Universidad Zaragoza
1542

Trabajo Fin de Grado

Deep Learning vs. Mecánica Estadística en el análisis de prevención de riesgos
Deep Learning vs. Statistical Mechanics for risk prevention analysis

Autor

Raúl Baigorri Martínez

Directores

Alfonso Tarancón Lafita
David Iñiguez Dieste

Facultad de ciencias
Grado en Física

28 de junio de 2019

ÍNDICE

1. INTRODUCCIÓN:

Hoy en día, las empresas o compañías informáticas como Google, IBM, Microsoft o Apple están centradas en el desarrollo de nuevas técnicas y algoritmos relacionados con la Inteligencia Artificial para manejar la gran cantidad y diversidad de datos que disponen, generados por la sociedad que utiliza sus plataformas en busca de información, por ello, debemos primero estudiar la teoría en la que se basan estos nuevos algoritmos para poder aplicarlos a nuestro trabajo.

1.1. INTRODUCCIÓN AL MACHINE LEARNING

Para entender bien los nuevos modelos de aprendizaje que están apareciendo en la nueva década debemos tener claros ciertos conceptos.

La Inteligencia Artificial (I.A.) es una "subdisciplina del campo de la informática que busca simular comportamientos inteligentes a través de la creación de máquinas"[1].

Dentro de esta podemos ver distintos tipos y métodos de comportamientos, en las que las máquinas suelen estar programadas para la realización de una tarea específica como el levantamiento de pesos, pegado de etiquetas, sellado, ordenamiento de objetos según determinadas características... funciones muy autómatas y específicas.

El Machine Learning es una "rama de la I.A. que busca conceder a las máquinas la capacidad de aprendizaje. Este aprendizaje, sucede mediante la generalización del conocimiento a partir de un conjunto de experiencias"[2].

La diferencia de esta rama con las demás es que en esta no se programa a la máquina para que realice una tarea específica, sino que se programa para que aprenda a través de una serie de datos, encuentre patrones o haga predicciones y aprenda a responder en base a lo aprendido.

"Dentro de esta rama existen diferentes técnicas o métodos para el aprendizaje utilizadas según el tipo de aplicaciones como los árboles de decisión, modelos de regresión, modelos de clasificación, técnicas de clusterización, y redes neuronales"[3].

Estas últimas son capaces de aprender de forma jerarquizada, la información se aprende por niveles y es solo con esta técnica con la que encontramos similitud con la capacidad humana de aprendizaje.

El Deep Learning es un modelo que trabaja con redes neuronales, imitando la conectividad del cerebro humano, clasificando conjuntos de datos y encontrando correlaciones entre ellos.

El objetivo de esta técnica es la creación de un modelo que intente imitar la realidad para dar el resultado pedido, por ello se necesita una interacción entre partes simples trabajando o interaccionando conjuntamente. Los tres tipos principales de redes neuronales los explicamos en el ANEXO 1[4, 5, 6].

1.2. CONTEXTO DEL TFG

Esta gran cantidad de datos que pueden obtener las distintas compañías informáticas y el valor que se puede extraer de ellos, han servido de conciencia para las grandes empresas, las cuales han tomado este recurso durante las últimas décadas.

Nosotros nos vamos a centrar en el sector de los seguros, donde gracias a la gran cantidad de datos que manejan las compañías de cada coche que van a vender y de cada comprador, pueden ayudarles a conocer mejor a sus clientes, los riesgos inherentes a cada contrato y actividad, pudiendo así establecer un precio del seguro estimado dependiente de las características de cada caso.

"Durante muchos años, las compañías han estado tratando con distintos modelos estadísticos y matemáticos para la determinación de los precios de los seguros dependiendo de los riesgos asociados a una póliza de seguro, utilizando distintas herramientas de *data mining* como modelos lineales generalizados hasta modelos bayesianos y de muy diversos tipos"[7].

Muy recientemente se están empezando a probar las técnicas de ***Deep Learning*** en algunos casos específicos del sector. Un ejemplo puede verse en la referencia (7) donde la compañía AXA predijo a los clientes que producían grandes pérdidas (en torno a un 1%) debido a sus accidentes de coche, que requerían pagos en torno a los 10.000 dolares. Con su detección sería capaz de optimizar el precio de sus pólizas.

Inicialmente se utilizaron métodos de Machine-Learning basados en *Random Forests*, el cual utiliza múltiples *Decision Trees* para el modelo predictivo, el cual llegaba a una precisión en torno al 40%. Con los modelos de redes neuronales llegaron a un porcentaje de predicción del 78% lo cual dio una ventaja significativa a la hora de optimizar costes y precios de seguros.

De este modo vemos que con Deep Learning podemos capturar las correlaciones entre las numerosas variables asociadas a cada cliente y las del contexto que le rodea con las variables de negocio que se intentan prever (el precio de la póliza, en nuestro caso) que son tan intrincadas que los modelos clásicos no siempre pueden capturarlas adecuadamente. En este sentido, el Deep Learning abre pues una nueva ventana de oportunidad para la creación de modelos predictivos y de caracterización de clientes.

1.3. DATOS UTILIZADOS

En nuestro caso disponemos de todas las variables de las que dispone la compañía sobre cada coche y cada comprador, desde las variables más directamente asociadas al propio riesgo asegurado (características del vehículo, edad y experiencia del conductor) hasta variables no asociadas con la determinación del precio del seguro como el número de kilómetros que típicamente va a recorrer el asegurado.

Para este trabajo, se ha hecho un filtro de las variables disponibles utilizado únicamente las variables que consideramos relevantes a la hora de determinar el precio del seguro (características del vehículo y del conductor principalmente) y limpieza de datos, ya que también se han descartado precios de seguros que no entraban dentro de la norma general, teniendo estos una desviación de la media muy por encima del resto.

Tenemos que indicar que trabajamos con una gran cantidad de datos, en total con 11.000.000 ejemplos de seguros proporcionados por tres compañías distintas de 43 ciudades y 6 modalidades. Por ello, hacemos una división de estos dependiendo de cada una de las tres etiquetas anteriores, llamando "modelo" a cada subdivisión de datos.

Obtenemos de esta manera un total de 489 modelos distintos, ya que cada compañía no nos ha proporcionado datos de todas las ciudades ni de todas las modalidades de los coches.

Aunque cada modelo tiene una serie de primas parecidas, estas tienen mucha variabilidad entre ciudades , tanto de valores como de número, ya que dependiendo de lo grande que sea la ciudad o el número de clientes de la compañía el modelo tendrá más o menos primas con las que podamos entrenar.

Además, debemos decir que tampoco tenemos todas las variables con las que trabajan las compañías, ya que vemos que en determinados datos, teniendo los mismos valores de todas las primas, obtenemos un precio del seguro distinto. Esto indica que las compañías poseen variables ocultas que no revelan y que nosotros no podemos manejar, como por ejemplo, la probabilidad de impago del cliente o probabilidad de accidente grave del cliente.

DATOS TOTALES	11.000.000
CIUDADES	43
COMPAÑIAS	3
MODALIDADES	6
TOTAL MODELOS	489

Tabla1: Esquema de la complejidad de los datos con los que trabajamos.

Los datos de cada modelo los guardamos en archivos con extensión ".csv" guardando cada ejemplo por filas y cada variable de cada ejemplo por columnas.

1.4. OBJETIVOS

Aunque sabemos que el Deep Learning es un método de predicción muy eficiente para algunas tareas como la predicción de caracteres o reconocimiento de objetos, una de las dudas más generales es la eficacia de los métodos de Deep Learning frente a la utilización de métodos estadísticos en la predicción de riesgos.Por ello vamos a comparar la eficacia en la predicción del precio de seguros mediante dos métodos distintos. Un primer método basado en la Mecánica Estadística y un segundo basado en el Deep Learning.

Obtendremos distintos resutados con cada método y los compararemos viendo el funcionamiento de cada uno, sus características, dificultades y condiciones en las que opera mejor cada uno de ellos

haciendo un estudio del error obtenido según el número de datos que disponemos para su entrenamiento.

2. MÉTODOS A COMPARAR:

2.1. ESTRUCTURA DE NUESTROS DATOS

Como ya hemos dicho anteriormente, disponemos de datos de seguros de coches de distintas ciudades separados en distintas compañías y modalidades.

Para la comparación "absoluta"[1]* entre los dos programas de predicción de precios vamos a trabajar con los mismos datos en ambos casos, dividiendo los datos proporcionados por las compañías de seguros (ya filtrados, limpiados y separada la variable del precio del seguro en un archivo distinto del resto de las variables) en datos de entrenamiento o train (80% de los datos totales da cada modelo) y datos de test o de control[2]* (20% de los datos totales da cada modelo).

Estos archivos los separaremos a su vez en otros dos, uno que contendrá la variable del precio para cada coche de cada modelo y otro que contendrá las 20 variables más relevantes que hemos considerado. Así pues tenemos una subdivisión en archivos de cada uno de:

Train_Data; Train_Precio; Test_Data; Test_Precio.

Las variables que van a intervenir en el precio del seguro pueden ser de distintos tipos como números enteros, reales o boleanos (0,1) que especifican si un coche determinado tiene cierta equipación o posee una característica determinada o no. Las variables son las siguientes:

-Booleanos: Formados por las distintas covers del coche y por la variable que determina si la compañía tiene zona de expedición o no.

cover_3, cover_4, cover_5, cover_6, cover_7, cover_8, cover_12, cover_14, expedition_zone

-Variables reales: El precio del coche y la variable deductible.

vehicle_price , deductible, (seguro con franquicia).

-Variables enteras: Formado por el resto de variables que indican el valor del precio

vehicle_years, driver_holder_type, vehicle_power,driver_age,experience,vehicle_type,km,

garage_type, years_without_accidents.

[1]* Ponemos "absoluta" ya que la comparación no va a ser totalmente igual, debido a que con un método intentaremos hacer la predicción ajustando los distintos parámetros, utilizando todos los datos de un archivo y con el otro operaremos utilizando solamente un grupo de estos en cada iteración. Además para que la comparación fuera absoluta el tiempo de programación de la CPU debería ser el mismo en ambos métodos, cosa que tampoco se cumple.

[2] *Nos referiremos a estos archivos con los nombres de train o entrenamiento y test o control indistintamente.

En cada uno de estos archivos tenemos N datos o seguros reales de coches que ha ofrecido la compañía divididos en distintas filas. Según el archivo con el que estaremos operando lo denominaremos N_train o N_test.

2.2. CARACTERÍSTICAS GENERALES

En ambos métodos, los cuales luego serán explicados más detalladamente, entrenamos primero o hacemos predicciones con los datos de train, obtenemos un resultado y un error comparando el precio obtenido con el precio real, y cambiando algunos parámetros de la función que determina el precio, hacemos que este error disminuya con cada iteración o paso de entrenamiento.

Aunque los métodos sean distintos, ya que uno trabaja con redes neuronales y otro trabaja viendo el problema como si la variable del precio fuera una función de las otras 20 variables restantes, y además no se entrenan de igual manera, estos operan de forma similar, comparando los resultados del valor predicho del precio con el valor real y buscando un valor menor del error entre ellos con el cambio de los parámetros de la red neuronal o de la función de aproximación.

Después de entrenar cada modelo con los datos en el archivo de train (como su propio nombre indica) obtenemos un error medio (MAE: Mean Absolute Error) y del tanto por ciento (MAPE: Mean Absolute Percentage Error) con los datos proporcionados en los archivos test. Esto lo podemos hacer al final del programa, después de haber entrenado suficiente con nuestros datos, o mientras vamos entrenando, viendo la evolución del error obtenido.

3. MECÁNICA ESTADÍSTICA:

El algoritmo de este método lo escribiremos en el lenguaje de programación C, el cual lo he estado utilizando durante el grado y estoy bastante familiarizado con él.

Como ya hemos dicho en el apartado anterior, en este algoritmo tratamos la variable del precio como una función aproximada del resto de variables que suponemos que influyen de manera distinta en la determinación de nuestro resultado.

Los valores más importantes con los que trabajamos en este programa son el MAE y el MAPE con los que iremos viendo la evolución de nuestro entrenamiento, ya que hacen referencia al error entre el valor predicho del precio y el valor real. El primero es un error absoluto y el segundo es un error en tanto por ciento como su nombre indica. Vienen dados por las siguientes fórmulas:

$$MAE = \sum_{n=0}^{N} |Precio_{Estimado} - Precio_{Real}|$$

$$MAPE = \sum_{n=0}^{N} \frac{|Precio_{Estimado} - Precio_{Real}|}{Precio_{Real}}$$

Aunque durante todo el grado hemos estado trabajando con el error cuadrático medio (RMSE), en este método entrenaremos y operaremos con estos dos errores ya que son con los que trabajan las empresas de seguros y ya que los resultados obtenidos son mejores, como podemos ver en el ANEXO 3.

$$RMSE = \sqrt{\frac{\sum_{n=0}^{N}(Precio_{Estimado} - Precio_{Real})^2}{N}}$$

Aunque trabajaremos principalmente con el error MAPE, generalmente este lo daremos y representaremos en %.

3.1. PROGRAMA PRINCIPAL.

Lo primero que hacemos es crear dos ficheros (Evolución y Resultados), con el nombre de cada Ciudad especificada con la compañía y la modalidad, para poder identificarla, en los que iremos escribiendo los resultados obtenidos del MAE y el MAPE, tanto de los datos de entrenamiento como de los datos de control cada cierto número de iteraciones, para ver su evolución.

Lo segundo que hacemos es leer los dos archivos ".csv" tanto de train como de test, pero esto lo hacemos de distinta manera y con dos funciones diferentes ya que con unos vamos a entrenar nuestros parámetros y con otros solo vamos a evaluar nuestras predicciones:

-Lectura de datos Train:

Leemos y guardamos todas las variables del archivo Train_Data en un tensor de tipo double: x[N_{Train}][20], donde denotaremos a cada variable específica como x_i^n, donde i se refiere al tipo de variable y n al número de seguro vendido de todos los N_Train proporcionados.

Después de esto hacemos varias operaciones por columnas del tensor, determinando si alguna variable toma el mismo valor para todos los datos proporcionados de ese modelo o si alguna toma siempre el valor 0 y mostrándolo por pantalla, ya que estas variables serán inútiles para el cálculo de la función del precio, ya que al ser todas iguales no ofrecerán ninguna diferencia entre el cálculo de los diversos precios, sin poder hacer ningún tipo de estadística con ellas.

Ahora procedemos a convertir nuestras variables x_i^n en variables normalizadas, para poder fijar los coeficientes de la función aproximción en un mismo rango, ya que al tener las distintas variables tienen distintas escalas, si lo quisiéramos hacer sin normalizarlas, también deberíamos fijar distintas escalas para los coeficientes.

Primero calculamos la media de todas las columnas, significando la media de los valores que toma cada variable de nuestro seguro y la suma de los cuadrados:

$$MEDIA_i \equiv \overline{X_i} \equiv \langle X_i \rangle = \frac{1}{N_{Train}} \sum_{n=0}^{N_{Train}} x_i^n$$

$$SC_i \equiv \overline{X_i^2} \equiv \langle X_i^2 \rangle = \frac{1}{N_{Train}} \sum_{n=0}^{N_{Train}} (x_i^n)^2$$

Así podemos calcular la desviación estándar:

$$\sigma_i = \sqrt{\langle X_i^2 \rangle - \langle X_i \rangle^2}$$

Y trabajar con una nueva variable normalizada para cada variable del seguro, que tendrá una media de cero y desviación 1:

$$\phi_i^n = \frac{X_i^n - \langle X_i \rangle}{\sigma_i}$$

Seguidamente, leemos y guardamos en un array los datos del fichero de precios y normalizamos la variable del mismo modo que hemos hecho para las 20 variables anteriores.

-Lectura de datos Test:

En este caso volvemos a leer y guardar todas las variables de los archivos Data en un tensor de tipo double: $x'[N_{Test}][20]$, donde denotaremos a cada variable específica como x'^n_i, donde i se refiere al tipo de variable y n al número de seguro vendido de todos los N_Test proporcionados.

3.2. FUNCION DE APROXIMACIÓN.

Inicialmente, la función que hemos escogido para determinar el precio la dividimos en una parte analítica, que es el desarrollo en serie de Taylor hasta el tercer término, más la suma de otros términos no analíticos, ya que sabemos que estos términos tienen presencia en este tipo de análisis, los cuales los explicamos más adelante. La fórmula de la función viene dada por la siguiente ecuación:

$$f(x_1, x_2, x_3 \ldots\ldots x_{20}) = \sum_{i=0}^{20} \left(\frac{\partial f}{\partial x_i}\right)\phi_i + \theta_1\left(\sum_{i=0}^{20} g_1\phi_i\right)\sum_{i=0}^{20} J_F^i sen(J_{FOURIER}^i \times \phi_i) + \frac{1}{N_{Train}}\sum_{i=0}^{20}\sum_{J=0}^{20}\left(\frac{\partial f}{\partial x_i \partial x_j}\right)\phi_i\phi_j + \frac{1}{N_{Train}}\theta_2\left(\sum_{i=0}^{20} g_2\,\phi_i\phi_j\right) + \frac{1}{N_{Train}^2}\sum_{i=0}^{20}\sum_{j\neq i}\sum_{k\neq i\neq j}\left(\frac{\partial f}{\partial x_i \partial x_j \partial x_k}\right)\phi_i\phi_j\,\phi_k$$

Donde los $\phi_i's$ son las variables de entrada normalizadas, constantes durante todo el proceso, y los demás parámetro multiplicativos o funcionales son variables y los que vamos a ir ajustando para encontrar el mínimo.

Al multiplicar la función por σ_{PRECIO} y sumarle la media obtenida con los datos de entrenamiento, entrará en el rango de los precios reales y obtendremos la *función de estimación* del precio.

Los términos significan:

- $\sum_{i=0}^{20}\left(\frac{\partial f}{\partial x_i}\right)\phi_i$: Hace referencia a la aproximación en primer orden en torno a un mínimo de una función analítica en las proximidades del mínimo. También lo podemos ver como los distintos pesos de las distintas características de cada coche, significando que no todas tienen la misma importancia individualmente a la hora de determinar el precio final.

- $\theta_1\left(\sum_{i=0}^{20} g_1\phi_i\right)$: Es un término no analítico que hace referencia a una función escalón en función de un sumatorio pesado de las distintas variables, de forma que si este supera un

determinado número adquiere u valor positivo y en caso contrario adquiere el mismo valor absoluto pero con signo negativo.
Este término es parecido a la activación de neuronas explicada en el ANEXO 4, por ello nos referiremos a él como término de activación.

- $\sum_{i=0}^{20} J_F^i sen(J_{FOURIER}^i \times \phi_i)$: Este término nos ayuda a obtener más rápido la forma de la función en torno a un mínimo, ya que sabemos que en torno a este, la función posee términos sinusoidales.

- $\sum_{i=0}^{20} \sum_{J=0}^{20} \left(\frac{\partial f}{\partial x_i \partial x_j}\right) \phi_i \phi_j$: Hace referencia a la derivada parcial de segundo orden en torno al mínimo de la función de estimación del precio como la aproximación en segundo orden. También la podemos ver como las relaciones entre las distintas características del seguro dos a dos a la hora de determinar el precio.

- $\theta_2 \left(\sum_{i=0}^{20} g_2 \phi_i \phi_j\right)$: Al igual que el tercer término del sumatorio, este es un término escalón, pero en este caso relacionando las distintas características por parejas.

- $\sum_{i=0}^{20} \sum_{j \neq i} \sum_{k \neq i \neq j} \left(\frac{\partial f}{\partial x_i \partial x_j \partial x_k}\right) \phi_i \phi_j \phi_k$: Hace referencia a la derivada parcial de tercer orden en torno al mínimo de la función de estimación del precio como la aproximación en el desarrollo en serie hasta tercer orden. También la podemos ver como las relaciones entre tres características del seguro cualesquiera a la hora de determinar el precio.

Las distintas derivadas de la función, coeficientes del término sinusoidal y funciones de activación son variables double generadas aleatoriamente en cada iteración de entrenamiento. En caso de que mejoren la predicción del precio de los seguros serán sustituidas por sus valores anteriores o no, siguiendo un algoritmo que explicaremos en el ANEXO 2.

La derivada de primer orden al igual que los coeficientes J's del término sinusoidal del sumatorio son vectores con tantas componentes como características tenemos de los coches. La derivada de segundo orden será un Tensor 20×20 y la derivada de tercer orden será un Tensor $20 \times 20 \times 20$.

Los términos de activación son variables double.

3.3. PROCESO DE ENTRENAMIENTO

Vamos a entrenar a nuestro modelo en una serie de iteraciones, entendiendo entrenar como ajustar nuestros parámetros de la función en cada iteración, dando valores aleatorios a los distintos coeficientes de la función de aproximación en cada una de ellas y quedándonos con estos valores aleatorios nuevos si el MAPE disminuye respecto al obtenido en la iteración anterior, buscando así el mínimo de este error que suponemos que será parecido al obtenido con los datos Test, ya que pertenecen a un mismo modelo.

Esta búsqueda del mínimo la haremos utilizando el Método de *Simulated Annealing*[8]. Por ello nuestra función MAPE será la energía de nuestro sistema.

Como no sabemos la forma que tiene nuestra energía, y seguramente no será suave, sino que al ser una función de veinte variables es posible que tenga varios mínimos relativos o que presente discontinuidades, no nos basta con empezar con cualquier solución e ir cambiando ligeramente la misma aceptando los cambios si la energía disminuye, sino que permitiremos fluctuaciones térmicas al sistema para que este sea capaz de recorrer toda nuestra función energía , pasar por varios mínimos relativos e ir buscando estos mínimos al bajar la temperatura progresivamente, siendo probablemente próximos al mínimo absoluto. Para ello utilizamos el *Algoritmo de Metrópolis*[9] y utilizaremos la *Mecánica Estadística de Boltzman*[10] para la probabilidad de una determinada configuración de nuestro sistema.

De este modo tenemos dos parámetros que dictarán la evolución de nuestro sistema principalmente:

-Delta (δ): Dicta el rango en el que se mueven los números aleatorios que dan valor a los coeficientes de la función de aproximación.

Con el objetivo de recorrer todos los valores de nuestra función de la energía, este empezará tomando valores en los que los números aleatorios generados uniformemente (que darán valor a los distintos coeficientes de la función de aproximación) estén en el rango (-0.5,0.5) ya que recordemos que a medida que pasan las iteraciones su valor irá disminuyendo linealmente, ya que suponemos que al encontrarnos cerca del mínimo absoluto de la función energía los cambios en los coeficientes deberán ser pequeños.

-Beta (β): Hace referencia a la Temperatura en la que se encuentra nuestro sistema, concretamente es proporcional al inverso de la Temperatura.

Al igual que el parámetro anterior, tiene el objetivo de permitirnos recorrer todos los valores de nuestra función energía. Este empezará tomando valores pequeños que significarán que estamos a una temperatura alta y nos permitirá recorrer un rango amplio de nuestra función, ya que debido a que la temperatura es alta, los valores de los coeficientes nuevos serán aceptados en el mayor de los casos (como vemos en el ANEXO 2). Su valor inicial deberá ser aquel que multiplicado por la variación energética inicial de $1 \rightarrow \beta_{INICIAL} \times \Delta E_{INICIAL} \approx 1$.

A medida que vamos iterando, su valor irá aumentando linealmente también, simulando un enfriamiento de nuestro sistema que haga que disminuya la aceptancia y nuestro punto se mueva en torno al mínimo más cercano.

Hay un parámetro que revisa o controla la evolución y, si es preciso, cambia el valor de los parámetros descritos anteriormente, este es la Aceptancia:

-Aceptancia: La aceptancia es el cociente del número de cambios tentativos que han sido aceptados entre el total después de realizar un proceso *sweep.*

Este proceso significa recorrer secuencialmente el vector que guarda los términos de primer orden de nuestra función, correspondientes a las distintas características según el número de iteración, recorrer secuencialmente los Tensores por filas, siendo el número de columna aleatoria en cada iteración y dar un número aleatorio para cada uno de los tres índices del Tensor de Einstein.

Sabemos cómo debe evolucionar nuestra aceptancia en la teoría. Al principio deseamos que esta sea casi igual que 1, ya que queremos explorar todas las configuraciones posibles o valores que puede

tomar nuestra función de energía, y a medida que vamos acercándonos al mínimo de nuestra función suponemos que esta irá disminuyendo hasta caer casi a cero cuando estemos muy próximos del mínimo. De este modo, si la vamos controlando podemos ser capaces de que el sistema evolucione como nosotros queramos.

Así imponemos que si esta toma valores pequeños al principio de la evolución del sistema (lo cual no queremos) disminuimos un poco el valor de β, o aumentamos la energía del sistema, siendo así más probable que el sistema pase por distintas configuraciones. Además programamos la disminución del valor de β en la siguiente iteración si la acaeptancia del *sweep* anterior ha sido muy baja.

También controlamos el valor del parámetro β. Como vamos disminuyendo su valor si la aceptancia es demasiado baja, este puede llegar a tomar valores muy pequeños, así que acotamos su valor en un mínimo.

3.4. EVOLUCIÓN DE LOS COEFICIENTES Y RESULTADOS GUARDADOS AL FINAL DE CADA ITERACIÓN

Necesitamos partir de una configuración inicial, la cual será la dada por todos los coeficientes de la función de aproximación igualados a cero, con lo cual, nuestra predicción inicial del precio será para todos los datos Test el precio medio obtenido con los datos Train.

Así, obtendremos un error inicial, operando con los datos de entrenamiento, del valor de la dispersión de nuestros datos. Para el error de Test obtendremos otro error, ya que estos tienen una media y desviación distinta, por lo que al dar siempre el precio medio obtenido con los datos de entrenamiento tendremos un resultado diferente aunque parecido, ya que los distintos datos pertenecen a un mismo modelo.

Una vez que entramos en el bucle de entrenamiento nos guardamos los distintos coeficientes en un archivo denominado “Configuración” en el cual reescribiremos los términos obtenidos al final de cada iteración o proceso *sweep* para partir de ellos en la siguiente.

Además de guardarnos estos coeficientes nos guardamos en el el archivo “Evolución” el valor de Beta, aceptancia, MAPE_Train y MAPE_Test.

En cada iteración comparamos los errores obtenidos con los anteriores guardándonos el error mínimo, tanto MAPE como MAE, obtenido con los datos de test, de forma que este será el que nos importará al final del entrenamiento, ya que será la mejor predicción que hemos obtenido.

3.5. MEJORAS EN EL PROGRAMA:

-Al principio entrenábamos los datos con el error cuadrático medio y veíamos la evolución del MAPE frente al número de iteraciones. Como debíamos trabajar con este último error, ya que es con este con el que trabajan las empresas de seguros decidimos entrenar nuestro programa también con este error, obteniendo mejores resultados.

-Después de una serie de pruebas para ver como evolucionaban los errores frente al número de iteraciones y frente al número de términos escogidos en la serie de nuestra función, observamos que si queríamos ejecutar los archivos con un tiempo de CPU que no fuera muy alto, el valor óptimo para el número de iteraciones era de 1500 y entrenamos escogiendo solamente los tres primeros términos de

la serie de Fourier. En el ANEXO 3 podemos ver un estudio más detallado de estos aspectos y una evolución de entrenamiento para un modelo grande con un número elevado de iteraciones, entrenándolo con todos los términos de la función aproximación.

El problema de los términos siguientes descartados es que además de multiplicar el tiempo de ejecución del programa, les cuesta mucho termalizar y su contribución empieza a tener efecto varias iteraciones después de empezar a trabajar con ellos en la estimación del resultado.

-Creación de programas adicionales con el objetivo de clasificar, ordenar y juntar los archivos obtenidos.

-Modificación del programa principal para la eficacia de la obtención de datos en el que en vez de ir compilando los distintos modelos uno por uno se le da como entrada un archivo el cual contiene los nombres de varios modelos y el cual ejecuta iterativamente.

4. DEEP LEARNING:

En este caso hemos trabajado con el lenguaje de programación Python[11] para la programación de nuestro código, que realiza toda la gestión de los procesos, tanto de entrada y salida de datos como de entrenamiento y test de los modelos, utilizando las librerías de Deep Learning de Tensorflow[12, 13].

Con este lenguaje no he trabajado a lo largo de la carrera, y ha sido necesario un estudio previo para su entendimiento y manejo.

En este caso, entrenaremos cada modelo con una red neuronal profunda (DNN) constituida por una capa externa de 20 neuronas, en la que se introducen las 20 variables de cada ejemplo del modelo, una capa externa, en la que obtenemos el precio estimado del seguro, y 5 capas ocultas, cada una con [150, 75, 50, 20, 10] neuronas consecutivamente y totalmente conectadas entre ellas. Estos números han sido elegidos a base de prueba y error, ya que no conocemos ningún estudio ni fórmula que dicte el número óptimo ni de capas ni de neuronas para la predicción de este caso.

En este método también devolveremos los errores definidos previamente de MAE y MAPE para evaluar la precisión de nuestros resultados, pero trabajaremos con el error cuadrático medio (RMSE) para entrenar nuestro modelo, como podemos ver en el ANEXO 5.

Al igual que en el método anterior, trabajaremos principalmente con el error MAPE, pero este lo daremos en % en la representación de la mayoría de gráficas.

4.1. PROGRAMA PRINCIPAL

El programa tiene distintas librerías y archivos, utilizados para el manejo de datos, explicados en el ANEXO 4[12,13,14,15,16,17,18,19]. El programa principal es el archivo llamado TRAIN_MODEL.py.

Este contiene el código para entrenar los distintos modelos, incluyendo otros archivos para cada una de sus distintas funcionalidades.

Lo primero que hacemos es indicar qué modelo vamos a entrenar, especificando su nombre, y crear una carpeta con su nombre en un determinado directorio, en la que guardamos los distintos parámetros de la red neuronal que vamos entrenando.

Después creamos un archivo llamado “estimación_’nombre del modelo’.dat” en el que guardaremos los valores del número de datos de entrenamiento, de testeo y los valores del MAPE, en tanto por ciento, y del MAE, obtenidos en la última iteración.

Mediante el archivo IRIS_DATA.py accedemos a cada archivo de nuestro modelo, obteniendo así cuatro variables DataFrame con los datos de cada archivo (train_x, train_y, test_x, test_y).

Seguidamente, definimos el número de iteraciones con el que vamos a entrenar nuestro modelo, siendo este distinto para cada modelo, en contra que en el método estadístico en el que era el mismo para todos los modelo. Su razón la podemos ver en el ANEXO 6.

Así pues, el número de iteraciones depende del número de datos con el que vayamos a entrenar, teniendo la ecuación siguiente:

$$train\_steps = int(N\_train * 25/batch\_size)$$

Donde la función $int()$ devuelve el número entero del número real entre corchetes, restándole a este último su parte decimal, N_train es el número de datos de entrenamiento y $batch\_size$ el número de datos que escogemos en cada “batch” de entrenamiento del modelo, siendo este número igual para todos los modelos y teniendo un valor de 230.

Una vez que tenemos definidas todas estas variables pasamos a definir nuestra red neuronal, utilizando el archivo LOAD_MODEL.py y utilizando otra vez el archivo IRIS_DATA.py, entrenamos nuestros datos de entrenamiento y obtenemos las predicciones, dadas por la red, de los datos de test.

Estas predicciones las guardamos en un array que utilizamos para compararlas con los datos reales de los precios.

Finalmente, calculamos los errores MAPE y MAE y guardamos estas variables en el archivo “estimación_nombre del modelo.dat”.

4.2. ENTRENAMIENTO DE LA RED:

En este archivo también especificamos el optimizador con el que trabajará la red neuronal, que en nuestro caso será Adagrad optimizer. Al igual que en el método anterior, lo que intentamos hacer es minimizar el valor de una función, llamada función de coste, que explicamos en el ANEXO 5[20, 21, 22].

4.2.1. ADAGRAD OPTIMIZER[23, 24] :

Este optimizador se basa en el método del descenso del gradiente y especifica un buen ratio de aprendizaje.

En mecánica estadística este ratio suele ser β, definido en el la explicación del entrenamiento del método estadístico, el cual es inversamente proporcional a la temperatura.

Otro método seguro de hacerlo, es escoger, para este ratio, el valor de una constante dividida por el tiempo, o computacionalmente hablando, el número de iteraciones que llevamos entrenando a nuestro modelo: $\alpha = \frac{C}{t}$, así la función de coste convergerá, pero esta asignación no es muy buena, ya que el proceso es muy lento.

En el algoritmo Adagrad el ratio depende de la concavidad de la función, de forma que si estamos en un punto donde la función es muy cóncava, indicando que estamos cerca de un mínimo, el ratio será muy pequeño, y si la función es casi plana, el ratio será muy grande.

Si trabajamos con varias variables, cada variable (o dimensión de la función) debe tener una tasa de aprendizaje distinta, ya que una función multidimensional no tiene el mismo gradiente para cada dimensión.

Empezamos con un vector $\vec{s}$ que inicializamos a cero: ($\vec{s}_{t=0} = \vec{0}$)
En cada iteración hacemos la operación: $\vec{s}_{t+1} = \vec{s}_t + \left(\vec{\nabla}F_{COSTE_t}\right)^2$.
Donde $\vec{\nabla}F_{COSTE_t}$ es el gradiente de la función de coste en el tiempo t y $\left(\vec{\nabla}F_{COSTE_t}\right)^2$ el vector $\vec{\nabla}F_{COSTE_t}$ con cada componente al cuadrado, no la multiplicación de vectores.
Así la variación de cada peso de cada variable evolucionará en cada iteración de la siguiente manera: $\vec{w}_{t+1} = \vec{w}_t - \frac{\alpha\vec{\nabla}F_{COSTE_t}}{\sqrt{\vec{s}_t+\varepsilon}}$
Donde α es la tasa de aprendizaje inicial y ε es un valor pequeño para no dividir por cero en la primera iteración cuando $\vec{s}_{t=0} = \vec{0}$.

Esta ecuación significa que cuando el gradiente sea muy grande, el paso va a ser muy pequeño, debido a que está dividiendo, y cuando nos encontremos en superficies muy inclinadas, daremos pasos pequeños hacia el mínimo de la función.

Como trabajamos en muchas dimensiones, puede ser que los valores del gradiente en estas sean muy dispares, dando pasos largos en una determinada dirección y cortos en otras en una misma iteración.

4.3. MEJORAS EN EL PROGRAMA:

Como más tarde veremos, este programa no funciona del todo bien para los modelos que tienen pocos datos de entrenamiento, y siendo falsa también la suposición de que el valor más pequeño obtenido del MAPE es el de la última iteración del entrenamiento.

Para estudiar por qué con los archivos pequeños obtenemos unos valores altos del MAPE y observar como entrena la red neuronal hemos realizado una serie de modificaciones en el archivo TRAIN_MODEL.py:

-Predicción tanto de los precios del archivo test como del archivo train y cálculo de las variables MAPE y MAE de ambos archivos, para ver la evolución de estos dos dependiendo del número de iteraciones.

- Creación de un archivo llamado “Evolución_nombre del modelo.dat”, en el que iremos guardando los errores MAPE y MAE tanto de los archivos de entrenamiento como de testeo cada cierto número de iteraciones, denominado “save_steps”, que tendrá el valor de:

$$\text{save\_steps} = int\left(\frac{\text{train\_steps}}{50}\right)$$

Ya que queremos dibujar una gráfica en la que se vea bien la evolución de nuestro entrenamiento, considerando así que con 50 puntos (o alguno más, ya que despreciamos el valor decimal del cociente) tendremos una buena visualización de esta.

-Adición de dos nuevas variables en el programa llamadas: error_minimo y precio_minimo.

En cada iteración evaluaremos los valores de MAPE y MAE de los archivos test guardándonos estos valores en las variables definidas si el valor del MAPE es el mínimo encontrado y si el número de iteraciones es mayor que cierto número ya que al principio estos errores pueden depender mucho del "batch" utilizado para la predicción.

-Creación de un nuevo archivo Python llamado "EJECUTA_VARIOS_MODELOS.py" para la eficacia de la obtención de datos.

Este archivo ejecuta sucesivamente varios modelos, leyendo, de un archivo de texto, el nombre de estos y guardando, en otro archivo de texto, el nombre del modelo entrenado, el número de datos de enternamiento y de test, y los errores mínimos de MAPE y MAE del archivo de test, obtenidos separando los datos de distintos modelos en distintas filas.

5. COMPARACIÓN DE RESULTADOS OBTENIDOS POR LOS DOS METODOS:

En este apartado procederemos a la comparación entre los dos métodos explicados anteriormente, analizaremos los resultados y compararemos sus entrenamientos y valores de los resultados.

Para ello, primero dibujamos los resultados obtenidos del error MAPE mínimo para todas las ciudades, dependiendo del número de datos con que hayamos entrenado a nuestro "modelo", ya que cuantos más datos de entrenamiento tengamos, mejor estadística podremos hacer y mejores predicciones obtendremos. Debemos indicar que, aunque en el método de Mecánica estadística sí que nos quedamos con el error minimo obtenido en la evolución del proceso, en el método de Deep_Learning nos quedamos con el error final, debido a que suponemos que este será el mínimo, aunque más adelante veremos que no es así.

Los resultados han sido los mostrados en la siguiente figura:

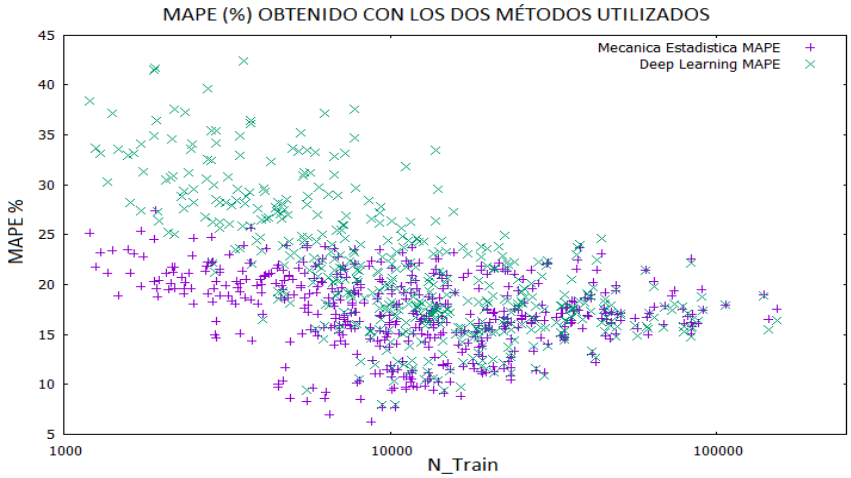

Figura 5.1: En esta figura podemos ver los valores mínimos obtenidos del error MAPE con los dos métodos en función del número de datos con el que se entrena cada modelo dibujando el eje x en escala logarítmica para su mejor visualización.

Observando la figura, podemos ver que los datos obtenidos con el método de Deep Learning son mucho menos precisos que los obtenidos por el estadístico (Mecánica Estadística), cuando entrenamos con pocos datos. Esta diferencia va disminuyendo a medida que aumentamos el número de datos de entrenamiento, hasta ser algo menores los obtenidos por el primer método mencionado.

Si observamos más detalladamente la figura, vemos que los errores empiezan a empeorar a partir de un número de unos 10000 datos de entrenamiento aproximadamente.

Por regla general vemos que en los dos métodos se cumple el hecho de que obtengamos mejores resultados dependiendo del número de datos con el que entrenamos, estando en concordancia con la teoría de que al tener más datos podemos realizar una estadística mejor.

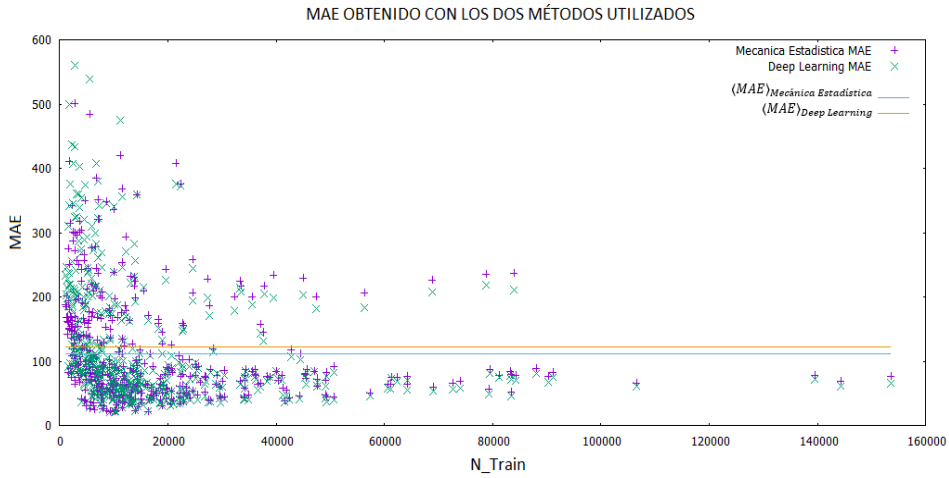

Figura 5.2: En esta figura podemos ver los valores mínimos obtenidos del MAE con los dos métodos en función del número de datos con el que se entrena cada modelo. También vemos una media del error de cada uno de los dos grupos de datos.

Con el valor del MAE pasa lo mismo, pero este error no se ve tan claro, ya que es el error absoluto y la diferencia entre estos comparada con el valor real no se aprecia tanto en la figura como en la anterior.

Como sabemos que no es lo mismo obtener un error determinado entrenando con un número de N_Train bajo (2.000 primas), que con uno alto (40.000 primas), teniendo más relevancia el segundo, ya que ha sido mejor calculado, debemos realizar la media ponderada de los resultados obtenidos mediante ambos métodos:

$$\langle MAPE \rangle = \frac{\sum_{i=0}^{N\_Modelos} MAPE_i \times N\_Train_i}{\sum_{i=0}^{N\_Modelos} N\_Train_i}$$

$$\langle MAE \rangle = \frac{\sum_{i=0}^{N\_Modelos} MAE_i \times N\_Train_i}{\sum_{i=0}^{N\_Modelos} N\_Train_i}$$

Teniendo unos valores de:

$$\langle MAPE \rangle_{Mecánica\ Estadística} = (16.95 \pm 0.91)\% \qquad \langle MAPE \rangle_{Deep\ Learning} = (18.32 \pm 0.87)\%$$
$$\langle MAE \rangle_{Mecánica\ Estadística} = (101.7 \pm 6.1)\ € \qquad \langle MAE \rangle_{Deep\ Learning} = (104.9\ \pm 5.8)\ €$$

Y obteniendo una diferencia entre los métodos de:

$$E_{MAPE} = \frac{\langle MAPE \rangle_{Mecánica\ Estadística} - \langle MAPE \rangle_{Deep\ Learning}}{\langle MAPE \rangle_{Mecánica\ Estadística}} \times 100 \cong 8\%$$

$$E_{MAE} = \frac{\langle MAE \rangle_{Mecánica\ Estadística} - \langle MAE \rangle_{Deep\ Learning}}{\langle MAE \rangle_{Mecánica\ Estadística}} \times 100 \cong 2\%$$

Así pues, aunque ambos métodos nos dan resultados bastante parecidos, sabemos que el método de redes neuronales no funciona muy bien cuando tenemos pocos datos de entrenamiento, por ello pasamos a estudiarlo.

5.1 DIFERENCIAS EN LA EVOLUCIÓN DE DISTINTOS MODELOS COMPARANDO LOS DOS MÉTODOS

Aunque en este apartado veremos los distintos resultados fijándonos solo en un modelo para cada clase, en el ANEXO 7 proporcionamos más figuras en las que podemos ver más ejemplos.

MODELOS GRANDES

Con modelos grandes nos referimos a modelos que contienen un número de datos de entrenamiento mayor que 20.000. Veamos cuál es su evolución utilizando los dos métodos:

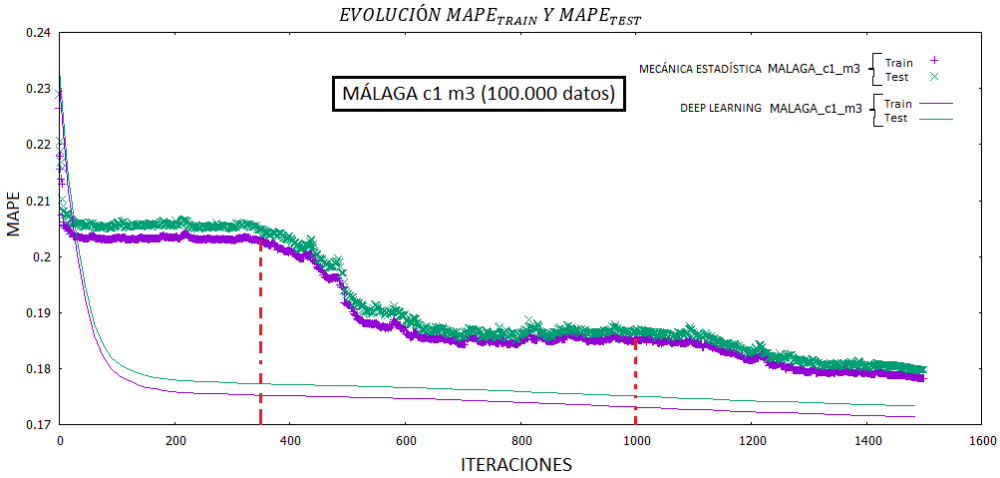

Figura 5.1.1: Representación de la evolución del modelo Málaga c1 m3 con 100.000 datos de entrenamiento, en el que vemos que el método de D.L. llega a errores más bajos que el método M.E.

En esta figura vemos que los modelos que contienen un número grande de datos de entrenamiento, dan mejores resultados con el método de Deep Learning, como nos esperábamos al ver la *Figura 5.1*. El valor del MAPE va disminuyendo su valor progresivamente hasta alcanzar un mínimo al final del entrenamiento.

MODELOS MEDIOS

Con modelos medios nos referimos a modelos desde 3.000 hasta 20.000 datos de entrenamiento:

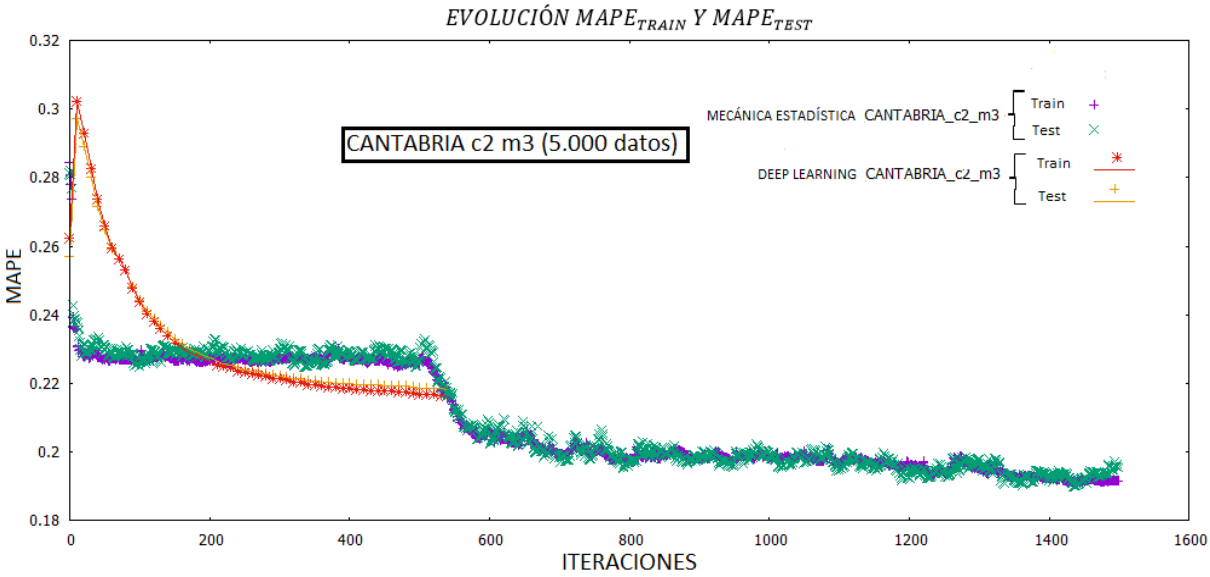

Figura 5.1.2: Representación de la evolución del modelo Cantabria c2 m3 con 5.000 datos de entrenamiento, en el que vemos una tendencia decreciente, por lo que un aumento de pasos de entrenamiento podría haber disminuido el error.

En esta figura apreciamos que estos modelos entrenados con el método de Deep Learning tienen muchas menos iteraciones de entrenamiento que los entrenados con el otro método.
Observamos que, aunque hay modelos con los que parece que ese número de iteraciones es suficiente, ya que aparentemente no disminuye mucho el valor del MAPE, hay otros en los que van disminuyendo progresivamente y no llegan a un estado estable en las iteraciones finales, dando la sensación de que si hubieran sido entrenados con un número de iteraciones mayor, hubiéramos obtenido mejores resultados.

MODELOS PEQUEÑOS

Con modelos medios nos referimos a modelos con 2.000 o menos datos de entrenamiento:

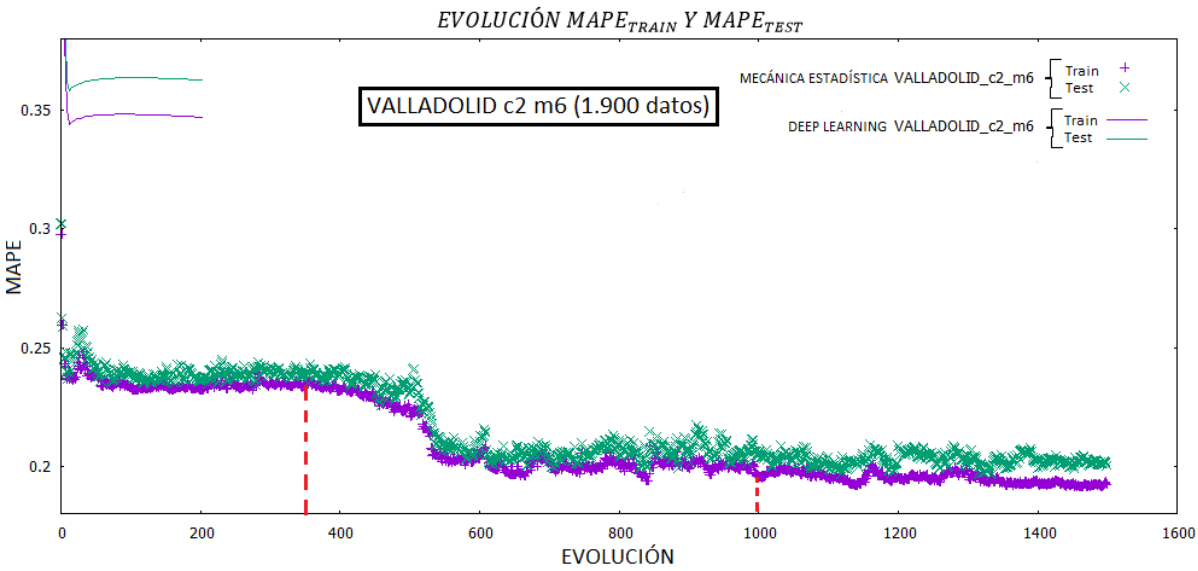

Figura 5.1.3 : Representación de la evolución del modelo Valladolid c2 m6 con 1.900 datos de entrenamiento.

Al igual que en el caso anterior, los modelos entrenados con el método de Deep Learning tienen muchas menos iteraciones de entrenamiento y volvemos a tener casos en los que el valor mínimo del MAPE obtenido podría haber disminuido si hubiéramos utilizado un número mayor de iteraciones, ya que su evolución presenta una tendencia descendente en las últimas.

Así, nos damos cuenta de que un aumento en el número de iteraciones de entrenamiento, para los archivos con menos de 20.000 datos, puede ser relevante en la mejora de nuestras predicciones obtenidas con el método de Deep Learning, y que este número puede ser un factor un condicionante para que los resultados de Deep Learning sean peores cuanto menor es el archivo.

Esto nos indica que nuestra suposición de que el entrenamiento para los archivos pequeños necesite menos iteraciones no sea del todo cierta y que la asignación proporcional de los pasos de entrenamiento al número de datos de cada modelo no sea muy buena.

Por otro lado, la suposición de que el error desciende conforme al entrenamiento también es falsa, ya que, como podemos comprobar, vemos que hay modelos en los que el valor del MAPE no es continuamente descendente, y en estos casos el valor final obtenido no será el mínimo. Por ello, otro factor que nos ayudaría a obtener mejores resultados con el método de Deep Learning sería escoger el valor mínimo del MAPE obtenido en la evolución, en vez del último obtenido.

Un aspecto característico de las figuras, es que en algunas obtenemos un valor del MAPE inicial mayor en los archivos Test que en los Train. Esto puede suceder si la dispersión de los precios del archivo de control es menor que la del archivo de entrenamiento y si el precio medio de los datos de entrenamiento queda entre el rango de los precios de control, echo bastante probable, ya que tenemos menos datos en los archivos de control que en los de entrenamiento, y estos son muy parecidos unos de los otros, haciendo posible las dos condiciones dichas anteriormente.

Otro aspecto que presentan es que los errores iniciales con Deep Learning son mucho mayores que los del método estadíctico. Esto es debido a que con este segundo método hacemos una aproximación

inicial, en la que estimamos que el valor del precio de los datos de control va a ser la media obtenida con los datos de entrenamiento, y con el primer método no hacemos ninguna, inicializando todos los parámetros de la red a cero.

Por último, analizando las gráficas, vemos que la evolución del método estadístico concuerda con el algoritmo establecido, ya que generalmente, el valor da una disminución brusca al añadir el segundo término (término de activación) en torno a las 350 iteraciones, indicando que este es un factor clave para la estimación final, y otra más suave al añadir el de Fourier, en torno a las 1000 iteraciones.

5.1.1 COMPARACIÓN DE LA EVOLUCIÓN CON DEEP LEARNING DE DISTINTOS MODELOS

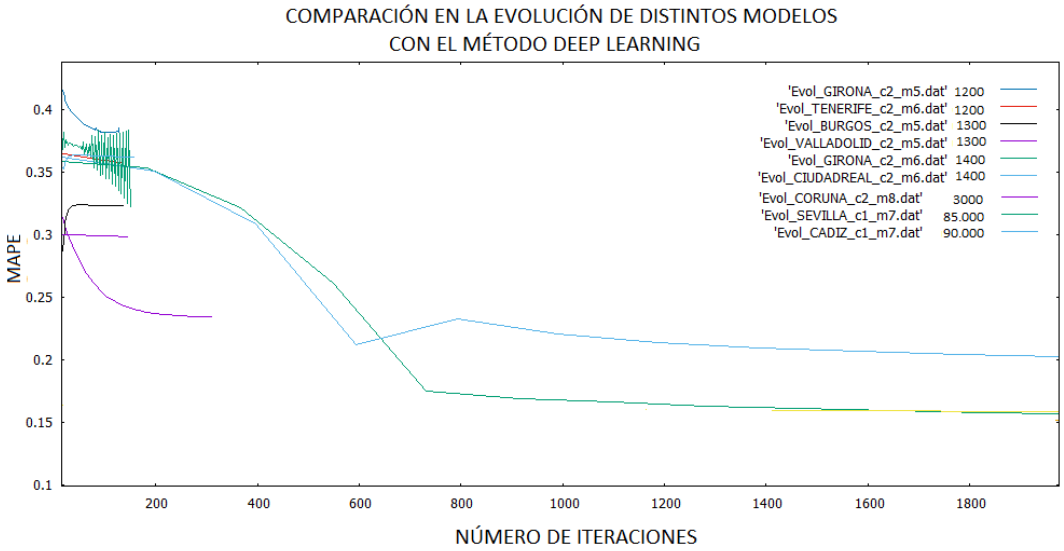

Figura 5.1.4 : Evolución con el método Deep Learning de distintos modelos con tamaños diferentes.

En esta gráfica hemos dibujado distintos modelos entrenados con Deep Learning y vemos que probablemente la asignación del número de iteraciones sea el factor más determinante en el hecho de que los errores obtenidos de modelos pequeños con este método sea peor que los obtenidos con el método estadístico.

5.2 EVOLUCIÓN DEL MAPE AUMENTANDO EL NÚMERO DE PASOS DE ENTRANEMIENTO

Debido a lo dicho en los dos apartados anteriores vamos a cambiar el algoritmo de nuestro código de Deep Learning, asignando un número de 1500 iteraciones (de igual manera que en el método estadístico):

$$train\_steps = 1500$$

Vamos a comparar estas evoluciones con las obtenidas en el apartado anterior y vamos a ver si este es un factor relevante en la determinación del error. Este proceso lo vamos a hacer solo con los archivos medios y pequeños, ya que hemos visto que con los grandes hemos obtenido resultados satisfactorios. Los resultados han sido los siguientes:

MODELOS MEDIOS

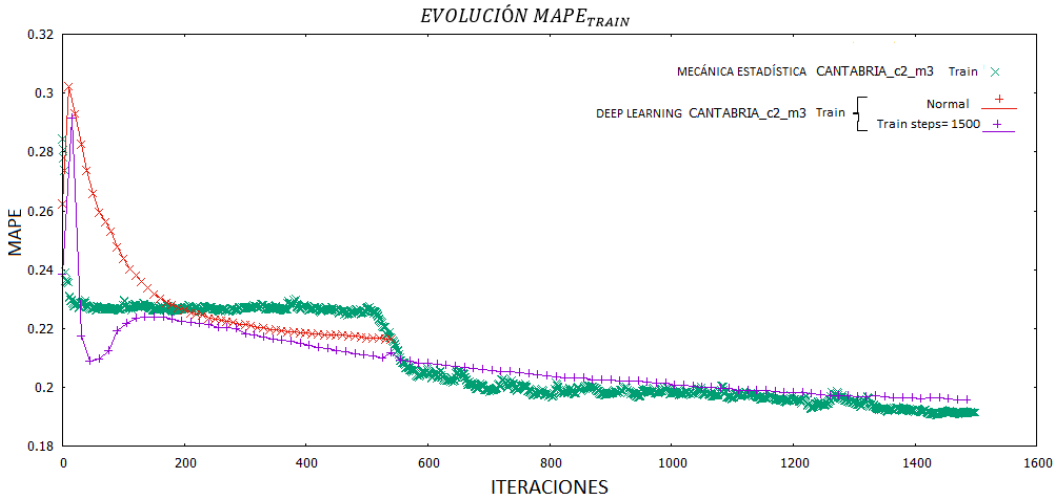

Figura 5.2.1 : Representación de la evolución del MAPE Train del modelo Cantabria c2 m3 con el método D.L. tomando 1500 iterciones de entrenamiento (dibujado en morado) y de las evoluciones representadas en el apartado anterior (dibujando M.E. en verde azulado y D.L. anterior en rojo).

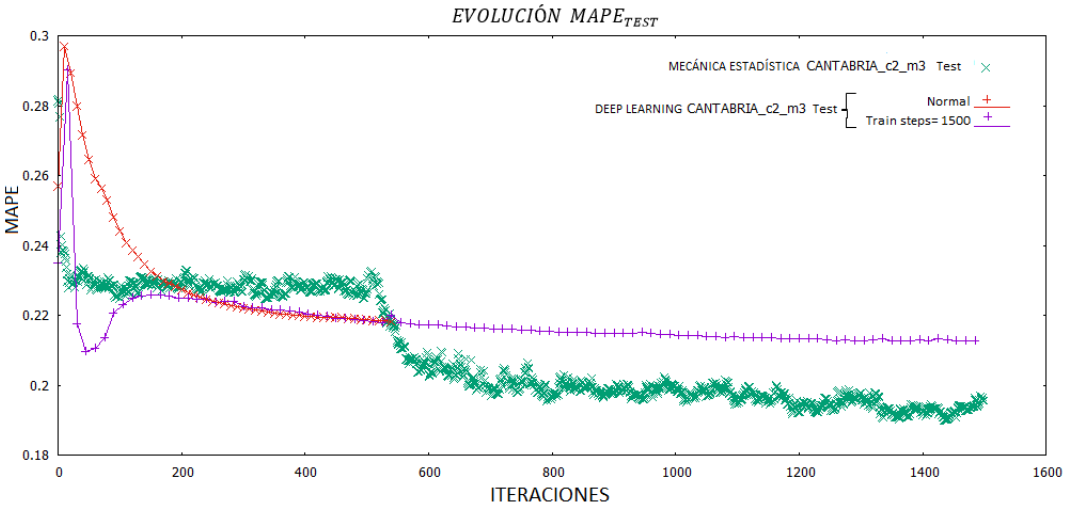

Figura 5.2.2: Representación de la evolución del MAPE Test del modelo Cantabria c2 m3 con el método D.L. tomando 1500 iterciones de entrenamiento (dibujado en morado) y de las evoluciones representadas en el apartado anterior (dibujando M.E. en verde azulado y D.L. anterior en rojo).

MODELOS PEQUEÑOS:

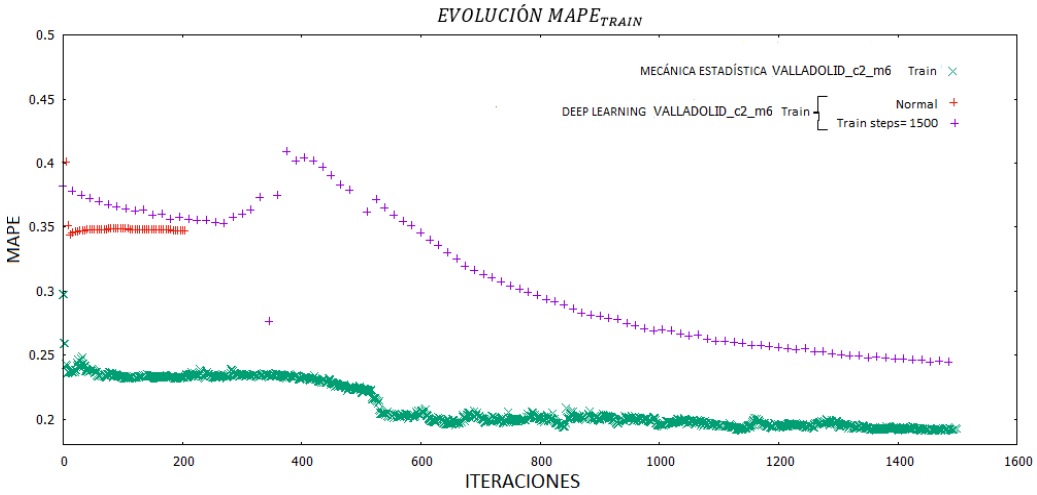

Figura 5.2.3 : Representación de la evolución del MAPE Train del modelo Valladolid c2 m6 con el método D.L. tomando 1500 iterciones de entrenamiento (dibujado en morado) y de las evoluciones representadas en el apartado anterior (dibujando M.E. en verde azulado y D.L. anterior en rojo).

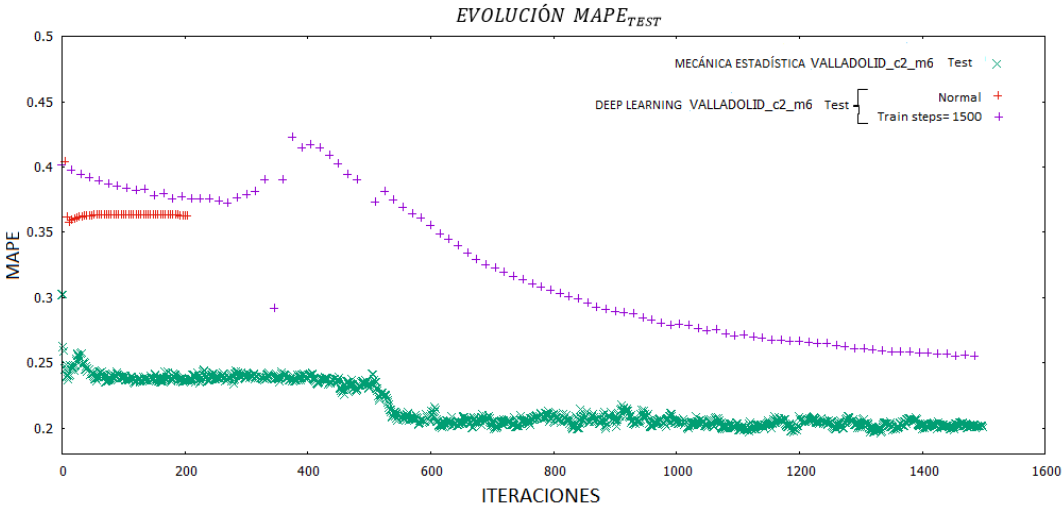

Figura 5.2.4 : Representación de la evolución del MAPE Test del modelo Valladolid c2 m6 con el método D.L. tomando 1500 iterciones de entrenamiento (dibujado en morado) y de las evoluciones representadas en el apartado anterior (dibujando M.E. en verde azulado y D.L. anterior en rojo).

Con estas gráficas comprobamos que, efectivamente, el aumento del número de iteraciones de entrenamiento es un factor clave para la mejor predicción del precio y que ambas suposiciones tomadas de las redes neuronales han sido falsas, obteniendo una mejora con el seguimiento de la evolución del MAPE, ya que para algunos modelos obtenemos valores mínimos en pasos intermedios del entrenamiento.

Estas las hemos hecho ya que, por la propia definición de la red neuronal, esta obtiene mejores resultados cuantos más datos de entrenamiento tenga, cuanto más homogéneos sean estos comparados con los de test y cuantas más iteraciones.

También las hemos hecho porque, en una primera prueba vimos que los valores obtenidos mediante este método no variaban mucho al multiplicar el número de iteraciones por cinco, como se puede ver en el ANEXO 6.

5.3 EVOLUCIÓN DEL MAPE PARA ARCHIVOS QUE HAN DADO UN VALOR GRANDE

Aunque en las gráficas del apartado anterior hemos visto que hay casos en los que el valor del MAPE toma valores menores que el valor final, su valor final ha mejorado considerablemente con el aumento de iteraciones, por ello, en nuestro caso vamos a comparar solamente los resultados finales obtenidos cambiando el número de iteraciones de entrenamiento para los modelos con los que hemos obtenido un valor del MAPE con el método de Deep Learning mayor que el 25%, ya que este ha sido aproximadamente el valor máximo obtenido con el método de Mecánica Estadística.

Esta decisión también la tomamos debido a que el seguimiento del entrenamiento cuesta ≅ 50 veces más de computar, debido a que el proceso más costoso con los archivos de Python es cargar las librerías y carpetas en las que almacenamos los valores de los distintos parámetros, y esto lo realizamos cada vez que dibujamos el valor del MAPE. Al dibujar cincuenta valores distintos del MAPE en cada proceso de entrenamiento, el tiempo en realizar el entrenamiento siguiendo la evolución es ≅ 50 veces más largo.

Los resultados los vamos a comparar tanto con los propios del método de Deep Learning, obtenidos anteriormente, como con los obtenidos con el método de Mecánica Estadística. Estos son los siguientes:

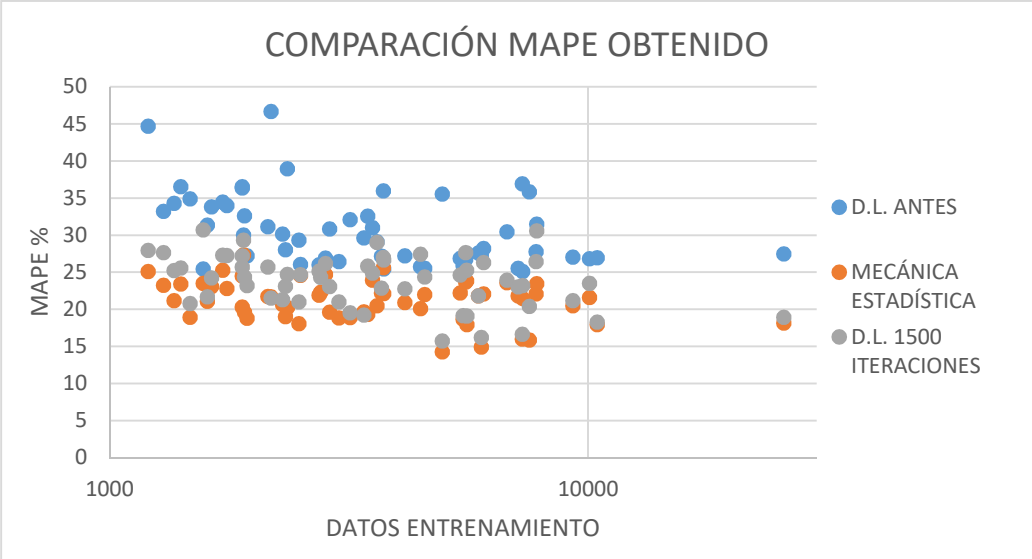

Figura 5.3.1: Comparación de los datos obtenidos del MAPE fijando el número de pasos de entrenamiento a 1500, con los resultados anteriores.

Vemos que los valores obtenidos han mejorado considerablemente obteniendo unos valores de medias ponderadas de:

$$\langle MAPE \rangle_{D.L.\ Antes} = 29.27\ \pm 4.88\ \%$$
$$\langle MAPE \rangle_{D.L.\ 1500\ Iter} = 22.83 \pm 2.76\ \%$$
$$\langle MAPE \rangle_{Mecánica\ Estadística} = 20.54 \pm 3.58\ \%$$

Obteniendo así una diferencia entre errores de:

$$ERROR_{D.L.'S} = \frac{\langle MAPE \rangle_{D.L.\ Antes} - \langle MAPE \rangle_{D.L.\ 1500\ Iter}}{\langle MAPE \rangle_{D.L.\ 1500\ Iter}} \times 100 = 28.21\%$$

$$ERROR_{D.L.\ ANTES} = \frac{\langle MAPE \rangle_{D.L.\ Antes} - \langle MAPE \rangle_{Mecánica\ Estadística}}{\langle MAPE \rangle_{Mecánica\ Estadística}} \times 100 = 42.51\%$$

$$ERROR_{D.L.\ 1500\ ITER} = \frac{\langle MAPE \rangle_{D.L.\ 1500\ Iter} - \langle MAPE \rangle_{Mecánica\ Estadística}}{\langle MAPE \rangle_{Mecánica\ Estadística}} \times 100 = 11.15\%$$

De este modo, hemos disminuido el error relativo de un 40% a un 10% aproximadamente comparando los datos con los obtenidos con el método estadístico, lo que significa que, aunque siguen siendo algo peores que los del otro método, incrementando el número de iteraciones hemos disminuido cuatro veces la diferencia entre estos.

6 CONCLUSIONES Y LÍNEAS FUTURAS:

Después del estudio y comparación de los datos obtenidos por ambos métodos, llegamos a la conclusión de que estos son muy parecidos, obteniendo medias ponderadas similares, con un error relativo menor que el 10%. Siendo algo peores los del método de Deep Learning cuando tenemos pocos datos de entrenamiento y mejorando conforme aumenta el número de estos datos hasta obtener errores más bajos que el otro método.

También se han visto distintos factores que influían en los malos resultados obtenidos con el método D.L. siendo el más relevante el número de iteraciones de entrenamiento.

Para la mejora de estos resultados se podría hacer una mezcla homogénea de datos de entrenamiento y control sin estar separados ordenadamente ya que puede haber un aumento o disminución en el valor del seguro de los coches dependiendo del mes o año en que se hayan realizado.

Además, se podría buscar un número óptimo de iteraciones de entrenamiento de la red, un seguimiento de la evolución de los errores obtenidos de control quedándonos con el mínimo, en lugar de escoger el final y distintas pruebas con otros valores de neuronas en cada capa y otro número de capas.

Del mismo modo, podríamos aumentar el número de iteraciones del método estadístico e incluir los distintos términos de la función de aproximación. Todo esto teniendo en cuenta el tiempo de CPU, dado que el tiempo de procesamiento debe ser similar.

Por último, podríamos estudiar los datos teniendo en cuenta en qué época del més o año se han vendido, teniendo en cuenta las distintas ofertas que realiza la compañía al final de més o en periodos de vacaciones y rebajas.

BIBLIOGRAFÍA:

[1] Grapsas, Tatiana. (2019, 26 de Febrero). Deep Learning: Entiende sobre la tendencia de la Inteligencia Artificial que copia al cerebro humano. Extraido el 15 de Junio de 2019 desde https://rockcontent.com/es/blog/deep-learning/.

[2] Grapsas, Tatiana. (2019, 26 de Febrero). Deep Learning: Entiende sobre la tendencia de la Inteligencia Artificial que copia al cerebro humano. Extraido el 15 de Junio de 2019 desde https://rockcontent.com/es/blog/deep-learning/.

[3] *Dot CSV.* (2017, 1 de Noviembre). ¿Qué es el Machine Learning?¿Y Deep Learning? Un mapa conceptual. Extraido el 3 de Marzo de 2019 desde https://www.youtube.com/watch?v=MRIv2IwFTPg&t=2s.

[4] *AMP Tech.* (2018, 20 de Mayo). Tipos de redes neuronales. Extraído el 3 de Marzo de 2019 desde https://www.youtube.com/watch?v=V5BYRPJThjE

[5] López Briega R. (2016, 2 de Agosto) Redes neuronales convolucionales con Tensorflow. Extraído el 5 de Marzo de 2019 desde https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/.

[6] J.C.Gonzalez (2017, 20 de Noviembre). USE OF CONVOLUTIONAL NEURAL NETWORK FOR IMAGE CLASIFICATION. Extraído el 5 de Marzo de 2019 desde https://www.apsl.net/blog/2017/11/20/use-convolutional-neural-network-image-classification/

[7] Kaz Sato (2017, 29 de Marzo). *Using machine learning for insurance pricing optimization.* Extraído el 20 de Marzo de 2019 desde https://cloud.google.com/blog/products/gcp/using-machine-learning-for-insurance-pricing-optimization.

[8] Tarancón Lafita, A. (2016, 4 de Febrero). *Fisica Computacional, Tomo 1, Teoría, Problemas y Prácticas.(pp.193-198,).*

[9] Tarancón Lafita, A. (2016, 4 de Febrero). *Fisica Computacional, Tomo 1, Teoría, Problemas y Prácticas.(pp.73-74,).*

[10] Tarancón Lafita, A. (2016, 4 de Febrero). *Fisica Computacional, Tomo 1, Teoría, Problemas y Prácticas.(pp.167-174).*

[11] Guido van Rossum. (2017, Octubre). *El tutorial de Python* .Recuperado de https://www.python.org.arg/

[12] Google Brain. (2015, 9 de Noviembre). Recuperado de https://www.tensorflow.org/

[13] González J. C. (2017-2018).*TENSORFLOW PARA PRINCIPIANTES*. Recuperado de https://www.apsl.net/blog/tag/redes-neuronales/

[14] Moya R. (2015, 30 de Octubre).Pandas en Python, con ejemplos, Parte 1 y Parte 2. Extraido el 25 de Marzo de 2019 de https://jarroba.com/pandas-python-ejemplos-parte-i-introduccion/

[15]Oliphant T.(2005). NumPy. Recuperado de https://www.numpy.org/

[16] NumPy.(25 de Junio de 2019). En Wikipedia, *la enciclopedia libre*. Recuperado el 25 de Marzo de 2019 de https://docs.python.org/3/library/os.html

[17]Sharma S. (2017, 6 de Septiembre). Activation Functions in Neural Networks. Extraido el 6 de Marzo de 2019 desde https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[18] Dot CSV (2018, 19 de Marzo). Parte 1: La Neurona. Extraido el 3 de Marzo de 2019 desde https://www.youtube.com/watch?v=MRIv2IwFTPg&t=2s

[19] Redes Neuronales. Extraido el 15 de Junio de 2019 de: https://ml4a.github.io/ml4a/es/neural_networks/

[20] Dot CSV (2017, 16 de Diciembre).Regresión Lineal y Mínimos Cuadrados Ordinarios. Extraido el 3 de Marzo de 2019 desde https://www.youtube.com/watch?v=k964_uNn3l0&t=20s

[21] Dot CSV (2018, 4 de Febrero).¿Qué es el Descenso del Gradiente? Algoritmo de Inteligencia Artificial. Extraido el 3 de Marzo de 2019 desde https://www.youtube.com/watch?v=A6FiCDoz8_4

[22] Dot CSV (2018, 14 de Octubre).¿Qué es una Red Neuronal? Parte3.5: Las matemáticas de Backpropagation. Extraido el 3 de Marzo de 2019 desde https://www.youtube.com/watch?v=A6FiCDoz8_4

[23] Weinberger K. (2018, 11 de Julio). Machine Learning Lecture 12 "Gradient Descent/ Newthon's Method". Extraido el 14 de Abril de 2019 desde https://www.youtube.com/watch?v=o6FfdP2uYh4

[24] Gylberth R. (2018, 3 de Mayo). An Introduction to AdaGrad. Extraido el 14 de Abril de 2019 desde https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827