es.wikipedia.org/wiki/Bolsa_de_valores

(14) Lilian Weng, 8 de Julio de 2017. https://lilianweng.github.io/lil-log/2017/07/08/predict-stock-prices-using-RNN-part-1.html

(15) Lilian Weng, 22 de Julio de 2017. https://lilianweng.github.io/lil-log/2017/07/22/predict-stock-prices-using-RNN-part-2.html

(16) Alexandr Honchar, 11 de Mayo de 2017. https://medium.com/machine-learning-world/neural-networks-for-algorithmic-trading-1-2-correct-time-series-forecasting-backtesting-9776bfd9e589

(17) Analytics Vidhya Content Team, 12 de Abril de 2016. https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/

(18) Sunil Ray, 13 de Setiembre de 2017. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

(19) Stack Overflow, 24 de Julio de 2017. https://stackoverflow.com/questions/45278286/how-to-choose-lstm-keras-parameters?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

(20) https://www.python.org/doc/essays/blurb/

(21) Wikipedia. https://es.wikipedia.org/wiki/Lenguaje_orientado_a_objetos

(22) Wikipedia. https://es.wikipedia.org/wiki/Python

(23) Python Software Foundation, 2001. https://pandas.pydata.org/

(24) Youtube. https://www.youtube.com/watch?v=5Vl-bK7tfD8&list=PLBAGcD3siRDittPwQDGIIAWkjz-RucAc7

(25) Wikipedia. https://en.wikipedia.org/wiki/Pandas_(software)

(26) Wikipedia. https://es.wikipedia.org/wiki/NumPy

(27) Numpy Developers. http://www.numpy.org/

(28) Bernard Brenyah, Enero de 2013. https://medium.com/python-data/setting-up-a-bollinger-band-with-python-28941e2fa300

(29) Keras Developers. https://keras.io/

(30) Wikipedia. https://en.wikipedia.org/wiki/PyCharm

(31) Wikipedia. https://es.wikipedia.org/wiki/Standard_%26_Poor%27s

(32) Wikipedia. https://es.wikipedia.org/wiki/S%26P_500

(33) Wikipedia. https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

(34) Wikipedia. https://es.wikipedia.org/wiki/Web_scraping

(35) Wikipedia. https://es.wikipedia.org/wiki/Beautiful_Soup

(36) Vladimir Ribakov, 22 de Agosto de 2016. https://vladimirribakov.com/bollinger-bands-ultimate-guide-part1/

11. ANEXO

11.1. Código

- Descarga de la base de entrenamiento y validación:

```
import bs4 as bs
import datetime as dt
import matplotlib.pyplot as plt
from matplotlib import style
import os
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import pickle
import requests
import sys
style.use('ggplot')


def get_sp500_tickers():
    request =
requests.get('https://en.wikipedia.org/wiki/List_of_S%26P_500_companie
s')
    soup = bs.BeautifulSoup(request.text, "lxml")
    info = soup.find ('table', {'class':'wikitable sortable'})
    tickers = []
    for row in info.findAll('tr')[1:]:
        ticker = row.findAll('td')[0].text
        tickers.append(ticker)

    with open("sp500tickers.pickle","wb") as f:
        pickle.dump(tickers,f)

    print(tickers)

    tickers.remove('BRK.B')
    tickers.remove('BF.B')

    return tickers

# get_sp500_tickers()

def get_data_from_quandl(reload_sp500=False):
    if reload_sp500:
        tickers = get_sp500_tickers()
    else:
        with open("sp500tickers.pickle","rb") as f:
            tickers = pickle.load(f)
    if not os.path.exists('stock_dfs'):
        os.makedirs('stock_dfs')
    tickers.remove('BRK.B')
    tickers.remove('BF.B')
    start = dt.datetime(2000,1,1)
    end = dt.datetime(2016,12,31)
```

```
    for ticker in tickers:
        print(ticker)
        if not os.path.exists('stock_dfs/{}.csv'.format(ticker)):
            df = web.DataReader(ticker, 'quandl', start, end)
            df.to_csv('stock_dfs/{}.csv'.format(ticker))
        else:
            print('Already have {}'.format(ticker))

# get_data_from_quandl()

def compile_data():
    with open("sp500tickers.pickle","rb") as f:
        tickers = pickle.load(f)
        tickers.remove('BRK.B')
        tickers.remove('BF.B')
    main_df = pd.DataFrame()

    for count,ticker in enumerate(tickers): #preguntar com va
count,ticker
        df = pd.read_csv('stock_dfs/{}.csv'.format(ticker))
        df.set_index('Date', inplace=True)

        df.rename(columns = {'AdjClose':ticker}, inplace=True)

df.drop(['Open','High','Low','Close','Volume','ExDividend','SplitRatio
','AdjOpen','AdjHigh','AdjLow','AdjVolume'], 1, inplace=True)

        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df, how='outer')

        if count % 10 == 0:
            print(count)
    main_df.iloc[::-1]

    print(main_df.head())
    main_df.to_csv('sp500_joined_closes.csv')

compile_data()


def visualize_data():
    df = pd.read_csv('sp500_joined_closes.csv')

    df_corr = df.corr()

    data = df_corr.values
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)

    heatmap = ax.pcolor(data, cmap=plt.cm.RdYlGn)
    fig.colorbar(heatmap)
    ax.set_xticks(np.arange(data.shape[0]) + 0.5, minor=False)
    ax.set_yticks(np.arange(data.shape[1]) + 0.5, minor=False)
    ax.invert_yaxis()
    ax.xaxis.tick_top()

    column_labels = df_corr.columns
    row_labels = df_corr.index
```

```
    ax.set_xticklabels(column_labels)
    ax.set_yticklabels(row_labels)
    plt.xticks(rotation=90)
    heatmap.set_clim(-1,1)
    plt.tight_layout()
    plt.show()

visualize_data()
```

- Descarga de los datos de test:

```
import bs4 as bs
import datetime as dt
import matplotlib.pyplot as plt
from matplotlib import style
import os
import numpy as py
import pandas as pd
import pandas_datareader.data as web
import pickle
import requests
import sys
style.use('ggplot')


def get_sp500_tickers():
    request =
requests.get('https://en.wikipedia.org/wiki/List_of_S%26P_500_companie
s')
    soup = bs.BeautifulSoup(request.text, "lxml")
    info = soup.find ('table', {'class':'wikitable sortable'})
    tickers = []
    for row in info.findAll('tr')[1:]:
        ticker = row.findAll('td')[0].text
        tickers.append(ticker)

    with open("sp500tickers.pickle","wb") as f:
        pickle.dump(tickers,f)

    print(tickers)

    tickers.remove('BRK.B')
    tickers.remove('BF.B')

    return tickers

# get_sp500_tickers()

def get_data_from_quandl(reload_sp500=False):
    if reload_sp500:
        tickers = get_sp500_tickers()
    else:
        with open("sp500tickers.pickle","rb") as f:
            tickers = pickle.load(f)
    if not os.path.exists('stock_dfs_test'):
        os.makedirs('stock_dfs_test')
    tickers.remove('BRK.B')
    tickers.remove('BF.B')
    start = dt.datetime(2017,1,1)
    end = dt.datetime(2017,12,31)

    for ticker in tickers:
        print(ticker)
        if not os.path.exists('stock_dfs_test/{}.csv'.format(ticker)):
            df = web.DataReader(ticker, 'quandl', start, end)
            df.to_csv('stock_dfs_test/{}.csv'.format(ticker))
        else:
```

```
            print('Already have {}'.format(ticker))

# get_data_from_quandl()

def compile_data():
    with open("sp500tickers.pickle","rb") as f:
        tickers = pickle.load(f)
        tickers.remove('BRK.B')
        tickers.remove('BF.B')
    main_df = pd.DataFrame()

    for count,ticker in enumerate(tickers): #preguntar com va
count,ticker
        df = pd.read_csv('stock_dfs_test/{}.csv'.format(ticker))
        df.set_index('Date', inplace=True)

        df.rename(columns = {'AdjClose':ticker}, inplace=True)

df.drop(['Open','High','Low','Close','Volume','ExDividend','SplitRatio
','AdjOpen','AdjHigh','AdjLow','AdjVolume'], 1, inplace=True)

        if main_df.empty:
            main_df = df
        else:
            main_df = main_df.join(df, how='outer')

        if count % 10 == 0:
            print(count)
    # main_df.iloc[::-1]

    print(main_df.head())
    main_df.to_csv('sp500_joined_closes_test.csv')

compile_data()
```

- Creación de labels y *features* para la red neuronal en entrenamiento y validación:

```
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
from sklearn import svm, neighbors, model_selection
from sklearn.ensemble import VotingClassifier, RandomForestClassifier


import sys


def process_data_for_labels(ticker):
    hm_days = 15 #days al futur pel preu per pujar o baixar x
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)

    name = ticker

    name = pd.DataFrame()
    name['AdjClose'] = df[ticker]

    print(name)

    name['15 Day MA'] = name['AdjClose'].rolling(window=15).mean()
    name['15 Day STD'] = name['AdjClose'].rolling(window=15).std()
    name['Upper Band'] = name['15 Day MA'] + (name['15 Day STD'] *
1.25)
    name['Upper Band'] = name['Upper Band'].shift(-15)
    name['Lower Band'] = name['15 Day MA'] - (name['15 Day STD'] *
1.25)
    name['Lower Band'] = name['Lower Band'].shift(-15)

    name[['AdjClose', '15 Day MA', 'Upper Band', 'Lower
Band']].plot(figsize=(12, 6))
    plt.title('15 Day Bollinger Band for {}'.format(ticker))
    plt.ylabel('Price (USD)')
    plt.show()

    print(name)

    tickers = df.columns.values.tolist()

    name.fillna(0, inplace=True)

    # Eliminamos las 15 primeras porque no tienen medias moviles para
calcular el target

    name = name[14:(len(name)-15)]

    # Creación del target

    name['{}_target'.format(ticker)] = np.NaN

    for i in range(name.shape[0]):
```

```
        if(name['AdjClose'][i] > name['Upper Band'][i]):
            name['{}_target'.format(ticker)][i] = 1
        elif (name['AdjClose'][i] < name['Lower Band'][i]):
            name['{}_target'.format(ticker)][i] = -1
        else:
            name['{}_target'.format(ticker)][i] = 0

    print(name)

    vals = name['{}_target'.format(ticker)].values.tolist()

    str_vals = [str(i) for i in vals]

    print('Data spread:', Counter(str_vals))

    return tickers, df, name

# process_data_for_labels('EBAY')
#
# sys.exit()
def differences(x):
    return x[-1] - x[0]

def extract_featuresets(ticker):

    tickers, df, y = process_data_for_labels(ticker)

    df_vals = pd.DataFrame()

    for j in tickers:

        df_vals['{}_MA_15'.format(j)] =
df[j].rolling(window=15).mean()
        df_vals['{}_MA_5'.format(j)] = df[j].rolling(window=5).mean()
        df_vals['{}_V1_15'.format(j)] =
df_vals['{}_MA_15'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V1_5'.format(j)] =
df_vals['{}_MA_5'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V2_15'.format(j)] =
df_vals['{}_V1_15'.format(j)].rolling(window=2).apply(differences)
        df_vals['{}_V2_5'.format(j)] =
df_vals['{}_V1_5'.format(j)].rolling(window=2).apply(differences)

    # df_vals = df_vals.replace([np.inf, -np.inf], 0)

    print(df_vals)

    df_vals.fillna(0, inplace=True)

    print(df_vals)

    X = df_vals.values[14:(df_vals.shape[0]-15)] #feature set
    print(X.shape[0])
    # hem de correr el valor target trenta posicions enrere
    y = y['{}_target'.format(ticker)].values #label

    print(len(y))
    print(y)

    X_train, X_test, y_train, y_test =
model_selection.train_test_split(X,
```

```
y,

test_size = 0.25,shuffle=False)
    np.save('X_train_sp.npy', X_train)
    np.save('X_test_sp.npy', X_test)
    np.save('y_train_sp.npy', y_train)
    np.save('y_test_sp.npy', y_test)

    return X, y, df

extract_featuresets('EBAY')
```

- Creación de labels y *features* para la red neuronal en test:

```
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pickle
from sklearn import svm, neighbors, model_selection
from sklearn.ensemble import VotingClassifier, RandomForestClassifier


import sys


def process_data_for_labels(ticker):
    hm_days = 15 #days al futur pel preu per pujar o baixar x
    df = pd.read_csv('sp500_joined_closes_test.csv', index_col=0)

    name = ticker

    name = pd.DataFrame()
    name['AdjClose'] = df[ticker]

    print(name)

    name['15 Day MA'] = name['AdjClose'].rolling(window=15).mean()
    name['15 Day STD'] = name['AdjClose'].rolling(window=15).std()
    name['Upper Band'] = name['15 Day MA'] + (name['15 Day STD'] *
1.25)
    name['Upper Band'] = name['Upper Band'].shift(-15)
    name['Lower Band'] = name['15 Day MA'] - (name['15 Day STD'] *
1.25)
    name['Lower Band'] = name['Lower Band'].shift(-15)

    print(name)

    tickers = df.columns.values.tolist()

    name.fillna(0, inplace=True)

    # Eliminamos las 15 primeras porque no tienen medias moviles para
calcular el target

    name = name[14:(len(name)-15)]

    # Creación del target

    name['{}_target'.format(ticker)] = np.NaN

    for i in range(name.shape[0]):
        if(name['AdjClose'][i] > name['Upper Band'][i]):
            name['{}_target'.format(ticker)][i] = 1
        elif (name['AdjClose'][i] < name['Lower Band'][i]):
            name['{}_target'.format(ticker)][i] = -1
        else:
            name['{}_target'.format(ticker)][i] = 0
```

```
    print(name)

    vals = name['{}_target'.format(ticker)].values.tolist()

    str_vals = [str(i) for i in vals]

    print('Data spread:', Counter(str_vals))

    return tickers, df, name

# process_data_for_labels('EBAY')
#
# sys.exit()
def differences(x):
    return x[-1] - x[0]

def extract_featuresets(ticker):

    tickers, df, y = process_data_for_labels(ticker)

    # df.fillna(0, inplace=True)

    # df = df.replace([np.inf, -np.inf], np.nan)
    # df.dropna(inplace=True)

    df_vals = pd.DataFrame()

    for j in tickers:

        df_vals['{}_MA_15'.format(j)] =
df[j].rolling(window=15).mean()
        df_vals['{}_MA_5'.format(j)] = df[j].rolling(window=5).mean()
        df_vals['{}_V1_15'.format(j)] =
df_vals['{}_MA_15'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V1_5'.format(j)] =
df_vals['{}_MA_5'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V2_15'.format(j)] =
df_vals['{}_V1_15'.format(j)].rolling(window=2).apply(differences)
        df_vals['{}_V2_5'.format(j)] =
df_vals['{}_V1_5'.format(j)].rolling(window=2).apply(differences)

    # df_vals = df_vals.replace([np.inf, -np.inf], 0)

    print(df_vals)

    df_vals.fillna(0, inplace=True)

    print(df_vals)

    X = df_vals.values[14:(df_vals.shape[0]-15)] #feature set
    print(X.shape[0])
    # hem de correr el valor target trenta posicions enrere
    y = y['{}_target'.format(ticker)].values #label

    print(len(y))
    print(y)

    # X_train, X_test, y_train, y_test =
model_selection.train_test_split(X,
    #
y,
```

```
    #
test_size = 0.25,shuffle=False)
    np.save('X_test_test.npy', X)
    np.save('y_test_test.npy', y)

    return X, y, df

extract_featuresets('EBAY')
```

- Creación de labels y *features* para la red neuronal recurrente en entrenamiento y validación:

```
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
from sklearn import svm, neighbors, model_selection
from sklearn.ensemble import VotingClassifier, RandomForestClassifier


import sys


def process_data_for_labels(ticker):
    hm_days = 15 #days al futur pel preu per pujar o baixar x
    df = pd.read_csv('sp500_joined_closes.csv', index_col=0)

    name = ticker

    name = pd.DataFrame()
    name['AdjClose'] = df[ticker]

    print(name)

    name['15 Day MA'] = name['AdjClose'].rolling(window=15).mean()
    name['15 Day STD'] = name['AdjClose'].rolling(window=15).std()
    name['Upper Band'] = name['15 Day MA'] + (name['15 Day STD'] *
1.25)
    name['Upper Band'] = name['Upper Band'].shift(-15)
    name['Lower Band'] = name['15 Day MA'] - (name['15 Day STD'] *
1.25)
    name['Lower Band'] = name['Lower Band'].shift(-15)

    name[['AdjClose', '15 Day MA', 'Upper Band', 'Lower
Band']].plot(figsize=(12, 6))
    plt.title('15 Day Bollinger Band for {}'.format(ticker))
    plt.ylabel('Price (USD)')
    plt.show()

    print(name)

    tickers = df.columns.values.tolist()

    name.fillna(0, inplace=True)

    # Eliminamos las 15 primeras porque no tienen medias moviles para
calcular el target

    # name = name[29:]
    name = name[14:(len(name)-15)]
    # Creación del target

    name['{}_target'.format(ticker)] = np.NaN

    for i in range(name.shape[0]):
```

```
        if(name['AdjClose'][i] > name['Upper Band'][i]):
            name['{}_target'.format(ticker)][i] = 1
        elif (name['AdjClose'][i] < name['Lower Band'][i]):
            name['{}_target'.format(ticker)][i] = -1
        else:
            name['{}_target'.format(ticker)][i] = 0

    print(name)

    vals = name['{}_target'.format(ticker)].values.tolist()

    str_vals = [str(i) for i in vals]

    print('Data spread:', Counter(str_vals))

    return tickers, df, name

# process_data_for_labels('EBAY')
#
# sys.exit()
def differences(x):
    return x[-1] - x[0]

def extract_featuresets(ticker):

    tickers, df, y = process_data_for_labels(ticker)

    df_vals = pd.DataFrame()

    for j in tickers:
        df_vals['{}_MA_15'.format(j)] =
df[j].rolling(window=15).mean()
        df_vals['{}_MA_5'.format(j)] = df[j].rolling(window=5).mean()
        df_vals['{}_V1_15'.format(j)] =
df_vals['{}_MA_15'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V1_5'.format(j)] =
df_vals['{}_MA_5'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V2_15'.format(j)] =
df_vals['{}_V1_15'.format(j)].rolling(window=2).apply(differences)
        df_vals['{}_V2_5'.format(j)] =
df_vals['{}_V1_5'.format(j)].rolling(window=2).apply(differences)

    print(df_vals)

    df_vals.fillna(0, inplace=True)

    print(df_vals)

    for column in df_vals.columns.values:
        for i in range(0, 15):
            df_vals['{}_{}d'.format(column, i)] =
df_vals[column].shift(i)

    print(df_vals)

    X = df_vals.values[14:(df_vals.shape[0] - 15)]  # feature set
    print(X.shape[0])

    y = y['{}_target'.format(ticker)].values #label

    print(len(y))
```

```
    print(y)

    X_train, X_test, y_train, y_test =
model_selection.train_test_split(X,

y,

test_size = 0.25,shuffle=False)

    np.save('X_tot_rnn.npy', X)
    np.save('y_tot_rnn.npy', y)

    np.save('X_train_sp_rnn.npy', X_train)
    np.save('X_test_sp_rnn.npy', X_test)
    np.save('y_train_sp_rnn.npy', y_train)
    np.save('y_test_sp_rnn.npy', y_test)

    return X, y, df

extract_featuresets('EBAY')
```

- Creación de labels y *features* para la red neuronal recurrente en test:

```
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
from sklearn import svm, neighbors, model_selection
from sklearn.ensemble import VotingClassifier, RandomForestClassifier


import sys


def process_data_for_labels(ticker):
    hm_days = 15 #days al futur pel preu per pujar o baixar x
    df = pd.read_csv('sp500_joined_closes_test.csv', index_col=0)

    name = ticker

    name = pd.DataFrame()
    name['AdjClose'] = df[ticker]

    print(name)

    name['15 Day MA'] = name['AdjClose'].rolling(window=15).mean()
    name['15 Day STD'] = name['AdjClose'].rolling(window=15).std()
    name['Upper Band'] = name['15 Day MA'] + (name['15 Day STD'] *
1.25)
    name['Upper Band'] = name['Upper Band'].shift(-15)
    name['Lower Band'] = name['15 Day MA'] - (name['15 Day STD'] *
1.25)
    name['Lower Band'] = name['Lower Band'].shift(-15)

    name[['AdjClose', '15 Day MA', 'Upper Band', 'Lower
Band']].plot(figsize=(12, 6))
    plt.title('15 Day Bollinger Band for {}'.format(ticker))
    plt.ylabel('Price (USD)')
    plt.show()

    print(name)

    tickers = df.columns.values.tolist()

    name.fillna(0, inplace=True)

    # Eliminamos las 15 primeras porque no tienen medias moviles para
calcular el target

    # name = name[29:]
    name = name[14:(len(name)-15)]
    # Creación del target

    name['{}_target'.format(ticker)] = np.NaN

    for i in range(name.shape[0]):
        if(name['AdjClose'][i] > name['Upper Band'][i]):
            name['{}_target'.format(ticker)][i] = 1
```

```
        elif (name['AdjClose'][i] < name['Lower Band'][i]):
            name['{}_target'.format(ticker)][i] = -1
        else:
            name['{}_target'.format(ticker)][i] = 0

    print(name)

    vals = name['{}_target'.format(ticker)].values.tolist()

    str_vals = [str(i) for i in vals]

    print('Data spread:', Counter(str_vals))

    return tickers, df, name

# process_data_for_labels('EBAY')
#
# sys.exit()
def differences(x):
    return x[-1] - x[0]

def extract_featuresets(ticker):

    tickers, df, y = process_data_for_labels(ticker)

    df_vals = pd.DataFrame()

    for j in tickers:
        df_vals['{}_MA_15'.format(j)] =
df[j].rolling(window=15).mean()
        df_vals['{}_MA_5'.format(j)] = df[j].rolling(window=5).mean()
        df_vals['{}_V1_15'.format(j)] =
df_vals['{}_MA_15'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V1_5'.format(j)] =
df_vals['{}_MA_5'.format(j)].rolling(window=5).apply(differences)
        df_vals['{}_V2_15'.format(j)] =
df_vals['{}_V1_15'.format(j)].rolling(window=2).apply(differences)
        df_vals['{}_V2_5'.format(j)] =
df_vals['{}_V1_5'.format(j)].rolling(window=2).apply(differences)

    print(df_vals)

    df_vals.fillna(0, inplace=True)

    print(df_vals)

    for column in df_vals.columns.values:
        for i in range(0, 15):
            df_vals['{}_{}d'.format(column, i)] =
df_vals[column].shift(i)

    print(df_vals)

    X = df_vals.values[14:(df_vals.shape[0] - 15)]  # feature set
    print(X.shape[0])

    y = y['{}_target'.format(ticker)].values #label

    print(len(y))
    print(y)
```

```
    np.save('X_test_test_rnn.npy', X)
    np.save('y_test_test_rnn.npy', y)

    return X, y, df

extract_featuresets('EBAY')
```

- Entrenamiento de la red neuronal:

```
import keras
import numpy as np
from collections import Counter
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from matplotlib import pyplot

from sklearn.model_selection import train_test_split

import sys

X_train = np.load('X_train_sp.npy')
X_test = np.load('X_test_sp.npy')
y_train = np.load('y_train_sp.npy')
y_test = np.load('y_test_sp.npy')

y_train = to_categorical(y_train, num_classes=3)
y_test = to_categorical(y_test, num_classes=3)

# sys.exit()

model = Sequential()
model.add(Dense(input_dim=3018, units=128, activation='relu'))
model.add(Dense(units=1500, activation='relu'))
model.add(Dense(units=3, activation='softmax'))

Adam = keras.optimizers.Adam(lr = 0.00001, decay=1e-6)

model.compile(loss='categorical_crossentropy',
              optimizer=Adam,
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=1000, batch_size=72, verbose=1,
validation_data=(X_test, y_test), shuffle=False)

predictions = model.predict(X_test)

print('Predicted spread:', Counter(np.argmax(predictions, axis=1)))

model.save('model_nn.h5')
```

- Entrenamiento de la red neuronal recurrente:

```
import numpy as np
from collections import Counter
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import to_categorical
from matplotlib import pyplot
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import sys

X_train = np.load('X_tot_rnn.npy')
# X_train = np.load('X_train_sp_rnn.npy')
X_test = np.load('X_test_sp_rnn.npy')
y_train = np.load('y_tot_rnn.npy')
# y_train = np.load('y_train_sp_rnn.npy')
y_test = np.load('y_test_sp_rnn.npy')
#
# X_train = X_train[:100,]
# X_test = X_test[:100,]
# y_train = y_train[:100]
# y_test = y_test[:100]

# print(y_train)
# sys.exit()


X_train = X_train.reshape((X_train.shape[0], 16, 3018))
X_test = X_test.reshape((X_test.shape[0], 16, 3018))

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

y_train = to_categorical(y_train, num_classes=3)
y_test = to_categorical(y_test, num_classes=3)

model = Sequential()
model.add(LSTM(512, input_shape= (X_train.shape[1],
X_train.shape[2])))
model.add(Dense(3, activation='softmax'))

# model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.summary()
from keras.optimizers import Adam
# Compile & run training
#model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.compile(optimizer=Adam(lr=0.00001, decay=1e-6),
loss='categorical_crossentropy', metrics=['accuracy'])

#-----------------------------------------------------------------------
```

```
-----------------------------------------#

from keras.callbacks import ModelCheckpoint

# define the checkpoint
filepath= "./weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
# checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1,
save_best_only=True, mode='min')

callbacks_list = [checkpoint]

# model.load_weights("weights-improvement-49-0.2095.hdf5")

model.fit(X_train, y_train,
          batch_size=72,
          # epochs=200,
          epochs=500,
          validation_split=0.25,
          callbacks=callbacks_list
          , shuffle=False
          )


predictions = model.predict(X_test)

print(predictions)

print('Predicted spread:', Counter(np.argmax(predictions, axis=1)))
```

- Resultados en test de la red neuronal:

```
from keras.models import load_model
import keras
import numpy as np
from sklearn import metrics
from collections import Counter
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import to_categorical

X = np.load('X_test_test.npy')
y = np.load('y_test_test.npy')

# y = to_categorical(y, num_classes=3)

model = load_model('model_nn.h5')

predictions = model.predict(X)

print('Predicted spread:', Counter(np.argmax(predictions, axis=1)))

str_vals = [str(i) for i in y]
print('Data spread:', Counter(str_vals))

print(y)

print(predictions)

print(metrics.accuracy_score(y, np.argmax(predictions, axis=1)))
```

- Resultados en test de la red neuronal recurrente:

```
from keras.models import load_model
import keras
import numpy as np
from sklearn import metrics
from collections import Counter
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import to_categorical

X = np.load('X_test_test_rnn.npy')
y = np.load('y_test_test_rnn.npy')

# y = to_categorical(y, num_classes=3)

model = load_model('guud.h5')

predictions = model.predict(X)

print('Predicted spread:', Counter(np.argmax(predictions, axis=1)))

str_vals = [str(i) for i in y]
print('Data spread:', Counter(str_vals))

print(y)

print(predictions)

print(metrics.accuracy_score(y, np.argmax(predictions, axis=1)))
```