Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Modelos de predicción de la vida útil de baterías de iones de Litio

Autor: Guillermo Flores Muñoz
Tutor: Juan Manuel Escaño González

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Modelos de predicción de la vida útil de baterías de iones de Litio

Autor:
Guillermo Flores Muñoz

Tutor:
Juan Manuel Escaño González

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo de Fin de Grado: Modelos de predicción de la vida útil de baterías de iones de Litio

Autor: Guillermo Flores Muñoz

Tutor: Juan Manuel Escaño González

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

Agradecimientos

Me gustaría dedicar dedicar unas palabras de agradecimiento a todas las personas que me han acompañado durante mi carrera universitaria.

Primero, a mis padres, sin los cuales jamás habría llegado hasta donde estoy. Siempre con su apoyo y esfuerzo para que siguiera adelante. También a mi hermano, que allanó el camino que seguí.

A mis compañeros y amigos, sobre todo a los que sufrieron conmigo hasta el final.

A mi tutor, Juan Manuel, por darme la oportunidad de realizar este trabajo y ayudarme a conseguirlo pese a las complicaciones del mundo este año.

Al resto de mi familia, particularmente mi tía, que se preocupó, como siempre, en ayudarme a encontrar lo que buscaba.

Y en especial a mi abuelo, que sé que está orgulloso allá donde esté.

Guillermo Flores Muñoz

Sevilla, 2020

Resumen

En este trabajo se pretende obtener un modelo capaz de predecir la vida útil de una batería de ion de litio a partir de datos de los primeros ciclos sin tener en cosideración las propiedades mecánicas o químicas de las baterías. Esto es, será una predicción exclusivamente estadística basada en datos medidos en los primeros ciclos de carga y descarga de un conjunto de baterías comerciales cicladas hasta el fallo. Con la penetración de las baterías en el mercado de consumo duradero, como los coches eléctricos, la duración de la vida útil de una batería toma una importancia que nunca había tenido debido a la frecuente renovación de los dispositivos en los que se usaban baterías, como los teléfonos móviles. Además, debido a la relativamente larga vida útil de las baterías, el análisis de rendimiento de esta se demora meses o años. Por ello, una predicción temprana abriría nuevas oportunidades de desarrollo, fabricación y optimización inalcanzables hasta ahora.

Abstract

In this work, the aim is to obtain a model capable of predicting the life of a lithium-ion battery from data from the first cycles without considering the mechanical or chemical properties of the batteries. That is, it will be a purely statistical prediction based on data measured in the first charge and discharge cycles of a set of commercial batteries cycled to failure. With the penetration of batteries in the consumer durables market, such as electric cars, the cycle life of a battery takes on an importance that it never had before due to the frequent renewal of devices where batteries were used, such as mobile phones. In addition, due to the relatively long life of batteries, the analysis of battery performance takes months or years. Therefore, an early prediction would open up new opportunities for development, manufacturing and optimization unattainable until now.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Motivación y objetivos

Las baterías de iones de litio se han utilizado comercialmente durante años en pequeños dispositivos como los teléfonos móviles, ordenadores portátiles, cámaras de video y dispositivos electrónicos similares. Debido a la rápida innovación en este campo y la frecuencia de renovación de los dispositivos por parte de los consumidores, la vida útil de las baterías jugó un papel menor entre sus especificaciones. Pero el comienzo de la penetración de baterías de iones de litio en el mercado de consumo duradero y bienes de inversión, como los vehículos eléctricos o híbridos, sistemas de almacenamiento temporales para fuentes de energía renovables, y también en el mercado de las baterías para vehículos convencionales, hace que se requiera una evaluación más sofisticada de la duración de la vida de la batería.

Desafortunadamente, las baterías de iones de litio son sistemas complejos de entender, y los procesos de su envejecimiento son aún más complicados. La disminución de la capacidad y de la energía no se originan por una sola causa, sino por varios procesos diversos y sus interacciones. Además, la mayoría de esos procesos no pueden estudiarse de forma independiente y se producen prácticamente al mismo tiempo, lo que complica la investigación de los mecanismos de degradación.

Las características de los materiales, así como las condiciones de almacenamiento y de ciclo, influyen en la duración y el rendimiento de las baterías. Dependiendo de la química de la célula, tanto en el estado de carga alto como el bajo pueden deteriorar el rendimiento y acortar la vida de la batería. A altas temperaturas, el decaimiento se acelera, pero las bajas temperaturas, especialmente durante la carga, también pueden tener un impacto negativo. Entre los parámetros de los materiales, la química de la superficie juega un papel importante tanto para los materiales del ánodo como para los del cátodo. En el cátodo, las transiciones de fase y los cambios estructurales en el material influyen fuertemente en el envejecimiento, mientras que los cambios en el material anódico se consideran de menor importancia. [1]

Debido a esta complejidad y unida, como ocurre con muchos sistemas químicos, mecánicos y electrónicos, la relativamente larga vida útil de las baterías conlleva un retraso en el análisis del rendimiento, a menudo desde muchos meses a años.

La predicción precisa de la vida útil utilizando datos de los primeros ciclos abriría nuevas oportunidades en la producción, el uso y la optimización de las baterías. Por ejemplo, los fabricantes podrían acelerar el ciclo de desarrollo de las células, validar rápidamente los nuevos procesos de fabricación y clasificar las nuevas células por su vida útil esperada. Del mismo modo, los usuarios finales podrían estiman la esperanza de vida de sus baterías.

Este proyecto tiene como objetivo replicar y ampliar un artículo publicado en la revista Nature [2], que va precisamente en esa dirección, usando datos y mediciones de los primeros ciclos de carga para predecir empíricamente con modelos estadísticos la vida útil de baterías de iones de litio sin tener en consideración los procesos físicos y químicos que se dan en la degradación de dichas baterías. El lenguaje de programación que se usará será Python.

Concretamente definimos vida útil de una batería o ciclo de vida como el número de ciclos de carga y descarga que una batería puede completar antes de perder su rendimiento. Ésta pérdida de rendimiento la situamos en el 80% de su capacidad nominal.

1.2 Estado Del Arte

La tarea de predecir la vida útil de las baterías de iones de litio es de crítica importancia dada su amplia utilidad, pero difícil debido a la no linealidad de su degradación respecto a los ciclos de carga y a una amplia variabilidad, incluso cuando se controlan las condiciones de funcionamiento.

Muchos estudios anteriores han modelado la vida útil de las baterías de iones de litio. Muchos autores han propuesto modelos físicos y semi-empíricos que tienen en cuenta diversos mecanismos como el crecimiento de la interfase electrolítica sólida [3], el revestimiento de litio [4], la pérdida de material activo [5] y el aumento de la impedancia [6]. Las predicciones la vida útil restante en los sistemas de gestión de baterías, resumida en estas revisiones [7] [8], a menudo se basan en estos modelos mecanicistas y semi-empíricos para la estimación del estado. Las mediciones diagnósticas especializadas como la eficiencia Faraday [9] y la espectroscopia de impedancia [10] pueden ser también utilizadas para la estimación de la vida útil. Mientras que estos modelos químicos y/o mecánicos específicos han mostrado cierto éxito predictivo, desarrollar modelos que describen completamente las células que se ciclan bajo determinadas condiciones operativas como, por ejemplo, la carga rápida, sigue siendo un desafío, dados las numerosas formas de degradación y su dependencia a las heterogeneidades térmicas y mecánicas dentro de una célula.

Los enfoques que utilizan técnicas estadísticas y de aprendizaje automático para predecir el ciclo de vida son alternativas atractivas y agnósticas a los mecanismos. Recientemente, los avances en potencia computacional y generación de datos han permitido que estas técnicas aceleren el progreso de una variedad de tareas, incluyendo la predicción de las propiedades de los materiales [11], la identificación de las rutas de síntesis química [12] y el descubrimiento de materiales para energía almacenamiento [13] y catálisis [14]. Cada vez hay más artículos científicos donde se aplican técnicas de aprendizaje de máquina para predecir la vida útil restante de las baterías utilizando datos recogidos en el laboratorio. Generalmente, estos trabajos hacen predicciones después de acumular datos correspondientes a la degradación de al menos el 25% a lo largo de la trayectoria hasta el fallo [15] o utilizando mediciones especializadas en el comienzo de la vida [16]. La predicción temprana y precisa de la vida útil con una degradación significativamente menor es un desafío debido al proceso de degradación no lineal (con una degradación insignificante de la capacidad en los primeros ciclos), así como los relativamente pequeños conjuntos de datos utilizados para fecha que abarcan un rango limitado de vidas de baterías [17]. Por ejemplo, se encontraron una débil correlación (ρ=0.1) entre los valores de capacidad en el ciclo 80 y los valores de capacidad en el ciclo 500 para 24 células que exhiben perfiles de degradación no lineales [18], que ilustran la dificultad de esta tarea.

La modelización basada en datos es una vía prometedora para el diagnóstico y el pronóstico de las baterías de iones de litio y permite aplicaciones emergentes en su desarrollo, fabricación y optimización. Se han desarrollado modelos de predicción del ciclo de vida utilizando datos de descarga de ciclo temprano que aún no muestran degradación de la capacidad, generados a partir de baterías comerciales de LFP/grafito ciclados en condiciones de carga rápida. Consiguiendo un nivel alto de precisión extrayendo características de las curvas de la capacidad de descarga en función del voltaje en lugar de sólo de las curvas de desvanecimiento de la capacidad, y sin utilizar datos de ciclos de diagnóstico lentos ni asumir conocimientos previos de la química celular y los mecanismos de degradación [19].

En resumen, las oportunidades de mejorar los modelos de predicción del estado del arte incluyen una mayor precisión, una predicción más temprana, una mayor interpretabilidad y una aplicación a una gama más amplia de condiciones de ciclado.

2 DATOS EMPLEADOS EN EL PROYECTO

En este capítulo se explicará todo lo relacionado con los datos utilizados, es decir, cómo se han obtenido y preparado los datos para poder trabajar con ellos de forma correcta.

Este proyecto no se ha centrado en el proceso de recolección de datos, puesto que el artículo del que se parte tiene la base de datos sobre la que se trabaja puesta a disposición pública. Sin embargo, sí que se han tenido que preparar para su uso en los modelos, principalmente detección y tratamiento de valores extraños encontrados.

2.1 Descripción de los datos

El conjunto de datos con el que trabajaremos [20], consta de 124 baterías comerciales de iones de litio que se someten a sucesivos ciclos hasta el fallo en condiciones de carga rápida. Estas células, fabricadas por A123 Systems (APR18650M1A), se han sometido a estos ciclos en un potenciostato LBT Arbin de 48 canales en una cámara de temperatura de convección forzada ajustada a 30°C. Las celdas tienen una capacidad nominal de 1,1 amperios hora (Ah) y un voltaje nominal de 3,3 voltios (V).

Todas las células de este conjunto de datos se cargan con una política de carga rápida de uno o dos pasos. Esta política tiene el formato "C1(Q1)-C2", en el que C1 y C2 son el primer y segundo paso a corriente constante, respectivamente, y Q1 es el estado de carga (State of charge o SOC, %) al que cambian las corrientes. El segundo paso de corriente termina en el 80% SOC, después de lo cual las células se cargan en 1C CC-CV. Los potenciales de corte superior e inferior son de 3,6 V y 2,0 V, respectivamente, que se ajustan a las especificaciones del fabricante. Estos potenciales de corte son fijos para todos los pasos de corriente, incluyendo la carga rápida; después de algunos ciclos, las células pueden alcanzar el potencial de corte superior durante la carga rápida, lo que lleva a una carga significativa de voltaje constante. Todas las células se descargan a 4C.

El conjunto de datos se divide en tres "lotes", que representan aproximadamente 48 células cada uno. Cada lote se define por una "fecha de lote", o la fecha en que se iniciaron las pruebas

Las mediciones de temperatura se realizan adjuntando un termopar tipo T con epoxi térmico (OMEGATHERM 201) y cinta de Kapton a la célula expuesta, después de haber quitado una pequeña sección del aislamiento de plástico. Las mediciones de temperatura no son perfectamente fiables; el contacto térmico entre el termopar y la célula puede variar sustancialmente, y el termopar a veces pierde el contacto durante el ciclo.

Las mediciones de resistencia interna se obtuvieron durante la carga al 80% de SOC promediando 10 pulsos de ±3,6C con un ancho de pulso de 30 ms o 33.

Los datos están proporcionados en formato .mat de MATLAB aunque pueden abrirse en Python, lenguaje en el que se desarrollará el modelo. También se proporciona un código para procesar y cargar estos datos, aunque como veremos, requieren de más procesado para poder trabajar con ellos y construir el modelo predictivo.

Este código carga los datos en una estructura diccionario o dict de Python. Los diccionarios se encuentran a veces en otros lenguajes como "memorias asociativas" o "vectores asociativos". A diferencia de las secuencias, que están indexadas por una serie de números, los diccionarios están indexados por claves [21]. En este caso cada clave lleva a un conjunto de datos que a su vez es también un diccionario. Esto se llama diccionario anidado y, aunque cumple su función, no es lo más ligero en

consumo de recursos ni lo más rápido para ejecutar cálculos sobre los datos.

El primer nivel de estos diccionarios anidados son cada una de las baterías (celdas). En total hay 124 claves, una por batería. Su nomenclatura es "b" (batch) seguida del número de lote más "c" (cell) seguida por un número en orden. Por ejemplo, "b1c0".

Los datos asociados a cada batería pueden agruparse en una de estas cuatro categorías, esquematizadas en la Figura 2-1, que serán el siguiente nivel del diccionario:

- Ciclos de vida ("cycle_life"): Número entero que informa del número de ciclos en los que la batería mantiene su capacidad de descarga por encima del 80%.
- Política de carga ("charge_policy"): Descripción de cómo ha sido cargada la batería tal y como se ha explicado anteriormente. Un ejemplo sería "3.6C(80%)-3.6C"
- Datos resumidos ("summary"): incluyen información única por ciclo incluido la resistencia interna, la capacidad de carga, la capacidad de descarga, la temperatura media, la temperatura mínima, la temperatura máxima, el tiempo de carga y el número de ciclo. Es otro nivel en el diccionario, las claves serán: 'IR', 'QC', 'QD', 'Tavg', 'Tmin', 'Tmax', 'chargetime', 'cycle'.
  Cada uno es un vector de tipo float cuya longitud será el número de ciclos de vida, con mediciones en cada uno de los ciclos.
- Ciclos ("cycles"): Es otro nivel anidado compuesto de claves numéricas correspondientes a cada ciclo de la batería en cuestión. Cada ciclo, a su vez, lo componen las siguientes claves que dan información por ciclo, incluido la corriente, la capacidad de carga, la capacidad de descarga, la capacidad de descarga linealmente interpolada, la temperatura, la temperatura linealmente interpolada, el voltaje, los vectores derivados de la descarga dQ/dV y el tiempo.
  Las claves son: 'I', 'Qc', 'Qd', 'Qdlin', 'T', 'Tdlin', 'V', 'dQdV', 't'

  Cada uno es un vector tipo float y la longitud varía dependiendo de las mediciones que se hayan hecho en el tiempo de descarga. Excepto para las dos mediciones linealmente interpoladas que miden mil.

Reordenaremos este diccionario y lo convertiremos en un DataFrame de *pandas*. También eliminaremos datos redundantes y los que no utilizaremos en el modelo.

En esta transformación haremos la primera limpieza de los datos ya que no son consistentes en cada uno de los tres lotes. Por ejemplo, en el lote 1 (b1) todas las mediciones resumen ("summary"') por ciclo empiezan con un valor 0 y tienen un ciclo más del que les corresponde. Tenemos que quitar la primera medida y desplazar el vector de ciclos para dejarlo como en los otros lotes. En cuando a las mediciones por ciclo ("cycles") empiezan en el ciclo 0 y todos los vectores son vectores sin mediciones. También hay que eliminarlo y desplazar el resto.

También dividiremos los datos en tres DataFrames que serán más ligeros para cargarlos en la RAM del limitado sistema con el que trabajamos. Uno de ellos será summary, compuesto por todos los datos del diccionario con el mismo nombre y haciendo de la columna cycle su índice.

Los otros serán Qdlin y Tdlin con los datos correspondientes a cada uno de los vectores del diccionario cycle. Las columnas serán los ciclos y el índice cada uno de los 1000 puntos en los que se interpolan estos datos.

Con los datos ya ordenados podemos hacer un estudio de cada una de las mediciones para ver cuales pueden funcionar mejor como predictores. En este estudio también iremos limpiando todos los posibles valores extraños conocidos por su palabra en inglés como 'outliers'.

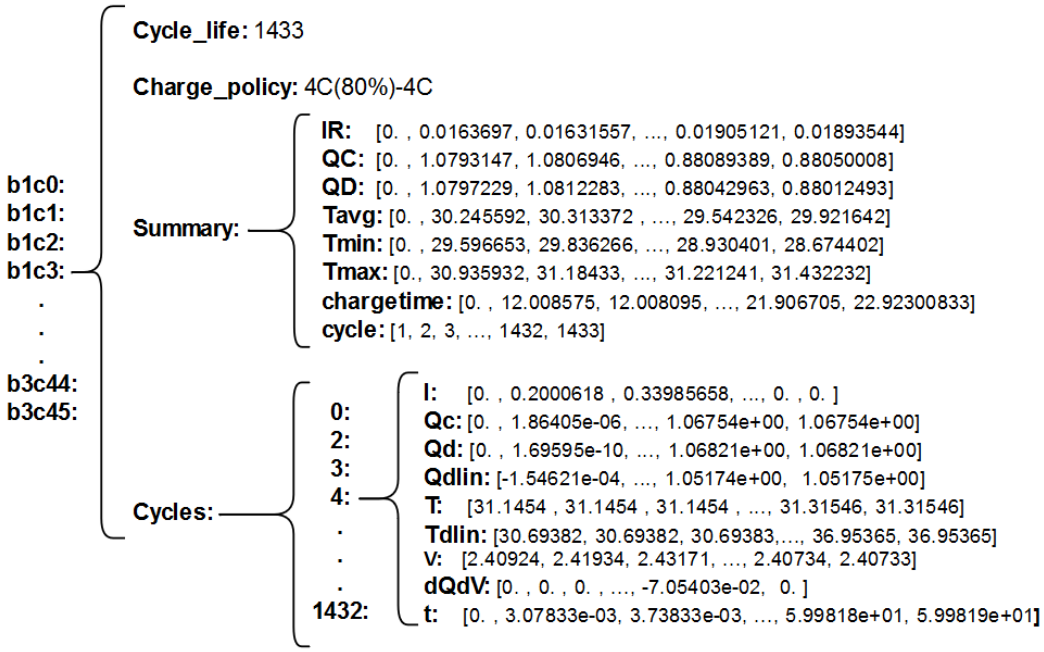

Figura 2-1. Esquema de la ordenación del dict de los datos proporcionados.

2.2 Análisis Exploratorio de Datos

El análisis exploratorio de datos es una forma de analizar datos definido por John W. Tukey [22]. Es el tratamiento estadístico al que se someten las muestras recogidas durante un proceso de investigación en cualquier campo científico.

Dividiremos esta sección en dos. En la primera observaremos detenidamente de manera gráfica todos los datos que nos son proporcionados. Veremos qué características puedan ser las más relevantes en términos de predicción de la vida útil. Será en ésta donde se realice también el tratamiento de los 'outliers'. En la segunda prepararemos los datos para introducirlos en el modelo, haciendo un estudio de correlaciones con el objetivo y

2.2.1 Visualización y tratamiento de los datos

Usamos la misma sección para la visualización y para la detección de 'outliers' por el simple hecho que es más fácil ver el antes y el después del tratamiento conforme se va haciendo y graficando.

Como hemos visto en la sección 2.1, los principales datos están recogidos en dos conjuntos, según si las mediciones están hechas una para cada ciclo o a lo largo del él.

Primero miraremos a las medidas tomadas por cada ciclo. Habrá sólo una por cada uno de los ciclos de cada batería. El eje x representa cada uno de esos ciclos mientras que el y representa la medida en cuestión. Cada línea corresponde a una batería y empezaremos dividiendo las gráficas con una leyenda de tres colores, uno por cada lote de baterías (b1, b2, b3). Estas medidas serán la capacidad de descarga, la resistencia interna y el tiempo de carga.

También analizaremos las medidas tomadas en cada ciclo. Habrá tantas como dure el ciclo en cuestión por cada uno de los ciclos de cada batería. Estás gráficas no pueden mostrar todas las baterías sino todos los ciclos de una sola batería.

2.2.1.1 Capacidad de descarga en función de los ciclos

Primero observamos la medida más importante del experimento. La capacidad de descarga, QD,

medida en amperios hora. El experimento termina cuando dicha medida llega al 80% de la capacidad nominal.

La capacidad de descarga QD de una batería o cualquier otro dispositivo de almacenamiento de energía es la cantidad de carga que puede ser retirada descargando desde el voltaje máximo de funcionamiento al mínimo [23].

La Figura 2-2 muestra la curva que hacen las mediciones al final de cada ciclo de la capacidad de descarga de cada batería.

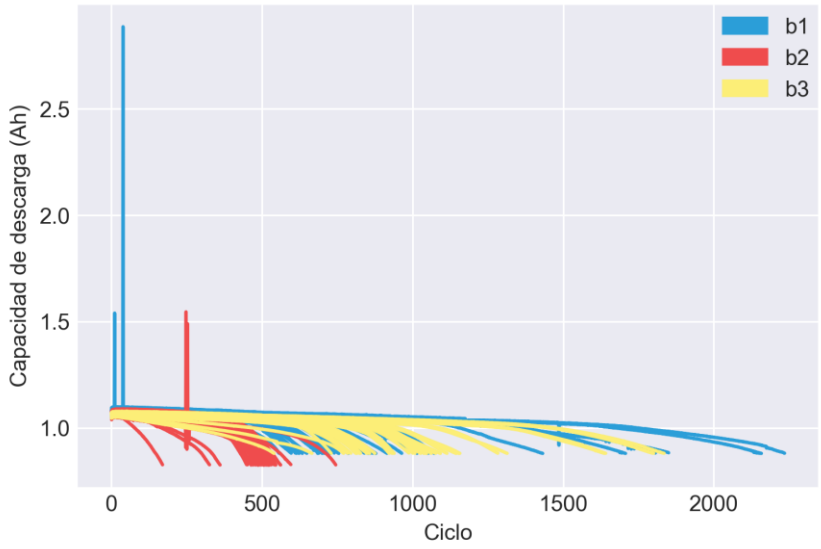

Figura 2-2. Capacidad de descarga de cada batería en cada ciclo de su vida

Como se puede observar, la gráfica -ni los datos- sin ningún procesamiento previo sirven para entender el proceso ni para construir un modelo que haga predicciones. Empezaremos a practicar el tratamiento de datos que veníamos hablando. Hay valores que multiplican por casi 3 el máximo que debería poder llegar esa medición (en este caso, 1.1 Ah). Las distorsiones en la gráfica la producen sólo unos pocos valores extraños debidos a un problema en las mediciones de ese punto en concreto o de una mala transcripción de los datos a la tabla. Por ello, al no ser datos reales, no aportan ninguna información relevante sino todo lo contrario. No son más que ruido que evita la correcta visualización de estos datos y lo que es peor, la correcta predicción del modelo.

La forma que usaremos para identificarlos será calcular si están dentro de un rango de valores esperables. Estos valores se ilustran en la Figura 2-3 con un sombreado del color de las marcas de las mediciones. Se calculan multiplicando el ajuste a los datos por un porcentaje. En este caso es un 5%.

La regresión usada es la local ponderada en la que el valor ajustado en x_k es el valor de un ajuste polinómico a los datos utilizando los mínimos cuadrados ponderados, donde el peso para (x_i, y_i) es grande si x_i está cerca de x_k y pequeño si no lo está [24].

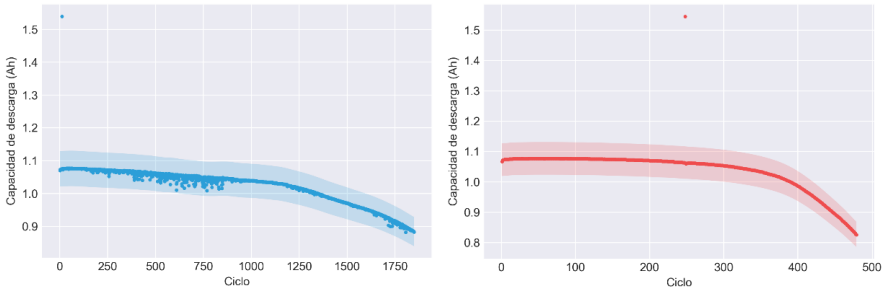

Figura 2-3. Dos curvas QD-Ciclos aleatorias (b1 y b2) con un rango de valores aceptables sombreados.

Aplicando este método en todas las baterías obtenemos la gráfica de la Figura 2-4a. Con los nuevos datos ya procesados encontramos un segundo problema. Las mediciones del segundo lote terminan más tarde que las de los otros dos lotes. El experimento debería terminar al 80% de la capacidad nominal de la batería, esto es, con una capacidad de descarga de 0.88 Ah. Sin embargo, las baterías del segundo lote terminan con una capacidad de descarga de 0.82 Ah. Eliminando las mediciones sobrantes nos quedaría la Figura 2-4b.

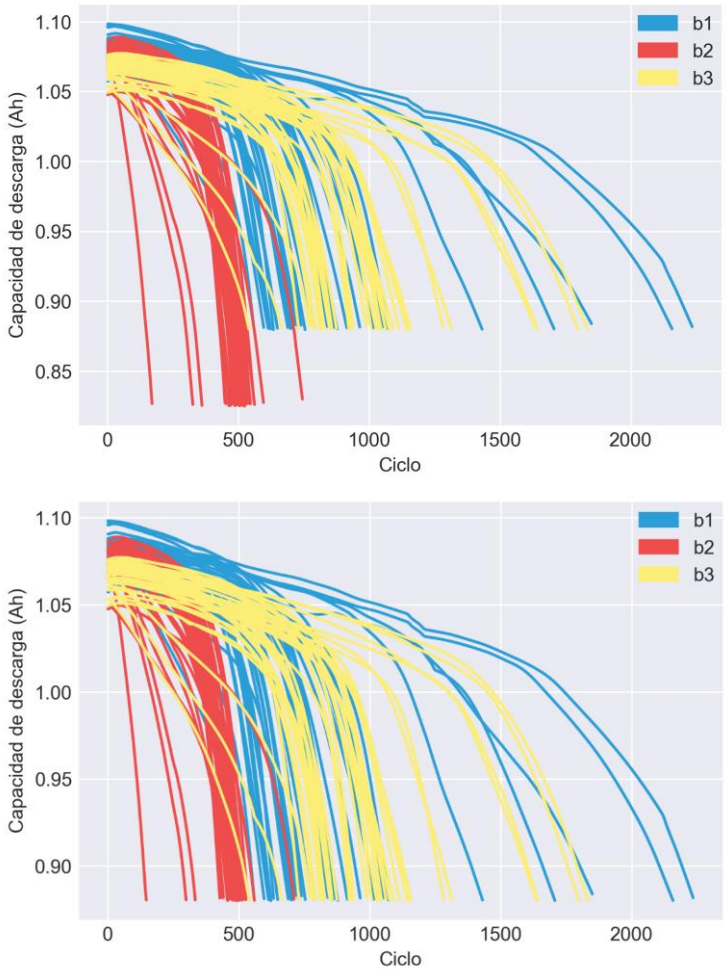

Figura 2-4. a) Curvas QD-Ciclos procesadas. b) Mismas curvas eliminando mediciones sobrantes. Nota: las gráficas están más suavizadas para su correcta visualización.

Una vez que tenemos todos los lotes igualados utilizaremos otra leyenda a partir de ahora que dará una mejor idea de los datos con los que estamos trabajando. Esta será un mapa de color que irá de más claro a más oscuro conforme la vida útil de cada batería sea más larga. Tal como se muestra en la Figura 2-5. En esta figura no parece que sea relevante, puesto que, al fin y al cabo, da la misma información que la figura 2-4-b y se puede ver la vida útil de cada batería sin necesidad del mapa de color. Sin embargo, para las gráficas posteriores, en las que solo se verán los primeros ciclos toma mucha más relevancia.

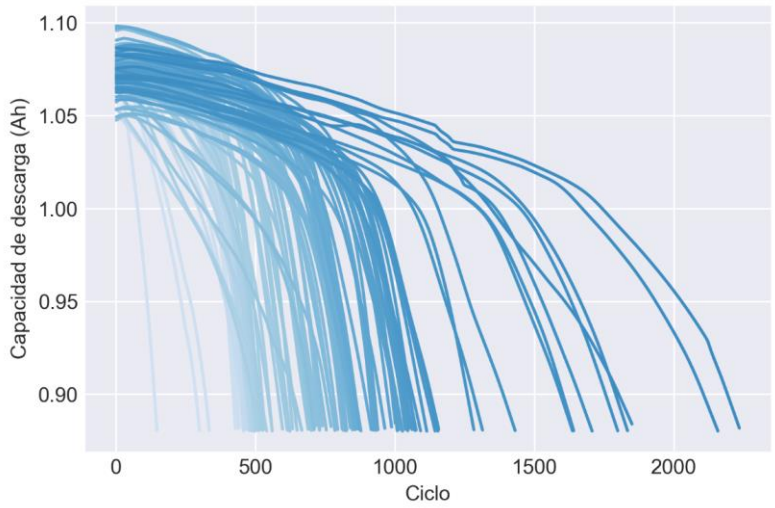

Figura 2-5. Curvas QD-Ciclos con mapa de calor. Cuanto más oscuro, más larga es la duración de la vida útil.

Con estos arreglos ya se pueden observar los ciclos de interés para este trabajo. Como veníamos diciendo, el objetivo de este proyecto es predecir la vida útil con datos sólo de los 100 primero ciclos. En las figuras vistas hasta ahora se ve una clara pérdida de rendimiento conforme van avanzando los ciclos, pero no se puede apreciar si esta pérdida de rendimiento es observable en los primeros ciclos. Sí sabíamos que la degradación era exponencial, y eso se hace evidente viendo las figuras. En la Figura 2-6 podemos comprobar, que no, no parece haber una pérdida de rendimiento a simple vista en estos 100 primeros ciclos para la mayoría de este conjunto de baterías.

No parece haber ninguna correlación entre la capacidad de descarga al inicio y la vida útil, ni con de la capacidad de descarga en el ciclo 100. Quizás sí puede apreciarse alguna correlación con la pendiente de la capacidad de descarga en los 100 primeros ciclos, al menos para algunas de las baterías cuya vida útil es más corta.

Como sabíamos, la degradación de una batería es exponencial. En los primeros ciclos apenas se llega a apreciar más que en las baterías cuya vida útil es poco más larga que los ciclos considerados. Esto hace que predecirla con tan pocos ciclos sea una tarea complicada.

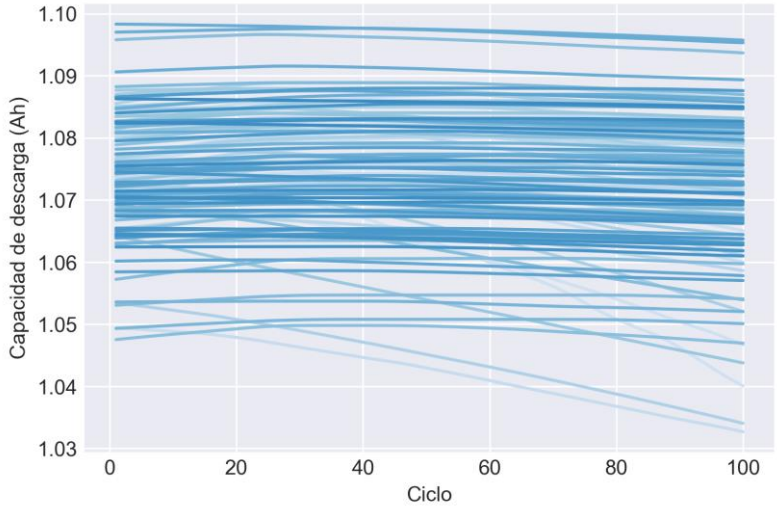

Figura 2-6. Curva QD-Ciclos en los 100 primeros ciclos. Nota: Se ha excluido una de las baterías con una vida inusualmente corta.

2.2.1.2 Resistencia Interna

Pasemos a la siguiente medición, la resistencia interna de la batería (IR) que está estrechamente relacionada con la disminución de a la potencia. Medida en Ohmios (Ω), también depende del estado de carga de la batería. A medida que la resistencia interna aumenta, la eficiencia de la batería disminuye y la estabilidad térmica se reduce, ya que una mayor cantidad de la energía de carga se convierte en calor. Esto hace que las mediciones de la resistencia interna en sucesivos ciclos de una batería aumenten conforme se vaya acercando al final de su vida útil. Sin embargo, tal y como pasa con las medidas de la capacidad de descarga, también tiene crecimiento exponencial y en los primeros ciclos no se puede apreciar cambios significativos.

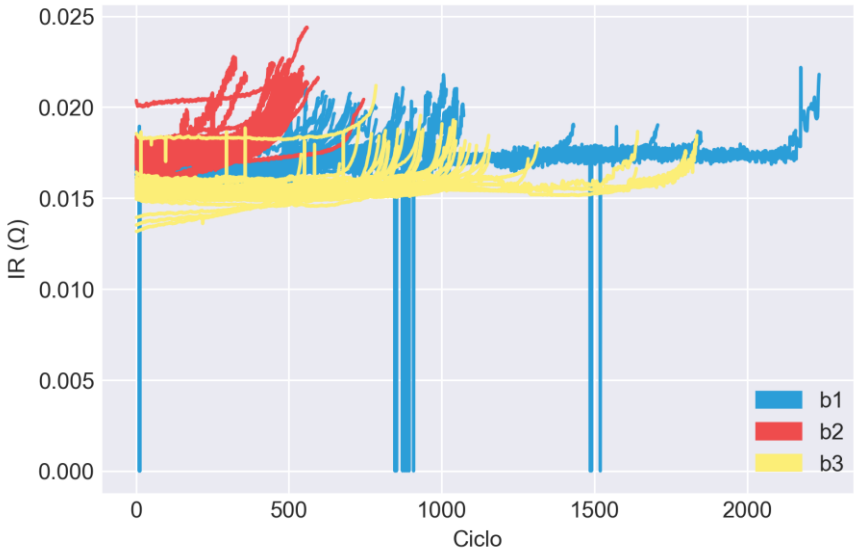

Figura 2-7. Resistencia interna de cada batería en cada ciclo de su vida.

Como puede verse en la Figura 2-7, y tal como les pasaba a las mediciones de la capacidad de descarga, también tiene algunas erróneas, sobretodo en el primer lote (b1).

Mirando a baterías individuales para hacerle el mismo tratamiento que a la capacidad de descarga obtenemos la Figura 2-8.

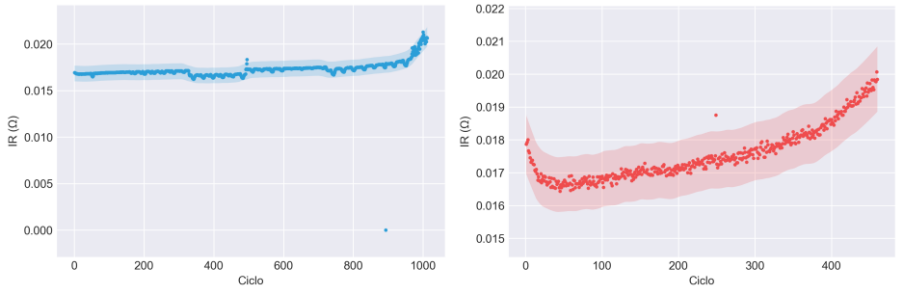

Figura 2-8. Dos curvas IR-Ciclos aleatorias (b1 y b2) con un rango de valores aceptables sombreado.

Tal como decía la teoría, en la Figura 2-9, se ve como, al contrario que la capacidad de la descarga, la resistencia interna de las baterías aumenta conforme se acercan a su vida final. También se aprecia el crecimiento exponencial, que también complican las predicciones tempranas basadas en esta medida.

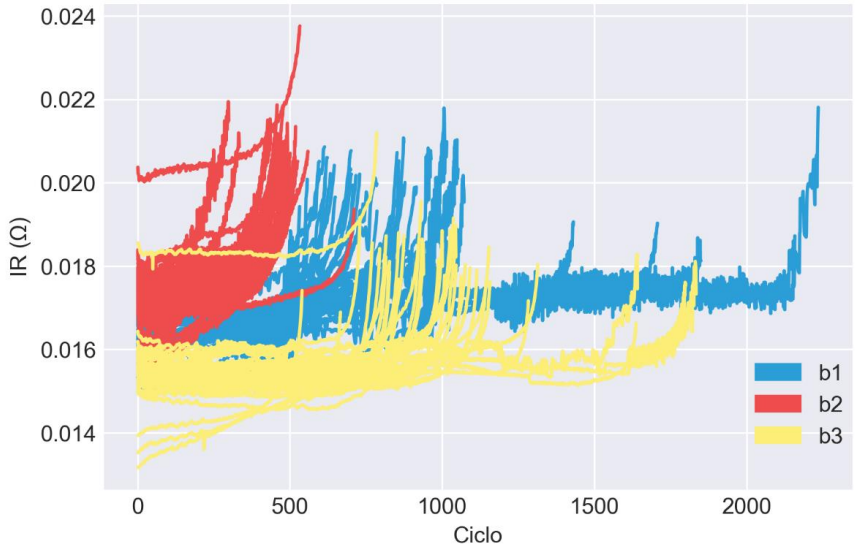

Figura 2-9. Curvas IR-Ciclos procesadas.

Una vez tratadas podemos ver cómo se comportan en los 100 primeros ciclos en la Figura 2-10. Aunque efectivamente, no vemos degradación en el sentido de crecimiento de la resistencia interna conforme avanzan los primeros ciclos, sí parece observarse cierta correlación entre la impedancia y la vida útil, ya que las curvas más altas están más claras (menor vida útil) que las bajas. También, las baterías que duran menos pueden tener mediciones más inestables y por lo tanto una mayor varianza, que podría usarse también como predictor del modelo.

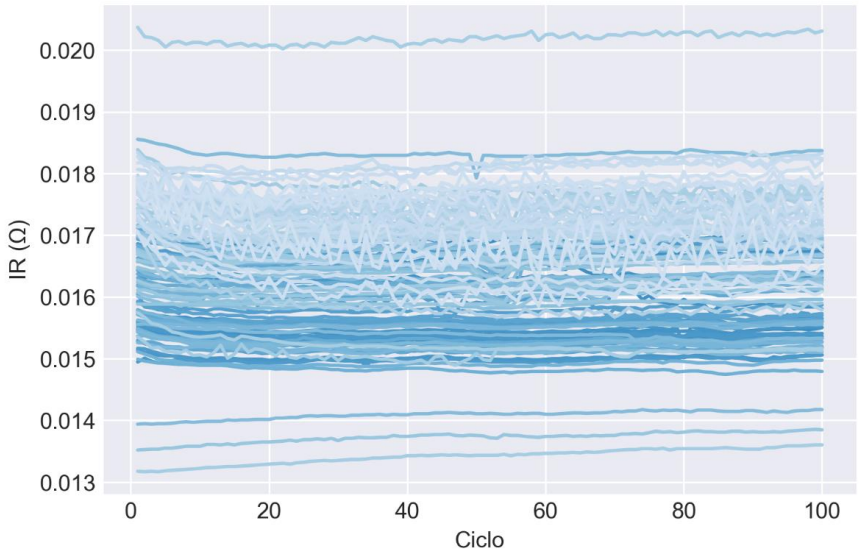

Figura 2-10. Curva IR-Ciclos en los 100 primeros ciclos.

2.2.1.3 Tiempo de Carga

La siguiente medida es el tiempo que tarda en cargarse cada batería en cada ciclo. Los datos en crudo pueden verse en la Figura 2-11.

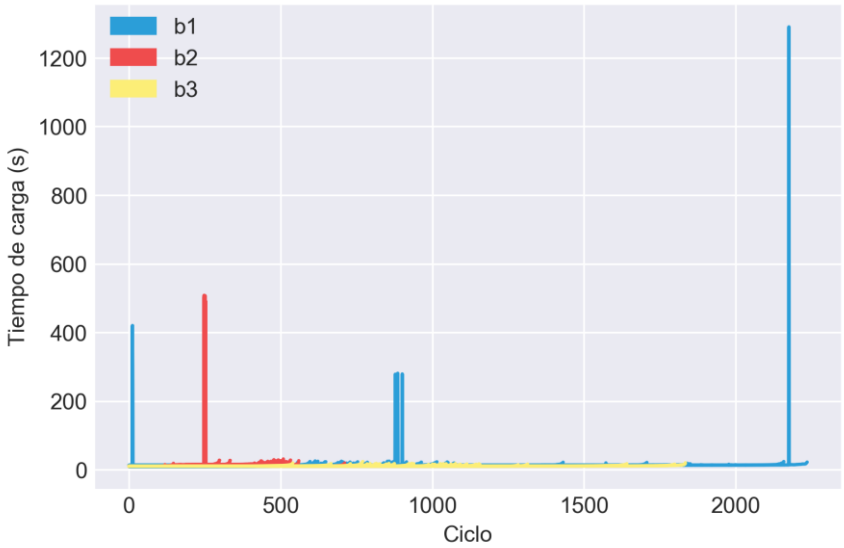

Figura 2-11. Tiempo de carga de cada batería en cada ciclo de su vida.

Llegando alguna medida a sobrepasar los 1200 segundos. Esta vez hay 'outliers' en todos los lotes. Después de un tratamiento igual que para los datos anteriores y que se puede ver en la Figura 2-12, comprobamos que tampoco se puede ver un cambio importante en el comportamiento de las baterías en cuanto a lo que tardan en cargarse hasta cerca del final de la vida de estas, que sufre, otra vez más, un crecimiento exponencial.

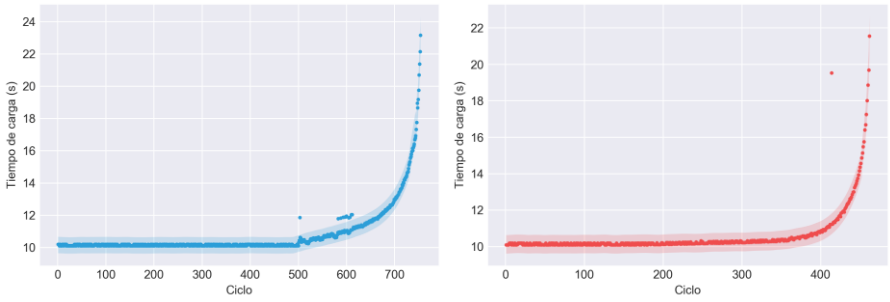

Figura 2-12. Dos curvas chargetime-Ciclos aleatorias (b1 y b2) con un rango de valores aceptables sombreado.

Procesando estos datos y mirando a los 100 primeros ciclos obtenemos la gráfica que puede observarse en la Figura 2-13. Volvemos a ver, como en el caso de la resistencia interna, aunque en una mayor proporción, una correlación entre el tiempo de carga en los primeros ciclos y la vida de la batería, ya que las baterías que más duran se encuentran en un nivel más alto en la gráfica, es decir, tardan más segundos en cargar en los primeros ciclos. Esto se debe principalmente al método de carga, sabemos que una carga rápida puede ser más perjudicial para la vida de la batería, aunque esto a nosotros no nos afecta, puesto que solo buscamos relación estadística entre variables y aquí la hay, lo que parece hacer de esta medida el mejor predictor que tenemos hasta ahora. Esto se comprobará en secciones posteriores con métodos estadísticos numéricos. También puede observarse una ligera correlación entre la pendiente de las medidas y la vida útil, al menos en algunas baterías.

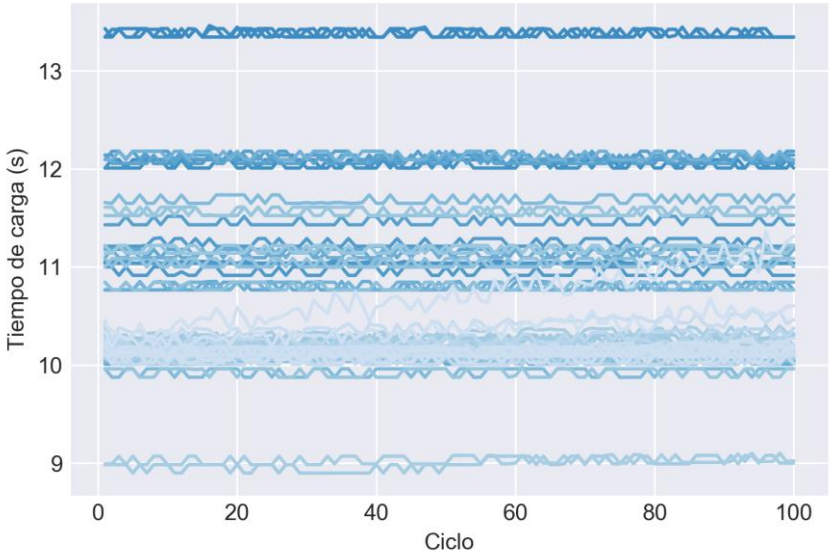

Figura 2-13. Curvas chargetime-Ciclos en los 100 primeros ciclos.

2.2.1.4 Temperatura máxima

Como medida de temperatura por ciclo nos quedamos con el máximo. La Figura 2-14 muestra las curvas que hace la temperatura máxima de cada ciclo por cada batería. Esta vez, ni siquiera hay un cambio recurrente cuando la batería sufre degradación.

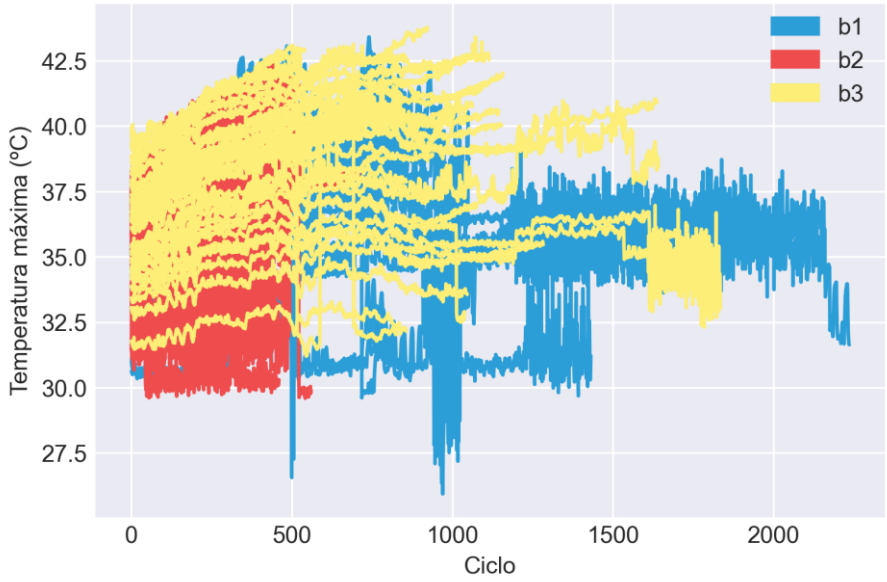

Figura 2-14. Temperatura máxima en cada ciclo.

No parece haber valores extraños, o al menos no valores que podamos asegurar que son extraños como sí pasaba en las otras medidas. La Figura 2-15 muestra los primeros 100 ciclos. A simple vista no encontramos ningún tipo de correlación con el objetivo.

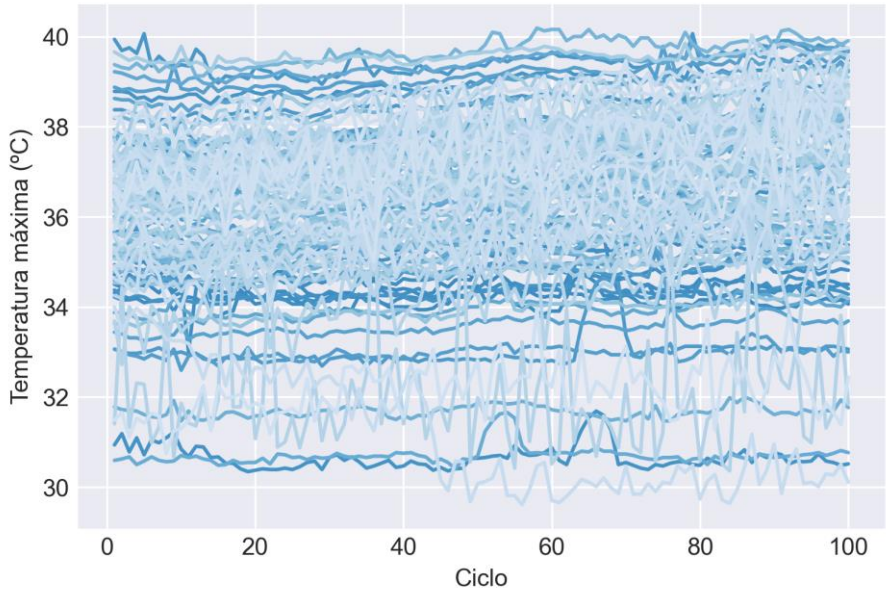

Figura 2-15. Curva Tmax-Ciclos en los 100 primeros ciclos.

Ahora pasaremos a las medidas tomadas en cada ciclo.

2.2.1.5 Capacidad de descarga en función del voltaje (Qdlin)

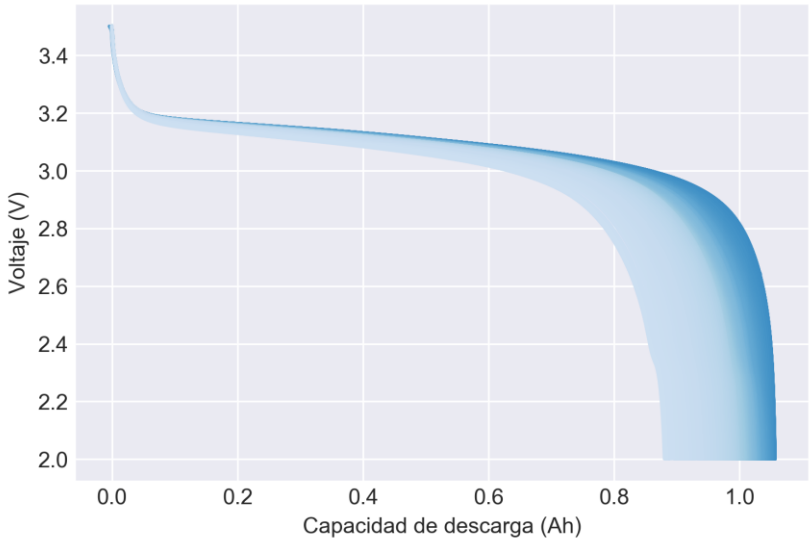

Figura 2-16. Curvas de la Capacidad de descarga frente al voltaje para una batería cualquiera.

Volvemos a la medición más importante, esta vez medida a lo largo de cada ciclo. Como cada ciclo tiene un número de medidas diferentes usaremos los datos interpolados linealmente en mil puntos y como el rango de voltaje es idéntico para cada ciclo, consideramos la capacidad como una función del voltaje, en contraposición al voltaje en función de la capacidad, para mantener una base uniforme para comparar los ciclos. Esta transformación, $\Delta Q(V)$, es de particular interés porque las curvas de voltaje y sus derivadas son una rica fuente de datos que es efectiva en el diagnóstico de la degradación [25].

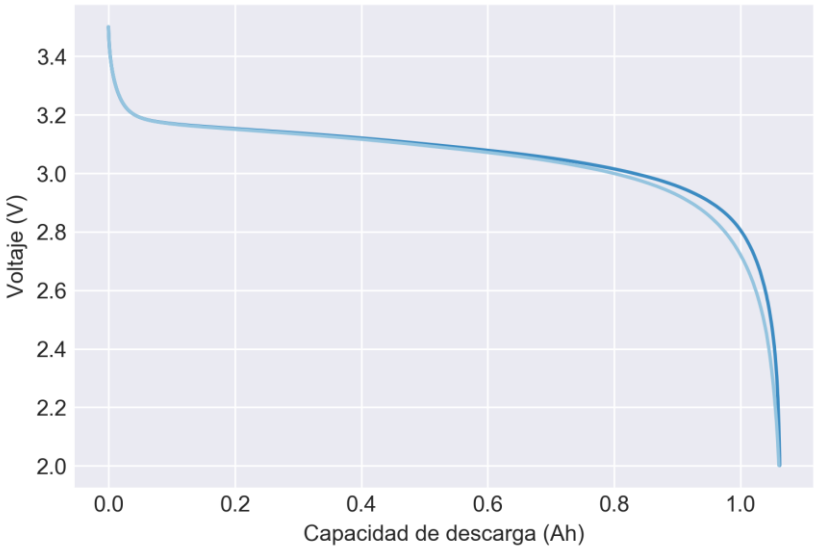

Figura 2-17. Curvas de la Capacidad de descarga frente al voltaje para una batería cualquiera de los ciclos 100 y 10.

La Figura 2-16 muestra todas las curvas de capacidad de la descarga frente al voltaje de una batería aleatoria. El mapa de color en este caso indica el paso de los ciclos ordenados hasta el final de su vida útil, aunque como se podía intuir, tiene perfecta correlación con la capacidad de descarga al final de cada ciclo. Y no solamente al final, también toda la curva que hacen las medidas está correlacionada con la vida de la batería. Usaremos eso en nuestro modelo predictivo, concretamente la diferencia entre la curva en los primeros ciclos y en el ciclo 100.

En la Figura 2-17 se puede apreciar la diferencia entre el ciclo 10 y el 100 de una batería del lote 1. La hipótesis es que, a más diferencia entre las curvas, más cerca estará la batería del final de vida útil.

2.2.1.6 Temperatura en función del voltaje (Tdlin)

La otra medición que tenemos interpolada y podemos medirla frente al voltaje es la temperatura en ºC. La Figura 2-18 muestra todas las curvas de capacidad de la descarga frente al voltaje de una batería aleatoria. El mapa de color, como para Qdlin, indica el paso de los ciclos ordenados hasta el final de su vida útil y, esta vez, no se aprecia correlación con la degradación. Aun así, también probaremos la misma característica de diferencia entre un ciclo 100 y 10 y probaremos su capacidad predictiva. Lo más interesante de esta gráfica es la gran varianza que hay en las mediciones de temperatura, probaremos distintas características basadas en la varianza de la temperatura para comprobar si a inestabilidad de la temperatura en la carga tiene alguna relación con la vida útil.

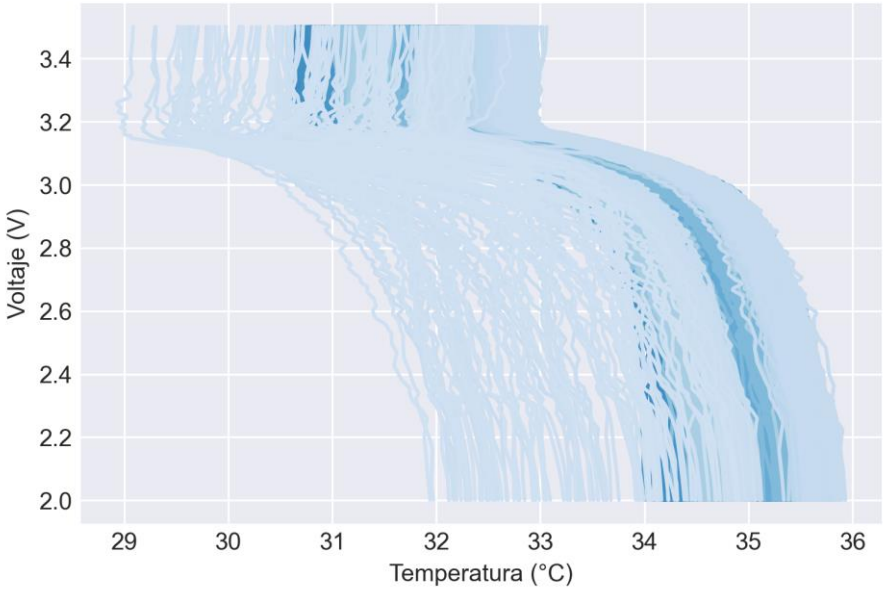

Figura 2-18. Curvas de la Temperatura frente al voltaje para una batería cualquiera.

De todas estas medidas tendremos que elegir, basándonos en su capacidad predictiva, un conjunto de características o variables de entrada que puedan ser introducidas en el modelo. Serán una magnitud escalar, por lo que tendremos que usar estadística descriptiva para resumir la información proporcionada por las mediciones.

2.2.1.7 Ciclos de vida

Por último, hablaremos de nuestra variable objetivo, la vida útil de la batería. Es la variable que queremos predecir. Es el número de ciclos que tiene una batería antes de llegar al 80% de su capacidad nominal.

Veremos la distribución de las baterías según su vida útil en un histograma dividido por los tres sets que vamos a utilizar para entrenar y validar los modelos, respetando la división que han seguido en el artículo original. Tendremos un set para el entrenamiento y dos para los test. Cada uno consta de unas 40 baterías. El último lote se tomó después de los primeros experimentos y conforma el segundo test.

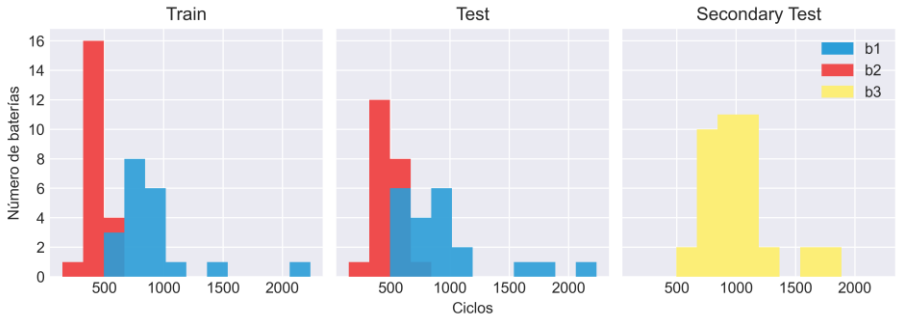

Figura 2-19. Histogramas de la distribución de las baterías por ciclos.

La Figura 2-19 muestra el histograma de la distribución. Vemos que la mayoría se concentra entre 500 y 1000 ciclos de vida, aunque hay alguna que incluso supera los 2000.

2.2.2 Selección de características

La selección de características es el proceso de reducir el número de variables de entrada cuando se desarrolla un modelo predictivo. La selección de características se centra principalmente en eliminar del modelo los predictores no informativos o redundantes. Es deseable reducir el número de variables de entrada tanto para reducir el costo computacional como, en algunos casos, para mejorar el rendimiento del modelo.

"La presencia de variables no informativas puede añadir incertidumbre a las predicciones y reducir la eficacia general del modelo." [26]

Los métodos de selección de características consisten en evaluar la relación entre cada variable de entrada y la variable objetivo utilizando técnicas estadísticas y seleccionando las variables de entrada que tienen la relación más fuerte con la variable objetivo.

La mayoría de estas técnicas son univariantes, lo que significa que evalúan cada predictor de forma aislada. En este caso, la existencia de predictores correlacionados permite seleccionar predictores importantes, pero redundantes. Las consecuencias obvias de esta cuestión son que se eligen demasiados predictores y, como resultado, surgen problemas de colinealidad.

La selección de características también está relacionada con las técnicas de reducción dimensional, en el sentido de que ambos métodos buscan menos variables de entrada para un modelo predictivo. La diferencia radica en que la selección de características selecciona las características que deben mantenerse o eliminarse del conjunto de datos, mientras que la reducción de la dimensionalidad crea una proyección de los datos que da lugar a características de entrada totalmente nuevas.

En este proyecto se usarán utilizarán técnicas estadísticas para evaluar la relación entre cada variable de entrada y la variable objetivo y, tomando estas puntuaciones como base, se elegirán las variables de

entrada que se utilicen en el modelo.

Normalmente tenemos una matriz $X \in \mathbb{R}^{n*p}$ con alta dimensionalidad, y una variable objetivo y. Un algoritmo de selección de características seleccionará un subconjunto de $k < p$ columnas, $X_S \in \mathbb{R}^{n*k}$, que son más relevantes para la variable objetivo y [27]

En estadística, la correlación o dependencia es cualquier relación estadística, ya sea causal o no, entre dos variables. En el sentido más amplio, la correlación es cualquier asociación estadística, aunque comúnmente se refiere al grado en que un par de variables se relacionan linealmente.

Las correlaciones son útiles porque pueden indicar una relación predictiva que puede ser explotada en la práctica. La correlación es la medida de la forma en que dos o más variables se relacionan entre sí. Hay varios coeficientes de correlación. El más común de ellos es el coeficiente de correlación de Pearson, que es sensible sólo a una relación lineal entre dos variables, por lo que mantendremos la relación lineal usando transformaciones logarítmicas[28].

El coeficiente de correlación de Pearson es la covarianza de las dos variables dividida por el producto de sus desviaciones estándar.

Usaremos la correlación tanto para medir la capacidad predictiva de las variables de entrada como para evitar colinearidad entre ellas.

La selección inicial de estas variables como posible entrada a nuestro modelo se ha tomado partiendo de las que usan los investigadores en el modelo original, la visualización de los datos y a las que hemos añadido otras nuevas basándonos en su habilidad predictiva, no en su significado físico.

La capacidad predictiva de estas variables iniciales la hemos medido, como ya se ha comentado, con el coeficiente de correlación de Pearson entre la propia variable transformada logarítmicamente para mantener la linealidad y la variable objetivo, es decir, la vida útil. Hemos seleccionado veinte características por prueba y error, manteniendo alguna con baja correlación con el objetivo con las que se demostrará que realmente son perjudiciales para el modelo y que el coeficiente de Pearson es una buena medida del valor predictivo para, al menos, nuestros datos concretos.

Las veinte variables se pueden dividir en grupos partiendo del conjunto de datos de los que han derivado. Las características de los mismos grupos son más propensas a la colinealidad, aunque se pueden obtener varias variables descriptivas de los mismos datos sin caer en ella.

- Basadas en la curva de la capacidad de descarga frente al voltaje.

  Son las más valiosas como predictores. Son funciones de la diferencia entre la curva en el ciclo 100 y la curva en el ciclo 10 para cada batería.

  $\Delta Q(V) = Q_{100}(V) - Q_{10}(V)$

  a) Mínimo = $\log(|\min(\Delta Q(V))|)$

  b) Varianza = $\log\left(\left|\frac{1}{p-1}\sum_{i=1}^{p}(\Delta Q(V)_i - \overline{\Delta Q}(V))^2\right|\right)$

  c) Área = $\log(\int \Delta Q(V) dV)$

- Basadas en la curva de temperatura frente al voltaje.

  $\Delta T(V) = T_{100}(V) - T_{10}(V)$

  d) Mínimo = $\log(|\min(\Delta T(V))|)$

  e) Varianza = $\log\left(\left|\frac{1}{p-1}\sum_{i=1}^{p}(\Delta T(V)_i - \overline{\Delta T}(V))^2\right|\right)$

  f) Área = $\log(\int \Delta T(V) dV)$

  g) Máximo de las varianzas = $\log\left(\max\left(\left|\frac{1}{p-1}\sum_{i=1}^{p}(T(V)_i - \bar{T}(V))^2\right|_0^{100}\right)\right)$

h) Varianza de las varianzas = $\log\left(\text{var}\left(\left|\frac{1}{p-1}\sum_{i=1}^{p}(T(V)_i - \bar{T}(V))^2\right|_0^{100}\right)\right)$

i) Máximo de la temperatura = $\max(T(V)|_0^{100})$

- Basadas en la curva de descarga frente a los ciclos

  j) Pendiente de la línea de ajuste lineal a la curva de la capacidad de descarga en los 100 primeros ciclos

  k) Intersección de la línea de ajuste lineal a la curva de la capacidad de descarga en los 100 primeros ciclos

  l) Capacidad de descarga en el ciclo 2

  m) Capacidad de descarga en el ciclo 100

- Basadas en el tiempo de carga

  n) Media del tiempo de carga en los 6 primeros ciclos $= \frac{1}{5}\sum_{i=2}^{6} CT_i$

  o) Pendiente de la línea del ajuste lineal a la curva del tiempo de carga en los 100 primeros ciclos

  p) Varianza = $log\ (|\frac{1}{p-1}\sum_{i=1}^{p}(CT_i - \overline{CT})^2|)$

- Basadas en la resistencia interna

  q) Mínimo de la resistencia interna = $\min(IR|_0^{100})$

  r) Diferencia de la resistencia interna en el ciclo 100 y el 2 = $IR_{100} - IR_2$

  s) Pendiente de la línea del ajuste lineal a la curva de la resistencia interna en los 100 primeros ciclos

  t) Varianza = $log\ (|\frac{1}{p-1}\sum_{i=1}^{p}(IR_i - \overline{IR})^2|)$

La Figura 2-20 Figura 2-20son gráficas de este conjunto de características cuyos ejes (y, x) son la vida útil de la batería (recordemos, nuestra variable objetivo) y la característica interpolada entre 0 y 1 para darle la misma importancia a cada una en el modelo predictivo. También aparece el coeficiente de correlación de pearson entre cada característica y el objetivo.

Los puntos de la gráfica corresponden sólo a las baterías seleccionadas para el entrenamiento y el cálculo de la correlación está también hecho sobre ellas. Así nos aseguramos una mejor generalización del modelo.

Como vemos hay variables con alta correlación con el objetivo y otras con una leve correlación o ninguna que pueden empeorar las predicciones del modelo. También se pueden observar semejanzas entre varias de las variables, lo que implica colinealidad. Para estudiar este hecho construiremos una matriz de correlación (Figura 2-21) entre las variables y descartaremos las que tengan menos correlación con el objetivo entre las relacionadas.

Se aprecian diferentes grupos de variables correlacionadas entre sí. Como esperábamos, las variables obtenidas del mismo conjunto de datos tienen más correlación entre ellas que con las variables de los demás conjuntos. Observamos, como ya se podía ver en la Figura 2-20, que las tres primeras variables son casi idénticas. Son las calculadas a partir de las curvas de capacidad de descarga. De entre ellas, la que más correlación tiene con el objetivo es la de la varianza (variable b). Eliminaremos las otras dos del modelo final.

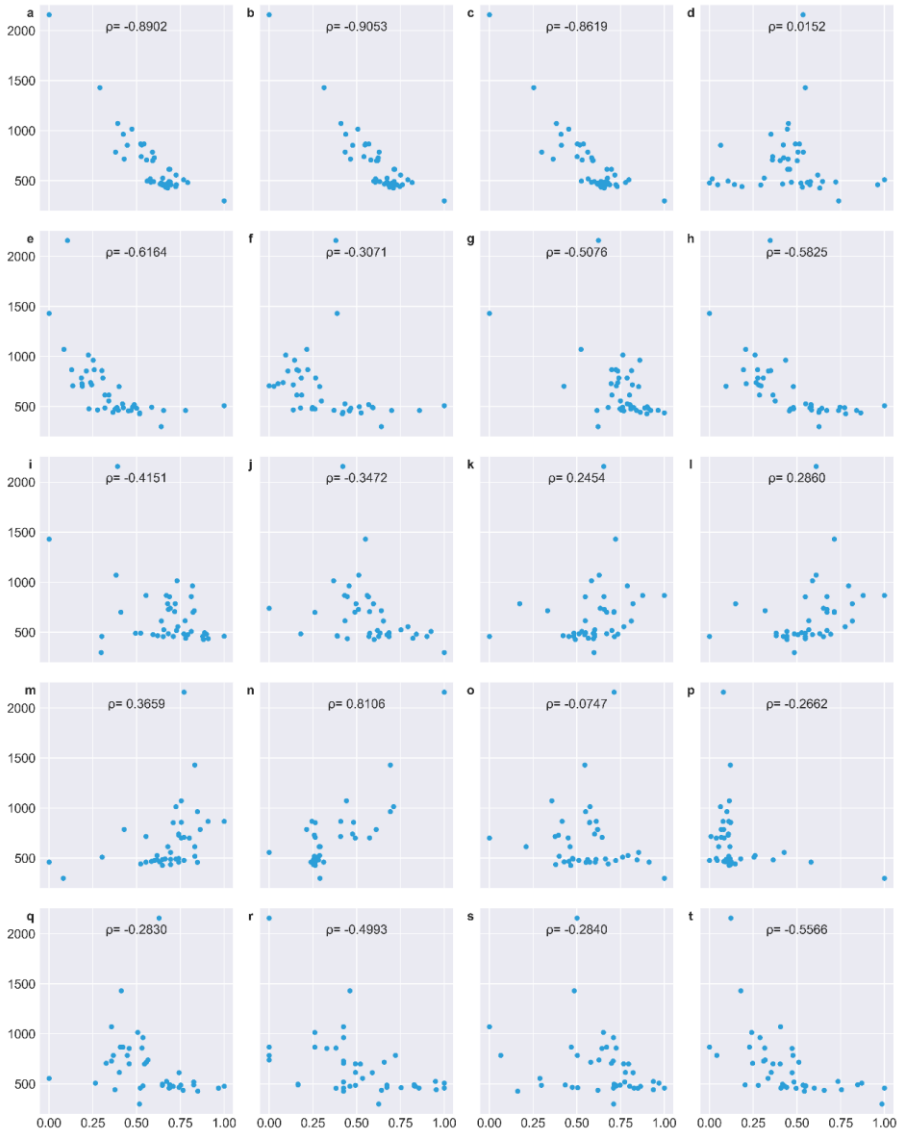

Figura 2-20. Características frente a la vida útil de la batería. **a**, mínimo de la diferencia entre la curva de la capacidad de descarga frente al voltaje en el ciclo 100 y 10 (ΔQ100-10(V)). **b**, varianza (ΔQ100-10(V)). **c**, área (ΔQ100-10(V)). **d**, mínimo de la diferencia entre la curva de la temperatura frente al voltaje en el ciclo 100 y 10 (ΔT100-10(V)). **e**, varianza (ΔT100-10(V)). **f**, área (ΔT100-10(V)). **g**, máximo de la varianza de las curvas de temperatura frente a voltaje en cada punto de voltaje. **h**, varianza de la varianza de las curvas de temperatura frente a voltaje en cada punto de voltaje. **i**, máximo de la temperatura. **j**, pendiente de la curva de la capacidad de descarga frente a la vida útil. **k**, intersección de la pendiente a la curva anterior. **l**, capacidad de descarga en el ciclo 2. **m**, capacidad de descarga en el ciclo 100. **n**, media del tiempo de carga de los ciclos 2 a 6. **o**, pendiente de la media del tiempo de carga frente a los ciclos. **p**, varianza de la media del tiempo de carga. **q**, mínimo de la resistencia interna. **r**, diferencia de la resistencia interna entre el ciclo 100 y 2. **s**, pendiente de la resistencia interna. **t**, varianza de la resistencia interna.

En el siguiente conjunto, el de las características obtenidas de la temperatura, vemos como no tienen tanta correlación como las del primer grupo, pero sigue siendo demasiado alta y puede perjudicar las predicciones. Entre la varianza de la diferencia entre el ciclo 100 y 10 (f) y el área (e), nos quedamos nuevamente con la varianza, que a su vez también es colineal con la varianza de las varianzas de todos los ciclos (h) la cual también se queda fuera del modelo. El máximo de la varianza (g) a su vez, es redundante con el máximo de las temperaturas (i) y mejor predictor, por lo que elimina a este último.

Las variables obtenidas de las curvas de la capacidad de descarga medida al final de cada ciclo tienen colinealidad perfecta entre la intersección de la curva y la recta de ajuste lineal (k) con la medición de la capacidad en el segundo ciclo (l), el cual tienen más correlación con el objetivo.

El conjunto de las características obtenidas del tiempo de carga no presenta una alta colinealidad, aunque sí las tiene con el primero de los conjuntos. Especialmente la media del tiempo de carga (n), uno de los mejores predictores. Mantendremos todas las variables de este conjunto.

Por último, de entre las variables de las medidas de resistencia interna eliminamos la diferencia entre el ciclo 100 y el 2 (r) por su alta dependencia con la variable de la varianza (t).

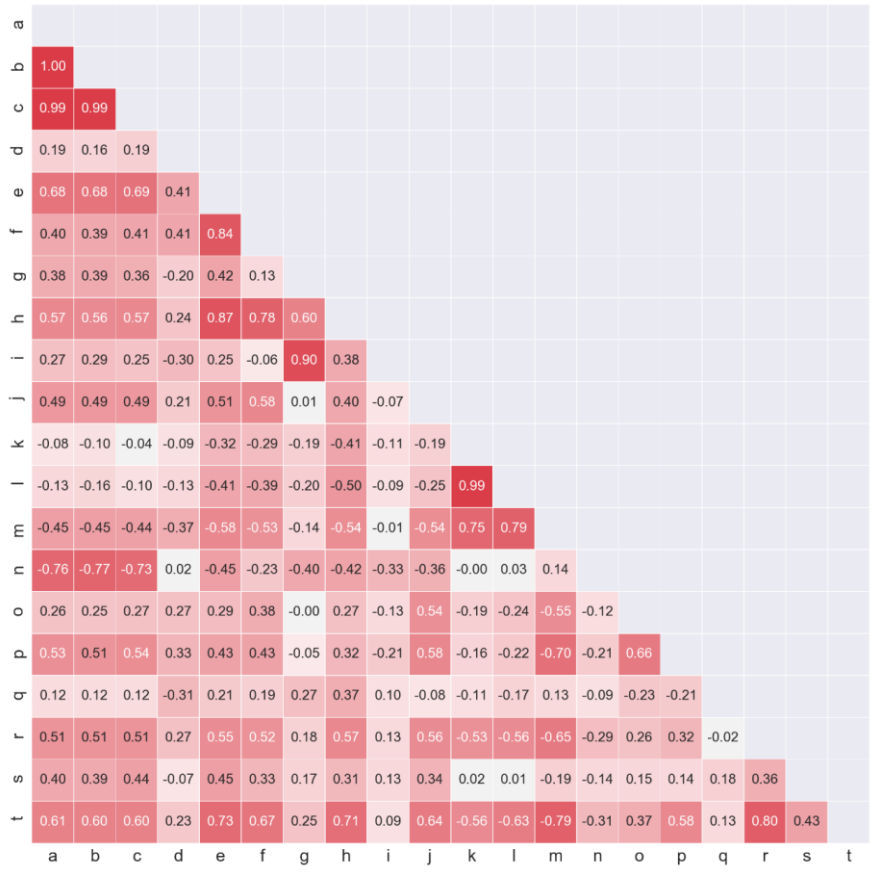

Figura 2-21. Matriz de correlación entre las posibles variables de entrada.

En total nos quedan 12 variables o características. Lo que no quiere decir que se vaya a usar las 12 para el modelo, puesto que no todas son buenos predictores, y tampoco se sabe a priori cuál será el límite en términos de correlación que haga de una variable un predictor suficientemente bueno como para mejorar los resultados. Iremos entrenando varios modelos empezando con solo una variable y añadiendo más progresivamente, cada vez incrementando en una nueva siendo ésta la siguiente de mayor puntuación en el análisis de correlaciones.

3 MODELOS

Como ya se ha comentado, el objetivo de este trabajo es predecir lo de la forma más precisa posible la vida útil de una batería usando los datos que hemos desarrollado en el capítulo anterior. Esto puede hacerse con varios modelos, llamados modelos de regresión, de los que se explicarán y se probarán los más usados y teóricamente válidos en este capítulo. Otra forma de predecir será agrupar las baterías en dos grupos según un umbral en la vida útil y clasificar las baterías en uno de esos dos grupos. Como este último problema es más sencillo, se utilizarán sólo datos de los primeros 5 ciclos en contraposición a los 100 que se usan para la predicción precisa que es la regresión. Dividiremos, pues, este capítulo en esas dos secciones, una para la regresión y otra para la clasificación.

3.1 Regresión

El análisis de regresión es una técnica estadística para investigar y modelar la relación entre las variables. Las aplicaciones de la regresión son numerosas y ocurren en casi todos los campos, incluyendo la ingeniería, las ciencias físicas y químicas, la economía, la administración, las ciencias biológicas y de la vida, y las ciencias sociales. De hecho, el análisis de regresión puede ser la técnica estadística más utilizada.

En casi todas las aplicaciones de la regresión, la ecuación de regresión es sólo una aproximación a la verdadera relación funcional entre las variables de interés. Estas relaciones funcionales se basan a menudo en la teoría física, química u otra teoría de ingeniería o científica, es decir, en el conocimiento del mecanismo subyacente. Por consiguiente, estos tipos de modelos se denominan a menudo modelos mecanicistas. Los modelos de regresión, por otra parte, se consideran modelos empíricos y su proximidad al modelo mecanicista depende de la complejidad de este y de lo bien escogidos que estén los datos que forman las variables de entrada. [29] En el caso de este trabajo, precisamente buscamos predecir de manera empírica lo que no se ha podido hacer con un estudio mecánico debido a su complejidad.

En general, la variable objetivo y puede estar relacionada con k regresores (o predictores), x_1, x_2, …, x_k, tal que

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{3–1}$$

A esto se le denomina modelo de regresión lineal múltiple porque hay más de un regresor involucrado. El adjetivo lineal se emplea para indicar que el modelo es lineal en sus parámetros β_0, β_1, $\beta_2, \ldots$, β_k, no porque y sea una función lineal de las x.

El objetivo del análisis de regresión es estimar los parámetros (a priori desconocidos) en el modelo. Este proceso se denomina ajuste del modelo a los datos. La siguiente fase del análisis de regresión es comprobar lo apropiado que es el modelo y la calidad de este. Si el modelo no logra generalizar a otros datos a los que se han utilizado para el ajuste significa que deben realizarse cambios, o bien en los datos de entrada o bien utilizando otras técnicas de regresión. Por ello, el análisis de regresión es un proceso iterativo.

Otro aspecto a tener en cuenta es que un modelo de regresión no implica causa efecto en la relación entre las variables, al igual que no lo implicaba el coeficiente de correlación. Incluso aunque exista una relación empíricamente fuerte entre una o más variables, no quiere decir que una cause la otra.

Para establecer esta causalidad, la relación entre los regresores y el objetivo debe tener una base fuera de la relación estadística de los datos, debe ser sugerida por consideraciones teóricas. El análisis de regresión puede ayudar a confirmar la relación causa efecto estudiada en la teoría mecanicista del proceso en cuestión, pero no puede proclamarla por sí sola. En este trabajo, aunque partimos de datos recogidos por expertos en el campo y elegidos por consideraciones teóricas, usamos únicamente motivos estadísticos para la elección entre estas variables y cualquier otra consideración queda fuera de los límites de este proyecto.

Hay diferentes técnicas para estimar los parámetros del modelo de regresión. Según el problema al que nos enfrentemos alguna puede funcionar mejor que otra, por eso es recomendable probar no sólo cambios en las variables de entrada si no en el propio modelo.

3.1.1 Mínimos Cuadrados Ordinarios (OLS)

El enfoque más común para estimar los parámetros del modelo es el de los mínimos cuadrados ordinarios (OLS, por sus siglas en inglés) donde las βs se eligen para minimizar

$$\sum_{i=1}^{n}[y_i - (\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_k x_{ki})]^2 \tag{3–2}$$

Para cada elección de los parámetros $\hat{\beta}$s estimados, la respuesta estimada esperada para una observación i será

$$\widehat{y_i} = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \hat{\beta}_2 x_{2i} + \cdots + \hat{\beta}_k x_{ki} \tag{3–3}$$

La diferencia entre el valor observado y_i y el valor esperado $\widehat{y_i}$ es llamada residuo. El método de los mínimos cuadrados ordinarios minimiza la suma de los cuadrados de los residuos [30].

Pese a ser la más usada por su simplicidad, es bien sabido que los OLS a menudo tiene un mal desempeño tanto en la predicción como en la interpretación. Se han propuesto técnicas de regularización para mejorarla. Añaden un límite en los coeficientes para que no se ajusten demasiado bien a los datos concretos e impida generalizar a los datos que nunca ha visto, esto se denomina overfitting. Por ejemplo, la regresión de Ridge [31] minimiza la suma residual de los cuadrados sujetos a un límite en la norma L2 de los coeficientes. Como un método de regularización, la regresión de Ridge logra su mejor predicción a través de una compensación entre el sesgo y la varianza. Sin embargo, la regresión de Ridge no puede producir un modelo parsimonioso, ya que siempre mantiene todos los predictores en el modelo.

Una prometedora técnica llamada Lasso fue propuesta por Tibshirani [32]. La Lasso es método de penalización de los mínimos cuadrados imponiendo un límite en la norma L1 en los coeficientes de regresión. La penalización L1 hace que el método Lasso regularice y haga selección de variables simultáneamente. Comparando la evaluación de las predicciones de la Lasso y la Ridge y se encontró que ninguna dominaba uniformemente sobre la otra [33]. Sin embargo, mientras la selección de variables se hace cada vez más importante en el análisis moderno de datos, la regularización Lasso se hace mucho más atractiva. En este trabajo ya se ha realizado un estudio previo de las características o variables de entrada. Sin embargo, para una mejor predicción, ayuda que haya también un filtro automático en el propio modelo.

3.1.2 Elastic Net

Los modelos de regresión lineal se pueden escribir de manera compacta usando notación matricial. Definiendo las siguientes matrices y vectores

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{k1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & \cdots & x_{kn} \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} \tag{3–4}$$

El modelo de regresión será

$$y = X\beta \tag{3–5}$$

El criterio Elastic Net queda, para λ_1, λ_2 no negativos y fijados

$$L(\lambda_1, \lambda_2, \beta) = |y - X\beta|^2 + \lambda_2|\beta|^2 + \lambda_2|\beta|_1 \tag{3–6}$$

Y el coeficiente estimado $\hat{\beta}$ saldrá de minimizar la ecuación

$$\hat{\beta} = \arg min_\beta \{L(\lambda_1, \lambda_2, \beta)\} \tag{3–7}$$

Esto puede verse como una penalización del método de los mínimos cuadrados. Sea $\alpha = \lambda_2/(\lambda_1 + \lambda_2)$; resolver $\hat{\beta}$ es equivalente a al problema de optimización

$$\hat{\beta} = \arg min_\beta |y - X\beta|^2, \qquad \text{sujeto a } (1 - \alpha)|\beta|_1 + \alpha|\beta|^2 \tag{3–8}$$

La función $(1 - \alpha)|\beta|_1 + \alpha|\beta|^2$ es la penalización Elastic Net, que es una combinación convexa de la Lasso y la Ridge. Para $\alpha = 1$, la Elastic Net es simple regularización Ridge, y para $\alpha = 0$, es Lasso [34].

Se optimizará el parámetro α para encontrar el mejor regularizador para nuestro modelo.

3.1.3 Perceptrón Multicapa

Las redes neuronales artificiales (RNA) son modelos matemáticos y computacionales inspirados en sistemas biológicos, adaptados y simulados en computadoras; han surgido como un intento de desarrollar modelos que emulen las características del cerebro humano. Las RNA están compuestas por un conjunto de elementos simples (neuronas) que se interconectan en paralelo y organizadas. Las redes neuronales artificiales conforman modelos que permiten describir muchos fenómenos del mundo real y que pueden ser comparados y aplicados con los modelos estadísticos en problemas de predicción, clasificación y clustering.

Una RNA está compuesta por un conjunto de neuronas, conectadas entre sí y por donde se envía la información necesaria. El modelo de una RNA se basa en la estructura de un grafo dirigido cuyos nodos representan las neuronas que están interconectadas a través de arcos dirigidos. Cada arco establece la relación entre nodos y tiene asociado un peso numérico w_{ji} que determina la fuerza y el signo de la conexión y sirve para propagar la activación de la neurona j hacía la neurona i. En la Figura 3-1 se muestra el esquema de una red neuronal con n neuronas en la capa de entrada, m neuronas en la capa oculta y una neurona en la capa de salida.

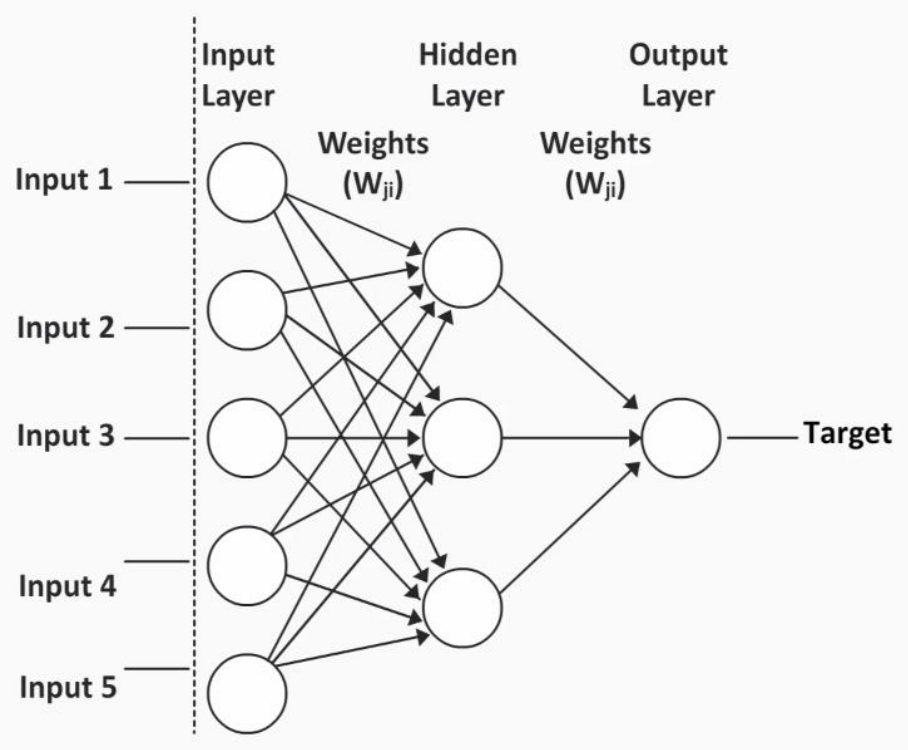

Figura 3-1. Estructura de un perceptrón multicapa [35].

Las neuronas se organizan por niveles o capas con un número determinado de neuronas en cada una de ellas y que describen la forma en que estas se interconectan entre sí. En una RNA, se pueden distinguir tres tipos de capas:

- Capa de entrada. Es la capa cuyos nodos (inputs) reciben directamente la información proveniente de las fuentes externas de la red (variables de entrada).
- Capas ocultas. Son internas a la red y no tienen contacto directo con el entorno exterior. Las neuronas de esta capa pueden estar interconectadas de distintas maneras, lo que determina, junto con su número, las diferentes topologías de las RN.
- Capa de salida. Los nodos (outputs) de esta capa permiten realizar la transferencia de información de la red hacía el exterior (variables de salida).

Una neurona recibe un conjunto de señales que representa la información del estado de activación de todas las neuronas con las que se encuentra conectadas. Sea i el valor de entrada de la neurona en un instante y la conexión entre la neurona x_i y j está ponderada por un peso w_{ij}. Entonces considerando un efecto aditivo, la entrada neta que recibe una neurona está dada por la expresión:

$$Net_j = w_0 + \sum_{i=1}^{N} w_{ij} x_i \tag{3–9}$$

La función de salida, conocida también como función de transferencia o activación, es en gran parte responsable del comportamiento de la red neuronal. La salida de una neurona se obtiene mediante la aplicación de la función de transferencia. Una de las activaciones más usadas es la función rectificadora conocida como unidad lineal rectificada (ReLU, por sus siglas en ingles), definida por $f(x) = \max(0, x)$.

El perceptrón simple consta de una capa de entrada, una capa oculta y una de salida; es un modelo de gran uso y resulta eficiente para modelar muchas situaciones del mundo real. La estructura del perceptrón está conformada por una capa de entrada con una neurona, que permite recibir los valores de las variables predictoras; una neurona en la capa oculta, donde se realiza la acumulación de la entrada (entrada neta) y la aplicación de la función de transferencia (generalmente una función umbral) para obtener la salida esperada, y la capa oculta de salida donde se compara la salida esperada con la salida observada; la diferencia sería el cálculo del error o residuo.

La única neurona de salida del perceptrón realiza la suma ponderada para obtener la entrada neta y restar un valor de umbral, el resultado se aplica a la función de transferencia. Los valores de los pesos pueden fijarse o hallarse utilizando diferentes algoritmos de entrenamiento de la red. El aprendizaje del perceptrón simple es el proceso de entrenamiento del mismo perceptrón. Consiste en calcular la combinación lineal a partir de un conjunto de variables de entrada, con un término de sesgo, aplicándole una función de activación, dando lugar a la salida de la red. Así, los pesos de la red se van ajustando por el método de aprendizaje supervisado por corrección de error, de tal manera que se va comparando la salida esperada con el valor de la variable de salida que se desea obtener, la diferencia es el error o residual. El aprendizaje termina cuando los residuales han superado un umbral o han dejado de mejorar.

El perceptrón multicapa (MLP) es una generalización del perceptrón simple en que se considera más de una capa oculta. Consiste en un conjunto de nodos organizados por capas, de modo que una neurona puede recibir entradas solo de aquellas situadas en la capa inmediatamente inferior. En general, en un MLP cada uno de los nodos calcula una combinación lineal de las entradas que llegan a él, le añade un sesgo y, finalmente, le aplica una función de activación, llamada también de transferencia, que por regla general traslada cualquier entrada real a un rango generalmente acotado, dando lugar así a la salida del nodo, que puede ser una de las entradas de un nuevo nodo. Tanto el perceptrón simple como multicapas puede tener una o más salidas, cada una de ellas con un conjunto de pesos y un sesgo asociados. Generalmente, se considera una misma función de activación o transferencia para cada nodo de la misma capa, aunque es posible usar diferentes funciones de activación para cada neurona.

Una diferencia importante entre las redes neuronales y los modelos estadísticos (como los de regresión vistos anteriormente) consiste en que los parámetros obtenidos por la red neuronal no son susceptibles de una interpretación práctica. No podemos saber inmediatamente cómo los pesos de la red o los valores de activación de las neuronas están relacionados con el conjunto de datos manejados. Así, a diferencia de los modelos estadísticos clásicos, no parece tan evidente conocer en una red el efecto que tiene cada variable explicativa sobre la variable de respuesta. Por tanto, es importante tener en cuenta que las similitudes que se puedan establecer entre RNA y modelos estadísticos siempre harán referencia al aspecto predictivo, pero no al aspecto explicativo.

Los modelos de regresión lineal simple o múltiple pueden ser identificados como un modelo de red neuronal perceptrón multicapa. Si el modelo neuronal posee una variable de entrada y una de salida, se tiene una regresión lineal simple. Si hay varias variables de entrada y solo una de salida, el modelo neuronal corresponde al modelo de regresión lineal múltiple.

Una regresión lineal múltiple corresponde a un modelo perceptrón simple sin capa oculta y función de activación de identidad. En este caso, la red neuronal tendrá tantos nodos de entrada como variables regresoras y solo un nodo de salida que corresponde a la variable dependiente, nuestro objetivo a predecir [36].

3.2 Clasificación

El siguiente objetivo del trabajo es clasificar las baterías en dos grupos según su ciclo de vida y usando solo datos de los 5 primeros ciclos, en contraposición a los 100 que hemos usado para la regresión. Los grupos serán 'High' para las baterías con más vida útil que 550 ciclos y 'Low' para las que tengan menos o igual. La selección de características ha partido de las calculadas para la regresión y añadiendo o quitando de manera empírica. Al final se ha optado por el siguiente grupo:

Para $\Delta Q(V) = Q_5(V) - Q_4(V)$,

- Mínimo = $log(|min(\Delta Q(V))|)$
- Varianza = $log\ (|\frac{1}{p-1}\sum_{i=1}^{p}(\Delta Q(V)_i - \overline{\Delta Q}(V))^2|)$

Para $\Delta T(V) = T_5(V) - T_4(V)$,

- Mínimo = $log(|min(\Delta T(V))|)$
- Varianza = $log\ (|\frac{1}{p-1}\sum_{i=1}^{p}(\Delta T(V)_i - \overline{\Delta T}(V))^2|)$

- Máximo de la temperatura = $\max(T(V)|_0^5)$
- Capacidad de descarga en el ciclo 2
- Media del tiempo de carga en los 5 primeros ciclos = $\frac{1}{4}\sum_{i=2}^{5} CT_i$
- Varianza de la resistencia interna = $log\ (|\frac{1}{p-1}\sum_{i=1}^{p}(IR_i - \overline{IR})^2|)$

Nótese como se han usado las mismas ecuaciones para sacar las mismas características, pero usando los primeros ciclos. En el caso de la diferencia entre dos ciclos característicos del principio y del final, han cambiado de ser el ciclo 100 y 10 por el 5 y el 4.

Se optimizará el parámetro α para encontrar el mejor regularizador para nuestro modelo.

3.2.1 Regresión Logística

Todas las regresiones que se han discutido hasta ahora se han caracterizado por una variable objetivo que es continua, pero modelar una variable categórica que tenga dos o más categorías es a veces necesario. En esta situación, la respuesta esperada $\hat{y}_i$ es la probabilidad condicional de los eventos de interés dados los valores de los predictores.

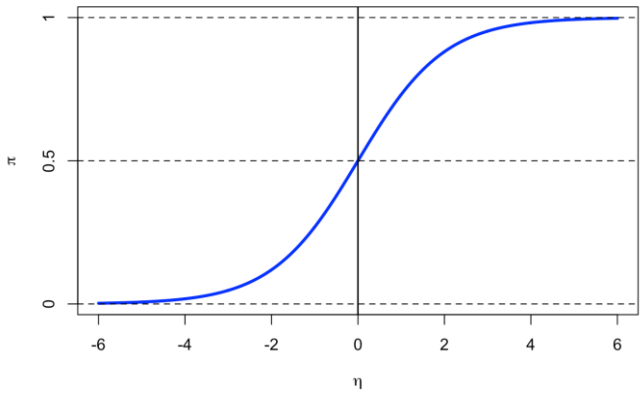

Figura 3-2. Curva de una función logit [37].

La función con la que trabaja la regresión logística que genera curvas en forma de S como la de la figura 3-2 es la función logit. Sea $\pi(x)$ la probabilidad de un suceso para los valores observados de los predictores x. La ratio de probabilidad de un suceso a fallar será,

$$\frac{\pi(x)}{1-\pi(x)} \tag{3–10}$$

(ya que la probabilidad de fallo es $1-\pi$). Nótese que puede variar de 0 a ∞ mientras que las probabilidades varían de 0 a 1.

La función logit se define como el logaritmo natural de la fórmula 3-10,

$$\ell(x) = log\left[\frac{\pi(x)}{1-\pi(x)}\right] \tag{3–11}$$

La hipótesis de la regresión logística relaciona linealmente el logit con los predictores, esto es

$$\ell(x) = log\left[\frac{\pi(x)}{1-\pi(x)}\right] = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k \tag{3–12}$$

Nótese que el logit varía entre -∞ y ∞ al variar la probabilidad entre 0 y 1, haciendo a los logits más aptos para el ajuste lineal. Resolviendo para $\pi(x)$ obtenemos el equivalente de la ecuación anterior y proporciona explícitamente la curva S para las probabilidades,

$$\pi_i(x) = \frac{e^{\beta_0+\beta_1 x_{1i}+\beta_2 x_{2i}+\cdots+\beta_k x_{ki}}}{1+\ e^{\beta_0+\beta_1 x_{1i}+\beta_2 x_{2i}+\cdots+\beta_k x_{ki}}} \tag{3–13}$$

El modelo plantea una relación aditiva/multiplicativa entre un predictor y las probabilidades, con un cambio multiplicativo en las probabilidades de éxito de $e^{\beta j}$ asociado a un aumento de una unidad en x_j que mantiene fijo todo lo demás en el modelo. Este valor representa la relación entre las probabilidades para x_j + 1 y las probabilidades para x_j. Una pendiente de 0 corresponde a una relación nula entre el logit (y por lo tanto la probabilidad) y el predictor dadas las demás variables del modelo, una pendiente positiva corresponde a una relación directa, y una pendiente negativa corresponde a una relación inversa [38].

4 IMPLEMENTACIÓN EN PYTHON

La aplicación práctica de todo lo que se ha tratado hasta ahora se explicará en este capítulo. La implementación se hará en Python.

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, con semántica dinámica, lo que lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso como lenguaje de scripts. La sintaxis simple y fácil de aprender de Python enfatiza la legibilidad y por lo tanto reduce el costo de mantenimiento del programa. Python soporta módulos y paquetes, lo que fomenta la modularidad del programa y la reutilización del código. El intérprete de Python y la extensa biblioteca estándar están disponibles en forma de código fuente para todas las plataformas principales, y pueden ser distribuidos libremente.

Es un lenguaje muy usado debido al aumento de productividad que proporciona. Como no hay un paso de compilación, el ciclo de edición-prueba-depuración es increíblemente rápido. Depurar programas Python es fácil: un error o una mala entrada nunca causará un fallo de segmentación. En cambio, cuando el intérprete descubre un error, hace una excepción. Cuando el programa no detecta la excepción, el intérprete imprime un rastro de la pila. Un depurador a nivel de fuente permite la inspección de variables locales y globales, la evaluación de expresiones arbitrarias, el establecimiento de puntos de ruptura, el paso por el código una línea a la vez, y así sucesivamente. El depurador está escrito en propio Python, lo que demuestra el poder introspectivo de Python. Por otro lado, a menudo la forma más rápida de depurar un programa es añadir unas cuantas declaraciones impresas a la fuente: el rápido ciclo de edición-prueba-depuración hace que este sencillo enfoque sea muy efectivo [39].

4.1 Librerías esenciales de Python

Para aquellos que están menos familiarizados con el ecosistema de datos de Python y las librerías en este trabajo, se hará un breve resumen de algunas de ellas [40].

4.1.1 NumPy

NumPy, abreviatura de Numerical Python, ha sido por mucho tiempo la piedra angular de la computación numérica en Python. Proporciona las estructuras de datos, los algoritmos y el soporte de otras librerías necesarios para la mayoría de las aplicaciones científicas que implican datos numéricos en Python.

Más allá de las rápidas capacidades de procesamiento de vectores que NumPy añade a Python, uno de sus usos primarios en el análisis de datos son como un contenedor para que los datos se pasen entre algoritmos y las librerías. Para los datos numéricos, las matrices NumPy son más eficientes para almacenar y manipular que las otras estructuras de datos incorporadas en Python. Además, las librerías escritas en un lenguaje de nivel inferior, como C o Fortran, pueden funcionar con los datos almacenados en una matriz NumPy.

Por lo tanto, muchas herramientas de computación numérica para Python asumen las matrices NumPy como estructura principal.

4.1.2 pandas

pandas proporciona estructuras de datos de alto nivel y funciones diseñadas para hacer que el trabajo con datos estructurados o tabulares sea rápido, fácil y expresivo. Desde su aparición en 2010, ha

ayudado a que Python sea un entorno de análisis de datos potente y productivo. Los principales objetos de los pandas que se utilizarán en este TFG son el DataFrame, una estructura de datos tabular y orientada a columnas con etiquetas tanto de filas como de columnas, y la Serie, un objeto matriz unidimensional etiquetado.

pandas combina las ideas de alto rendimiento y de computación en matriz de NumPy con las capacidades de manipulación flexible de datos de las hojas de cálculo y las bases de datos relacionales (como SQL). Proporciona una sofisticada funcionalidad de indexación para facilitar la remodelación, la división y el corte, las agregaciones y la selección de subconjuntos de datos. Dado que la manipulación, preparación y limpieza de datos es tan importante en un TFG de este tipo, pandas será la librería más utilizada.

4.1.3 matplotlib

matplotlib es la librería de Python más popular para producir gráficos y otras visualizaciones de datos bidimensionales. Está diseñada para crear gráficos adecuados para su publicación. Aunque hay otras librerías de visualización disponibles para los programadores de Python, matplotlib es la más utilizada y como tal tiene generalmente una buena integración con el resto del ecosistema.

4.1.4 scikit-learn

Desde el inicio del proyecto en 2010, scikit-learn se ha convertido en el principal kit de herramientas de aprendizaje de máquinas de uso general para los programadores de Python. Incluye submódulos para tales modelos como:

Clasificación: SVM, nearest neighbors, random forest, regresión logística, etc.

Regresión: Lasso, ridge, etc.

Clustering: k-means, spectral clustering, etc.

Reducción de la dimensionalidad: PCA, selección de características, factorización de matrices, etc.

Selección de modelos: Grid search, validación cruzada, métricas, etc.

Preprocesamiento: Extracción de características, normalización, etc.

scikit-learn ha sido fundamental para permitir que la Python sea un lenguaje de programación de ciencia de datos productivo.

4.2 Descripción del software

Partimos de la estructura de datos .mat proporcionada en tres archivos, uno por lote de baterías. También se proporcionan unos códigos en Python que leen estos datos y construyen un solo archivo Pickle. El módulo Pickle implementa protocolos binarios para serializar y deserializar una estructura de objetos Python. "Pickling" es el proceso por el cual una jerarquía de objetos Python se convierte en un flujo de bytes, y "unpickling" es la operación inversa, por la cual un flujo de bytes (de un archivo binario o un objeto similar a los bytes) se convierte de nuevo en una jerarquía de objetos [41]. Por desgracia, dado el sistema poco potente con el que trabajamos, no era posible usar este módulo para agrupar los datos ya que daba un error de memoria. Buscando alternativas, la más ligera que encontramos fue el módulo Joblib que computa estos procesos de serialización y desserialización como un pipeline [42].

Una vez conseguido el acceso a estos datos encontramos que son un diccionario anidado, como ya explicamos en el capítulo 2, y procedemos a convertirlos en una estructura DataFrame de pandas. También aprovechamos esta conversión para hacer coincidir el inicio de las mediciones en todas las columnas y para volver a separarlos en tres estructuras, esta vez por columnas (mediciones) en vez de por lotes (filas), así será más fácil de manejar en la memoria y siempre podremos acceder a todos los

datos para una medición concreta cuando necesitemos hacer gráficas o estudio de correlaciones.

La división de las baterías en sets de datos de entrenamiento y tests también viene especificado en estos códigos y no se ha alterado en este trabajo para poder hacer mejores comparaciones.

El siguiente paso del proyecto era comprender mejor los datos, y para ello lo mejor que se puede hacer son visualizaciones de éstos. La librería que usamos para esto es matplotlib. A la vez que vamos haciendo las visualizaciones, nos vamos percatando de la cantidad de valores extraños que hay en los datos, por lo que usamos este mismo archivo para ir tratándolos de manera secuencial. Usamos la función lowess de la librería statsmodels con distintos parámetros de suavizamiento para, al multiplicarlo por un porcentaje –en este caso un $\pm 5\%$-, obtener el rango de valores esperados. Observamos, por las gráficas que los que se salen de ese rango, lo hacen por una cantidad muy grande, son medidas mal tomada. Por eso, se decide igualar todo punto que se salga de ese rango a la propia línea de ajuste lowless, en vez de al límite del rango.

El cálculo de las características a partir de estos datos se hace con distintas librerías. Principalmente se usa NumPy, para obtener todo lo relacionado con la estadística descriptiva de las mediciones. Por ejemplo, cuando buscamos un mínimo usamos la función numy.min() o cuando calculamos la varianza usamos numpy.var(). Para calcular otro tipo de datos, como las pendientes, usamos la librería sklearn. Y para calcular el área entre dos curvas usamos la función area_between_two_curves, de la librería específica similaritymeasures.

Con esto obtenemos 20 variables distintas, que podrían ser la entrada a los modelos. Sin embargo, sabemos que los malos predictores pueden empeorar los resultados. También si dos variables de entrada dan la misma información. Por eso hacemos un estudio de correlación, primero entre la entrada y el objetivo y luego a las entradas entre sí.

Antes de calcular las correlaciones tenemos que dejar todas las características en la misma escala. Así, los modelos le dan a todas la misma importancia y no tienen sesgo las que tengan valores más altos. Entre varias formas de escalar los datos elegimos la más simple, MinMax, normalizándolos entre 0 y 1. La fórmula que usa para ello es

$$\frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{4–1}$$

El principal problema de este escalador es su sensibilidad a los outliers, sin embargo, ya hemos tratado estos con anterioridad. La función en Python es MinMaxScaler de la librería sklearn.

Otro tema a tener en cuenta es que tenemos que escalar los datos en función sólo de los de entrenamiento, para no afectar a las predicciones, lo que trampearía los resultados puesto que le habríamos dado información del test durante el entrenamiento. Esto hace que, aunque los datos del entrenamiento estén normalizados entre 0 y 1, los de los test pueden salirse de estos rangos si son mayores o menores que el máximo o el mínimo de los datos del entrenamiento con el que se han normalizado.

Las correlaciones las calculamos nuevamente con NumPy, y la visualización de las características normalizadas (ahora podemos usar los mismos ejes) con matplotlib.

Descartamos las variables que sufren colinealidad. Se podía haber hecho de forma algorítmica, pero no tenía sentido puesto que era más rápido hacerlo manualmente.

Nos quedan 12 características, y aunque hemos evitado la colinealidad entre ellas, sabemos que no todas son buenos predictores. Por ello, usamos la función sklearn.SelectKBest con criterio de correlación para elegir las k mejores características. De manera iterativa iremos metiendo cada vez más variables empezando con el mejor predictor en cuanto a su correlación con el objetivo y añadiendo en cada iteración una variable más, siendo ésta la siguiente mejor variable.

Antes de hacer este proceso iterativo nos tenemos que parar en los propios modelos. Aquí domina la

librería de aprendizaje de máquina por excelencia, sklearn. Tanto los tres modelos de regresión, incluida la red neuronal, como el clasificador son funciones de esta librería. También lo son el análisis de las métricas de los resultados, como el error porcentual medio o la matriz de confusión. Más importarte sería pararse en la elección de los parámetros de estos modelos.

Para cada una de las iteraciones de las que hemos hablado, se prueban diferentes hiperparámetros, es decir, los parámetros prefijados que no elige el propio modelo conforme a los datos que se le administran. Como, por ejemplo, las regulaciones. Siempre probamos los mismos hiperparámetros y, para cada iteración, nos quedamos con los que den mejores resultados. Para hacer esto usamos una técnica llamada cross-validation o validación cruzada, que divide los datos en *k folds* o partes y deja una de las partes fuera del entrenamiento y ajusta los coeficientes para esos hiperparámetros concretos, repitiendo el proceso con los otros *folds* para cada parámetro y así ver de una manera más generalizada cual se comporta mejor. En nuestro caso usaremos 4 *folds*. La función que se encarga de esto es sklearn.GridSearchCV. La fuerza de la regularización L2 de la red neuronal se probará para cada iteración con éste método, al igual que para el parámetro α de la Elastic Net.

El percertrón multicapa, además, se ha decidido con sólo 3 neuronas en una única capa oculta, aparte de la capa de entrada con una neurona por variable de entrada y la de salida, con una neurona que determinará el objetivo. Añadir más neuronas en la capa oculta o más capas ocultas provocaba overfitting incluso con una regularización fuerte, que como ya hemos explicado en algún capítulo anterior, se refiere al ajuste demasiado preciso a los datos del entrenamiento en concreto impidiendo su generalización a datos que no haya visto el modelo. El optimizador será Adam y las activaciones ReLU.

En el caso de la clasificación, el modelo no tiene una única salida con un resultado concreto que intenta predecir la variable objetivo de la batería en cuestión. Tiene dos salidas que suman uno y serán las probabilidades de que dicha batería pertenezca a cada clase (“High” y “Low”). Como en el caso de los modelos de regresión lineal, tenemos distintos hiperparámetros que hay que elegir. Como, por ejemplo, las regularizaciones. Usaremos, como ya se hizo en la regresión, una validación cruzada para obtener los mejores resultados. Un nuevo parámetro a tener en cuenta es el peso de las clases. En un set de datos balanceado, donde ninguna clase predomina en número sobre la otra, no habría usar este parámetro. Pero con el que trabajamos hay más baterías de clase “High” que de “Low” por lo que darles un peso más alto a las baterías “Low” para compensarlo es una buena práctica.

Todos los códigos pueden verse en el anexo.

4.3 Errores cometidos en la implementación

La implementación detallada del apartado anterior explica sólo el código definitivo del proyecto. Hasta llegar a este se ha pasado por un proceso iterativo en el que ha habido errores y correcciones.

El error principal que se ha cometido en el proyecto fue empezar directamente por los modelos sin entender ni, como más tarde se vio necesario, tratar los datos. Se crearon una serie variables de entrada sin realizar ninguna clase de estudio previo de correlaciones con el objetivo ni entre ellas y esto llevó, como es lógico a resultados poco satisfactorios. Fue entonces cuando nos percatamos que había valores extraños en algunas características de ciertas baterías, e intentamos corregirlas en las propias variables creadas, no en los datos en crudo. El tratamiento fue parecido al que finalmente usamos, pero al hacerlo sobre las variables, los resultados obtenidos, aunque mejores que en un primer momento, podían ser tramposos. Otro error en esta etapa del proyecto fue normalizar las variables usando todos los datos en la fórmula 4-1, en vez de sólo los datos del entrenamiento. Esto podía llevar también a mejores resultados aparentes sobre nuestros tests, pero desde luego a un peor modelo en cuanto a su generalización.

Finalmente se optó por empezar por el principio, transformar las estructuras dict de Python a

pd.DataFrame para poder estudiarlas mejor y fue cuando nos dimos cuenta de que no sólo había valores extraños, si no incoherencias en el inicio de las mediciones.

La siguiente iteración fue directamente suavizar los datos con la función statsmodels.lowess buscando el mejor parámetro de suavizamiento. Sin embargo, sabemos que así se puede perder información de mediciones reales, no errores en las mediciones, que es lo que buscamos.

Por último, llegamos a la implementación definitiva que ya ha sido explicada. También destacar que, aunque para cada subproceso también ha habido iteraciones para hacerlo lo mejor posible, no son tan relevantes como las comentadas en esta sección.

5 RESULTADOS

En éste capítulo detallaremos los resultados obtenidos y los compararemos con los del artículo original, comprobando si hemos conseguido mejorar sus predicciones. Las pruebas se harán en las mismas condiciones que el artículo, separando los sets de datos en los mismos conjuntos de baterías que ellos. Hay dos tests, uno en el que hay baterías de los dos primeros lotes (mismos lotes del entrenamiento) y otro en el que sólo hay baterías del tercer lote, obtenido después de los primeros experimentos.

5.1 Regresión

Como ya se ha comentado en capítulos anteriores, para ver con cuántas variables funcionan mejor los modelos de regresión, se ha usado un proceso iterativo en el que se han añadido cada vez más variables empezado por la mejor y siguiendo un orden decreciente de la correlación con el objetivo. Usaremos dos métricas para medir la eficacia de los modelos. La primera de ellas será el error porcentual medio (MAPE, ecuación 5-1), que mide la precisión de las predicciones como un porcentaje.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{y}_i - y_i|}{y_i} \tag{5–1}$$

Como vemos en la Figura 5-1, la predicción va mejorando al ir introduciendo más características hasta que llega un punto en el que las características nuevas son malos predictores y empeora. Especialmente en el primer test, el segundo tiene unos resultados más erráticos. El mejor modelo será con k = 9, que minimiza el error de ambos test, aunque en el segundo haya modelos con menos variables con resultados ligeramente mejores.

Los tres modelos obtienen un resultado muy parecido (Tabla 5-1**Error! Reference source not found.**), mejorando los resultados del artículo original en más de 4 puntos en el primer test, y quedándose muy cerca de los resultados en el segundo test. Es también de destacar el poco error que tiene el modelo original en los datos de entrenamiento en comparación con los de este trabajo, pudiendo estar sufriendo overfitting lo que podría dar lugar a una menor generalización, aunque no es algo que pueda afirmarse taxativamente sin tener acceso a más datos. En cualquier caso, que nuestros modelos tengan un error similar en los datos de entrenamiento y los del test incluso en la red neural, siendo este un sistema más complejo y propenso al overfitting, parece indicar que funcionarían parecidos para cualquier set de las mismas características.

La Tabla 5-2 muestra otra métrica importante. La raíz del error cuadrático medio (RMSE, ecuación 5-2). Es la desviación típica de los residuos o errores en la predicción. Los residuos, recordemos, miden cómo de lejos están las predicciones de la variable objetivo. El error cuadrático medio es la medida de cómo de dispersos están estos residuos o, en otras palabras, cómo de concentrados están alrededor de la línea de ajuste de la regresión.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2} \tag{5–2}$$

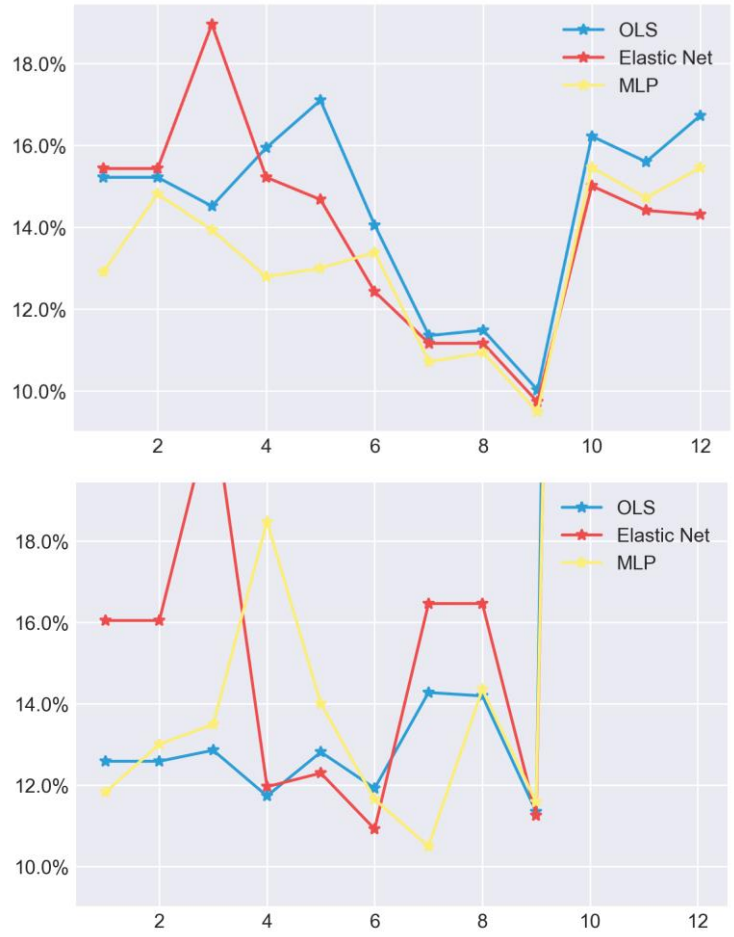

Figura 5-1. Error porcentual medio para cada uno de los modelos del proceso iterativo con k variables de entrada. Cada gráfica es un set de test distintos.

Los resultados son equivalentes a los del error porcentual medio. La diferencia entre estas dos medidas radica en que mientras el MAPE da información relativa y puede ser comparada con otros modelos de otros problemas distintos, el RMSE da información cuantitativa sobre el problema concreto. Con esta medida sabemos en cuántos ciclos nos estamos equivocando de media en la predicción de nuestra vida útil. Además, tomar la raíz cuadrada del promedio de errores cuadrados tiene algunas implicaciones interesantes para el RMSE. Dado que se les hace el cuadrado a los errores antes de ser promediados, el RMSE da un peso relativamente alto a los errores grandes. Por eso suele usarse también como función de optimización para alguna clase de modelos estadísticos.

Tabla 5-1. Error porcentual medio de cada modelo con k = 9 para cada set.

Set	OLS	Elastic Net	MLP	Original
Train	10.93	10.78	10.47	**5.6**
Test	10.02	9.73	**9.49**	14.1
Secondary test	11.34	11.26	11.58	**10.7**

Tabla 5-2. Raíz del error cuadrático medio de cada modelo con k = 9 para cada set.

Set	OLS	Elastic Net	MLP	Original
Train	88.303	88.367	88.544	**51**
Test	106.965	**106.861**	109.191	118
Secondary test	**191.429**	192.697	195.026	214

Por último, la Figura 5-2. Predicción frente a la observación para todos los modelos.muestra las diferencias entre la predicción y la observación. La línea gris sería la perfecta predicción y los puntos son las predicciones que hemos obtenido coloreados según lotes, como hemos venido haciendo en todo el trabajo. Vemos como los modelos tienen prácticamente la misma predicción para todos las baterías, y que, efectivamente, están cerca de la observación. También se observa una tendencia a fallar en las baterías con más vida útil, quedándose corto en la predicción en los dos conjuntos de test para todos los modelos. Esto se debe, probablemente, a un sesgo en el set de entrenamiento, donde hay pocas baterías con una vida larga.

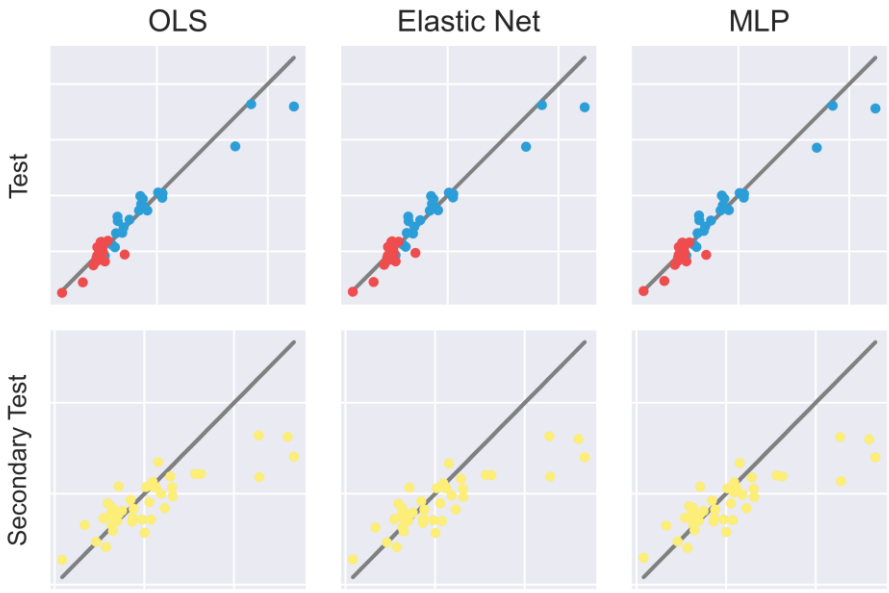

Figura 5-2. Predicción frente a la observación para todos los modelos.

5.2 Clasificación

En cuanto a la clasificación, sólo presentamos una regresión logística, aunque se han probado prototipos de otros clasificadores como los árboles de decisión o los SVC, sin que se hayan obtenido mejores resultados que en la regresión logística.

Tabla 5-3. Resultados del modelo de clasificación.

Set	Métrica	Regresión Logística	Original
Train	Precisión	**100**	97.4
Test		**95.35**	92.7
Secondary test		95.0	**97.5**
Train	Pérdida Logística	**0.3576**	-
Test		**0.3954**	-
Secondary test		**0.4256**	-

Obtenemos los resultados presentados en la

Tabla 5-3. Al igual que en la regresión, volvemos a ver una ligera mejoría en el primer test y algo de empeoramiento en el segundo. Esto podría deberse a la forma de tratar los valores extraños presentes en los datos, que puede hacer que los sets tengan distinto sesgo. También se ha calculado la pérdida logística, un mejor indicador que la precisión. Según el glosario sobre aprendizaje automático de Google Developers, la pérdida es la "medición de la distancia entre las predicciones de un modelo y su etiqueta. Para describirla de manera más pesimista, se trata de una medición de qué tan malo es el modelo. [43]" Otra forma de definirla es la confianza que tiene el modelo en sus predicciones. La fórmula de la pérdida logística es,

$$H_p = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i) \tag{5–3}$$

Donde y_i es la etiqueta y $\hat{y}_i$ es la probabilidad predicha de su clase.

El artículo original no da información sobre la pérdida para que podamos compararla con nuestro modelo.

En total, de las 83 baterías que componen ambos test, el modelo sólo falla en su clasificación en 4, frente a las 5 del modelo original.

Tabla 5-4. Matrices de confusión.

	Test			Secondary Test	
	High	Low		High	Low
High	**20**	**2**	High	**38**	**1**
Low	**0**	**21**	Low	**1**	**0**

La Tabla 5-4 muestra las matrices de confusión de los dos test. Una matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo de clasificación. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las observaciones reales. En nuestro caso vemos que en el primer test tenemos 20 baterías de la clase 'High' todas bien clasificadas por nuestro modelo y 23 baterías de la clase 'Low', de las cuales 2 han sido mal clasificadas como 'High'. En el test secundario tenemos 39 baterías de la clase 'High' con un solo fallo en la clasificación y sólo una batería observada como 'Low' cuya clasificación es errónea.

6 CONCLUSIONES

Los resultados presentados hacen pensar que no es tanto la complejidad del modelo, si no lo bien que los datos caracterizan el proceso que se intenta estudiar y predecir. Sobre todo, cuando no se tiene una gran cantidad de datos con los que trabajar en modelos más sofisticados que sean capaces de entender las relaciones más complejas que se pueden dar entre estos.

Como hemos visto sobre todo en la red neuronal, intentar mejorar las predicciones por la parte del modelo añadiendo complejidad a sus cálculos provoca overfitting y empeora los resultados y la generalización. Al final, la mejor red neuronal que hemos diseñado tenía sólo una capa oculta con 3 únicas neuronas, una minucia si lo comparamos con redes del estado del arte que emplean millones de parámetros.

Otro tema a tener en cuenta, como hemos explicado en la sección 4.3, es que por muy buenos que fueran los modelos, no conseguían una buena predicción si los datos sobre los que entrenaban tenían incluso sólo unas pocas mediciones erróneas.

Todo esto hace pensar que lo verdaderamente importante a nivel práctico es poder trabajar con el mayor número de datos posible y que éstos estén lo mejor tomados y procesados.

En relación a nuestro problema concreto podemos decir que, efectivamente, los modelos predictivos basados en datos son una opción prometedora para el diagnóstico y el pronóstico de baterías de ion de litio y pueden abrir nuevas oportunidades en su desarrollo, fabricación y optimización.

Se han desarrollado diversos modelos predictivos que usan sólo datos de los primeros ciclos cuando todavía no han mostrado degradación obteniendo predicciones numéricas concretas con un error de menos del 10% usando sólo los 100 primeros ciclos y menos de un 5% cuando sólo se buscaba clasificar en dos grupos usando los 5 primeros ciclos. Éste nivel de predicción se consigue calculando características de entrada al modelo basadas en las curvas de capacidad de descarga frente al voltaje, en vez de usar solo las curvas de capacidad de descarga en relación a los ciclos. Se usan también, otras características extraídas del tiempo de carga, las curvas de temperatura frente al voltaje o la resistencia interna, aunque las más útiles como predictoras, como se ha estudiado en el análisis de correlación, son las primeras.

Todo esto se ha hecho sin conocimiento previo de los mecanismos de degradación de las células o a su química, dándole aún más relevancia a la predicción estadística. Este enfoque puede complementar y ayudar a otros enfoques especializados basados en modelos físicos o semi-empíricos.

En general, este trabajo incide en la idea de que los modelos basados en datos son de una gran relevancia para entender y desarrollar sistemas complejos como pueden serlo las baterías de ion de litio.

REFERENCIAS

[1] J. Vetter, P. Novák, M. Wagner, C. Veit, K.-C. Moller y J. Besenhard, «Ageing mechanisms in lithium-ion batteries,» *J. Power Sources,* 2005.

[2] K. A. Severson, P. M. Attia, N. Jin, N. Perkins y B. Jiang, «Data-driven prediction of battery cycle life before capacity degradation,» *Nature,* 2019.

[3] J. Christensen y J. A. Newman, «A mathematical model for the lithium-ion negative electrode solid interphase.,» *J. Electrochem. Soc.,* 2004.

[4] X. Yang, Y. Leng, G. Zhang, S. Ge y C.-Y. Wang, «Modeling of lithium plating induced aging of lithium-ion batteries: transition from linear to nonlinear aging.,» *J. Power Sources ,* 2017.

[5] Q. Zhang y R. White, «Capacity fade analysis of a lithium ion cell.,» *J. Power Sources,* 2008.

[6] V. Ramadesigan, «Parameter estimation and capacity fade analysis of lithium-ion batteries using reformulated models.,» *J. Electrochem. Soc.,* 2011.

[7] V. Ramadesigan, «Modeling and simulation of lithium-ion batteries from a systems engineering perspective.,» *J. Electrochem. Soc.,* 2012.

[8] W. Waag, C. Fleischer y D. U. Sauer, «Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles.,» *J. Power Sources,* 2013.

[9] B. J. C., «Evaluation of effects of additives in wound Li-ion cells through high precision coulometry.,» *J. Electrochem. Soc.,* 2013.

[10] C. T. Love, M. B. V. Virji, R. E. Rocheleau y K. E. Swider-Lyons, «State-of-health monitoring of 18650 4S packs with a single-point impedance diagnostic.,» *J. Power Sources,* 2014.

[11] P. Raccuglia, «Machine-learning-assisted materials discovery using failed experiments.,» *Nature,* 2016.

[12] M. H. S. Segler, M. Preuss y M. P. Waller, «Planning chemical syntheses with deep neural networks and symbolic AI,» *Nature,* 2018.

[13] A. D. Sendek, «Holistic computational structure screening of more than 12000 candidates for solid lithium-ion conductor materials,» *Energy Environ. Sci.,* 2017.

[14] Z. W. Ulissi, «Machine-learning methods enable exhaustive searches for active bimetallic facets and reveal active site motifs for CO2 reduction,» *ACS Catal,* 2017.

[15] A. Nuhic, T. Terzimehic, T. Soczka-Guth, M. Buchholz y K. Dietmayer, «Health diagnosis and remaining useful life prognostics of lithium-ion batteries using data-driven methods,» *J. Power*

Sources, 2013.

[16] T. Baumhöfer, M. Brühl, S. Rothgang y D. U. Sauer, «Production caused variation in capacity aging trend and correlation to initial cell performance,» *J. Power Sources*, 2014.

[17] B. Saha y K. Goebel, «Battery data set,» *NASA Ames Progn. Data Repos.*, 2007.

[18] S. J. Harris, D. J. Harris y C. Li, «Failure statistics for commercial lithium ion batteries: a study of 24 pouch cells,» *J. Power Sources,* 2017.

[19] K. A. Severson, P. M. Attia, N. Jin, N. Perkins y B. Jiang, ref. 2.

[20] K. A. Severson, P. M. Attia, N. Jin, N. Perkins y B. Jiang, «Data-driven prediction of battery cycle life before capacity degradation,» 2019. [En línea]. Available: https://data.matr.io/1/projects/5c48dd2bc625d700019f3204.

[21] Python S. Foundation, «Python Documentation,» 2020. [En línea]. Available: https://docs.python.org/3/tutorial/datastructures.html.

[22] J. Tukey, Exploratory data analysis. Addison-Wesley Publishing Company, 1977.

[23] C. Nebl, F. Steger y H.-G. Schweiger, «Discharge Capacity of Energy Storages as a Function of the Discharge Current –Expanding Peukert's equation,» *J. Electrochem. Sci.,* 2016.

[24] W. Cleveland, «Robust Locally Weighted Regression and Smoothing Scatterplots,» *J. American Stat. Assoc.,* 1979.

[25] D. Anseán, « Operando lithium plating quantification and early detection of a commercial LiFePO4 cell cycles under dynamic driving schedule,» *J. Power Sources,* 2017.

[26] M. Kuhn y K. Johnson, Applied Predictive Modeling. Springer-Verlag New York, 2013.

[27] T. Huijskens, «Mutual information-based feature selection,» 2017. [En línea]. Available: https://thuijskens.github.io/2017/10/07/feature-selection/.

[28] D. C. Monstgomery, Introduction to linear regression analysis. Wiley,1982.

[29] D. C. Montgomery, ref 28.

[30] S. Chatterjee y J. S. Simonoff, Handbook of Regression Analysis. Wiley, 2013.

[31] A. Hoerl y R. Kennard, «Ridge regression: Biased estimation for nonorthogonal problems,» *American Stat. Assoc.*, 1970.

[32] R. Tibshirani, «Regression shrinkage and selection via the lasso» *J. Royal Stat. Soc.*, 1996.

[33] W. Fu, «Penalized regressions: the bridge versus the lasso,» *J. Comp. and Graph. Stat.*, 1998.

[34] H. Zou y T. Hastie, «Regularization and Variable Selection via the Elastic Net,» *J. Royal Stat.*

Soc., 2005.

[35] J. Salcedo, *Machine Learning for Data Mining*, Packt Publishing, 2009.

[36] C. M. Ch., «Modelos de regresión lineal con redes neuronales,» *Anales Científicos*, 2014.

[37] K. Hartmann, J. Krois y B. Waske, «E-Learning Project SOGA: Statistics and Geospatial Data Analysis.,» *Freie Universitaet Berlin*, 2018.

[38] S. Chatterjee y J. S. Simonoff, ref 30.

[39] Python S. Fundation, «Python,» 2020. [En línea]. Available: https://www.python.org/doc/essays/blurb/.

[40] W. McKinney, Python for Data Analysis. O'Reilly, 2017.

[41] Python S. Fundation, «Python Documentation,» 2020. [En línea]. Available: https://docs.python.org/3/library/pickle.html.

[42] Joblib developers, «Joblib Documentation,» 2018. [En línea]. Available: https://joblib.readthedocs.io/en/latest/.

[43] Google Developers, «Machine Learning Glossary,» 2020. [En línea]. Available: https://developers.google.com/machine-learning/glossary?hl=es-419.

ANEXO

Batch1.py

Extrae los datos del primer lote del fichero '2017-05-12_batchdata_updated_struct_errorcorrect.mat' de Matlab y los guarda en formato joblib.

```
import h5py
import numpy as np
import joblib

matFilename = '../Data/2017-05-12_batchdata_updated_struct_errorcorrect.mat'
f = h5py.File(matFilename)

batch = f['batch']

num_cells = batch['summary'].shape[0]
bat_dict = {}
for i in range(num_cells):
    cl = f[batch['cycle_life'][i,0]][()]
    policy = f[batch['policy_readable'][i,0]][()].tobytes()[::2].decode()
    summary_IR = np.hstack(f[batch['summary'][i,0]]['IR'][0,:].tolist())
    summary_QC = np.hstack(f[batch['summary'][i,0]]['QCharge'][0,:].tolist())
    summary_QD =
np.hstack(f[batch['summary'][i,0]]['QDischarge'][0,:].tolist())
    summary_TA = np.hstack(f[batch['summary'][i,0]]['Tavg'][0,:].tolist())
    summary_TM = np.hstack(f[batch['summary'][i,0]]['Tmin'][0,:].tolist())
    summary_TX = np.hstack(f[batch['summary'][i,0]]['Tmax'][0,:].tolist())
    summary_CT =
np.hstack(f[batch['summary'][i,0]]['chargetime'][0,:].tolist())
    summary_CY = np.hstack(f[batch['summary'][i,0]]['cycle'][0,:].tolist())
    summary = {'IR': summary_IR, 'QC': summary_QC, 'QD': summary_QD, 'Tavg':
               summary_TA, 'Tmin': summary_TM, 'Tmax': summary_TX,
'chargetime': summary_CT,
               'cycle': summary_CY}
    cycles = f[batch['cycles'][i,0]]
    cycle_dict = {}
    for j in range(cycles['I'].shape[0]):
        I = np.hstack((f[cycles['I'][j,0]][()]))
        Qc = np.hstack((f[cycles['Qc'][j,0]][()]))
        Qd = np.hstack((f[cycles['Qd'][j,0]][()]))
        Qdlin = np.hstack((f[cycles['Qdlin'][j,0]][()]))
        T = np.hstack((f[cycles['T'][j,0]][()]))
        Tdlin = np.hstack((f[cycles['Tdlin'][j,0]][()]))
        V = np.hstack((f[cycles['V'][j,0]][()]))
        dQdV = np.hstack((f[cycles['discharge_dQdV'][j,0]][()]))
        t = np.hstack((f[cycles['t'][j,0]][()]))
        cd = {'I': I, 'Qc': Qc, 'Qd': Qd, 'Qdlin': Qdlin, 'T': T, 'Tdlin':
Tdlin, 'V':V, 'dQdV': dQdV, 't':t}
        cycle_dict[str(j)] = cd
```

```
    cell_dict = {'cycle_life': cl, 'charge_policy':policy, 'summary':
summary, 'cycles': cycle_dict}
    key = 'b1c' + str(i)
    bat_dict[key]=   cell_dict

joblib.dump(bat_dict, '../Data/batch1.joblib')
```

Batch2.py

Extrae los datos del segundo lote del fichero '2017-06-30_batchdata_updated_struct_errorcorrect.mat' de Matlab y los guarda en formato joblib.

```
import h5py
import numpy as np
import joblib

matFilename = '../Data/2017-06-30_batchdata_updated_struct_errorcorrect.mat'
f = h5py.File(matFilename)

batch = f['batch']

num_cells = batch['summary'].shape[0]
bat_dict = {}
for i in range(num_cells):
    cl = f[batch['cycle_life'][i,0]][()]
    policy = f[batch['policy_readable'][i,0]][()].tobytes()[::2].decode()
    summary_IR = np.hstack(f[batch['summary'][i,0]]['IR'][0,:].tolist())
    summary_QC = np.hstack(f[batch['summary'][i,0]]['QCharge'][0,:].tolist())
    summary_QD =
np.hstack(f[batch['summary'][i,0]]['QDischarge'][0,:].tolist())
    summary_TA = np.hstack(f[batch['summary'][i,0]]['Tavg'][0,:].tolist())
    summary_TM = np.hstack(f[batch['summary'][i,0]]['Tmin'][0,:].tolist())
    summary_TX = np.hstack(f[batch['summary'][i,0]]['Tmax'][0,:].tolist())
    summary_CT =
np.hstack(f[batch['summary'][i,0]]['chargetime'][0,:].tolist())
    summary_CY = np.hstack(f[batch['summary'][i,0]]['cycle'][0,:].tolist())
    summary = {'IR': summary_IR, 'QC': summary_QC, 'QD': summary_QD, 'Tavg':
               summary_TA, 'Tmin': summary_TM, 'Tmax': summary_TX,
'chargetime': summary_CT,
               'cycle': summary_CY}
    cycles = f[batch['cycles'][i,0]]
    cycle_dict = {}
    for j in range(cycles['I'].shape[0]):
        I = np.hstack((f[cycles['I'][j,0]][()]))
        Qc = np.hstack((f[cycles['Qc'][j,0]][()]))
        Qd = np.hstack((f[cycles['Qd'][j,0]][()]))
        Qdlin = np.hstack((f[cycles['Qdlin'][j,0]][()]))
        T = np.hstack((f[cycles['T'][j,0]][()]))
        Tdlin = np.hstack((f[cycles['Tdlin'][j,0]][()]))
        V = np.hstack((f[cycles['V'][j,0]][()]))
        dQdV = np.hstack((f[cycles['discharge_dQdV'][j,0]][()]))
        t = np.hstack((f[cycles['t'][j,0]][()]))
        cd = {'I': I, 'Qc': Qc, 'Qd': Qd, 'Qdlin': Qdlin, 'T': T, 'Tdlin':
Tdlin, 'V':V, 'dQdV': dQdV, 't':t}
        cycle_dict[str(j)] = cd
```

```
    cell_dict = {'cycle_life': cl, 'charge_policy':policy, 'summary':
summary, 'cycles': cycle_dict}
    key = 'b2c' + str(i)
    bat_dict[key]=   cell_dict

joblib.dump(bat_dict, '../Data/batch2.joblib')
```

Batch3.py

Extrae los datos del tercer lote del fichero '2018-04-12_batchdata_updated_struct_errorcorrect.mat' de Matlab y los guarda en formato joblib.

```
import h5py
import numpy as np
import joblib

matFilename = '../Data/2018-04-12_batchdata_updated_struct_errorcorrect.mat'
f = h5py.File(matFilename)

batch = f['batch']

num_cells = batch['summary'].shape[0]
bat_dict = {}
for i in range(num_cells):
    cl = f[batch['cycle_life'][i,0]][()]
    policy = f[batch['policy_readable'][i,0]][()].tobytes()[::2].decode()
    summary_IR = np.hstack(f[batch['summary'][i,0]]['IR'][0,:].tolist())
    summary_QC = np.hstack(f[batch['summary'][i,0]]['QCharge'][0,:].tolist())
    summary_QD =
np.hstack(f[batch['summary'][i,0]]['QDischarge'][0,:].tolist())
    summary_TA = np.hstack(f[batch['summary'][i,0]]['Tavg'][0,:].tolist())
    summary_TM = np.hstack(f[batch['summary'][i,0]]['Tmin'][0,:].tolist())
    summary_TX = np.hstack(f[batch['summary'][i,0]]['Tmax'][0,:].tolist())
    summary_CT =
np.hstack(f[batch['summary'][i,0]]['chargetime'][0,:].tolist())
    summary_CY = np.hstack(f[batch['summary'][i,0]]['cycle'][0,:].tolist())
    summary = {'IR': summary_IR, 'QC': summary_QC, 'QD': summary_QD, 'Tavg':
               summary_TA, 'Tmin': summary_TM, 'Tmax': summary_TX,
'chargetime': summary_CT,
               'cycle': summary_CY}
    cycles = f[batch['cycles'][i,0]]
    cycle_dict = {}
    for j in range(cycles['I'].shape[0]):
        I = np.hstack((f[cycles['I'][j,0]][()]))
        Qc = np.hstack((f[cycles['Qc'][j,0]][()]))
        Qd = np.hstack((f[cycles['Qd'][j,0]][()]))
        Qdlin = np.hstack((f[cycles['Qdlin'][j,0]][()]))
        T = np.hstack((f[cycles['T'][j,0]][()]))
        Tdlin = np.hstack((f[cycles['Tdlin'][j,0]][()]))
        V = np.hstack((f[cycles['V'][j,0]][()]))
        dQdV = np.hstack((f[cycles['discharge_dQdV'][j,0]][()]))
        t = np.hstack((f[cycles['t'][j,0]][()]))
        cd = {'I': I, 'Qc': Qc, 'Qd': Qd, 'Qdlin': Qdlin, 'T': T, 'Tdlin':
Tdlin, 'V':V, 'dQdV': dQdV, 't':t}
        cycle_dict[str(j)] = cd

    cell_dict = {'cycle_life': cl, 'charge_policy':policy, 'summary':
summary, 'cycles': cycle_dict}
```

```
    key = 'b3c' + str(i)
    bat_dict[key]=   cell_dict

joblib.dump(bat_dict, '../Data/batch3.joblib')
```

LoadData.py

Abre los ficheros joblib, elimina las baterías defectuosas y los une en un dict de Python. Convierte el dict en un pd.DataFrame y desplaza las columnas para que empiecen todas a la vez.

```
import numpy as np
import matplotlib.pyplot as plt
import joblib
import pandas as pd
import time

def LoadData():

    batch1=joblib.load('../Data/batch1.joblib')
    #remove batteries that do not reach 80% capacity
    del batch1['b1c8']
    del batch1['b1c10']
    del batch1['b1c12']
    del batch1['b1c13']
    del batch1['b1c22']

    numBat1 = len(batch1.keys())



    batch2 = joblib.load('../Data/batch2.joblib')

    # There are four cells from batch1 that carried into batch2, we'll remove
the data from batch2
    # and put it with the correct cell from batch1
    batch2_keys = ['b2c7', 'b2c8', 'b2c9', 'b2c15', 'b2c16']
    batch1_keys = ['b1c0', 'b1c1', 'b1c2', 'b1c3', 'b1c4']
    add_len = [662, 981, 1060, 208, 482];

    for i, bk in enumerate(batch1_keys):
        batch1[bk]['cycle_life'] = batch1[bk]['cycle_life'] + add_len[i]
        for j in batch1[bk]['summary'].keys():
            if j == 'cycle':
                batch1[bk]['summary'][j] =
np.hstack((batch1[bk]['summary'][j], batch2[batch2_keys[i]]['summary'][j] +
len(batch1[bk]['summary'][j])))
            else:
                batch1[bk]['summary'][j] =
np.hstack((batch1[bk]['summary'][j], batch2[batch2_keys[i]]['summary'][j]))
        last_cycle = len(batch1[bk]['cycles'].keys())
        for j, jk in enumerate(batch2[batch2_keys[i]]['cycles'].keys()):
            batch1[bk]['cycles'][str(last_cycle + j)] =
batch2[batch2_keys[i]]['cycles'][jk]

    del batch2['b2c7']
    del batch2['b2c8']
    del batch2['b2c9']
    del batch2['b2c15']
    del batch2['b2c16']
```

```
    numBat2 = len(batch2.keys())


    batch3 = joblib.load('../Data/batch3.joblib')
    # remove noisy channels from batch3
    del batch3['b3c37']
    del batch3['b3c2']
    del batch3['b3c23']
    del batch3['b3c32']
    del batch3['b3c38']
    del batch3['b3c39']

    numBat3 = len(batch3.keys())


    numBat = numBat1 + numBat2 + numBat3

    bat_dict = {**batch1, **batch2, **batch3}

    ix=[]
    for bat in bat_dict.keys():
        if 'b1' in bat:
            for cycle in bat_dict[bat]['summary']['cycle'][:-1]:
                ix.append((bat,int(cycle)))
        else:
            for cycle in bat_dict[bat]['summary']['cycle']:
                ix.append((bat,int(cycle)))

    col_summary = [key for key in bat_dict['b1c0']['summary'].keys()][:-1]

    bat_summary_df = pd.DataFrame(index = pd.MultiIndex.from_tuples(ix,
names=('Battery', 'Cycle')),
                                  columns = col_summary)

    ix=[]
    for bat in bat_dict.keys():
        for cycle in range(1,101):
            ix.append((bat,cycle))

    bat_Qdlin_df = pd.DataFrame(index = pd.MultiIndex.from_tuples(ix,
names=('Battery', 'Cycle')),
                                columns = range(1000))

    bat_Qdlin_df.columns.name='Qdlin'

    bat_Tdlin_df = pd.DataFrame(index = pd.MultiIndex.from_tuples(ix,
names=('Battery', 'Cycle')),
                                columns = range(1000))

    bat_Tdlin_df.columns.name='Tdlin'

    for bat in bat_dict.keys():
        print('Batería: ', bat)
        for cycle in bat_summary_df.loc[bat].index:
            print('\rCiclo: ', cycle, end='\r')
            time.sleep(0.2)
            for metric in bat_summary_df.columns:
                if 'b1' in bat:
bat_summary_df.loc[bat,cycle][metric]=bat_dict[bat]['summary'][metric][cycle]
                else:
```

```
bat_summary_df.loc[bat,cycle][metric]=bat_dict[bat]['summary'][metric][cycle-
1]



for cycle in bat_Qdlin_df.loc[bat].index:
                if 'b1' in bat:

bat_Qdlin_df.loc[bat,cycle]=bat_dict[bat]['cycles'][str(cycle)]['Qdlin']

bat_Tdlin_df.loc[bat,cycle]=bat_dict[bat]['cycles'][str(cycle)]['Tdlin']
                else:

bat_Qdlin_df.loc[bat,cycle]=bat_dict[bat]['cycles'][str(cycle-1)]['Qdlin']

bat_Tdlin_df.loc[bat,cycle]=bat_dict[bat]['cycles'][str(cycle-1)]['Tdlin']
        print('\r')
```

Plot.py

Visualiza todos los datos que se van a usar y los procesa para que no haya valores extraños.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.ticker import MaxNLocator
import matplotlib as mpl
import joblib
import pandas as pd
from cycler import cycler
import statsmodels.api as sm

summary=joblib.load('../Data/bat_summary_df.joblib')
Qdlin=joblib.load('../Data/Qdlin.joblib')
Tdlin=joblib.load('../Data/Tdlin.joblib')

#Tratamiento manual de un outlier
arr=np.array(summary.loc['b1c18','IR'])
arr[:53]=arr[53]
summary.loc['b1c18','IR']=arr


#Paleta https://coolors.co/ef4c4d-2b9ed8-542838-7adc54-fcee77


#Figura_1
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary.loc[bat,'QD'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
```

```
plt.legend(bbox_to_anchor=(1.015, 1.015),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()
#Figure_2


summary_sm = summary.copy().astype(np.float)
summary_outlier = summary.copy().astype(np.float)
for bat in summary.index.get_level_values('Battery').unique():
    summary_sm.loc[bat,'QD']=sm.nonparametric.lowess(summary.loc[bat,'QD'],
summary.loc[bat,'QD'].index, frac=0.06)[:,1]
    summary_outlier.loc[bat,'QD']=np.array(list(map(lambda a, b: a if
(a<b*1.05 and a>b*0.95) else b, summary.loc[bat,'QD'],
summary_sm.loc[bat,'QD'])))

for bat in summary.index.get_level_values('Battery').unique():
    if 'b1' in bat: c='#2b9ed8'
    if 'b2' in bat: c='#ef4c4d'
    if 'b3' in bat: c='#fcee77'
    fig, ax = plt.subplots(dpi=300)
    ax.scatter(summary.loc[bat,'QD'].index,summary.loc[bat,'QD'],
marker='.',linewidth=0.0001,c=c)

ax.fill_between(summary.loc[bat,'QD'].index,summary_sm.loc[bat,'QD']*1.05,
summary_sm.loc[bat,'QD']*0.95, alpha=0.2, facecolor=c)
    plt.xlabel('Ciclo')
    plt.ylabel('Capacidad de descarga (Ah)')
    plt.show()


#Grafica suavizada para mejor visualización
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary_sm.loc[bat,'QD'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1.015, 1.015),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()

summary=summary[summary_sm.loc[:,'QD']>0.88]
summary_outlier=summary_outlier[summary_sm.loc[:,'QD']>0.88]
summary_sm=summary_sm[summary_sm.loc[:,'QD']>0.88]


#Figure_3
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary_sm.loc[bat,'QD'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
```

```
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1.015, 1.015),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()

#Figure_4
cycles=[]
for bat in summary_sm.index.get_level_values('Battery').unique():
    cycles.append(summary_sm.loc[bat,'QD'].index[-1])

cycles_series=pd.Series(cycles,
index=summary_sm.index.get_level_values('Battery').unique())

colormap = plt.cm.Blues
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values(ascending=False).index:
    plt.plot(summary_sm.loc[bat,'QD'])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()

#Figure_5
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.2, 0.65, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values().index:
    plt.plot(summary_sm.loc[bat,'QD'][:100])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()

#Figure_6
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.2, 0.65, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values().index[1:]:
    plt.plot(summary_sm.loc[bat,'QD'][:100])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()

#IR

#Figura_7
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary.loc[bat,'IR'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1.015, 0.30),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
```

```
plt.ylabel('IR (Ω)')
plt.show()


#Outlier
for bat in summary.index.get_level_values('Battery').unique():
    summary_sm.loc[bat,'IR']=sm.nonparametric.lowess(summary.loc[bat,'IR'],
summary.loc[bat,'IR'].index, frac=0.06)[:,1]
    summary_outlier.loc[bat,'IR']=np.array(list(map(lambda a, b: a if
(a<b*1.05 and a>b*0.95) else b, summary.loc[bat,'IR'],
summary_sm.loc[bat,'IR'])))


for bat in summary.index.get_level_values('Battery').unique():
    if 'b1' in bat: c='#2b9ed8'
    if 'b2' in bat: c='#ef4c4d'
    if 'b3' in bat: c='#fcee77'
    fig, ax = plt.subplots(dpi=300)
    ax.scatter(summary.loc[bat,'IR'].index,summary.loc[bat,'IR'],
marker='.',linewidth=0.0001,c=c)

ax.fill_between(summary.loc[bat,'IR'].index,summary_sm.loc[bat,'IR']*1.05,
summary_sm.loc[bat,'IR']*0.95, alpha=0.2, facecolor=c)
    plt.xlabel('Ciclo')
    plt.ylabel('IR (Ω)')
    plt.show()


#Figure_8
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary_outlier.loc[bat,'IR'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1.015, 0.30),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('IR (Ω)')
plt.show()

#Figure_9
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary_outlier.loc[bat,'IR'][:100])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1.015, 0.30),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('IR (Ω)')
plt.show()

#Figure_10
colormap = plt.cm.Blues
```

```
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values(ascending=False).index:
    plt.plot(summary_outlier.loc[bat,'IR'])
plt.xlabel('Ciclo')
plt.ylabel('Capacidad de descarga (Ah)')
plt.show()

#Figure_11
colormap = plt.cm.Blues
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values(ascending=False).index:
    plt.plot(summary_outlier.loc[bat,'IR'][:100])
plt.xlabel('Ciclo')
plt.ylabel('IR (Ω)')
plt.show()

#chargetime

#Figure_12
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary.loc[bat,'chargetime'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(0.165, 1.015),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('Tiempo de carga (s)')
plt.show()


#Outlier
for bat in summary.index.get_level_values('Battery').unique():

summary_sm.loc[bat,'chargetime']=np.concatenate((sm.nonparametric.lowess(summ
ary.loc[bat,'chargetime'], summary.loc[bat,'chargetime'].index, frac=0.06)[:-
10,1],np.array(summary.loc[bat,'chargetime'][-10:])))
    summary_outlier.loc[bat,'chargetime']=np.array(list(map(lambda a, b: a if
(a<b*1.05 and a>b*0.95) else b, summary.loc[bat,'chargetime'],
summary_sm.loc[bat,'chargetime'])))


for bat in summary.index.get_level_values('Battery').unique():
    if 'b1' in bat: c='#2b9ed8'
    if 'b2' in bat: c='#ef4c4d'
    if 'b3' in bat: c='#fcee77'
    fig, ax = plt.subplots(dpi=300)

ax.scatter(summary.loc[bat,'chargetime'].index,summary.loc[bat,'chargetime'],
marker='.',linewidth=0.0001,c=c)
```

```
ax.fill_between(summary.loc[bat,'chargetime'].index,summary_sm.loc[bat,'charg
etime']*1.05, summary_sm.loc[bat,'chargetime']*0.95, alpha=0.2, facecolor=c)
    plt.xlabel('Ciclo')
    plt.ylabel('Tiempo de carga (s)')
    plt.show()
#Figure_14
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary_outlier.loc[bat,'chargetime'][:100])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1.015, 1.015),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('Tiempo de carga (s)')
plt.show()

#Figure_15
colormap = plt.cm.Blues
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values(ascending=False).index:
    plt.plot(summary_outlier.loc[bat,'chargetime'][:100])
plt.xlabel('Ciclo')
plt.ylabel('Tiempo de carga (s)')
plt.show()

#Temp Max
plt.figure(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in summary.index.get_level_values('Battery').unique():
    line, = plt.plot(summary.loc[bat,'Tmax'])
    if 'b1' in bat: line.set_color('#2b9ed8')
    if 'b2' in bat: line.set_color('#ef4c4d')
    if 'b3' in bat: line.set_color('#fcee77')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1, 1.015),handles=[b1, b2, b3])
plt.xlabel('Ciclo')
plt.ylabel('Temperatura máxima (°C)')
plt.show()

colormap = plt.cm.Blues
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, 124)])))
fig, ax = plt.subplots(dpi=300)
mpl.style.use('seaborn-darkgrid')
for bat in cycles_series.sort_values(ascending=False).index:
    plt.plot(summary_outlier.loc[bat,'Tmax'][:100])
plt.xlabel('Ciclo')
plt.ylabel('Temperatura máxima (°C)')
plt.show()
```

```
##########################QDLIN#################################
colormap = plt.cm.Blues
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, len(Qdlin.loc['b3c45'].index))])))
fig, ax = plt.subplots(dpi=300)
for cycle in Qdlin.loc['b3c45'].index:
    plt.plot(Qdlin.loc['b3c45',cycle],np.arange(3.5,2,-0.0015))
plt.ylabel('Voltaje (V)')
plt.xlabel('Capacidad de descarga (Ah)')
plt.show()

plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.4, 2)])))
fig, ax = plt.subplots(dpi=300)
for cycle in [2,100]:
    plt.plot(Qdlin.loc['b3c45',cycle],np.arange(3.5,2,-0.0015))
plt.ylabel('Voltaje (V)')
plt.xlabel('Capacidad de descarga (Ah)')
plt.show()

##########################TDLIN#################################

colormap = plt.cm.Blues
plt.rc('axes', prop_cycle=(cycler('color',[colormap(i) for i in
np.linspace(0.65, 0.2, len(Tdlin.loc['b3c45'].index))])))
fig, ax = plt.subplots(dpi=300)
for cycle in Tdlin.loc['b3c45'].index:
    plt.plot(Tdlin.loc['b3c45',cycle],np.arange(3.5,2,-0.0015))
plt.ylabel('Voltaje (V)')
plt.xlabel('Temperatura (°C)')
plt.show()

##########################Cycle Lyfe#############################

Y = pd.DataFrame(features_df['Cycle Life'].copy())
Y['batch']=1
Y['batch'][['b2' in index for index in Y.index]]=2
Y['batch'][['b3' in index for index in Y.index]]=3
train =Y.copy().iloc[train_ind]
test = Y.copy().iloc[test_ind]
secondary_test = Y.copy().iloc[secondary_test_ind]

fig, ax = plt.subplots(nrows=1, ncols=3, dpi=500, sharey='row', sharex='row',
figsize=(10,3))
fig.subplots_adjust(wspace = 0.05)
ax[0].set_title('Train')
train[train['batch']==2]['Cycle Life'].hist(ax=ax[0],
color='#ef4c4d',range=[Y['Cycle Life'].min(), Y['Cycle Life'].max()],
bins=12)
train[train['batch']==1]['Cycle Life'].hist(ax=ax[0],
color='#2b9ed8',range=[Y['Cycle Life'].min(), Y['Cycle Life'].max()],
bins=12,alpha=0.9)
ax[1].set_title('Test')
test[test['batch']==2]['Cycle Life'].hist(ax=ax[1],
color='#ef4c4d',range=[Y['Cycle Life'].min(), Y['Cycle Life'].max()],
bins=12)
test[test['batch']==1]['Cycle Life'].hist(ax=ax[1],
color='#2b9ed8',range=[Y['Cycle Life'].min(), Y['Cycle Life'].max()],
bins=12, alpha=0.9)
ax[2].set_title('Secondary Test')
secondary_test['Cycle Life'].hist(ax=ax[2], color='#fcee77',range=[Y['Cycle
Life'].min(), Y['Cycle Life'].max()], bins=12)
```

```
ax[0].yaxis.set_major_locator(MaxNLocator(integer=True))
ax[0].set_ylabel('Número de baterías')
ax[1].set_xlabel('Ciclos')
b1 = mpatches.Patch(color='#2b9ed8', label='b1')
b2 = mpatches.Patch(color='#ef4c4d', label='b2')
b3 = mpatches.Patch(color='#fcee77', label='b3')
plt.legend(bbox_to_anchor=(1, 1),handles=[b1, b2, b3])

########################Model#################################

col=Ytest.copy()

col[['b1' in index for index in col.index]]='#2b9ed8'

col[['b2' in index for index in col.index]]='#ef4c4d'

col2=Ytest2.copy()

col2[['b3' in index for index in col2.index]]='#fcee77'

fig, axes = plt.subplots(nrows=2, ncols=3, dpi=500, sharey='row')
fig.subplots_adjust(hspace=0.1,wspace = 0.14)
i=0
for ax, test in zip(axes.flatten(), [[Ypred_regr, Ytest],[Ypred_EN,
Ytest],[Ypred_sgd, Ytest],[Ypred2_regr, Ytest2],[Ypred2_EN,
Ytest2],[Ypred2_sgd, Ytest2]]):
    if i==3:
        col=col2
    ax.plot(test[1],test[1],color='gray',zorder=1)
    ax.scatter(test[1], test[0], marker='.', color=col,zorder=2)
    ax.set_yticklabels([])
    ax.set_xticklabels([])
    if i==0:
        ax.set_ylabel('Test')
        ax.set_title('OLS')
    if i==1:
        ax.set_title('Elastic Net')
    if i==2:
        ax.set_title('MLP')
    if i==3:
        ax.set_ylabel('Secondary Test')
    i+=1
```

Load_and_split.py

Separa los datos en tres conjuntos: entrenamiento, primer y segundo test.

```
import numpy as np
import joblib

def test_ind(numBat = 124, numBat1 = 41, numBat2 = 43, numBat3 = 40):
    test_ind = np.hstack((np.arange(0,(numBat1+numBat2),2),83))
    train_ind = np.arange(1,(numBat1+numBat2-1),2)
    secondary_test_ind = np.arange(numBat-numBat3,numBat)
    return test_ind, train_ind, secondary_test_ind
```

PrepareFeatures.py

Crea las variables de entrada a los modelos, tanto de regresión como de clasificación. En el caso de regresión, hace el estudio de correraciones, lo visualiza y elimina las características con colinealidad. Selecciona las k mejores.

```
import joblib
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from scipy.stats import pearsonr
import numpy as np
import pandas as pd
import similaritymeasures
from LoadData import test_ind
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
import sys
import string
from sklearn.feature_selection import SelectKBest


epsilon=sys.float_info.epsilon

summary_outlier = joblib.load('../Data/summary_outlier.joblib')
Qdlin = joblib.load('../Data/Qdlin.joblib')
Tdlin = joblib.load('../Data/Tdlin.joblib')

def PrepareFeatures(summary=summary_outlier,Qdlin=Qdlin,mode='regr'):

    n_cells = 124

    cycle_life = np.zeros(n_cells)
    life_class = []

    #Atributos (Features)
    # 1. ΔQ(V)
    min_Qd_100_10 = np.zeros(n_cells)
    var_Qd_100_10 = np.zeros(n_cells)
    min_Qd_5_4 = np.zeros(n_cells)
    var_Qd_5_4 = np.zeros(n_cells)
    areaQd = np.zeros(n_cells)

    min_Td_100_10 = np.zeros(n_cells)
    var_Td_100_10 = np.zeros(n_cells)
    min_Td_5_4 = np.zeros(n_cells)
    var_Td_5_4 = np.zeros(n_cells)
    areaTd = np.zeros(n_cells)
    Var1 = np.zeros(n_cells)
    Var2 = np.zeros(n_cells)
    # 2. Curva de capacidad de descarga
    slope_2_100 = np.zeros(n_cells) #Pendiente del ajuste lineal a la curva
    intercept_2_100 = np.zeros(n_cells)  #Intersección
    Qd_2 = np.zeros(n_cells)
    Qd_100 = np.zeros(n_cells)

    # 3. Otros Atributos
    Avg_t_2_6 = np.zeros(n_cells)
```

```
    Avg_t_2_5 = np.zeros(n_cells)
    Max_T = np.zeros(n_cells)
    Sum_T_2_5 = np.zeros(n_cells)
    min_IR_2_100 = np.zeros(n_cells)
    diff_IR_100_2 = np.zeros(n_cells)
    slope_IR = np.zeros(n_cells)
    slope_CT = np.zeros(n_cells)
    var_IR = np.zeros(n_cells)
    var_CT = np.zeros(n_cells)

    index=summary.index.get_level_values('Battery').unique()

    for i, bat in enumerate(index):
        cycle_life[i] = summary.loc[bat].index[-1]

        if mode == 'regr':
            # 1. ΔQ100-10(V)
            Qd_100_10 = Qdlin.loc[bat,100] - Qdlin.loc[bat,10]
            min_Qd_100_10[i] = np.log(np.abs(np.min(Qd_100_10)))
            var_Qd_100_10[i] = np.log(np.var(Qd_100_10))
            areaQd[i] =
np.log(similaritymeasures.area_between_two_curves(np.array([np.array(Qdlin.lo
c[bat,100]),np.array(Qdlin.loc[bat,100].index)]).T,
np.array([np.array(Qdlin.loc[bat,10]),np.array(Qdlin.loc[bat,10].index)]).T))

            Td_100_10 = np.mean(Tdlin.loc[bat][95:100]) -
np.mean(Tdlin.loc[bat][5:10])
            min_Td_100_10[i] = np.log(np.abs(np.min(Td_100_10)))
            var_Td_100_10[i] = np.log(np.var(Td_100_10))
            areaTd[i] =
np.log(similaritymeasures.area_between_two_curves(np.array([np.array(Tdlin.lo
c[bat,100]),np.array(Tdlin.loc[bat,100].index)]).T,
np.array([np.array(Tdlin.loc[bat,10]),np.array(Tdlin.loc[bat,10].index)]).T))


            Var1[i] = np.log(np.max(np.var(Tdlin.loc[bat][:100].T)))
            Var2[i] = np.log(np.var(np.var(Tdlin.loc[bat][:100].T)))
            # 2. Curva de capacidad de descarga
            QD = summary.loc[bat,'QD'][:100]
            QDix = np.array(QD.index,dtype=np.int8).reshape(-1,1)
            lin_reg_2_100 = LinearRegression()
            lin_reg_2_100.fit(QDix, QD)

            slope_2_100[i] = np.log(np.abs(lin_reg_2_100.coef_[0])+epsilon)
            intercept_2_100[i] = np.log(lin_reg_2_100.intercept_)
            Qd_2[i] = np.log(QD[2])
            Qd_100[i] = np.log(QD[100])

            # 3. Otros Atributos
            Avg_t_2_6[i] = np.mean(summary.loc[bat,'chargetime'][2:6])
            min_IR_2_100[i] = np.log(np.min(summary.loc[bat,'IR'][:100]))
            diff_IR_100_2[i] = np.log(np.abs(summary.loc[bat,'IR'][100] -
summary.loc[bat,'IR'][2]))
            Max_T[i] = np.log(np.max(summary.loc[bat,'Tmax'][:100]))
            CT = summary.loc[bat,'chargetime'][:100]
            CTix = np.array(CT.index,dtype=np.int8).reshape(-1,1)
            lin_reg_2_100 = LinearRegression()
            lin_reg_2_100.fit(CTix, CT)
            var_CT[i]=np.log(np.var(CT)+epsilon)
            slope_CT[i] = np.log(np.abs(lin_reg_2_100.coef_[0])+epsilon)
            IR = summary.loc[bat,'IR'][:100]
            IRix = np.array(IR.index,dtype=np.int8).reshape(-1,1)
```

```
            lin_reg_2_100 = LinearRegression()
            lin_reg_2_100.fit(IRix, IR)
            var_IR[i] = np.log(np.var(IR))
            slope_IR[i] = np.log(np.abs(lin_reg_2_100.coef_[0])+epsilon)

        elif mode == 'class':
            #Clasificación
            Qd_5_4 = Qdlin.loc[bat,5] - Qdlin.loc[bat,4]

            min_Qd_5_4[i] = np.log10(np.abs(np.min(Qd_5_4)))
            var_Qd_5_4[i] = np.log10(np.var(Qd_5_4))
            Td_5_4 = Tdlin.loc[bat,5] - Tdlin.loc[bat,4]
            min_Td_5_4[i] = np.log(np.abs(np.min(Td_5_4)))
            var_Td_5_4[i] = np.log(np.var(Td_5_4))
            Avg_t_2_5[i] = np.mean(summary.loc[bat,'chargetime'][2:5])
            Qd_2[i] = np.log(summary.loc[bat,'QD'][2])
            var_IR[i] = np.log(np.var(summary.loc[bat,'IR'][:5])+epsilon)
            Max_T[i] = np.log(np.max(summary.loc[bat,'Tmax'][:5]))

            life_class.append('high' if cycle_life[i]>550 else 'low')

    if mode=='regr':
        features_df = pd.DataFrame({
        #Discharge/Voltage
        "a) ΔQ100-10(V) min": min_Qd_100_10, #a
        "b) ΔQ100-10(V) var": var_Qd_100_10, #b
        'c) ΔQ100-10(V) Área': areaQd, #c
        #Temp
        "d) ΔT100-10(V) min": min_Td_100_10, #d
        "e) ΔT100-10(V) var": var_Td_100_10, #e
        'f) ΔT100-10(V) Área': areaTd, #f
        'g) Var1': Var1, #g
        'h) Var2': Var2, #h
        "i) Max Temp": Max_T, #i
        #Discharge/Cicles
        "j) Slope Qd(2-100)": slope_2_100, #j
        "k) Intercept Slope-Curve (2-100)": intercept_2_100, #k
        "l) Qd 2": Qd_2, #l
        "m) Qd 100": Qd_100, #m
        #Charge Time
        "n) Mean chargetime 2-6": Avg_t_2_6, #n
        'o) Slope CT': slope_CT, #o
        "p) Var CT": var_CT, #p
        #Internal Resistance
        "q) Min IR 2-100": min_IR_2_100, #q
        "r) ΔIR 2-100": diff_IR_100_2, #r
        "s) Slope IR": slope_IR, #s
        "t) Var IR": var_IR, #t
        #Target
        "Cycle Life": cycle_life,
        },index=index)
    elif mode=='class':
        features_df = pd.DataFrame({
        "ΔQ5-4(V) min": min_Qd_5_4,
        "ΔQ5-4(V) var": var_Qd_5_4,
        "ΔT5-4(V) min":min_Td_5_4,
        "ΔT5-4(V) var":var_Td_5_4,
        "Max Temp": Max_T,
        "Mean chargetime 2-5": Avg_t_2_5,
        "Qd 2": Qd_2,
        "Var IR": var_IR,
        "Cycle Life": life_class,
```

```
        },index=index)

    return features_df

features_df=PrepareFeatures()


scaler = MinMaxScaler()

test_ind, train_ind, secondary_test_ind = test_ind()

train =features_df.copy().iloc[train_ind]
test = features_df.copy().iloc[test_ind]
secondary_test = features_df.copy().iloc[secondary_test_ind]

scaler.fit(train.iloc[:,:-1])

Xtrain = pd.DataFrame(scaler.transform(train.iloc[:,:-
1]),index=train.index,columns=train.columns[:-1])
Ytrain = train.loc[:,'Cycle Life']

Xtest = pd.DataFrame(scaler.transform(test.iloc[:,:-
1]),index=test.index,columns=test.columns[:-1])
Ytest = test.loc[:,'Cycle Life']

Xtest2 = pd.DataFrame(scaler.transform(secondary_test.iloc[:,:-
1]),index=secondary_test.index,columns=secondary_test.columns[:-1])
Ytest2 = secondary_test.loc[:,'Cycle Life']

mpl.style.use('seaborn-darkgrid')
fig, axes = plt.subplots(nrows=5, ncols=4,
dpi=500,figsize=(12,16),sharex='col', sharey='row')
fig.subplots_adjust(hspace=0.1,wspace = 0.14)

for ax, feature, letter in zip(axes.flatten(),
Xtrain.columns,string.ascii_lowercase):
    ax.scatter(Xtrain[feature], Ytrain, marker='.', color='#2b9ed8')
    ax.set_ylabel(letter, rotation='horizontal', weight='bold')
    ax.yaxis.set_label_coords(-0.05,0.93)
    corr=np.corrcoef(Xtrain[feature],Ytrain)[0,1]
    ax.text(0.5, 1700, 'p = %.4f' % corr, ha='center')

#Colinealidad
corr = Xtrain.corr()

mask = np.triu(np.ones_like(corr, dtype=np.bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9),dpi=300)

# Generate a custom diverging colormap
cmap = sns.diverging_palette(10, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, vmin=-1, center=0,
            square=True,  cbar=False, linewidths=.2, annot=True,
fmt=".2f",annot_kws={"fontsize":8},
            xticklabels=string.ascii_lowercase[:20],
yticklabels=string.ascii_lowercase[:20])

Xtrain_full, Xtest_full, Xtest2_full = Xtrain.copy(), Xtest.copy(),
Xtest2.copy()
```

```
Xtrain.drop(["a) ΔQ100-10(V) min",'c) ΔQ100-10(V) Área','h) Var2' ,'f) ΔT100-
10(V) Área',"i) Max Temp","k) Intercept Slope-Curve (2-100)", 'm) Qd 100',
'r) ΔIR 2-100'],axis=1,inplace=True)

Xtest = Xtest[Xtrain.columns]

Xtest2= Xtest2[Xtrain.columns]

# Proceso iterativo

# for k in range(2,13):
#     SelectK= SelectKBest(pcorr, k=k)
#     SelectK.fit(Xtrain, Ytrain)

#     Xtrain_kbest = pd.DataFrame(SelectK.transform(Xtrain),
index=Xtrain.index, columns=Xtrain.columns[SelectK.get_support()])

#     Xtest_kbest = pd.DataFrame(SelectK.transform(Xtest), index=Xtest.index,
columns=Xtrain.columns[SelectK.get_support()])

#     Xtest2_kbest = pd.DataFrame(SelectK.transform(Xtest2),
index=Xtest2.index, columns=Xtrain.columns[SelectK.get_support()])
#     print('k = ',k)

#     Errors, Ypred_regr, Ypred2_regr, Ypred_EN, Ypred2_EN, Ypred_sgd,
Ypred2_sgd
=TrainRegreModel(Xtrain=Xtrain_kbest,Ytrain=Ytrain,Xtest=Xtest_kbest,Ytest=Yt
est,Xtest2=Xtest2_kbest,Ytest2=Ytest2)
#     Error_k=pd.concat([Error_k,Errors])
def pcorr(X,Y):
    scores, pvalues = [], []
    for column in range(X.shape[1]):
        cur_score, cur_p = pearsonr(X[:,column],Y)
        scores.append(abs(cur_score))
        pvalues.append(cur_p)
    return (np.array(scores),np.array(pvalues))

SelectK= SelectKBest(pcorr, k=9)
SelectK.fit(Xtrain, Ytrain)

Xtrain_kbest = pd.DataFrame(SelectK.transform(Xtrain), index=Xtrain.index,
columns=Xtrain.columns[SelectK.get_support()])

Xtest_kbest = pd.DataFrame(SelectK.transform(Xtest), index=Xtest.index,
columns=Xtrain.columns[SelectK.get_support()])

Xtest2_kbest = pd.DataFrame(SelectK.transform(Xtest2), index=Xtest2.index,
columns=Xtrain.columns[SelectK.get_support()])
```

Model.py

Crea y entrena los modelos de regresión y clasificación.

```
from sklearn.linear_model import
ElasticNet,LinearRegression,LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPRegressor
import numpy as np
import pandas as pd
```

```
from sklearn.metrics import mean_squared_error, accuracy_score,
confusion_matrix, classification_report, log_loss


def mpe(test,pred):
   mpe=sum(abs(test-pred)/test)/pred.size*100
   return mpe

def
TrainRegreModel(Xtrain=Xtrain_kbest,Ytrain=Ytrain,Xtest=Xtest_kbest,Ytest=Yte
st,Xtest2=Xtest2_kbest,Ytest2=Ytest2):
    linspace = np.linspace(0.05,1,20)

    #OLS
    regr = LinearRegression()
    regr.fit(Xtrain, Ytrain)
    Ytrainpred_regr = regr.predict(Xtrain)
    Ypred_regr = regr.predict(Xtest)
    Ypred2_regr = regr.predict(Xtest2)

    regrtrain_mpe=mpe(Ytrain,Ytrainpred_regr)
    regr_mpe=mpe(Ytest,Ypred_regr)
    regr2_mpe=mpe(Ytest2,Ypred2_regr)
    regrtrain_rmse=np.sqrt(mean_squared_error(Ytrain,Ytrainpred_regr))
    regr_rmse=np.sqrt(mean_squared_error(Ytest,Ypred_regr))
    regr2_rmse=np.sqrt(mean_squared_error(Ytest2,Ypred2_regr))

    #Elastic Net
    parameters = {
        "alpha": linspace,
        "l1_ratio": linspace,
        'max_iter':[200000]
    }

    ElasNet = ElasticNet()
    ElasNet = GridSearchCV(ElasNet, parameters, cv=4)
    ElasNet.fit(Xtrain, Ytrain)
    Ytrainpred_EN = ElasNet.predict(Xtrain)
    Ypred_EN = ElasNet.predict(Xtest)
    Ypred2_EN = ElasNet.predict(Xtest2)

    ElasNettrain_mpe=mpe(Ytrain,Ytrainpred_EN)
    ElasNet_mpe=mpe(Ytest,Ypred_EN)
    ElasNet2_mpe=mpe(Ytest2,Ypred2_EN)
    ElasNettrain_rmse=np.sqrt(mean_squared_error(Ytrain,Ytrainpred_EN))
    ElasNet_rmse=np.sqrt(mean_squared_error(Ytest,Ypred_EN))
    ElasNet2_rmse=np.sqrt(mean_squared_error(Ytest2,Ypred2_EN))

    #MLP
    parameters = {
        'hidden_layer_sizes':[(3,)],
        "alpha": np.linspace(25,35,3),
        'max_iter': [200000]
        }

    sgd = MLPRegressor()
    sgd = GridSearchCV(sgd, parameters, cv=4, verbose=True, n_jobs=-1)
    sgd.fit(Xtrain,Ytrain)

    Ytrainpred_sgd = sgd.predict(Xtrain)
    Ypred_sgd = sgd.predict(Xtest)
    Ypred2_sgd = sgd.predict(Xtest2)
```

```
    sgdtrain_mpe=mpe(Ytrain,Ytrainpred_sgd)
    sgd_mpe=mpe(Ytest,Ypred_sgd)
    sgd2_mpe=mpe(Ytest2,Ypred2_sgd)
    sgdtrain_rmse=np.sqrt(mean_squared_error(Ytrain,Ytrainpred_sgd))
    sgd_rmse=np.sqrt(mean_squared_error(Ytest,Ypred_sgd))
    sgd2_rmse=np.sqrt(mean_squared_error(Ytest2,Ypred2_sgd))

    #Tabla de errores de cada modelo
    columns = [["Mean Percentage Error"]*3+["Root Mean Squared Error"]*3,
               ['OLS', 'Elastic Net', 'MLP']*2]

Errors=pd.DataFrame([[regrtrain_mpe,ElasNettrain_mpe,sgdtrain_mpe,regrtrain_r
mse,ElasNettrain_rmse,sgdtrain_rmse],

[regr_mpe,ElasNet_mpe,sgd_mpe,regr_rmse,ElasNet_rmse,sgd_rmse],

[regr2_mpe,ElasNet2_mpe,sgd2_mpe,regr2_rmse,ElasNet2_rmse,sgd2_rmse]],
                         index=['Train','Test','Secondary
Test'],columns=pd.MultiIndex.from_arrays(columns))

    with pd.option_context('display.max_rows', None, 'display.max_columns',
None):
        print(Errors)

    return Errors, Ypred_regr, Ypred2_regr, Ypred_EN, Ypred2_EN, Ypred_sgd,
Ypred2_sgd

Errors, Ypred_regr, Ypred2_regr, Ypred_EN, Ypred2_EN, Ypred_sgd, Ypred2_sgd
=TrainRegreModel()

def
TrainClasModel(Xtrain=Xtrain,Ytrain=Ytrain,Xtest=Xtest,Ytest=Ytest,Xtest2=Xte
st2,Ytest2=Ytest2):
    linspace = np.linspace(0.05,1,20)
    parameters = {"C": linspace,
                  "penalty": ['l1','l2'],
                  "solver": ['liblinear'],
                  "max_iter": [200000],
                  "class_weight": ['balanced']
                 }

    logreg = LogisticRegression()
    clas = GridSearchCV(logreg, parameters, cv=4)

    clas.fit(Xtrain,Ytrain)

    Ytrainpred = clas.predict(Xtrain)
    Ytestpred = clas.predict(Xtest)
    Ytest2pred = clas.predict(Xtest2)

    Ytrainprob = clas.predict_proba(Xtrain)
    Ytestprob = clas.predict_proba(Xtest)
    Ytest2prob = clas.predict_proba(Xtest2)

    print('Train loss: ', log_loss(Ytrain, Ytrainprob), '\nTest loss: ',
          log_loss(Ytest, Ytestprob), '\nSecondary Test loss: ',
          log_loss(Ytest2, Ytest2prob), '\n')

    Train=pd.DataFrame(confusion_matrix(Ytrain,Ytrainpred))
    Test=pd.DataFrame(confusion_matrix(Ytest,Ytestpred))
    Test2=pd.DataFrame(confusion_matrix(Ytest2,Ytest2pred))
```

```
    print(classification_report(Ytest,Ytestpred),'\n')
    print(classification_report(Ytest2,Ytest2pred))
    return Train, Test, Test2, Ytrainprob, Ytestprob, Ytest2prob
Train, Test, Test2, Ytrainprob, Ytestprob, Ytest2prob=TrainClasModel()
```