Un enfoque de Machine Learning para la detección de averías en motores c.c. basado en procesamiento de audio.

Trabajo de grado presentado como requisito para optar al titulo de Ingeniero Físico

Rommel Stiward Prieto Estacio

Director:
D.Sc. Diego Alberto Bravo Montenegro

Universidad del Cauca
Facultad de Ciencias Naturales, Exactas y de la Educación
Programa de Ingeniería Física
Popayán, 2023

Nota de aceptación

Director: D.Sc. Diego Alberto Bravo M.

Jurado: Ph.D. Carlos Felipe Rengifo R.

Jurado: Ph.D. Rubiel Vargas.

Popayán, 7 de Diciembre 2023.

Agradecimientos

En este espacio quiero agradecer desde el fondo de mi corazón a las personas que me acompañaron no solo durante la producción de esta investigación sino durante toda mi formación profesional.

Primero debo agradecer a Dios por todas las oportunidades que he recibido a lo largo de mi vida, igual que por las capacidades que me dio y las personas de las que me ha rodeado siempre; no hay un paso que haya dado en el camino hasta este momento que no haya sido guiado por ti, Señor, espero seguirte honrando en cada acontecimiento de mi vida.

A mi director de tesis, mi mentor y mi amigo el Dr. Diego Alberto Bravo Montenegro; no podría alegrarme más de que me haya tendido su mano y ofrecido su conocimiento y apoyo desde que me enseñó por primera vez hasta la última conclusión aquí escrita. Gracias por confiar en mis capacidades incluso cuando yo mismo no lo hacía, por impulsarme a hacer lo mejor de mí mismo y ayudarme a construir el ingeniero que seré a lo largo de mi vida profesional. En algún momento me dijo que sabía que haríamos grandes cosas juntos, y estoy seguro de que este es solo el principio. Nunca olvidaré todo lo que aprendí cubierto por su ala.

A mis papás, Rommel Prieto y Belis Estacio, que cada día de mi vida se han esforzado para darme las mejores oportunidades. Se llenan mis ojos de lágrimas al ser consciente de los sacrificios que han hecho por mí y mis hermanos y lo felices que somos cada que puedo volver a casa. En mi casa aprendí a ser fuerte y perseverar, y en cada momento de la carrera recibí todo el apoyo y el impulso de mis padres para dar lo mejor de mí mismo. Los amo infinitamente, y ustedes no están detrás de cada cosa que logre, están a mi lado; no hay nada que pueda alcanzar en la vida que no sea un reflejo del proyecto que ustedes hicieron de mí y de mis hermanos. Así mismo, agradezco a mi familia, a los descendientes de Magdalena y de Miriam por estar a mi lado en cada paso del proceso y darme valiosas lecciones de vida desde su amor y sabiduría.

Durante mi educación superior tuve la fortuna de hacer un buen número de amigos, a quienes agradezco su acompañamiento durante este proceso de trabajo de grado y por darme un lugar de comodidad cuando estaba lejos de mi hogar; debo hacer mención especial a Ester, Mateo y Sebastián, quienes estuvieron conmigo desde el principio hasta el final, días y noches enteras

viéndome quemar circuitos y programar aplicaciones. No entendí el significado de la amistad hasta que los encontré a ustedes, y les he aprendido tanto que no terminaré de agradecerles nunca; mi mamá me dijo alguna vez que las amistades de la Universidad eran para toda la vida, y con ustedes no quiero que sea diferente.

En mi formación encontré profesores que me indujeron conocimiento invaluable; les agradezco cada clase, cada asesoría y lección que me ofrecieron con la mejor disposición, cada vez que me corrigieron con la mejor intención y construyeron a este ingeniero físico. Finalmente, le hago la dedicatoria a mi programa y a mis compañeros: estudiar ingeniería física fue la mejor decisión de mi vida y me encontré con las personas correctas siempre, gracias por abrirme las puertas y por dejarme retribuirte en alguna medida, espero poner tu nombre en alto y convertirme en tu orgullo.

Gracias a cada persona que acompañó a este hijo de Buenaventura.

Misión cumplida.

Lista de Contenidos

Lista de Figuras

Lista de Tablas

Introducción General

Los motores eléctricos son parte del día a día de la humanidad desde hace más de un siglo, tanto que rara vez se piensa sobre su presencia en la vida cotidiana: locomotoras, taladros, automóviles, grandes fábricas; la construcción del mundo moderno se basa en los motores para el avance de la civilización. El fundamento, a grandes rasgos, se basa en una transformación de energía: de electromagnética a energía mecánica, sin embargo, es natural que existan distintas clasificaciones para los motores, siendo una aquella sobre su alimentación. Es entonces donde se encuentra al motor de corriente continua (c.c.), primer tipo de motor ampliamente utilizado y que a pesar de verse reemplazado en muchas instancias por motores de corriente alterna, sigue siendo preferido en la fabricación de vehículos eléctricos, elevadores, bombas y algunos tipos de locomotoras, entre otros.

No obstante, al igual que todo sistema los motores se ven sujetos a fallas después de un tiempo de operación. Esto puede deberse a entornos complejos de operación como alta temperatura, presencia de polvo, humedad y vibración, por lo que es necesario el establecimiento de herramientas para el diagnóstico y monitoreo de las condiciones de estos dispositivos.

Para Borgi [1], el mundo está atravesando la llamada "Cuarta Revolución Industrial", caracterizada por la colección de gran cantidad de datos que permiten simplificar la toma de decisiones; esto ha llevado a la construcción de un conjunto de metodologías que faciliten esta integración para la toma de decisiones. Dentro de la inteligencia artificial, el Machine Learning emerge como una poderosa herramienta para la producción de algoritmos predictivos. En una industria cada vez más demandante el Aprendizaje de Máquinas (ML) se alza como una herramienta cada vez más imprescindible. La industria, cada vez más *inteligente*, encuentra en el ML conocimiento computacional a partir de observaciones e interacciones con el mundo exterior y permite la construcción de sistemas que automáticamente mejoran con experiencia pues posibilitan la extracción de las relaciones fundamentales de un proceso a partir de clasificación, evaluación y optimización. De igual forma, Franciosi [2] describe impactos no despreciables, económicos, ambientales y sociales como las consecuencias de un

pobre manejo del mantenimiento que pueden afectar gravemente la sostenibilidad industrial.

En la reactivación económica dentro del marco de la postpandemia por COVID-19, y con las dificultades que está presentando la industria actual se ha encontrado la necesidad de disminuir todas las formas de costos posibles: económicos, de talento humano, de producción, o de tiempo. Todos estos son activos financieros de los cuales debe evitarse el desgaste al máximo. El mantenimiento es una parte fundamental en discutiblemente todas las empresas de la actualidad, pues es la herramienta con la que se garantiza la optimización de los procesos industriales, y de esta optimización no solo depende el éxito y la competitividad de la empresa, además podrían depender vidas y recursos de toda índole. Las estrategias de mantenimiento deben encontrar un balance entre la minimización de perjuicios a la producción y de costos, por lo que el mantenimiento predictivo es potencialmente el mecanismo más adecuado para este fin. Pero ¿cómo debe ser esa estrategia de mantenimiento predictivo para satisfacer las necesidades de hoy? Los procesos industriales son ricos en señales acústicas; el sonido de un motor, una turbina o cualquier sistema electromecánico se puede aprovechar ampliamente para determinar si un proceso industrial está funcionando adecuadamente. La limitada audición del ser humano puede ser, en situaciones igualmente limitadas, suficiente: un obrero con fina audición puede escuchar que hay algo en su planta que no está funcionando adecuadamente, pero es factible afirmar que cuando un humano puede escuchar este tipo de señales es consecuencia de una falla que pudo ser remediable con anterioridad, y por ende menos problemática para el adecuado funcionamiento de un proceso industrial. El estudio de las señales emitidas dentro de estos sistemas puede proporcionar información suficiente para encontrar dichas características que permitirían a un modelo de aprendizaje automático establecer *in situ*, de forma rápida y precisa si existe un fallo en alguna parte del sistema mediante la comparación con lo que se consideraría su funcionamiento típico.

Objetivos

Objetivo general

Obtener un modelo predictivo de clasificación que determine el estado de funcionamiento de un motor de corriente directa sin escobillas, BLDC a partir de señales de audio.

Objetivos específicos

- Generar experimentalmente un conjunto de datos entrada-salida de audio para la clasificación de motores BLDC.
- Entrenar un modelo predictivo de clasificación mediante distintas técnicas de aprendizaje automático.
- Diseñar un mínimo producto viable que permita la implementación del modelo predictivo.

Descripción metodológica

Esta investigación se comprende de 5 etapas principales, derivadas de la metodología *Cross-Industry Standard Process for Data Mining*, CRISP-DM por sus siglas en inglés; la primera etapa busca entender el sistema físico a utilizar: el motor de corriente directa sin escobillas, sus afecciones y la relación que guardan con el sonido. La segunda etapa consta del diseño de un experimento que permitiera obtener señales de audio, para posteriormente procesarlas y un conjunto de datos que pudiera entrenar modelos de clasificación, evaluarlos y finalmente realizar una acción de despliegue básica conocida como Mínimo Producto Viable y una solución IoT fundamentada en el marco de la Industria 4.0.

Impacto del trabajo

La solución propuesta tendría impactos en los sectores económicos, industriales, ambientales y productivos de la sociedad, pues el mantenimiento predictivo tiene como fin el alargar la vida útil de los dispositivos que hacen parte de la industria; es una estrategia que, una vez se implementa, es mucho más amigable ante todos estos sectores que, por ejemplo, el mantenimiento correctivo, cuyo fin ya se ha expuesto como esperar a que el dispositivo falle y entonces intentar repararlo o directamente reemplazarlo, lo cual ya implica un probable y considerable detrimento económico y perjuicio ambiental, además de una disminución operacional que afecta directamente la productividad de una planta. La estrategia propuesta busca, además de sentar una metodología de PdM para motores, permitir que se base en ser no invasiva, rápida y fiable que faculte la toma de decisiones industriales.

Organización del trabajo

La presente monografía está compuesta por cinco capítulos; en el primer capítulo se estudia una revisión sistemática de trabajos previos basados en la aplicación de técnicas de aprendizaje automático en procesos industriales, el mantenimiento predictivo de motores BLDC y otros tipos de motores, análisis de audio y las señales que pueden ser efectivas para este fin.

En el segundo capítulo se explica el principio funcionamiento de los motores BLDC, se plantea su modelo matemático y se realiza el proceso de caracterización para obtener la función de transferencia en estado pleno de funcionamiento del motor modelo a utilizar en el desarrollo experimental. Además, se detalla el experimento realizado para la obtención de los datos de entrada y salida y su preprocesamiento para la obtención de las características probadas y seleccionadas para alimentar los modelos predictivos a entrenar

En el tercer capítulo se introducen las técnicas utilizadas para el desarrollo del mínimo producto viable en forma de aplicación que permita la implementación de los modelos seleccionados en un entorno industrial. Se exponen las metodologías utilizadas, sus ventajas y desventajas y las características que les acompañan.

El capítulo cuarto presenta los resultados obtenidos y su discusión; se exponen las matrices de confusión y otras métricas de evaluación que, una vez comparadas con la naturaleza estadística de cada modelo permitirían establecer diferencias, ventajas y desventajas en cada etapa del desarrollo de la investigación.

El quinto capítulo entrega las conclusiones de este trabajo de grado en investigación; se resumen las metodologías, resultados y análisis obtenidos y se destacan sus principales contribuciones. Además, se plantean desarrollos futuros en la línea del procesamiento de señales, inteligencia artificial y automatización.

Lista de Publicaciones

- **Dataset of audio signals from Brushless DC motors for Predictive Maintenance**. *Data in Brief*. Volumen 50. October 2023. pág. 1-5. ISSN 2352-3409. `https://doi.org/10.1016/j.dib.2023.109569`.
- **Machine Learning and Audio Signal Processing for Predictive Maintenance: A review**. *IEEE 6th Colombian Conference on Automatic Control (CCAC 2023)*. 979-8-3503-2472-3/23/ ©2023 IEEE

Capítulo 1

Machine Hearing y Mantenimiento Predictivo de motores DC: una revisión

1.1. El aprendizaje automático

Álvarez, [3] describe el aprendizaje automático, o *Machine Learning* como un conjunto de técnicas utilizadas para enseñar a las máquinas a hacer lo que los seres humanos hacen naturalmente: aprender de la experiencia. A partir de métodos computacionales para aprender información de datos sin depender de un modelo matemático que los caracterice y mejorar su rendimiento cuando mejora la cantidad de información relevante disponible.

1.1.1. Tipos de aprendizaje automático

Existen diferentes metodologías de Machine Learning dependiendo del objetivo a alcanzar; un resumen de estos tipos de aprendizaje se puede encontrar en la figura 1.1. El tipo de aprendizaje, supervisado o no supervisado, se refiere a la naturaleza de los datos de entrenamiento: un algoritmo de Machine Learning supervisado conoce la entrada y la salida de los datos, su valor real, mientras que un algoritmo no supervisado solo conoce la entrada y a partir de esto estudia las relaciones ocultas entre la información analizada. Igualmente, cada tipo de entrenamiento tiene una subdivisión en dependencia de la naturaleza de los datos de salida, es decir, si la variable de salida es continua o discreta. Un resumen de los subtipos en función del tipo de aprendizaje y la variable de salida se presenta en la figura 1.2.

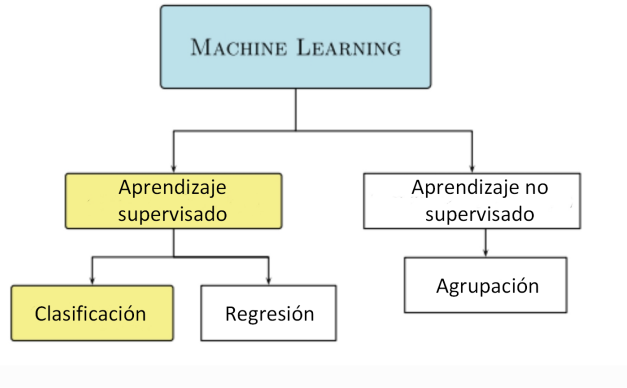

Figura 1.1: Tipos de aprendizaje automático. [3]

Figura 1.2: Clasificación del ML según tipo de aprendizaje y variable de salida - Elaboración propia

1.1.2. El proceso del Machine Learning

El entranmiento, validación y despliegue de un modelo de aprendizaje se compone de distintas etapas que se ejecutan de manera iterativa. La investigación preliminar muestra un consenso en estas etapas, citando a Álvarez[3] y Carvalho[4] se puede sintetizar el proceso en cinco etapas principales:

- Extracción y Carga de los datos de entrada y salida.
- Extracción de características.
- Construcción del modelo.
- Validación del modelo.
- Despliegue del modelo.

La selección de un algoritmo adecuado no solo depende del tipo de aprendizaje utilizado sino de la naturaleza y tamaño de los datos y la disponibilidad computacional. Algunos algoritmos pueden aplicarse a distintas técnicas de aprendizaje, por ejemplo, las *Máquinas de Soporte Vectorial* que serán explicadas matemáticamente en el capítulo 2 tienen aplicaciones en regresión y clasificación. Algunos autores separan el proceso de aprendizaje automático en subetapas distintas o evitan la iteratividad del proceso incluyendo una etapa de mantenimiento del modelo en la que se repiten fases previas para mejorar el rendimiento, sin embargo, la figura 1.3 resume estas etapas en un diagrama de flujo.

1.2. El mantenimiento predictivo

Una vez descrito el proceso de Machine Learning, es preciso enfocarse en sus aplicaciones: Cinar [5] sugiere que los algoritmos de aprendizaje automático se pueden aplicar sobre datos recolectados y permitir el diagnóstico y detección de averías. Esta aplicación del Machine Learning infunde inteligencia en los sistemas físicos para que puedan aprender automáticamente y adaptarse a entornos cambiantes utilizando experiencia histórica en la etapa de entrenamiento. Reforzando esta definición, Carvalho [4] revisa que la aplicación de enfoques analíticos como el sugerido puede lograr ventajas como la reducción de costos de mantenimiento, reducción en fallos de maquinaria, disminución en detenciones para reparación,

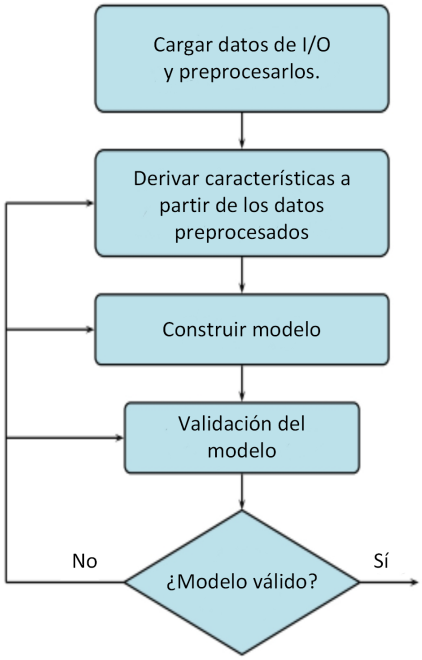

Figura 1.3: Flujo de trabajo del proceso de ML.[3]

disminución en el inventario necesario de repuestos, aumentos en la producción, mejoramientos de la seguridad operacional, verificación de las reparaciones, beneficios generales, entre otros.

1.2.1. Tipos de mantenimiento

Es preciso entonces describir los principales tipos de mantenimiento que se realizan actualmente en la industria, de forma que se puedan establecer ventajas y desventajas entre ellos y sugerir un enfoque adecuado para el contexto actual. La discusión alrededor del mantenimiento clasifica tres tipos:

Mantenimiento Correctivo

También conocido como Run-to-Failure (funcionamiento hasta el fallo, R2F) o mantenimiento no planeado. Se describe como la más sencilla entre todas las técnicas de mantenimiento, esto pues no se lleva a cabo hasta que el equipo no ha dejado de funcionar completamente. Para Nunes [6], el mantenimiento correctivo es la más primitiva de todas las estrategias, pues implica una detención en la producción mientras la pieza se reemplaza o repara. Entre

sus desventajas están un alto tiempo de detenimiento y un alto riesgo de fallos secundarios.

Mantenimiento Preventivo

También conocido como mantenimiento programado o mantenimiento basado en el tiempo, y abreviado PvM. Esta técnica es la primera en surgir como una solución a los altos costos que genera el mantenimiento correctivo: se basa en un mantenimiento periódico que inspecciona y mantiene los componentes a partir de un programa con la intención de anticipar fallas. Aunque no es la más reciente o más efectiva de las técnicas de mantenimiento, se mantiene como la estrategia más ampliamente extendida en la industria, y generalmente se realiza en periodos iguales, aunque los ciclos de trabajo proporcionados por el fabricante también pueden ser tenidos en cuenta; el PvM tiende a ser una metodología efectiva para evitar fallas mientras se mejora la eficiencia, sin embargo, su principal desventaja es que puede aumentar costos operacionales en consecuencia de tomar medidas correctivas de manera innecesaria.

Mantenimiento Predictivo

A veces conocido como mantenimiento basado en estadísticas y abreviado como PdM, utiliza herramientas predictivas para determinar si las acciones de mantenimiento pueden o no ser necesarias de forma que puedan ser programadas. Permite la detección de averías en una etapa temprana basada en técnicas como modelos de Machine Learning entrenados con datos históricos recuperados de sensores y dispositivos capaces de medir, monitorear y procesar parámetros como señales acústicas, corriente, voltaje, temperatura, vibraciones, entre otros. El mantenimiento predictivo puede ser considerado una evolución del mantenimiento basado en condición, pues alinea los paradigmas del Internet de las Cosas (IoT, *Internet of Things*) con automatización, ingeniería y analítica de datos. El PdM ofrece un balance entre las ventajas de los mantenimientos correctivo y preventivo al maximizar la vida útil de un componente y su tiempo de funcionamiento simultáneamente.

De acuerdo con [7], el sector de fabricación y manufactura ha notado una tendencia al aumento de uso de la colección e interpretación de datos como una herramienta para la toma de decisiones y el mejoramiento de la productividad, sostenibilidad y calidad de productos. El aprendizaje automático ha probado ser una de las herramientas más poderosas para el desarrollo de algoritmos predictivos, pues sus técnicas tienen la habilidad de manejar datos de alta dimensionalidad y multiples variables, extrayendo relaciones ocultas entre ellos. Nacchia [8] afirma que las ténicas de Machine Learning hacen uso de tres principios fundamentales: representación, evaluación y clasificación, pues permiten obtener más de menos

al mejorar según su uso. Como se ha mencionado anteriormente, la naturaleza de los datos analizados depende del objetivo del sistema; en esta investigación se utilizará la clasificación basada en audio, conocida como *machine hearing* [9].

1.2.2. El PdM como herramienta I4.0 en sistemas electromecánicos: Estado del Arte.

Los sistemas electromecánicos se eligen como objetivo de investigación por su propensión a largas jornadas de uso continuo. A continuación se hace una revisión de algunas investigaciones previas relevantes para el mantenimiento predictivo de estos elementos.

Modeling and Fault Detection of Brushless Direct Current Motor by Deep Learning Sensor Data Fusion - Suawa et al.

Un primer caso de estudio es la investigación realizada por Suawa [10], en la que se hace un énfasis sobre la importancia y la rica representación de fenómenos observados que se obtiene al combinar datos de múltiples sensores y analizan detección de averías en un motor de corriente directa sin escobillas a partir de distintas combinaciones de datos de sonido y vibración que alimentan modelos de aprendizaje profundo basados en redes neuronales convolucionales y en métodos de Long Short-Term Memory. La metodología propuesta por los investigadores se fundamenta en facultar la implementación de técnicas de mantenimiento predictivo sin tener mayor conocimiento técnico sobre el sistema estudiado y los datos obtenidos, pues sostienen que los ingenieros de Machine Learning muchas veces se enfrentan a conjuntos de datos de los que no tienen el conocimiento necesario. El esquema propuesto en esta investigación se detalla en la figura 1.4

Para la adquisición de datos utilizaron una board Redpitaya, un PC en miniatura que cuenta con un System-on-Chip Xilinx Zynq 7010 porque permite recolectar los datos a una alta tasa de muestreo y transmitirla mediante Ethernet; dicha tasa de muestreo era de 1953.125 kHz para el micrófono y acelerómetro, y los datos se recolectaron en forma de archivo de texto.

Se entrenaron modelos con la información de cada sensor individualmente y se encontró que, para los modelos de redes neuronales convolucionales profundas (DCNN, *Deep Convolutional Neural Network*), las señales del acelerómetro tuvieron una exactitud del 71.1 %, mientras que las del micrófono tuvieron un 96.8 %. Cuando se entrenó una red de LSTM se obtuvieron exactitudes del 64.3 % y del 63.8 % respectivamente. Al entrenar modelos de datos fusionados se obtuvo una notoria mejoría en el caso LSTM, obteniendo 73.6 % de exactitud, mientras

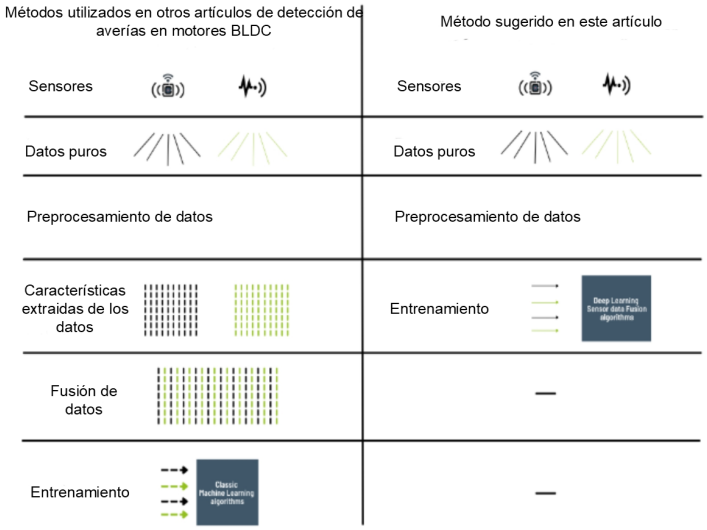

Figura 1.4: Metodología de Suawa para PdM de motores BLDC[10].

que para la DCNN se obtuvo un 98 % ; los autores hacen la claridad de que, en el caso DCNN solo se podía obtener al rededor de un 4 % de mejoría respecto al caso individual; cuando se hizo un enfoque combinado entre CNN y LSTM se obtuvo una métrica de exactitud del 93.8 %.

Cascade Convolutional Network with Progressive Optimization for Motor Fault Diagnosis under Nonstationary Conditions - Wang et al.

La investigación de Wang [11] hace un enfasis en la popularidad de las redes neuronales convolucionales en la detección de averías en motores gracias a su poderosa habilidad para la extracción de características, pero menciona su tendencia a la pérdidda de información, proponiendo como solución el uso de redes neuronales convolucionales en cascada (C-CNN, *Cascade-Convolutional Neural Networks*) para el diagnóstico de averías en condiciones no estacionarias, realizando el experimento con ejemplares a velocidad constante y a velocidad variable. Hace la observación de que, en el procesamiento de señales suele emplearse análisisestadístico, transformada de Fourier, transformada de paquetes de onda, descomposición de modelo empírico, entre otras técnicas, pero justifica la elección de expeirmento en que las técnicas clásicas de procesamiento de señales, extracción de características y reconocimiento de patrones suelen verse limitadas al enfrentarse con las condiciones cambiantes de un entorno industrial, elevando así el aprendizaje profundo como el conjunto de técnicas más deseable a utilizar.

En la metodología propuesta, Wang utiliza señales de vibración muestreadas a 6.4 kHz con nueve ejemplares en distinto estado de funcionamiento, obteniendo una exactitud del 99.62 % en el modelo C-CNN con optimización progresiva, además de proveer un análisis sobre la confusión entre algunas clases

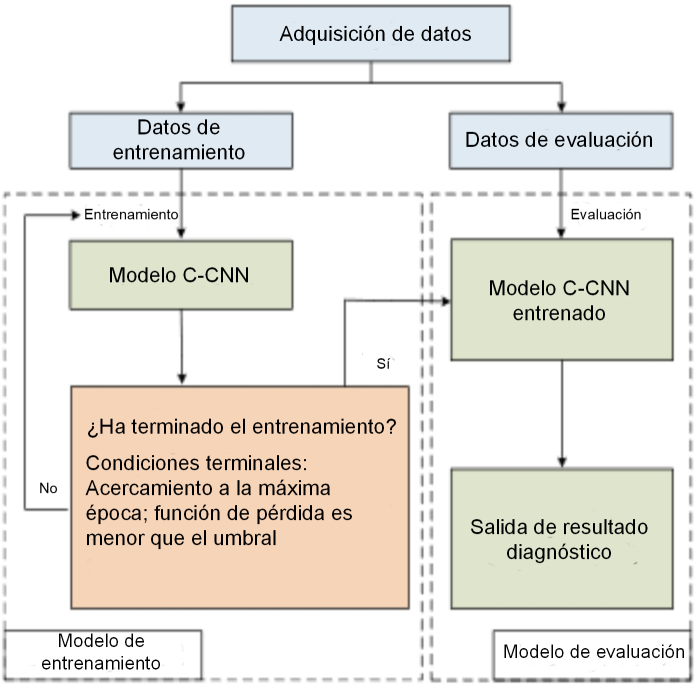

Figura 1.5: Metodología de Wang para PdM de motores[12].

A sound based method for fault detection with statistical feature extraction in UAV Motors - Altinors, Yol, Yaman.

En esta investigación conducida en la Universidad de Firat en Turquía, Altinors [13] propone un método basado exclusivamente en el sonido para la detección en tiempo real de averías en motores de aeronaves no tripuladas (UAV) con aplicaciones militares; se selecciona el enfoque de audio con la intención de disminuir la complejidad del método, descartando las señales de vibración y voltaje. Altinors propone la detección de cuatro clases: motor sano, fallo excéntrico, fallo de hélice y fallo de balinera. La figura 1.6 muestra los casos en motores BLDC A2212, cuyos datos fueron muestreados a 44 kHz y almacenados en formato m4a; en la fase de extracción de características ocuparon seis descriptores: promedio, desviación estandar, varianza, correlación, curtosis y asimetría; todo el método fue desarrollado en MATLAB

2020a y se entrenaron modelos de árbol de decisión (DT, *Decision Tree*), máquinas de soporte vectorial (SVM, *Support Vector Machines*) y k-Nearest Neighbors (kNN). Se obtuvieron las mejores exactitudes con los modelos de SVM; para motores de 1400 KV del 91.25 %, para ejemplares de 2200 KV se obtuvo una exactitud del 99.75 % en SVM y kNN, y en el caso de 2700 KV del 90.00 %.

Como el método propuesto involucra la detección en tiempo real, se implementaron los modelos en un sistema embebido que permite a un operador conectado a internet realizar el diagnóstico a distancia. La figura 1.7 muestra el esquema propuesto para dicho fin.

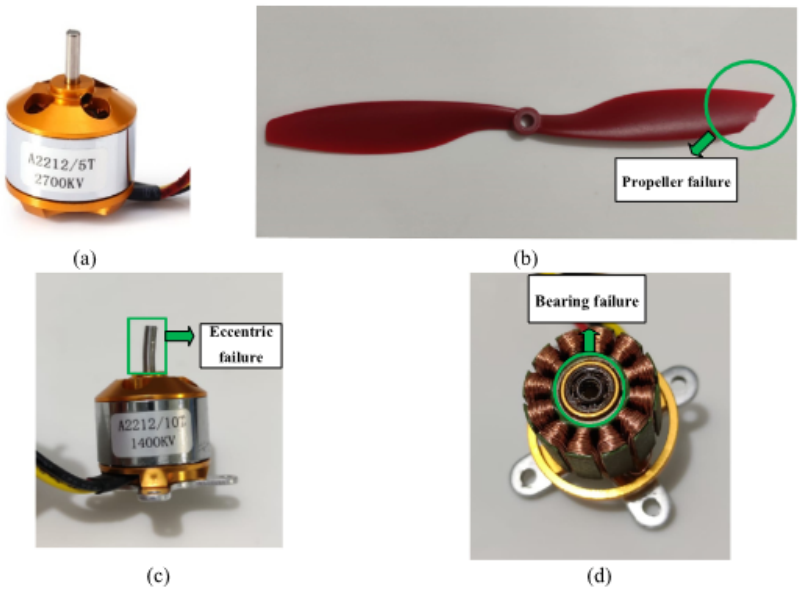

Figura 1.6: Clases consideradas por Altinors[13].

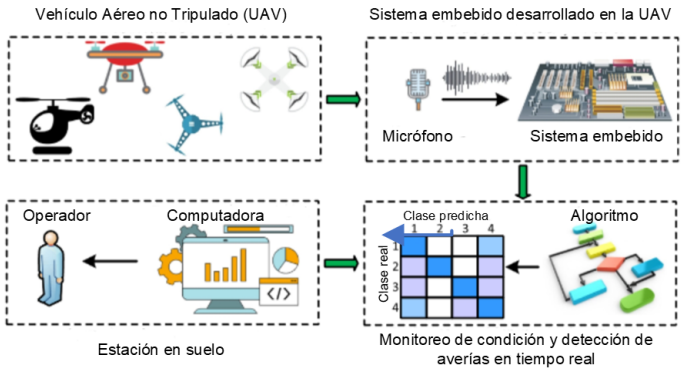

Figura 1.7: Esquema de análisis en tiempo real propuesto por Altinors[13].

La tabla 1.1 hace referencia a investigaciones conducidas sobre detección de averías sistemas

electromecánicos y relaciona los métodos de aprendizaje automático con la naturaleza de las señales adquiridas, además de enseñar métricas de evaluación presentadas por sus autores. Los significados de los acrónimos de cada método se relacionan en la tabla 1.2. Un artículo de revisión sistemática completo se encuentra en la sección de Anexos.

Tabla 1.1: Descripción de los métodos utilizados en el material revisado.

Autor	**Método**	**Señales adquiridas**	**Métricas**
Babu et al. [14]	-	Multisensor	MSE: 0.0606
Cakir et al. [15]	SVM, k-NN	Multisensor	Acc: 0.999
Carrera et al. [16]	CNN	Sonido	Acc: 0.9617
Duc Nguyen et al. [17]	k-NN, DT, LG, SVM	Corriente	AUC: 0.995
Glowacz [18]	Nearest Neighbour	Sonido	-
Han et al. [19]	k-NN, CNN, SVM	Vibración, sonido	Acc: 1.000
Hesabi [20]	LSTM	-	Acc: 0.97
Li et al. [21]	SVM	Sonido	Acc: 0.9592
Lu. [22]	CNN	Vibración	Acc: 1.000
Nikfar [23]	SVM	Vibración, temperatura	Acc: 1.000
Pacheco-Chérrez [24]	SVM	Vibración, sonido	Acc = 0.9666
Patel and Giri. [25]	RF	Vibración	Acc: 0.997
Patil and Wani. [26]	CNN	Sonido	Acc: 0.95
Raja . [27]	SVC, RF, DT, k-NN	Corriente	Acc: 0.97
Shao [28]	DCNN	Vibración, corriente	Acc: 0.9983
Shifat and Hur. [29]	ANN	Vibración, corriente	Acc: 0.98
Souza et al. [30]	CNN	Vibración	Acc: 0.9725
Strantzalis et al.	CNN	Sonido	Acc: 0.978
Suawa et al. [10]	DCNN, LSTM	Vibración, sonido	Acc: 0.988
Wang et al. [12]	Bayesian inference	Vibración	Acc $>$ 0.9
Wang et al. [11]	C-CNN	Vibración	Acc: 0.9449
Xu et al. [31]	LSTM, GRU, TCN	Vibración	Acc: 0.9390
Yaman, Yol, Altinors [32]	SVM	Sonido	Acc: 1.000
Yao et al. [33]	CNN	Sonido	Acc: 0.9175

Excluyendo los algoritmos que se relacionen con el aprendizaje profundo, los algoritmos más utilizados en la clasificación de fallas son las máquinas de soporte vectorial (SVM), el algoritmo k-Nearest Neighbors (kNN), los Decision Trees (DT) y los Random Forest (RF), y las señales más comunes son de vibración y sonido. Se selecciona el motor BLDC como sistema electromecánico de interés para estudiar las señales acústicas como datos de entrenamiento para la detección de averías.

Tabla 1.2: Significado de los acrónimos de los métodos.

Acrónimo	**Método**
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
C-CNN	Cascade Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
DT	Decision Tree
GRU	Gated Recurrent Unit Neural Network
k-NN	k-Nearest Neighbors
LSTM	Long-Short Term Memory Neural Network
RF	Random Forest
SVM	Support Vector Machines
TCN	Temporal Convolution Neural Network

1.3. Estudio del motor DC

El motor de corriente continua fue el primer tipo de motor ampliamente utilizado, y a pesar de que ha perdido muchos campos de aplicación en favor de los motores de corriente alterna, también son relativamente simples de entender, y su funcionamiento es extensible a otros motores haciéndolos sujetos de investigación muy útiles al ser un vehículo ideal de aprendizaje [34].

1.3.1. Modelo matemático del motor DC

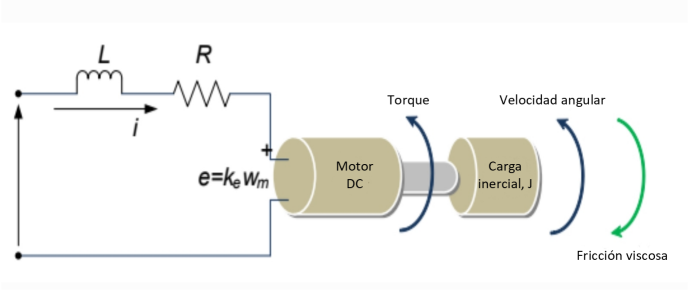

Figura 1.8: Arreglo electromecánico de un motor c.c. con escobillas. [35]

La figura 1.8 muestra un arreglo electromecánico de un motor DC convencional; la fuerza electromotriz en el motor estará definida como $e = k_m\omega$, en la que se define a k_m como una constante del motor y ω como su velocidad angular. A partir de la ley de voltaje de Kirchoff

se infiere entonces:

$$V_s = RI + L\frac{dI}{dt} + e \tag{1.1}$$

Una vez alcanzado el estado estacionario, la expresión anterior se convierte en

$$V_s = RI + e \tag{1.2}$$

Sin embargo, para generalizar se utilizará la ecuación 1, de la cual se despeja e

$$e = -RI - L\frac{dI}{dt} + V_s \tag{1.3}$$

Por otra parte, a partir de la segunda ley de Newton se puede analizar la sección mecánica del motor:

$$\tau_m = k_f\omega + J\frac{d\omega}{dt} + \tau_L \tag{1.4}$$

Donde τ_e es el torque eléctrico, J la inercia del rotor, k_f la constante de fricción y τ_l una supuesta carga mecánica que se asume cero.

Para establecer una relación diferencial entre ambas etapas del sistema electromecánico se definen la constante de torque y la constante de fuerza contraelectromotriz, $k_T = \frac{\tau_e}{\omega}$ y $k_e = \frac{\omega}{e}$

Haciendo esto se puede establecer el siguiente sistema de ecuaciones diferenciales.

$$\frac{dI}{dt} = -I\frac{R}{L} - \omega\frac{k_e}{L} + V_s\frac{1}{L}$$

$$\frac{d\omega}{dt} = I\frac{k_T}{J} - \omega\frac{k_f}{J} + \tau_L\frac{1}{J}$$

Este sistema de ecuaciones se resuelve haciendo uso de la transformada de Laplace con condiciones iniciales iguales a cero, con lo que se obtiene la función de transferencia $G(s)$:

$$G(s) = \frac{\omega}{V_s} = \frac{k_T}{JLs^2 + (RJ + K_fL)s + K_fR + k_ek_T} \tag{1.5}$$

Para simplificar la función de transferencia se toman las siguientes consideraciones:

$$\begin{aligned} k_f &\to 0 \\ \Rightarrow RJ &>>> k_f L; \\ k_e k_T &>>> RK_f \end{aligned}$$

Por lo que la función de transferencia puede reescribirse como:

$$G(s) = \frac{k_T}{JLs^2 + RJs + k_e k_T} \tag{1.6}$$

Para finalizar, se introducen las constantes de tiempo del motor DC:

$$\tau_m = \frac{RJ}{k_e k_T} \tag{1.7}$$

$$\tau_e = \frac{L}{R}$$

Con la que se obtiene la expresión final para la función de transferencia de un motor DC [35]:

$$G(s) = \frac{\frac{1}{k_e}}{\tau_m \tau_e s^2 + \tau_m s + 1} \tag{1.8}$$

1.4. El motor BLDC

El motor c.c. convencional cuenta con escobillas adheridas al estator, en cambio, el motor *brushless*, sin escobillas, hace la conmutación por control electrónico; además, los embobinados del estator en el motor BLDC se energizan para generar la rotación sin ningún contacto entre estator y rotor. Además, en los motores sin escobillas se hace uso del efecto Hall como dispositivos sensores. El arreglo más común en un motor brushless es de tres fases por su eficiencia y bajo momento de torsión, además de alta precisión en el control.

1.4.1. Modelo matemático del motor BLDC

El modelo matemático no es muy diferente en estructura, salvo por la inclusión de las tres fases generando un cambio en las constantes mecánica y eléctrica de la ecuación 1.8. La figura

1.9 muestra un arreglo electromecánico del motor BLDC en una configuración simétrica.

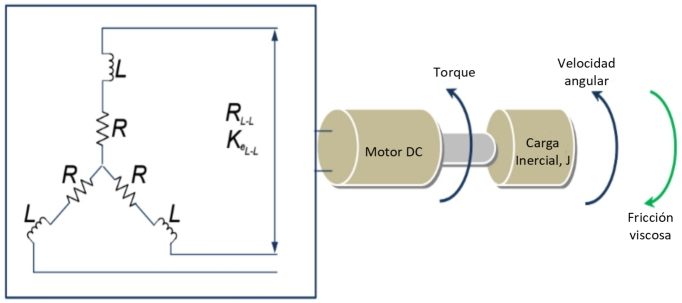

Figura 1.9: Arreglo trifásico para un motor BLDC. [35]

Con las suposiciones consideradas, las constantes τ_m y τ_e se convierten en:

$$\tau_m = \frac{3JR}{K_e K_t}, \tau_e = \frac{L}{3R} \tag{1.9}$$

Considerando los efectos de la fase y siendo R_ϕ la resistencia entre fases y $K_{e(L-L)}$ la constante contraelectromotriz de cada fase, la constante de tiempo mecánica se convierte en:

$$\tau_m = \frac{3R_\phi J}{K_e K_t} \tag{1.10}$$

Relacionando potencia mecánica y potencia eléctrica también se obtiene

$$K_e = 0.0605 K_t \tag{1.11}$$

Para finalmente obtener la función de transferencia del motor BLDC en la ecuación 1.12.

$$G(s) = \frac{\frac{1}{K_e}}{\tau_m \tau_s s^2 + \tau_m s + 1} \tag{1.12}$$

1.5. Resumen del capítulo 1

En este capítulo se entregó una revisión teórica y referencial de los conceptos que se aplicaron en esta investigación: se introdujo al Machine Learning, el Mantenimiento Predictivo, al motor de corriente continua y su presentación sin escobillas, el motor BLDC, el vehículo de aprendizaje que actuó como sujeto en capítulos posteriores.

Capítulo 2

Metodología para diagnóstico de averías en motores BLDC basado en procesamiento de audio

2.1. Comprobación experimental del modelo matemático

2.1.1. Caracterización de un motor DC

El proceso de caracterización involucra la conducción de una serie de experimentos que permitan comprobar experimentalmente el modelo fisico-matemático de una planta; en esta investigación es importante conducir un ensayo de caracterización para conocer en cierta medida el sistema electromecánico a manipular y asociarlo a las posibles averías que se podrían encontrar. Estos ensayos estáticos o dinámicos se concentran en encontrar parámetros que tengan un sentido físico, y para objetivos de esta investigación se limitará a las resistencias e inductancias de armadura.

2.2. Diseño del experimento y adquisición de los datos

La obtención del conjunto de datos es crucial para el desarrollo de buenos modelos de aprendizaje automático. Existen varias consideraciones que se deben tomar antes de diseñar un experimento. El sistema físico a analizar debe conocerse, caracterizarse y a partir de este punto establecer las consideraciones necesarias para el experimento; la investigación reali-

zada en el capítulo 1 establece ciertos lineamientos para la experimentación con motores BLDC y la obtención de un dataset basado en audio, incluidas distancia entre el tambor y el eje del micrófono. A continuación, se describirá la ruta metodológica que se siguió en la obtención del conjunto de datos:

2.2.1. Etapa 1: Selección del universo

Al decidir con qué referencia de motor experimentar se consideró la popularidad en aplicaciones. En el diseño de aeronaves UAV de modelo de entrada se ha encontrado un común uso del motor BLDC A2212/6T 2200 KV, por lo que se seleccionaron cuatro ejemplares de este motor como universo de estudio. Algunas características del A2212/6T 2200 KV se recogen en la tabla 2.1.

Tabla 2.1: Tabla de características del motor A2212/6T 2200 kV.

Dimensiones	27.5 mm x 30 mm
Peso	49 g
Diámetro del eje	3.17 mm
Resistencia interna	90 mΩ
ESC mínimo recomendado	18 A
Máxima eficiencia	80 %
Batería recomendada	2-3s LiPo

Para la operación del A2212 de la figura 2.3 se requiere un controlador electrónico de velocidad, ESC por sus siglas en inglés: *Electronic Speed Controller*. Según los requerimientos del motor se selecciona el ESC de la figura 2.2 que tiene las características en la tabla 2.2. Además, las características de la batería Zippy Compact 2100 seleccionada para alimentar el motor se recogen en la tabla 2.3, y se puede observar en la figura 2.1.

Tabla 2.2: Tabla de características del ESC 30A.

Dimensiones	45 mm x 25 mm x 11 mm
Peso	25 g
Salida BEC	5V -2A
Corriente máxima	30 A

Una vez establecido el universo de motores a analizar y el equipo necesario para su funcionamiento es imprescindible establecer las condiciones operacionales que serán estudiadas. El trabajo de Altinors basado en el análisis de aeronaves no tripuladas describe cuatro tipos de fallo, y considera tres de ellos para la experimentación:

Figura 2.1: Batería Zippy Compact 2100

Figura 2.2: Electronic Speed Controler de 30A.

Figura 2.3: Motor BLDC A2212

Tabla 2.3: Tabla de características de la batería Zippy 2100 mAh.

Dimensiones	135 mm x 45 mm x 18 mm
Capacidad	2100 mAh
Voltaje	3S1P / 9.9 V
Descarga	30 C constante / 60 C ráfaga

- Fallo de balinera: Un fallo asociado a los balines del motor, que se desgastan después de un tiempo operacional y al someterse a condiciones de polvo, humedad y temperatura. Tiene implicaciones directas en la estructura del motor.
- Fallo excéntrico Principalmente visto en aplicaciones de aeronaves no tripuladas, implica un desbalance en el eje del motor que se puede provocar por choques. Para efectos de esta investigación será excluido
- Fallo de hélice Similar al fallo excéntrico pero de menor gravedad, se provoca usualmente por una colisión o por desgaste en el acople de la hélice.

Fundamentados en esto y excluyendo el fallo excéntrico por motivos operacionales y de seguridad, la etapa experimental de esta investigación se consideraron tres combinaciones distintas de análisis:

- Ejemplares sanos - Ejemplares con fallo en hélice (H-P, de *Healthy-Bearing*).
- Ejemplares sanos - Ejemplares en fallo. (H-F, de *Healthy-Faulty*).
- Ejemplares sanos - ejemplares con fallo en hélice - ejemplares con fallo en balinera y embobinado (HPB, de *Healthy-Propeller-Bearing*).

En este momento se constituyó una hipótesis: en la categorización HPB, introducida al final de la investigación, se espera un nivel mayor de confusión entre la categoría Balinera-Embobinado y las categorías Sano y Hélice pues los sonidos producidos por la categoría B tenían similitudes con las otras dos. Posteriormente se desarrollará esta hipótesis en el análisis de resultados.

2.2.2. Etapa 2: Condiciones para la adquisición de datos I/O

En el capítulo 1 se estableció la importancia que le dan múltiples autores a las condiciones en las que se adquiere la información de los sistemas electromecánicos. Las condiciones experimentales muchas veces se conforman respecto al contexto en el que se realice dicha experimentación, los materiales a utilizar y el tipo de estudio a realizar. Por ejemplo, Altinors fundamenta sus condiciones operacionales basándose en el micrófono utilizado, el laboratorio en el que se realizó la adquisición y el software y hardware computacional de grabación. Se establecieron las siguientes condiciones reglamentarias para la adquisición de datos:

- Dispositivo de grabación: Micrófono capacitivo MCJR-005
- Distancia entre el tambor del micrófono y el eje del motor: 22 cm.
- Frecuencia de muestreo: 16 kHz
- Duración de la pista: 10 s.
- Software de grabación: Audacity.

Algunos textos de la literatura como Altinors [32] recomiendan una distancia mayor entre el motor y el micrófono MCJR-005 de la figura 2.4, esta prerrogativa podría fundamentarse en la significancia dentro de un contexto operacional, sin embargo, el micrófono utilizado tiene una sensibilidad bastante alta, lo que lo hace muy susceptible al ruido exterior. Así mismo, también recomiendan una frecuencia de muestreo superior, sin embargo el MCR-005

Figura 2.4: Micrófono MCJR-005

permite una máxima frecuencia de 16 kHz. La metodología de grabado se fundamenta en hacer pasos regulares en el suministro de voltaje a los motores con la intención de captar su sonido en más condiciones operacionales y su sonido se captura haciendo uso del software libre Audacity para Windows durante diez segundos. El proceso es realizado para los cuatro motores del universo y se detalla con mayor profundidad a continuación.

2.2.3. Etapa 3: Montaje experimental

El proceso experimental seguido guarda una cierta similitud con el proceso de caracterización mecánica de un motor DC: se varía el voltaje inducido, variando la velocidad angular y adquiriendo una señal de interés: en el proceso de caracterización mecánica esta señal es la velocidad angular del motor, mientras que en esta investigación es su sonido. Al ser alimentados por una batería independiente y no por una fuente variable de voltaje, es necesario incluir una etapa que permita con mayor facilidad la manipulación del ESC.

Existen varias formas de controlar la velocidad angular de un motor brushless; la primera que se implementó en esta investigación es más ortodoxa: se basa en el uso de potenciómetros o controladores de radiofrecuencia cuyo valor sea adecuado mediante un microcontrolador y escrito en el ESC. Utilizar esta alternativa no garantiza mayor control sobre el paso entre voltajes para la grabación, lo que llevó a la idea de una nueva alternativa más afín con la naturaleza de la investigación: el mantenimiento predictivo es parte fundamental de la

Industria 4.0, así que pueden utilizarse otras estrategias de la I4.0 para adquirir las señales.

El segundo enfoque se basó en herramientas del Internet de las Cosas, por lo que requirió el reemplazo de Arduino Uno como microcontrolador por la placa ESP32, que cuenta con conexiones Wi-Fi y Bluetooth aprovechables para el IoT. Detalles de la placa y su salida de pines pueden verse en la figura 2.5

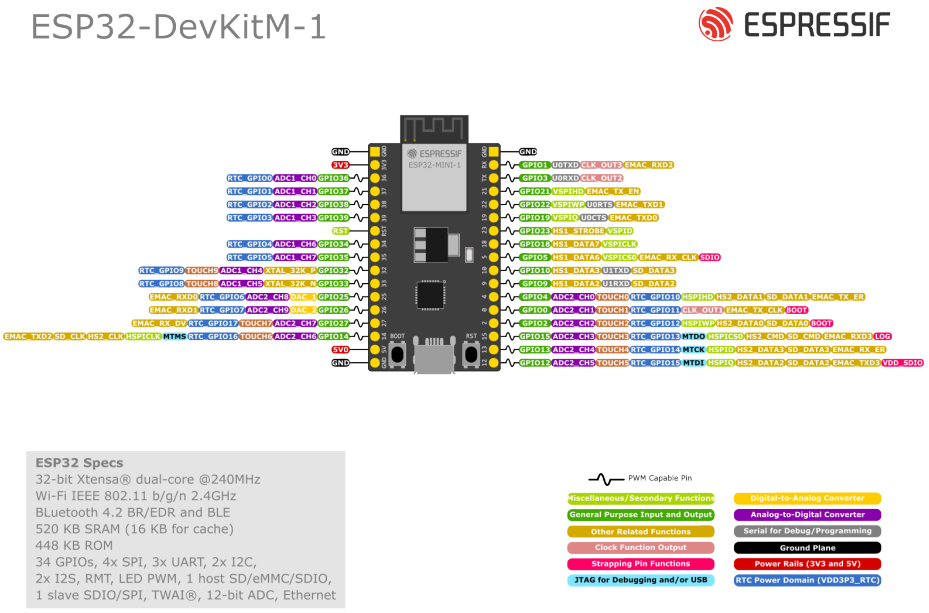

Figura 2.5: Salida de pines de la ESP32.[36]

El uso de IoT aunque en principio parece una complicación evitable, proporciona varias ventajas, siendo la más importante la estabilidad que brinda al experimentador en la selección de un voltaje. Para esto se diseñó una aplicación en Blynk, una plataforma que facilita las implementaciones de IoT con microcontroladores como el ESP32. La aplicación en Blynk no requería más que un slider que permitiera el control de velocidad; en la sección de Anexos se incluye el código .ino implementado con el que se recibe un valor de la aplicación y se escribe en el controlador de velocidad como periodo de los pulsos PWM, variando así la velocidad del motor.

La figura 2.6 muestra el diagrama de conexiones utilizado en el experimento. Una vez solucionado el problema de variación de velocidad, se inició el proceso de grabación con Audacity, un software libre para Windows para la grabación y edición de audio. Los motores 1 y 2 se sometieron a condiciones normales y falla de hélice, mientras que los motores 3 y 4 proveen información de condiciones normales y fallos en balinera y embobinado. Los valores digitales que se escribieron en el controlador de velocidad y hacían operacional al motor oscilaban

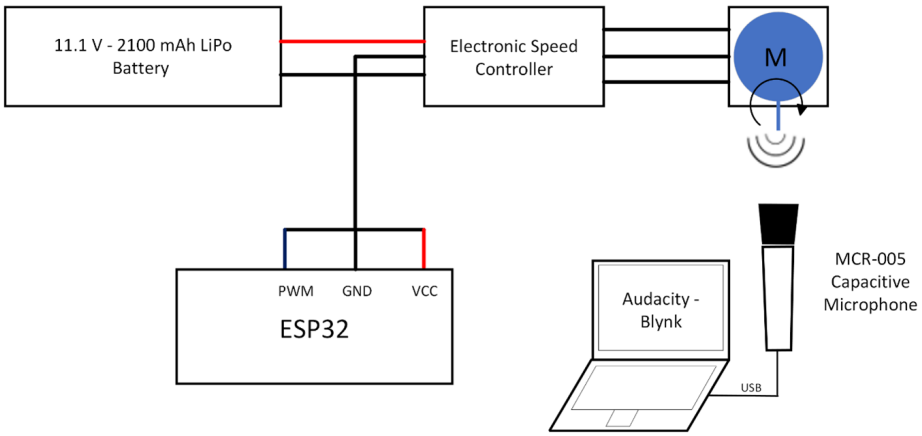

Figura 2.6: Diagrama de conexiones para la adquisición de datos, elaboración propia.

entre 1450 y 2000. Los archivos de audio se guardaron en formato *.wav* por su alta compatibilidad. Para garantizar homogeneidad, los archivos son recortados por software en la etapa posterior. Una vez obtenidas 43 grabaciones de los cuatro ejemplares en distintos modos de operación, se cargan los datos en MATLAB R2022b, se validan los datos para evitar posible información perdida y se almacenan en una matriz con dimensiones 43x160000. Este conjunto de datos crudos será el sujeto principal de las etapas de análisis, procesamiento y entrenamiento. Gráficas mostrando ejemplos de las señales de audio previas al procesamiento se muestran en 2.7, mientras que la figura 2.8 muestra el audio normalizado de un ejemplar de cada clase seleccionado aleatoriamente.

2.3. Análisis y extracción de características

Las características que podrían ser representativas en la construcción de una matriz de descriptores se infirieron a partir de la investigación descrita en el capítulo 1 y experiencia de proyectos anteriores. Uno de los focos principales de esta investigación es demostrar la efectividad de los métodos espectrales, en comparación con los más ampliamente utilizados métodos estadísticos. Por ejemplo, Altinors [13] basa su enfoque en seis descriptores: media, desviación estándar, varianza, correlación, curtosis y asimetría. A continuación se presenta una descripción matemática de la selección preliminar de características. Además, se describe el proceso para la obtención de cada característica en MATLAB.

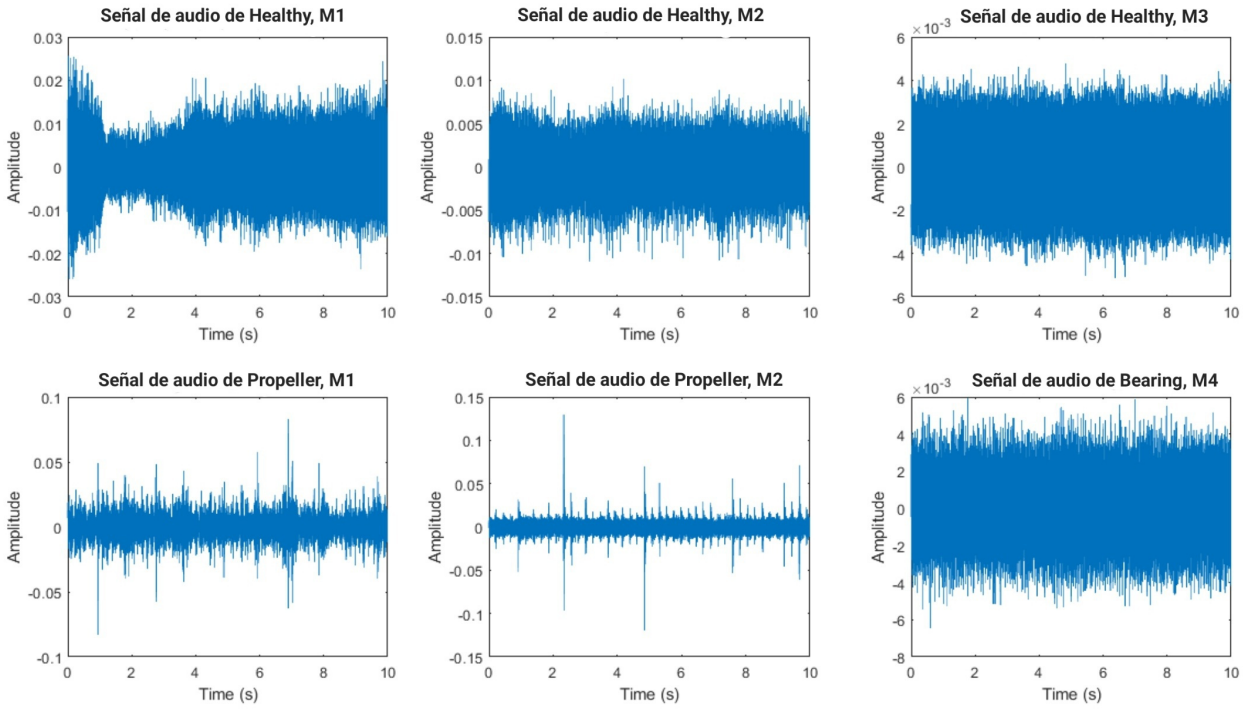

Figura 2.7: Señales de audio sin procesar.

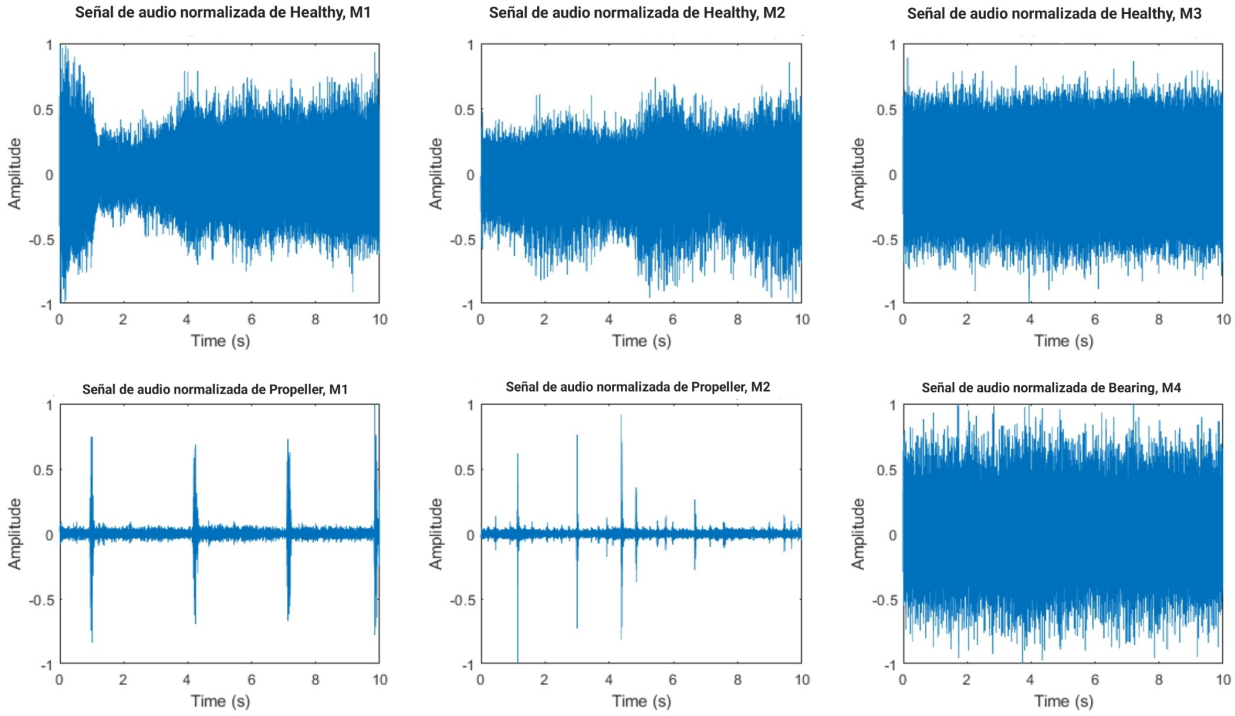

Figura 2.8: Señales de audio normalizadas.

2.3.1. Características de naturaleza estadística

Media aritmética

Ante un conjunto de datos $x = x_1, x_2, ..., x_n$ se define la media aritmética o *promedio* como

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.1}$$

Desviación estándar

Considerando el mismo conjunto de datos, se define la desviación estándar como una medida de dispersión de los datos. Matemáticamente se expresa

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2.2}$$

De forma que una desviación estándar pequeña representa una alta agrupación de los datos, mientras que una desviación estándar alta representa una dispersión relevante en los datos; la desviación estándar determina la forma de la distribución probabilística.

Asimetría estadística

De aquí en adelante referida por su nombre en inglés, *skewness*, es una medida del grado de asimetría de una distribución; en su gráfica de frecuencias, una distribución asimétrica mostrará una tendencia hacia un lado, donde tiende a encontrarse la media de los datos. La skewness se define por la ecuación 2.3.

$$Skewness = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{Y_i - \bar{Y}}{\sigma}\right)^3 \tag{2.3}$$

Curtosis

Como consecuencia de la skewness nace la curtosis, a veces escrita *kurtosis* por su nombre en inglés: si una distribución tiene una tendencia asimétrica, la kurtosis mide qué tan asimétrica es, es decir, qué tan representativa es la *cola* de la distribución; en análisis de características

esto puede intuirse como una tendencia de la distribución a los outliers; similar a la skewness, se define en la ecuación 2.3.[37]

$$Kurtosis = \frac{1}{N}\sum_{i=1}^{N}(\frac{Y_i - \bar{Y}}{\sigma})^4 \tag{2.4}$$

Tasa de cruce por cero

La zero-crossing rate es una medida del número de veces que una señal cruza el eje. En MATLAB se calcula con la función `zcr(x,fs)` y se computa en cada cuadro temporal de la señal.

2.3.2. Características de naturaleza espectral

Aunque el tratamiento estadístico es el más extendido en la revisión de literatura, muchas veces el análisis temporal de una señal no es suficiente. Estudiar la naturaleza de la señal en otros dominios puede resultar complementario, así que se introduce el análisis en dominio de la frecuencia como una herramienta de la que se extraen características importantes.

Transformada de Fourier

Ante una señal continua aperiódica $x(t)$, se define una función $X(F)$, denominada *transformada de Fourier de* $x(t)$ en la ecuación 2.5

$$X(F) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi Ft}dt \tag{2.5}$$

En la ecuación 2.5 se puede apreciar la transformación del dominio temporal al dominio de la frecuencia. Igualmente, toda transformación de dominio tiene su reverso, como dicta la ecuación 2.6 definiendo la *transformada inversa de Fourier*

$$x(t) = \int_{-\infty}^{\infty} X(F)e^{j2\pi Ft}dF \tag{2.6}$$

Una de las condiciones para la existencia de la transformada de Fourier de una función es que la señal $x(t)$ sea absolutamente integrable, y en consecuencia tenga energía finita. Sin embargo, la transformación al dominio de la frecuencia dada por la ecuación 2.5 no

es computacionalmente apta, por lo que se requiere el uso de la transformada discreta de Fourier de una señal aperiódica discreta en el tiempo $x(n)$. Su transformada de Fourier es

$$X(\omega) = \sum_{-\infty}^{\infty} x(n)e^{-j\omega n}$$

Se define entonces su transformada discreta de Fourier (DFT) como

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}, k = 0, 1, 2, ..., N-1 \tag{2.7}$$

La transformada discreta de Fourier se calcula haciendo uso de un algoritmo FFT

Introducir la transformada discreta de Fourier permite la introducción del periodogramaP_{xx}, un estimado del espectro de densidad de potencia

$$P_{xx} = \frac{1}{N}|X(f)|^2 \tag{2.8}$$

En MATLAB, la función `fft(x)` calcula la transformada discreta de Fourier de un vector x.

Densidad de Potencia Espectral

La Densidad de Potencia Espectral (*PSD* por sus siglas en inglés, *Power Spectral Density*), es una representación de la potencia contenida en cada frecuencia de una señal. El método del periodograma es un método paramétrico, es decir, hace suposiciones sobre la generación de los datos. Existen métodos no paramétricos como los descritos por Barlett (1948), Blackman y Turkey (1958), pero el más popular es el método de Welch basado en un promediado de periodogramas modificados y aplicar una función de ventana a segmentaciones de los datos que podrían o no solaparse entre sí. El método de Welch introduce finalmente una modificación al periodograma dada en la ecuación 2.9.

$$P_{xx}^{i}(f) = \frac{1}{MU}\left|\sum_{n=0}^{M-1} x_i(n)w(n)e^{-j2\pi fn}\right|^2, i = 0, 1, ..., L-1 \tag{2.9}$$

Donde M es la longitud de los segmentos y U es un factor de normalización que se selecciona como

$$U = \frac{1}{M} \sum_{n=0}^{M-1} w^2(n) \tag{2.10}$$

Finalmente, al ser un método no paramétrico, la densidad de potencia de Welch es una estimación cuyo valor viene dado por la ecuación 2.11, donde L es el número de segmentos. [38]

$$P_{xx}^{W}(f) = \frac{1}{L} \sum_{i=0}^{L-1} P_{xx}^{\ i}(f) \tag{2.11}$$

En MATLAB, el cálculo de la densidad de potencia espectral de Welch se calcula con la función `pwelch`.

Radio armónico

Un descriptor que podría ser interesante en el análisis de frecuencia es la armonicidad del audio; el radio armónico se refiere a la razón de la potencia de una frecuencia fundamental respecto a la potencia total en un cuadro de audio. [39]

El radio armónico se calcula en MATLAB con la función `harmonicRatio(audioIn,fs)`, que recibe una señal de audio y su frecuencia de muestreo. Internamente, se determina la autocorrelación normalizada de la señal con la ecuación 2.12.

$$\Gamma(m) = \frac{\sum_{n=1}^{N} s(n)s(n-m)}{\sqrt{\sum_{n=1}^{N} s(n)^2 \sum_{n=0}^{N} (s-m)^2}}, m = 1, 2, ..., M \tag{2.12}$$

Donde s es un cuadro de audio con N elementos y M es el retraso máximo del cálculo asociado a una frecuencia fundamental mínima de 25 Hz. El estimado inicial del radio armónico se define en la ecuación , y posteriormente se mejora haciendo uso de interpolación parabólica.

$$\beta HR = \max_{M_0 \leq m \leq M} \Gamma(m) \tag{2.13}$$

Centroide espectral

El centroide espectral representa la ubicación del centro de masas de un espectro. Se calcula considerando una distribución cuyos valores son las frecuencias presentes al hacer transformada de Fourier y las probabilidades de observar una frecuencia son su amplitud normalizada.

$$centroid = \frac{\sum_{k=b_1}^{b_2} f_k s_k}{\sum_{k=b_1}^{b_2} s_k} \tag{2.14}$$

La ecuación 2.14 es una forma de promedio ponderado; f_k es la frecuencia en Hertz de la celda k, s_k es el valor espectral de la celda y b_1, b_2 son sus bordes. En MATLAB, la función `spectralCentroid(x,f)` calcula el centroide espectral de la señal en el tiempo.

Planura espectral

La planura espectral, *flatness*, mide la razón entre la media geométrica del espectro y su media aritmética[40]. En MATLAB se calcula con la función `spectralFlatness(audio,fs)` que hace el cálculo de la ecuación 2.15 sigue la misma convención del centroide espectral.

$$flatness = \frac{\left(\prod_{k=b_1}^{b_2} s_k\right)^{\frac{1}{b_2-b_1}}}{\frac{1}{b1-b2}\sum_{k=b_1}^{b_2} s_k} \tag{2.15}$$

Entropía espectral

La entropía espectral de una señal también sirve como una medición de potencia espectral tratando su distribución de potencia normalizada como una distribución de probabilidad y calcula su entropía Shannon. Este descriptor tiene usos en la detección y diagnóstico de fallas. Sea una señal $x(n)$ y $X(m)$, se define su probabilidad $P(m)$ como

$$P(m) = \frac{S(m)}{\sum_i S(i)} \tag{2.16}$$

Y su entropía espectral H en la ecuación 2.17

$$H = -\sum_{m=1}^{N} P(m) \log_2 P(m) \tag{2.17}$$

En MATLAB, la entropía espectral del método Shannon puede calcularse con la función `pentropy(x)`.[41]

2.4. Análisis Exploratorio de Datos

Al construir una matriz de características que entrenará un modelo de clasificación es importante seleccionar aquellas que sean más relevantes en la determinación de la salida deseada. Los algoritmos de aprendizaje automático pueden verse afectados por condiciones como el overfitting, que consiste en un sobreentrenamiento del modelo en el que se aprende a memorizar los datos de entrenamiento en lugar de poder inferir a partir de un vector de características. Otra condición que se intenta evitar al entrenar modelos de aprendizaje es la maldición de la dimensionalidad; este fenómeno, descrito por primera vez en 1961 por Bellman planteando el siguiente ejemplo:

Un ejemplo para visualizar la maldición de la dimensionalidad es descrito por Bellman de la siguiente forma[42]: Asumiento una función de salida $u = u(t)$ especificada sobre $0 \geq t \geq 1$ por sus valores en los puntos $k(0.1), k = 0, 1, 2, ..., 99$, se tabulan 100 valores. Asimismo, si $u = u(s, t)$, una función de dos variables, especificada entre $0 \geq s, t \geq 1$ ante una matriz similar a la especificada anteriormente, se requieren $100x100 = 10^4$ valores. Si la función fuese de tres variables, se requieren 10^6 valores.

La maldición de la dimensionalidad se refiere a las consecuencias computacionales de una matriz con un número representativo de características. Para Press[42], podría asumirse que entre mayor número de características aumenta la capacidad predictiva del modelo, pero esto solo es cierto si dichas características realmente ofrecen predictibilidad; de lo contrario, lo único que se consigue es deteriorar el rendimiento del modelo y aumentar el costo computacional necesario para su funcionamiento. Un primer acercamiento a aquellas características que sí presentan mayor relevancia para el entrenamiento del modelo se obtiene al hacer cálculos de correlación. La correlación representa qué relación de significancia tiene una variable respecto a otra, y puede ser de dos tipos: positiva y negativa. El coeficiente de correlación un valor entre -1 y 1; la metodología más común para su cálculo es la Correlación de Pearson.

La asociación entre variables ordinales más usada es el coeficiente de correlación lineal de

Pearson, denotado r. Para pares de cantidades $(x_i, y_i), i = 1, 2, ..., N$, el coeficiente Pearson se calcula con la ecuación:

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} \tag{2.18}$$

Donde $\bar{x}$ es la media de los x_i y $\bar{y}$ es la media de los y_i

El coeficiente de correlación de Pearson es calculado para todo el conjunto de datos en MATLAB mediante la función *corrcoef*. La función recibe dos argumentos que retornan una matriz simétrica cuya diagonal principal es 1 por convención y los demás valores representan los coeficientes de correlación.

En la revisión de investigaciones antecedentes se encuentra frecuentemente la indicación de que las señales acústicas deben ser normalizadas, es decir, la magnitud de la señal debe convertirse a una razón entre los valores de sí misma que la limite entre 0 y 1. La normalización descrita por Altinors[13] se conoce comunmente cono normalización min-max se representa matemáticamente en la ecuación

$$Z' = \frac{Z - Z_{min}}{Z_{max} - Z_{min}} \tag{2.19}$$

Se calcularon coeficientes de correlación en distintas combinaciónes respecto a la normalización de variables y se seleccionó aquella que proporcionara la mejor diferenciación entre las categorías; por ejemplo, la desviación estándar es un descriptor que proporciona un peso considerable según su coeficiente de correlación de Pearson al normalizarla, sin embargo, normalizándola este coeficiente disminuye considerablemente; Debido a las condiciones de adquisición de datos planteadas para esta investigación, descriptores estadísticos como la media no fueron normalizados.

Todos los descriptores espectrales fueron sometidos a tratamientos estadísticos que permitieran calcular coeficientes de correlación: media, valor mínimo, valor máximo y desviación estándar de cada descriptor podrían proporcionar una característica útil para la construcción del dataset final. Las figuras 2.9, 2.10, 2.11 presentan los resultados de correlación obtenidos en cada uno de los tres grupos de control.

La construcción del dataset preliminar resulta en una matriz de dimensiones 43x24, donde las dos primeras columnas representan el nombre del archivo y su clase.

A partir de las correlaciones se establecieron umbrales para considerar características dentro del grupo final. En cualquier caso, se sostuvo que $|r| \geq 0.5$ como una de las estrategias para

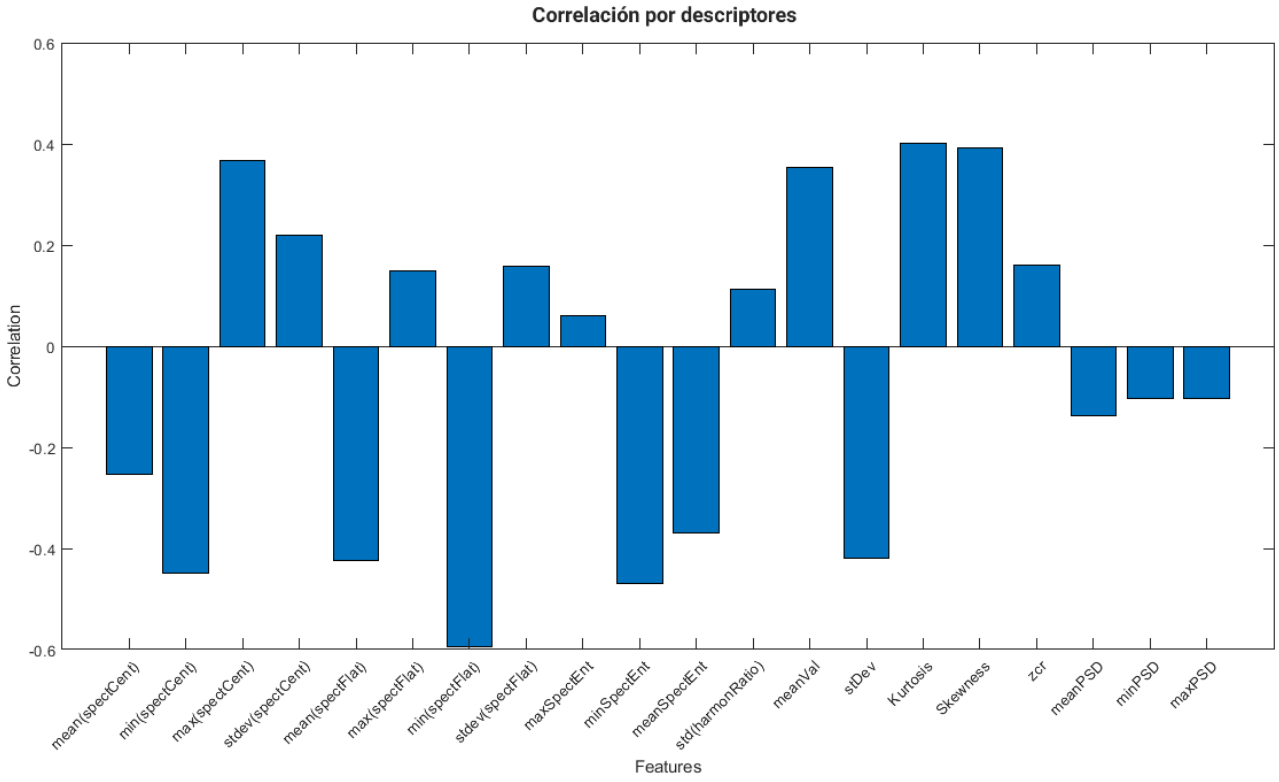

Figura 2.9: Gráfico comparativo de correlaciones para el grupo H-F

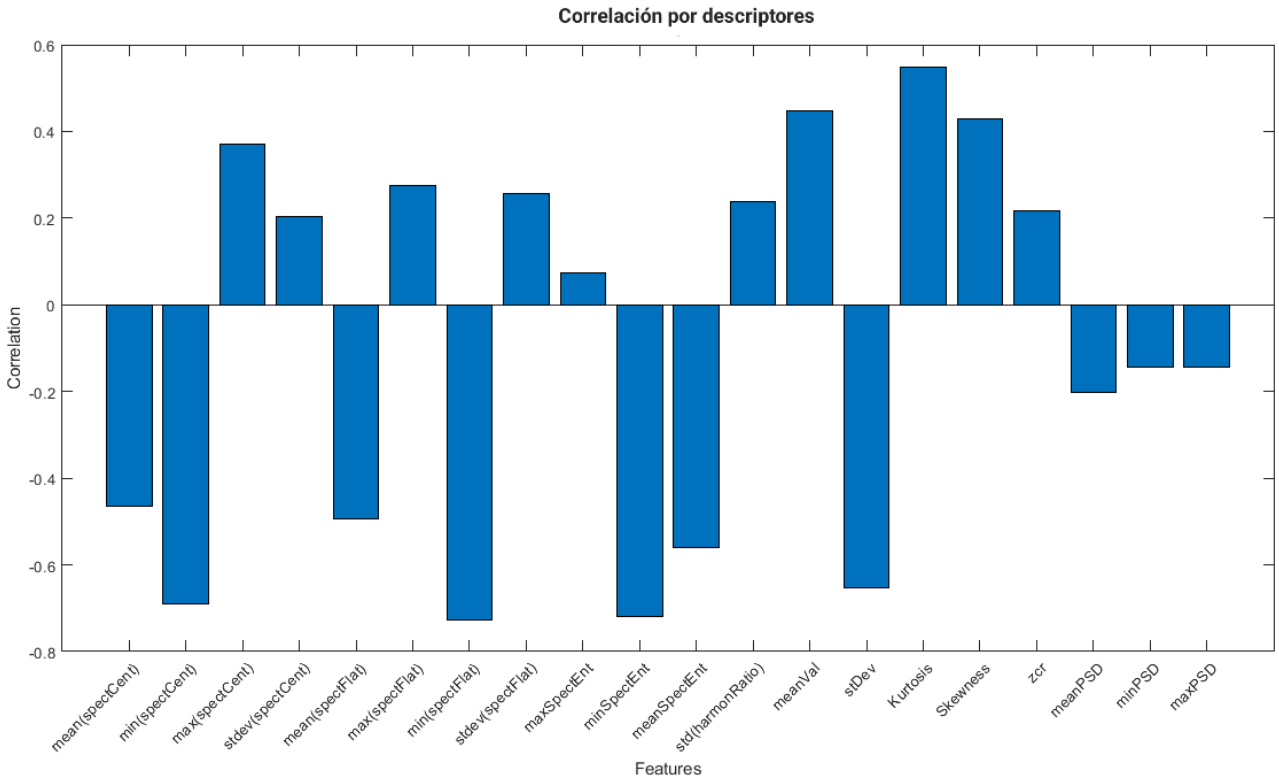

Figura 2.10: Gráfico comparativo de correlaciones para el grupo H-P

evitar la maldición de la dimensionalidad. A partir de esto, se determinó que en los tres grupos de control se seleccionan como características el mínimo y máximo del centroide espectral, el mínimo de la planura espectral, la entropía espectral mínima y media, la desviación estándar y la curtosis.

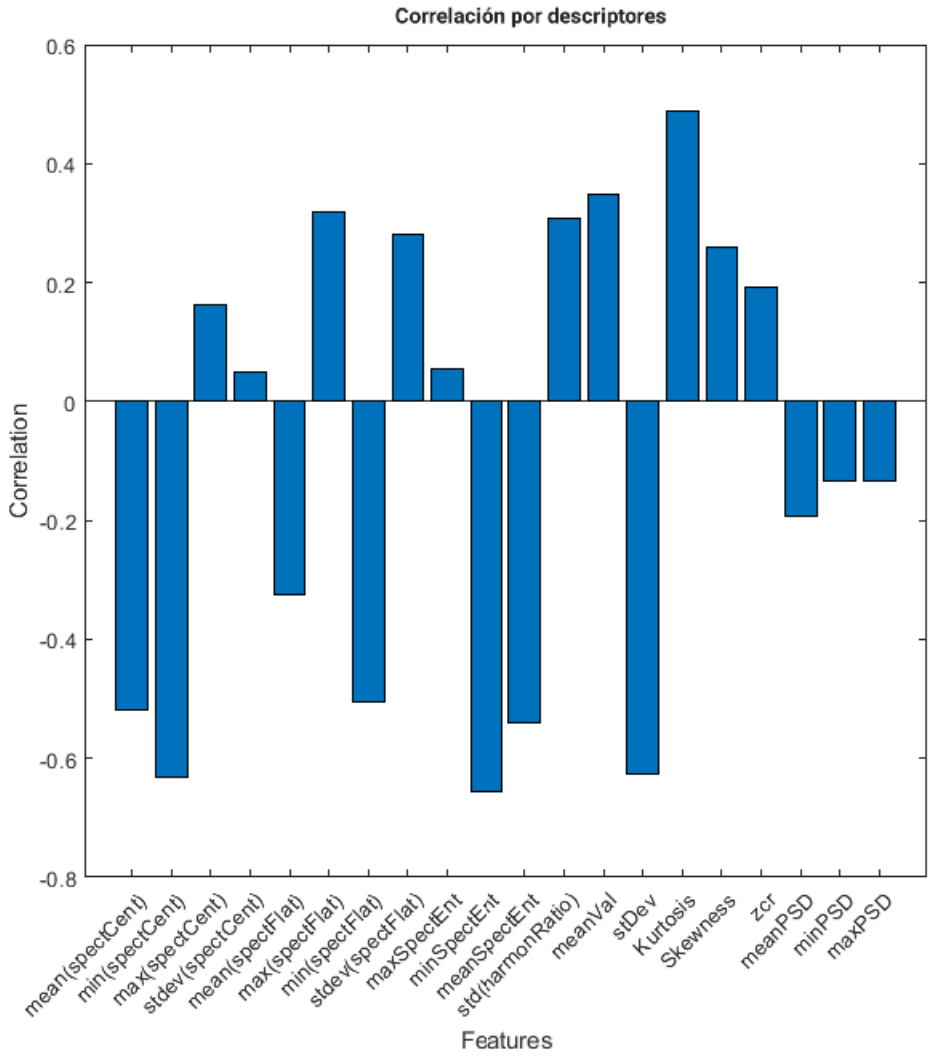

Figura 2.11: Gráfico comparativo de correlaciones para el grupo HPB

Una vez realizado el análisis de correlación se reduce la dimensionalidad del dataset preliminar a 43×8, con las siete características seleccionadas y se puede continuar con la etapa de selección y entrenamiento de un modelo de aprendizaje automático.

2.5. Entrenamiento de los modelos predictivos

Hasta ahora la descripción de la metodología para extracción de características se ha descrito de manera exclusiva en MATLAB, pero también resultará interesante mostrar de forma paralela el desarrollo del entrenamiento de modelos predictivos en Python. Antes de proceder con la descripción de los modelos a entrenar, es importante explicar su importación a Python. Para este fin se requiere el uso de dos librerías que deben instalarse mediante la linea de comandos: SciPy y Pandas. Estas librerías no solo se necesitan para la transformación de datos sino para realizar predicciones nuevas.

Entonces, en MATLAB se exporta un archivo *.mat* conteniendo el dataset de entrenamiento, mientras que en un script de Python se utiliza el método *loadmat* de la siguiente forma:

```
from scipy import io
import pandas as pd
...
mat_contents = io.loadmat('dataset.mat')

df = pd.DataFrame(newDS,
columns = ['Class','minSpectCent','maxSpectCent','minSpectFlat',
'minSpectEnt','meanSpectEnt','StDev','Kurtosis'])

df['Class'] = df['Class'].astype('int')
```

La fracción de código anterior convierte el dataset en un dataframe de Pandas, de forma que será comprensible para los métodos de entrenamiento posteriores. El conjunto de datos fue guardado en formato *.mat* en lugar del más popular y estandarizado archivo de valores separados por coma, *.csv*, pues en las etapas más tempranas del desarrollo de esta investigación MATLAB era el entorno de trabajo exclusivo.

El capítulo 1 estableció los tipos de aprendizaje automático en relación con el tipo de variables involucradas en el entrenamiento y en razón de la salida que se desea obtener del modelo. Esta investigación se involucra exclusivamente con el aprendizaje supervisado, y al tener una variable de respuesta de naturaleza discreta se estudiarán algoritmos de clasificación. En una etapa posterior del documento se hará una breve revisión sobre las posibilidades de aplicar metodologías de regresión en problemáticas similares. Igualmente, el capítulo 1 hace

una breve introducción a los métodos de clasificación, sin embargo, es importante recordar aspectos puntuales sobre esta técnica de aprendizaje supervisado.

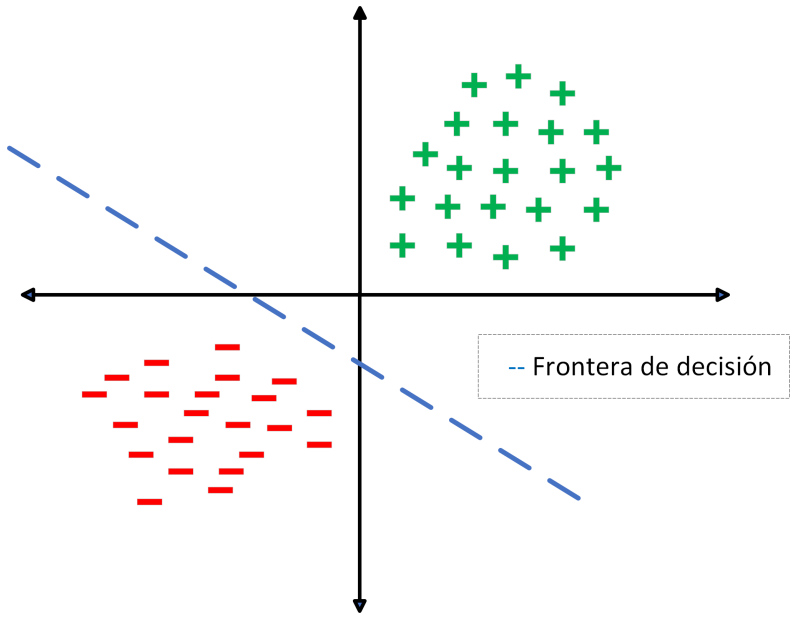

Figura 2.12: La frontera de decisión es una de las técnicas utilizadas en clasificación para la separación binaria.

Las técnicas de clasificación predicen respuestas discretas y categorizan los datos; la figura 2.12 muestra una de las formas en la que esta categorización se hace en algunos algoritmos de clasificación. La selección de un algoritmo de aprendizaje automático es un proceso principalmente de prueba y error, pero que también puede basarse en la comprensión de naturaleza matemática de los modelos. Además, es importante que la selección de un algoritmo específico cumplan un balance entre cuatro aspectos fundamentales: la velocidad de entrenamiento, el uso de memoria, la precisión al predecir sobre datos nuevos y la transparencia e interpretabilidad, es decir, la facilidad para entender por qué un algoritmo asigna determinada categoría a un conjunto de datos. Otro criterio que debe tenerse en cuenta al elegir el algoritmo a implementar es el número de categorías que se espera obtener como salida, esto es clasificación binaria y multiclase. Este último aspecto es particularmente importante en esta investigación considerando los grupos de control que conforman el conjunto de datos; para dos de los grupos de control es importante pues solo se requiere clasificación binaria, sin embargo para el grupo HPB se necesita un algoritmo multiclase.

A partir de estas condiciones y de los resultados de la revisión sistemática del capítulo 1 se establecen tres algoritmos como objetivo de entrenamiento: k-Vecinos Más Cercanos (*k-Nearest Neighbors*, kNN), árboles de decisión (*Decision Trees*, DT) y Máquinas de Soporte

Vectorial (*Support Vector Machines*, SVM). Profundización matemática y los casos de uso de estos algoritmos de clasificación se provee a continuación.

2.5.1. k-Vecinos más Cercanos - kNN

Muchas técnicas de aprendizaje automático están basadas la generalización de los datos, llevando a que el resultado se fundamente en el parecido que tengan datos nuevos con aquellos que entrenaron el modelo. El aprendizaje basado en vecinos cercanos es el más conocido de estos modelos de tipo *lazy learning*. Cuando un modelo kNN (siglas del inglés *k-Nearest Neighbors*) es presentado con nueva información utiliza una técnica específica de medición de distancia para asociarla a una categoría específica. La medición de distancia depende explícitamente del modelo; la distancia Minkowski de orden $p > 0$ se define en la ecuación 2.20.

Si $X = \mathbb{R}^d$

$$Dis_p(x, y) = (\sum_{j=1}^{d} |x_j - y_j|^p)^{\frac{1}{p}} = ||x - y||_p \tag{2.20}$$

Donde $||z||_p = (\sum_{j=1}^{d} |z|^p)^{\frac{1}{p}}$ es la norma p del vector z. Entonces, se define asímismo a la conocida distancia Euclidiana como el caso particular para p = 2 en la ecuación 2.21.

$$Dis_2(x, y) = \sqrt{\sum_{j=1}^{d} (x_j - y_j)^2} = \sqrt{(x - y)^T (x - y)} \tag{2.21}$$

Los modelos basados en distancia se formulan en términos de un número de ejemplares, y las reglas de decisión se definen en términos de los ejemplares más cercanos o *vecinos*. Los ejemplares pueden definirse como centroides que encuentran un centro de masa de acuerdo con la métrica de distancia elegida, o medoides que encuentren el punto de datos más localmente centrado. En el caso de $k = 1$, este clasificador se llama simplemente el *vecino cercano*, cuya fundamentación es sencilla: memorizar el conjunto de datos completo y establecer una frontera de decisión. El clasificador NN sugiere un alto riesgo de overfitting si los datos son limitados, ruidosos o poco representativos. En contraposición, el clasificador de Nearest Neighbor es algorítmicamente rápido de entrenar: el tiempo de entrenamiento es del orden $O(n)$ para almacenar n ejemplares, pero clasificar una sola instancia también toma O(n) tiempo, pues esta debe compararse individualmente con cada ejemplar de entrenamiento para determinar su cercanía a alguno. Además, otra desventaja del clasificador NN es su propensión a la maldición de la dimensionalidad.

El clasificador kNN es una extensión en la que se toman "*votaciones*" entre los $k \geq 1$ ejemplares más cercanos de la instancia a clasificar y la salida predicha es aquella mayoritaria. Aunque resulte contraintuitivo, un mayor número de vecinos no necesariamente implica una mejor predicción: el refinamiento del algoritmo aumenta con k pero luego disminuye, o viéndolo de una forma más técnica, un k pequeño aumenta la sensibilidad al ruido mientras que un k muy grande difumina las fronteras de decisión.

A partir de lo descrito anteriormente, se pueden establecer algunas ventajas y desventajas del clasificador de k-Nearest Neighbors:

- Ventajas:
  - Sencillo y rápido de implementar
  - Efectivo ante fronteras de decisión no muy compejas.
- Desventajas:
  - Computacionalmente costoso, en particular con datasets grandes.
  - Requiere una adecuada estimación del número de vecinos y de la técnica de medición de distancia.

En MATLAB®, la función *fitcknn* permite entrenar un modelo kNN. La función hace parte del Statistics and Machine Learning Toolbox y en principio recibe dos argumentos: el primero es una matriz X con los datos de muestra cuyas filas corresponden las observaciones y columnas representan las características o predictores, mientras que el segundo argumento representa los vectores de salida Y que representan las categorías de clasificación. Algunos argumentos opcionales que puede recibir *fitcknn* son $'Distance'$ para personalizar una métrica de distancia y $'NumNeighbors'$ para establecer el número de vecinos a considerar. Entonces, como ejemplo, la función kNN puede escribirse como:

```
Mdl = fitcknn(X,Y,'Distance','euclidian','NumNeighbors',3)
```

La salida, *Mdl*, es un modelo que ante la función *predict* retorna una clasificación, por ejemplo:

```
label = predict(Mdl,X)
```

Donde *label* es la predicción asignada por el modelo, Mdl al indicarle una matriz de características X.

En Python, se necesita instalar la librería **scikit-learn** mediante la línea de comandos:

```
pip install sklearn
```

Entonces, se puede entrenar un modelo kNN importando el siguiente método:

```
from sklearn.neighbors import KNeighborsClassifier
```

Y se utiliza la función *KNeighborsClassifier*, que recibe como argumentos un número de vecinos de la siguiente forma:

```
knn = KNeighborsClassifier(n_neigh)\\
knn.fit(X_train,y_train)
```

El método *fit* del modelo *knn* entrena el modelo con los datos de entrada y salida, X_{train} y y_{train} respectivamente.

En este momento es imperativo explicar la selección adecuada del número k de vecinos: esta se hace de manera recursiva; se sabe que el número de vecinos seleccionados impactará de manera positiva o negativa al modelo, así que se selecciona un número mínimo y máximo de vecinos a considerar y una métrica de evaluación. Las métricas de evaluación para algoritmos de clasificación se profundizarán en la sección 2.3.4, sin embargo, es imprescindible introducir una de ellas en este momento: la exactitud o *accuracy* es la métrica más directa respecto al rendimiento de un modelo de clasificación, y se utiliza ampliamente como una guía en la selección correcta del número de vecinos. La ecuación 2.22 define el accuracy como un valor entre 0 y 1.

$$Accuracy = \frac{\#aciertos}{\#predicciones} \tag{2.22}$$

La exactitud como métrica de selección de vecinos es rápida pero puede no ser indicada si existe un considerable desbalance entre las clases.

Otra forma de hacer la selección de vecinos es la validación cruzada, que divide el dataset de entrenamiento en varios pliegues y hace subdatasets de entrenamiento y evaluación, prueba con distinto número de vecinos y les asigna un puntaje.

Ambas, la validación cruzada y la técnica de exactitud, se hacen a partir de un bucle for, asignando los puntajes de cada métrica a una posición en un arreglo numérico y encontrando el índice de dicho arreglo que contenga la mayor puntuación; dicho índice representa el número óptimo de vecinos.

La necesidad de un algoritmo recursivo para la selección de vecinos también podría representar una desventaja del algoritmo kNN pues requiere el entrenamiento de modelos con distinto número de vecinos para la selección óptima, aumentando así el tiempo y el costo computacional.

2.5.2. Árboles de Decisión - DT

Muchos algoritmos de aprendizaje automático pueden utilizarse en aplicaciones de clasificación y regresión; el algoritmo de k-Nearest Neighbors es un ejemplo, pero también están entre ellos los árboles de decisión, *Decision Trees, DT*; a diferencia del método kNN, los DT no memorizan el dataset y trazan una frontera de decisión para clasificar, sino que dividen el dataset en subconjuntos dependiendo de un criterio de decisión. Esto es, una división binaria recursiva que termina generando la estructura de árbol que se puede observar en la figura 2.13.

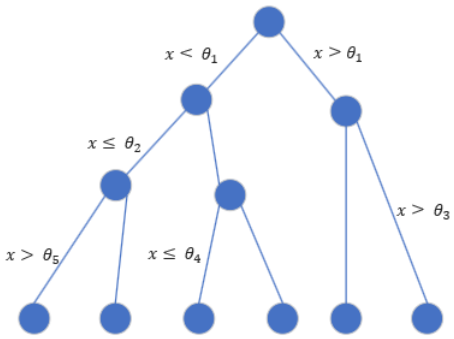

Figura 2.13: La clasificación en un árbol de decisión se hace al comparar la información nueva con unos pesos establecidos.

En primera instancia, el espacio de características x se divide en dos regiones según se compara con los parámetros θ del modelo, entonces si $x_1 > \theta_1$ o $x_1 \leq \theta_1$ se crean subregiones que comparan si $x_2 > \theta_2$ o $x_2 \leq \theta_2$. A partir de esto, y una vez se alcance un criterio de decisión cada subregión del modelo se asocia a una clase.

Entre los criterios que pueden utilizarse para separar una región de otra se encuentra la entropía, obteniendo su nombre de la variable termodinámica con la que puede hacerse una

analogía: es una medición del desorden o impureza de un conjunto de datos, y se en los árboles de decisión representa una medida de homogeneidad de las muestras en un nodo. La entropía del subconjunto S está descrita matemáticamente por la ecuación 2.23, donde P_i representa la proporción de datos de la clase i. La ecuación 2.24 explica el cálculo de la entropía en casos de clasificación binaria ($i = 2$). Una entropía de 0 representa homogeneidad total del conjunto de datos, mientras que una tendencia a 1 representa desorden total.

$$Entropia(S) = \sum_{i=1}^{n}(-P_i \log_2(P_i)) \tag{2.23}$$

$$Entropia(S) = -P_1 \log_2(P_1) - P_2 \log_2(P_2) \tag{2.24}$$

La entropía tiene asociada otra métrica con la que guarda una relación inversa: la ganancia de información. Esta métrica se utiliza para decidir qué característica se divide en determinado nodo, es decir, qué tanta información puede proporcionar para obtener una mejor separación de los datos. La ganancia de información entonces depende de la entropía del subconjunto actual y de la característica a dividir; la ecuación 2.25 define matemáticamente esta métrica.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} * Entropy(S_v) \tag{2.25}$$

La ganancia depende entonces de la entropía del subconjunto de datos actual, la fracción $\frac{|S_v|}{S}$ donde el subconjunto S_v es aquel en que la característica A tiene el valor v entre todos los valores posibles que puede tomar. La característica con mayor ganancia de información se selecciona como el siguiente criterio de división en el árbol.

Otro criterio importante es la impureza Gini, cuyo valor es una representación de lo ordenado que está un subconjunto de datos. Esta métrica se calcula según las proporciones de cada clase en el dataset, según se escribe en la ecuación 2.26, donde P_i representa la proporción de la clase i en el subconjunto de datos.

$$Gini = 1 - \sum_{i=1}^{n} P_i^2 \tag{2.26}$$

Un hiperparámetro importante en la implementación de un algoritmo DT es la profundidad máxima, que se refiere al nivel máximo de crecimiento que puede alcanzar un árbol de decisión; es importante seleccionar una profundidad máxima como estrategia para evitar

el overfitting, maximizar la eficiencia computacional y disminuir el acercamiento a outliers del conjunto de entrenamiento. La selección de una profundidad máxima puede hacerse de manera similar a la selección de un número de vecinos en un modelo kNN: a partir de validación cruzada o de otra métrica de evaluación.

El estudio de los árboles de decisión permite describir ventajas y desventajas sobre este clasificador:

- Ventajas
  - Fácilmente interpretable
  - Sistema de pesos útil para la selección de características
  - Minimización del costo computacional
- Desventajas
  - Tendencia al overfitting
  - Sensibilidad a cambios en el dataset
  - Rendimiento bajo en comparación a otros modelos.

En MATLAB, la función `fitctree` entrena un modelo de clasificación de Decision Tree; al igual que en el entrenamiento de modelos kNN, se requiere la instalación del Statistics and Machine Learning Toolbox. La función recibe dos argumentos obligatorios: la matriz de características X y los datos de salida Y; entre sus argumentos opcionales está `'CrossVal'` que permite hacer validación cruzada como forma de seleccionar la profundidad máxima, `'MaxDepth` que seguido de un entero establece la profundidad máxima del árbol y `MaxNumSplits` para el máximo número de divisiones. La función `fitctree` se puede escribir entonces como

```
Mdl = fitctree(X,Y,'CrossVal','on','MaxNumSplits',5)
```

De igual forma, la salida *Mdl* se puede someter a su propio método *predict* para hacer predicciones ante información nueva. En Python se entrena un modelo de árbol de decisión haciendo uso de `scikit-learn` de la siguiente forma:

```
from sklearn.tree import DecisionTreeClassifier
...
dtModel = DecisionTreeClassifier(random_state=42)
dtModel.fit(X_train,y_train)
```

Como se ha descrito previamente, el método *fit* entrena el modelo según los datos de entrenamiento X_train y y_train. La línea `random_state=42` controla la aleatoreidad del estimador, y recibe otros hiperparámetros como el criterio, cuyos valores pueden ser `'entropy'`, `'log_loss'` o la opción por defecto, `'gini'`, o la profundidad máxima.

2.5.3. Máquinas de Soporte Vectorial - SVM

Entre los algoritmos más poderosos de aprendizaje automática están las Máquinas de Soporte Vectorial, *Support Vector Machines, SVM*. El primer aspecto importante de los modelos SVM es que también se fundamentan en una frontera de decisión, aquí llamadas *hiperplanos*, pero en comparación con técnicas como kNN dicha frontera de decisión es la solución a un problema de optimización, así que en comparación con los árboles de decisión, la categorización se hace de manera global en lugar de localmente. El hiperplano seleccionado es uno en el que el márgen se maximiza, de forma que la separación de clases en un espacio de características se hace más efectiva. Las máquinas de soporte vectorial tienden a mostrar mejores resultados en problemas de clasificación binaria, en cuyo caso la clasificación responde a una función signo, como se muestra en la ecuación 2.27, donde $\vec{w} \cdot \vec{x}$ representa el producto entre cada característica x y su peso w, y b el *bias* del modelo.

$$y = \text{sign}(\vec{w} \cdot \vec{x} + b) \tag{2.27}$$

El problema de optimización que permite alcanzar el máximo margen posible asumiendo que no se permita ninguna clasificación errónea es encontrar el mínimo valor de los pesos que satisfaga la ecuación 2.28

$$\text{mín}\left(\frac{1}{2}\|w\|\right) : y_n(\vec{w} \cdot \vec{x} + b) \geq 1 \quad \forall n \tag{2.28}$$

Los soportes vectoriales son aquellos datos que están más cerca al hiperplano e influencian fuertemente su orientación y posicionamiento. La separabilidad lineal de los datos es relevante, pues el algoritmo SVM utiliza aquellos datos linealmente no separables como una forma de penalización; sin embargo, los datos no separables linealmente también son manejables para las máquinas de soporte vectorial, pues permite el uso de kernels que permitan llevar dichos datos a un espacio de alta dimensionalidad que permita encontrar una frontera de decisión lineal. Al introducir un kernel, la decisión planteada en la ecuación 2.27 se transforma en la ecuación

$$y(x) = \text{sign}\left(\sum_{n=1}^{N} a_n t_n k(x, x_n) + b\right) \tag{2.29}$$

Donde a_n representa los multiplicadores de Lagrange, t_n la clase conocida y $k(x, x_n)$ el kernel a utilizar. A continuación se describen algunos de los kérnels más comunes en aplicaciones de SVM.

El kernel lineal es el más sencillo de todos, asímismo su utilidad está en datos linealmente separables; matemáticamente solo representa un producto escalar entre dos vectores de características, definido en la ecuación 2.30

$$k(x_i, x_j) = \vec{x_i} \cdot \vec{x_j} \tag{2.30}$$

Luego está el kernel polinomial, que puede capturar mejor la relación entre los datos al resolver un polinomio de grado d; la ecuación 2.31 describe matemáticamente este kérnel; los hiperparámetros r y d deben ser elegidos cuidadosamente para evitar el overfitting.

$$K(x_i, x_j) = (\vec{x_i} \cdot \vec{x_j} + r)^d \tag{2.31}$$

A partir de aquí se introducen kérnels más complejos, que si bien pueden entender mejor muchas relaciones complejas entre los datos comparten la particularidad de ser muy sensibles a los hiperparámetros con los que se sintonicen, estos son el kernel radio-basis-function (rbf), descrito en la ecuación 2.32 y el kernel sigmoide, definido en la ecuación 2.33

$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}} \tag{2.32}$$

$$K(x_i, x_j) = \tanh(\alpha(\vec{x_i} \cdot \vec{x_j}) + \beta) \tag{2.33}$$

Aunque su fuerte sea la clasificación binaria, los modelos SVM también pueden entrenarse en clasificación multiclase mediante ECOC, *Error Correcting Output Codes*; el ECOC crea clasificadores binarios y los cruza entre sí para *simular* una clasificación multiclase. Si hay un desbalance entre las clases, el ECOC puede ser un enfoque robusto para las máquinas de soporte vectorial. Otra metodología es One-vs-One (OvO), donde se seleccionan pares de clases para generar clasificadores binarios y luego la clasificación mayoritaria se asigna como la final.

A partir de lo anterior se pueden establecer algunas consideraciones en el uso de modelos SVM en esta investigación; dos de los grupos de control presentan clasificación binaria, pero el grupo HBF es clasificación multiclase, por lo que requerirá el uso de alguna estrategia ECOC u OvO para poder implementar este algoritmo. Se recogen a continuación ventajas y desventajas de las máquinas de soporte vectorial.

- Ventajas
  - Fuerte generalización al solucionar un problema de optimización
  - Posibilidad de trabajar con datos no separables linealmente usando kérnels.
  - Soluciones específicas para el manejo de múltiples clases.
- Desventajas
  - Dependen fuertemente de la correcta selección de hiperparámetros
  - Matemáticamente complejos y de baja interpretabilidad humana.
  - El almacenamiento de los vectores y el uso de kérnels aumenta considerablemente el costo computacional.

En MATLAB existen dos funciones para entrenar modelos SVM, dependiendo de si la clasificación es binaria o multiclase se utilizarán las funciones `fitcsvm` o `fitcecoc` respectivamente. Las funciones reciben los mismos argumentos obligatorios que los casos kNN y DT. Como se puede intuir por su nombre, MATLAB utiliza ECOC como estrategia para los casos multiclase.

```
\\ Para un modelo de clasificación binaria
Mdl = fitcsvm(X,y)
\\ O para clasificación multiclase
Mdl = fitcecoc(X,y)
```

Las funciones fitcecoc y fitcsvm reciben los mismos hiperparámetros, como `Standardize`, una variable booleana que estandariza los predictores numéricos, `KernelFunction` para seleccionar un kernel `rbf`, `polynomial` o `linear`; MATLAB no tiene soporte directo para el kernel sigmoide, pero se puede personalizar.

Para entrenar un modelo SVM en Python se importa el clasificador de la librería `sklearn`. El método SVC es general para clasificación binaria o multiclase, y en tal caso utiliza el método

One-vs-One para entregar una clasificación final. El principal hiperparámetro a controlar en la creación de la instancia es el kernel a utilizar. Por defecto se utiliza el kernel gaussiano, pero se puede seleccionar `linear`, `poly` y `sigmoid` como alternativas.

```
from sklearn import svm

svmModel = svm.SVC(kernel='linear')
svmModel.fit(X_train,y_train)
```

Los modelos de aprendizaje generados en esta sección de la metodología se guardan en disco haciendo uso de la librería joblib. La siguiente fracción de código muestra un ejemplo donde se guarda el árbol de decisión `dtModel` en un archivo `dtModel1.joblib` almacenado en la ruta del script.

```
from joblib import dump

dump(dtModel,'dtModel1.joblib')
```

2.6. Evaluación y mantenimiento de los modelos predictivos

Una vez obtenidos modelos de aprendizaje es imprescindible someterlos a métricas de evaluación que permitan hacer ajustes y mantenimiento a los modelos. En esta sección se describirán algunas de las métricas de evaluación más utilizadas en algoritmos de clasificación.

En la sección anterior se definió la primera de ellas: la exactitud, la más simple de todas y un primer acercamiento al rendimiento de un modelo. La ecuación 2.22 se reescribe a continuación como recordatorio:

$$Accuracy = \frac{\#\, aciertos}{\#\, predicciones}$$

Sin embargo, el accuracy puede malinterpretarse si hay un desbalance entre las clases. Es preciso entonces introducir dos nociones, la primera de notación:

- Positivo: caso asignado a una clase

- Negativo: caso no asignado a una clase

Los positivos y negativos pueden ser verdaderos o falsos, entonces un verdadero positivo y un verdadero negativo representan una clasificación (o en su defecto no clasificación) como acertada. Introducir esta notación sirve como vehículo para la siguiente noción: en problemas de clasificación binaria los positivos y negativos responden a las dos clases, sin embargo, en casos multiclase los positivos se refieren a *correctamente clasificado en esta clase*, o *correctamente no clasificado en esta clase*. Entendiendo esto, la ecuación 2.34 introduce la precisión como siguiente métrica, una razón entre los casos correctamente clasificados como pertenecientes a una clase respecto a todos los datos clasificados en ella.

$$Precision = \frac{Verdaderos\,Positivos}{Verdaderos\,Positivos + Falsos\,Positivos} \tag{2.34}$$

En contraposición a la precisión está el *recall*, definido en la ecuación 2.35; el recall representa la fracción de casos que fueron clasificados en una clase en razón de todos los casos que debieron clasificarse en ella.

$$Recall = \frac{Verdaderos\,Positivos}{Verdaderos\,Positives + Falsos\,Negativos} \tag{2.35}$$

Un balance entre la precisión y el recall se puede percibir mediante el F1-Score, que combina ambas métricas en una media armónica según la ecuación 2.36

$$F1 - score = 2\frac{Precisin \times Recall}{Precisin + Recall} \tag{2.36}$$

La métrica más extendida e interpretable de los algoritmos de clasificación es la matriz de confusión, un arreglo representativo de las clasificaciones positivas y negativas de un modelo predictivo. Su organización en el caso binario se observa en la figura 2.14

La matriz de confusión será la primordial métrica a demostrar en cada modelo generado. En la figura 2.14 las secciones verdes representan las predicciones realizadas correctamente; las filas representan categorías predichas, de forma que la suma de los verdaderos positivos (TP) y los falsos positivos (FP) representan todos los datos que pertenecen a la clase positiva; el mismo razonamiento se aplica para la clase negativa.

Las métricas de evaluación se obtienen presentando al modelo con nueva información, el subconjunto de prueba descrito al inicio de la sección 2.3. Una vez se obtienen las métricas descritas en esta sección, estas deben ser útiles para realizar modificaciones a los hiperparámetros descritos en cada modelo; por ejemplo, aunque el kérnel de un modelo SVM puede

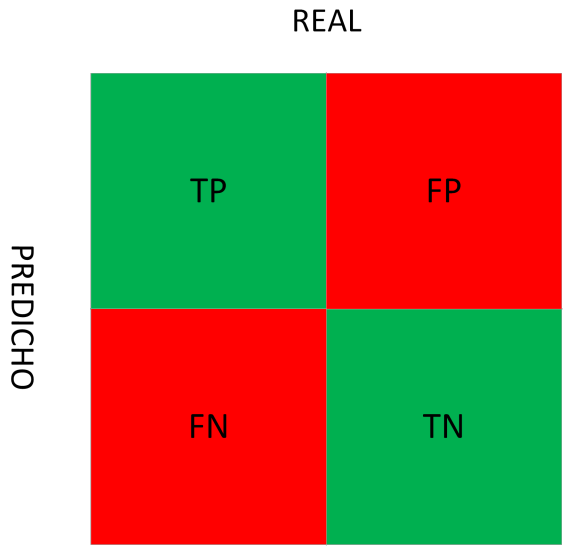

Figura 2.14: Matriz de confusión en clasificación binaria

seleccionarse entendiendo las propiedades del conjunto de datos, no es una práctica inusual entrenar distintos modelos SVM variando el kérnel y utilizar las métricas como estrategia de selección para aquel que proporcione el mejor ajuste.

Aunque propiamente no se considera mantenimiento del modelo, ajustar el dataset puede ser una estrategia que mejore las métricas de evaluación; por ejemplo, se pueden obtener los pesos del modelo predictivo para determinar la importancia de características o reconsiderar un procesamiento específico de las señales. Cuando se ha evaluado el modelo y se hacen los ajustes necesarios para su optimización, es preciso construir una aplicación de despliegue para la utilización final.

2.7. Resumen del capítulo 2

El capítulo 2 introdujo al desarrollo del experimento, la conformación del conjunto de datos, el análisis exploratorio de datos como herramienta para la extracción de características y entendimiento de los datos, el estudio de descriptores estadísticos y espectrales, el estudio y análisis de algoritmos de aprendizaje automático como los k-Nearest Neighbors, Decision Trees y Support Vector Machines, sus métricas de evaluación y técnicas que pueden favorecer al mantenimiento como etapa posterior a la validación de los modelos.

Capítulo 3

Metodología de Despliegue

3.1. Desarrollo del Mínimo Producto Viable en MATLAB.®

MATLAB ofrece a los desarrolladores el diseño de interfaces gráficas de usuario haciendo uso del App Designer; en este proyecto solo se utilizó el MATLAB App Designer como una herramienta de despliegue del Mínimo Producto Viable: la mínima acción que permita el uso de los modelos de entrenamiento. En App Designer, cuya interfaz principal para una aplicación en blanco se observa en la figura 3.1 , se utilizan elementos predefinidos que al presentarse un evento producen una acción programada por detrás mediante las llamadas *Callbacks*.

3.2. Desarrollo de una herramienta IoT para la detección de averías y agendamiento de mantenimiento predictivo

El principal foco de esta investigación es el mantenimiento predictivo, constituido dentro del marco de la Industria 4.0; al desarrollar una herramienta que permita el uso de modelos de aprendizaje automático dentro del PdM puede resultar atractivo construir una aplicación basada en Internet de las Cosas: la interacción entre sistemas físicos y sistemas de la información en la nube que faciliten aún más la implementación de los modelos entrenados.

La construcción de una aplicación IoT puede verse como una secuencia de capas que tienen una representación real:

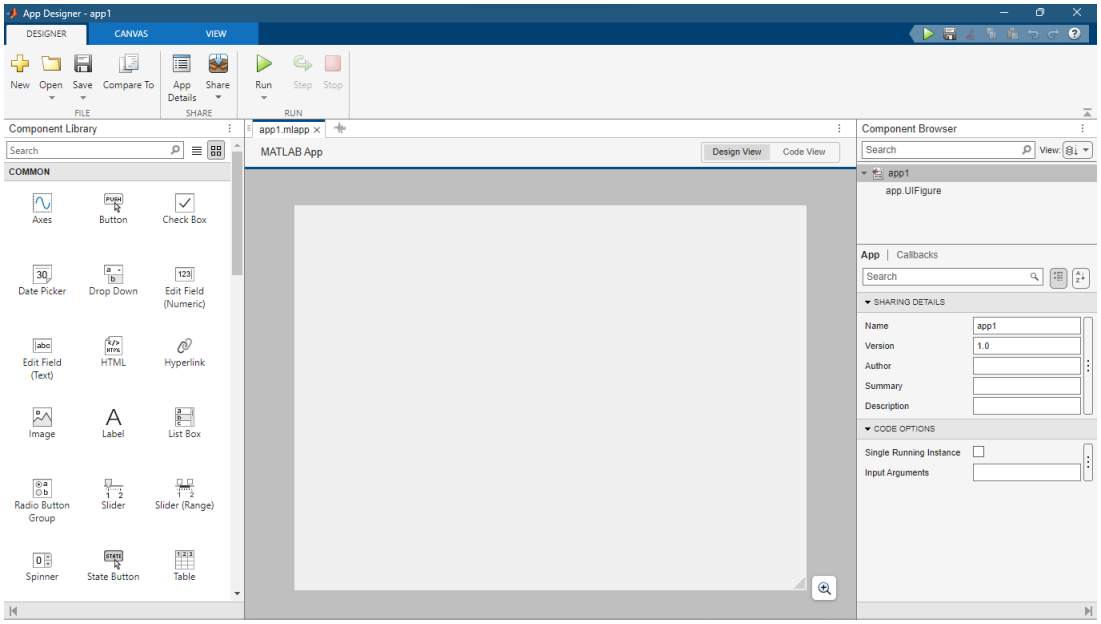

Figura 3.1: Interfaz de diseño del MATLAB App Designer

- Capa de percepción: se refiere a la planta física: los sensores y actuadores que constituyen el sistema a controlar y están asociados mediante un microcontrolador.
- Capa de conectividad: es una capa de transición encargada de llevar los datos digitales del microcontrolador a la nube. Utiliza protocolos como Ethernet, Bluetooth o WiFi para transmitir la información a un servicio en la nube como la Google Cloud Platform, Microsoft Azure o Amazon Web Services.
- Capa de aplicación: la capa más alta de la jerarquía donde se procesa la información obtenida de los servicios de computación en la nube y se entrega al usuario permitiéndole tomar decisiones respecto a la planta física.

El modelo de capas anterior es una simplificación del modelo de interconexión de sistemas abiertos, modelo OSI por sus siglas en inglés. Los componentes de la capa de conectividad se sobreponen entre la capa de percepción y la capa de aplicación. Depende de la segunda capa que las capas externas se comuniquen entre sí.

3.2.1. Capa de Percepción

En el desarrollo de esta capa se siguió un esquema similar al de la sección 2.2 para la adquisición de datos, pero se descartó Blynk como herramienta de control al no ser necesaria

Tabla 3.1: Atributos de la clase 'Motor'

Atributo	Tipo
Pin	integer
Status	boolean
Test	boolean
Fault	integer

la variación manual de la velocidad y al tratarse de un entorno pago que no permite la libertad creativa de la herramienta a diseñar. El microcontrolador ESP32 sigue siendo la herramienta que permitirá controlar los actuadores, que en la etapa de despliegue son dos motores con sus baterías y ESCs. La distancia al micrófono se sostiene igual que en la sección 2.2. El contenido del archivo .ino se puede recuperar en la sección de Anexos.

3.2.2. Capa de Conectividad

Previamente se describió la superposición entre la capa de conectividad y las otras capas del modelo IoT: el microcontrolador tendrá las funcionalidades de acceso a una red WiFi que permita la transmisión de datos con una base de datos en tiempo real de Google Firebase, el servicio en la nube elegido para almacenar la información de la aplicación y el sistema físico. La comunicación entre las capas física y de aplicación puede hacerse sin la intermediación de Google Firebase pero es significativamente más compleja; Google Firebase es ampliamente utilizado en aplicaciones de IoT independientemente de su tamaño por sus herramientas, que incluyen: control en tiempo real (Google Firebase Realtime Database, RTDB), control de accesibilidad y seguridad, acceso mediante API y facilidad para la operación multiplataforma.

Para configurar el acceso de la ESP32 a la base de datos se debe hacer uso de la librería Firebase for ESP32 and ESP8266 [43]. Las herramientas que constituyen las fase de conectividad y de aplicación fueron seleccionadas porque permiten plantear los ejemplares físicos como instancias de una clase. A partir de esto, se instaura la clase Motor como estructura para la base de datos; los atributos de esta clase y sus respectivos tipos de datos se encuentran relacionados en la tabla 3.1.

3.2.3. Capa de Aplicación

El desarrollo de la capa de aplicación se refiere a la etapa que recibe la información del sistema físico, la transforma y permite al usuario final la interacción con los sensores y actuadores

que estén presentes en él. La capa de aplicación requiere la selección de técnicas que logren este objetivo; existen opciones de uso libre y multiplataforma como la mencionada Blynk y la Arduino IoT Platform; aunque estas tecnologías ofrecen facilidad en uso son considerablemente más limitadas que una opción de desarrollo propia. Además, una herramienta propia permite la personalización al ser diseñada explícitamente con un fin en lugar de hacer uso de opciones preexistentes; existen múltiples frameworks que se pueden utilizar para el desarrollo web, pero en esta investigación se hace uso de Django, un framework web de alto nivel para Python que motiva el desarrollo rápido y un diseño limpio y pragmático [44]. Entre las ventajas listadas por sus desarrolladores están:

- Velocidad
- Manejo completo de autenticación, administración, mapeo, feeds, entre otros.
- Seguridad
- Escalabilidad
- Versatilidad

Django provee al desarrollador de muchas herramientas para la construcción completa de una aplicación full-stack. Una aplicación creada en Django se fundamenta en el uso de modelos, una extensión de las clases, eje de la programación orientada a objetos. Las clases tienen atributos y métodos, y son la estructura de la que se construyen instancias (e.i., objetos), y en Django su construcción es, en la práctica, idéntica. En principio, el único modelo necesario para desarrollar la aplicación de despliegue es el mismo que se crea en la capa de conectividad; la declaración de modelos en Django se hace en `models.py`, un archivo generado automáticamente al crear un proyecto nuevo. La declaración de Motor se hace de la siguiente manera:

```
from django.db import models

class Motor(models.Model)
    name = models.CharField(max_length=20,blank=True,null=True)
    pin = models.IntegerField(blank=False,null=True,default=1)
    fault = models.IntegerField(blank=True,null=True)
    status = models.BooleanField(blank=True,null=False,default=True)
    firebaseExists = models.BooleanField(blank=True,null=False,default=False)
```

Se pueden notar al comparar la declaración anterior dos diferencias con la tabla 3.1: la ausencia del atributo 'Test', que se omite al ser transitivo y representar solo el inicio de la prueba, y la presencia de un campo booleano 'firebaseExists', cuya función es revisar la existencia del motor en Google Firebase. Crear un modelo nuevo o modificar uno existente requiere la interacción con la línea de comandos respecto a las llamadas migraciones.

```
python manage.py makemigrations
-
python manage.py migrate
```

Estos comandos introducen a otra ventaja de Django: la directa integración que tiene con bases de datos relacionales. Al ejecutarlos por primera vez, `makemigrations` y `migrate` producen una base de datos en SQLite3; aunque la existencia de los modelos basados en clases es para evitar el contacto directo con SQL, Django también permite la escritura directa de consultas SQL en models.py.

Crear un modelo es solo el primer paso, la interacción del usuario con los modelos es realmente el fuerte; ante esto se crean las vistas, *views.py*, donde se escriben los métodos que hacen consultas HTTP entre el front-end y el back-end. La primera vista que se debe construir es la creación de una instancia motor; la creación de vistas va asociada directamente al desarrollo del front-end; para la creación de instancias debe crearse primero un formulario web. Todo el diseño front-end es facilitado por Django mediante el uso de plantillas, *templates*, una extensión al código HTML que evita la reescritura al introducirle conceptos de programación como los condicionales y búcles, además de permitir el uso de objetos y sus atributos directamente en su escritura. Además, es necesario asociar el front-end (código HTML, CSS y JavaScript generado por plantillas) con el back-end (vistas y modelos) mediante urls.

El corazón de este proyecto Django es el acceso a los modelos de aprendizaje y a la generación de un nuevo vector de características que pueda utilizarse para predecir el estado del motor BLDC. Para esto se utilizan tres métodos en el archivo `views.py`. El primero de ellos es `record_audio()`, que haciendo uso de la librería PyAudio accede a un micrófono USB y graba el sonido del motor durante 10 segundos sampleando a 16 kHz simulando las condiciones del montaje experimental con el que se obtuvo el dataset de entrenamiento.

```
def record_audio():
p = pyaudio.PyAudio()
```

```
    device_index = 1  # Replace with the appropriate device index

    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    input=True,
                    input_device_index=device_index,
                    frames_per_buffer=CHUNK)

    frames = []

    print("Recording...")

    for _ in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
        data = stream.read(CHUNK)
        frames.append(data)

    print("Recording finished.")

    stream.stop_stream()
    stream.close()
    p.terminate()

    audio_data = b''.join(frames)
    return audio_data
```

El segundo método se encarga de hacer el procesamiento y generar el vector de características; a diferencia de MATLAB, Python no cuenta directamente con los métodos matemáticos que se utilizaron para producir la matriz de características original, así que se requiere el uso de dos librerías: NumPy y Librosa; la primera extiende las funcionalidades matemáticas nativas de Python mientras que Librosa tiene como objetivo dotarle de técnicas para procesamiento de audio. Librosa permite calcular el espectrograma, NumPy normalizarlo y extraer características como la entropía espectral. Al final, se produce un vector de características con la siguiente linea de código.

```
features_vector = np.array([
min_spectral_centroid,
max_spectral_centroid,
min_spectral_flatness,
min_spectral_entropy,
mean_spectral_entropy,
std_dev,
kurto
])
```

El último método realmente se compone de dos: un modo manual y un modo automático. Para efectos de metodología se describirá el modo automático: esta vista recibe una ID del motor y se encarga de cambiar el estado en Firebase apagando los demás motores y solo dejando en funcionamiento el motor a probar, abrir la grabación llamando a `record_audio()` y procesar el audio llamando a `preProcessing(audio_array)`. Una vez se obtiene el vector de características, autoMode busca todos los archivos de extensión `.joblib`, que son los modelos de aprendizaje automático guardados previamente, los carga, predice y almacena cada pareja algoritmo-predicción como un key-value de un diccionario. Este diccionario se retorna como un fichero JSON que en el front-end se presenta al usuario como una tabla de resultados. La diferencia entre los métodos autoMode y manualMode es que, en lugar de buscar todos los modelos de aprendizaje, manualMode permite al usuario seleccionar el grupo de control y el algoritmo con el que desea realizar la clasificación.

La figura 3.2 muestra un resumen de las herramientas utilizadas en el desarrollo de la aplicación IoT relacionándolas con la capa a la que contribuyen; operando en conjunto resultan en una aplicación funcional que apoye en el diagnóstico de averías y la programación de mantenimiento

3.3. Resumen del capítulo 3

En este capítulo se introdujeron las metodologías de despliegue a aplicar, es decir, acciones que siendo mínimas o aplicables permiten la implementación e interacción de usuario con los modelos de aprendizaje descritos en el capítulo 2. Se estableció al MATLAB App Designer como el entorno de desarrollo de Interfaz Gráfica de Usuario para el Producto Mínimo Viable (la acción mínima de despliegue), y se introdujo a Django y Google Firebase como

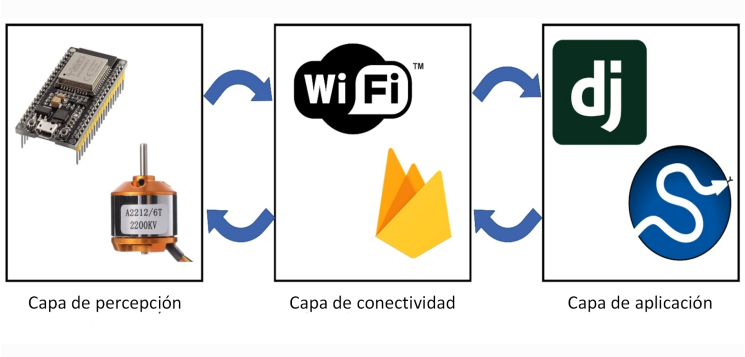

Figura 3.2: Estructura IoT utilizada para el desarrollo de la aplicación, elaboración propia.

herramientas adicionales para la construcción de capas en un sistema de Internet de las Cosas.

Capítulo 4

Resultados y discusión

A partir de la metodología descrita en los capítulos 2 y 3, se comproboó experimentalmente la función de transferencia de un motor c.c., se extrajeron los datos de audio y se construyeron modelos k-Nearest Neighbor, Decision Trees y Support Vector Machines para cada uno de los tres grupos de control (Healthy-Faulty, Healthy-Propeller, Healthy-Propeller-Bearing). En esta sección se analizan los resultados obtenidos en cada modelo de aprendizaje, sus métricas de evaluación y un análisis cualitativo sobre cada uno. Posteriormente, se exponen las aplicaciones de despliegue diseñadas en MATLAB App Designer y Django-Firebase.

4.1. Ensayos de caracterización

En el capítulo 1 se describió al motor c.c. como un excelente vehículo pedagógico para entender sistemas electromecánicos más grandes y de mayor aplicabilidad industrial y se describió matemáticamente; igualmente, en el capítulo 2 se planteó la metodología para la comprobación experimental de su modelo matemático y su función de transferencia. Es necesario entender el funcionamiento de un sistema para poder describirlo y manejarlo con mayor fluidez. Los motores BLDC utilizados para la construcción de los modelos de aprendizaje, aunque matemáticamente similares, ya son dispositivos de mayor aplicabilidad y más complejidad que los motores de corriente continua con escobillas. De igual manera, al haber definido los fallos a estudiar solamente es necesaria la comparación entre ejemplares sanos y ejemplares de la clase Bearing, pues estos sí involucran un fallo interno en el motor asociable a dos características físicas: su resistencia e inductancia de armadura. Haciendo uso de un medidor LCR UT603 se realizaron mediciones en las tres fases de tres ejemplares: dos sanos y uno con falla de bobinado y balinera. Los resultados se recogen en la tabla

Tabla 4.1: Mediciones de armadura en tres ejemplares

	Resistencia ($m\Omega$)	Inductancia (mH)
Motor 1		
Fase A-B	0.2	0.013
Fase B-C	0.23	0.012
Fase A-C	0.2	0.014
Motor 2		
Fase A-B	0.23	0.014
Fase B-C	0.2	0.012
Fase A-C	0.2	0.014
Motor 3		
Fase A-B	0.38	0.036
Fase B-C	0.37	0.034
Fase A-C	0.37	0.030

Las mediciones de características estáticas ya son un resultado revelador: el ejemplar con fallo tiene resistencia e inductancia de armadura mayor a los ejemplares sanos. Esto es lógico teniendo en cuenta que los daños en bobinado representan una dificultad para que el campo magnético fluya adecuadamente a través del circuito y disminuya su velocidad de rotación. Esta misma dificultad en el flujo de campo magnético produce un aumento en la resistencia de armadura del motor.

4.2. Modelos de aprendizaje obtenidos

4.2.1. Grupo de control HF

El grupo HF se compone de 21 archivos de audio .WAV referentes a la clase H representando a los motores sanos y 22 referentes a motores etiquetados como F, motores en fallo. La figura 4.1 es una representación de cada descriptor respecto a su etiqueta en forma de distribución del kérnel. Asímismo, la figura 4.2 muestra diagramas de cajas como una distribución por deciles de los datos de cada categoría.

Al entrenar un modelo kNN para el grupo de control HF con las opciones por defecto de cada lenguaje de programación se obtienen las métricas de evaluación de la tabla 4.2, aunque visualmente podría resultar más sencillo de entender el rendimiento del modelo a partir de su matriz de confusión en la figura 4.3.

Asímismo, el modelo Decision Tree consigue las métricas de evaluación relacionadas en la tabla 4.3, y su matriz de confusión en la figura 4.4.

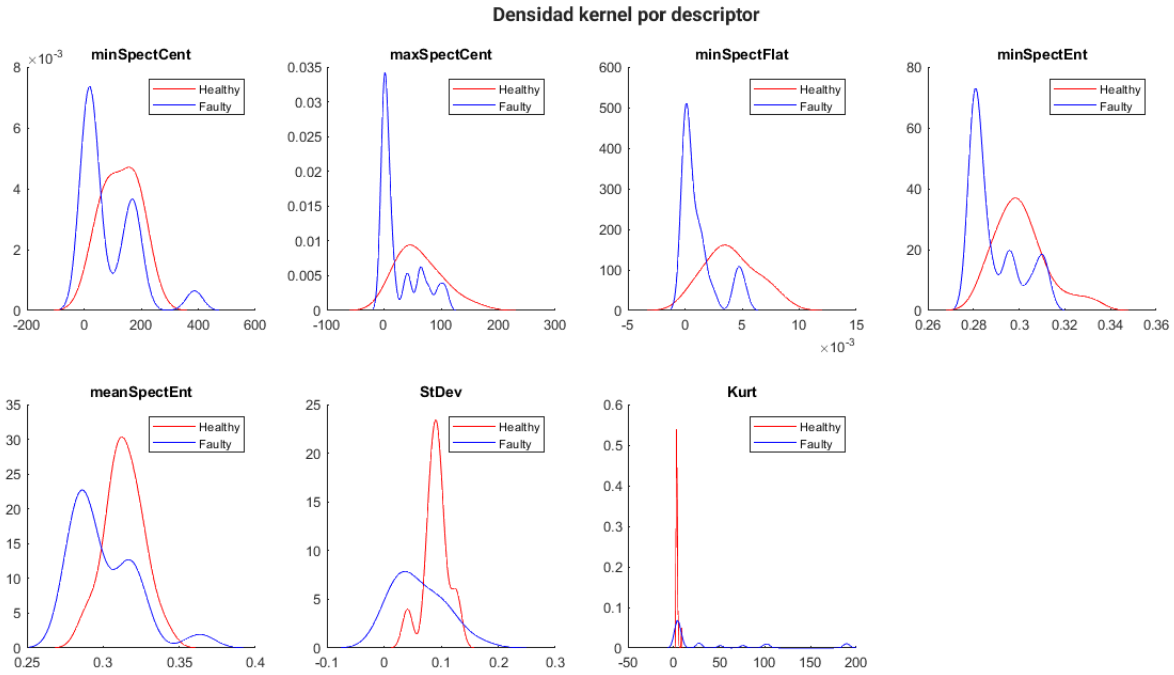

Figura 4.1: Estimación de la densidad del kernel para el grupo HF

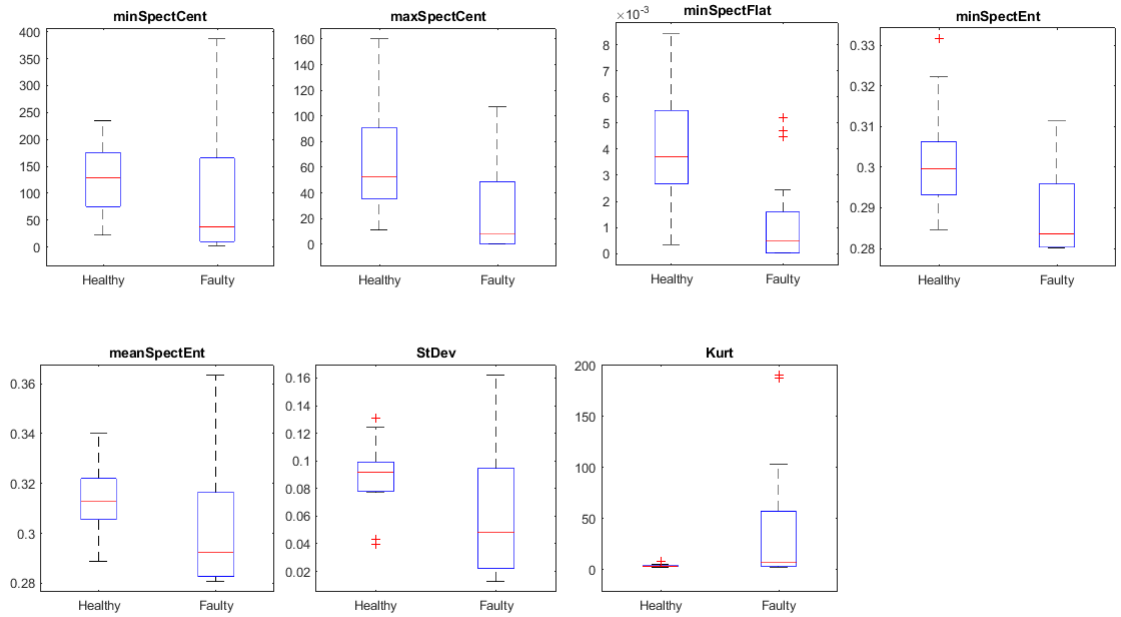

Figura 4.2: Diagramas de caja para los descriptores del grupo HF.

Tabla 4.2: Métricas de evaluación para el modelo kNN del grupo HF.

Métrica	MATLAB (Neighbors=10)	Python (Neighbors=7)
Accuracy	0.84615	0.69230
Precission	 0.7145 1.0000	 0.5700 0.8300
Macro-avg precission	0.8572	0.7000
Recall	 1.0000 0.7500	 0.80000 0.62000
Macro-avg Recall	0.8750	0.71000
F1-Score	 0.8333 0.8571	 0.67000 0.71000
Macro-avg F1-score	0.8452	0.69000

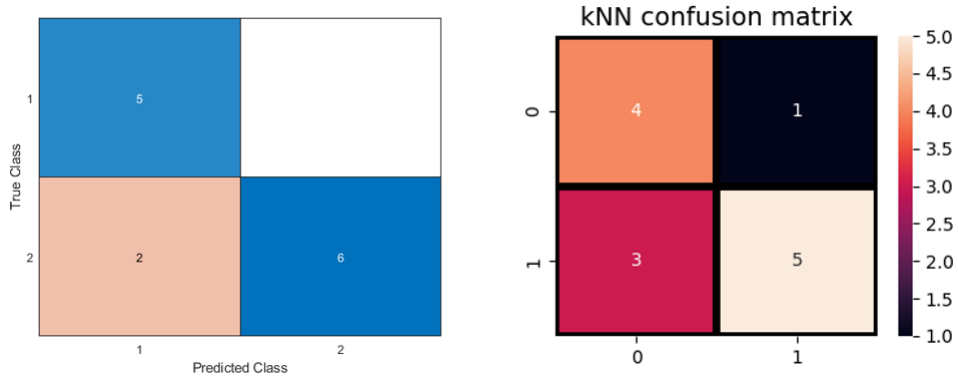

Figura 4.3: Matrices de confusión de los modelos kNN para el grupo HF

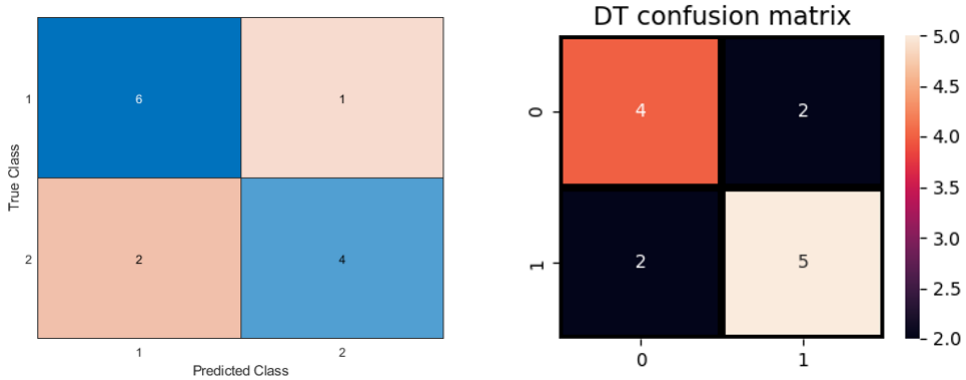

Figura 4.4: Matriz de confusión de los modelo DT para el grupo HF

Tabla 4.3: Métricas de evaluación para el modelo DT del grupo HF.

Métrica	MATLAB	Python
Accuracy	0.76923	0.69230
Precission		
	0.75000	0.67000
	0.80000	0.71000
Macro-avg precission	0.77500	0.69000
Recall		
	0.85710	0.67000
	0.66670	0.71000
Macro-avg Recall	0.76190	0.69000
F1-Score		
	0.80000	0.67000
	0.72730	0.71000
Macro-avg F1-score	0.76365	0.69000

Finalmente, el modelo SVM en MATLAB se entrena con la función `fitcsvm` al tratarse de clasificación binaria, con un kernel lineal, mientras que en Python se entrena con un kérnel radial-basis function (rbf). Para este modelo se obtienen las métricas de evaluación relacionadas en la tabla 4.4, y la matriz de confusión en la figura 4.5.

Tabla 4.4: Métricas de evaluación para el modelo SVM del grupo HF.

Métrica	MATLAB (linear)	Python (rbf)
Accuracy	0.92308	0.84615
Precission		
	1.00000	0.83000
	0.83333	0.86000
Macro-avg precission	0.91665	0.85000
Recall		
	0.87500	0.83000
	1.00000	0.86000
Macro-avg Recall	0.93750	0.85000
F1-Score		
	0.93333	0.83000
	0.90910	0.86000
Macro-avg F1-score	0.92121	0.85000

En el análisis exploratorio de datos, y particularmente en las figuras 4.1 y 4.2 se puede apreciar que las características seleccionadas proporcionan diferenciación efectiva entre las clases. Los resultados de esta etapa comprueban la popularidad de los métodos estadísticos como descriptores, por ejemplo, la kurtosis y la desviación estándar son particularmente

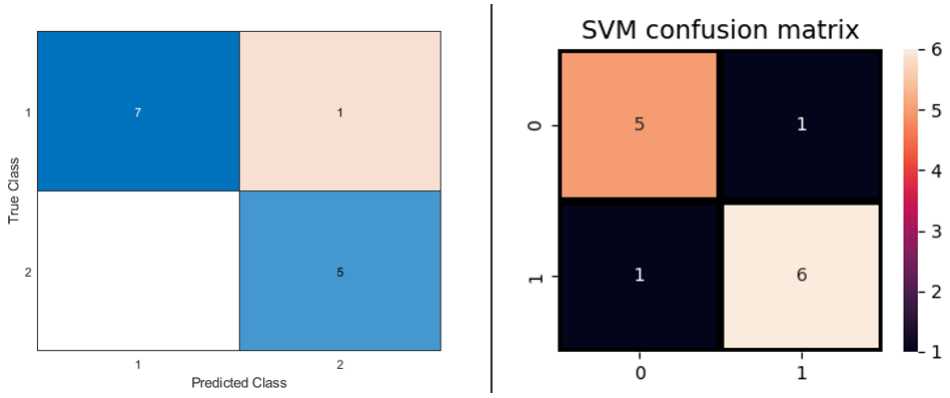

Figura 4.5: Matriz de confusión del modelo SVM en MATLAB

diferenciadoras, sin embargo, también empieza a comprobar experimentalmente la hipótesis de utilizar métodos espectrales como variables de entrada. Esta diferenciabilidad apreciable a simple vista también se mantiene al analizar las métricas de los modelos obtenidos: aunque esta investigación cuenta con un número de observaciones limitado en comparación con otras similares, los modelos mantienen altas métricas sin caer en el overfitting: el modelo con peor desempeño en este grupo de control fue el Decision Tree, con una accuracy del 76.9 %, y esto es congruente con la descripción del algoritmo DT: una de sus desventajas es el bajo rendimiento a comparación de otros modelos, aunque podría intuirse este resultado como una consecuencia de evitar el overfitting, otra de las tendencias que tiene este algoritmo. Por otra parte, el mejor resultado se obtuvo con el modelo SVM, que alcanzó una accuracy del 92.3 %; esto va acorde a la naturaleza del algoritmo pues su fuerte está en la solución del problema de optimización que construye su barrera de decisión. La alta precisión del modelo SVM también puede representar la separación lineal de los datos, pues no requirió de un kérnel más fuerte que el lineal para poder alcanzar sus métricas. El rendimiento del modelo SVM también se debe a que este grupo de control es de clasificación binaria, un punto en el que los modelos SVM no suelen presentar dificultades. Esta misma causalidad puede aplicarse al modelo kNN, que suele ser efectivo ante fronteras de decisión no muy complejas. Entender la naturaleza de los algoritmos como SVM y kNN lleva a intuír una posible dificultad que podrían enfrentar en una clasificación multilclase como el grupo HPB.

4.2.2. Grupo HP

El grupo Healthy-Propeller se compone de 36 observaciones: 22 correspondientes a ejemplares sanos y 14 a ejemplares con fallo en hélice; esto evidencia un desbalance de clases que podría tener repercusiones en las métricas de modelos como las máquinas de soporte vectorial, sin

embargo, es preciso observar los resultados del Análisis Exploratorio de Datos para esta categoría, evidenciar si mejora o empeora la diferenciabilidad entre clases y a partir de las métricas establecer las repercusiones positivas o negativas que tuviera este grupo de control. Con este fin, las figuras 4.6 y 4.7 muestran la estimación de densidad del kernel y los diagramas de caja respectivamente.

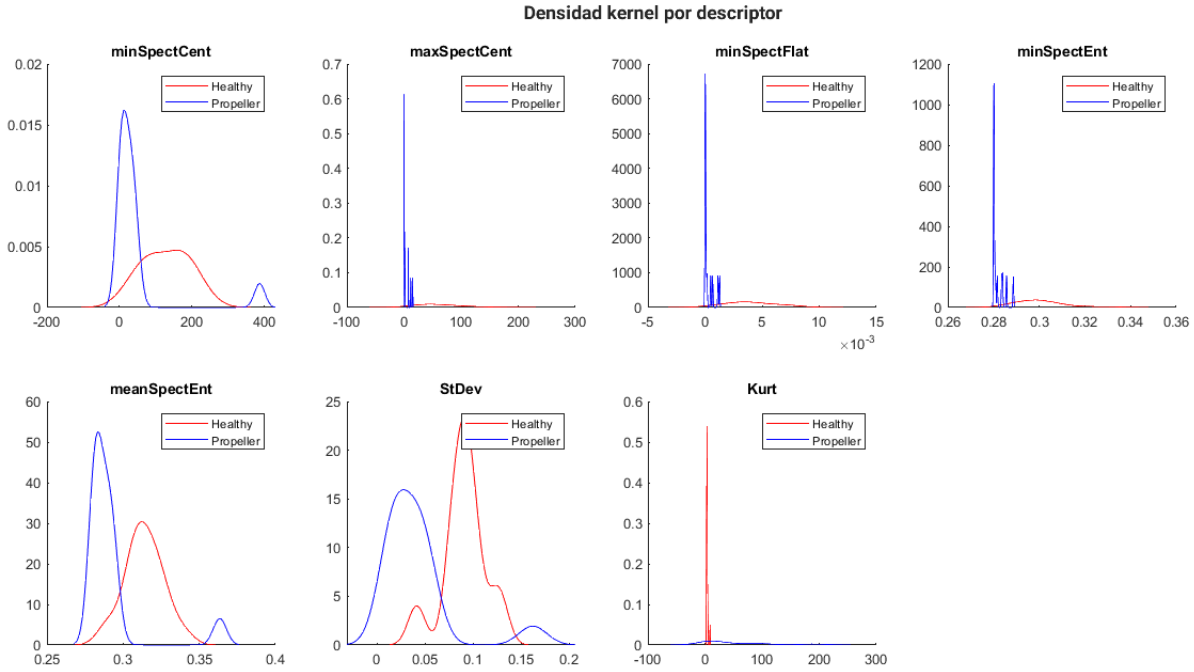

Figura 4.6: Estimación de la densidad del kernel para el grupo HP

Al comparar las distribuciones y los diagramas de cajas del grupo HF y el grupo HP se nota una mejora considerable en la calidad de los descriptores: todas las distribuciones de la clase Propeller presentan su pico en un valor mucho menor respecto a los de la clase de fallos en general; este resultado introduce dos implicaciones: las clases Healthy-Propeller son considerablemente mejor diferenciables que el grupo HF, por lo que se espera una mejora en el rendimiento del modelo Decision Tree ya que podría definir con más rigor los criterios de separación. La otra implicación es que, al ser la clase Faulty un combinado de las clases Propeller y Bearing, la mejora en la diferenciación podría representar una baja diferenciabilidad entre la clase Bearing y la clase Healthy.

Los modelos kNN entrenados para este grupo de control tienen las métricas de evaluación que se relacionen en la tabla 4.5 y las matrices de confusión de las figuras 4.8.

El modelo DT entrenado para el grupo Healthy-Propeller tiene las métricas de evaluación

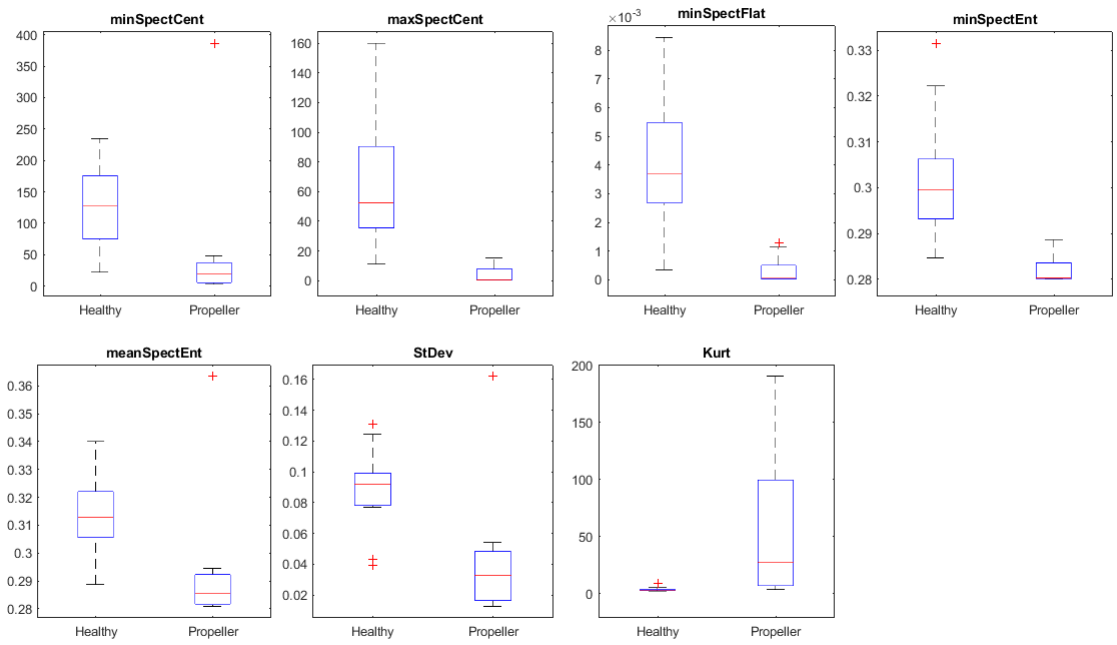

Figura 4.7: Diagramas de caja para los descriptores del grupo HP.

Tabla 4.5: Métricas de evaluación para el modelo kNN del grupo HF.

Métrica	MATLAB (Neighbors=1)	Python (Neighbors=7)
Accuracy	0.90909	0.90909
Precission		
	1.00000	1.00000
	0.75000	0.83000
Macro-avg precission	0.87500	0.92000
Recall		
	0.87500	0.83000
	1.00000	1.00000
Macro-avg Recall	0.93750	0.92000
F1-Score		
	0.93333	0.91000
	0.85710	0.91000
Macro-avg F1-score	0.89521	0.91000

de la tabla 4.6 y la matriz de confusión de la gráfica 4.9.

Ante el grupo de control HP, el modelo DT presenta una mejora de rendimiento del 5 % respecto al grupo HF; esto es congruente con el análisis exploratorio de los datos. La estructura del árbol se presenta en la figura 4.10, y esta muestra que con dos características es suficiente para que el modelo clasifique una entrada nueva como una de las categorías.

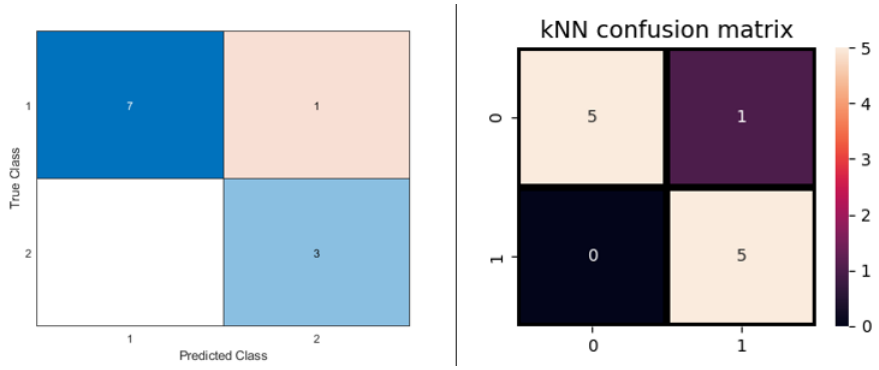

Figura 4.8: Matriz de confusión del modelo kNN del grupo HP en MATLAB.

Tabla 4.6: Métricas de evaluación para el modelo DT del grupo HP.

Métrica	MATLAB	Python
Accuracy	0.81818	0.81818
Precission		
	0.85710	0.75000
	0.75000	1.00000
Macro-avg precission	0.80355	0.88000
Recall		
	0.85710	1.00000
	0.75000	0.60000
Macro-avg Recall	0.80355	0.80000
F1-Score		
	0.85710	0.86000
	0.75000	0.75000
Macro-avg F1-score	0.80355	0.80500

Una clasificación tan rápida muestra que las características que seleccionó el modelo fueron lo suficientemente diferenciables entre categorías.

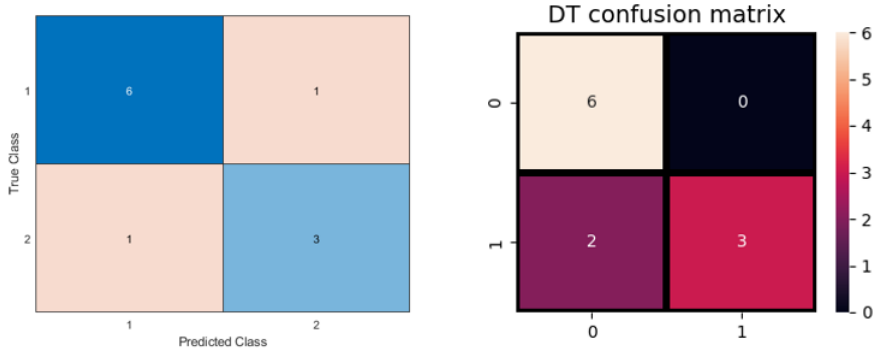

Figura 4.9: Matrices de confusión de los modelos DT para el grupo HP.

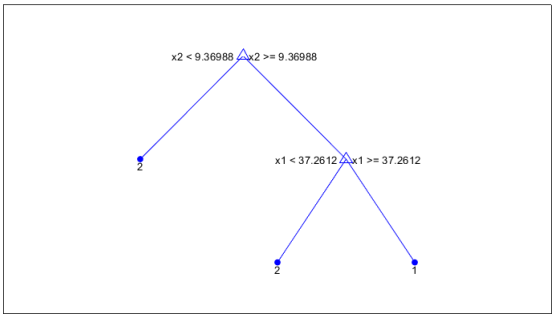

Figura 4.10: Estructura del árbol de decisión para el grupo HP.

Finalmente, el modelo SVM entrenado para el grupo de control obtiene las métricas de evaluación de la tabla 4.7 y la matriz de confusión de la figura 4.11.

Tabla 4.7: Métricas de evaluación para el modelo SVM del grupo HP.

Métrica	MATLAB (linear)	Python
Accuracy	0.81818	0.81818
Precission		
	1.00000	0.83000
	0.60000	0.80000
Macro-avg precission	0.80000	0.81500
Recall		
	0.75000	0.83000
	1.00000	0.80000
Macro-avg Recall	0.87500	0.81500
F1-Score		
	0.85710	0.83000
	0.75000	0.80000
Macro-avg F1-score	0.80355	0.81500

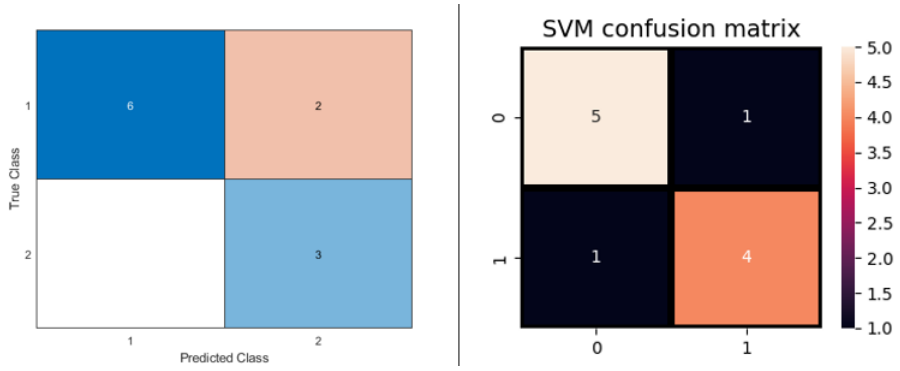

Figura 4.11: Matrices de confusión de los modelos SVM para el grupo HP.

Las métricas del modelo SVM, especialmente su precision y recall indican una fuerte habilidad para generalizar información de la clase Healthy, sin embargo, el desbalance entre las clases efectivamente afecta la confusión respecto a la clase Propeller; esto va en concordancia con las ventajas y desventajas de las máquinas de soporte vectorial pues un desbalance de clases tiende a que el modelo favorezca a la clase mayor. En general, los resultados de este grupo de control presentan una mejoría respecto al grupo anterior. Las características elegidas como descriptores muestran una fuerte separabilidad de los datos.

4.2.3. Grupo HPB

El grupo HPB incluye siete observaciones realizadas en motores con fallo de balinera y embobinado; esto representa una profundización en la problemática planteada para el grupo de control Healthy-Propeller al desbalancear aún más las clases y ya se han observado las consecuencias que esto tiene en el rendimiento de los modelos. Las figuras 4.12 y 4.13 muestran el análisis exploratorio de datos hecho a las observaciones del grupo HPB.

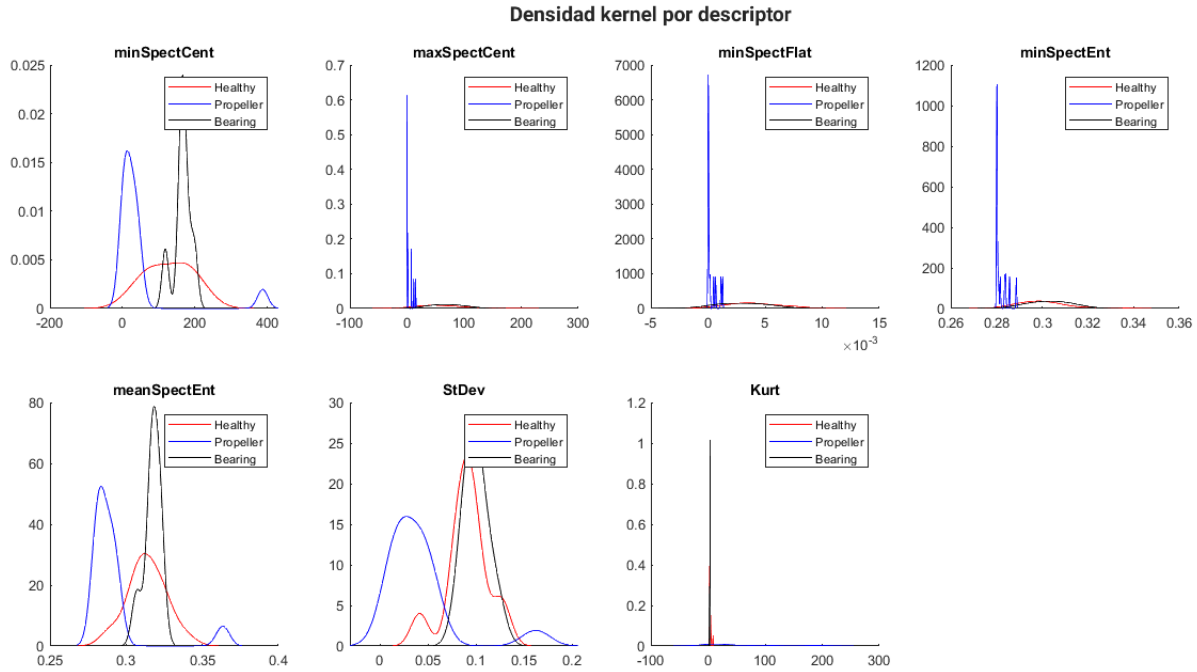

Figura 4.12: Estimación de la densidad del kernel para el grupo HPB

El Análisis Exploratiorio de Datos ya indica que este grupo será más complejo de separar: la categoría Bearing muestra similitudes con las clases Healthy y Propeller en varias características, y es de esperarse que en las métricas de los modelos entrenados exista un nivel

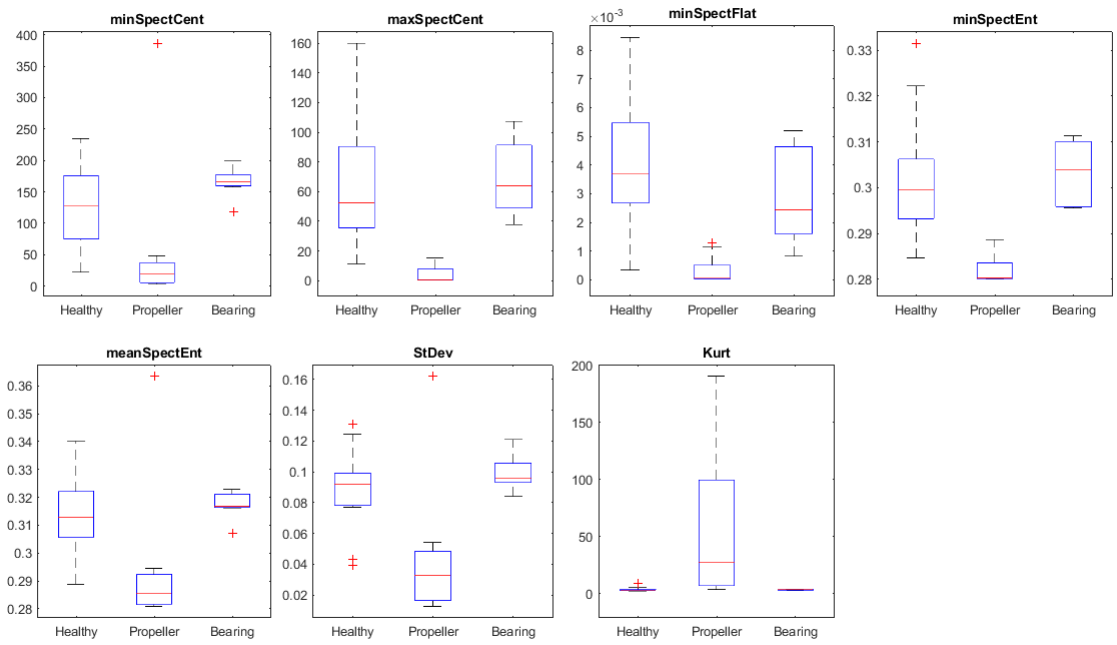

Figura 4.13: Diagramas de caja para los descriptores del grupo HPB.

considerable de confusión principalmente entre las clases Healthy y Bearing. Esto podría representar un mayor esfuerzo en el modelo de Decision Tree para establecer criterios, y en los modelos kNN y SVM para trazar fronteras de decisión e hiperplanos que separen las categorías. En este grupo las métricas no solo se resumen mediante la media macro, sino que también se ocupa la media proporcional para medir el rendimiento de los modelos con base en la presencia de las clases.

Los modelos kNN entrenados para este grupo de control tienen las métricas de evaluación de la tabla 4.8 y las matrices de confusión de la figura 4.14.

En la evaluación del modelo de Python se encuentra que las métricas referentes a la clase 3 están evaluadas en 0.00000; esto se debe a que en el momento de dividir aleatoriamente el dataset original en subconjuntos de entrenamiento y evaluación, no se asignan elementos de esta clase al subconjunto de evaluación (todos los datos de la clase 3 se están utilizando en entrenamiento). Para evitar esto, lo primero que se puede hacer es intentar cambiar la semilla de aleatoriedad que se utiliza en la función `train_test_split`, pero la solución más recomendable es adquirir más observaciones que fortalezcan el dataset; esto tiene sentido, pues solo hay siete observaciones pertenecientes a la clase Bearing. Debido a esto es necesaria la introducción de las métricas por pesos, **weighted**, así la valoración se hace respecto a la presencia de cada clase en el conjunto de evaluación y no de forma general como se hace en el

Tabla 4.8: Métricas de evaluación para el modelo kNN del grupo HPB.

Métrica	MATLAB (Neighbors: 10)	Python (Neighbors: 13)
Accuracy	0.76923	0.76923
Precission		
	1.00000	0.67000
	0.60000	1.00000
	0.50000	0.00000
Macro-avg precission	0.70000	0.56000
Weighted precission	0.86923	0.69000
Recall		
	0.66667	1.00000
	1.00000	0.80000
	1.00000	0.00000
Macro-avg recall	0.88889	0.60000
Weighted recall	0.76923	0.77777
F1-Score		
	0.80000	0.80000
	0.75000	0.89000
	0.66667	0.00000
Macro-avg F1-score	0.73890	0.56000
Weighted F1-Score	0.77820	0.71000

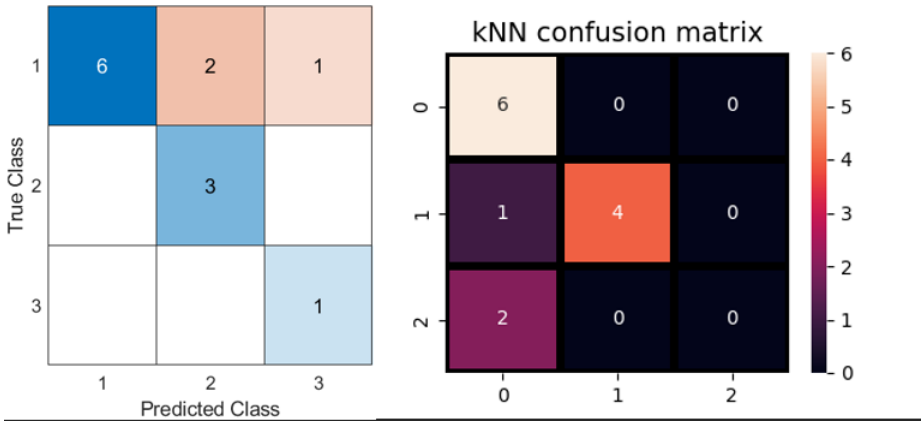

Figura 4.14: Matrices de confusión de los modelos kNN para el grupo HPB.

macro-average. En el modelo de MATLAB sí se asignan elementos de la clase 3 al conjunto de entrenamiento, pero las métricas weighted también resuelven en cierta proporción el sesgo que general el desbalance de clases.

Las matrices de confusión de este modelo indican que la generalización entre la clase 1 y 3 se empieza a perder, como reveló el análisis de datos. Podría deberse a que el conjunto de entrenamiento esté teniendo problemas de dimensionalidad, generando más confusión entre

las dos categorías. También existe un pequeño margen de confusión entre las clases Propeller y Healthy, pero esta ya se ha observado y analizado previamente. Este comportamiento podrá confirmarse analizando el modelo DT, cuyas métricas se reúnen en la tabla 4.9 y sus matrices de confusión en la figura 4.15.

Tabla 4.9: Métricas de evaluación para el modelo DT del grupo HPB.

Métrica	MATLAB	Python
Accuracy	0.61538	0.69230
Precission		
	0.60000	0.75000
	0.57140	1.00000
	1.00000	0.40000
Macro-avg precission	0.86667	0.72000
Weighted precission	0.68350	0.79000
Recall		
	0.50000	0.50000
	1.00000	0.80000
	0.33333	1.00000
Macro-avg recall	0.61111	0.77000
Weighted recall	0.61537	0.69000
F1-Score		
	0.54550	0.60000
	0.72730	0.89000
	0.50000	0.57000
Macro-avg F1-score	0.59093	0.69000
Weighted F1-Score	0.59093	0.71000

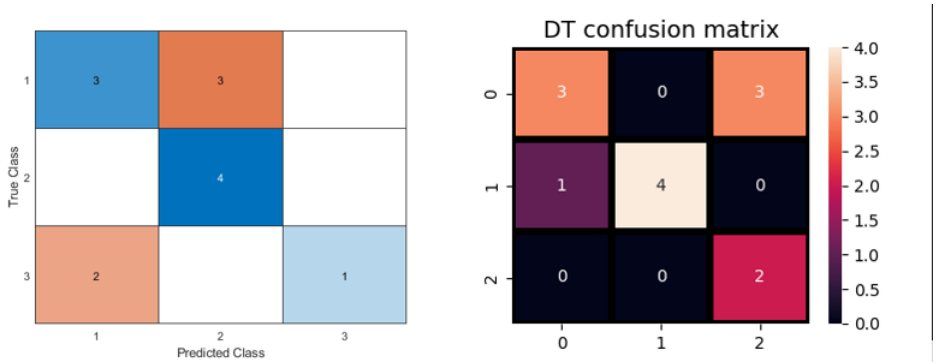

Figura 4.15: Matrices de confusión de los modelos DT para el grupo HPB.

El subconjunto de evaluación del modelo DT sí incluye observaciones de la clase 3, y aunque presenta disminución en la confusión el descenso en su rendimiento respecto a los dos grupos de control anteriores representa una dificultad separando los datos; esto valida el análisis

hecho sobre el modelo kNN: no solo el número de observaciones de la clase 3 es muy bajo, sino que los descriptores seleccionados no son suficientes para esta clase. Al final del capítulo se resumirán las razones por las que esto podría deberse y cómo corregirse, pero es preciso revisar el comportamiento del modelo SVM para fortalecer los resultados encontrados; el orden seleccionado para mostrar las métricas de los modelos deja a las máquinas de soporte vectorial porque, a pesar (y como consecuencia) de ser el más matemáticamente complejo de todos, se ve afectado por las variables que afectan a los modelos kNN y DT; las métricas de la tabla 4.10 y las matrices de confusión de la figura 4.16 confirman este suceso.

Tabla 4.10: Métricas de evaluación para el modelo SVM del grupo HPB.

Métrica	MATLAB (kernel: linear)	Python (kernel: rbf)
Accuracy	0.53846	0.84615
Precission		
	0.71430	0.83000
	1.00000	0.86000
	0.20000	0.00000
Macro-avg precission	0.63810	0.56000
Weighted precission	0.67913	0.78000
Recall		
	0.55560	0.83000
	0.50000	1.00000
	0.50000	0.00000
Macro-avg recall	0.51853	0.61000
Weighted recall	0.68639	0.85000
F1-Score		
	0.62500	0.83000
	0.66667	0.92000
	0.28570	0.00000
Macro-avg F1-score	0.52579	0.59000
Weighted F1-Score	0.57921	0.81000

El modelo de clasificación SVM obtenido efectivamente sufre por las condiciones ya establecidas: en el modelo MATLAB la exactitud tiene una disminución muy alta respecto a los otros grupos de control, mientras que en el modelo Python, aunque fuerte en exactitud (alcanza el 84 %), no incluye elementos de la clase 3 pero mantiene métricas relativamente altas. Al principio de esta sección se especificó que este problema era multiclase, y en el capítulo 2 se explicó el modelo SVM y el tratamiento que hace para entrenar modelos multiclase: se puede entrenar un modelo ECOC o uno OvO para entregar un resultado a partir de una combinación de clasificadores binarios. La comparación de resultados entre los modelos SVM

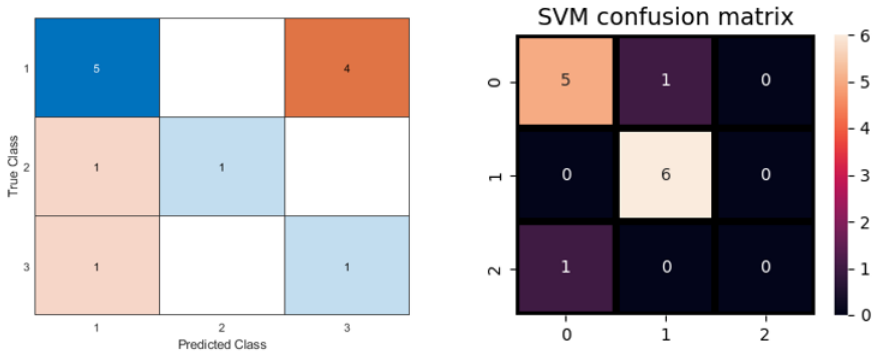

Figura 4.16: Matrices de confusión de los modelos SVM para el grupo HPB.

entrenados indica que, al menos para este conjunto de datos, la combinación utilizada en Python (Kernel RBF - One-vs-One) entrega mejores resultados que la utilizada en MATLAB (Kernel lineal - ECOC). Esto va acorde con la sensibilidad que tienen los modelos SVM ante la correcta selección de hiperparámetros.

El análisis de los modelos obtenidos en este grupo de control puede resumirse en:

- Deben aumentarse las observaciones de la categoría Bearing.
- Los ejemplares de fallo en embobinado y balinera probablemente no cuentan con suficiente daño para dejar de clasificarse como saludables.
- Una vez aumentado el tamaño del dataset es preciso repetir el análisis de correlaciones para determinar la importancia de los predictores seleccionados; además, se debe repetir el Análisis Exploratorio de Datos para seleccionar las características que mejor diferencien las tres clases.
- Ante los problemas de confusión entre las clases Healthy y Bearing, se recomienda la implementación de algoritmos kNN

Los resultados obtenidos muestran que una combinación de métodos estadísticos y espectrales es un enfoque sólido para la detección de averías en motores BLDC a partir del sonido; la selección de algoritmos basados en revisión sistemática de literatura y naturaleza matemática de los modelos de predicción indica que los modelos k-Nearest Neighbors, Decision Tree y Support Vector Machines son opciones robustas para la implementación de técnicas de mantenimiento predictivo como herramienta dentro del marco de la Industria 4.0.

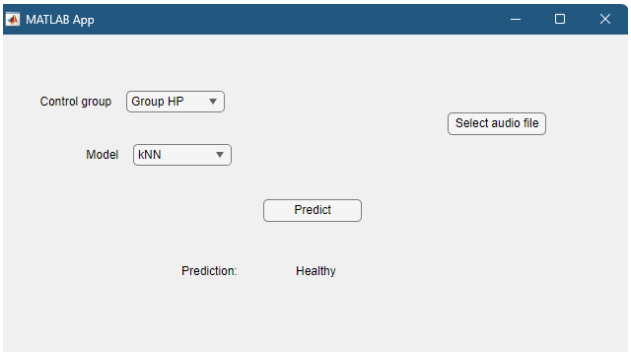

Figura 4.17: Mínimo Producto Viable en MATLAB App Designer

4.3. Mínimo Producto Viable

Una vez entrenados los modelos en ambos lenguajes de programación, se deben tomar acciones de despliegue; generalmente se entrega un mínimo producto viable, la mínima aplicación que permita el uso del modelo. Ya se ha descrito esta metodología en el capítulo 3; en MATLAB, una vez se entrenan modelos de aprendizaje se pueden guardar escribiendo en el script o en la ventana de comandos

```
save('model','modelVarName')
```

Guardando el modelo en la carpeta donde esté el workspace. La aplicación diseñada en el App Designer se ve como muestra la figura.

El usuario tiene la opción de seleccionar el modelo de aprendizaje con el que desea realizar la predicción y el archivo de audio que desea inyectar. Así, internamente se hace el preprocesamiento y la obtención del vector de características, se hace una predicción y se muestra al usuario. El enfoque aplicado con el MATLAB App Desginer es un despliegue básico y rápido de modelos Machine Learning.

4.4. Aplicación IoT en Django-Firebase

El enfoque de MATLAB App Designer, aunque funcional, carece de aplicabilidad: el usuario debe encargarse de la grabación y carga del archivo, por lo que como entregable final se desarrolló una herramienta IoT utilizando Django, Google Firebase y el microcontrolador ESP32.

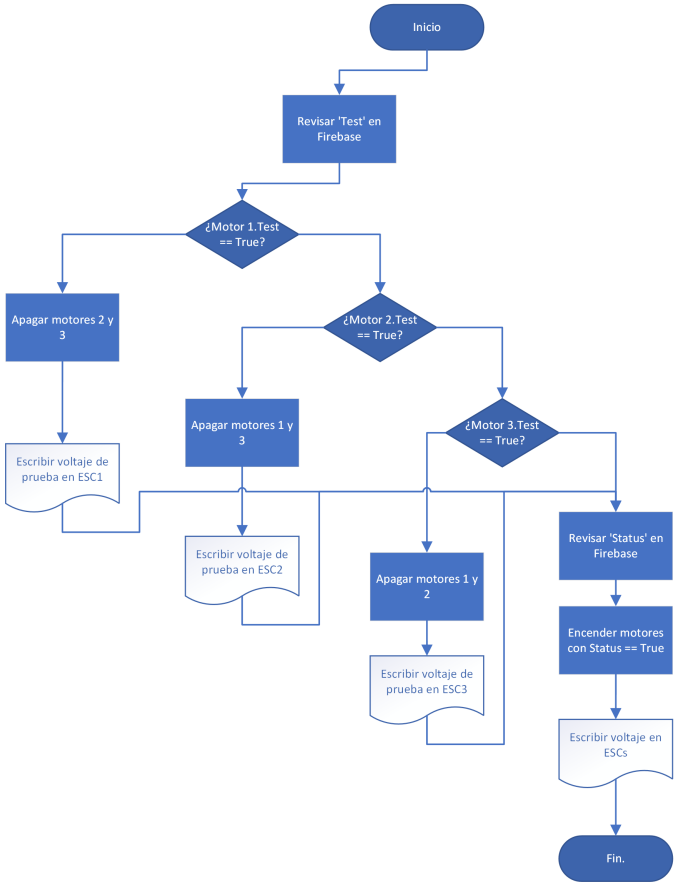

Figura 4.18: Diagrama de flujo para la capa de percepción

4.4.1. Capa de percepción

En la implementación de la capa de percepción se repiten las condiciones experimentales de adquisición de datos planteadas en el capítulo 2, pero en lugar de implementar un algoritmo en Blynk se utiliza el planteamiento del diagrama de flujo de la figura 4.18. En este montaje se utilizan tres motores A2212 simultáneamente y se establece la velocidad de los pulsos en 1600 μs.

El código, disponible en los Anexos del 6 continúa revisando los atributos 'Test' y 'Status' de RTDB de Firebase.

4.4.2. Capa de conectividad

Como ya se describió en el capítulo 3, la capa de conectividad se encarga de asociar la información percibida en una planta física con la capa de aplicación; este proceso se llevó a

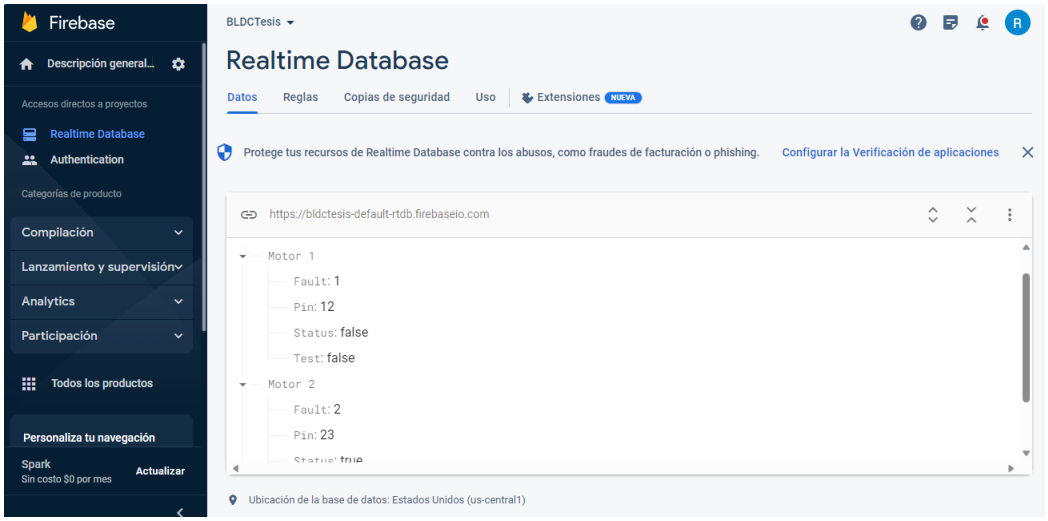

Figura 4.19: Captura de pantalla de la base de datos en tiempo real.

cabo en Google Firebase permitiendo una interacción entre la tarjeta ESP32 y la aplicación web de Django; la figura 4.19 muestra una captura de pantalla de la base de datos en tiempo real donde se observan las instancias generadas de la clase motor y sus atributos. El atributo **Fault** se refiere al tipo de fallo encontrado en el motor posterior al análisis, **Pin** es un marcador de tipo entero que registra el pin de entrada/salida al que está conectado el ESC del motor en la capa de percepción, **Status** es una bandera booleana que representa el estado operacional del motor: representa un dispositivo apagado o encendido desde la capa de aplicación, y **Test** está asociado al estado de identificación de averías: solo un motor puede tener el flanco Test en alto a la vez, pues cuando el estado es alto se apagan temporalmente los demás motores para hacer la adquisición de datos adecuadamente.

La base de datos en tiempo real de Firebase debe conectarse a las demás capas; para el ESP32 se utiliza el bloque de código

```
#include <ESP32Servo.h>
#include <FirebaseESP32.h>
#include <WiFi.h>
#include <WiFiClient.h>
//#include <Firebase_ESP_Client.h>

#include "addons/RTDBHelper.h"
#include "addons/TokenHelper.h"
#define API_KEY "AIzaSyAJHYrVgc3buthhVj9ne0CYxg3vYTF3fE8"
```

```
#define DATABASE_URL "https://bldctesis-default-rtdb.firebaseio.com/"

const int escPin1 = 5;
const int escPin2 = 14;
const int escPin3 = 25;

char ssid[] = "Stiw";
char pass[] ="1007784376";

bool signupOK = false;

FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  esc1.attach(escPin1);
  esc2.attach(escPin2);
  esc3.attach(escPin3);
  // pinMode(escPin1,OUTPUT);
  // pinMode(escPin2,OUTPUT);
  // pinMode(escPin3,OUTPUT);
  WiFi.begin(ssid,pass);
  Serial.print("Connecting to WiFi");
  while(WiFi.status()!=WL_CONNECTED){
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();
```

```
    config.api_key = API_KEY;
    config.database_url = DATABASE_URL;

    if(Firebase.signUp(&config,&auth,"","")){
      Serial.println("Ok");
      signupOK = true;
    }
    else{
      Serial.printf("%s\n",config.signer.signupError.message.c_str());
    }

    config.token_status_callback = tokenStatusCallback;

    Firebase.begin(&config,&auth);
    Firebase.reconnectWiFi(true);
    delay(1000);

  }
```

Mientras que en la aplicación de Django se utiliza el bloque de código

```
    config = {
    "apiKey": "PRIVATE-API-KEY",
    "authDomain": "bldctesis.firebaseapp.com",
    "databaseURL": "https://bldctesis-default-rtdb.firebaseio.com",
    "projectId": "bldctesis",
    "storageBucket": "bldctesis.appspot.com",
    "messagingSenderId": "164535771215",
    "appId": "1:164535771215:web:4bfcf62b6c729d4633ca3a"
}

firebase=pyrebase.initialize_app(config)
authe = firebase.auth()
database=firebase.database()
```

4.4.3. Capa de aplicación.

La capa de aplicación es aquella con la que interactúa principalmente el usuario, una aplicación web diseñada en Django como se describió en el capítulo 3. El procesamiento de los datos se hizo utilizando dos librerías: NumPy y Librosa, una librería gratuita para el procesamiento de audio, y la adquisición de la señal se hace utilizando la librería PyAudio. En las vistas de la función se implementó el método `recordAudio` que abre el micrófono MCR-005 a partir de indexación cuando está conectado a un puerto USB. El método empleado retorna un archivo de audio de diez segundos muestreado a 16 KHz simulando las condiciones de entrenamiento para facilitar la clasificación a los modelos:

```
CHUNK = 1024
FORMAT = pyaudio.paFloat32
CHANNELS = 1
RATE = 16000
RECORD_SECONDS = 10

def record_audio():
    p = pyaudio.PyAudio()

    device_index = 1  # Replace with the appropriate device index

    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    input=True,
                    input_device_index=device_index,
                    frames_per_buffer=CHUNK)

    frames = []

    print("Recording...")

    for _ in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
        data = stream.read(CHUNK)
        frames.append(data)
```

```
    print("Recording finished.")

    stream.stop_stream()
    stream.close()
    p.terminate()

    audio_data = b''.join(frames)
    return audio_data
```

El arreglo `audio_data` es el argumento de la función `preProcessing`, que utilizando Librosa y NumPy regresa el vector de características con el que se realizará la predicción.

```
def preProcessing(audio_data):
    sr = RATE
    spectral_centroids = librosa.feature.spectral_centroid(y=audio_data,sr=sr)[0]
    spectral_flatness = librosa.feature.spectral_flatness(y=audio_data)[0]

    spectrogram = np.abs(librosa.stft(audio_data)) ** 2
    normalized_spectrogram = spectrogram / np.sum(spectrogram)
    # Normalize the spectrogram
    spectral_entropy = -np.sum(normalized_spectrogram *
    np.log2(normalized_spectrogram + 1e-6))
    std_dev = (np.std(audio_data))
    kurto = kurtosis(audio_data)
    min_spectral_centroid = np.min(spectral_centroids)
    max_spectral_centroid = np.max(spectral_centroids)

    min_spectral_flatness = np.min(spectral_flatness)

    min_spectral_entropy = np.min(spectral_entropy)
    mean_spectral_entropy = np.mean(spectral_entropy)

    features_vector = np.array([
    min_spectral_centroid,
    max_spectral_centroid,
```

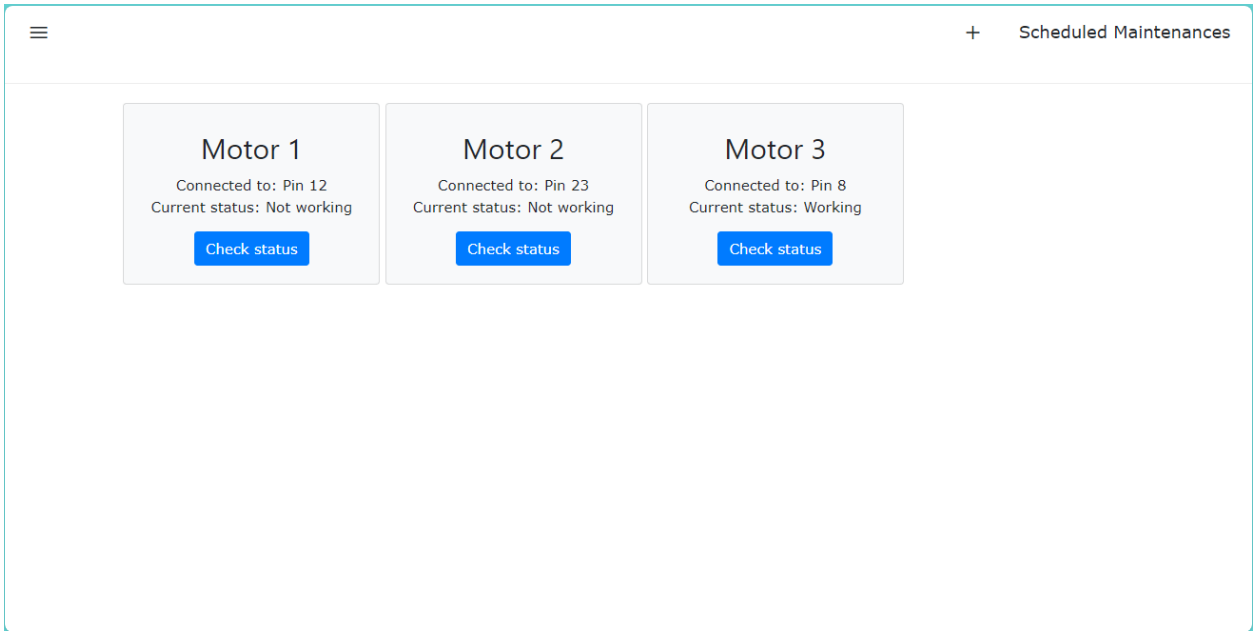

Figura 4.20: Vista principal de usuario en la aplicación Django.

```
min_spectral_flatness,
min_spectral_entropy,
mean_spectral_entropy,
std_dev,
kurto
])

return features_vector
```

La figura 4.20 muestra la interfaz principal de la aplicación de usuario; en esta vista se listan todos los motores obtenidos de la base de datos, el pin al que están conectados y su actual estado de funcionamiento; además, se observa en la figura 4.21 haciendo clic en el símbolo + se abre un formulario para crear una nueva instancia de la clase Motor en la base de datos local y en Google Firebase. Como se puede abstraer, el llamado al método `preProcessing` se hace desde un último método que se encarga de cargar los modelos de predicción, hacer la clasificación en tiempo real y enviarla como parte de una respuesta asíncrona de JavaScript a la vista de usuario. Para esto, se hicieron dos métodos: **modo automático**, observable en la figura 4.22, en el que se hace la predicción con todos los modelos y se entregan los resultados en una tabla, además ofreciéndole al usuario la posibilidad de apagar el motor y programar mantenimiento. La ventana modal que contiene el modo automático y sus resultados se

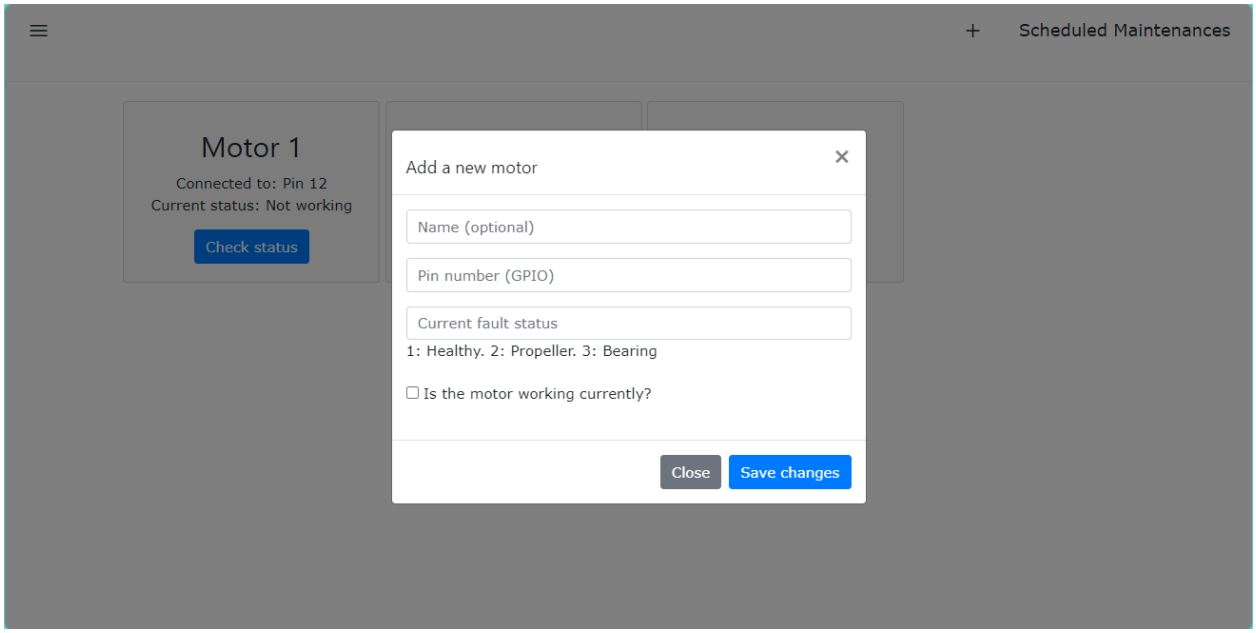

Figura 4.21: Formulario para añadir nuevo motor.

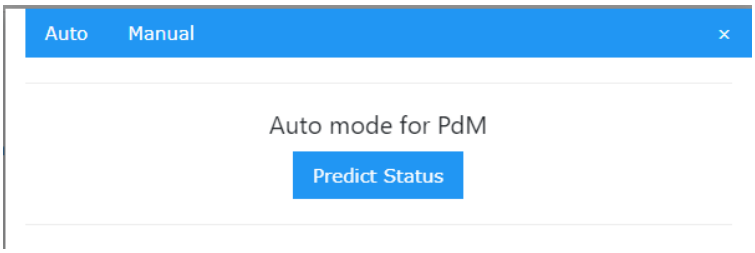

Figura 4.22: Modo automático de clasificación.

pueden observar en las figuras 4.22 y 4.23.

En el modo manual observado en la figura 4.24 se ofrece al usuario más personalización: se permite la selección de un grupo de control y de un modelo en específico, entregando solo la predicción obtenida por dicho modelo; este modo es más afín al usuario familiarizado con las métricas de evaluación y la naturaleza matemática de los algoritmos utilizados.

Al programar un mantenimiento se cambian dos atributos del motor en la base de datos local: una bandera booleana que lo indica como programado y un campo DateField que indica la fecha en que fue programado dicho mantenimiento; de esta forma, se incluye en la vista *Scheduled Maintenances*, como muestra la figura 4.25.

Finalmente, como detalle técnico se presenta la interfaz de administración de Django con un ejemplo de instancia para la clase Motor en la figura 4.26.

La aplicación IoT generada utilizando el ESP32, Google Firebase, Django y todas las libre-

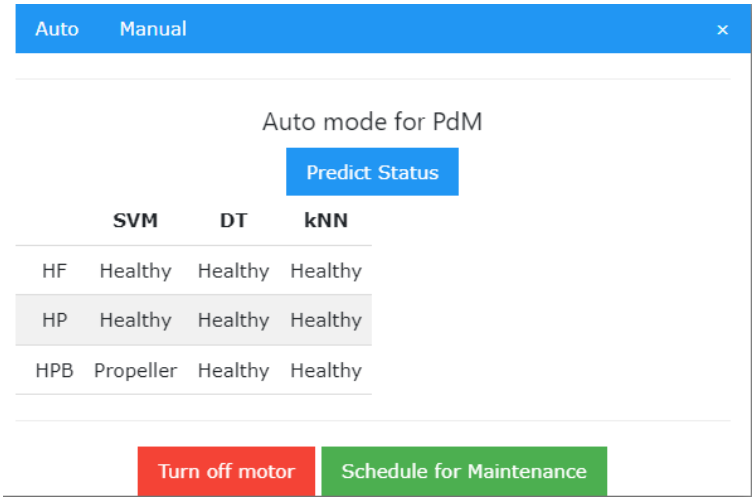

Figura 4.23: Resultados del modo automático.

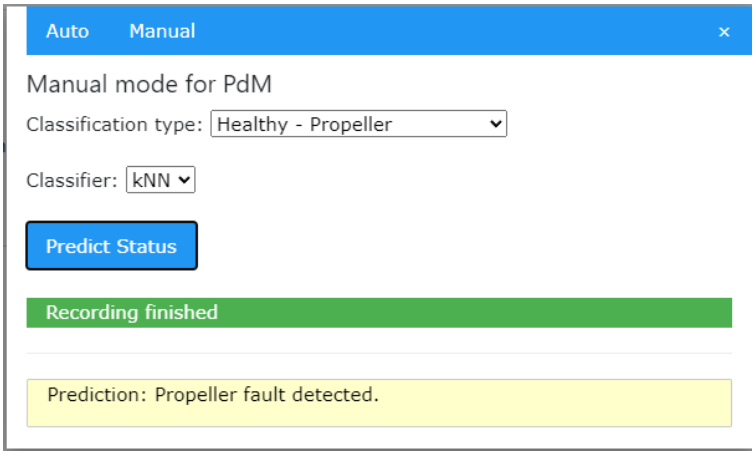

Figura 4.24: Modo manual y su presentación de resultados.

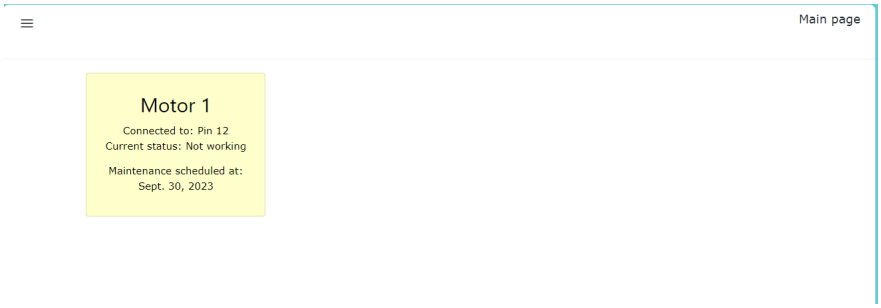

Figura 4.25: Vista de mantenimientos programados en la aplicación.

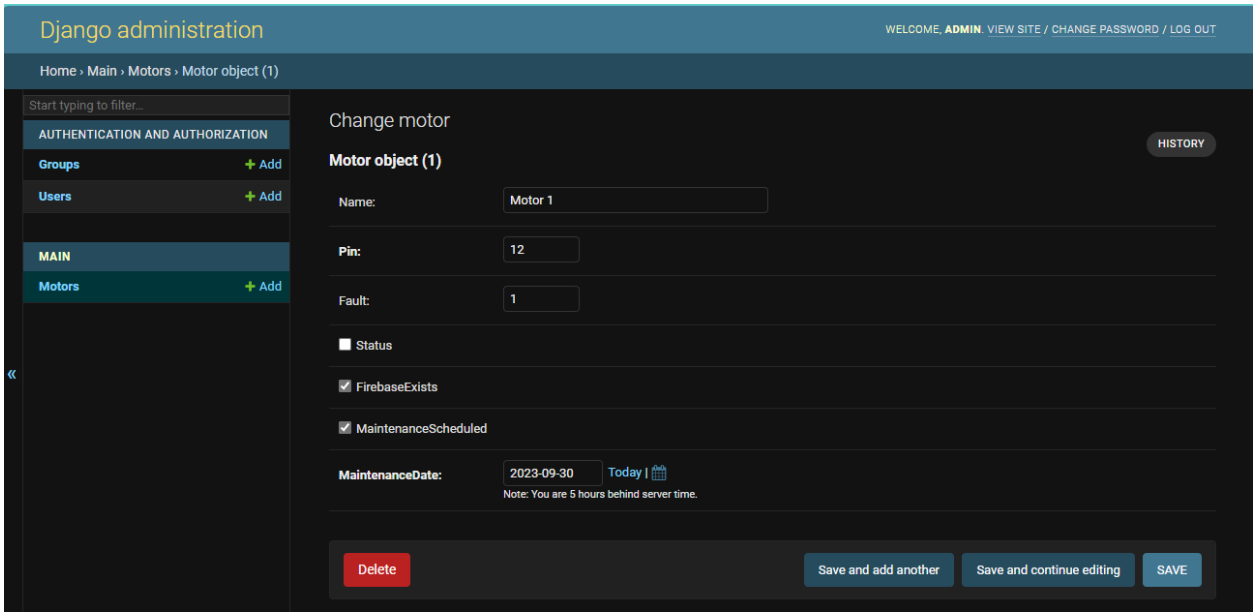

Figura 4.26: Instancia de Motor vista desde la Administración de Django

rías aplicadas en esta sección se muestran como un acercamiento altamente funcional para el campo de aplicaciones industriales conocido como Industrial Internet of Things (IIoT): permite satisfactoriamente una integración de servicios y experiencia de usuario asociados a los objetivos de la Industria 4.0.

En resumen, en este capítulo se presentan los resultados de caracterización estática, las métricas de evaluación de los modelos kNN, DT y SVM entrenados y su análisis respecto a su naturaleza matemática y al conjunto de datos, y los resultados de aplicaciones de despliegue producidas en MATLAB App Designer y Django. Los códigos implementados para obtener estos resultados se encuentran en la sección de Anexos 6.

4.5. Resumen del capítulo 4

En el capítulo de resultados se presentaron las métricas de evaluación de los modelos de aprendizaje entrenados, se efectúan análisis sobre su rendimiento, los factores que afectan a cada algoritmo debido a su naturaleza matemática y al conjunto de datos utilizados y se presentan soluciones a estas afecciones; además, se entregan resultados sobre las acciones de despliegue desarrolladas en MATLAB App Designer y Django-Firebase-ESP32, haciendo un balance entre las aplicaciones de despliegue rápidas y las industrialmente aplicables.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones generales

A partir del material estudiado y los resultados obtenidos en las secciones anteriores se pueden establecer conclusiones importantes; la primera es que el análisis mixto estadístico-espectral de señales de audio se muestra como una solución fuerte para la detección de averías en motores BLDC. Esta investigación se condujo con un dataset de 43 observaciones y bajo el análisis de una sola señal y siete descriptores fueron suficientes para obtener modelos con hasta un 92 % de exactitud y métricas de evaluación fuertes, valores cercanos a los enfoques multisensores revisados en la literatura; aunque es importante destacar que los resultados de los modelos de aprendizaje indican confiabilidad, es igual de necesario indicar que las métricas de evaluación se pueden mejorar con tres acciones principales: aumentar el número de observaciones y ejemplares, solucionar el desbalance de clases y hacer una selección mayor y más rigurosa de características; como se mencionó en la sección de resultados, estos tres factores tuvieron una incidencia considerable en los modelos obtenidos; el principal problema del grupo HPB fue la falta de observaciones pertenecientes a la clase 3 y lo poco representativa que era la avería: el Análisis Exploratorio indicaba lo que podría describirse como una avería no tan grave, congruente con el ensayo de caracterización estática: aunque había una diferencia en inductancias y resistencias de los ejemplares, no era tan grande, haciendo que la diferenciabilidad entre clases no fuera tan fuerte como se comprobó en las métricas de los modelos HPB.

Respecto a los modelos de aprendizaje entrenados, el algoritmo de k-Nearest Neighbors parece mostrar el mejor desempeño entre los tres algoritmos comparados en cada grupo de control, teniendo en cuenta su simplicidad y su bajo costo computacional. Las matrices de

confusión de los modelos de aprendizaje entrenados muestran que se evitó satisfactoriamente caer en el sobremuestreo: los modelos no se aprendieron los datos. Aunque los modelos SVM sean matemáticamente más rigurosos que los kNN y DT no significa que siempre sean una solución adecuada, precisamente su rigurosidad hace que sea susceptible a muchos factores externos y es una buena práctica la determinación apropiada de su uso teniendo en cuenta su complejidad computacional y matemática.

Así, se comprueban las ventajas y desventajas de cada algoritmo aplicado, además de encontrarse que entrenar un modelo de aprendizaje automático con los mismos datos en dos entornos de programación distintos no necesariamente implica obtener los mismos resultados: MATLAB y Python utilizan opciones por defecto distintas y no aplican el mismo escalamiento de datos al entrenar, esto sin contar con la aleatoriedad interna que se utilice para separar el conjunto de datos; por ejemplo, en los modelos SVM de multiples clases la simulación en MATLAB se hace mediante ECOC y en Python mediante One-vs-One.

En cuanto a la implementación y despliegue de los modelos de Machine Learning, en el marco de la Cuarta Revolución Industrial es preciso brindar soluciones funcionales y de aplicabilidad alta; el despliegue rápido hecho en MATLAB no es una solución industrial, es una primera solución de desarrollo generada con el fin de probar los modelos ante información nueva; el enfoque de IIoT no solamente va en la línea de la poca invasibidad que se desea con el mantenimiento predictivo, sino que también es una solución más personalizable y optimizable, aunque en una aplicación industrial puede resultar más costosa debido al uso de servicios privados. La tarjeta ESP32 utilizada como eje de la capa de percepción ha tenido un surgimiento bastante fuerte gracias a su versatilidad, la disponibilidad de protocolos de comunicación, facilidad de implementar y comunidad de soporte, y la aplicación en Django solidifica las técnicas de mantenimiento predictivo e Internet de las Cosas que se obtuvieron con esta investigación.

5.2. Trabajos futuros

Los problemas enfrentados en el desarrollo experimental de este trabajo de investigación se convierten en oportunidades de trabajos futuros, por ejemplo, la baja diferencialidad del tipo Bearing es importante para tomar el siguiente paso en el mantenimiento predictivo: la estimación de la vida útil (RUL, *Remaining Useful Life*), pues así como el trabajo desarrollado se fundamentó en la detección de averías, es decir, algoritmos de clasificación, la estimación de la vida útil implica el desarrollo de modelos de regresión para los que se necesitan ejem-

plares con varios niveles de daño. Así mismo, la implementación de la regresión para obtener RUL implica una ampliación bastante considerable del tamaño del conjunto de datos; a este conjunto de datos puede aplicarse la misma metodología de esta investigación mejorando las métricas de evaluación de los modelos de clasificación mientras se construyen modelos nuevos de aprendizaje. En conjunto, pueden aplicarse técnicas de aprendizaje profundo y aprendizaje por refuerzo donde los modelos aprendan por sí mismos sobre el entorno, mejorando la aplicación industrial. Podría ser interesante la implementación de un enfoque multisensor: análisis de vibración y temperatura podrían representar señales importantes y poco invasivas que indiquen la condición del motor. Es preciso anotar que, aunque esta investigación se enfocó en motores BLDC, al principio se indicó que este sistema se utiliza por su sencillez y extensibilidad a otros sistemas electromecánicos.

Como sugerencia final de trabajos futuros se puede implementar el aprendizaje automático diréctamente en hardware: se permite la reducción de costos, latencia, se aumenta la velocidad, eficiencia y escalabilidad, haciendo que el costo computacional pase a ser una preocupación menor y se puedan implementar modelos de Deep Learning y Reinforcement Learning.

Bibliografía

[1] T. Borgi, A. Hidri, B. Neef y M. S. Naceur, «Data Analytics for predictive maintenance of Industrial Robots,» *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, 2017. DOI: 10.1109/aset.2017.7983729.

[2] C. Franciosi, V. Di Pasquale, R. Iannone y S. Miranda, «Multi-stakeholder perspectives on indicators for sustainable maintenance performance in production contexts: An exploratory study,» *Journal of Quality in Maintenance Engineering*, vol. 27, n.º 2, págs. 308-330, 2020. DOI: 10.1108/jqme-03-2019-0033.

[3] L.-I. Alvarez, C. Lozano y D. Bravo, «Machine learning for predictive maintenance scheduling of distribution transformers,» *Journal of Quality in Maintenance Engineering*, ene. de 2022. DOI: 10.1108/JQME-06-2021-0052.

[4] T. P. Carvalho, F. A. Soares, R. Vita, R. d. P. Francisco, J. P. Basto y S. G. Alcalá, «A Systematic Literature Review of Machine Learning Methods Applied to Predictive Maintenance,» *Computers and Industrial Engineering*, vol. 137, nov. de 2019, ISSN: 03608352. DOI: 10.1016/j.cie.2019.106024.

[5] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael y B. Safaei, «Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0,» *Sustainability*, vol. 12, n.º 19, pág. 8211, oct. de 2020, ISSN: 2071-1050. DOI: 10.3390/su12198211. (visitado 20-12-2022).

[6] P. Nunes, J. Santos y E. Rocha, «Challenges in predictive maintenance – A review,» *CIRP Journal of Manufacturing Science and Technology*, vol. 40, págs. 53-67, feb. de 2023, ISSN: 17555817. DOI: 10.1016/j.cirpj.2022.11.004. (visitado 20-12-2022).

[7] A. L. Bowler, M. P. Pound y N. J. Watson, «A Review of Ultrasonic Sensing and Machine Learning Methods to Monitor Industrial Processes,» *Ultrasonics*, vol. 124, ago. de 2022, ISSN: 0041624X. DOI: 10.1016/j.ultras.2022.106776.

[8] M. Nacchia, F. Fruggiero, A. Lambiase y K. Bruton, «A Systematic Mapping of the Advancing Use of Machine Learning Techniques for Predictive Maintenance in the Manufacturing Sector,» *Applied Sciences (Switzerland)*, vol. 11, n.º 6, mar. de 2021, ISSN: 20763417. DOI: 10.3390/app11062546.

[9] K. M. Rashid y J. Louis, «Activity Identification in Modular Construction Using Audio Signals and Machine Learning,» *Automation in Construction*, vol. 119, nov. de 2020, ISSN: 09265805. DOI: 10.1016/j.autcon.2020.103361.

[10] P. Suawa, T. Meisel, M. Jongmanns, M. Huebner y M. Reichenbach, «Modeling and Fault Detection of Brushless Direct Current Motor by Deep Learning Sensor Data Fusion,» *Sensors*, vol. 22, n.º 9, pág. 3516, mayo de 2022, ISSN: 1424-8220. DOI: 10.3390/s22093516. (visitado 25-03-2023).

[11] F. Wang, R. Liu, Q. Hu y X. Chen, «Cascade Convolutional Neural Network with Progressive Optimization for Motor Fault Diagnosis under Nonstationary Conditions,» *IEEE Transactions on Industrial Informatics*, vol. 17, n.º 4, págs. 2511-2521, abr. de 2021, ISSN: 19410050. DOI: 10.1109/TII.2020.3003353.

[12] J. Wang, Y. Liang, Y. Zheng, R. X. Gao y F. Zhang, «An integrated fault diagnosis and prognosis approach for predictive maintenance of wind turbine bearing with limited samples,» *Renewable Energy*, vol. 145, págs. 642-650, ene. de 2020, ISSN: 09601481. DOI: 10.1016/j.renene.2019.06.103. (visitado 20-12-2022).

[13] A. Altinors, F. Yol y O. Yaman, «A Sound Based Method for Fault Detection with Statistical Feature Extraction in UAV Motors,» *Applied Acoustics*, vol. 183, dic. de 2021, ISSN: 1872910X. DOI: 10.1016/j.apacoust.2021.108325.

[14] E. Babu, J. Francis, E. Thomas, R. Cherian y S. S. Sunandhan, «Predictive Analysis of Induction Motor Using Current, Vibration and Acoustic Signals,» en *2022 2nd International Conference on Power Electronics & IoT Applications in Renewable Energy and Its Control (PARC)*, Mathura, India: IEEE, ene. de 2022, págs. 1-7, ISBN: 978-1-66543-215-3. DOI: 10.1109/PARC52418.2022.9726688. (visitado 25-03-2023).

[15] M. Cakir, M. A. Guvenc y S. Mistikoglu, «The experimental application of popular machine learning algorithms on predictive maintenance and the design of IIoT based condition monitoring system,» *Computers & Industrial Engineering*, vol. 151, pág. 106 948, ene. de 2021, ISSN: 03608352. DOI: 10.1016/j.cie.2020.106948. (visitado 20-12-2022).

[16] J. D. Carrera, P. V. Matute, E. Pelaez y F. R. Loayza, «Fault Diagnosing for Electric Motors Based in Noise Data Classification Using Deep Learning,» en *2022 IEEE ANDESCON*, Barranquilla, Colombia: IEEE, nov. de 2022, págs. 1-6, ISBN: 978-1-66548-854-9. DOI: 10.1109/ANDESCON56260.2022.9989522. (visitado 25-03-2023).

[17] V. Duc Nguyen, E. Zwanenburg, S. Limmer, W. Luijben, T. Back y M. Olhofer, «A Combination of Fourier Transform and Machine Learning for Fault Detection and Diagnosis of Induction Motors,» en *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, Yinchuan, China: IEEE, ago. de 2021, págs. 344-351, ISBN: 978-1-66544-391-3. DOI: 10.1109/DSA52907.2021.00053. (visitado 25-03-2023).

[18] A. Glowacz, «Acoustic Based Fault Diagnosis of Three-Phase Induction Motor,» *Applied Acoustics*, vol. 137, págs. 82-89, ago. de 2018, ISSN: 1872910X. DOI: 10.1016/j.apacoust.2018.03.010.

[19] S. Han, N. Mannan, D. C. Stein, K. R. Pattipati y G. M. Bollas, «Classification and Regression Models of Audio and Vibration Signals for Machine State Monitoring in Precision Machining Systems,» *Journal of Manufacturing Systems*, vol. 61, págs. 45-53, oct. de 2021, ISSN: 02786125. DOI: 10.1016/j.jmsy.2021.08.004.

[20] H. Hesabi, M. Nourelfath y A. Hajji, «A Deep Learning Predictive Model for Selective Maintenance Optimization,» *Reliability Engineering and System Safety*, vol. 219, mar. de 2022, ISSN: 09518320. DOI: 10.1016/j.ress.2021.108191.

[21] Z. Li, R. Liu y D. Wu, «Data-Driven Smart Manufacturing: Tool Wear Monitoring with Audio Signals and Machine Learning,» *Journal of Manufacturing Processes*, vol. 48, págs. 66-76, dic. de 2019, ISSN: 15266125. DOI: 10.1016/j.jmapro.2019.10.020.

[22] S. Lu, G. Qian, Q. He, F. Liu, Y. Liu y Q. Wang, «In Situ Motor Fault Diagnosis Using Enhanced Convolutional Neural Network in an Embedded System,» *IEEE Sensors Journal*, vol. 20, n.º 15, págs. 8287-8296, ago. de 2020, ISSN: 15581748. DOI: 10.1109/JSEN.2019.2911299.

[23] M. Nikfar, J. Bitencourt y K. Mykoniatis, «A Two-Phase Machine Learning Approach for Predictive Maintenance of Low Voltage Industrial Motors,» *Procedia Computer Science*, vol. 200, págs. 111-120, 2022, ISSN: 18770509. DOI: 10.1016/j.procs.2022.01.210. (visitado 20-12-2022).

[24] J. Pacheco-Chérrez, J. A. Fortoul-Díaz, F. Cortés-Santacruz, L. María Aloso-Valerdi y D. I. Ibarra-Zarate, «Bearing fault detection with vibration and acoustic signals: Comparison among different machine leaning classification methods,» *Engineering Failure Analysis*, vol. 139, pág. 106 515, sep. de 2022, ISSN: 13506307. DOI: 10.1016/j.engfailanal.2022.106515. (visitado 20-12-2022).

[25] R. K. Patel y V. Giri, «Feature selection and classification of mechanical fault of an induction motor using random forest classifier,» *Perspectives in Science*, vol. 8, págs. 334-337, sep. de 2016, ISSN: 22130209. DOI: 10.1016/j.pisc.2016.04.068. (visitado 20-12-2022).

[26] S. Patil y K. Wani, «Gear fault detection using noise analysis and machine learning algorithm with YAMNet pretrained network,» *Materials Today: Proceedings*, S2214785322061211, sep. de 2022, ISSN: 22147853. DOI: 10.1016/j.matpr.2022.09.307. (visitado 20-12-2022).

[27] H. A. Raja, H. Raval, T. Vaimann, A. Kallaste, A. Rassolkin y A. Belahcen, «Cost-Efficient Real-Time Condition Monitoring and Fault Diagnostics System for BLDC Motor Using IoT and Machine Learning,» en *2022 International Conference on Diagnostics in Electrical Engineering (Diagnostika)*, Pilsen, Czech Republic: IEEE, sep. de 2022, págs. 1-4, ISBN: 978-1-66548-082-6. DOI: 10.1109/Diagnostika55131.2022.9905102. (visitado 25-03-2023).

[28] S. Shao, R. Yan, Y. Lu, P. Wang y R. X. Gao, «DCNN-Based Multi-Signal Induction Motor Fault Diagnosis,» *IEEE Transactions on Instrumentation and Measurement*, vol. 69, n.º 6, págs. 2658-2669, jun. de 2020, ISSN: 15579662. DOI: 10.1109/TIM.2019.2925247.

[29] T. A. Shifat y J. W. Hur, «ANN Assisted Multi Sensor Information Fusion for BLDC Motor Fault Diagnosis,» *IEEE Access*, vol. 9, págs. 9429-9441, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3050243.

[30] R. M. Souza, E. G. Nascimento, U. A. Miranda, W. J. Silva y H. A. Lepikson, «Deep learning for diagnosis and classification of faults in industrial rotating machinery,» *Computers & Industrial Engineering*, vol. 153, pág. 107 060, mar. de 2021, ISSN: 03608352. DOI: 10.1016/j.cie.2020.107060. (visitado 20-12-2022).

[31] Z. Xu, D. Yu e Y. Hu, «Motor Fault Diagnosis Method Based on Deep Learning,» en *2022 11th International Conference of Information and Communication Technology (ICTech))*, Wuhan, China: IEEE, feb. de 2022, págs. 236-239, ISBN: 978-1-66549-694-0. DOI: 10.1109/ICTech55460.2022.00054. (visitado 20-12-2022).

[32] O. Yaman, F. Yol y A. Altinors, «A Fault Detection Method Based on Embedded Feature Extraction and SVM Classification for UAV Motors,» *Microprocessors and Microsystems*, vol. 94, pág. 104 683, oct. de 2022, ISSN: 01419331. DOI: 10.1016/j.micpro.2022.104683.

[33] Y. Yao, G. Gui, S. Yang y S. Zhang, «An adaptive anti-noise network with recursive attention mechanism for gear fault diagnosis in real-industrial noise environment condition,» *Measurement*, vol. 186, pág. 110 169, dic. de 2021, ISSN: 02632241. DOI: 10.1016/j.measurement.2021.110169. (visitado 20-12-2022).

[34] A. Hughes y B. Drury, *Electric motors and drives fundamentals, types and applications*. Elsevier, 2013.

[35] O. Oguntoyinbo, «PID Control of Brushless DC Motor and Robot Trajectory Planning Simulation with MATLAB®/SIMULINK®,» Tesis doct., University of Applied Sciences, Finlandia, 2009.

[36] Espressif. «ESP-IDF Programming Guide.» (2023), dirección: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/user-guide-devkitm-1.html.

[37] M. R. Spiegel, *Theory and Problems of Statistics*. McGraw-Hill, 1992.

[38] J. G. Proakis y D. G. Manolakis, *Tratamiento Digital de Señales*. Madrid: PEARSON EDUCACIÓN S.A., 2007, ISBN: 978-84-8322-347-5.

[39] D. Mitrović, M. Zeppelzauer y C. Breiteneder, «Chapter 3 - Features for Content-Based Audio Retrieval,» en *Advances in Computers: Improving the Web*, ép. Advances in Computers, vol. 78, Elsevier, 2010, págs. 71-150. DOI: https://doi.org/10.1016/S0065-2458(10)78003-7. dirección: https://www.sciencedirect.com/science/article/pii/S0065245810780037.

[40] J. Johnston, «Transform coding of audio signals using Perceptual Noise Criteria,» *IEEE Journal on Selected Areas in Communications*, vol. 6, n.º 2, págs. 314-323, 1988. DOI: 10.1109/49.608.

[41] Y. N. Pan, J. Chen y X. L. Li, «Spectral entropy: A complementary index for rolling element bearing performance degradation assessment,» *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 223, n.º 5, págs. 1223-1231, 2008. DOI: 10.1243/09544062jmes1224.

[42] W. H. Press, ed., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge ; New York: Cambridge University Press, 1992, ISBN: 978-0-521-43108-8.

[43] mobizt. «Firebase Realtime Database Arduino Library for ESP32.» Accessed: May 20, 2023. (2023), dirección: https://github.com/mobizt/Firebase-ESP32.

[44] Django Software Foundation. «Django Web Framework.» Accessed: January 2023. (2023), dirección: https://www.djangoproject.com/.

Capítulo 6

Anexos

6.1. Software utilizado

- Audacity.
- MATLAB R2023a.
  - Signal Processing Toolbox.
  - Audio Toolbox.
  - App Designer.
- Python:
  - Django.
  - NumPy.
  - SciPy.
  - Librosa.
  - Pandas.
  - Scikit-Learn.
  - Pyrebase.
  - PyAudio.
- Visual Studio Code.
- Google Firebase.

6.2. Cronograma de actividades

Fase	Nov 2022	Dic 2022	Ene 2023	Feb 2023	Mar 2023	Abr 2023	May 2023	Jun 2023	Jul 2023	Ago 2023	Sep 2023	Oct 2023
Fase Inicial												
Investigación preliminar												
Estudio del estado del arte												
Adquisición de datos												
Caracterización de los sujetos												
Diseño del experimento												
Conformación del dataset												
Entrenamiento de los modelos												
Validación y mantenimiento de los modelos												
Acciones de despliegue												
Diseño del MVP en App Designer												
Implementación de aplicación IoT en Django												
Redacción del documento												

Tabla 6.1: Cronograma de Investigación

6.3. Artículo de revisión sistemática - IEEE CCAC 2023.

Machine Learning and Audio Signal Processing for Predictive Maintenance: A review

Rommel Prieto, Diego Bravo
Physics Department
Universidad del Cauca
Popayán, Colombia
Email: {stiwardprieto, dibravo} @unicauca.edu.co

***Abstract*—This work shows a systematic review of literature of machine learning and audio processing for applications in preventive maintenance. It intends to prove how audio signal processing combined with Machine Leanring techniques can produce a powerfool tool to detect anomalies and malfunctions in electromechanical devices. The document describes a review of art state and the algorithms that can be used for preventive maintenance applications. When reviewed, the literature proves that Machine Learning and Deep Learning approaches provide high accuracy results as tools for PdM, and that Support Vector Machines, k-Nearest Neighbors and Convolutional Neural Networks are the most used approaches as they achieve the highest evaluation metrics, and prove sound, vibration and current to be the most popular signals to train ML-PdM oriented models.**

***Index Terms*—Audio Signal Processing, Machine Learning, Predictive Maintenance, Support Vector Machines, Industry 4.0, Classification Algortithms.**

I. Introduction

Throughout modern history humanity has faced transformative events known as "industrial revolutions"; the First Industrial Revolution introduced steam machines and hydraulic energy, allowing the transition from the usage of humans and animals; this revolution is followed by the one that came with fuel and gas, alongside with the invention of telephone, which brought mass production and the origins of automatization, followed by the arrival of Programmable Logical Controllers that introducted to data analysis. Industry is currently going through the Fourth Industrial Revolution, also known as Industry 4.0 (I4.0); the goal is the integration between physical and digital systems, or as Lee [12] defines it: the pursuit of autonomous industrial systems base on Artificial Intelligence, Big Data, Data Analytics, Cloud Computing, Internet of Things, among others, implying the possibility of automatizing data collection from machines and components. Cinar [34] claims Machine learning algorithms can be applied over the collected data and allow fault detection and diagnosis. The techniques often used for this end infuse intelligence within systems so they can learn automatically and adapt to changing environments using historical experience in the training stages.

As Carvalho [5] reviews, the appliance of analytic approaches make it possible to achieve advantages such as manteinance cost reduction, machine fault reduction, repair stop reduction, spare parts inventory reduction, spare part life increasing, increasing production, improvement in operator safety, repair verification, overall profit, among others. The discussion about mainteinance classifies it in three main types:

1) Corrective maintenance
2) Preventive maintenance - PvM
3) Predictive maintenance - PdM

A summary of maintenance types is depicted in Fig. 1. The selection of a maintenance strategy is crucial for the improvement of equipment condition, reducing the equipment failure rates and minimization of maintenance costs, while maximizing the life of equipment. According to this, predictive maintenance would be the most promising strategy during the I4.0 due to optimization in the management of assets. An evolution for PdM already exists: prescriptive maintenance, described by Nunes as a step-further that uses the predictions made to make relevant suggestions which lead to the best course of action to take with the failure in order to increase the Remaining Useful Life (RUL), but prescriptive maintenance requires a solid consolidation of PdM in order to be implemented.

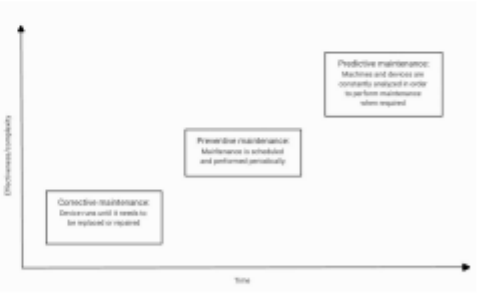

Fig. 1. Representation of maintenance types throughout time and their effectiveness.

According to Bowler [2], the manufacturing sector has depicted a trend to increase the use of collection and interpretation of data as a tool for decision making and the improvement of productivity, sustainability and product quality. Machine Learning has proven to be one of the most powerful tools for the development of predictive algorithms as its approaches

have the abilitty to handle high-dimensional and multivariable data, whilst extracting hidden relationships within data: an outcome can be predicted by a previously trained model based on historical input data and its output behavior. Nacchia [17] claims ML techniques make use of three main principles: representation, evaluation and classification, as they allow to get more from less by improving with performances. The effectivness of ML is measured by its ability to efficently handle information. As it has been stated previously, the nature of the data that is analyzed depends on the goal of the system; one approach of particular interest is the audio-based classification or *machine hearing* [23]. The main goal of this article is to review the methods, developments, approaches and challenges of machine hearing as a tool for predictive maintenance, especifically referring to those that can be applied to electromechanical systems.

II. Machine Learning approaches for PdM: a systematic review.

In order to find the most suitable articles for this systematic literature review, the following research questions were held:

- What Machine Learning techniques are being applied for Predictive Maintenance?
- Which of the ML techniques are more suitable for audio analysis?
- How are PdM algorithms applied to electromechanical systems?
- How is the data collected for PdM algorithms in electromechanical systems?

This questions are solved by researching for articles in literature databases such as EBSCO, ScienceDirect and IEEEXplore. While performing the selection, some inclusion and exclusion criteria where held, such as:

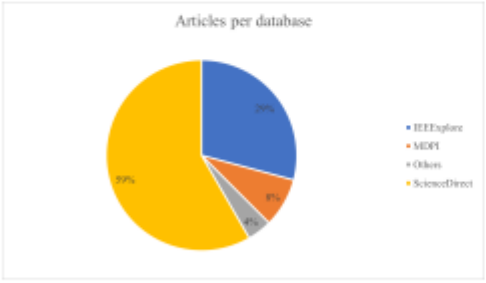

Fig. 2. Number of articles per database

- Exclusion criteria:
  - Studies published prior to 2018, as the main interest of this review is in the most current research to PdM.
  - Research that is not related to PdM of electromechanical systems.
- Inclusion criteria:
  - Studies that compare different machine learning techniques for fault diagnosis in electromechanical systems, including studies that present comparative results of accuracy and speed of the evaluated models.
  - Studies that address in-situ fault diagnosis. This includes studies that present solutions to identify and analyze faults, and studies that present experimental results to validate the performance of these proposals.
  - Works that focus on presenting techniques for analyzing and processing audio signals in order to extract features that might be helpful in training an ML algorithm.

Fig. 2 makes a comparison of the number of articles and their source databases.

A. A systematic approach to the design and implementation of an ML-PdM algorithm

After undergoing the necessary review, a systematic structure for designing and executing a machine learning (ML)-based Predictive Maintenance (PdM) algorithm can be outlined. These procedural steps are widely recognized and must be described:

1) Data selection: The first step in designing an ML algorithm is to identify and select the most relevant data for the problem at hand. This involves gathering data from different sources and selecting the data that best represents the problem domain. It is crucial to ensure the data quality and the information it may contain that can be aid the model to learn and make accurate predictions. This data is often retrieved by performing an experiment of a specific nature, for example: Yaman [32] executes the data acquisition on brushless DC motors embedded in an UAV system, testing them under four different conditions: healthy motor, bearing fault, winding fault and both bearing and winding faults. All motors were driven with different input voltages and their output voltages and currents are recorded.

2) Data pre-processing: Once data has been selected, it needs to be pre-processed to prepare it for use in the Machine Learning model. This phase involves tasks as cleaning the data, handling missing values and scaling or normalizing the data to ensure that it is in the appropriate range in order for it to be used to improve learning of an ML model. As an example, the approach proposed by Li [13]in smart manufacturing based on audio signals performs a low-frequency noise filtering and normalization of the audio signals collected from machinery, and uses tools and methods like Fast Fourier Transform to perform the feature extraction that will feed a Support Vector Machine (SVM) or Rain Forest (RF) algorithm.

Manjare [16] describes the identification of condition indicators as an extra step needed for PdM applications that can be categorized within the data selection and feature extraction phase. Their investigation describes said indicators as signal that belong to time domain, frequency domain or both. An example of this is provided by Wang et al. [30]

while researching a method for fault diagnosis and prognosis for wind turbines; the framework proposed is shown in Fig. 3.

3) Model selection: The next step is to select an appropriate ML model that is best suited to the problem at hand. The choice of a specific model depends on factors like the problem to address and the computational resources [7]. A classification of the Machine Learning techniques and the data nature for the reviewed material can be found in table I. Evaluation metrics have been included as they have been reported by their authors; accuracy has been selected as the main metric as it is the most spread metric among the authors, but when accuracy is not available said metric is specified. The meaning of the acronyms used in I is explained in II. Methods such as Convolutional Neural Networks (CNN) are separated to showcase the special configurations authors have used while implementing them, so that C-CNN is a Cascade CNN and DCNN is just a variation of CNN with a larger amount of layers than a regular CNN.

The selection process of a model often is performed by testing different models and comparing for the best accuracy results. For instance, Pacheco-Chérrez et al. [19] used vibration and acoustic signals to train several machine learning classifiers, including Naive Bayes (NB), k-Nearest Neighbors (k-NN), Decision Trees (DT), Random Forests (RF), Support Vector Machines (SVM), and Artificial Neural Networks (ANN). The results proved that SVM with a radial basis function kernel was the best approach when including acoustic signals, and also suggested these signals may be more suitable for fault detection than just vibration signals. Also, it is worth noticing that, even though SVM had the highest values of accuracy, precision, recall and F1-score, all the tested methods produced very high results.

4) Model training and validation: Model training is a critical step while developing machine learning models. Its goal is to minimize the difference between the predictions made by the model and the actual values in training data. This is typically made by the definition of a loss function, and its results help with optimization of the model. Optimization algorithms such as the method proposed by Luo [15] show an advance in the improvement of convergence rates and general performance of models. Models are trained over a training dataset and evaluated over a validation dataset in order to test if the model is able to generalize to new data and to avoid overfitting to the training data. This validation is often obtained by splitting a full dataset into two, the first part for training whilst the second, unknown to the model, is used to evaluate it.

III. Main ML techniques and approaches.

While reviewing research material that proposes a new approach to PdM in industrial machinery, mainly turbines and motors, the choice of a proper Machine Learning algorithm can be highly representative in the results obtained. According to this information, Convolutional Neural Networks and Support Vector Machines are the most frequent supervised learning models, followed by k-Nearest Neighbours, whilst eleven techniques show a minor relevance in the state-of-art research. A brief overview on SVM, k-NN and CNN is given next.

A. Support Vector Machines

Support Vector Machines are highly popular in PdM and fault detection as it shows high performance when working with high-dimensional, complex datasets . SVMs classify data by finding the decision boundary hyperplane in the input feature space that separates all classes from each other. One big advantage of the SVM model is that it contains a regularization parameter [10], which allows to control overfitting and achieve a better generalization. A disadvantage of SVMs is that they can be slow, and time complexity is usually between $O(m^2n)$ and $O(m^3n)$, having m as the number of observations and n as the number of features. Another disadvantage is the selection of a proper kernel, as there are many and it is very relevant to the model performance, [27].

B. Convolutional Neural Networks

Artificial Neural Networks consist of simple computing units that can communicate between each other simulating the synapses process that happens inside of the nervous system. Artificial Neural Networks (ANNs) work with large datasets by using complex multi-layered architectures. As complex as these networks can get, they are the opening stage for Deep Learning, and there are many types of Neural Networks that can be performed depending on the relation neurons and layers have with each other. [27]. According to Han [9], Convolutional Neural Networks consist of convolution layers that learn temporal and spectral structures of feature maps, followed by pooling layers that simplify the convolutional layers output, thus making CNNs a suitable option for signal processing. Like SVMs, CNNs can be used for modeling highly non-linear systems, and if data is available it results convenient as an update might be required.

C. k-Nearest Neighbours

Cakir [3] describes the kNN classifier as one of the most easy to applicate and simple ML classifiers. It tries to classify new samples together by comparing it to the k-nearest neighbors in the training dataset, classifying it as the most common label among said neighbors. As a simple model there are some disadvantages as the computational cost and it might not work properly with large datasets. It is often used as a baseline for more complex models.

The popularity of these techniques and their high evaluation metrics may be influenced by factors as the nature of the data; depending on the experimental setup and sensors used, the size of the training dataset could be representative to a specific algorithm; Neural Networks and SVM tend to achieve higher metrics than k-NN as the models are usually trained with vibration and sound signals, sampled at very high rates thus producing a large dataset that Neural Networks and SVMs tend to handle better than k-NN models; another relevant factor is the number of classes, this being one of the reasons why k-NN

TABLE I
DESCRIPTION OF METHODS USED IN THE REVIEWED MATERIAL

Author	Method	Signals acquired	Metrics
Babu et al. [1]	-	Current, vibration, sound	MSE: 0.0606
Cakir et al. [3]	SVM, k-NN	Vibration, sound, rotational speed, current and temperature	Acc - SVM, kNN: 0.999
Carrera et al. [4]	CNN	Sound	Acc: 0.9617
Duc Nguyen et al. [6]	k-NN, DT, RF, LG, SVM	Current	AUC: 0.995
Glowacz [8]	Nearest Neighbour	Sound	-
Han et al. [9]	k-NN, CNN, SVM	Vibration, sound	Acc: 1.000
Hesabi, Nourelfath, Hajji [11]	LSTM	-	Acc: 0.97
Li et al. [13]	SVM	Sound	Acc: 0.9592
Lu et al. [14]	CNN	Vibration	Acc: 1.000
Nikfar, Bbiencourt, Mykoniatis. [18]	SVM	Vibration, temperature	Acc: 1.000
Pacheco-Chérrez et al. [19]	SVM	Vibration, sound	Acc = 0.9666
Patel and Giri. [20]	RF	Vibration	Acc: 0.997
Patil and Wani. [21]	CNN	Sound	Acc: 0.95
Raja et al. [22]	SVC, RF, DT, k-NN	Current	Acc - SVC: 0.97, RF: 0.93
Shao et al. [24]	DCNN	Vibration, current	Acc: 0.9983
Shifat and Hur. [25]	ANN	Vibration, current	Acc: 0.98
Souza et al. [26]	CNN	Vibration	Acc: 0.9725
Strantzalis et al.	CNN	Sound	Acc: 0.978
Suawa et al. [28]	DCNN, CNN-LSTM, LSTM	Vibration, sound	Acc: 0.988, 0.935, 0.756
Wang et al. [30]	Bayesian inference	Vibration	Acc ¿ 0.9
Wang et al. [29]	C-CNN	Vibration	Acc: 0.9449
Xu et al. [31]	LSTM, GRU, TCN	Vibration	Acc: 0.9390, 0.8954, 0.9583
Yaman, Yol, Altinors [32]	SVM	Sound	Acc: 1.000
Yao et al. [33]	CNN	Sound	Acc: 0.9175

TABLE II
MEANING OF THE METHOD ACRONYMS.

Acronym	Method
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
C-CNN	Cascade Convolutional Neural Network
DCNN	Deep Convolutional Neural Network
DT	Decision Tree
GRU	Gated Recurrent Unit Neural Network
k-NN	k-Nearest Neighbors
LSTM	Long-Short Term Memory Neural Network
RF	Random Forest
SVM	Support Vector Machines
TCN	Temporal Convolution Neural Network

models are often baseline models, as it struggles with higher number of classes. A relevant result is that k-NNs tend to be very good at classifying healthy devices against failure, but struggles when classifying a fail mode.

IV. Conclusions and Future Work

As I4.0 establishes and we enter the age of data, it is important that Predictive Maintenance can act as a tool to improve industrial productivity. This review focused on electromechanical systems, and establishes audio analysis as crucial information to consider while designing a Machine Learning that can predict the current status of machinery, as the acoustic signals appeared in a considerable amount of the reviewed material for classification of faults. Support Vector Machines, Convolutional Neural Networks and k-Nearest Neighbors were found to be the most used algorithms to build classification and regression models.

As for future works, a PdM-ML algorithm for fault detection in DC motors based in acoustic signals will be designed and deployed, as this research proved it plausible and desirable. Audio signals from motors will be recorded at distances between 25 cm and 45 cm at a 16.000 kHz frequency. The systematic reviewed has established both temporal and spectral features to be relevant, so Central Tendency Measurements, skewness, correlation, kurtosis, Fast-Fourier Transform, and Power Spectrum Density must be analyzed to build the dataset. Metrics for the three main algorithms must be compared in order to select the best model for deployment.

Acknowledgment

The authors would like to recognize and express their sincere gratitude to Universidad del Cauca [ISNI: 0000 0001 2158 6862] (Colombia), for the financial support granted during this project.

References

[1] E. Babu, J. Francis, E. Thomas, R. Cherian, and S. S. Sunandhan, "Predictive Analysis of Induction Motor using Current, Vibration and Acoustic Signals," in *2022 2nd International Conference on Power*

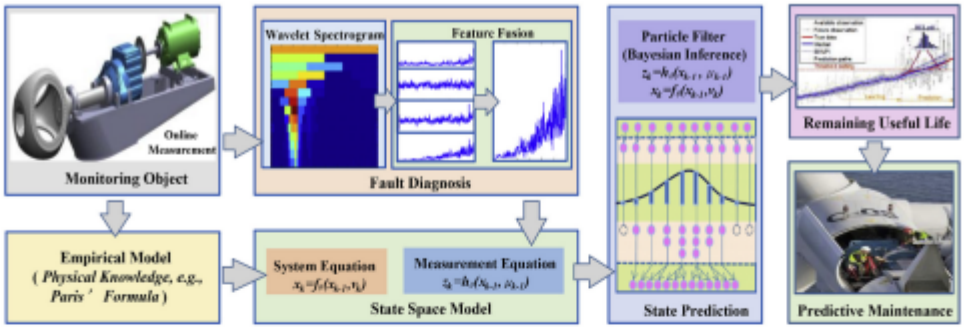

Fig. 3. Framework proposed by Wang et al. [30]

Electronics & IoT Applications in Renewable Energy and Its Control (PARC). Mathura, India: IEEE, Jan. 2022, pp. 1–7.

[2] A. L. Bowler, M. P. Pound, and N. J. Watson, "A review of ultrasonic sensing and machine learning methods to monitor industrial processes," *Ultrasonics*, vol. 124, Aug. 2022, publisher: Elsevier B.V.

[3] M. Cakir, M. A. Guvenc, and S. Mistikoglu, "The experimental application of popular machine learning algorithms on predictive maintenance and the design of IIoT based condition monitoring system," *Computers & Industrial Engineering*, vol. 151, p. 106948, Jan. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360835220306252

[4] J. D. Carrera, P. V. Matute, E. Pelaez, and F. R. Loayza, "Fault Diagnosing for Electric Motors Based in Noise Data Classification Using Deep Learning," in *2022 IEEE ANDESCON*. Barranquilla, Colombia: IEEE, Nov. 2022, pp. 1–6.

[5] T. P. Carvalho, F. A. Soares, R. Vita, R. d. P. Francisco, J. P. Basto, and S. G. Alcalá, "A systematic literature review of machine learning methods applied to predictive maintenance," *Computers and Industrial Engineering*, vol. 137, Nov. 2019, publisher: Elsevier Ltd.

[6] V. Duc Nguyen, E. Zwanenburg, S. Limmer, W. Luijben, T. Back, and M. Olhofer, "A Combination of Fourier Transform and Machine Learning for Fault Detection and Diagnosis of Induction Motors," in *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. Yinchuan, China: IEEE, Aug. 2021, pp. 344–351.

[7] V. Dutordoir, J. Hensman, M. van der Wilk, C. H. Ek, Z. Ghahramani, and N. Durrande, "Deep neural networks as point estimates for deep gaussian processes," in *Advances in Neural Information Processing Systems*, 2021.

[8] A. Glowacz, "Acoustic based fault diagnosis of three-phase induction motor," *Applied Acoustics*, vol. 137, pp. 82–89, Aug. 2018, publisher: Elsevier Ltd.

[9] S. Han, N. Mannan, D. C. Stein, K. R. Pattipati, and G. M. Bollas, "Classification and regression models of audio and vibration signals for machine state monitoring in precision machining systems," *Journal of Manufacturing Systems*, vol. 61, pp. 45–53, Oct. 2021, publisher: Elsevier B.V.

[10] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., ser. Springer Series in Statistics. New York, NY: Springer, 2009.

[11] H. Hesabi, M. Nourelfath, and A. Hajji, "A deep learning predictive model for selective maintenance optimization," *Reliability Engineering and System Safety*, vol. 219, Mar. 2022, publisher: Elsevier Ltd.

[12] W. J. Lee, H. Wu, H. Yun, H. Kim, M. B. Jun, and J. W. Sutherland, "Predictive Maintenance of Machine Tool Systems Using Artificial Intelligence Techniques Applied to Machine Condition Data," *Procedia CIRP*, vol. 80, pp. 506–511, 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2212827118312988

[13] Z. Li, R. Liu, and D. Wu, "Data-driven smart manufacturing: Tool wear monitoring with audio signals and machine learning," *Journal of Manufacturing Processes*, vol. 48, pp. 66–76, Dec. 2019, publisher: Elsevier Ltd.

[14] S. Lu, G. Qian, Q. He, F. Liu, Y. Liu, and Q. Wang, "In Situ Motor Fault Diagnosis Using Enhanced Convolutional Neural Network in an Embedded System," *IEEE Sensors Journal*, vol. 20, no. 15, pp. 8287–8296, Aug. 2020, publisher: Institute of Electrical and Electronics Engineers Inc.

[15] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in *Proceedings of the International Conference on Learning Representations*, 2019.

[16] A. A. Manjare and B. G. Patil, "A Review: Condition Based Techniques and Predictive Maintenance for Motor," in *Proceedings - International Conference on Artificial Intelligence and Smart Systems, ICAIS 2021*. Institute of Electrical and Electronics Engineers Inc., Mar. 2021, pp. 807–813.

[17] M. Nacchia, F. Fruggiero, A. Lambiase, and K. Bruton, "A systematic mapping of the advancing use of machine learning techniques for predictive maintenance in the manufacturing sector," *Applied Sciences (Switzerland)*, vol. 11, no. 6, Mar. 2021, publisher: MDPI AG.

[18] M. Nikfar, J. Bitencourt, and K. Mykoniatis, "A Two-Phase Machine Learning Approach for Predictive Maintenance of Low Voltage Industrial Motors," *Procedia Computer Science*, vol. 200, pp. 111–120, 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050922002198

[19] J. Pacheco-Chérrez, J. A. Fortoul-Díaz, F. Cortés-Santacruz, L. María Aloso-Valerdi, and D. I. Ibarra-Zarate, "Bearing fault detection with vibration and acoustic signals: Comparison among different machine leaning classification methods," *Engineering Failure Analysis*, vol. 139, p. 106515, Sep. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1350630722004897

[20] R. K. Patel and V. Giri, "Feature selection and classification of mechanical fault of an induction motor using random forest classifier," *Perspectives in Science*, vol. 8, pp. 334–337, Sep. 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2213020916300908

[21] S. Patil and K. Wani, "Gear fault detection using noise analysis and machine learning algorithm with YAMNet pretrained network," *Materials Today: Proceedings*, p. S2214785322061211, Sep. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2214785322061211

[22] H. A. Raja, H. Raval, T. Vaimann, A. Kallaste, A. Rassolkin, and A. Belahcen, "Cost-efficient real-time condition monitoring and fault diagnostics system for BLDC motor using IoT and Machine learning," in *2022 International Conference on Diagnostics in Electrical Engineering (Diagnostika)*. Pilsen, Czech Republic: IEEE, Sep. 2022, pp. 1–4.

[23] K. M. Rashid and J. Louis, "Activity identification in modular construction using audio signals and machine learning," *Automation in Construction*, vol. 119, Nov. 2020, publisher: Elsevier B.V.

[24] S. Shao, R. Yan, Y. Lu, P. Wang, and R. X. Gao, "DCNN-Based

multi-signal induction motor fault diagnosis," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 6, pp. 2658–2669, Jun. 2020, publisher: Institute of Electrical and Electronics Engineers Inc.

[25] T. A. Shifat and J. W. Hur, "ANN Assisted Multi Sensor Information Fusion for BLDC Motor Fault Diagnosis," *IEEE Access*, vol. 9, pp. 9429–9441, 2021, publisher: Institute of Electrical and Electronics Engineers Inc.

[26] R. M. Souza, E. G. Nascimento, U. A. Miranda, W. J. Silva, and H. A. Lepikson, "Deep learning for diagnosis and classification of faults in industrial rotating machinery," *Computers & Industrial Engineering*, vol. 153, p. 107060, Mar. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360835220307300

[27] A. Stetco, F. Dinmohammadi, X. Zhao, V. Robu, D. Flynn, M. Barnes, J. Keane, and G. Nenadic, "Machine learning methods for wind turbine condition monitoring: A review," *Renewable Energy*, vol. 133, pp. 620–635, Apr. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S096014811831231X

[28] P. Suawa, T. Meisel, M. Jongmanns, M. Huebner, and M. Reichenbach, "Modeling and Fault Detection of Brushless Direct Current Motor by Deep Learning Sensor Data Fusion," *Sensors*, vol. 22, no. 9, p. 3516, May 2022.

[29] F. Wang, R. Liu, Q. Hu, and X. Chen, "Cascade Convolutional Neural Network with Progressive Optimization for Motor Fault Diagnosis under Nonstationary Conditions," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2511–2521, Apr. 2021, publisher: IEEE Computer Society.

[30] J. Wang, Y. Liang, Y. Zheng, R. X. Gao, and F. Zhang, "An integrated fault diagnosis and prognosis approach for predictive maintenance of wind turbine bearing with limited samples," *Renewable Energy*, vol. 145, pp. 642–650, Jan. 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0960148119309371

[31] Z. Xu, D. Yu, and Y. Hu, "Motor Fault Diagnosis Method Based on Deep Learning," in *2022 11th International Conference of Information and Communication Technology (ICTech)*. Wuhan, China: IEEE, Feb. 2022, pp. 236–239. [Online]. Available: https://ieeexplore.ieee.org/document/9849605/

[32] O. Yaman, F. Yol, and A. Altinors, "A Fault Detection Method Based on Embedded Feature Extraction and SVM Classification for UAV Motors," *Microprocessors and Microsystems*, vol. 94, p. 104683, Oct. 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0141933122002137

[33] Y. Yao, G. Gui, S. Yang, and S. Zhang, "An adaptive anti-noise network with recursive attention mechanism for gear fault diagnosis in real-industrial noise environment condition," *Measurement*, vol. 186, p. 110169, Dec. 2021. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0263224121010848

[34] Z. M. Çınar, A. Abdussalam Nuhu, Q. Zeeshan, O. Korhan, M. Asmael, and B. Safaei, "Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0," *Sustainability*, vol. 12, no. 19, p. 8211, Oct. 2020. [Online]. Available: https://www.mdpi.com/2071-1050/12/19/8211

6.4. Códigos en MATLAB.

6.4.1. Código de generación del dataset.

```
clear; clc;
% Carga de archivos de audio
audioFolders = {'Healthy','Propeller','Bearing'};

dataset3 = cell(0,21);

for idx = 1:length(audioFolders)
    currentFolder = audioFolders{idx};
    subFolders = dir(currentFolder);
    subFolders = subFolders([subFolders.isdir]);
    subFolders = subFolders(~ismember({subFolders.name}, {'.', '..'}));

    for subIdx = 1:length(subFolders)
        subFolder = fullfile(currentFolder,subFolders(subIdx).name);
        files = dir(fullfile(subFolder,'*.wav'));
        length(files)
        for fileIdx = 1:length(files)
            fpath = fullfile(subFolder,files(fileIdx).name);
            [audiosignal,fs] = audioread(fpath);

            %Resamplear si fs != 16 kHz.

            if fs ~= 16000
                resample(audiosignal,16000,fs);
                fs=16000;
            end


            meanVal = mean(audiosignal);
            harmonRatio = std(harmonicRatio(audiosignal,fs));
            % Normalización
            audiosignal = (audiosignal-min(audiosignal))
```

```
            /(max(audiosignal)-min(audiosignal));

            stDev = std(audiosignal);

            spectCent = spectralCentroid(audiosignal,fs);
            meanSpectCent = mean(spectCent);
            maxSpectCent = max(spectCent);
            minSpectCent = min(spectCent);
            stDevSpectCent = std(spectCent);
            spectFlat = spectralFlatness(audiosignal,fs);
            maxSpectEnt = max(pentropy(audiosignal(:,1),fs));
            minSpectEnt = min(pentropy(audiosignal(:,1),fs));
            meanSpectEnt = mean(pentropy(audiosignal(:,1),fs));
            audioKurtosis = kurtosis(audiosignal);
            maxKurtosis = max(audioKurtosis);
            minKurtosis = min(audioKurtosis);
            audioSkewness = skewness(audiosignal);
            maxSkewness = max(audioSkewness);
            minSkewness = min(audioSkewness);
            zcr = zerocrossrate(audiosignal);
            [psd,~] = pwelch(audiosignal(:,1),hamming(1024),512,[],fs);
            if strcmp(currentFolder,'Bearing')
                status = 3;
            elseif strcmp(currentFolder,'Healthy')
                status = 1;
            else
                status = 2;
            end
            newRow = {files(fileIdx).name,status,mean(spectCent(:,1)),
            min(spectCent(:,1)),max(spectCent(:,1)),std(spectCent(:,1)),mean(spect
            %if newRow{2} ~= 0
                dataset3 = [dataset3;newRow];
            %end
        end
    end
```

```
end
save('dataset_hpb',"dataset3")
```

6.4.2. Código de Análisis Exploratorio de Datos.

```
clear;clc;
load('newDS_HPB.mat')

featureNames =
{'minSpectCent','maxSpectCent','minSpectFlat', 'minSpectEnt','meanSpectEnt','StDev

minSpectCentH = newDS(1:22,2);
minSpectCentF = newDS(23:36,2);
minSpectCentB = newDS(37:43,2);

maxSpectCentH = newDS(1:22,3);
maxSpectCentF = newDS(23:36,3);
maxSpectCentB = newDS(37:43,3);

minSpectFlatH = newDS(1:22,4);
minSpectFlatF = newDS(23:36,4);
minSpectFlatB = newDS(37:43,4);

minSpectEntH = newDS(1:22,5);
minSpectEntF = newDS(23:36,5);
minSpectEntB = newDS(37:43,5);

meanSpectEntH = newDS(1:22,6);
meanSpectEntF = newDS(23:36,6);
meanSpectEntB = newDS(37:43,6);

StDevH = newDS(1:22,7);
StDevF = newDS(23:36,7);
StDevB = newDS(37:43,7);
```

```
KurtH = newDS(1:22,8);
KurtF = newDS(23:36,8);
KurtB = newDS(37:43,8);

% Create subplots
figure;

% Define colors for the plots
colors = {'k', 'r', 'b'};

for i = 1:length(featureNames)
    subplot(2,4,i);
    hold on;

    % Plot healthy data
    [f_h, x_h] = ksdensity(eval([featureNames{i} 'H']));
    plot(x_h, f_h, '-r');

    % Plot faulty data
    [f_f, x_f] = ksdensity(eval([featureNames{i} 'F']));
    plot(x_f, f_f, '-b');

    [f_f, x_f] = ksdensity(eval([featureNames{i} 'B']));
    plot(x_f, f_f, '-k');

    % Add title and legend
    title(featureNames{i});
    legend('Healthy', 'Propeller','Bearing');

    hold off;
end

% Adjust layout
sgtitle('Kernel Density Plots');
```

```
figure;

for i = 1:length(featureNames)
    subplot(2,4,i);

    % Combine Healthy and Faulty data for boxplot
    data = [newDS(1:22, i+1); newDS(23:36, i+1); newDS(37:43, i+1)];
    labels = [ones(22,1); 2*ones(14,1); 3*ones(7,1)];

    % Create box plot
    boxplot(data, labels, 'Labels', {'Healthy', 'Propeller','Bearing'});

    % Set the title
    title(featureNames{i});
end

% Adjust layout
%sgtitle('Box Plots for Healthy and Faulty Observations');
```

6.4.3. Entrenamiento de los modelos kNN.

```
clear;clc;
load("newDS_HPB.mat")
features = newDS(:,2:end);
labels = newDS(:,1);

trainRatio = 0.7;
numObservations = size(features, 1);
numTrain = round(trainRatio * numObservations);
indices = randperm(numObservations);
trainIndices = indices(1:numTrain);
testIndices = indices(numTrain+1:end);
trainFeatures = features(trainIndices, :);
```

```
trainLabels = labels(trainIndices, :);
testFeatures = features(testIndices, :);
testLabels = labels(testIndices, :);

% Set the range of neighbors to try
neighbors = 1:10;

% Initialize variables to store evaluation results
accuracy = zeros(1, length(neighbors));

% Iterate over each number of neighbors
for k = neighbors
    % Train the KNN model
    knnModel = fitcknn(trainFeatures, trainLabels, 'NumNeighbors', k);

    % Predict labels for the test set
    predictedLabels = predict(knnModel, testFeatures);

    % Compute the accuracy
    accuracy(k) = sum(predictedLabels == testLabels) / numel(testLabels);
end

% Find the best number of neighbors
bestNeighbors = find(accuracy == max(accuracy));

% Select the best number of neighbors
bestK = bestNeighbors(1);

% Train the final KNN model using the best number of neighbors
knnModel = fitcknn(trainFeatures, trainLabels, 'NumNeighbors', bestK);

% Predict labels for the test set using the best model
predictedLabels = predict(knnModel, testFeatures);

% Compute the confusion matrix
```

```
C = confusionmat(testLabels, predictedLabels);
cc = confusionchart(testLabels,predictedLabels);
% Calculate the accuracy, precision, recall, and F1-score
accuracy = sum(diag(C)) / sum(C(:));
precision = diag(C) ./ sum(C, 1)';
recall = diag(C) ./ sum(C, 2);
f1Score = 2 * (precision .* recall) ./ (precision + recall);

% Display the confusion matrix and evaluation metrics
disp('Confusion Matrix:');
disp(C);
disp('Evaluation Metrics:');
disp(['Accuracy: ', num2str(accuracy)]);
disp('Precision:');
disp(precision);
disp('Recall:');
disp(recall);
disp('F1-Score:');
disp(f1Score);
```

6.4.4. Entrenamiento de los modelos DT.

```
clear;clc;
load("newDS_HPB.mat")
features = newDS(:,2:end);
labels = newDS(:,1);


% Split the dataset into training and testing sets
trainRatio = 0.70;


numObservations = size(features, 1);
numTrain = round(trainRatio * numObservations);
```

```
indices = randperm(numObservations);
trainIndices = indices(1:numTrain);
testIndices = indices(numTrain+1:end);
trainFeatures = features(trainIndices, :);
trainLabels = labels(trainIndices, :);
testFeatures = features(testIndices, :);
testLabels = labels(testIndices, :);

% Train the Decision Tree model
dtModel = fitctree(trainFeatures, trainLabels);

% Predict the labels for the test set using the trained model
predictedLabels = predict(dtModel, testFeatures);

% Evaluate the performance of the model
accuracy = sum(predictedLabels == testLabels) / numel(testLabels);
fprintf('Accuracy: %.2f%%\n', accuracy * 100);



% Compute the confusion matrix
C = confusionmat(testLabels, predictedLabels);
Cc = confusionchart(testLabels,predictedLabels);
% Display the confusion matrix
disp('Confusion Matrix:');
disp(C);
accuracy = sum(diag(C)) / sum(C(:));
precision = diag(C) ./ sum(C, 1)';
recall = diag(C) ./ sum(C, 2);
f1Score = 2 * (precision .* recall) ./ (precision + recall);

% Display the evaluation metrics
disp('Evaluation Metrics:');
disp(['Accuracy: ', num2str(accuracy)]);
disp('Precision:');
disp(precision);
```

```
disp('Recall:');
disp(recall);
disp('F1-Score:');
disp(f1Score);
```

6.4.5. Entrenamiento de los modelos SVM.

```
clear;clc;
load("newDS_HPB.mat")
features = newDS(:,2:end);
labels = newDS(:,1);



trainRatio = 0.70;
numObservations = size(features, 1);
numTrain = round(trainRatio * numObservations);
indices = randperm(numObservations);
trainIndices = indices(1:numTrain);
testIndices = indices(numTrain+1:end);
trainFeatures = features(trainIndices, :);
trainLabels = labels(trainIndices, :);
testFeatures = features(testIndices, :);
testLabels = labels(testIndices, :);

% Train the SVM model
svmModel = fitcecoc(trainFeatures, trainLabels);

% Predict labels for the test set
predictedLabels = predict(svmModel, testFeatures);

% Compute the confusion matrix
C = confusionmat(testLabels, predictedLabels);
cc = confusionchart(testLabels,predictedLabels);
% Calculate the accuracy, precision, recall, and F1-score
accuracy = sum(diag(C)) / sum(C(:));
```

```
precision = diag(C) ./ sum(C, 1)';
recall = diag(C) ./ sum(C, 2);
f1Score = 2 * (precision .* recall) ./ (precision + recall);

% Calculate the overall metrics
avgPrecision = mean(precision);
avgRecall = mean(recall);
avgF1Score = mean(f1Score);

% Display the confusion matrix
disp('Confusion Matrix:');
disp(C);

% Display the evaluation metrics
disp('Evaluation Metrics:');
disp(['Accuracy: ', num2str(accuracy)]);
disp('Precision:');
disp(precision);
disp(['Average Precision: ', num2str(avgPrecision)]);
disp('Recall:');
disp(recall);
disp(['Average Recall: ', num2str(avgRecall)]);
disp('F1-Score:');
disp(f1Score);
disp(['Average F1-Score: ', num2str(avgF1Score)]);
```

6.5. Código del MVP en MATLAB App Designer.

```
classdef MVP < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        PredictionResultLabel       matlab.ui.control.Label
```

```
    PredictionLabel             matlab.ui.control.Label
    PredictButton               matlab.ui.control.Button
    loadFileButton              matlab.ui.control.Button
    ModelDropDown               matlab.ui.control.DropDown
    ModelDropDownLabel          matlab.ui.control.Label
    ControlgroupDropDown        matlab.ui.control.DropDown
    ControlgroupDropDownLabel   matlab.ui.control.Label
end


properties (Access = public)
    SelectedFilePath string % Description
    CurrentGroup string
    CurrentModel string
end


% Callbacks that handle component events
methods (Access = private)

    % Button pushed function: loadFileButton
    function loadFileButtonPushed(app, event)
        [fileName, filePath] = uigetfile('*.wav');
        if fileName ~= 0
            app.SelectedFilePath = fullfile(filePath, fileName);
        end
    end

    % Value changed function: ControlgroupDropDown
    function ControlgroupDropDownValueChanged(app, event)
        selectedGroup = app.ControlgroupDropDown.Value;
        app.CurrentGroup = selectedGroup;
    end

    % Value changed function: ModelDropDown
```

```
function ModelDropDownValueChanged(app, event)
    selectedModel = app.ModelDropDown.Value;
    app.CurrentModel = selectedModel;
end

% Button pushed function: PredictButton
function PredictButtonPushed(app, event)
    function audioData = preprocessData(filePath)
        [audiosignal,fs] = audioread(filePath);
        if fs ~= 16000
            resample(audiosignal,16000,fs);
            fs=16000;
        end
        audiosignal =
        (audiosignal-min(audiosignal))/(max(audiosignal)-min(audiosignal))

        spectCent = spectralCentroid(audiosignal,fs);
        minSpectCent = min(spectCent);
        maxSpectCent = max(spectCent);
        spectFlat = spectralFlatness(audiosignal,fs);
        minSpectFlat = min(spectFlat);
        minSpectEnt = min(pentropy(audiosignal(:,1),fs));
        meanSpectEnt = mean(pentropy(audiosignal(:,1),fs));
        stDev = std(audiosignal);
        kurt = kurtosis(audiosignal);
        audioData
        = [minSpectCent,maxSpectCent,minSpectFlat,minSpectEnt,meanSpectEnt
    end

    disp(['Current Group: ' app.CurrentGroup]);
    disp(['Current Model: ' app.CurrentModel]);
    currentModel = app.CurrentModel{1};

    modelFile = fullfile(app.CurrentGroup, [currentModel 'Model.mat']);
    disp(['ModelFile: ',modelFile]);
```

```
    switch app.CurrentModel
        case 'kNN'
            variableName = 'knnModel';
        case 'SVM'
            variableName = 'svmModel';
        case 'DT'
            variableName = 'dtModel';
        otherwise
            disp('Invalid model selected');
            return;
    end
    if exist(modelFile, 'file') == 2
        % Load the selected model and get the specified variable
        loadedData = load(modelFile, variableName);
        model = loadedData.(variableName);
    else
        disp(['Error: Model file not found - ' modelFile]);
        return;
    end
    % Preprocess the data
    audioData = preprocessData(app.SelectedFilePath);

    % Perform prediction
    prediction = predict(model, audioData);

    % Display the prediction result
    displayResult(app, prediction);
end



function displayResult(app, prediction)

    if strcmp(app.CurrentGroup, 'Group HF')
```

```
                if prediction == 1
                    app.PredictionResultLabel.Text = 'Healthy';
                else
                    app.PredictionResultLabel.Text = 'Faulty';
                end
            elseif strcmp(app.CurrentGroup, 'Group HP')
                if prediction == 1
                    app.PredictionResultLabel.Text = 'Healthy';
                else
                    app.PredictionResultLabel.Text = 'Propeller';
                end
            elseif strcmp(app.CurrentGroup, 'Group HPB')
                switch prediction
                    case 1
                        app.PredictionResultLabel.Text = 'Healthy';
                    case 2
                        app.PredictionResultLabel.Text = 'Propeller';
                    case 3
                        app.PredictionResultLabel.Text = 'Bearing';
                end
            end
            app.PredictionResultLabel.Visible = "on";
            app.PredictionLabel.Visible = "on";
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 640 480];
```

```
app.UIFigure.Name = 'MATLAB App';

% Create ControlgroupDropDownLabel
app.ControlgroupDropDownLabel = uilabel(app.UIFigure);
app.ControlgroupDropDownLabel.HorizontalAlignment = 'right';
app.ControlgroupDropDownLabel.Position = [39 402 78 22];
app.ControlgroupDropDownLabel.Text = 'Control group';

% Create ControlgroupDropDown
app.ControlgroupDropDown = uidropdown(app.UIFigure);
app.ControlgroupDropDown.Items =
{'Group HF', 'Group HP', 'Group HPB'};
app.ControlgroupDropDown.ValueChangedFcn =
createCallbackFcn(app, @ControlgroupDropDownValueChanged, true);
app.ControlgroupDropDown.Position = [132 402 100 22];
app.ControlgroupDropDown.Value = 'Group HF';

% Create ModelDropDownLabel
app.ModelDropDownLabel = uilabel(app.UIFigure);
app.ModelDropDownLabel.HorizontalAlignment = 'right';
app.ModelDropDownLabel.Position = [86 348 38 22];
app.ModelDropDownLabel.Text = 'Model';

% Create ModelDropDown
app.ModelDropDown = uidropdown(app.UIFigure);
app.ModelDropDown.Items = {'kNN', 'DT', 'SVM'};
app.ModelDropDown.ValueChangedFcn =
createCallbackFcn(app, @ModelDropDownValueChanged, true);
app.ModelDropDown.Position = [139 348 100 22];
app.ModelDropDown.Value = 'kNN';

% Create loadFileButton
app.loadFileButton = uibutton(app.UIFigure, 'push');
app.loadFileButton.ButtonPushedFcn =
createCallbackFcn(app, @loadFileButtonPushed, true);
```

```
            app.loadFileButton.Position = [458 379 100 23];
            app.loadFileButton.Text = 'Select audio file';

            % Create PredictButton
            app.PredictButton = uibutton(app.UIFigure, 'push');
            app.PredictButton.ButtonPushedFcn =
            createCallbackFcn(app, @PredictButtonPushed, true);
            app.PredictButton.Position = [271 291 100 23];
            app.PredictButton.Text = 'Predict';

            % Create PredictionLabel
            app.PredictionLabel = uilabel(app.UIFigure);
            app.PredictionLabel.Visible = 'off';
            app.PredictionLabel.Position = [188 221 76 40];
            app.PredictionLabel.Text = 'Prediction:';

            % Create PredictionResultLabel
            app.PredictionResultLabel = uilabel(app.UIFigure);
            app.PredictionResultLabel.Visible = 'off';
            app.PredictionResultLabel.Position = [304 230 98 22];
            app.PredictionResultLabel.Text = 'Healthy';

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = MVP

            % Create UIFigure and components
            createComponents(app)
```

```
            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

6.6. Código de generación de modelos en Python.

```
from scipy import io
import pandas as pd
from sklearn import svm,metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,
ConfusionMatrixDisplay,classification_report,recall_score,accuracy_score,precision
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
import numpy as np
import seaborn as sns
from joblib import dump
# En este código se generan los modelos de sano vs fallo.
mat_contents = io.loadmat('newDS_HPB.mat')
```

```
newDS = mat_contents['newDS']

#newDS = newDS[(newDS[:, 0] == 1) | (newDS[:, 0] == 2)]

df = pd.DataFrame(newDS, columns = ['Class','minSpectCent','maxSpectCent',
'minSpectFlat','minSpectEnt','meanSpectEnt','StDev','Kurtosis'])
df['Class'] = df['Class'].astype('int')



X = df[['minSpectCent','maxSpectCent','minSpectFlat',
'minSpectEnt','meanSpectEnt',
'StDev','Kurtosis']]
y = df['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,
random_state=23)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Modelo SVM 2.

svmModel = svm.SVC(kernel='rbf')
svmModel.fit(X_train,y_train)

yhat = svmModel.predict(X_test)
accuracy = svmModel.score(X_test,y_test)

print(accuracy)

import seaborn as sns
plt.figure(figsize=(4,3))
```

```
sns.heatmap(confusion_matrix(y_test,yhat),
            annot=True,fmt='d',linecolor='k',linewidths=3)
plt.title("SVM confusion matrix",fontsize=14)
plt.show()


report = classification_report(y_test,yhat)
print(report)

dump(svmModel,'svmModel3.joblib')


# Modelo kNN

""" k_values = [i for i in range (1,15)]
scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    score = cross_val_score(knn, X, y, cv=5)
    scores.append(np.mean(score))

sns.lineplot(x = k_values, y = scores, marker = 'o')
plt.xlabel("K Values")
plt.ylabel("Accuracy Score")
plt.show()

best_index = np.argmax(scores)
best_k = k_values[best_index]

knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)


print("Accuracy:", accuracy)

plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test,y_pred),
            annot=True,fmt='d',linecolor='k',linewidths=3)
plt.title("kNN confusion matrix",fontsize=14)
plt.show()

report = classification_report(y_test,y_pred)
print(report)

dump(knn,'knnModel3.joblib') """

# DT Model
"""dtModel = DecisionTreeClassifier(random_state=23)
dtModel.fit(X_train,y_train)

yhat = dtModel.predict(X_test)

accuracy = dtModel.score(X_test,y_test)

print(accuracy)

plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test,yhat),
            annot=True,fmt='d',linecolor='k',linewidths=3)
plt.title("DT confusion matrix",fontsize=14)
plt.show()


report = classification_report(y_test,yhat)
print(report)
```

```
dump(dtModel,'dtModel3.joblib')"""
```

6.7. Código del archivo views.py en Django.

```
from django.shortcuts import render
from django.http import HttpResponseRedirect,JsonResponse
from django.urls import reverse
from .models import Motor
import pyrebase
import joblib
from scipy.stats import kurtosis
import numpy as np
import pyaudio
import librosa
import os
import json
import glob
# Create your views here.
from django.http import HttpResponse
from datetime import datetime


MODEL_DIR = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'ml_models')

CHUNK = 1024
FORMAT = pyaudio.paFloat32
CHANNELS = 1
RATE = 16000
RECORD_SECONDS = 10


config = {
    "apiKey": "AIzaSyAJHYrVgc3buthhVj9ne0CYxg3vYTF3fE8",
    "authDomain": "bldctesis.firebaseapp.com",
```

```
    "databaseURL": "https://bldctesis-default-rtdb.firebaseio.com",
    "projectId": "bldctesis",
    "storageBucket": "bldctesis.appspot.com",
    "messagingSenderId": "164535771215",
    "appId": "1:164535771215:web:4bfcf62b6c729d4633ca3a"
}

firebase=pyrebase.initialize_app(config)
authe = firebase.auth()
database=firebase.database()




def index(request):
    motors = []
    motorspin = []
    motorstatus = []
    for motor in Motor.objects.all():
        motors.append(motor)
        motorspin.append(motor.pin)
        motorstatus.append(motor.status)
        if motor.firebaseExists is False:

            database.child(motor.name).set({
                "Pin":motor.pin,
                "Fault":motor.fault,
                "Status":motor.status,
                "Test":False,
            })
            motor.firebaseExists = True
            motor.save()
        else:
            pass
    return render(request,"main/index.html",{
```

```
            'motors':motors,
            'motorspin':motorspin,
            'motorstatus':motorstatus,
        })

    def createMotor(request):
        if request.method == "POST":
            if request.headers.get('x-requested-with') == 'XMLHttpRequest':

                title = request.POST.get('title')
                print(title)
                pin = request.POST.get('pin')
                print(pin)
                if request.POST.get('status') == "false":
                    status = False
                else:
                    status = True
                fault = request.POST.get('fault')
                print(int(fault))
                motor = Motor.objects.create(name=title)
                motor.pin = int(pin)
                motor.fault = int(fault)
                motor.status = status
                motor.save()
                response = {
                    'msg':'Motor saved.'
                }
                return JsonResponse(response)
            else:
                response = {
                    'msg':'Error'
                }
                return JsonResponse(response)
        else:
            return HttpResponseRedirect(reverse(index))
```

```
def record_audio():
    p = pyaudio.PyAudio()

    device_index = 1

    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=RATE,
                    input=True,
                    input_device_index=device_index,
                    frames_per_buffer=CHUNK)

    frames = []

    print("Recording...")

    for _ in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
        data = stream.read(CHUNK)
        frames.append(data)

    print("Recording finished.")

    stream.stop_stream()
    stream.close()
    p.terminate()

    audio_data = b''.join(frames)
    return audio_data

def preProcessing(audio_data):
    sr = RATE
    spectral_centroids = librosa.feature.spectral_centroid(y=audio_data,sr=sr)[0]
    spectral_flatness = librosa.feature.spectral_flatness(y=audio_data)[0]
```

```
    spectrogram = np.abs(librosa.stft(audio_data)) ** 2
    normalized_spectrogram = spectrogram / np.sum(spectrogram)
    # Normalize the spectrogram
    spectral_entropy = -np.sum(normalized_spectrogram *
    np.log2(normalized_spectrogram + 1e-6))
    std_dev = (np.std(audio_data))
    kurto = kurtosis(audio_data)
    min_spectral_centroid = np.min(spectral_centroids)
    max_spectral_centroid = np.max(spectral_centroids)

    min_spectral_flatness = np.min(spectral_flatness)

    min_spectral_entropy = np.min(spectral_entropy)
    mean_spectral_entropy = np.mean(spectral_entropy)

    features_vector = np.array([
    min_spectral_centroid,
    max_spectral_centroid,
    min_spectral_flatness,
    min_spectral_entropy,
    mean_spectral_entropy,
    std_dev,
    kurto
    ])

    return features_vector
def checkMotor(request):
    if request.method == "POST":
        if request.headers.get('x-requested-with') == 'XMLHttpRequest':
            mid = int(request.POST.get('motorId'))
            currentMotor = Motor.objects.get(id=mid)
            #print(currentMotor.name)
            selector1 = request.POST.get('selector1')
            selector2 = request.POST.get('selector2')
            #print(selector1)
```

```
            #print(selector2)
            motors = database.get().val()
            for motor_key, motor_data in motors.items():
                #print(motor_key)
                if motor_key == currentMotor.name:
                    # Update the Test field to True for the specific motor
                    database.child(motor_key).update({"Status": True})
                else:
                    # Update the Test field to False for other motors
                    database.child(motor_key).update({"Status": False})
            audio_data = record_audio()
            audio_array = np.frombuffer(audio_data, dtype=np.float32)
            audio_array =
            (audio_array - np.min(audio_array))/(np.max(audio_array)-np.min(audio_
            featVect = preProcessing(audio_array)

            featVect = featVect.reshape(1,-1)
            model_path = os.path.join
            (MODEL_DIR, selector2, f'{selector2}_{selector1}.joblib')
            print(model_path)
            if os.path.exists(model_path):
                model = joblib.load(model_path)
                result = model.predict(featVect)
                print(result.tolist())
            else:
                print("No result")

            response = {
                'msg':'Ok',
                'result': result.tolist()[0]
            }
            return JsonResponse(response)
        else:
            response = {
                'msg' : 'Error'
```

```
            }
            return JsonResponse(response)

        else:
            return HttpResponseRedirect(reverse(index))

    def autoMode(request):
        if request.method == "POST":
            if request.headers.get('x-requested-with') == 'XMLHttpRequest':
                mid = int(request.POST.get('motorId'))
                currentMotor = Motor.objects.get(id=mid)
                #print(currentMotor.name)
                motors = database.get().val()
                for motor_key, motor_data in motors.items():
                    #print(motor_key)
                    if motor_key == currentMotor.name:
                        # Update the Test field to True for the specific motor
                        database.child(motor_key).update({"Status": True})
                    else:
                        # Update the Test field to False for other motors
                        database.child(motor_key).update({"Status": False})
                audio_data = record_audio()
                audio_array = np.frombuffer(audio_data, dtype=np.float32)
                audio_array = (audio_array - np.min(audio_array))

                /(np.max(audio_array)-np.min(audio_array))
                featVect = preProcessing(audio_array)
                featVect = featVect.reshape(1,-1)
                model_files =

                [f for f in os.listdir(MODEL_DIR) if f.endswith('.joblib')]
                predictions = {}

                for root, dirs, files in os.walk(MODEL_DIR):
                    for model_file in files:
```

```
                    if model_file.endswith('.joblib'):
                        model_path = os.path.join(root, model_file)
                        print(model_path)
                        # Load the model
                        model = joblib.load(model_path)
                        result = model.predict(featVect)
                        key = model_file
                        predictions[key] = int(result[0])
            response = {
                'msg':'Ok',
                'result': json.dumps(predictions),
            }
            print(predictions)
            return JsonResponse(response)
        else:
            response = {
                'msg' : 'Error'
            }
            return JsonResponse(response)

    else:
        return HttpResponseRedirect(reverse(index))


def decision(request):
    if request.method == "POST":
        if request.headers.get('x-requested-with') == 'XMLHttpRequest':
            mid = int(request.POST.get('motorId'))
            button = int(request.POST.get('button'))
            currentMotor = Motor.objects.get(id=mid)
            #print(currentMotor.name)
            if(button == 1):
                currentMotor.status=False
            else:
                currentMotor.maintenanceScheduled = True
```

```
                currentMotor.maintenanceDate =
                datetime.strptime(request.POST.get('date'),'%m/%d/%Y').date()
            currentMotor.save()
            response = {
                'msg':'Ok, maintenance scheduled or motor stopped.',
            }
            return JsonResponse(response)
        else:
            response = {
                'msg' : 'Error'
            }
            return JsonResponse(response)

    else:
        return HttpResponseRedirect(reverse(index))


def maintList(request):
    motors = []
    motorspin = []
    motorstatus = []
    motorsfailure = []
    motorsdate = []
    for motor in Motor.objects.filter(maintenanceScheduled=True):
        print(motor.name)
        motors.append(motor)
        motorspin.append(motor.pin)
        motorstatus.append(motor.status)
        motorsfailure.append(motor.faultMode)
        motorsdate.append(motor.maintenanceDate)
    print(len(motors))
    return render(request,"main/maintenance.html",{
        'motors':motors,
        'motorspin':motorspin,
        'motorstatus':motorstatus,
```

```
        'motorsfailure':motorsfailure,
        'motorsdate':motorsdate,
    })
```

6.8. Estructura del modelo Motor en Django.

```
from django.db import models

# Create your models here.
class Motor(models.Model):
    name=models.CharField(max_length=20,blank=True,null=True)
    pin = models.IntegerField(blank=False,null=True,default=1)
    faultMode = (
        ('1',1),
        ('2',2),
        ('3',3),
    )
    fault = models.IntegerField(blank=True,null=True)
    status = models.BooleanField(blank=True,null=False,default=True)
    firebaseExists = models.BooleanField(blank=True,null=False,default=False)
    maintenanceScheduled =
    models.BooleanField(blank=True,null=False,default=False)
    maintenanceDate = models.DateField()
```

6.9. Código de la ESP32 para la adquisición de datos en Blynk.

```
#include <ESP32Servo.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
// Define the pin for the ESC signal
const int escPin = 4;
#define BLYNK_TEMPLATE_ID "TMPL2Z60p-Q6M"
```

```
#define BLYNK_TEMPLATE_NAME "BLDC"
#define BLYNK_AUTH_TOKEN "RNa_eaw--yARKMfiyrlwMSxNEpD_aePt"
#define BLYNK_PRINT Serial
// Create a Servo object
Servo esc;


char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Stiw";
char pass[] = "1007784376";

void setup() {
  // Initialize the serial communication
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass,"blynk.cloud",80);
  // Attach the ESC to the signal pin
  esc.attach(escPin);
  delay(1000);
}

void loop() {
  // Read the potentiometer value
  Blynk.run();

}
BLYNK_WRITE(V4){
  int speed = param.asInt();
  esc.writeMicroseconds(speed);
  Serial.print("Pin value: ");
  Serial.print(speed);
  delay(100);
}
```

6.10. Código de la ESP32 para la aplicación IoT.

```
#include <ESP32Servo.h>
#include <FirebaseESP32.h>
#include <WiFi.h>
#include <WiFiClient.h>
//#include <Firebase_ESP_Client.h>

#include "addons/RTDBHelper.h"
#include "addons/TokenHelper.h"
#define API_KEY "AIzaSyAJHYrVgc3buthhVj9ne0CYxg3vYTF3fE8"

#define DATABASE_URL "https://bldctesis-default-rtdb.firebaseio.com/"

const int escPin1 = 5;
const int escPin2 = 14;
const int escPin3 = 25;

char ssid[] = "Stiw";
char pass[] ="1007784376";

bool signupOK = false;

FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

Servo esc1,esc2,esc3;



void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  esc1.attach(escPin1);
```

```
esc2.attach(escPin2);
esc3.attach(escPin3);
// pinMode(escPin1,OUTPUT);
// pinMode(escPin2,OUTPUT);
// pinMode(escPin3,OUTPUT);
WiFi.begin(ssid,pass);
Serial.print("Connecting to WiFi");
while(WiFi.status()!=WL_CONNECTED){
  Serial.print(".");
  delay(300);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();

config.api_key = API_KEY;
config.database_url = DATABASE_URL;

if(Firebase.signUp(&config,&auth,"","")){
  Serial.println("Ok");
  signupOK = true;
}
else{
  Serial.printf("%s\n",config.signer.signupError.message.c_str());
}

config.token_status_callback = tokenStatusCallback;

Firebase.begin(&config,&auth);
Firebase.reconnectWiFi(true);
delay(1000);
esc1.writeMicroseconds(1600);
esc2.writeMicroseconds(1600);
esc3.writeMicroseconds(1600);
```

```
}

void loop() {
  // put your main code here, to run repeatedly:
  if (Firebase.getBool(fbdo, "/Motor 1/Test")) {
    digitalWrite(escPin1, HIGH);
    esc1.writeMicroseconds(1600);
    digitalWrite(escPin2, LOW);
    digitalWrite(escPin3, LOW);
  } else if (Firebase.getBool(fbdo, "/Motor 2/Test")) {
    digitalWrite(escPin1, LOW);
    digitalWrite(escPin2, HIGH);
    esc2.writeMicroseconds(1600);
    digitalWrite(escPin3, LOW);
  } else if (Firebase.getBool(fbdo, "/Motor 3/Test")) {
    digitalWrite(escPin1, LOW);
    digitalWrite(escPin2, LOW);
    digitalWrite(escPin3, HIGH);
    esc3.writeMicroseconds(1600);
  } else {
    if(Firebase.getBool(fbdo, "/Motor 1/Status")){
      digitalWrite(escPin1,HIGH);
      esc1.writeMicroseconds(1600);
    }
    else{
      digitalWrite(escPin1,LOW);
    }

    if(Firebase.getBool(fbdo, "/Motor 2/Status")){
      digitalWrite(escPin2,HIGH);
      esc2.writeMicroseconds(1600);
    }
    else{
      digitalWrite(escPin2,LOW);
    }
```

```
        if(Firebase.getBool(fbdo, "Motor 3/Status")){
          digitalWrite(escPin3,HIGH);
          esc3.writeMicroseconds(1600);
        }
        else{
          digitalWrite(escPin3,LOW);
        }

      }
    }
```