Aplicación de técnicas de Machine Learning a la predicción de fallos de discos mediante el uso de Spark
(Applying Machine Learning Techniques to Predicting Disk Failures Using Spark)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN CIENCIA DE DATOS

Autor: Alejandro Villanueva Noriega

Director\es: Javier Cacheiro López

Septiembre - 2021

A mi familia, por haberme dado la oportunidad
de formarme académicamente en esta
Universidad, y sobretodo por haberme
educado en lo personal.
A mis amigos, por ser un claro apoyo
incondicional y desconexión en todo momento.
A mi director de trabajo, Javier,
y a Jose Fuentes,
por haberme guiado en la elaboración
de este trabajo de una forma
empática y cercana.

Resumen

Se han utilizado diferentes algoritmos de Machine Learning aplicados a la predicción de fallos en discos duros utilizando los atributos SMART. El problema de clasificación consiste en detectar anomalías en una serie temporal de datos ruidosos. A lo largo de todo el trabajo se hizo uso de Spark para la paralelización de procesos con el fin de reducir los tiempos de ejecución.

Abstract

Different Machine Learning algorithms applied to hard disk failure prediction using SMART attributes have been used. The classification problem consists of detecting anomalies in a time series of noisy data. Throughout the work, Spark was used for process parallelisation in order to reduce execution times.

Key Words: Predictive Maintenance, Spark, Python, Machine Learning, Binary Classification, Hard Disk Drives, SMART

Índice general

Capítulo 1

Introducción

1.1 Mantenimiento Predictivo

En los últimos años se ha producido una explosión del interés por el mantenimiento predictivo. Pero, ¿cómo podemos definir el mantenimiento predictivo? ¿A qué se debe este auge? El mantenimiento es todo el conjunto de acciones que se realizan para que las máquinas y equipos funcionen como se supone que deben hacerlo.

Una estrategia de mantenimiento habitual es la de "Run to Failure" R2F, a veces denominada también mantenimiento correctivo. Consiste en que se arregla una máquina en el instante en el que la misma se rompe. Este tipo de estrategia puede dar lugar a muchos resultados imprevisibles, y roturas inesperadas de máquinas [1].

Por otro lado, existe el mantenimiento preventivo, en el que las reparaciones se llevan a cabo según un calendario regular y orientado al tiempo. Con este enfoque es posible reducir el número de roturas inesperadas [1]. Ahora bien, una gran parte del mantenimiento que se realiza puede ser totalmente innecesario, y pese a ello, se podrían tener roturas inesperadas ya que no se tiene en cuenta el estado puntual de la máquina.

Una alternativa a las estrategias previas es el mantenimiento predictivo, basado en la salud estimada del equipo. Este enfoque puede permitir reducir la incertidumbre de las actividades de mantenimiento, realizando un mantenimiento en el momento adecuado. Para ello, es necesaria la recopilación de grandes cantidades de datos, y la capacidad de preprocesar y analizar los mismos.

El flujo básico de trabajo viene mostrado en la Figura 1.1, dónde las principales fases son la recopilación de datos, la estimación del estado de la máquina, dónde se analizan los datos tomados; y la ejecución de la acción consecuente en función del estado de la máquina.

1.2 Hard Disk Drives

En esta era de la computación en la nube y el Big Data, la fiabilidad de los sistemas de almacenamiento en la nube es un reto importante al que se enfrentan las empresas tecnológicas. Según [2]. [3], el disco duro supone aproximadamente del orden del 80 % de los fallas en Data Centers, dónde comprobaron que alrededor del 8 % de todos sus servidores pueden verse al menos una vez en fallo en un año determinado, siendo esta cifra mayor para aquellas máquinas con muchos de discos duros. La gran escala de un de un

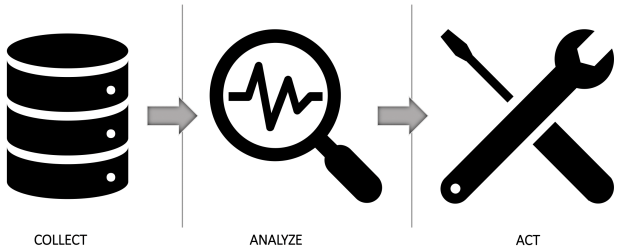


Figura 1.1: Flujo básico de trabajo de un sistema de mantenimiento predictivo.

centro de datos magnifica la probabilidad de fallo de los discos duros, haciendo que los fallos de los discos duros sean uno de los principales casos de mantenimiento.

Los discos duros fueron inventados hace más de 50 años y se utilizan en los ordenadores personales desde mediados de la década de 1980. El microprocesador del ordenador es el encargado de realizar las operaciones y cálculos, mientras que es el disco duro el que proporciona al ordenador su prodigiosa memoria y le permite almacenar fotos digitales, archivos de música, documentos de texto, etc.

Todos los ordenadores necesitan de almacenamiento de información a largo plazo. Este tipo de almacenamiento se conoce como almacenamiento secundario, mientras que la Random Access Memory RAM se conoce como almacenamiento primario.

Un disco duro es un dispositivo de almacenamiento de información no volátil, es decir, que no necesita de un aporte energético constante para conservar dicha información, al contrario que las memorias RAM; y que cuya información puede ser alterada en cualquier momento [4].

La RAM conocida como memoria de acceso aleatorio es el almacenamiento principal de un ordenador. Cuando se utiliza un archivo o se trabaja sobre el mismo, este estará almacenado temporalmente en la memoria RAM. Esta memoria es la que permite realizar tareas comunes como abrir aplicaciones, jugar a juegos, pasar rápidamente de una tarea a otra sin perder el progreso. En resumen, la capacidad de la RAM va ligada a la fluidez y rapidez del ordenador. La RAM es una memoria volátil. Esta característica significa que no puede almacenar información una vez que se apaga el sistema. Por ejemplo, si se copia un fragmento de texto en el portapapeles y se reinicia el ordenador, este fragmento habrá sido olvidado. Esto se debe a que solo se almacenó temporalmente en la RAM. Ahora bien, esta memoria hace posible que un ordenador acceda a los datos en un orden aleatorio y, por lo tanto, lee y escribe mucho más rápido que el almacenamiento secundario de una computadora.

En general, el almacenamiento secundario viene compuesto en dos formas: las unidades de disco duro HDD y las unidades de estado sólido SSD. Estos dispositivos de almacenamiento secundarios pueden ser extraíbles. De esta forma se pueden reemplazar o actualizar

en un ordenador, así como trasladarlos a otro.

Las unidades de disco duro o HDDs están compuestas por piezas mecánicas, de ahí que se les denomine discos duros mecánicos. Estos están formados por discos magnetizados unidos por un mismo eje, conocidos como platos. En cada plato y en cada una de sus caras, un cabezal de lectura/escritura lee o graba los datos sobre los discos. Estos platos están compuestos por miles de millones de áreas diminutas. Mediante los cabezales de escritura/lectura, cada una de esas zonas puede magnetizarse para almacenar un 1 o desmagnetizarse para almacenar un 0 de forma independiente. El magnetismo se utiliza en el almacenamiento informático porque sigue almacenando información incluso cuando se desconecta la corriente. Estos platos giran normalmente entre frecuencias de los 5400 y 15.000 r.p.m. Cuanto más rápido gira el disco magnético, más rápido el ordenador puede acceder a la información. El tamaño de estos discos duros mecánicos suele estar entre los 6 y los 9 cm. Actualmente, los HDD internos pueden llegar a alcanzar una capacidad máxima de 20 TB. Desde la aparición de las SSD, las unidades de disco duro se usan poco como almacenamiento secundario en los ordenadores, pero aún son una fiable opción como almacenamiento externo [5].

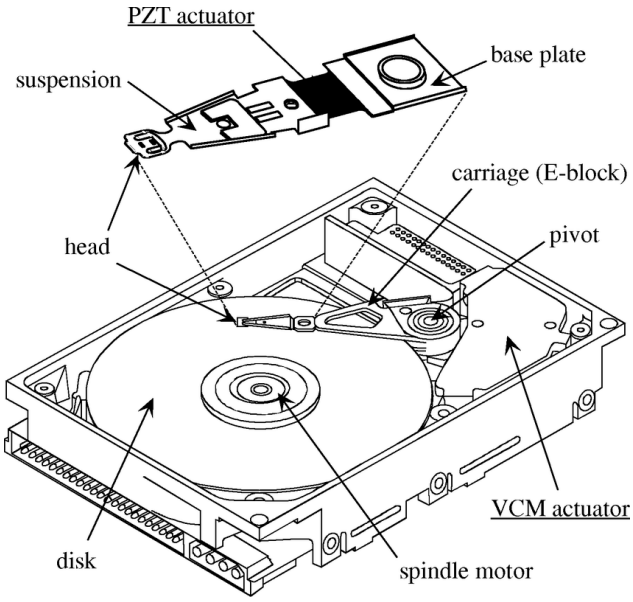


Figura 1.2: Esquema básico de un disco duro. [6]

Se muestra un esquema de los componentes del mismo en la Figura 1.2. Lo más importante de una memoria no solo es el almacenar información, sino poder acceder a ella [4]. Estos datos que se almacenan en el disco duro siguen un patrón muy ordenado en cada uno de los platos. Los bits de información están dispuestos en trayectorias circulares concéntricas llamadas pistas. Cada una de estas pistas esta dividida en áreas llamadas sectores. En el disco duro se almacena también un mapa de dichos sectores que ya se han utilizado,

y otros que aún están por utilizar. Este mapa de sectores en Windows se denomina File Allocation Table FAT. Cuando el ordenador quiere almacenar o leer información, recoge la localización del mapa de sectores y ordena al cabezal de lectura-escritura que se desplace por el plato hasta el lugar dónde insertará o leerá los datos.

En suma, existe una interfaz, llamada controlador, que comunica el entramado mecánico de un disco y su correspondiente ordenador. Consiste de un pequeño circuito que activa los actuadores, selecciona las pistas específicas para la lectura y escritura, y transforma los flujos de datos paralelos que vienen desde el ordenador en flujos de datos en serie para ser escritos en el disco, y viceversa [4].

1.2.1 SMART

Los sistemas de almacenamiento a gran escala se utilizan ampliamente en diferentes ámbitos, incluidos aquellos sistemas informáticos de alto rendimiento y proveedores de servicios de Internet. Las grandes escalas de sistemas de almacenamiento incrementan en gran medida la incidencia de los fallos en los discos duros HDDs, conduciendo a pérdidas de datos difíciles de recuperar e incluso permanentes.

Para garantizar las fiabilidad y estabilidad de estos sistemas de almacenamiento, se monitorizan las condiciones de funcionamiento de los discos duros en tiempo real. En general, se utilizan sensores térmicos, acelerómetros, contadores o incluso sensores de emisión acústica. Este sistema de monitorización se denomina SMART, Self Monitoring Analysis and Reporting Technology [7]. Su principal funcionalidad es generar un histórico de una serie de variables de dichos discos duros. Uno de los principales problemas de estos sistemas de monitorización es que existen muchas incoherencias en las estadísticas ya que muchos de los fabricantes de discos duros utilizan diferentes definiciones y mediciones. Cabe destacar a su vez que existen dos principales tipos de fallos en los HDD: predecibles y no predecibles.

Los fallos predecibles incluyen las averías que aparecen en el tiempo y que están causadas por una mecánica defectuosa del HDD o por aquellos daños superficiales. Por otro lado, los fallos no predecibles están causados por acontecimientos repentinos, como pueden ser subidas de tensión repentinas, o daños inesperados en los circuitos del disco duro o la unidad de estado sólido. La tecnología SMART únicamente nos ayuda a intentar predecir aquellos fallos predecibles.

Con la ayuda de enfoques estadísticos, técnicas de Machine Learning e incluso Deep Learning, se han propuesto varios métodos para mejorar la precisión en la predicción de fallos de dichos discos duros. Estos métodos han logrados grandes rendimientos de predicción, aunque tienen que combatir principalmente con dos problemas.

El primer problema se debe a que los diferentes discos duros varían considerablemente en función del fabricante. Por consecuente, se hace una tarea extremadamente compleja el generalizar un modelo para todos los discos, ya que el rendimiento de los métodos de predicción depende en gran medida de la búsqueda de los mejores parámetros. Por otra parte, otro problema con el que tienen que lidiar estos métodos es que existe una gran diferencia en el número de unidades en fallo y unidades con estado normal. Este desequilibrio en los datos dificulta el rendimiento de la predicción de fallos en los discos duros [8], [9].

Los atributos SMART surgieron tras la necesidad de alertar de forma temprana si un disco iba a tener un fallo inminente. Para ello, se crearon estos sistemas de monitorización con el fin de intentar estimar la salud del disco en todo momento. Estos atributos SMART vienen descritos y mostrados en la Tabla A.1 del Anexo A.

Los atributos que son funcionales para unos modelos pueden no serlo para otro [10], y de igual forma, ocurre entre fabricantes. El intervalo de estos atributos suele ser de 0 a 100 y para algunos otros de 0 a 255, aunque no existe ninguna norma de normalización de estos atributos para los fabricantes.

Las variables más críticas en las fallas de los discos duros son los atributos 5, 12, 187, 188, 189, 190, 198, 199 y 200 [11].

Nuestro objetivo es desarrollar un método eficaz de predicción de fallos en HDDs mediante algoritmos de clasificación de Machine Learning, identificando las unidades que van a fallar con una ventana concreta de días de antelación.

Capítulo 2

Fundamento Teórico y Estado del Arte

2.1 Series Temporales

Una serie temporal es un conjunto de N observaciones que están ordenadas y equidistantes en tiempo de una unidad observable en diferentes momentos. Cuando se observa una única variable se denominan series univariantes o escalares mientras que si se trata de varias características se denomina serie multivariante o vectorial [12].

En general, una serie temporal univariante se representa de la siguiente forma:

$$y_1, y_2, ..., y_N \; ; \; (y_t)_{t=1}^{N} \; ; \; (y_t : t = 1, ..., N)$$

dónde y_t es la observación número t $(1 \leq t \leq N)$ de la serie y N es el número de observaciones. Por consecuente, esta serie puede venir representada por un vector columna $\mathbf{y} \equiv [y_1, y_2, ..., y_N]'$ de orden $N \times 1$.

Por otro lado, una serie temporal multivariante o vectorial se representa de la siguiente forma:

$$\boldsymbol{y_1}, \boldsymbol{y_2}, ..., \boldsymbol{y_N} \; ; \; (\boldsymbol{y_t})_{t=1}^{N} \; ; \; (\boldsymbol{y_t} : t = 1, ..., N)$$

dónde $\boldsymbol{y_t} \equiv [y_{t1}, y_{t2}, ..., y_{tM}]'$ $(M \geq 2)$ es la observación número t $(1 \leq t \leq N)$ de la serie, N es el número de observaciones y M es la variable observada. Por consecuente, esta serie puede venir representada por una matriz $\boldsymbol{Y}$ de orden $N \times M$:

$$\boldsymbol{Y} \equiv \begin{bmatrix} \boldsymbol{y_1'} \\ \boldsymbol{y_2'} \\ \vdots \\ \boldsymbol{y_N'} \end{bmatrix} \equiv \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1M} \\ y_{21} & y_{22} & \cdots & y_{2M} \\ \vdots & \vdots & ... & \vdots \\ y_{N1} & y_{N2} & \cdots & y_{NM} \end{bmatrix}$$

dónde y_{tj} es la observación número t $(1 \leq t \leq N)$ de la serie sobre la variable número j $(1 \leq j \leq M)$.

2.2 Predicción de fallos en HDDs

La predicción de fallas en HDDs es una tarea relevante y muy demandada no solo por aquellos proveedores de servicios en la nube, sino que es también un área de investigación interesante dentro de la comunidad científica [13]. Gracias a la introducción de los datos de monitorización SMART se pudo enfocar el análisis de la predicción con enfoques de Machine y Deep Learning.

El método de predicción más extendido y utilizado por la gran mayoría de la comunidad es la clasificación binaria. El propósito principal de los diferentes trabajos que utilizan este enfoque es determinar si un HDD fallará en un determinado periodo de tiempo o no.

Botezatu et al. [14] proponen un enfoque de este tipo utilizando Transfer-Learning y Regularized Greedy Forests para la clasificación, obteniendo una precisión final de 98% en la predicción de los fallos con 10-15 días de antelación. Por otra parte, Xiao et al. [15] proponen un mecanismo para predecir estas fallas en los HDDs mediante una clasificación Random Forest, cuya construcción permite al modelo seguir aprendiendo a medida que se van recogiendo más datos. En suma, Murray et al. [16] desarrollan un nuevo algoritmo basado en el aprendizaje de múltiples instancias y el clasificador Naive-Bayes; prometedor para la detección de aquellos eventos raros en series temporales de datos ruidosos y no distribuidos paramétricamente, insisten. Además, también se han desarrollado modelos con redes neuronales. Sun et al. [17] proponen un enfoque basado en redes neuronales convolucionales para predecir dichas fallas.

Por otra parte, otro enfoque propuesto es la modelización de la tarea de predicción de fallos como un enfoque de detección de anomalías. Wang et al. [18] utilizan la distancia de Mahalanobis, la transformación de Box-Cox y las pruebas de razon de verosimilitud generalizada, aplicados a este nuevo enfoque. Hamerly y Elkan [19] hacen uso de este enfoque proponiendo un modelo que es mezcla de dos submodelos, un clasificador Naive-Bayes con maximización de expectativas y otro clasificador Naive-Bayes para detectar las anomalías. Aussel et al. [20] realizan un estudio en el que se evalúa el rendimiento de los algorítmos de clasificación SVM, Random Forest y XGBoost, así como el impacto y relación con la extracción de características. Zhu et al. [21] presentan un modelo de red neuronal recurrente y un modelo basado en SVMs para la predicción de la salud del disco duro.

También, se han propuesto otro tipos de modelos con el fin de inferir el tiempo hasta el fallo como variable continua. Chaves et al. [22] utilizan una red bayesiana para intentar obtener este tiempo hasta el fallo, prediciendo así la salud del disco.

2.3 Algoritmos de clasificación

En el desarrollo de este trabajo se han utilizado tres algoritmos de Machine Learning para la tarea de clasificación. Estos son la regresión logística binaria, las máquinas de vectores soporte y Random Forest. Estos algoritmos serán descritos en más detalle a posteriori.

El principal objetivo de estas tareas de clasificación es intentar aproximar aquella desconocida función que relaciona las variables predictoras con la variable objetivo. Esta función

desconocida puede definirse como $Y = f(x_1, x_2, ..., x_n)$ siendo Y la variable objetivo, en nuestro caso categórica, y $x_1, x_2, ..., x_n$ son las características.

2.3.1 Regresión Logística Binaria

La regresión logística es un método estadístico cuyo propósito es modelar la probabilidad de una variable categórica binaria (dos posibles valores) o multinomial (más de dos posibles valores) en función de una o más variables independientes.

Consiste en, dado un conjunto de observaciones $\{y_i, x_{i1}, ..., x_{np}\}_{i=1}^{n}$, obtener los parámetros β_p que describen la ecuación siguiente:

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}$$

dónde $\{x_{i1}, ..., x_{ip}\}$ son las características y y_i la variable a clasificar, para una observación concreta.

Si la variable a clasificar es binaria (0 y 1), matemáticamente es posible resolver el modelo de regresión lineal por mínimos cuadrados. No obstante, dado que se pueden obtener valores distintos de 0 y 1, distintos de lo valores de categoría y fuera de los rangos de probabilidad, se introduce la sigmoide. La regresión logística transforma el valor resultado de la regresión lineal con esta función sigmoide de tal forma que el resultado final pertenece al intervalo [0,1].

La función sigmoidal se define a continuación:

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

2.3.2 Linear Support Vector Machine

Las máquinas de vectores soporte SVM intentan ajustar un hiperplano de separación óptimo entre las clases centrándose en las observaciones de entrenamiento que se encuentran en el borde de las de las distribuciones de clase, denominados los vectores de soporte. Todas las demás muestras de entrenamiento se descartan de hecho, ya que no contribuyen a la estimación de la ubicación del hiperplano. Por consecuente, se puede tener una gran precisión con un conjunto de entrenamiento pequeño [23].

Este método de clasificación viene mostrado en la Figura 2.1. La distancia entre estos vectores soporte que definen el hiperplano de separación se denomina margen.

Un hiperplano en el espacio de características se define mediante la siguiente ecuación:

$$wx + b = 0$$

dónde x es un punto situado en el hiperplano, w es la normal al hiperplano y b es el sesgo.

El margen entre dos hiperplanos paralelos al hiperplano óptimo será de $\|w/2\|$. En general, en los problemas de clasificación estaremos en la situación (b) de la Figura 2.1, dónde las clases no son linealmente separables. Por consecuente, tendremos que penalizar aquellas clases que se clasificarán mal. El problema de optimización en las máquinas de vectores soporte es la minimización de estos errores por la mala clasificación de algunas observaciones mientras se intenta maximizar el margen:

$$min\left[\frac{\|w^2\|}{2} + C\sum_{i=1}^{n}\xi_i\right]$$

dónde la primera parte de la ecuación pretender maximizar el margen entre las clases; y la segunda se refiere a la penalización de aquellas observaciones que son mal clasificadas. El valor de C es un parámetro de control de balance entre estas dos consideraciones. De esta forma si C es un valor pequeño, la penalización por mala clasificación es baja, por lo que se elige un hiperplano con un gran margen a expensas de un mayor número de errores de clasificación; mientras que si C es un valor grande, el algoritmo intenta minimizar el número de observaciones mal clasificadas resultando un margen mas pequeño.

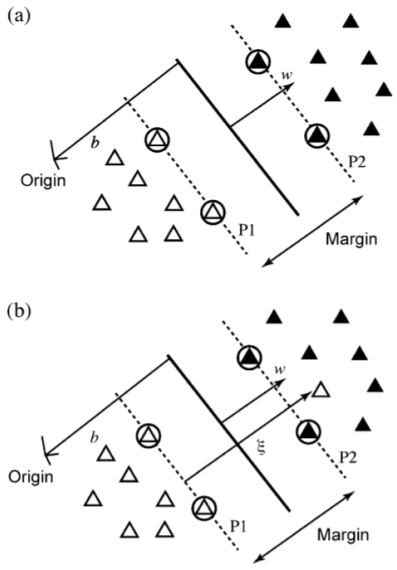


Figura 2.1: (a) La situación linealmente separable y (b) situación con observaciones no tan definidas. Las observaciones rodeadas representan los vectores soporte. [23]

2.3.3 Random Forest

El algoritmo Random Forest es un método de Machine Learning que utiliza el concepto de aprendizaje por *Ensemble*. Este tipo de métodos utilizan el concepto de generar un modelo global combinando un gran número n de modelos no correlacionados. Es decir, se entrena un modelo que es combinación de modelos más débiles [20]. Los resultados de este conjunto de modelos se fusionan en un modelo de predicción global. Las dos principales formas de combinar estos aprendizajes débiles son el boosting y el bagging. El boosting mejora iterativamente los modelos, centrándose en las instancias que fueron clasificadas

de forma errónea en las iteraciones anteriores. Por otra parte, el bagging genera muchos modelos individuales y proporciona un resultado derivado de la decisión mayoritaria.

El algoritmo Random Forest pertenece a la categoría de aprendizaje bagging. Este algoritmo crea arboles de decisión basados en los datos de entrenamiento, y a posteriori, agrega los resultados de estos modelos débiles para generar un predicción final.

Los árboles de decisión son candidatos ideales para los métodos de conjunto, ya que suelen tener un sesgo bajo y una varianza alta, lo que los hace muy que se beneficien del proceso de promediación [24]. Al aplicar esta técnica de conjunto, se evita la desventaja principal de los árboles de decisión, su gran tendencia a sobreajustar los datos de entrenamiento. Ahora bien, para ello estos árboles no han de tener ningún tipo de correlación. Por consecuente, se utilizan únicamente características seleccionadas al azar así como las muestras de entrenamiento, es decir, para cada árbol de decisión, se muestrea un subconjunto de todos los datos de entrenamiento con reemplazo y, a continuación, el espacio de características de cada muestra también se muestrea de forma aleatoria.

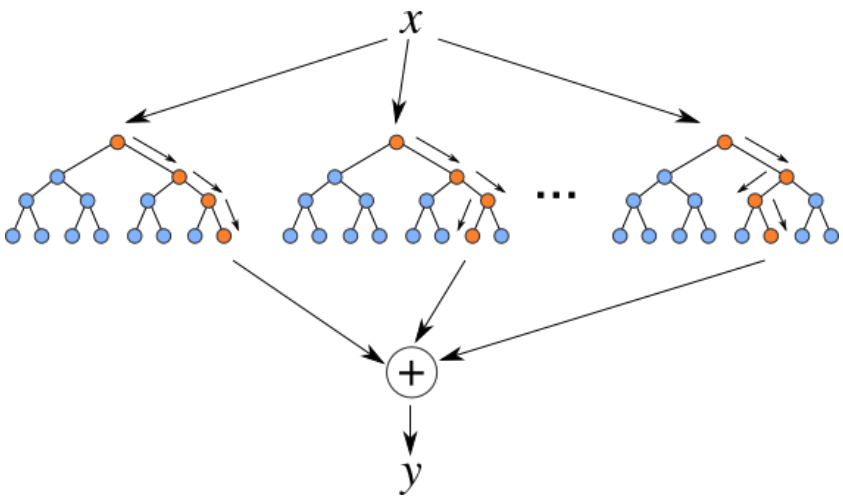


Figura 2.2: Esquema del algoritmo Random Forest. [25]

El algoritmo puede utilizarse tanto para regresión o clasificación, ya sea binaria o multiclase. En cuanto a la regresión, las predicción final será la media aritmética de las predicciones individuales; mientras que en el caso de la clasificación será la clase con mayor número de apariciones. Este procedimiento viene esquematizado en la Figura 2.2

Los principales hiperparámetros del clasificador Random Forest son los siguientes:

- **max_depth**: La profundidad máxima se define como el camino más largo entre el nodo raíz y el nodo hoja.
- **min_samples_leaf**: Este hiperparámetro de Random Forest especifica el número mínimo de muestras que deben estar presentes en el nodo hoja después de dividir un nodo.
- **n_estimators**: Número de árboles en el bosque.

- **max_sample**: Este hiperparámetro determina qué fracción del conjunto de datos original se da a cualquier árbol individual.
- **max_features**: Esto se asemeja al número de características máximas proporcionadas a cada árbol en un bosque aleatorio.
- **bootstrap**: Método de muestreo de puntos de datos (con o sin reemplazo).
- **criterion**: Mide la calidad de cada división. Se puede utilizar la impureza de Gini, o la Entropía, basada en la ganancia de información.

2.3.4 Reducción de la dimensión

La gran mayoría de los datos que se manejan hoy en día, suelen tener una alta dimensionalidad. La reducción de la dimensionalidad es la transformación de los datos de alta dimensión en una representación significativa de dimensionalidad reducida.

Lo idóneo es que esta representación reducida de los datos posea una dimensionalidad que se corresponda con la dimensionalidad intrínseca de los datos, es decir, que posea el número mínimo de parámetros necesarios para caracterizar de las propiedades observadas de los datos

La reducción de la dimensionalidad es relevante en la gran mayoría de ámbitos del Machine Learning, ya que facilita, entre otras cosas, la clasificación, la visualización y la compresión de datos de alta dimensión. Existen diferentes métodos de reducción de la dimensión, ya sean lineales o no lineales.

PCA: Principal Components Analysis

El análisis de componentes principales PCA es una técnica de lineal de reducción de la dimensionalidad, lo que significa que realiza la reducción de la dimensionalidad incrustando los datos en un subespacio lineal de menor dimensionalidad [26].

Este método construye una representación de baja dimensionalidad de los datos que describe la mayor parte de la varianza de los mismo. Esto se hace encontrando una base lineal de dimensionalidad reducida para los datos, en en la que la cantidad de varianza de los datos sea máxima.

2.4 Computación Distribuida: Apache Spark

Apache Spark es un framework de procesamiento de datos distribuidos a gran escala, ya sea en instalaciones de Data Centers o datos en la nube [27]. Desde sus orígenes en el AMPLab de la Universidad de Berkeley en 2009, Apache Spark se ha convertido en uno de los principales motores unificados de procesamiento distribuido de Big Data del mundo [28].

Además, Spark puede desplegarse de diversas maneras, proporciona enlaces nativos para los lenguajes de programación Java, Scala, Python y R. En suma, como se muestra en la Figura 2.3, Spark posee de librerías con APIs de Machine Learning *MLlib*, SQL para consultas interactivas *Spark-SQL*, procesamiento de flujos *Structured Streaming* para interactuar con datos en tiempo real, y el procesamiento de grafos *GraphX*.

Sin llegar a muchos detalle, la infraestructura de una aplicación Spark, mostrada en la Figura 2.4, consta de un driver que es responsable de orquestar las operaciones paralelas

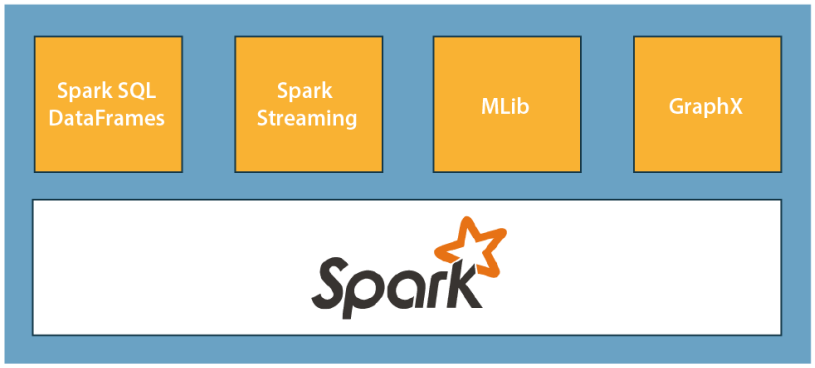


Figura 2.3: Componentes de Apache Spark y cojunto de APIs. [29]

en el clúster de Spark. Este driver accede a los componentes distribuidos en el clúster (los ejecutores de Spark y el gestor del clúster) a través de una SparkSession. El driver de la aplicación Spark es el responsable de instanciar una SparkSession, comunicarse con con el gestor del clúster, solicitar recursos como CPU, memoria, etc, transformar todas las operaciones de Spark en cómputos de grafo, planificar y distribuir su ejecución como tareas entre los ejecutores de Spark. El gestor de clústeres es responsable de la gestión y asignación de recursos para el clúster de nodos en los que se ejecuta su aplicación Spark. Por otro lado, un ejecutor de Spark se ejecuta en cada nodo trabajador del clúster. Los ejecutores se comunican con el driver y son los responsables de ejecutar las tareas en los trabajadores.

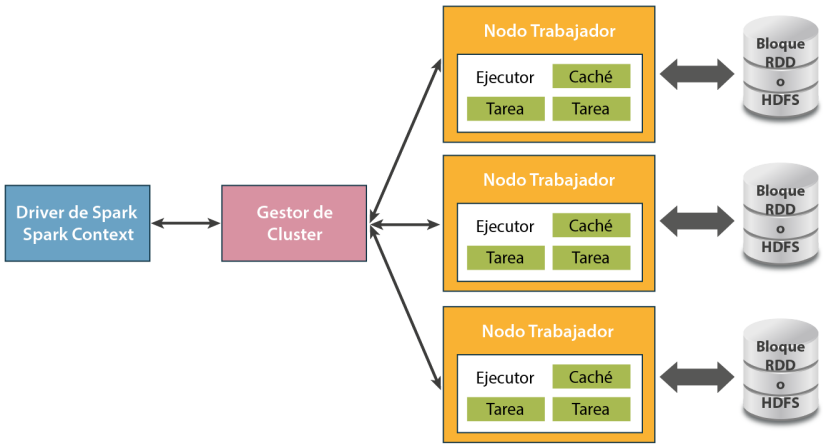


Figura 2.4: Arquitectura de una aplicación Spark. [29]

Los datos físicos se distribuyen en en el almacenamiento como particiones que residen en Hadoop Distributed File System HDFS o en el almacenamiento en la nube. A cada ejecutor de la aplicación de Spark se le asigna una tarea en función de su localidad, permitiendo que los estos procesen sólo los datos que están cerca de ellos, minimizando el ancho de banda de la red [27].

Por otra parte, las operaciones de Spark sobre los datos distribuidos se pueden categorizar en dos tipos: transformaciones y acciones.

Las transformaciones, en el caso de los DataFrames por ejemplo, transforman este objeto de Spark en un nuevo DataFrame sin alterar los datos originales. Una transformación no cambiará el DataFrame original, sino que devolverá los resultados transformados de la operación en un nuevo DataFrame. Estas transformaciones se evalúan de forma *lazy*, es decir, el resultado no se calcula en el momento de ser invocada la transformación, sino que se van registrando en cadena. Esta evaluación *lazy* permite retrasar la ejecución de Spark hasta que se invoque una acción. En ese momento, se reorganizan y optimizan las transformaciones en etapas para una ejecución más eficiente.

Spark proporciona un conjunto de métodos comunes para leer datos desde diferentes fuentes. La fuente de datos por defecto en Spark es Parquet. Parquet es utilizado por muchos frameworks y plataformas de procesamiento de Big Data ya que es un formato de archivo columnar, open-source y autodescriptivo, que permite un gran ahorro en el espacio almacenamiento y un acceso rápido a las columnas de datos.

Mientras que formatos como CSV o Avro están basados en filas, un fichero Parquet está orientado a columnas [30], es decir, los valores de cada columna de una tabla se registran uno al lado del otro. Esta diferencia en el registro de los datos se muestra en la Figura 2.5. Además, este formato de archivos es autodescriptivo, es decir, en sus metadatos, vienen incluidos el esquema y la estructura.

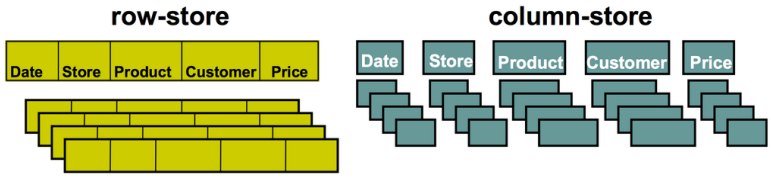


Figura 2.5: Contraste entre un almacenamiento en formato fila y columnar. [30]

A diferencia de los formatos de archivo basados en filas, como CSV, Parquet tiene como propósito optimizar su rendimiento. Cuando se ejecutan consultas en su sistema de archivos, se centra solo en los datos relevantes, escaneando un conjunto menor de datos para responder a la consulta. Por ejemplo, se considera una tabla con 1000 columnas. Normalmente, solo se necesitarán un subconjunto de las mismas. El uso de archivos Parquet permite obtener solo las columnas necesarias y sus valores, cargarlos en memoria y responder a la consulta; mientras que si se utilizar un formato de archivo basado en fila, habría que cargar toda la tabla en memoria, conllevando a un peor rendimiento de la consulta.

2.5 Clasificación Desequilibrada

Un problema de clasificación desequilibrada es un ejemplo de problema de clasificación en el que la cantidad de observaciones de una clase difiere enormemente de la cantidad de observaciones de la otra clase (problema de clasificación binaria) o del resto de clases (problema de clasificación multiclase). En estos problemas de clasificación, una o más clases están subrepresentadas dentro del dataset [31]. Existen muchas situaciones en las que estamos ante problemas de este tipo, como la detección de anomalías, diagnósticos médicos, o como en nuestro caso, la detección de fallos. Cuando se trabaja con un problema de clasificación desequilibrada, la clase minoritaria suele ser la de mayor interés.

Los algoritmos comunes de clasificación tienden a ser sesgados hacia la clase mayoritaria, ya que las reglas que predicen correctamente esas observaciones, se ponderan positivamente a favor de la métrica de validación. Sin embargo, las reglas específicas, que en general son las que tienden a predecir la clase minoritaria, pueden ser ignoradas [31]. Por consecuente, el modelo no es capaz de predecir correctamente la clase minoritaria. Es más, consideremos un datasets de clasificación binaria cuya relación entre clases es 1:100. Si el modelo predice siempre que cualquier observación pertenece a la clase mayoritaria, la *accuracy* del mismo será del 99% sobre los datos de test, pero un 0% sobre los datos de clase minoritaria. Por consecuente, en los problemas de clasificación binaria desequilibrada, la *accuracy* sobre el conjunto de datos de test no es una buena métrica de validación del modelo. Las principales métricas base de validación en este tipo de problemas son las siguientes:

- **TP: True positives**: Un verdadero positivo es un resultado en el que el modelo predice correctamente la clase positiva.
- **TN: True negatives**: Un verdadero negativo es un resultado en el que el modelo predice correctamente la clase negativa.
- **FP: False positives**: Un falso positivo es un resultado en el que el modelo predice erróneamente la clase positiva.
- **FN: False negatives**: Un falso negativo es un resultado en el que el modelo predice erróneamente la clase negativa.

A partir de estas métricas de validación, se construye la matriz de confusión, dónde cada fila representa la clase real de la observación, mientras que cada columna la clase predicha. Esta matriz de confusión aporta no solo métricas del modelo en general sino también sobre las diferentes clases.

Otro tipo de métricas comunes en clasificación imbalanceada son la *Sensitivity* y *Specificity* así como la *Precision* y el *Recall*.

- **Sensitivity** $= TP/(TP+FN)$: Expresa cómo de bien se ha predicho la clase positiva.
- **Specificity** $= TN/(FP+TN)$: Expresa cómo de bien se ha predicho la clase negativa.

  La *Sensitivity* y la *Specificity* pueden combinarse en una única métrica que balancea ambos conceptos llamada $G-Mean$.

  - **G-Mean** $= \sqrt{Sensitivity \cdot Specificity}$
- **Precision** $= TP/(TP+FN)$: Resume la fracción de ejemplos asignados a la clase positiva que pertenecen a la clase positiva.

- **Recall** $= TP/(TP + FN)$: Resume lo bien que se predijo la clase positiva. Es la misma métrica que la *Sensitivity*.

  La *Precision* y la *Recall* pueden combinarse en una única métrica que balancea ambos conceptos llamada $F - Score$.

  - **F-Score** $= 2 \cdot Precision \cdot Recall/(Precision + Recall)$

Otra métrica de validación es la curva ROC Receiver Operating Characteristic, que resume el análisis de la validación de los clasificadores binarios en función de su capacidad para discriminar las clases. La curva ROC resume el comportamiento del modelo calculando la tasa de falsos positivos (*Recall* ó *Sensitivity*) y la tasa de verdaderos positivos para un conjunto de predicciones sobre diferentes umbrales. Cada umbral es un punto dentro de la gráfica, formando una curva como se muestra en la Figura 2.6.

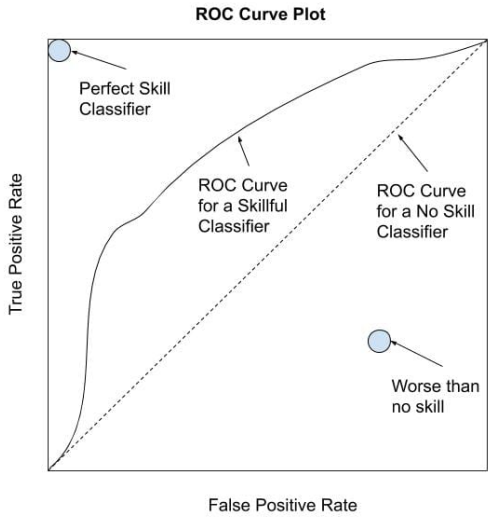


Figura 2.6: Curva ROC. [32]

Un modelo simple que predice siempre la clase mayoritaria estará representado por la diagonal. El mejor modelo posible será un punto en la parte superior izquierda de la gráfica. Para poder comparar diferentes modelos de una forma cuantitativa se suele utilizar el área bajo la curva AUC, dónde está métrica va desde 1 para el mejor modelo y 0 para un modelo que no predice ninguna observación correctamente.

Por otra parte, para intentar mejorar el rendimiento de estos algoritmos de clasificación en problemas de datos imbalanceados, se desarrollan diferentes técnicas con el fin de distinguir correctamente la clase minoritaria. Estas técnicas se pueden resumir en cuatro grupos principales en función de su enfoque: a nivel algoritmo, a nivel de datos, a nivel de aprendizaje sensible o a nivel de aprendizaje por *ensemble*.

En nuestro problema, abordaremos las técnicas a nivel de datos, dónde el propósito fundamental de las misma es re-equilibrar la distribución de las clases mediante el muestreo del espacio de datos. De tal forma, se evita una modificación del algoritmo de aprendizaje ya que este paso es previo al aprendizaje del modelo. Además, estas técnicas son independientes del clasificador subyacente [31].

Las técnicas de muestreo se pueden clasificar en tres familias. Los métodos de submuestreo, que crean un subconjunto del conjunto de datos original, mediante la eliminación de observaciones, normalmente instancias de la clase mayoritaria. Por otra parte, los métodos de sobremuestreo, que amplifican el conjunto de datos creando, en general, nuevas muestras de la clase minoritaria. Finalmente, las técnicas híbridas que combinan ambos enfoques.

SMOTE: Synthetic Minority Oversampling Technique

El algoritmo SMOTE es una técnica de sobremuestreo. En vez de replicar las observaciones de la clase minoritaria, SMOTE genera nuevas observaciones sintéticas [33]. Estas nuevas observaciones se generan mediante interpolaciones con las observaciones de la clase a reproducir.

El procedimiento de este algoritmo viene mostrado en la Figura 2.7. Consiste en la elección de un conjunto de puntos aleatorios de clase minoritaria, en la Figura 2.7 se muestra solo un punto x_1. Para este conjunto de puntos, se identifican los K, 5 por defecto, vecinos más cercanos, x_i con $\{x_2, ..., x_6\}$. Con estos K vecinos para cada punto se interpola de una forma aleatoria nuevos puntos (puntos rojos) pertenecientes a esta clase minoritaria.

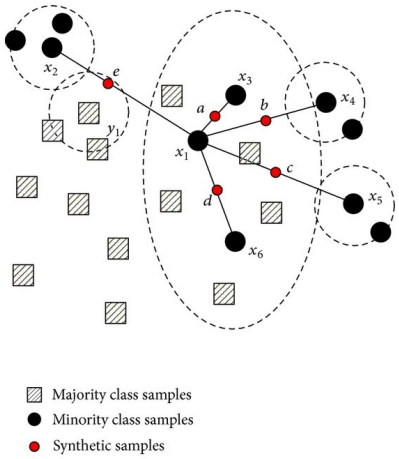


Figura 2.7: Procedimiento del algoritmo SMOTE. [34]

Capítulo 3

Descripción y Preprocesado de los Datos

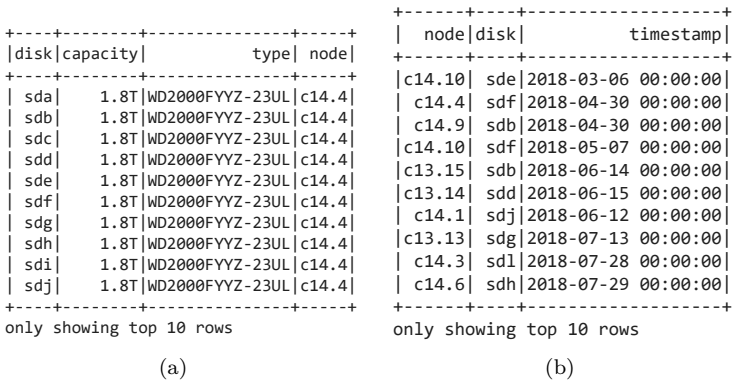
Los datos utilizados durante todo el estudio son datos procedentes de las instalaciones del Centro de Supercomputación de Galicia CESGA, dónde se monitorizan los atributos SMART para diferentes discos duros de un clúster. Se han utilizado tres ficheros: un fichero en formato parquet que recoge las series temporales de estos atributos para diferentes discos; otro fichero en formato parquet que recoge las fechas en las que se producen cambios de mantenimiento en los discos y finalmente un fichero de texto plano dónde se caracteriza el tipo de disco para cada uno de los mismos.

El primer fichero parquet se carga como DataFrame de Spark para su posterior ETL. Este DataFrame está formateado de tal forma que cada una de sus filas corresponden a un único disco, dónde las columnas de atributos SMART son series temporales en formato MapType, con clave timestamp del registro y valor la medida del atributo. Se muestra un slice del DataFrame en la Figura 3.1. La frecuencia en la que se registraban los atributos SMART para cada disco es de 15 minutos; y se tienen datos desde el 01/01/2018 hasta el 01/06/2021.

```
+----+-----+----------+----------+--------------------------------+
|disk| node|start_time|  end_time|smart_Power_Cycle_Count_raw_value|
+----+-----+----------+----------+--------------------------------+
| sdc|c14.1|1514764800|1622505600|             [1547817301 -> 12...|
| sdj|c14.1|1514764800|1622505600|             [1547817301 -> 16...|
| sdd|c14.2|1514764800|1622505600|             [1548071101 -> 92...|
| sdl|c14.3|1514764800|1622505600|             [1548081901 -> 7....|
| sdf|c14.4|1514764800|1622505600|             [1548081902 -> 84...|
| sdl|c14.4|1514764800|1622505600|             [1548081902 -> 17...|
| sde|c14.5|1514764800|1622505600|             [1548777601 -> 97...|
| sdg|c14.5|1514764800|1622505600|             [1548777601 -> 97...|
| sdj|c14.5|1514764800|1622505600|             [1548777601 -> 17...|
| sdk|c14.5|1514764800|1622505600|             [1548777601 -> 17...|
+----+-----+----------+----------+--------------------------------+
only showing top 10 rows
```

Figura 3.1: Formato de los datos de las series temporales previo a la ETL.

El segundo fichero parquet corresponde con los timestamps de los fallos para los diferentes discos. En consecuencia, bastó con formatear los tipos y cruzar los datos con el primer fichero DataFrame, añadiendo así al previo las fechas y horas de los fallos de cada uno de los discos. Finalmente, el último fichero de texto, se formateó para poder unirlo al DataFrame, añadiendo así la característica del tipo de disco. Se muestran en la Figura 3.2 los dos DataFrames formateados.



disk	capacity	type	node
sda	1.8T	WD2000FYYZ-23UL	c14.4
sdb	1.8T	WD2000FYYZ-23UL	c14.4
sdc	1.8T	WD2000FYYZ-23UL	c14.4
sdd	1.8T	WD2000FYYZ-23UL	c14.4
sde	1.8T	WD2000FYYZ-23UL	c14.4
sdf	1.8T	WD2000FYYZ-23UL	c14.4
sdg	1.8T	WD2000FYYZ-23UL	c14.4
sdh	1.8T	WD2000FYYZ-23UL	c14.4
sdi	1.8T	WD2000FYYZ-23UL	c14.4
sdj	1.8T	WD2000FYYZ-23UL	c14.4

only showing top 10 rows

(a)

node	disk	timestamp
c14.10	sde	2018-03-06 00:00:00
c14.4	sdf	2018-04-30 00:00:00
c14.9	sdb	2018-04-30 00:00:00
c14.10	sdf	2018-05-07 00:00:00
c13.15	sdb	2018-06-14 00:00:00
c13.14	sdd	2018-06-15 00:00:00
c14.1	sdj	2018-06-12 00:00:00
c13.13	sdg	2018-07-13 00:00:00
c14.3	sdl	2018-07-28 00:00:00
c14.6	sdh	2018-07-29 00:00:00

only showing top 10 rows

(b)

Figura 3.2: (a) DataFrame final formateado de las características de los diferentes discos. (b) DataFrame final formateado de las fechas de fallos de los diferentes discos.

Debido a problemas en la extracción de los datos, las series temporales estaban divididas y por consecuente, para cada disco se tenían dos filas. La primera parte de la ETL consistió en fusionar estás filas y formatear todas las columnas al tipo deseado; obteniendo un total de 61 filas, es decir, se están explorando 61 discos independientes. A la hora de cruzar los datos de las fechas de fallo, encontramos que había discos que no tenían ningún registro de fallo. Estos fueron omitidos.

Una vez fusionadas las series temporales, se filtran las fechas hasta el día del fallo para cada disco. Si el disco tiene mas de un fallo se toma el de fecha más antigua. Asimismo, se añaden las características, tipo de disco y capacidad, a cada uno de los discos. Se muestra en la Figura 3.3 la distribución de los tipos de discos. Dado que los atributos SMART pueden diferir entre fabricantes, se ha seleccionado únicamente el disco con más observaciones, *WDC WD2003FZEX-0*. Por consecuente, estamos estudiando 22 discos independientes.

Como se comentó previamente, cada una de las celdas del DataFrame corresponde a las series temporales de cada uno de los discos. Estas series temporales se exploran, transformando el DataFrame de tal forma que cada una de las filas corresponden a un registro temporal de un disco en concreto, es decir, expandimos estas celdas únicas por discos y las concatenamos unas con otras, obteniendo un total de 1.609.317 filas.

Para este tipo disco en concreto, se observa que existen 9 atributos que siempre aparecen como nulos, y por tanto los eliminamos. Parece ser que el fabricante de este tipo de discos no monitoriza estos 9 atributos en concreto. Además, aquellas filas que poseen algún atributo

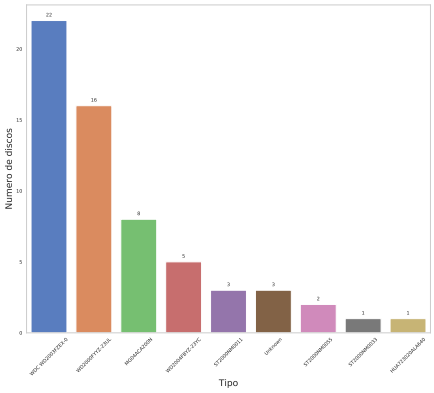


Figura 3.3: Distribución de los tipos de discos.

con valor nulo las eliminamos también. Obteniendo un total de 1,609,115 filas. Se muestran en las Figuras A.1 y A.2 del Anexo A las series temporales de dos discos diferentes.

Una vez se tiene el DataFrame formateado se procede al etiquetado de las instancias de falla en los datos.

El primer problema que surge a la hora de intentar resolver este problema de mantenimiento predictivo es que se desea predecir que un disco va a fallar con una ventana de días de antelación, denominada horizonte o lag. A lo largo de todo el trabajo se han definido tres horizontes de falla, 1, 2 y 7 días, y por consecuente, serán problemas de clasificación diferentes, dónde se entrenará un modelo independiente para cada uno de ellos. Cabe destacar que si definimos un horizonte de falla de N días, las observaciones de los N días antes del fallo real también se etiquetaron como fallos; generando distintos desequilibrios entre la distribución de las clases. Esta forma de etiquetado viene mostrada en la Figura 3.4. Además, se muestran en la Tabla 3.2 las distintas distribuciones de las observaciones en base si han sido etiquetadas como fallo o no para los distintos horizontes.

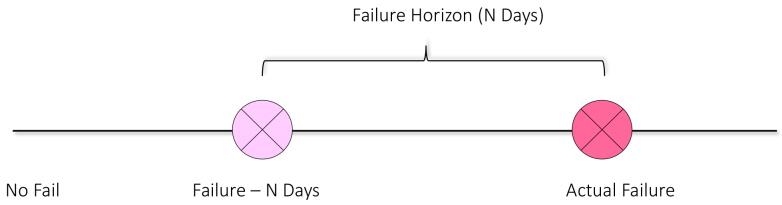


Figura 3.4: Metodología de asignación de fallas a los datos.

Clase	*Lag = 7* días	*Lag = 2* días	*Lag = 1* días
1 - Fallo	14,415	3,916	1,920
0 - No-Fallo	1,594,700	1,605,199	1,607,195
Proporción	1:110	1:410	1:840

Tabla 3.1: Distribución y proporción de las clases para los distintos horizontes analizados.

Se observan las proporciones 1:110, 1:410, 1:840, respectivamente. Para hacer frente a este problema de desequilibrio entre clases, se utiliza el algoritmo SMOTE. Dada la complejidad del problema de la detección de fallos y su enorme desequilibrio entre clases, se obtuvieron resultados paupérrimos tras el muestreo del algoritmo SMOTE para el re-equilibrio entre clases. Por consecuente, se afrontó este problema haciendo uso de un submuestreo aleatorio sin reemplazo. Se muestra en la Tabla la re-distribución de los DataSets de entrenamiento para los diferentes lags, una vez realizado el submuestreo.

Clase	*Lag = 7* días	*Lag = 2* días	*Lag = 1* días
1 - Fallo	8,653	2,356	1,127
0 - No-Fallo	8.469	2,307	1,079

Tabla 3.2: Distribución y proporción de las clases para los distintos horizontes analizados.

Además, tras varios entrenamientos de modelos, se observó que el uso de ciertas medidas SMART distorsionaba la generalización del modelo; por tanto, se omitieron las mismas, utilizando únicamente las más destacadas en la literatura [11].

Por último, para el entrenamiento de los algoritmos, se tomó como DataSet de entrenamiento una proporción aleatoria del 0.6 sobre el DataFrame global. Ya que el desequilibrio prevalece tras hacer la partición, es unicamente en el DataSet de entrenamiento dónde se realiza el submuestreo. Finalmente, el DataSet de test se corresponde con el 0.4 restante del global.

Capítulo 4

Análisis y Resultados

En primer lugar, cabe destacar que el balanceo de las diferentes clases se llevó a cabo en primera instancia mediante el algoritmo de SMOTE para la ventana temporal de 7 días. No obstante, no se generalizaba bien a la hora de predecir los fallos y por consecuente se omitió este modo de re-equilibrio de las instancias de clase. Se llevó a cabo mediante submuestreo aleatorio sin reemplazo.

La proporción de las clases obviamente dependerá de la ventana temporal que utilicemos.

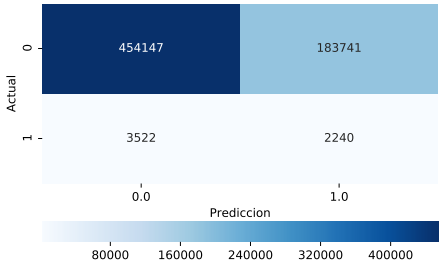
4.1 Ventana temporal: 7 días

Se muestran en las Figuras 4.1, 4.2 y 4.3 los resultados de los algoritmos utilizando el DataFrame etiquetado con un horizonte de 7 días.

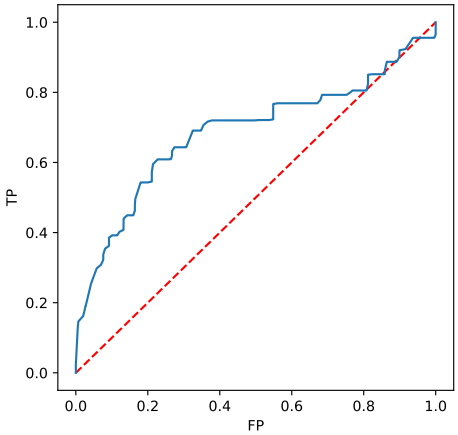
Se observa cómo ninguno de los modelos generaliza correctamente el comportamiento de la anomalía. Es más, se obtienen valores grandes de falsas alarmas, 3522, 3758 y 2703, respectivamente. Los valores del área bajo la curva ROC no superan el 0.6, y se obtienen *Specificity* bajas, salvo para el modelo SVM, con una *Specificity* de 0.8. En suma, los valores de *Sensitivity* que hacen constancia de la *accuracy* sobre la falla, son muy bajos, con un máximo de 0.564 para el modelo Random Forest.

El hecho de tomar un horizonte de 7 días puede haber infectado la fracción de falla del DataSet, ya que puede que los atributos SMART a siete días previos del fallo no den cuenta del desgaste del mismo; haciendo que el modelo no generalice de una forma correcta.

Regresión Logística Binaria



(a) Matriz de confusión.



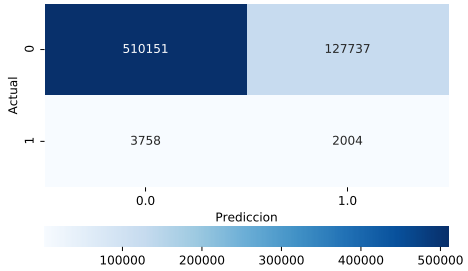
(b) Curva ROC. AUC = 0.531

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.012	0.233	0.389
0 - No-Fallo	0.992	0.829	0.712

(c) Distintas métricas de validación.

Figura 4.1: Métricas de validación para el modelo de regresión logística sobre los datos submuestrados con un horizonte de 7 días. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.389; *Specificity* = 0.712; *G-Mean* = 0.526 .

Linear Support Vector Machine



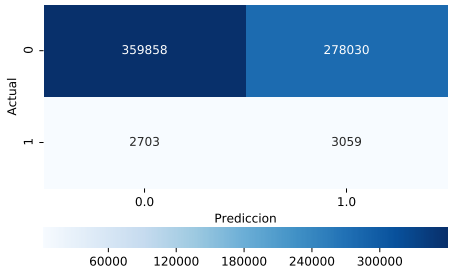
(a) Matriz de confusión.

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.015	0.030	0.348
0 - No-Fallo	0.993	0.886	0.800

(b)

Figura 4.2: Métricas de validación para el modelo de SVM sobre los datos submuestrados con un horizonte de 7 días. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.348; *Specificity* = 0.800; AUC = 0.528; *G-Mean* = 0.528 .

Random Forest



(a) Matriz de confusión.

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.011	0.021	0.531
0 - No-Fallo	0.993	0.719	0.564

(b)

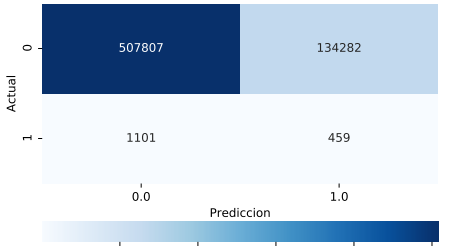
Figura 4.3: Métricas de validación para el modelo de Random Forest sobre los datos submuestrados con un horizonte de 7 días. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.564; *Specificity* = 0.531; AUC = 0.548; *G-Mean* = 0.547 .

4.2 Ventana temporal: 2 días

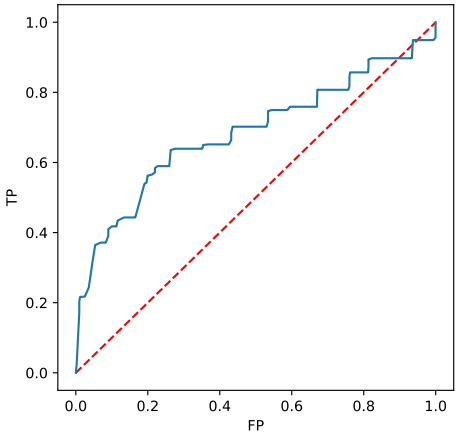
Se muestran en las Figuras 4.4, 4.5 y 4.6 los resultados de los algoritmos utilizando el DataFrame etiquetado con un horizonte de 2 días.

Se observa cómo ninguno de los modelos generaliza correctamente el comportamiento de la anomalía. Es más, se obtienen valores grandes de falsas alarmas, 1101, 1259 y 834, respectivamente. Estos valores de falsas alarmas vienen implícitos en los bajos valores de *Specificity*, dando evidencia de que estos modelos para este horizonte en concreto no son capaces de predecir de forma correcta. En este caso, como el lag es menor, se obtienen menos datos de fallo. Para el modelo de Random Forest, se obtiene un valor para el AUC de 0.611, con un valor de *Specificity* de 0.465, dando cuenta del poco poder de predicción de fallo.

Regresión Logística Binaria



(a) Matriz de confusión.



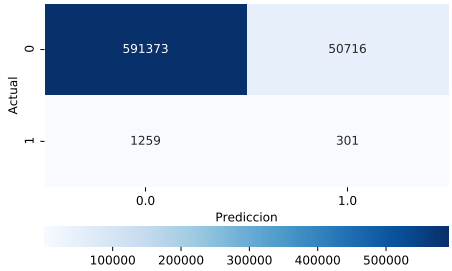
(b) Curva ROC. AUC = 0.562

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.003	0.007	0.294
0 - No-Fallo	0.998	0.882	0.791

(c) Distintas métricas de validación.

Figura 4.4: Métricas de validación para el modelo de regresión logística sobre los datos submuestrados con un horizonte de 2 días. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.294; *Specificity* = 0.791; *G-Mean* = 0.482 .

Linear Support Vector Machine



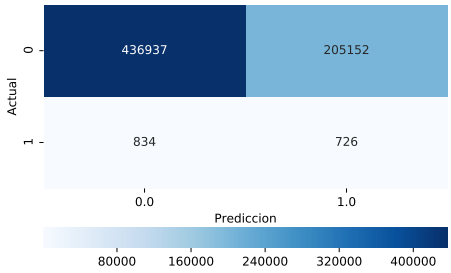
(a) Matriz de confusión.

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	006	0.115	0.193
0 - No-Fallo	0.998	0.958	0.921

(b)

Figura 4.5: Métricas de validación para el modelo de SVM sobre los datos submuestrados con un horizonte de 2 días. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.193; *Specificity* = 0.921; AUC = 0.551; *G-Mean* = 0.422 .

Random Forests



(a) Matriz de confusión.

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.004	0.007	0.465
0 - No-Fallo	0.998	0.810	0.681

(b)

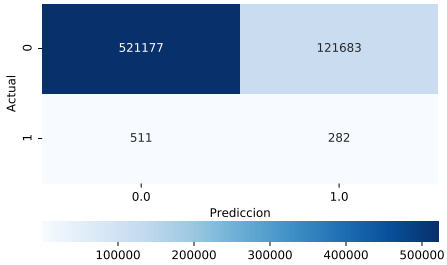
Figura 4.6: Métricas de validación para el modelo de Random Forest sobre los datos submuestrados con un horizonte de 2 días. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.465; *Specificity* = 0.681; AUC = 0.611; *G-Mean* = 0.563 .

4.3 Ventana temporal: 1 día

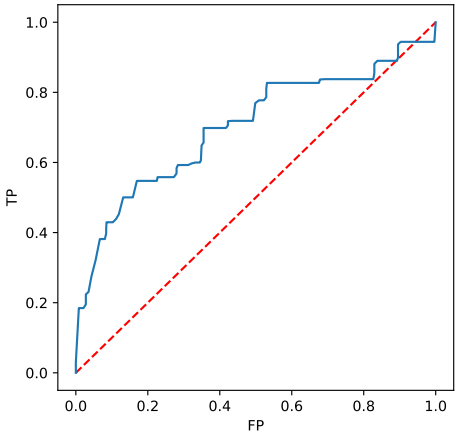
Se muestran en las Figuras 4.7, 4.8 y 4.9 los resultados de los algoritmos utilizando el DataFrame etiquetado con un horizonte de 1 día.

Se observa cómo ninguno de los modelos generaliza correctamente el comportamiento de la anomalía. Es más, se obtienen valores bajos de *Specificity* que dan en consecuencia un gran número de falsas alarmas, 511, 676 y 335, respectivamente. El modelo de regresión logística binaria tiene un valor para el área bajo la curva ROC de 0.637, pero no es capaz de predecir correctamente el fallo, obteniendo un valor de *Specificity* de 0.356.

Regresión Logística Binaria



(a) Matriz de confusión.



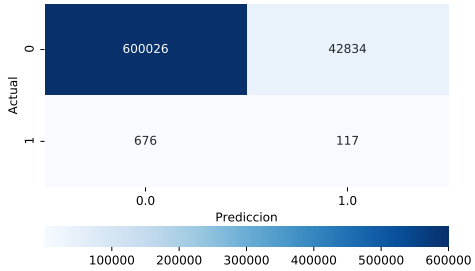
(b) Curva ROC. AUC = 0.637

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.002	0.004	0.356
0 - No-Fallo	0.999	0.895	0.811

(c) Distintas métricas de validación.

Figura 4.7: Métricas de validación para el modelo de regresión logística sobre los datos submuestrados con un horizonte de 1 día. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.356; *Specificity* = 0.811; *G-Mean* = 0.537 .

Linear Support Vector Machine



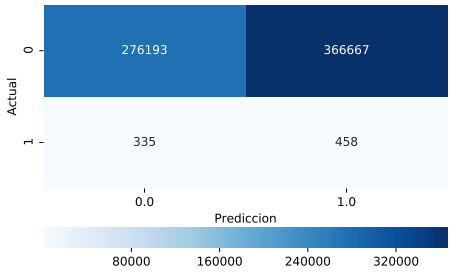
(a) Matriz de confusión.

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.003	0.005	0.148
0 - No-Fallo	0.999	0.965	0.933

(b)

Figura 4.8: Métricas de validación para el modelo de SVM sobre los datos submuestrados con un horizonte de 1 día. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.148; *Specificity* = 0.933; AUC = 0.572; *G-Mean* = 0.372 .

Random Forests



(a) Matriz de confusión.

Clase	*Precision*	*F-Score*	*Recall*
1 - Fallo	0.001	0.002	0.578
0 - No-Fallo	0.999	0.601	0.430

(b)

Figura 4.9: Métricas de validación para el modelo de Random Forest sobre los datos submuestrados con un horizonte de 1 día. En suma, el resto de métricas son las siguientes *Sensitivity* = 0.430; *Specificity* = 0.578; AUC = 0.528; *G-Mean* = 0.499 .

Todo el proceso de ETL y el entrenamiento de los diferentes modelos se encuentra en diferentes Jupyter Notebooks en un repositorio de GitHub https://github.com/avn-mds/TFM.

Capítulo 5

Conclusiones

En el presente trabajo se han utilizado diferentes algoritmos de clasificación con el fin de predecir a N días de antelación el fallo de un disco duro. Se muestran las conclusiones más importantes obtenidas a lo largo del estudio:

- Los resultados de los diferentes modelos evidencian la dificultad de los problemas de clasificación imbalanceada, obteniendo modelos que no tienen una capacidad óptima de predicción, con un gran número de falsas alarmas. Este hecho no generaría ningún tipo de valor añadido, ya que una implementación de mantenimiento preventivo se ajustaría de una forma más adecuada al problema.
- Se ha evidenciado la potencia de procesamiento de PySpark para el proceso de la ETL, reduciendo los tiempos de ejecución.
- Se ha comprendido el principal problema del mantenimiento predictivo, la dificultad de predicción con N días de antelación. Se observa que para un lag N grande, se tienen más datos de fallo, pero por otro lado, se aleja demasiado del punto de falla pudiendo tener datos que no dan cuenta de los efectos de fallo. Por otra parte, para un lag N corto, se tienen muy pocos datos de fallo.
- Se ha comprendido la dificultad de predicción de las fallas en discos dada la poca estandarización de las distintas medidas SMART.
- Una posible mejora es el entrenamiento de modelos con memoria como las LSTM, o añadir agregaciones semanales a los modelos, con el fin de dar cuenta del desgaste acumulado en los discos.

Apéndice A

Anexo 1

SMART ID	Atributo	Descripción
smart_1	Read Error Rate Rate of hardware	Errores de lectura que se produjeron al leer datos de una superficie de disco.
smart_2	Throughput Performance	Rendimiento de una unidad de disco duro.
smart_3	Spin-Up Time	Tiempo medio en milisegundos de giro del cabezal (desde cero RPM hasta que está totalmente operativo).
smart_4	Start/Stop Count	Recuento de los ciclos de arranque/parada del cabezal.
smart_5	Reallocated Sectors Count	Recuento de sectores reasignados.
smart_7	Seek Error Rate	Índice de errores de búsqueda de los cabezales magnéticos.
smart_8	Seek Time Performance	Rendimiento medio de las operaciones de búsqueda de los cabezas magnéticas.
smart_9	Power-On Hours	Recuento de horas en estado de encendido.
smart_10	Spin Retry Count	Un recuento total de los intentos de inicio de giro para alcanzar la velocidad de funcionamiento.
smart_11	Recalibration Retries	Recuento de que se solicitó la recalibración.
smart_12	Power Cycle Count	Recuento de los ciclos de encendido y apagado del disco duro completo.
smart_184	End-to-End error	Recuento de los errores de paridad que se producen en la ruta de datos al soporte a través de la memoria caché de la unidad RAM.
smart_187	Reported Uncorrectable Errors	Recuento de errores que no pudieron ser recuperados utilizando el hardware ECC.
smart_188	Command Timeout	Recuento de las operaciones abortadas debido a que el disco duro de la unidad de disco duro.

smart_189	High Fly Writes	Recuento de información reescrita o reasignada a lo largo de la vida de la la unidad.
smart_190	Temperature Difference	El valor es igual a (100-temp. C), lo que permite fabricante establecer un umbral mínimo que corresponde a una temperatura máxima.
smart_191	G-sense Error Rate	Recuento de los errores resultantes de los inducido por golpes y vibraciones.
smart_192	Power-off Retract Count	Número de ciclos de apagado o retracción de emergencia de emergencia.
smart_193	Load Cycle Count	Recuento de ciclos de carga/descarga en la posición de posición de la zona de aterrizaje de la cabeza.
smart_194	Temperature	Indica la temperatura del dispositivo.
smart_195	Hardware ECC Recovered	Tiempo entre errores corregidos por ECC.
smart_196	Reallocation Event Count	Recuento de los intentos de transferir datos de los sectores reasignados a un área de reserva. Se cuentan tanto los intentos y los intentos fallidos se cuentan.
smart_197	Current Pending Sector Count	Recuento de sectores "inestables" (en espera de ser reasignación, debido a errores de lectura irrecuperables errores de lectura).
smart_198	Uncorrectable Sector Count	El recuento total de errores no corregibles al lectura/escritura de un sector.
smart_199	UltraDMA CRC Error Count	Recuento de errores en la transferencia de datos a través de el cable de interfaz según lo determinado por ICRC (Interface Cyclic Redundancy Check).
smart_200	Multi-Zone Error Rate	Recuento de errores encontrados al escribir un sector
smart_201	Soft Read Error Rate	Recuento indica el número de errores de lectura errores de lectura del software.
smart_223	Load/Unload Retry Count	Recuento de veces que la cabeza cambia de posición.
smart_240	Head Flying Hours	Tiempo empleado durante el posicionamiento de los cabezales de los cabezales.

Tabla A.1: Descripción de los diferentes atributos SMART. [10]

5, 12, 187, 188, 189, 190, 198, 199 y 200

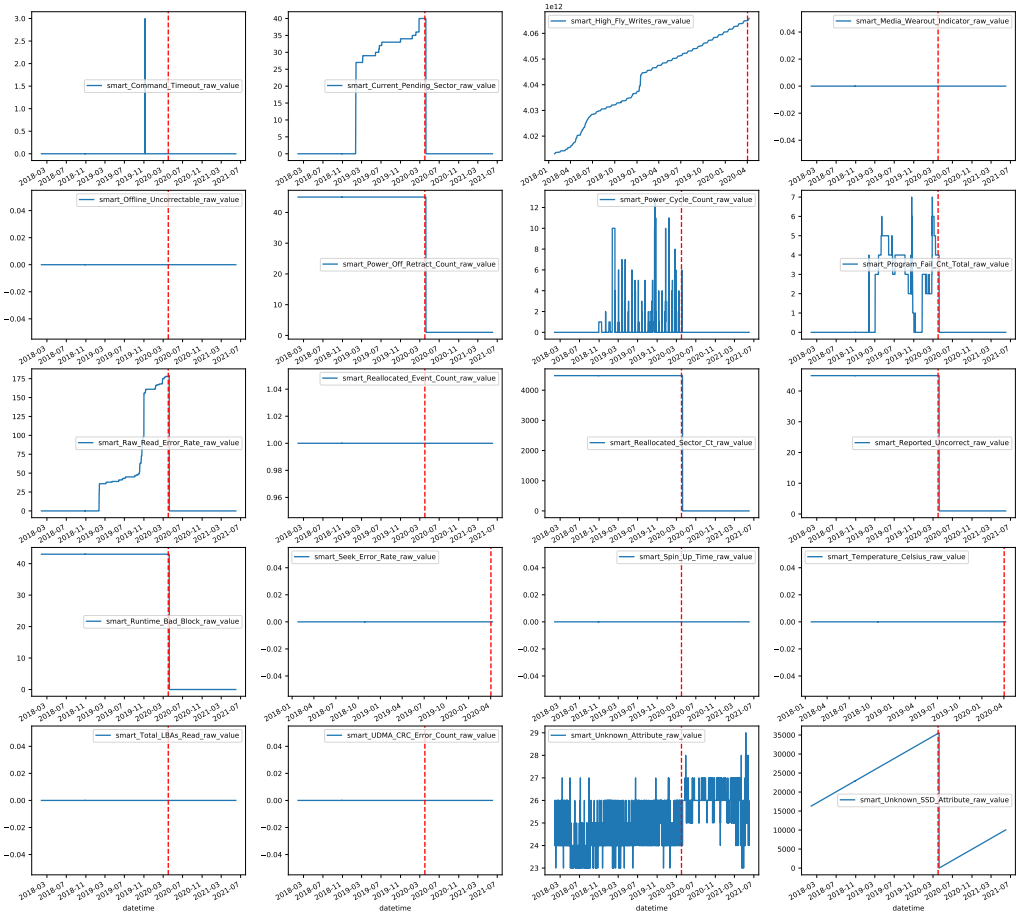


Figura A.1: Series temporales de los diferentes atributos SMART para un disco en concreto. La línea vertical corresponde al día de falla. Disco: *sdh*, Nodo: *c13.12*

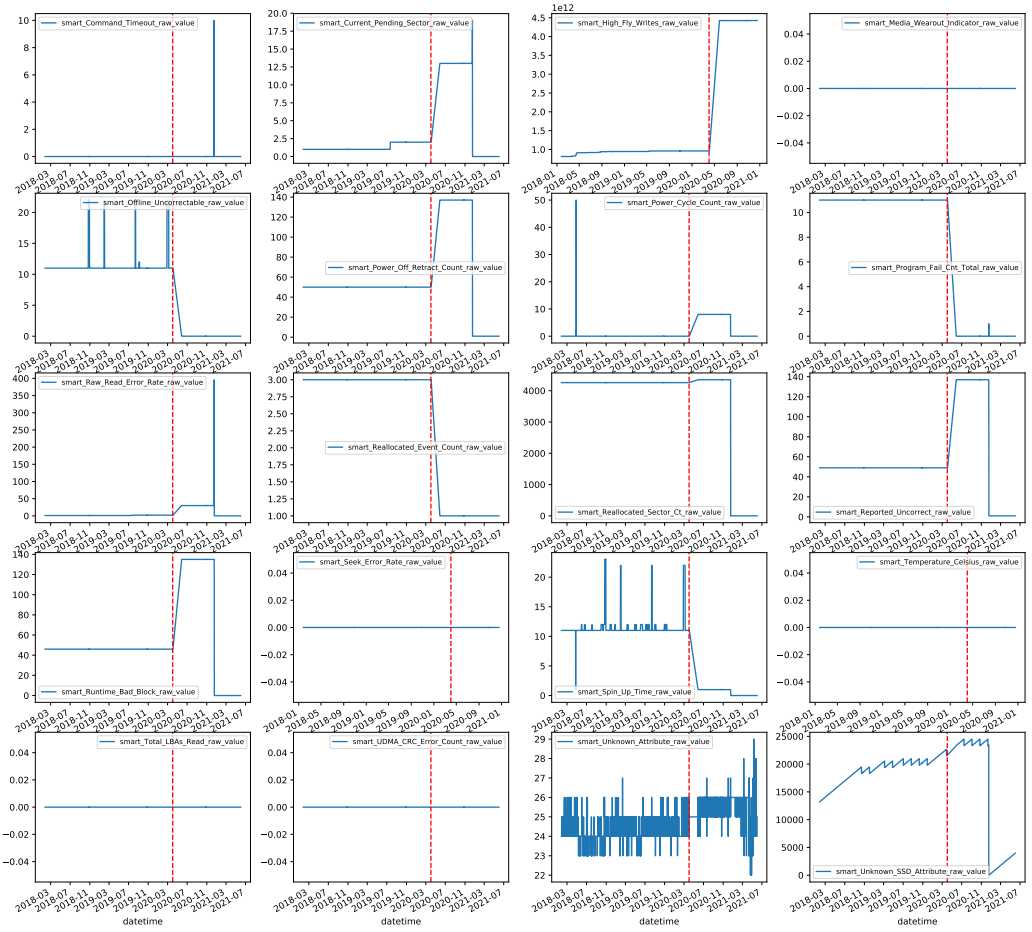


Figura A.2: Series temporales de los diferentes atributos SMART para un disco en concreto. La línea vertical corresponde al día de falla. Disco: *sdb*, Nodo: *c13.1*

Bibliografía

[1] Machine learning for predictive maintenance - part 1. https://datatonic.com/insights/machine-learning-predictive-maintenance/, Apr 2021.

[2] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 193–204, New York, NY, USA, 2010. Association for Computing Machinery.

[3] Guosai Wang, Lifei Zhang, and Wei Xu. What can we learn from four years of data center hardware failures? In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 25–36. IEEE, 2017.

[4] A. Silberschatz, H.F. Korth, S. Sudarshan, and F.S. Pérez. *Fundamentos de bases de datos*. McGraw-Hill, 2006.

[5] Chris Woodford. How does a hard drive work? https://www.explainthatstuff.com/harddrive.html, Aug 2020.

[6] Hidehiko Numasato and Masayoshi Tomizuka. Setting control and performance of dual-actuator systems for hard disk drives. *Mechatronics, IEEE/ASME Transactions on*, 8:431 – 438, 01 2004.

[7] Riccardo Pinciroli, Lishan Yang, Jacob Alter, and Evgenia Smirni. The life and death of ssds and hdds: Similarities, differences, and prediction models. *arXiv preprint arXiv:2012.12373*, 2020.

[8] Jing Shen, Jian Wan, Se-Jung Lim, and Lifeng Yu. Random-forest-based failure prediction for hard disk drives. *International Journal of Distributed Sensor Networks*, 14(11):1550147718806480, 2018.

[9] Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Transactions on Computers*, 65(11):3502–3508, 2016.

[10] Henish Hemendra Balu. Predicting hard disk drive failures and misbehavior. 2020.

[11] Nicolas Aussel, Samuel Jaulin, Guillaume Gandon, Yohan Petetin, Eriza Fazli, and Sophie Chabridon. Predictive models of hard drive failures based on operational data. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 619–625, 2017.

[12] William WS Wei. Time series analysis. In *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2*. 2006.

[13] Marwin Züfle, Christian Krupitzer, Florian Erhard, Johannes Grohmann, and Samuel Kounev. To fail or not to fail: Predicting hard disk drive failure time windows. In *MMB*, pages 19–36, 2020.

[14] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. Predicting disk replacement towards reliable data centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2016.

[15] Jiang Xiao, Zhuang Xiong, Song Wu, Yusheng Yi, Hai Jin, and Kan Hu. Disk failure prediction in data centers via online learning. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[16] Joseph F Murray, Gordon F Hughes, Kenneth Kreutz-Delgado, and Dale Schuurmans. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 6(5), 2005.

[17] Xiaoyi Sun, Krishnendu Chakrabarty, Ruirui Huang, Yiquan Chen, Bing Zhao, Hai Cao, Yinhe Han, Xiaoyao Liang, and Li Jiang. System-level hardware failure prediction using deep learning. In *2019 56th ACM/IEEE design automation conference (DAC)*, pages 1–6. IEEE, 2019.

[18] Yu Wang, Eden WM Ma, Tommy WS Chow, and Kwok-Leung Tsui. A two-step parametric method for failure prediction in hard disk drives. *IEEE Transactions on industrial informatics*, 10(1):419–430, 2013.

[19] Greg Hamerly, Charles Elkan, et al. Bayesian approaches to failure prediction for disk drives. In *ICML*, volume 1, pages 202–209. Citeseer, 2001.

[20] Nicolas Aussel, Samuel Jaulin, Guillaume Gandon, Yohan Petetin, Eriza Fazli, and Sophie Chabridon. Predictive models of hard drive failures based on operational data. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 619–625. IEEE, 2017.

[21] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. Proactive drive failure prediction for large scale storage systems. In *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*, pages 1–5. IEEE, 2013.

[22] Iago C Chaves, Manoel Rui P de Paula, Lucas GM Leite, Joao Paulo P Gomes, and Javam C Machado. Hard disk drive failure prediction method based on a bayesian network. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[23] Giles M Foody and Ajay Mathur. Toward intelligent training of supervised image classifications: directing training data acquisition for svm classification. *Remote Sensing of Environment*, 93(1-2):107–117, 2004.

[24] Gilles Louppe. Understanding random forests. *Cornell University Library*, 10, 2014.

[25] Jagandeep Singh. Random forest: Pros and cons. https://medium.datadriveninvestor.com/random-forest-pros-and-cons-c1c42fb64f04, Dec 2020.

[26] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[27] Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Inc., 1st edition, 2015.

[28] Hugh Watson. Update tutorial: Big data analytics: Concepts, technology, and applications. *Communications of the Association for Information Systems*, 44:364–379, 01 2019.

[29] Por Ari Handler Gamboa , Por, , Ari Handler GamboaEstudiante de Grado en Informática en la UAM., Ari Handler Gamboa, and Estudiante de Grado en Informática en la UAM. Introducción a apache spark - batch y streaming. https://www.adictosaltrabajo.com/2015/11/16/introduccion-a-apache-spark-batch-y-streaming/, Jun 2019.

[30] What is a columnar database? definition and related faqs. https://www.omnisci.com/technical-glossary/columnar-database.

[31] Alberto Fernández, Salvador García, Mikel Galar, Ronaldo C Prati, Bartosz Krawczyk, and Francisco Herrera. *Learning from imbalanced data sets*, volume 10. Springer, 2018.

[32] Jason Brownlee. Tour of evaluation metrics for imbalanced classification. https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/, Apr 2021.

[33] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[34] Feng Hu and Hang Li. A novel boundary oversampling algorithm based on neighborhood rough set model: Nrsboundary-smote. *Mathematical Problems in Engineering*, 2013, 11 2013.