Universidad de Valladolid

Grado en Estadística

Facultad de Ciencias

Diagnóstico de fallos de rodamientos en motores de inducción en estado estacionario mediante técnicas boosting y redes neuronales

Autor

Manuel Astorgano Antón

Tutor

Miguel Alejandro Fernández Temprano

Trabajo de Fin de Grado

Resumen

Los motores de inducción están presentes en la mayoría de procesos industriales. Como todo componente mecánico, dichos motores están sujetos a averías. Debido a los grandes costes de inspección y reparación, surge la necesidad de monitorizar los motores para detectar fallos de manera no invasiva, es decir, sin intervenir en su funcionamiento. El análisis de la corriente, el sonido y las vibraciones que genera el motor resultan útiles para la detección de fallos.

En este trabajo se plantea un problema de clasificación de fallos en rodamientos de motores de inducción en estado estacionario analizando los datos asociados a las frecuencias características de dichos fallos. Se cuenta con conjuntos de datos basados en la señal eléctrica, el sonido y las vibraciones de los motores que permiten clasificarlos en seis estados en función de su nivel de deterioro.

Para resolver el problema de la clasificación se utilizan técnicas de inteligencia artificial modernas como son el boosting o las redes neuronales junto con metodología estadística. Esto permite determinar la influencia de distintos factores que afectan a la precisión de la detección de fallos. Además, se explora la interpretabilidad de los modelos obtenidos, permitiendo obtener las variables más importantes a la hora de clasificar así como la influencia de sus valores.

Finalmente, se obtienen conclusiones interesantes como que el uso de estadísticos de orden superior, el boosting de árboles de decisión o el conjunto de datos de señal eléctrica generan los diagnósticos de deterioro del motor más precisos.

Abstract

Induction motors are used in most industrial processes. As every mechanical component, these motors are bound to eventually fail. Due to both high inspection and reparation costs, the need to monitor the motors in a non-invasive way, i.e. without intervening on their operation, appears. The analysis of electrical current, sound and vibration generated by the motor are useful in the process of said fault detection.

This project proposes a classification problem for detecting faults in bearings during the steady-state performance of an induction motor by analyzing the data associated to the characteristic frequencies of these faults. Three data sets are available which contain information regarding electrical current, sound and vibration, enabling a motor classification into six states based on their wear and damage.

In order to solve this problem, state of the art artificial intelligence techniques such as boosting and neural networks joined with statistical methodology are used. This allows to determine the influence of several factors which have an impact in the accuracy of fault detection. Furthermore, an interpretability analysis is conducted, thus providing both the most important features and their values which play a key role in the classification.

Finally, the results show that using higher-order statistics, boosted decision trees or the electrical current data set generate the most accurate diagnosis of wear in motor bearings.

Agradecimientos

A Miguel Alejandro por su tiempo, ayuda y consejos durante el desarrollo del trabajo.
A los profesores del Grado que me han transmitido las bases, los conocimientos y el interés en la estadística.
A mi familia y amigos por su ayuda y apoyo durante estos años.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Descripción del problema

Un motor de inducción o motor asíncrono es un motor eléctrico de corriente alterna capaz de producir energía mecánica a partir de energía eléctrica mediante inducción electromagnética [1]. Se utiliza en máquinas de la industria, en electrodomésticos de la vida cotidiana o en vechículos eléctricos.

Debido al uso de este tipo de motor en la gran mayoría de procesos industriales, la importancia de la monitorización de su estado para detectar fallos ha aumentado durante los últimos años. Los fallos mecánicos suelen manifestarse mediante la generación de vibraciones y sonido excesivo.

Existen diversas técnicas para diagnosticar fallos mecánicos en motores de inducción: análisis de vibraciones, análisis de flujo magnético de dispersión, análisis de sonido o análisis del espectro de corriente del motor (*Motor Current Signature Analysis*, MCSA) [2].

Para un tipo de fallo en particular: la contaminación de rodamientos por partículas, el análisis de sonido, vibraciones y de corriente resulta útil debido a que es no invasivo, lo que significa que no se interfiere con el funcionamiento del motor a la hora de obtener las mediciones.

En este trabajo se analizarán varios conjuntos de datos relacionados con el sonido, las vibraciones y la corriente para detectar fallos en rodamientos de motores en estado estacionario. Primero, se procesarán los datos y se emplearán técnicas de reducción de dimensionalidad para seleccionar variables de interés. Después, se entranará un modelo de clasificación para diagnosticar el estado de los motores. Para ello, se utilizarán tanto técnicas boosting como redes neuronales debido a los buenos resultados de trabajos anteriores (ver [3], [4]) y su aumento de popularidad por el éxito en multitud de áreas de estudio. Posteriormente se realizará una

comparativa de resultados y una evaluación de los métodos. Finalmente, se aplicarán técnicas de interpretación para obtener el razonamiento interno de los clasificadores y facilitar su comprensión disminuyendo el efecto de «caja negra».

1.2. Objetivos

La capacidad de diagnosticar fallos prematuramente en motores industriales supone un gran avance debido al aumento de la seguridad y eficiencia energética que proporciona. Para ello, se utilizan cada vez más modelos de inteligencia artificial ya que evitan recurrir a inspecciones invasivas y generan buenos resultados. Hoy día, los algoritmos más populares están basados en técnicas boosting y aprendizaje profundo (redes neuronales de gran tamaño). Los objetivos de este trabajo son:

- Analizar datos de fallos de motores en tres ámbitos: corriente eléctrica, sonido y vibraciones.
- Estudiar cómo las técnicas boosting y redes neuronales clasifican fallos de motores para posteriormente compararlas.
- Obtener información interpretable a partir de modelos opacos, comprobando qué variables influyen en la clasificación.

1.3. Estructura

El documento se organiza de la siguiente manera:

1. Metodología. Introducción a los modelos y explicación de conceptos teóricos que se van a aplicar.
2. Datos. Descripción, procesamiento, transformaciones y selección de variables.
3. Análisis. Construcción de modelos para los diferentes conjuntos de datos con su correspondiente evaluación, interpretación y comparativa.
4. Conclusiones. Resumen de resultados, ideas de mejora y trabajo futuro.

1.4. Asignaturas relacionadas

Las asignaturas con más peso a la hora de realizar este proyecto han sido las siguientes:

- Análisis de Datos, Análisis Multivariante y Análisis de Datos Categóricos. Explican los fundamentos del problema de la clasificación e introducen los modelos predictivos así como los métodos para evaluarlos.
- Regresión y ANOVA, Modelos Lineal y Modelos Estadísticos Avanzados. Desarrollan los procedimientos de análisis de la varianza.
- Métodos Estadísticos de Computación Intensiva. Se introduce la simulación de modelos y técnicas de regularización de los mismos.
- Computación Estadística. Introducción al lenguaje de programación R, donde se adquieren conocimientos tecnológicos en el marco de la estadística.
- Técnicas de Aprendizaje Automático y Minería de Datos. Introducción al boosting, redes neuronales y Python.

Capítulo 2

Metodología

Durante este capítulo se exponen las bases teóricas de los dos métodos que se van a emplear para abordar el problema de la clasificación: boosting de árboles de decisión y redes neuronales. Además, se abordan cuestiones sobre la evaluación e interpretación de los modelos utilizados.

2.1. Boosting de árboles de decisión

2.1.1. Árboles de decisión

El aprendizaje mediante árboles de decisión o inducción de árboles de decisión es un modelo ampliamente usado en estadística, minería de datos y *machine learning*. Emplea un árbol de decisión como modelo predictivo para, a partir de unas observaciones representadas como ramas, extraer conclusiones representadas como hojas. Es un algoritmo sencillo de construir e implementar.

Fueron introducidos en 1963 por James N. Morgan y John A. Sonquist [5]. No obstante, comenzaron a ganar popularidad con las publicaciones «Árboles de clasificación y regresión» de Breiman (*Classification and Regression Trees*, CART) [6], «ID3» y su última versión «C4.5» [7], [8].

De forma general, un árbol de decisión está formado por nodos y se construye generando condiciones sobre las variables de un conjunto de datos. Los nodos internos representan las condiciones y los externos (hojas), las decisiones basadas en las condiciones. Para generar las condiciones, se utilizan métricas como el índice de Gini o la entropía, de forma que las observaciones que se clasifican en cada hoja pertenezcan a la misma clase. La forma de clasificación es muy similar al procedimiento que seguiría un humano «haciendo preguntas», lo que confiere

a los árboles una alta interpretabilidad.
No obstante, los árboles de decisión por si solos no suelen ser una buena elección de modelo ya que las desventajas sobrepasan a las ventajas:

- Poca robustez. Pequeños cambios en los datos pueden significar grandes cambios en la estructura (condiciones) del árbol.
- Poca precisión. Actualmente, casi todos los algoritmos alternativos a los árboles generan mejores resultados con datos similares.
- Para datos con variables categóricas con múltiples niveles, los árboles suelen favorecer a los atributos con más niveles [9].
- Los cálculos para construirlos se complican cuando el espacio de clases no es divisible mediante hiperrectángulos, es decir, las clases del conjunto de datos no forman espacios rectangulares al ser representadas. Un problema linealmente separable provoca un alto sobreajuste si se utiliza un árbol (Figura 2.1.1).

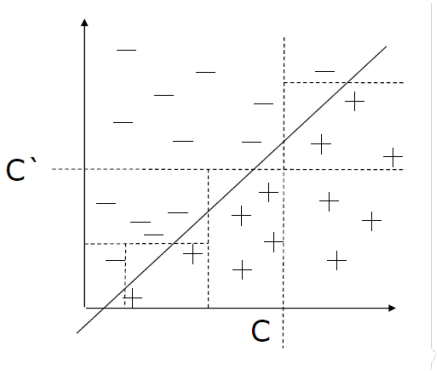


Figura 2.1.1: Particiones rectangulares obtenidas con un árbol de clasificación sobre un problema linealmente separable.

2.1.2. AdaBoost

Dentro del aprendizaje automático supervisado, el boosting o potenciación es un meta-algoritmo basado en ensembles (construir un modelo a partir varios modelos más sencillos) que tiene como objetivo reducir el sesgo y la varianza del modelo final haciendo uso de clasificadores débiles (clasificador con poco poder predictivo). Nace de la pregunta «¿Puede un conjunto de clasificadores débiles crear un clasificador fuerte?» propuesta por Kearns y Valiant [10]. Es posible utilizar boosting con casi cualquier tipo de clasificador, pero lo más habitual es emplear árboles de decisión ya que se aprovecha su sencillez e interpretabilidad a la vez que se mitigan las desventajas vistas anteriormente.

AdaBoost o *Adaptive Boosting* [11] es un algoritmo popular e importante históricamente ya que fue el primer algoritmo capaz de adaptarse a los clasificadores débiles. A pesar de que los clasificadores iniciales sean malos, mientras el rendimiento de cada uno sea mejor que una elección aleatoria, el modelo final convergerá a un clasificador fuerte.
AdaBoost se utiliza típicamente con «tocones de decisión» o *decision stumps*, es decir, árboles con un solo nivel/disyunción. Éstos serán los clasificadores débiles. La idea del algoritmo consiste en construir un modelo sencillo que asigna el mismo peso a todas las observaciones. A continuación, se re-asignan los pesos de forma que a las observaciones mal clasificadas les corresponda un peso mayor. De esta forma, los individuos con pesos mayores tendrán más importancia en el siguiente modelo. Se entrena durante un número de iteraciones hasta que disminuya el error de clasificación. Podemos ver el procedimiento a grandes rasgos en la Figura 2.1.2

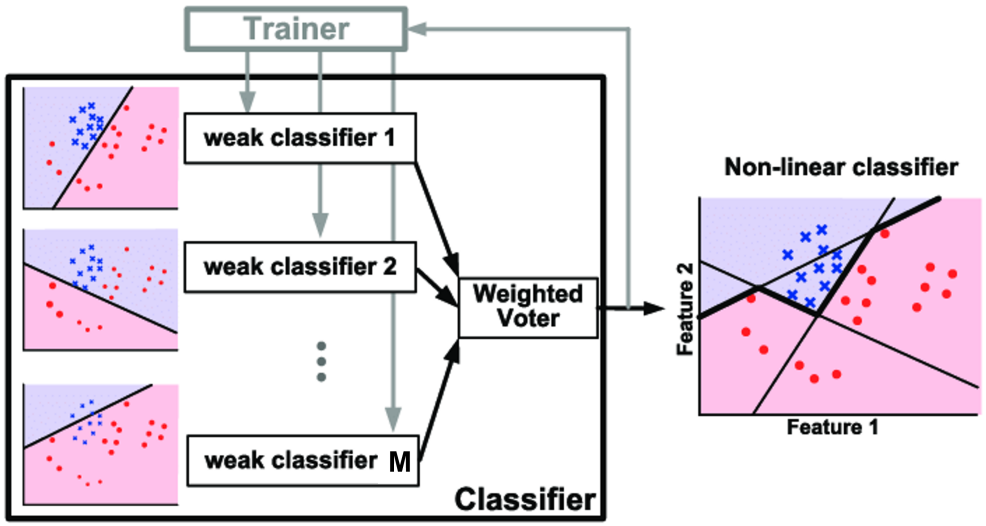


Figura 2.1.2: Visualización de AdaBoost.

Suponiendo que se tienen unos datos de entrenamiento $\{(x_i, y_i)\}_{i=1}^N$, donde $x_i \in \mathbb{R}^K$ e $y_i \in \{-1, 1\}$, donde las y_i son la variable respuesta y las x_i, las variables que se usan para clasificar. Se cuenta con un número grande M de clasificadores débiles $f_m(x) \in \{-1, 1\}$ y una función de pérdida I en $[0, 1]$ definida así:

$$I(f_m(x), y) = \begin{cases} 0 & \text{si } f_m(x_i) = y_i \\ 1 & \text{si } f_m(x_i) \neq y_i \end{cases} \tag{2.1.1}$$

El algoritmo de AdaBoost funciona de la siguiente manera:

1. Desde $i = 1$ hasta N, $w_i^{(1)} = 1$. Fijar los pesos de cada observación a 1.

2. Desde $m = 1$ hasta M entrenar el clasificador m minimizando la función objetivo $\epsilon_m = \frac{\sum_{i=1}^{N} w_i^{(m)} I(f_m(x_i) \neq y_i)}{\sum_{i=1}^{N} w_i^{(m)}}$. Obtener $\alpha_m = ln\frac{1-\epsilon_m}{\epsilon_m}$.

3. Para cada i obtener $w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m I(f_m(x_i) \neq y_i)}$. Actualizar los pesos.

Por último, se obtiene un clasificador final basado en una combinación lineal de los clasificadores débiles:

$$g(x) = sign(\sum_{m=1}^{M} \alpha_m f_m(x)) \tag{2.1.2}$$

En el algoritmo aparece un término ϵ_m, que es el error asociado al clasificador f_m y se calcula con la proporción de errores mal clasificados. A partir de ϵ_m se obtiene α_m denominado influencia, potestad o *amount of say*. Como se observa en el paso de actualización de los pesos, α_m indica la importancia que tendrá un clasificador a la hora de decidir sobre una observación.
Comprobamos que la actualización de los pesos se realiza en base a α_m. Si éste es positivo, el peso aumenta y si es negativo, el peso disminuye.
Finalmente, observamos en la Figura 2.1.3 el procedimiento en detalle [12]. Cada sub-Figura muestra el número m de clasificadores base entrenados hasta el momento junto con la banda de decisión del más reciente (línea discontinua) y la línea de decisión combinada (línea continua verde). Cada observación se representa con un círculo y su clase, con un color. Sus pesos vienen dados por el tamaño del círculo. De esta forma, comprobamos que a los ejemplos mal clasificados en la iteración $m = 1$ se les asigna un peso alto en la iteración $m = 2$.

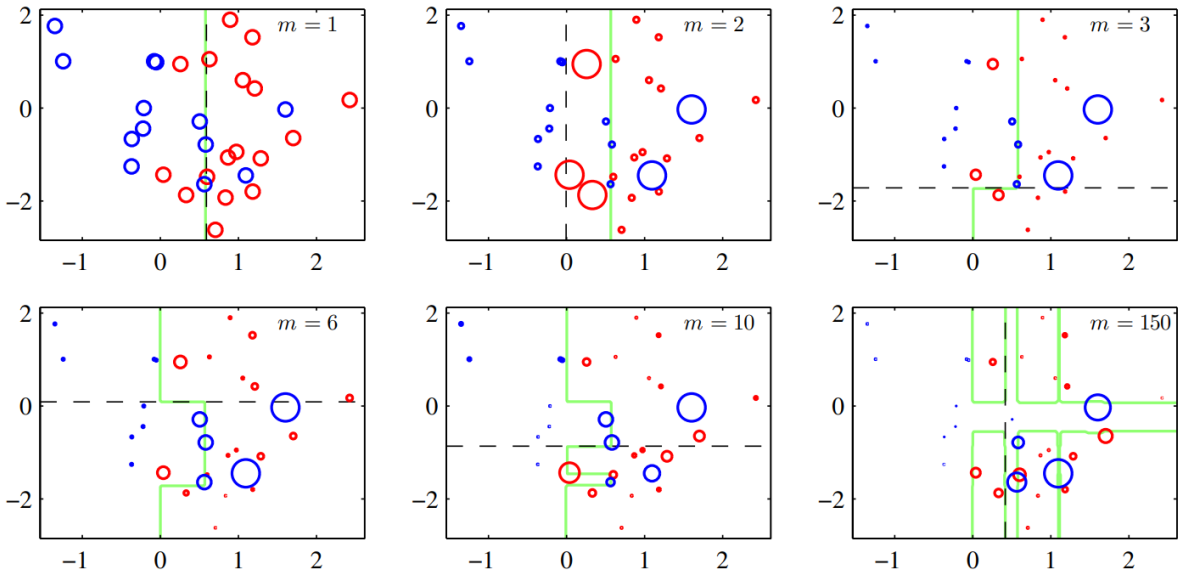


Figura 2.1.3: Iteraciones de AdaBoost [12].

2.1.3. Gradient boosting

Similar a AdaBoost, el gradient boosting o potenciación del gradiente permite obtener predicciones a partir de un ensemble de clasificadores débiles que normalmente son árboles de decisión. No obstante, parte de la idea de Breiman de que el boosting puede interpretarse como un algoritmo de optimización sobre una función de pérdida adecuada [13].
Los árboles potenciados por gradiente se desarrollaron explícitamente por Friedman [14] a la vez que lo hacen Mason, Baxter, Barlett y Frean [15]. En estos documentos se explica la versión del algoritmo como un descenso del gradiente funcional iterativo, es decir, los algoritmos optimizan una función de coste sobre un espacio de funciones eligiendo iterativamente una función que apunta hacia la dirección negativa del gradiente.

Algoritmo general

Inicialmente, contamos con una variable dependiente y y un vector de variables independientes x que se relacionan con alguna distribución de probabilidad. El objetivo es encontrar una función $\hat{F}(x)$ que aproxime a la variable dependiente a partir de la independiente. Para ello, se introduce una función de coste o pérdida $L(y, F(x))$ y se minimiza:

$$\hat{F} = argmin_F E_{x,y}[L(y, F(x))] \tag{2.1.3}$$

Se asume que y toma valores reales y se busca una aproximación $\hat{F}(x)$ en la forma de una suma ponderada de funciones $h_i(x) \in \mathcal{H}$. Cada una de estas funciones equivale a uno de los anteriores clasificadores débiles que componen el ensemble.

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + constante \tag{2.1.4}$$

Denotamos el set de entrenamiento como muestras conocidas de x e y: $\{(x_1, y_1), ..., (x_n, y_n)\}$. A partir de él, se busca una aproximación $\hat{F}(x)$ que minimice el valor medio de la función de pérdida en el conjunto de entrenamiento. Se parte de una función $F_0(x)$ y se expande incrementalmente mediante un procedimiento *greedy* o voraz:

$$F_0(x) = argmin_\gamma \sum_{i=1}^{n} L(y_i, \gamma) \tag{2.1.5}$$

$$F_m(x) = F_{m-1}(x) + argmin_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h_m(x_i)) \right] \tag{2.1.6}$$

Donde $h_m \in \mathcal{H}$ es una función de clasificadores débiles.
Como elegir h a partir de L es casi imposible computacionalmente, se simplifica el problema aplicando el descenso del gradiente funcional. La idea básica es encontrar un mínimo local de la función de pérdida iterando sobre $F_m(x)$ con $\gamma > 0$, haciendo uso de los residuos de la iteración m (denominados pseudo-residuos).

$$F_m(x) = F_{m-1}(x) - \gamma \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i)) \tag{2.1.7}$$

Es posible optimizar γ buscando un valor para el cual la función de pérdida tenga un mínimo:

$$\gamma_m = argmin_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))) \tag{2.1.8}$$

En la solución anterior, el modelo ajustado con los pseudo-residuos es $h_m(x_i) = -\nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))$.
Finalmente se actualiza el modelo de acuerdo con:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) \tag{2.1.9}$$

Donde γ_m es 2.1.8 (*learning rate* o tasa de aprendizaje).

Algoritmo para el problema de multiclasificación

La clasificación multiclase es un caso específico del algoritmo anterior. Resulta de gran interés puesto que los estados de los motores entran en esta categoría. Se emplean árboles de decisión de regresión logística y en cada iteración se calcula un árbol para cada clase k. A continuación, se describen los pasos a seguir junto con algunos cambios que adaptan el descenso del gradiente a los árboles de decisión [14].
Comenzamos especificando una función de pérdida adecuada para un problema de clasificación en K clases:

$$L(\{y_k, F_k(x)\}_1^K) = -\sum_{k=1}^{K} y_k \log p_k(x) \tag{2.1.10}$$

$y_k = 1 \in \{0, 1\}$ y cada clase se denota por k. $p_k(x) = P(y_k = 1|x)$. Con esto, se puede expresar $F(x)$ en función de las clases k:

$$F_k(x) = \log p_k(x) - \frac{1}{K} \sum_{l=1}^{K} \log p_l(x) \tag{2.1.11}$$

Y obtener de forma equivalente las probabilidades p en función de F:

$$p_k(x) = \frac{\exp(F_k(x))}{\sum_{l=1}^{K} \exp(F_l(x))} \tag{2.1.12}$$

Sustituyendo el resultado de la ecuación 2.1.12 en 2.1.10 y tomando la primera derivada se obtiene:

$$\tilde{y}_{ik} = \left[\frac{\partial L(\{y_{il}, F_l(x_i)\}_{l=1}^{K})}{\partial F_k(x_i)} \right]_{\{F_l(x)=F_{l,m-1}(x)\}_1^K} = y_{ik} - p_{k,m-1}(x_i) \tag{2.1.13}$$

En este caso calculamos $p_{k,m-1}(x)$ a partir de $F_{k,m-1}(x)$ en la ecuación 2.1.12. Como consecuencia se inducen K árboles en cada iteración m con el objetivo de predecir los residuales de cada clase. Cada uno de los árboles tiene J hojas que se distribuyen en regiones $\{R_{jkm}\}_{j=1}^{J}$. Las actualizaciones de los pesos γ_{jkm} se obtienen con una ecuación similar a 2.1.8. Dicho resultado no es sencillo de calcular, por lo que se computa una solución aproximada con el algoritmo de Newton-Raphson empleando una aproximación diagonal de la matriz Hessiana. El problema se separa en tantos subproblemas como hojas tenga el árbol:

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} \tilde{y}_{ik}}{\sum_{x_i \in R_{jkm}} |\tilde{y}_{ik}|(1 - |\tilde{y}_{ik}|)} \tag{2.1.14}$$

Finalmente, se obtiene una ecuación de actualización del modelo parecida a 2.1.9:

$$F_{km}(x) = F_{k,m-1}(x) - \sum_{j=1}^{J} \gamma_{jkm} 1(x \in R_{jkm}) \tag{2.1.15}$$

2.1.4. XGBoost

XGBoost [16] (*eXtreme Gradient Boosting*) es una librería de código abierto disponible en varios lenguajes de programación que ofrece una implementación eficiente y generalizada del gradient boosting. Ha ganado atención durante los últimos años debido a su buen rendimiento en multitud de problemas y competiciones de inteligencia artificial. Incluye tanto mejoras algorítmicas como computacionales, lo que permite crear modelos precisos y robustos de forma rápida. Algunas de las características que XGBoost amplía con respecto a la potenciación del gradiente tradicional, explicada anteriormente, son:

- Paralelización. A pesar de que el gradient boosting se identifica por ser un proceso iterativo en el cual se induce un clasificador a partir de los errores del anterior, XGBoost construye los árboles en paralelo. Inicialmente escanea todas las instancias de entrenamiento y define particiones independientes que van a aparecer en cada árbol. A continuación, procesa las

particiones o caminos de forma paralela.

- Poda. El criterio de construcción de árboles del boosting clásico es un método voraz y su parada ocurre cuando se obtiene un valor negativo en la función de pérdida. XGBoost introduce un parámetro de «profundidad máxima» que construye el árbol hasta la profundidad especificada y poda las ramas con pérdidas negativas, lo que incrementa el rendimiento predictivo y computacional.
- Optimización del *hardware*. Permite aprovechar las especificaciones del ordenador al máximo, haciendo uso tanto de los núcleos del procesador como de la tarjeta gráfica. Esto hace factible entrenar un modelo complejo hasta en ordenadores poco potentes.
- Regularización. Incluye penalizaciones para modelos complejos tanto con regularización Lasso (L1) como Ridge (L2).
- Búsqueda de particiones teniendo en cuenta datos ausentes. En la gran mayoría de problemas, los datos no están completos. Se incluye un pequeño sub-algoritmo que permite aprender los patrones de «escasez» de los datos. Se basa en enumerar los valores ausentes (desde el principio hasta el final y viceversa) y obtener predicciones en las dos direcciones, eligiendo la mejor.

A continuación se explican algunas características de interés con más detalle, siguiendo [16].

Esquema de cuantiles ponderados

Uno de los pasos más importantes a la hora de construir un árbol es la elección de candidatos para generar las disyunciones o condiciones sobre los datos. El *weighted quantile sketch* es un sub-algoritmo cuya función es proponer los mejores candidatos posibles. Normalmente se utilizan los cuantiles para distribuir a los candidatos de forma uniforme. En resumen, los datos se dividen en sub-conjuntos y se calculan los cuantiles en paralelo para obtener un histograma aproximado de los cuantiles de todos los datos. En XGBoost se añaden pesos a dichos cuantiles de forma que los pesos en cada cuantil sean aproximadamente los mismos.
Este procedimiento es más eficaz y el aumento de eficiencia es más notorio cuanto mayor sea el conjunto de datos.

Regularización de la función objetivo

Se incluye un término de regularización $\Omega(f)$ que suaviza los pesos finales con el objetivo de paliar el sobreajuste. Basta con hacer $\Omega(f) = 0$ para obtener un modelo sin regularización.

La nueva función objetivo empleando la notación de secciones anteriores sería:

$$\mathcal{L} = \sum_i L + \sum_k \Omega(f_k) \tag{2.1.16}$$

$$\Omega(f) = \alpha J + \frac{1}{2}\lambda ||\gamma||^2 \tag{2.1.17}$$

En la ecuación anterior, L es la función objetivo original, J es el número de hojas del árbol y γ, sus pesos. Los parámetros de regularización son α y λ.

Sub-muestreo de columnas y decrecimiento

Estas dos técnicas son sencillas y efectivas. El sub-muestreo de columnas, como su nombre indica, consiste en utilizar una muestra aleatoria de las variables en la construcción de cada árbol. Previene el sobreajuste aún más que el tradicional sub-muestreo de filas (también incluido) y acelera la computación en paralelo. Esta técnica proviene de los conocidos bosques aleatorios o *random forests* de Breiman [17].
El decrecimiento, introducido por Friedman en [18] permite escalar los pesos de un nuevo clasificador por un factor η después de cada iteración del algoritmo. De una forma parecida al *learning rate*, el decrecimiento permite reducir la influencia de cada árbol individual dejando «espacio» para que nuevos árboles mejoren el modelo.

2.2. Redes neuronales

En esta sección se explica la teoría detrás de las redes neuronales: funcionamiento básico, funciones de activación y entrenamiento.

2.2.1. Estructura básica

La idea de las redes neuronales aparece en 1943 con el trabajo de McCulloch y Pitts [19]. A partir de ahí, Rosenblatt propone el modelo del perceptrón simple [20], que es lo que se utiliza hoy día como base de una red neuronal. Una neurona g es una suma ponderada:

$$g(x) = \sum_{i=1}^{n} w_i x_i \tag{2.2.1}$$

Donde x es un vector con valores de variables y w los pesos asignados. La neurona se activa según una función f y produce una salida binaria $\{0, 1\}$ dependiendo de un valor umbral b

denominado *bias*.

$$y = f(g(\mathbf{x})) = \begin{cases} 1 & \text{si } g(\mathbf{x}) + b > 0 \\ 0 & \text{si } g(\mathbf{x}) + b \leq 0 \end{cases} \tag{2.2.2}$$

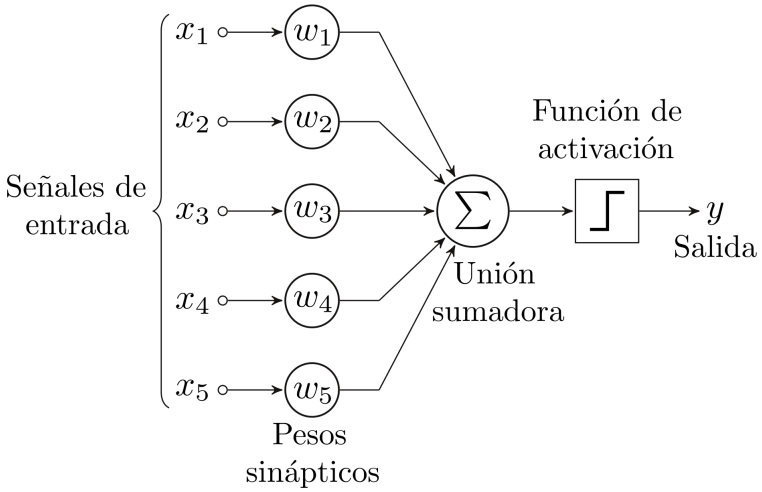


Figura 2.2.1: Perceptrón simple.

En la Figura 2.2.1 se observan las partes de una neurona junto con su función de activación, que se explicará en la siguiente sección.
Normalmente, las neuronas se organizan en capas y cada capa suele tener múltiples neuronas. Esto da lugar a las conocidas redes neuronales.

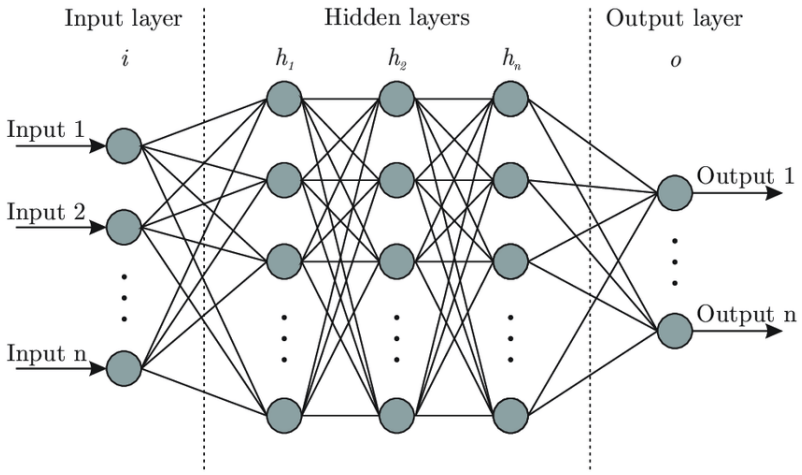


Figura 2.2.2: Estructura de una red neuronal.

En la Figura 2.2.2 observamos la estructura general de una red en la que todas las capas

adyacentes están conectadas. La capa de entrada (izquierda) consta de tantas neuronas como variables. Las capas ocultas se eligen de forma arbitraria. Finalmente, la capa de salida consta de tantas neuronas como clases tenga nuestro problema.

2.2.2. Funciones de activación

La función de activación de una neurona genera su salida dado un conjunto de entradas. Cada neurona tiene una asignada y no tienen por qué coincidir. Existen muchos tipos de funciones de activación y su elección depende del tipo de problema que se esté tratando. A continuación explicamos las más importantes.

Sigmoide

También denominada logística, toma valores en el rango $[0, 1]$, lo que permite interpretar sus salidas como una probabilidad.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2.3}$$

Suele utilizarse en la última capa de una red por su interpretabilidad, pero no es recomendable hacerlo en más capas debido a su coste computacional.

Rectificador lineal

La función ReLU o *Rectified Linear Unit* es eficiente y rápida de calcular. Presenta valores en el intervalo $[0, \infty]$.

$$f(x) = max(0, x) \tag{2.2.4}$$

Softmax

Basada en la función sigmoide, es muy empleada en problemas de multiclasificación. Proporciona una estimación de la distribución de probabilidades de las clases, por lo que la suma de sus salidas es 1.

$$f_i(z_i) = \frac{e^{z_i}}{\sum_n^{i=0} e^{z_i}} \tag{2.2.5}$$

Donde z_i es un vector de tamaño n.

2.2.3. Entrenamiento

El entrenamiento de una red consiste en ajustar los pesos y *bias* de cada neurona de forma que las predicciones sean tan cercanas a la realidad como sea posible, como cualquier otro modelo.

Se introduce el término «conjunto de validación», que permite evaluar las salidas de la red en cada iteración para saber si está realizando un ajuste adecuado. Consiste en reservar un pequeño grupo de observaciones del set de datos original para guiar a la red hacia la dirección adecuada en la actualización de los pesos.

Función de pérdida

Para evaluar el rendimiento del modelo durante el entrenamiento es necesario emplear una función de pérdida o función de coste. Dicha función calcula la distancia entre la salida generada por la red y la salida real, lo que permite evaluar la calidad del ajuste a los datos. Dependiendo de los datos que tengamos será necesario emplear diferentes tipos de funciones. Aquí se explican las utilizadas en este trabajo:

- Error cuadrático medio: $MSE(y, \tilde{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$. Normalmente se utiliza en problemas de regresión donde la variable a predecir es continua.
- Entropía cruzada categórica (*categorical crossentropy*). Utilizada en problemas de clasificación con varias clases.
  $CCE = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$

Como se explicará en la sección 2.3.1, es posible usar el MSE en el problema de los motores para penalizar menos la clasificación incorrecta entre clases adyacentes.

Descenso del gradiente

El descenso del gradiente es el algoritmo por el cual se minimiza la función objetivo. Consiste en explorar la función objetivo C que en nuestro caso es la de pérdida y minimizarla empleando un procedimiento similar al explicado en la sección 2.1.3.
Para minimizar C hay que calcular su gradiente ∇C, lo que necesita del cálculo del gradiente de cada observación del conjunto de entrenamiento x, ∇C_x. Este procedimiento es computacionalmente exhaustivo por lo que se recurre al descenso del gradiente estocástico o *Stochastic Gradient Descent*, SGD.
La idea es sencilla: consiste en seleccionar aleatoriamente m observaciones del set de datos original y obtener el gradiente de esa muestra, aproximándose al gradiente real. El muestreo se realiza sin reemplazamiento y cuando se agotan las observaciones se ha completado una época o *epoch* de entrenamiento.
En la práctica, se introducen pequeñas heurísticas como la modificación de la tasa de aprendizaje durante el entrenamiento o la adición de un término de inercia. Ambas permiten escapar de mínimos locales y «encauzar» el enternamiento en la dirección adecuada.

Retropropagación

El algoritmo de retropropagación o *backpropagation* [21] permite calcular de forma rápida y eficiente los gradientes de la sección anterior. No es hasta 1986 cuando comienza a utilizarse y las redes neuronales ganan popularidad, puesto que el entrenamiento hasta ese momento era lento y la mayoría de problemas grandes eran inabordables. La idea general es realizar una evaluación del error del modelo hacia atrás, partiendo de la última capa hasta la primera actualizando los parámetros de la red.

Para llevar a cabo el algoritmo de retropropagación, debemos encontrar una expresión concreta de la derivada parcial de la función de pérdida C respecto a cualquier peso w o bias b de la red neuronal.

$$\frac{\partial C}{\partial w} \text{ y } \frac{\partial C}{\partial b} \tag{2.2.6}$$

A continuación definimos la nomenclatura y los pasos para obtener la expresión que buscamos:

- w^l es la matriz de pesos de las neuronas que conectan con la capa l. Cada elemento w^l_{jk} es el peso de la neurona k-ésima en la capa $l-1$ que conecta con la neurona j-ésima en la capa l.
- b^l es el bias o sesgo de la neurona j-ésima en la capa l. Cada elemento b^l_j es el bias de la neurona j.
- a^l es el vector de activaciones de la capa l. Cada elemento a^l_j es la activación de la neurona j.
- z^l es la suma ponderada de las neuronas de la capa l. $z_l = w^l a^{l-1} + b^l$.
- δ^l es el vector de errores asociado a la capa l. Cada elemento δ^l_j es el error de la neurona j, definido como $\delta^l_j = \frac{\partial C}{\partial z^l_j}$.

Ahora se puede obtener una expresión para a^l a partir de la función de activación f y su correspondiente suma ponderada.

$$a^l = f(z^l) \tag{2.2.7}$$

Como queremos conocer cómo un cambio sobre los parámetros afectan a la función de pérdida, tenemos que calcular los errores para cada capa y relacionarlos con las derivadas:

$$\frac{\partial C}{\partial w^l_{jk}} \text{ y } \frac{\partial C}{\partial b^l_j} \tag{2.2.8}$$

La retropropagación se fundamenta en cuatro fórmulas que calculan el error δ^l y el gradiente de la función de pérdida. Como ya hemos dicho, se comienza por la capa de salida:

1. Se obtiene la ecuación del error para la capa de salida (δ^L).

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} f'(z_j^L) \tag{2.2.9}$$

2. Con el error de la última capa, es posible obtener una ecuación del error para la capa anterior y así sucesivamente. Este paso es lo que da nombre al algoritmo.

$$\delta^l = ((w^{l+1})^t \delta^{l+1}) \cdot f'(z^l) \tag{2.2.10}$$

   Es importante darse cuenta que se traspone la matriz de pesos $(w^{l+1})^t$.

3. Finalmente se obtienen las ecuaciones que buscábamos: la derivada de la función de pérdida en función de los bias y del peso. Estas ecuaciones proporcionan el gradiente de la función de coste.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.2.11}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.2.12}$$

Optimizador

El optimizador es un hiperparámetro de las redes neuronales encargado de reducir la función de pérdida. Habitualmente se basan en métodos de descenso del gradiente y ofrecen rendimientos distintos. Los más usados a día de hoy son dos:

- Descenso del gradiente estocástico o SGD.
- Adam o *Adaptive Moment Estimation*. Calcula los momentos de primer y segundo orden de los gradientes, es decir, la media y la varianza.

Estos dos optimizadores se caracterizan por un compromiso entre velocidad de convergencia y capacidad de generalización. Adam es el más rápido ya que el cálculo de los momentos es computacionalmente eficiente. Sin embargo, está demostrado que SGD con inercia generaliza mejor [22], aunque sea más lento. En este trabajo, el conjunto de datos es relativamente pequeño, por lo que la opción elegida será SGD.

Medidas contra el sobreajuste

Las redes neuronales son conocidas por su tendencia al sobreajuste. Para corregirlo, es posible aplicar técnicas ya conocidas como la regularización L1 y L2, además de otras más específicas de las redes neuronales.

- Parada tempranda. Consiste en parar el entrenamiento de la red cuando el rendimiento del modelo sobre el conjunto de entrenamiento comience a empeorar, lo que ocurre cuando se entrena durante demasiadas épocas.
- *Dropout*. Se trata de eliminar aleatoriamente neuronas con una probabilidad fijada como hiperparámetro durante el proceso de entrenamiento. Esto equivale a entrenar redes diferentes que pueden sobreajustar de formas distintas, lo que finalmente reduce el sobreajuste.

2.3. Construcción y evaluación de modelos

A continuación se exponen cuestiones relacionadas con las métricas del error de un modelo y los procedimientos para entrenar un clasificador eligiendo correctamente la división del conjunto de datos y los hiperparámetros del mismo.

2.3.1. Medidas del error

Existen varias métricas para medir el error, en este trabajo se usarán las siguientes:

- Error de clasificación: $Error(y, \tilde{y}) = 1 - accuracy = 1 - \frac{1}{n}\sum_{i=1}^{n} I(y_i = \tilde{y}_i)$
- Error cuadrático medio: $MSE(y, \tilde{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$. Normalmente se calcula su raíz cuadrada para poder interpretarlo (RMSE).
- Error absoluto medio: $MAE(y, \tilde{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \tilde{y}_i|$.

Donde y es un vector de tamaño n con valores reales e $\tilde{y}$ contiene los predichos.
El error cuadrático y el absoluto no penalizan tanto como el error de clasificación la confusión de clases adyacentes. El MAE es el más sencillo de interpretar pero es difícil de optimizar en comparación con los otros dos.

2.3.2. División del conjunto de datos

A la hora de entrenar un modelo, buscamos estimar su rendimiendo de la forma más precisa posible y que ese rendimiento sea el mejor posible. Sin embargo, obtener una estimación precisa del error y que éste sea bajo es un compromiso que en cierta medida depende de la forma en la que elijamos los datos para entrenar el modelo.
Si utilizamos el conjunto de datos completo para entrenar, el modelo se ajustará lo mejor posible a esos datos. Inicialmente puede parecer una buena idea, pero existe el problema del sobreajuste: el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien con los datos reales, que son los que nos interesan.
La solución a esto es sencilla, hay que dividir el conjunto de datos en dos: entrenamiento (train) y prueba (test). Los datos de entrenamiento permiten ajustar los parámetros del modelo mientras que los de prueba evalúan su rendimiento ya que nunca han sido vistos por el clasificador. A partir de esta idea se desarrollan varias formas de entrenar y evaluar un modelo, que dependen tanto de la variedad como del tamaño del conjunto de datos. Además, es recomendable estratificar los sub-conjuntos que obtengamos, es decir, mantener la misma distribución de clases tras dividir el conjunto de datos original.

Hold-out

Coincide con el método «general» explicado anteriormente, siendo el más sencillo. Consiste en dividir el conjunto total de datos en dos: entrenamiento y test. Es fácil de implementar pero presenta desventajas. La primera es que si el conjunto de datos no es lo suficientemente grande, el entrenamiento o la estimación del error no serán fiables. Puede remediarse repitiendo el procedimiento varias veces, pero esto provoca que los conjuntos se solapen entre repeticiones generando dependencia entre ellos y de nuevo la estimación del error (promedio) sería demasiado optimista.

Validación cruzada

La validación cruzada, *K-fold* o *cross validation* (K-CV) permite estimar la tasa de error de la forma más realista posible. El procedimiento a seguir es el siguiente (Figura 2.3.1):

1. Dividir el conjunto de datos en K particiones estratificadas.
2. Entrenar con $K-1$ particiones y obtener la estimación del error con la restante (test).
3. Repetir K veces cambiando la partición de test por otra no usada hasta el momento.
4. Obtener la tasa de error final como la media de todas las iteraciones.

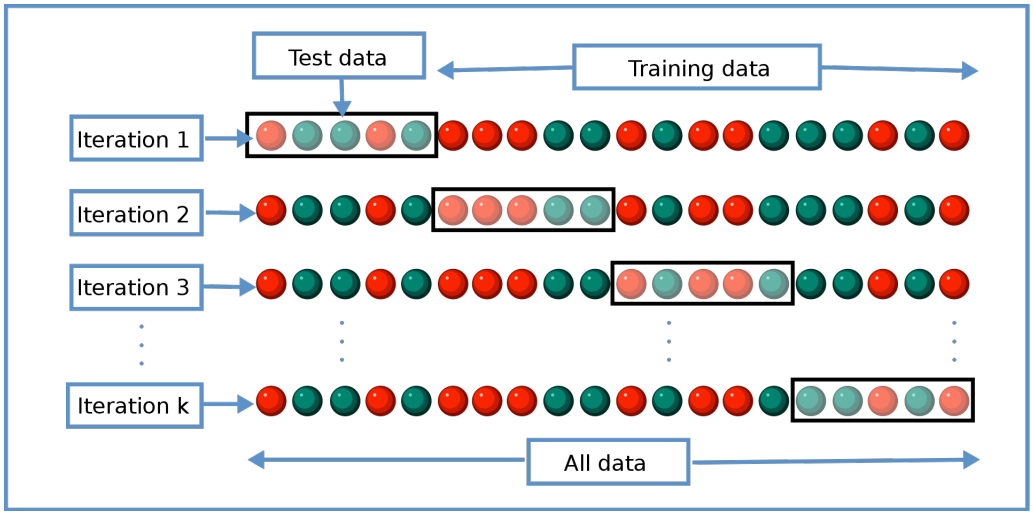


Figura 2.3.1: Diagrama de validación cruzada.

Esta técnica evita el solapamiento de los conjuntos de hold-out repetido. Además, se pueden realizar repeticiones de validación cruzada disminuyendo la variabilidad del error a costa de solapar los conjuntos de datos, aunque el problema de la dependencia es mucho menos preocupante.

Normalmente se toman valores $K = 5$ ó $K = 10$. Casos particulares donde $K = n$ (siendo n el tamaño del conjunto de datos) se denominan *Leave One Out* o LOO (dejar uno fuera). Cuanto mayor sea K, mayor será el coste computacional ya que conlleva construir K clasificadores.

2.3.3. Búsqueda de hiperparámetros

La búsqueda de hiperparámetros de un modelo es esencial, ya que el rendimiento depende en muchos casos de ellos. En este trabajo se emplea la búsqueda en rejilla aleatorizada o *randomized grid search*. Consiste en establecer unos rangos de búsqueda para cada hiperparámetro del modelo y entrenar sus posibles combinaciones aleatorizadas durante un número de iteraciones prefijadas. Lógicamente, se elegirán los hiperparámetros que mejores resultados generen.

Para ello, se reserva un conjunto de entrenamiento y test. Acto seguido, se realiza un procedimiento de validación cruzada repetida con los datos de entrenamiento a la vez que se buscan los parámetros mediante *randomized grid search*. En cada iteración de validación cruzada se prueba en los datos de test el modelo con los mejores hiperparámetros encontrados. Finalmente, se eligen los hiperparámetros que mejores resultados hayan dado entre todas las repeticiones de validación cruzada.

2.4. Interpretabilidad de modelos complejos

En muchas aplicaciones, entender por qué un modelo realiza una predicción en concreto puede ser tan importante como la precisión de dicha predicción. Hoy en día, los modelos que mejor rendimiento ofrecen suelen ser bastante complejos y resultan muy complicados de interpretar, especialmente los ensembles y las redes neuronales. Es por eso que se busca un compromiso entre precisión e interpretabilidad. En esta sección se presentan varios marcos de trabajo que permiten interpretar estos modelos: LIME [23] y SHAP [24].

2.4.1. LIME

Las «Explicaciones locales interpretables e independientes del modelo» (*Local Interpretable Model-agnostic Explanations*, LIME) [23] es una técnica de interpretación de modelos complejos que tiene como objetivo alcanzar la interpretabilidad, independencia, fidelidad local y perspectiva global en un modelo. Las dos últimas características hacen referencia a que la interpretabilidad ha de ser fiable para observaciones concretas y tiene que ser representativa del modelo. En resumen, se busca interpretar cómo se clasifica una observación individual x.
Contamos inicialmente con un modelo interpretable g y una función que mide su complejidad $\Omega(g)$. La complejidad en un árbol puede ser su profundidad mientras que en una red neuronal puede ser el número de neuronas. También tenemos un modelo no interpretable denominado f y una medida de proximidad entre instancias x y z que llamaremos $\pi_x(z)$. Finalmente definimos $\mathcal{L}(f, g, \pi_x(z))$, que nos indica cómo de poco fiable es g a la hora de aproximar f en el espacio próximo a π_x. Nuestro objetivo (ecuación 2.4.1) es minimizar $\mathcal{L}$ y mantener $\Omega(f)$ lo suficientemente bajo para que sea interpretable por un humano:

$$\xi(x) = argmin_g \mathcal{L}(f, g, \pi_x) + \Omega(f) \tag{2.4.1}$$

El procedimiento general del algoritmo consiste en tomar una observación x y perturbarla generando ruido, obteniendo un pequeño conjunto de datos Z formado por observaciones z. Con estas observaciones perturbadas y ponderadas con π_x se entrena el clasificador g.
En la Figura 2.4.1 se observa un ejemplo. La clases de la función f que buscamos interpretar se definen por el fondo rosa y azul. La instancia x que buscamos interpretar es la cruz roja grande. El algoritmo LIME muestrea las instancias z, obtiene predicciones usando f y las pondera por proximidad a la observación x. Dichos pesos se representan por tamaño. Finalmente, la explicación generada es la línea discontinua, que se aproxima razonablemente a la función f.

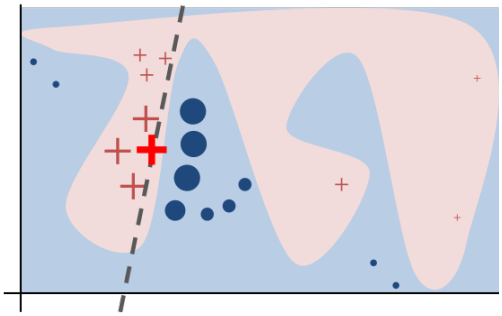


Figura 2.4.1: Ejemplo de explicación LIME, obtenida de [23].

2.4.2. Shapley value

El valor de Shapley [25] proporciona una manera de explicar predicciones de modelos no lineales en el ámbito de *machine learning*. Interpretando un modelo entrenado con una serie de variables como un valor de una función en una coalición de jugadores, los valores de Shapley son una forma natural de calcular qué variables contribuyen a una predicción.

La idea general se entiende fácilmente con un ejemplo [26]. Suponemos inicialmente que contamos con un modelo que predice los precios de un apartamento y obtenemos una predicción de 300000 euros para las siguientes variables: 50 m^2 de área, segundo piso, con un parque y mascotas prohibidas. Conocemos que la predicción media para todas las viviendas es de 310000 euros. Para averiguar el efecto de cada variable en la predicción en un modelo de regresión solo tenemos que observar los pesos. Sin embargo, en un modelo complejo no lineal hay que buscar otra solución. El valor de Shapley consiste en distribuir la predicción de forma equitativa entre las variables. En el ejemplo anterior, el parque ha podido contribuir con 30000 euros, los metros cuadrados con 10000, el segundo piso con 0 y la prohibición de mascotas con -50000. Si sumamos todo esto obtenemos -10000 euros, que es la diferencia entre el precio predicho y la media para todos los apartamentos.

Para calcular el valor de Shapley para una variable es necesario obtener la media de las contribuciones marginales de esa variable a todos los posibles grupos de variables.

Suponiendo que contamos con un modelo de regresión $\hat{f}(x) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$ donde x es la observación para la que queremos obtener las contribuciones y β_j el peso de la variable x_j. La contribución ϕ_j a la variable j-ésima de la predicción es:

$$\phi_j(\hat{f}) = \beta_j x_j - \beta_j E(X_j) \tag{2.4.2}$$

Sumando las contribuciones de cada variable para una instancia se obtiene:

$$\sum_{j=1}^{p} \phi_j(\hat{f}) = \hat{f}(x) - E(\hat{f}(X)) \tag{2.4.3}$$

Sin embargo, esto no es tan sencillo para modelos con una estructura de pesos distinta a la regresión, como son el boosting de árboles o las redes neuronales. Se utiliza una teoría de juego cooperativo. El valor de Shapley de una variable es la contribución a la predicción, ponderada y sumada sobre todas las posibles combinaciones de valores:

$$\phi_j(val) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \frac{|S|!(p - |S| - 1)!}{p!}(val(S \cup \{j\}) - val(S)) \tag{2.4.4}$$

Donde S es un sub-conjunto de las variables usadas en el modelo, x es un vector de valores de la instancia que se quiere explicar y p, el número de variables.
Para obtener los valores de Shapley cuando se tiene un gran número de variables, encontrar la solución exacta es difícil puesto que los sub-conjuntos de variables crecen exponencialmente. Se propone una aproximación por el método de Monte-Carlo:

$$\hat{\phi}_j = \frac{1}{M} \sum_{m=1}^{M} (\hat{f}(x^m_{+j}) - \hat{f}(x^m_{-j})) \tag{2.4.5}$$

En este caso $\hat{f}(x^m_{+j})$ es la predicción para x a la cual se le han reemplazado aleatoriamente un conjunto de valores de variables de una observación aleatoria z a excepción del valor de la variable a predecir j.

2.4.3. SHAP

Finalmente, llegamos a las explicaciones aditivas de Shapley (*SHapley Additive exPlanations*, SHAP) [24]. Se trata de un método que combina las técnicas de LIME y valores de Shapley para asignar a cada variable una importancia para una predicción en particular. El modelo explicativo propuesto por SHAP es el siguiente:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2.4.6}$$

Donde $z' \in \{0,1\}^M$ y M es el número de variables simplificadas. Utilizando la misma notación que en 2.4.1, x' son entradas simplificadas a través de una función $h_x(x') = x$. De esta forma,

$g(z') = f(h_x(z'))$ cuando $z' \approx x'$.
Las propiedades que busca SHAP son:

- Precisión local. Cuando se aproxima el modelo original f a una entrada x, la precisión local necesita que el modelo explicativo al menos genere la misma predicción que f para la entrada simplificada x'.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{2.4.7}$$

- Ausencia. Si las entradas simplificadas x' representan la presencia de una variable, la propiedad de ausencia requiere que las variables ausentes en la entrada original no tengan impacto.

$$x'_i = 0 \Rightarrow \phi_i = 0 \tag{2.4.8}$$

- Consistencia. Si un modelo cambia de forma que la contribución de una entrada simplificada x' se mantiene igual o aumenta sin tener en cuenta las otras entradas, la contribución de esa entrada no debería disminuir. Expresado matemáticamente, dado $f_x(z') = f(h_x(z'))$ y $z' \backslash i$ (fijar $z'_i = 0$). Para dos modelos f y f', si

$$f'_x(z') - f'_x(z' \backslash i) \geq f_x(z') - f_x(z' \backslash i) \tag{2.4.9}$$

  para cada entrada $z' \in \{0,1\}^M$, entonces $\phi_i(f', x) \geq \phi_i(f, x)$.

Finalmente, solo existe un modelo explicativo g que cumpla la ecuación 2.4.6 y las tres propiedades anteriores:

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \backslash i)] \tag{2.4.10}$$

Donde $|z'|$ es el número de entradas en z' que no son 0 y $z' \subseteq x'$ representa todos los vectores z' donde las entradas distintas de 0 son un subconjunto de las entradas distintas de 0 en x'.
Existen tres métodos para estimar los valores SHAP: *KernelSHAP*, *TreeSHAP* y *DeepSHAP*. El primero es válido para todos los modelos, mientras que los dos últimos son útiles para árboles y redes neuronales respectivamente y son los que usaremos.

- *TreeSHAP*. Es específico para ensembles de árboles y tiene un coste computacional polinómico. Esto es posible gracias a la estructura de un modelo de árboles y las propiedades

aditivas de los valores de Shapley: el valor SHAP de un ensemble es la suma de los SHAP de sus árboles. Sin embargo, no es una aproximación completamente fiable ya que se estima la esperanza condicional directamente utilizando información calculada durante el entrenamiento del modelo. Esto significa que no atribuye influencia a variables que no intervienen en el modelo ya que los árboles no contienen información sobre las variables que no aparecen [27].

- *DeepSHAP*. Mejora la eficiencia computacional en modelos de redes neuronales combinando SHAP con *DeepLIFT* [28], un método para extraer las variables importantes en redes neuronales mediante la comparación de activaciones de cada neurona con una «activación de referencia».

Capítulo 3

Datos

3.1. Descripción del conjunto de datos

El conjunto de datos completo contiene información sobre las frecuencias características asociadas a fallos en rodamientos de motores de inducción que trabajan en estado estacionario. A los motores se les introdujo carburo de silicio en sus rodamientos para dañarlos de forma controlada. Dependiendo de su estado, se clasifican en 6 clases $\{1, 2, 3, 4, 5, 6\}$ siendo 1 un motor en estado óptimo y 6 uno completamente dañado. Este set de datos ha sido obtenido y facilitado por profesores del departamento de Ingeniería Eléctrica de la Universidad de Valladolid.
El conjunto de datos principal se divide en tres subconjuntos que contienen información distinta sobre el mismo problema: Corriente, Sonido y Vibración.

3.1.1. Corriente eléctrica

Para abreviar, nos referiremos a él como conjunto de Red. Contiene datos sobre la señal eléctrica que alimenta a los motores. Dichos datos se obtienen aplicando la transformada de Fourier para pasar del dominio del tiempo al de la frecuencia, obteniendo el espectro de frecuencia. Se dispone de dos conjuntos (carga de trabajo alta y baja). Cada uno cuenta con 60 observaciones, 969 columnas que describen la señal eléctrica y una adicional que contiene el estado del motor. Cada uno de estos dos conjuntos mide tres fases distintas de la red en otras tres tablas. Las columnas contienen cuatro conjuntos de variables con valores numéricos relacionados con las frecuencias asociadas a las pistas de rodamiento externas e internas, defectos en las esferas de rodamiento y defectos en el retenedor. Cada conjunto de variables cuenta con su número de armónico (de 1 a 11) y su banda (de 1 a 11 y de 1 a -11). De esta forma se tienen $4 \cdot 11 \cdot 22 = 969$ variables.

Los cuatro grupos de variables son los siguientes:

- FTF (*Fundamental Train Frequency*). Frecuencia de operación del retenedor.
- BSF (*Ball Spin Frequency*). Frecuencia característica empleada para detectar un fallo en el rodamiento.
- BPFO (*Ball Pass Frequency Outer Race*). Frecuencia característica usada para localizar un defecto en la pista externa.
- BPFI (*Ball Pass Frequency Inner Race*). Frecuencia característica para detectar un defecto en la pista interna.

Aunque es más habitual que el motor funcione en la práctica en carga alta, se ha hecho el experimento también en carga de trabajo baja para comprobar si ésta pudiera facilitar la detección de fallos en el motor.
Además, se obtienen estadísticos resumen sobre la onda de corriente en el dominio temporal que actúan como conjunto de variables alternativo y que también se utilizarán en las siguientes secciones:

- `m1` $= \frac{1}{n}\sum_{i=1}^{n} x_i$ Momento de primer orden (media).
- `m2` $= \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$ Momento de segundo orden (varianza).
- `m3` $= \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3$ Momento de tercer orden.
- `m4` $= \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^4$ Momento de cuarto orden.
- `c1` $= m_1$ Cumulante de primer orden (media).
- `c2` $= m_2 - m_1^2$ Cumulante de segundo orden.
- `c3` $= m_3 - 3m_1m_2 + 2m_1^3$ Cumulante de tercer orden.
- `c4` $= m_4 + m_3m_1 - 3m_2^2 + 12m_2m_1^2 - 6m_1^4$ Cumulante de cuarto orden.
- `c6` $= \frac{1}{n \cdot m_2^3}\sum_{i=1}^{n}(x_i - \overline{x})^6$Cumulante de sexto orden.
- `skew` $= \frac{m_3}{(\sqrt{m_2})^4}$) *Skewness* o asimetría.
- `kurt` $= \frac{m_4}{(\sqrt{m_2})^4}$) Curtosis.
- `am` $= \frac{1}{n}\sum_{i=1}^{n}|x_i|$ Promedio de valores absolutos.
- `xp` $= \text{máx}\,|x|$ Máximo valor absoluto.
- `xr` $= (\frac{1}{n}\sum_{i=1}^{n}|x_i|)^2$ Valor cuadrático medio.

- $\texttt{cf} = \frac{\texttt{xp}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x-\overline{x})^2}}$ Factor de cresta.
- $\texttt{sf} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x-\overline{x})^2}}{|\overline{x}|}$ Factor de forma.

3.1.2. Sonido

Analizando el periodograma de las grabaciones con micrófono de los motores en funcionamiento, se obtienen de nuevo dos sets de datos (carga alta y baja) con 60 filas, 16 variables correspondientes a los estadísticos de orden superior descritos en la Sección 3.1.1 y la clasificación del motor.

3.1.3. Vibración

Analizando el periodograma de los valores recogidos por el acelerómetro, se obtienen seis conjuntos de datos (carga alta y baja por los ejes X, Y, Z correspondientes a la vibración) con 60 filas, 16 variables correspondientes a los estadísticos de orden superior de la Sección 3.1.1 y la clasificación del motor.

3.2. Procesamiento

Inicialmente, se realizan operaciones sencillas sobre los conjuntos de datos para limpiarlos y mejorar su usabilidad. También se lleva a cabo un análisis de componentes principales para buscar relaciones entre las variables e intentar reducir su dimensionalidad.

3.2.1. Transformaciones iniciales

A cada subconjunto de datos se le han aplicado distintas transformaciones, que coinciden para los sets de carga alta y baja:

- Red. Los datos erróneos (con valor -200) que ocurren debido a fallos en el instrumento de medida o falta de precisión, se sustituyen por nulos. De acuerdo a las recomendaciones de los expertos del Departamento de Ingeniería Eléctrica se utiliza la media de 3 subconjuntos correspondientes a las fases de la onda sin tener en cuenta los valores nulos anteriores con el objetivo de reducir el número de variables desde el principio. Finalmente, las celdas con valor nulo tras la operación anterior se eliminan.
  Con respecto al conjunto de estadísticos, se concatenan las variables de cada fase de la onda, manteniendo las 60 filas pero triplicando el número de columnas.

- Sonido. Sin transformaciones.
- Vibraciones. Se han concatenado las columnas de cada eje de vibración, añadiendo las variables correspondientes a cada eje. Se mantienen las 60 observaciones y se triplica el número de columnas.

3.2.2. Análisis de componentes principales

A continuación se explican los resultados obtenidos tras realizar un análisis de componentes principales sobre los conjuntos de datos transformados.

Red

Para los datos de **Red Carga Alta** observamos en la Figura 3.2.1 que la componente 1 recoge casi toda la variabilidad de los datos. Según el scree plot podemos tomar las dos primeras componentes y mantener un 81 % de la variabilidad total. Sin embargo, el 90 % se alcanza con 7 componentes, que suelen considerarse demasiadas.

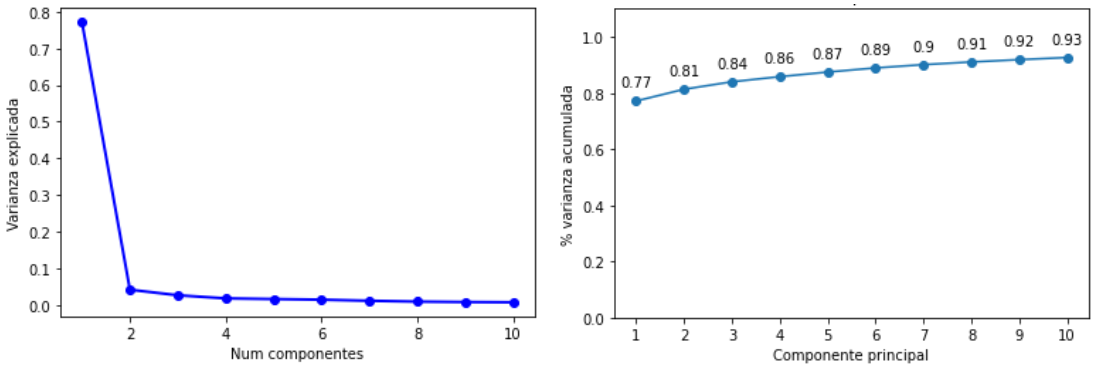


Figura 3.2.1: Scree plot y porcentaje de varianza acumulada para Red Carga Alta.

En el gráfico de individuos de la Figura 3.2.2 comprobamos que cada estado se centra en una parte razonablemente pequeña del espacio (sobre todo los estados poco dañados 1 al 3). Sin embargo, los puntos están demasiado mezclados como para considerar que los estados están separados.

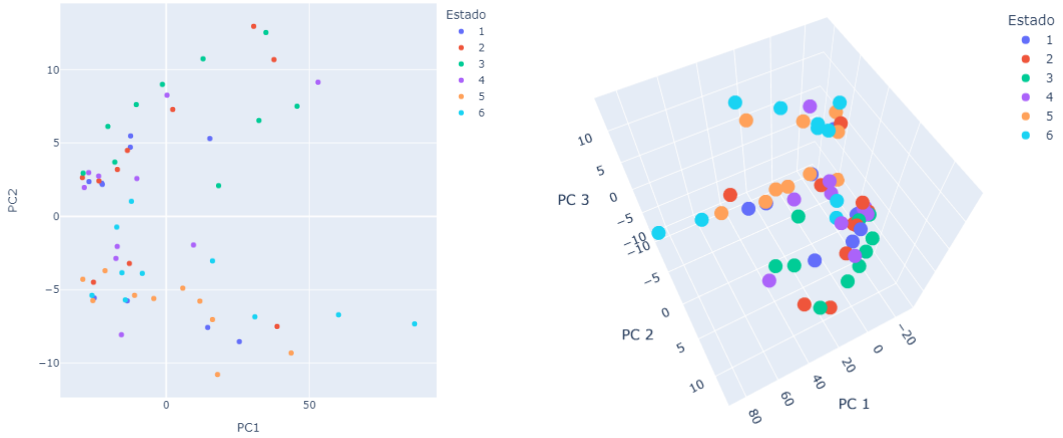


Figura 3.2.2: Scatter plot para individuos en las dos y tres primeras componentes para Red Carga Alta.

Destacamos que las correlaciones o cargas de las variables con la primera componente principal son muy elevadas. De 965 variables, 589 tienen un coeficiente de correlación con la primera componente superior a 0.9 en valor absoluto. Además, 237 lo tienen superior a 0.99. Esto disminuye notablemente para la segunda componente, que solo presenta una variable con un coeficiente superior a 0.8.

Para los datos de **Red Carga Baja**, los resultados son similares. La primera componente recoge aún más variabilidad que en los datos de carga alta, aunque la conclusión sigue siendo que lo más indicado es tomar las 2 ó 3 primeras componentes según la Figura 3.2.3.

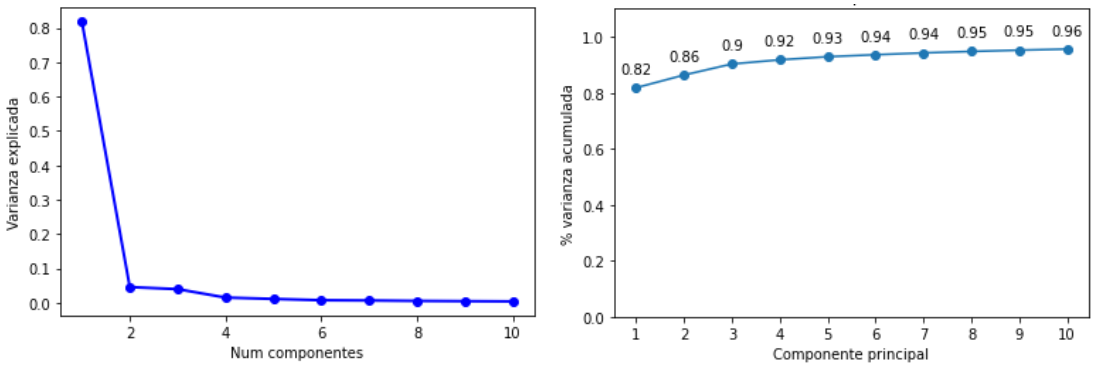


Figura 3.2.3: Scree plot y porcentaje de varianza acumulada para Red Carga Baja.

Los gráficos de individuos de la Figura 3.2.4 parecen concentrar en áreas más reducidas los

estados, pero se siguen mezclando demasiado entre ellos. Conforme se deteriora el motor en los últimos estados, los puntos parecen separarse más.

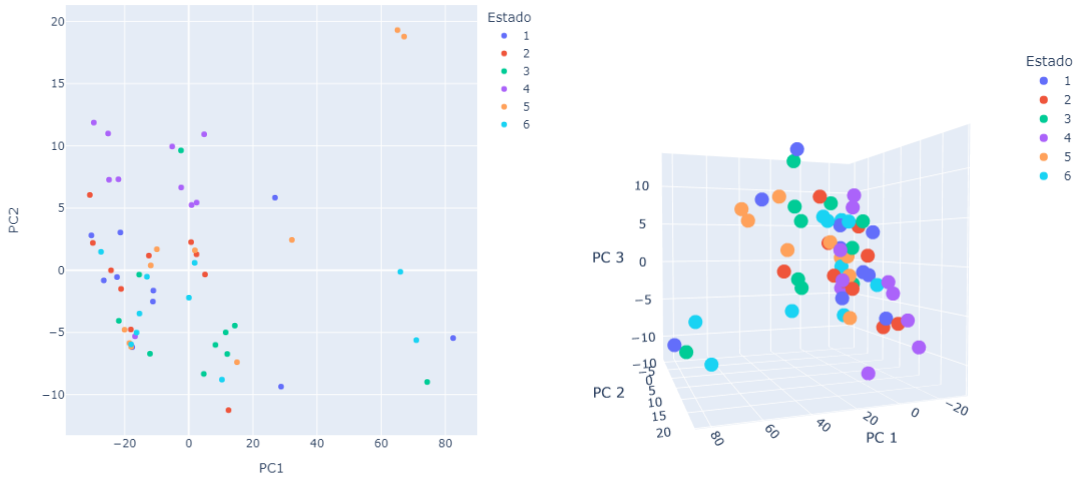


Figura 3.2.4: Scatter plot para individuos en las dos y tres primeras componentes para Red Carga Baja.

A continuación, en los datos de **Estadísticos de Red Carga Alta** observamos en la Figura 3.2.5 que la variabilidad se recoge de forma más gradual que en los conjuntos anteriores.

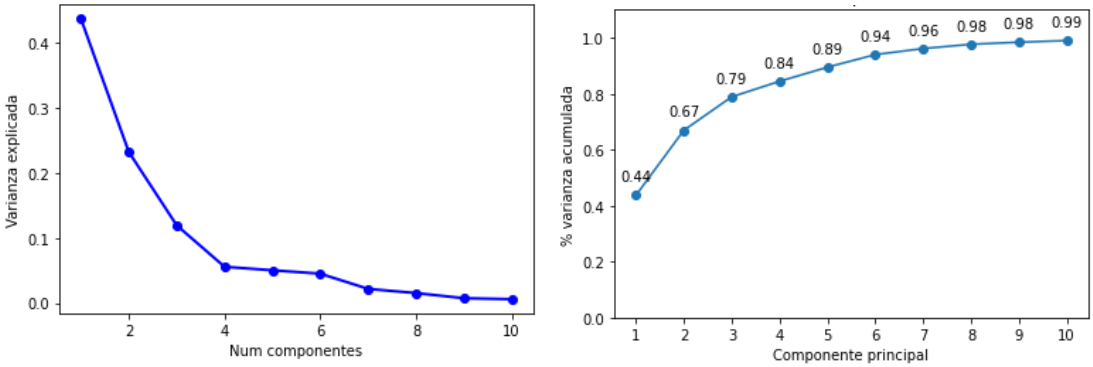


Figura 3.2.5: Scree plot y porcentaje de varianza acumulada para Estadísticos Red Carga Alta.

En esta ocasión se muestra también en la Figura 3.2.6 el gráfico de cargas, ya que el número más reducido de variables lo permite. Destacamos que las variables iguales de distintas fases (etiquetadas con `{variable}_{fase}`) se agrupan. El gráfico de individuos muestra una separación mucho más clara, especialmente en las tres primeras componentes.

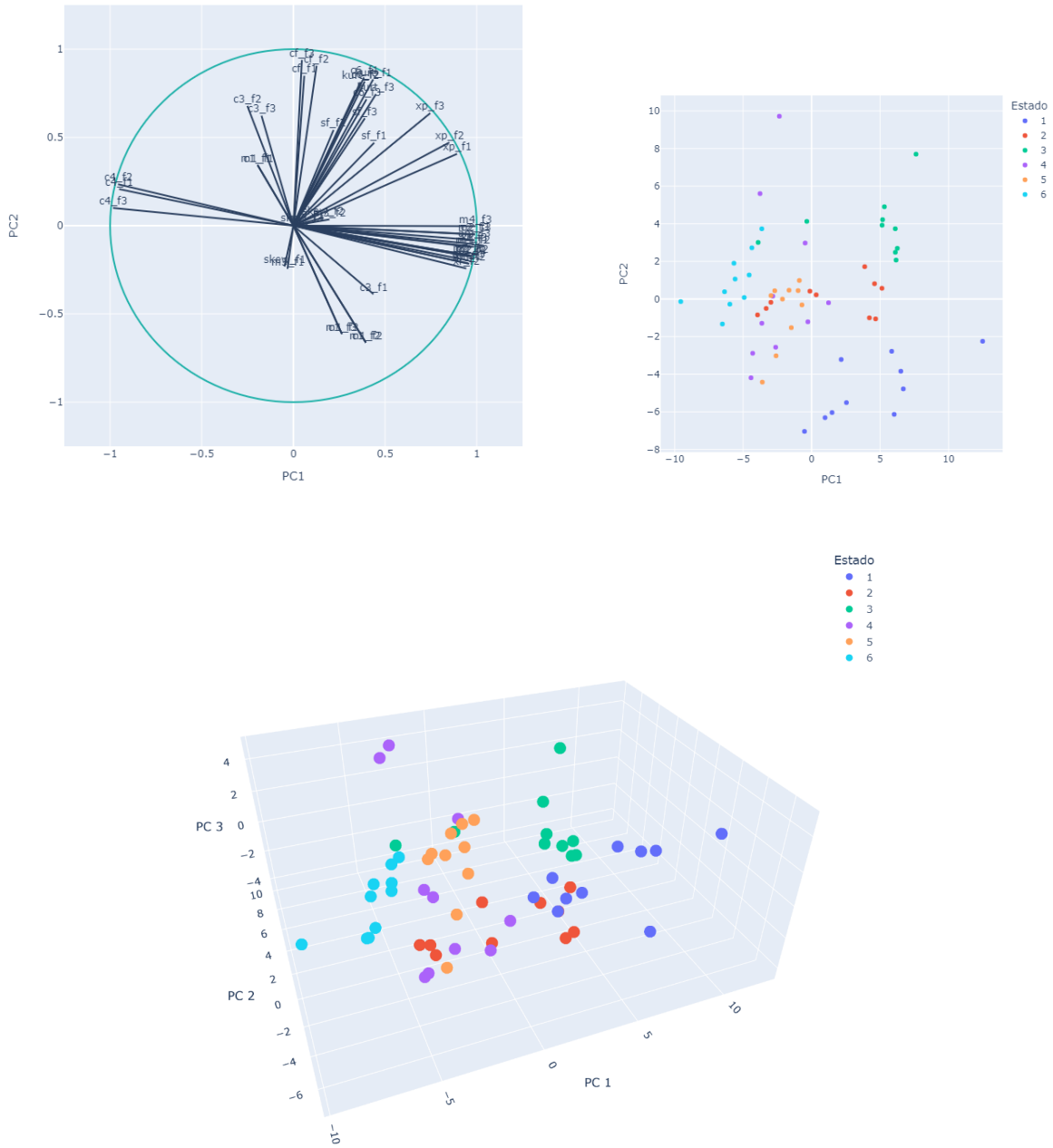


Figura 3.2.6: Gráfico de cargas y scatter plot para individuos en las dos y tres primeras componentes para Estadísticos Red Carga Alta.

Finalmente, en los datos de **Estadísticos de Red Carga Baja** observamos en la Figura 3.2.7 un comportamiento similar al conjunto anterior, donde se recoge gradualmente la variabilidad hasta llegar al 90 % en la sexta componente.

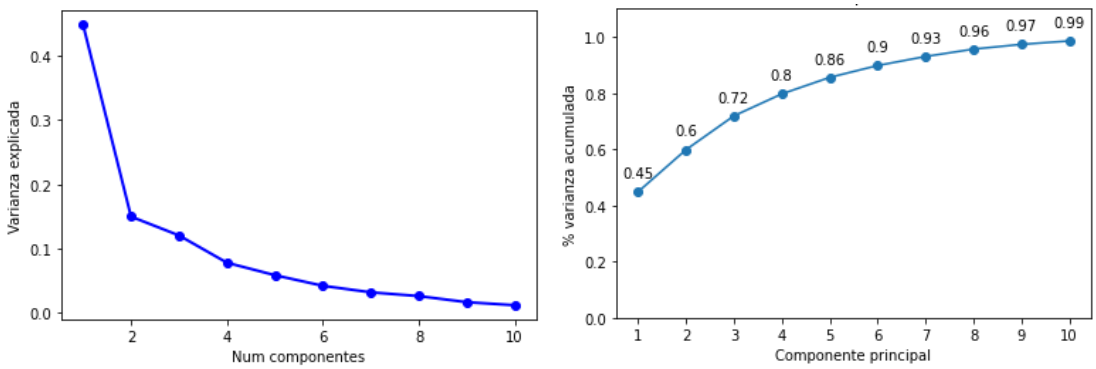


Figura 3.2.7: Scree plot y porcentaje de varianza acumulada para Estadísticos Red Carga Baja.

Como se ve en la Figura 3.2.8, las variables se agrupan de forma similar en el gráfico de cargas y en el gráfico de individuos se aprecia una mejor separación de las clases, especialmente en el de las tres primeras componentes.

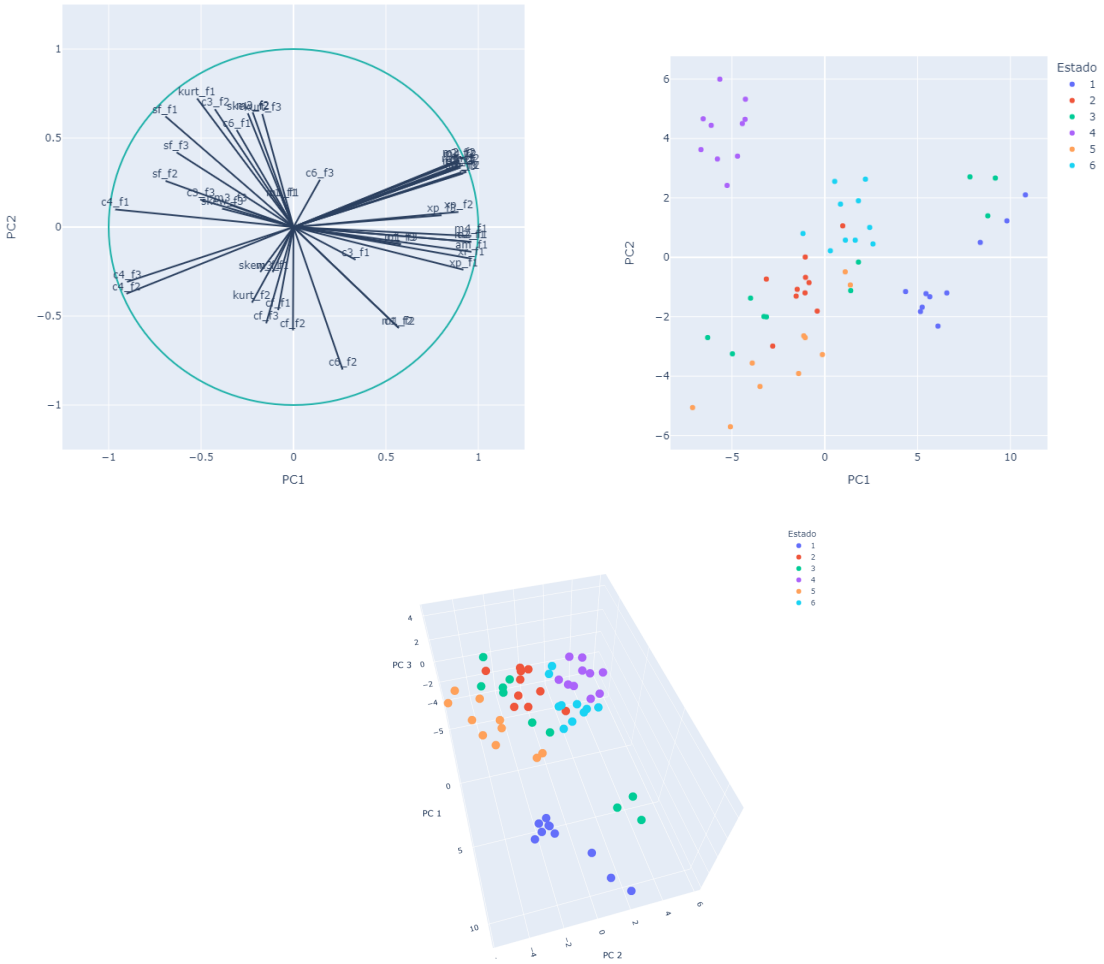


Figura 3.2.8: Gráfico de cargas y scatter plot para individuos en las dos y tres primeras componentes para Estadísticos Red Carga Baja.

Sonido

En el conjunto de datos de **Sonido Carga Alta**, la primera componente recoge casi un 60 % de la variabilidad. En este caso, comprobamos que a partir de la sexta componente se alcanza el 100 % de variabilidad explicada 3.2.9. Tomar las dos o tres primeras sería lo más adecuado.

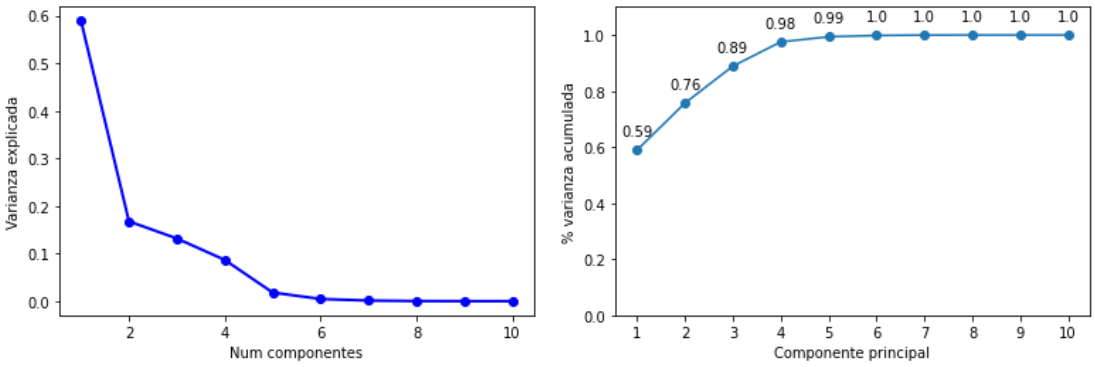


Figura 3.2.9: Scree plot y porcentaje de varianza acumulada para Sonido Carga Alta.

A continuación, observamos el gráfico de cargas 3.2.10 (correlaciones de las variables con las dos primeras componentes). En él comprobamos que las variables agrupadas corresponden a estadísticos que se calculan de forma similar o directamente a partir de otro estadístico. Casi todas las variables tienen correlaciones relativamente altas con los dos ejes, a excepción de `m3` y `skew`. Hay que destacar que un ángulo cercano a 90 grados entre las variables es indicador de que son casi incorreladas. El gráfico de individuos 3.2.10 muestra unos resultados mucho mejores que en el conjunto de datos de red. Las clases menos deterioradas parecen formar grupos en rectas. Destacan los estados 5 y 6 ya que presentan valores más extremos y se encuentran más esparcidos.

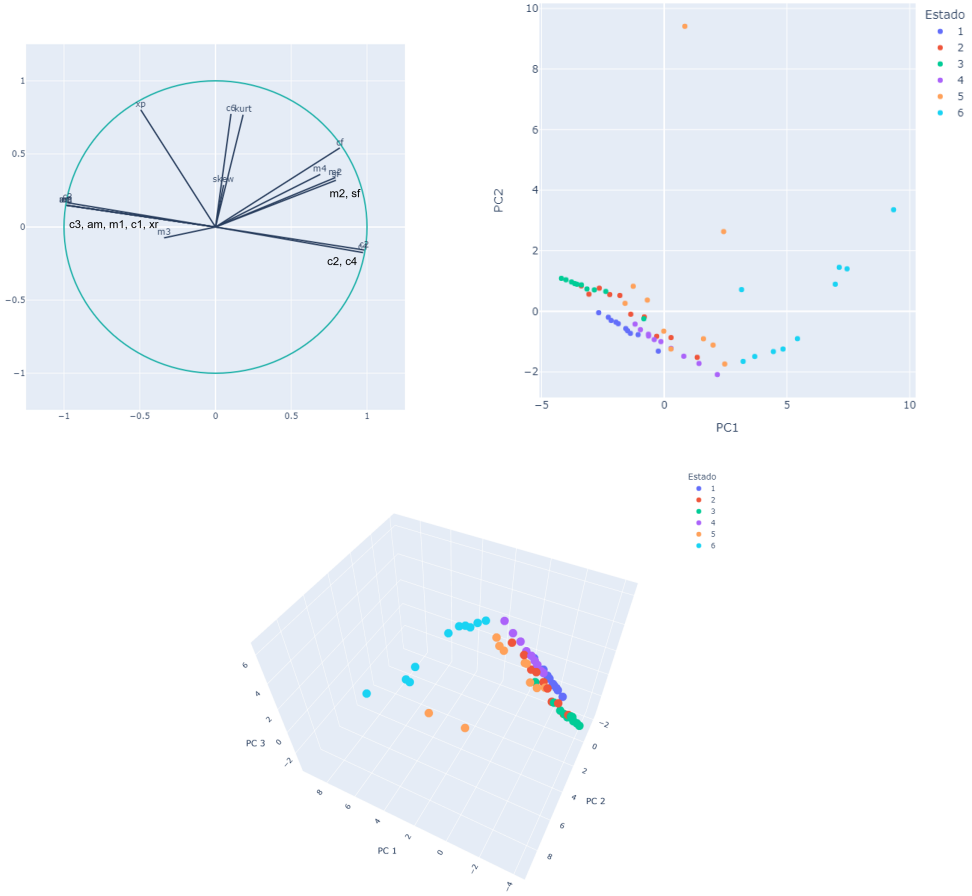


Figura 3.2.10: Gráfico de cargas y scatter plot para individuos en las dos y tres primeras componentes para Sonido Carga Alta.

Para los datos **Sonido Carga Baja** comprobamos que tanto el scree plot y la varianza acumulada son bastante similares al conjunto de carga alta según la Figura 3.2.11. Tomaremos también las tres primeras componentes.

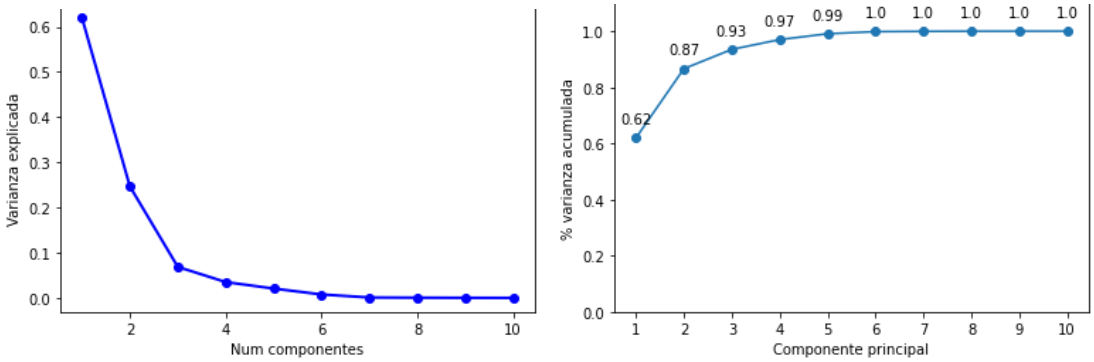


Figura 3.2.11: Scree plot y porcentaje de varianza acumulada para Sonido Carga Baja.

Lo mismo ocurre con el gráfico de cargas, que es muy similar al del subconjunto de carga alta. En este caso las correlaciones con las componentes son más altas y variables como `skew` y `kurt` pasan a estar correladas de forma negativa con la segunda componente y de forma positiva con la primera.

A continuación, en el gráfico de individuos mostrado en la Figura 3.2.12 se observa que la separación de clases es mucho más clara que en los motores en carga alta. A medida que aumenta el deterioro del motor, los puntos dejan de alinearse. De nuevo, los estados 5 y 6 son más irregulares.

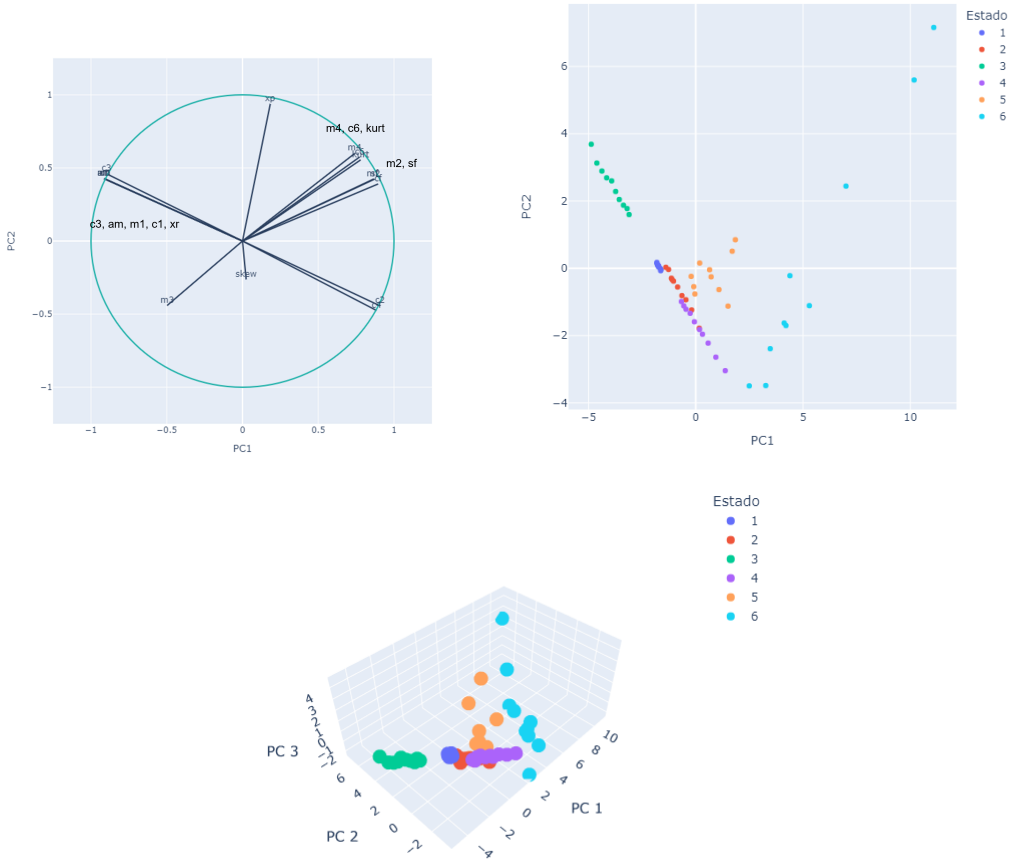


Figura 3.2.12: Gráfico de cargas y scatter plot para individuos en las dos y tres primeras componentes para Sonido Carga Baja.

Vibración

En este conjunto de datos recordamos que las vibraciones se registran en tres ejes (X, Y, Z) por lo que realizamos el análisis para el conjunto completo con todos los ejes y en cada eje por separado. De esta forma podemos identificar el eje que más información proporciona

sobre el estado del motor. Primero, para **Vibraciones XYZ Carga Alta** observamos que son necesarias 6 componentes para alcanzar el 90 % de variabilidad acumulada según la Figura 3.2.13. Esto puede deberse a que el conjunto de datos tiene el triple de variables en comparación con el de sonido.

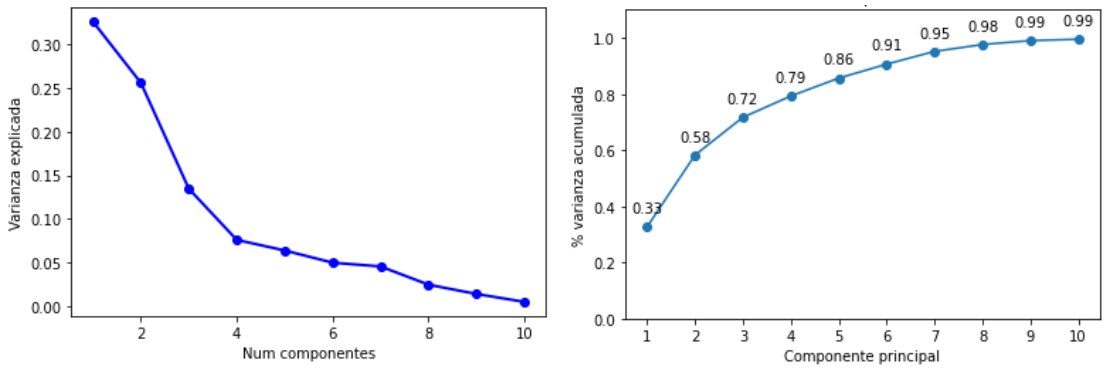


Figura 3.2.13: Scree plot y porcentaje de varianza acumulada para Vibraciones XYZ Carga Alta.

En el gráfico de cargas (Figura 3.2.14) para los tres ejes comprobamos que se agrupan las variables pertenecientes al mismo eje.
En los gráficos de individuos 3.2.14 destaca la tendencia a formar rectas con los estados del motor agrupados. En este caso el estado 6 es el más irregular y destaca el estado 5 por estar muy relacionado con el 3.

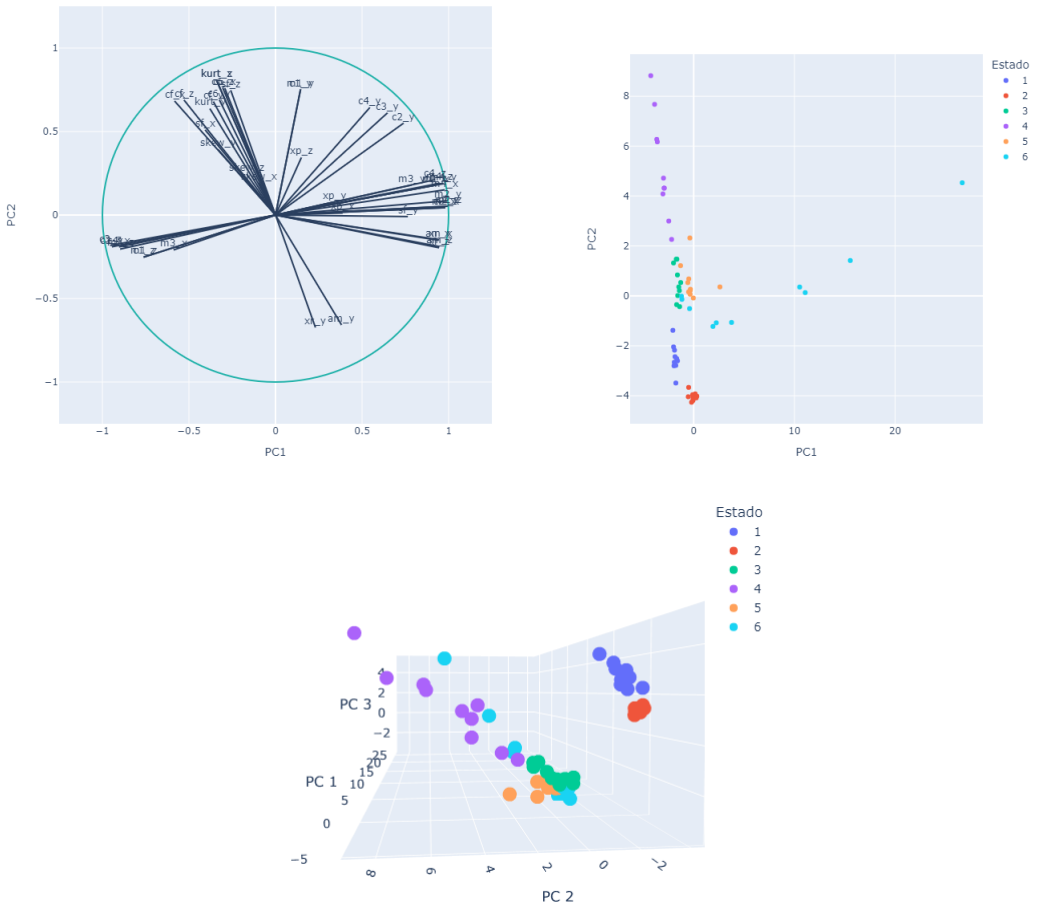
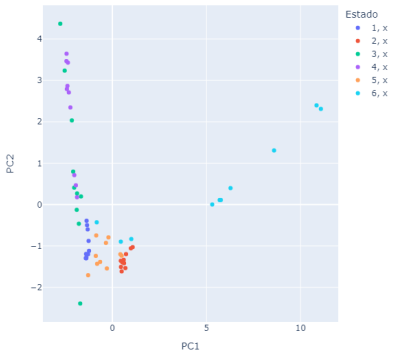
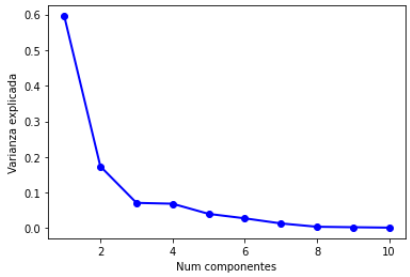


Figura 3.2.14: Scatter plot para individuos en las dos y tres primeras componentes para Vibraciones XYZ Carga Alta.

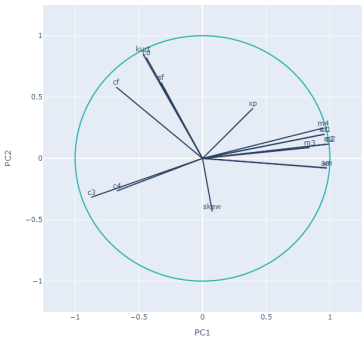
Separando por ejes el análisis en la Figura 3.2.15, se ve claramente que el eje que mejor discrimina los estados es el Y, mientras que el X y Z son bastante similares. Los scree plot indican que las 3 primeras componentes acumulan suficiente variabilidad. Finalmente, en los gráficos de cargas destaca el eje Y al ser bastante distinto al resto y presentar en sus variables correlaciones con las componentes algo más elevadas.



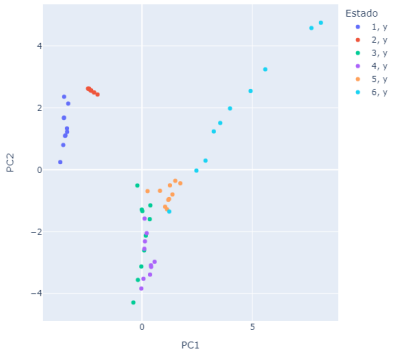
Individuos eje X



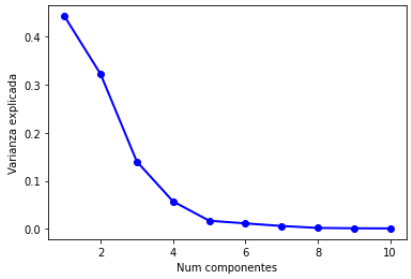
Scree plot eje X



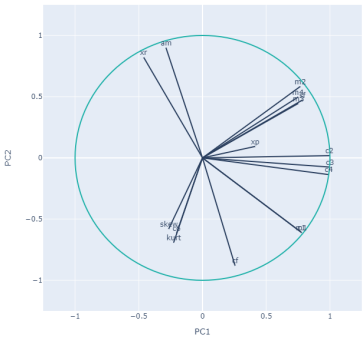
Cargas eje X



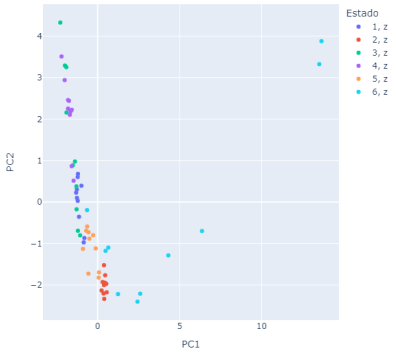
Individuos eje Y



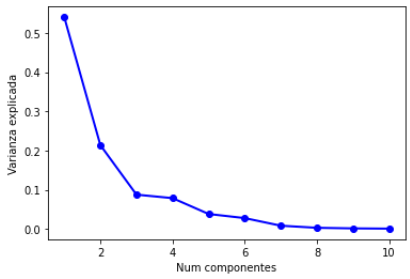
Scree plot eje Y



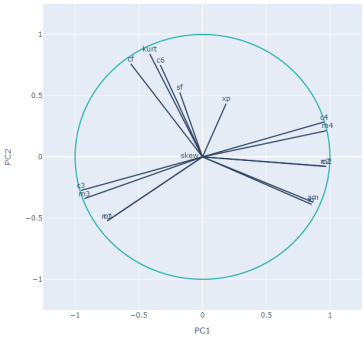
Cargas eje Y



Individuos eje Z



Scree plot eje Z



Cargas eje Z

Figura 3.2.15: Plots para cada eje por separado de Vibraciones Carga Alta.

Finalmente, el análisis para **Vibraciones XYZ Carga Baja**, al igual que en carga alta, genera conclusiones a favor de que el eje Y discrimina mejor que el resto.

En el scree plot de la Figura 3.2.16 vuelven a ser necesarias 6 componenetes para acumular el

90 % de la varianza. Sin embargo, las primeras acumulan más que en carga alta.

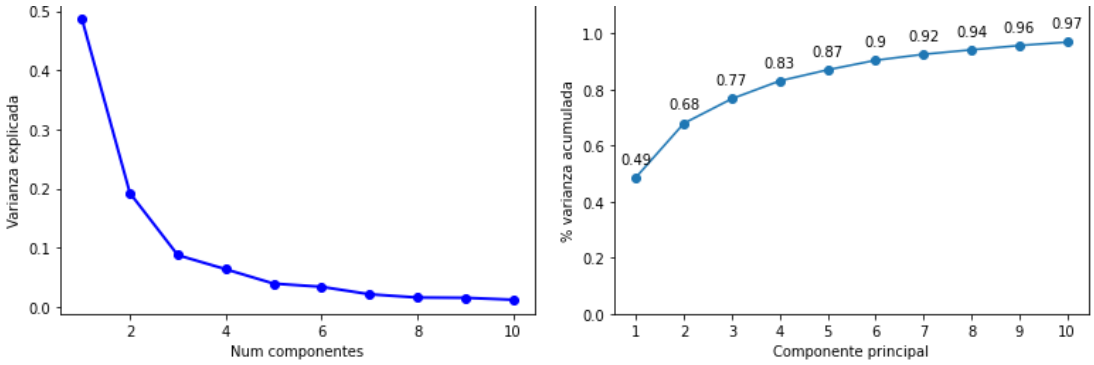


Figura 3.2.16: Scree plot y porcentaje de varianza acumulada para Vibraciones XYZ Carga Baja.

El gráfico de cargas mostrado en la Figura 3.2.17 agrupa de nuevo los ejes y algunos signos de las correlaciones cambian con respecto al conjunto de carga alta.
El gráfico de individuos 3.2.17 es similar al de carga alta. Ahora el estado 5 se separa mejor del 3 y vuelve a dejar de alinearse de la misma forma que el resto, al igual que el 6.

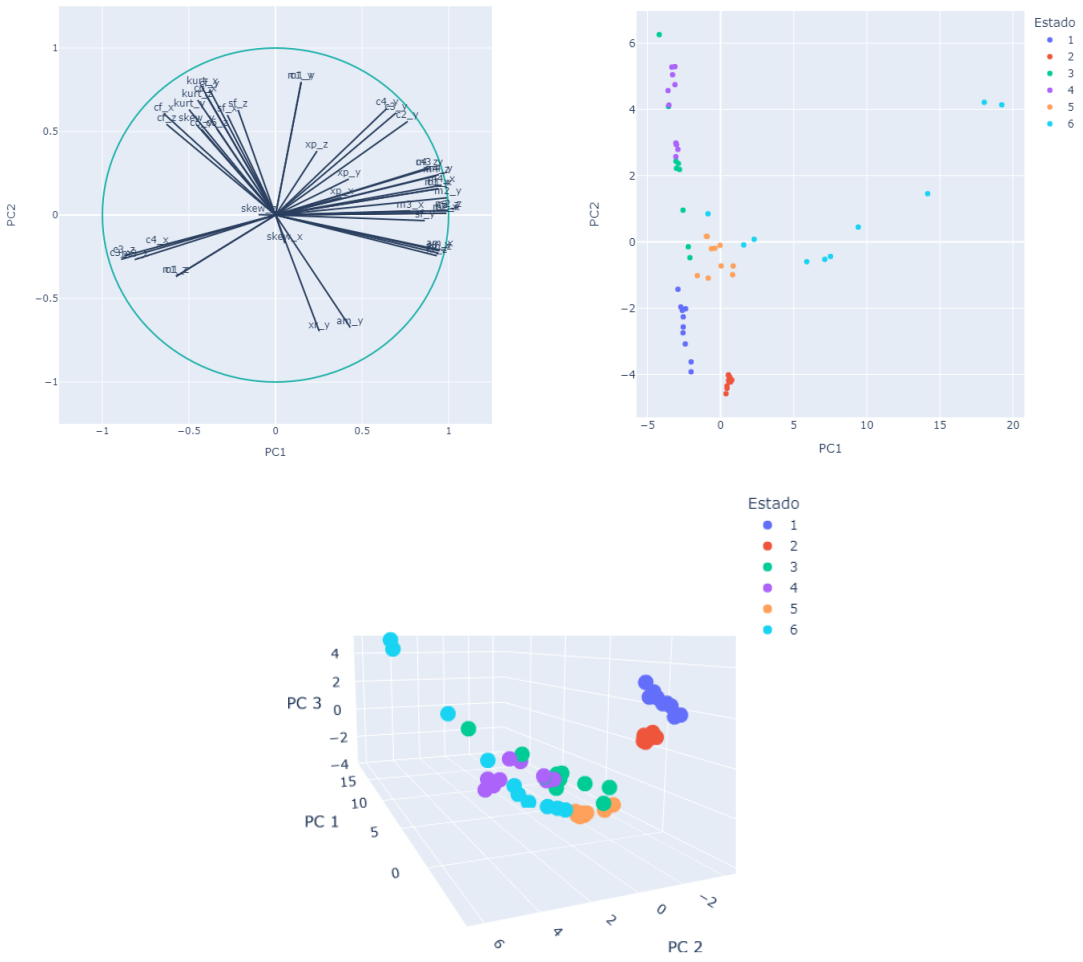
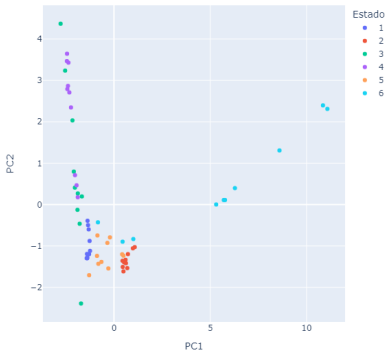
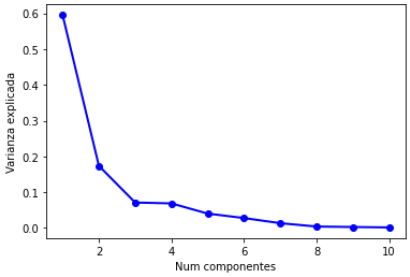


Figura 3.2.17: Scatter plot para individuos en las dos y tres primeras componentes para Vibraciones XYZ Carga Baja.

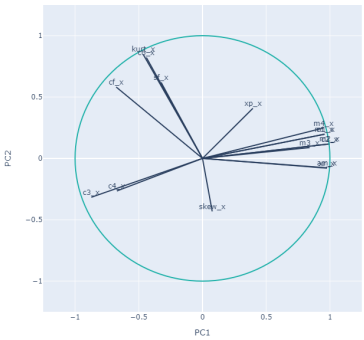
Finalmente, separando el análisis en los tres ejes (Figura 3.2.18) observamos que el eje Y ofrece una mejor discriminación de las clases o estados al igual que en el conjunto de datos separado por ejes para carga baja.



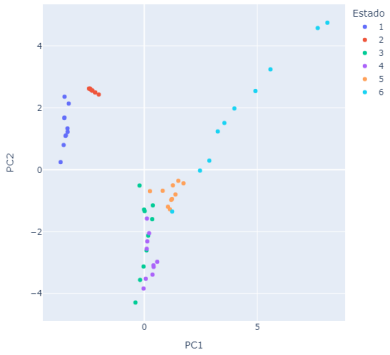
Individuos eje X



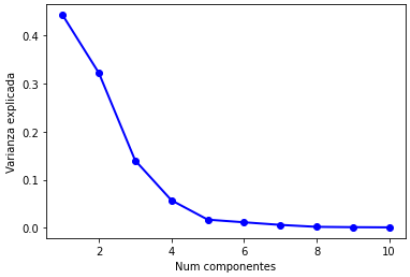
Scree plot eje X



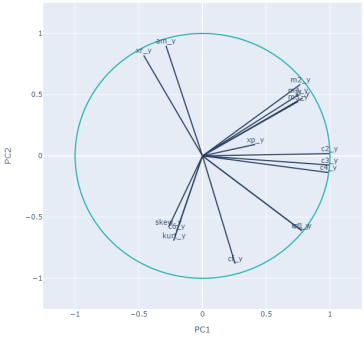
Cargas eje X



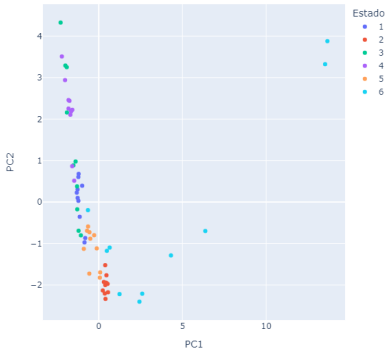
Individuos eje Y



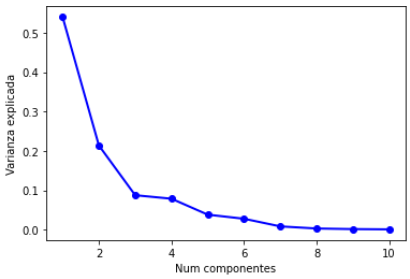
Scree plot eje Y



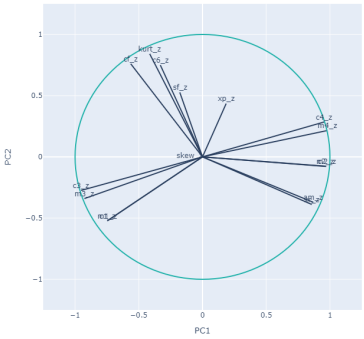
Cargas eje Y



Individuos eje Z



Scree plot eje Z



Cargas eje Z

Figura 3.2.18: Plots para cada eje por separado de Vibraciones Carga Baja.

3.2.3. Selección de variables

Tras el análisis de componentes principales, podemos generar los conjuntos de datos con los que se entrenarán los clasificadores.

Red

Al tratarse del set con más columnas, el análisis de componentes permitirá reducir bastante la dimensión manteniendo casi toda la información de los datos. Se trabajará con tres conjuntos tanto para carga alta como baja:

- Conjunto original formado por 965 variables.
- Conjunto formado por las 7 primeras componentes principales ya que acumulan el 90 % de la variabilidad total.
- A partir del análisis de componentes principales se elige la variable con mayor correlación con la primera componente (`BSF_5_6`) y se eliminan aquellas cuya correlación es superior a 0.9 ya que aportan información similar. De esta forma, el conjunto final tiene 316 variables.

Para el caso de los estadísticos de orden superior, inicialmente eliminamos las variables que se calculan igual: `c1`. Eliminamos `c6` porque es casi igual que `kurt` y `am` ya que está muy asociada a `xr`. Ahora, para cada nivel de carga se obtienen nuevos subconjuntos:

- Conjunto con las variables concatenadas en las columnas. Se toman las variables correspondientes a las tres fases y se unen en un solo conjunto.
- Conjunto con las 6 primeras componentes principales, que corresponden al 90 % de variabilidad acumulada.
- Se separan las fases en varios conjuntos, obteniendo uno para cada fase.

Sonido

Al ser el cálculo de estadísticos equivalente al conjunto de Red, se eliminan las mismas variables. De esta forma obtenemos dos conjuntos para cada subconjunto de carga alta y baja:

- Conjunto con las variables eliminadas con 13 columnas en total.
- Tres primeras componentes principales (necesarias para acumular el 90 % de variabilidad).

Vibración

Con las variables a eliminar, ocurre lo mismo que con el conjunto de Sonido y se descartan los mismos estadísticos, obteniéndose tres conjuntos para carga alta y baja:

- Conjunto con las variables eliminadas con 40 columnas en total (correspondientes a cada eje al igual que con las fases de Red).
- Conjunto con las 3 primeras componentes principales (necesarias para acumular el 90 % de variabilidad).
- Se separan los ejes en varios conjuntos, obteniendo uno para cada eje.

Capítulo 4

Análisis

En este capítulo se explica el procedimiento seguido para obtener los modelos predictivos. Primero se define el experimento que se ha realizado y a continuación se realiza un análisis de la varianza (ANOVA) sobre diversos factores que afectan a los modelos para comprobar las diferencias entre sus rendimientos. Finalmente, para los resultados significativos obtenidos con los ANOVA, se realiza un filtrado y un posterior análisis más detallado de su clasificación.

4.1. Experimento

El procedimiento realizado para obtener las medidas del error de clasificación con las que se trabaja en las próximas secciones es el siguiente:

1. División del conjunto de datos mediante *hold-out*. Se utiliza una partición de entrenamiento (*train*) y otra de prueba (*test*) con proporciones 0.8 y 0.2 respectivamente.
2. Con la partición *train* se realiza validación cruzada de 5 particiones con búsqueda de hiperparámetros aleatoria. Se eligen 5 particiones en vez de 10 debido a que el conjunto de datos es bastante pequeño.
3. Se construye un modelo con los mejores hiperparámetros del paso anterior y se estima su tasa de error con el conjunto de prueba *test*. De esta forma obtenemos un estimador del error lo menos sesgado posible, ya que son datos que el clasificador no ha visto nunca.
4. Se repiten los pasos anteriores 20 veces para reducir la variabilidad en la estimación del error.

Las combinaciones de conjuntos de datos sobre los que se aplica el experimento son:

- Datos originales y datos resumidos mediante el análisis de componentes principales (ACP).
- Nivel de carga del motor: Alta y Baja (CA, CB).
- Algoritmo XGBoost y redes neuronales.
- Datos de red, sonido y vibraciones. Adicionalmente, las vibraciones se dividen en tres ejes. Los datos de red se dividen en tres fases y dos cojuntos de variables: estadísticos de orden superior y frecuencias de los rodamientos explicados en la sección 3.1.1.

Los hiperparámetros que se afinan en el modelo de boosting son la tasa de aprendizaje, el número de árboles, su profundidad y el peso en las hojas. Adicionalmente se buscan parámetros de regularización como L1, L2 y la reducción en la función de pérdida en particiones. Finalmente se ajustan los parámetros de muestreo tanto en filas como en columnas. Destacamos que no se realiza una búsqueda exhaustiva de hiperparámetros, sino que es una búsqueda aleatoria que para tras 100 iteraciones ó 10 iteraciones sin mejora en el error de clasificación.
El modelo planteado mediante redes neuronales es «sencillo» debido a las pocas observaciones y a la tendencia al sobreajuste de las redes. Se utiliza un modelo de 3 capas ocultas con activación ReLU. Para prevenir el sobreajuste se utiliza *dropout* con probabilidad 0.2 y normalización de las entradas. La búsqueda de hiperparámetros se realiza sobre la tasa de aprendizaje y las épocas de entrenamiento.
Todos los experimentos se han realizado en Python con las librerías `XGBoost` [29], `PyTorch` [30] para construir los modelos y `SciKit-Learn` [31] para evaluarlos.

4.2. Tasas de error iniciales

A continuación se exponen en los Cuadros 4.2.1 y 4.2.2 las tasas de error medias obtenidas en las 20 repeticiones de validación cruzada, separadas por conjuntos de datos, tanto con XGBoost como con redes neuronales.

Conjunto	**Variables**	**Nivel de carga**	**Transformación**	
			Original	**PCA**
Red	Frecuencias	CA	0.625	0.721
		CB	0.654	0.658
	Estadísticos	CA	0.038	0.283
		CB	0.013	0.217
Sonido	Estadísticos	CA	0.183	0.438
		CB	0.017	0.162
Vibraciones	Estadísticos	CA	0.086	0.15
		CB	0.075	0.13

Cuadro 4.2.1: Tasas de error iniciales para XGBoost.

Conjunto	**Variables**	**Nivel de carga**	**Transformación**	
			Original	**PCA**
Red	Frecuencias	CA	0.604	0.717
		CB	0.7	0.738
	Estadísticos	CA	0.03	0.217
		CB	0	0.108
Sonido	Estadísticos	CA	0.308	0.308
		CB	0.067	0.058
Vibraciones	Estadísticos	CA	0.083	0.05
		CB	0.192	0.06

Cuadro 4.2.2: Tasas de error iniciales para redes neuronales.

Las tasas de error para el conjunto de Frecuencias de Red son muy malas a pesar de haber realizado una selección de variables, extraer las componentes principales y una búsqueda de hiperparámetros.

El rendimiento para el resto de conjuntos basados en los estadísticos de orden superior es mucho mejor. Las tasas de error no superan el 0.2 salvo en casos puntuales y el conjunto de variables sin transformar es mejor que las componentes principales en casi todos los casos.

El boosting da un comportamiento más estable que las redes neuronales ya que sus tasas de error son menos variables entre experimentos.

En las secciones siguientes se añade más granularidad al problema definiendo factores más específicos sobre el conjunto de datos.

4.3. Factores

Con el objetivo de elegir el mejor clasificador, el mejor conjunto de datos (según calidad de clasificación) y reducir el tamaño del problema, definimos una serie de factores que permitirán, mediante un análisis de la varianza, seleccionar aquellos conjuntos que den mejores resultados. Los factores no son comunes para todo el set de datos, por lo que el análisis del error se realiza en dos etapas. La primera seleccionará los niveles óptimos para los factores no comunes. La segunda permitirá evaluar todos factores restantes puesto que los conjuntos disponibles tienen los mismos niveles. En los Cuadros 4.3.1, 4.3.2 y 4.3.3 observamos los factores definidos y sus detalles.

Factor	**Niveles**	**Significado**
conjunto	`red`, `sonido`, `vibraciones`	Conjunto de datos.
transf	`original`, `pca`	Transformación de los datos: variables sin transformar o componentes principales.
nc	`ca`, `cb`	Nivel de carga de trabajo en el motor: alta o baja.
alg	`xgb`, `nn`	Algoritmo de clasificación: boosting o redes neuronales.

Cuadro 4.3.1: Factores para todos los conjuntos de datos.

Factor	**Niveles**	**Significado**
fase	`f1, f2, f3, todos`	Variables para cada fase de la onda.
variables	`est, frec`	Conjunto de variables: estadísticos o frecuencias

Cuadro 4.3.2: Factores para el conjunto de datos de Red.

Para cada combinación de factores se realiza una ejecución del experimento descrito en la sección 4.1. De esta forma, se tienen para el conjunto de factores del Cuadro 4.3.2 4 Fases, 2 Variables, 2 Transformaciones, 2 Niveles de carga y 2 Algoritmos.

Factor	**Niveles**	**Significado**
eje	`x, y, z, todos`	Eje de vibración.

Cuadro 4.3.3: Factores para el conjunto de datos de Vibraciones.

Para los factores del Cuadro 4.3.3 contamos con 4 Ejes, 2 Transformaciones, 2 Niveles de carga y 2 Algoritmos. Finalmente en el Cuadro 4.3.1 hay 3 Conjuntos, 2 Transformaciones, 2 Niveles de carga y 2 Algoritmos. Todo esto nos da 1280, 640 y 480 observaciones respectivamente.
Los ANOVA se realizan tanto para las tasas de error como para el MAE como se explica en la sección 2.3.1.

4.4. Análisis de los ejes de vibración

Inicialmente realizamos un análisis sobre el conjunto de vibraciones para determinar qué eje es el que mejor clasifica. Realizamos un ANOVA de un factor sobre el error de clasificación y el MAE con 4 niveles correspondientes a los ejes X, Y, Z y Todos. Los resultados del experimento aparecen en los Cuadros 4.4.1 y 4.4.2.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
eje	3	26.895	8.965	99.638	2e-15
Residuals	636	57.225	0.090		

Cuadro 4.4.1: ANOVA sobre la tasa de error. Factor eje de vibración.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
eje	3	1.278	0.426	33.899	2e-15
Residuals	636	7.989	0.013		

Cuadro 4.4.2: ANOVA sobre el MAE. Factor eje de vibración.

Las diferencias entre los ejes son muy significativas. Procedemos a realizar un test *post-hoc* para encontrar las diferencias entre cada par de ejes. Se realiza un test de Duncan sobre la tasa de error con $\alpha = 0.05$. El resultado está en el Cuadro 4.4.3.

	Error	**Std**	**R**	**Min**	**Max**	**Groups**
todos	0.104	0.087	160	0	0.416	a
x	0.159	0.118	160	0	0.583	b
y	0.136	0.095	160	0	0.333	b
z	0.226	0.139	160	0	0.583	c
Alpha:	0.05	Df Error:	636			

Cuadro 4.4.3: Test de Duncan para la tasa de error de los Ejes de vibración.

En la columna *Groups* observamos los grupos de ejes que ha encontrado el test, de forma que en las filas etiquetadas con la misma letra no se han encontrado diferencias significativas. El factor *todos* (los tres ejes) genera una clasificación distinta al resto, mientras que el X y el Y ofrecen resultados similares. El eje Z es distinto del resto.
Representando las métricas del error mediante un boxplot en la Figura 4.4.1 observamos dichas diferencias entre ellas.

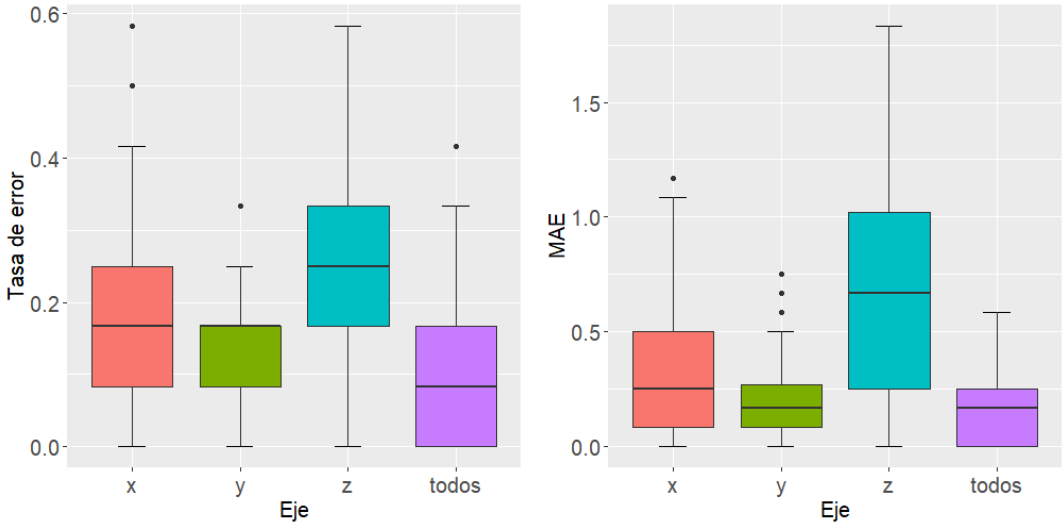


Figura 4.4.1: Boxplot para las métricas del error según Eje. Tasa de error (izquierda) y MAE (derecha).

Hacer uso de todos los ejes de vibración mejora la clasificación de los modelos, llegándose a obtener tasas de error nulas. Destaca entre el eje Y porque genera resultados mucho mejores que los otros dos.
Concluimos que utilizar todos los ejes dará el mejor rendimiento, por lo que a partir de ahora se trabajará con todos los ejes.

4.5. Análisis del conjunto de Red

De la misma forma que se ha hecho con el conjunto de vibración, procedemos a comparar los sets de datos de Red. Primero analizamos la fase de la onda para el conjunto de estadísticos para después comparar dicho conjunto con el de frecuencias asociadas a las pistas de rodamiento.

4.5.1. Fase de la onda

Se comparan las fases de la onda (F1, F2, F3 y Todas) en el conjunto de estadísticos mediante un ANOVA de un factor sobre las métricas del error, de forma análoga a la Sección 4.4. Observamos el resultado en los Cuadros 4.5.1 y 4.5.2.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
fase	3	0.838	0.279	12.338	8e-8
Residuals	476	10.781	0.023		

Cuadro 4.5.1: ANOVA sobre la tasa de error. Factor fases de onda.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
fase	3	3.295	1.098	12.274	9e-8
Residuals	476	42.592	0.089		

Cuadro 4.5.2: ANOVA sobre el MAE. Factor fases de onda.

Comprobamos que se detectan diferencias en ambas métricas del error. De nuevo, se lleva a cabo el test de Duncan para comprobar qué fases se diferencian entre sí de forma significativa. Los resultados aparecen en el Cuadro 4.5.3.

	Error	Std	R	Min	Max	Groups
todos	0.139	0.129	120	0	0.500	a
f1	0.198	0.123	120	0	0.500	b
f2	0.234	0.162	120	0	0.583	bc
f3	0.247	0.179	120	0	0.666	c
Alpha:	0.05	Df Error:	476			

Cuadro 4.5.3: Test de Duncan para la tasa de error de las Fases de Red.

Representamos a continuación en un gráfico de cajas (Figura 4.5.1) los datos sobre los que se ha efectuado el estudio.

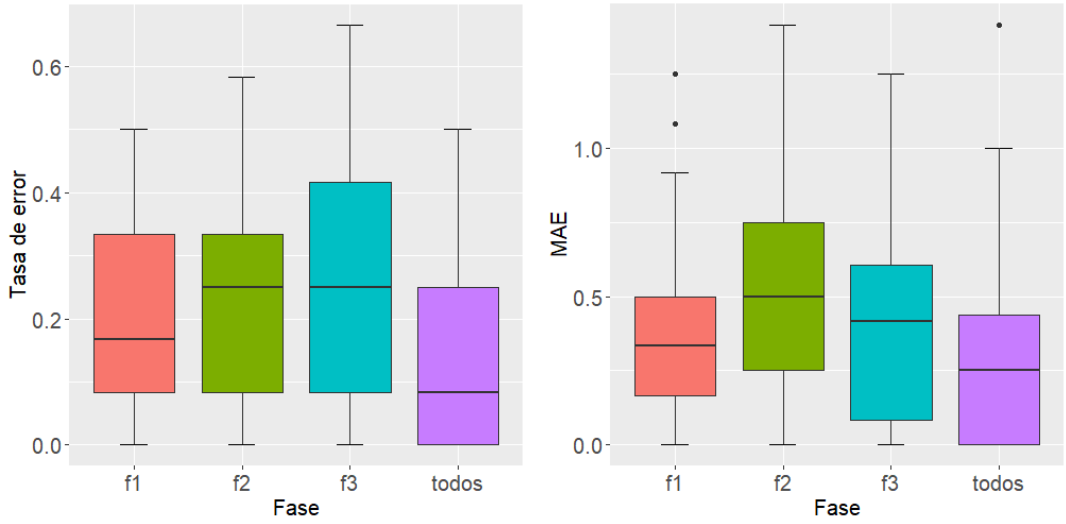


Figura 4.5.1: Boxplot para las métricas del error según Fase. Tasa de error (izquierda) y MAE (derecha).

Es fácil ver que incluir todas las fases en la clasificación mejora el rendimiento del modelo. Todos los resultados en el experimento con todas las fases son mejores, llegando a tener tasas

de error nulas. Destaca la Fase 1 entre el resto al ser sus resultados ligeramente mejores.
A la vista de estos resultados se trabajará con todas las fases.

4.5.2. Set de variables

A continuación, realizamos otro ANOVA de un factor sobre el conjunto de variables que toma dos niveles: Estadísticos (est) o Frecuencias (frec). En las primeras pruebas se ha comprobado que los resultados de los estadísticos son mucho mejores. Estos resultados se confirman con la información de los Cuadros 4.5.4 y 4.5.5.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
variables	1	25.359	25.359	1638.5	2e-15
Residuals	318	4.922	0.016		

Cuadro 4.5.4: ANOVA sobre la tasa de error. Factor Conjunto.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
variables	1	137.157	137.157	984.85	2e-15
Residuals	318	44.287	0.139		

Cuadro 4.5.5: ANOVA sobre el MAE. Factor Conjunto.

Las diferencias son muy significativas, como era de esperar. Las podemos observar en los diagramas de cajas de la Figura 4.5.2.

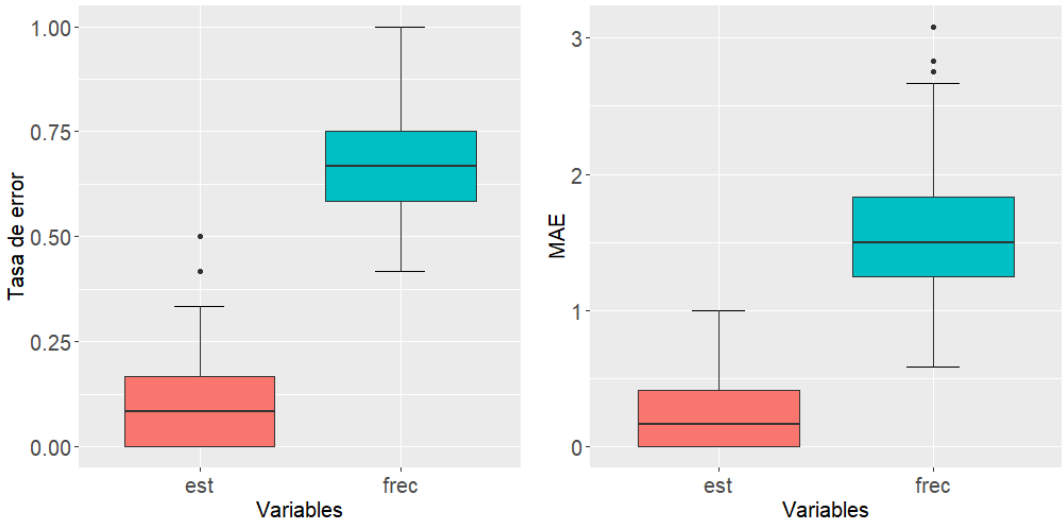


Figura 4.5.2: Boxplot para las métricas del error según Variables. Tasa de error (izquierda) y MAE (derecha).

El conjunto de estadísticos da tasas de error muy buenas, que no llegan a superar el 0.5. Sin embargo, el set de datos de frecuencias genera resultados demasiado malos y casi aleatorios. De ahora en adelante se utilizará el conjunto de estadísticos en los datos de Red.

4.6. Análisis general

Finalmente, ya que se han filtrado los conjuntos de datos con factores que no eran transversales para el set de datos general, podemos realizar un ANOVA de múltiples factores con interacción para comparar todos los resultados. Los Cuadros 4.6.1 y 4.6.2 muestran los resultados de dicho análisis.

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
transf	1	0.986	0.986	137.853	0.000
nc	1	0.971	0.971	135.748	0.000
alg	1	0.077	0.077	10.784	0.001
conjunto	2	0.753	0.377	52.678	0.000
transf:nc	1	0.090	0.090	12.629	0.000
transf:alg	1	0.608	0.608	85.044	0.000
nc:alg	1	0.002	0.002	0.245	0.621
transf:conjunto	2	0.762	0.381	53.297	0.000
nc:conjunto	2	1.369	0.685	95.765	0.000
alg:conjunto	2	0.028	0.014	1.945	0.144
transf:nc:alg	1	0.000	0.000	0.051	0.822
transf:nc:conjunto	2	0.000	0.000	0.026	0.974
transf:alg:conjunto	2	0.076	0.038	5.284	0.005
nc:alg:conjunto	2	0.068	0.034	4.762	0.008
transf:nc:alg:conjunto	2	0.048	0.024	3.341	0.036
Residuals	456	3.260	0.007		

Cuadro 4.6.1: ANOVA sobre la tasa de error. Factores Transformación, NC, Alg y Conjunto.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
transf	1	4.126	4.126	108.777	0.000
nc	1	4.095	4.095	107.964	0.000
alg	1	0.033	0.033	0.879	0.349
conjunto	2	4.254	2.127	56.087	0.000
transf:nc	1	0.033	0.033	0.879	0.349
transf:alg	1	1.595	1.595	42.047	0.000
nc:alg	1	0.026	0.026	0.673	0.412
transf:conjunto	2	2.452	1.226	32.330	0.000
nc:conjunto	2	5.093	2.547	67.149	0.000
alg:conjunto	2	0.328	0.164	4.327	0.014
transf:nc:alg	1	0.013	0.013	0.343	0.558
transf:nc:conjunto	2	0.161	0.080	2.116	0.122
transf:alg:conjunto	2	0.369	0.185	4.876	0.008
nc:alg:conjunto	2	0.103	0.052	1.363	0.257
transf:nc:alg:conjunto	2	0.055	0.028	0.729	0.483
Residuals	456	17.294	0.038		

Cuadro 4.6.2: ANOVA sobre el MAE. Factores Transformación, NC, Alg y Conjunto.

Comprobamos que los grados de libertad de los residuales son bastante elevados para tener tan solo 60 clasificaciones de motores, lo que provocará que se detecten diferencias en el test de la F cuando las sumas de cuadrados sean pequeñas.

Debido a que la hipótesis de independencia que debería cumplir un ANOVA no es del todo cierta puesto que una misma observación puede utilizarse varias veces en la validación cruzada repetida, las diferencias que se detectan mediante los p-valores no son completamente fiables. Como solución se opta por elegir un p-valor más estricto, tomando como corte el 0.005. De esta forma, se reduce el error de tipo I: las diferencias que se detectan a este nivel tienen más probabilidades de serlo [3].

Existen diferencias entre la información de los dos cuadros anteriores. Específicamente en el factor Algoritmo (alg), se detectan diferencias usando la tasa de error, pero no con el MAE. Podemos deducir de este resultado que un algoritmo realiza menos clasificaciones erróneas pero esas clasificaciones se confunden entre clases más alejadas, mientras que otro se confunde en más ocasiones (mayor tasa de error) pero entre clases más próximas (mismo MAE). Esto se debe a que el MAE penaliza menos el equivocarse entre el estado 1 y el 2 que entre el 1 el 6. De momento consideraremos las diferencias como significativas para estudiar más a fondo el resultado en secciones posteriores. Restringiendo el p-valor a 0.005 y eliminando hacia atrás progresivamente las interacciones de orden más alto, las tablas ANOVA con los factores y sus interacciones significativos se encuentran en los Cuadros 4.6.3 y 4.6.4.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
transf	1	0.986	0.986	132.472	0.000
nc	1	0.971	0.971	130.449	0.000
alg	1	0.077	0.077	10.363	0.001
conjunto	2	0.753	0.377	50.621	0.000
transf:nc	1	0.091	0.091	12.137	0.001
transf:alg	1	0.608	0.608	81.724	0.000
transf:conjunto	2	0.762	0.381	51.216	0.000
nc:conjunto	2	1.369	0.685	92.027	0.000
Residuals	468	3.482	0.007		

Cuadro 4.6.3: ANOVA sobre la tasa de error. Factores significativos Transformación, NC, Alg y Conjunto.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
transf	1	4.126	4.126	105.217	0.000
nc	1	4.095	4.094	104.429	0.000
alg	1	0.033	0.033	0.850	0.357
conjunto	2	4.254	2.127	54.251	0.000
transf:nc	1	0.033	0.033	0.850	0.357
transf:alg	1	1.595	1.595	40.670	0.000
transf:conjunto	2	2.452	1.226	31.272	0.000
nc:conjunto	2	5.093	2.547	64.951	0.000
Residuals	468	18.350	0.039		

Cuadro 4.6.4: ANOVA sobre el MAE. Factores significativos Transformación, NC, Alg y Conjunto.

Si bien en un caso como este lo relevante van a ser las interacciones entre los factores, se hará un estudio completo tanto de interacciones presentes en el modelo como de los factores principales.

4.6.1. Factor Conjunto

Los factores asociados a los tres conjuntos de datos (conjunto: red, sonido y vibraciones) son significativos. Llevamos a cabo un test de Duncan para comprobar las diferencias entre los conjuntos (ver Cuadros 4.6.5 y 4.6.6). La calidad de clasificación es diferente entre ellos. En la Figura 4.6.1 observamos los diagramas de cajas correspondientes.

	Error	Std	R	Min	Max	Groups
red	0.114	0.131	160	0	0.500	b
sonido	0.192	0.165	160	0	0.583	a
vibraciones	0.104	0.087	160	0	0.416	b
Alpha:	0.05	Df Error:	468			

Cuadro 4.6.5: Test de Duncan para la tasa de error de los Conjuntos.

	MAE	Std	R	Min	Max	Groups
red	0.240	0.278	160	0	1	b
sonido	0.390	0.354	160	0	1.412	a
vibraciones	0.164	0.147	160	0	0.583	c
Alpha:	0.05	Df Error:	468			

Cuadro 4.6.6: Test de Duncan para el MAE de los Conjuntos.

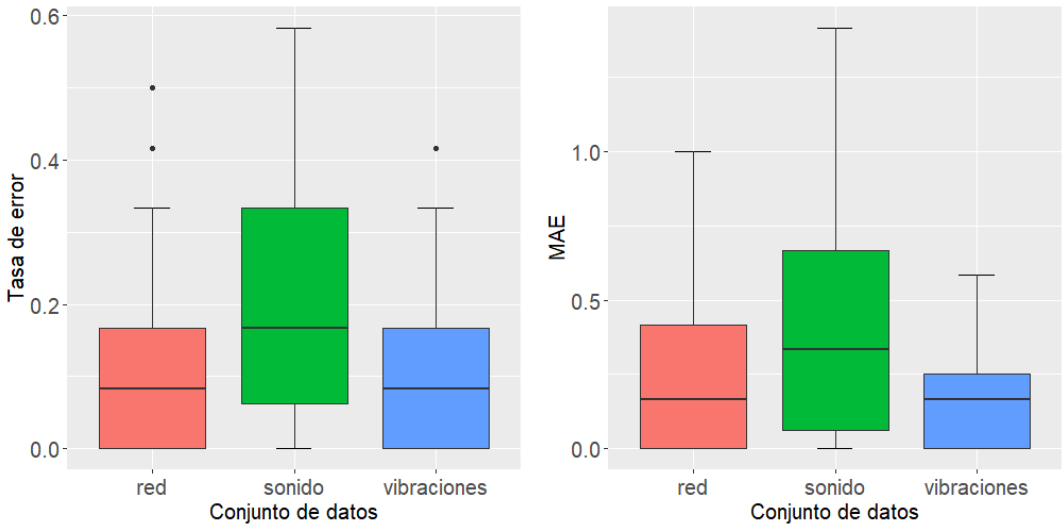


Figura 4.6.1: Boxplot para las métricas del error según Conjunto. Tasa de error (izquierda) y MAE (derecha).

Comprobamos que el conjunto de datos de sonido genera los peores resultados en base a las medidas del error con diferencia. Su mediana para ambas métricas es la más alta y la tasa de error llega a alcanzar el 0.6. Esto último puede deberse al mal comportamiento con el conjunto transformado mediante ACP con XGBoost. Existe una interacción entre Transformación y Conjunto, por lo que se estudiará con más detalle en secciones siguientes.
En cuanto al conjunto de Red y Vibraciones, sus tasas de error son muy similares pero su MAE es bastante diferente. Comprobamos que el conjunto de vibraciones tiene menos variabilidad o lo que es lo mismo, sus errores se producen entre clases próximas entre sí.

4.6.2. Factor Nivel de carga

Los niveles de carga a los que trabaja el motor (nc: ca y cb, carga alta y baja respectivamente), como era de esperar, influyen en la clasificación. Comprobamos sus diferencias en la Figura 4.6.2.

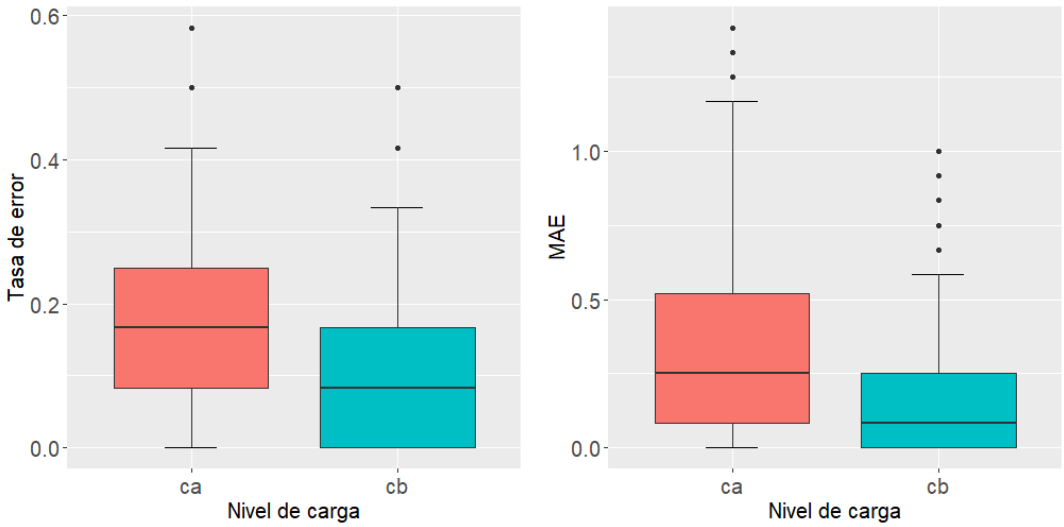


Figura 4.6.2: Boxplot para las métricas del error según Nivel de carga (nc). Tasa de error (izquierda) y MAE (derecha).

El factor Nivel de carga alto tiene peores resultados, aunque las diferencias no son tan acusadas como se pensaba. Utilizar el nivel de carga bajo resultará en un clasificador con mejor rendimiento. Este resultado no es deseable puesto que los motores están diseñados para trabajar en el Nivel de carga alto y ser lo más productivos posible. Sin embargo, es lógico pensar que se produzcan más valores atípicos y errores de precisión en las mediciones que resulten de un ruido mayor en los datos de carga alta, ruido que dificulta el ajuste de los modelos.

4.6.3. Factor Transformación

El factor Transformación del conjunto (original, pca) influye en la calidad de clasificación. Inicialmente al ver los resultados del análisis de componentes principales y la gran cantidad de varianza que acumulaban las primeras componentes se pensó que el rendimiento con las variables transformadas sería bueno. Podemos ver los resultados en la Figura 4.6.3.

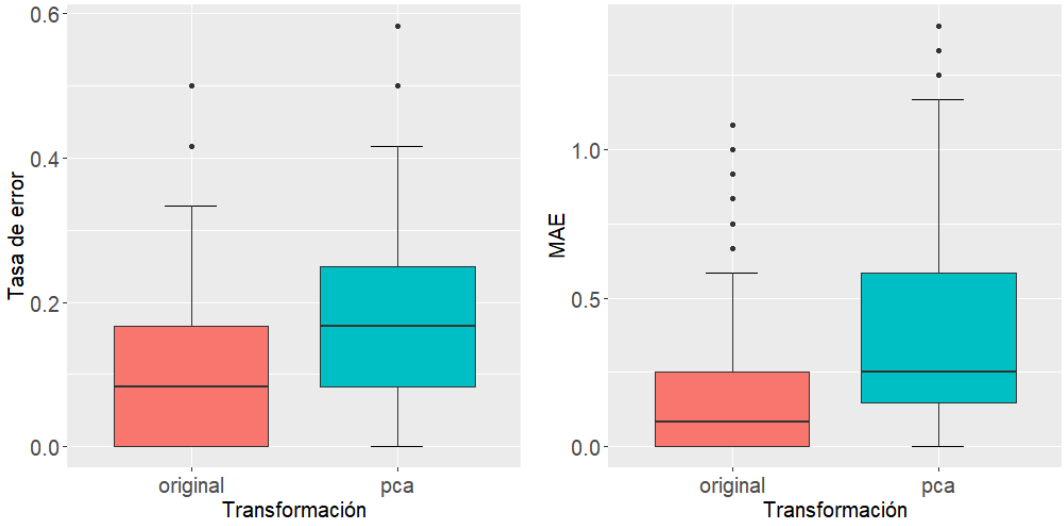


Figura 4.6.3: Boxplot para las métricas del error según Transformación del conjunto. Tasa de error (izquierda) y MAE (derecha).

En ambos casos no utilizar las componentes principales para clasificar proporciona mejores resultados. A pesar de que las componentes acumulan el 90 % de variabilidad y en todos los conjuntos cuentan con menos variables, la calidad de los modelos empeora. Destacamos que incluso si no se hubieran detectado diferencias, sería preferible no realizar la transformación ya que se pierde interpretabilidad.

4.6.4. Factor Algoritmo

El factor Algoritmo (alg) resulta interesante, puesto que se detectan diferencias entre las tasas de error pero no entre el MAE. Se observa en la Figura 4.6.4.

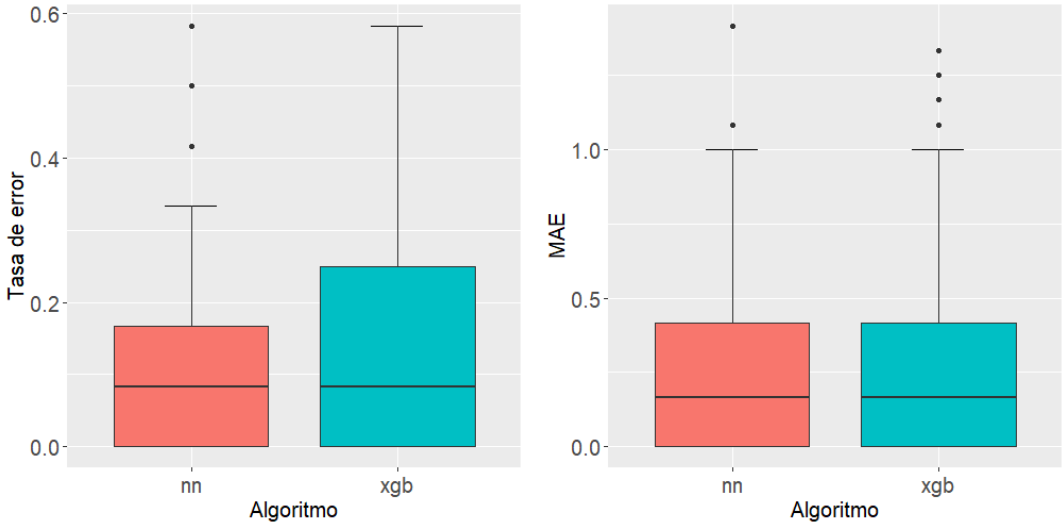


Figura 4.6.4: Boxplot para las métricas del error según Algoritmo (alg). Tasa de error (izquierda) y MAE (derecha).

Comprobamos que las medianas para ambas métricas del error en los dos algoritmos coinciden y su rango de valores es casi el mismo, siendo ligeramente menor en boosting. XGBoost genera un comportamiento más estable a la vista del MAE y aunque cometa más errores de clasificación, lo hace entre clases próximas.

4.6.5. Interacción Conjunto:Transformación

La primera interacción significativa es la relativa a los factores Conjunto y Transformación. En la Figura 4.6.5 podemos observar los resultados.

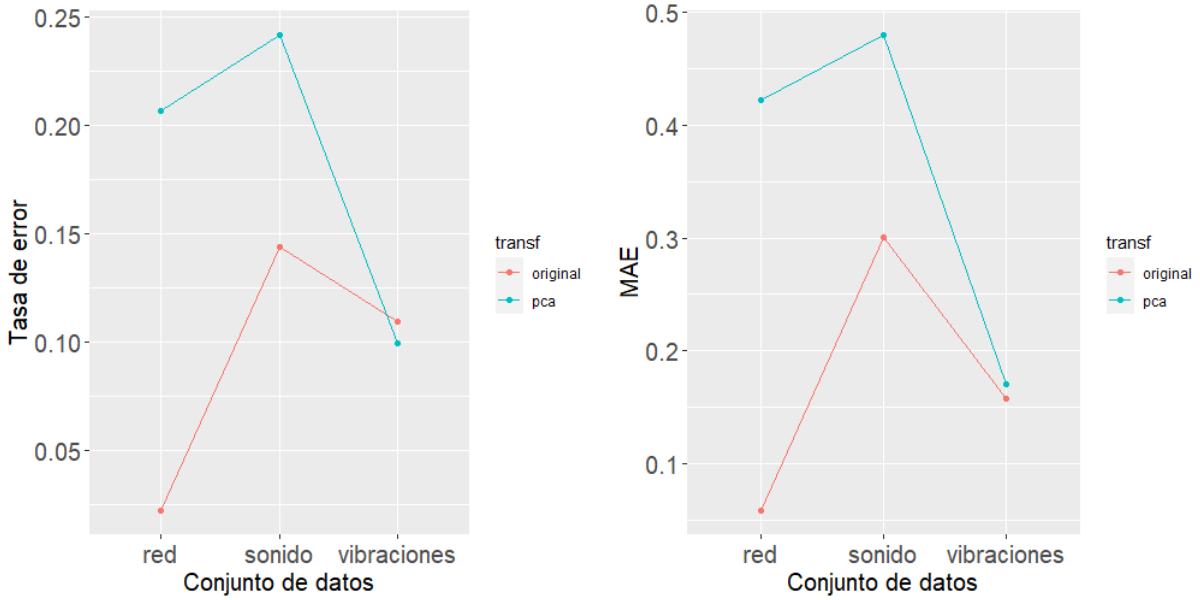


Figura 4.6.5: Gráfico de interacción para las medias de las métricas del error según Conjunto*Transformación. Tasa de error (izquierda) y MAE (derecha).

Dado un conjunto de datos, comprobamos si la utilización de las componentes principales en vez de las variables proporciona las los mejores resultados. Sin duda, sin transformar las variables, el conjunto de Red predice casi a la perfección el estado del motor mientras que al tomar las componentes principales su rendimiento empeora considerablemente. Este comportamiento es similar también para el conjunto de datos de sonido, siendo éste el que peor clasifica de los tres. Finalmente, observamos que el conjunto de vibraciones es el único que mejora ligeramente su tasa de error (pero no el MAE) al sustituir las variables a sus componentes principales.

4.6.6. Interacción Conjunto:Nivel de carga

Otra interacción que resulta significativa es la de los factores Conjunto de datos y el Nivel de carga de los motores, visualizada en la Figura 4.6.6.

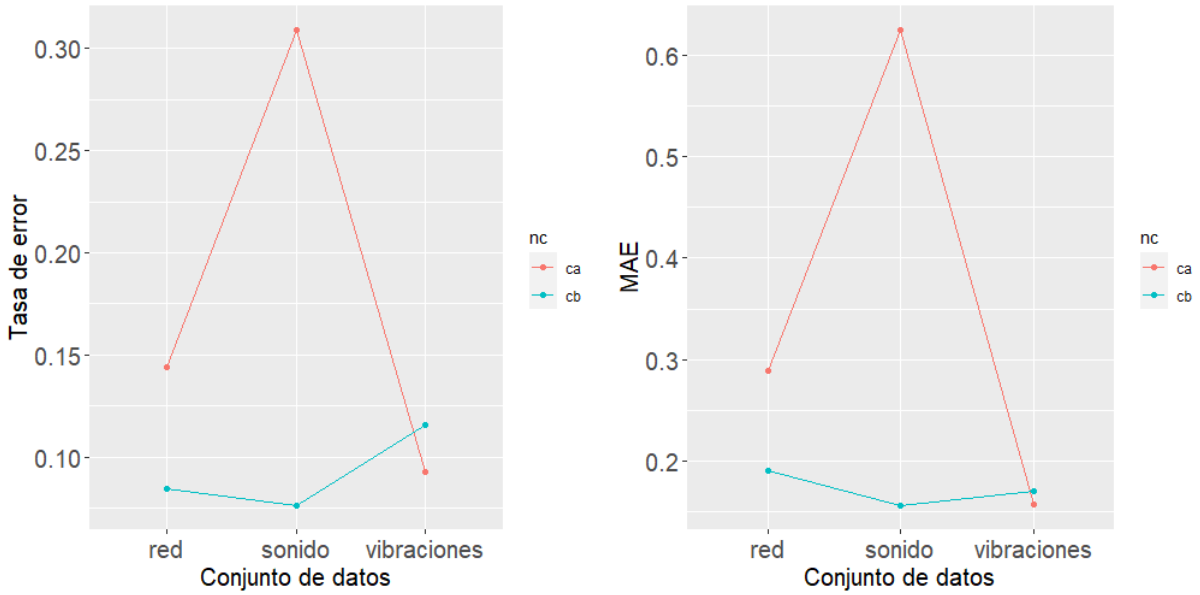


Figura 4.6.6: Gráfico de interacción para las medias de las métricas del error según Conjunto*Nivel de carga. Tasa de error (izquierda) y MAE (derecha).

En este caso, los gráficos para las dos métricas del error son esencialmente iguales. Para el conjunto de red, las diferencias entre los niveles de carga son relativamente bajas (en torno a un 0.05 de diferencia entre errores). En el caso del sonido, esta diferencia es bastante acusada y la diferencia media de errores entre niveles de carga es de casi 0.2. Finalmente, en el conjunto de vibraciones se observa la única ocurrencia en el que la clasificación por nivel de carga mejora, por poco, en carga alta.
Este resultado es interesante para el departamento de Ingeniería Eléctrica. Utilizar el conjunto de vibraciones para detectar fallos en carga alta significa que, si se utiliza este método, no es necesario reducir la carga del motor para obtener los mejores resultados. En cambio, con otros métodos el hecho de reducir la carga del motor resulta beneficioso a la hora de detectar problemas, especialmente con el conjunto de sonido.

4.6.7. Interacción Transformación:Nivel de carga

Como se observa en los Cuadros 4.6.3 y 4.6.4, la interacción no es significativa para el MAE pero sí para la tasa de error. Los resultados gráficos aparecen en la Figura 4.6.7.

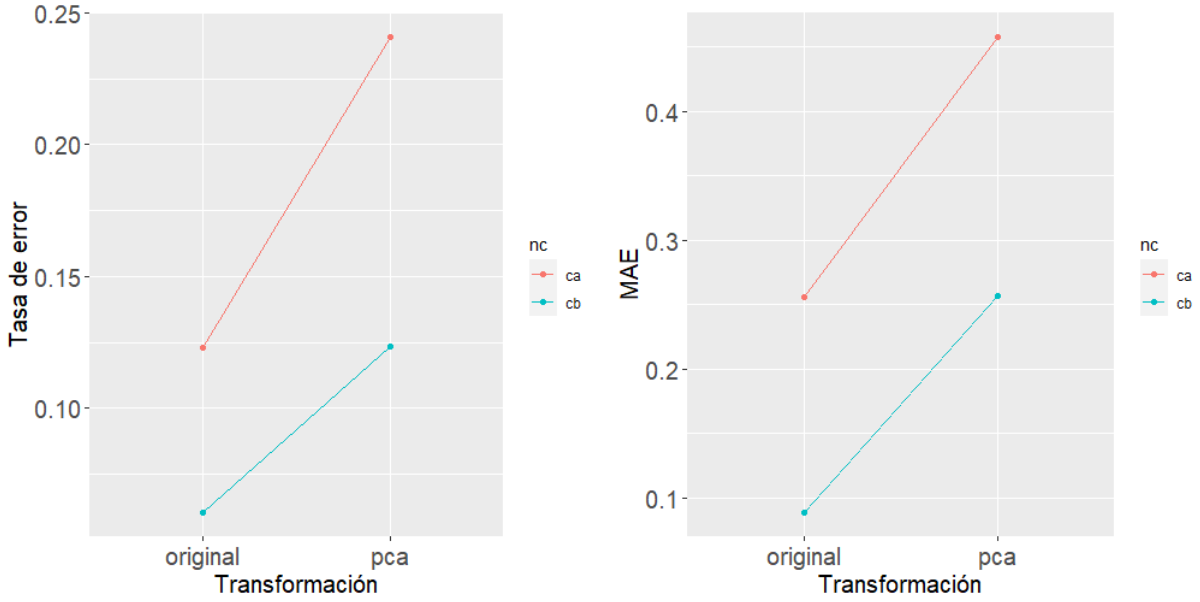


Figura 4.6.7: Gráfico de interacción para las medias de las métricas del error según Transformación*Nivel de carga. Tasa de error (izquierda) y MAE (derecha).

Dada una transformación de los datos, la clasificación es mejor en los conjuntos con variables originales y con nivel de carga bajo. La diferencia entre niveles de carga es menor para las tasas de error del conjunto sin transformar que para el MAE. Destacamos que en ambas métricas, es casi igual utilizar el conjunto de carga alta sin transformar y el de carga baja transformado mediante ACP.

4.6.8. Interacción Transformación:Algoritmo

Finalmente, observamos la última interacción significativa que relaciona los factores Transformación del conjunto con el Algoritmo. Los resultados se muestran en la Figura 4.6.8.

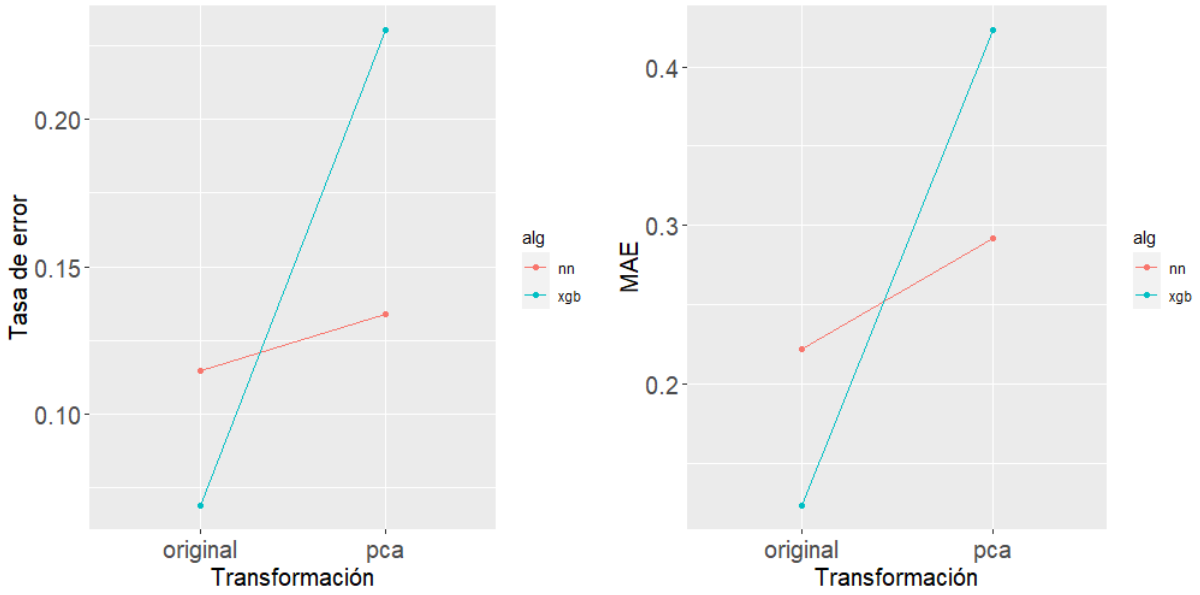


Figura 4.6.8: Gráfico de interacción para las medias de las métricas del error según Transformación*Algoritmo. Tasa de error (izquierda) y MAE (derecha).

Claramente el comportamiento de XGBoost es muy superior al de las redes neuronales cuando no se transforman los datos. No obstante, cuando se utilizan las componentes principales las métricas del boosting empeoran bastante. Debido a que en general las componentes principales presentan resultados peores que usar las variables originales, se añade un punto a favor de XGBoost.

4.7. Selección del conjunto de datos

Con los resultados obtenidos en los ANOVA anteriores, podemos filtrar los conjuntos de datos en función de los factores para reducir la dimensión del problema.

- Tomaremos todos los Conjuntos de datos, ya que uno de los objetivos del trabajo es comprobar si combinarlos genera mejores resultados.
- Tomaremos el Nivel de carga alto. Se ha demostrado que la carga baja proporciona mejores predicciones, pero en la práctica los motores funcionan siempre en carga alta.
- Sobre el factor Transformación, se decide no transformar los datos ya que las componentes principales generan resultados peores en todos los casos.
- A la hora de elegir el Algoritmo es importante tener en cuenta sus interacciones. Como hemos descartado las componentes principales, los resultados de XGBoost son mejores

que la red neuronal. Además, es un modelo más sencillo tanto de construir como de entrenar (en torno a cuatro veces más rápido y con un menor consumo de recursos), se comporta mejor a la hora de buscar hiperparámetros y en general es más estable. Por tanto, utilizaremos boosting como algoritmo final.

4.8. Combinación del factor Conjunto

A continuación se presenta un análisis con boosting y Nivel de carga alto sobre los conjuntos de datos de Red, Sonido y Vibraciones con las variables originales. Destacamos que en el conjunto de Red se utilizan los estadísticos con todas las fases de la onda y en Vibraciones, todos los ejes. A continuación, en el Cuadro 4.8.1 se muestran las tasas de error y los MAE del experimento explicado anteriormente (validación cruzada de 10 particiones repetida 20 veces).

Métrica	**Conjunto**	**Media**	**D. est.**	**Mín.**	**Q1**	**Mediana**	**Q3**	**Máx.**
Error	Red	0.021	0.037	0	0	0	0.021	0.083
	Sonido	0.167	0.069	0	0.167	0.167	0.167	0.333
	Vibración	0.083	0.0759	0	0	0.083	0.083	0.25
	Red+Sonido	0.021	0.046	0	0	0	0	0.167
	Red+Vibración	0.033	0.063	0	0	0	0.083	0.25
	Sonido+Vibración	0.021	0.046	0	0	0	0	0.167
	Todos	0.012	0.031	0	0	0	0	0.083
MAE	Red	0.067	0.119	0	0	0	0.063	0.333
	Sonido	0.371	0.188	0	0.25	0.333	0.5	0.833
	Vibración	0.133	0.128	0	0	0.133	0.25	0.417
	Red+Sonido	0.054	0.112	0	0	0	0	0.333
	Red+Vibración	0.079	0.125	0	0	0	0.25	0.333
	Sonido+Vibración	0.054	0.13	0	0	0	0	0.5
	Todos	0.029	0.078	0	0	0	0	0.25

Cuadro 4.8.1: Tasa de error y MAE para las combinaciones de Conjuntos.

En términos generales, todos los conjuntos dan buenos resultados. Destaca el conjunto de sonido por ser el que peores resultados da. Como era de esperar, la combinación de todos los conjuntos da las mejores predicciones. Sorprendentemente, el set de Red es el siguiente con tasas de error bastante bajas y estables.

A continuación realizamos un ANOVA sobre las tasas de error y los MAE para comprobar si hay diferencias significativas entre los clasificadores de los conjuntos. El resultado aparece en los Cuadros 4.8.2 y 4.8.3.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
conjunto	6	0.4031	0.0672	23.08	0
Residuals	133	0.3872	0.0029		

Cuadro 4.8.2: ANOVA sobre la tasa de error de los Conjuntos.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
conjunto	6	1.743	0.2905	17.74	0
Residuals	133	2.177	0.0164		

Cuadro 4.8.3: ANOVA sobre el MAE de los Conjuntos.

Como se han detectado diferencias significativas, se realiza un test de Duncan para comprobar cómo se agrupan los conjuntos. Los resultados están en el Cuadro 4.8.4 donde se agrupa el test para la tasa de error y el MAE.

	Error	Groups	MAE	Groups
red	0.021	c	0.066	bc
sonido	0.171	a	0.371	a
vibraciones	0.075	b	0.133	b
red+sonido	0.021	c	0.054	bc
red+vib	0.033	c	0.079	bc
sonido+vib	0.021	c	0.054	bc
todos	0.004	c	0.013	c
Alpha:	0.05		Df Error:	133

Cuadro 4.8.4: Test de Duncan para las tasas de error y MAE de los Conjuntos.

Con respecto a las tasas de error, las diferencias entre el conjunto de sonido y vibraciones son significativas, mientras que el resto de conjuntos se pueden considerar equivalentes. Si nos fijamos en el MAE, las vibraciones se incluyen en el grupo de conjuntos combinados a excepción del factor *todos*.

Para visualizar las diferencias, en la Figura 4.8.1 se muestran las métricas del error en forma de diagramas de cajas. Ambas son similares excepto para el sonido, que se distribuyen en un rango mayor. El conjunto de Red junto con las combinaciones del resto presenta el mejor comportamiento y sería el más indicado ya que es el más simple.

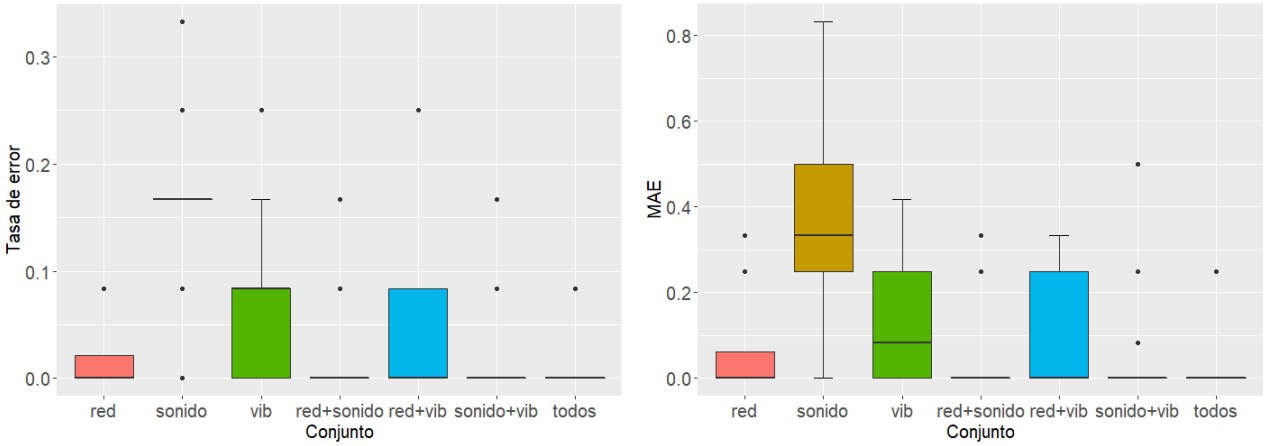


Figura 4.8.1: Boxplot para las métricas del error según combinaciones de Conjuntos. Tasa de error (izquierda) y MAE (derecha).

Finalmente, escogemos los conjuntos originales por separado y la combinación de todos a pesar de que no es significativamente distinto al de Red. Hacemos esto porque es interesante comparar la importancia de las variables a la hora de su interpretación en un conjunto más grande que las contenga todas y permita evaluarlas al mismo nivel.

Capítulo 5

Resultados

En esta sección se presentan los resultados finales del modelo para cada conjunto de datos, además de un análisis de las variables con gráficos de importancia e interpretabilidad mediante SHAP.
Presentamos también las matrices de confusión, que contienen los valores reales de cada estado del motor junto con la predicción del modelo. En ellas para cada ejecución del experimento, se utilizan 48 instancias para entrenar con validación cruzada y 12 como test, estratificando por clases. Se repite 20 veces y se promedian los resultados. Finalmente se condicionan por filas para obtener una estimación de la probabilidad de clasificar un estado de la fila i como otro de la columna j. Destacamos el uso de dos gráficos que utilizaremos:

- *Feature importance plot* o gráfico de importancia de las variables. El ensemble de árboles construido con XGBoost proporciona el número de veces que aparece cada variable en sus particiones. Aquellas que aparecen más veces se consideran más importantes a la hora de clasificar.
- *Beeswarm plot* o gráfico de enjambre. Se representan los valores SHAP de cada observación para un estado (clasificación) de los motores en el eje X con las variables en el eje Y. Se incluye una tercera dimensión con los colores rojo y azul. El color rojo indica un valor alto en la variable correspondiente y el azul, uno bajo. Valores SHAP altos favorecen la clasificación al estado representado, mientras que un valor bajo favorece la clasificación a cualquiera de los otros. Las variables se ordenan de forma descendente según su valor SHAP.

5.1. Red

Los resultados son bastante buenos a pesar del pequeño tamaño del conjunto de entrenamiento. En el Cuadro 5.1.1 se observa que los principales errores del modelo se encuentran en la clasificación de los estados intermedios.

		Predicho					
		E1	**E2**	**E3**	**E4**	**E5**	**E6**
Real	**E1**	1	0	0	0	0	0
	E2	0	0.975	0	0	0.025	0
	E3	0	0	0.95	0	0	0.05
	E4	0.025	0	0	0.975	0	0
	E5	0	0	0	0	1	0
	E6	0	0.025	0	0	0	0.975

Cuadro 5.1.1: Matriz de confusión condicionada por filas del conjunto Red.

La nomenclatura de las variables para este clasificador es `{conjunto}_{variable}_{fase}`, donde conjunto es siempre Red, variable se refiere a los estadísticos de orden superior y fase a la fase de la onda. Se presenta una descripción más detallada en la sección 3.1.1.
Continuando con el estudio de las variables, el modelo utiliza `m1_f2` (promedio de la fase 2) como variable principal para clasificar, seguido del factor de forma de la fase 1 o `sf_f1` como se observa en la Figura 5.1.1.

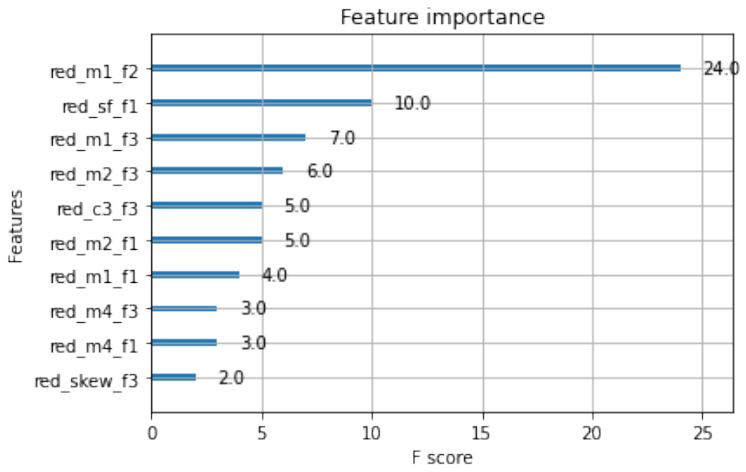


Figura 5.1.1: Gráfico de importancia para el conjunto Red para las 10 variables más influyentes.

El gráfico mostrado en la Figura 5.1.2 muestra que SHAP coincide generalmente con las particiones que realiza el modelo. Ofrece una granularidad mayor, ya que podemos observar qué

estados se clasifican con cada variable. El promedio de la fase 2 ayuda a clasificar los estados 1, 4 y 5. A pesar de su importancia, el factor de forma de la fase 1 solo influye en el estado 3. Destaca por ser el único estado que se clasifica utilizando solamente una variable, lo que puede llevarnos a pensar que esa es una de las causas de que el estado 3 sea aquel con más tasa de error.

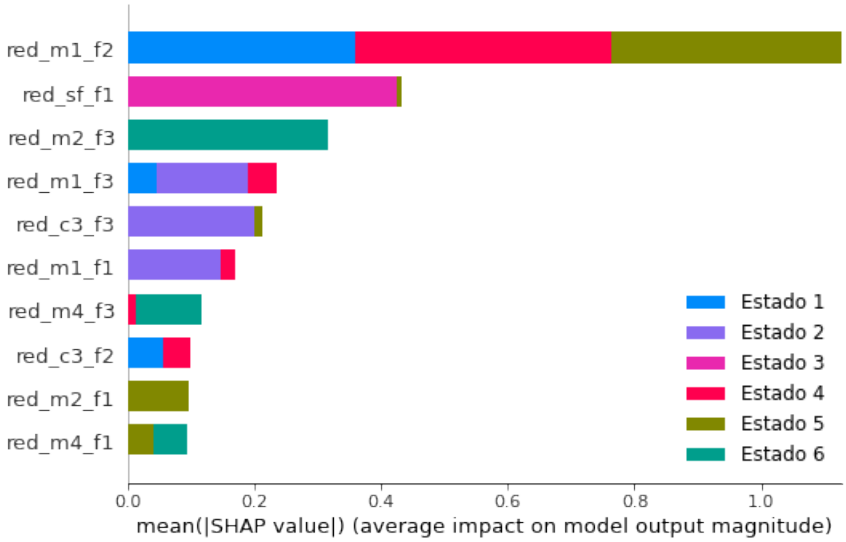
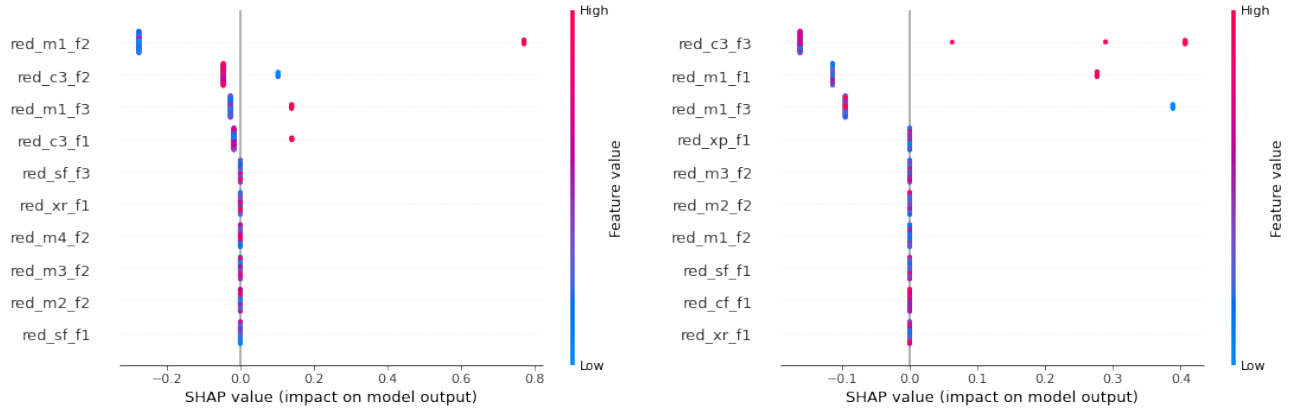


Figura 5.1.2: Gráfico resumen de SHAP sobre el conjunto Red para las 10 variables más influyentes.

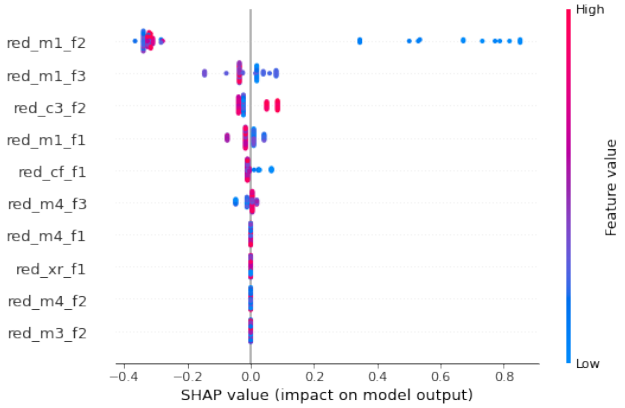
A continuación analizamos la clasificación de cada estado con los gráficos de enjambre de la Figura 5.1.3.



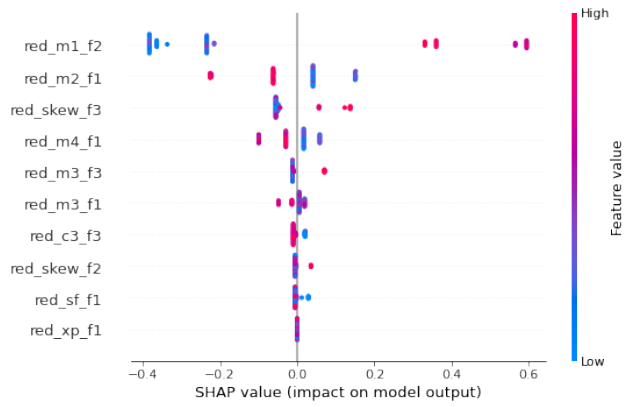
Estado 1

Estado 2

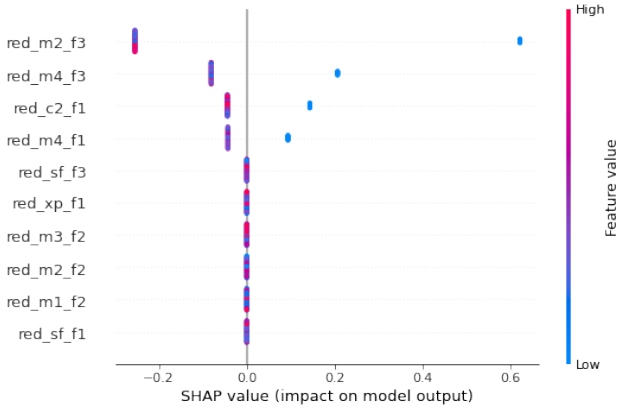
Estado 3



Estado 4



Estado 5



Estado 6

Figura 5.1.3: *Beeswarm plots* para la clasificación del conjunto Red.

- Estado 1. Valores altos del promedio de la fase 2 contribuyen a clasificar en este estado. Ocurre lo mismo con el promedio de la fase 3 y el cumulante 3 de la fase 1. Destaca el cumulante 3 de la fase 2, ya que medidas altas contribuyen a etiquetar la observación

como otro estado.

- Estado 2. Valores altos de `c3_f3` y `m1_f1` motivan la clasificación a este estado. En el promedio de la fase 3 observamos que la distribución de valores no es clara.
- Estado 3. Medidas elevadas de los factores de forma de las fases 1 y 2 clasifican las observaciones en este estado. Esto refuerza nuestra teoría sobre la mala clasificación al usar solo una variable.
- Estado 4. El promedio de la fase 2 como se vio anteriormente es influyente en la clasificación del estado 4. Valores bajos clasifican a este estado. Destacamos aún más las medidas altas, que contribuyen a clasificar hacia otro estado.
- Estado 5. El momento 1 de la fase 2 influye claramente en el etiquetado de los motores, donde valores altos se marcan como estado 5. Ocurre lo mismo con el apuntamiento de la fase 3. La varianza (`m2`) y el momento 4 de la fase 1 clasifican a este estado para valores bajos.
- Estado 6. Finalmente comprobamos que medidas bajas en la varianza y el momento 4 de la fase 3 influyen en la clasificación en el estado 6. No obstante, la distribución de valores no se encuentra demasiado separada en comparación con los otros gráficos, por lo que es razonable pensar que la clasificación se realiza por «descarte» y de ahí se confunde con el estado 3 (ver Cuadro 5.1.1).

5.2. Sonido

Las matrices de confusión para el ensemble construido con el conjunto de Sonido se presentan en el Cuadro 5.2.1. Es el set de datos que peores resultados obtiene, especialmente al utilizarse el nivel de carga alto. Comprobamos que de nuevo, se confunden los estados intermedios y aumenta el error de clasificación para los estados más deteriorados, especialmente el 6. Debido a que el conjunto de carga baja daba tasas de error muy bajas, podemos pensar que se debe a que los micrófonos empleados no son lo suficientemente precisos cuando el motor genera un ruido más potente en el nivel de carga alto.

		Predicho					
		E1	**E2**	**E3**	**E4**	**E5**	**E6**
Real	**E1**	1	0	0	0	0	0
	E2	0	0.9	0	0	0	0.1
	E3	0	0	0.75	0	0.175	0.075
	E4	0	0.15	0	0.725	0	0.125
	E5	0	0.15	0	0	0.85	0
	E6	0.05	0	0.15	0.075	0.05	0.625

Cuadro 5.2.1: Matriz de confusión condicionada por filas del conjunto Sonido.

La nomenclatura de las variables en este experimento es `{conjunto}_{variable}`, donde conjunto es siempre Sonido y variable se refiere a los estadísticos de orden superior.
A continuación, se muestra la importancia de las variables en la Figura 5.2.1. La media y la varianza destacan por aparecer en el mayor número de ensembles. En segundo lugar aparecen el factor de forma, el apuntamiento y la kurtosis.

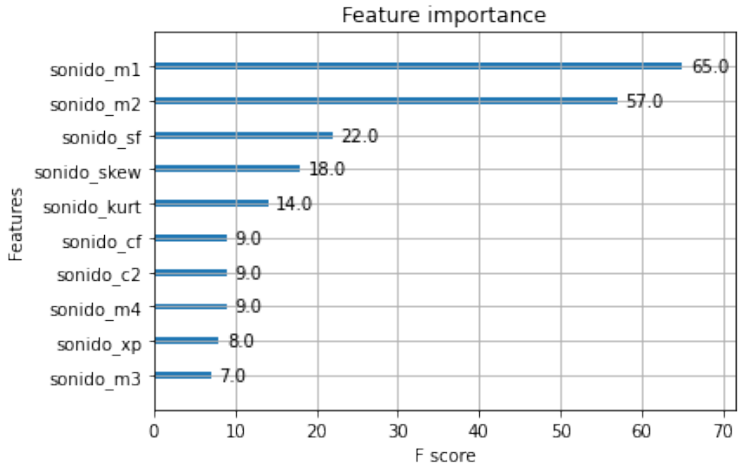


Figura 5.2.1: Gráfico de importancia para el conjunto Sonido para las 10 variables más influyentes.

Si comparamos los resultados anteriores con los obtenidos con SHAP, veremos que son bastante parecidos según la Figura 5.2.2. La media, la varianza y el factor de forma acumulan gran parte de la importancia. Además, con estas tres variables se clasifican todos los estados.

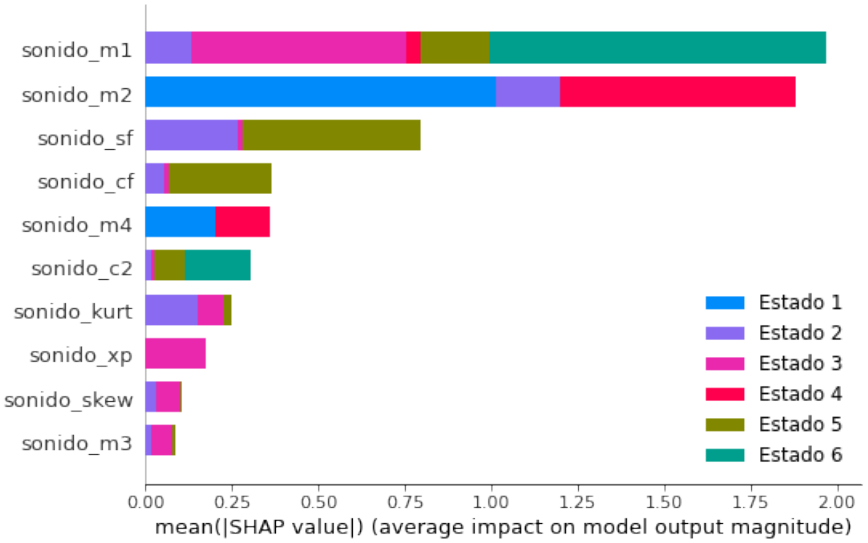
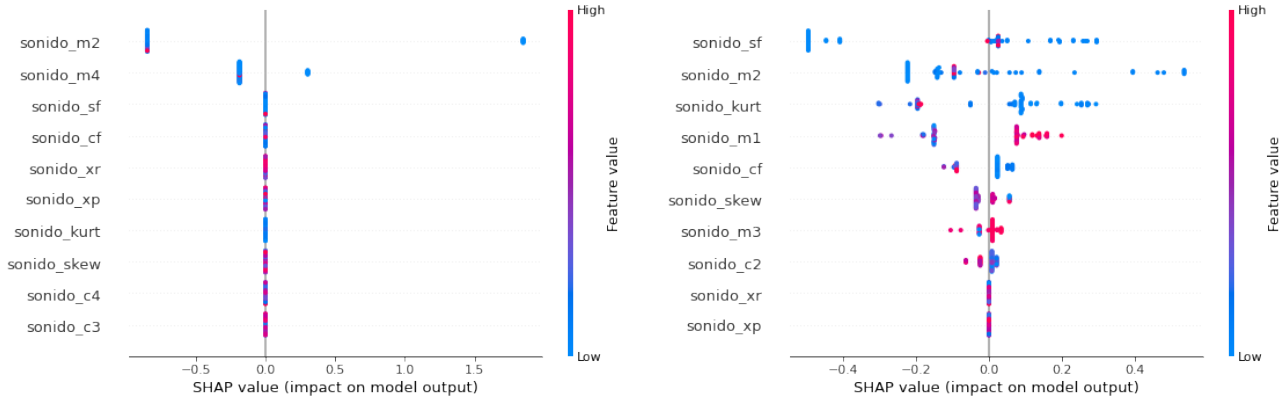


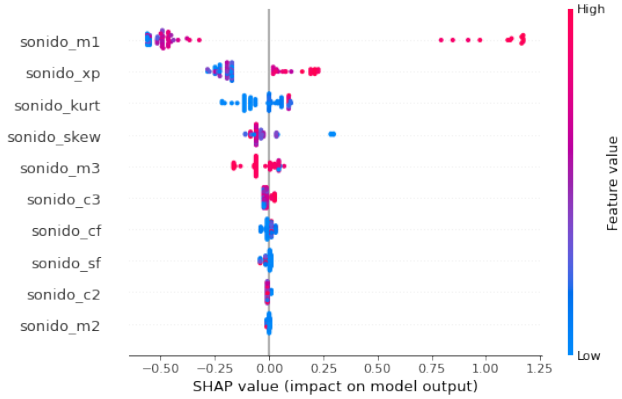
Figura 5.2.2: Gráfico resumen de SHAP sobre el conjunto Sonido para las 10 variables más influyentes.

Con la información anterior podemos pensar que el modelo genera buenos resultados. No obstante, con las visualizaciones de la Figura 5.2.3 podemos comprobar dónde falla el modelo.

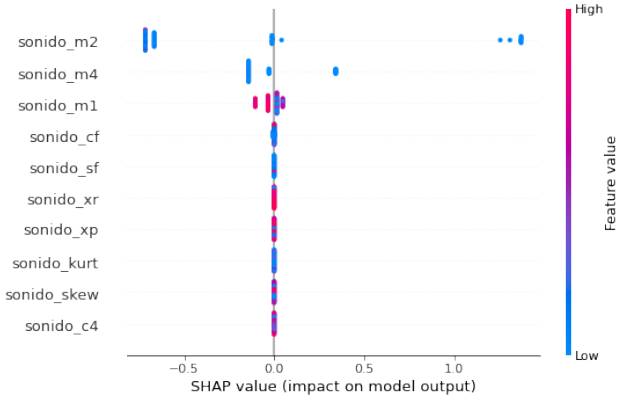


Estado 1

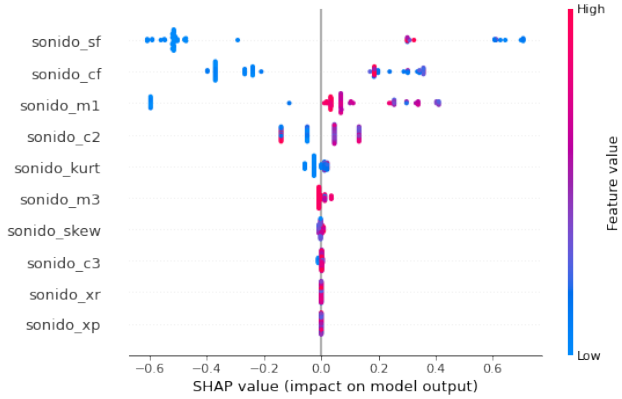
Estado 2



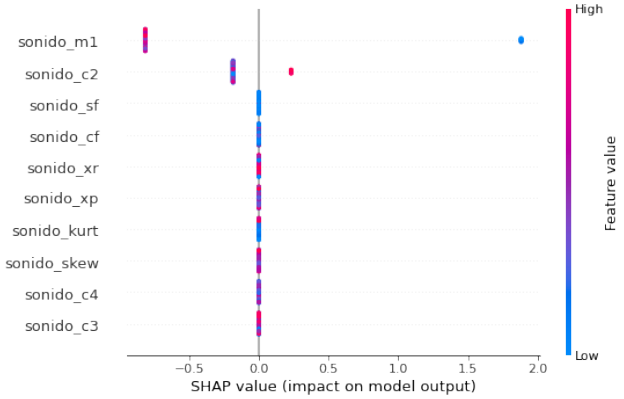
Estado 3



Estado 4



Estado 5



Estado 6

Figura 5.2.3: *Beeswarm plots* para la clasificación del conjunto Sonido.

- Estado 1. La distribución de valores no parece ser muy diferenciada. Valores bajos de la varianza y el momento 4 contribuyen a clasificar al estado 1. Parece funcionar ya que las matrices de confusión no muestran errores para E1.

- Estado 2. De forma similar al estado anterior, medidas bajas para el factor de forma, la varianza y la curtosis clasifican hacia el estado 2. Destaca el promedio, cuya distribución de puntos está bien dividida y valores altos clasifican a este estado.
- Estado 3. Valores altos del promedio y el máximo valor absoluto influyen para clasificar una observación en este estado.
- Estado 4. Las distribuciones del gráfico están poco separadas. Destaca el promedio, para el que valores altos evitan que se clasifique en el estado 4.
- Estado 5. De nuevo, las distribuciones de medidas se diferencian bastante mal y comienza a justificarse la mala clasificación de los estados más deteriorados (5 y 6).
- Estado 6. Finalmente, para el estado 6 se toman el promedio y el cumulante 2 como variables más influyentes. Valores altos de la primera contribuyen a clasificar hacia otro estado, mientras que valores altos de la segunda clasifican como estado 6. Comprobamos que `m1` también influye en el estado 3, lo que explica las observaciones mal clasificadas en la matriz de confusión.

5.3. Vibración

Los resultados para el set de Vibración son bastante mejores que los de Sonido. De nuevo, parecen confundirse los estados intermedios, esta vez en menor medida. La matriz de confusión con los resultados se muestra en el Cuadro 5.3.1.

		Predicho					
		E1	**E2**	**E3**	**E4**	**E5**	**E6**
Real	**E1**	1	0	0	0	0	0
	E2	0	0.95	0	0	0.05	0
	E3	0	0	0.875	0.025	0	0.1
	E4	0	0	0.025	0.925	0	0.05
	E5	0	0.05	0	0.025	0.925	0
	E6	0	0	0.1	0	0.025	0.875

Cuadro 5.3.1: Matriz de confusión condicionada por filas del conjunto Vibración.

Las variables toman nombres de esta forma: `{conjunto}_{variable}_{eje}`, donde el conjunto es siempre vibración, variable es el estadístico y eje, el eje X, Y o Z.
Según los análisis realizados anteriormente, el eje Y demuestra ser el más importante a la hora

de clasificar. Específicamente al utilizar la varianza. No obstante, en este ensemble se utilizan casi todas las variables por igual según la Figura 5.3.1.

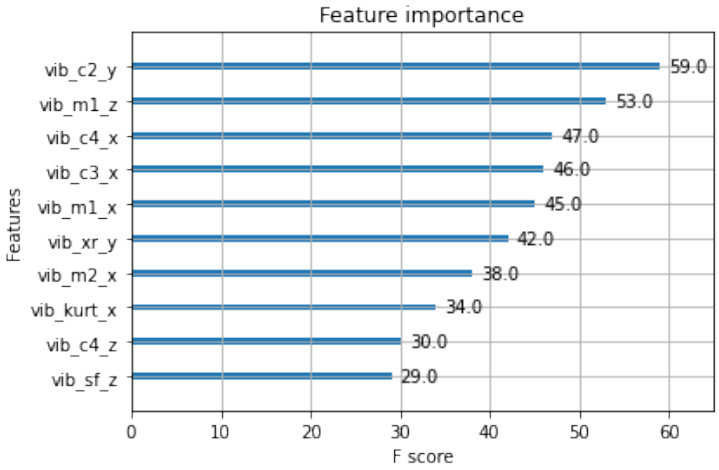


Figura 5.3.1: Gráfico de importancia para el conjunto Sonido para las 10 variables más influyentes.

En este caso, la importancia de las variables que obtiene XGBoost coincide con la que explica posteriormente SHAP como observamos en la Figura 5.3.2. Las variables parecen utilizarse para clasificar a un estado en concreto con pequeñas excepciones para la varianza en el eje y el valor cuadrático medio en el eje Y (`xr_y`).

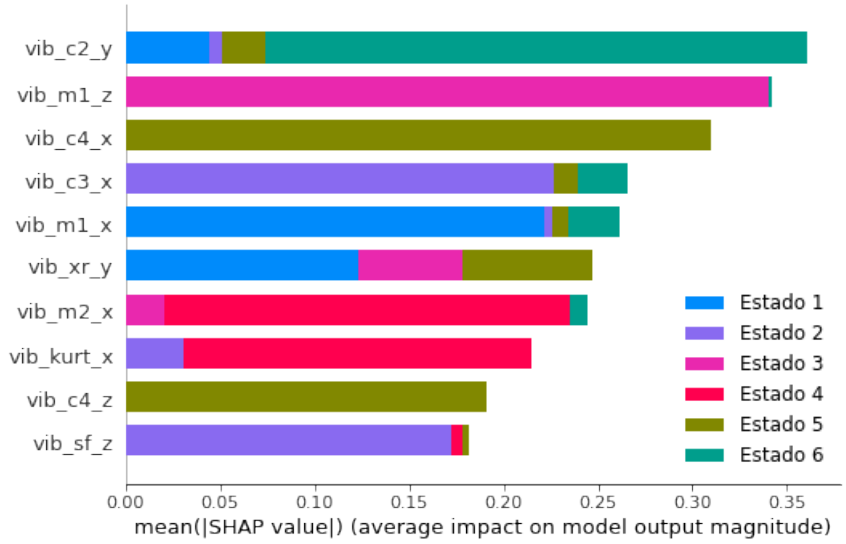
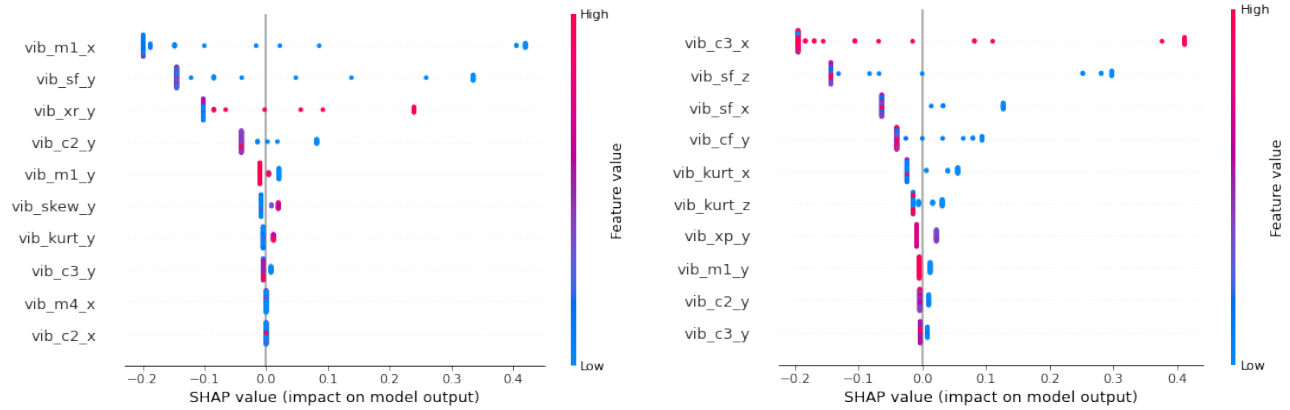


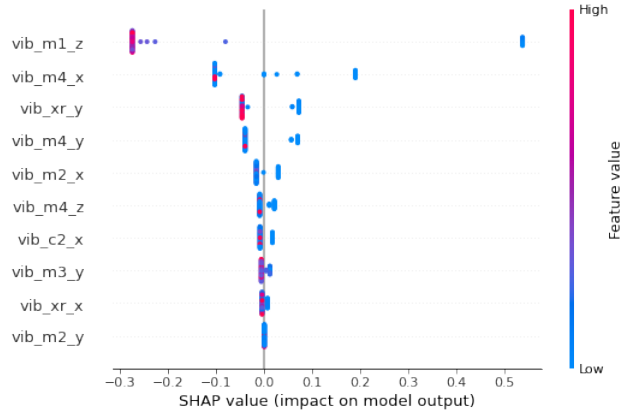
Figura 5.3.2: Gráfico resumen de SHAP sobre el conjunto Vibración para las 10 variables más influyentes.

Profundizando en los estados individuales, se muestran los gráficos de enjambre en la Figura 5.3.3.

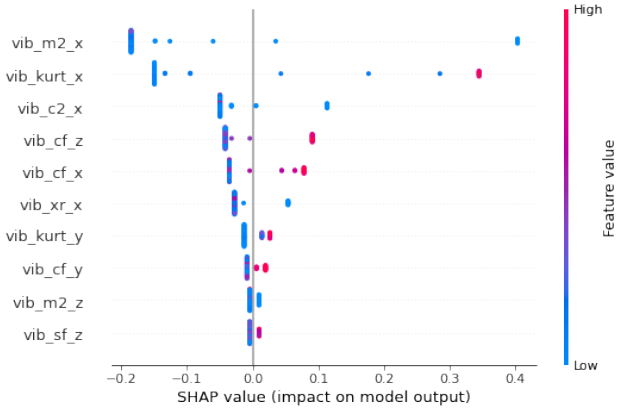


Estado 1

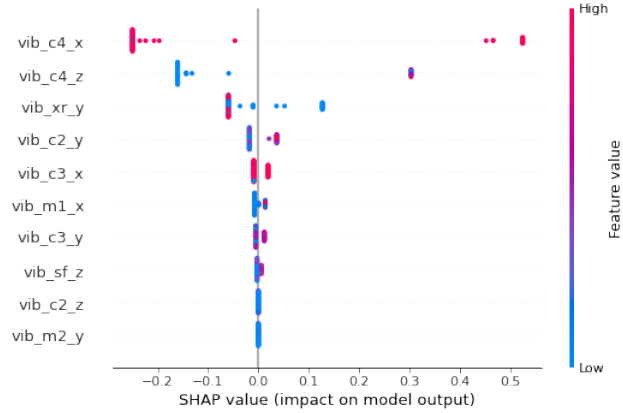
Estado 2



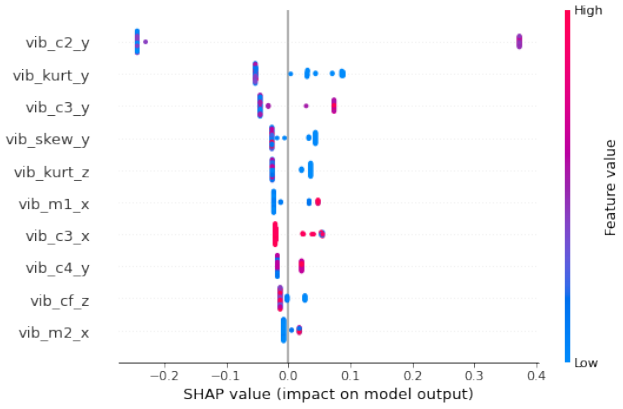
Estado 3



Estado 4



Estado 5



Estado 6

Figura 5.3.3: *Beeswarm plots* para la clasificación del conjunto Vibración.

- Estado 1. La distribución del valor cuadrático medio en el eje Y parece ser la más dife-

renciada. Valores altos clasifican al estado 1.

- Estado 2. En este caso, el cumulante 3 para el eje X es la variable más importante, donde valores elevados contribuyen a clasificar en este estado. Parece que valores altos del promedio del eje Y, a pesar de no tener mucha importancia según SHAP, permiten clasificar bastante bien hacia otros estados.
- Estado 3. Valores altos del promedio del eje Z evitan que se clasifique hacia el estado 3. Ocurre lo mismo con `xr` (valor cuadrático medio) en el eje Y.
- Estado 4. Las distribuciones de medidas comienzan a empeorar. Destaca la kurtosis del eje X, en la que valores altos contribuyen a clasificar al estado 4.
- Estado 5. De la misma forma que en el estado anterior, variables poco importantes como la varianza del eje Y son las que mejores distribuciones presentan, donde valores altos contribuyen a clasificar hacia el estado 5.
- Estado 6. Al igual que en los otros estados más deteriorados, la separación de colores no es demasiado intensa. Destaca la varianza para el eje Y, donde valores medios (color morado) contribuyen a clasificar al estado 6.

5.4. Combinación de los tres

Como último experimento, se decide unir todos los conjuntos para comprobar cómo influyen todas las variables sobre el modelo. A pesar de que no se encontraron diferencias significativas con el conjunto de Red (aunque los resultados son mejores) resultará interesante analizar las predicciones y las variables en conjunto.
La matriz de confusión es casi perfecta. Los errores que normalmente se presentan en los estados intermedios desaparecen. La combinación de variables parece mitigar el problema que presentaban los anteriores conjuntos. Se presentan los resultados en el Cuadro 5.4.1

		Predicho					
		E1	**E2**	**E3**	**E4**	**E5**	**E6**
Real	**E1**	1	0	0	0	0	0
	E2	0	1	0	0	0	0
	E3	0	0	1	0	0	0
	E4	0	0	0	1	0	0
	E5	0	0	0	0	1	0
	E6	0	0	0.025	0	0	0.975

Cuadro 5.4.1: Matriz de confusión condicionada por filas para todos los conjuntos.

La nomenclatura de variables en esta última parte es `{conjunto}_{variable}_{eje/fase}`, donde el conjunto puede ser cada uno de los anteriores (Red, Sonido, Vibración), variable es el estadístico, la fase se refiere al conjunto de Red y el eje se refiere al conjunto de Vibración.
En la Figura 5.4.1 se muestran las variables más importantes según las apariciones en el ensemble. Destacamos que la mayoría pertenecen al conjunto de Red y Vibraciones, siendo solo el promedio del Sonido la única que aparece en el top 10.

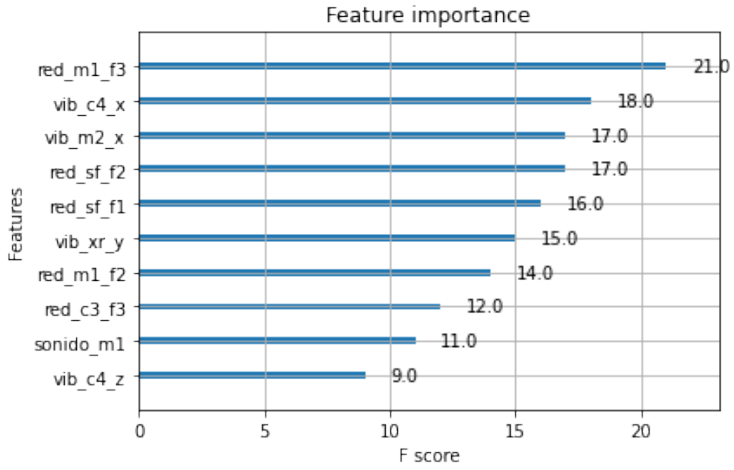


Figura 5.4.1: Gráfico de importancia para el todos los conjuntos para las 10 variables más influyentes.

La importancia estimada con SHAP se observa en la Figura 5.4.2, donde casi todas las variables se utilizan para clasificar solamente a uno o dos estados. Destacamos que en la gran mayoría de los casos, las variables de los conjuntos de Vibraciones y Sonido son importantes en los estados más deteriorados (4, 5 y 6) mientras que el conjunto de Red destaca a la hora de clasificar los estados menos deteriorados (1, 2 y 3).

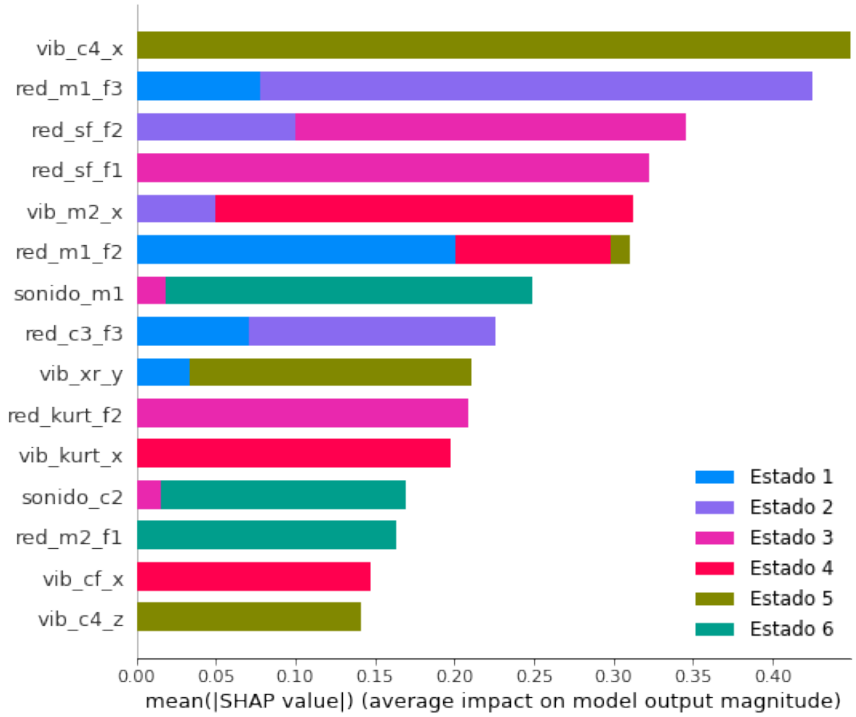
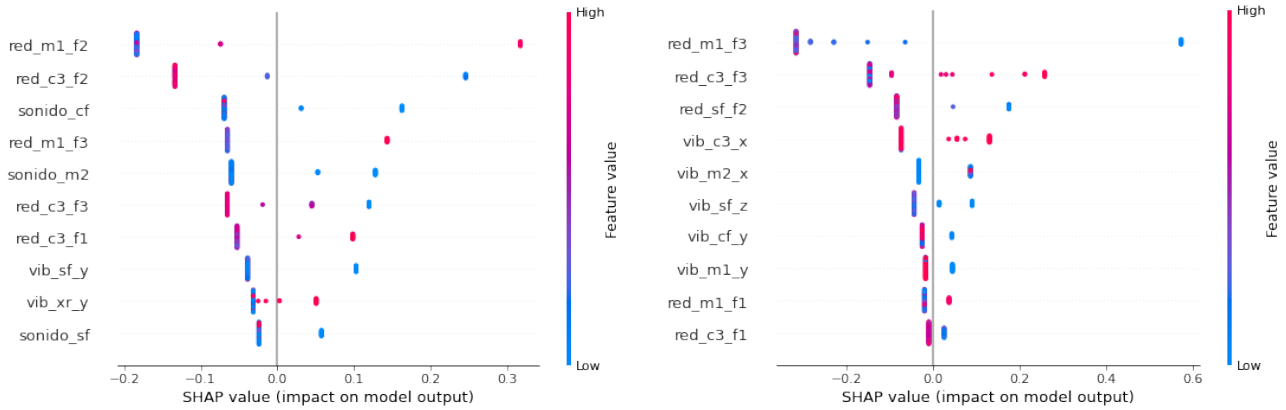


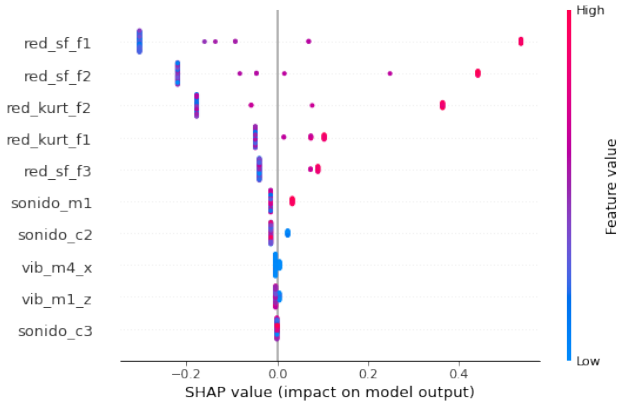
Figura 5.4.2: Gráfico resumen de SHAP sobre todos los conjuntos para las 15 variables más influyentes.

Finalmente, analizamos los estados de los motores individualmente en la Figura 5.4.3. Inicialmente se comprueba que se utilizan más variables para clasificar y las separaciones de sus valores es más clara.

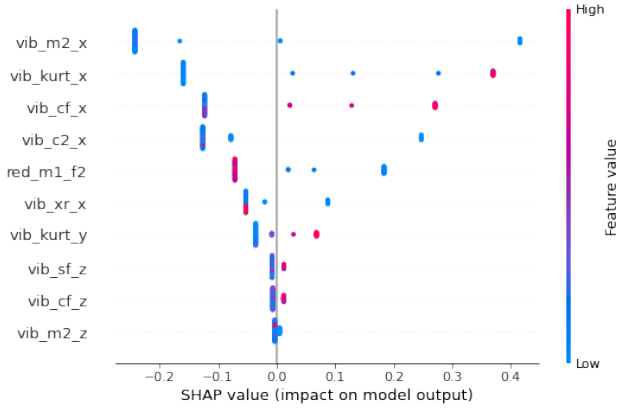


Estado 1

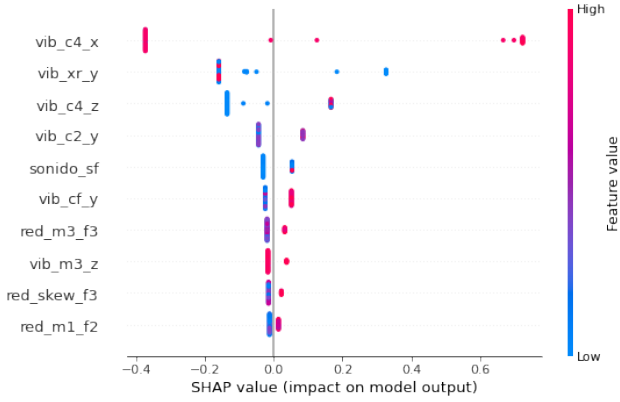
Estado 2



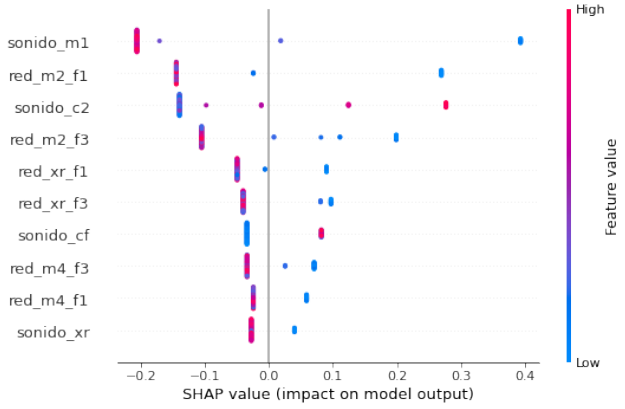
Estado 3



Estado 4



Estado 5



Estado 6

Figura 5.4.3: *Beeswarm plots* para la clasificación de todos los conjuntos.

- Estado 1. El conjunto de Red es útil para este estado, donde medidas altas del promedio de la fase 2 contribuyen a clasificar en el estado 1 y valores altos del cumulante 3 lo hacen hacia otros estados.

- Estado 2. De nuevo, el conjunto de Red aparece en las variables con más importancia. Destaca la media de la fase 3, donde medidas bajas clasifican las observaciones como estado 2. Valores altos del cumulante 3 de la fase 3 también lo hacen.
- Estado 3. Continúa la prevalencia del conjunto de Red sobre el resto. En este caso , el factor de forma de las fases 1 y 2 y la curtosis de las fases 1 y 2 clasifican hacia el estado 3 en sus valores altos.
- Estado 4. A medida que el motor comienza a deteriorarse, destaca el conjunto de Vibración en el eje X. La varianza tiene una separación de colores poco clara, pero valores altos de la curtosis y el factor de cresta tienden claramente a clasificar como estado 4.
- Estado 5. El conjunto de Vibraciones, especialmente el valor cuadrático medio del eje Y (valores bajos clasifican hacia el estado 5) y el cumulante 4 del eje Z (valores medios-altos clasifican hacia el estado 5) destacan sobre el resto.
- Estado 6. Finalmente, valores bajos del promedio en el conjunto de Sonido clasifican a los motores en el estado 6. Destaca el conjunto de Red con la varianza en la fase 1, donde medidas bajas clasifican hacia el estado 6. Ocurre al contrario con `sonido_c2`, para el cual valores altos clasifican en este estado.

Capítulo 6

Conclusiones y trabajo futuro

En este último apartado se exponen las conclusiones obtenidas a partir del análisis realizado en el proyecto. Además, se presentan posibles mejoras del trabajo y estudios futuros.

Conclusiones

- Como puede verse en la Sección 4.5, el conjunto de datos de Red basado en estadísticos de orden superior proporciona resultados mucho mejores que el basado en las frecuencias de las pistas de rodamiento. Reducir el número de variables resumiéndolas con estadísticos genera modelos más robustos con medidas del error mucho más bajas, permitiendo clasificar con precisión el estado de un motor de inducción.
- Según se describe en las Secciones 3.2.2 y 4.6, el análisis de componentes principales ha sido de gran utilidad a la hora de realizar una selección de variables. Ha simplificado enormemente el conjunto de red basado en frecuencias de las pistas de rodamiento manteniendo la precisión de clasificación y ha permitido identificar variables muy correlacionadas en el conjunto de estadísticos de orden superior. Sin embargo, a la hora de realizar la clasificación utilizando las propias componentes principales, los resultados son mucho peores que con los conjuntos originales, especialmente con boosting.
- El boosting ha probado ser un modelo más adecuado que las redes neuronales para esta tarea de clasificación. Debido a la escasa cantidad de observaciones, la red neuronal construida no debería tener demasiados parámetros ni ser demasiado compleja para evitar el sobreajuste. El modelo de boosting de árboles de clasificación contaba con una mayor optimización gracias a la búsqueda más exhaustiva de hiperparámetros. Esto es posible

debido a su simplicidad y a los tiempos de entrenamiento más bajos en comparación con las redes. Las métricas del error y el rendimiento de ambos algoritmos se presentan en la Sección 4.6.

- Según los resultados de la Sección 4.6.6, se ha demostrado que el nivel de carga del motor influye en el diagnóstico del estado de deterioro. Generalmente, el nivel de carga bajo proporciona una clasificación mejor. Sin embargo, a la hora de analizar la interacción del nivel de carga con los conjuntos de datos (Red, Sonido y Vibraciones) comprobamos que el conjunto de vibraciones es el único que mejora sus resultados con los motores en carga alta. Esto es de gran interés debido a que es el nivel de carga utilizado en la práctica.
- Se muestra en la Sección 4.8 que una vez se realiza una reducción del problema eligiendo el algoritmo, nivel de carga y transformación adecuados se llega a la conclusión de que los datos de red proporcionan una mejor clasificación que los de sonido y vibraciones. A la hora de agrupar los conjuntos para utilizar su combinación de variables para clasificar la mejora no es significativa, es decir, utilizar los datos de red no son significativamente diferentes de los obtenidos con las combinaciones de todos los conjuntos. Esta información es de utilidad puesto que la corriente es más sencilla de medir que el sonido o las vibraciones, ya que solo se necesita un cuadro eléctrico y una pinza amperimétrica, que son muy comunes en la industria. En cuanto a las vibraciones, se utiliza un acelerómetro que es más complejo y la medida depende de la distancia del acelerómetro al motor. Finalmente, el micrófono para medir el sonido puede verse afectado por otras fuentes de sonido que haya cerca.
- Como se observa en la Sección 5, la interpretabilidad de los modelos mediante SHAP ha sido de utilidad para reconocer las variables más importantes a la hora de clasificar cada estado de deterioro. Sin embargo, al entrar en un análisis más específico teniendo en cuenta los valores que toman las variables, algunos resultados son directamente no interpretables debido a la distribución desequilibrada de los valores de las variables.

Trabajo futuro y posibles mejoras

- Ampliar el estudio a partir de datos basados en variadores, que no han sido utilizados debido a su escasez.
- Ampliar el conjunto de datos estudiado con más observaciones para asentar las conclusiones obtenidas.

- La clasificación en 6 estados de deterioro es buena, pero en ocasiones puede ser más útil reducir el número de estados de deterioro combinando estados para que la clasificación mejore. Incluir más datos de motores en estado 1 (sin deteriorar) y combinar el resto de estados (2 al 6) sería interesante.
- Realizar una búsqueda de hiperparámetros exhaustiva en vez de aleatorizada para ambos modelos. Ampliar dicha búsqueda especialmente para el modelo de redes neuronales, utilizando una GPU más potente para obtener tiempos de entrenamiento razonables.
- Comparar los resultados con otros modelos más sencillos.
- Desarrollo de un artículo científico con los resultados del proyecto para una revista de impacto.

Bibliografía

[1] Wikipedia. «Electromagnetic induction». (2022), dirección: https://en.wikipedia.org/wiki/Electromagnetic_induction.

[2] P. A. Delgado-Arredondo, D. Morinigo-Sotelo, R. A. Osornio-Rios, J. G. Avina-Cervantes, H. Rostro-Gonzalez y R. d. J. Romero-Troncoso, «Methodology for fault detection in induction motors via sound and vibration signals», *Mechanical Systems and Signal Processing*, vol. 83, págs. 568-589, ene. de 2017. DOI: 10.1016/j.ymssp.2016.06.032.

[3] A. Barón, «Detección y Clasificación de Fallos en Motores mediante Procedimientos Boosting», *Trabajo Fin de Grado, Universidad de Valladolid*, 2020. dirección: http://uvadoc.uva.es/handle/10324/43777.

[4] M. Toquero, «Clasificación de fallos en motores en estado transitorio mediante redes neuronales», *Trabajo Fin de Grado, Universidad de Valladolid*, 2021. dirección: https://uvadoc.uva.es/handle/10324/50494.

[5] J. N. Morgan y J. A. Sonquist, «Problems in the Analysis of Survey Data, and a Proposal», *Journal of the American Statistical Association*, vol. 58, n.º 302, págs. 415-434, 1963.

[6] L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone, *Classification and Regression Trees*. Wadsworth y Brooks, 1984.

[7] J. R. Quinlan, «Induction of Decision Trees», *Mach. Learn.*, vol. 1, n.º 1, págs. 81-106, 1986.

[8] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[9] H. Deng, G. Runger y E. Tuv, «Bias of Importance Measures for Multi-valued Attributes and Solutions», *Lecture Notes in Computer Science*, vol. 6792, págs. 293-300, jun. de 2011.

[10] M. Kearns y L. Valiant, «Cryptographic Limitations on Learning Boolean Formulae and Finite Automata», 1989.

[11] Y. Freund y R. E. Schapire, «A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting», *Journal of Computer and System Sciences*, vol. 55, n.º 1, págs. 119-139, 1997.

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[13] L. Breiman, «Arcing the edge», 1997.

[14] J. Friedman, «Greedy Function Approximation: A Gradient Boosting Machine», *The Annals of Statistics*, vol. 29, nov. de 2000.

[15] L. Mason, J. Baxter, P. Bartlett y M. Frean, «Boosting Algorithms as Gradient Descent», vol. 12, 1999.

[16] T. Chen y C. Guestrin, «XGBoost: A Scalable Tree Boosting System», KDD '16, págs. 785-794, 2016.

[17] L. Breiman, «Random Forests», *Machine Learning*, vol. 45, n.º 1, págs. 5-32, 2001.

[18] J. H. Friedman, «Stochastic gradient boosting», *Computational Statistics & Data Analysis*, vol. 38, n.º 4, págs. 367-378, 2002.

[19] W. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *The bulletin of mathematical biophysics*, vol. 5, págs. 115-133, 1943.

[20] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain.», *Psychological Review*, vol. 6, n.º 65, págs. 386-408, 1958.

[21] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors», *Nature*, vol. 323, págs. 533-536, 1986.

[22] J. Zhang e I. Mitliagkas, «YellowFin and the Art of Momentum Tuning», 2018.

[23] M. T. Ribeiro, S. Singh y C. Guestrin, «"Why Should I Trust You?": Explaining the Predictions of Any Classifier», 2016.

[24] S. Lundberg y S.-I. Lee, «A Unified Approach to Interpreting Model Predictions», 2017.

[25] L. S. Shapley, *A Value for N-Person Games*. RAND Corporation, 1952.

[26] C. Molnar, *Interpretable Machine Learning, A Guide for Making Black Box Models Explainable*, 2.ª ed. 2022. dirección: https://christophm.github.io/interpretable-ml-book.

[27] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger y S. Friedler, «Problems with Shapley-value-based explanations as feature importance measures», 2020.

[28] A. Shrikumar, P. Greenside, A. Shcherbina y A. Kundaje, «Not Just a Black Box: Learning Important Features Through Propagating Activation Differences», 2016.

[29] GitHub. «XGBoost». (2021), dirección: https://github.com/dmlc/xgboost.

[30] ——, «PyTorch». (2022), dirección: https://github.com/pytorch/pytorch.

[31] ——, «Scikit-Learn». (2022), dirección: https://github.com/scikit-learn/scikit-learn.

Apéndice

Fragmentos de código utilizados para realizar el trabajo. Se muestran ejemplos concretos que permiten obtener todos los resultados modificando las entradas.

Carga de datos y análisis de componentes principales

Se ha realizado con las librerías `pandas` y `sklearn` en Python.
Obtener un DataFrame concatenando varios y tratando las observaciones incorrectas.

```
# Lectura y procesado inicial
dfs = {}
for i in range(3):
    dfs[i] = pd.read_excel('DatosRedCA.xlsx', sheet_name=i).replace(-200, np.nan)

df_red_ca = pd.concat([dfs[0], dfs[1], dfs[2]]).groupby(level=0).mean()
df_red_ca = df_red_ca.rename(columns={'Unnamed: 968': 'Estado'})
df_red_ca.head()
```

Obtener las primeras `n_comp` componentes principales con su *scree plot* y gráfico de varianza acumulada.

```
# PCA
def custom_pca(df, n_comp):
    scaler = StandardScaler()
    if 'Eje' in df.columns:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-2]),
            columns=df.columns[:-2])
    else:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]),
            columns=df.columns[:-1])

```

```
    pca = PCA(n_components=n_comp)
    pca_fit = pca.fit(scaled_df)

    pc_vals = np.arange(pca.n_components_) + 1
    plt.plot(pc_vals, pca.explained_variance_ratio_, 'o-', linewidth=2, color='blue')
    plt.title('PCA')
    plt.xlabel('Num componentes')
    plt.ylabel('Varianza explicada')
    plt.show()


    prop_varianza_acum = pca_fit.explained_variance_ratio_.cumsum()
    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(6, 4))
    ax.plot(
        np.arange(n_comp) + 1,
        prop_varianza_acum,
        marker = 'o'
    )

    for x, y in zip(np.arange(n_comp) + 1, prop_varianza_acum):
        label = round(y, 2)
        ax.annotate(
            label,
            (x,y),
            textcoords="offset points",
            xytext=(0,10),
            ha='center'
        )

    ax.set_ylim(0, 1.1)
    ax.set_xticks(np.arange(pca_fit.n_components_) + 1)
    ax.set_title('Pct. de varianza explicada acumulada')
    ax.set_xlabel('Componente principal')
    ax.set_ylabel('% varianza acumulada')
    plt.show()

    print(prop_varianza_acum)
    print(pca.explained_variance_)
    print(pca_fit.explained_variance_ratio_)
```

Obtener la matriz de cargas del análisis de las primeras `n_comp` componentes principales.

```python
# matriz de cargas PCA
def loading_matrix(df, n_comp):
    scaler = StandardScaler()
    if 'Eje' in df.columns:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-2]),
            columns=df.columns[:-2])
    else:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]),
            columns=df.columns[:-1])

    pca = PCA(n_components=n_comp)
    pca_fit = pca.fit(scaled_df)

    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)
    loading_matrix = pd.DataFrame(loadings[:,0:2], columns=['PC1', 'PC2'])
    return loading_matrix
```

Obtener un gráfico de individuos de las variables de las primeras 2 componentes principales dado un umbral `threshold` de correlación.

```python
# grafico de individuos PCA
def pca_indiv_map(df, threshold):
    scaler = StandardScaler()
    if 'Eje' in df.columns:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-2]),
            columns=df.columns[:-2])
    else:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]),
            columns=df.columns[:-1])

    pca = PCA(n_components=2)
    components = pca.fit_transform(scaled_df)

    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

    if 'Eje' in df.columns:
        fig = px.scatter(components, x=0, y=1,
            color=df['Estado'].astype('category'),
                symbol=df['Eje'].astype('category'))
    else:
        fig = px.scatter(components, x=0, y=1, color=df['Estado'].astype('category'))

```

```
    fig.update_layout(width=600, height=600, xaxis_title='PC1', yaxis_title='PC2',
        legend_title='Estado')
    fig.show()
```

Obtener el gráfico de individuos de las tres primeras componentes principales.

```
# grafico de individuos 3D PCA
def pca_3d(df):
    scaler = StandardScaler()
    if 'Eje' in df.columns:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-2]),
            columns=df.columns[:-2])
    else:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]),
            columns=df.columns[:-1])

    pca = PCA(n_components=3)
    components = pca.fit_transform(scaled_df)

    total_var = pca.explained_variance_ratio_.sum() * 100

    if 'Eje' in df.columns:
        fig = px.scatter_3d(
            components, x=0, y=1, z=2,
                color=df['Estado'].astype('category'), symbol=df['Eje'].astype('category'),
            title=f'Total Explained Variance: {total_var:.2f}%',
            labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'},
            width=600,
            height=600
        )
    else:
        fig = px.scatter_3d(
        components, x=0, y=1, z=2, color=df['Estado'].astype('category'),
        title=f'Total Explained Variance: {total_var:.2f}%',
        labels={'0': 'PC 1', '1': 'PC 2', '2': 'PC 3'},
        width=600,
        height=600
        )

    fig.update_layout(legend_title='Estado', width=900, height=900)
    fig.show()
```

Obtener el gráfico de cargas para las dos primeras componentes principales dado un umbral `threshold` de correlación entre variables.

```python
# grafico de cargas PCA
def loadings_map(df, threshold):
    scaler = StandardScaler()
    if 'Eje' in df.columns:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-2]),
            columns=df.columns[:-2])
    else:
        scaled_df = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]),
            columns=df.columns[:-1])

    pca = PCA(n_components=2)
    components = pca.fit_transform(scaled_df)

    loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

    fig = go.Figure(layout_yaxis_range=[-1.25,1.25], layout_xaxis_range=[-1.25,1.25])

    for i, feature in enumerate(scaled_df.columns):
        if np.abs(loadings[i, 0]) > threshold:
            fig.add_shape(
                type='line',
                x0=0, y0=0,
                x1=loadings[i, 0],
                y1=loadings[i, 1]
            )
            fig.add_annotation(
                x=loadings[i, 0],
                y=loadings[i, 1],
                ax=0, ay=0,
                xanchor="center",
                yanchor="bottom",
                text=feature,
            )

    fig.add_shape(type="circle",
        xref="x", yref="y",
        x0=-1, y0=-1, x1=1, y1=1,
        line_color="LightSeaGreen",
    )
    fig.update_layout(width=700, height=700, xaxis_title='PC1', yaxis_title='PC2')
```

```
    fig.show()
```

Clasificación utilizando XGBoost

Función de búsqueda de hiperparámetros con búsqueda aleatoria en cuadrícula para XGBoost con la librería `xgboost` en Python.

```
target = 'Estado'

# busqueda en cuadricula aleatorizada
def param_grid_search(df, factors, predictors, rep=20, cv_folds=5, ):
    X = df[predictors]
    y = df[target]
    print(X, y)

    alg = xgb.XGBClassifier()
    params = {'learning_rate': np.arange(0.01, 0.51, 0.1),
                'n_estimators': [1, 10, 50, 100, 1000],
                'min_child_weight': np.arange(1, 10, 2),
                'max_depth': np.arange(0, 30, 10),
                'gamma': [i/10.0 for i in range(0,5)],
                'subsample': [i/10.0 for i in range(6,10)],
                'colsample_bytree': [i/10.0 for i in range(6,10)],
                'reg_alpha': [0, 0.1, 0.5, 1, 2, 5],
                'reg_lambda': [0, 0.1, 0.5, 1, 2, 5],
                'objective': ['multi:softmax'],
                'num_class': [6],
                'use_label_encoder': [False]}
    error = []
    mae = []
    final_conf_matrix = np.zeros([6,6], dtype=int)
    for r in range(0, rep):
        print(f'Repeticion {r}')
        X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
            test_size=0.2, random_state=rep)
        clf = RandomizedSearchCV(alg, params, cv=cv_folds, n_iter=100, n_jobs=10,
            verbose=0, random_state=42)
        clf.fit(X_train, y_train, eval_set=[(X_train, y_train)],
            eval_metric='mlogloss',
            early_stopping_rounds=10,
```

```
            verbose=False)

        print(f'Mejores hiperparametros: ', clf.best_params_)
        y_pred = clf.predict(X_test)
        print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
        error.append(1-accuracy_score(y_test, y_pred))
        mae.append(mean_absolute_error(y_test, y_pred))
        conf_matrix = confusion_matrix(y_test, y_pred)
        final_conf_matrix = final_conf_matrix + conf_matrix

    df_out = pd.DataFrame({'error': error, 'mae': mae})
    df_out['tipo'] = factors[0] #
    df_out['nc'] = factors[1]
    df_out['conjunto'] = factors[2]
    df_out['alg'] = factors[3]
    clf.best_estimator_.save_model(f'model_name.json')
    print(final_conf_matrix)
    return df_out
```

Clasificación utilizando redes neuronales

Creación del conjunto de datos, arquitectura del modelo y bucle de entrenamiento de la red neuronal con la librería `PyTorch` en Python.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

target = 'Estado'

# Dataset y arquitectura
class ClassifierDataset(Dataset):

    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data

    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]

    def __len__ (self):
```

```
        return len(self.X_data)

class MulticlassClassification(nn.Module):
    def __init__(self, num_feature, num_class):
        super(MulticlassClassification, self).__init__()

        self.layer_1 = nn.Linear(num_feature, 512)
        self.layer_2 = nn.Linear(512, 512)
        self.layer_3 = nn.Linear(512, 128)
        self.layer_4 = nn.Linear(128, 64)
        self.layer_out = nn.Linear(64, num_class)

        self.relu1 = nn.ReLU()
        self.relu2 = nn.ReLU()
        self.relu3 = nn.ReLU()
        self.relu4 = nn.ReLU()

        self.dropout1 = nn.Dropout(p=0.2)
        self.dropout2 = nn.Dropout(p=0.2)
        self.dropout3 = nn.Dropout(p=0.2)

        self.batchnorm1 = nn.BatchNorm1d(512)
        self.batchnorm2 = nn.BatchNorm1d(512)
        self.batchnorm3 = nn.BatchNorm1d(128)
        self.batchnorm4 = nn.BatchNorm1d(64)

    def forward(self, x):
        x = self.layer_1(x)
        x = self.batchnorm1(x)
        x = self.relu1(x)

        x = self.layer_2(x)
        x = self.batchnorm2(x)
        x = self.relu2(x)
        x = self.dropout1(x)

        x = self.layer_3(x)
        x = self.batchnorm3(x)
        x = self.relu3(x)
        x = self.dropout2(x)

        x = self.layer_4(x)
```

```
        x = self.batchnorm4(x)
        x = self.relu4(x)
        x = self.dropout3(x)

        x = self.layer_out(x)

        return x

# bucle de entrenamiento
for e in range(0, n_epoch):
    train_epoch_loss = 0
    train_epoch_acc = 0
    model.train()
    for X_train_batch, y_train_batch in train_loader:
        X_train_batch, y_train_batch = X_train_batch.to(device),
            y_train_batch.to(device)
        optimizer.zero_grad()

        y_train_pred = model(X_train_batch)

        train_loss = criterion(y_train_pred, y_train_batch)
        train_acc = multi_acc(y_train_pred, y_train_batch)

        if e % 500 == 0:
            print(f'\tEpoch {e}: {train_loss.item()}')

        train_loss.backward()
        optimizer.step()

        train_epoch_loss += train_loss.item()
        train_epoch_acc += train_acc.item()

y_pred_list = []
with torch.no_grad():
    model.eval()
    for X_batch, _ in test_loader:
        X_batch = X_batch.to(device)
        y_test_pred = model(X_batch)
        _, y_pred_tags = torch.max(y_test_pred, dim = 1)
        y_pred_list.append(y_pred_tags.cpu().numpy())
y_pred_list = [a.squeeze().tolist() for a in y_pred_list]
```

Interpretabilidad con SHAP

Obtener gráficos de importancia a partir del ensemble de XGBoost y los valores SHAP cargando los datos y el modelo previamente entrenado en Python.

```
df = pd.read_csv('datos_procesados/red_ca.csv')
df['Estado'] = df['Estado'].astype('int')-1

model = XGBClassifier()
model.load_model('models/xgb_red_ca.json')

target = 'Estado'
predictors = [x for x in df.columns if x not in [target]]

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(df[predictors], df[target])

classes = [f'Estado {i}' for i in range(1,7)]
shap.summary_plot(shap_values, df[predictors], max_display=15, plot_type='bar',
    class_names=classes, class_inds=[0,1,2,3,4,5])
plt.show()

xgb.plot_importance(model.get_booster(), max_num_features=10)

shap.summary_plot(shap_values[0], df[predictors], max_display=10, plot_type='dot')
```

Análisis y gráficos

Obtener los resultados de los ANOVA y tests de Duncan así como los gráficos de cajas e interacción con R.

```
aov <- aov(lm(mae ~ tipo+nc+alg+conjunto+tipo:nc+tipo:alg+tipo:conjunto+
             nc:conjunto
         , data=df))
dunc <- duncan.test(y=aov, trt='conjunto', group=T, alpha=0.05, main=NULL, console=T)


ggplot(df, aes(x = nc, y = error, fill = nc)) +
    stat_boxplot(geom = "errorbar", width = 0.25) +
    geom_boxplot() +
```

```
    theme(legend.position = "none") +
    xlab('Nivel de carga') +
    ylab('Tasa de error') +
    theme(axis.text = element_text(size = 15),
          axis.title = element_text(size = 15))

ggplot(df, aes(x = conjunto, color = nc, group = nc, y = error)) +
    stat_summary(fun = mean, geom = "point") +
    stat_summary(fun = mean, geom = "line") +
    xlab('Conjunto de datos') +
    ylab('Tasa de error') +
    theme(axis.text = element_text(size = 15),
          axis.title = element_text(size = 15))
```