Universidad de Valladolid

Grado en Estadística

Detección y Clasificación de Fallos en Motores mediante Procedimientos Boosting

Trabajo de Fin de Grado

Autor

Alejandro Barón García

Tutor

Miguel Alejandro Fernández Temprano

Resumen

En cualquier tipo de motor, los fallos siempre están presentes. Una inspección del motor podría indicarnos el estado del mismo, pudiendo así preveer futuras averías. Sin embargo, puede existir imposibilidad de parar un motor para inspeccionarlo ya sea porque su funcionamiento es continuo o por ser una tarea costosa que implique montado y desmontado de numerosas piezas.

Mediante la toma de medidas de una corriente inducida en el variador del motor por vibraciones debidas a fallos en el rotor, podemos tratar de predecir el estado del motor sin hacer una inspección invasiva. Esta metodología de mantenimiento predictivo se conoce como *Motor Current Signature Analysis*.

En este trabajo se presenta una metodología estadística basada en técnicas de Boosting para clasificar los motores en un estado de deterioro según las medidas de esta corriente inducida. La metodología desarrollada explora los diferentes procedimientos boosting existentes en la literatura, incluyendo aquellos de más reciente desarrollo, y determina que existen factores que hacen que deban utilizarse distintos clasificadores para la detección de fallos. Además permite, no solo saber qué variables deben utilizarse en cada configuración de motor para obtener los mejores resultados en lo que se refiere a la detección y clasificación de fallos, sino que también permite establecer cómo cambian estas variables dependiendo del grado de deterioro del motor, lo que es una aportación relevante al estado del arte en lo que se refiere a la detección y clasificación de fallos en este tipo de situaciones.

Esta memoria plasma el conocimiento desarrollado para el Trabajo de Fin de Grado de Estadística de la Universidad de Valladolid. Desde el punto de vista estadístico, el interés del trabajo reside en el uso de *boosting*, una metodología estadística que está en boga en la comunidad científica y que cada vez más demuestra dar buenos resultados, en establecer una metodología de interpretabilidad para estos modelos y en la aplicación de restricciones isotónicas para la mejora de la predicción.

Abstract

In any kind of motor, failure is always present. Close up inspection could tell us its current state, therefore avoiding future problems. However, we can't always check up the motor in detail, because it needs to be working constantly or disassembling it is not viable.

By taking some current measures induced by vibrations caused by deterioration in the rotor, we can try to predict its current wearing status without performing an invasive check-up. This predictive maintenance technique is known as Motor Current Signature Analysis.

This project showcases a Boosting-driven statistical methodology for motor classification according to their wearing status using the induced current measurement. The methodology developed explores the different boosting procedures existing in the literature, including those of more recent development, and determines that there are factors that make it necessary to use different classifiers for fault detection. Furthermore, it allows not only to know which variables should be used in each engine configuration to obtain the best results in terms of fault detection and classification, but also to establish how these variables change depending on the degree of engine deterioration, which is a relevant contribution to the state of the art in terms of fault detection and classification in this type of situation.

This document shows the development for the Final Degree Project for the Statistics Degree at the University of Valladolid. From a statistical point of view, the interest of this project is to apply boosting techniques, a state of the art algorithm that continuously proves to give good results, to develop an interpetation methodology for these methods and to include isotonic restrictions to improve predictive performance.

Considerad lo que es correcto y verdadero
Practicad y cultivad la ciencia
Familiarizaos con las artes
Conoced los principios del oficio
Entended el prejuicio y beneficio de cada cosa
Aprended a ver cada cosa con exactitud
Tomad conciencia de lo que no es obvio
Sed cuidadosos incluso en los asuntos pequeños
No hagáis nada que sea inútil

- Miyamoto Musashi -

Quiero agradecer:
A mis padres Edith y Juan Luis por su apoyo incondicional y ánimo hasta aquí, el final del camino.
A mi tutor Miguel Alejandro su ayuda y esfuerzos brindados en el desarrollo de este trabajo, sin su guía esto no habría sido posible.
A todos aquellos que me han acompañado en este peregrinaje, tanto a los que han estado desde siempre como a los recién llegados, sois tantos que no cabéis aquí.

Quiero dedicarle este trabajo:
A mi abuelo Juan por ser un ejemplo para mí. Aunque las circunstancias no le brindaron la posibilidad de acceder a la ciencia por la vía académica, siempre persiguió (y persigue) el conocimiento por su cuenta con el afán científico que me ha llevado hasta aquí.

Índice

1. Introducción

1.1 Descripción del problema

Los motores de inducción están presentes en múltiples aplicaciones eléctricas. Desde grandes máquinas en fábricas hasta pequeños electrodomésticos, son la fuente de energía para muchos aparatos. Tradicionalmente se usaban motores de corriente continua, pero hoy en día priman los motores de corriente alterna.

Estos motores suelen ser resistentes, sin embargo, el uso y el desgaste provoca que estos fallen. Debido a que entre un 80% y un 85% del consumo energético de la industria es efectuado por motores de inducción [1], la necesidad de mantenimiento predictivo motiva el uso de técnicas estadísticas para tratar de estudiar el estado del motor sin necesidad de realizar una inspección invasiva, de tal forma que podamos ver con antelación si un motor va a fallar y así evitar un posible peligro de fallo mientras el motor está funcionando, evitando así que se pare el sistema y aumentando la eficiencia del mismo.

Uno de los defectos más comunes en estos motores por el uso y desgaste está en los rotores. Las fisuras o desgastes que estos sufren provocan vibraciones, que a su vez llevan a una oscilación en el motor. Estas oscilaciones inducen, por el fenómeno del electromagnetismo, en la corriente de alimentación un doble armónico (el superior o *Upper Sideband Harmonic*, **USH**, e inferior o *Lower Sideband Harmonic*, **LSH**) respecto a la onda de la corriente suministrada. Esta técnica de mantenimiento predictivo mediante el estudio de los armónicos inducidos se conoce como **Motor Current Signature Analysis**. Un elemento muy relevante en el problema de la detección y clasificación de fallos en motores es el variador de corriente.

Los variadores o inversores de frecuencia son dispositivos que procesan la energía suministrada, de tal forma que esta sea apta para cada motor (i.e. adecuándola a la frecuencia necesaria para garantizar un correcto funcionamiento del mismo, reduciendo así el consumo y el coste de funcionamiento del motor).

A priori, sabemos que a mayor oscilación (por causa de un fallo más severo), mayor será el valor de los armónicos superior e inferior que han sido inducidos. Basándonos en esta hipótesis,

trataremos de realizar una predicción del estado del motor en función de los valores de estos armónicos a lo largo de un periodo de tiempo. Sin embargo, como se indicaba al principio de esta sección, el valor del armónico se sospecha que depende tanto del modelo de **variador** empleado como del **nivel de carga** suministrada al motor.

1.2 Objetivos del proyecto

En este Trabajo de Fin de Grado en Estadística se presentará un estudio realizado con objeto de encontrar un algoritmo predictivo basado en técnicas de Boosting para predecir el estado de deterioro de un motor de inducción. En algunos estudios anteriores [2] se aplicaron métodos de ensemble para el estudio de este problema mediante simulación.

Las técnicas de Boosting cada vez son más populares. Según `https://www.import.io/post/how-to-win-a-kaggle-competition/` podemos ver que el novedoso algoritmo XGBoost, que se estudiará en este TFG, ostenta la hegemonía junto a las redes neuronales en las competiciones del popular sitio Kaggle. Esta popularidad se debe tanto a los buenos resultados que proporciona como a la reciente explosión computacional democratizada cada vez a más gente gracias a los avances recientes de la informática.

Los objetivos de este TFG son los siguientes:

1. **Emplear diferentes técnicas Boosting** para clasificar motores según su estado de deterioro.
2. **Estudiar qué influencia tienen los distintos factores** (modelo de inversor, nivel de carga) en la precisión de los clasificadores.
3. **Comprender qué información** es necesaria para clasificar un motor.
4. **Establecer una metodología de interpretabilidad** de modelos opacos. Los modelos boosting tienen una alta capacidad predictiva pero su interpretabilidad es compleja.
5. **Estudiar cómo se relacionan las medidas** de la corriente con el estado de deterioro según el instante del tiempo en el que se manifiesta esa medida mediante la metodología de interpretabilidad.
6. **Considerar restricciones de monotonía** y estudio de la monotonía de los datos con el objetivo de mejorar la capacidad predictiva de los modelos. Se tiene la intuición de que esta metodología podría mejorar la clasificación debido a que la relación entre las medidas del armónico y la respuesta (estado de deterioro) es monótona creciente; a mayor valor del armónico inducido, mayor deterioro.

Para poder crear un modelo predictivo, este TFG se llevará en colaboración con el Departamento de Ingeniería Eléctrica de la Universidad de Valladolid los cuales han proporcionado los datos con los que se ha realizado el estudio.

1.3 Estructura del documento

Esta memoria se compone los siguientes capítulos:

- **Metodología:** En el que se expone de forma teórica los conceptos, algoritmos y procedimientos de selección que se desarrollarán en el TFG.
- **Descripción y procesado de los datos:** En el que se muestra el conjunto de datos, el preprocesado necesario, así como la selección/transformación de las variables.
- **Análisis mediante técnicas Boosting y resultados:** En el que se desarrolla una comparación de modelos boosting y se estudia el problema según la hipótesis desarrollada por el mejor modelo o modelos.
- **Análisis mediante técnicas Boosting con restricciones isotónicas y resultados:** En el que se adapta la metodología del capítulo anterior estableciendo restricciones de monotonía entre las variables explicativas y la respuesta.
- **Conclusiones:** En el que se resumen los resultados de este trabajo de fin de grado y se sugieren posibles mejoras o trabajo futuro respecto a lo realizado.

1.4 Relación con las asignaturas del grado

Los contenidos de este trabajo de fin de grado se basan en mayor medida en las siguientes asignaturas:

- **Análisis de Datos, Análisis Multivariante y Análisis de Datos Categóricos**: en las que se exponen los algoritmos base para los algoritmos boosting así como el estudio y análisis de métricas y evaluación de clasificadores.
- **Computación Estadística**: En la que se adquieren competencias para el desarrollo de mediante programación en R.
- **Métodos Estadísticos de Computación Intensiva**: En la que se muestran procedimientos estadísticos basados en simulación, así como técnicas de regularización.
- **Regresión y ANOVA y Modelos Estadísticos Avanzados**: En las que se desarrollan los contenidos de la metodología ANOVA y su estudio.
- **Técnicas de Aprendizaje Automático**: En la que se realiza una introducción a python y a clasificadores base de los que parten los métodos boosting.

1.4.1 Ampliación de Materia

En cuanto a elementos tratados en este trabajo que son novedad respecto a los contenidos del Grado:

- **Algoritmos de Boosting**: Se utilizarán algoritmos boosting, concretamente AdaBoost, XGBoost, LightGBM y LogitBoost, así como introducción de regularización Dropout en XGBoost.
- **Métodos de Interpretabilidad**: SHAP y gráficos mediante simulación para interpretabilidad de algoritmos de clasificación opacos.
- **Clasificadores con restricciones monótonas**: de cara a utilizar la información a priori que invita a pensar que habría monotonía en los datos.

2. Metodología

En este capítulo, en primer lugar, se describen métodos genéricos para el problema de clasificación supervisada. Seguidamente, se detallan los algoritmos que se usan en el TFG y finalmente, se exponen cuestiones metodológicas llevadas a cabo para el estudio de los datos y los modelos.

2.1 El problema de la clasificación

Uno de los problemas más en boga hoy en día en la estadística es de clasificación supervisada, es decir, determinar la pertenencia de una observación a un grupo entre K posibles cuando se dispone de una muestra de observaciones cuyo grupo de pertenencia es conocido. Sin embargo, que hoy en día sea importante no quiere decir que sea nuevo.

2.1.1 LDA

Ya en 1936 Ronald Fisher [3] propuso la técnica de Linear Discriminant Analysis para la clasificación de plantas en el archiconocido dataset `iris`. Mediante esta técnica (para el caso de dos grupos), se establece un hiperplano de separación de tal forma que proyectando sobre este, los grupos queden lo más separados posibles. De esta forma podremos separar las observaciones en dos grupos estableciendo un umbral que divida las categorías. En la Figura 2.1 se muestra un ejemplo visual del discriminante lineal.

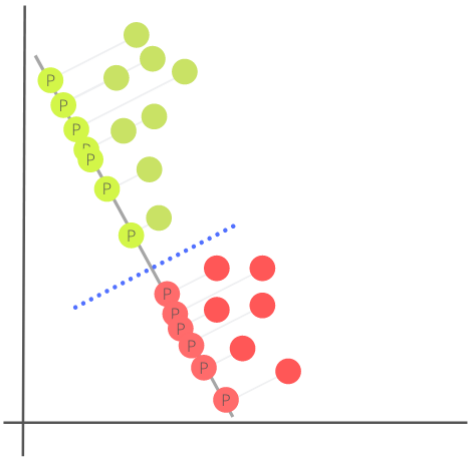


Figura 2.1: Ejemplo de LDA

Este es uno de los modelos más sencillos de clasificación, por ende, altamente interpretable. En el caso multiclase, se puede expresar formalmente de la siguiente manera: si tenemos $k = 1, 2..., K$ grupos, asumiendo igualdad de varianzas entre grupos se puede expresar como:

$$\text{Clasificar } x \text{ en } t \text{ si } \left(x-\mu_t\right)^\top \Sigma^{-1} \left(x-\mu_t\right) \leq \left(x-\mu_k\right)^\top \Sigma^{-1} \left(x-\mu_k\right), k = 1, \ldots, K \qquad (2.1)$$

En la práctica, al ser desconocidos, μ y Σ se sustituyen por sus estimadores $\hat{\mu}$ y S.

2.1.2 Árboles de Decisión

Los árboles de decisión surgieron ya en 1963, cuando James Nelson Morgan y John A. Sonquist presentaron el primer árbol de regresión en su artículo *Problems in the Analysis of Survey Data, and a Proposal*[4]. Este algoritmo, muy popular en la minería de datos, se basa en un grafo con forma de árbol. Cada nodo representa una disyunción entorno a una variable, pudiendo ser estas variables numéricas o categóricas (sin necesidad de hacer variables dummy) [5]. Para calcular las disyunciones, se siguen criterios basados en métricas como el índice de impureza de Gini o la ganancia de información (entropía).

Estas disyunciones aplicadas de forma consecutiva a una observación siguiendo un camino del grafo desde la raíz hasta llegar a las hojas permiten clasificar/estimar la variable respuesta de un individuo. Este modo de clasificación basado en disyunciones es muy parecido al razonamiento humano, por lo que la ventaja de los árboles es su alta interpretabilidad [5].

Sin embargo, los árboles presentan dos problemas principales. En primer lugar, debido a esta simplicidad en la construcción, su capacidad predictiva no es en general la mejor. En segundo lugar, son sensibles al sobreajuste si no son podados. Además, son poco robustos, pequeñas

perturbaciones en los datos implican modificaciones en la estructura del árbol [5]. Sin embargo, mediante la combinación en ensembles estos problemas se pueden paliar.

2.1.3 Redes neuronales artificiales

En contraposición a la sencillez del LDA y de los árboles, existen las redes neuronales. Los primeros modelos, de Warren McCulloch y Walter Pitts [6] datan de 1943. Sin embargo, debido a su compleja estructura, no ha sido hasta esta década cuando han empezado a ganar relevancia por el alto coste computacional que conlleva entrenarlas.

Consideradas por algunos autores como un método de ensemble (originalmente consistían en el encadenado de regresiones logísticas realizadas en pequeñas unidades de cómputo o neuronas), han trascendido esta estructura (i.e. Redes Neuronales Convolucionales, Recurrentes,con distintas funciones de activación a la logística...) para dar cabida a una nueva rama de la inteligencia artificial, el Deep Learning.

Las aplicaciones de las redes neuronales son cada vez más numerosas, especialmente en aprendizaje no estructurado. Algunos ejemplos son *Computer vision*, procesado del lenguaje natural, generación artificial de imágenes... [7] [8]

Sin embargo, a pesar de su alta capacidad predictiva y rango de aplicabilidad, son un modelo muy complejo. Tan complejo que algunos de los modelos más recientes, tienen hasta 2.6 mil millones de parámetros, como *Meena*, el *chatbot* de Google, dinamitando el concepto que era la máxima a seguir en estadística, la parsimonia o Navaja de Ockham, es decir, buscar los modelos más sencillos. Esta complejidad los vuelve, en muchos casos, poco o nada interpretables, por lo cual si nuestro interés es meramente predictivo pueden ser una opción, pero se antojan complicadas a la hora de explicar la realidad, que es para lo que sirve un modelo.

2.1.4 Boosting

Basadas en las ideas de Kearns y Valiant [9] [10] de finales de los años 80, las técnicas de Boosting en el ámbito del Aprendizaje Automático se basan en la idea de aumentar la capacidad predictiva de un algoritmo (meta-algoritmia) mediante la combinación de clasificadores llamados débiles por su pobre capacidad predictiva. La intención tras la combinación de estos clasificadores es una reducción del sesgo y la varianza, dando lugar a un clasificador combinado con mejor capacidad predictiva/propiedades que un clasificador único complejo.

Aunque no hay limitaciones cuando se crea el ensemble de clasificadores, el clasificador base más popular son los árboles de decisión, debido a su fácil interpretabilidad y eficiencia en el entrenamiento. En este Trabajo se estudiarán métodos de Boosting basados en árboles. La construcción de ensembles a grandes rasgos es la siguiente [5]:

1. Sea $\hat{f}(x) = 0$ nuestro predictor inicial.

2. Sea M el número de árboles dentro del ensemble, d el tamaño del árbol, y en el primer paso los residuos $r_i = y_i$, las observaciones.

3. Para cada $m = 1 \dots M$

   - Entrenar un árbol $\hat{f}^m$ de tamaño d con las variables X y la respuesta r (residuos)
   - Actualizar nuestro clasificador $\hat{f}(x) = \hat{f}(x) + \lambda \hat{f}^m$, donde λ es nuestro parámetro de decrecimiento (*shrinkage*)
   - Actualizar los residuos $r_i = r_i - \lambda \hat{f}^m$

4. En $\hat{f}(x)$ tendremos nuestro clasificador final

2.2 Boosting en Árboles de Decisión: AdaBoost

AdaBoost [11] (abreviatura de *Adaptative Boosting*), es un metaalgoritmo basado en técnicas de Boosting. Fue creado por Yoav Freund y Robert Schapire en 2003.
En el caso del AdaBoost, los weak learners serían los llamados tocones (*stumps*), es decir, árboles de una sola disyunción. Combinándolos se obtiene un clasificador complejo conocido como un bosque de tocones (*Forest of Stumps*).

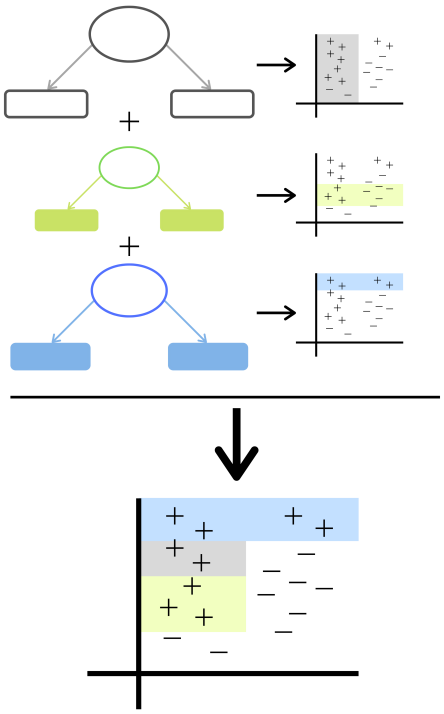


Figura 2.2: Visualización de AdaBoost

La idea detrás de AdaBoost es la de tratar de clasificar con mayor atención las observaciones difíciles. Inicialmente, todas las observaciones $i = 1..n$ tienen un peso $w_i = 1/n$, siendo n el número de observaciones. Estos pesos se van actualizando, incrementándose si la observación es "problemática" a la hora de clasificar. Ya que los pesos se actualizan, llamaremos $w_{i,m}$ al peso de la observación i para inducir el stump m.

Primero, se crea el primer stump t_m (nuestro clasificador débil) al que llamaremos t_0. En la primera iteración todos los pesos $w_{i,0}$ son iguales , así que podemos ignorarlos. Este primer stump será escogido como el que minimice la medida del error que se escojamos. Las más usadas son

- **Gini:** $I_G(p) = 1 - \sum_{i=1}^{J} {p_i}^2$
- **Entropía:** $H(T) = -\sum_{i=1}^{J} p_i \log_2 p_i$

Donde p_i=fraccion de elementos etiquetados con clase i en la hoja.

Sin embargo no todos los stumps tendrán la misma potestad (*amount of say*) a la hora de clasificar una observación, es decir, unos tendrán más voz que otros a la hora de tomar la decisión. Esta potestad vendrá determinada según la capacidad de clasificar que tiene ese stump. Denominaremos el error e_m asociado al stump t_m como la suma de los pesos de las observaciones mal clasificadas E ($e_m = \sum_i^E w_{i,m}$). Estos errores determinan la postestad a la que llamaremos α_m mediante alguno de los siguientes criterios, de tal forma que un e_j bajo implica un α_m alto y viceversa. Hay que señalar que un e_m alto implicaría un α_m negativo. De este modo, si un stump vota por una clasificación k, este α_m negativo hará que su voto pase a ser $No\ k$. Para determinar el *amount of saying*, estas son las tres propuestas principales:

- Breiman $\alpha_m = \frac{1}{2}\log(\frac{1-e_m}{e_m})$
- Freund $\alpha_m = \log(\frac{1-e_m}{e_m})$
- Zhu $\alpha_m = \log(\frac{1-e_m}{e_m}) + \log(k-1)$ (siendo k el número de clases)

Una vez determinado el t_0 y el α_0, pasamos a actualizar los pesos de la muestra, de tal forma que se enfatice la necesidad de clasficar bien las observaciones que t_0 clasificó mal aumentando el $w_{i,1}$ asociado a estas (o disminuirlo si estaba bien clasificada). Una forma de ajustar este peso es la siguiente:

- Aumentar peso: $w'_{i,m+1} = w_{i,m} * e^{\alpha_m}$
- Disminuir peso: $w'_{i,m+1} = w_{i,m} * e^{-\alpha_m}$

Una vez calculados los nuevos pesos para todas las observaciones, hemos de normalizarlos por un factor $\tau_m = \sum_{i=0}^{n} w_{i,m}$. Gracias a estos nuevos pesos, podremos determinar el siguiente stump eligiendo la variable que mejor clasifica usando por ejemplo un indice de Gini Ponderado o hacer un muestreo aleatorio simple con reemplazamiento de tamaño n asociando probabilidad de ser muestreado al peso de la observación, y generar el nuevo stump con esa muestra.

Una vez se ha inducido un Bosque de stumps, la clasificación se realiza del siguiente modo:

1. Para cada stump, ver qué clase asigna a la nueva observación
2. Para los stumps que han asignado la clase k, calcular la suma de los α_m
3. Asignar la clase que tenga la mayor suma de los α_m

2.3 Boosting en Árboles de Decisión: Gradient Boosting

Otro algoritmo del estado del arte que proporciona muy buenos resultados es **Gradient Boosting** (GB). En este TFG se usarán la versiones **XGBoost** [12] y **LightGBM**[13]. Al igual que con AdaBoost, Gradient Boosting se construyen sucesivos árboles que van tratando de predecir los residuos de la predicción de los árboles anteriores. Sin embargo, parte de otra idea.

Como argumenta Friedman[14] en su artículo, que es la guía que se ha utilizado para desarrollar esta sección, la optimización de funciones generalmente se contempla desde un punto de vista de un espacio numérico más que paramétrico. Estos algoritmos de optimización numérica nos invitan a pensar en una estrecha relación entre los algoritmos de descenso por gradiente y los clasificadores creados desde una perspectiva aditiva de funciones (como se puede considerar a los métodos de Boosting ya que son una sucesión acumulada de predicciones de los árboles del ensemble).

Tomando esta aproximación de tratar de llegar a la siguiente solución por métodos de máximo descenso por gradiente, podemos inducir árboles paso a paso basándonos en la solución (árbol) anterior.

2.3.1 Optimización de Funciones

Si tenemos una función $F(x, P)$ donde $\mathbf{P}$ es nuestro conjunto de parámetros, y queremos optimizarla respecto a una función de pérdida $\Phi(\mathbf{P})$, tendremos que encontrar el conjunto de parámetros $\mathbf{P}$ tal que:

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \Phi(\mathbf{P}) \tag{2.2}$$

$$\Phi(\mathbf{P}) = E_{y,\mathbf{x}} L(y, F(\mathbf{x}; \mathbf{P})) \tag{2.3}$$

Obteniendo nuestro óptimo $\mathbf{F}^*$

$$F^*(\mathbf{x}) = F(\mathbf{x}; \mathbf{P}^*) \tag{2.4}$$

Normalmente expresamos $\mathbf{P}^*$ como $\mathbf{P}^* = \sum p_m$, siendo p_m cada uno de los pasos tomados por descenso de gradiente (que se muestra en la siguiente sección).

2.3.2 Descenso de gradiente

Los métodos de descenso de gradiente son métodos de optimización muy populares hoy en día. La idea detrás de ellos es la de ir tomando pasos guiados en la dirección que nos marca el gradiente de la función que queremos optimizar.
Sea nuestra función de pérdida $L(y, F(x; \mathbf{P}))$. Sea $\mathbf{P}_m$ el punto en el que nos encontramos tras m iteraciones ($\mathbf{P}_m = \sum_{i=0}^{m} p_i$, siendo p_i el incremento del paso i). El siguiente paso a dar vendrá indicado por la dirección del gradiente:

$$\mathbf{g}_m = \{g_{jm}\} = \left\{ \left[\frac{\partial \Phi(\mathbf{P})}{\partial P_j} \right]_{\mathbf{P}=\mathbf{P}_{m-1}} \right\} \tag{2.5}$$

donde

$$\mathbf{P}_{m-1} = \sum_{i=0}^{m-1} \mathbf{p}_i \tag{2.6}$$

Entonces el siguiente paso a dar $\mathbf{p}_m$ será

$$\mathrm{p}_m = -\rho_m \mathbf{g}_m \tag{2.7}$$

donde ρ_m nos marca la magnitud de esa dirección

$$\rho_m = \arg\min_{\rho} \Phi\left(\mathbf{P}_{m-1} - \rho \mathbf{g}_m\right) \tag{2.8}$$

F(x) como parámetro

Sin embargo, si consideramos en sí misma a la función $F(x)$ evaluada en el punto x como un parámetro a optimizar, podremos buscar directamente una aproximación aditiva por pasos hacia el óptimo

$$F^*(\mathbf{x}) = \sum f^m(x) \tag{2.9}$$

Siendo $f^m(x)$ el árbol inducido en cada paso m (guiados por la aproximación de descenso de gradiente, buscando el árbol óptimo en cada paso). Por lo que podemos ver que

$$cf_m(\mathbf{x}) = -\rho_m g_m(\mathbf{x}) \tag{2.10}$$

$$g_m(\mathbf{x}) = \left[\frac{\partial \phi(F(\mathbf{x}))}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} = \left[\frac{\partial E_y[L(y, F(\mathbf{x}))|\mathbf{x}]}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{2.11}$$

2.3.3 Gradient Descent con funciones como parámetros

Sin embargo, tenemos que sumar que nuestro espacio de prueba $\mathbf{x}$ está limitado al conjunto de datos que tenemos observados, por lo que el cálculo de esperanzas como en la Ec. 2.3 puede no ser preciso . Podemos restringir nuestra $F(x; \mathbf{P})$ a un conjunto de funciones parametrizado tal que $P \to a, \beta$ como sigue

$$F\left(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M\right) = \sum_{m=1}^{M} \beta_m h\left(\mathbf{x}; \mathbf{a}_m\right) \tag{2.12}$$

Donde $h(x; \mathbf{a})$ es una función simple de entrada x y parámetros $\mathbf{a}$, de tal forma que tendremos un espacio de prueba *suavizado* a partir de nuestros datos observados (que son un espacio discreto de x):

$$F\left(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M\right) = \sum_{m=1}^{M} \beta_m h\left(\mathbf{x}; \mathbf{a}_m\right) \tag{2.13}$$

Entonces nuestros parámetros óptimos α, β se pueden calcular como sigue (reformulando la Ec. 2.2)

$$\{\beta_m, \mathbf{a}_m\}_1^M = \arg \min_{\{\beta'_m, \mathbf{a}'_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta'_m h\left(\mathbf{x}_i; \mathbf{a}'_m\right)\right) \tag{2.14}$$

Esta optimización en la mayoría de situaciones es muy compleja, por lo que se opta por una aproximación greedy "*basada en etapas o stagewise*" según la función F_m obtenida en la etapa anterior

$$(\beta_m, \mathbf{a}_m) = \arg \min_{\beta, \mathbf{a}} \sum_{i=1}^{N} L\left(y_i, F_{m-1}\left(\mathbf{x}_i\right) + \beta h\left(\mathbf{x}_i; \mathbf{a}\right)\right) \tag{2.15}$$

Lo que ya no nos lleva al óptimo $\mathbf{F}^*$ sino a una función que es un óptimo local en cada etapa de la forma

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h\left(\mathbf{x}; \mathbf{a}_m\right) \tag{2.16}$$

A diferencia de las aproximaciones stepwise (como un descenso de gradiente), no se reajustan los términos que ya hemos estimado, sino que se van añadiendo los nuevos cálculos y modelos a un óptimo construido como la adición de todos los óptimos estimados en pasos anteriores.
Pero, ¿cómo encontramos el $h(x; a_m)$? No es sencillo encontrar el conjunto de parámetros que minimice la pérdida L, o la pérdida puede ser dificil de evaluar en F. A partir de (2.15) podemos pensar en que $\beta_m * h(x; a_m)$ es el paso greedy hacia la búsqueda del óptimo F^*. Como vimos

Algorithm 1: Gradient_Boost

1. $F_0(\mathbf{x}) = \arg\min_\rho \sum_{i=1}^{N} L(y_i, \rho)$
2. For $m = 1$ to M do:
3. $\quad \tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \; i = 1, N$
4. $\quad \mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$
5. $\quad \rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$
6. $\quad F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$
7. endFor

end Algorithm

Figura 2.3: Pseudocódigo del algoritmo de boosting como descenso de gradiente

en (2.10), podemos pensar en **menos** descenso de gradiente como (recordemos que podemos considerar $F(x)$ como un parámetro como se comentó en la sección 2.3.1)

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{2.17}$$

Sin embargo como se comenta al principio de esta sección, tenemos el problema de haber observado un subespacio muestral $\mathbf{x}$. Esto nos lleva a ver que g_m solo estará definido en los datos observados. Una posible solución es buscar el $h(x; \mathbf{a})$ que más se parezca (que esté más correlado) con el $-g_m(x)$ contenido en el espacio muestral observado R^N

$$\mathbf{a}_m = \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{N} [-g_m(\mathbf{x}_i) - \beta h(\mathbf{x}_i; \mathbf{a})]^2 \tag{2.18}$$

Seguidamente, podemos calcular el tamaño del paso del gradiente ρ_m

$$\rho_m = \arg\min_\rho \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m)) \tag{2.19}$$

Y finalmente obtenemos el estimador en el paso m como la suma de los anteriores más el nuevo h "regulado" por el tamaño de paso ρ_m

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m) \tag{2.20}$$

Básicamente, se basa en la idea de ajustar el conjunto de parámetros $\mathbf{a}_m$ respecto a los **pseudoresiduales**

$$\{\tilde{y}_i = -g_m(\mathbf{x}_i)\}_{i=1}^{N} \tag{2.21}$$

en vez de como hacíamos 2.14 que surgió por la necesidad de suavizar el espacio discreto observado hacia uno continuo (restringíamos F a ser suave)

2.3.4 Árboles de regresión como algoritmo base

Supongamos que cada algoritmo base que conforma el ensemble es un árbol de regresión. Podemos considerar nuestro algoritmo base h

$$h\left(\mathbf{x};\{b_j,R_j\}_1^J\right)=\sum_{j=1}^{J}b_j1\left(\mathbf{x}\in R_j\right) \tag{2.22}$$

Donde $\{R_j\}_1^J$ son las regiones de los nodos terminales del árbol. La línea 6 del algoritmo de la 2.3 pasa a ser

$$F_m(\mathbf{x})=F_{m-1}(\mathbf{x})+\rho_m\sum_{j=1}^{J}b_{jm}1\left(\mathbf{x}\in R_{jm}\right) \tag{2.23}$$

Pero tenemos que tener en cuenta que las regiones de un árbol de regresión son disjuntas, luego si $x\in R_j$ entonces $h\left(\mathbf{x};\{b_j,R_j\}_1^J\right)=h(x)=b_j$ (lo usaremos más adelante). Si tomamos $\gamma_{jm}=\rho_m b_{jm}$, podemos expresar la ecuación anterior como

$$F_m(\mathbf{x})=F_{m-1}(\mathbf{x})+\sum_{j=1}^{J}\gamma_{jm}1\left(\mathbf{x}\in R_{jm}\right) \tag{2.24}$$

Para mejorar el ajuste, podemos considerar el paso anterior como la suma de J funciones base (una por región)

$$\{\gamma_{jm}\}_1^J=\arg\min_{\{\gamma_j\}_1^j}\sum_{i=1}^{N}L\left(y_i,F_{m-1}\left(\mathbf{x}_i\right)+\sum_{j=1}^{J}\gamma_j1\left(\mathbf{x}\in R_{jm}\right)\right) \tag{2.25}$$

Teniendo en cuenta la naturaleza disjunta de estas regiones (y por ende las funciones son independientes a la hora de optimizar), y aplicando el resultado que vimos sobre $h(x)=b_j$ si $x\in R_j$, podemos expresar esta optimización como

$$\gamma_{jm}=\arg\min_{\gamma}\sum_{x_i\in R_{jm}}L\left(y_i,F_{m-1}\left(x_i\right)+\gamma\right) \tag{2.26}$$

2.3.5 Clasificación mediante árboles y Regresión logística multiclase

Si tenemos varias clases (como es el caso de nuestro problema de motores) podemos basarnos en árboles de regresión que realicen la regresión logística, y expresar nuestra función de pérdida como la logistic-loss multiclase, teniendo en cuenta que en cada paso se estimará un árbol para cada clase K $F_{mk}(x)$ (creándose así un ensemble de grupos de árboles, quedando en cada grupo

cada clase predicha por un árbol frente al resto)

$$L\left(\{y_k, F_k(\mathbf{x})\}_1^K\right) = -\sum_{k=1}^{K} y_k \log p_k(\mathbf{x}) \tag{2.27}$$

Donde $y_k = 1(clase = k) \in \{0, 1\}$ y $p_k(\mathbf{x}) = \Pr(y_k = 1|\mathbf{x})$
Podemos formular nuestra $F(x)$ para la regresión logística con K clases en función de la probabilidad de las clases

$$F_k(\mathbf{x}) = \log p_k(\mathbf{x}) - \frac{1}{K}\sum_{l=1}^{K} \log p_l(\mathbf{x}) \tag{2.28}$$

O bien invertir esta igualdad y expresar las probabilidades en función de F

$$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) \,/\, \sum_{l=1}^{K} \exp(F_l(\mathbf{x})) \tag{2.29}$$

Una vez hecho esto, podemos expresar los pseudoresiduales (2.21) como sigue en función de $p_k(x)$

$$\tilde{y}_{ik} = -\left[\frac{\partial L\left(\{y_{il}, F_l(\mathbf{x}_i)\}_{l=1}^K\right)}{\partial F_k(\mathbf{x}_i)}\right]_{\{F_l(\mathbf{x})=F_{l,m-1}(\mathbf{x})\}_{\mathbf{F}}^K} = y_{ik} - p_{k,m-1}(\mathbf{x}_i) \tag{2.30}$$

Por tanto podemos buscar los γ_{jmk} asociados a cada clase aplicado a la regresión logística, siendo $\phi(y_k, F_k) = -y_k * log(p_k(x))$

$$\{\gamma_{jkm}\} = \arg\min_{\{\gamma_j\}_k} \sum_{i=1}^{N}\sum_{k=1}^{K} \phi\left(y_{ik}, F_{k,m-1}(\mathbf{x}_i) + \sum_{j=1}^{J} \gamma_{jk} 1(\mathbf{x}_i \in R_{jm})\right) \tag{2.31}$$

Esta ecuación no tiene solución directa. No se va a demostrar porque excede los contenidos de este TFG pero se puede aproximar por una iteración del algoritmo de Newton-Raphson mediante una aproximación diagonal del Hessiano, descomponiéndose así el problema en uno separado para cada hoja de cada árbol (R_{jkm})

$$\gamma_{jkm} = \frac{K-1}{K}\frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}|\,(1 - |\tilde{y}_{ik}|)} \tag{2.32}$$

Podemos entonces reformular el algoritmo de la Figura 2.3 al de la 2.4. Una vez hemos obtenido los estimadores finales en el paso M para cada clase $\{F_{kM}(x)\}_1^K$ podremos obtener los estimadores de las $p_k(x)$ con la Ec. 2.29

Algorithm 6: L_K-TreeBoost

$F_{k0}(\mathbf{x}) = 0, \quad k = 1, K$

For $m = 1$ to M do:

$p_k(\mathbf{x}) = \exp(F_k(\mathbf{x})) / \sum_{l=1}^{K} \exp(F_l(\mathbf{x})), \quad k = 1, K$

For $k = 1$ to K do:

$\tilde{y}_{ik} = y_{ik} - p_k(\mathbf{x}_i), \quad i = 1, N$

$\{R_{jkm}\}_{j=1}^{J} = J$–terminal node $tree(\{\tilde{y}_{ik}, \mathbf{x}_i\}_1^N)$

$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{\mathbf{x}_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{\mathbf{x}_i \in R_{jkm}} |\tilde{y}_{ik}| (1-|\tilde{y}_{ik}|)}, \quad j = 1, J$

$F_{km}(\mathbf{x}) = F_{k,m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jkm} 1(\mathbf{x} \in R_{jkm})$

endFor

endFor

end Algorithm

Figura 2.4: Gradient Boosting en clasificación multiclase

2.4 XGBoost: una generalización del Gradient Boosting

XGBoost [12] es una implementación y generalización del Gradient Boosting. Su flexibilidad, capacidad computacional y predictiva le han proporcionado una posición hegemónica en las competiciones de análisis de datos.
Entre las ventajas de XGBoost frente al GB tradicional se encuentran:

- **Regularización**: A diferencia de las Gradient Boosting Machines tradicionales, XGBoost contempla la opción de añadir regularización Ridge (L1) y Lasso (L2) a la función objetivo. Con esto se consigue introducir una selección de variables que ayuda a paliar el sobreajuste en caso de existir.

- **Procesamiento Paralelo**: Puede sonar paradójico, el gradient boosting se construye sobre el árbol del paso anterior, luego ¿cómo se puede paralelizar algo que tiene naturaleza necesariamente paralela? La paralelización se realiza a nivel de nodo, de tal forma que se dos caminos independientes en el árbol serán procesados de forma paralela.

- **Flexibilidad**: Permite jugar con la función objetivo/ de pérdida (acepta funciones propias) que se quiere optimizar

- **Poda de Árboles**: A diferencia de las GBM que paran de construir el árbol una vez encuentran una pérdida negativa, XGBoost crea el árbol hasta la profundidad indicada. Una vez creado, poda aquellas hojas que tengan pérdidas negativas, lo que flexibiliza la estructura de los árboles.

- **Gestión de datos dispersos**: Provee una implementación eficiente para datos con muchos *missing*, aunque en nuestro conjunto de datos no hay luego esta propiedad no la aprovecharemos.

- **Integración con scikit-learn**: Si usamos la versión de python, podremos integrarlo con la librería **sklearn**, la más popular para aprendizaje automático actualmente.
- **Restricciones isotónicas**: En secciones posteriores se estudiará el boosting isotónico, el cual está integrado en XGBoost, tanto en su versión para R como para Python.

2.4.1 Regularización

Como vimos en la ecuación 2.26, nuestra función de pérdida era la logística. Podemos añadir un término $\Omega(f)$ para la regularización

$$L(\phi) = \sum_i l\left(\hat{y}_i, y_i\right) + \sum_m \Omega\left(f_m\right) \text{ donde } \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2 \tag{2.33}$$

Donde T es el número de hojas en el árbol y w son los pesos de la regresión en cada hoja aprendidos por el árbol nuevo. Este término añadido de regularización forzará a reducir los valores de T y w, llevando el árbol a ser más pequeño y a cometer un menor sobreajuste. Si ambos parámetros valen 0, volvemos al caso sin regularización.

2.4.2 Muestreo de variables y observaciones

Una de las técnicas más utilizadas en **Bagging** [15] es el muestreo de columnas. En cada iteración, al árbol de esa iteración no se le proporcionan todas las variables sino que solo se le proporciona un subconjunto aleatorio de estas. Podemos hacer lo mismo con las observaciones, de tal forma que a cada árbol se le está presentando un escenario distinto de clasificación. Con esto, estamos introduciendo diversidad en el ensemble, combatiendo así el sobreajuste al estar contemplando subcasos diversos.

2.4.3 Decrecimiento

XGBoost además introduce una técnica parecida a la que utiliza Friedman en su artículo *Stochastic Gradient Boosting* [16] y muy similar a la que implementa AdaBoost. Si introducimos un parámetro η de decrecimiento multiplicando la respuesta de los sucesivos árboles (es decir, estamos *reduciendo* el impacto de su respuesta en la global) se ha demostrado empiricamente [16] que valores bajos de η proporcionan mejores tasas de error.
De hecho, con la pérdida logística, tamaño máximo de árbol de 1 (stump) y con $\eta = 1/2 * log(\frac{1-e_j}{e_j})$ se puede ver que AdaBoost es una particularización del Gradient Boosting

2.5 LightGBM: El principal rival de XGBoost

LightGBM [13] es la otra gran alternativa frente a XGBoost. De la mano de Microsoft, promete ser una librería que implementa gradient boosting de forma distribuida, precisa y mucho más rápida que las opciones existentes.

Comparte las mismas funcionalidades que XGBoost implementando nuevas técnicas que agilizan el procesamiento.
Según los autores [13], la principal razón es que para cada variable, los frameworks existentes escanean todas las instancias para estimar la ganancia de información y así determinar los puntos de división o *split* en cada nodo, lo cual es una tarea computacionalmente pesada y lenta. LightGBM introduce dos técnicas novedosas: **Gradient-based One-Side Sampling (GOSS)** para combatir el problema por la parte de las observaciones y **Exclusive Feature Bundling (EFB)** para combatir el problema por la parte del número de variables.

2.5.1 Gradient-based One-Side Sampling

La siguiente exposición se basa en el algoritmo propuesto por [13]. Como hemos visto antes, en AdaBoost se tiene en cuenta la importancia de cada observación a la hora de estimar los árboles, siendo el peso de la observación un indicador de esta importancia.
Sin embargo, en gradient boosting no tenemos un indicador de esa importancia que nos permitía remuestrear con reemplazamiento las observaciones para así enfatizar la clasificación de las *problemáticas*. Sin embargo, el gradiente aportado por cada observación en Gradient Boosting también informa sobre esta importancia. Si el gradiente asociado a una observación es pequeño, esto quiere decir que la observación ya está bien clasificada. Si eliminásemos las observaciones con un gradiente bajo estaríamos modificando el conjunto de datos, empeorando así la clasificación del mismo. Aquí es donde entra en juego **GOSS**.
GOSS mantiene las observaciones con gradiente alto y realiza un muestreo sobre las observaciones con gradiente bajo. Para compensar esta modificación en el conjunto de datos, cuando se calcula la ganancia de información, GOSS introduce una constante que multiplica las muestras con gradiente pequeño del siguiente modo.
GOSS ordena los datos en funcion del valor absoluto de su gradiente y selecciona el $ax100\%$. Seguidamente, muestrea de forma aleatoria sobre las restantes un $bx100\%$. Después de esto, a la hora de calcular la ganancia de información, GOSS multiplica estas instancias muestreadas por una constante $k = \frac{1-a}{b}$, de tal forma que se enfatizará la clasificación en las peor clasificadas (gradiente alto) mientras que gracias a la constante, la distribución de los datos no cambiará en exceso al multiplicar las que tengan un error de clasificación bajo (gradiente bajo).

2.5.2 Exclusive Feature Bundling

Debido a que los datos de alta dimensión suelen tener una estructura dispersa, LightGBM aprovechándose de esto propone juntar varias variables que sean excluyentes (i.e. por ejemplo cuando casi nunca toman valor 0 al mismo tiempo, cosa que se puede dar al codificar con dummies) en un solo paquete o *bundle*. De este paquete, se extrae una sola variable fruto de la combinación de las anteriores sin pérdida de información. De este modo, la complejidad del problema disminuye de $O(|Observaciones|x|Variables|)$ to $O(|Datos|x|Bundle|)$, siempre

que $|Bundle| << |Variables|$. Con esta combinación de variables excluyentes se acelera el entrenamiento del ensemble.
Se puede leer el proceso en el documento original del algoritmo [13]. En este TFG no tenemos problema de datos dispersos, así que esta característica se menciona de forma exclusivamente bibliográfica.

2.6 XGBoost DART: Dropout en Árboles de decisión

DART (Dropout in multiple Aditive Regression Trees)[17] consiste en una técnica de supresión de árboles del ensemble con el objetivo de paliar el sobreajuste. Esta técnica de regularización surgió en las redes neuronales, de tal forma que se "bloquean" neuronas de forma aleatoria en el entrenamiento para evitar que alguna neurona de las primeras capas sea demasiado influyente en las capas finales, lo que provocaría que la red sobreajuste a los ejemplos del conjunto de entrenamiento. Esta técnica fue patentada por Google en 2005 y ha demostrado ser efectiva en muchos modelos [18]
En el caso de los árboles de decisión, los autores proponen, en cada iteración del descenso de gradiente no se tiene en cuenta todo el conjunto de árboles entrenado hasta la iteración actual, sino que solo se considera un subconjunto de árboles muestreado aleatoriamente sin reemplazamiento de los ya entrenados. Con este subconjunto se calcula el siguiente paso del descenso de gradiente , es decir, el nuevo árbol solo es influenciado por un subconjunto de sus antecesores, provocando así una situación similar al dropout en redes neuronales. XGBoost trae este booster incorporado como parámetro en su paquete de Python.

2.7 Boosting basado en regresión logística: LogitBoost

En el capítulo 5 veremos métodos de boosting que incorporan información externa al modelo en forma de restricciones isotónicas. Algunos de los métodos allí propuestos se basan e **LogitBoost** [19].
Al igual que los algoritmos de boosting basados en árboles antes expuestos, se construye un modelo aditivo $F(x) = \sum_{m=1}^{M} f_m(x)$. Sin embargo ahora f_m no es un árbol de decisión sino una regresión logística, que para el problema de dos clases se puede formular como sigue siendo $\pi(x) = P(y = 1|x)$:

$$f(x) = \log\left(\frac{\pi(x)}{1 - \pi(x))}\right) \tag{2.34}$$

El algoritmo de construcción del modelo de LogitBoost consiste en los siguientes pasos:

1. $w_i = \frac{1}{n}, \pi(x) = \frac{1}{2}\ \forall x_i \in D$,$F(x) = 0$
2. Para $m = 1, ..., M$:

(a) Calcular los nuevos $w_i = \pi(x)(1 - \pi(x))$y los pseudoresiduales $z_i = \frac{y_i^* - \pi(x)}{w_i}$

(b) Ajustar con una regresión ponderada $f_m(x)$ sobre los z_i usando los pesos w_i.En las versiones más habituales de LogitBoost, $f_m(x)$ se basa en un solo predictor para cada paso.

(c) Actualizar $F(x) = f_m(x)$ y $\pi(x) = \frac{1}{1+e^{-F(x)}}$

3. Clasificar como 1 si $\pi(x) > 0.5$, 0 en caso contrario.

Durante este proceso iterativo se añadirán restricciones isotónicas que darán lugar a los algoritmos que compararemos en el capítulo 5 junto a los ya estudiados.

2.8 Interpretabilidad de modelos complejos

Los modelos de alta complejidad como los ensembles o las redes neuronales poseen una alta capacidad predictiva, pero su interpretabilidad es muy compleja o directamente nula.
Para ello, surgen alternativas que tratan de estudiar cómo cambios en las variables explicativas alteran la predicción para la varible respuesta. A continuación se muestran dos de las alternativas más populares hoy en día, **LIME** y **SHAP**. En este capítulo se muestra una transcripción y explicación del capítulo 5 del libro de Cristoph Molnar [20] y de los papers originales de LIME [21] y de SHAP [22]

2.8.1 LIME

Según los creadores de Local Interpretable Model-Agnostic Explanations o **LIME** [21], un *explicador* de modelos ideal tendría las siguientes propiedades:

- **Interpretabilidad:** Debería proporcionar una relación cuantitativa entre las variables explicativas y la respuesta.
- **Fidelidad Local:** Una interpretabilidad puede que no sea fidedigna para todo el modelo, pero al menos ha de serlo de forma local.
- **Independiente del Modelo:** El interpretador debería ser independiente del modelo.
- **Perspectiva global:** El interpretador debería mostrar información representativa para el usuario, de tal forma que el usuario tenga una intuición global del modelo.

Comenzamos explicando **LIME**, el cual permite interpretar el cómo se clasificó una observación individual x.

Compromiso Fidelidad-Interpretabilidad y Fidelidad

Sea $g \in G$ un posible modelo interpretable de todos los modelos interpretables G (regresión, árboles de decisión...). Definamos la complejidad de ese modelo como $\Omega(g)$, y la fiabilidad de ese clasificador respecto a otro clasificador f que no es interpretable como $\mathcal{L}(f, g, \pi_x$, donde $\pi_x(z)$ es la métrica de proximidad de una observación z respecto a la observación x de la que queremos explicar o interpretar su predicción. Podemos definir una función objetivo para encontrar ese interpretador g:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}\left(f, g, \pi_x\right) + \Omega(g) \tag{2.35}$$

De este modo, evitaremos que la complejidad Ω se dispare mientras que se maximizará la fidelidad (también llamada de pérdida) $\mathcal{L}$. Sin embargo, esta función es muy difícil de encontrar y de optimizar. En la práctica, la complejidad del modelo es fijada por el usuario, optimizando solo la función de fidelidad.

Muestreo para exploración local

Para entrenar el modelo que será nuestro interpretador local de la clasificación de nuestra observación $x \in \mathbb{R}^d$, utilizaremos la observación interpretable x' (Debido a que x puede requerir una transormación de la entrada en problemas como análisis de textos o imágenes, pudiendo representar estas como un vector binario). En nuestro caso, por ser datos tabulares, las observaciones son directamente interpretables. En la Figura 2.5 se muestra el algoritmo que proponen los autores cuando g es un clasificador *Lasso*

Algorithm 1 Sparse Linear Explanations using LIME

Require: Classifier f, Number of samples N
Require: Instance x, and its interpretable version x'
Require: Similarity kernel π_x, Length of explanation K
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ **do**
    $z'_i \leftarrow sample\_around(x')$
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso$(\mathcal{Z}, K)$ ▷ with z'_i as features, $f(z)$ as target
  return w

Figura 2.5: Algoritmo de LIME

La idea del algoritmo es entrenar con una muestra generada entorno a x'. Una vez seleccionado el subconjunto d' de variables a estudiar (en nuestro caso todas, tal que $x' = x$) , se perturba

esta observación (en nuestro caso, añadiendo ruido gaussiano a x') dando lugar al conjunto Z'. Reconstruyendo a partir de x, obtenemos el conjunto $\mathcal{Z}$ (en nuestro caso, por ser datos tabulares, $Z' = \mathcal{Z}$) con nuestra muestra generada, y con la que entrenaremos el clasificador g ponderando según π_x.
Los autores proponen como métrica de proximidad π_x un kernel exponencial sobre una función de distancia D con amplitud σ :

$$\pi_x(z) = exp(\frac{-D(x,z)^2}{\sigma^2}) \tag{2.36}$$

Y como función de pérdida a optimizar, el ECM ponderado mediante π_x:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z) * (f(z) - g(z'))^2 \tag{2.37}$$

En la 2.6 obtenida de [21], a modo ilustrativo de LIME vemos en azul y rosa las zonas de clasificación del modelo a interpretar f. En el símbolo + de color rojo intenso la observación x', y alrededor de ella las observaciones z.

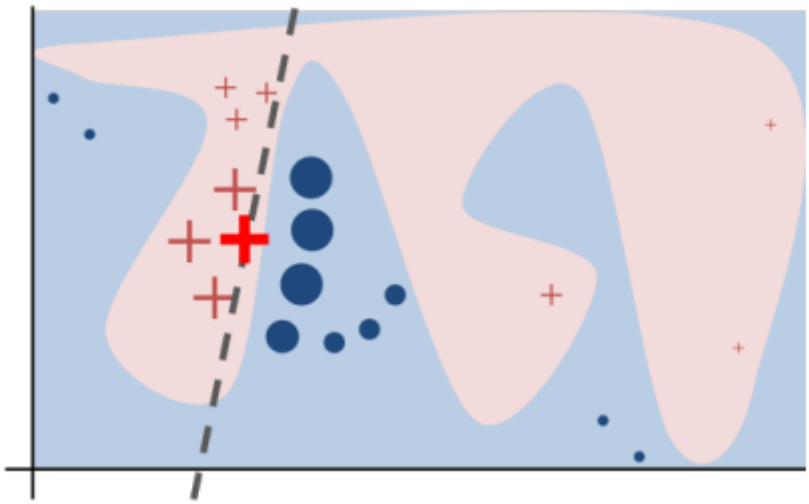


Figura 2.6: Ejemplo de LIME

2.8.2 Shapley Values

Antes de explicar SHAP, hay que explicar los **Shapley Values** [23]. Nacidos de la teoría de juegos, se basan en la idea de que una predicción se puede explicar de tal forma que cada variable es un jugador, en la que el premio u objetivo es la predicción. Los Shapley values nos dicen cómo distribuir la predicción de forma "justa" entre los jugadores o variables.
Supongamos que tenemos tres variables explicativas, X_1, X_2, X_3 y una predicción R. Una posible explicación de cómo afectó cada variable a la predicción es que todas contribuyeron con contribuciones ϕ_j, de tal forma que $\phi_1 + \phi_2 + \phi_3 = R$. Si nuestro modelo fuese lineal,

las contribuciones ϕ_j de la variable X_j a la predicción de la observación x_i podrían entenderse como la diferencia entre la predicción y el valor esperado para la predicción en esa variable:

$$\phi_j(\hat{f}) = \beta_j x_{ij} - E(\beta_j X_j) = \beta_j x_{ij} - \beta_j E(X_j) \tag{2.38}$$

Sin embargo generalmente no se tiene una interpretabilidad directa para modelos complejos como ensembles o redes neuronales. La forma en la que se calculan los **Shapley Values** para una variable es mediante la media de las contribuciones marginales de esa variable a todas las posibles **coaliciones de variables**.

Una coalición de variables no es más que un subconjunto de variables $S \subseteq F$ donde F es el conjunto de todas las variables del modelo sin la variable de la que queremos estudiar su contribución. Supongamos que queremos estudiar la contribución de X_3 a la respuesta. En nuestro ejemplo, las posibles coaliciones son:

- $\emptyset$
- $\{X_1\}$ $\{X_2\}$
- $\{X_1, X_2\}$

Para cada una de estas coaliciones calcularemos el promedio de las diferencias de las predicciones marginales para el subconjunto S con y sin la variable de interés de forma ponderada, es decir:

$$\phi_j = \sum_{S \subset (X_1,\ldots,X_p)\setminus(X_j)} \frac{|S|!(p-|S|-1)!}{p!} val(S \cup \{X_j\}) - val(S)) \tag{2.39}$$

Para calcular la predicción marginalizada:

$$val_x(S) = \int \hat{f}(X_1, \ldots, X_p) d\mathbb{P}_{X \notin S} - E_X(\hat{f}(X)) \tag{2.40}$$

Por ejemplo, en nuestro caso para $S = \{X_1 = x_1\}$.

$$val_x(S) = val_x(\{X_1\}) = \int_{\mathbb{R}} \hat{f}(x_1, X_2, X_3) d\mathbb{P}_{X_2,X_3} - E_X(\hat{f}(X)) \tag{2.41}$$

Si hubiese otra variable sobre la que calcular la integral, volveríamos a integrar sobre esa variable. En la práctica, hay propuestas como métodos de Montecarlo [24], en la que, variable a variable, se van simulando puntos utilizando una mezcla de la observación x' y otra z' de forma aleatoria, obteniendo el valor de la variable de interés X_j de x' o de z' según interese e iterando. De forma resumida, se itera M veces, generando x^m_{+j} (observación con la variable j) y x^m_{-j} (observación sin ella)

- $x^m_{+j} = (x_{(1)}, \ldots, x_{(j-1)}, x^m_{(j)}, z^m_{(j+1)}, \ldots, z^m_{(p)})$
- $x^m_{-j} = (x_{(1)}, \ldots, x_{(j-1)}, z^m_{(j)}, z^m_{(j+1)}, \ldots, z^m_{(p)})$

En cada iteración se obtiene $\phi_j^m = \hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m)$. Promediando, obtenemos el estimador de Shapley:

$$\phi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^m \tag{2.42}$$

Propiedades

Los Shapley values cumplen las siguientes propiedades que en teoría de juegos se consideran requisitos para hablar de un pago justo o equitativo a los jugadores:

- **Eficiencia**: La suma de las contribuciones es la diferencia entre el valor esperado y la predicción

$$\sum\nolimits_{j=1}^{p} \phi_j = \hat{f}(x) - E_X(\hat{f}(\mathbf{X})) \tag{2.43}$$

- **Simetría**: Si dos variables contribuyen de igual forma a todas sus posbiles coaliciones, sus contribuciones son equivalentes

$$\forall j, k \text{ tal que } val(S \cup \{x_j\}) = val(S \cup \{x_k\}), S \subseteq \{x_1, \ldots, x_p\} \setminus \{x_j, x_k\} \text{ entonces } \phi_j = \phi_k \tag{2.44}$$

- **Dummy**: Si una variable no cambia la predicción, su contribución debe ser cero

$$\forall j \text{ tal que } val(S \cup \{x_j\}) = val(S), S \subseteq \{x_1, \ldots, x_p\} \text{ entonces } \phi_j = 0 \tag{2.45}$$

- **Aditividad**: Si el juego tiene pagos o recompensas p^1, p^2, la contribución de una variable es la suma de las contribuciones a cada pago $\phi_j = \phi_j^1 + \phi_j^2$. Esta propiedad es muy interesante en nuestro caso, ya que en nuestro problema de ensembles, cada ensemble tiene un pago distinto a repartir entre las variables.

Ventajas

- La contribución, a diferencia de **LIME**, se reparte más homogéneamente entre todas las variables.
- A la hora de marginalizar para estudiar x, puedes elegir sobre qué subconjunto de los datos hacerlo, a diferencia de **LIME** que genera un subcojunto localmente mediante perturbaciones.
- Se basa en una teoría sólida y contrastada como la teoría de juegos. **LIME** asume linealidad local cosa que no tiene que suceder.

Desventajas

- **Computacionalmente exhaustivo**: Tenemos 2^p posibles coaliciones de variables.
- **Interpretabilidad**:La interpretación debe ser la siguiente: Dados los valores observados para x, cuál fue la contribución de las variables **en la diferencia** entre la predicción para x y la predicción promedio.
- **Necesidad de los datos**: A diferencia de **LIME**, no basta con tener solo la predicción.
- **No proporciona un modelo**: No se puede elucubrar sobre qué efectos tendrían cambios en las variables en la predicción.
- **Poco robusto frente a outliers**.

2.8.3 Shapley Additive Explanations: SHAP

SHAP [22] es una extensión de los Shapley Values. Como vimos, un *jugador* podía ser una variable pero también podemos considerar grupos de variables, como hace LIME. Esto aúna ambos métodos. SHAP especifica el modelo explicativo como sigue:

$$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j \tag{2.46}$$

En este modelo, los Shapley Values ϕ_j son los coeficientes del modelo LIME, aunando así ambas metodologías.
A diferencia de como vimos en la sección 2.8.1, x y x' ya no son equivalentes para datos tabulares. Ahora, $x' = \{1\}^P$, es decir, x' es un vector de 1 representando una coalición de todas las variables, siendo coalición un conjunto de variables en las que se basaban los Shapley Values. Muestrear entorno a x' como hacíamos en LIME consistirá en muestrear coaliciones de variables z'. También necesitamos una función h_x para *reconstruir* la muestra original a partir de las observaciones z', es decir:

$$h_x \text{ tal que } h_x(z') = z \tag{2.47}$$

A la hora de *reconstruir* z, lo haremos de la siguiente forma. En las variables para las que $z'_j = 1$, recuperaremos el valor de la variable j que tomó x. En las que no, tomaremos un valor aleatorio observado para esa variable. Este muestreo es sobre la marginal de las variables presentes (integrando sobre las ausentes), es decir, basándonos en la idea de que, asumiendo independencia de las variables:

$$f(h_x(z')) = E(f(z)|z_S) = E_{z_{\overline{S}}|z_S}(f(z)) \approx E_{z_{\overline{S}}}(f(z)) \approx f(z_S, E(z_{\overline{S}})) \tag{2.48}$$

Siendo S el conjunto de variables X_j tal que $z_j = 1$.

Propiedades de SHAP

Aparte de las propiedades de los Shapley values, las cuales cumple porque recordemos que estamos calculando estos, SHAP añade las siguientes propiedades:

1. **Fidelidad local**: Si tomamos $\phi_0 = E_X(\hat{f}(x))$, la Eq. 2.46 es equivalente a la propiedad de eficiencia 2.43 de los Shapleys

2. **Variables ausentes implican coeficientes nulos**. Es decir, que si $x'_j = 0$,$\phi_j = 0$. En la práctica se traduce en que variables constantes tienen Shapleys nulos.

3. **Consistencia**: Sea $z' \backslash j$ la observación z' con $z'_j = 0$. Supongamos que $f_x(z') = f(h_x(z'))$. Esta propiedad nos indica que si se cumple que para dos modelos f' y f'

$$f'_x(z') - f'_x(z'_{\backslash j}) \geq f_x(z') - f_x(z'_{\backslash j}) \tag{2.49}$$

   Para todos los $z' \in \{0,1\}^P$, entonces

$$\phi_j(f', x) \geq \phi_j(f, x) \tag{2.50}$$

   Esta propiedad indica que si un modelo cambia de tal forma que la contribución marginal de una variable crece o se mantiene, el valor shapley ϕ_j crece o se mantiene en la misma medida.

Los autores de SHAP proponen dos métodos para estimar los Shapley Values vía SHAP, KernelSHAP[22] y TreeSHAP [25], una versión más eficiente para modelos basados en árboles. A continuación explicaré el primero a modo ilustrativo de cómo funciona SHAP. TreeSHAP, debido a su alta complejidad no se desarrollará pero sí será empleado en secciones posteriores.

Kernel SHAP

Para cada observación x, computamos los Shapley Values de cada variable como sigue:

1. Muestrear coaliciones $z'_k \in \{0,1\}^P, k = 1..K$

2. Calcular $f(h_x(z))$

3. Calcular los pesos con el kernel de SHAP.

4. Ajustar un modelo lineal local ponderado como hacíamos en LIME

5. Los coeficientes β_j de ese modelo serán los Shapley values ϕ_j

La diferencia con LIME está en el kernel que define los pesos según la proximidad. En el caso de SHAP, se da más peso a las coaliciones z' que sean muy representativas; con muchas variables

(se verá bien el efecto de las pocas variables no presentes) o pocas (se verá bien el efecto de las pocas variables presentes). El kernel es el siguiente:

$$\pi_x(z') = \frac{(P-1)}{\begin{pmatrix} P \\ |z'| \end{pmatrix} |z'| (P - |z'|)} \tag{2.51}$$

Para muestrear las coaliciones, no se hace de forma aleatoria. Es más representativo muestrear, en primer lugar coaliciones con un solo 0 o con un solo 1 (hay un total de P-2). Seguidamente, con dos 0 o con dos 1, así hasta tener las K observaciones/coaliciones z', y entrenamos un modelo lineal optimizando el ECM ponderado que vimos en la ecuación 2.37:

$$g(z') = \phi_0 + \sum_{j=1}^{P} \phi_j z'_j \tag{2.52}$$

Obteniendo así los Shapley values de **SHAP** en los ϕ_j del modelo.

Sobre TreeSHAP:

El problema de Kernel Shap es que, para modelos de ensemble, la complejidad es de $O(TL2^P)$, donde T es el número de árboles, L el máximo número de hojas. Los autores propusieron TreeSHAP [25] como una alternativa para modelos de ensembles en la que la complejidad se reducía a $O(TLD^2)$, donde D es la profundidad máxima del árbol. El problema es, a diferencia de como vimos en KernelSHAP es que TreeSHAP muestrea sobre distribuciones condicionales en vez de marginales como hacíamos en la Ec. 2.48, y eso provoca que no estemos calculando exactamente valores Shapley. Además, el hecho de muestrear así puede provocar que variables irrelevantes tomen valores Shapley no nulos.

Sin embargo en la implementación práctica de la librería `shap` utilizan marginales. Además, a la hora de muestrear los valores de las variables que no están en la coalición, permite utilizar un conjunto auxiliar (que no disponemos) en la aproximación *true to the data*, o muestrear los valores en función de los splits y las hojas del modelo en la aproximación *true to the model*. Utilizaremos la segunda ya que queremos explicar el modelo. Además, debido a su alta eficiencia, utilizaremos TreeSHAP al estar estudiando modelos de ensemble aunque sea menos exacto.

Ventajas de SHAP

1. Todas las de los Shapley values
2. Aúna LIME y Shapley
3. Versiones mucho más rápidas de cómputo que con los Shapleys para ensembles de árboles

Desventajas

1. KernelSHAP es lento y asume independencia de las variables.

2. TreeSHAP puede dar valores erróneos de Shapleys para variables no relevantes por el muestreo condicional

2.9 Otras cuestiones metodológicas

Métricas del error

Dos métricas se tendrán en cuenta a la hora de evaluar clasificadores:

- **Error de clasificación**: $e(y, \hat{y}) = \frac{\sum_{i=0}^{n} I_{y_i \neq \hat{y}_i}}{n}$
- **Mean absolute error**: $MAE(y, \hat{y}) = \frac{\sum_{i=0}^{n} |y_i - \hat{y}_i|}{n}$

La ventaja del MAE sobre el Error de clasificación es el hecho de que no penaliza tanto el equivocarse en clases adyacentes como lo hace el error de clasificación, que considera todos los fallos como equivalentes.

División Train/Test

Cuando estimamos un modelo, lo hacemos con la intención de explicar la realidad. Para entrenar modelos, empleamos datos sobre la realidad que queremos explicar. Pero los datos que tenemos son una observación parcial de la realidad, no se puede obtener todo el conjunto de escenarios posibles. De hecho, cuando entrenamos un modelo, explicamos una respuesta en función de lo **observado**. Si evaluamos el rendimiento del modelo en los datos con los que lo hemos entrenado, podemos cometer el error de pensar que nuestro modelo es una buena aproximación a la realidad porque tiene un buen rendimiento en los datos que ha conocido en el entrenamiento. Sin embargo, podemos estar cometiendo un problema de **sobreajuste**, es decir, nuestro modelo ajusta muy bien los datos con los que ha entrenado pero a la hora de generalizar y probar casos que no conoció en el subconjunto de entrenamiento o **train**, nos proporciona un rendimiento paupérrimo.
Para evitar caer en la tentación de juzgar la capacidad predictiva de un modelo, debemos dividir nuestro conjunto de datos en 2 partes, **train** y **test**. Con **train**, entrenaremos el modelo y con **test**, conjunto de datos que el modelo no ve durante el entrenamiento, juzgaremos la capacidad predictiva real del modelo.

2.9.1 Validación cruzada

Sin embargo, el estimar la capacidad de predicción (en nuestro caso la tasa de error **e**) de un modelo únicamente con una prueba nos daría un estimador del error con mucha variabilidad, que además es sensible a la partición train/test que estemos haciendo.
Si dividimos nuestro conjunto de datos en k particiones independientes podremos entrenar con $k-1$ particiones que harán de train y la restante de test. De este modo, iterando hasta utilizar

cada partición como test, y promediando la tasa de errores, tendremos un estimador del error con menor varianza por ser el promedio de varias observaciones. Este método se conoce como validación cruzada (conocido como *cross validation*, XV)

2.9.2 Validación cruzada e hiperparámetros

Como hemos visto, mediante la validación cruzada podemos obtener una tasa del error realista. Gracias a esto, podemos probar conjuntos de hiperparámetros y compararlos de forma consistente, quedándonos con los mejores hiperparámetros. La validación cruzada proporciona una tasa de error con menor varianza, de tal forma que al repetir 5 veces el experimento, nos aseguramos que una tasa de error es buena en distintos escenarios y no ha sido buena de casualidad por la división train/test.
Sin embargo, el estimador del error obtenido mediante validación cruzada es más optimista que la tasa de error real. Es por esto que reservaremos una porción de los datos para una partición extra llamada *test*, que nunca se verá hasta que, mediante la validación cruzada sobre el train split, haya encontrado el mejor conjunto de hiperparámetros. Una vez obtenido ese mejor modelo con los mejores hiperparámetros, decidido mediante XV, evaluaremos sobre la partición *test* para obtener la tasa de error de generalización, un estimador más realista (ya que son datos que nunca ha visto, que es lo que pasará en el futuro).

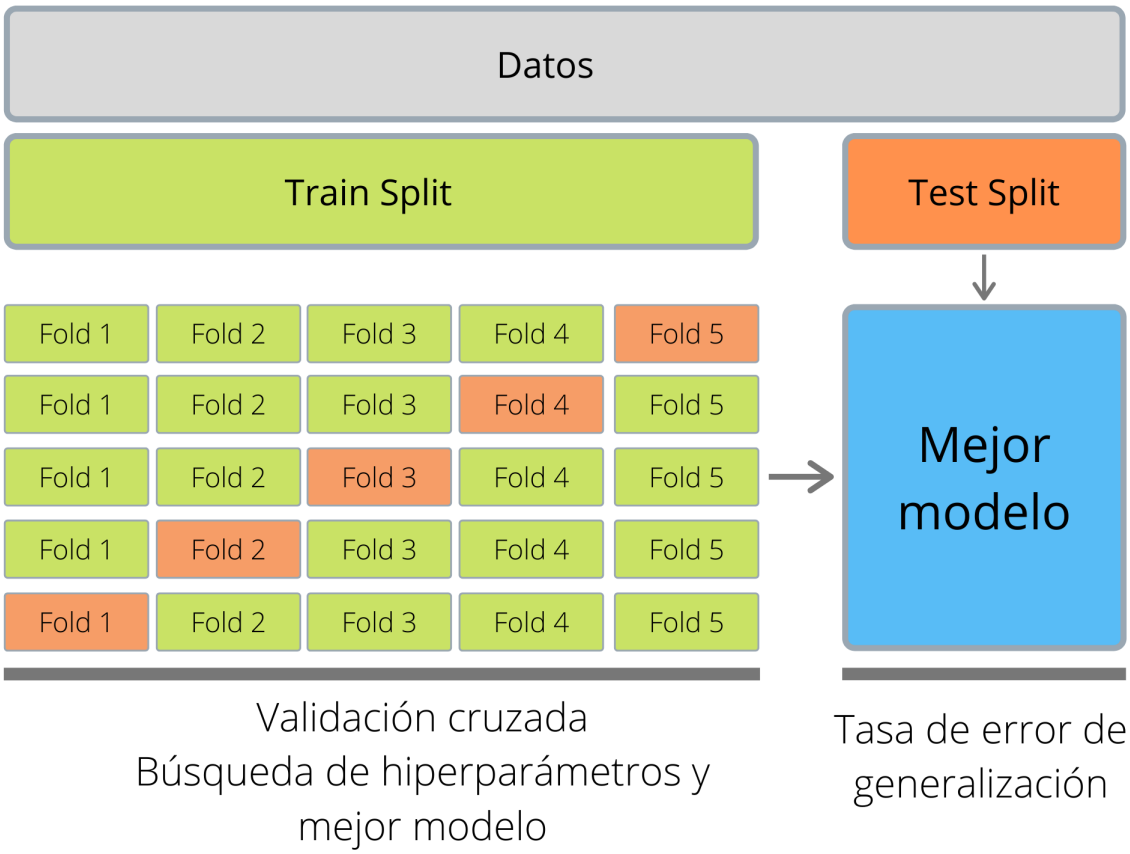


Figura 2.7: Esquema de desarrollo con validación cruzada 5-fold

2.9.3 Validación cruzada repetida

Además de realizar una validación cruzada para obtener un estimador más *"realista"* (con menor varianza) del error, podemos repetir el procedimiento anterior (XV+Test) varias veces (cambiando las particiones train y test en cada repetición de forma aleatoria). De este modo, tendremos varios estimadores del error (e_r, mientras que con el método anterior solo obtendríamos uno. En este TFG se realizará *5-fold XV* y se repetirá 20 veces.

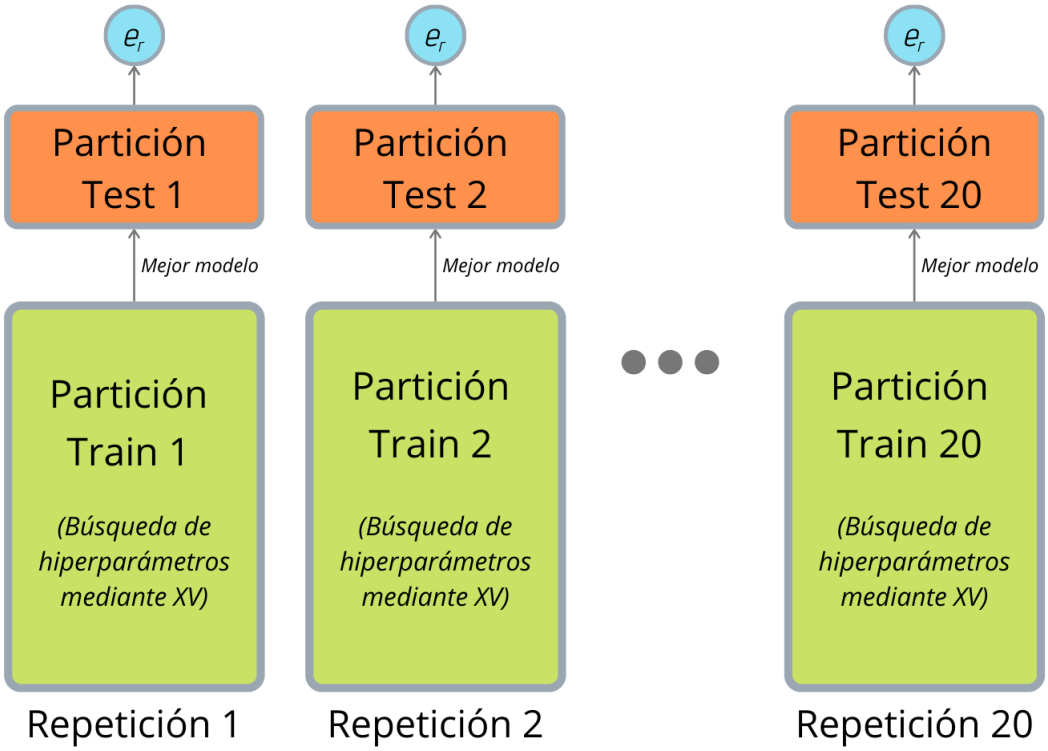


Figura 2.8: Validación cruzada + Test repetido

2.9.4 Búsqueda de hiperparámetros

Uno de los mayores problemas que debemos abordar a la hora de ajustar un modelo predictivo es el ajuste de hiperparámetros. En el caso de AdaBoost y XGBoost, tenemos los siguientes hiperparámetros:

AdaBoost

- Número de árboles. Esto determinará cuántos *stumps* se estimarán para realizar la predicción
- Función de actualización del *amount of saying*. Al ser un problema multiclass, necesitamos usar la actualización de *Zhu*.

Reg. L1	Learning rate	Profundidad de los árboles	Número de árboles
0.5	0.05	4	100

Tabla 2.1: Ejemplo de individuo de la pobación

XGBoost

Los hiperparámetros que consideraremos en primer lugar son:

- Regularización $L1$ a
- Learning rate $\alpha \in (0, 1)$: Un valor bajo implicará dar pequeños pasos en el refinado de la predicción. En la práctica
- Profundidad de los árboles
- Número de árboles

Para XGBoost, el espacio de estados en el que buscar una combinación de hiperparámetros óptima es computacionalmente inviable para las máquinas de las que disponemos. Es por esto que necesitamos una metaheurística para tratar de buscar un conjunto de hiperparámetros óptimos, al menos localmente. XGBoost tiene muchos más hiperparámetros, pero en primera instancia hemos probado estos debido a la carga computacional que supone la prueba de hiperparámetros en R (como veremos en el desarrollo del trabajo, XGBoost tan solo probando estos pocos hiperparámetros ya proporciona mejores resultados que AdaBoost, pero en las secciones finales, con implementaciones en Python podemos probar más ya que es un lenguaje más eficiente y con búsqueda de hiperparámetros incorporada de forma nativa). A continuación se presentan los algoritmos genéticos como metaheurísitca propuesta para este problema.

Algoritmo Genético:

Desde 1950, ya Alan Turing [26] sugirió la idea de un sistema computacional que evolucionase siguiendo los principios de la naturaleza. Un algoritmo genético es una metaheurística inspirada por los procesos de selección natural que sufren todos los seres vivos. La idea detrás de este proceso es que los individuos más aptos son los que se reproducen y pasan a tener descendientes, transmitiendo a estos sus características a las que llamamos **genes**. Me familiaricé con estos algoritmos en la asignatura de Algoritmos y Computación, donde desarrollé un proyecto didáctico en el que explicamos y mostramos una aplicación de aprendizaje por refuerzo con redes neuronales [27].
En nuestro caso, nuestra **población** de individuos estará compuesta por combinaciones de hiperparámetros. Un individuo tendrá por **genes** los hiperparámetros que listamos arriba. En la tabla 2.1 vemos un posible individuo
Los pasos que sigue el algoritmo son los siguientes:

1. **Inicialización de la población:** Creamos n individuos con genes (parámetros) aleatorios inicializados con una distribución uniforme.

2. **Selección de los más aptos:** Mediante validación cruzada sobre el conjunto de *test*, asignamos a cada individuo de la población un **score**, es decir, cuán apto es ese individuo. Solo los mejores individuos sobreviven, escogidos en función de su score, en este caso el error de validación cruzada. Lo interesante de escoger el error de validación cruzada como score no es solo que es un estimador del error con menor varianza, sino que la idea detrás de los algoritmos genéticos es que solo sobreviven los más aptos. Si un conjunto de hiperparámetros sobrevive varias iteraciones seguidas, quiere decir que ha proporcionado una tasa de error baja de forma consistente, es decir, está en cierto modo demostrando ser el mejor sean cual sean las particiones de la validación cruzada (que ya de por sí evita escoger hiperparámetros que cometan sobreajuste).

   Una vez hemos evaluado a todos los individuos, procedemos a seleccionar a los más aptos (los que minimizan ese error), en este el percentil 0.5.

3. **Reproducción:** Con los individuos más aptos, crearemos un nuevo set de hijos hasta alcanzar el tamaño de población n. En este problema se seleccionaron parejas de padres de forma aleatoria, y se seleccionaron de forma complementaria y aleatoria 2 parámetros de un padre y 2 de otro.

4. **Mutación:** Una vez hemos obtenido a los hijos de los más aptos, introducimos una pequeña mutación en los parámetros. Para los parámetros de regularización y *learning rate*, mutan con un porcentaje k que sigue una distribución $N(1, 0.15)$, de tal forma que varíen en un porcentaje hacia arriba o hacia abajo del 15% aproximadamente. En el caso del número de rondas y de árboles, del 10%.

5. **Hijo aleatorio:** Para dar a la metaheurística la capacidad de escapar del problema del mínimo local, en cada generación un hijo será sustituido por otro completamente aleatorio. De esta forma escapamos de una posible "endogamia" introduciendo parámetros nuevos en la población.

6. Volvemos al paso 2, tantas veces como nº de generaciones deseemos

De este modo, se espera que el algoritmo consiga llegar a una solución óptima (al menos localmente) y sin tener que recorrer el espacio de posibles valores de los hiperparámetros que sufre una explosión combinatoria debido al gran abanico de valores que pueden tomar estos.

Problemas del Algoritmo genético

Los principales problemas que sufren los algoritmos genéticos vienen por dos causas principalmente:

- **Implementación no consistente:** Los algoritmos genéticos son lo más parecido que hay a una receta de cocina en la computación. Hay unas guías generales pero pasos como la **reproducción** y **mutación** no hay nada definido, es decisión propia del implementador el cómo llevara a cabo ambas (puedes mutar en mayor o menor medida , generar hijos cogiendo más o menos atributos de cada padre...).

- **Carga computacional:** La razón de probar pocos hiperparámetros en primera instancia es que si se quieren probar muchos, deberíamos aumentar el tamaño de la población para que un mayor conjunto de posibles características estén recogidas en la población inicial. Si ponemos pocos individuos, la probabilidad de encontrar una combinación mejor se ve reducida a mayor número de genes.

Random Search

Una vez se trabaje Python, ya no preciso de un algoritmo genético implementado por mi para la búsqueda de hiperparámetros. XGBoost y LightGBM traen de forma nativa un *wrapper* para `scikit-learn`, la cual mediante una sencilla orden (`RandomSearchCV`) permite buscar el mejor conjunto de hiperparámetros indicándole el grid o posibles valores que se desean probar. Realizando varias iteraciones (es decir, muestrando conjuntos de valores para los hiperparámetros de forma aleatoria en cada iteración) y mediante validación cruzada, devuelve el mejor modelo con los mejores hiperparámetros según el criterio de la tasa de error.
Este módulo de `sklearn` soporta trabajo con varios hilos,a diferencia del algoritmo genético que no tiene paralelismo, lo cual aumenta la eficiencia del proceso en gran medida. Utilizaremos 100 repeticiones.

3. Descripción y procesado de los datos

3.1 Descripción del conjunto de datos

El conjunto de datos se compone de 900 observaciones de experimentos sobre motores, a los cuales se indujo fallos de forma controlada (perforando y dañando el rotor en cada experimento). Los experimentos fueron llevados a cabo por el departamento de Ingeniería Eléctrica de la Universidad de Valladolid. Estas observaciones son dobles, es decir, para un mismo experimento se tienen dos observaciones, una correspondiente al valor de cada uno de los dos armónicos inducidos sobre la onda de la corriente de alimentación. De estos 450 experimentos se tomó registro de:

- **Modelo del Variador:** Tres tipos de modelo, *AB, ABB* y *TM*
- **Nivel de carga:** Se suministraron dos niveles de carga, *NC1* y *NC2* a la hora de medir los armónicos.
- **Valor de los armónicos inducidos:** Para cada motor, se registró la medición de los armónicos inferior (*LSH*) y superior (*USH*) en 2048 instantes de tiempo en milisegundos desde que el motor se arranca.
- **Estado del motor:** Se clasificaron los 450 motores según su estado de deterioro, pudiendo tomar valores desde *R1* (motor en buenas condiciones) hasta *R5* (motor gravemente dañado).

La muestra de las 450 observaciones está balanceada, es decir, se tienen el mismo número de observaciones para cada combinación de valores de modelo, nivel de carga y estado del motor, lo cual es beneficioso de cara a entrenar modelos ya que evita que algún modelo o nivel de carga esté sobrerepresentado en el modelo, empeorando así la capacidad predictiva de este.

3.1.1 Modificación del conjunto de datos

Como se comentó antes, tenemos para cada motor una observación doble. Por ello, la transformación que debemos hacer es la de juntar por columnas las medidas de *LSH* y *USH* para cada

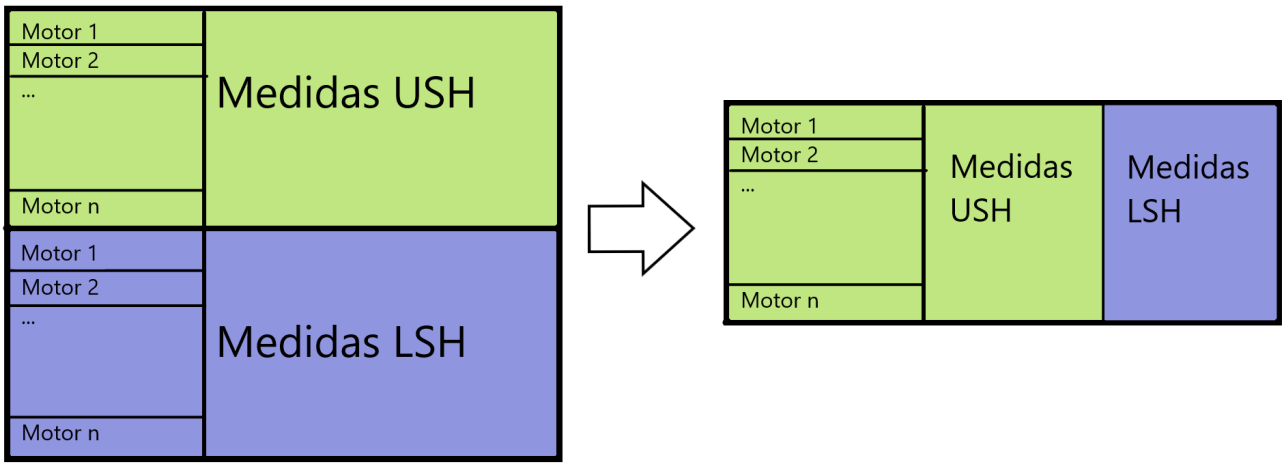


Figura 3.1: Transformación del dataset

motor. La Figura 3.1 muestra un esquema de esta transformación. Esta transformación puede ser problemática, ya que estamos "reduciendo" el tamaño de muestra n y estamos duplicando el número de variables p.

3.1.2 Previsualización del problema

A continuación se presenta un estudio descriptivo previo de la distribución de las clases y su relación con las medidas de los armónicos. La Figura 3.2 en su parte izquierda representa a modo de ejemplo la evolución de la medida del armónico *USH* a lo largo de los 2048 instantes del tiempo, coloreando los valores según el estado de deterioro del motor (rojo=R1, verde=R2, azul=R3, morado=R4 y marrón=R5). Si hacemos un corte transversal (parte derecha de la figura) en el instante de tiempo 1500, podemos ver cómo se distribuyen los valores de las ondas medidas en ese instante en función de la clase mediante 5 boxplots.

En este caso, lo que se aprecia es que para las clases R1-R3 (deterioros bajos), los valores de las ondas siguen una distribución semejante, toman un rango de valores parecido e inferior a los deterioros altos. Esto es una tendencia general en las ondas en todos los instantes de tiempo, salvo por los instantes iniciales, cuando el motor acaba de ser arrancado que hace que las medidas tengan una alta variabilidad, y los instantes finales.

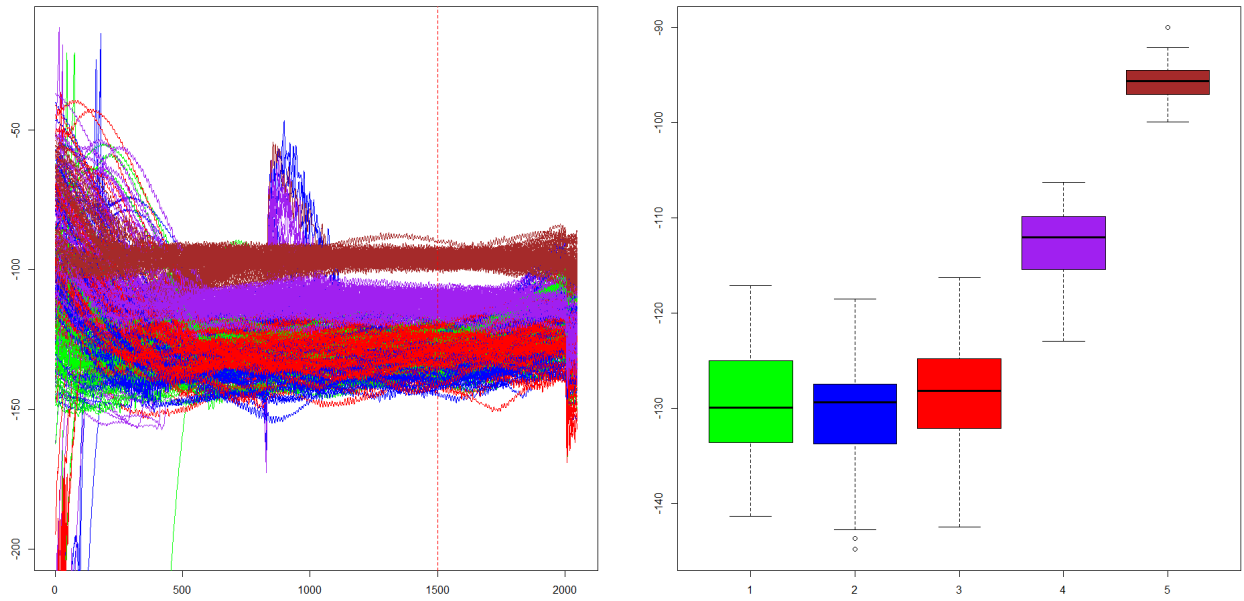


Figura 3.2: Ondas y Boxplots en t=1500 para la distribución de los valores según las clases del armónico USH

Este esquema se da también cuando observamos las ondas particularmente para una combinación específica de Modelo de inversor y Nivel de carga, como podemos ver en la figura 3.3, en la que se muestran para 75 motores con inversor *ABB* y nivel de carga *2*. En esta vemos cómo también se ven bien diferenciadas los estados de deterioro *R4,R5* respecto a *R1,R2,R3*. De hecho parece que estas últimas se diferencian un poco mejor, pero hablaremos sobre particularizar el estudio a cada combinación de factores *Model x Level* más adelante. La principal idea detrás de este Trabajo de Fin de grado es que una vez se ha estabilizado la onda, se puede utilizar los valores que esta toma y tratar de predecir el estado de deterioro del motor (valores altos de los armónicos inducidos indicarían mayor deterioro del rotor debido a una mayor vibración).

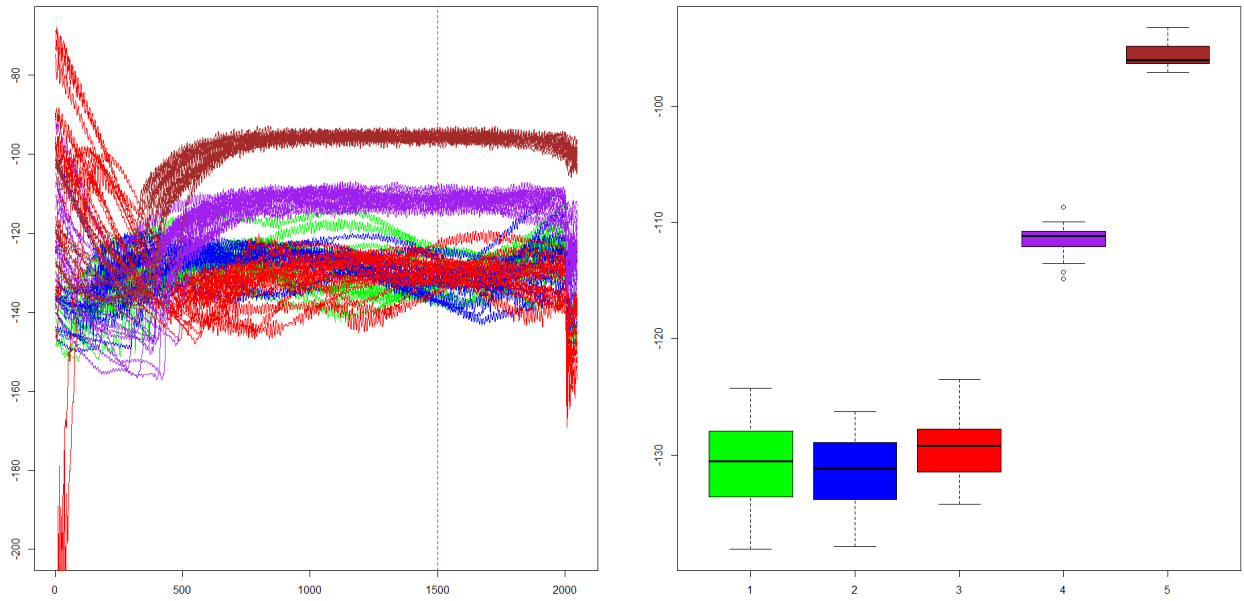


Figura 3.3: Ondas y Boxplots en t=1500 para la distribución de los valores según las clases del armónico USH para inversores ABB a nivel de carga NC2

3.2 Problemática de las variables

Debido a que tenemos la medición de 2048 instantes del tiempo en milisegundos para cada armónico, la información que nos proporcionan dos variables cercanas en el tiempo quizás sea redundante (lo que induce problemas de multicolinealidad, sobreajuste, necesidad de un mayor número de datos y altos tiempos de ejecución en el entrenamiento del modelo). Estamos ante un problema con una altísima dimensionalidad (tras la transformación del dataset de la Fig. 3.1, 4096 variables numéricas + 3 categóricas) para el número de observaciones que tenemos ($n = 450$) que, a priori, debería ser reducible, tratando de resumir esa redundancia que tenemos, especialmente una vez la medida de la onda se estabiliza.
Por esto, vamos a enfrentar dos conjuntos de variables:

- **Set de Variables 1**: Crearemos entornos cada 250 unidades de tiempo, tomando la media de los 10 valores situados entorno a estas marcas temporales($t \pm 5$). Además, viendo que en los primeros y últimos momentos de tiempo la medición es muy inestable, tomaremos solo mediciones desde el instante 250 hasta el 2000.

- **Set de Variables 2**: El original, tomando los 2048 instantes de tiempo para ambos armónicos.

A priori, el set de variables 1 podría proporcionar mejores resultados y tiempos de ejecución por los problemas que da esta sobredimensionalidad comentados.

3.3 Casos

De cara a no caer en la situación de que un set de variables sea mejor en algún escenario concreto, se realizará una validación cruzada *5-fold* para cada uno de los siguientes "casos". Entendemos por caso el entrenar un clasificador por separado para un grupo de subconjunto de los datos. A continuación se explican los subconjuntos para cada caso

- **Caso global:** Un subconjunto, proporcionando todas las observaciones y estimando un clasificador global para todas ellas.
- **Caso Banda:** Dos subconjuntos, cada uno proporcionando todas las observaciones pero solo las variables correspondientes a un armónico.
- **Caso Model x Level:** Seis subconjuntos de datos, creando un clasificador para cada combinación de Model x Level (75 observaciones en cada uno).
- **Caso Model:** Tres subconjuntos de datos, creando un clasificador para cada nivel de Model.
- **Caso Level:** Dos subconjuntos de datos, creando un clasificador para cada nivel de Level.

3.4 Resultados sobre sets de variables y Toma de decisiones

A continuación se adjuntan los resultados de los entrenamientos en todos los casos anteriormente comentados, considerando los algoritmos AdaBoost y XGBoost descritos en las secciones 2.2 y 2.4

Caso global

	Set de Variables 1		**Set de Variables 2**	
Algoritmo	*Error 5-XV*	*Error Test*	*Error 5-XV*	*Error Test*
AdaBoost	0.306	0.32	0.29	0.307
XGBoost	0.265	0.245	0.287	0.271

Tabla 3.1: Tasas de error: Caso Global

Caso Banda

	USH			
	Set de Variables 1		**Set de Variables 2**	
Algoritmo	*Error 5-XV*	*Error Test*	*Error 5-XV*	*Error Test*
AdaBoost	0.3	0.386	0.322	0.366
XGBoost	0.3	0.311	0.32	0.30

Tabla 3.2: Tasas de error: Caso Banda=USH

	LSH			
	Set de Variables 1		**Set de Variables 2**	
Algoritmo	*Error 5-XV*	*Error Test*	*Error 5-XV*	*Error Test*
AdaBoost	0.31	0.333	0.33	0.326
XGBoost	0.2437	0.2847	0.34	0.30

Tabla 3.3: Tasas de error: Caso Banda=LSH

Caso Model x Level:

		Set de Variables 1		**Set de Variables 2**	
Algoritmo	**Model x Level**	*Error 5-XV*	*Error Test*	*Error 5-XV*	*Error Test*
AdaBoost	*AB x NC1*	0.28	0.40	0.26	0.92
	ABB x NC1	0.30	0.16	0.34	0.32
	TM x NC1	0.30	0.24	0.34	0.40
	AB x NC2	0.26	0.28	0.32	0.32
	ABB x NC2	0.24	0.24	0.32	0.32
	TM x NC2	0.24	0.20	0.32	0.32
XGBoost	*AB x NC1*	0.20	0.40	0.34	0.32
	ABB x NC1	0.26	0.24	0.32	0.20
	TM x NC1	0.16	0.32	0.44	0.04
	AB x NC2	0.26	0.16	0.34	0.40
	ABB x NC2	0.18	0.28	0.34	0.20
	TM x NC2	0.18	0.12	0.26	0.24

Tabla 3.4: Tasas de error: Caso ModelxLevel

Caso Model

		Set de Variables 1		Set de Variables 2	
Algoritmo	**Model**	*Error 5-XV*	*Error Test*	*Error 5-XV*	*Error Test*
AdaBoost	*AB*	0.27	0.22	0.27	0.22
	ABB	0.28	0.34	0.28	0.34
	TM	0.21	0.10	0.21	0.10
XGBoost	*AB*	0.27	0.28	0.27	0.22
	ABB	0.24	0.34	0.29	0.28
	TM	0.17	0.24	0.27	0.23

Tabla 3.5: Tasas de error: Caso Model

Caso Level

		Set de Variables 1		Set de Variables 2	
Algoritmo	**Level**	*Error 5-XV*	*Error Test*	*Error 5-XV*	*Error Test*
AdaBoost	*NC1*	0.293	0.386	0.33	0.386
	NC2	0.266	0.20	0.253	0.320
XGBoost	*NC1*	0.280	0.333	0.413	0.30
	NC2	0.246	0.226	0.286	0.24

Tabla 3.6: Tasas de error: Caso Level

3.4.1 Toma de decisiones

Los resultados parecen confirmar la hipótesis de que utilizar el set de Variables 1 (reduciendo la dimensionalidad del problema de 2052 variables a 10) es equivalente a usar el set de Variables 2. Es cierto que solo estamos mirando el valor del estimador puntual del error, con una sola repetición, por lo que no estamos teniendo en cuenta la variabilidad del mismo. Sin embargo, al estar usando un error promedio de validación cruzada *5-fold* tenemos garantía de que esta variabilidad ha sido reducida.

Por último, otra justificación es la carga computacional y problemas de ajuste que supone trabajar con un problema con $n = 450, p = 4100$. Los tiempos de cómputo han sido muchísimo más altos con el set de variables 2, además de que dificulta la interpretación del modelo al estar en un problema $n >> p$ (dimensionalidad muy alta). Con esta justificación simplemente se podría haber optado por el set de Variables 1 y evitar una semana de cómputo que llevaron estas pruebas, pero de este modo ha sido corroborada empíricamente, por lo que a partir de ahora se trabajará con este set reducido.

4. Análisis mediante técnicas boosting y resultados

4.1 Descripción de los factores

De ahora en adelante se trabajará únicamente con el set de variables 1 (entornos). De estudios similares que se realizaron anteriormente [1][28][29], se tiene la intuición que al entrenar un clasificador separado para cada combinación de *Model* x *Level* se obtienen mejores resultados a la hora de clasificar. Es por esto que para tratar de confirmarlo, entrenaremos un clasificador para cada combinación *Model x Level*.

También se pensó que, por ser armónicos de la corriente de entrada, estas corrientes inducidas deberían ser "simétricas" respecto a la principal, por lo que podría ser que se pudiera entrenar proporcionando información del *USH* o del *LSH* por ser esta de alguna forma, redundante (Si son completamente simétricos, tomarían el mismo valor solo que en momentos de tiempo distintos). Esto es interesante de cara a reducir la dimensionalidad del problema, por lo que estudiaremos también entrenando con ambas mediciones por separado.

De cara a estudiar la influencia de los factores Model y Level (interpretables desde el punto de el Análisis de la Varianza (ANOVA) como un " tratamiento " a los individuos de la población) en la predicción, de qué algoritmo es mejor (estudiaremos AdaBoost, XGBoost,LightGBM y XGBoost-Dart) y de qué subconjunto de variables del set de variables 1 (Solo entrenar con las medidas de LSH, USH o con las dos), se realizarán 20 repeticiones de entrenamiento de un clasificador para cada combinación de factores. Así, tendremos 3 Modelos, 2 Niveles de carga, 4 Algoritmos y 3 subsets de variables, teniendo entonces 72 posibles combinaciones de factores. De cada una de esas 72 se realizarán 20 repeticiones, por lo que tendremos 1440 observaciones, cada una correspondiente al error sobre la partición de test del clasificador tras haber elegido los mejores hiperparámetros con un conjunto de entrenamiento y validación cruzada . Estas tasas de error serán estudiadas mediante un ANOVA para ver qué combinación de factores proporciona las mejores tasas de error y cómo interactúan entre sí.

Nombre del Factor	Niveles	Descripción
Model	*AB, ABB, TM*	Modelo del inversor que lleva el motor
Level	*NC1, NC2*	Nivel de carga al que se midió en el experimento para cada motor
Band	*LSH, USH, Todo*	Variables que se proporcionan al modelo para clasificar (armónico superior, inferior o los dos)
Algoritmo	*AdaBoost, XGBoost, LightGBM, DART*	Algoritmo que se entrena para clasificar

Tabla 4.1: Factores a estudiar en las observaciones del error obtenidas en el test al entrenar en cada combinación

4.2 Notas sobre la obtención de las observaciones según factor

Para cada combinación de factores, las observaciones se obtienen de forma distinta:

- **Factores Model y Level:** Estos factores ya se aplicaron en el muestreo, cada observación indica las medidas para un modelo y nivel de carga específico. Al entrenar el algoritmo, solo se le proporcionará las observaciones pertenecientes al subconjunto de cada combinación *Model x Level*.
- **Factor Banda:** Proporcionaremos las variables asociadas a las medidas del armónico *LSH, USH* o las de los dos (Todo).
- **Factor Algoritmo:** Entrenaremos cada algoritmo con los datos proporcionados por las observaciones restringidas con los factores *Model* y *Level* y con las variables que indique el Factor Banda.

Así mismo, la obtención de las ejecuciones de XGBoost, LightGBM y XGBoostDart se realizarán en Python, debido a una mayor eficiencia de las librerías y a la incorporación de forma nativa de búsqueda de hiperparámetros mediante RandomSearch.

4.3 ANOVA sobre los factores

Las Tablas 4.2 y 4.3 muestran el ANOVA sobre las métricas de la sección 2.9

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Model	2	0.75	0.37	59.57	0.0000
Level	1	0.46	0.46	73.56	0.0000
Band	2	0.95	0.48	75.84	0.0000
Algoritmo	3	0.66	0.22	35.29	0.0000
Model:Level	2	0.41	0.20	32.64	0.0000
Model:Band	4	0.10	0.02	3.80	0.0044
Level:Band	2	0.05	0.02	3.86	0.0213
Model:Algoritmo	6	0.02	0.00	0.56	0.7617
Level:Algoritmo	3	0.03	0.01	1.63	0.1807
Band:Algoritmo	6	0.16	0.03	4.17	0.0004
Model:Level:Band	4	0.45	0.11	17.92	0.0000
Model:Level:Algoritmo	6	0.06	0.01	1.51	0.1724
Model:Band:Algoritmo	12	0.13	0.01	1.67	0.0672
Level:Band:Algoritmo	6	0.06	0.01	1.70	0.1168
Model:Level:Band:Algoritmo	12	0.12	0.01	1.66	0.0711
Residuals	1368	8.57	0.01		

Tabla 4.2: Tabla ANOVA sobre la tasa de error de los algoritmos y los factores estudiados

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Model	2	5.75	2.87	119.39	0.0000
Level	1	1.42	1.42	59.18	0.0000
Band	2	3.39	1.69	70.35	0.0000
Algoritmo	3	4.04	1.35	56.01	0.0000
Model:Level	2	1.49	0.75	31.02	0.0000
Model:Band	4	0.20	0.05	2.04	0.0866
Level:Band	2	0.18	0.09	3.72	0.0244
Model:Algoritmo	6	0.13	0.02	0.92	0.4796
Level:Algoritmo	3	0.05	0.02	0.64	0.5884
Band:Algoritmo	6	0.36	0.06	2.47	0.0220
Model:Level:Band	4	0.53	0.13	5.48	0.0002
Model:Level:Algoritmo	6	0.31	0.05	2.13	0.0475
Model:Band:Algoritmo	12	0.22	0.02	0.75	0.7075
Level:Band:Algoritmo	6	0.73	0.12	5.08	0.0000
Model:Level:Band:Algoritmo	12	0.07	0.01	0.24	0.9961
Residuals	1368	32.91	0.02		

Tabla 4.3: Tabla ANOVA sobre el MAE de los algoritmos y los factores estudiados

Si observamos la tabla 4.2, vemos que los grados de libertad de los residuales son altísimos. Esto va a implicar que valores muy pequeños de las sumas de cuadrados para los efectos de los factores (e interacciones) sean detectadas como efectos significativas en el test de la F. Pero, ¿podemos aceptar que los grados de libertad sean tan altos? Tras obtener los resultados se consideró que esta *explosión* de los grados de libertad que era excesiva para el número de

observaciones de motores que estamos manejando. A continuación se expone el razonamiento de por qué estos grados de libertad no son *fidedignos*.

Cuando realizamos un ANOVA, tenemos dos hipótesis principales a asumir: Normalidad de los residuos e independencia de las observaciones. Debido a la segunda, sabemos que los $Gl_{residuals} = n_{observaciones} - Gl_{modelo}$. Pero, en este caso nuestras observaciones son evaluaciones del error de clasificación obtenidas mediante validaciones cruzadas repetidas.

Cuando realizamos validación cruzada, nos aseguramos de que todas las particiones sean disjuntas e independientes, de tal forma que una observación caiga en una y solo una partición (ver Fig. 2.7). Nuestro estimador del error es el promedio de los e_k, obtenidos en cada una de las k particiones de test. Si suponemos normalidad en los e_k nuestro estimador es un promedio de normales independientes, por lo que por el teorema central del límite podemos asegurar que el estimador del error de clasificación que obtenemos tendrá una varianza reducida en función del número de *folds* o particiones.

$$e_c \sim N(e, \sigma^2) \tag{4.1}$$

$$\hat{e} = \frac{\sum_c^{n_{folds}} e_c}{n_{folds}} \sim N(e, \frac{\sigma^2}{n_{folds}}) \tag{4.2}$$

Sin embargo, cuando realizamos validación cruzada repetida, los errores no son independientes. En la Figura 4.1 se ilustra este problema. Observemos el individuo (motor) i . Como se aprecia en la imagen, esta observación puede caer varias veces en la partición Test, debido a que las repetición del proceso no tiene en cuenta las anteriores para que sean disjuntas. Esto hace que los e_r no sean independientes (no es una dependencia muy fuerte, pero existe)

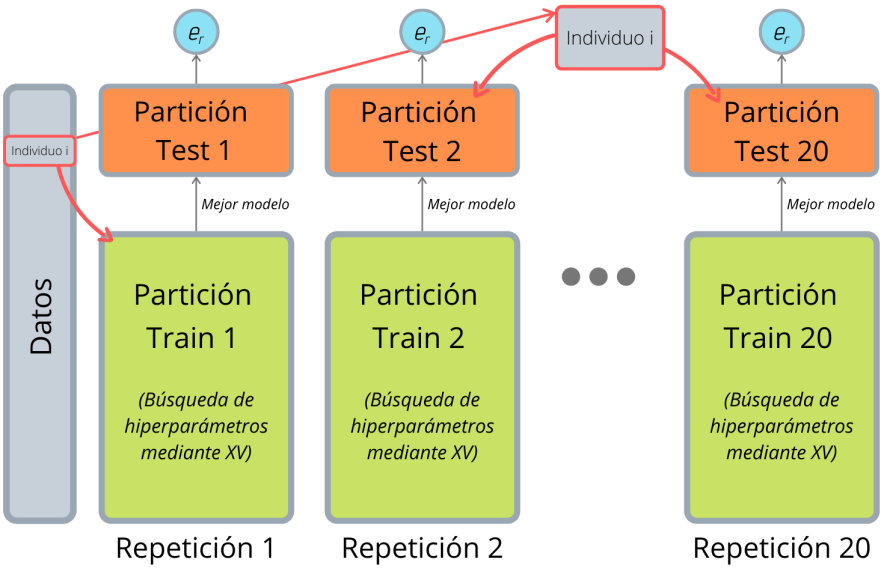


Figura 4.1: Una observación puede caer en varias particiones, haciendo que los e_r no sean independientes

Como consecuencia de esto, estamos viendo que los grados de libertad se han aumentado de forma ficticia (recordemos que los grados de libertad son análogos a la cantidad de información que tenemos. Por mucho que repitamos experimentos sobre las mismas observaciones, la información no aumenta más de un cierto límite). Este aumento de los grados de libertad también está asociado a una reducción ficticia de la varianza del estimador, por lo que al realizar un test de la F para comparar efectos, encontrará diferencias significativas con mayor facilidad.
La consecuencia de esto es que el $p-valor$ que aparece en la tabla ANOVA no es del todo fiable, es demasiado "optimista" en cuanto a que detecta diferencias significativas que en verdad no lo son (error de tipo I muy alto). Por esto, hay dos alternativas:

- Reducir los grados de libertad de forma manual
- Ser mucho más estrictos con el p-valor

La primera además de ser mucho menos rigurosa (estaríamos "cocinando" los resultados), carece de un fundamento o metodología para ser realizada. La segunda, sin embargo, aprovecha la metodología subyacente en el ANOVA que aunque no cumpla la hipótesis de independencia, puede valernos para estudiar qué impacto tienen los efectos paliando ese alto error de tipo I detectado.

Habitualmente, se establece un $p - valor = 0.05$ como corte. En este caso, se utilizará un $p - valor = 0.005$. De esta forma, eliminando las interacciones no significativas, el ANOVA queda de la siguiente forma:

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Model	2	0.75	0.37	58.64	0.0000
Level	1	0.46	0.46	72.41	0.0000
Band	2	0.95	0.48	74.65	0.0000
Algoritmo	3	0.66	0.22	34.74	0.0000
Model:Level	2	0.41	0.20	32.13	0.0000
Model:Band	4	0.10	0.02	3.74	0.0049
Band:Algoritmo	6	0.16	0.03	4.11	0.0004
Model:Level:Band	6	0.50	0.08	13.03	0.0000
Residuals	1413	8.99	0.01		

Tabla 4.4: ANOVA con los factores significativos sobre la tasa de error

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Model	2	5.75	2.87	119.89	0.0000
Level	1	1.42	1.42	59.43	0.0000
Band	2	3.39	1.69	70.64	0.0000
Algoritmo	3	4.04	1.35	56.25	0.0000
Model:Level	2	1.49	0.75	31.15	0.0000
Model:Band	4	0.20	0.05	2.05	0.0854
Model:Level:Band	6	0.71	0.12	4.92	0.0001
Level:Band:Algoritmo	15	1.14	0.08	3.16	0.0000
Residuals	1404	33.64	0.02		

Tabla 4.5: ANOVA con los factores significativos sobre el MAE

En las siguientes secciones se expone el análisis de estos resultados, estudiando los factores significativos y sus interacciones. Posteriormente, y basándonos en estos resultados, se realizará el estudio de los modelos según la combinación de factores con la tasa de error más baja. Los resultados de los estimadores promedio de los errores para cada algoritmo están disponibles en las tablas A.1 y A.2 de los anexos.

4.3.1 Factor Model

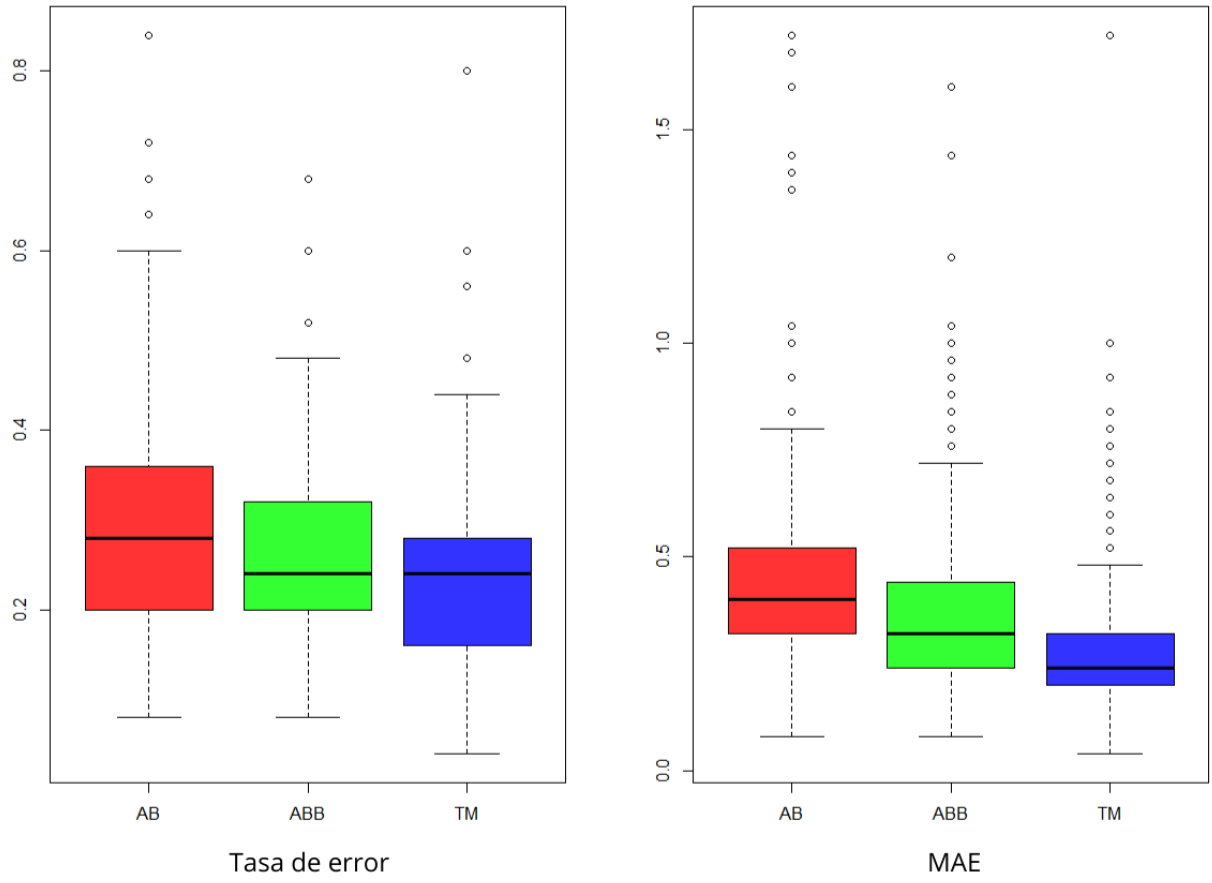


Figura 4.2: Boxplots de las tasas de error según modelo

El efecto del factor *Modelo* es significativo, por lo que el particularizar la predicción según el modelo tiene sentido.

Como podemos ver en los boxplot, para motores con el inversor *TM* la predicción es más precisa proporcionando tasas de error más bajas que para los otros dos. Aunque para inversores *ABB* el valor mediano del error parece similar, la distribución de los datos es más asimétrica hacia la derecha (valores superiores). Además, viendo el boxplot de *TM* vemos que el primer y tercer cuartil es toman valores más bajos para TM, así como los "bigotes" inferior y superior del boxplot. A pesar de que hay un outlier más alto para *TM*, las evidencias anteriores indican que clasificadores para *TM* serán más precisos que para *ABB*.

El inversor *AB* proporciona las tasas de error más altas que los otros dos. Sin embargo, el ANOVA indica que también hay una interacción *Model x Level*, que analizaremos posteriormente.

4.3.2 Factor Level

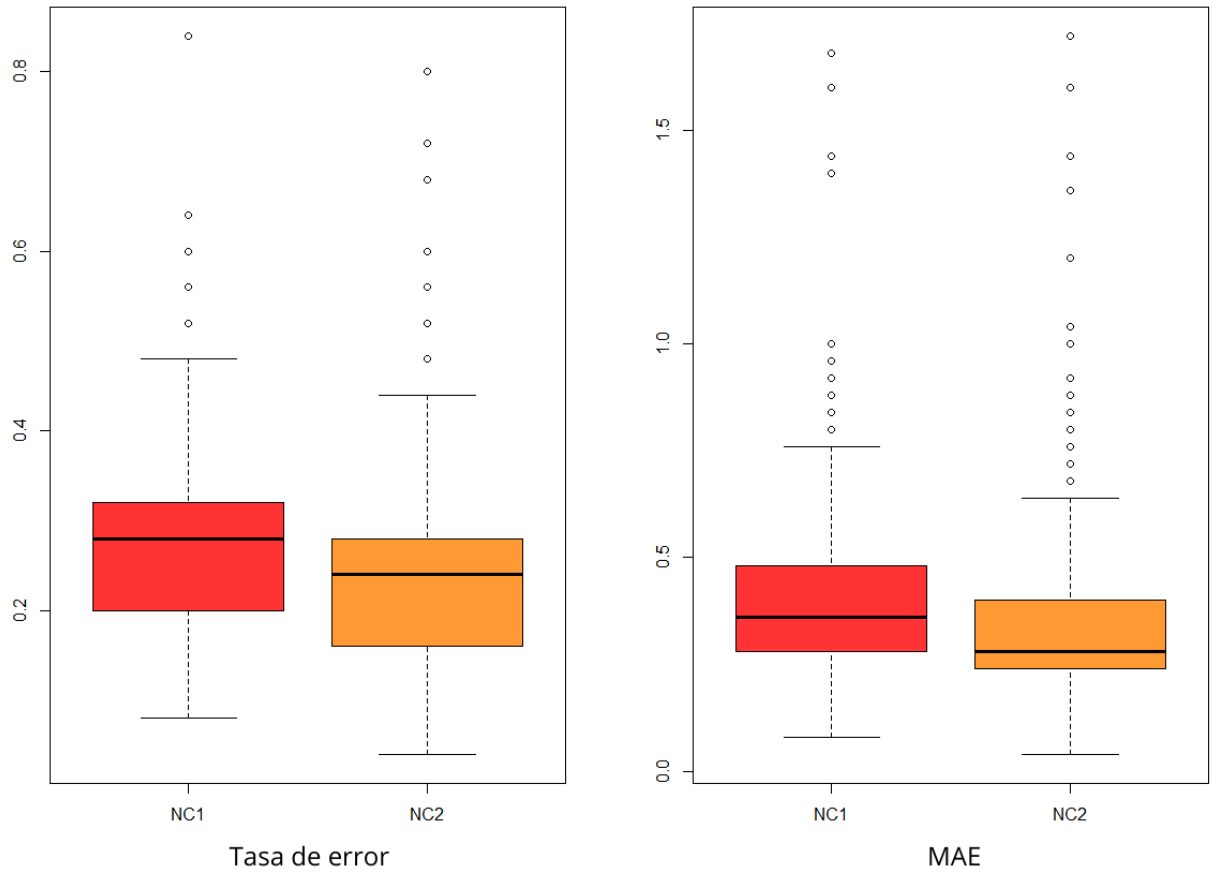


Figura 4.3: Boxplots de las tasas de error según el nivel de carga

Así como particularizar la predicción por modelo hemos visto que tiene sentido, también lo tiene para los distintos niveles de carga. Al igual que el factor *Model*, el factor *Level* o nivel de carga también es significativo.
A la hora de predecir el estado de deterioro de un motor, esto nos indica si solo nos fijamos en el nivel de carga, medir a nivel de carga 2 proporcionará una precisión mayor que al nivel de carga 1.

4.3.3 Factor Band

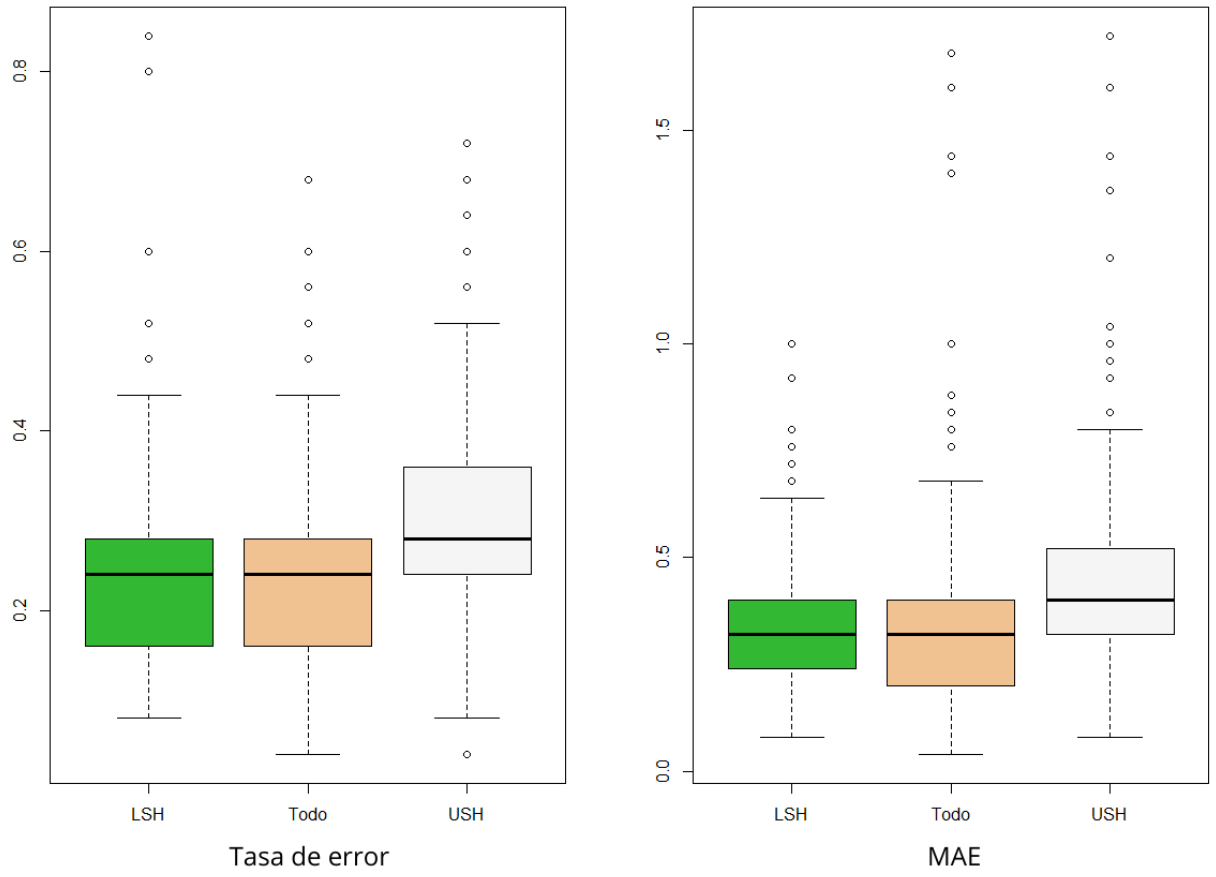


Figura 4.4: Boxplots de las tasas de error según la banda

El efecto de este factor es también interesante. En este caso el factor *Band* indicaba qué información se proporcionaba al clasificador (las medidas del armónico inferior (*LSH*) únicamente, las del superior (*USH*) o las de ambos(*Todo*)) con objeto de tratar de reducir la dimensionalidad del problema de clasificación a la mitad en variables.

Como se aprecia en el boxplot, el proporcionar solo el armónico *USH* proporciona tasas de error más altas. Sin embargo, si nos fijamos en *LSH* frente a *Todo*, la diferencia es ínfima. Parece que el extremo inferior del boxplot *Todo* toma valores más bajos y los outliers superiores también. Pero la diferencia es tan pequeña que dado que aparecen interacciones *Model x Band* y *Model x Level x Band*, estudiaremos el proporcionar qué información según el modelo más adelante.

4.3.4 Factor Algoritmo

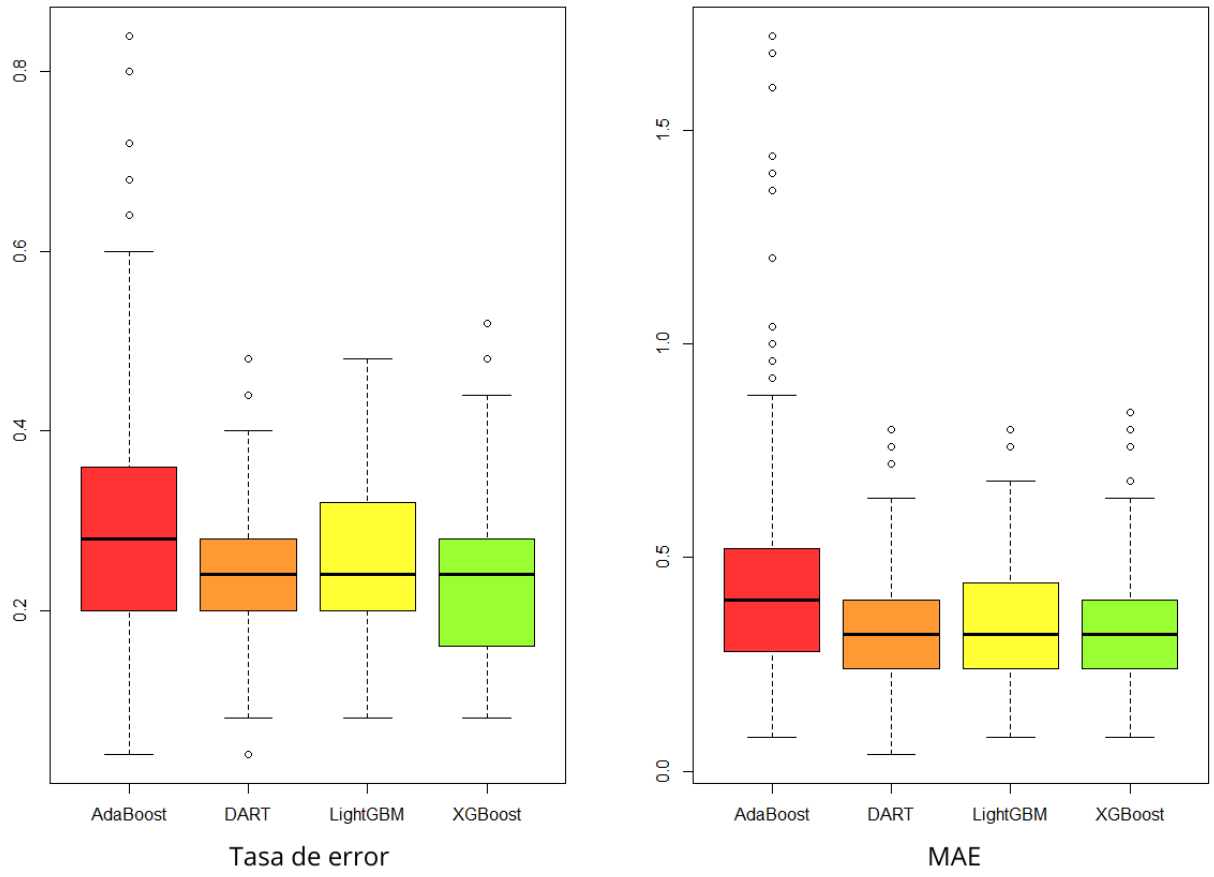


Figura 4.5: Boxplots de las tasas de error según el algoritmo

Mediante este factor podemos estudiar qué algoritmo es, en general, más preciso a la hora de clasificar y así compararlos.

AdaBoost proporciona los peores resultados (y además su distribución tiene poco apuntamiento). Para los otros tres, el mejor parece XGBoost por su primer cuartil, porque la mediana es prácticamente la misma para *DART,LightGBM* y *XGBoost*. Por lo que a priori parece el más preciso para este problema según la tasa de error, pero esto no es así según el MAE. Podemos particularizar para cada cada combinación *Modelo y Nivel de Carga* gracias a la interacción triple *Model x Level x Band* y ver qué algoritmo usar para cada combinación de los otros factores.

4.3.5 Interacción Band*Algoritmo

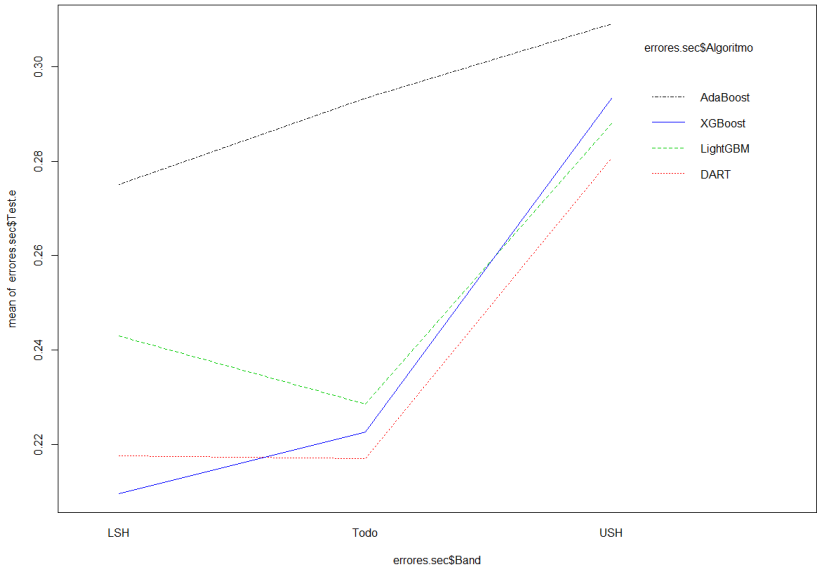


Figura 4.6: Grafico de interacción Banda x Algoritmo

Esta interacción nos habla de, proporcionada una información, qué algoritmo predice mejor el estado de deterioro de un motor (o, en el otro sentido, para cada algoritmo qué información es mejor proporcionar). A simple vista, proporcionar solo la información del armónico *USH* da los peores resultados para todos los algoritmos.
Como vemos, si solo proporcionamos la información relativa al armónico *LSH*, el algoritmo que mejor predice es XGBoost, dando la tasa de error más baja de los cuatro algoritmos estudiados. Sin embargo, si proporcionamos la información de los dos armónicos (*Todo*, que como vimos en la sección anterior podía ser interesante), vemos que DART también proporciona buenas tasas de error, similares a las de proporcionar solo LSH. Como vimos en la sección 2.6, el dropout es una técnica muy potente contra el sobreajuste y eficaz en problemas con alta dimensionalidad. Esto explicaría por qué parece que en el caso de DART no hay prácticamente diferencia entre proporcionar solo *LSH* o proporcionar *Todo*.

4.3.6 Interacción Model*Level*Band

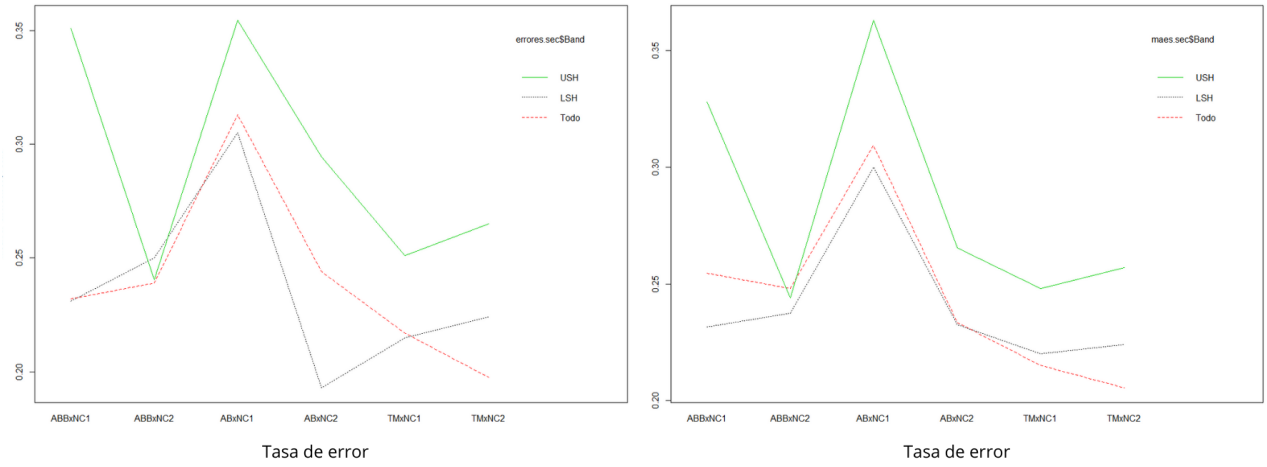


Figura 4.7: Gráfico de interacción Model x Nivel de carga x Banda

Esta interacción triple, aunque difícil de estudiar por ser triple, nos proporciona un escenario concreto a la hora de predecir para cada modelo de inversor.

Como vemos en el gráfico, podemos ver en el eje de abscisas las distintas combinaciones *Model x Level*. Si las comparamos por pares de *Model*, podemos ver en cuál de los dos niveles de carga y para qué "armónico" se obtienen mejores resultados.

Para el modelo ABB como vemos el error más bajo se obtiene, con el armónico LSH y nivel de carga 1, aunque seguido muy de cerca de utilizar los dos armónicos (*Todo*).

Para el modelo AB, tendríamos que emplear el nivel de carga 2 y proporcionar la información del armónico LSH. Esta combinación es la que proporciona las tasas de error más bajas, inferiores en promedio al 0.2

Para el modelo TM, también mediríamos a nivel de carga 2. Sin embargo, para este modelo y nivel de carga, a la vista de los resultados es bastante mejor proporcionar la información de los dos armónicos de cara a una mejor predicción

Estos resultados son iguales para el MAE, por lo que se usarán para decidir cómo entrenar modelos para estudiar el problema

4.3.7 Interacion Level*Algoritmo*Band

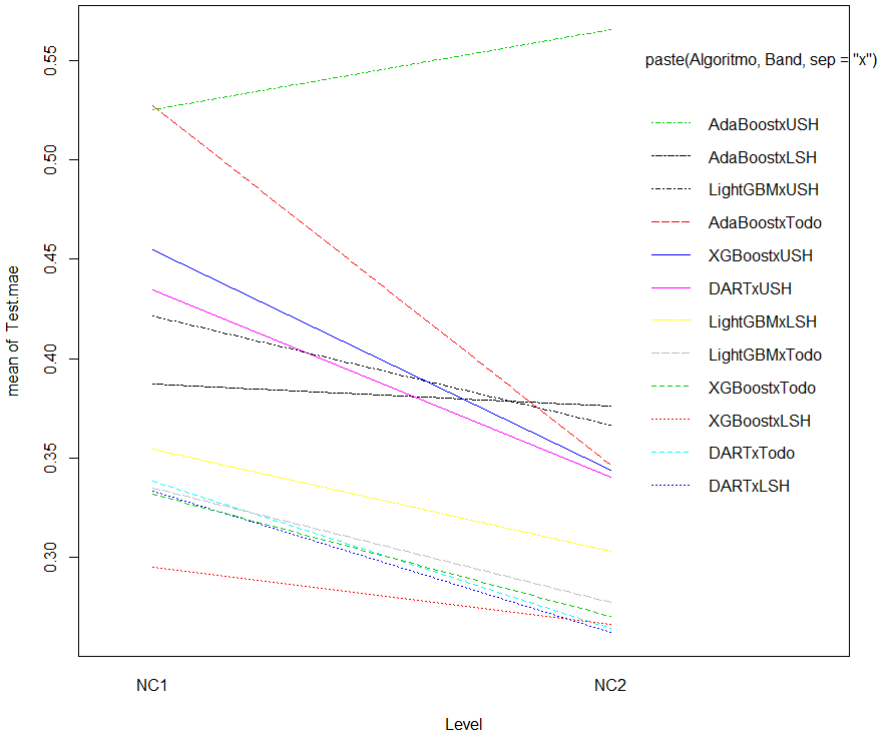


Figura 4.8: Grafico de interacción LevelxAlgoxBand para el MAE

Esta interacción es solo significativa para el MAE. Sin embargo, a la vista de la figura podemos ver que en general el patrón es común de menor tasa de error a Nivel de Carga 2 para todos los algoritmos salvo para AdaBoost con USH. Si se quita AdaBoost del ANOVA para el MAE, podemos ver si la interacción sigue siendo significativa

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Model	2	3.76	1.88	178.24	0.0000
Level	1	1.21	1.21	114.72	0.0000
Band	2	1.98	0.99	93.82	0.0000
Algoritmo	2	0.05	0.03	2.53	0.0800
Model:Level	2	0.73	0.37	34.65	0.0000
Model:Band	4	0.09	0.02	2.14	0.0741
Level:Band	2	0.06	0.03	2.85	0.0586
Model:Algoritmo	4	0.04	0.01	0.99	0.4107
Level:Algoritmo	2	0.03	0.01	1.34	0.2611
Band:Algoritmo	4	0.10	0.02	2.33	0.0540
Model:Level:Band	4	0.37	0.09	8.86	0.0000
Model:Level:Algoritmo	4	0.05	0.01	1.07	0.3693
Model:Band:Algoritmo	8	0.11	0.01	1.29	0.2438
Level:Band:Algoritmo	4	0.05	0.01	1.24	0.2927
Model:Level:Band:Algoritmo	8	0.06	0.01	0.73	0.6643
Residuals	1026	10.83	0.01		

Tabla 4.6: ANOVA para el MAE sin AdaBoost

Como vemos, esa anomalía era la causa de la significación de este factor (el p-valor ha aumentado hasta 0.6643), el cual nos permite mantener las conclusiones que teníamos de las secciones anteriores.

4.4 Estudio descriptivo de modelos

En esta memoria se han expuesto múltiples evidencias de que los tres inversores estudiados (*AB, ABB*y *TM*) se comportan de forma distinta y por tanto necesitan de un estudio personalizado para cada inversor. A continuación, y en base a las conclusiones de las secciones 4.3.6 y 4.3.5 se expone un estudio individualizado para cada modelo de inversor sobre la importancia de las variables a la hora de clasificar según lo que nos indique un modelo entrenado en exclusiva para combinación de *Model x Level x Band*. De este modo, podemos ver, según el modelo, la distribución de las clases entorno a los instantes de tiempo

4.4.1 Inversor AB

Para el inversor AB, teníamos con la figuras 4.6 y 4.7 que se debería medir a *NC2*, proporcionando a *XGBoost* la información del armónico *LSH*. Con esta información, se entrenará al clasificador solo con las observaciones e información del armónico correspondientes, y se realizará un estudio descriptivo de las variables más importantes. La tabla 4.7 muestra el promedio de 20 matrices de confusión sobre el test para este clasificador, entrenando con 50 observaciones y reservando 25 para test de forma estratificada.

AB	Predicho					
Real		*R1*	*R2*	*R3*	*R4*	*R5*
	R1	3.9	0.2	0.85	0	0.05
	R2	0.45	3.8	0.75	0	0
	R3	1.8	0.5	2.7	0	0
	R4	0.05	0.05	0	4.9	0
	R5	0	0	0	0	5

Tabla 4.7: Matriz de confusión promediada en 20 repeticiones para AB

AB	Predicho					
Real		*R1*	*R2*	*R3*	*R4*	*R5*
	R1	0.78	0.04	0.17	0	0.01
	R2	0.09	0.76	0.15	0	0
	R3	0.36	0.1	0.54	0	0
	R4	0.01	0.01	0	0.98	0
	R5	0	0	0	0	1

Tabla 4.8: Matriz de confusión (condicionada por filas) promediada en 20 repeticiones para AB

Importancia de las variables

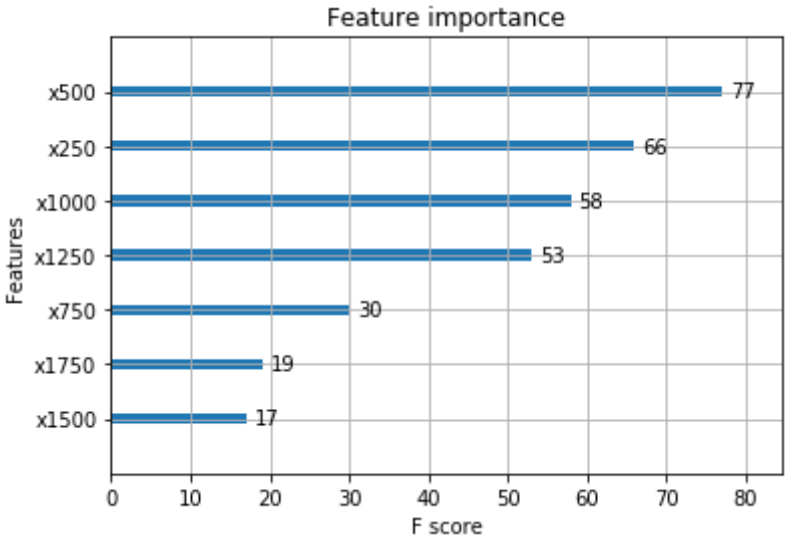


Figura 4.9: Importancia de las variables

Como vemos, las variables con mayor importancia (siendo el F-score el número de veces que aparecen en los árboles del ensemble) son las medidas del armónico *LSH* en los instantes de tiempo 500ms,250ms y 1000ms.
Para poder estudiar cómo se distribuyen las clases entorno a estas variables, realizaremos una simulación de tipo Montecarlo lanzando 1000000 puntos de tal forma que recorramos todos los

posibles valores que toman estas tres variables (entre el mínimo y el máximo de las mismas en el conjunto de datos de entrenamiento) y poder ver por áreas cómo está clasificando XGBoost.

Estudio de las variables

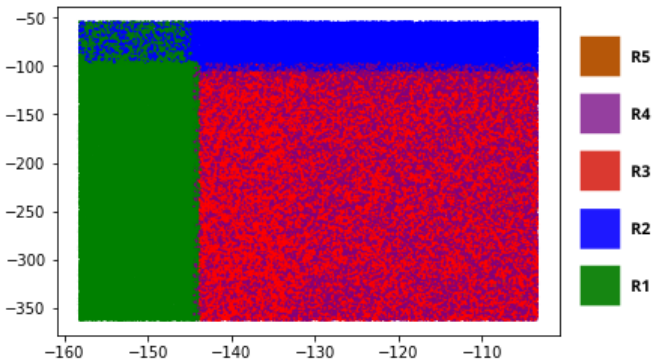


Figura 4.10: Distribución de las clases en las variables x250 y x500

Como vemos, se diferencian claramente las regiones que determina el boosting. En una región a la izquierda tenemos los motores en estado R1, es decir, los menos dañados. En la parte superior, los R2, entremezclándose estas regiones en la parte superior izquierda, pero claramente diferenciadas de una tercera región donde se sitúan R3, R4 y R5.

Sin embargo, esta visión en dos dimensiones no nos está dejando ver el concepto aprendido por el algoritmo de forma fidedigna (de hecho parece no haber puntos marrones prácticamente), ya que es una proyección de un espacio $\mathbb{R}^7$ (tenemos 7 variables). Visualicemos en 3 dimensiones la distribución de las clases, en conjunto y por separado.

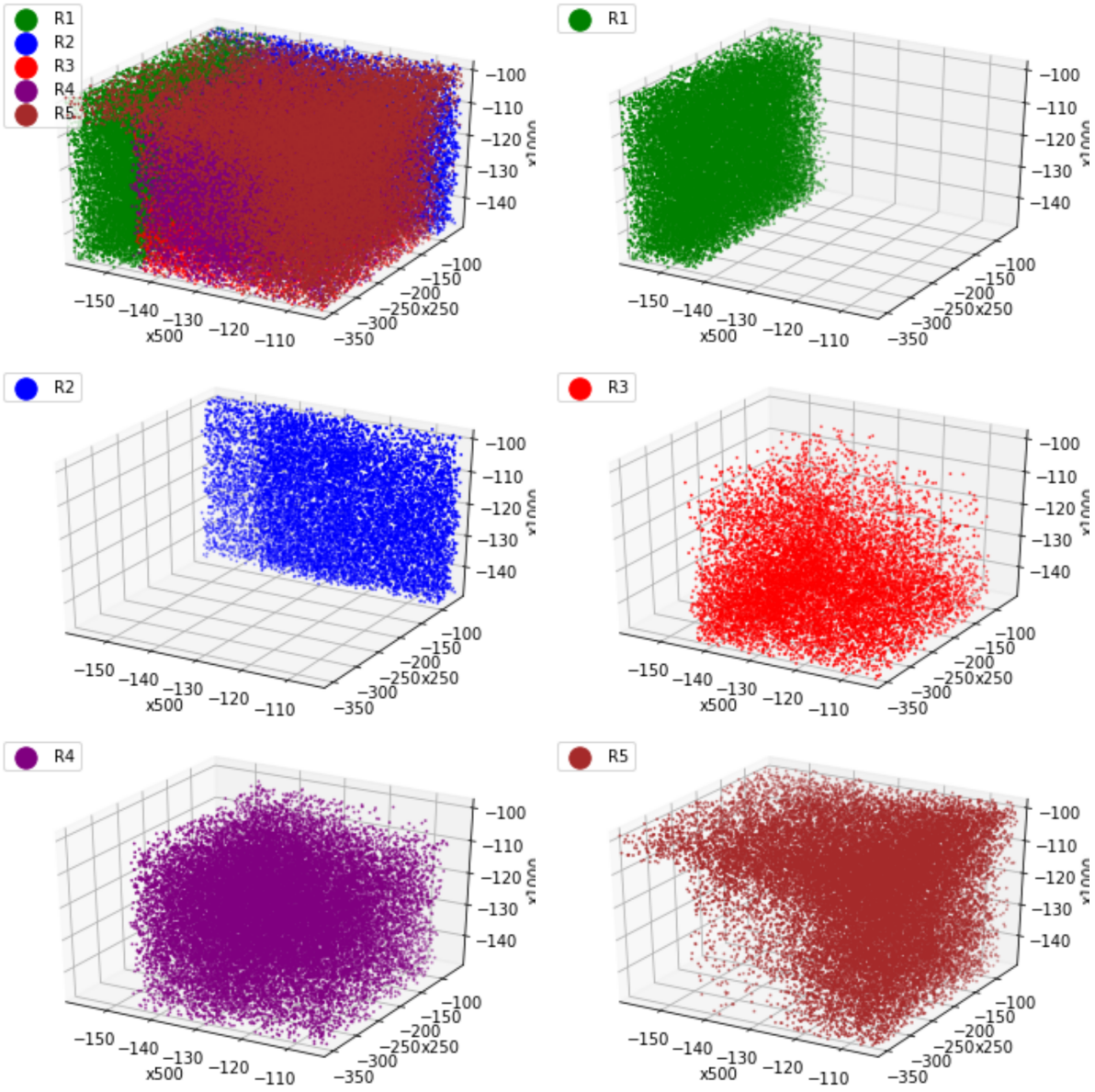


Figura 4.11: Distribución de la predicción de las clases en x500,x250 y x1000

Con esta figura podemos ver mejor las nubes de puntos. Podemos apreciar claramente dos paralelepípedos para R1 y R2, similar a lo que vimos en la figura 4.10. Sin embargo, podemos ver que aunque las clases R3, R4 y R5 están contenidas en una región como antes, vemos como las clases R5 y R3 están orientadas hacia valores superiores e inferiores de x1000 respectivamente. Vamos a realizar 3 cortes para x1000 y proyectar, de cara a ver si la distribución cambia.

Para x1000$\in (-110, 100)$ podemos estudiar esa capa superior que parece existir la distribución de R5, para x1000$\in (-140, -110)$ parce haber una mayor concentración de puntos morados y para x1000$\in (-150, 140)$ de puntos rojos.

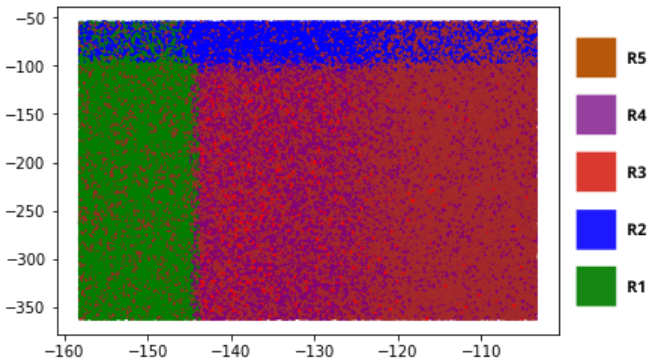


Figura 4.12: Distribución de las clases en las variables x250 y x500 para x1000$\in (-110, -100)$

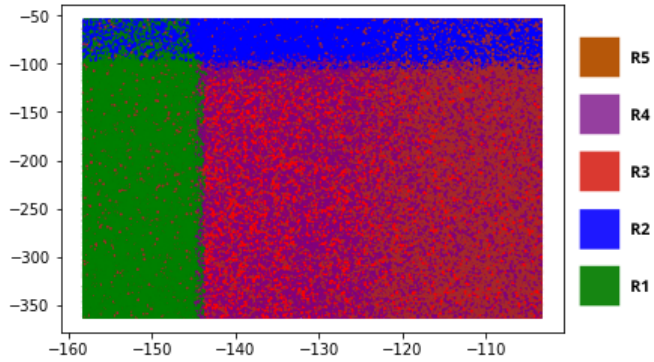


Figura 4.13: Distribución de las clases en las variables x250 y x500 para x1000$\in (-140, -110)$

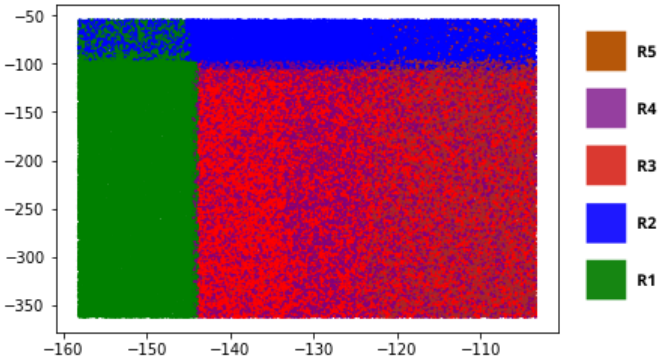


Figura 4.14: Distribución de las clases en las variables x250 y x500 para x1000$\in (-150, -140)$

Como vemos en las figuras, esto parece que es cierto. En valores más altos de x1000 (Fig. 4.12), aparecen muchos más puntos R5 (marrones) que en la Figura 4.10 no aparecía, especialmente concentrados para valores de x250 superiores a -120. De hecho, en este corte parece que incluso se entremezclan puntos marrones con los verdes y con los azules en mayor medida, cosa que no pasaba en la vista inicial.

Si pasamos a los valores intermedios (Fig. 4.13) vemos como los puntos morados aparecen con mayor frecuencia en detrimento de los marrones, que van disminuyendo según descendemos en valores de x1000. Y finalmente, si seguimos descendiendo a los valores más bajos de x1000, en la Fig. 4.14 podemos ver como los puntos rojos aparecen con mucha mayor frecuencia, justo lo que se veía con los gráficos 3D.

SHAP

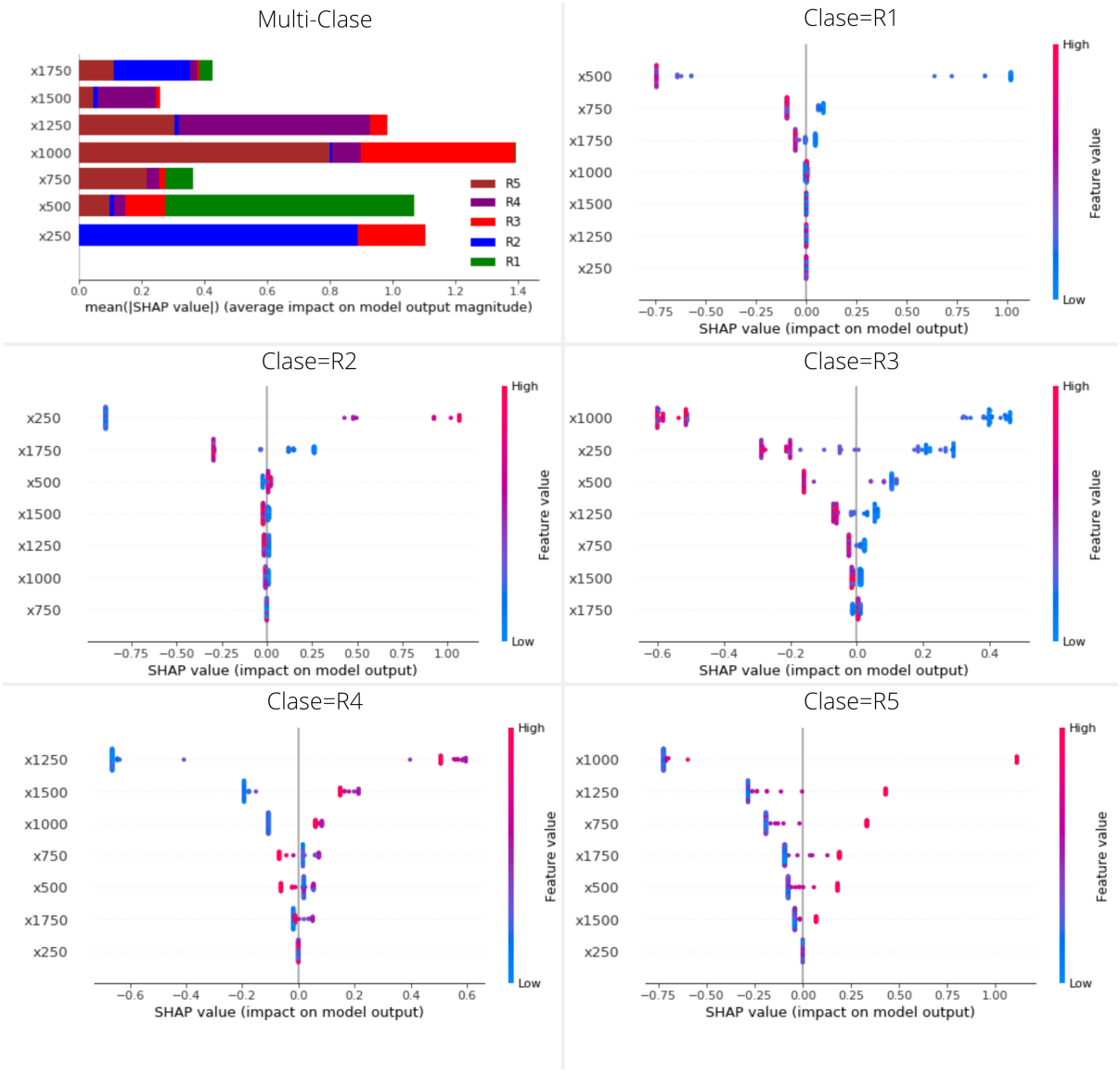


Figura 4.15: Valores SHAP para motores AB a NC2 con XGBoost y LSH

En el primer gráfico de la Fig. 4.15, podemos ver en un diagrama de barras apiladas, la contribución de cada variable a cada una de las clases. Por ejemplo, para la clase R5 (marrón),

las variables que más contribuyen son x1000 y x1250. Si miramos su gráfico particular, podemos ver para cada observación un gradiente de color, mostrando si tomó valores altos (colores rojos) o bajos (azules) en esa variable. En el eje X figura el valor SHAP de ese punto para esa variable, es decir, la influencia de esa variable en la clasificación de ese punto.
A continuación se muestra un análisis para cada clase por separado.

1. *R1*: A la vista del gráfico conjunto,se deduce de forma clara que la variable más importante fue x500. Mirando el gráfico particular, vemos que medidas altas en el instante x500 motivaron la como NO R1, y medidas bajas la clasificación como R1.

2. *R2*: A la vista del gráfico conjunto, se ve que las variables más importantes fueron x250 y x1750. Mirando su gráfico particular podemos decir que valores altos en el instante x250 motivaron la clasificación como R2 y valores bajos como NO R2, y en x1750 valores altos influyeron en contra de esta clase mientras que valores altos a favor.

3. *R3*: Las variables más importantes fueron x1000, x250 y x500. Valores bajos en estas tres variables motivaron clasificar como R3 y valores altos incitan a clasificar como no R3.

4. *R4*: Las variables más influyentes fueron x1250 y x1500. Valores altos de estas variables motivaron clasificar como R4, y valores bajos como no R4.

5. *R5*: Valores altos en x1000 x1250 y x750 (en general para todas) motivaron clasificaciones en R5, y valores bajos al contrario.

Conclusiones sobre el inversor AB

Tanto los Shapley Values como el F-score con la visualización nos llevan en la misma dirección. Las variables más influyentes son x250 para motores en estado R2 (en las que los motores en este estado toman valores altos), x500 para motores en estado R1 (en la que motores en este estado toman valores bajos) y x1000 y x1250 para separar motores R4 y R5 (en las que estos toman valores más altos).
Volvamos a los datos originales. Veamos qué pasa al hacer un corte transversal a la serie temporal de las corrientes medidas, separando por clase, en los instantes de tiempo que nos indica el modelo.

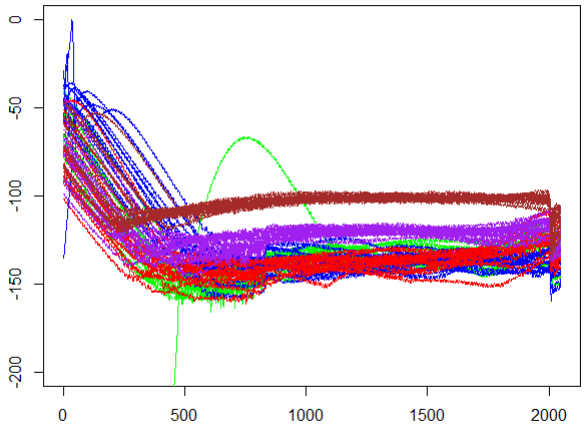


Figura 4.16: Secuencia de los valores del armónico LSH para inversores AB a Nivel de Carga 2

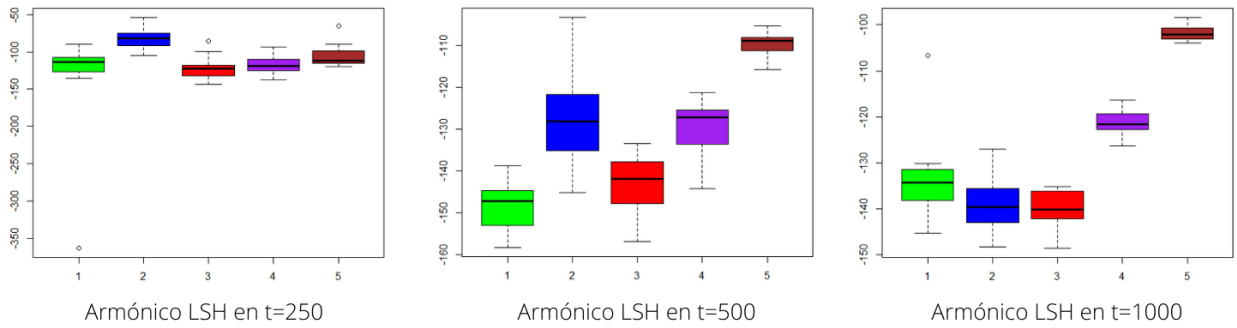


Figura 4.17: Distribución de los valores del armónico LSH por clase en las variables más importantes en el modelo sobre los datos originales de inversores AB a NC2

Como podemos ver, el modelo ha encontrado cortes en los que se separa muy bien la clase R2(azul) de las otras (t=250). Esto es interesante porque como vimos en la Fig 3.2, R1, R2 y R3 eran las más difíciles de discernir.
Así mismo, en instante t=1000, R4 y R5 están claramente separadas, pero con t=500 y t=1000 parece estar separando otra vez R1, que toma valores ligeramente distintos a los de R2 y R3 en estas dos variables

4.4.2 Inversor ABB

Para el inversor ABB, teníamos que la tasa de error más baja se alcanzaba para medidas a nivel de carga 1 , proporcionando al clasificador la información del armónico LSH y por tanto clasificando con XGBoost. La tabla 4.9 muestra el promedio de 20 matrices de confusión sobre

el test para este clasificador, entrenando con 50 observaciones y reservando 25 para test de forma estratificada.

ABB	**Predicho**					
Real		*R1*	*R2*	*R3*	*R4*	*R5*
	R1	2.7	0.55	1.35	0	0.4
	R2	0.65	3.95	0.05	0.35	0
	R3	0.8	0.05	4.15	0	0
	R4	0	0	0	5	0
	R5	0	0	0	0	5

Tabla 4.9: Matriz de confusión promediada en 20 repeticiones para ABB

ABB	**Predicho**					
Real		*R1*	*R2*	*R3*	*R4*	*R5*
	R1	0.54	0.11	0.27	0	0.08
	R2	0.13	0.79	0.01	0.07	0
	R3	0.16	0.01	0.83	0	0
	R4	0	0	0	1	0
	R5	0	0	0	0	1

Tabla 4.10: Matriz de confusión (condicionada por filas) promediada en 20 repeticiones para ABB

Importancia de las variables

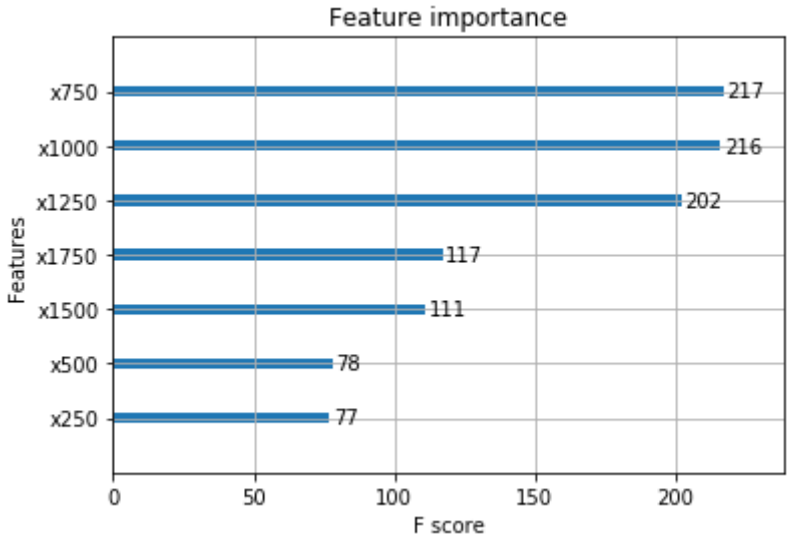


Figura 4.18: Importancia de las variables

Estudio de las variables

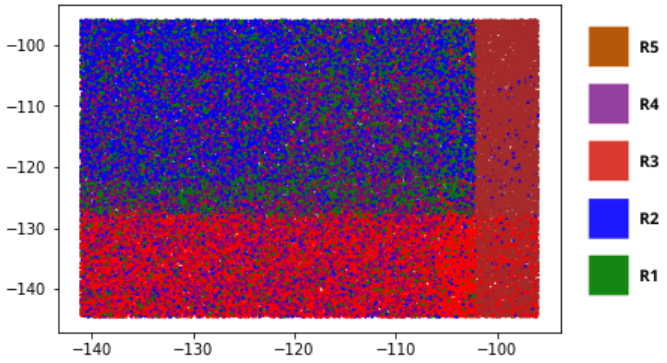


Figura 4.19: Proyección en las variables x750 y x1000

Como podemos ver, se aprecian dos franjas para las clases R3(rojo) y R5(marrón), y otra zona para R1,R2 y R4. Sin embargo, vamos a verlo en 3D para hacernos una mejor idea de cómo está separando esa franja Azul-Verde-Morada

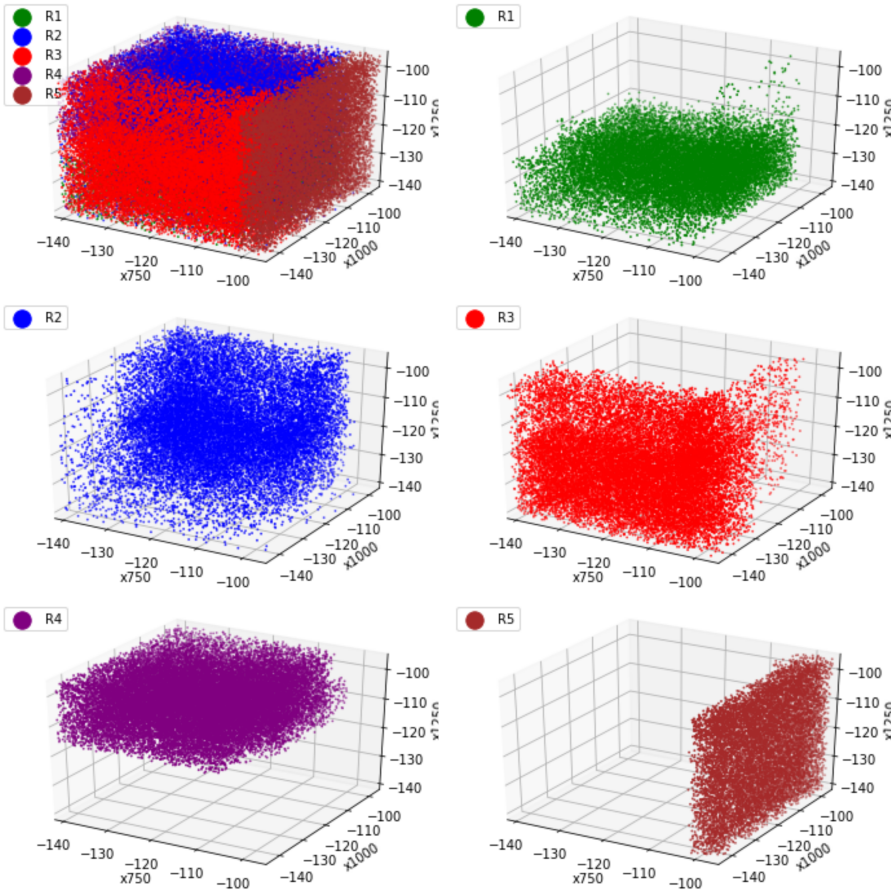


Figura 4.20: Distribución de los puntos por clases en las variables x750 x1000 y x1250

Como vemos, esa región está separada en tres niveles, valores bajos para la variable x1250 indican una mayor presencia de motores en estado R1 (verdes), en valores medios dominan los R2(azules) y en valores altos motores en estado R4(morados).

Podemos comprobar esto haciendo una proyección para x750 y x1000 cuando x1250 toma valores inferiores a -130, a diferencia de la Fig 4.19 , predominan los valores verdes.

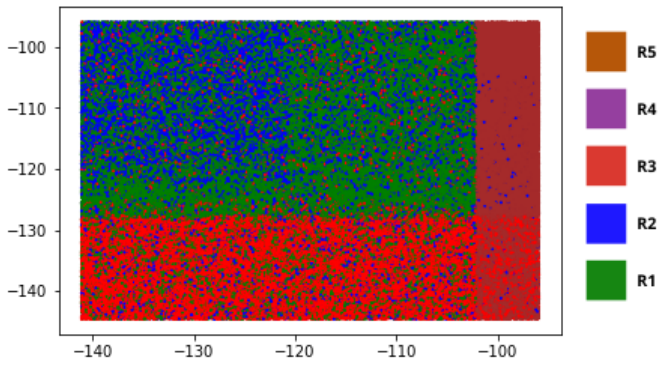


Figura 4.21: Proyeccion en las variables x750 y x1000 cuando x1250< -130

SHAP para motores ABB

A continuación (Fig. 4.22) se presenta un estudio basado en SHAP para motores

- *R1*: Valores bajos (puntos azules) tienen un SHAP alto y viceversa, es decir, que tomar valores bajos en esta variable favorece la clasificación del motor como en estado R1
- *R2*: En el caso de la clase R2 en la sección anterior vimos que es la más compleja de separar. En los SHAP esto se refleja, parece que la importancia de las variables está bastante repartida. De hecho en la variable x500 los SHAP indican que valores bajos motivan la NO clasificación como R2, podríamos estar hablando de que el algoritmo tome R2 por descarte.
- *R3*: Para esta clase la variable más importante es x1000, en la que valores bajos motivan la clasificación como R3, y valores altos la no clasificación. x1500 también parece tener impacto en este estado de motores en la misma dirección.
- *R4*: En el caso de R4 parece que la clave está en la variable x1250. Motores que tomen valores altos en esta variable se clasifican como dañados. Motores que tomen valores bajos serán clasificados como NO R4.
- *R5*: La idea es la misma que la de R4 pero en variables de tiempo anteriores o más tempranas(valores altos en x750 motivan la clasificación como R5, es decir, grave deterioro). Esto nos puede estar indicando que los motores R4 tardan más en manifestar en sus medidas ese deterioro que induce un valor alto en la medida de los armónicos.

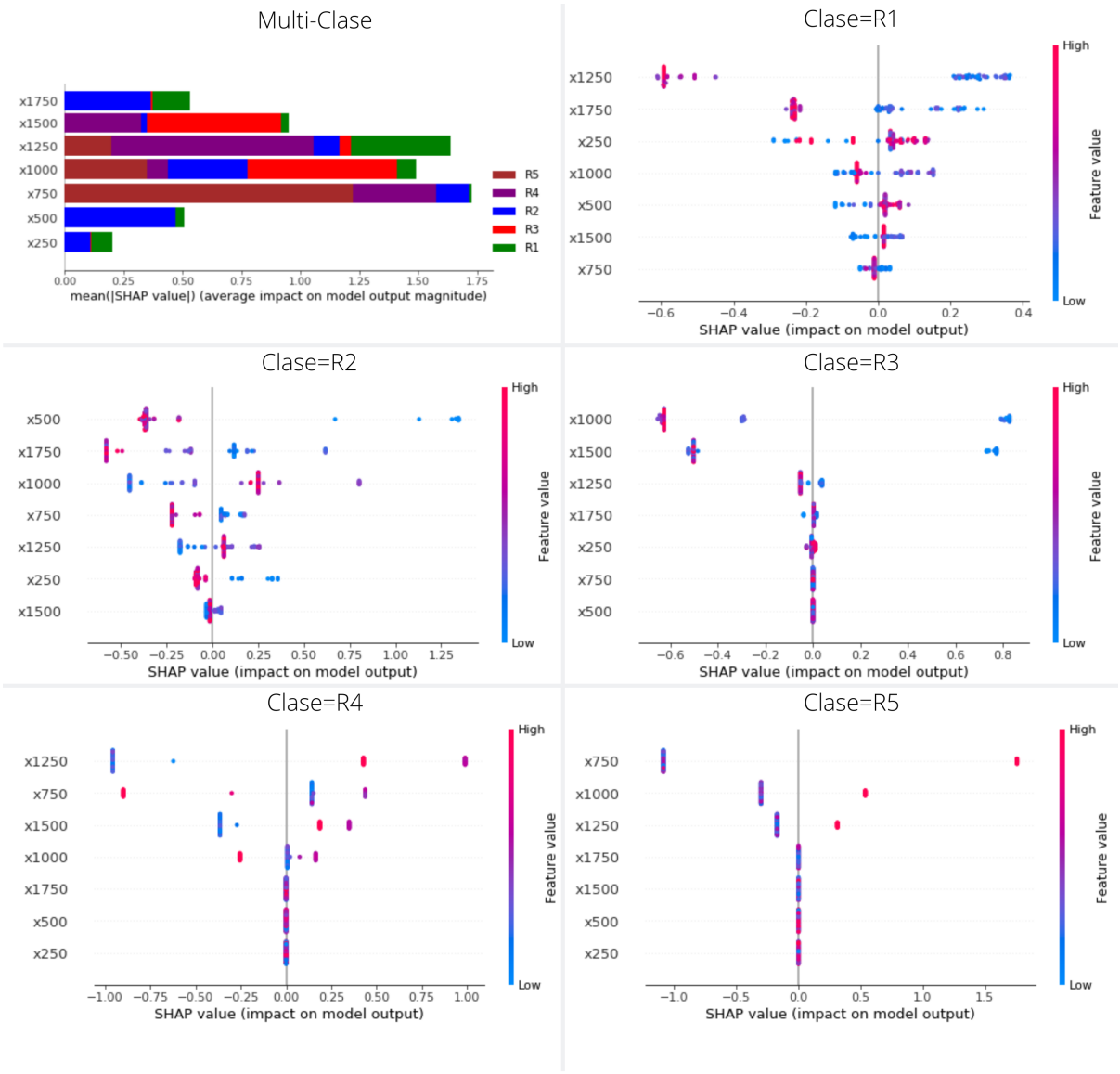


Figura 4.22: Valores SHAP para motores ABB a NC1 con XGBoost y LSH

Conclusiones sobre el inversor ABB

En este caso, las variables más importantes parecen estar en los momentos finales. Así como para motores AB las variables x250, x500 y x1000 son las más importantes, en este caso son x750, x1000 y x1250, seguidas por x1500 de cerca tanto en el F-score como en SHAP.
Podemos ver sin embargo que en estos motores, de forma clara, la separación en R3 es más fácil que en motores AB, concretamente en x1000 y en x1250, mientras que en AB se separaba mejor la clase R2. Si volvemos a los datos originales podemos verificar esto:

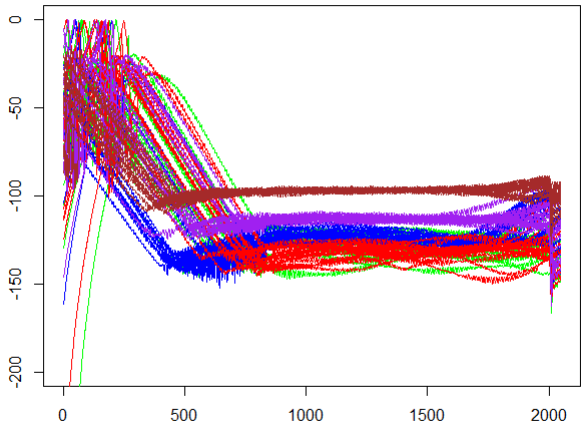


Figura 4.23: Secuencia de los valores del armónico LSH para inversores AB a Nivel de Carga 2

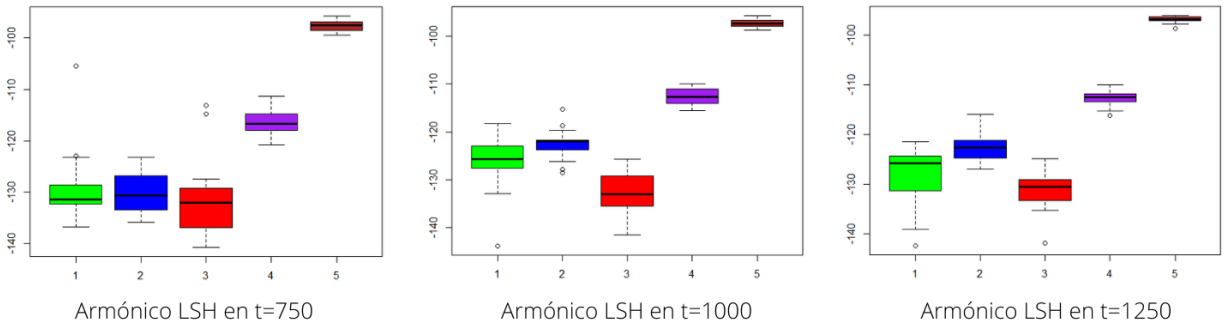


Figura 4.24: Distribución de los valores del armónico LSH por clase en las variables más importantes en el modelo sobre los datos originales de inversores ABB a NC1

Aquí podemos apreciar como en el instante t=1000 , la clase R3 se separa del resto tomando valores más bajos. Las clases R4 y R5 están completamente separadas y parece que en este inversor discernir entre estado de deterioro R1 y R2 no es tan fácil, no parece separarse clara-

mente en ninguna variable salvo en t=1000, aunque como vimos en la Figura 4.20 el modelo parece haber encontrado una separación con las variables mencionadas.

4.4.3 Inversor TM

El inversor TM, siguiendo la tónica de este TFG, tiene un comportamiento distinto al de AB y ABB. Para clasificar este tipo de inversores, es mejor proporcionar la información de los dos armónicos a nivel de carga 2, y utilizar DART (recordemos, XGBoost con Dropout), para clasificar. La tabla 4.11 muestra el promedio de 20 matrices de confusión sobre el test para este clasificador, entrenando con 50 observaciones y reservando 25 para test de forma estratificada.

TM	**Predicho**					
Real		*R1*	*R2*	*R3*	*R4*	*R5*
	R1	4.7	0	0.3	0	0
	R2	0.05	3.3	1.65	0	0
	R3	0	1.45	3.05	0.2	0.3
	R4	0	0	0	5	0
	R5	0	0	0	0.1	4.9

Tabla 4.11: Matriz de confusión promediada en 20 repeticiones para TM

TM	**Predicho**					
Real		*R1*	*R2*	*R3*	*R4*	*R5*
	R1	0.94	0	0.06	0	0
	R2	0.01	0.66	0.33	0	0
	R3	0	0.29	0.61	0.04	0.06
	R4	0	0	0	1	0
	R5	0	0	0	0.02	0.98

Tabla 4.12: Matriz de confusión (condicionada por filas) promediada en 20 repeticiones para TM

Importancia de las variables

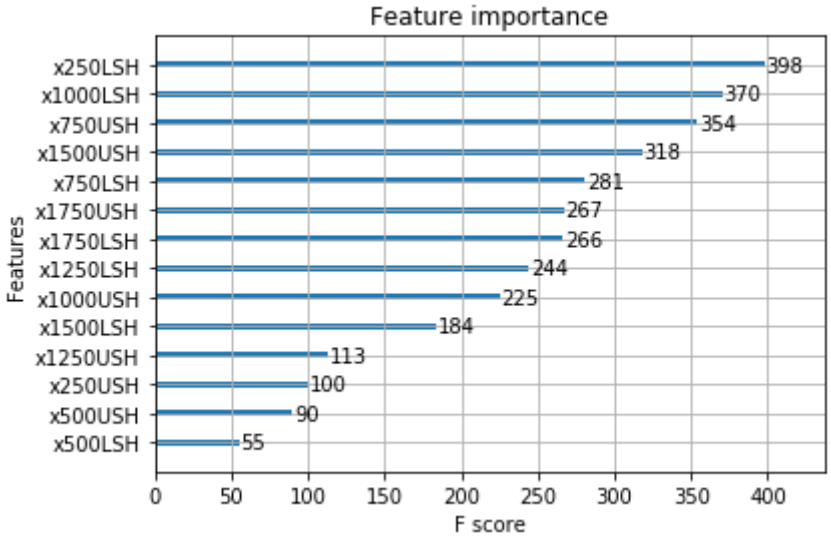


Figura 4.25: Importancia de las variables

Como podemos ver, en las variables más importantes se encuentran medidas de ambos armónicos. Es decir, el clasificador está aprovechando la información que para AB y ABB resulta innecesaria (e incluso perturba la clasificación empeorando la tasa de error al incluirla).
Vamos a ver la distribución de los puntos en torno a estas variables.

Estudio de las variables

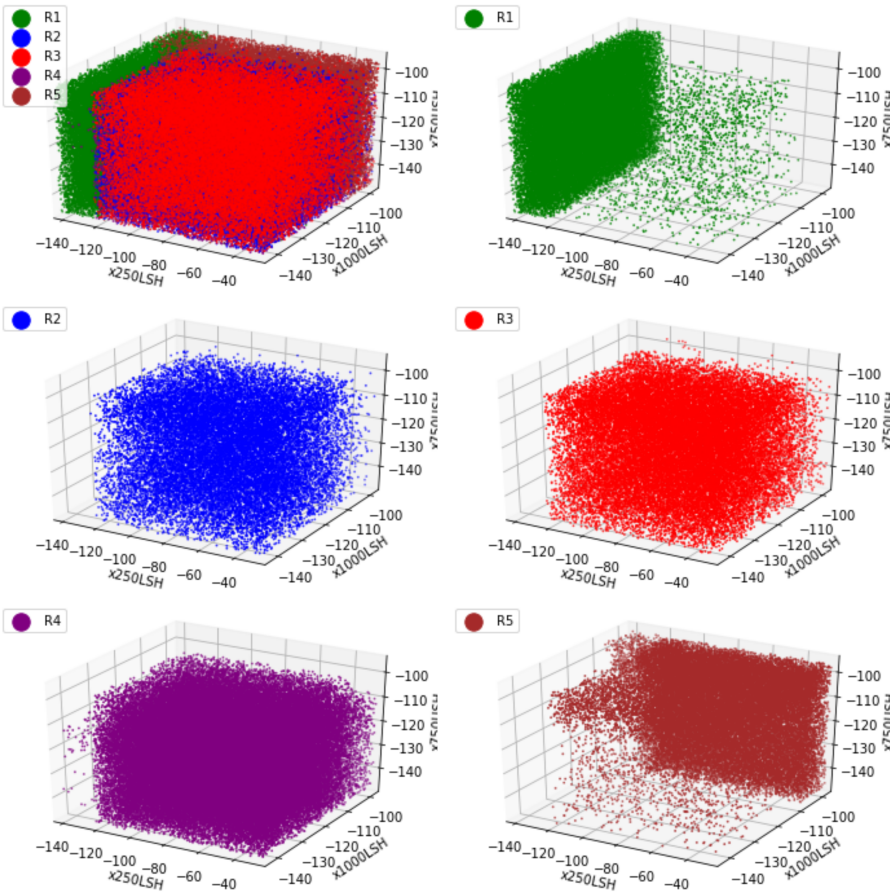


Figura 4.26: Distribución de los puntos por clases en las variables x250LSH x1000LSH y x750USH

Aquí podemos ver cómo clasifica perfectamente los motores R1 y R5 en dos regiones claramente separadas del resto. Sin embargo, R2, R3 y R4 parecen ocupar el mismo volumen. Dado que TMxNC2 era la combinación con la segunda tasa de error más baja (Fig. 4.7), ¿a qué se debe? La clave está en la importancia de las variables. En el caso de TM, no hay un salto tan diferenciado entre la tercera y cuarta variable (es decir, su importancia es similar, de hecho, la diferencia es bastante suave según va disminuyendo la importancia de las variables). Veamos la distribución de los puntos según la 2a,3ra y 4ta variable por importancia.

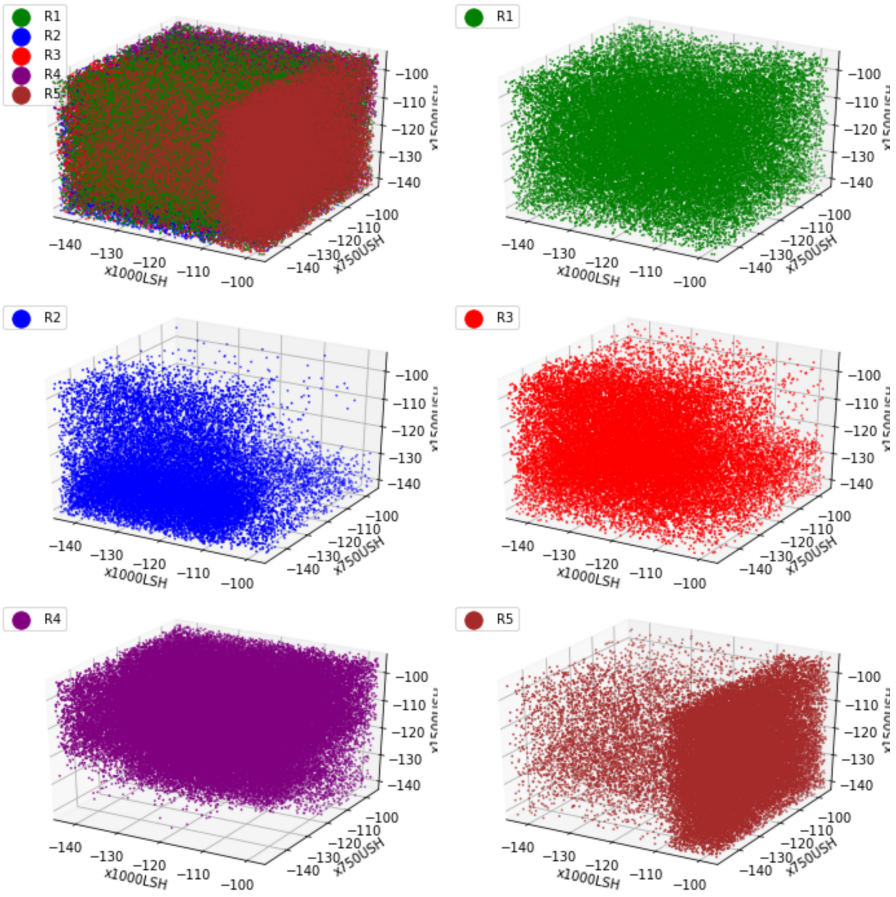


Figura 4.27: Distribución de los puntos por clases en las variables x1000LSH, x750USH y x1500USH

Aquí vemos que estas variables son las encargadas de separar las tres clases que entraban antes en conflicto. La masa de R4 está situada hacia valores superiores, mientras que la de R2 está en la esquina inferior izquierda (valores inferiores). R3 sigue entremezclándose con ambas, pero estará siendo separada en otra variable.

SHAP para motores TM

En este caso no se muestran los Shapleys estimados mediante TreeSHAP, sino mediante KernelSHAP. Esto es debido a que para XGBoost-Dart no está implementado el algoritmo TreeSHAP.

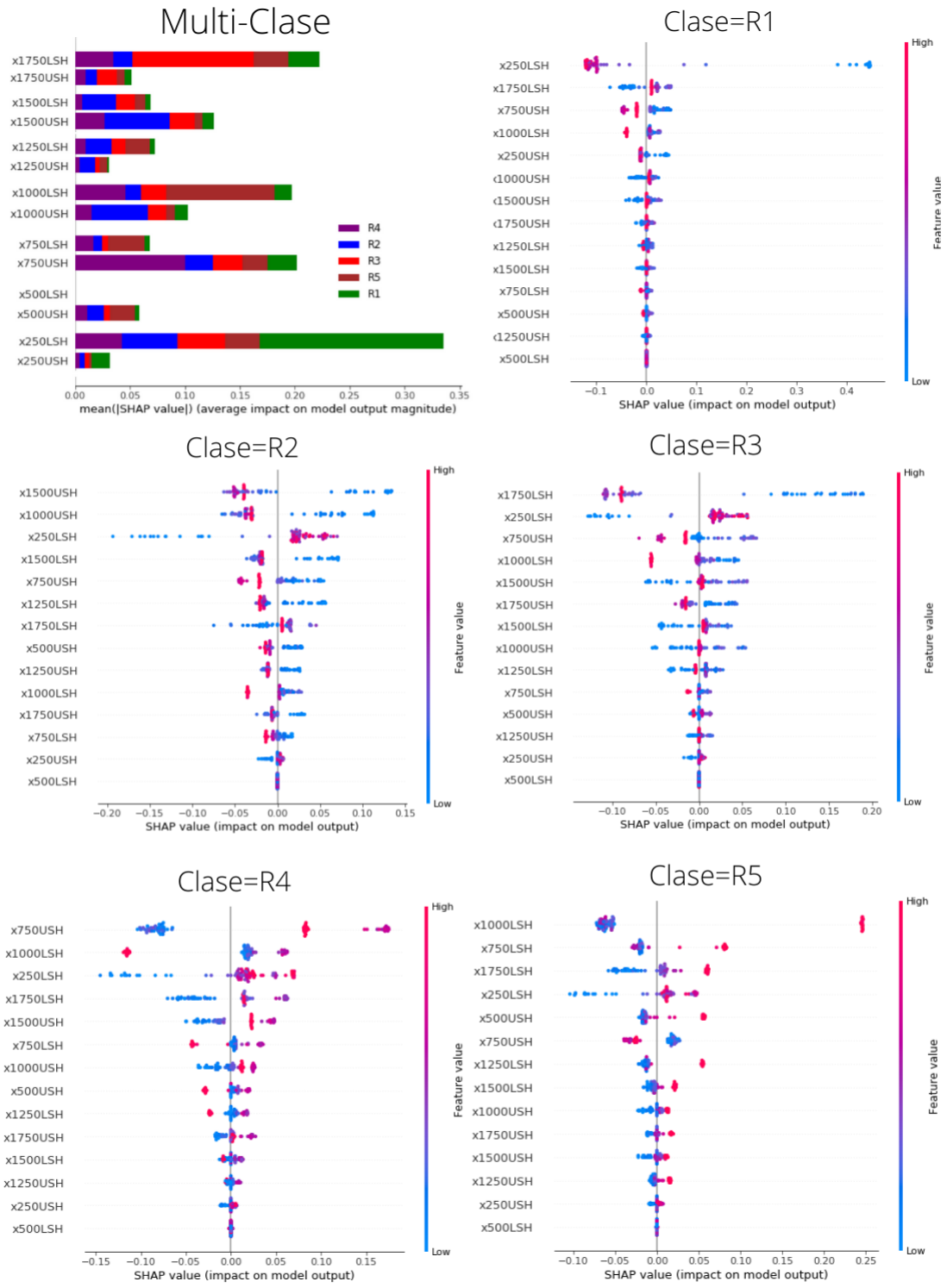


Figura 4.28: Valores SHAP para motores TM a NC1 con XGBoost-DART y Ambos Armónicos

- *R1*: Hay dos instantes de tiempo en el armónico LSH que son los más importantes. Valores bajos en x250LSH favorecen la clasificación en R1 (y valores altos favorecen la no clasificación), y en bastante menor medida valores bajos en x1750LSH favorecen la clasificación en R1.

- *R2*: La clave de este estado de deterioro parece estar en el armónico USH y en valores bajos de este en los instantes x1500USH y x1000USH, lo cual favorece la clasificación como R2. Así mismo, en x250LSH valores altos en esa variable favorecen la clasificación como R2, lo cual es extraño. Esto lo comentaremos en las conclusiones.

- *R3*: En el caso de este estado de deterioro medio, la variable x1750LSH juega un papel muy importante. Es interesante porque en los otros motores, las variables tan tardías no influían tanto para R3. Tomar valores bajos en esta variable favorece la clasificación en R3 y altos, en contra.

- *R4*: En este estado de deterioro la variable más influyente es x750USH Valores altos en esta juegan a favor de la clasificación en R4 y bajos, en contra. Sin embargo, en la segunda variable más influyente, es curioso que hay valores altos con shapleys altos y valores altos con shapleys bajos. Esto a priori parece contradictorio, pero la interpretación más factible es que valores bajos discriminan R4 vs R5 y valores altos R4 vs R123.

- *R5*: En el deterioro más grave, el armónico LSH fue suficiente para discriminarlo del resto, concretamente en los instantes x1000LSH y x750LSH, valores altos favorecen la clasificación en R5, y valores bajos en NO R5.

Conclusiones sobre el inversor TM

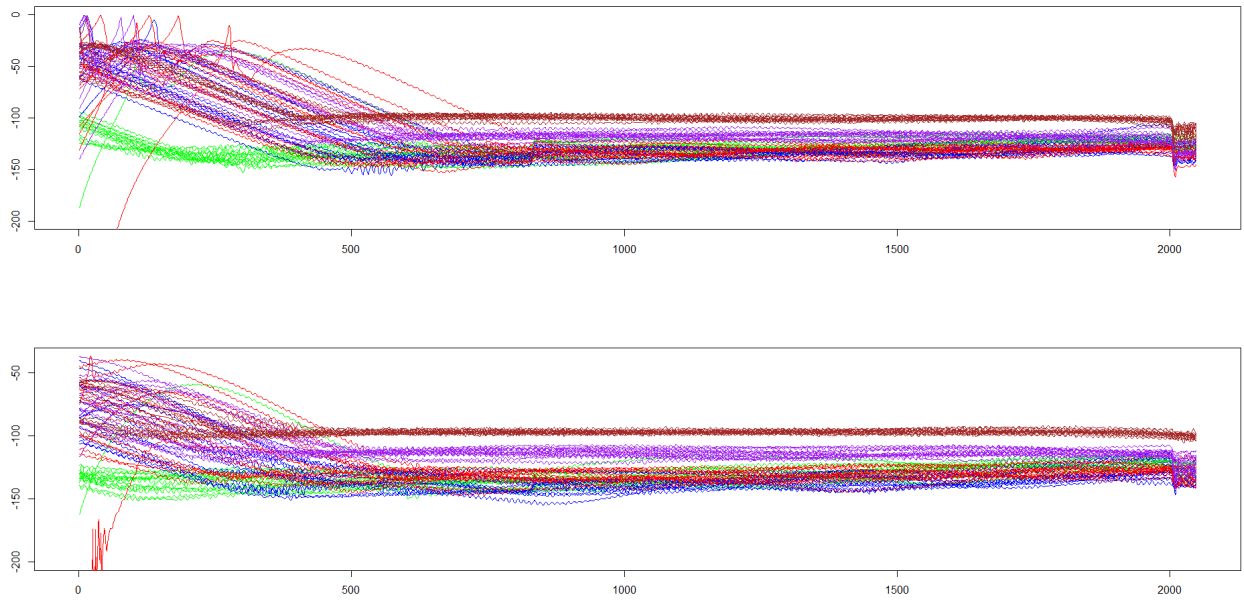


Figura 4.29: Serie temporal con las medidas de los armónicos LSH (arriba) y USH(abajo) para motores con inversor TM

Armónico LSH

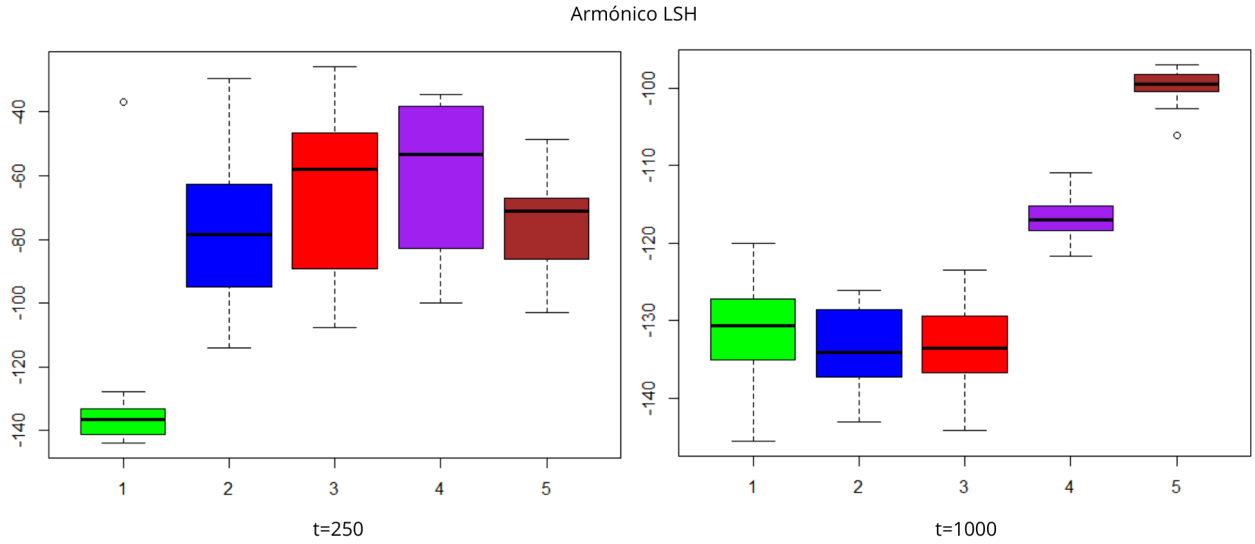


Figura 4.30: Distribución de valores del armónico LSH por clase

Como vemos en las variables originales, para el instante 250 del armónico LSH, a diferencia de los dos motores anteriores, la que se separa claramente en este momento es la clase R1 (verde), lo cual se vio reflejado en los SHAP, mientras que las otras cuatro no aprecen ser separables en esta variable. Además la amplitud de la caja R1 es mucho menor que las otras, lo cual nos puede estar indicando que la onda para este modelo de inversor, en condiciones de no deterioro del rotor, está ya estabilizada en los primeros instantes de tiempo.

Para t=1000, siguiendo el patrón general, se separan las clases 4 y 5 del resto.

Finalmente, en la sección de SHAP se comentó que había un comportamiento extraño en R3, la variable x1750LSH era la más influyente

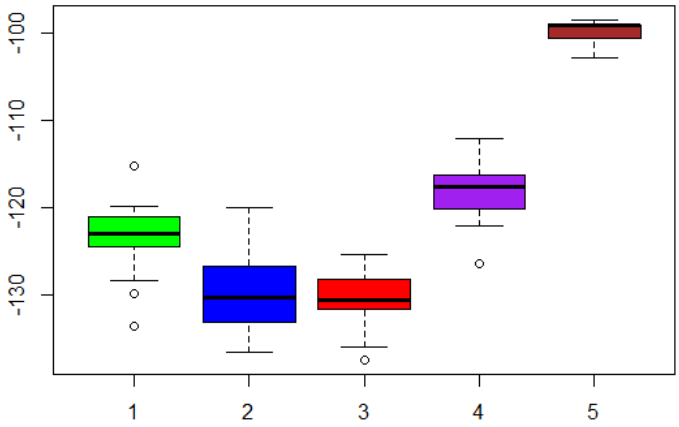


Figura 4.31: Distribución de valores del armónico LSH en t=1750 por clase

A la vista de los datos originales, parece que sí que R3 toma valores más bajos que R1 y R2. Este resultado es extraño, y se recomendaría una revisión por parte de alguien con conocimiento del dominio para buscar causas, ya que a priori un motor con R3 estaría más dañado que uno de R12 y por tanto, su valor en este armónico debería ser más alto.

Armónico USH

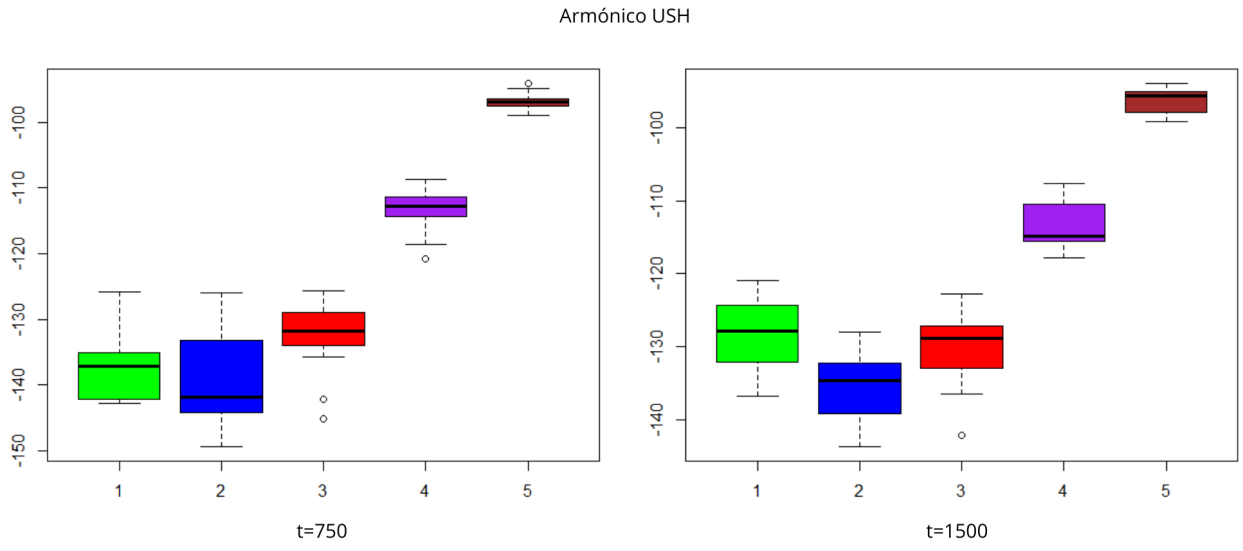


Figura 4.32: Distribución de valores del armónico USH por clase

Aquí vemos cómo se repite el hecho de que XGBoost (con DART en este caso), separa la clase R2 de R1 y R3 en t=1500, pero esta vez en el armónico USH. Es interesante también mencionar que para t=750 también parece estar separando R3, o al menos ha detectado que la caja es menos ancha (menor variabilidad) en ese instante de tiempo.

5. Análisis mediante técnicas Boosting isotónicas

5.1 Motivación

En este capítulo vamos a tratar los datos con tecnicas monótonas porque se sabe que, a priori, a mayor estado de deterioro de un motor, mayor será el valor de los armónicos inducidos, es decir, en el problema en el que nos encontramos, podemos suponer que la relación entre la medida de los armónicos de la corriente inducida y el estado de deterioro es monotóna creciente. Los métodos que tienen en cuenta la monotonía están diseñados para tener en cuenta esta información y, de ese modo, obtener mejores resultados que los métodos estándar que se consideraron en el capítulo anterior. [30]

5.1.1 Relaciones de Isotonía de las variables con la respuesta

Se dice que una función $f : X \to \mathbb{R}$ es isotónica con respecto al orden $\preceq$ si

$$\forall x, y \in X, x \preceq y \Rightarrow f(x) \leq f(y).$$

De este modo si, por ejemplo, $X = \mathbb{R}$ y $\preceq$ es el orden habitual las funciones isotónicas con respecto a este orden son exactamente las funciones monótonas crecientes y las funciones monótonas decrecientes son aquellas funciones f para las que $-f$ es isotónica con respecto a este orden. Aunque en este trabajo hablaremos de isotonía en general, nos referiremos, con evidente abuso de lenguaje y a menos que se indique lo contrario, a una función como isotónica cuando sea o bien monótona creciente o monótona decreciente.
Trasladado al contexto de regresión, diremos que una función de regresión es isotónica (monótona creciente o decreciente) con respecto a una variable regresora si la curva ajustada es siempre creciente o decreciente con respecto a dicha variable regresora.

5.2 Métodos e Implementación de la isotonía

Tanto la librería de XGBoost como la de LightGBM permiten la introducción de restricciones de monotonía en el ajuste del modelo mediante el paso como parámetro de un vector $\{0,1\}^p$ donde p es el número de variables, y los 1 representan que la variable p es monótona por lo que repetiremos la metodología del punto anterior para tratar de mejorar lo resultados obtenidos. Además, añadiremos otras técnicas boosting como *ASILB,AMILB,CSILB,CMILB* basadas en LogitBoost (paquete `isoboost` [30]) isotónico y técnicas isotónicas no boosting basadas en discriminante lineal (paquete `dawai` [31]). A continuación se describe cómo implementan cada una de estas familias de métodos la isotonía, según los describen los autores en los trabajos originales.

5.2.1 Monotonía en isoboost

Los algoritmos de este paquete son una versión isotónica de LogitBoost, el cual se explicó en la sección 2.7. Principalmente, tenemos dos familias de métodos:

SILB

Los métodos basados en Simple Isotonic LogitBoost imponen que en la construcción del ensemble $F(x) = \sum_{m=1}^{M} f_m(x_{j_m})$, siendo M el número de iteraciones (es decir, en cada iteración se utiliza una variable). El algoritmo de construcción del ensemble es el siguiente:
Sea $I \cup D$ el conjunto de índices de variables sobre las que se quiere imponer restricciones, donde I son los índices de las variables con monotonía creciente y D los de las variables con monotonía decreciente

1. Comenzar con $w_i = \frac{1}{n}, i = 1, \ldots, n, F(\mathrm{x}) = 0$ y $\pi(\mathbf{x}_i) = \frac{1}{2}$ for $i = 1, \ldots, n$

2. Repetir M veces:

   2.1 Calcular $w_i = \pi(\mathbf{x}_i)(1 - \pi(\mathbf{x}_i)), z_i = \frac{y_i - \pi(\mathbf{x}_i)}{w_i}, i = 1, \ldots, n$

   2.2. Para $j = 1, \ldots .d$:

   a) Si $j \in I \cup D$. ajustar una regresión isotónica (creciente para I y decreciente para D) ponderada $f_j(x)$ of z_i to x_{ij} con pesos w_i

   b) Si $j \notin I \cup D$, ajustar un *stump* binario $f_j(x)$ por mínimos cuadrados para z_i sobre x_{ij} con pesos w_i

   2.3. Considerar $h = \arg\min_{j \in \{1,\ldots,d\}} \sum_{i=1}^{n} w_i (z_i - f_j(x_{ij}))^2$, y actualizar $F(\mathbf{x}) = F(\mathbf{x}) + f_h(x_h)$ y $\pi(\mathbf{x}) = \frac{1}{1+e^{-F(\mathbf{x})}}$

3. Clasificar como 0 si $\pi(x) < 0.5$, 1 si $\pi(x) \geq 0.5$

MLIB

En cambio, los métodos basados en Multiple Isotonic LogitBoost imponen que $F(x) = \sum_{j=1}^{p} f_j(x_j)$ (es decir,se utilizan todas las variables en cada paso). El algoritmo es el siguiente:

1. Comenzar con pesos $w_i = 1/n$ $i = 1, \ldots, n, F(x) = 0$ y $\pi(x_i) = 0.5, i = 1, \ldots, n$
2. Repetir hasta que converja:

   Calcular $w_i = \pi(\mathbf{x}_i)(1 - \pi(\mathbf{x}_i)), z_i = \frac{y_i - \pi(\mathbf{x}_i)}{w_i}, i = 1, \ldots, n$

   Ajustar una regresión aditiva ponderada mediante el algoritmo de *backfitting* $F(x) = \sum_{j=1}^{d} f_j(x_j)$ de z_i sobre x, con pesos w_i, siendo $f_j(x_j)$ una regresión isotónica si $j \in I \cup D$ o una regresión lineal si $j \notin I \cup D, j = 1, \ldots$. d

   Actualizar $\pi(\mathbf{x}) = \frac{1}{1+e^{-F(\mathbf{x})}}$
3. Clasificar como 0 si $\pi(x) < 0.5$, 1 si $\pi(x) \geq 0.5$. Como F(x) es una suma de d términos, el riesgo de sobreajuste no es grande como sucede con otras técnicas de boosting al emplear muchas iteraciones.

Dentro de esos dos métodos, los que se exponen en este TFG del paquete `isoboost` se basan en dos familias de modelos de *odds* proporcionales:

- Categorías adyacentes: Donde se impone que $log(\frac{\pi_j}{\pi_{j+1}}) = \alpha_j + X\beta$
- Probabilidad acumulada: Donde se impone que $log(\frac{P(Y \leq j)}{P(Y \geq j)}) = \alpha_j + X\beta$

Combinando estas dos familias (**A***djacent categories* y **C***umulative probabilities*) con los métodos **MILB** y **SILB**, obtendremos los cuatro métodos que se han estudiado **ASILB, AMILB, CSILB, CMILB** de este paquete. Cabe destacar que hasta ahora, en aras de la comprensibilidad, se ha mostrado el problema de clasificación binario. A la hora de pasar al problema multiclase, hay que tener en cuenta que tendremos un problema de regresión logística multiclase en cada paso de la construcción del modelo boosting. Para una información más completa véase [30]

5.2.2 Monotonía en dawai

En este paquete, se proporcionan las funciones `rlda,rqda` para estimar un discriminante lineal (ver sec. 2.1.1) con restricciones. Estos métodos no son métodos boosting como los que se están considerando en el resto de este trabajo sino que se basan en la imposición de restricciones de orden sobre las medias de las poblaciones. Se consideran en este trabajo como punto de comparación de la eficiencia de los distintos métodos isotónicos.
Debido a que la muestra no es grande (más pequeña aún porque recordemos que para combinación de *Model x Level x Band* se entrena un clasificador), no es posible utilizar el método `rqda` (discriminante cuadrático con restricciones).

Imposición de orden sobre las medias

La imposición de las restricciones de monotonía entre grupos se realiza mediante la obtención de la proyección del estimador μ sobre un cono C definido por las restricciones $a^\top$ impuestas sobre las medias . Es decir, siendo p el número de predictores se busca el vector $\mu^\top$ tal que:

$$\left(\mu_1^\top, \ldots, \mu_k^\top\right)^\top \in C = \left\{x \in \mathbb{R}^{pk} : a_j^\top x \geq 0, j = 1, \ldots, q\right\} \tag{5.1}$$

Para obtener este estimador de medias proyectado, se realiza el siguiente procedimiento iterativo en función de γ:

$$\widehat{\mu}^{\gamma(m)} = \boldsymbol{p}_{S_*^{-1}}\left(\widehat{\mu}^{\gamma(m-1)}|C\right) - \gamma \boldsymbol{p}_{S_*^{-1}}\left(\widehat{\boldsymbol{\mu}}^{\gamma(m-1)}|C^P\right), m = 1, 2, M \tag{5.2}$$

donde $\widehat{\mu}^{\gamma(0)} = \left(\bar{Y}_1^\top, \ldots, \bar{Y}_k^\top\right)^\top \in \mathbb{R}^{pk}$, es decir, el vector de medias para cada grupo en las variables $X_1, ...X_p$, $p_{S_*^{-1}}(Y|C)$, es la proyección de $Y \in \mathbb{R}^{pk}$ sobre el cono C usando la métrica definida por la matriz $S_*^{-1} = \left[\text{diag}\left(\frac{S}{n_1}, \frac{S}{n_2}, \ldots, \frac{S}{n_k}\right)\right]^{-1}$ y el cono $C^P = \left\{y \in \mathbb{R}^{pk} : y^\top S_*^{-1} x \leq 0, x \in C\right\}$ es el cono polar de C.
De este modo, tendremos una imposición de la monotonía entre las clases aplicando este estimador proyectado en la fórmula de la sección 2.1.1

5.2.3 Monotonía en XGBoost y LightGBM

La monotonía en estas dos librerías se hace a nivel de árbol, de la siguiente forma:
Supongamos que tenemos como variable explicativa X, sobre la que queremos imponer una restricción de monotonía, esto es, para todo par de valores x_1, x_2 tales que $x_1 < x_2$, $w_n(x_1) < w_n(x_2)$ (siendo w el valor de la hoja del split en la que caería cada observación a la altura n del árbol).

1. En el nodo raíz ($nodo_0$) no se aplican restricciones ya que $w(x) = w_0 \, \forall x$.

2. Si X es seleccionada para crear una disyunción en el árbol (en el $nodo_n$), se comprueba si el umbral elegido para esa disyunción (u_{lim}) cumple la restricción de monotonía ($w_{izquierdo,n+t}(x) < w_{derecho,n+t}(x)$). Si no lo cumple, se desecha el valor umbral para ese split y se busca otro.

3. Si la variable X es utilizada otra vez en un split más profundo (llamémoslo $nodo_{n+t}$) del árbol, de cara a garantizar que la monotonía se cumpla en todo el árbol, se fuerza que los splits izquierdo y derecho cumplan la monotonía de tal forma que su peso sea menor que la media de su padre y el hermano de su padre. Es decir, se arrastra el valor medio de los ancestros de tal forma que, además de la restricción del paso 2 $w_{izquierdo,n+t}(x), w_{derecho,n+t}(x) < media(w_{izquierdo,n}(x), w_{derecho,n}(x))$

5.3 Resultados monótonos vs no monótonos

Una vez obtenidas las tasas de error (disponibles en la Tabla A.1 de los anexos) de los experimentos con monotonía, vamos a compararlas con las obtenidas en el apartado anterior. Para obtener los resultados se impusieron restricciones de monotonía a todas las variables explicativas.

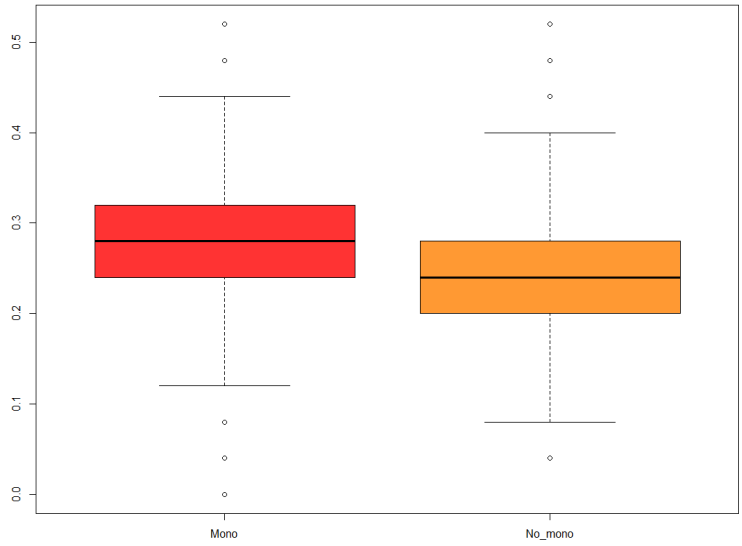


Figura 5.1: Distribución de la tasa de error para algoritmos monótonos vs no monótonos

En general, sin discernir por algoritmo, podemos ver que las restricciones monotónicas no parece que proporcionen una mejor clasificación. Si visualizamos separando por algoritmo:

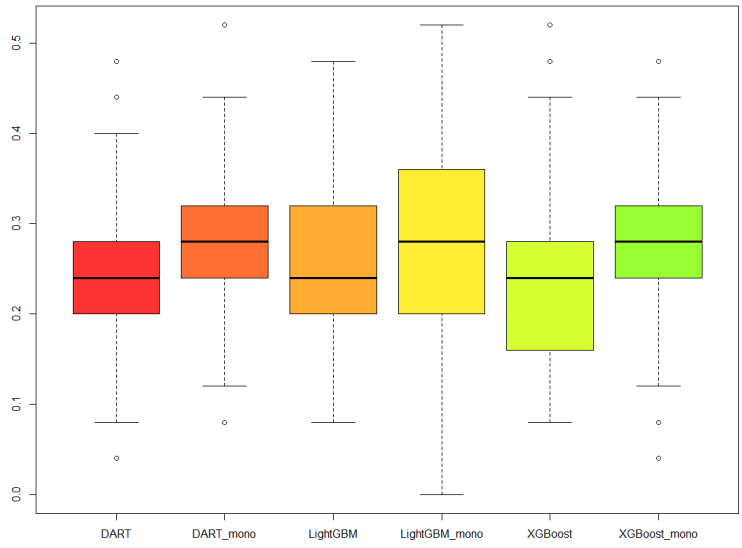


Figura 5.2: Distribución de la tasa de error para algoritmos monótonos vs no monótonos según el algoritmo

Vemos que en ningún caso la versión monotónica mejora a la no monotónica. A continuación se analizarán las causas de por qué esta relación de no monotonía que a priori sí existe no mejora la predicción del estado de deterioro de los motores.

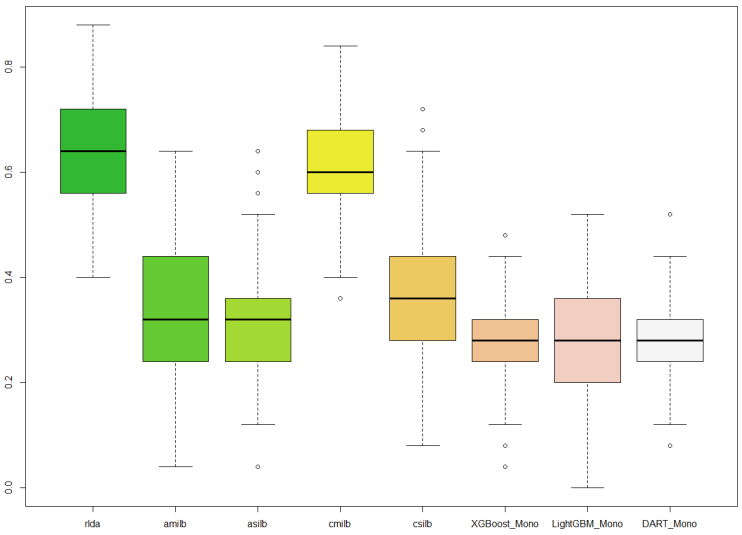


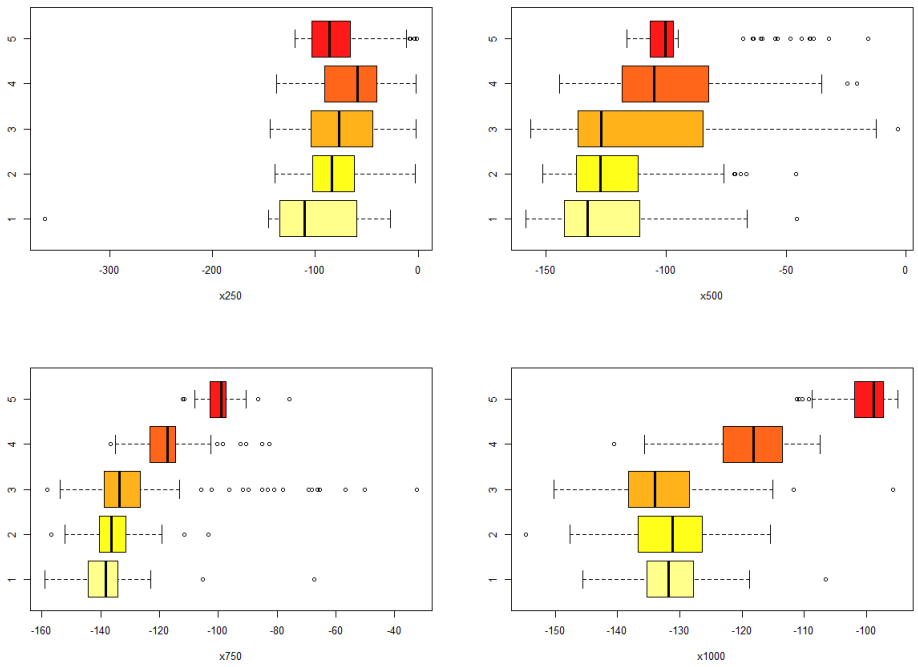
Figura 5.3: Clasificadores basados en LogitBoost isotónico v.s. clasificadores basados en gradient boosting isotónico

Si comparamos los nuevos algoritmos planteados frente al los que ya teníamos basados en árboles (en versión monotónica) tenemos que no los mejoran (por tanto tampoco mejoran a las versiones no monotónicas de estos, que como hemos visto eran mejores)

5.4 Diagnósticos sobre monotonía

Dado que a priori la relación es monótona, en esta sección se va a estudiar el por qué las restricciones de monotonía parecen no estar funcionando. De cara a entender el por qué el incluir restricciones de monotonía no ayuda a una mejor predicción, a continuación se muestran, en primer lugar, una serie de gráficos descriptivos para ver cómo se distribuyen las clases para los distintos valores del tiempo.
En segundo lugar, mediante medidas de isotonía multivariantes, estudiaremos qué valores toman estos estadísticos de isotonía en nuestro problema y cómo se comparan con los valores de otros problemas en los que sí que sabemos que las restricciones isotónicas mejoran la capacidad predictiva de nuestros modelos.

5.4.1 Distribución de las clases (deterioro) en el armónico LSH según los instantes de tiempo



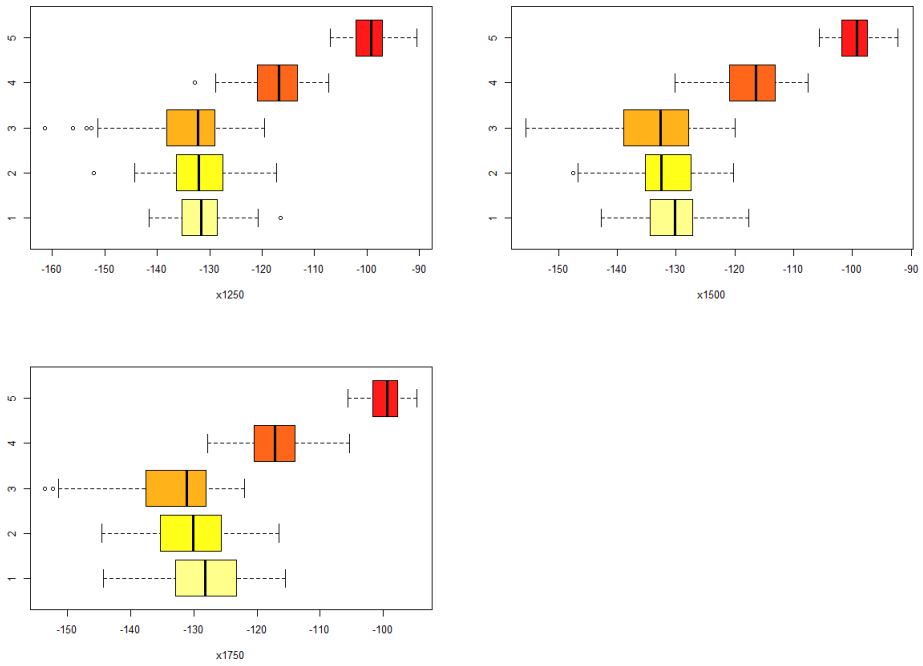


Figura 5.4: Distribución de valores según las clases en el armonico LSH

En los gráficos anteriores podemos ver cómo se distribuyen las clases (en un gradiente de color según el deterioro) para los valores en los puntos temporales que habíamos tomado como variables explicativas.
Si la relación entre las variables y la respuesta fuese monotónica, deberíamos ver una tendencia hacia una diagonal en los gráficos, esto es, valores de rojo más intensos (más deterioro) hacia valores más altos de la variable.
Como se aprecia en los gráficos, esto no se cumple de forma sistemática. Las tres primeras clases parece que salvo para el instante x750 no cumplen esta tendencia diagonal. Sin embargo, las clases R4 y R5 sí que parecen cumplirla (como viene siendo habitual, el comportamiento de R4 y R5 es distinto a las otras)

5.4.2 Distribución de las clases (deterioro) en el armónico USH según los instantes de tiempo

En el armónico USH sucede lo mismo que en armónico LSH, solo que en este no hay ni si quiera una variable en la que se refleje la supuesta monotonía como en x750 para el armónico LSH.

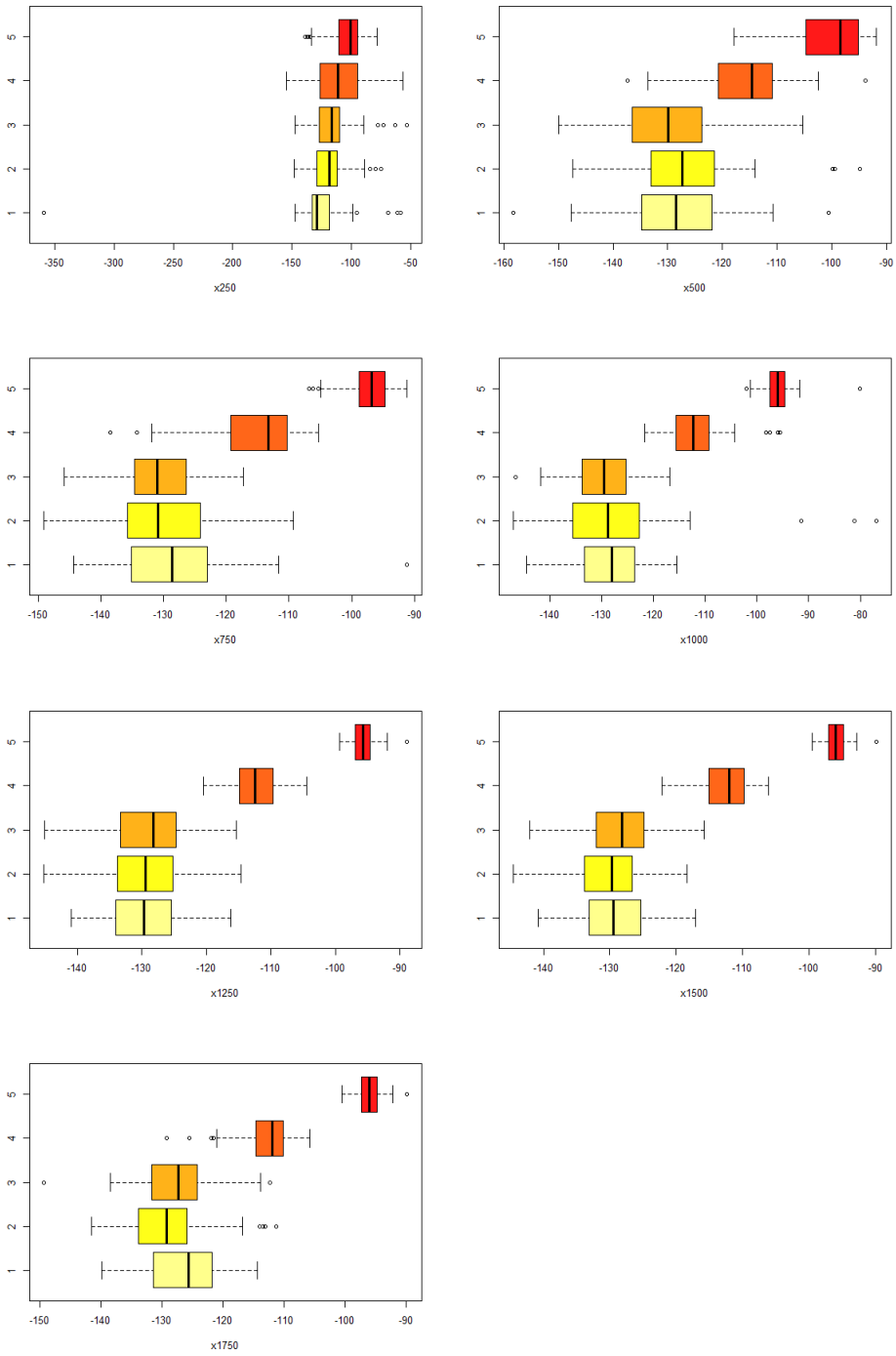


Figura 5.5: Distribución de valores según las clases en el armonico USH

5.4.3 Medidas de Isotonía

Existen métricas para el cálculo de isotonía de las variables explicativas respuesta respecto a las explicativas. Según un meta-estudio sobre artículos de algoritmos isotónicos [32], la más

popular es el **Non-Monotonic Index** [33] o NMI:

$$NMI = \frac{1}{n(n-1)} \sum_{x \in D} NClash(x) \tag{5.3}$$

Donde $NClash(x)$ es el número de pares *clash* o discordantes con x (aquellos en los que las variables explicativas cumplen todas ellas la monotonía respecto a las de x pero la respuesta no). Sin embargo, esta métrica es dependiente del número de pares comparables, el cual está influenciado por el número de variables (a más variables, menos pares comparables). Por ello también es interesante analizar otros estadísticos que se proponen en el artículo antes citado como el de Goodman-Kurskal:

$$\gamma_1 = \frac{S_+ - S_-}{S_+ + S_-} \tag{5.4}$$

$$S_- = \sum_{x \in D} NClash(x) \tag{5.5}$$

$$S_+ = \sum_{x \in D} NMonot(x) \tag{5.6}$$

Donde $NMonot(x)$ es el número de pares que cumplen las restricciones de monotonía respecto a x. También se propone otro índice, el *Frequency of Monotonicity* o **FOM**, siendo $\#P$ el número de pares totales:

$$FOM = \frac{S_+}{\#P} \tag{5.7}$$

Debido a la dependencia del número de variables empleadas, si se quieren utilizar subconjuntos de variables (debido a que cambia el número de pares comparables) en vez de todas las disponibles, estas métricas parecen insuficientes. A mayores de los propuestos en el artículo [32] queremos proponer dos métricas nuevas:
El porcentaje de pares comparables que cumplen la monotonía:

$$\gamma_3 = \frac{S_+}{S_+ + S_-} \tag{5.8}$$

El porcentaje de pares comparables respecto al total:

$$\gamma_4 = \frac{S_+ + S_-}{\#P} \tag{5.9}$$

En la tabla 5.1 podemos ver los valores que toman estos estadísticos para las variables consideradas por separado. En la tabla 5.2 podemos ver algunos subconjuntos de variables que, a la vista de los gráficos de la sección anterior podrían cumplir la monotonía:

Armónico	X250	X500	X750	X1000	X1250	X1500	X1750	**NMI**	γ_1	**FOM**	γ_3	γ_4
LSH	X							0.2662	-0.0647	0.4676	0.4676	1
		X						0.2205	0.118	0.559	0.559	1
			X					0.1668	0.3328	0.6664	0.6664	1
				X				0.1786	0.2856	0.6428	0.6428	1
					X			0.1694	0.3224	0.6612	0.6612	1
						X		0.1703	0.3187	0.6594	0.6594	1
							X	0.1775	0.2899	0.645	0.645	1
USH	X							0.2298	0.0806	0.5403	0.5403	1
		X						0.1817	0.2734	0.6367	0.6367	1
			X					0.1738	0.3049	0.6524	0.6524	1
				X				0.1676	0.3295	0.6647	0.6647	1
					X			0.1583	0.3668	0.6834	0.6834	1
						X		0.1552	0.3794	0.6897	0.6897	1
							X	0.167	0.332	0.666	0.666	1

Tabla 5.1: Estadisticos de monotonía para las variables consideradas una a una

Armónico	X250	X500	X750	X1000	X1250	X1500	X1750	**NMI**	γ_1	**FOM**	γ_3	γ_4
LSH	X	X	X	X	X	X	X	0.0263	0.698	0.2962	0.849	0.3488
		X	X	X	X			0.0495	0.6245	0.4285	0.8122	0.5276
			X	X	X			0.0773	0.5643	0.5548	0.7822	0.7093
USH	X	X	X	X	X	X	X	0.034	0.7046	0.3924	0.8523	0.4604
		X	X	X	X			0.0806	0.556	0.5647	0.778	0.7257
			X	X	X			0.0994	0.5043	0.6034	0.7522	0.8022

Tabla 5.2: Estadisticos de monotonía para distintos conjuntos de variables

Para comprobar si efectivamente estos diagnósticos nos están informando de que los datos que estamos considerando en este trabajo no son lo "suficientemente monótonos" como para que las técnicas isotónicas funcionen bien se han calculado los valores de estas medidas para los datos motors, del paquete `isoboost`, en los que se sabe que los algoritmos isotónicos mejoraron la predicción, considerando las variables una a una, el conjunto de todas ellas y el subconjunto que proporciona los mejores resultados [30].Los resultados obtenidos para este conjunto de datos aparecen en la tabla 5.3

Amplitude L1	Amplitude U1	Amplitude L5	Amplitude U5	Amplitude L7	Amplitude U7	**NMI**	γ_1	**FOM**	γ_3	γ_4
X						0.1409	0.4362	0.7181	0.7181	1
	X					0.1745	0.302	0.651	0.651	1
		X				0.1419	0.4325	0.7162	0.7162	1
			X			0.2401	0.0394	0.5197	0.5197	1
				X		0.2713	-0.0852	0.4574	0.4574	1
					X	0.3459	-0.3836	0.3082	0.3082	1
X	X	X	X	X	X	0.0078	0.7894	0.1334	0.8947	0.149052
X	X	X				0.0629	0.665	0.6263	0.8328	0.75197133

Tabla 5.3: Estadísticos de monotonía para los datos del paquete `isoboost`

A la vista de los resultados, podemos ver que para los datos de `isoboost`, los estadísticos indican que existe un mayor grado de monotonía en estos. El **NMI** toma valores más bajos para los datos de `isoboost`, y tanto **FOM** como $\gamma 1$ y $\gamma 3$ toman valores más altos para los datos de `isoboost` también. De hecho, los autores de `isoboost` proponen utilizar las tres

primeras variables porque proporcionaban los mejores resultados en la predicción. En la tabla 5.3 se aprecia que son las que tienen los mejores valores de los estadíticos FOM, $\gamma 3$ y $\gamma 4$. Esto también se ve reflejado en la Fig. 5.6, son las variables en las que la monotonía parece cumplirse para todas las clases.
Sin embargo, hay variables en las que los estadísticos toman peores valores que en nuestros datos. Como vemos en la Fig. 5.6, a diferencia de nuestros datos, para las variables en las que la monotonía esta presente, lo está para todas las clases. En nuestros datos, por el contrario, parece que a la vista de las figuras 5.4 y 5.5 la monotonía en muchos casos solo se cumplía para las clases *R4,R5*.

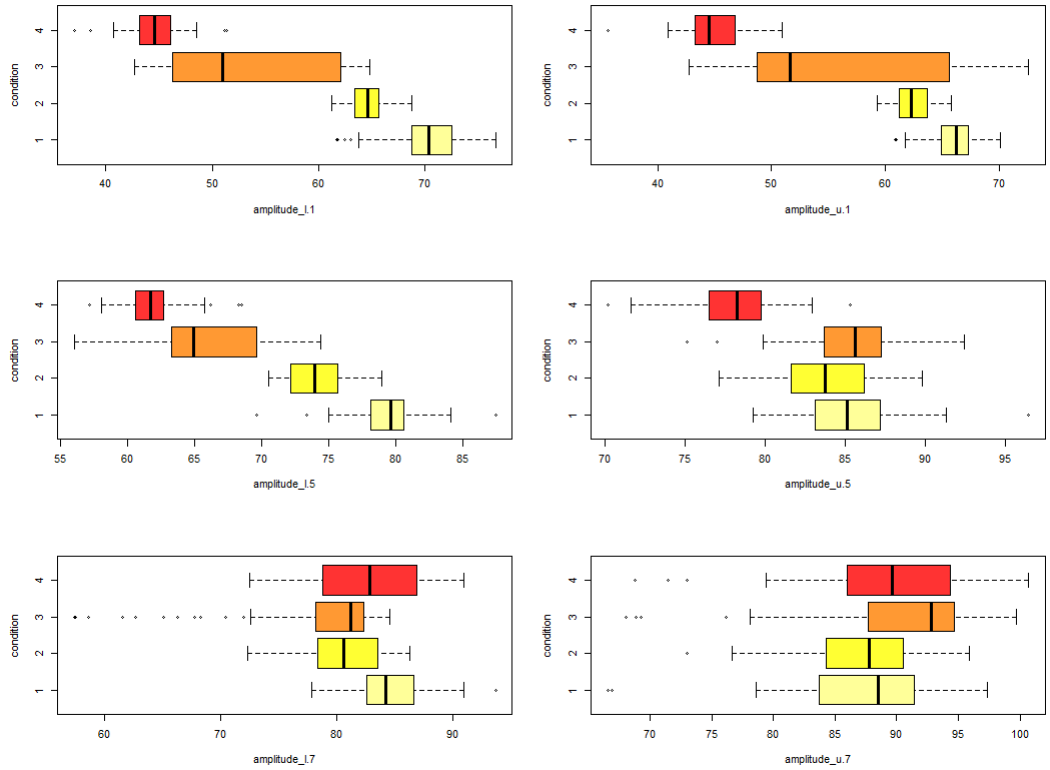


Figura 5.6: Distribución de las clases según armónico para los datos de isoboost

Siguiendo con el análisis, a continuación se muestra el conteo de pares que cumplen la monotonía (S_+) para todas las variables separado por clases en nuestros datos, y en otra tabla para los datos de `isoboost` utilizando las variables recomendadas por los autores en [30] y todas las disponibles:

LSH					
	1	**2**	**3**	**4**	**5**
1	X	0.2045	0.2536	0.3193	0.3213
2		X	0.1725	0.2251	0.2234
3			X	0.2464	0.2455
4				X	0.3806
5					X

USH					
	1	**2**	**3**	**4**	**5**
1	X	0.4023	0.4470	0.5265	0.6009
2		X	0.3660	0.4303	0.4889
3			X	0.4602	0.5218
4				X	0.6454
5					X

Tabla 5.4: **FOM** por pares de clases de nuestro conjunto de datos que cumplen la monotonía por clases para LSH (izda) y USH(dcha)

L.1+U.1+L.5				
	1	**2**	**3**	**4**
1	X	0.8401	0.6972	0.8841
2		X	0.6894	0.9288
3			X	0.7362
4				X

Todo				
	1	**2**	**3**	**4**
1	X	0.2688	0.1916	0.2350
2		X	0.1697	0.2193
3			X	0.0497
4				X

Tabla 5.5: **FOM** de los datos de `isoboost` que cumplen la monotonía por clases

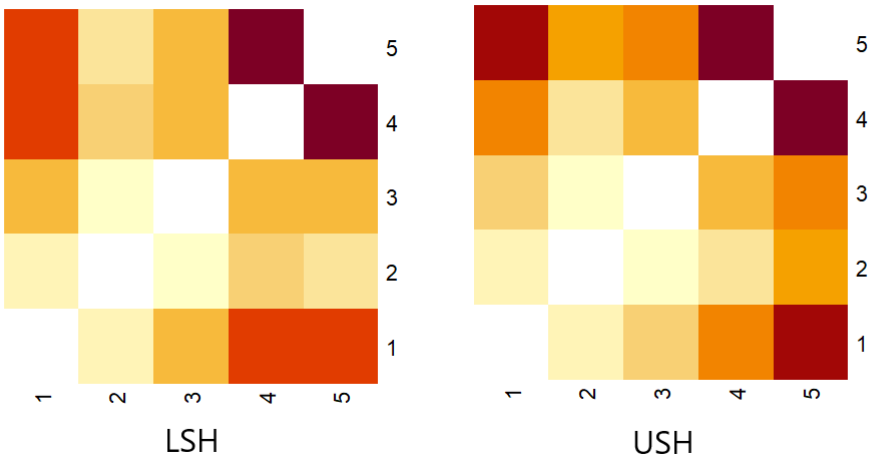


Figura 5.7: Heatmap del conteo de pares que cumplen la monotonía por clases para USH y LSH

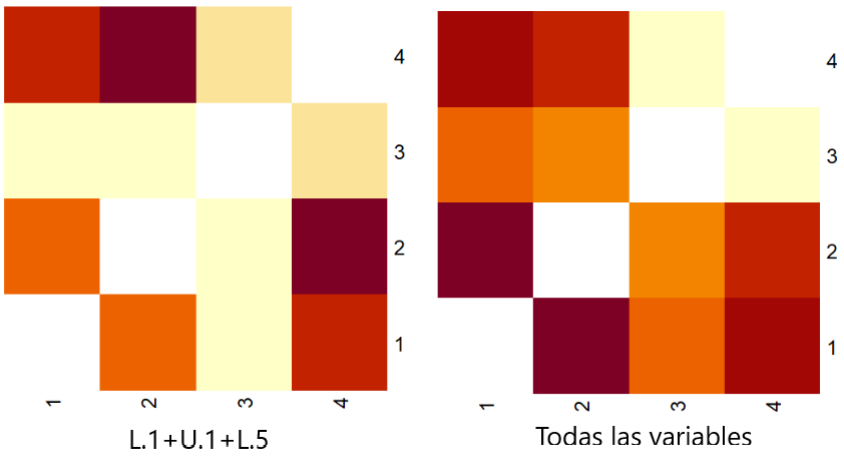


Figura 5.8: Heatmap del conteo de pares que cumplen la monotonía por clases para los datos de isoboost

Podemos ver que para nuestros datos las zonas más oscuras son las de R1,R2,R3 vs R4,R5. En los datos `isoboost`, si utilizamos las variables recomendadas por los autores, vemos que los colores más intensos están siempre hacia arriba en columnas (es decir, cuanto más alejadas están las clases en deterioro, mejor se cumple la monotonía). En caso de estudiar todas las variables, la clase 1 es la única que parece cumplir la monotonía frente al resto.

También es interesante comentar que el armónico USH parece cumplir mejor la monotonía según la Tabla 5.4.3 que el armónico LSH.

Finalmente, a la vista de los resultados podemos decir que los datos con los que se ha realizado este TFG no presentan monotonía de manera sólida y por ello la inclusión de restricciones monotónicas no parece contribuir a una mejor predicción.

6. Conclusiones y trabajo futuro

A continuación se exponen las conclusiones obtenidas a partir del estudio de este TFG.

6.1 Conclusiones sobre los modelos Boosting e influencia de los factores

- Se ha demostrado que, a la hora de determinar el estado de deterioro de un motor mediante la medida de las corrientes inducidas por la degradación del rotor, inversores diferentes requieren modelos predictivos diferentes y que en esos modelos predictivos pueden influir de manera distinta los diferentes niveles de carga suministrado al motor.

- La información de un único armónico es suficiente para clasificar inversores AB y ABB, confirmando así la sospecha de que el USH es redundante. Sin embargo, para el inversor TM parece conveniente el uso conjunto de USH y LSH.

- El mejor algoritmo de boosting para este problema es XGBoost o XGBoost con DART. LightGBM es más rápido pero proporciona peores resultados. AdaBoost es el peor en cualquier situación.

- El inversor que proporciona mejores tasas de acierto es TM, pero su comportamiento es distinto al de los otros. Según informaciones recibidas desde el departamento de Ingeniería Eléctrica, el inversor TM tiene un comportamiento más inestable lo que podría dificultar la clasificación de su estado de deterioro, pero en este TFG se consigue una mejor clasificación a costa de la inclusión información sobre el armónico USH y la introducción de Dropout.

- La clasificación de motores en estado R4 o R5 (los dos estados de deterioro más altos) es directa, el problema está entre las clases R1, R2 y R3 (deterioros más bajos). Los algoritmos han sido capaces de encontrar separación entre estas clases, que según el estudio descriptivo inicial parecían difíciles de separar.

- Además, con el estudio basado en los Shapley Values proporcionados por SHAP hemos encontrado relación entre los instantes temporales y la manifestación de los estados de deterioro de los motores.

- El trabajo que se ha desarrollado en este TFG no solo ha permitido establecer algoritmos complejos para clasificación, sino que se ha expuesto una metodología sólida para explicar qué variables y de qué forma estas permiten separar las clases, cuando en este tipo de algoritmos opacos no es fácil. Esta metodología no es solo aplicable a clasificadores complejos sino que no depende del clasificador utilizado

6.2 Conclusiones sobre el estudio de la isotonía

- Hemos observado que los métodos de clasificación que tienen en cuenta la monotonía no funcionan bien en este caso a pesar de que el planteamiento teórico del problema hacía pensar en que deberían tener buen desempeño.

- Con el objetivo de explicar el pobre comportamiento en este caso de los métodos isotónicos, hemos establecido una metodología para el análisis basado en estadísticos para la detección de las situaciones en las que este problema puede aparecer. Para ello, aparte de las medidas expuestas en [32], hemos propuesto dos estadísticos nuevos para el estudio de la monotonía en los datos, γ_3 y γ_4.

- Siguiendo esta metodología, hemos visto que en nuestros datos la causa de del mal comportamiento de los métodos isotónicos se debe a que, entre los datos disponibles, para las las clases R1, R2 y R3 no parece haber un comportamiento monótono, aunque sí se cumpliría entre R123, R4 y R5.

- En los datos de `isoboost`, hemos encontrado la causa de por qué esas variables proporcionaban los mejores resultados en el trabajo [30] a la hora de usar clasificadores isotónicos.

6.3 Trabajo futuro y posibles mejoras

A continuación se enumeran algunas líneas de trabajo que han surgido en el estudio desarrollado en este TFG y que quedan pendientes de futuro desarrollo.

- Recolección de más datos de cara a tener una muestra más representativa del problema a estudiar. Si se quiere hacer una aplicación en entornos reales, dado que los datos son de laboratorio, parece conveniente recolectar datos en un entorno real con ruido estadístico.

- Entrenamiento con más tiempo de cara a buscar mejores hiperparámetros.

- Existen alternativas de ensemble (no boosting) que no hemos probado como stacking o redes neuronales, cuya interpretabilidad es peor pero su capacidad de clasificación puede ser mayor. No obstante, con la metodología de interpretación expuesta en el capítulo 2 se podrían estudiar estos modelos.
- Comparar con modelos que no sean basados en ensembles.
- Revisión de los resultados por parte de un experto en el dominio, para poder mejorar el análisis de conocimiento generado a partir de los modelos.
- Estudio más detallado de las medidas de monotonía propuestas en este trabajo, propuesta de nuevas medidas relacionadas con ellas y valoración de su posible utilidad para determinar a priori la utilidad de los métodos isotónicos.
- Desarrollo de publicaciones científicas de impacto para exponer el conocimiento obtenido en este trabajo que no aparece en la literatura actual.

A. Anexos

A.1 Tablas de resultados

A.1.1 Tasas de error de clasificación para los algoritmos

		Error de test promedio					
Algoritmo	**Monotonía**	**Media**	**Mínimo**	**Q=0.25**	**Q=0.5**	**Q=0.75**	**Máximo**
rlda	*Mono*	0.631	0.40	0.56	0.64	0.72	0.88
amilb	*Mono*	0.332	0.04	0.24	0.32	0.44	0.64
asilb	*Mono*	0.310	0.04	0.24	0.32	0.36	0.64
cmilb	*Mono*	0.611	0.36	0.56	0.60	0.68	0.84
csilb	*Mono*	0.353	0.08	0.28	0.36	0.44	0.72
XGBoost_Mono	*Mono*	0.272	0.04	0.24	0.28	0.32	0.48
LightGBM_Mono	*Mono*	0.283	0.00	0.20	0.28	0.36	0.52
DART_Mono	*Mono*	0.275	0.08	0.24	0.28	0.32	0.52
XGBoost	*No_mono*	0.241	0.08	0.16	0.24	0.28	0.52
LightGBM	*No_mono*	0.253	0.08	0.20	0.24	0.32	0.48
DART	*No_mono*	0.238	0.04	0.20	0.24	0.28	0.48
AdaBoost	*No_mono*	0.292	0.04	0.20	0.28	0.36	0.84

Tabla A.1: Resultados de las tasas de error test promediadas para cada algoritmo

A.1.2 MAEs para los algoritmos basados en árboles

		MAE de test promedio					
Algoritmo	**Monotonía**	**Media**	**Mínimo**	**Q=0.25**	**Q=0.5**	**Q=0.75**	**Máximo**
DART	*Mono*	0.3925556	0.08	0.28	0.36	0.48	1.12
LightGBM	*Mono*	0.3928889	0.00	0.28	0.40	0.48	0.88
XGBoost	*Mono*	0.3768889	0.04	0.28	0.36	0.48	0.92
AdaBoost	*No_mono*	0.4546667	0.08	0.28	0.40	0.52	1.72
DART	*No_mono*	0.3291111	0.04	0.24	0.32	0.40	0.80
LightGBM	*No_mono*	0.3430000	0.08	0.24	0.32	0.44	0.80
XGBoost	*No_mono*	0.3272222	0.08	0.24	0.32	0.40	0.84

Tabla A.2: Resultados de los MAES de test promediadas para cada algoritmo

A.2 Generación del conjunto de datos con Set de Variables 1

```
## leer datos


X <- read.csv2("D:/Users/Desktop/TFG/datos3.csv")

# Resumen variables
summary(X[,1:5])
#########################################################################

#Vamos a agrupar las variables temporales de 250 en 250, haciendo una media por
    tramos temporales
tiempototal<-ncol(X)-4
#Xagrupado<-X[X$band==sort(X$band),1:4]

XLSH<-X[X$band=="LSH",]
XUSH<-X[X$band=="USH",]
colnames(XLSH)<-paste(colnames(XLSH),"LSH",sep="")
colnames(XUSH)<-paste(colnames(XUSH),"USH",sep="")

#Marcas temporales
marcas<-seq(250,1750,by=250)
entorno.amplitud<-5

Xagrupado<-X[X$band=="LSH",c(1,3,4)]
nombres<-colnames(Xagrupado)
```

```
#Union de LSH y USH por columnas
for(i in 1:(length(marcas)-1)){

   #Tramo a promediar
   tramo<-(marcas[i]-entorno.amplitud):(marcas[i]+entorno.amplitud)+5
   #Unimos las media del tramo para cada observacion
   Xagrupado<-cbind(Xagrupado,
   7rowMeans(XLSH[,tramo]),
   rowMeans(XUSH[,tramo]))
   #Cambiamos el nombre
   nombretramoL<-paste("xLSH",marcas[i],sep="")
   nombretramoU<-paste("xUSH",marcas[i],sep="")
   nombres<-c(nombres,nombretramoL,nombretramoU)
}

colnames(Xagrupado)<-nombres

#Por comodidad, pongo la clase a predecir al final y drop de la banda
Xagrupado<-Xagrupado[,c(1,3:ncol(Xagrupado),2)]
colnames(Xagrupado)[c(1:2,ncol(Xagrupado))]<-c("model","level","state")
Xagrupado$state<-as.factor(Xagrupado$state)
```

A.3 Funcion de entrenamiento y prueba de hiperparámetros para AdaBoost

```
library(adabag)
library(caret)

splitandtrain<-function(data,trees=1:10, xv=5){

   train.index <- createDataPartition(data$state, p = .66, list = FALSE)
   train <- data[ train.index,]
   test  <- data[-train.index,]

   errores<-c()
   for(ntrees in trees){
      print(c(ntrees,"arboles"))
```

```
    errores<-c(errores,boosting.cv(state~., data=train, boos=TRUE,
        mfinal=4,coeflearn = "Zhu",v=xv)$error)

  }

  model<-boosting(state~.,data=train,boos=TRUE,
  mfinal=which.min(errores),coeflearn = "Zhu")

  results<-list("train"=train,"test"=test,"bestmodel"=model,
  "bestntree"=which.min(errores),"xv.err"=errores, "bestxv.err"=min(errores))
  return(results)

}
```

A.4 Algoritmo Genético para XGBoost

```
#Funcion de inicializacion de la poblacion
initPop<-function(popsize=10){
  pop<-data.frame(a=runif(popsize,0,1),
  b=runif(popsize,0,1),
  eta=runif(popsize,0,0.3),
  depth=sample(1:5,popsize,replace=TRUE),
  rondas=sample(1:100,popsize,replace=TRUE),
  score=rep(0,popsize))
  return(pop)
}

#Funcion de seleccion de los mas aptos y entrenamiento
selectandbreed<-function(pop){
  popsize<-nrow(pop)/2
  pop<-pop[order(pop$score),]
  pop<-pop[1:(popsize),] #salvamos los 50% mejores
  #Creamos la nueva poblacion
  hijos<-pop
  for(i in 1:popsize){
    padre1<-pop[sample(1:popsize,1),]
    padre2<-pop[sample(1:popsize,1),]
    gen1<-sample(1:ncol(padre1),ncol(padre1)/2)
    gen2<- -1*gen1
    nuevohijo<-mutate(cbind(padre1[,gen1],padre2[,gen2]))
```

```
      hijos<-rbind(hijos,nuevohijo)
   }
   #Sustituimos al ultimo hijo por uno aleatorio para poder escapar del minimo local
   hijos[nrow(hijos),]<-initPop(1)
   return(hijos)
}

#Funcion de mutacion
mutate<-function(hijo){

   n.hijo<-hijo
   n.hijo$a<-hijo$a*rnorm(1,1,0.15)
   n.hijo$b<-hijo$b*rnorm(1,1,0.15)
   n.hijo$eta<-hijo$eta*rnorm(1,1,0.1)
   n.hijo$depth<-hijo$depth+sample(c(-1,1),1)
   n.hijo$rondas<-round(hijo$rondas*rnorm(1,1,0.15))
   n.hijo$score<-0

   return(n.hijo)
}
```

A.5 Función de entrenamiento y búsqueda de hiperparámetros de XGBoost en R

```
splitandtrain<-function(data,popsize=10,ngen=30,verb=FALSE){

   #Train test split
   train.index <- createDataPartition(data$state, p = .66, list = FALSE)
   train <- as.matrix(data[ train.index,])
   test  <- as.matrix(data[-train.index,])

   #Matriz xgboost,utilidad que facilita el manejo de la libreria
   variables<-1:(ncol(data)-1)
   target<-ncol(data)
   dtrain <- xgb.DMatrix(data = train[,variables], label = train[,target])
   dtest <- xgb.DMatrix(data = test[,variables], label = test[,target])
   pop<-initPop(popsize)
   for(gen in 1:ngen){
      print(c("Generacion",gen))
```

```
      print(pop)
      for(i in 1:nrow(pop)){
         ind<-pop[i,]
         param<-list(max_depth=ind$depth,alpha=ind$a,beta=ind$b,eta=ind$eta)
         clasif<-xgb.cv(nfold=5,data = dtrain, nrounds = ind$rondas, objective =
             "multi:softmax",num_class=5,params=param,verbose=FALSE )
         pop[i,"score"]<-clasif$evaluation_log$test_merror_mean[ind$rondas]
      }
      print("Scores")
      print(pop)
      print("Reproduccion")
      pop<-selectandbreed(pop)
      print(pop)
   }
   #devolvemos el optimo
   i.op<-1
   bestind<-pop[i.op,]
   param.op<-list(max_depth=bestind$depth,alpha=bestind$a,beta=bestind$b,eta=bestind$eta)
   model<-xgboost(data = dtrain, nrounds = bestind$rondas, objective =
       "multi:softmax"
   ,num_class=5,params=param )
   results<-list("train"=dtrain,"dtest"=dtest,"test"=test,"bestmodel"=model,
   "bestparam"=param.op,"bestxv.err"=bestind$score,"lastpop"=pop)

   return(results)

}
```

A.6 Ejemplo de LightGBM monótono en python

```
import lightgbm as lgb
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

import time

start = time.time()
datos=pd.read_csv("datos/datosagrupados.csv",sep=",")
```

```
datos=datos.drop(datos.columns[0],axis=1)

from sklearn.model_selection import RandomizedSearchCV
estimator = lgb.LGBMClassifier(monotone_constraints=[1,1,1,1,1,1,1])

errores=[]
subsets=[["AB","NC1"],
      ["ABB","NC1"],
      ["TM","NC1"],
      ["AB","NC2"],
      ["ABB","NC2"],
      ["TM","NC2"]]

param_grid = {
      'lambda_l1':[0,0.1,0.5,1,2,5],
      'lambda_l2':[0,0.1,0.5,1,2,5],
      'learning_rate': np.arange(0.05,0.55,0.1),
      'n_estimators': [1,10,50,100,1000],
      'min_child_samples': [1,10,50],
      'max_depth': np.arange(0,40,10),
      'num_leaves': np.arange(5,104,5),
      'feature_fraction': [0.7,0.8,0.9,1.0],
      'bagging_fraction': [0.7,0.8,0.9,1.0],
      'bagging_freq': [5,6,7,8]
}

erroresmxl=[]
maemxl=[]
for i in range(0,len(subsets)):

   df=datos.loc[(datos["model"]==subsets[i][0]) & (datos["level"]==subsets[i][1]) &
       (datos["band"]=="LSH")]

   X=df.iloc[:,:-1]
   y=df.iloc[:,-1]
   X=X.drop(["model","level","band"],axis=1)
   "La clase a int"
   y=pd.factorize(y)[0]

   errores=[]
   mae=[]
   for repeticion in range(0,20):
      print("Combinacion",i,"repeticion",repeticion)
```

```
        X_train, X_test, y_train, y_test = train_test_split(X, y,
        stratify=y,
        test_size=0.33)

        gbm = RandomizedSearchCV(estimator, param_grid,cv=5,n_iter=100,n_jobs=-1)
        gbm.fit(X_train, y_train,eval_set=[(X_train, y_train)],
        eval_metric="multi_logloss",early_stopping_rounds=10)

        print('Mejores hiperparametros:', gbm.best_params_)
        y_predgrid = gbm.predict(X_test)
        # eval
        print('Accuracy:', accuracy_score(y_test, y_predgrid))

        errores.append(1-accuracy_score(y_test, y_predgrid))
        mae.append(mean_absolute_error(y_test,y_predgrid))
    maemxl.append(mae)
    erroresmxl.append(errores)

resultados=pd.DataFrame(erroresmxl)
resultados.to_csv("resultadosLGBM/erroresLGBM_LSH_mono_100.csv")
resultados=pd.DataFrame(maemxl)
resultados.to_csv("resultadosLGBM/maeLGBM_LSH_mono_100.csv")
print("End")
end = time.time()
print("Elapsed time",end - start)
```

A.6.1 Para XGBoost y XGBoost DART

Para que no sea monótono, basta con quitar el parámetro de monotonía. (Lo mismo para LightGBM). `booster="dart",rate_drop=0.1,skip_drop=0.5` son los parámetros propios de XGBoost-DART

```
params = {
'eta':[0, 0.1,0.5,0.9],
'lambda':[0,0.1,0.5,1,2,5],
'min_child_weight': [1, 5, 10],
'gamma': [0.5, 1, 1.5, 2, 5],
'subsample': [0.6, 0.8, 1.0],
'colsample_bytree': [0.6, 0.8, 1.0],
'max_depth': [3, 4, 5]
}
```

```
estim = xgb.XGBClassifier(learning_rate=0.02,
    n_estimators=600,early_stopping_rounds=20,
    objective='multi:softmax',num_class=5,
silent=True,booster="dart",sample_type="uniform"
,normalize_type="tree",
rate_drop=0.1,skip_drop=0.5,
monotone_constraints=(1,1,1,1,1,1,1, 1,1,1,1,1,1,1))
```

A.7 Script para el cálculo de estadísticos de monotonía

Nota: Este ejemplo es para los datos del paquete `isoboost`.

```
library(isoboost)
data(motors)
datos<-motors

X<-datos[,-7]
y<-as.numeric(datos[,ncol(datos)])
n<-nrow(X)
mono_cols<-list(c(1,2,3),c(1,2,3,4,5,6)) #Columnas a comprobar
pairs<-t(combn(1:n,2))

for(k in 1:length(mono_cols)){
   mono_col<-mono_cols[[k]]
   nclash<-0
   nmonot<-0

   p<-0
   for(w in 1:nrow(pairs)){
      i<-pairs[w,1]
      j<-pairs[w,2]

      p<-p+1

      x1<-X[i,mono_col]
      y1<-y[i]
      x2<-X[j,mono_col]
      y2<-y[j]

      imenorquej<-mean(x1<x2)==1 #Si la media de los valores de x1<x2 es 1, es que
```

```
        todo son trues y todas las variables de x1 son mayores
    imayorquej<-mean(x1>x2)==1

    if(imenorquej){
      #Si x1<x2
      nclash<-nclash+ (y1<=y2)
      nmonot<-nmonot+ (y1>y2)
    } else if(imayorquej) {
      #Si x1>x2
      nclash<-nclash+ (y1>=y2)# e y1>y2, clash
      nmonot<-nmonot+ (y1<y2) # e y1<y2, exito
    }
  }
  nmi<-nclash/(n*(n-1))
  s.menos<-nclash
  s.mas<-nmonot

  gamma1<-(s.mas-s.menos)/(s.mas+s.menos)
  fom<-s.mas/p
  gamma3<-s.mas/(s.mas+s.menos)
  gamma4<-(s.mas+s.menos)/p

  print(c(" ",paste(colnames(X)[mono_col])))
  print(c("NMI:=",round(nmi,4),"Gamma:=",round(gamma1,4)))
  print(c("FOM:=",round(fom,4),"Gamma3:=",round(gamma3,4
      ),"Gamma4:=",round(gamma4,8)))

}
```

A.8 Scripts para el cálculo de valores SHAP

A.8.1 KernelSHAP

```
#Mejor estimador de la validacion cruzada
model=estim.best_estimator_

shap.initjs()
def pred_wrap(data):
data_wrap=pd.DataFrame(data,columns=X.columns)
return(model.predict_proba(data_wrap))
```

```
explainer = shap.KernelExplainer(pred_wrap,X)
shap_values=explainer.shap_values(X)
```

A.8.2 TreeSHAP

```
#Mejor estimador de la validacion cruzada
model=estim.best_estimator_
shap.initjs()

explainer = shap.TreeExplainer(model)
shap_values=explainer.shap_values(X)
```

A.8.3 Gráficos

Utilidades para los graficos

```
#Guardamos el nombre de las columnas porque se entrena con numpy arrays
feature_names=X.columns
# Utilidades de color
classes=["R"+str(i) for i in range(1,6)]
colors =['green','blue','red','purple','brown']

# Crear el mapa de colores
class_inds = numpy.argsort([-np.abs(shap_values[i]).mean() for i in
    range(len(shap_values))])
cmap = plt_colors.ListedColormap(np.array(colors)[class_inds])
```

Individual para cada clase

```
clase=0
shap.summary_plot(shap_values[clase], features, feature_names, color=cmap,
    class_names=classes)
```

Conjunto

```
shap.summary_plot(shap_values, features, feature_names, color=cmap,
    class_names=classes)
```

Índice de Figuras

Índice de Tablas

Bibliografía

[1] José Ignacio Fernández Villafáñez. "Diagnóstico de fallos en rodamientos de motores eléctricos mediante técnicas lasso". *Trabajo de Fin de Grado, Escuela de Ingenierías Industriales,Universidad de Valladolid* (2018).

[2] Sergio P. Santos and Jose Aifredo F. Costa. "A comparison between hybrid and non-hybrid classifiers in diagnosis of induction motor faults". *Proceedings - 2008 IEEE 11th International Conference on Computational Science and Engineering, CSE 2008*. 2008. ISBN: 9780769531939. DOI: 10.1109/CSE.2008.60.

[3] F.R.S. R. A. Fisher, Sc.D. "The use of multiple measurements in taxonomic problems". *Annals of Eugenics* (1936). ISSN: 19454589. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

[4] James N. Morgan and John A. Sonquist. "Problems in the Analysis of Survey Data, and a Proposal". *Journal of the American Statistical Association* (1963). ISSN: 01621459. DOI: 10.2307/2283276.

[5] Gareth James et al. *An introduction to Statistical Learning*. 2000. ISBN: 978-1-4614-7137-0. DOI: 10.1007/978-1-4614-7138-7. arXiv: arXiv:1011.1669v3.

[6] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". *The Bulletin of Mathematical Biophysics* (1943). ISSN: 00074985. DOI: 10.1007/BF02478259.

[7] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. "Deep learning-based image recognition for autonomous driving". *IATSS Research* 43.4 (Dec. 2019), pp. 244–252. DOI: 10.1016/j.iatssr.2019.11.008. URL: https://doi.org/10.1016/j.iatssr.2019.11.008.

[8] *Google Duplex: A.I. Assistant Calls Local Businesses To Make Appointments*. 2018, visited on the 2nd of June 2020. URL: https://www.youtube.com/watch?v=D5VN56jQMWM.

[9] Michael Kearns. "Thoughts on Hypothesis Boosting". Unpublished manuscript. Dec. 1988.

[10] M. Kearns and L. G. Valiant. "Crytographic Limitations on Learning Boolean Formulae and Finite Automata". *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. STOC '89. Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 433–444. ISBN: 0897913078. DOI: 10.1145/73007.73049. URL: https://doi.org/10.1145/73007.73049.

[11] Yoav Freund and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 904. Springer Verlag, 1995, pp. 23–37. ISBN: 9783540591191. DOI: 10.1006/jcss.1997.1504.

[12] Tianqi Chen and Carlos Guestrin. "XGBoost". *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD-16* (2016). DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[13] Guolin Ke et al. "LightGBM: A highly efficient gradient boosting decision tree". *Advances in Neural Information Processing Systems*. 2017.

[14] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine". *Annals of Statistics* (2001). ISSN: 00905364. DOI: 10.2307/2699986.

[15] Leo Breiman. "Random forests". *Machine Learning* (2001). ISSN: 08856125. DOI: 10.1023/A:1010933404324.

[16] Jerome H. Friedman. "Stochastic gradient boosting". *Computational Statistics and Data Analysis* (2002). ISSN: 01679473. DOI: 10.1016/S0167-9473(01)00065-2.

[17] K. V. Rashmi and Ran Gilad-Bachrach. "DART: Dropouts meet multiple additive regression trees". *Journal of Machine Learning Research*. 2015. arXiv: 1505.01866.

[18] Geoffrey E. Hinton Alexander Krizhevsky Ilya Sutskever Nitish Srivastva. "System and method for addressing overfitting in a neural network". US9406017B2. 2005.

[19] Jerome Friedman and T Hastie. "Additive logistic regression: a statistical view of boosting, 1998". *URL citeseer. nj. nec. com/friedman98additive. html* (1998).

[20] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2019.

[21] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why should i trust you?" Explaining the predictions of any classifier". *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778.

[22] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. arXiv: 1705.07874 [cs.AI].

[23] Lloyd S Shapley. *A value for n-person games*. Tech. rep. Rand Corp Santa Monica CA, 1952.

[24] Erik Štrumbelj and Igor Kononenko. "Explaining prediction models and individual predictions with feature contributions". *Knowledge and Information Systems* (2014). ISSN: 02193116. DOI: 10.1007/s10115-013-0679-x.

[25] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. "Consistent Individualized Feature Attribution for Tree Ensembles". *CoRR* abs/1802.03888 (2018). arXiv: 1802.03888. URL: http://arxiv.org/abs/1802.03888.

[26] A. M. Turing. "Computing machinery and intelligence". *Machine Intelligence: Perspectives on the Computational Model*. 2012. ISBN: 9781136525049. DOI: 10.1093/mind/lix.236.433.

[27] Alejandro Barón García and Alberto Calvo Madurga. *A nature-inspired didactic approach to genetic algorithms & neural networks*. Tech. rep. Universidad de Valladolid, 2019. URL: https://github.com/AlejandroBaron/didactic-ga/blob/master/documentation.pdf.

[28] Raúl Granados Romero. "Diagnóstico de fallos en el rotor de motores eléctricos en estado transitorio mediante técnicas estadística". *Trabajo de Fin de Grado, Escuela de Ingenierías Industriales,Universidad de Valladolid* (2017).

[29] Rubio del Rey Fernando. "Análisis de la influencia del variador en el diagnóstico de los fallos de motores mediante técnicas estadísticas". *Trabajo de Fin de Grado,Escuela de Ingenierías Industriales,Universidad de Valladolid* (2016).

[30] David Conde et al. "Isotonic boosting classification rules". *Accepted for publication Advances in Data Analysis and Classification* (2020). DOI: 10.1007/s11634-020-00404-9.

[31] David Conde et al. "dawai: An R Package for Discriminant Analysis with Additional Information". *Journal of Statistical Software* 66.10 (2015), pp. 1–19. URL: http://www.jstatsoft.org/v66/i10/.

[32] José Ramón Cano et al. "Monotonic classification: An overview on algorithms, performance measures and data sets". *Neurocomputing* (2019). ISSN: 18728286. DOI: 10.1016/j.neucom.2019.02.024. arXiv: 1811.07155.

[33] H. A. M. Daniels and M. V. Velikova. "Derivation of monotone decision models from noisy data". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 36.5 (2006), pp. 705–710.