ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE MECÁNICA

CARRERA MANTENIMIENTO INDUSTRIAL

"CLASIFICACIÓN DE FALLAS EN RODAMIENTOS UTILIZANDO DATOS DE ANÁLISIS DE ULTRASONIDO BASADO EN UNA MÁQUINA DE APRENDIZAJE EXTREMO"

Trabajo de Integración Curricular

Tipo: Proyecto de Investigación

Presentado para optar al grado académico de:

INGENIERO EN MANTENIMIENTO INDUSTRIAL

AUTOR:

DIEGO HERNAN BARAHONA DEFAZ

Riobamba – Ecuador

2023

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE MECÁNICA

CARRERA MANTENIMIENTO INDUSTRIAL

"CLASIFICACIÓN DE FALLAS EN RODAMIENTOS UTILIZANDO DATOS DE ANÁLISIS DE ULTRASONIDO BASADO EN UNA MÁQUINA DE APRENDIZAJE EXTREMO"

Trabajo de Integración Curricular

Tipo: Proyecto de Investigación

Presentado para optar al grado académico de:

INGENIERO EN MANTENIMIENTO INDUSTRIAL

AUTOR: DIEGO HERNAN BARAHONA DEFAZ

DIRECTOR: ING. FÉLIX ANTONIO GARCÍA MORA

Riobamba – Ecuador

2023

Yo, DIEGO HERNAN BARAHONA DEFAZ, declaro que el presente Trabajo de Integración Curricular es de mi autoría y los resultados del mismo son auténticos. Los textos en el documento que provienen de otras fuentes están debidamente citados y referenciados.

Como autor asumo la responsabilidad legal y académica de los contenidos de este Trabajo de Integración Curricular; el patrimonio intelectual pertenece a la Escuela Superior Politécnica de Chimborazo.

Riobamba, 30 de noviembre de 2023

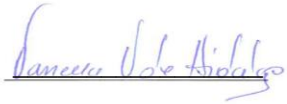
Diego Hernan Barahona Defaz
050342935-9

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE MECÁNICA

CARRERA MANTENIMIENTO INDUSTRIAL

El Tribunal del Trabajo de Integración Curricular certifica que: El Trabajo de Integración Curricular; Tipo: Proyecto de Investigación, "**CLASIFICACIÓN DE FALLAS EN RODAMIENTOS UTILIZANDO DATOS DE ANÁLISIS DE ULTRASONIDO BASADO EN UNA MÁQUINA DE APRENDIZAJE EXTREMO"**, realizado por el señor: **DIEGO HERNAN BARAHONA DEFAZ**, ha sido minuciosamente revisado por los Miembros del Tribunal del Trabajo de Integración Curricular, el mismo que cumple con los requisitos científicos, técnicos, legales, en tal virtud el Tribunal Autoriza su presentación.

	FIRMA	FECHA
Ing. Marco Antonio Ordoñez Viñan **PRESIDENTE DEL TRIBUNAL**		2023-11-30
Ing. Félix Antonio García Mora **DIRECTOR DE TRABAJO DE INTEGRACIÓN CURRICULAR**		2023-11-30
Ing. Vanessa Lorena Valverde González **ASESOR DE TRABAJO DE INTEGRACIÓN CURRICULAR**		2023-11-30

DEDICATORIA

El esfuerzo de este trabajo se lo dedico sin duda alguna a mis padres: Alfredo Barahona y Beatriz Defaz, quienes nunca dejaron de creer en mi a pesar los tropiezos que he atravesado en la vida. Además, dedico a la persona que se convirtió en mi complemento para compartir el resto de mi vida, Amelia.

Diego

AGRADECIMIENTO

Agradezco a la Escuela Superior Politécnica de Chimborazo, una institución de gran prestigio la cual me abrió las puertas para poder estudiar en la Carrera de Mantenimiento Industrial, a sus docentes por ser los que forman directamente a los futuros profesionales, especialmente al Ing. Félix García e Ing. Vanessa Valverde quienes son los que están detrás de este trabajo como director y asesor respectivamente, finalmente agradecer a toda mi familia que siempre están pendientes de mí.

Diego

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIÓNES

ÍNDICE DE ANEXOS

RESUMEN

El principal objetivo de este trabajo investigativo fue clasificar fallas de rodamientos a partir de una base de datos con parámetros de ultrasonido aplicando el algoritmo máquina de aprendizaje extremo (ELM), para esto primero se adquirió la base de datos obtenida de un trabajo de integración curricular que se encuentra disponible en el repositorio de la Escuela Superior Politécnica de Chimborazo. Seguidamente se realizó un análisis exploratorio de la base de datos filtrando los parámetros referidos a ultrasonido donde se evidenció la existencia de datos faltantes en cada estado de falla del rodamiento por lo que se hizo un pretratamiento incrementando la base de datos en cada estado de falla mediante la técnica del sobre muestreo. Antes de aplicar ELM se hizo una partición de los datos totales en datos de entrenamiento y datos de prueba para posteriormente extraer un conjunto de características que va presentando al analizar los datos y así poder ingresar al algoritmo a entrenarse; seguido se creó la clase ELM ajustando los híper parámetros principales como el número de neuronas en la capa oculta, función de activación entre otros que son necesarios para su entrenamiento. Una vez que se entrenó el modelo con este conjunto de datos se probó ingresando los datos de prueba y se evaluó los resultados permitiendo clasificar con la máxima efectividad obtenida los rodamientos que se encuentre en los diferentes estados de falla. La precisión en el conjunto de entrenamiento resulto del 94.08% y la precisión en el conjunto de prueba dio como resultado 90,43%. Con esto se puede concluir que es que se alcanzó el límite del rendimiento del modelo con los parámetros y datos proporcionados en este modelo de aprendizaje lo que se recomendaría contar con mayor cantidad de parámetros iniciales para un mejor rendimiento del modelo ELM aplicado.

Palabras clave: <MÁQUINA DE APRENDIZAJE EXTREMO>, <MANTENIMIENTO PREDICTIVO>, <RODAMIENTOS>, <ULTRASONIDO>, <APRENDIZAJE DE MÁQUINAS>.

SUMMARY

The main objective of this research work was to classify bearing failures from a database with ultrasound parameters by applying the extreme learning machine (ELM) algorithm, for this purpose, first the database obtained from a curricular integration work available in the repository of Escuela Superior Politécnica de Chimborazo was acquired. Next, an exploratory analysis of the database was carried out by filtering the parameters referred to ultrasound where it was evidenced the existence of missing data in each state of bearing failure, so a pre-treatment was made by increasing the database in each state of failure by means of the oversampling technique. Before applying ELM, a partition of the total data into training data and test data was made to subsequently extract a set of characteristics that are presented when analyzing the data and thus be able to enter the algorithm to be trained; then the ELM class was created by adjusting the main hyper parameters such as the number of neurons in the hidden layer, activation function among others that are necessary for training. Once the model was trained with this data set, it was tested by entering the test data and the results were evaluated allowing to classify with the maximum effectiveness obtained the bearings found in the different failure states. The accuracy in the training set resulted in 94.08% and the accuracy in the test set resulted in 90.43%. With this it can be concluded that is that the limit of the model performance was reached with the parameters and data provided in this learning model what would be recommended to have more initial parameters for better performance of the applied ELM model.

Keywords: <EXTREME LEARNING MACHINE (ELM) >, <PREDICTIVE MAINTENANCE>, <BEARINGS>, <ULTRASOUND>, <MACHINE LEARNING>.

Lic. Sandra Paulina Porras Pumalema MSc.
C.I. 0603357062

INTRODUCCIÓN

Los rodamientos son utilizados para una amplia variedad de aplicaciones industriales por su principal función de reducir la fricción y el desgaste entre pieza móviles. Las fallas en los rodamientos pueden causar consecuencias en otros componentes de la maquinaria, como engranajes, ejes y motores, lo que puede aumentar el tiempo de inactividad de la máquina y los costos de reparación. El ultrasonido es una técnica del mantenimiento muy útil para diagnosticar problemas en los rodamientos en una etapa temprana además de recopilar información ya que permite medir parámetros como frecuencia e intensidad del sonido y mediante un análisis se pueden indicar problemas en el rodamiento.

El mantenimiento en la industria 4.0 se basa en el uso de tecnologías avanzadas como el Internet de las cosas (IoT), la inteligencia artificial (IA) y el aprendizaje automático (ML) para mejorar la efectividad del mantenimiento ya que actualmente se genera más información a partir de las máquinas, esto permite a las empresas recopilar datos de sus equipos y utilizarlos para predecir y prevenir problemas antes de que ocurran. Máquina de aprendizaje extremo (ELM) es un algoritmo de aprendizaje automático que pertenece a la familia de las redes neuronales artificiales y más específicamente a las redes neuronales (*feedforward*) que se basa en una estructura de una sola capa así como un proceso de entrenamiento simple, se utiliza para resolver problemas de clasificación y regresión, además, ha demostrado ser muy eficiente en términos de tiempo de entrenamiento y precisión en comparación con otros algoritmos de redes neuronales tradicionales. ELM es utilizado en una variedad de aplicaciones, como el reconocimiento de patrones, el procesamiento de señales, el control automático, el diagnóstico de fallos, entre otros.

En este trabajo de integración curricular se propone un método más para la clasificación de fallas en rodamientos utilizando el algoritmo máquina de aprendizaje extremo ELM que se basa en entrenar el modelo con un conjunto de datos de ultrasonido que contengan información sobre 5 distintos modos de falla en rodamientos. La base de datos a utilizar pertenece a un trabajo de integración curricular de la carrera de Mantenimiento Industrial desarrollado en la ESPOCH y a partir de esta información se realizará todo el desarrollo de la metodología. Una vez entrenado, el modelo es capaz de clasificar nuevos datos de ultrasonido en función de su similitud con los datos de entrenamiento, y determinar el estado correspondiente del rodamiento. Al finalizar se evalúa los resultados sobre la efectividad del método mediante las métricas como son: la matriz de confusión, f1-score, precisión y recall.

CAPÍTULO I

1. PROBLEMA DE INVESTIGACIÓN

1.1. Planteamiento del problema

Uno de los componentes más importantes de las máquinas rotativas son los rodamientos siendo en donde está enfocado este estudio. Su mal funcionamiento puede ser muy determinante al momento de diagnosticar una máquina para encontrar la raíz de la falla ya que muchas veces se pasa por alto pensar que el rodamiento puede llegar a causar fallas severas a una máquina. Por lo general se ha utilizado métodos basado en experiencia, inspecciones regulares o programas de mantenimiento preventivo para diagnosticar el estado del rodamiento sin embargo estos métodos son cada vez más tradicionales y pueden no ser suficientes ya que pueden generar mantenimiento innecesario o, en algunos casos, no detectar problemas potenciales antes de que se conviertan en fallas graves. Existen también técnicas como el análisis de vibración, termografía o ultrasonido que ayudan a detectar anormalidades de forma más exacta recopilando información que es examinada por un especialista dando muy buenos resultados, pero cuando es una gran cantidad de datos hace complicado ser analizado por métodos manuales donde su efectividad puede bajar, esto hace necesario emplear una técnica de diagnóstico basado en datos que permita clasificar fallas de rodamientos más temprano y de mejor manera.

1.2. Limitaciones y delimitaciones

Este estudio se sumerge en el campo de la inteligencia artificial, específicamente en el aprendizaje de máquinas, un ámbito que ha experimentado avances significativos en los últimos tiempos. La complejidad y la rapidez con la que evoluciona la inteligencia artificial representan un desafío, ya que este tema puede resultar menos familiar en el contexto del mantenimiento industrial. No obstante, esta circunstancia se percibe como una oportunidad valiosa para adquirir conocimientos en un área en constante desarrollo. El tiempo de investigación necesario para comprender y aplicar adecuadamente el algoritmo de máquina de aprendizaje extremo (ELM) puede considerarse como una limitación inicial. Sin embargo, con la disponibilidad de recursos y acceso libre a la información, se ve como una oportunidad para explorar y profundizar en el ELM, una herramienta prometedora en el campo de la clasificación de fallas en rodamientos utilizando datos de ultrasonido.

1.3. Problema General de Investigación

¿Cómo puede lograrse una clasificación automatizada y rápida de fallas en rodamientos a través del análisis eficiente de datos de ultrasonido?

1.4. Problemas específicos de investigación

¿Qué se debe hacer antes de empezar a trabajar con una base de datos obtenida?

¿De qué forma se puede preparar la información antes de entrenar la base de datos?

¿Qué se busca al realizar pruebas de entrenamiento antes de establecer un modelo final?

¿Qué se debe hacer una vez que se tiene seleccionado los parámetros para evaluar los resultados?

1.5. Objetivos

1.5.1. Objetivo general

Clasificar las fallas en rodamientos utilizando datos de análisis de ultrasonido basado en una máquina de aprendizaje extremo.

1.5.2. Objetivos específicos

Preprocesar la base de datos de ultrasonidos en rodamientos obtenida.

Dividir la base de datos de rodamientos en datos de entrenamiento y datos de prueba.

Encontrar las características de extracción que mejor correlacionan a los datos de falla.

Aplicar el algoritmo de clasificación de aprendizaje extremo a los datos de falla de ultrasonido para rodamientos.

Verificar la efectividad del modelo de clasificación de aprendizaje extremo para detectar fallas utilizando datos de análisis de ultrasonido.

1.6. Justificación

Las máquinas con partes móviles vibran al momento de su funcionamiento, y estas vibraciones a veces se pueden percibir directamente de diferente manera, por ejemplo; cuando se coloca una mano sobre la máquina o cuando el sonido emitido por la máquina es audible al oído humano (Besa Antonio, Carballeira Javier, 2018, p. 87). Uno de los componentes más importantes en estas máquinas son los rodamientos, encargados de reducir la fricción principalmente de los ejes. Se estima que un 20% de las fallas de las máquinas son causadas por rodamientos. Comúnmente los rodamientos se reemplazan cuando han cumplido ciertas horas de funcionamiento o también a veces se ha cambiado de manera imprevista por alguna falla severa.

Estas actividades de mantenimiento van siendo menos eficientes, por tanto, en caso de falla es necesario poder estimar su ocurrencia por medio del mantenimiento predictivo que es un conjunto de técnicas que se aplican con el fin de estimar las fallas. (Gallará I, Pontelli D, 2020, p. 77).Las herramientas tecnológicas han jugado un papel muy importante en el diagnostico en el mantenimiento siendo el aprendizaje de máquinas una de estas.

Machine Learning o también aprendizaje de máquina, es un concepto nuevo que consiste en descubrir posibles relaciones entre nuestros datos mediante algoritmos que se han desarrollado hace pocos años y que implementan grandes empresas como facebook y google. Hay muy poca información al respecto (Nolasco Jorge, 2018, p. 256). Se centra en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender a partir de datos.

Varios de estos métodos para el diagnóstico se han aplicado generalmente en máquinas rotativas comúnmente en las CNC que tiene elementos vitales para su funcionamiento siendo los rodamientos uno de ellos. La detección de fallas en rodamientos especialmente en una etapa temprana genera ahorros tanto en recursos económicos como humanos por tal razón es necesario poder contar con un sistema inteligente de detección de fallas que sea automático, económico, eficaz y fácil de usar (Chuya Jorge, 2019, p. 2).

1.7. Hipótesis

La aplicación del método de aprendizaje extrema a una base de datos de ultrasonidos en rodamientos permite clasificar las fallas.

1.7.1. Variable dependiente

Clasificación de fallas.

1.7.2. Variable independiente

- F1-score
- Recall
- Precisión
- Matriz de confusión

CAPÍTULO II

2. MARCO TEÓRICO

2.1. Antecedentes de investigación

El mantenimiento nace principalmente de la necesidad de conservar los equipos o máquinas en buenas condiciones de funcionamiento, pero esto no impide que con el tiempo se produzcan degradaciones o deterioro de las máquinas causando que presenten síntomas de fallo.

Dentro de las máquinas rotativas hay componentes importantes principalmente los rodamientos, que puede fallar fácilmente. Para diagnosticar los síntomas existe diversidad de técnicas que en por su parte (García Félix, 2021.) mediante el análisis de señales de vibraciones y utilizando técnicas de aprendizaje de máquinas ha obtenido un modelo que por medio del reconocimiento de las particularidades de datos puede clasificar cuando un rodamiento está en falla.

Con el crecimiento de la industria 4.0 las máquinas tienen a generar cada vez más datos que se pueden analizar mediante diferentes técnicas de aprendizaje de máquinas para tomar decisiones de mantenimiento más confiables; debido a la gran cantidad de datos generados se convierte en un desafío humano poder analizarlo de manera que, se ha desarrollado un modelo de aprendizaje de máquinas para la detección de fallas (Vilema Pablo, 2022).

Un gran número de actividades modernas va generando información que puede ser aprovechado y mejorado en algún punto siendo muy importante para la toma de decisiones. Partiendo de esa referencia los modelos de aprendizaje de máquinas han cobrado mucha importancia en estos últimos tiempos dentro del campo de la inteligencia artificial, entre ellas destaca las máquinas de aprendizaje extremo (Castro Fausto, 2019).

Se cree que el análisis de ultrasonido es el método más confiable para la detección de fallas en los rodamientos ya que la presencia de síntomas aparece antes que varie la frecuencia o la temperatura. Además, las fallas causadas por la fatiga, el desgaste de la superficie y los problemas de lubricación se pueden detectar con esta técnica siendo una de las más eficientes (Graciá Luis, 2021, p. 22).

2.2. Referencias Teóricas

2.2.3. Rodamientos

Los rodamientos son elementos muy importantes para las máquinas rotativas debido a que sirven como soportes entre ejes para reducir la fricción que se produce cuando giran. Es importante revisar de forma general las partes del rodamiento como podemos observar en la Tabla 1-2; en este caso se va a tomar de ejemplo un rodamiento rígido de bolas.

Tabla 2-1: Descripción de las partes del rodamiento

N°	COMPONENTE	DESCRIPCIÓN
1 y 2	Pista externa e interna	La pista de rodamientos permite colocar el eje de la máquina sin problemas de fricción
3	Elementos rodantes	Más conocidos como bolas y permiten que la pista exterior e interior giren independientemente entre ellas y así reduzcan la fricción.
4	Jaula	Su función es mantener separadas a las bolas para evitar que se unan y produzcan un desgaste anormal.
5	Obturación	Está ubicado en la parte lateral del rodamiento actúa como protector ya que no permite que entre nada que pueda provocar una falla.

Fuente: (Graciá Luis, 2021)

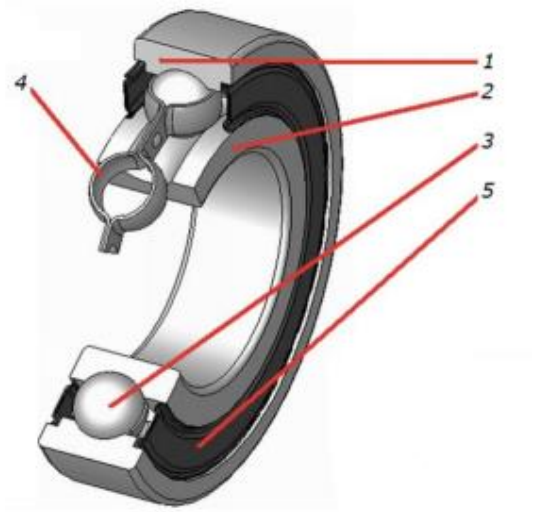


Ilustración 2-1: Rodamiento rígido de bolas.

Fuente:(Graciá Luis, 2021)

2.2.4. Importancia de los rodamientos

Los rodamientos son componentes mecánicos que se utilizan para soportar cargas y reducir la fricción entre dos piezas en movimiento. Los rodamientos son esenciales en una amplia variedad de aplicaciones industriales y tienen varias funciones importantes. La importancia de los rodamientos se da por estas razones:

- **Permiten el movimiento suave:** Los rodamientos permiten que las piezas en movimiento se desplacen de manera suave y reduciendo la fricción al máximo, lo que disminuye el desgaste y aumenta la eficiencia.

- **Soportan cargas:** Los rodamientos pueden soportar cargas significativas y distribuirlas de manera uniforme, lo que los hace ideales para aplicaciones de alta carga.

- **Reducen el ruido:** Los rodamientos de alta calidad pueden reducir el ruido y las vibraciones, lo que los hace ideales para aplicaciones en las que se requiere un funcionamiento silencioso.

- **Aumenta la vida útil de los equipos:** Los rodamientos de alta calidad pueden prolongar la vida útil de los equipos en los que se utilizan, lo que reduce los costos de mantenimiento y mejora la productividad.

2.2.5. Técnicas utilizadas para el diagnóstico de fallas en rodamientos

La detección de fallas es muy importante y permitirá identificar mediante parámetros y condiciones que se encuentra el rodamiento. Hoy en día, hay distintas técnicas de mantenimiento predictivo que se aplican para analizar el estado de los rodamientos. Entre ellas destacan:

- Análisis con ultrasonidos.
- Análisis de vibraciones.
- Análisis por firma eléctrica.
- Análisis de temperaturas.

Las técnicas que más se utiliza son el análisis por ultrasonido y el análisis por vibración. El análisis con ultrasonido se considera un método mucho más efectivo para detectar las fallas en rodamientos, aunque su costo es mayor a comparación con el análisis por vibración. Con el ultrasonido se puede detectar de mejor manera algunos síntomas que puede presentar el rodamiento como desgastes en superficies, problemas con el lubricante y fallos a fatiga (Graciá Luis, 2021, p. 22).

2.2.6. Análisis por ultrasonido

2.2.6.1. El ultrasonido

Es una vibración mecánica con un alcance superior al alcance audible para el oído humano, transmitida a través de un medio físico que puede ser registrada y medida en Hertz (Hz) por equipos creados para tal fin debido a que el oído humano solamente puede percibir sonidos que su frecuencia se encuentre entre 20 Hz y 20kHz. (Sánchez Ana, 2017, p. 46).

El análisis con ultrasonidos es una técnica no destructiva eficiente para la detección temprana de problemas en los rodamientos, lo que permite tomar medidas para evitar fallas mayores y costosas. También es útil para evaluar la condición de los rodamientos durante el mantenimiento preventivo y puede ayudar a planificar el reemplazo de los rodamientos antes de que ocurra una falla.

2.2.6.2. Ultrasonido Activo y Pasivo.

El diagnóstico de rodamientos con ultrasonido pasivo es una técnica que se utiliza para detectar problemas en los rodamientos mediante el análisis de las ondas de sonido emitidas por el rodamiento. A diferencia del diagnóstico de rodamientos con ultrasonido activo, en el cual se utiliza un transductor para generar ondas de sonido, en el diagnóstico pasivo se utiliza un micrófono para capturar las ondas de sonido emitidas por el rodamiento.

2.2.7. Límites de audibilidad

2.2.7.1. Frecuencia de ultrasonido (Hz)

Los límites de la audibilidad son los rangos más altos y bajos de frecuencia que pueden ser percibidos por el oído humano. Estos límites pueden cambiar con la edad.

Tabla 2-2: Límites de audibilidad

Rango	Límites
Infrasonido	Tonos cuyas frecuencias estén por debajo de los 16 Hz.
Sonido audible	Todos los tonos cuya frecuencia estén dentro de 16Hz hasta los 20 kHz.
Ultrasonido	Los tonos con frecuencias por encima de los 20 kHz.

Fuente:(Sánchez Ana, 2017)

En la Tabla 2-2 se describen los límites de audibilidad en donde el límite inferior es aproximadamente de 20 Hz, lo que significa que las frecuencias por debajo de 20 Hz no son percibidas por el oído humano. El rango audible va de los 20 Hz hasta los 20 kHz, donde todo sonido dentro de este rango se puede percibir por el oído humano. Por último, el límite superior de la audibilidad es aproximadamente de 20.000 Hz, lo que significa que las frecuencias por encima de 20.000 Hz también son imperceptibles para el oído humano.

De igual manera se puede distinguir entre los distintos rangos en la Ilustración 2-2.

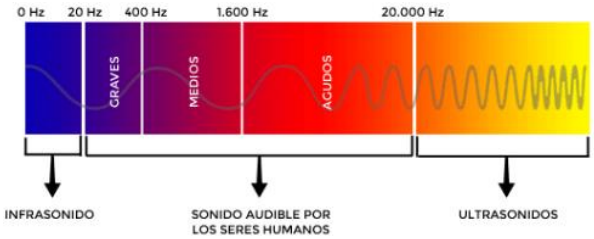


Ilustración 2-2: Rangos de frecuencias del sonido.

Fuente: (programar fácil, 2022)

2.2.7.2. Intensidad del sonido (dB)

La unidad para medir la intensidad del sonido se llama decibelio. Los decibelios se utilizan para expresar la diferencia de intensidad entre dos sonidos y se basan en la escala logarítmica de intensidad de sonido. Esto significa que un aumento de 10 dB en la intensidad del sonido representa un aumento de 10 veces en la intensidad del sonido.

Uno de los factores para la degradación auditiva es una excesiva exposición a volúmenes altos de presión sonora que van más allá del límite que el oído humano puede soportar, de lo contrario se empieza a experimentar dolor además del riesgo de perder el sentido del oído. El límite de intensidad de sonido para el oído se encuentra sobre los 130 dB en donde se presentan

molestias si se está expuesto constantemente, en la Tabla 3-2 se describen algunos ejemplos cuando va incrementando la intensidad del sonido. (Zafra Julián, 2018, p. 62).

Tabla 2-3: Descripción de los niveles de sonido soportable al oído.

DESCRIPCIÓN	INTENSIDAD DEL SONIDO (dB)
Detonaciones	140
Umbral de dolor	130
	120
Tren subterráneo	110
Aviones	100
Ruido promedio en fábricas	90
	80
Conversación promedio	70
	60
Oficina promedio	50
Ambiente residencial	40
Susurro silencioso	30
	20
	10
Umbral de audición	0

Fuente:(Zafra Julián, 2018)

2.2.8. Deterioro de los rodamientos

Anteriormente se describió que los rodamientos están conformados por diferentes elementos como son: pista exterior, pista interior, jaula o canastilla y elementos rodantes. El desgaste que sufre las diferentes partes del rodamiento producirá una o algunas frecuencias características que pueden ser identificados y analizados. Se tiene cuatro posibles frecuencias en el de deterioro en el rodamiento que son:

- **BPFO** (Frecuencia de deterioro de la pista exterior), es la frecuencia que se desplazan los elementos rodantes con la presencia de un desperfecto en la pista externa cada vez que se cumple un giro completo del eje (Guangaxi Pilar, 2022).

- **BPFI** (Frecuencia de deterioro de la pista inferior), es la frecuencia con que se desplaza los elementos rodantes con la presencia de un desperfecto en la pista interna cada vez que se cumple un giro completo del eje (Guangaxi Pilar, 2022).

- **BSF** (Frecuencia de deterioro de los elementos rodantes), es la frecuencia con que se desplazan los elementos rodantes o más bien el número de vueltas que hace una bola cada vez que el eje realiza un giro completo.(Guangaxi Pilar, 2022).

- **FTF** (Frecuencia de deterioro de la canastilla), es la frecuencia de giro de la canastilla donde se encuentran los elementos rodantes cada vez que el eje realice un giro completo (Guangaxi Pilar, 2022).

2.2.9. Software

2.2.9.1. Jupyter Notebook

Teniendo en cuenta que hoy en día la información generada por cualquier máquina u objeto se lo registra de manera digital, hace que en análisis de los datos y la interpretación de estos sea cada vez más dificultoso de interpretar y explicar.

Ilustración 2-3: Logo de aplicación Jupyter.

Fuente: (Jupyter, 2022)

Jupyter Notebook es una opción gratuita de herramienta informática creada para ser un cuaderno de matemática en donde se puede desarrollar diferentes tipos de aplicaciones mediante código además de poder compartir los cálculos relacionados, código y documentos de texto. Es una de las aplicaciones de Anaconda *Navigator* de código abierto, actúa como un cuaderno de

laboratorio virtual que admite, datos, código, flujos de trabajo y visualizaciones que detallan el proceso del desarrollo. Es legible por humanos y máquina (Randles et al., 2017).

2.2.10. Procesadores CPU, GPU y TPU

Dentro del aprendizaje de máquinas se utilizan estos procesadores que son parte del hardware de un computador para analizar y entender grandes cantidades de datos y luego utilizar los resultados de ese análisis para realizar predicciones o tomar decisiones. Sin embargo, cada tipo de procesador tiene sus propias fortalezas y debilidades en términos de su capacidad para realizar este tipo de cálculos.

El componente más importante del hardware de un computador tiene el nombre de unidad central de procesamiento más conocida como CPU, que es el principal encargado para poder interpretar instrucciones y para ejecutar procesos. La CPU dentro del área del aprendizaje de máquinas ha sido la única herramienta computacional para en entrenamiento y la inferencia de modelos, pero la razón principal del progreso en el aprendizaje automático se da con el uso de nuevos elementos de hardware para aumentar la velocidad computacional del entrenamiento incluido el uso de unidades de procesamiento de tensores (TPU) y unidades de procesamiento de gráficos (GPU) (Gaviña Javier, 2022).

2.2.10.1. CPU

Conocida por sus siglas en inglés (*Central Processing Unit)* es el componente fundamental de un computador donde es el encargado de realizar la mayoría de las operaciones aritméticas y lógicas, además de controlar el flujo de datos. (Gaviña Javier, 2022)

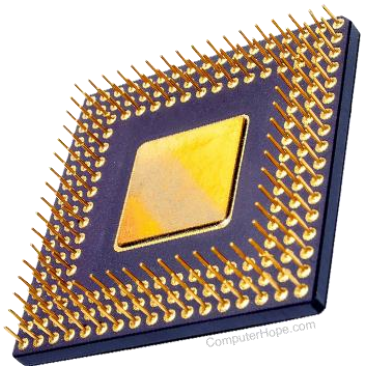


Ilustración 2-4: Unidad Central de Procesamiento (CPU)

Fuente: (computerhope, 2022**)**

Son versátiles y pueden realizar muchas tareas, pero no son tan rápidos cono los GPU o los TPU. Por tal razón, los CPU pueden ser adecuados para aprendizaje de máquinas que no requieran que su rendimiento para cálculos sea el máximo.

2.2.10.2. GPU

Un GPU (*Graphics Processing Unit)* o unidad de procesamiento gráfico. Es un tipo de procesador diseñado para acelerar la generación de imágenes en una computadora. Los GPU tienen una gran cantidad de núcleos de procesamiento que pueden ejecutar operaciones matemáticas en paralelo, lo que los hace ideales para acelerar el procesamiento de gráficos y cálculos matemáticos complejos. Esto los hace muy útiles para acelerar el entrenamiento de modelos de aprendizaje profundo (*deep learning*), que requiere el cálculo de muchas operaciones matemáticas en paralelo.

La GPU incrementa enormemente las operaciones gracias a la elevada cantidad de núcleos de procesamiento que contienen. Es por ello por lo que la GPU logra maximizar el througput (número de operaciones por segundo), lo que la convierte adecuada para operaciones matemáticas en paralelo, como las redes neuronales profundas. Sin embargo, en el cálculo de algoritmos de Machine Learning simples, puede llegar a ser óptimo el uso de la CPU (Gaviña Javier, 2022).

Ilustración 2-5: Unidad de Prosador Gráfico.

Fuente: (computerhope, 2022)

2.2.10.3. TPU

Una TPU (*Tensor processing Unit) o unidad de procesamiento tensorial.* Es un tipo de procesador diseñado específicamente para acelerar el procesamiento de operaciones matemáticas utilizadas en el aprendizaje profundo (*depp learning).* Los TPU están optimizados para ejecutar operaciones matemáticas de precisión de alta velocidad y son muy eficientes en

términos de consumo de energía. Esto los hace ideales para acelerar el entrenamiento de modelos de aprendizaje profundo.

Es posible utilizar TPU en la nube a través de Google Colab, una plataforma gratuita de Jupyter Notebook que proporciona recursos informáticos en la nube para ejecutar y compartir código de Python. Una de las características es que permite utilizar TPU de forma gratuita y sin configuración adicional.

Decimos que una TPU es un circuito con gran número de núcleos, muchos más que los ofrecidos por una GPU, especializados en realizar operaciones compuestas sobre tipos de datos simples. Esto les permite incrementar el rendimiento respecto a las GPU y las CPU (Gaviña Javier, 2022).

Ilustración 2-6: Unidad de Procesamiento Tensorial.
Fuente: (computerhope, 2022)

2.2.11. Extracción de características

Es un proceso fundamental en el análisis de datos. Consiste en tomar los datos en su forma original y transformarlos en un conjunto de características más informativas que capturan las propiedades relevantes y distintivas de los datos. Esto puede implicar crear nuevas variables derivadas, combinar características existentes y ajustar los datos para que sean más adecuados para el análisis.

2.2.11.1. Extractor de Características TSFEL

El Extractor de Características TSFEL (*Time Series Feature Extraction Library*) es una herramienta poderosa utilizada en análisis de señales y aprendizaje automático para extraer información relevante y distintiva de conjuntos de datos de series temporales. TSFEL ofrece

una amplia variedad de más de 60 características que se calculan de forma automática en tres dominios diferentes: temporal, estadístico y espectral.

El Extractor de Características TSFEL automatiza el cálculo de estas características, lo que ahorra tiempo y esfuerzo en la extracción de características (Barandas Marília et al., 2020).

- **Dominio Temporal:** En este dominio, TSFEL calcula características directamente en el dominio del tiempo de la señal. Esto incluye características como la media, la mediana, la desviación estándar, la asimetría y la curtosis. Estas características capturan información básica sobre la distribución y las tendencias en la serie temporal.

Tabla 2-4: Características en el dominio Temporal.

Características	Costo Computacional
Energia absoluta	1
Área bajo la curva	1
Autocorrelación	1
Centroide	1
Entropía	1
Diferencia media absoluta	1
Diferencia media	1
Diferencia absoluta mediana	1
Diferencia mediana	1
Puntos de Inflexion negativos	1
Distancia pico a pico	1
Puntos de Inflexion positive	1
Distancia de la señal	1
Pendiente	1
Suma diferencia absoluta	1
Energía total	1
Tasa de cruce por cero	1
Picos de barrio	1

Fuente:(Barandas Marília et al., 2020)

Los valores numéricos en la columna "Costo Computacional" indican una estimación relativa de la cantidad de recursos necesarios para realizar el cálculo de cada característica en función de la complejidad de la operación matemática involucrada y el tamaño de los datos.

- **Dominio Estadístico:** En el dominio estadístico, TSFEL calcula características que proporcionan información sobre la distribución estadística de la señal. Esto incluye características como el valor del percentil, la entropía, entre otros. Estas características pueden revelar detalles sobre la variabilidad y la forma de la señal.

Tabla 2-5: Características en el dominio Estadístico.

Características	Costo Computacional
ECDF	1
Percentil ECDF	1
Recuento de percentiles ECDF	1
Histograma	1
Rango intercuartil	1
Curtosis	1
Máx	1
Significar	1
Desviación media absoluta	1
Mediana	1
Desviación absoluta mediana	1
Mínimo	1
Media cuadrática	1
Oblicuidad	1
Desviación estándar	1
Diferencia	1

Fuente:(Barandas Marília et al., 2020)

- **Dominio Espectral:** En este dominio, TSFEL analiza la señal en el espacio de frecuencia, calculando características como la energía espectral, la potencia en diferentes bandas de frecuencia y la entropía espectral. Estas características son útiles para comprender las propiedades frecuenciales y las componentes armónicas de la señal.

Tabla 2-6: Características en el dominio Espectral.

Características	Costo Computacional
Coeficiente medio FFT	1
Frecuencia fundamental	1
Energía de rango humano	2
LPCC	1
MFCC	1
Espectro de potencia máxima	1
Frecuencia máxima	1
Frecuencia media	1
Ancho de banda de potencia	1
Centroide espectral	2
Disminución espectral	1
Distancia espectral	1
Entropía espectral	1
Curtosis espectral	2
Puntos de inflexión positivos espectrales	1
Roll-off espectral	1
roll-on espectral	1
asimetría espectral	2
Pendiente espectral2	1

Propagación espectral	1
Variación espectral	1
Wavelet media absoluta	2
Energía de onda	2
Desviación estándar de wavelet	2
entropía wavelet	2
Varianza wavelet	2

Fuente:(Barandas Marília et al., 2020)

2.2.12. Aprendizaje de Maquinas

Es también conocido como aprendizaje automático y es la ciencia que permite que las computadoras aprendan en base a los datos. En lugar de ofrecer una solución paso a paso para cada necesidad específica como lo hacen los métodos de programación tradicionales, el campo del aprendizaje automático se enfoca en desarrollar algoritmos generales que pueden derivar patrones de diferentes tipos de datos. Existe diferentes opciones que podemos encontrar las cuales se clasifican en los siguientes tipos de aprendizaje de máquinas.(Bobadilla Jesús, 2020, p. 10)

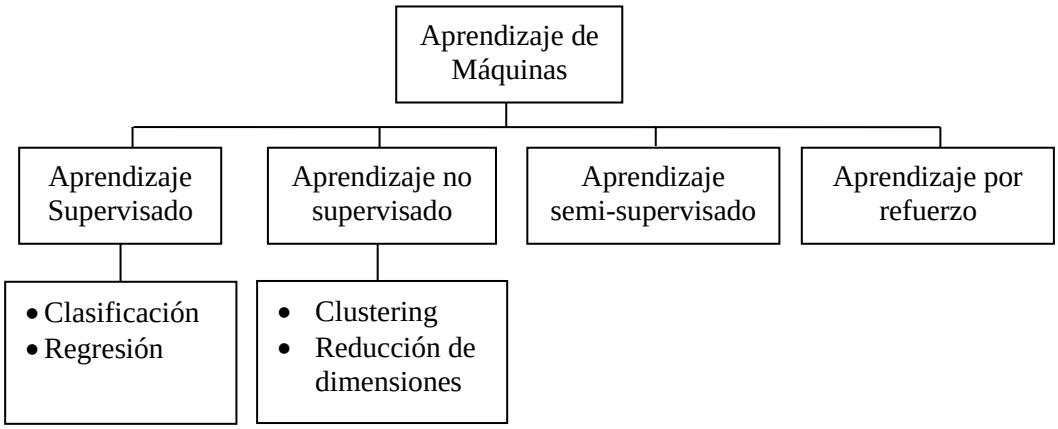


Ilustración 2-7: Clasificación del aprendizaje de máquinas

Realizado por: Barahona, Diego, 2023

2.2.13. Aprendizaje supervisado

Este tipo de aprendizaje consiste en presentar un conjunto de pruebas de los cuales conocemos la respuesta. Lo que deseamos es formular algún tipo de regla o correspondencia que nos permita aproximarnos a la respuesta. Partiendo de ese conjunto de pruebas se puede usar diferentes algoritmos de clasificación de aprendizaje de máquinas con el objetivo de "entrenar" un modelo y al finalizar dicho entrenamiento, predecir la etiqueta correspondiente incluso de datos no incluidos en el conjunto de datos originales.

2.2.14. Redes Neuronales

Las redes neuronales es una de las técnicas dentro de la inteligencia artificial al igual que el aprendizaje de máquinas y aprendizaje profundo en donde ha crecido de gran manera en la actualidad.

Una red neuronal artificial (RNA) es un sistema elaborado para poder interpretar estímulos como en las neuronas que existe en nuestro cerebro en donde miles de señales ingresan y estas deben ser procesadas para obtener una sola respuesta. Al existir millones de neuronas y que estén conectadas entre sí, hace posible que se formen las "redes neuronales". Este método se empezó con una simulación abstracta del sistema nervioso humano haciendo referencia mediate las neuronas interconectadas entre si teniendo un parecido a las conexiones de axones y dendritas en nuestro sistema nervioso. (Mercado Franklin, 2018, p. 14)

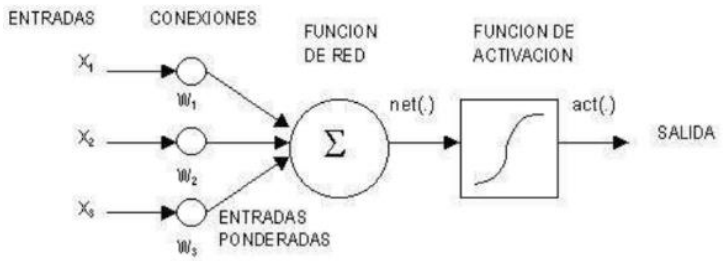


Ilustración 2-8: Representación de una red neuronal artificial.

Fuente: (Mercado Franklin, 2018)

Dentro de la Ilustración 8-2. se representa una red neuronal artificial siendo X el vector de entrada, seguido de W que es el peso asociado a cada entrada para luego se pueda realizar un producto entre las entradas y sus pesos siendo conocida como función de red.

La salida de la función de red se evalúa en la función de activación que produce la salida de la unidad de proceso. De esta manera se puede observar que el comportamiento será como una neurona biológica de manera sencilla. Una representación vectorial del funcionamiento básico de una neurona artificial se indica según la siguiente expresión:

$$O = f(X * W)$$

Donde la función f puede ser una función lineal, una función umbral o una función no lineal que simule con mayor exactitud las características de transferencia no lineales de las neuronas biológicas.(Mercado Franklin 2018, p. 15).

Por las entradas Xi llegan unos valores que pueden ser enteros, reales o binarios. Estos valores podrían ser señales que enviarían otras neuronas. Los pesos que hay en las sinapsis equivaldrían en la neurona biológica a los mecanismos que existen en las sinapsis para transmitir la señal.

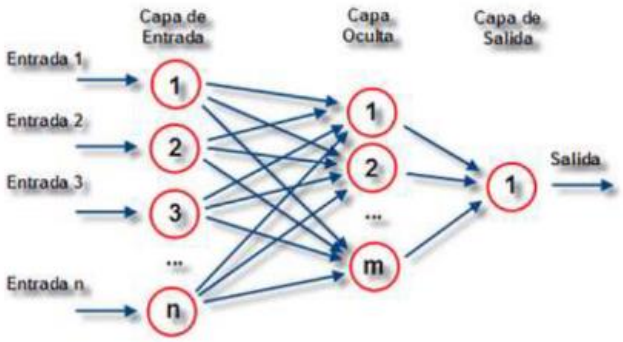


Ilustración 2-9: Representación del perceptrón multicapa.

Fuente: (Mercado Franklin, 2018)

La salida de la función de red llega a la de activación que transforma este valor en otro en el dominio que trabajen las salidas de las neuronas. El valor de la salida cumpliría la función de la tasa de disparo de las neuronas biológicas.

2.2.15. Máquina de Aprendizaje Extremo

Se caracteriza por ser un método de entrenamiento de forma rápida para las redes SLFN (*Single Layer Feed forward Networks)* o redes de alimentación directa de una sola capa.

El Aprendizaje de Maquina Extremo o (*Extreme Learning Machines)* ELM trabaja en redes monocapa hacia delante o de tipo "*feedforward*" SLFN de una sola capa oculta. Comparado con las técnicas de inteligencia computacional tradicionales, ELM mantiene la capacidad de generalización y un entrenamiento mucho más rápido con menor intervención humana. Porque los pesos de la capa oculta no necesitan ser calculados, es decir, dicha capa no necesita ser entrenada (Wang et al., 2022, p. 2).

Su idea principal es asignar aleatoriamente los pesos entre las entradas y la capa oculta, y luego calcular directamente los pesos de la capa de salida utilizando una técnica de regresión lineal. Esto permite que el proceso de entrenamiento sea extremadamente rápido en comparación con otros algoritmos de aprendizaje profundo más complejos. Sin embargo, es importante tener en cuenta que el rendimiento del ELM puede depender en gran medida de la elección adecuada del número de neuronas en la capa oculta y la función de activación utilizada. A diferencia de otros algoritmos de aprendizaje profundo, la Máquina de Aprendizaje Extremo tiene una etapa de

entrenamiento muy rápida y no requiere ajustar hiper parámetros complejos (Wang et al., 2022, p. 2).

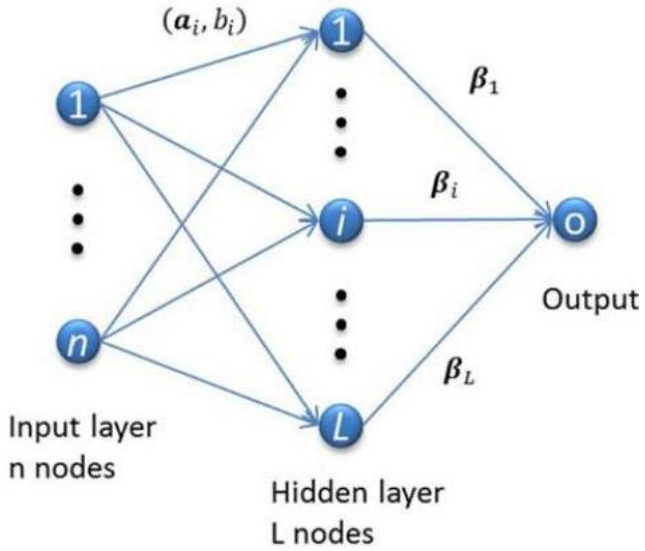


Ilustración 2-10: Estructura de ELM

Fuente: (Li Xudong, 2018)

El proceso de entrenamiento de ELM consta de los siguientes pasos:

a) **Entrada de datos e inicialización de pesos y sesgos:** Primero, se deben proporcionar los datos de entrenamiento. Cada ejemplo de entrenamiento debe estar representado por un vector de características (atributos) y su correspondiente etiqueta de salida. Luego se generan aleatoriamente los pesos y sesgos (bias) de las conexiones entre las capas de entrada y ocultas de la red neuronal. Esto se hace solo una vez y no se requiere ajustarlos durante el proceso de entrenamiento.

b) **Cálculo de la capa oculta:** Se calculan las salidas de la capa oculta utilizando las funciones de activación no lineales (por ejemplo, funciones sigmoides, funciones Relu, etc.). Esta es la etapa "extrema" del algoritmo ya que se puede realizar de manera rápida mediante el uso de álgebra lineal y la inversión de matrices específicamente se puede obtener por la inversa de Moore-Penrose ya que es un problema lineal.

c) **Cálculo de los pesos de salida y predicciones:** Una vez obtenidas las salidas de la capa oculta, se calculan los pesos de salida utilizando un enfoque de mínimos cuadrados. Con los pesos de salida calculados, la red ELM está lista para hacer predicciones en nuevos datos.

2.2.15.1. Inicialización de la clase ELM

El algoritmo ELM no se encuentra está disponible en ninguna distribución estándar de Python. Para utilizar el algoritmo ELM en Jupyter Notebook o en cualquier otro entorno se puede descargar el código fuente de la biblioteca ELM desde su repositorio en GitHub u optar por una biblioteca ya existente que implemente ELM. Se instalar sus dependencias utilizando herramientas como “pip” (el gestor de paquetes de Python).

Después se puede importar en Jupyter Notebook mediante la declaración de importación estándar de Python “import elm”. Finalmente se puede utilizarla para entrenar y aplicar redes neuronales de avance con una sola capa oculta utilizando el algoritmo ELM (Li Xudong, 2018).

En la Ilustración 11-2. se visualiza como se inicializa la clase ELM con los siguientes parámetros:

- **hidden_units:** Número de neuronas en la capa oculta.
- **activation_function:** Función de activación utilizada en la capa oculta.
- **x:** Datos de entrenamiento de forma (muestras, características).
- **y:** Etiquetas de entrenamiento de forma (muestras,).
- **C:** Parámetro de regularización.
- **elm_type:** Tipo de problema, puede ser 'clf' para clasificación o 'reg' para regresión.
- **one_hot:** Un valor booleano que indica si las etiquetas deben transformarse a one-hot encoding (solo en problemas de clasificación).
- **random_type:** Método de inicialización de los pesos, puede ser 'uniform' o 'normal'

```
# Crear el modelo ELM para esta iteración
model = elm.elm(hidden_units=1700, activation_function='relu',
                random_type='normal', x=x_train_scaled, y=y_train_encoded,
                C=0.1, elm_type='clf', one_hot=True)

# Entrenar el modelo en esta iteración
beta, train_accuracy, running_time = model.fit('no_re')
```

Ilustración 2-11: Inicialización de ELM con los parámetros principales.

Fuente: (Li Xudong, 2018)-

2.2.15.2. Funciones de activación y método “input2hidden”

El código fuente implementa varias funciones de activación en la capa oculta, incluyendo 'sigmoid', 'relu', 'sin', 'tanh' y 'leaky_relu'. La función “input2hidden” toma las características de

entrada “x” y calcula la salida de la capa oculta “H” aplicando la función de activación correspondiente a las neuronas.

```
if self.activation_function == 'relu':
    self.H = self.temH * (self.temH > 0)
```

Ilustración 2-12: Configuración de la función de activación RELU.

Fuente: (Li Xudong, 2018).

2.2.15.3. Método “hidden2output”

El método “hidden2output” toma la salida de la capa oculta **H** y calcula la salida final multiplicando **H** por la matriz de pesos beta.

```
# compute the output
def __hidden2output(self, H):
    self.output = np.dot(H.T, self.beta)
    return self.output
```

Ilustración 2-13: Configuración para la capa de salida.

Fuente: (Li Xudong, 2018)-

2.2.15.4. Método fit:

El método “fit” entrena el modelo ELM y calcula la matriz de pesos beta. Se puede utilizar diferentes algoritmos para calcular beta, los cuales son:

- **no_re:** Sin regularización.
- **solution1:** Algoritmo más rápido 1.
- **solution2:** Algoritmo más rápido 2.

```
# no regularization
if algorithm == 'no_re':
    self.beta = np.dot(pinv2(self.H.T), self.y_temp)
```

Ilustración 2-14: Configuración del método “fit”.

Fuente: (Li Xudong, 2018).

Dependiendo del tipo de problema (clf o reg), se realiza el procesamiento necesario para las etiquetas. Luego, se calcula la matriz H para los datos de entrenamiento y se utiliza el algoritmo seleccionado para calcular beta.

La función también calcula la salida de entrenamiento “result”, y si el problema es de clasificación, aplica la función Softmax para obtener probabilidades de clases. Finalmente, se evalúa el rendimiento del entrenamiento, ya sea mediante precisión clasificación o RMSE regresión (Li Xudong, 2018)-.

2.2.15.5. Método predict:

El método “predict” toma datos de entrada “x”, calcula la salida de la capa oculta H y luego calcula las predicciones finales “y_”, utilizando la matriz de pesos beta.

```
def predict(self, x):
    self.H = self.__input2hidden(x)
    self.y_ = self.__hidden2output(self.H)
    if self.elm_type == 'clf':
        self.y_ = np.where(self.y_ == np.max(self.y_, axis=1).reshape(-1, 1))[1]

    return self.y_
```

Ilustración 2-15: Configuración del método “predict”.

Fuente: (Li Xudong, 2018)-.

2.2.15.6. Método score:

El método score toma datos de entrada “**x”** y etiquetas “**y”**, y calcula el rendimiento del modelo utilizando las predicciones realizadas por el método “predict”.

```
def score(self, x, y):
    self.prediction = self.predict(x)
    if self.elm_type == 'clf':
        self.correct = 0
        for i in range(y.shape[0]):
            if self.prediction[i] == y[i]:
                self.correct += 1
        self.test_score = self.correct/y.shape[0]
```

Ilustración 2-16: Configuración para el método “score”

Fuente: (Li Xudong, 2018).

2.2.16. Matriz de confusión.

La matriz de confusión o también conocida como matriz de error, es una herramienta que se utiliza para evaluar el rendimiento de un modelo de clasificación al comparar sus predicciones con las clases reales de los datos

Tabla 2-7: Matriz de confusión para clasificación binaria.

		Predicción	
		0	1
Real	0	TN	FP
	1	FN	TP

Realizado por: Barahona Diego, 2023

- Verdadero negativo (TN): representa el número de predicciones negativas (0) que se clasificaron correctamente.
- Falso positivo (FP): representa el número de predicciones positivas (1) que fueron clasificadas de manera incorrecta.
- Falso negativo (FN): representa el número de predicciones negativas que fueron clasificadas incorrectamente.
- Verdadero positivo (TP): representa el número de predicciones positivas que se clasificaron correctamente (Sotaquirá Miguel 2022).

En una clasificación multiclase con 5 clases, la matriz de confusión tendrá una forma similar a la siguiente:

Tabla 2-8: Matriz de confusión para clasificación multiclase.

		Predicción				
Real		Clase 0	Clase 1	Clase 2	Clase 3	Clase 4
	Clase 0	TP0	FN0	FN0	FN0	FN0
	Clase 1	FN1	TP1	FN1	FN1	FN1
	Clase 2	FN2	FN2	TP2	FN2	FN2
	Clase 3	FN3	FN3	FN3	TP3	FN3
	Clase 4	FN4	FN4	FN4	FN4	TP4

Realizado por: Barahona Diego, 2023

Donde:

- TP (True Positive): Indica el número de instancias que fueron clasificadas correctamente como la clase correspondiente. Por ejemplo, TP1 es el número de instancias de la clase 1 que fueron correctamente clasificadas como clase 1.

- FN (False Negative): Indica el número de instancias que pertenecen a la clase correspondiente, pero fueron incorrectamente clasificadas como otras clases. Por ejemplo, FN0 es el número de instancias de la clase 0 que fueron erróneamente clasificadas como otras clases en lugar de clase 0 (Sotaquirá Miguel, 2022).

Cada fila de la matriz representa una clase real, y cada columna representa la predicción del modelo para una clase. La diagonal principal (de arriba a la izquierda a abajo a la derecha) contiene los valores TP, que representan las predicciones correctas para cada clase.

El objetivo es tener la mayor cantidad posible de valores en la diagonal principal, ya que eso indicaría predicciones precisas. A partir de la matriz de confusión, se puede calcular varias métricas de evaluación, como la precisión, el recall (tasa de verdaderos positivos), el F1-score y otras, para cada clase por separado. Esto dará una idea de cómo el modelo está funcionando para cada clase en particular y en general para todas las clases.

2.2.16.1. Precisión.

La precisión es la tasa de positivos que se predijeron como positivos y en realidad también fueron positivos (Pensamientos de mariposa, 2023). Se lo calcula como:

$$\frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

2.2.16.2. Exactitud.

Representa el número de predicciones correctas realizadas por el clasificador (Pensamientos de mariposa, 2023) es la proporción de todas las instancias que fueron clasificadas correctamente. se lo calcula como:

$$\frac{\mathrm{TP0} + \mathrm{TP1} + \mathrm{TP2} + \mathrm{TP3} + \mathrm{TP4}}{\text{Total de instancias}}$$

2.2.16.3. Sensibilidad (recall).

Representa la proporción de casos positivos reales que fueron clasificados correctamente(Pensamientos de mariposa, 2023), se lo calcula como:

$$\frac{\mathrm{TP}_{\chi}}{\mathrm{TP}_{\chi} + \mathrm{FN}_{\chi}}$$

2.2.16.4. Especificidad.

Representa la proporción de casos negativos reales que se clasificaron correctamente(Pensamientos de mariposa, 2023), se lo calcula como:

$$\frac{\mathrm{TN}}{\mathrm{TP} + \mathrm{FN}}$$

La especificidad no se aplica directamente en una clasificación multiclase, ya que no hay una única clase de "Negativa". Esta métrica es más relevante en problemas de clasificación binaria.

2.2.16.5. Puntaje F1

Representa el promedio ponderado de la precisión y la sensibilidad (Pensamientos de mariposa, 2023), se lo calcula como:

$$2(\frac{Precision\ para\ Clase\ x \times Recall\ para\ Clase\ x}{Precision\ para\ Clase\ x + Recall\ para\ Clase\ x})$$

2.2.17. Aprendizaje no supervisado

Este tipo de aprendizaje no conocemos la respuesta, pero si conocemos algunas características o propiedades. El aprendizaje no supervisado es similar al método que utilizamos para aprender hablar cuando somos bebes, en un principio escuchamos hablar a nuestros padres y no entendemos nada; pero a medida que vamos escuchando miles de conversaciones, nuestro cerebro comenzará a formar un modelo sobre cómo funciona el lenguaje y comenzaremos a reconocer patrones y a esperar ciertos sonidos.(Nolasco Jorge, 2018)

CAPÍTULO III

3. MARCO METODOLÓGICO

3.1. Enfoque de investigación

Conociendo la importancia y lo fundamental que puede llegar ser los rodamientos principalmente dentro de las máquinas rotativas se ha ido desarrollando algunos métodos que van siendo cada vez más efectivos para su identificación de fallas mediante técnicas de diagnóstico.

Con la identificación a tiempo de fallas en rodamientos se puede llegar a ser muy decisivo al momento de identificar la causa raíz de los fallos de la máquina. Para realizar este proceso es necesario contar con información sobre el historial de funcionamiento del rodamiento. Hay que tener en cuenta que un solo registro no puede decirnos suficiente como para extraer alguna característica en particular lo que significa que necesitamos una gran cantidad de registros en este caso de análisis de rodamientos con ultrasonido.

El análisis de datos es un enfoque comúnmente utilizado en el enfoque mixto de investigación, ya que permite combinar y analizar datos cuantitativos y cualitativos. En el análisis de datos mixtos, se recopilan y analizan tanto datos numéricos como datos de texto. Los datos numéricos se pueden analizar utilizando técnicas estadísticas, mientras que los datos textuales o verbales se pueden analizar utilizando técnicas de análisis de contenido.

3.2. Diseño de Investigación

3.2.1. *Obtención de la base de datos*

Para empezar a trabajar es necesario una base de datos de análisis de fallas en rodamientos y para esto se utilizará una base de datos obtenidas mediante la aplicación de diferentes técnicas de mantenimiento basado en la condición en rodamientos, siendo parte del Trabajo de Integración Curricular de (Guangaxi Pilar, 2022). Disponible en el repositorio de la Escuela Superior Politécnica de Chimborazo.

3.2.1.1. *Descripción de los datos*

La base de datos obtenida se trata del análisis de la evolución de fallas en rodamientos mediante la aplicación de diferentes técnicas del mantenimiento basado en la condición como son: análisis de vibraciones, ultrasonido, temperatura y energía. Está conformada por 2008 filas y 113 columnas haciendo un total de 226904 datos recolectados los cuales están distribuidos de acuerdo con la técnica aplicada (Guangaxi Pilar, 2022, p. 51).

- Para el análisis de vibración se tiene 2008 filas y 14 columnas sumando 28112 datos equivalente al 12,39% de los datos totales.(Guangaxi Pilar, 2022)

- Para el análisis termográfico se tiene 2008 filas y 9 columnas sumando 18072 datos equivalente al 7,96% de los datos totales.(Guangaxi Pilar, 2022)

- Para el análisis de ultrasonido se tiene 2008 filas y 2 columnas sumando 4016 datos equivalente al 1,77% del total de datos.(Guangaxi Pilar, 2022)

- Para el análisis de energía se tiene 2008 filas y 88 columnas sumando 176704 datos que equivale al 77,88% del total de datos.(Guangaxi Pilar, 2022)

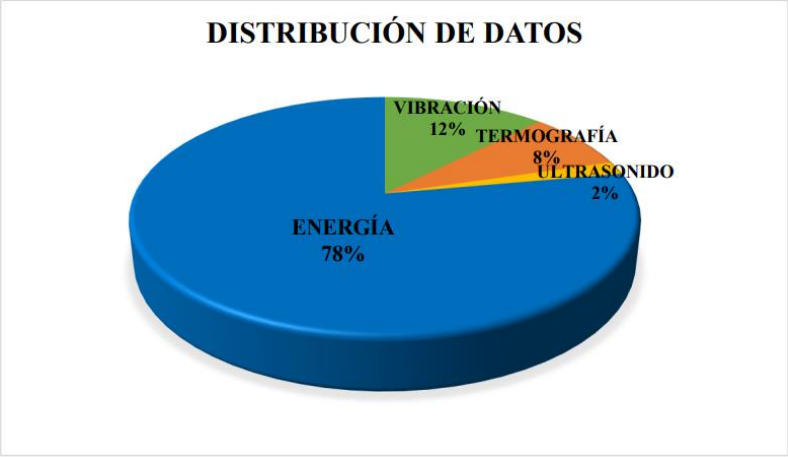


Ilustración 3-1: Distribución de datos con diferentes técnicas del mantenimiento.

Fuente: (Guangaxi Pilar, 2022)

3.2.1.2. *Librerías utilizadas para el pretratamiento de datos.*

Las librerías son colecciones de módulos y funciones que se puede importar y utilizar en un propio código. Esto permite aprovechar el trabajo previo de otros desarrolladores y ahorra

tiempo al no tener que escribir todo el código necesario desde cero. Estas herramientas encapsulan el conocimiento y la experiencia de otros desarrolladores, representando una vía eficiente para aprovechar el trabajo previo y acelerar el proceso de creación de software.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import functions_IF as fns                # Módulo propio
```

Ilustración 3-2: Diferentes librerías utilizadas en Python.

Realizado por: Barahona Diego, 2023.

3.2.1.3. *Análisis exploratorio de los datos.*

Permite obtener una visión general de la estructura y formato de los datos. Esto incluye identificar el tipo de variables presentes, la distribución de valores, y la presencia de datos faltantes. Durante el análisis exploratorio, es posible identificar la necesidad de transformaciones en los datos, como la normalización, escalado o manejo de datos faltantes. Estas preparaciones son esenciales para garantizar la calidad de entrada en modelos de aprendizaje automático.

Para poder cargar la información se puede cambiar el formato excel a un formato csv. Después se cargan los datos almacenados en el archivo "roddata.csv." y muestra una descripción general de los datos.

	Item	Frecuencia (Hz)	Vrms (mm/s)	Frecuencia 1 (HZ)	Amplitud 1 (gE)	BPFO (Hz)	Frecuencia 2 (Hz)	Amplitud 2 (gE)
0	1	59.89	8.83	301.01	0.01	300.33	764.02	0.0
1	2	59.88	9.48	299.86	0.01	299.65	762.87	0.0
2	3	59.87	9.25	299.86	0.01	299.71	762.87	0.0
3	4	59.88	8.73	299.86	0.01	299.70	762.87	0.0
4	5	59.87	8.75	299.86	0.02	299.72	762.87	0.0

5 rows × 119 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2007 entries, 0 to 2006
Columns: 119 entries, Item to Sonido
dtypes: float64(116), int64(1), object(2)
memory usage: 1.8+ MB
```

Ilustración 3-3: Visualización de la base de datos.

Realizado por: Barahona Diego, 2023.

En la Ilustración 3-3. se observa la descripción de la información de manera general que nos dice que se cuenta con 2007 filas de entradas y 119 columnas de entradas, seguidamente la información de las primeras filas de forma general. Para poder visualizar la manera que están distribuidos los datos se realiza relacionando la columna “Estado Rodamiento” e “Item”.

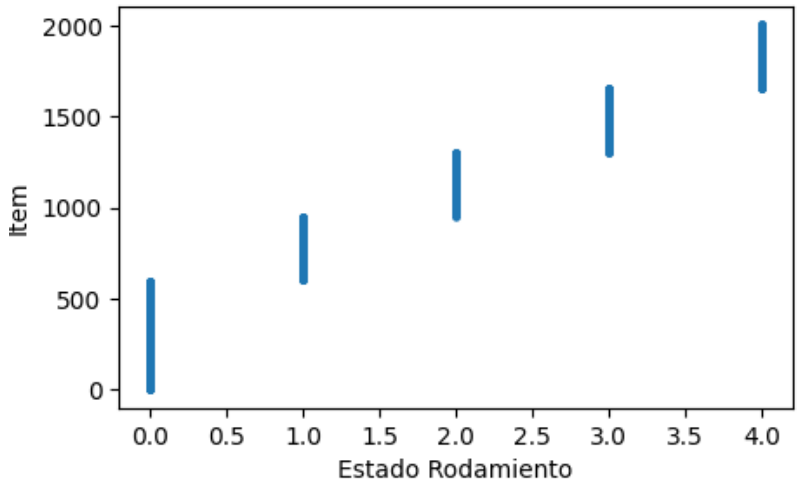


Ilustración 3-4: Dispersión de los datos en relación con el estado de rodamiento.

Realizado por: Barahona Diego, 2023.

Como se observa en la Ilustración 4-3. del total de mediciones que son 2007 ítems, están distribuidas en cinco diferentes estados de rodamientos los cuales están descritos de la siguiente Tabla 1-3.

Tabla 3-1: Descripción de la toma de datos.

Estado Rodamiento	Descripción	Número datos
0	Rodamiento sin fallas.	600
1	Rodamiento con falla en pista externa.	352
2	Rodamiento con falla en pista interna.	352
3	Rodamiento con falla en canastilla.	352
4	Fallas en elementos rodantes.	352

Fuente: (Guangaxi Pilar, 2022)

3.2.2. *Preprocesamiento de los datos*

El preprocesamiento de datos es una fase crítica en el análisis de datos que implica la preparación y transformación de los datos brutos para garantizar que sean adecuados y útiles para su análisis.

Se necesita separar los datos en base a la variable objetivo “Estado Rodamiento” además de las columnas relacionadas al ultrasonido (Frecuencia de ultrasonido e Intensidad del sonido) que interesa analizar además de otras columnas como: Ítem y Estado Rodamiento que sirven para complementar el análisis de los datos.

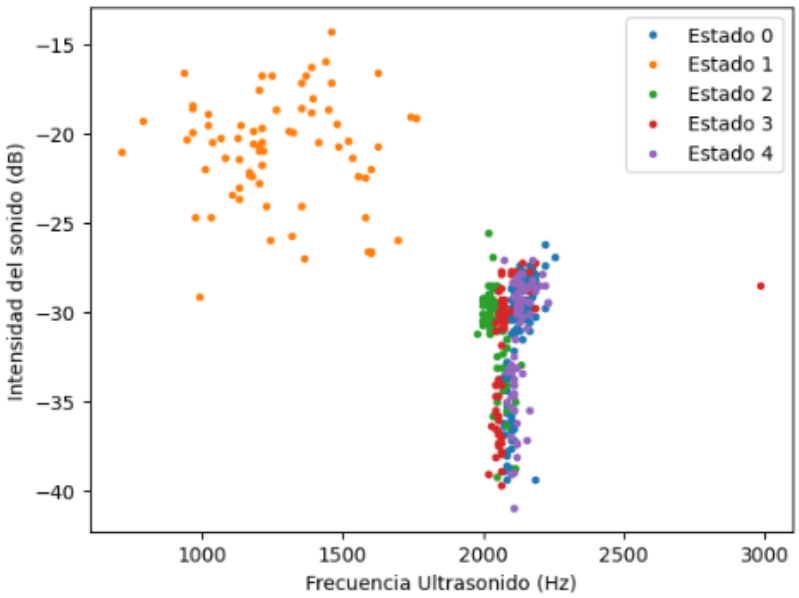


Ilustración 3-5: Distribución de datos por estado de rodamiento.

Realizado por: Barahona Diego, 2023.

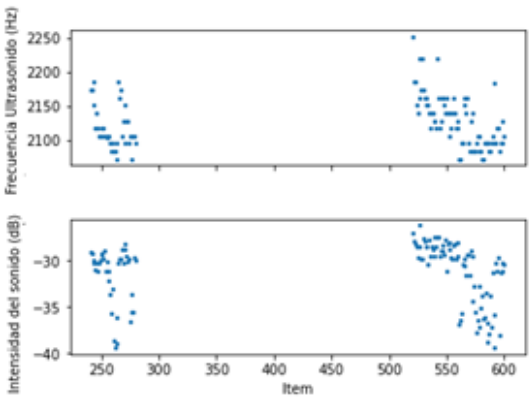
Para esto se puede utilizar la función “query” que filtra las columnas y su código se encuentra anexado. En la Ilustración 5-3. se visualiza un gráfico de dispersión donde se muestra la distribución de los datos en relación con las columnas Frecuencia de ultrasonido e Intensidad del sonido.

3.2.2.1. *Analizando datos vacíos.*

a) Agrupación de estado 0.

Para analizar los datos vacíos en el Estado 0 y en el resto se va a tomar en cuenta las columnas “Frecuencia Ultrasonido (Hz)” e “Intensidad del sonido (dB)”. Se describe los datos para verificar cuantos datos faltantes existen y según la descripción detallada en la Ilustración 6-3 A. se observa que en las columnas “Frecuencia Ultrasonido (Hz)” e “Intensidad del sonido (dB)” se detecta 480 datos faltantes y que solo existen 120 registros de datos haciendo un total de 600 existentes, pero con valores faltantes; además, en la parte B representa un diagrama de dispersión que muestra la manera que están distribuidos los datos del estado 0.

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    480
Intensidad del sonido (dB)     480
dtype: int64
count     120.000000
mean     2125.083333
std        36.121314
min      2073.000000
25%      2096.000000
50%      2118.000000
75%      2152.000000
max      2252.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    120.000000
mean     -31.271083
std        3.306273
min      -39.400000
25%      -33.457500
50%      -29.950000
75%      -28.850000
max      -26.180000
Name: Intensidad del sonido (dB), dtype: float64
```



A B

Ilustración 3-6: a) Descripción de Estado 0. b) Diagrama de dispersión Estado 0.

Realizado por: Barahona Diego, 2023.

b) Agrupación de estado 1.

Se realiza el procedimiento anterior de describir los datos faltantes para luego poder visualizar con el diagrama de dispersión. Así mismo en la descripción detallada en la Ilustración 7-3 A. se detectan que en las columnas “Frecuencia Ultrasonido (Hz)” e “Intensidad del sonido (dB)” se detecta 280 datos faltantes y que solo existen 72 registros de datos haciendo un total de 352 datos entre los dos. Se grafica el diagrama de dispersión para ver la existencia de datos faltantes.

A B

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      72.000000
mean     1272.593056
std       232.952934
min       712.400000
25%      1128.250000
50%      1220.500000
75%      1450.000000
max      1762.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -20.908333
std       3.047213
min     -29.120000
25%     -22.420000
50%     -20.530000
75%     -18.882500
max     -14.300000
Name: Intensidad del sonido (dB), dtype: float64
```

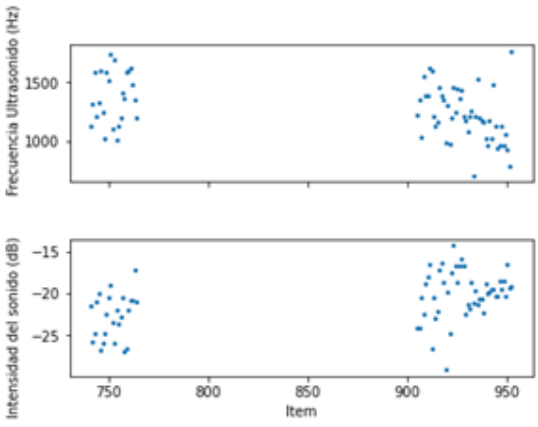


Ilustración 3-7: a) Descripción del Estado 1. b) Dispersión en Estado 1.

Realizado por: Barahona Diego, 2023.

c) Agrupación de estado 2.

De la misma manera en la Ilustración 8-3 A. se detectan que existen 280 datos faltantes en las columnas “Frecuencia Ultrasonido (Hz)” e “Intensidad del sonido (dB)” y también se ve que

hay solo 72 registros de datos que sumado los dos da como resultado 352 registros seguido de su respectivo diagrama de dispersión de datos.

A

```
# Estado 2
fns.show_states_analysis(df_st2, lst_cols)
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      72.000000
mean     2037.694444
std        30.804535
min      1976.000000
25%      2021.250000
50%      2032.000000
75%      2049.000000
max      2132.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -31.214444
std       2.643797
min     -39.240000
25%     -32.587500
50%     -30.420000
75%     -29.480000
max     -25.600000
Name: Intensidad del sonido (dB), dtype: float64
```

B

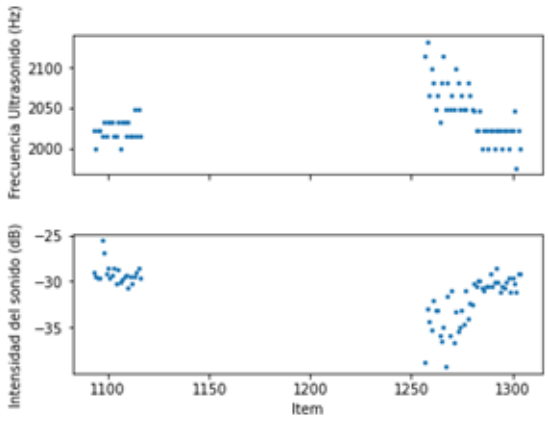


Ilustración 3-8: a) Descripción del Estado 2. b) Dispersión en Estado 2.

Realizado por: Barahona Diego, 2023.

d) Agrupación de estado 3.

Nuevamente en la Ilustración 9-3 A. se detectan que existen solo 71 registros de datos en las columnas “Frecuencia Ultrasonido (Hz)” e “Intensidad del sonido (dB)” así también que existen 280 datos faltantes dando un total de 351 registros. Se grafica su respectivo diagrama de dispersión para detectar la distribución de los datos.

A

```
# Estado 3
fns.show_states_analysis(df_st3, lst_cols)
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      71.000000
mean     2087.971831
std       113.835090
min      2017.000000
25%      2051.000000
50%      2062.000000
75%      2096.000000
max      2984.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    71.000000
mean    -31.775634
std       3.648708
min     -39.710000
25%     -35.070000
50%     -30.110000
```

B

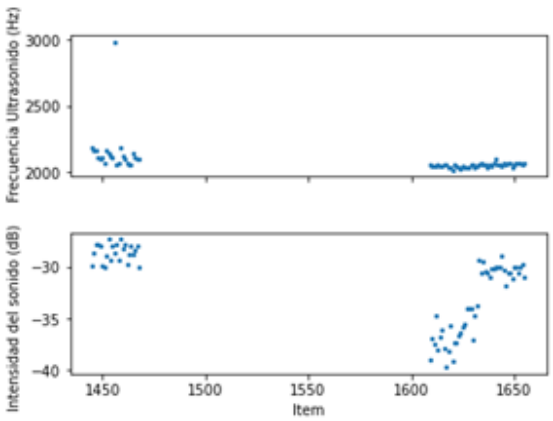


Ilustración 3-9: a) Descripción del Estado 3. b) Dispersión en Estado 3.

Realizado por: Barahona Diego, 2023.

e) Agrupación de estado 4.

En la Ilustración 10-3 A. se detectan que existen solo 72 registros de datos en las columnas "Frecuencia Ultrasonido (Hz)" e "Intensidad del sonido (dB)" y una cantidad de 280 datos faltantes seguido del diagrama de dispersión que muestra la distribución de los datos.

A

```
# Estado 4
fns.show_states_analysis(df_st4, lst_cols)
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      72.000000
mean     2131.111111
std        32.953897
min      2073.000000
25%      2107.000000
50%      2118.000000
75%      2140.000000
max      2230.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -31.323889
std       3.424509
min     -40.970000
25%     -33.810000
50%     -29.800000
```

B

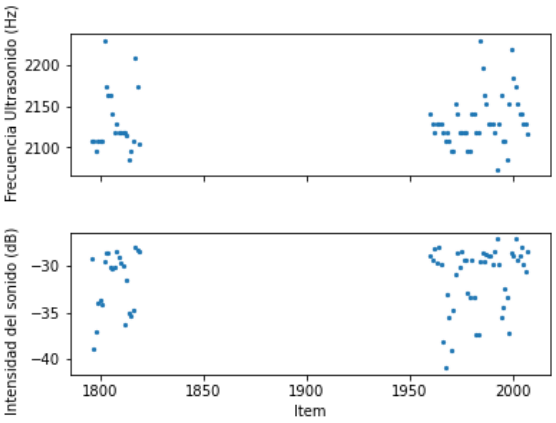


Ilustración 3-10: a) Descripción del Estado 4. b) Dispersión en Estado 4.

Realizado por: Barahona Diego, 2023.

3.2.2.2. *Filtrado de datos vacíos.*

Dado que existen datos faltantes en la base de datos se filtrará los datos de las columnas de Intensidad del sonido (dB) y frecuencia ultrasonido (Hz) sin tomar en cuenta los datos faltantes en cada uno de los estados del rodamiento. En la Ilustración 11-3. se observa un solo ejemplo del estado 0 con los datos agrupados sin datos faltantes y de la misma manera se realiza para los demás estados. El procedimiento completo se encuentra anexada.

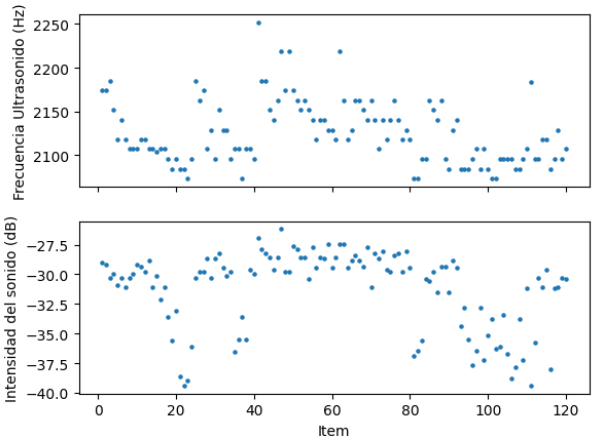


Ilustración 3-11: Dispersión sin datos faltantes en Estado 0

Realizado por: Barahona Diego, 2023.

Realizado el filtrado de los datos faltantes en cada estado del rodamiento, se concatena y se guarda en un archivo CSV con el nombre "df_IF_sonido.csv" en donde se encuentran las columnas: Item, Frecuencia Ultrasonido (Hz) y Estado Rodamiento.

3.2.2.3. *Análisis del estado de rodamiento.*

Se utiliza la biblioteca matplotlib y seaborn para graficar un diagrama de barras y se representa la distribución de datos por estado de los rodamientos como se representa en la Ilustración 12-3.

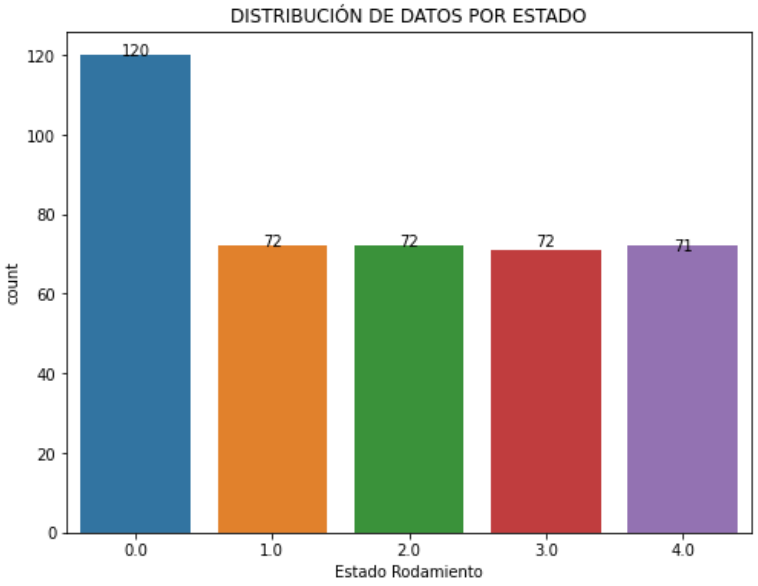


Ilustración 3-12: Distribución de datos una vez filtrado valores faltantes.

Realizado por: Barahona Diego, 2023.

Se puede evidenciar que en el estado 0 existen 120 datos mientras que en los estados:1, 2 y 3 existen 72 datos hasta 71 datos en el estado 4 únicamente. Como son muy pocos datos para empezar a trabajar con el modelo entonces se utiliza algunas funciones que permiten balancear y sobre muestrear datos.

a) Sobre muestreo de datos

Dado a que se observó un desbalanceo de datos antes, el Dataframe fue sobre muestreado hacia 25000 datos utilizando el comando SMOTE que pertenece a la librería de imblearn. Para poder analizar se genera un gráfico de barras que representa la distribución de datos en los cinco estados de funcionamiento del rodamiento, cada barra muestra el número de datos para cada estado como se observa en la Ilustración 13-3. Una vez realizado el sobre muestreo se guarda en Dataframe generado con el nombre: "IF_sonido_25k.csv".

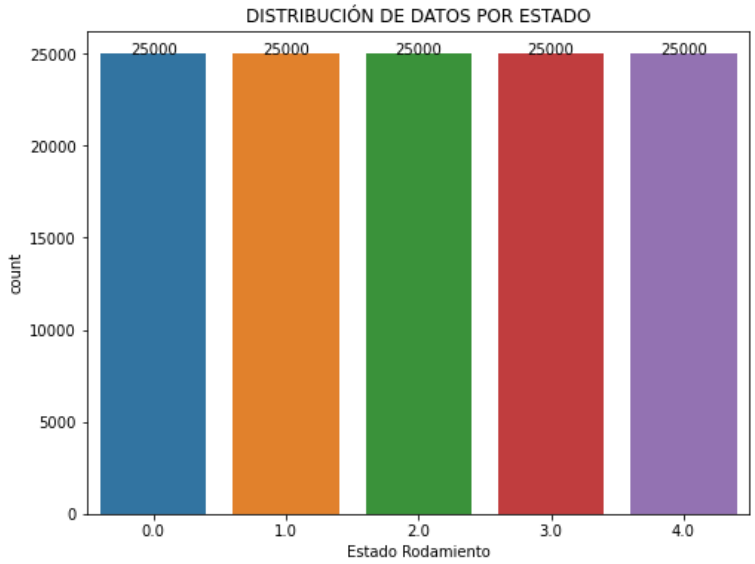


Ilustración 3-13: Distribución de datos balanceado a 25000.

Realizado por: Barahona Diego, 2023.

3.2.3. *División de la base de datos.*

Para la división de los datos se va a tomar un porcentaje del 20% de los datos totales para la prueba mientras que el 80 % de los datos se utilizará para el entrenamiento.

```
y1_train.value_counts()
```

```
Estado Rodamiento
1.0    20086
0.0    20024
4.0    19999
2.0    19993
3.0    19898
Name: count, dtype: int64
```

(a)

```
y1_test.value_counts()
```

```
Estado Rodamiento
3.0    5102
2.0    5007
4.0    5001
0.0    4976
1.0    4914
Name: count, dtype: int64
```

(b)

Ilustración 3-14: División de datos. (a) entrenamiento y (b) prueba.

Realizado por: Barahona Diego, 2023.

En la Ilustración 14-3. se muestra la cantidad de datos obtenidos. Se denomina “y1_train” a los datos de entrenamiento que son aproximadamente 20.000 siendo el ochenta por ciento de los datos totales. De la misma manera se denomina “y1_test” a los datos que servirá para la prueba del modelo y es menor alrededor de 5.000 que representa el 20 por ciento de los datos totales.

Nuevamente se guarda los Dataframe generados como “IF_25k_train.csv” y “IF_25_test.csv” pero para trabajar de mejor manera se los llama “df_train” y “df_test” respectivamente representados en la Ilustración 15-3.

Concatenación y guardado IF_25k_train

```
df_train= pd.concat([X1_train,y1_train], axis=1)
df_train.to_csv("data/IF_25k_train.csv")
```

Concatenación y guardado IF_25_test

```
df_test= pd.concat([X1_test,y1_test], axis=1)
df_test.to_csv("data/IF_25k_test.csv")
```

Ilustración 3-15: Datos de entrenamiento y prueba.

Realizado por: Barahona Diego, 2023.

3.2.4. *Extracción de características*

Antes de comenzar la extracción de las características hay que realizar alguna modificación como: eliminar la columna generada por defecto de nombre "Unnamed: 0" y clasificar cada estado de falla para realizar de manera individual la extracción de la característica donde la variable objetivo es "Estado Rodamiento".

3.2.4.1. *Extracción de características en datos de entrenamiento "df_train".*

Se le asigna las variables de IF_0, IF_1, IF_2, IF_3 y IF_4 para cada estado de rodamiento por individual para separar los datos individualmente, además se muestra solamente los datos de IF_0 para comprobar la filtración de los datos, este paso se lo realizo anteriormente mediante la función "query". Ilustración 16-3.

```
IF_0 = df_train.query('`Estado Rodamiento` == 0')
IF_1 = df_train.query('`Estado Rodamiento` == 1')
IF_2 = df_train.query('`Estado Rodamiento` == 2')
IF_3 = df_train.query('`Estado Rodamiento` == 3')
IF_4 = df_train.query('`Estado Rodamiento` == 4')
```

Ilustración 3-16: Clasificación individual de estado del rodamiento en entrenamiento.

Realizado por: Barahona Diego, 2023.

Para la extracción de características se va a utilizar la librería TSFEL en cada uno de los modos de falla del rodamiento para al final concatenar en un solo documento todas las características para entrenamiento extraídas que dio como resultado 280 columnas. Para visualizar las características extraídas se muestra únicamente los datos de IF_0, el resto de procedimiento se lo puede encontrar en Anexos. Ilustración 17-3.

*** Feature extraction finished ***

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0
0	5.370682e+07	232.436256	5.370682e+07	0.055025	2.0	9.0	2084.000000
1	5.313260e+07	231.645000	5.313260e+07	0.054818	2.0	9.0	2073.000000
2	5.483465e+07	235.298849	5.483465e+07	0.055189	2.0	9.0	2105.368974
3	5.411012e+07	233.575000	5.411012e+07	0.055026	2.0	9.0	2107.000000
4	5.377603e+07	232.660233	5.377603e+07	0.055203	2.0	9.0	2084.000000
...	...	...	...	...	...	...	...
1663	5.379720e+07	232.645000	5.379720e+07	0.054584	2.0	9.0	2084.000000
1664	5.374864e+07	232.760000	5.374864e+07	0.055581	2.0	9.0	2084.000000
1665	5.503303e+07	235.594747	5.503303e+07	0.055499	2.0	9.0	2107.000000
1666	5.431373e+07	234.210000	5.431373e+07	0.054722	2.0	9.0	2107.000000
1667	5.407704e+07	233.360000	5.407704e+07	0.054837	2.0	9.0	2084.000000

1668 rows × 280 columns

Ilustración 3-17: Características extraídas de "IF_0" para entrenamiento.

Realizado por: Barahona Diego, 2023.

Seguidamente se concatena las características extraídas para entrenamiento a cada estado de rodamiento y se lo guarda en un solo documento de nombre "features_train_fin.csv" para poder utilizar más adelante

3.2.4.2. *Extracción de características en datos de prueba "df_test".*

Se le asigna las variables de IF_0t, IF_1t, IF_2t, IF_3t y IF_4t para cada estado de rodamiento por individual, además se muestra solamente los datos de IF_0t para comprobar la filtración de los datos, este paso se lo realizo anteriormente mediante la función "query". Ilustración 18-3.

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
4414	2107.0	-28.690000	0.0
17638	2084.0	-37.125439	0.0
23341	2107.0	-28.690000	0.0
6903	2084.0	-33.991107	0.0
10950	2163.0	-28.248403	0.0
...	...	...	...
16201	2107.0	-29.700588	0.0
4421	2152.0	-28.619601	0.0
16446	2096.0	-35.625150	0.0
7818	2107.0	-30.898271	0.0
20884	2174.0	-29.707699	0.0

4976 rows × 3 columns

Ilustración 3-18: Clasificación individual de la variable objetivo de prueba.

Realizado por: Barahona Diego, 2023.

Para la extracción de características se va a utilizar la librería TSFEL en cada uno de los modos de falla del rodamiento para al final concatenar en un solo documento todas las características para entrenamiento extraídas que dio como resultado 280 columnas. Para visualizar las características extraídas se muestra únicamente los datos de IF_0t, el resto de procedimiento se lo puede encontrar en Anexos. Ilustración 19-3.

```
*** Feature extraction finished ***
```

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0
0	5.341745e+07	232.145000	5.341745e+07	0.055079	2.0	9.0	2084.0
1	5.403551e+07	233.274252	5.403551e+07	0.054690	2.0	9.0	2096.0
2	5.382727e+07	233.115013	5.382727e+07	0.054615	2.0	9.0	2084.0
3	5.367632e+07	232.801278	5.367632e+07	0.055069	2.0	9.0	2084.0
4	5.516313e+07	235.623511	5.516313e+07	0.054229	2.0	9.0	2073.0
...	...	...	...	...	...	...	...
409	5.379780e+07	232.876560	5.379780e+07	0.054941	2.0	9.0	2096.0
410	5.534673e+07	236.365137	5.534673e+07	0.054309	2.0	9.0	2096.0
411	5.431334e+07	234.090000	5.431334e+07	0.055003	2.0	9.0	2084.0
412	5.411379e+07	233.168750	5.411379e+07	0.054769	2.0	9.0	2084.0
413	5.427250e+07	234.000000	5.427250e+07	0.054769	2.0	9.0	2096.0

414 rows × 280 columns

Ilustración 3-19: Características extraídas de "IF_0t" para prueba.

Realizado por: Barahona Diego, 2023.

Nuevamente se realiza la concatenación de las características extraídas para la prueba a cada estado de rodamiento y se lo guarda en un solo documento de nombre "features_test_fin.csv" para su uso más adelante.

3.3.5. Aplicación del algoritmo ELM a la base de datos de ultrasonido.

a) Importación de librerías para el modelo ELM

Se importan las librerías necesarias para implementar el algoritmo ELM. Iniciando por el mismo "elm" que es una biblioteca personalizada creada por el autor (Li Xudong, 2018) para implementar la Máquina de Aprendizaje Extremo (ELM), contiene la clase ElmNetwork, que proporciona métodos para entrenar y utilizar el modelo ELM. "numpy" para trabajar con matrices, "pandas" para trabajar con DataFrames y realizar el preprocesamiento de datos, además "sklearn" que proporciona herramientas de aprendizaje automático para tareas de clasificación y regresión.

```
import elm
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, LabelEncoder
```

Ilustración 3-20: Librerías utilizadas para el modelo ELM.

Realizado por: Barahona Diego, 2023.

b) Preprocesamiento de datos

Se llama a los archivos "csv" que contiene los datos separados anteriormente para entrenamiento y prueba. Ilustración 21-3.

```
# se llama los archivos de la división de datos para entrenamiento y prueba.

df_train = pd.read_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/features_train_fin.csv")

df_test = pd.read_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/features_test_fin.csv")
df_train
```

Ilustración 3-21: Dirección del archivo a ingresar al modelo ELM.

Realizado por: Barahona Diego, 2023.

Se utiliza el objeto "RobustScaler" previamente creado para escalar las características en el conjunto de entrenamiento (x_train) y el conjunto de prueba (x_test). Por último, se reinicia el índice del DataFrame para asegurarse de que sea secuencial desde 0 hasta n-1 lo que evita posibles problemas durante el entrenamiento y la evaluación del modelo.

```
# Crear un escalador RobustScaler
stdsc = RobustScaler()

# Escalar las características del conjunto de entrenamiento
x_train_scaled = stdsc.fit_transform(x_train)

# Escalar las características del conjunto de prueba
x_test_scaled = stdsc.transform(x_test)

# Convertir las etiquetas a valores numéricos enteros en el rango de 0 a n-1
label_encoder = LabelEncoder()

# Codificar las etiquetas del conjunto de entrenamiento
y_train_encoded = label_encoder.fit_transform(y_train)

# Codificar las etiquetas del conjunto de prueba
y_test_encoded = label_encoder.transform(y_test)
```

Ilustración 3-22: Escalado de datos y codificación de etiquetas.

Realizado por: Barahona Diego, 2023.

3.2.4.3. *Entrenamiento del modelo ELM*

Se crea un objeto "elm.elm" (representando el modelo ELM) con los siguientes argumentos:

- **hidden_units:** 1700 neuronas
- **activation_function**: Relu en este caso.
- **random_type:** Tipo de generación de pesos aleatorios en la capa oculta (distribución normal).
- **x:** Conjunto de características de entrenamiento.
- **y:** Etiquetas del objetivo codificadas correspondientes al conjunto de entrenamiento.
- **C:** Parámetro de regularización de la ELM.
- **elm_type:** Tipo de ELM ('clf' para clasificación en este caso).
- **one_hot:** Si es True, las etiquetas objetivo se codificarán en formato "one-hot encoding".

```
# Crear el modelo ELM para esta iteración
model = elm.elm(hidden_units=1700, activation_function='relu',
                random_type='normal', x=x_train_scaled, y=y_train_encoded,
                C=0.1, elm_type='clf', one_hot=True)

# Entrenar el modelo en esta iteración
beta, train_accuracy, running_time = model.fit('no_re')
```

Ilustración 3-23: Entrenamiento del modelo ELM.

Realizado por: Barahona Diego, 2023.

Luego, el modelo se entrena con la función “fit ('no_re')”, donde “no_re” indica que no se utilizará el algoritmo recursivo para calcular los pesos de la capa de salida.

3.2.4.4. *Evaluación del modelo:*

Se realiza la predicción en el conjunto de prueba utilizando “predict”, se puede presentar variación en los porcentajes de precisión tanto en conjunto de entrenamiento como de prueba. Utilizar una validación cruzada puede proporcionar una estimación más robusta del rendimiento del modelo al considerar múltiples divisiones de datos.

CAPÍTULO IV

4. ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

A partir de este capítulo se realiza la interpretación y discusión de los resultados obtenidos al momento de ejecutar el modelo de clasificación de fallas en rodamientos mediante datos de ultrasonido. Se busca determinar el mayor porcentaje de efectividad del modelo ELM ejecutando el código de programación en la plataforma google colab con la finalidad de optimizar tiempos mediante el uso de aceleradores como la TPU. Además, se aplican varios algoritmos de aprendizaje de máquinas con el fin de comparar el rendimiento de los demás algoritmos.

4.1. Análisis con características extraídas de los datos de ultrasonido

4.1.1. Resultados del entrenamiento y prueba del modelo ELM.

```
Train Accuracy: 0.9416636658264314
Test Accuracy: 0.885151369533878
Classifier running time: 15.318721517000085
```

Ilustración 4-1: Resultados del entrenamiento y prueba con 1700 neuronas.

Realizado por: Barahona Diego, 2023.

Los resultados que se obtuvo mediante la validación cruzada muestran el rendimiento del modelo ELM en cada división de la validación cruzada. Cada “fold” representa una división diferente de los datos de entrenamiento y prueba, lo que brinda una idea de cómo se comporta el modelo en diferentes conjuntos de datos.

Además, se interpreta en los resultados que el promedio de todas las precisiones obtenidas en el conjunto de entrenamiento es de 94.16% mientras que el promedio de todas las precisiones en el conjunto de prueba es de 88.51% así como el tiempo que se demora en realizar todo este proceso matemático es de 15.31 segundos como se puede observar en la Ilustración 1-4. La brecha entre la precisión del entrenamiento y la precisión de la prueba, en este caso, no parece ser muy grande, lo que sugiere que el modelo generaliza razonablemente bien a datos desconocidos.

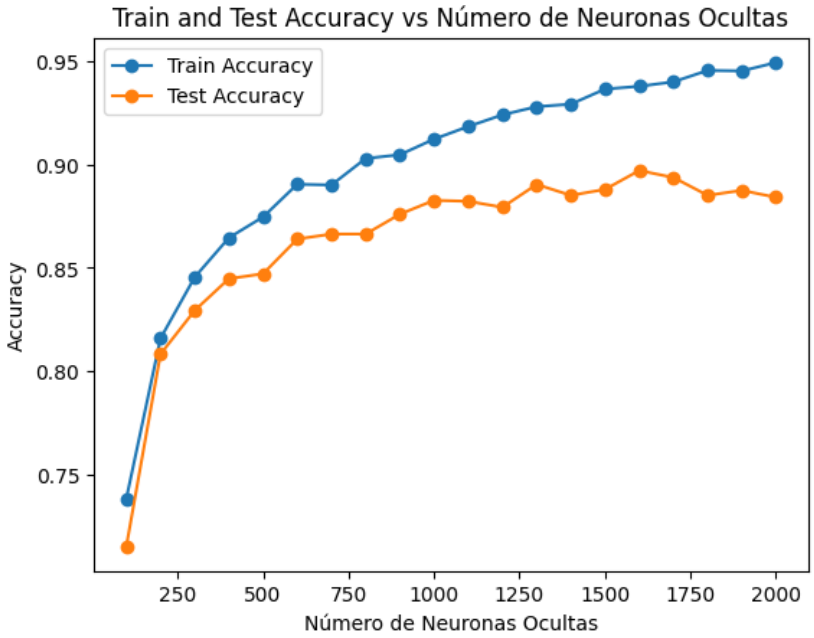


Ilustración 4-2: Diagrama de precisión de los datos de entrenamiento y prueba.

Realizado por: Barahona Diego, 2023.

4.1.2. Matriz de confusión del modelo de entrenamiento.

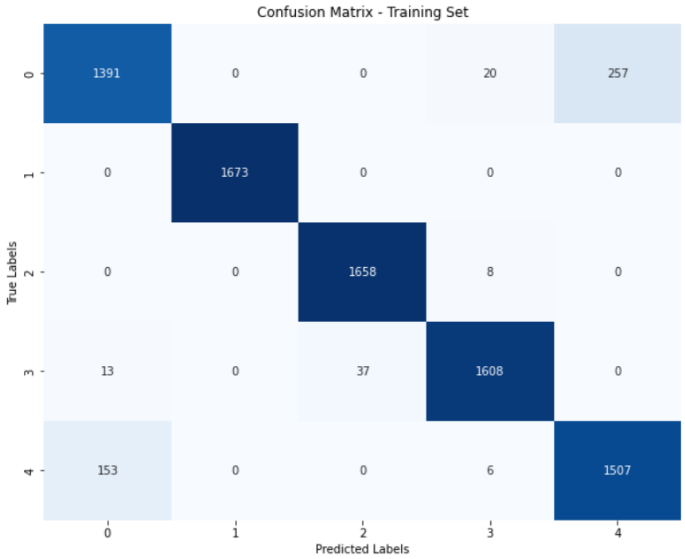


Ilustración 4-3: Matriz de confusión del conjunto de entrenamiento.

Realizado por: Barahona Diego, 2023.

Esta matriz muestra la cantidad de ejemplos dentro del conjunto de entrenamiento clasificados correcta e incorrectamente para cada clase, tanto en las filas (True Labels) como en las columnas (Predicted Labels). Ilustración 3-4.

- La primera fila representa el estado real "Estado 0". El modelo predijo correctamente 1391 ejemplos como "Estado 0", mientras que se confundió con 20 ejemplos clasificándolos erróneamente como "Estado 3" y 257 ejemplos como "Estado 4".

- La segunda fila representa el estado real "Estado 1". El modelo predijo correctamente 1673 ejemplos como "Estado 1".

- La tercera fila representa el estado real "Estado 2". El modelo predijo correctamente 1658 ejemplos como "Estado 2", mientras que se confundió con 8 ejemplos clasificándolos erróneamente como "Estado 3".

- La cuarta fila representa el estado real "Estado 3". El modelo predijo correctamente 1608 ejemplos como "Estado 3", mientras que se confundió con 13 ejemplos clasificándolos erróneamente como "Estado 0", y 37 ejemplos como "Estado 2".

- La quinta fila representa el estado real "Estado 4". El modelo predijo correctamente 1507 ejemplos como "Estado 4", mientras que se confundió con 153 ejemplos clasificándolos erróneamente como "Estado 0" y 6 ejemplos. Como “Estado 3”

	precision	recall	f1-score	support
0	0.89	0.83	0.86	1668
1	1.00	1.00	1.00	1673
2	0.98	1.00	0.99	1666
3	0.98	0.97	0.97	1658
4	0.85	0.90	0.88	1666
accuracy			0.94	8331
macro avg	0.94	0.94	0.94	8331
weighted avg	0.94	0.94	0.94	8331

```
Accuracy: 94.07033969511464%
Precision: 94.10329151497679%
Sensibilidad: 94.07033969511464%
```

Ilustración 4-4: Evaluación de las métricas en el conjunto de prueba.

Realizado por: Barahona Diego, 2023.

- **Precisión:** La precisión es la proporción de ejemplos clasificados como positivos que son realmente positivos. Se calcula para cada clase por separado. En este caso, los valores de precisión para cada clase son: 89% para la clase 0, 100% para la clase 1, 98% para la clase 2, 98% para la clase 3 y 85% para la clase 4. Ilustración 4-4.

- **Recall (Sensibilidad):** La sensibilidad es la proporción de ejemplos positivos que fueron correctamente clasificados como positivos. También se calcula para cada clase por separado. En este caso, los valores de sensibilidad para cada clase son: 83% para la clase 0, 100% para la clase 1, 100% para la clase 2, 97% para la clase 3 y 90% para la clase 4.

- **F1-score:** El F1-score es una medida que combina la precisión y la sensibilidad en una sola métrica. Representa el promedio armónico de precisión y sensibilidad y es útil cuando hay un desequilibrio entre las clases. En este caso, los valores de F1-score para cada clase son: 86% para la clase 0, 100% para la clase 1, 99% para la clase 2, 97% para la clase 3 y 88% para la clase 4.

4.1.3. Matriz de confusión del modelo de prueba.

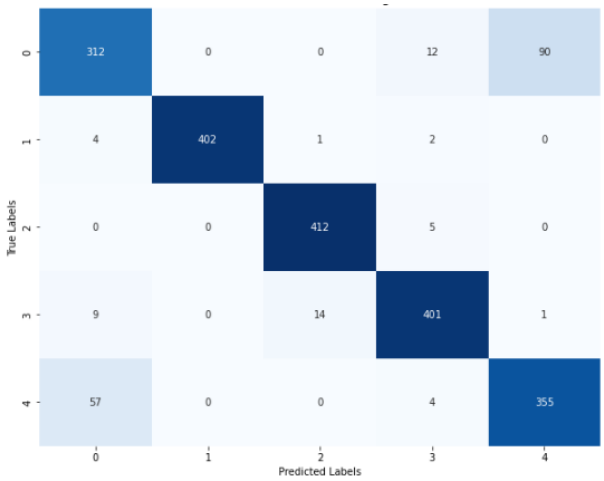


Ilustración 4-5: Matriz de confusión del modelo ELM de prueba.

Realizado por: Barahona Diego, 2023.

Esta matriz muestra la cantidad de ejemplos clasificados correcta e incorrectamente para cada clase, tanto en las filas (True Labels) como en las columnas (Predicted Labels). Ilustración 5-4.

- La primera fila representa el estado real "Estado 0". El modelo predijo correctamente 312 ejemplos como "Estado 0", mientras que se confundió con 12 ejemplos clasificándolos erróneamente como "Estado 3" y 90 ejemplos como "Estado 4".

- La segunda fila representa el estado real "Estado 1". El modelo predijo correctamente 402 ejemplos como "Estado 1", mientras que clasificó incorrectamente 4 ejemplos como “Estado 0”, 1 ejemplo como "Estado 2", y 2 ejemplos como “Estado 3”.

- La tercera fila representa el estado real "Estado 2". El modelo predijo correctamente 412 ejemplos como "Estado 2", mientras que se confundió con 5 ejemplos clasificándolos erróneamente como "Estado 3".

- La cuarta fila representa el estado real "Estado 3". El modelo predijo correctamente 401 ejemplos como "Estado 3", mientras que se confundió con 9 ejemplos clasificándolos erróneamente como "Estado 0", 14 ejemplos como "Estado 2" y 1 ejemplo como “Estado 4”.

- La quinta fila representa el estado real "Estado 4". El modelo predijo correctamente 355 ejemplos como "Estado 4", mientras que se confundió con 57 ejemplos clasificándolos erróneamente como "Estado 0" y 4 ejemplos. Como “Estado 3”

	precision	recall	f1-score	support
0	0.82	0.75	0.78	414
1	1.00	0.98	0.99	409
2	0.96	0.99	0.98	417
3	0.95	0.94	0.94	425
4	0.80	0.85	0.82	416
accuracy			0.90	2081
macro avg	0.90	0.90	0.90	2081
weighted avg	0.90	0.90	0.90	2081

```
Accuracy: 90.43728976453627%
Precision: 90.46392333785782%
Sensibilidad: 90.43728976453627%
```

Ilustración 4-6: Análisis de las métricas de evaluación.

Realizado por: Barahona Diego, 2023.

- **Precisión:** La precisión es la proporción de ejemplos clasificados como positivos que son realmente positivos. Se calcula para cada clase por separado. En este caso, los valores de precisión para cada clase son: 82% para la clase 0, 100% para la clase 1, 96% para la clase 2, 95% para la clase 3 y 80% para la clase 4. Ilustración 6-4.

- **Recall (Sensibilidad):** La sensibilidad es la proporción de ejemplos positivos que fueron correctamente clasificados como positivos. También se calcula para cada clase por separado. En este caso, los valores de sensibilidad para cada clase son: 75% para la clase 0, 98% para la clase 1, 99% para la clase 2, 94% para la clase 3 y 85% para la clase 4.

- **F1-score:** El F1-score es una medida que combina la precisión y la sensibilidad en una sola métrica. Representa el promedio armónico de precisión y sensibilidad y es útil

cuando hay un desequilibrio entre las clases. En este caso, los valores de F1-score para cada clase son: 78% para la clase 0, 99% para la clase 1, 98% para la clase 2, 94% para la clase 3 y 82% para la clase 4.

4.1.4. Comparación del modelo ELM con otros algoritmos para clasificación.

Tabla 4-1: Comparación del ELM con otros algoritmos de clasificación.

MÉTODO	EXACTITUD	PRECISIÓN	SENSIBILIDAD
ELM	90.43%	90.46%	90.43%
Random Forest	96.44%	96.44%	96.44%
XGBoost	99.13%	99.13%	99.13%
Naive Bayes	90.67%	90.76%	90.67%
Árbol de decisión	96.92%	96.92%	96.92%
KNN	81.30%	82.32%	81.30%

Realizado por: Barahona Diego; 2023.

En la tabla proporcionada, se puede observar que el algoritmo aplicado ELM tiene una exactitud máxima en la fase de prueba del 90.43%, una precisión del 90.46% y una sensibilidad del 90.43%. Comparado con otros algoritmos en la lista, ELM muestra un rendimiento sólido y consistente en todas estas métricas. Su exactitud y sensibilidad cercanas al 90% indican que ELM tiene una capacidad aceptable para clasificar muestras correctamente y capturar verdaderos positivos de manera efectiva. Por otro lado, KNN parece ser el menos preciso y sensible de los métodos evaluados en esta comparativa.

Sin embargo, cuando se compara con los demás algoritmos para clasificación, como XGBoost con su 99.13% en todas las métricas, es evidente que ELM no alcanza los niveles de rendimiento excepcionales de estos métodos. Random Forest y el Árbol de Decisión también superan a ELM en términos de precisión, sensibilidad y exactitud. Si bien no alcanza las tasas de precisión y sensibilidad más altas observadas en XGBoost, sigue siendo una opción factible para tareas de clasificación en este caso con datos de análisis de ultrasonido en rodamientos.

CAPÍTULO V

5. CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

No se puede trabajar directamente con datos faltantes en el algoritmo ELM debido a su dependencia en operaciones matriciales y funciones de activación que requieren valores numéricos completos. Estas operaciones y funciones no están diseñadas para manejar la ausencia de datos ya que podría generar errores en los cálculos y resultados inconsistentes por lo que se requiere un preprocesamiento mediante el sobre muestreo como en este caso.

La división de datos en conjuntos de entrenamiento y prueba es una práctica esencial en el aprendizaje automático. Esta separación permite evaluar el rendimiento del modelo en datos no vistos, evitando problemas de sobreajuste y proporcionando una estimación confiable de cómo se comportarán en situaciones del mundo real. Además, al dividir los datos se puede comparar diferentes modelos y algoritmos de manera justa utilizando los mismos conjuntos de prueba.

El modelo ELM ha logrado un rendimiento máximo del 94.16% en términos de precisión en el conjunto de entrenamiento y 88.51% en el conjunto de prueba. La precisión en el conjunto de entrenamiento es ligeramente mayor que en el conjunto de prueba. Con esto se puede concluir que es que se alcanzó el límite del rendimiento del modelo con las características y datos proporcionados.

La eficiencia de ELM su etapa "extrema" se debe a que los pesos y sesgos se generan aleatoriamente y se mantienen fijos durante todo el proceso de entrenamiento. Esto diferencia con otros algoritmos de aprendizaje automático, como las redes neuronales convencionales, que requieren una fase iterativa de ajuste de pesos mediante algoritmos de optimización, lo que implica muchos cálculos repetitivos

El tiempo de entrenamiento de aproximadamente 15.31 segundos significa que el modelo ELM tardó ese tiempo en ajustar sus parámetros y pesos internos utilizando los datos de entrenamiento proporcionados. Durante este tiempo el modelo ELM analizó los datos de entrenamiento, calculó las activaciones de las neuronas ocultas y ajustó los pesos de manera iterativa para minimizar el error entre las salidas pronosticadas y las etiquetas reales.

5.2. RECOMENDACIONES

Para aplicar el algoritmo ELM de manera efectiva se recomienda realizar un preprocesamiento completo antes de la etapa de entrenamiento. Es crucial emplear técnicas adecuadas para llenar los valores faltantes, evitando así interrupciones en las operaciones matriciales y en las funciones de activación de ELM. Este enfoque de preprocesamiento garantizará que el algoritmo funcione correctamente y permitirá aprovechar al máximo los datos disponibles para obtener resultados precisos

La elección del porcentaje de división entre entrenamiento y prueba puede depender de varios factores, se recomienda considerar las características específicas del conjunto de datos incluida la cantidad total de datos disponibles, la complejidad del modelo y la naturaleza del problema para tomar una decisión sobre el porcentaje de división que mejor se adapte la situación. Cuanto menos datos se tenga, más importante será maximizar la cantidad de datos utilizados en el entrenamiento.

La función de activación por defecto puede ser diferente según el tipo de problema (clasificación o regresión) y el paquete utilizado. Para problemas de clasificación, a menudo se utiliza la función sigmoide o relu (rectified linear unit). Para problemas de regresión, la función de activación puede ser lineal o tanh (tangente hiperbólica).

Si el conjunto de datos tiene un desequilibrio de clases, es decir, algunas clases tienen muchas más instancias que otras, esto puede afectar la capacidad del modelo para aprender patrones de las clases minoritarias. Hay que considerar el desequilibrio de clases utilizando técnicas de equilibrio de datos, como sobre muestreo o sub muestreo.

El modelo ELM puede funcionar bien para algunos tipos de problemas de clasificación, pero puede no ser la mejor opción para otros como puede ser considerado este caso. En vista de eso se recomienda explorar otros algoritmos de aprendizaje automático o redes neuronales más avanzadas si es necesario.

BIBLIOGRAFÍA

1. **BARANDAS MARÍLIA, FOLGADO DUARTE y FERNANDES LETICIA**. "TSFEL: Time Series Feature Extraction Library." [en línea]. 2020. Recuperado a partir de : https://github.com/fraunhoferportugal/tsfel/tree/master [consultado 17 agosto 2023].

2. **BESA ANTONIO y CARBALLEIRA JAVIER.** *Análisis de fallas de estructuras y elementos mecánicos* [en línea]. 2° Edición. Valencia : Editorial Universitat Politécnica de Valencia . Recuperado a partir de : https://elibro.net/es/ereader/espoch/57456?page=99. [consultado 17 octubre 2022].

3. **BOBADILLA JESÚS.** *Machine Learning y Deep Learning* [en línea]. Ra-Ma 2020. Madrid : RA-MA Editorial. ISBN 978-84-9964-889-7. Recuperado a partir de : https://elibro.net/es/ereader/espoch/222698?page=10 [consultado 13 octubre 2022].

4. **CASTRO FAUSTO.** Entrenamiento Comprimido Basado en Máquinas de Aprendizaje Extremo. *Scielo* [en línea]. Recuperado a partir de : https://www.scielo.cl/pdf/infotec/v30n4/0718-0764-infotec-30-04-00227.pdf [consultado 17 octubre 2022].

5. **CHUYA JORGE.** *Sistema Inteligente de Diagnóstico de Fallas en Máquinas Rotativas usando el Enfoque de Aprendizaje Automático* [en línea]. Monterrey : Instituto Tecnológico y de Estudios Superiores de Monterrey. Recuperado a partir de : http://hdl.handle.net/11285/633052 [consultado 17 octubre 2022].

6. **GALLARÁ I y PONTELLI D.** *Mantenimiento Industrial* [en línea]. 1° Edición. Córdova : Universitas - Editorial. ISBN 978-987-572-358-0. Recuperado a partir de : https://elibro.net/es/ereader/espoch/172527 [consultado 13 octubre 2022].

7. **GARCÍA FÉLIX.** Clasificación de fallas en rodamientos utilizando aprendizaje de máquinas. *Dominio de las Ciencias*. DOI http://dx.doi.org/10.23857/dc.v7i4.2082.

8. **GAVIÑA JAVIER.** *Estimación de Pose y sus Aplicaciones*. . Valencia : Universidad de Valencia.

9. **GRACIÁ LUIS.** *DESARROLLO DE UN SISTEMA DE DIAGNÓSTICO DE RODAMIENTOS MEDIANTE ANÁLISIS DE RUIDO* [en línea]. Valencia : UNIVERSITAT POLITÉCNICA DE VALÉNCIA. Recuperado a partir de : https://riunet.upv.es/bitstream/handle/10251/166355/Grac%c3%a1%20-%20DESARROLLO%20DE%20UN%20SISTEMA%20DE%20DIAGN%c3%93STICO%20DE%20RODAMIENTOS%20MEDIANTE%20AN%c3%81LISIS%20DE%20RUIDO.pdf?sequence=2&isAllowed=y [consultado 19 octubre 2022].

10. **GUANGAXI PILAR.** *OBTENCIÓN DE UNA BASE DE DATOS DE VIBRACIÓN, TEMPERATURA, ENERGÍA Y ULTRASONIDO DE LA EVOLUCIÓN DE LOS MODOS DE FALLOS EN RODAMIENTOS*. . Riobamba : Escuela Superior Politécnica de Chimborazo.

11. **LI XUDONG.** *Máquina de aprendizaje extremo (ELM): código de Python* [en línea]. Beijing : 2018. Recuperado a partir de : https://github.com/5663015/elm/tree/master [consultado 17 agosto 2023].

12. **MERCADO FRANKLIN.** *Máquina de aprendizaje extremo para la predicción temprana de heladas en valles mesotermicos de Bolivia* [en línea]. Cochabamba : UNIVERSIDAD CATÓLICA BOLIVIANA “SAN PABLO”. Recuperado a partir de : https://www.researchgate.net/publication/322369589_Maquina_de_aprendizaje_extremo_para_la_prediccion_temprana_de_heladas_en_valles_mesotermicos_de_Bolivia [consultado 23 noviembre 2022].

13. **NOLASCO VALENZUELA JORGE SANTIAGO.** 2018. *Python Aplicaciones prácticas* [en línea]. Ra-Ma 2018. Madrid : RA-RA Editorial. ISBN 978-84-9964-758-6. Recuperado a partir de : https://www.alphaeditorialcloud.com/reader/python-aplicaciones-practicas?location=1 [consultado 12 octubre 2022].

14. **PENSAMIENTOS DE MARIPOSA.** ¿Qué es una matriz de confusión en el aprendizaje automático? [en línea]. 27 enero 2023. Recuperado a partir de : https://geekflare.com/es/confusion-matrix-in-machine-learning/ [consultado 20 agosto 2023].

15. **RANDLES ET AL.** Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study. *IEEE Xplorer*. p. 1.

16. **SÁNCHEZ ANA.** *TÉCNICAS DE MANTENIMIENTO PREDICTIVO. METODOLOGIA DE APLICACIÓN EN* **LAS** *ORGANIZACIONES* [en línea]. Bogotá : Universidad Católica de Colombia. Recuperado a partir de : https://repository.ucatolica.edu.co/server/api/core/bitstreams/7b9c12fb-9ffc-4dad-b1c0-8e9e60dc90b4/content [consultado 2 enero 2023].

17. **SOTAQUIRÁ MIGUEL.** La Matriz de Confusión. [en línea]. 9 septiembre 2022. Recuperado a partir de : https://www.codificandobits.com/blog/matriz-de-confusion/ [consultado 20 agosto 2023].

18. **VILEMA** PABLO**.** Aprendizaje de máquina para mantenimiento predictivo: un problema de clasificación binaria. *Conciencia Digital* [en línea]. Recuperado a partir de : https://cienciadigital.org/revistacienciadigital2/index.php/ConcienciaDigital/article/view/2150/5270 [consultado 13 octubre 2022].

19. **WANG, Jian et al**. A review on extreme learning machine. *Multimedia Tools and Applications*. Vol. 81, número 29, pp. 41611–41660. DOI 10.1007/s11042-021-11007-7.

20. **ZAFRA JULIÁN.** Ingeniería *del Sonido* [en línea]. Ra-Ma 2018. Madrid. Recuperado a partir de : https://elibro.net/es/ereader/espoch/106578 [consultado 3 enero 2023].

ANEXOS

Librerías para el pretratamiento

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import KNNImputer
```

[2]:
```
import functions_IF as fns                # Módulo propio
```

1. Lectura de base de datos original

1.1 Carga de base de datos

Se cargan los datos almacenados en el archivo "roddata.csv"

[3]:
```
df_rod = pd.read_csv('data/roddata.csv')
display(df_rod.head())
display(df_rod.info())
```

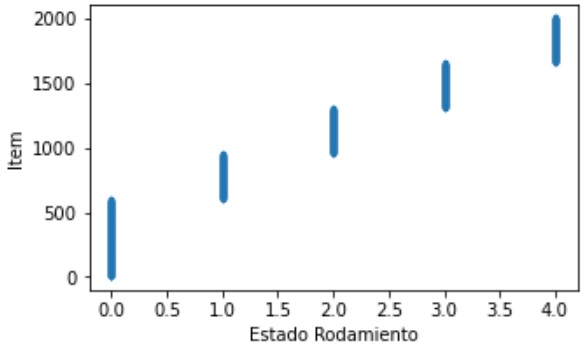
	Item	Frecuencia (Hz)	Vrms (mm/s)	Frecuencia 1 (HZ)	Amplitud 1 (gE)	BPFO (Hz)	Frecuencia 2 (Hz)	Amplitud 2 (gE)	BPFI (Hz)	Frecuenci 3 (Hz
0	1	59.89	8.83	301.01	0.01	300.33	764.02	0.0	764.06	21.8
1	2	59.88	9.48	299.86	0.01	299.65	762.87	0.0	762.35	21.8
2	3	59.87	9.25	299.86	0.01	299.71	762.87	0.0	762.50	21.8
3	4	59.88	8.73	299.86	0.01	299.70	762.87	0.0	762.46	21.8
4	5	59.87	8.75	299.86	0.02	299.72	762.87	0.0	762.51	21.8

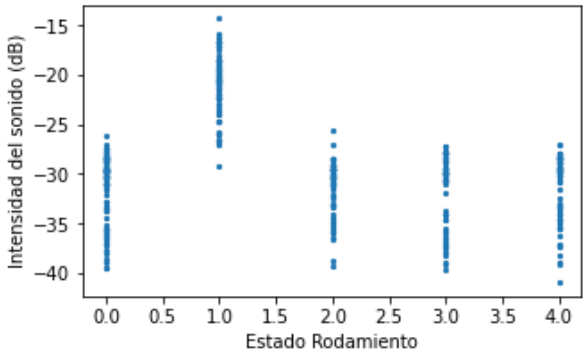
5 rows × 119 columns

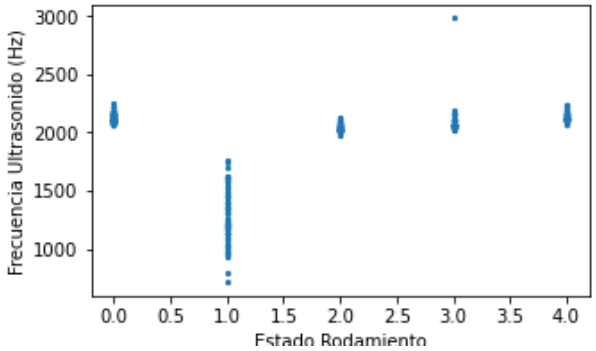
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2007 entries, 0 to 2006
Columns: 119 entries, Item to Sonido
```

[4]:
```
df_rod.plot.scatter(x="Estado Rodamiento", y="Item", figsize=(5,3), marker='.')
df_rod.plot.scatter(x="Estado Rodamiento", y="Intensidad del sonido (dB)", figsize=(5
df_rod.plot.scatter(x="Estado Rodamiento", y="Frecuencia Ultrasonido (Hz)", figsize=(
df_rod.plot.scatter(y="Intensidad del sonido (dB)", x="Frecuencia Ultrasonido (Hz)",
```

[4]:
```
<AxesSubplot:xlabel='Frecuencia Ultrasonido (Hz)', ylabel='Intensidad del sonido (d
B)'>
```







[5]:
```
df_state = df_rod.groupby("Estado Rodamiento").size() # Agrupa los diferentes estados
display(df_state) # mostrar
```

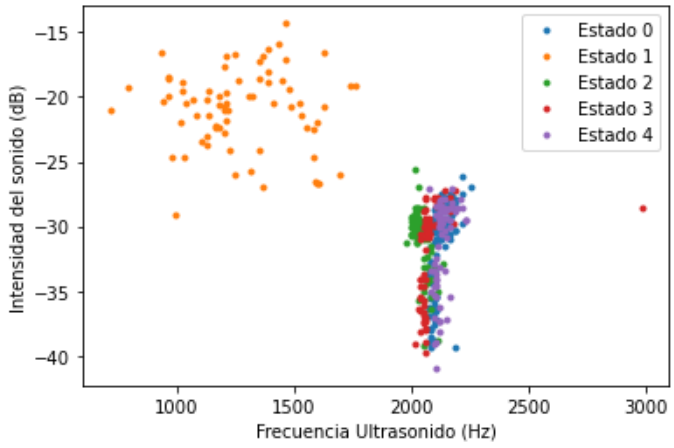
```
Estado Rodamiento
0.0    600
1.0    352
2.0    352
3.0    351
4.0    352
dtype: int64
```

[6]:
```
lst_cols_ext = ['Item','Frecuencia Ultrasonido (Hz)','Intensidad del sonido (dB)' ,

df_st0 = df_rod.query('`Estado Rodamiento` == 0')[lst_cols_ext] # Con "query" se sepa
df_st1 = df_rod.query('`Estado Rodamiento` == 1')[lst_cols_ext] # separa el Estado 1
df_st2 = df_rod.query('`Estado Rodamiento` == 2')[lst_cols_ext] # separa el Estado 2
df_st3 = df_rod.query('`Estado Rodamiento` == 3')[lst_cols_ext] # separa el Estado 3
df_st4 = df_rod.query('`Estado Rodamiento` == 4')[lst_cols_ext] # separa el Estado 4
df_st0
```

[6]:

	Item	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
0	1	NaN	NaN	0.0
1	2	NaN	NaN	0.0
2	3	NaN	NaN	0.0
3	4	NaN	NaN	0.0

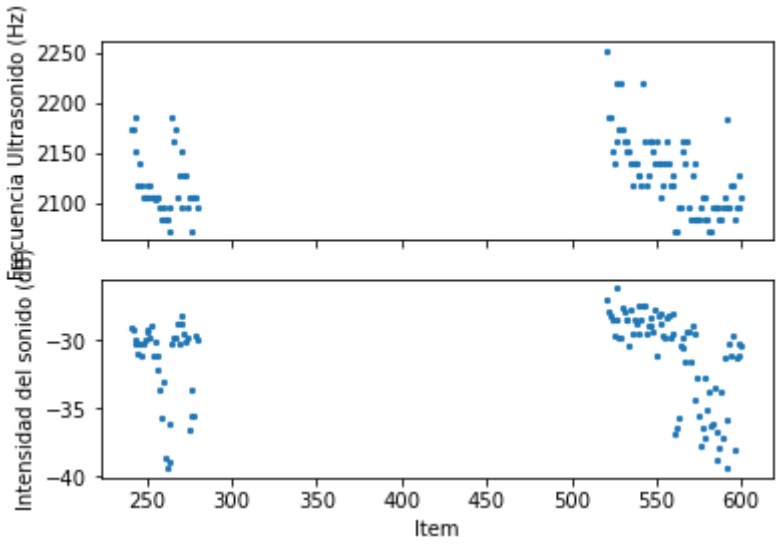
[7]:
```
# Distribución de datos por estado del rodamiento
ls_state = [df_st0, df_st1, df_st2, df_st3, df_st4]
ls_cols =["Frecuencia Ultrasonido (Hz)", "Intensidad del sonido (dB)"]

fns.show_state_data(ls_state, ls_cols)
```

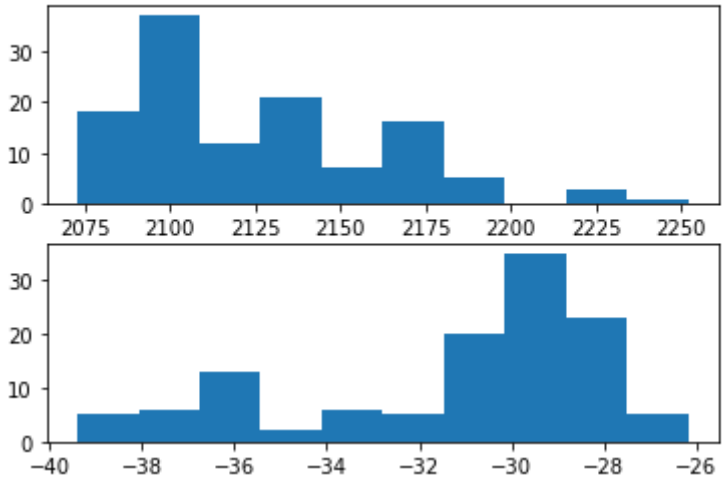


2. Analizando datos vacios

[8]:
```
# Estado 0
lst_cols = ['Item', 'Frecuencia Ultrasonido (Hz)', 'Intensidad del sonido (dB)']
fns.show_states_analysis(df_st0, lst_cols)
```

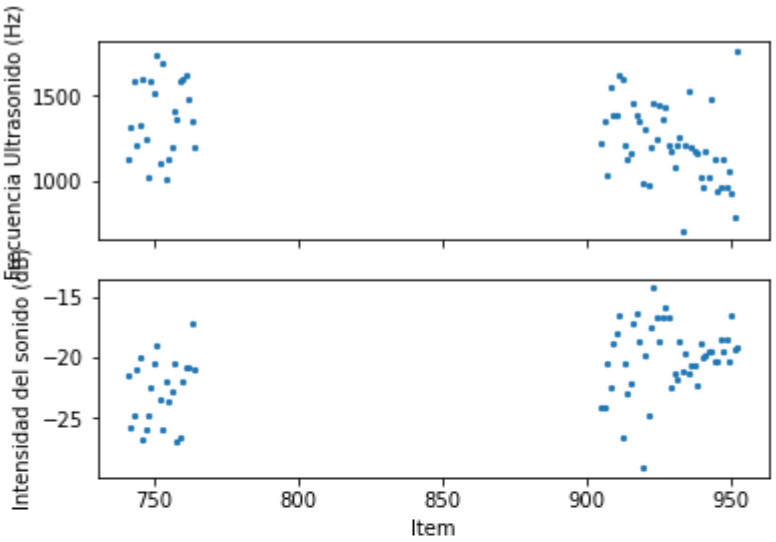
```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    480
Intensidad del sonido (dB)     480
dtype: int64
count     120.000000
mean     2125.083333
std        36.121314
min      2073.000000
25%      2096.000000
50%      2118.000000
75%      2152.000000
max      2252.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    120.000000
mean     -31.271083
std        3.306273
min      -39.400000
25%      -33.457500
50%      -29.950000
75%      -28.850000
max      -26.180000
```

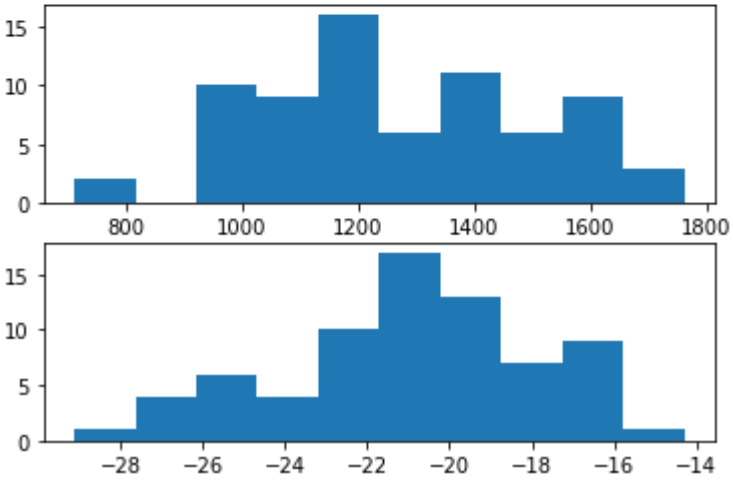




[9]:
```
# Estado 1
fns.show_states_analysis(df_st1, lst_cols) #  muestra descripción de df_st1
```

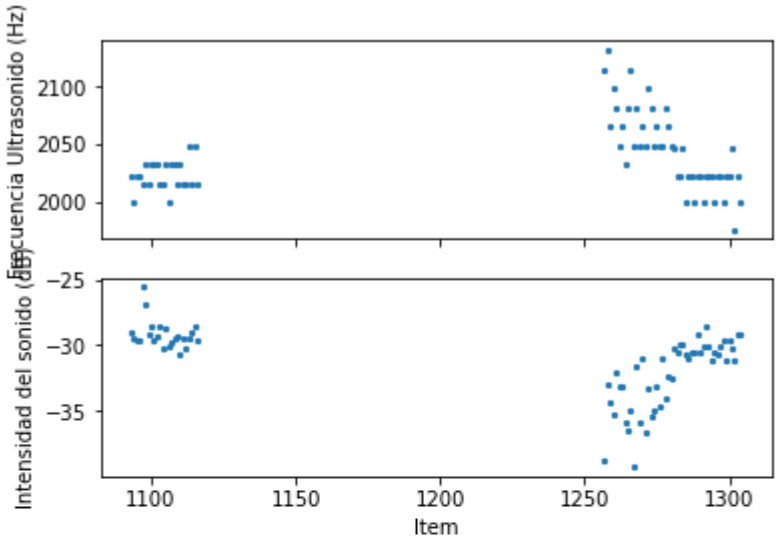
```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      72.000000
mean     1272.593056
std       232.952934
min       712.400000
25%      1128.250000
50%      1220.500000
75%      1450.000000
max      1762.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -20.908333
std       3.047213
min     -29.120000
25%     -22.420000
50%     -20.530000
75%     -18.882500
max     -14.300000
Name: Intensidad del sonido (dB), dtype: float64
```

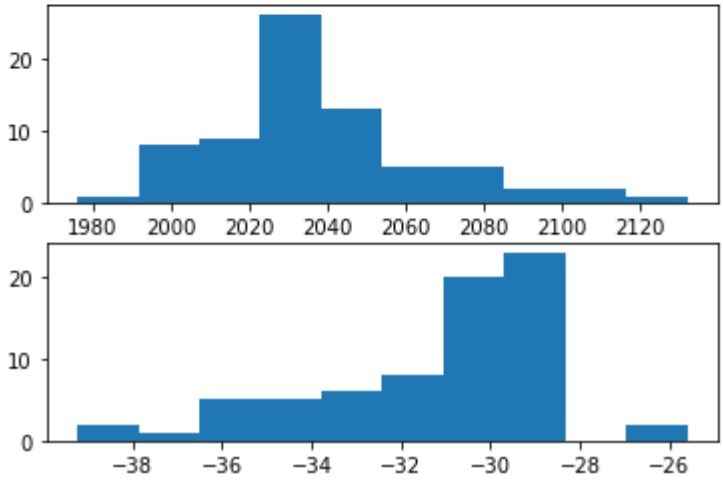




[10]:
```
# Estado 2
fns.show_states_analysis(df_st2, lst_cols) #  muestra descripción de df_st2
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      72.000000
mean     2037.694444
std        30.804535
min      1976.000000
25%      2021.250000
50%      2032.000000
75%      2049.000000
max      2132.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -31.214444
std       2.643797
min     -39.240000
25%     -32.587500
50%     -30.420000
75%     -29.480000
max     -25.600000
Name: Intensidad del sonido (dB), dtype: float64
```





[11]:
```
# Estado 3
fns.show_states_analysis(df_st3, lst_cols) #  muestra descripción de df_st3
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      71.000000
mean     2087.971831
std       113.835090
min      2017.000000
25%      2051.000000
50%      2062.000000
75%      2096.000000
max      2984.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    71.000000
mean    -31.775634
std       3.648708
min     -39.710000
25%     -35.070000
50%     -30.110000
```

[12]:
```
# Estado 4
fns.show_states_analysis(df_st4, lst_cols) #  muestra descripción de df_st4
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    280
Intensidad del sonido (dB)     280
dtype: int64
count      72.000000
mean     2131.111111
std        32.953897
min      2073.000000
25%      2107.000000
50%      2118.000000
75%      2140.000000
max      2230.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -31.323889
std       3.424509
min     -40.970000
25%     -33.810000
50%     -29.800000
```

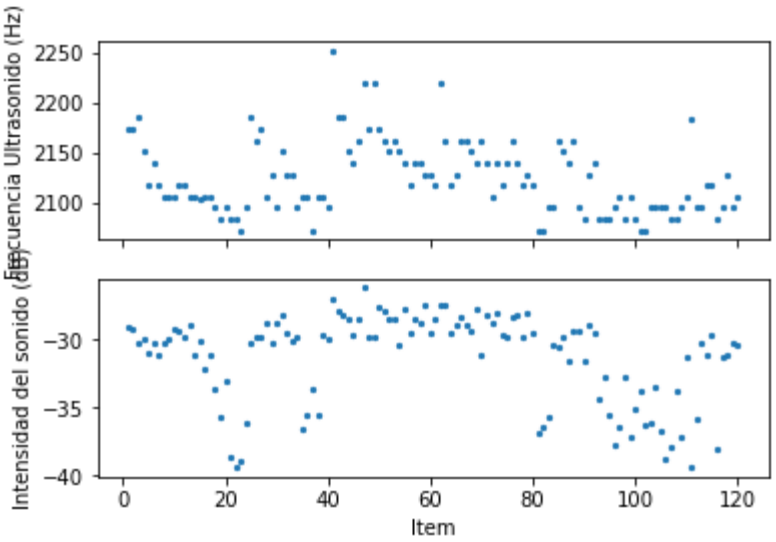
PRIMERA Y ÚNICA ALTERNATIVA

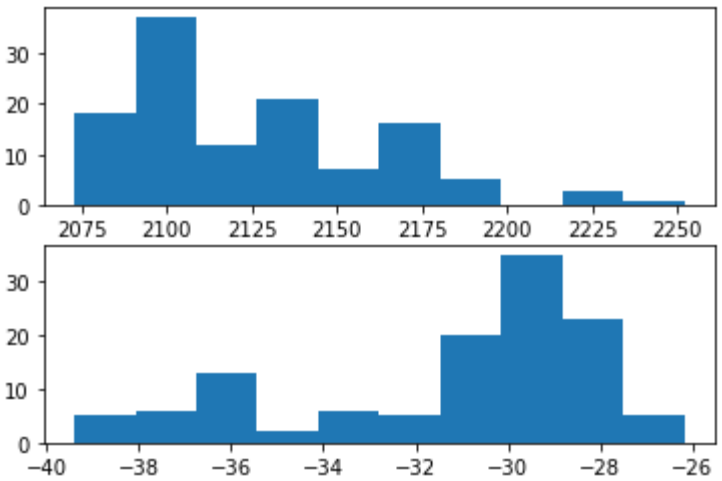
Se elimina los datos vacios

[13]:
```
df_st0_dropped = fns.drop_data_null(df_st0) # df_st0_dropped será el nuevo dataframe
df_st1_dropped = fns.drop_data_null(df_st1) # df_st1_dropped será el nuevo dataframe
df_st2_dropped = fns.drop_data_null(df_st2) # df_st2_dropped será el nuevo dataframe
df_st3_dropped = fns.drop_data_null(df_st3) # df_st3_dropped será el nuevo dataframe
df_st4_dropped = fns.drop_data_null(df_st4) # df_st4_dropped será el nuevo dataframe

df_st0_dropped.describe()
display(df_st0_dropped.head())
```

[14]:
```
fns.show_states_analysis(df_st0_dropped, lst_cols)
```

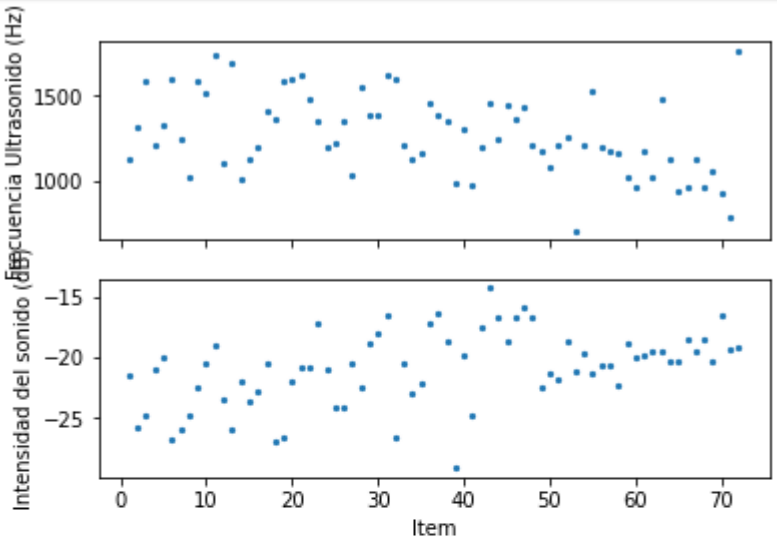
```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    0
Intensidad del sonido (dB)     0
dtype: int64
count     120.000000
mean     2125.083333
std        36.121314
min      2073.000000
25%      2096.000000
50%      2118.000000
75%      2152.000000
max      2252.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    120.000000
mean     -31.271083
std        3.306273
min      -39.400000
25%      -33.457500
50%      -29.950000
75%      -28.850000
max      -26.180000
Name: Intensidad del sonido (dB), dtype: float64
```

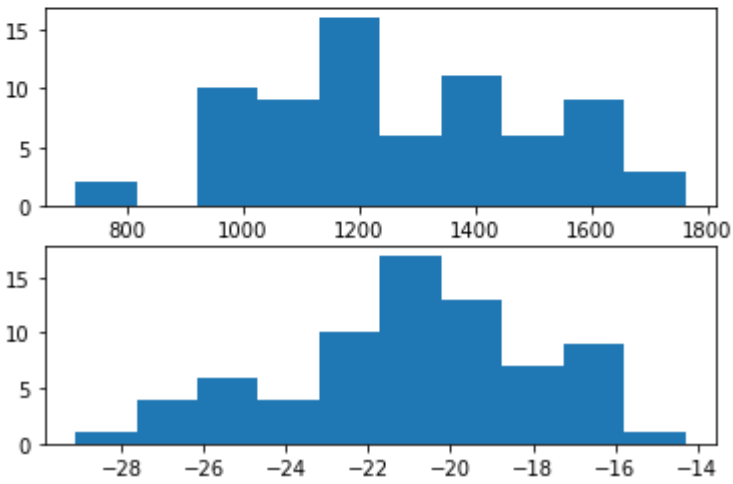




[15]:
```
fns.show_states_analysis(df_st1_dropped, lst_cols)
```

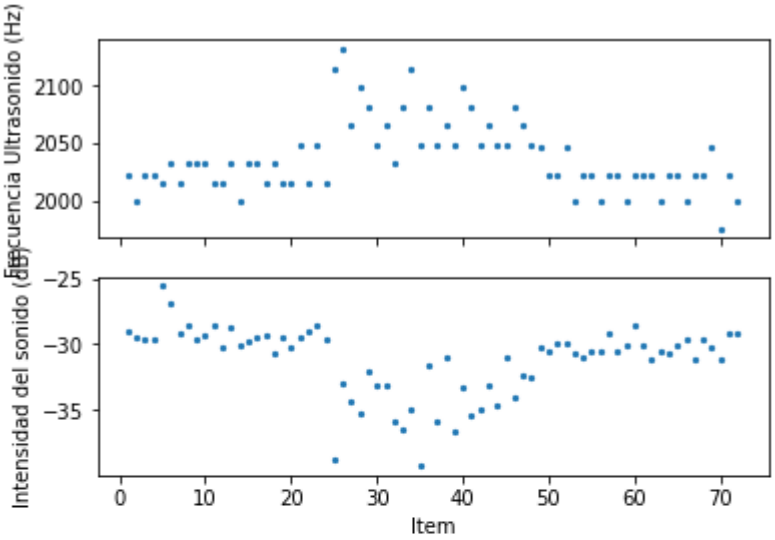
```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    0
Intensidad del sonido (dB)     0
dtype: int64
count      72.000000
mean     1272.593056
std       232.952934
min       712.400000
25%      1128.250000
50%      1220.500000
75%      1450.000000
max      1762.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -20.908333
std       3.047213
min     -29.120000
25%     -22.420000
50%     -20.530000
75%     -18.882500
max     -14.300000
Name: Intensidad del sonido (dB), dtype: float64
```

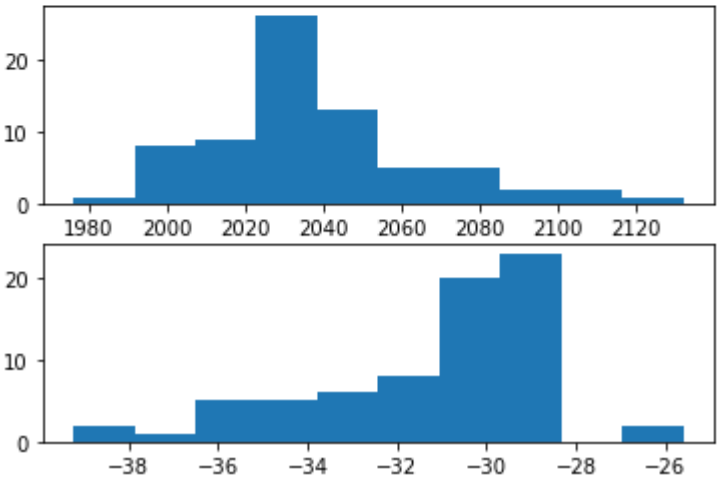




[16]:
```
fns.show_states_analysis(df_st2_dropped, lst_cols)
```

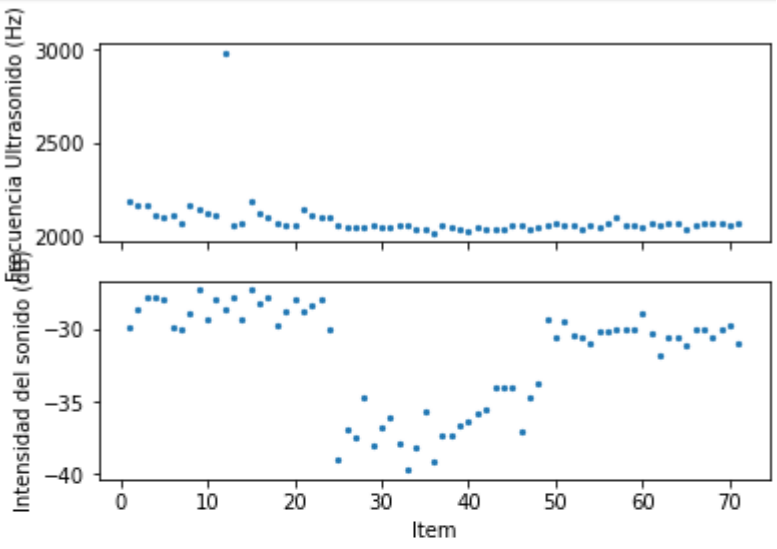
```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    0
Intensidad del sonido (dB)     0
dtype: int64
count      72.000000
mean     2037.694444
std        30.804535
min      1976.000000
25%      2021.250000
50%      2032.000000
75%      2049.000000
max      2132.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -31.214444
std       2.643797
min     -39.240000
25%     -32.587500
50%     -30.420000
75%     -29.480000
max     -25.600000
Name: Intensidad del sonido (dB), dtype: float64
```

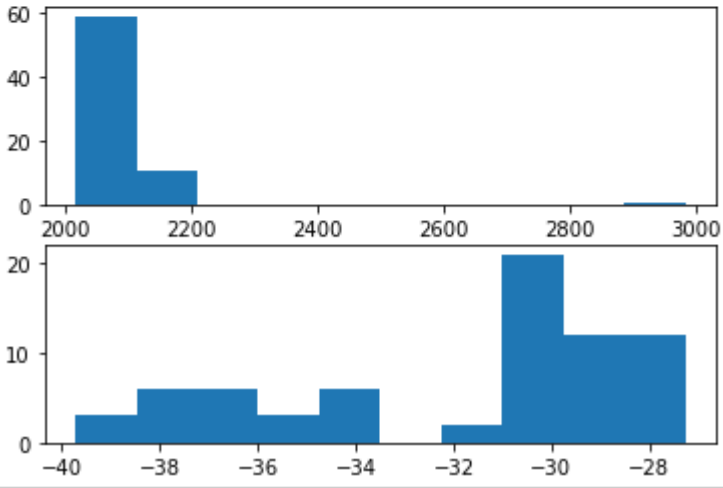




[17]:
```
# Estado 3
fns.show_states_analysis(df_st3_dropped, lst_cols)
```

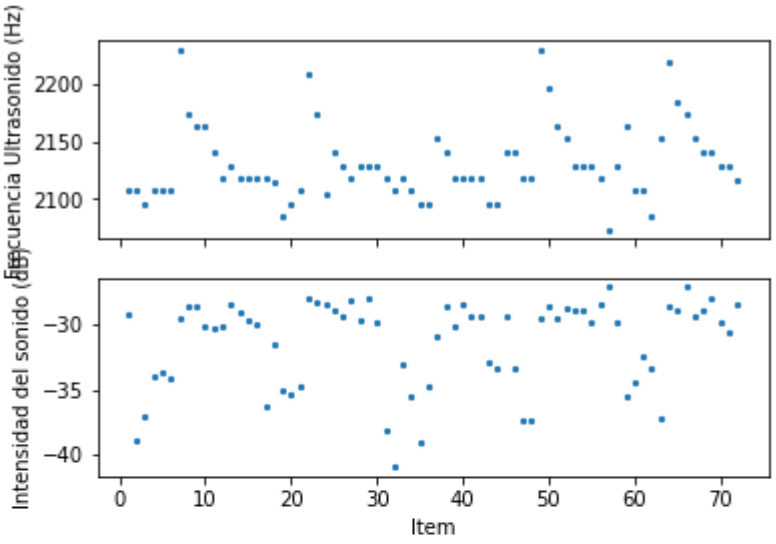
```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    0
Intensidad del sonido (dB)     0
dtype: int64
count      71.000000
mean     2087.971831
std       113.835090
min      2017.000000
25%      2051.000000
50%      2062.000000
75%      2096.000000
max      2984.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    71.000000
mean    -31.775634
std       3.648708
min     -39.710000
25%     -35.070000
50%     -30.110000
75%     -29.085000
max     -27.280000
Name: Intensidad del sonido (dB), dtype: float64
```

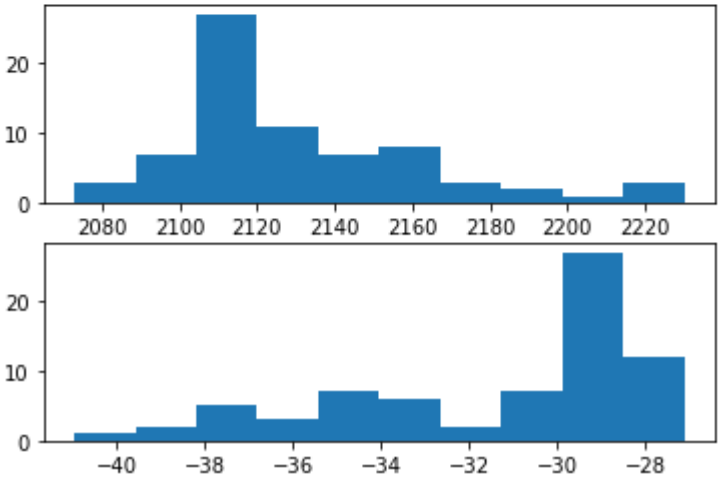




[18]:
```
# Estado 4
fns.show_states_analysis(df_st4_dropped, lst_cols)
```

```
Número de datos vacíos:
Frecuencia Ultrasonido (Hz)    0
Intensidad del sonido (dB)     0
dtype: int64
count      72.000000
mean     2131.111111
std        32.953897
min      2073.000000
25%      2107.000000
50%      2118.000000
75%      2140.000000
max      2230.000000
Name: Frecuencia Ultrasonido (Hz), dtype: float64
count    72.000000
mean    -31.323889
std       3.424509
min     -40.970000
25%     -33.810000
50%     -29.800000
75%     -28.810000
max     -27.120000
Name: Intensidad del sonido (dB), dtype: float64
```





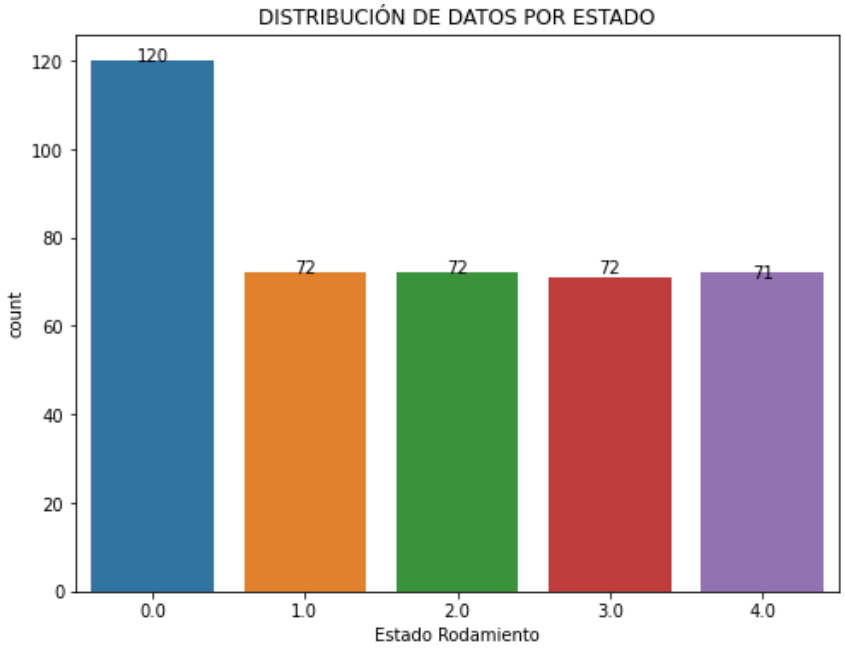
[19]:
```
l_df_dropped = [df_st0_dropped, df_st1_dropped, df_st2_dropped, df_st3_dropped, df_st
df_IF_sonido = pd.concat(l_df_dropped)
df_IF_sonido.to_csv('data/df_IF_sonido.csv') # Se guarda el archivo CSV
display(df_IF_sonido.columns)
```

```
Index(['Item', 'Frecuencia Ultrasonido (Hz)', 'Intensidad del sonido (dB)',
       'Estado Rodamiento'],
      dtype='object')
```

Análisis general del Estado de Rodamiento

Intensidad y Frecuecia

[20]:
```
plt.figure(figsize = (8,6)) # Descripción de la medida del gráfico.
g=sns.countplot(x='Estado Rodamiento' , data=df_IF_sonido) # se define el eje X y el
for i, u in enumerate(df_IF_sonido['Estado Rodamiento'].value_counts().values):
    g.text(i, u, str(u), ha='center')

plt.title('DISTRIBUCIÓN DE DATOS POR ESTADO') # se agrega título
plt.show()
```



[21]:
```
IF_sonido_0 = df_IF_sonido.query('`Estado Rodamiento` == 0') # Con "query" se separa
IF_sonido_1 = df_IF_sonido.query('`Estado Rodamiento` == 1') # separa el Estado 1 y s
IF_sonido_2 = df_IF_sonido.query('`Estado Rodamiento` == 2') # separa el Estado 2 y s
IF_sonido_3 = df_IF_sonido.query('`Estado Rodamiento` == 3') # separa el Estado 3 y s
IF_sonido_4 = df_IF_sonido.query('`Estado Rodamiento` == 4') # separa el Estado 4 y s
IF_sonido_0
```

[21]:

	Item	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
240	1.0	2174.0	-29.01	0.0
241	2.0	2174.0	-29.17	0.0
242	3.0	2185.0	-30.27	0.0
243	4.0	2152.0	-29.95	0.0
244	5.0	2118.0	-30.90	0.0
...	...	...	...	...
595	116.0	2084.0	-38.03	0.0
596	117.0	2096.0	-31.21	0.0
597	118.0	2129.0	-31.06	0.0
598	119.0	2096.0	-30.27	0.0
599	120.0	2107.0	-30.43	0.0

120 rows × 4 columns

[22]:
```
IF_sonido_filtrado = df_IF_sonido.drop('Item', axis=1) # se quita la columna "Item"
IF_sonido_filtrado.head()
```

[22]:

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
240	2174.0	-29.01	0.0
241	2174.0	-29.17	0.0

Sobremuestreo a 25000

[23]:
```
from imblearn.over_sampling import SMOTE

# Definir variables
Xsm = IF_sonido_filtrado.copy()
ysm = Xsm.pop('Estado Rodamiento')

# Crear diccionario de sampling_strategy
sampling_strategy = {0: 25000, 1: 25000, 2: 25000, 3: 25000, 4: 25000}

# Crear instancia de SMOTE con el diccionario de sampling_strategy
sm = SMOTE(sampling_strategy=sampling_strategy)

# Realizar el sobremuestreo
X_res, y_res = sm.fit_resample(Xsm, ysm)
```

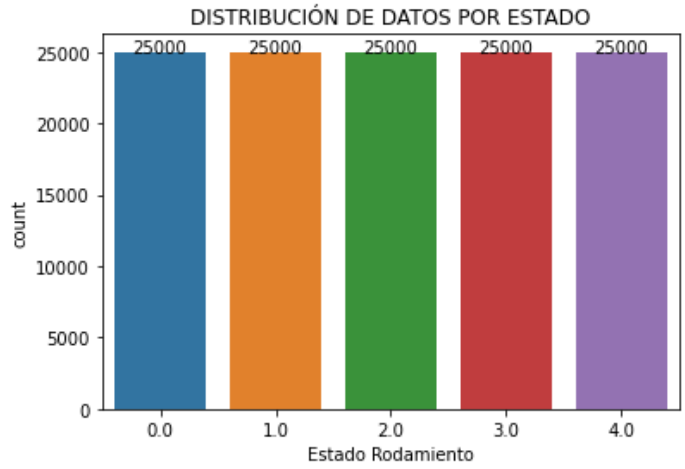
[24]:
```
IF_sonido_25k= pd.concat([X_res,y_res], axis=1)
IF_sonido_25k
```

[24]:

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
0	2174.000000	-29.010000	0.0
1	2174.000000	-29.170000	0.0
2	2185.000000	-30.270000	0.0
3	2152.000000	-29.950000	0.0
4	2118.000000	-30.900000	0.0
...	...	...	...
124995	2107.000000	-33.085594	4.0
124996	2117.058937	-28.422348	4.0
124997	2118.000000	-28.085179	4.0
124998	2163.000000	-33.887930	4.0
124999	2140.000000	-29.815319	4.0

125000 rows × 3 columns

[25]:
```
# plt.figure(figsize = (8,6)) # Descripción de la medda del gráfico.
g=sns.countplot(x='Estado Rodamiento' , data=IF_sonido_25k) # se define el eje X y el
for i, u in enumerate(IF_sonido_25k['Estado Rodamiento'].value_counts().values):
    g.text(i, u, str(u), ha='center')

plt.title('DISTRIBUCIÓN DE DATOS POR ESTADO') # se agrega título
plt.show()
```



[26]:
```
# guardamos el dataframe sobremuestreado directo a 25k
IF_sonido_25k.to_csv('data/IF_sonido_25k.csv')
```

EXTRACCIÓN DE CARACTERÍSTICAS, DIVISIÓN DE DATOS Y COMPARACIÓN CON DIFERENTES ALGORITMOS

Importación de librerías

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tsfel
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, classification_report
from sklearn.preprocessing import RobustScaler
from sklearn. ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

[2]:
```
df_25k_IF =
pd.read_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/

# df_25k_F_sonido ==> Dataframe con Frecuencia e Intensidad
df_25k_IF
```

División de datos

[3]:
```
X1= df_25k_IF.copy() # Definiendo variables
y1= X1.pop('Estado Rodamiento')
```

[4]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1,y1, random_state=0, test_size= 0.20)
```

[5]:
```
y1_train.value_counts()
```

[5]:
```
Estado Rodamiento
1.0    20086
0.0    20024
4.0    19999
2.0    19993
3.0    19898
Name: count, dtype: int64
```

[6]:
```
y1_test.value_counts()
```

[6]:
```
Estado Rodamiento
3.0    5102
2.0    5007
4.0    5001
0.0    4976
1.0    4914
Name: count, dtype: int64
```

Concatenación y guardado IF_25k_train

[7]:
```
df_train= pd.concat([X1_train,y1_train], axis=1)
df_train.to_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/IF_25k_train.csv")
```

Concatenación y guardado IF_25_test

[8]:
```
df_test= pd.concat([X1_test,y1_test], axis=1)
df_test.to_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/IF_25k_test.csv")
```

Extracción de características

[9]:
```
df_train # llamamos al df
```

[9]:

	Unnamed: 0	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
2628	2628	2107.000000	-35.742205	0.0
2076	2076	2096.000000	-30.341270	0.0
104597	104597	2107.000000	-37.445445	4.0
49039	49039	1472.908376	-19.320246	1.0
23535	23535	2107.000000	-29.071341	0.0
...	...	...	...	...
45891	45891	952.892696	-21.763346	1.0
117952	117952	2107.000000	-33.768955	4.0
42613	42613	1117.170116	-23.245542	1.0
43567	43567	1600.000000	-25.392148	1.0
68268	68268	2016.000000	-29.113662	2.0

100000 rows × 4 columns

[10]:
```
df_train= df_train.drop(['Unnamed: 0'], axis= 1) # Se quita la columna genrada por defecto.
df_train
```

[10]:

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
2628	2107.000000	-35.742205	0.0
2076	2096.000000	-30.341270	0.0
104597	2107.000000	-37.445445	4.0
49039	1472.908376	-19.320246	1.0
23535	2107.000000	-29.071341	0.0
...	...	...	...
45891	952.892696	-21.763346	1.0
117952	2107.000000	-33.768955	4.0
42613	1117.170116	-23.245542	1.0
43567	1600.000000	-25.392148	1.0
68268	2016.000000	-29.113662	2.0

100000 rows × 3 columns

Clasifico por falla

[11]:
```
IF_0 = df_train.query('`Estado Rodamiento` == 0') #se separa el df_train y se nombra IF_0
IF_1 = df_train.query('`Estado Rodamiento` == 1') #se separa el df_train y se nombra IF_1
IF_2 = df_train.query('`Estado Rodamiento` == 2') #se separa el df_train y se nombra IF_2
IF_3 = df_train.query('`Estado Rodamiento` == 3') #se separa el df_train y se nombra IF_3
IF_4 = df_train.query('`Estado Rodamiento` == 4') #se separa el df_train y se nombra IF_4

IF_0
```

[11]:

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
2628	2107.000000	-35.742205	0.0
2076	2096.000000	-30.341270	0.0
23535	2107.000000	-29.071341	0.0
1599	2096.000000	-35.185883	0.0
4626	2140.000000	-27.881954	0.0
...	...	...	...
7877	2107.000000	-31.348120	0.0
6921	2140.000000	-27.853084	0.0
18983	2084.000000	-38.503534	0.0
17089	2184.423299	-35.535280	0.0
21243	2118.000000	-29.335647	0.0

20024 rows × 3 columns

Extrae características para clases 0 Train

[12]:
```
# define variables
X0T= IF_0.copy()
y0T= X0T.pop('Estado Rodamiento')
```

[13]:
```
json_path = "C:/Users/PC/anaconda3/Lib/site-packages/tsfel/feature_extraction/features.json"
cfg_file = tsfel.load_json(json_path)
features_train0 = tsfel.time_series_features_extractor(cfg_file, X0T, window_size=12)
features_train0
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\1328964999.py:3: UserWarning: Using default sampling frequency set i
  features_train0 = tsfel.time_series_features_extractor(cfg_file, X0T, window_size=12)
Progress: 100% Complete
```

```
*** Feature extraction finished ***
```

[13]:

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0	0_ECDF Percentile_1	0_ECDF_0	0_ECDF_1	...
0	5.432047e+07	234.220000	5.432047e+07	0.055260	2107.0	2107.0	2107.0	2107.0	0.083333	0.166667	...
1	5.350150e+07	232.405961	5.350150e+07	0.055387	2.0	9.0	2084.0	2129.0	0.083333	0.166667	...
2	5.418374e+07	233.835000	5.418374e+07	0.054507	2.0	9.0	2073.0	2152.0	0.083333	0.166667	...
3	5.365375e+07	232.765000	5.365375e+07	0.054691	2.0	9.0	2084.0	2118.0	0.083333	0.166667	...
4	5.475345e+07	234.840817	5.475345e+07	0.054786	2.0	9.0	2096.0	2163.0	0.083333	0.166667	...
...	...	...	...	...	...	...	...	...	...	...	...
1663	5.398385e+07	233.365000	5.398385e+07	0.054908	2.0	9.0	2084.0	2140.0	0.083333	0.166667	...
1664	5.365634e+07	232.145158	5.365634e+07	0.055734	2.0	9.0	2084.0	2107.0	0.083333	0.166667	...
1665	5.406268e+07	233.382141	5.406268e+07	0.054894	2.0	9.0	2084.0	2140.0	0.083333	0.166667	...
1666	5.413458e+07	233.650000	5.413458e+07	0.055276	2.0	9.0	2084.0	2140.0	0.083333	0.166667	...
1667	5.384539e+07	233.055000	5.384539e+07	0.054965	2.0	9.0	2096.0	2118.0	0.083333	0.166667	...

1668 rows × 280 columns

Extraer características estado 1 Train

[14]:
```
# define variables
X1T= IF_1.copy()
y1T= X1T.pop('Estado Rodamiento')
```

[15]:
```
features_train1 = tsfel.time_series_features_extractor(cfg_file, X1T, window_size=12)
features_train1
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\2750706999.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_train1 = tsfel.time_series_features_extractor(cfg_file, X1T, window_size=12)
Progress: 100% Complete
```

Extraer características estado 2 Train

[16]:
```
#define variables
X2T= IF_2.copy()
y2T= X2T.pop('Estado Rodamiento')
```

[17]:
```
features_train2 = tsfel.time_series_features_extractor(cfg_file, X2T, window_size=12)
features_train2
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\2098666172.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_train2 = tsfel.time_series_features_extractor(cfg_file, X2T, window_size=12)
Progress: 100% Complete
```

Extraer características estado 3 Train

[18]:
```
#define variables
X3T= IF_3.copy()
y3T= X3T.pop('Estado Rodamiento')
```

[19]:
```
features_train3 = tsfel.time_series_features_extractor(cfg_file, X3T, window_size=12)
features_train3
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\3410105915.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_train3 = tsfel.time_series_features_extractor(cfg_file, X3T, window_size=12)
Progress: 100% Complete
```

Extraer características estado 4 Train

[20]:
```
#define variables
X4T= IF_4.copy()
y4T= X4T.pop('Estado Rodamiento')
```

[21]:
```
features_train4 = tsfel.time_series_features_extractor(cfg_file, X4T, window_size=12)
features_train4
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\3406221801.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_train4 = tsfel.time_series_features_extractor(cfg_file, X4T, window_size=12)
Progress: 100% Complete
```

Añadimos Estado Rodamiento

[23]:
```
new_features_train0= features_train0.assign(Estado_Rodamiento = 0)
```

[24]:
```
new_features_train1= features_train1.assign(Estado_Rodamiento = 1)
```

[25]:
```
new_features_train2= features_train2.assign(Estado_Rodamiento = 2)
```

[26]:
```
new_features_train3= features_train3.assign(Estado_Rodamiento = 3)
```

[27]:
```
new_features_train4= features_train4.assign(Estado_Rodamiento = 4)
```

concateno los features 5 clases

[28]:
```
features_train_fin = pd.concat([new_features_train0,new_features_train1,new_features_train2,new_features_train3,new_features_train4], axis=0)
features_train_fin.to_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/features_train_fin.csv")
```

[29]:
```
features_train_fin
```

[29]:

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0	0_ECDF Percentile_1	0_ECDF_0	0_ECDF_1	...	1_Wavelet variance_1	1_Wavelet variance_2	1_Wavelet variance_3
0	5.432047e+07	234.220000	5.432047e+07	0.055260	2107.0	2107.0	2107.0	2107.000000	0.083333	0.166667	...	75.245152	57.804074	75.364156
1	5.350150e+07	232.405961	5.350150e+07	0.055387	2.0	9.0	2084.0	2129.000000	0.083333	0.166667	...	132.991155	73.388156	67.043697
2	5.418374e+07	233.835000	5.418374e+07	0.054507	2.0	9.0	2073.0	2152.000000	0.083333	0.166667	...	127.926651	88.734359	94.583531
3	5.365375e+07	232.765000	5.365375e+07	0.054691	2.0	9.0	2084.0	2118.000000	0.083333	0.166667	...	43.212363	55.435750	97.938106
4	5.475345e+07	234.840817	5.475345e+07	0.054786	2.0	9.0	2096.0	2163.000000	0.083333	0.166667	...	115.391923	74.470189	82.699122
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1661	5.429829e+07	233.903939	5.429829e+07	0.054873	2.0	9.0	2096.0	2140.000000	0.083333	0.166667	...	34.781155	82.417212	145.437145
1662	5.533019e+07	236.194128	5.533019e+07	0.055405	2.0	9.0	2107.0	2169.159547	0.083333	0.166667	...	35.308898	81.654300	135.916770
1663	5.444949e+07	234.457301	5.444949e+07	0.054692	2.0	9.0	2096.0	2140.000000	0.083333	0.166667	...	76.749739	72.657564	102.552468
1664	5.384998e+07	233.090116	5.384998e+07	0.055130	2.0	9.0	2107.0	2118.000000	0.083333	0.166667	...	127.220983	79.517894	85.231709
1665	5.431713e+07	233.881313	5.431713e+07	0.055292	2.0	9.0	2096.0	2140.000000	0.083333	0.166667	...	156.156471	84.983602	71.959733

8331 rows × 281 columns

Ingeniería de caraceristicas TEST

[30]:
```
df_test
```

[30]:

	Unnamed: 0	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
94619	94619	2096.000000	-28.746985	3.0
365	365	2118.000000	-38.140000	4.0
4414	4414	2184.791655	-30.549296	0.0
17638	17638	2096.000000	-37.812781	0.0
109801	109801	2188.928359	-28.739292	4.0
...	...	...	...	...
117976	117976	2187.144181	-28.789573	4.0
7818	7818	2096.000000	-33.361539	0.0
103614	103614	2118.000000	-29.320000	4.0
74925	74925	2023.000000	-30.957043	2.0
20884	20884	2084.000000	-31.749382	0.0

25000 rows × 4 columns

[31]:
```
df_test= df_test.drop(['Unnamed: 0'], axis= 1) # Se quita la columna generada por defecto.
df_test
```

[31]:

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
94619	2096.000000	-28.746985	3.0
365	2118.000000	-38.140000	4.0
4414	2184.791655	-30.549296	0.0
17638	2096.000000	-37.812781	0.0
109801	2188.928359	-28.739292	4.0
...	...	...	...
117976	2187.144181	-28.789573	4.0
7818	2096.000000	-33.361539	0.0
103614	2118.000000	-29.320000	4.0
74925	2023.000000	-30.957043	2.0
20884	2084.000000	-31.749382	0.0

25000 rows × 3 columns

Clasifico por falla

[32]:
```
IF_0t = df_test.query('`Estado Rodamiento` == 0')
IF_1t = df_test.query('`Estado Rodamiento` == 1')
IF_2t = df_test.query('`Estado Rodamiento` == 2')
IF_3t = df_test.query('`Estado Rodamiento` == 3')
IF_4t = df_test.query('`Estado Rodamiento` == 4')
IF_0t
```

[32]:

	Frecuencia Ultrasonido (Hz)	Intensidad del sonido (dB)	Estado Rodamiento
4414	2184.791655	-30.549296	0.0
17638	2096.000000	-37.812781	0.0
23341	2222.024858	-29.539679	0.0
6903	2163.000000	-28.198516	0.0
10950	2107.000000	-30.210754	0.0
...	...	...	...
16201	2163.000000	-27.979304	0.0
4421	2174.000000	-29.366601	0.0
16446	2140.000000	-28.518197	0.0
7818	2096.000000	-33.361539	0.0
20884	2084.000000	-31.749382	0.0

4976 rows × 3 columns

Extrae características para estado 0 Test

[33]:
```
# define variables
X0t= IF_0t.copy()
y0t= X0t.pop('Estado Rodamiento')
```

[34]:
```
json_path = "C:/Users/PC/anaconda3/Lib/site-packages/tsfel/feature_extraction/features.json"
cfg_file = tsfel.load_json(json_path)

features_test0 = tsfel.time_series_features_extractor(cfg_file, X0t, window_size=12)
features_test0
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\986266411.py:4: UserWarning: Using default sampling frequency set in configuration file.
  features_test0 = tsfel.time_series_features_extractor(cfg_file, X0t, window_size=12)
Progress: 100% Complete
```

Progress: 100% Complete

*** Feature extraction finished ***

[34]:

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0	0_ECDF Percentile_1	0_ECDF_0	0_ECDF_1	...
0	5.510039e+07	235.253863	5.510039e+07	0.054711	2.0	9.0	2096.0	2163.00000	0.083333	0.166667	...
1	5.470565e+07	234.755746	5.470565e+07	0.054396	2.0	9.0	2084.0	2166.00839	0.083333	0.166667	...
2	5.424712e+07	233.166425	5.424712e+07	0.055579	2.0	9.0	2084.0	2129.00000	0.083333	0.166667	...
3	5.450894e+07	234.335000	5.450894e+07	0.055177	2.0	9.0	2107.0	2152.00000	0.083333	0.166667	...
4	5.447229e+07	234.548260	5.447229e+07	0.055259	2.0	9.0	2084.0	2174.00000	0.083333	0.166667	...
...	...	...	...	...	...	...	...	...	...	...	...
409	5.404249e+07	232.990000	5.404249e+07	0.054598	2.0	9.0	2096.0	2129.00000	0.083333	0.166667	...
410	5.445413e+07	233.695485	5.445413e+07	0.055141	2.0	9.0	2096.0	2140.00000	0.083333	0.166667	...
411	5.416743e+07	233.637197	5.416743e+07	0.054867	2.0	9.0	2096.0	2140.00000	0.083333	0.166667	...
412	5.422552e+07	233.871591	5.422552e+07	0.055017	2.0	9.0	2096.0	2140.00000	0.083333	0.166667	...
413	5.428667e+07	233.809498	5.428667e+07	0.056171	2.0	9.0	2073.0	2152.00000	0.083333	0.166667	...

414 rows × 280 columns

Extraer características en estado 1 Test

[35]:
```
# define variables
X1t= IF_1t.copy()
y1t= X1t.pop('Estado Rodamiento')
```

[36]:
```
features_test1 = tsfel.time_series_features_extractor(cfg_file, X1t, window_size=12)
features_test1
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\3150920148.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_test1 = tsfel.time_series_features_extractor(cfg_file, X1t, window_size=12)
```

Progress: 100% Complete

Extraer características en estado 2 Test

[37]:
```
# define variables
X2t= IF_2t.copy()
y2t= X2t.pop('Estado Rodamiento')
```

[38]:
```
features_test2 = tsfel.time_series_features_extractor(cfg_file, X2t, window_size=12)
features_test2
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\1602508592.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_test2 = tsfel.time_series_features_extractor(cfg_file, X2t, window_size=12)
```

Progress: 100% Complete

Extraer características en estado 3 Test

[39]:
```
# define variables
X3t= IF_3t.copy()
y3t= X3t.pop('Estado Rodamiento')
```

[40]:
```
features_test3 = tsfel.time_series_features_extractor(cfg_file, X3t, window_size=12)
features_test3
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\666665190.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_test3 = tsfel.time_series_features_extractor(cfg_file, X3t, window_size=12)
```

Progress: 100% Complete

Extraer características en estado 4 Test

[41]:
```
# define variables
X4t= IF_4t.copy()
y4t= X4t.pop('Estado Rodamiento')
```

[42]:
```
features_test4 = tsfel.time_series_features_extractor(cfg_file, X4t, window_size=12)
features_test4
```

```
*** Feature extraction started ***
C:\Users\PC\AppData\Local\Temp\ipykernel_16320\2413033814.py:1: UserWarning: Using default sampling frequency set in configuration file.
  features_test4 = tsfel.time_series_features_extractor(cfg_file, X4t, window_size=12)
Progress: 100% Complete
```

Añade columna Estado_Rodamiento

[43]:
```
new_features_test0= features_test0.assign(Estado_Rodamiento = 0)
```

[44]:
```
new_features_test1= features_test1.assign(Estado_Rodamiento = 1)
```

[45]:
```
new_features_test2= features_test2.assign(Estado_Rodamiento = 2)
```

[46]:
```
new_features_test3= features_test3.assign(Estado_Rodamiento = 3)
```

[47]:
```
new_features_test4= features_test4.assign(Estado_Rodamiento = 4)
```

Concateno los 5 estados de falla

[48]:
```
features_test_fin = pd.concat([new_features_test0,new_features_test1,new_features_test2,new_features_test3,new_features_test4], axis=0)
features_test_fin.to_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/features_test_fin.csv")
```

Concateno los archivos de Entrenamiento y Prueba

[49]:
```
features_fin = pd.concat([features_train_fin, features_test_fin], axis=0, ignore_index=True)
features_fin.to_csv("C:/Users/PC/Documents/DIEGO ESPOCH/PROGRAMACIÓN PYTHON TESIS/data/features_fin.csv")
```

Escalado y Normalización de caracteriticas originales

[50]:
```
X_traino= features_train_fin.copy()
y_traino= X_traino.pop('Estado_Rodamiento')
```

[51]:
```
X_testo= features_test_fin.copy()
y_testo= X_testo.pop('Estado_Rodamiento')
```

[52]:
```
scaler = RobustScaler()
scaler.fit(X_traino)
features_train_normo = scaler.transform(X_traino)
features_test_normo = scaler.transform(X_testo)
```

Clasificador Random Forest

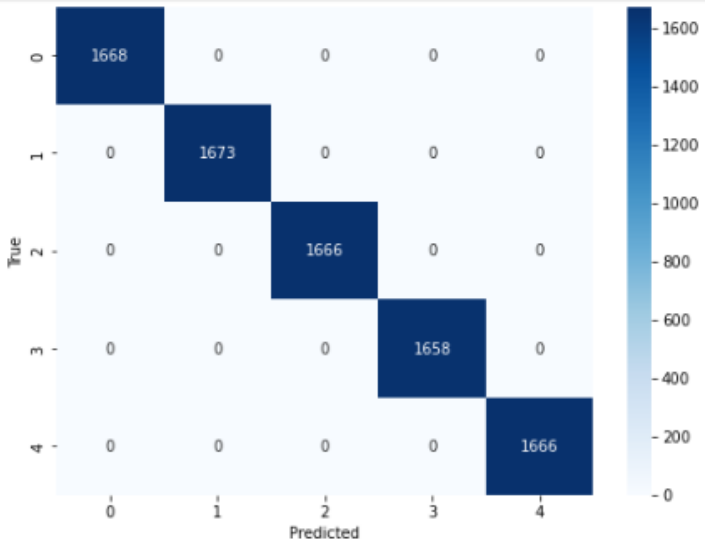
Entrenamiento del modelo

[53]:
```
# Entremaniento del modelo

classifier = RandomForestClassifier()
classifier.fit(features_train_normo, y_traino)


#predicción en el conjunto de prueba

y_predict = classifier.predict(features_train_normo)
accuracy = accuracy_score(y_traino, y_predict) * 100
precision = precision_score(y_traino, y_predict, average='weighted') * 100
recall = recall_score(y_traino, y_predict, average='weighted') * 100
print(classification_report(y_traino, y_predict))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Sensibilidad: " + str(recall) + '%')
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1668
           1       1.00      1.00      1.00      1673
           2       1.00      1.00      1.00      1666
           3       1.00      1.00      1.00      1658
           4       1.00      1.00      1.00      1666

    accuracy                           1.00      8331
   macro avg       1.00      1.00      1.00      8331
weighted avg       1.00      1.00      1.00      8331

Accuracy: 100.0%
Precision: 100.0%
Sensibilidad: 100.0%
```

Matriz de confusión de Entrenamiento RANDOM FOREST

[54]:
```
# Calcula la matriz de confusión
cm = confusion_matrix(y_traino, y_predict)

# Obtiene las etiquetas de las clases
class_labels = np.unique(np.concatenate((y_traino, y_predict)))

# Crea una figura de matplotlib
plt.figure(figsize=(8, 6))

# Crea un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=class_labels, yticklabels=class_labels)

# Configura las etiquetas de los ejes
plt.xlabel('Predicted')
plt.ylabel('True')

# Muestra la matriz de confusión
plt.show()
```



Prueba del modelo

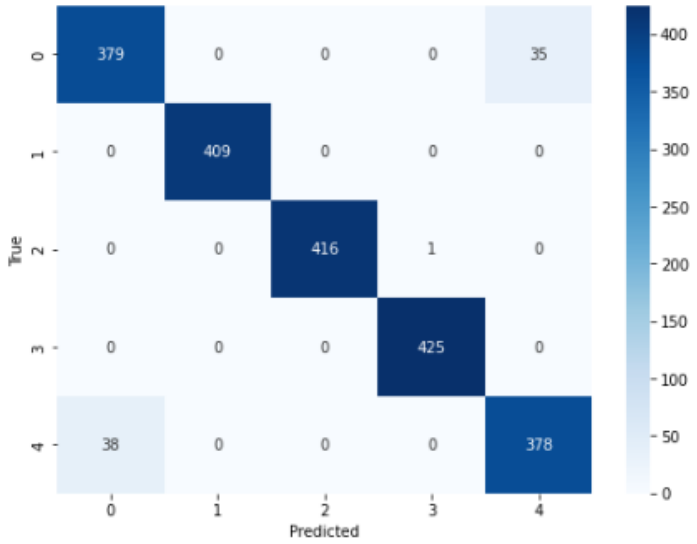
[55]:
```
#predicción en el conjunto de prueba

y_predict = classifier.predict(features_test_normo)
accuracy = accuracy_score(y_testo, y_predict) * 100
precision = precision_score(y_testo, y_predict, average='weighted') * 100
recall = recall_score(y_testo, y_predict, average='weighted') * 100
print(classification_report(y_testo, y_predict))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Sensibilidad: " + str(recall) + '%')
```

```
              precision    recall  f1-score   support

           0       0.91      0.92      0.91       414
           1       1.00      1.00      1.00       409
           2       1.00      1.00      1.00       417
           3       1.00      1.00      1.00       425
           4       0.92      0.91      0.91       416

    accuracy                           0.96      2081
   macro avg       0.96      0.96      0.96      2081
weighted avg       0.96      0.96      0.96      2081

Accuracy: 96.44401729937529%
Precision: 96.44505044443388%
Sensibilidad: 96.44401729937529%
```

[56]:

```
# Calcula la matriz de confusión
cm = confusion_matrix(y_testo, y_predict)

# Obtiene las etiquetas de las clases
class_labels = np.unique(np.concatenate((y_testo, y_predict)))

# Crea una figura de matplotlib
plt.figure(figsize=(8, 6))

# Crea un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=class_labels, yticklabels=class_labels)

# Configura las etiquetas de los ejes
plt.xlabel('Predicted')
plt.ylabel('True')
```

```
# Muestra la matriz de confusión
plt.show()
```



Xgboost

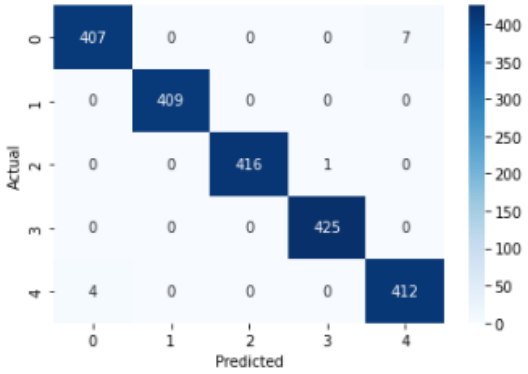
In [56]:
```
XGBC = XGBClassifier()
XGBC.fit(features_train_normo, y_traino)

y_predict1 = classifier.predict(features_train_normo)
accuracy = accuracy_score(y_traino, y_predict1) * 100
precision = precision_score(y_traino, y_predict1, average='weighted') * 100
recall = recall_score(y_traino, y_predict1, average='weighted') * 100
print(classification_report(y_traino, y_predict1))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Sensibilidad: " + str(recall) + '%')
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1668
           1       1.00      1.00      1.00      1673
           2       1.00      1.00      1.00      1666
           3       1.00      1.00      1.00      1658
           4       1.00      1.00      1.00      1666

    accuracy                           1.00      8331
   macro avg       1.00      1.00      1.00      8331
weighted avg       1.00      1.00      1.00      8331

Accuracy: 100.0%
Precision: 100.0%
Sensibilidad: 100.0%
```

In [57]:
```

y_predict1 = XGBC.predict(features_test_normo)
accuracy = accuracy_score(y_testo, y_predict1) * 100
precision = precision_score(y_testo, y_predict1, average='weighted') * 100
recall = recall_score(y_testo, y_predict1, average='weighted') * 100
print(classification_report(y_testo, y_predict1))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Sensibilidad: " + str(recall) + '%')
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       414
           1       1.00      1.00      1.00       409
           2       1.00      1.00      1.00       417
           3       1.00      1.00      1.00       425
           4       0.98      0.99      0.99       416

    accuracy                           0.99      2081
   macro avg       0.99      0.99      0.99      2081
weighted avg       0.99      0.99      0.99      2081

Accuracy: 99.42335415665545%
Precision: 99.42447235316149%
Sensibilidad: 99.42335415665545%
```

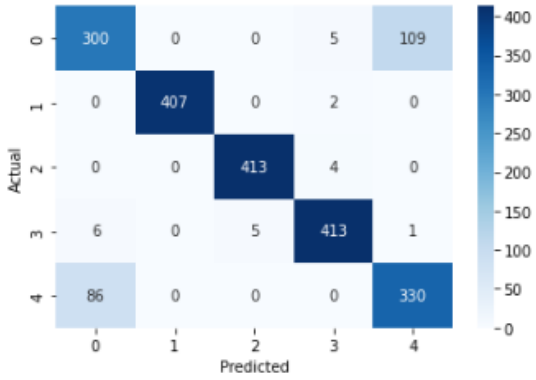
In [58]:
```
# Crear la matriz de confusión
cm = confusion_matrix(y_testo, y_predict1)

# Mostrar la matriz de confusión en forma de heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Naive Bayes

In [59]:
```
# Crear y entrenar el clasificador Naive Bayes
GNB = GaussianNB()
GNB.fit(features_train_normo, y_traino)

# Realizar predicciones en el conjunto de prueba
y_predict2 = GNB.predict(features_test_normo)

# Calcular las métricas
accuracy = accuracy_score(y_testo, y_predict2) * 100
precision = precision_score(y_testo, y_predict2, average='weighted') * 100
recall = recall_score(y_testo, y_predict2, average='weighted') * 100

# Imprimir los resultados
print(classification_report(y_testo, y_predict2))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Recall: " + str(recall) + '%')

# Crear la matriz de confusión
cm = confusion_matrix(y_testo, y_predict2)

# Mostrar la matriz de confusión en forma de heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.77      0.72      0.74       414
           1       1.00      1.00      1.00       409
           2       0.99      0.99      0.99       417
           3       0.97      0.97      0.97       425
           4       0.75      0.79      0.77       416

    accuracy                           0.90      2081
   macro avg       0.90      0.90      0.90      2081
weighted avg       0.90      0.90      0.90      2081

Accuracy: 89.52426717924075%
Precision: 89.56380360500428%
Recall: 89.52426717924075%
```



Arbol de decisión

In [60]:
```
# Crear y entrenar el clasificador Naive Bayes
DTC = DecisionTreeClassifier()
DTC.fit(features_train_normo, y_traino)

# Realizar predicciones en el conjunto de prueba
y_predict3 = DTC.predict(features_test_normo)

# Calcular las métricas
accuracy = accuracy_score(y_testo, y_predict3) * 100
precision = precision_score(y_testo, y_predict3, average='weighted') * 100
recall = recall_score(y_testo, y_predict3, average='weighted') * 100

# Imprimir los resultados
print(classification_report(y_testo, y_predict3))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Recall: " + str(recall) + '%')

# Crear la matriz de confusión
cm = confusion_matrix(y_testo, y_predict3)

# Mostrar la matriz de confusión en forma de heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

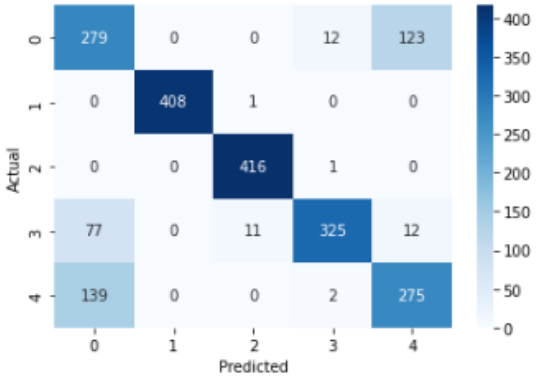
```
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       414
           1       1.00      1.00      1.00       409
           2       1.00      1.00      1.00       417
           3       1.00      1.00      1.00       425
           4       0.91      0.92      0.92       416

    accuracy                           0.97      2081
   macro avg       0.97      0.97      0.97      2081
weighted avg       0.97      0.97      0.97      2081

Accuracy: 96.58817876021143%
Precision: 96.58975382752087%
Recall: 96.58817876021143%
```

KNN

In [61]:
```python
# Crear y entrenar el clasificador Naive Bayes
KNN = KNeighborsClassifier()
KNN.fit(features_train_normo, y_traino)

# Realizar predicciones en el conjunto de prueba
y_predict4 = KNN.predict(features_test_normo)

# Calcular las métricas
accuracy = accuracy_score(y_testo, y_predict4) * 100
precision = precision_score(y_testo, y_predict4, average='weighted') * 100
recall = recall_score(y_testo, y_predict4, average='weighted') * 100

# Imprimir los resultados
print(classification_report(y_testo, y_predict4))
print("Accuracy: " + str(accuracy) + '%')
print("Precision: " + str(precision) + '%')
print("Recall: " + str(recall) + '%')

# Crear la matriz de confusión
cm = confusion_matrix(y_testo, y_predict4)

# Mostrar la matriz de confusión en forma de heatmap
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

	precision	recall	f1-score	support
0	0.56	0.67	0.61	414
1	1.00	1.00	1.00	409
2	0.97	1.00	0.98	417
3	0.96	0.76	0.85	425
4	0.67	0.66	0.67	416
accuracy			0.82	2081
macro avg	0.83	0.82	0.82	2081
weighted avg	0.83	0.82	0.82	2081

```
Accuracy: 81.8356559346468%
Precision: 83.27382296993322%
Recall: 81.8356559346468%
```



CONFIGURACIÓN DEL ELM EN GOOGLE COLAB con la finalizad de utilizar los aceleradores.

```
[ ] from google.colab import drive

    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount('
```

```
[ ] import numpy as np
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import RobustScaler, LabelEncoder
```

```
[ ]
    df_train = pd.read_csv("/content/drive/MyDrive/PROGRAMACIÓN PYTHON TESIS/data/features_train_fin.csv")
    df_test = pd.read_csv("/content/drive/MyDrive/PROGRAMACIÓN PYTHON TESIS/data/features_test_fin.csv")
    df_train
```

	Unnamed: 0	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0
0	0	5.374068e+07	232.920000	5.374068e+07	0.055175	2.0	9.0	2084.000000
1	1	5.425652e+07	233.949155	5.425652e+07	0.054675	2.0	9.0	2084.000000
2	2	5.436803e+07	233.995000	5.436803e+07	0.055085	2.0	9.0	2096.000000
3	3	5.451246e+07	234.220000	5.451246e+07	0.055392	2152.0	2152.0	2152.000000
4	4	5.354777e+07	232.470000	5.354777e+07	0.055300	2.0	9.0	2084.000000

```
[ ] df_trainfil= df_train.drop(['Unnamed: 0'], axis= 1) # Se quita la columna genrada por defecto.

    df_testfil= df_test.drop(['Unnamed: 0'], axis= 1)
```

```
[ ] x_train = df_trainfil.copy() # Definiendo variables
    y_train = x_train.pop('Estado_Rodamiento')

    x_test = df_testfil.copy()
    y_test = x_test.pop('Estado_Rodamiento')
    df_testfil
```

	0_Absolute energy	0_Area under the curve	0_Autocorrelation	0_Centroid	0_ECDF Percentile Count_0	0_ECDF Percentile Count_1	0_ECDF Percentile_0	0_ECDF Percentile_1
0	5.394087e+07	233.366211	5.394087e+07	0.054493	2.0	9.0	2073.000000	2129.000000
1	5.321436e+07	231.670000	5.321436e+07	0.055368	2.0	9.0	2073.000000	2129.000000
2	5.455530e+07	234.437188	5.455530e+07	0.054537	2.0	9.0	2084.000000	2152.000000
3	5.450721e+07	234.270000	5.450721e+07	0.055111	2.0	9.0	2096.000000	2152.000000
4	5.418288e+07	233.610000	5.418288e+07	0.054606	2.0	9.0	2096.000000	2152.000000

```
[ ] y_train.value_counts() # muestra la cantidad de valores para entrenar.
```

```
1    1673
0    1668
2    1666
4    1666
3    1658
Name: Estado_Rodamiento, dtype: int64
```

```
[ ] y_test.value_counts() # muestra la cantidad de valores para la prueba.
```

```
3    425
2    417
4    416
0    414
1    409
Name: Estado_Rodamiento, dtype: int64
```

```
[ ] # Crear un escalador RobustScaler
    stdsc = RobustScaler()

    # Escalar las características del conjunto de entrenamiento
    x_train_scaled = stdsc.fit_transform(x_train)

    # Escalar las características del conjunto de prueba
    x_test_scaled = stdsc.transform(x_test)

    # Convertir las etiquetas a valores numéricos enteros en el rango de 0 a n-1
    label_encoder = LabelEncoder()

    # Codificar las etiquetas del conjunto de entrenamiento
    y_train_encoded = label_encoder.fit_transform(y_train)

    # Codificar las etiquetas del conjunto de prueba
    y_test_encoded = label_encoder.transform(y_test)
```

```
[ ]  from sklearn.model_selection import StratifiedKFold
     # Definir el número de folds para la validación cruzada
     num_folds = 5


     # Crear un objeto StratifiedKFold
     stratified_kfold = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)



     # Seleccionar el fold que deseas analizar (por ejemplo, Fold 1)
     fold_to_analyze = 0

     # Listas para almacenar los resultados
     betas = []
     train_accuracies = []
     running_times = []
     test_predictions = []
     test_accuracies = []
```

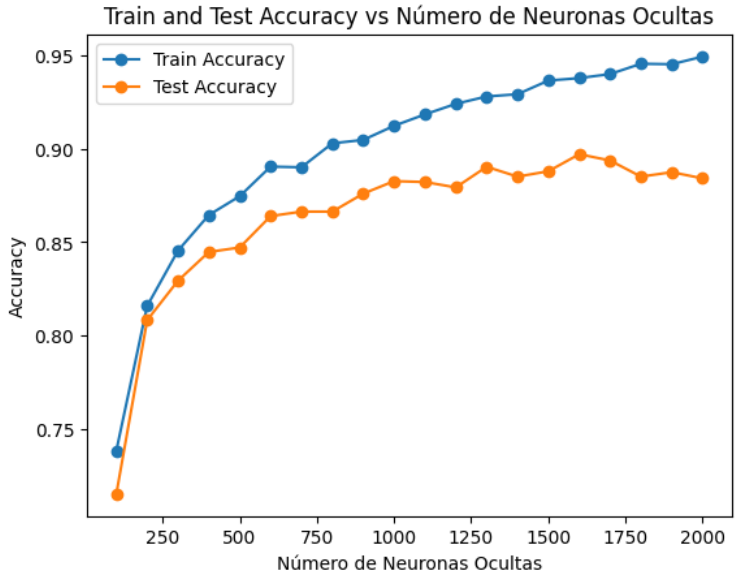
```
[ ]  # Crear el modelo ELM para esta iteración
     model = elm(hidden_units=1700, activation_function='relu',
                 random_type='normal', x=x_train_scaled, y=y_train_encoded,
                 C=0.1, elm_type='clf', one_hot=True)

     # Entrenar el modelo
     beta, train_accuracy, running_time = model.fit('no_re')

     # Evaluar el modelo en el conjunto de prueba
     prediction = model.predict(x_test_scaled)
     test_accuracy = np.mean(prediction == y_test_encoded)

     # Imprimir resultados
     print("Train Accuracy:", train_accuracy)
     print("Test Accuracy:", test_accuracy)
     print("Classifier running time:", running_time)
```

```
Train Accuracy: 0.9416636658264314
Test Accuracy: 0.885151369533878
Classifier running time: 15.318721517000085
```

```
[ ]
     # Definir el rango de neuronas ocultas
     hidden_units_range = list(range(100, 2001, 100))

     # Listas para almacenar resultados
     train_accuracies = []
     test_accuracies = []
     running_times = []

     # Iterar sobre diferentes números de neuronas en la capa oculta
     for hidden_units in hidden_units_range:
         # Crear el modelo ELM para esta iteración
         model = elm(hidden_units=hidden_units, activation_function='relu',
                     random_type='normal', x=x_train_scaled, y=y_train_encoded,
                     C=0.1, elm_type='clf', one_hot=True)

         # Medir el tiempo de ejecución
         start_time = time.time()

         # Entrenar el modelo en esta iteración
         _, train_accuracy, _ = model.fit('no_re')

         # Medir el tiempo de ejecución
         end_time = time.time()
         running_time = end_time - start_time
```

```
[ ]     # Evaluar el modelo en el conjunto de prueba
        test_accuracy = model.score(x_test_scaled, y_test_encoded)

        # Almacenar resultados
        train_accuracies.append(train_accuracy)
        test_accuracies.append(test_accuracy)
        running_times.append(running_time)

        # Imprimir resultados de esta iteración
        print(f"Hidden Units = {hidden_units}, Train Accuracy = {train_accuracy}, Test Accuracy = {test_accuracy},

    # Graficar los resultados
    plt.plot(hidden_units_range, train_accuracies, 'o-', label='Train Accuracy')
    plt.plot(hidden_units_range, test_accuracies, 'o-', label='Test Accuracy')
    plt.title('Train and Test Accuracy vs Número de Neuronas Ocultas')
    plt.xlabel('Número de Neuronas Ocultas')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

```
Hidden Units = 100, Train Accuracy = 0.7382066978754052, Test Accuracy = 0.7150408457472369, Running Time = 0.4770970344543457 seconds
Hidden Units = 200, Train Accuracy = 0.8158684431640859, Test Accuracy = 0.8082652570879385, Running Time = 0.7358372211456299 seconds
Hidden Units = 300, Train Accuracy = 0.8455167446885128, Test Accuracy = 0.8294089380105718, Running Time = 0.8233978748321533 seconds
Hidden Units = 400, Train Accuracy = 0.8646020885848037, Test Accuracy = 0.8447861604997597, Running Time = 0.8761048316955566 seconds
Hidden Units = 500, Train Accuracy = 0.8746849117752971, Test Accuracy = 0.8471888515136954, Running Time = 1.2708735466003418 seconds
Hidden Units = 600, Train Accuracy = 0.8905293482175008, Test Accuracy = 0.8640076886112446, Running Time = 2.820756196975708 seconds
Hidden Units = 700, Train Accuracy = 0.8901692473892699, Test Accuracy = 0.8664103796251802, Running Time = 3.6965243816375732 seconds
Hidden Units = 800, Train Accuracy = 0.9030128435962069, Test Accuracy = 0.8664103796251802, Running Time = 3.7838048934936523 seconds
Hidden Units = 900, Train Accuracy = 0.9048133477373664, Test Accuracy = 0.8760211436809227, Running Time = 3.2301509380340576 seconds
Hidden Units = 1000, Train Accuracy = 0.9123754651302365, Test Accuracy = 0.8827486785199423, Running Time = 3.920074462890625 seconds
Hidden Units = 1100, Train Accuracy = 0.9184971792101788, Test Accuracy = 0.8822681403171552, Running Time = 7.501230955123901 seconds
Hidden Units = 1200, Train Accuracy = 0.9241387588524786, Test Accuracy = 0.8793849111004325, Running Time = 6.509760141372681 seconds
Hidden Units = 1300, Train Accuracy = 0.9280998679630297, Test Accuracy = 0.8904372897645363, Running Time = 7.371062994003296 seconds
Hidden Units = 1400, Train Accuracy = 0.929300204057136, Test Accuracy = 0.885151369533878, Running Time = 10.474178791046143 seconds
Hidden Units = 1500, Train Accuracy = 0.9366222542311847, Test Accuracy = 0.8880345987506006, Running Time = 11.332549095153809 seconds
Hidden Units = 1600, Train Accuracy = 0.9379426239347017, Test Accuracy = 0.897164824603556, Running Time = 11.314677476882935 seconds
Hidden Units = 1700, Train Accuracy = 0.9401032289040931, Test Accuracy = 0.8938010571840461, Running Time = 15.220899820327759 seconds
Hidden Units = 1800, Train Accuracy = 0.9456247749369824, Test Accuracy = 0.885151369533878, Running Time = 16.924935340881348 seconds
Hidden Units = 1900, Train Accuracy = 0.9453847077181611, Test Accuracy = 0.8875540605478136, Running Time = 18.167630672454834 seconds
Hidden Units = 2000, Train Accuracy = 0.9494658504381227, Test Accuracy = 0.8841902931283037, Running Time = 19.81543231010437 seconds
```



ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

CERTIFICADO DE CUMPLIMIENTO DE LA GUÍA PARA NORMALIZACIÓN DE TRABAJOS DE FIN DE GRADO

Fecha de entrega: 09/ 02 / 2024

INFORMACIÓN DEL AUTOR
Nombres – Apellidos: DIEGO HERNAN BARAHONA DEFAZ
INFORMACIÓN INSTITUCIONAL
Facultad: MECÁNICA
Carrera: MANTENIMIENTO INDUSTRIAL
Título a optar: INGENIERO DE MANTENIMIENTO
 **Firma del Director del Trabajo de Integración Curricular**    **Firma del Asesor del Trabajo de Integración Curricular**