UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Comparación de algoritmos de inteligencia artificial para la caracterización de emisiones acústicas en materiales compuestos de fibra de carbono (CFRP)

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

AUTOR/A: Martínez Cases, Cristina

Tutor/a: Giner Maravilla, Eugenio

Cotutor/a: Infante García, Diego

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

TRABAJO FIN DE MÁSTER UNIVERSITARIO EN INGENIERÍA AERONÁUTICA

Comparación de algoritmos de inteligencia artificial para la caracterización de emisiones acústicas en polímeros reforzados con fibra de carbono

Autora
Cristina Martínez Cases

Tutores
Eugenio Giner Maravilla
Diego Infante García

Universitat Politècnica de València
Escuela Técnica Superior de Ingeniería del Diseño

Máster en Ingeniería Aeronáutica

Curso 2022/2023

Valencia, 31 de julio de 2023

Resumen

En la actualidad, el uso de la inteligencia artificial (IA) aplicada a materiales compuestos estructurales se presenta cada vez más en la industria. Una de las aplicaciones es la monitorización de las señales de emisiones acústicas producidas durante el funcionamiento de determinados componentes. Con el análisis mediante IA de dichas señales se consigue obtener información en tiempo real de la integridad de los mismos, pudiendo incluso llegar a identificar modos de fallo.

Es por ello por lo que este estudio se centra en la comparación de diversos algoritmos de inteligencia artificial para la caracterización de emisiones acústicas generadas en un material compuesto de fibra de carbono.

En primer lugar, se realiza una exhaustiva revisión del estado del arte, estableciendo las bases teóricas necesarias; se analizan los diversos tipos y las características de los algoritmos de inteligencia artificial, así como los fundamentos de las emisiones acústicas y el comportamiento de los materiales compuestos reforzados con fibra de carbono.

Posteriormente, utilizando el programa MATLAB, se desarrollan varios tipos de algoritmos de caracterización empleando distintos parámetros. Para evaluar su desempeño, se lleva a cabo una toma de datos experimental de las emisiones acústicas generadas por diferentes impactos sobre una placa plana de fibra de carbono.

Tras ello, estos datos se procesan para extraer las características más relevantes de las señales. Posteriormente, se introducen en los algoritmos diseñados previamente para la creación de los modelos. Finalmente se evalúa el rendimiento de cada algoritmo y se exponen las ventajas e inconvenientes asociadas a su uso para finalizar con la designación de un modelo como óptimo.

Palabras clave

Inteligencia Artificial; Materiales Compuestos; Emisiones Acústicas; Fibra de Carbono; MATLAB; CFRP; Aprendizaje Autónomo;

Abstract

Nowadays, the use of artificial intelligence (AI) applied to structural composite materials is increasingly being applied in industry. One of the applications is the monitoring of acoustic emission signals produced during the operation of components. By analysing these signals using AI, it is possible to obtain real-time information on the integrity of the components and even identify failure modes.

For this reason, this study focuses on the comparison of different artificial intelligence algorithms for the characterisation of acoustic emissions generated in a carbon fibre composite material.

Firstly, an exhaustive review of the state of the art is carried out, establishing the necessary theoretical bases. Various types and characteristics of artificial intelligence algorithms are analysed, as well as the fundamentals of acoustic emissions and the behaviour of carbon fibre reinforced composite materials (CFRP).

Afterwards, using MATLAB software, several types of characterisation algorithms are developed using different parameters. To evaluate their performance, an experimental data collection of acoustic emissions generated by different impacts on a carbon fibre plate is carried out. These data are processed to extract the most relevant characteristics of the signals.

After that, they are introduced into the algorithms previously designed for the creation of the models. Finally, the performance of each algorithm is evaluated and the advantages and disadvantages associated with their use are presented. Ending with the designation of a model as optimal.

Key words

Artificial Intelligence; Composite Materials; Acoustic Emissions; Carbon Fibre; MATLAB; CFRP; Machine Learning;

Agradecimientos

A mis tutores, Eugenio y Diego, y al resto de compañeros del Departamento de Ingeniería Mecánica y de Materiales de la UPV, gracias por darme la oportunidad de trabajar con vosotros estos meses, por tratar de enseñarme todo lo que está en vuestra mano, por mostraros cercanos en todo momento y hacer de este tiempo una gran experiencia.

Gracias por el apoyo y la dedicación mostrada en todo momento.

A mis padres, Encarna y Victoriano, y al resto de la familia, gracias por darme la oportunidad de continuar mis estudios lejos de casa, por buscar lo mejor para mi en todo momento, por el apoyo, por creer en mí y ayudarme a perseguir mis sueños.

Gracias por vuestro sacrificio y dedicación.

A ti Arturo, gracias por estar a mi lado durante estos años, apoyarme en mis decisiones, darme ánimos cuando más lo he necesitado, por ayudarme a ver todo desde otra perspectiva y aguantarme en los momentos más difíciles.

Gracias por tu paciencia y apoyo.

Al resto de amigos y compañeros que realmente han vivido cada momento y han colaborado de mejor o peor manera a poner fin a esta etapa.

Gracias por estar siempre ahí.

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

Abreviaturas y acrónimos

hit	Señal de emisión acústica transitoria
ANN	*Artificial Neural Network*
CFRP	*Carbon Fiber Reinforced Polymer*
DDT	*Duration Discrimination Time*
EA	Emisiones Acústicas
HCA	*Hierarchical Cluster Analysis*
IA	Intelligencia Artificial
KNN	K-Nearest Neighbors
LOOCV	*Leave one out cross validation*
PCA	*Principal Component Analysis*
SVM	*Support Vector Machine*

Símbolos

A	Amplitud
C_L	Velocidad de propagación onda longitudinal
C_T	Velocidad de propagación onda transitoria
E	Módulo de Young
f_s	Frecuencia de adquisición
k	Número de vecinos en un algoritmo KNN

Letras griegas

ν	Coeficiente de poisson
ρ	Densidad

Parte I

MEMORIA

Capítulo 1

Introducción

Los requerimientos exigidos por la industria en cuanto a las propiedades de los materiales son cada vez más exigentes. Se busca un mejor rendimiento de los componentes, tratando de maximizar su resistencia y rigidez al mismo tiempo que se intenta minimizar su peso y coste. Este hecho ha derivado en la mejora de criterios de diseño y procesos de fabricación en materiales comúnmente utilizados, como el acero, pero también en el auge de otros materiales avanzados, como son los materiales compuestos.

La industria del transporte, tanto del automóvil como la aeronáutica, son los grandes beneficiarios de su utilización. En estas áreas la reducción de peso es fundamental, tanto para ahorrar costes como para mejorar prestaciones. El uso de metal se ha visto sustituido por polímeros y plásticos reforzados (materiales compuestos de fibra de carbono, de vidrio, kevlar...) ya que se consigue una reducción significativa de peso al contar con elevados valores de rigidez y resistencia especificas.

Los materiales compuestos son generalmente no isótropos, lo cual hace que sus propiedades sean superiores si la carga es aplicada en la dirección de las fibras, pero inferiores en la dirección perpendicular a las mismas, hecho que puede desencadenar en fallos catastróficos. Como solución a este problema, se plantea la utilización de laminados, en los cuales con la combinación de distintas orientaciones se consigue reducir este efecto.

El uso de materiales compuestos presenta importantes inconvenientes en cuanto a daños preexistentes o defectos internos, ya que la estructura del material puede estar dañada sin apreciarse evidencias en la superficie. Además, no presentan un comportamiento plástico previo a la rotura por lo que no existe una advertencia que anticipe el fallo.

La caracterización de fallos en piezas fabricadas con estos metales se realiza mediante ensayos no destructivos como rayos X, estudio por ultrasonidos o con el uso de microscopios ópticos. Todas estas técnicas, presentan la desventaja de no ser capaces de realizar una monitorización en tiempo real, lo que sí se consigue con el empleo de emisiones acústicas. Una alternativa es el empleo de emisiones acústicas para la monitorización, técnica que presenta buenos resultados en la detección de grietas, incluso

en la fase de iniciación, además de permitir distinguir entre roturas de matriz, fibras, delaminaciones o despegues.

Sin embargo, un problema derivado del uso de emisiones acústicas es su interpretación. Durante la monitorización de una pieza, se recogen una gran cantidad de emisiones, cada una con características diferentes. Procesar estos datos, averiguar su origen, o prever su fallo es una tarea compleja, la cual se facilita con el uso de técnicas de inteligencia artificial.

En los últimos años, los avances en inteligencia artificial han revolucionado muchos campos de estudio e investigación, incluida la caracterización de materiales. Los algoritmos ofrecen la capacidad de procesar grandes volúmenes de datos, extrayendo patrones y características significativas para su análisis.

La inteligencia artificial engloba numerosas funcionalidades como el control robótico, el procesamiento de imágenes o el campo de aprendizaje autónomo. Este último, también conocido como *Machine Learning*, trata de aprender sobre datos y reconocer patrones para crear algoritmos capaces de predecir resultados en base a su aprendizaje. Estas funcionalidades lo hacen especialmente útil para el análisis de emisiones acústicas.

Dentro de *Machine Learning* existen diversos tipos de algoritmos, en función de la forma en la que clasifican los datos. El rendimiento de los mismos, depende de factores como: el tipo de datos, la cantidad disponible de los mismos o el problema a resolver. Por ello, se decide estudiar cuál de estos modelos es el que presenta un mejor rendimiento para la caracterización de emisiones acústicas en materiales compuestos de fibra de carbono.

1.1. Estado del Arte

A continuación, se realiza un análisis de los estudios previos y la investigación existente en relación con el uso de inteligencia artificial para la caracterización de fallos en materiales compuestos. La revisión de la literatura sirve como base para comprender el contexto actual y las contribuciones que se han realizado hasta la fecha.

Por una parte, existen numerosos estudios que demuestran el auge en la utilización de materiales compuestos en diversos ámbitos y en especial en la aeronáutica. El informe de seguridad de la FAA (*Federal Aviation Administration*) de 2014 sobre el uso de materiales compuestos [1], expone la evolución de su uso desde la época de 1980 hasta ahora. En los inicios, sólo se utilizaban los materiales compuestos en partes de la cola de las aeronaves, pero actualmente, se han construido aeronaves como el Boeing 787 Dreamliner, que utiliza materiales compuestos para fuselaje y ala. Este avión llega a acumular hasta un 50 % del peso total de material compuesto, excluyendo los motores. En la (Figura 1.1) se muestra un gráfico sobre la evolución de este porcentaje

a lo largo de los años en las aeronaves militares y comerciales obtenido de *Radical innovation in scaling up: Boeing' s Dreamliner and the challenge of socio-technical transitions* de Rebecca Slayton et al. [2].

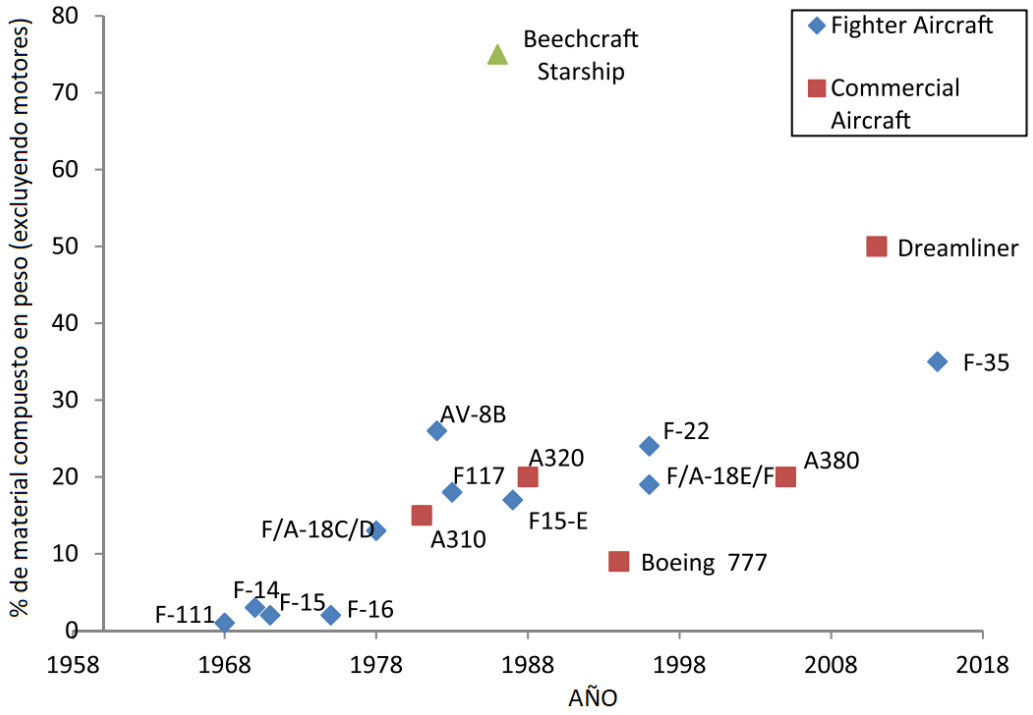

Figura 1.1: Evolución del uso de materiales compuestos en aeronaves [2].

El incremento de su uso ha motivado el estudio técnicas de caracterización de fallos, entre los cuales, se encuentra el análisis de las emisiones acústicas producidas por los materiales y su posterior evaluación con métodos basados en el uso de algoritmos de inteligencia artificial. De la revisión en la literatura, son numerosas las investigaciones en este ámbito. Entre los artículos publicados, cabe destacar:

- *Damage mechanism identification in composites via machine learning and acoustic emission* de C. Nuir et al. (2021) [3]: Asume que existen diferencias en las emisiones acústicas producidas por cada tipo de fallo (ruptura de matriz o fibras, delaminaciones y despegues), las procesa utilizando transformadas de Wavelet y las agrupa con algoritmos no supervisados de IA.

- *A waveform-based clustering and machine learning method for damage mode identification in CFRP laminates* de Jie Wang et al. (2023) [4]: Analiza las señales acústicas de diversos laminados de CFRP con defectos aplicando transformadas de Wavelet y utiliza una red neuronal para separar los tipos de fallos en diferentes *clusters*.

- *Acoustic emission data based deep learning approach for classification and detection of damage-sources in a composite panel* de Shirsendu Sikdar et al. (2022) [5]: Utiliza una red neuronal para analizar la información de señales acústicas a las

cuales se le aplica la transformada de Wavelet. Estas señales provienen de daños provocados de forma artificial sobre un laminado de CFRP. Trata de clasificar las señales según la región de la placa en la que se realizan.

- *Acoustic emission for interlaminar toughness testing of CFRP: Evaluation of the crack growth due to burst analysis* Fabian Lissek et al. (2018) [6]: Analiza los parámetros temporales de las emisiones acústicas producidas por tres laminados de CFRP con orientaciones distintas sometidos a un ensayo DCB (*Double cantilever beam*).

- *An acoustic-emission characterization of the failure modes in polymer-composite materials* de M.Giordano et al. (1998) [7]: Realiza un análisis espectral de frecuencia en probetas de fibra de carbono sometidas a un ensayo de tracción. Trata de aislar fallos de rotura de fibras.

- *Classification of delamination and matrix cracking in carbon fibre composite plates using acoustic emission* (AE) de Dirk Aljets et al. (2009) [8]: Entrena una red neuronal para distinguir fallos por delaminaciones de fallos por ruptura de matriz. Para ello, se llevan a cabo ensayos de tracción y pandeo y se analiza el contenido temporal de las señales acústicas emitidas.

- *Damage analysis of composite CFRP tubes using acoustic emission monitoring and pattern recognition approach* Michal Sofrer et al. (2021) [9]: Realiza ensayos de flexión en tres puntos para recoger emisiones acústicas producidas por tubos de CFRP. Utiliza algorimos no supervisados para clasificar la información obtenida, la cual ha sido previamente procesada utilizando SFTT (Transformada de Fourier de Término Reducido).

- *Damage classification in carbon fibre composites using acoustic emission: A comparison of three techniques* de John P. McCrory et al. (2014) [10]: Trata de localizar y clasificar con el uso de emisiones acústicas el tipo de daño ocurrido en una placa de fibra de carbono. Extrae datos temporales y de los modos de onda para comparar resultados de tres técnicas: una red neuronal, un algoritmo no supervisado y la técnica MAR (Measured Amplitude Ratio). Utiliza técnicas de DIC (Digital image correlation) y ultrasonidos para validar los resultados.

- *Do high frequency acoustic emission events always represent fibre failure in CFRP laminates?* Fatih E. Oz et al. (2017) [11]: Demuestra que la ruptura de fibras en un compuesto CFRP no siempre se corresponde con frecuencias altas. Para ello, analiza las emisiones acústicas producidas por el ensayo de probetas a tracción mediante algoritmos no supervisados. Además, se realiza DIC en la superficie y análisis con microscopio óptico para etiquetar las respuestas.

- *Identifying damage mechanisms of composites by acoustic emission and supervised machine learning* Renato S.M. Almeida et al. (2023) [12]: Se llevan a cabo diversos ensayos para obtener información sobre emisiones acústicas de los diferentes tipos de fallo en un material compuesto de matriz cerámica. Para clasificar

los datos se emplea un algoritmo supervisado KNN (*k nearest neighbors*) que toma como entrada las características temporales de la señal.

- *A Neural Network Framework for Validating Information–Theoretics Parameters in the Applications of Acoustic Emission Technique for Mechanical Characterization of Materials* Claudia Barile et al. (2022) [13]: Utiliza información temporal y transformadas de Wavelet para clasificar datos de ensayos a tracción de probetas de CFRP. Emplea el algoritmo de *k-means* ++ y una red neuronal para clasificar los datos.

- *Unsupervised learning for classification of acoustic emission events from tensile and bending experiments with open-hole carbon fiber composite samples* Hisham A. Sawan et al. (2014) [14]: Se comparan varios métodos no supervisados para la clasificación de emisiones acústicas en probetas de distintos laminados de CFRP sometidas a ensayos de flexión y tracción. Combina el uso de datos en el dominio de la frecuencia y en el del tiempo. Los resultados se etiquetan con el uso de DIC y datos de la fuerza aplicada.

1.2. Objetivos

El objetivo principal de este trabajo es realizar una comparación de distintos algoritmos de inteligencia artificial para la caracterización de emisiones acústicas en polímeros reforzados con fibra de carbono y evaluar la influencia distintos parámetros en su rendimiento. Para lograr el objetivo principal se plantean una serie de objetivos que son:

- Comparar distintos algoritmos de inteligencia artificial para la clasificación de emisiones acústicas.
- Identificar el algoritmo de inteligencia artificial óptimo en términos de precisión y coste computacional.
- Comprensión de los principales fundamentos teóricos de materiales compuestos, emisiones acústicas e inteligencia artificial.
- Familiarización con el funcionamiento del equipo de adquisición de emisiones acústicas.
- Realización de ensayos experimentales para la recogida de las emisiones acústicas necesarias para la tarea de clasificación.
- Estudio de los diversos métodos de procesamiento de señales y elección del más óptimo.
- Familiarización con el entorno de programación de MATLAB para la creación de algoritmos de clasificación.

1.3. Estructura del Trabajo

Para cumplir con los objetivos, se decide estructurar la memoria del trabajo de la siguiente forma:

- En este primer capítulo, se realiza una introducción sobre el uso de materiales compuestos y el estudio de su comportamiento mecánico mediante el análisis de emisiones acústicas con el uso de inteligencia artificial. Se realiza una revisión del estado del arte y se presentan los objetivos del trabajo.
- En el segundo capítulo, se explican los fundamentos teóricos de los campos involucrados en el estudio: materiales compuestos, emisiones acústicas e inteligencia artificial.
- En el tercer capítulo, se exponen las características del ensayo llevado a cabo para la caracterización de emisiones acústicas en CFRP, además de los métodos y parámetros utilizados para la toma de datos.
- En el cuarto capítulo, se hace una revisión de las técnicas de procesamiento de señales empleadas para utilizar los datos de emisiones acústicas como entrada de los modelos de inteligencia artificial.
- En el quinto capítulo, se realiza una descripción de los parámetros elegidos para crear los modelos, así como su arquitectura y los criterios utilizados para la evaluación de su rendimiento.
- En el sexto capitulo, se analizan los resultados obtenidos de la implementación de cada uno de los modelos, detallando cuáles son los que poseen mejores características.
- En el séptimo capítulo, se extraen conclusiones de los resultados obtenidos y se hacen propuestas de trabajos futuros.
- El trabajo concluye con un último apartado de bibliografía

Tras la memoria se incluyen otras tres partes:

- El anexo, donde se incluye una sección con el código empleado en los capítulos 4 y 5 y otra con la relación de trabajo con los objetivos de desarrollo sostenible de la agenda 2030.
- El pliego de condiciones, que recoge las regulaciones, condiciones técnicas y legales enmarcadas en la elaboración del proyecto.
- El presupuesto, que contiene la relación de costes necesarios para el desarrollo del trabajo.

Capítulo 2

Marco Teórico

Para lograr la correcta comprensión de cada uno de los campos involucrados en este estudio se exponen sus bases teóricas. Se define cada uno de los ámbitos, sus características más relevantes y la relación existente entre los mismos.

2.1. Materiales Compuestos Reforzados con Fibra de Carbono

Se considera un material compuesto como una combinación de dos o más materiales a escala macroscópica, con propiedades que lo hacen útil ingenierilmente. Es decir, todo material compuesto queda caracterizado por constar de dos o más materiales distintos separables mecánicamente. Además, se debe contar con la posibilidad de controlar la dispersión de un material en otro a la hora de su fabricación, a fin de poder optimizar las propiedades finales. Los materiales compuestos también poseen un efecto sinérgico, es decir, sus propiedades en conjunto son superiores a las propiedades de los componentes por separado [15].

2.1.1. Clasificación de los materiales compuestos

Compuestos reforzados con fibras

Se componen de fibras orientadas convenientemente embebidas en un aglutinante o matriz. La geometría de las fibras determina la rigidez y resistencia y la matriz permite transmitir los esfuerzos a las fibras además de protegerlas [15].

Compuestos laminados

Están formados por capas o láminas de dos o mas materiales diferentes unidas entre sí.

Entre ellos se encuentran los laminados de compuestos reforzados con fibras, estos constituyen una clase mixta de material compuesto al ser cada lámina que forma el

laminado a su vez un compuesto reforzado con fibra. Esta estructura permite disponer cada capa según la orientación idónea que requiera una aplicación en concreto (Figura 2.1) [15].

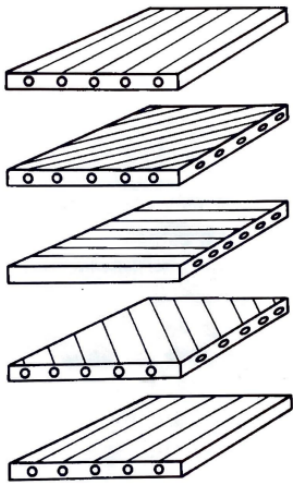

Figura 2.1: Estructura de un laminado. Cada lámina es un compuesto reforzado [15].

Compuestos reforzados con partículas

Se trata de distintos elementos aglutinados sin una disposición concreta. Ejemplos de ello son el hormigón, el combustible sólido utilizado en cohetes o las varillas de reactores nucleares [15].

2.1.2. Ventajas e inconvenientes de su uso

Como principal ventaja sobre materiales tradicionales, se encuentra la capacidad de lograr una resistencia y rigidez específicas muy elevadas. A ello, se deben sumar características únicas en términos de resistencia a la fatiga, impacto, corrosión, etc. Estas pueden lograrse con la optimización del diseño a través de la colocación de las fibras de refuerzo en posiciones y orientaciones adecuadas y la utilización de una correcta fracción de volumen [15].

En cuanto a los inconvenientes, por una parte, aparece un complicado diseño derivado de acoplamientos entre extensión, flexión y/o torsión que no están presentes en materiales isótropos al que se une la presencia de efectos de borde en láminas debidas a la distribución de tensiones en los extremos de fibra. Además, pueden aparecer delaminaciones en los laminados por causa de las tensiones cortantes en las uniones entre láminas. A todo esto se debe sumar que su coste es elevado, al ser el proceso de fabricación mucho más caro; las uniones son complicadas y voluminosas y pueden presentar numerosos defectos específicos derivados del proceso de fabricación como son: huecos entre fibras, curado incompleto, fibras rotas, dañadas o mal alineadas, espesor no uniforme, etc. [15].

2.1.3. Aplicaciones

Múltiples son las aplicaciones de los materiales compuestos, entre ellas se encuentran: la aeronáutica y astronáutica, industrias donde la reducción de peso es un aspecto crítico; automoción y ferrocarril, con la fabricación de múltiples variedades de piezas; industria de generación de energía eólica (aerogeneradores); industria química que aprovecha las ventajas de combinar resistencia a alta presión y a la corrosión; medicina; electrónica y electricidad; material deportivo; electrodomésticos; construcción y edificación, etc. [15].

2.2. Emisiones Acústicas

2.2.1. Definición

Emisión acústica (EA) es el término que se utiliza para describir las ondas de tensión mecánica producidas cuando se libera energía de deformación rápidamente debido a la ocurrencia de cambios micro estructurales en un material [16].

Una onda de tensión propagándose en un medio que tiene una relación tensión-deformación lineal se conoce como onda elástica [17]. Estas ondas se propagan y pueden ser detectadas por los sensores apropiados para su posterior análisis [18].

2.2.2. Aplicaciones

El estudio de las emisiones acústicas producidas por los materiales, es una herramienta potente, que se utiliza en un amplio rango de aplicaciones, como puede ser: análisis de fugas en recipientes a presión y reactores; monitorización de estructuras; inspección de elementos soldados; investigaciones sísmicas y geológicas, etc. [18].

En el caso de CFRP, existen diversos mecanismos de fallo que pueden generar emisiones acústicas cuando el material es sometido a esfuerzos, tales como ruptura de la matriz, ruptura de fibras, ocurrencia de delaminaciones o despegue de fibras (Figura 2.2) [16].

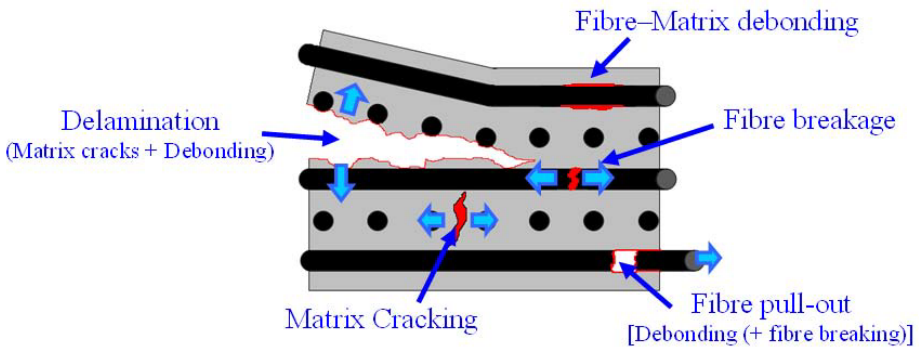

Figura 2.2: Tipos de fallo en laminados de material compuesto [8].

2.2.3. Ventajas e inconvenientes de su uso

El uso de mecanismos de detección de emisiones acústicas sobre otros métodos de detección de fallos o de procesos de deformación, presenta la ventaja de que utiliza información de dicha fractura o deformación en el momento que ocurre, a diferencia de otras técnicas como puede ser el uso de microscopios ópticos. La principal ventaja que presenta es que, con una monitorización continua de una pieza, los eventos acústicos pueden detectarse en el momento en el que ocurren, independientemente de la parte de la que provengan [16].

Por contra, una gran desventaja que se presenta es la dificultad de interpretar la información recibida para relacionarla de forma objetiva con las posibles fuentes de emisión acústica [16].

2.2.4. Tipos de señales acústicas

Se pueden diferenciar dos tipos de señales acústicas: continuas o transitorias (Figura 2.3).

En las señales transitorias, también llamadas golpes, *hits* o *burst*, el principio y final de la señal se puede distinguir claramente del ruido de fondo mientras que en las señales continuas, se observan variaciones de amplitud y frecuencia, pero la señal nunca termina [18].

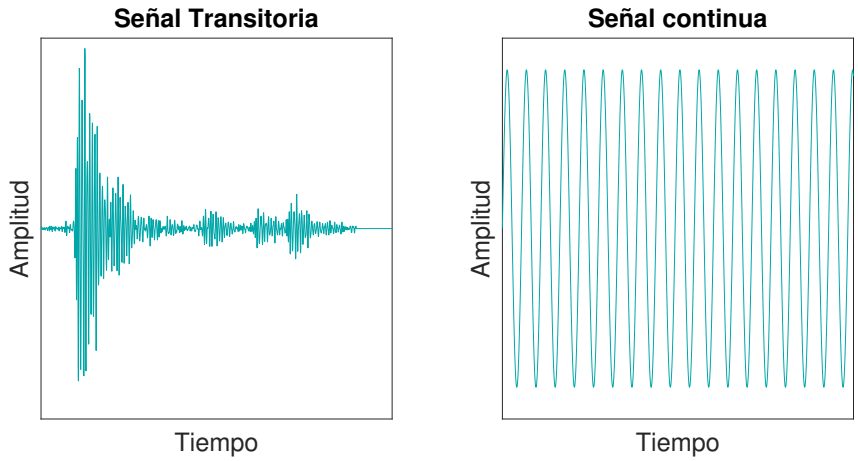

Figura 2.3: Señal transitoria (izquierda) y continua (derecha).

Un ejemplo de señal continua sería el ruido generado por la fricción o el flujo constante de un líquido en una tubería. Por otro lado, una señal transitoria podría ser la fractura de un componente o la iniciación y propagación de grietas [18].

2.2.5. Modos de propagación en sólidos

Durante la propagación de una onda, los movimientos de las partículas se transfieren dinámicamente a las partículas adyacentes. Cada partícula vibra entorno a su posición pero no se desplaza. Este movimiento se propaga con una velocidad característica de onda [19].

El movimiento de las ondas se define en base a la orientación del movimiento de las partículas y la dirección de propagación. Se pueden distinguir varios tipos de onda (Figura 2.4):

- Ondas longitudinales: También llamadas simétricas S_0 o primarias (*P-wave*). Son ondas donde el movimiento de las partículas es en la dirección de propagación de la onda [8]. Puede propagarse en todos los medios, sólido, liquido o gas y es la más rápida de las ondas elásticas [19].

- Ondas transversales: También llamadas antisimétricas A_0 o secundarias (*S-wave*). Son ondas donde el movimiento de las partículas es perpendicular a la dirección de propagación de la onda [8]. Puede propagarse en sólidos pero no en líquidos o gases. La velocidad de propagación es proporcional a la de una onda longitudinal según la Ecuación 2.1 [19].

$$C_T = \sqrt{\frac{1-2\nu}{2(1-\nu)}} C_L \tag{2.1}$$

  Donde ν es el coeficiente de Poisson, C_T la velocidad de propagación de la onda transversal y C_L la velocidad de propagación de la onda longitudinal.

- Ondas Rayleigh: Son ondas superficiales en el que las partículas se mueven formando una elipse. Este movimiento decrece con la profundidad [19].

- Ondas guiadas: También conocidas como *Lamb waves*, se producen cuando la longitud de onda es mayor que el espesor del material. A causa de la reflexión y superposición de ondas esta se puede propagar de forma simétrica o antisimétrica [8].

La existencia de varios modos de propagación se debe a la distribución de fuerzas en el material. Las ondas simétricas están producidas por fuerzas normales y las antisimétricas por esfuerzos de cortadura. Una onda puede cambiar su modo de propagación en la interfaz de dos medios por causa de reflexión o deflexión [8].

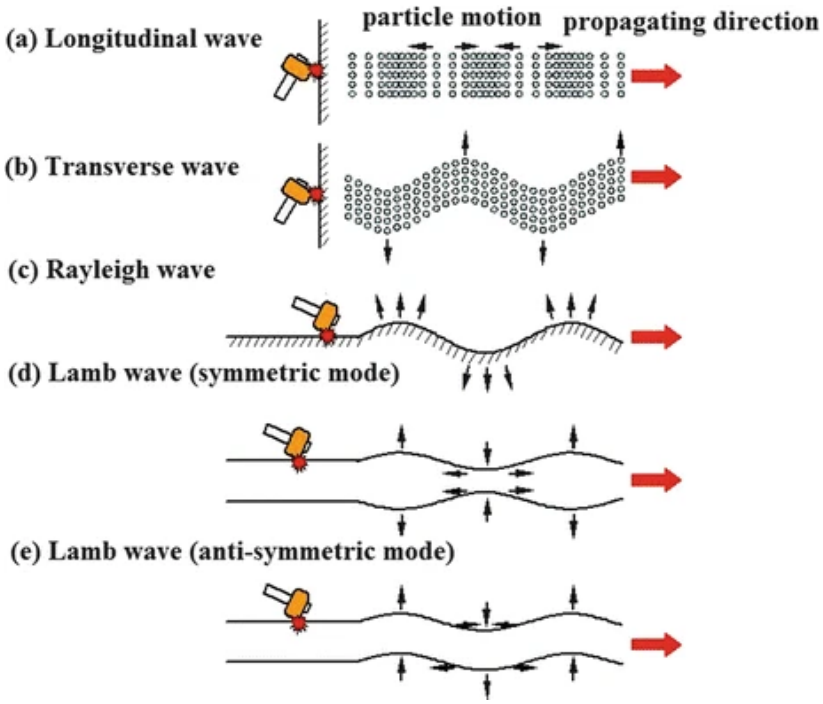

Figura 2.4: Tipos de onda según su movimiento [19].

2.2.6. Velocidad de propagación

La velocidad a la que se propaga una onda elástica en un sólido depende de las propiedades del material y del tipo de onda.

La velocidad de propagación de una onda longitudinal C_L $[m/s]$ se expresa en función del módulo de *Young* E, el coeficiente de Poisson υ y la densidad ρ del material a través del cual se propaga una onda (Ecuación 2.2) [19].

$$C_L = \sqrt{\frac{(1-\upsilon)E}{(1+\upsilon)(1-2\upsilon)\rho}} \tag{2.2}$$

La velocidad de propagación de una onda transversal, de la combinación de las Ecuaciones 2.1 y 2.2 resulta:

$$C_T = \sqrt{\frac{E}{2(1+\upsilon)\rho}} \tag{2.3}$$

La velocidad de una onda *Rayleigh* es aproximadamente un 90 % de la onda transversal y la de una *Lamb Wave* varía según el espesor del material, frecuencia y modo de propagación [19].

2.2.7. Atenuación

Durante la propagación de la onda elástica por el material compuesto se alteran las características de la señal original.

Por una parte, el cambio de un material a otro, como puede ser que la onda pase de la matriz a la fibra, o a la interfase, produce una reflexión de parte de la energía de la onda. Dependiendo del ángulo de entrada y la velocidad de propagación, la dirección de la onda transmitida cambia según la Ley de Snell. El ratio de energía reflejada y desviada de la onda depende de la diferencia de impedancia acústica entre los materiales implicados. Esta impedancia es función de la densidad de los materiales por lo que también esta condicionado por la temperatura y presión del material. Debido a estos efectos, el modo de propagación de una onda puede cambiar al pasar de un material a otro [8].

Por otro lado, todas las ondas mecánicas experimentan atenuación cuando se propagan ya que parte de la energía se convierte en calor debido a la fricción y absorción en el material lo cual genera cambios en la amplitud de la señal original. A esto se debe sumar la difracción de ondas debida a la presencia de poros en el material que también provoca atenuación [8].

2.2.8. Características de un *hit*

Los parámetros que describen una señal acústica transitoria en el dominio del tiempo se definen como: duración, *treshold, peak amplitude, rise time, counts* y energía. Estos se observan en la Figura 2.5 [20].

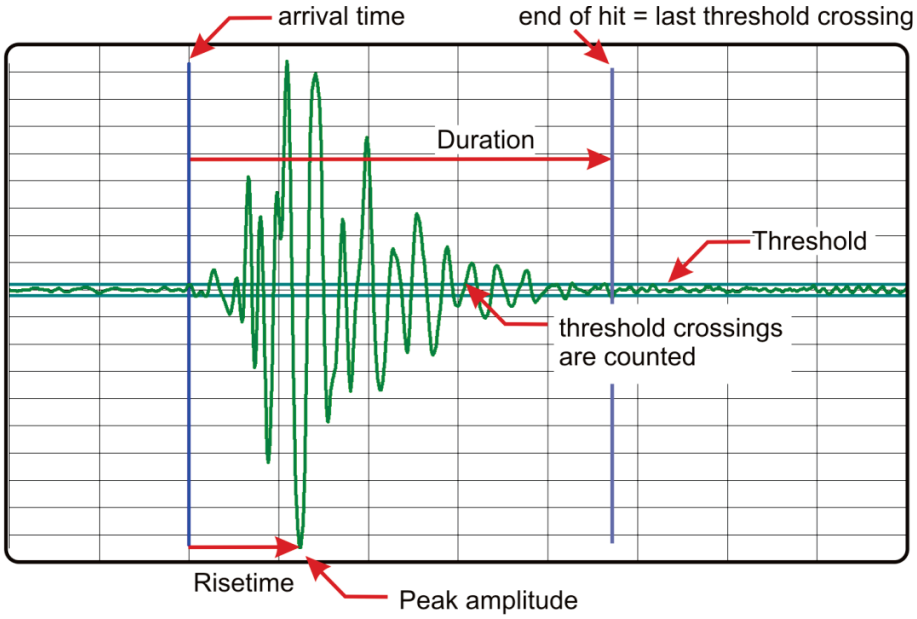

Figura 2.5: Parámetros característicos de una señal transitoria [20].

- *Peak amplitude* A: Es el valor máximo de tensión alcanzado por la señal en toda su duración. Se mide en μV o dB.

$$A_{dB} = 20 \log\left(\frac{A}{A_{ref}}\right); \qquad A_{ref} = 10\mu V \tag{2.4}$$

- *Threshold*: Se define como el umbral mínimo para la detección de una señal. Permite su diferenciación del ruido de fondo.

- Duración: Tiempo entre el comienzo y final de un hit. El inicio se corresponde con el momento en el que la señal supera el umbral de detección por primera vez y el final cuando lo hace por última.

- *Rise time* o tiempo de llegada: Tiempo transcurrido desde el inicio de un *hit* hasta alcanzar la *Peak amplitude*.

- *Counts* o conteo: Número de veces que la señal cruza el *threshold* en dirección positiva.

- Energía: Se define como la integral sobre la ventana de tiempo definida del cuadrado de la tensión eléctrica producida u_i. Como se trata de señales discretas, esta integral se convierte en un sumatorio de todas las muestras.

$$E = \frac{1}{f_s}\sum_{i=1}^{N} u_i^2 \tag{2.5}$$

  Donde f_s es la frecuencia de adquisición de datos en Hz y N el número de muestras tomadas para un transitorio.

2.3. Inteligencia Artificial

La inteligencia artificial se define como la disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico [21].

2.3.1. *Machine Learning*

El *Machine Learning*, o aprendizaje automático, es una rama de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a los sistemas informáticos aprender y tomar decisiones sobre datos sin intervención humana de forma directa [21].

2.3.2. Terminología

Para llevar a cabo un proceso de *Machine Learning* se debe trabajar con bases de datos. Los datos tomados se recogen en el *data set* donde cada muestra o instancia, contiene la descripción de un evento u objeto. Una instancia describe características del objeto, estas descripciones se llaman atributos o *features*, a los cuales les corresponden unos valores [22].

El proceso de utilizar *Machine Learning* para construir los modelos se llama entrenamiento o aprendizaje. Entrenando el algoritmo con el *data set*, se consigue el modelo, utilizado como término general para referirse a la salida del patrón obtenido de los datos [22].

Los datos se dividen para ser utilizados en la fase de entrenamiento y la de test, normalmente esta división es de un 80 % y un 20 %. A estos datos se les denomina datos de entrenamiento y de test [22].

La predicción de una muestra, se denomina *label* y a una muestra con su *label* se le llama ejemplo [22].

Cuando la predicción es un valor discreto, se trata de un problema de clasificación, mientras que si el resultado es continuo es de regresión. Además de predicciones, se pueden hacer agrupaciones de datos o *clustering*, según si los objetos comparten ciertas características [22].

Para evaluar el rendimiento de un modelo se utilizan métricas como la precisión, la tasa de error o la exactitud [23].

2.3.3. Tipos de *Machine Learning*

Una forma de clasificar los métodos de *Machine Learning* es según la forma en la que aprende el modelo, según si este es entrenado con supervisión humana o no.

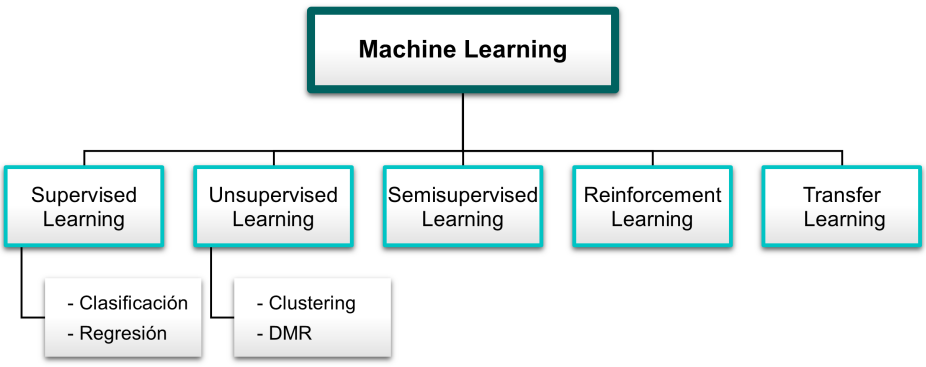

Figura 2.6: Esquema clasificación algoritmos de *Machine Learning*.

De acuerdo con la Figura 2.6, los diferentes tipos en los que se pueden clasificar los algoritmos son:

- *Supervised Learning*: Los datos de entrada al modelo incluyen los *labels*. El modelo predice la clase de datos desconocidos basándose en la información anterior conocida sobre las clases de objetos cuyas características son similares. Dentro de este grupo se encuentran algoritmos de regresión lineal y logística, SVM, árboles de decisión, KNN... [24]. Atendiendo al tipo de salidas de un algoritmo supervisado, se puede distinguir entre algoritmos de clasificación o regresión:

  - Clasificación: A cada elemento de entrada se le asigna una clase, el algoritmo asigna los objetos nuevos de entrada a cada una de las clases en función de las características de los datos proporcionados. Un ejemplo sería asignar la letra correspondiente del alfabeto a la entrada de un caracter manuscrito [25].

  - Regresión: Los resultados dados por el modelo son variables continuas. Un ejemplo sería un algoritmo que trata de predecir el valor de una casa en base a características como la ubicación, tamaño, antigüedad... [25].

- *Unsupervised Learning*: Los datos de entrada al modelo no están etiquetados por lo que el algoritmo busca patrones que los agrupan según su similitud. Algunos de los algoritmos son: *K-means*, HCA, PCA... [24]. Existen dos tipos de aprendizaje no supervisado: el agrupamiento o *clustering* y la reducción dimensional (DRM):

  - *Clustering*: Estos algoritmos tratan de agrupar los objetos en un número determinado de grupos, también llamados *clusters*, haciendo que los objetos que presentan características similares pertenezcan al mismo *cluster* [25].
  - *Dimensionality Reduction Methods* (DRM): En este grupo se incluyen aquellos algoritmos que tratan de representar datos de grandes dimensiones en espacios de menor dimensión. Resultan especialmente útiles para reducir la memoria necesaria para albergar los archivos y disminuir el tiempo de resolución. [25]

- *Semisupervised Learning*: Este tipo de algoritmos posee la capacidad de analizar datos etiquetados de forma parcial. Normalmente la mayoría de estos datos no están etiquetados y sólo unos pocos incluyen *labels*. Estos algoritmos son combinaciones de supervisados y no supervisados como por ejemplo DBNs (Deep Belief Networks)[23].

- *Reinforcement Learning*: El algoritmo trata de encontrar una secuencia de operaciones que lleguen a una solución objetivo que maximiza una recompensa, para ello se eligen o evitan acciones basadas en sus consecuencias [26]. Un ejemplo sería un algoritmo que juega a ajedrez y aprende de las decisiones que le llevan a la victoria o a la derrota o un modelo que trata de salir de un laberinto probando diferentes caminos.

- *Transfer Learning*: En este grupo se incluyen los algoritmos que utilizan información de otros modelos para mejorar su rendimiento. Resultan especialmente útiles cuando se dispone de pocos datos [27].

2.3.4. Algoritmos supervisados de clasificación

Atendiendo a las características de los datos a analizar, al tratarse de una caracterización de diversos impactos conocidos sobre un componente, se emplean los algoritmos supervisados de clasificación que se detallan a continuación.

Árboles de decisión

Un árbol de decisión sigue la estructura de un árbol (Figura 2.7), clasificando los datos desde la raíz hasta las hojas, donde los nodos representan las variables de entrada y las ramas las fronteras de decisión. De esta forma los datos se van clasificando en diversos grupos, las hojas, basados en los *features* de entrada [22].

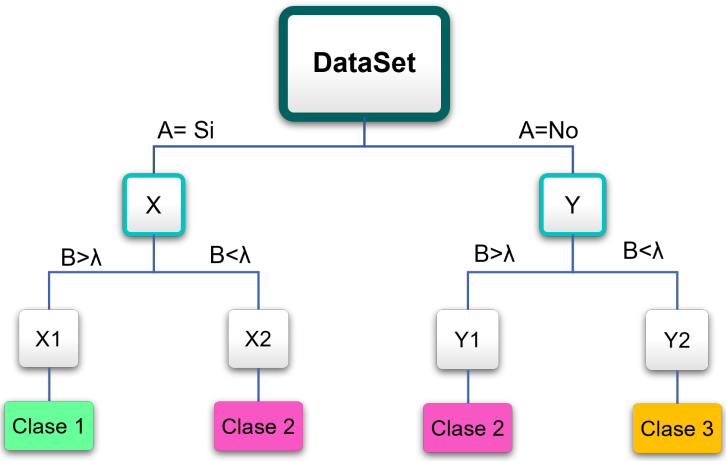

Figura 2.7: Esquema algoritmo árbol de decisión para 3 clases y dos *features*.

En la estructura del árbol, se debe determinar el número máximo de divisiones, lo que serían los nodos de ramificación; el criterio de división y cuantas divisiones hacer para un dato incorrecto [28].

KNN

Los algoritmos KNN (*K-Nearest Neighbors*) clasifican buscando datos similares en los ya proporcionados. En este caso no hay una fase de entrenamiento específica antes de la clasificación sino que se categorizan nuevos datos en base a los ya existentes [29].

El modelo primero calcula la distancia entre el objeto a clasificar y el resto de eventos, de ellos coge los k vecinos que tienen la menor distancia y en base a los *label* de estos se categoriza como una u otra clase (Figura 2.8) [29].

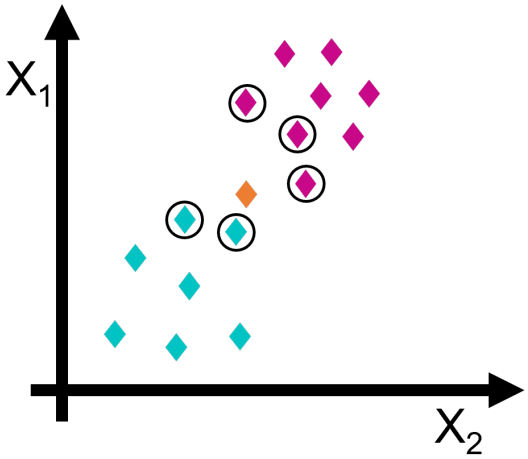

Figura 2.8: Esquema algoritmo KNN para 3 clases con $k = 5$.

Antes de clasificar los datos se debe decidir el número de vecinos k, la definición de distancia a utilizar y la forma de ponderar dicha distancia. Estos valores, pueden tomarse de forma arbitraria o bien utilizar validación cruzada para encontrar un valor óptimo [29].

Los tipos de distancia que existen son:

- Distancia euclídea: Para un espacio de n dimensiones, la distancia euclídea entre dos objetos i, j se define como [27]:

$$d(i,j) = \sqrt{\sum_{a=1}^{n}(x_{ia} - x_{ja})^2} \tag{2.6}$$

- Distancia Manhattan o *City Block*: Para un espacio de n dimensiones, mide la distancia entre dos objetos i, j utilizando únicamente segmentos ortogonales a los ejes [30]:

$$d(i,j) = \sum_{a=1}^{n}|x_{ia} - x_{ja}| \tag{2.7}$$

- Distancia Minkowski: Es una generalización de las distancias euclídea y Manhattan [30]:

$$d(i,j) = \sqrt[h]{\sum_{a=1}^{n}|x_{ia} - x_{ja}|^h} \tag{2.8}$$

- Distancia Chebyshev: También llamada distancia suprema, es una generalización de la distancia Manhattan cuando $h \to \infty$. Para dos objetos i, j se corresponde con la mayor de sus distancias a lo largo de cualquiera de los ejes [30]:

$$d(i,j) = \lim_{h\to\infty} \sqrt[h]{\sum_{a=1}^{n} |x_{ia} - x_{ja}|^h} = \max_{a}^{n} |x_{ia} - x_{ja}| \tag{2.9}$$

En la Figura 2.9 se comparan estas distancias.

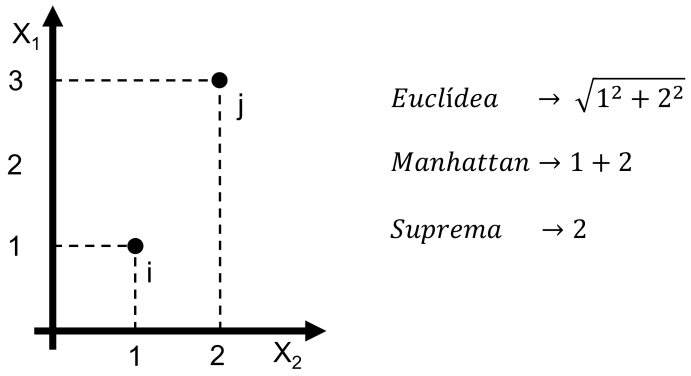

Figura 2.9: Distancias euclídea, Manhattan y suprema para dos objetos *i, j*.

- Distancia Mahalanobis: Se basa en la distancia euclídea, tratando de ajustarla en función de la importancia de los *features* y su relación. Para ello, utiliza la matriz inversa de covarianza S o bien una matriz personalizada de pesos M [30]. Para dos vectores **i, j** que contienen los datos de dos objetos, la distancia se define como:

$$d(\mathbf{i},\mathbf{j}) = \sqrt{(\mathbf{i}-\mathbf{j})^T \cdot S^{-1} \cdot (\mathbf{i}-\mathbf{j})} \tag{2.10}$$

- *Cosine similarity*: Compara dos vectores evaluando el coseno del ángulo comprendido entre ellos. Esta distancia está acotada en el intervalo [-1 1] siendo 1 cuando el ángulo es 0°, 0 cuando son ortogonales y 1 si van en sentido contrario [24]. Para dos vectores **i, j** que contienen los datos de dos objetos, la distancia se corresponde con:

$$d(\mathbf{i},\mathbf{j}) = cos(\mathbf{i},\mathbf{j}) = \frac{\mathbf{i}\cdot\mathbf{j}}{||\mathbf{i}|| \cdot ||\mathbf{j}||} \tag{2.11}$$

- Coeficiente de correlación lineal: Compara la relación lineal existente entre dos variables aleatorias *i,j*. Toma valores en el intervalo [-1 1]. Los valores extremos representan una relación lineal perfecta entre las variables que puede ser positiva o negativa. El valor 0, implica la ausencia de relación lineal. Se utiliza la relación lineal simple definida como [31]:

$$d(i,j) = \frac{Cov(i,j)}{\sigma_i \sigma_j} \tag{2.12}$$

Donde σ_i y σ_j representan la desviación típica dada por la Ecuación 2.13 y $Cov(i,j)$ la covarianza según la Ecuación 2.14 siendo N el número de observaciones.

$$\sigma_i = \sqrt{\frac{\sum_k^N (i_k - \bar{i})^2}{N}} \qquad \sigma_j = \sqrt{\frac{\sum_k^N (j_k - \bar{j})^2}{N}} \tag{2.13}$$

$$Cov(i,j) = \frac{\sum_k^N [(i_k - \bar{i})(j_k - \bar{j})]}{N} \tag{2.14}$$

- Coeficiente de correlación de Spearman: Es una medida no paramétrica que determina la relación existente entre dos sets de datos. Al igual que el coeficiente de correlación lineal, toma valores en el intervalo [-1 1]. Para un espacio de n dimensiones, el coeficiente entre dos objetos i, j se define según [31]:

$$d(i,j) = 1 - \frac{6 \sum_{k=1}^n (x_{ik} - x_{jk})^2}{n(n^2 - 1)} \tag{2.15}$$

- Distancia Hamming: Indica el porcentaje de *features* que son diferentes para dos objetos [28].

- Coeficiente *Jaccard*: Indica el porcentaje de *features* distintos de 0 que son diferentes para dos objetos dados. Como valor de distancia se utiliza 1 menos este coeficiente [28].

La ponderación de la distancia puede ser: equivalente, cuando la distribución es equitativa; inversa, cuando se pondera por la inversa de la distancia o inversa al cuadrado si se utiliza la inversa de la distancia al cuadrado [28].

La forma más sencilla de clasificar es por pluralidad, determinando la clase a la que pertenece un nuevo objeto en base a la que pertenecen la mayor parte de sus vecinos. Sin embargo, también se desarrollan algoritmos KNN que otorgan un peso distinto a cada vecino en función de su distancia al nuevo objeto [32].

Naive Bayes

El algoritmo de clasificación *Naive Bayes* asume que todos los *features* son independientes uno de otro a la hora de agrupar los datos. Utiliza los datos de entrenamiento para calcular una función de distribución probabilística, normal o continua (*kernel*), para cada una de las clases. Con ello, se obtiene la probabilidad de pertenecer a dicha clase en base al valor de sus *features*. Se utiliza la probabilidad condicional para calcular la probabilidad de que un objeto pertenezca a una clase u otra y la predicción final se realiza asignando la clase con la probabilidad más elevada. A pesar de no ser siempre cierto el hecho de asumir que los *features* son independientes, en la práctica el algoritmo resulta bastante efectivo [33].

El modelo se basa en el teorema de Bayes, con el cual se obtiene la probabilidad de ocurrencia de un evento A sabiendo que otro evento B ha ocurrido (P(A|B)). Este teorema se rige por la Ecuación 2.16 [34].

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.16}$$

Donde P(A) es la probabilidad de A, P(B) la probabilidad de B y P(B|A) es la probabilidad de que ocurra B sabiendo que ha ocurrido A.

Ante nuevos datos de entrada, el algoritmo usa las probabilidades obtenidas en el entrenamiento para predecir la clase de los objetos con el Terorema de Bayes (Figura 2.10)[24].

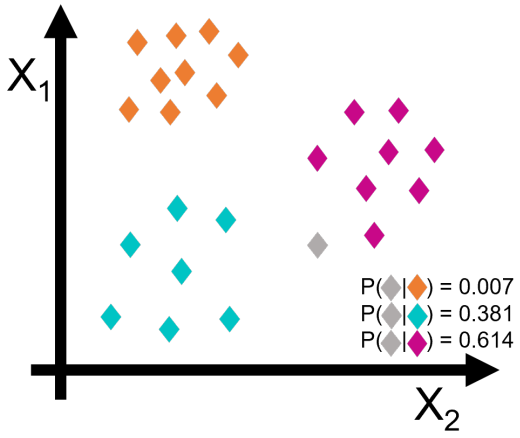

Figura 2.10: Esquema Algoritmo *Naive Bayes* para tres clases.

Análisis discriminante

El análisis discriminante, al igual que el algoritmo de *Naive Bayes*, utiliza los datos de entrenamiento para calcular la distribución que siguen los *features* de cada una de las clases. Pero, a diferencia del anterior, en este algoritmo no se utiliza la probabilidad de pertenencia a cada una de las clases sino que se calculan las ecuaciones de las fronteras entre clases. Además, en este caso no se asume que los *features* sean independientes [35].

Se asignan distribuciones normales multivariantes (Ecuación 2.17) a cada una de las clases y en base a su media y desviación típica se calculan las fronteras (también llamadas hiperplanos) hallando los puntos en los cuales la probabilidad de pertenecer a cada una de las clases es la misma [36].

$$p(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_k|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{2.17}$$

Para conformar el modelo se debe encontrar la distribución de cada una de las clases en función de la media μ y la desviación típica Σ de los datos de entrenamiento [37].

A los nuevos datos se les asigna una clase u otra en función del lado de la frontera en el que se encuentren.

En el caso de que se asuma que todas las distribuciones tienen la misma forma, y por tanto la misma varianza, el problema se simplifica, siendo todas las fronteras lineales. En el caso de trabajar con distribuciones con distinta varianza, las fronteras serán cuadráticas (Figura 2.11) [36].

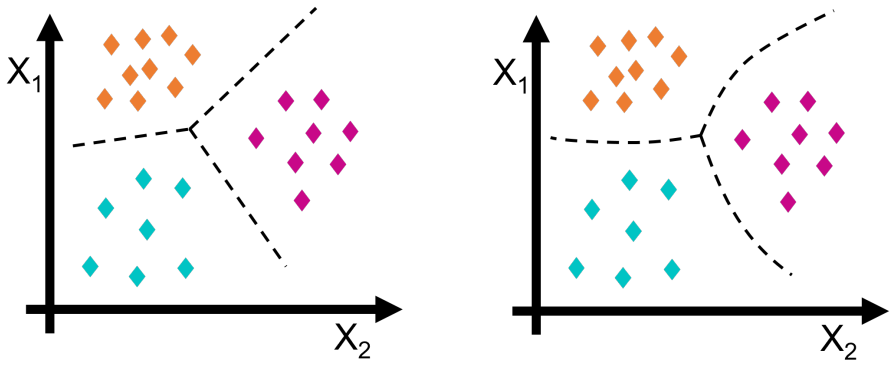

Figura 2.11: Esquemas algoritmos análisis discriminante para tres clases. (izquierda, misma varianza; derecha, distinta varianza).

SVM

Los algoritmos SVM (*Support Vector Machine*) o máquinas de vectores de soporte, clasifican los datos buscando hiperplanos que establecen una frontera entre los objetos en el espacio multidimensional que forman los distintos *features*. Se buscan las fronteras que están lo mas lejos posible de cada observación [24].

Un algoritmo SVM se determina con los siguientes parámetros:

- Hiperplano: Para un espacio de n dimensiones, donde n es el número de *features* de los objetos a clasificar, un hiperplano es una superficie de $n-1$ dimensiones que separa y clasifica los sets de datos [24].
- Vector soporte: Son los objetos críticos de cada clase al ser los más cercanos al hiperplano, si estos datos desaparecen, la posición del hiperplano cambia [24].
- Margen: Es la distancia del hiperplano a los vectores soporte. Un mayor margen, implica una mayor confianza en la clasificación [24].

De los infinitos hiperplanos que pueden separar un conjunto de datos, se trata de encontrar el óptimo que proporcione una exactitud y precisión adecuadas para valores desconocidos. El objetivo es encontrar aquellos hiperplanos que logren el mayor margen posible para todos los conjuntos de datos, incluso si eso implica clasificar incorrectamente algunos de los objetos. Esto se debe a que llegar a un modelo inflexible, que ajuste perfectamente los datos de entrenamiento, puede resultar en *overfitting*[1] (Figura 2.12).

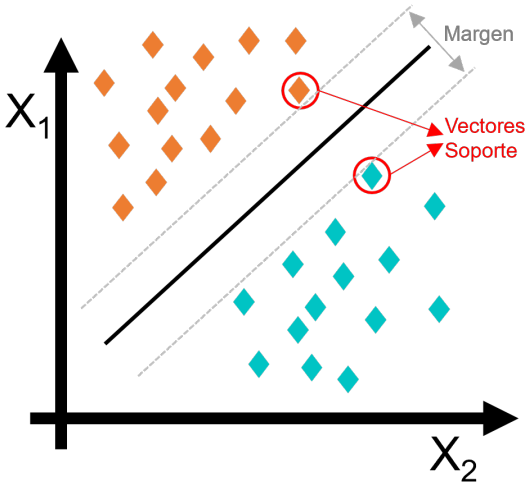

Figura 2.12: Esquema Algoritmo SVM para dos clases.

Una vez creado el modelo, el algoritmo clasifica los nuevos datos ubicándolos en el espacio y eligiendo su clase en función de qué lado del plano se encuentren.

[1]Se dice que un modelo presenta *overfitting* cuando se ajusta perfectamente a los datos de entrenamiento pero no es capaz de clasificar de forma correcta nuevos objetos

Redes neuronales artificiales

Las redes neuronales artificiales (*Artificial Neural Network*, ANN) basan su funcionamiento en la imitación del sistema nervioso humano. El sistema nervioso, coordina las acciones humanas transmitiendo señales entre diferentes partes del cuerpo. Estos movimientos se coordinan por medio de un tipo especial de células llamadas neuronas. Las neuronas tienen una estructura especial que les permite conectarse unas a otras y transmitir y recibir información [24].

Una neurona biológica está formada por tres partes fundamentales que se distinguen en la Figura 2.13: las dendritas, que reciben señales de neuronas vecinas; el soma, que constituye el cuerpo principal de la neurona, acumula y procesa la información de las diferentes dendritas y el axón que transmite la información a las neuronas vecinas a través de sus terminales. Entre el terminal de un axón y las dendritas hay un pequeño espacio conocido como sinapsis, este espacio puede excitarse o inhibirse para realizar la conexión o no entre neuronas [24].

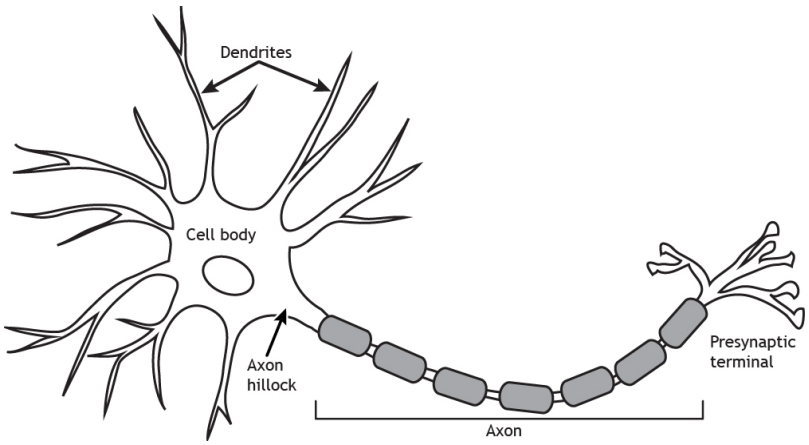

Figura 2.13: Estructura biológica de una neurona [38].

Una neurona artificial o perceptrón, es el elemento de funcionamiento básico de la red. Funciona de forma semejante a una neurona biológica: recibe una señal de input x_i procedente del «exterior» o de la salida de otras neuronas y a través de los distintos elementos que componen la red devuelve el output (Figura 2.14) [39].

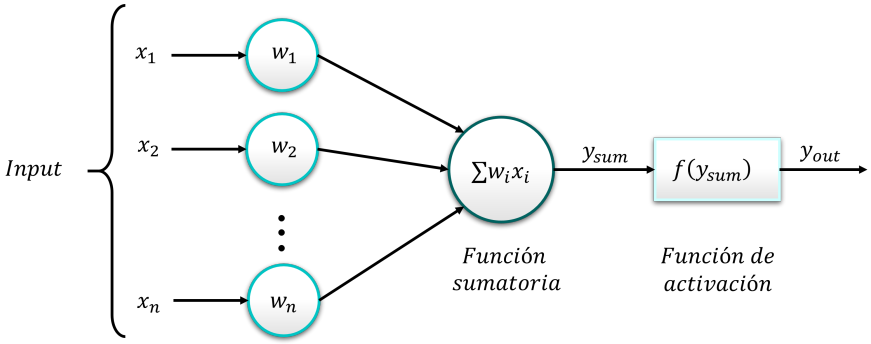

Figura 2.14: Esquema neurona artificial.

Los elementos que componen una neurona son [24]:

- Pesos de entrada w_i: La señal x_i combinada con el peso w_i define la influencia del input. Este efecto puede ser: positivo, suponiendo un efecto excitativo; o negativo siendo en ese caso el efecto inhibitorio. Esta repercusión se refleja en el output, que es la función sumatoria.
- Función sumatoria: Se encarga de sumar cada input computado con su peso. Además se suele incluir un *bias* b que ajusta el input de la función de activación (Ecuación 2.18).

$$y_{sum} = b + \sum_{i=1}^{n} w_i \cdot x_i \tag{2.18}$$

- Función de activación: Transmite la señal de output cuando se supera un determinado valor umbral o se cumple con los requisitos de la función (Ecuación 2.19).

$$y_{out} = f(y_{sum}) \tag{2.19}$$

Existen varios tipos de funciones de activación. Entre las mas utilizadas se encuentran [24]:

- Función identidad: La salida es la misma que la entrada.

$$y_{out} = y_{sum} \tag{2.20}$$

- Función escalón: Cuando el valor de entrada es negativo devuelve un 0 y cuando es 0 o positivo devuelve un 1.

$$y_{\text{out}} = f\left(y_{\text{sum}}\right) = \begin{cases} 1, & y_{sum} \geq 0 \\ 0, & y_{sum} < 0 \end{cases} \tag{2.21}$$

- *Threshold*: Un *Threshold* funciona como la función escalón pero utilizando un valor umbral a en lugar de 0. Cuando el valor de entrada es menor que el umbral devuelve un 0 y cuando coincide con el umbral o es un valor mayor devuelve un 1.

$$y_{\text{out}} = f\left(y_{\text{sum}}\right) = \begin{cases} 1, & y_{sum} \geq a \\ 0, & y_{sum} < a \end{cases} \tag{2.22}$$

- *ReLU*: *Rectified Linear Unit* es la función de activación más popular en redes convolucionales y *deep learning*. La función devuelve 0 para valores menores que 0 y el valor de entrada y_{sum} cuando es mayor o igual a 0.

$$y_{out} = \begin{cases} y_{sum}, & y_{sum} \geq 0 \\ 0, & y_{sum} < 0 \end{cases} \tag{2.23}$$

- Sigmoidea: Es la función más utilizada en redes neuronales, ya que se trata de una función derivable y por tanto, continua, algo que necesitan muchos tipos de algoritmos. Existen dos tipos de funciones sigmoideas: binaria y bipolar, variando la salida entre 0 y 1 en el primer caso y entre -1 y 1 en el segundo.

$$y_{out} = \frac{1}{1 + e^{-k\, y_{sum}}}; \qquad y_{out} = \frac{1 - e^{-k\, y_{sum}}}{1 + e^{-k\, y_{sum}}}; \tag{2.24}$$

Binaria *Bipolar*

Donde k es un parámetro que indica la pendiente de la función, se encuentra en el rango [0, 1] y debe ajustarse a la inclinación deseada.

- Tangente hiperbólica: Función de activación continua similar a la función sigmoidea. Habitualmente se utiliza en las redes neuronales que se propagan hacia detrás.

$$y_{out} = \frac{e^{y_{sum}} - e^{-y_{sum}}}{e^{y_{sum}} + e^{-y_{sum}}}; \tag{2.25}$$

Definidos todos los elementos de una red neuronal, se puede establecer una analogía directa entre la actividad biológica y las redes neuronales artificiales (Tabla D.5, Figura 2.15).

Tabla 2.1: Analogía entre una neurona biológica y una artificial [40].

Neurona biológica		**Neurona Artificial**
Señales que llegan a la sinapsis	→	Entradas a la neurona
Carácter excitador o inhibidor de la sinapsis de entrada	→	Pesos de entrada
Estímulo total	→	Función sumatoria
Activación o no	→	Función de activación
Respuesta	→	Función de salida

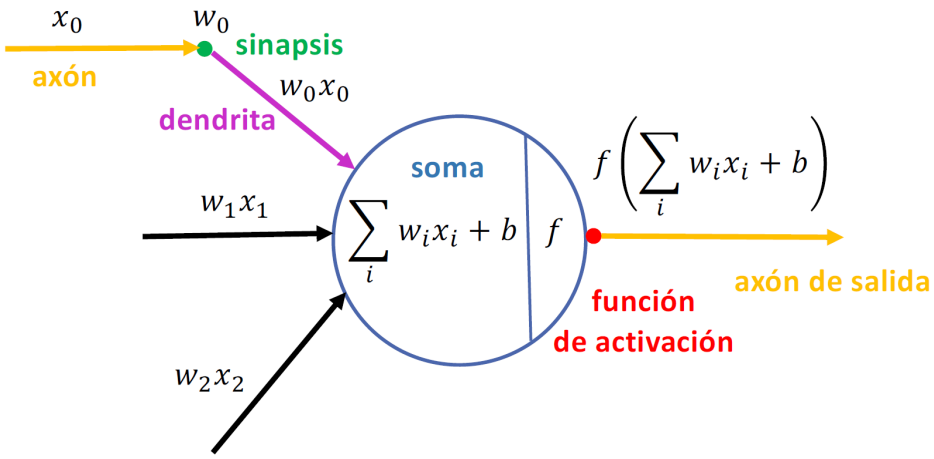

Figura 2.15: Esquema analogía entre una neurona biológica y una artificial [41].

La forma de interconectar los componentes constituye la arquitectura de la red. Esta arquitectura debe satisfacer una serie de requisitos [24]:

- Como mínimo, debe haber dos capas de neuronas en la red, input y output *layer*.
- Puede haber capas de neuronas intermedias entre el input y output llamadas capas ocultas o *hidden layers*
- Las neuronas pueden estar conectadas a una, todas o alguna de las neuronas de la capa siguiente.
- Pueden existir una o varias señales de salida. Si hay varias señales de salida, pueden estar conectadas entre sí.
- La salida de una capa puede ser la entrada a neuronas de esa capa o de capas anteriores.

Cumpliendo con estas condiciones, se definen varios tipos de arquitectura posibles dentro de una red neuronal [24]:

- Red neuronal monocapa: Es la red neuronal más básica, consta de las capas de salida y entrada. Se considera monocapa pese a ser dos *layers* ya que la capa de input no realiza ninguna operación, simplemente pasan las señales y las operaciones sólo las realiza la última capa (Figura 2.16).
- Red neuronal multicapa: En este caso, existen una o varias *hidden layers* entre el input y el output. El número de neuronas en cada red puede ser diferente(Figura 2.17).

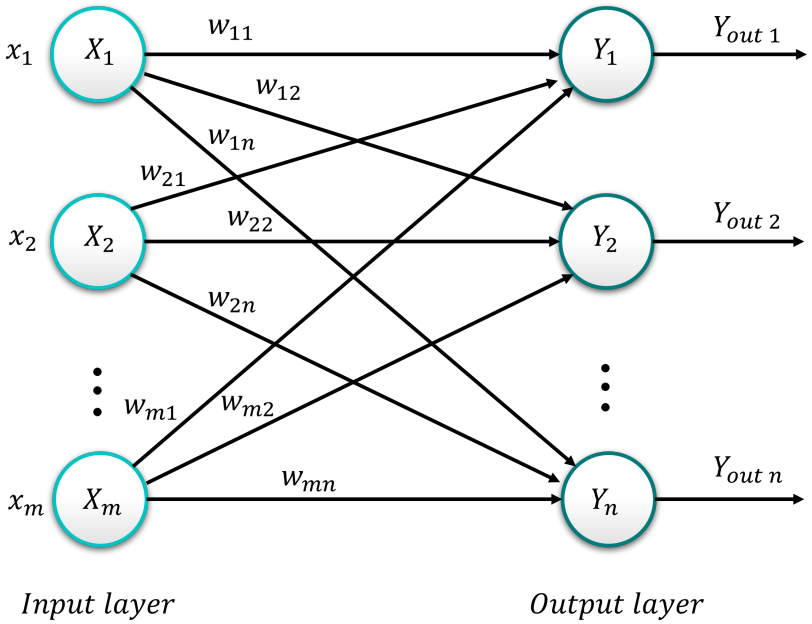

Figura 2.16: Esquema red neuronal monocapa.

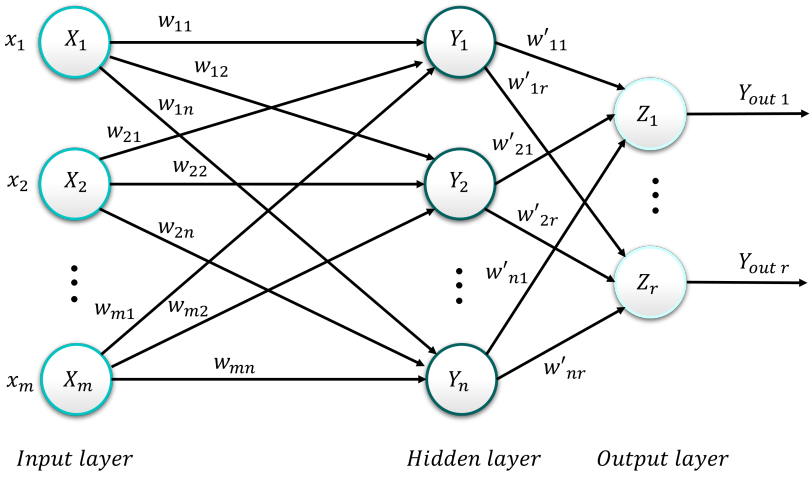

Figura 2.17: Esquema red neuronal multicapa.

Tanto en las redes multicapa como en las monocapa, la señal siempre se transmite en el mismo sentido, del input al output. Esto se conoce como red no recurrente o prealimentada.

- Red neuronal competitiva: Presenta una estructura similar a la de una red monocapa, con la diferencia de que las neuronas de salida están conectadas unas a otras. Esta conexión puede ser parcial o total. En estas redes, las neuronas de la capa de salida, compiten entre sí para representar la entrada (Figura 2.18).

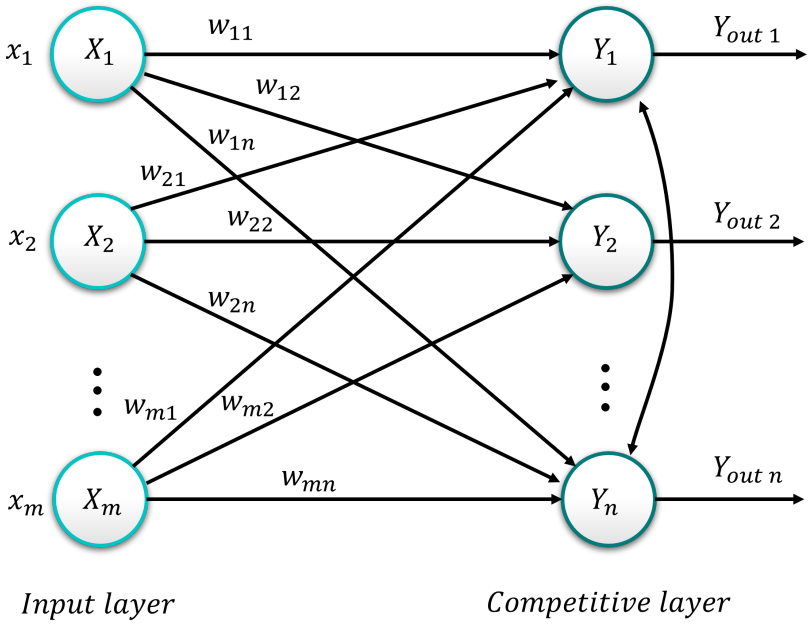

Figura 2.18: Esquema red neuronal competitiva.

- Red neuronal recurrente: A diferencia de las redes anteriores, en las redes recurrentes, existe un bucle de información de feedback de la capa de output a la de input o incluso en una misma capa (Figura 2.19).

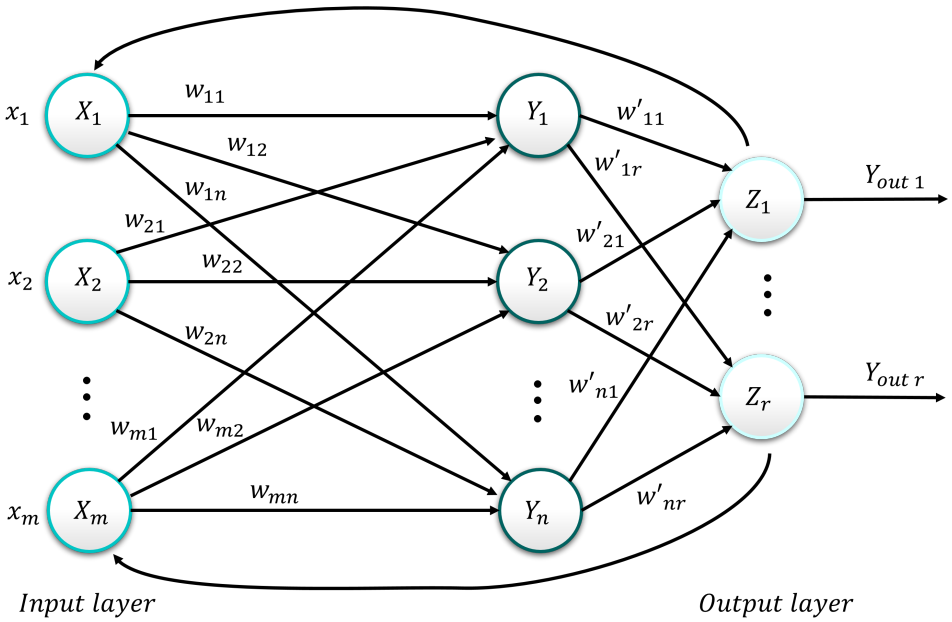

Figura 2.19: Esquema red neuronal recurrente.

- Red convolucional: Su estructura es mucho más compleja que las anteriores. Se utilizan en la clasificación de imágenes. Como este no es el objetivo del trabajo quedan fuera del estudio.

2.3.5. Proceso de aprendizaje

El proceso de aprendizaje de un algoritmo sigue el esquema de la Figura 2.20.

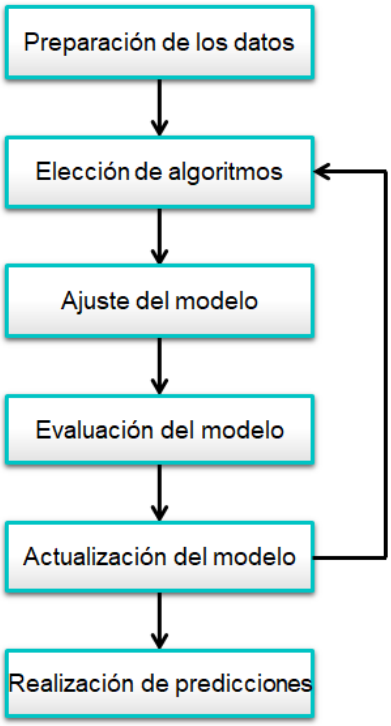

Figura 2.20: Esquema proceso de aprendizaje de un modelo.

Con los datos en el formato adecuado, se elige un algoritmo y se crea el modelo utilizando los datos de entrenamiento. El rendimiento de este modelo se evalúa con los datos de test. En función de su rendimiento y la precisión esperada, se actualizan parámetros del modelo para repetir el proceso anterior y mejorar los resultados. Por último, con el modelo final se realizan las predicciones.

Existen numerosos algoritmos que pueden conformar un modelo dado un conjunto de datos. A continuación, se presentan diversas técnicas que se utilizan para mejorar el rendimiento de los mismos y seleccionar el óptimo.

Validación de datos

La forma de evaluar el funcionamiento de los modelos es hacer predicciones con los datos de test y calcular el error cometido.

A la hora de elegir un modelo u otro, una posible forma de elección sería calcular la precisión de varios y coger el que tenga mayor valor. Sin embargo el método de validación puede influir en su precisión:

- *Holdout validation*: También conocido como método de retención, consiste en reservar un porcentaje de los datos para la validación [28].

Con el uso de este método, la precisión se calcula con una parte concreta del set de datos. En este caso, es posible que un algoritmo determinado tenga un buen rendimiento para esos datos en particular pero que no sea capaz de generalizar para otros datos, o por el contrario, que para unos datos concretos no tenga una buena precisión, pero para datos en general funcione correctamente (Figura 2.21) [36].

Figura 2.21: Esquema *holdout validation*.

- *K-fold cross validation*: Con el fin de evitar el problema planteado en el método anterior, se emplea la validación cruzada. Este procedimiento consiste en repetir la evaluación del modelo k veces donde, en cada iteración, se realiza una división diferente para los datos de entrenamiento y test. Para cada una de estas iteraciones se evalúa el desempeño del modelo, calculando un promedio de rendimiento general (Figura 2.22) [24].

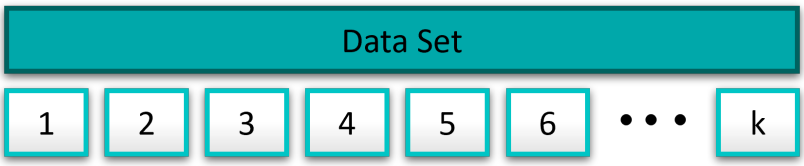

Figura 2.22: Esquema *k-fold cross validation*.

- LOOCV (*Leave one out cross validation*): Es el caso extremo de validación cruzada donde cada uno de los datos se utiliza como test. De esta forma se maximiza el número de datos de entrenamiento, pero también el tiempo de ejecución ya que se tienen que hacer tantas iteraciones como número de instancias contiene el set de datos (Figura 2.23). En la práctica no se suele utilizar [24].

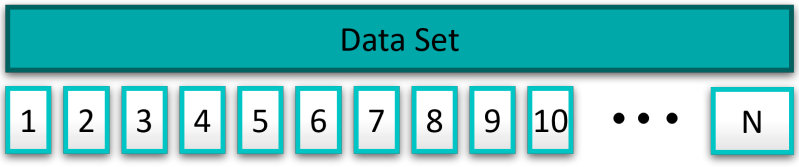

Figura 2.23: Esquema *leave one out cross validation*.

Reducción de *features*

Los problemas de *Machine Learning* a menudo implican utilizar datos de gran dimensión con una numerosa cantidad de *features*, lo cual supone un coste computacional grande.

La técnica de reducción de *features* proporciona beneficios en el tiempo de cálculo además de generar modelos simplificados más fáciles de interpretar. Normalmente los modelos en los que se realiza la reducción de *features* son capaces de generalizar mejor con nuevos datos de entrada ya que en general, contar con un menor número de *features* implica una mayor precisión [24].

Existen dos formas comunes de llevar a cabo este proceso que son la transformación y selección de *features*.

- Transformación de *features*: Este método transforma las coordenadas espaciales de las instancias de entrada.

  El método mas utilizado para la transformación de datos es el PCA (*Principal Component Analysis*), que transforma un espacio de datos de n dimensiones, en uno de las mismas dimensiones pero con componentes ortogonales, es decir, independientes unas de otras.

  El vector de *features* de entrada de una instancia del set de datos se transforma a un espacio vectorial con vectores directores llamados componentes principales, ortogonales unos a otros. Cada instancia se transforma por tanto en un espacio distinto, cuyas componentes principales son independientes. Las componentes principales explican la variabilidad en los datos.

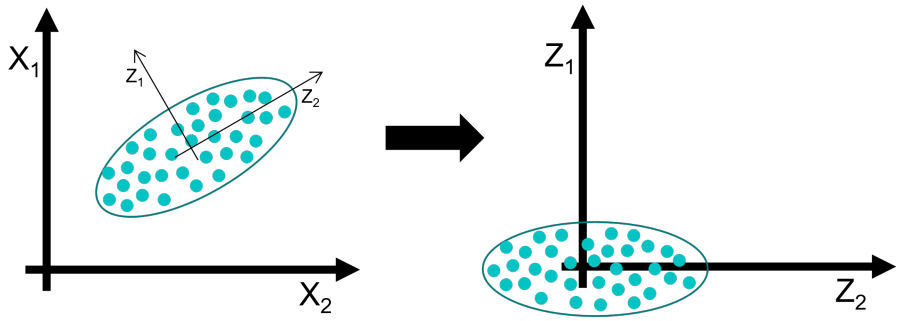

Figura 2.24: Esquema *principal component analysis*.

En la Figura 2.24 se muestra un ejemplo básico de PCA para dos *features*. La muestra de datos se centra en el espacio para después alinear los nuevos ejes con las direcciones de máxima varianza. Si la varianza de uno de los dos ejes es lo suficientemente pequeña (como sería en este caso en el eje de ordenadas), se

puede ignorar, lo cual reduciría el espacio a una dimensión [39].

Para un set de datos, las variables transformadas contienen la misma información que los datos originales. Sin embargo, aquellas que presentan una varianza menor pueden atribuirse a ruido más que a información [36]. Con ello, el PCA se puede utilizar para una reducción de *features*, descartando aquellas componentes que no son consideradas principales, es decir, componentes que tengan una varianza menor a un umbral.

- Selección de *features*: Este método selecciona parte de las variables observadas.

Los sets de datos a menudo, pueden contener atributos que no tienen ninguna relación con la respuesta deseada. Estos datos no deben incluirse en el modelo. Por otro lado, pueden existir *features* que estén fuertemente correlacionados, de los cuales, sólo uno debe incluirse en el modelo [36].

Existen varios métodos llamados *Feature Ranking Algorithms* que tratan de asignar una puntuación a cada uno de los *features* en base a lo relevantes que son para el modelo de acuerdo con una métrica dada. Algoritmos comunes son: método chi cuadrado, MRMR (*Minimum Redundancy Maximum Relevance*), ANOVA (Análisis de la varianza) y *Kruskal Wallis*. Estos métodos tienen cada uno una base teórica distinta, pero todos ellos buscan ordenar los *features* de mayor a menor notoriedad para el análisis en función de su contribución a la diferenciación de los datos. Una vez ordenados, se eligen los más relevantes para el entrenamiento del modelo [36].

Otra forma más sencilla, pero también válida para la selección de features, es la selección de *features* de forma secuencial. Este método consiste en añadir y quitar variables en un determinado modelo en cada iteración y evaluar el efecto sobre la precisión.

Optimización de hiperparámetros

Un hiperparámetro es un parámetro de un algoritmo de aprendizaje pero no del modelo. Por ello, no se ve afectado por el proceso de aprendizaje, debe fijarse antes del entrenamiento y permanece constante durante el mismo [23]. Ejemplos de hiperparámetros son el número de vecinos en un algoritmo KNN o la cantidad de neuronas en una capa oculta de una red neuronal.

La selección adecuada de hiperparámetros puede tener un impacto significativo en la precisión del modelo aunque elegir los adecuados es complejo y requiere un gran consumo de tiempo [23]. Frente a este problema se plantean algoritmos de optimización de hiperparámetros que entrenan los modelos con combinaciones de los mismos hasta conseguir el conjunto que presenta mayor precisión [23].

Ensemble learning

El método de *ensemble learning*, también conocido como aprendizaje por conjuntos, entrena varios métodos para clasificar un mismo *dataset*.

Existen algoritmos que se consideran *weak learners* (aprendices débiles), lo que significa que son especialmente sensibles a los datos de entrenamiento. Sin embargo, con la combinación de varios se pueden lograr buenos resultados [23].

La Figura 2.25 muestra un esquema de la que sería una arquitectura típica de un *Ensemble*. Este conjunto, contiene varios algoritmos entrenados con los mismos datos, que pueden ser iguales o diferentes, las salidas de los mismos se combinan para lograr un resultado de clasificación más preciso [42].

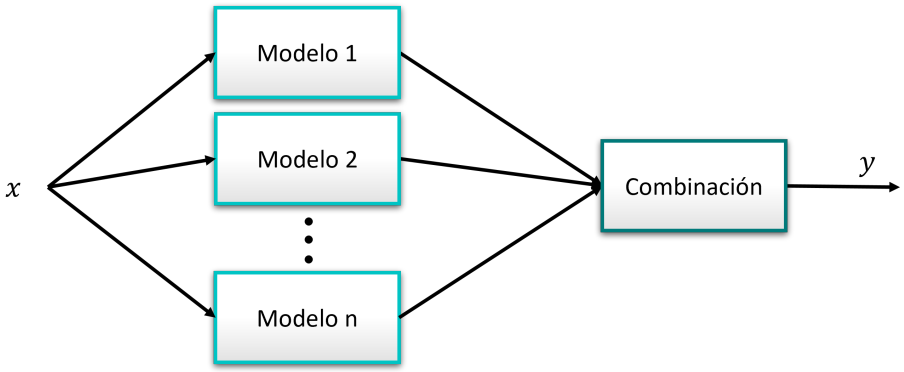

Figura 2.25: Esquema *ensemble learning*.

Uno de los conjuntos mas conocidos es la agrupación de árboles de decisión, llamada *Random Forest*, que a pesar de su simplicidad, es uno de los algoritmos de *Machine Learning* más potentes disponibles actualmente [23].

Capítulo 3

Diseño Experimental y Toma de Datos

La toma de datos de emisiones acústicas de CFRP para su posterior análisis mediante algoritmos de IA se lleva a cabo sobre una placa de fibra de carbono a la cual se le acopla el equipo de emisiones acústicas conectado al software de adquisición.

3.1. Definición del Ensayo

Para el ensayo, se elige una placa plana de fibra de carbono unidireccional de dimensiones 300x50x1.2 mm, donde la orientación de las fibras está alineada de forma paralela a la dirección de mayor longitud. Se elige este tipo de placa, cuyas características son lo más sencillas posibles, con el fin de minimizar los efectos producidos por la presencia de las diferentes formas de onda derivadas de los distintos modos de propagación en sólidos. Además, se trata de mitigar los efectos de la atenuación derivados de la presencia de diferentes velocidades de propagación entre fibras y matriz.

Los impactos sobre la placa se realizan con hielo, arena y una esfera metálica, simulando de esta forma impactos que pueden suceder de forma habitual sobre piezas fabricadas con CFRP en una aeronave, como pueden ser: el desprendimiento de hielo formado durante el vuelo, la presencia de arena en pista o el impacto de un FOD (Foreign Object Damage) como puede ser un tornillo (Figura 3.1). Para el ensayo se dejan caer estos objetos desde una altura constante de 40 cm.

Figura 3.1: Objetos utilizados para ejercer los impactos (esfera, hielo y arena) sobre la placa unidireccional de CFRP en la que se realizan los ensayos.

3.2. Adquisición de Datos

La toma de datos se realiza mediante dos sensores de emisiones acústicas VS150-M directamente conectados a la placa. Estos se colocan a idéntica distancia del punto de impacto con el fin de evitar la influencia de la atenuación (Figura 3.2).

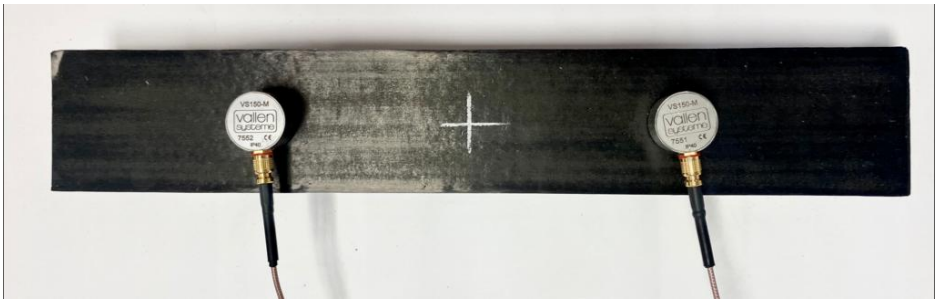

Figura 3.2: Ubicación de los sensores sobre la placa de CFRP.

Los sensores de emisiones acústicas son sensores piezoeléctricos que convierten el movimiento de la superficie de la placa, producido por la onda de sonido elástica, en señales eléctricas que pueden ser procesadas por el equipo de medida [43].

Los sensores se fijan a la placa con pinzas de compresión, utilizando como acoplante grasa sintética Loctite SuperLube (Figura 3.3).

Figura 3.3: Sujeción de los sensores a la placa de CFRP.

Las emisiones acústicas son monitorizadas por el sistema LinWave de Vallen Systeme, el cual cuenta con dos canales de adquisición. A este dispositivo se conectan los dos sensores VS150-M sin preamplificador, cuya banda de frecuencias de operación se sitúa entre 100 y 400 kHz (Figura 3.4).

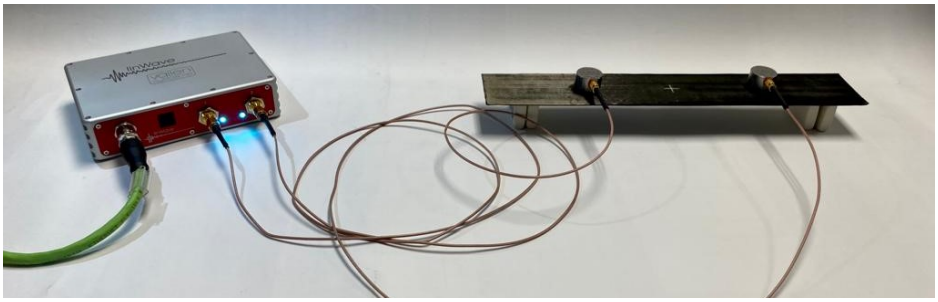

Figura 3.4: Montaje sistema de adquisición de emisiones acústicas.

La frecuencia de adquisición de datos (*sampling rate*) se establece en 2 MHz y el umbral en 35dB, con el fin de evitar interferencias generadas por ruidos.

Las señales se recogen en forma de transitorios, comúnmente conocidos como *hits*. Se deben determinar parámetros asociados a la toma de este tipo de señales como es el DDT, *Pre-trigger* y *Post-Duration*.

El DDT (*Duration Discrimination Time*), diferencia unos hits de otros, separando las señales. Una nueva señal es detectada cuando no ha sucedido ningún cruce del umbral en una ventana de tiempo mayor o igual al DDT. El *Pre-trigger* indica el intervalo de tiempo anterior al *trigger* en el cual se deben recoger los datos de la onda, siendo el *trigger* el momento en el cual la señal cruza el umbral por primera vez. Similar al *Pre-trigger*, el *Post-Duration* indica el intervalo de tiempo tras el último cruce de umbral durante el que se recoge información de la señal (Figura 3.5).

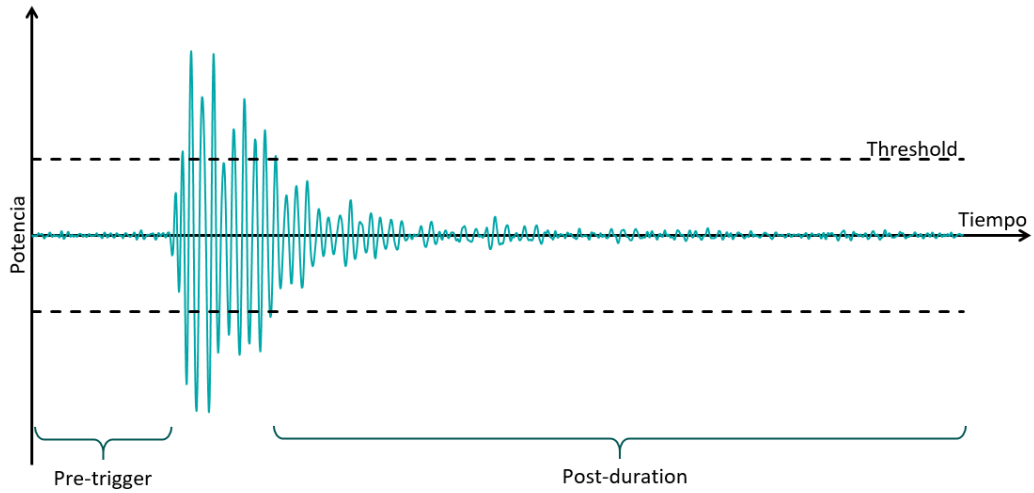

Figura 3.5: Representación gráfica de un *hit*.

De la experiencia en otros ensayos, se establece un DDT de 250 μs, un *Pre-trigger* de 100μs y un *Post-Duration* de 700μs.

Previo a la recogida de información, se lleva a cabo la prueba *Hsu-Nielsen, Pencil lead break AE test*, que consiste en la rotura de una mina de lápiz a 30° con el objetivo de excitar todas las frecuencias, consiguiendo una amplitud de señal elevada y comprobando con ello la correcta colocación de los sensores (Figura 3.6).

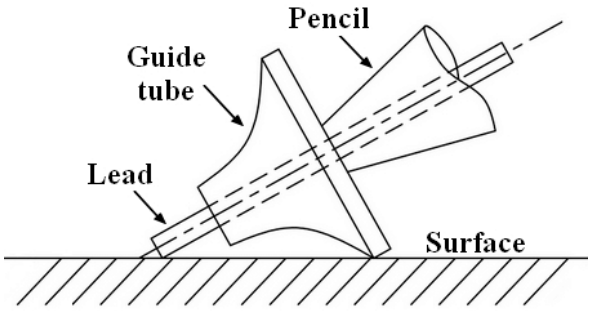

Figura 3.6: *Hsu-Nielsen source* [44].

Se realizan tres ensayos para el registro de los datos. Los impactos se realizan en el centro de la placa y desde una misma altura con el fin de evitar introducir variabilidad en los resultados.

Capítulo 4

Procesamiento de Señales

Para el tratamiento de datos se utilizan los archivos que genera el software de la empresa Vallen Systeme GmbH. La toma de datos genera tres tipos de archivos, que se diferencian por su extensión. El archivo *.pridb* contiene los descriptores de la señal acústica, el *.tradb* almacena los datos de los transitorios y el *.vaex* se utiliza para visualizar los datos en el software de adquisición.

Se utiliza el *script ExtraccionDatatos.py*, disponible en el Anexo, para leer los valores de los archivos *.pridb* y *.tradb*. Se emplea la librería vallenae en *python* de Vallen Systeme GmbH para extraer los datos de medida de emisiones acústicas, tanto los datos descriptores de los *hits* como los transitorios. Posteriormente, estos datos se exportan a archivos tipo «.csv» para su posterior lectura y procesamiento con MATLAB.

El *script* de MATLAB *Procesamiento.m*, disponible en el Anexo, importa, lee los archivos y los procesa. Los datos obtenidos son: para cada *hit*, los valores del transitorio y las variables de la Tabla 4.1.

Tabla 4.1: Datos de emisiones acústicas recogidos.

Variable	**Descripción**
time	Tiempo transcurrido desde el inicio del ensayo en segundos
channel	Número de canal que detecta el *hit*
amplitude	Amplitud pico en voltios
duration	Duración del *hit* en segundos
energy	Energía en eu[1]
rms	Valor cuadrático medio del ruido de fondo al inicio del hit en voltios
rise time	Tiempo desde el primer cruce de umbral hasta la amplitud pico en segundos
counts	Número de veces que la señal cruza el umbral en dirección positiva
trai	Índice del transitorio

[1] 1eu es una unidad arbitraria de energía del programa que se debe escalar con la resistencia del sensor y preamplificador, normalmente $R = 10k\Omega$. $E[J] = \gamma \frac{1}{R} E[eu]$, con $\gamma = 10^{-14}$.

Los datos del transitorio presentan la dificultad de que cada *hit* tiene una duración diferente, lo cual puede ser problemático a la hora de operar con la señal. Para evitar este inconveniente, se recortan todos los transitorios a la duración del que tiene un menor valor. Esta operación puede realizarse sin conllevar una pérdida de información gracias a haber establecido un valor elevado de *post duration* en comparación con la duración típica de los *hits*. Aún con ello, estos datos no pueden utilizarse en bruto, por lo tanto, deben ser procesados para obtener información útil para la clasificación.

4.1. Filtrado de la Señal

Un aspecto fundamental del procesamiento de señales digitales es el proceso de filtrado que trata de transformar la señal de entrada en una de salida con mejores características, utilizando para ello un filtro digital.

Un filtro digital es un sistema que realiza operaciones matemáticas sobre una señal discreta para reducir o potenciar ciertos aspectos de dicha señal [45], dentro del mismo, se encuentran entre otros procedimientos la realización de transformadas de Fourier para pasar el contenido de la señal del espacio temporal al de frecuencia o el uso de filtros para la reducción de ruido.

4.1.1. Ventanas de filtrado para la reducción de *leakage*

Contar con una señal discreta y finita, cuya frecuencia de muestreo es menor que la mínima necesaria, resulta en una distorsión de la señal registrada. Este hecho, conlleva la aparición de componentes de frecuencias ficticias en el espectro de frecuencia y errores de *leakage* que producen una pérdida de información. En la Figura 4.1 se muestra la transformada de una señal, en la que se observa el lóbulo principal, correspondiente a la frecuencia real de la señal y los secundarios generados por las frecuencias espurias.

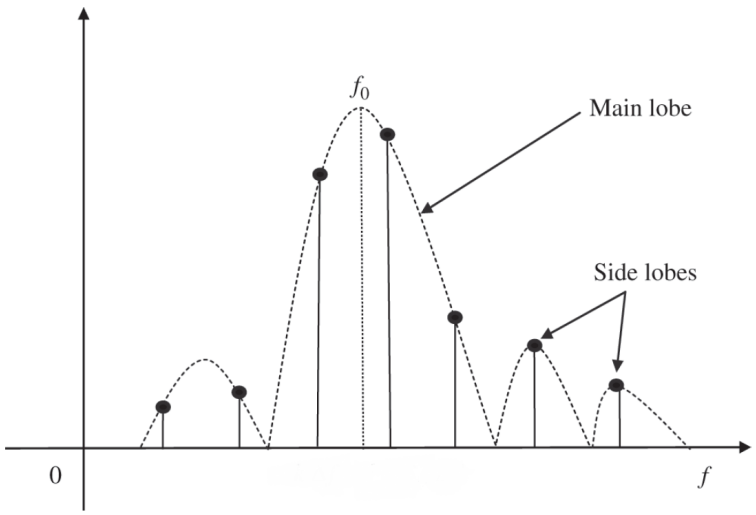

Figura 4.1: Espectro de una señal debido al truncamiento [45].

Para mitigar dicho efecto, se aplica una proceso de filtrado a los transitorios con el uso de ventanas. Dicho proceso consiste en multiplicar la señal de entrada por la correspondiente a la ventana. Estas ventanas cambian la forma se la señal, disminuyendo su amplitud en los extremos. Existen diversas funciones que definen varios tipos de ventanas, entre las más comunes se encuentran: rectangular, triangular, Hanning, Hamming, Blackman y Kaiser. En las figuras 4.2 y 4.3 se muestra la forma en el dominio tiempo y frecuencia de estas ventanas.

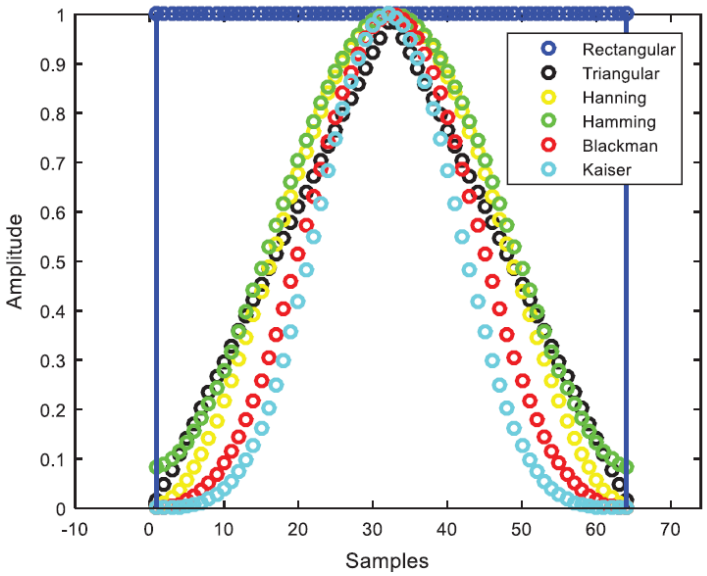

Figura 4.2: Amplitud en el dominio tiempo de varias funciones correspondientes a ventanas [45].

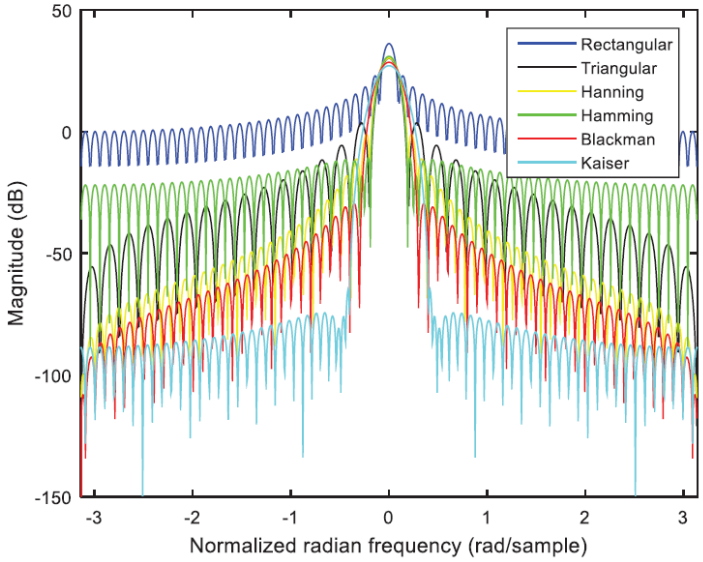

Figura 4.3: Espectro en frecuencia de varias funciones correspondientes a ventanas [45].

En el contexto del análisis de señales, la selección de una ventana adecuada es esencial para lograr una óptima resolución. El objetivo es utilizar una ventana que trate de conseguir que el lóbulo principal de la señal tenga una anchura mínima y amplitud máxima, lo cual se traduce en una mayor resolución ya que una baja resolución puede hacer que la señal se vea fuertemente influenciada por el ruido de fondo. Para ello, se debe reducir la amplitud de los lóbulos secundarios con el objetivo de incrementar la energía del principal.

De todas las ventanas mostradas, la de Kaiser posee parámetros ajustables que proporcionan una mayor flexibilidad en su forma. Esta propiedad, junto con el hecho de que consigue atenuar de mejor forma los lóbulos producidos por efecto de *leakage* hacen que resulte óptima en comparación con el resto [45].

4.1.2. Método Welch

El análisis espectral de la señal extrae la potencia del espectro de la misma, obteniendo información del contenido en frecuencia.

Una forma de estimar la densidad espectral es el periodograma que consiste simplemente en realizar la transformada discreta de Fourier de la señal y calcular su valor absoluto. Este método no resulta un buen estimador espectral ya que su varianza no converge a cero, incluso si la longitud de la señal tiende a infinito (Figura 4.4) [45].

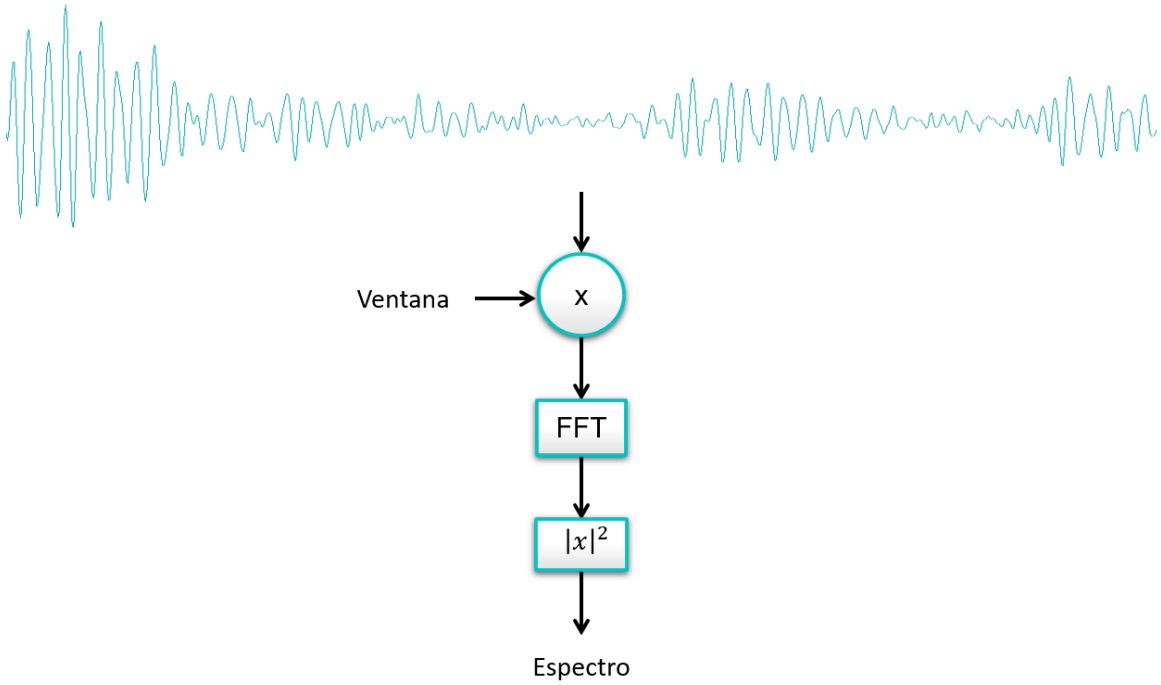

Figura 4.4: Esquema periodograma.

El método Welch es un método mejorado para la estimación de la potencia espectral

basado en el uso del periodograma, pero reduciendo su varianza la la hora de estimar la densidad de potencia espectral. El esquema de este método se muestra en la Figura 4.5.

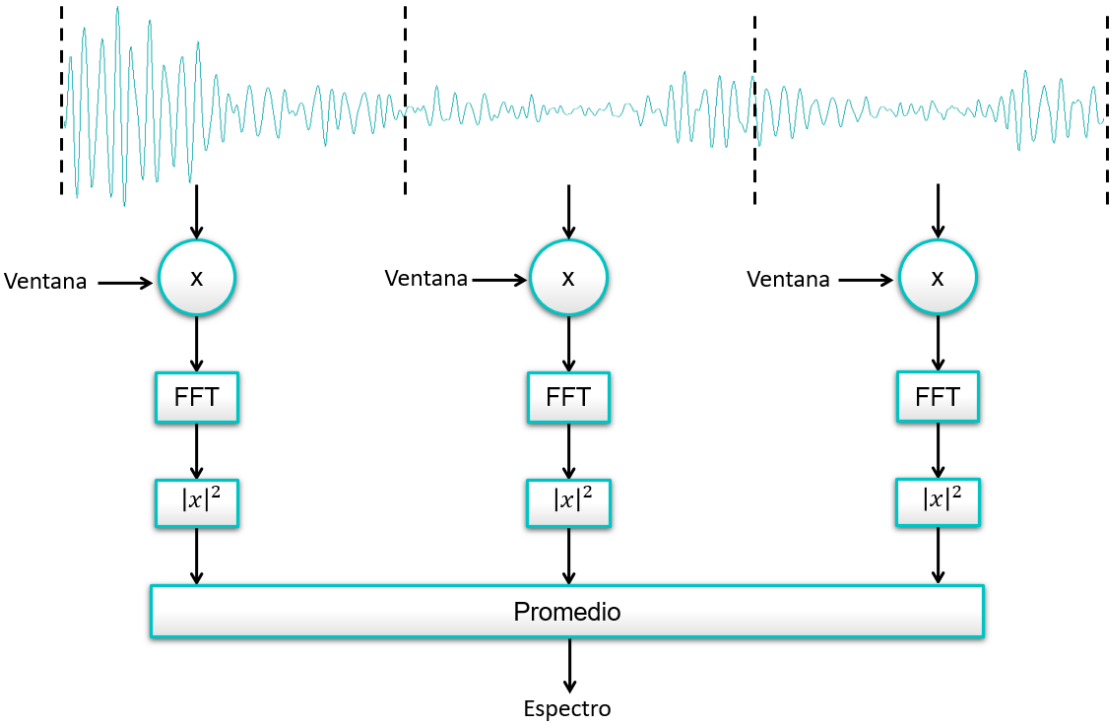

Figura 4.5: Esquema método Welch sin superposición.

El procedimiento empleado para llevar a cabo este método consiste en primer lugar, en dividir la señal original en un número predeterminado de segmentos, los cuales pueden estar superpuestos. A cada uno de ellos, se les aplica una ventana de filtrado, seguido de la realización de la transformada de Fourier. A continuación, se calcula el valor absoluto de los coeficientes obtenidos y se realiza la media de todos los segmentos.

Mediante este proceso se reduce la varianza de las estimaciones de las medidas individuales de potencia ya que se eliminan las frecuencias que no son relevantes en todos los segmentos y se acentúan las que sí. En la Figura 4.6 se muestra un esquema del proceso de eliminación de las frecuencias no relevantes donde las frecuencias que no aparecen en todos los segmentos se eliminan y las que sí se acentúan.

Para el procesamiento de los transitorios obtenidos se elige emplear el método Welch dividiendo la señal en 5 segmentos con una superposición de 0.5 a los que se aplica una ventana tipo Kaiser con $\beta = 10$.

En la Figura 4.7 se pueden observar las diferencias en el espectro de un transitorio elegido al azar dependiendo de la forma de estimación de densidad espectral utilizada.

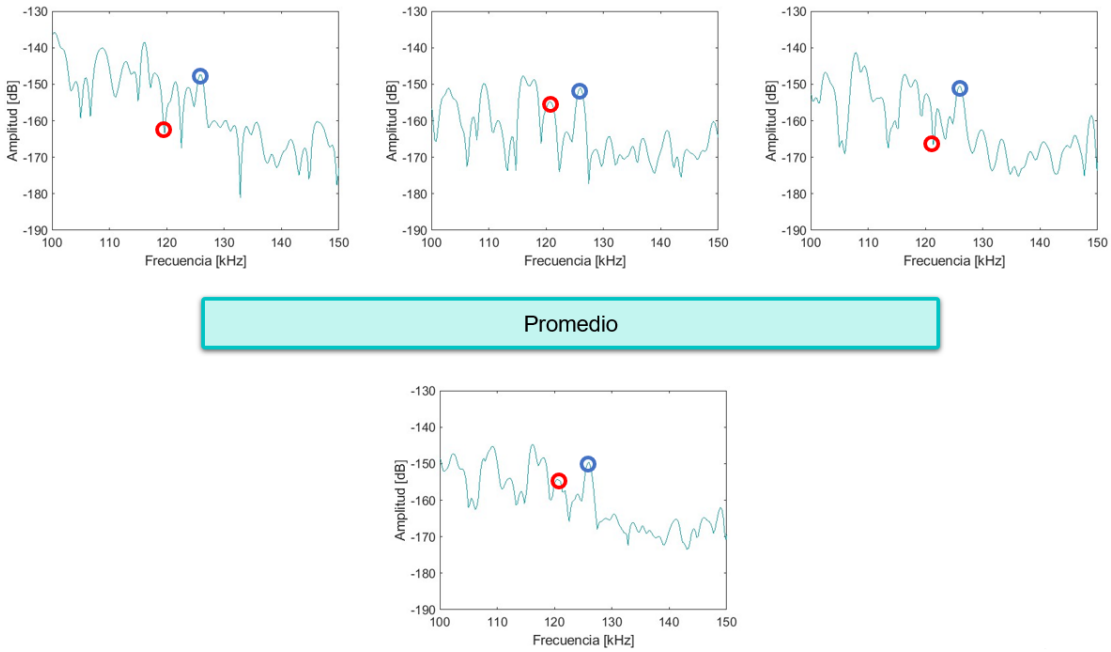

Figura 4.6: Proceso de transformación de la señal mediante el método Welch.

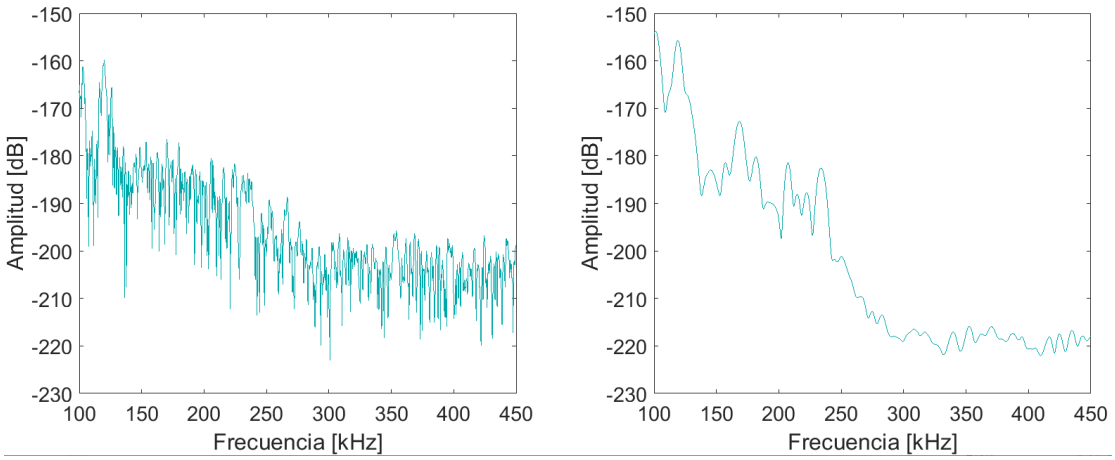

Figura 4.7: Espectro de una misma señal a la que se le aplica la transformada de Fourier (Izquierda) y el método Welch (Derecha).

Capítulo 5

Aplicación de Modelos de Inteligencia Artificial

5.1. Selección de *Features*

Una vez obtenidos los datos y procesados, se pasa a realizar la selección de los parámetros de entrada al modelo.

De los datos descriptores de las señales proporcionados por el software de adquisición, se eliminan aquellos que no proporcionan información útil para la clasificación como son: el tiempo, canal, *rms* y número de transitorio. Se considera que estos no aportan valor a la hora de diferenciar unos *hits* de otros.

En cuanto a los datos obtenidos del transitorio de las señales, se decide coger la ubicación de los 10 picos de frecuencia más relevantes, ordenados de mayor a menor potencia ya que se considera estas frecuencias como características propias de cada uno de los impactos.

Por tanto, los *features* elgidos para la clasificación son 15:

- Amplitud
- Duración
- *Counts*
- *Rise time*
- Energía
- Frecuencias (1 a 10)

5.2. Diseño de Modelos

Para la creación de los modelos de clasificación se hace uso de la aplicación *Classification Learner* de MATLAB. En ella se introducen todos los *features* y se realiza el diseño de cada uno de los modelos disponibles para su posterior análisis.

Se dividen los datos en un 80 % de entrenamiento y un 20 % de test. Además, al disponer de relativamente pocos datos, para aumentar el valor de precisión, se hace uso de la validación cruzada, utilizando un valor de $k = 10$, lo que implica la división

de los datos en 10 subgrupos para el entrenamiento y validación.

Los *features* se estandarizan a la entrada al modelo para evitar que la escala de sus valores influya en la respuesta del modelo.

Los modelos que se eligen para la comparación del rendimiento son: árbol de decisión, análisis discriminante, KNN, *Naive Bayes*, SVM, red neuronal y *ensemble* de árboles de decisión (*Random forest*).

La arquitectura de estos modelos no se establece de antemano sino que se elige una opción disponible en todos los modelos que es la de optimización de hiperparámetros. Con el uso de este recurso, se entrena cada modelo utilizando el set de datos de entrenamiento con varias combinaciones de hiperparámetros, evaluando su desempeño y seleccionando la configuración que presenta una mayor precisión. Finalmente, se utilizan los datos de test para la validación de los modelos.

5.3. Evaluación de Modelos

Las métricas utilizadas para la evaluación de modelos se desarrollan para el caso más sencillo, que es una clasificación binaria. Luego estas se extrapolan y aplican a problemas a las tres clases existentes [23]. En una clasificación binaria, los datos se dividen en las categorías:

- *True Positive* (TP): El modelo clasifica un dato verdadero como verdadero.
- *False Negative* (FN): El modelo clasifica un dato verdadero como falso.
- *True Negative* (TN): El modelo clasifica un dato falso como falso.
- *False Positive*(FP): El modelo clasifica un dato falso como verdadero.

Existen diversos parámetros de evaluación de modelos, entre los más utilizados se encuentran:

- Precisión: Ratio de valores verdaderos entre todos los clasificados como verdaderos.

$$Precisión = \frac{TP}{TP + FP} \tag{5.1}$$

- *Recall* o Sensibilidad: Ratio de valores verdaderos entre los valores realmente verdaderos.

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

- *Accuracy* o Exactitud: Ratio de instancias predichas de forma correcta entre el total.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{5.3}$$

- Tasa de error: Ratio de instancias predichas de forma erronea entre el total.

$$Tasa\ de\ error = \frac{FP + FN}{TP + TN + FN + FP} = 1 - Accuracy \tag{5.4}$$

- F_1 *Score*: Media armónica de la precisión y sensibilidad. De esta forma los valores bajos toman importancia, requiriéndose tanto valores de precisión como de sensibilidad altos para obtener un valor de F_1 *Score* alto. De esta forma, se da la misma importancia a los valores de precisión y exactitud.

$$F_1 = 2 \cdot \frac{Precisión \cdot Recall}{Precisión + Recall} \tag{5.5}$$

Una herramienta comúnmente utilizada para presentar estos datos como resumen de los resultados proporcionados por los algoritmos de clasificación es la matriz de confusión (Figura 5.1) [23], donde lo idoneo sería que todos los datos se encontrasen en la diagonal principal, lográndose en ese caso una clasificación perfecta con una precisión del 100 %.

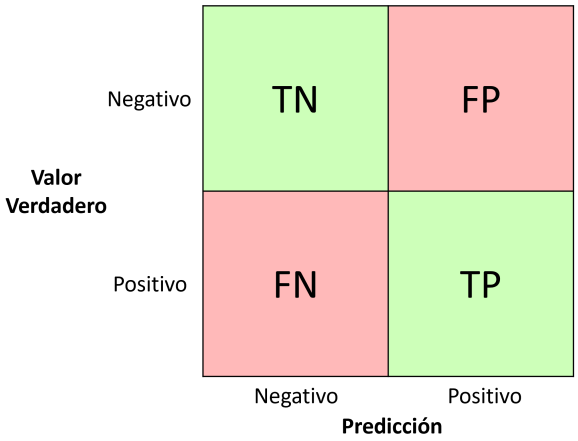

Figura 5.1: Matriz de confusión para un problema binario.

Capítulo 6

Análisis de Resultados

En este apartado se presentan y analizan los resultados más significativos obtenidos de la aplicación de distintos modelos de clasificación de *Machine Learning* a los datos recogidos de emisiones acústicas. De cada modelo se muestran y analizan los resultados de las métricas de evaluación y el tiempo de entrenamiento.

Para el ajuste de los modelos se elige la opción de optimización de hiperparámetros, por lo que éstos no se consideran en el análisis. La app obtiene los que mejor rendimiento presentan tras ejecutar un algoritmo de optimización. Sin embargo, sí que se realizan varios entrenamientos de cada uno de los modelos variando distintos parámetros para ver su influencia y tratar de lograr una mayor eficacia.

6.1. Datos de Emisiones Acústicas

El ensayo se realiza con el objetivo de conseguir una cantidad de datos balanceados ya que si se tiene un número similar de datos de cada tipo de los impactos, este no influirá en los resultados. Tras su realización, el número de datos obtenido se muestra en la Tabla 6.1.

Tabla 6.1: Datos recogidos de emisiones acústicas.

Material	**Número de datos o *hits***
Arena	2720
Esfera metálica	1111
Hielo	1266

Las condiciones físicas de la arena dificultan la tarea de aislar cada impacto, por ello, resulta complicado conseguir un número determinado de datos de esta clase. Para que este hecho no influya en el análisis, se cogen 1111 muestras de cada categoría de forma aleatoria. Este número se corresponde con la clase esfera metálica, de la cual se dispone un menor numero de instancias.

6.2. Modelos de *Machine Learning*

Se decide partir de un caso base y desde el mismo, realizar variaciones para comprobar la influencia de distintas características. Los resultados mostrados se corresponden con los dados por el set de test, datos que no ha visto el modelo y no han sido utilizados para el ajuste de los parámetros.

La funciones resultantes de los modelos entrenados se adjuntan en el Anexo. Con la información disponible en dichas funciones, se pueden realizar predicciones para nuevos datos. Para el resto de casos, la estructura de las funciones es similar, únicamente cambian los valores de los parámetros e hiperparámetros.

6.2.1. Caso base

Para el caso base se emplea lo expuesto en la Capítulo 5. Los *features* de entrada al modelo son: amplitud, duración, energía, *counts*, *rise time* y los 10 primeros picos de frecuencia. Para validación se utiliza el método *K-fold cross validation* con $k = 10$.

En la Tabla 6.2 se muestran las métricas de evaluación por clases y globales para cada uno de los modelos y en la Tabla 6.3 se indican sus tiempos de evaluación.

Analizando las métricas de todos los modelos, el que mejores resultados presenta es el *Random Forest*. Su *accuracy* y *F1 Score* son los más elevados, con un valor de 84.7 %. Además, es también el modelo que presenta menor tasa de error. Por otra parte, en cuanto a los resultados por clases, depende de los modelos, pero de forma general, el hielo y la esfera son los que mayor tasa de error presentan. Esto significa que los modelos son capaces de distinguir bien la clase arena pero se confunden con las clases hielo y esfera metálica. Una posible explicación a este hecho es que pese a ser materiales diferentes, los impactos tienen características similares ya que sus dimensiones y peso son similares en comparación con las de la arena.

Atendiendo a los tiempos de cálculo, los modelos que mejores resultados otorgan son el árbol de decisión y el algoritmo KNN, siendo los peores la Red Neuronal y SVM. En cuanto al modelo que presenta mejores resultados, el *Random Forest*, su tiempo es bastante moderado, resultando este de unos 5 minutos.

De forma general, extraer que los modelos *Naive Bayes* y análisis discriminante, no obtienen muy buenos resultados aunque sí buenos tiempos de ejecución. Tras ellos, los algoritmos de árbol de decisión y KNN presentan muy buenos tiempos de cálculo, inferiores a un minuto, con un *accuracy* de 71.9 % y 77.3 %, valores que se consideran como válidos. A continuación, se observan los modelos de red neuronal y SVM que presentan también un *accuracy* elevado, de 73.7 % y 72.4 % pero tienen tiempos de evaluación mayores, de 62 y 30 minutos respectivamente. Por último, el modelo que presenta mejores resultados con un tiempo moderado y que por tanto debería considerarse como óptimo es el *Random Forest*.

Tabla 6.2: Métricas de evaluación Caso Base.

Modelo	Clase	Precisión	Recall	Accuracy	Tasa de error	F1 Score
Árbol de decisión	Arena	72.3 %	65.8 %	65.8 %	34.2 %	68.9 %
	Hielo	69.8 %	77.9 %	77.9 %	22.1 %	73.6 %
	Esfera	74.1 %	72.1 %	72.1 %	27.9 %	73.1 %
	Global	72.0 %	71.9 %	71.9 %	28.1 %	71.8 %
Análisis Discriminante	Arena	50.0 %	74.3 %	74.3 %	25.7 %	59.8 %
	Hielo	66.7 %	51.4 %	51.4 %	48.9 %	58.0 %
	Esfera	53.9 %	40.1 %	40.1 %	59.9 %	46.0 %
	Global	56.9 %	55.3 %	55.3 %	44.7 %	54.6 %
Naive Bayes	Arena	50.5 %	88.3 %	88.3 %	11.7 %	64.3 %
	Hielo	78.9 %	57.2 %	57.2 %	42.8 %	66.3 %
	Esfera	74.4 %	39.2 %	39.2 %	60.8 %	51.3 %
	Global	67.9 %	61.6 %	61.6 %	38.4 %	60.6 %
SVM	Arena	75.3 %	63.1 %	63.1 %	36.9 %	68.6 %
	Hielo	72.8 %	79.7 %	79.7 %	20.3 %	76.1 %
	Esfera	69.6 %	74.3 %	74.3 %	25.7 %	71.9 %
	Global	72.6 %	72.4 %	72.4 %	27.6 %	72.2 %
KNN	Arena	66.1 %	89.6 %	89.6 %	10.4 %	76.1 %
	Hielo	87.6 %	79.3 %	79.3 %	20.7 %	83.2 %
	Esfera	85.4 %	63.1 %	63.1 %	36.9 %	72.5 %
	Global	79.7 %	77.3 %	77.3 %	22.7 %	77.3 %
Random Forest	Arena	78.2 %	90.5 %	90.5 %	9.5 %	83.9 %
	Hielo	90.7 %	83.8 %	83.8 %	11.2 %	87.1 %
	Esfera	86.8 %	79.7 %	79.7 %	20.3 %	83.1 %
	Global	85.2 %	84.7 %	84.7 %	15.3 %	84.7 %
Red Neuronal	Arena	75.1 %	64.0 %	64.0 %	36.0 %	69.1 %
	Hielo	75.7 %	80.2 %	80.2 %	19.8 %	77.9 %
	Esfera	70.7 %	77.0 %	77.0 %	23.0 %	73.7 %
	Global	73.8 %	73.7 %	73.7 %	26.3 %	73.6 %

Tabla 6.3: Tiempos de ejecución Caso Base.

Modelo	Tiempo de ejecución [s]
Árbol de decisión	53
Análisis Discriminante	65
Naive Bayes	286
SVM	1976
KNN	59
Random Forest	305
Red Neuronal	3784

6.2.2. Selección de *features*

Partiendo del caso base, se realiza una selección de *features*. Se abordan dos orientaciones distintas de entrenamiento de los modelos: uno que considera únicamente los atributos temporales y otro que se enfoca exclusivamente en el contenido en frecuencia. Para el estudio en frecuencia solamente se tiene en cuenta la localización de los 10 primeros picos de frecuencia obtenidos al realizar la transformada tipo Welch de las señales. Los otros 5 *features* (amplitud, energía, duración, *counts* y *rise time*), son los que se utilizan en el estudio temporal.

Las métricas de evaluación se muestran en las tablas 6.4 y 6.5 y los tiempos de validación en las tablas 6.6 y 6.7.

Tabla 6.4: Métricas de evaluación utilizando *features* del contenido en frecuencia.

Modelo	**Clase**	**Precisión**	**Recall**	**Accuracy**	**Tasa de error**	**F1 Score**
Árbol de decisión	Arena	56.2 %	55.4 %	55.4 %	44.6 %	55.8 %
	Hielo	67.8 %	71.2 %	71.2 %	28.8 %	69.5 %
	Esfera	58.9 %	56.8 %	56.8 %	43.2 %	57.8 %
	Global	61.0 %	61.1 %	61.1 %	38.9 %	61.0 %
Análisis Discriminante	Arena	46.5 %	27.0 %	27.0 %	73.0 %	34.2 %
	Hielo	39.7 %	74.3 %	74.3 %	25.7 %	51.7 %
	Esfera	42.1 %	23.0 %	23.0 %	77.0 %	29.7 %
	Global	42.8 %	41.4 %	41.4 %	58.6 %	38.5 %
Naive Bayes	Arena	48.1 %	28.8 %	28.8 %	71.2 %	36.1 %
	Hielo	45.4 %	66.2 %	66.2 %	33.8 %	53.8 %
	Esfera	41.1 %	38.7 %	38.7 %	61.3 %	39.9 %
	Global	44.9 %	44.6 %	44.6 %	55.4 %	43.3 %
SVM	Arena	52.6 %	100.0 %	100.0 %	0.0 %	68.9 %
	Hielo	100.0 %	58.1 %	58.1 %	41.9 %	73.5 %
	Esfera	100.0 %	51.8 %	51.8 %	48.2 %	68.2 %
	Global	84.2 %	70.0 %	70.0 %	30.0 %	70.2 %
KNN	Arena	65.1 %	63.1 %	63.1 %	36.9 %	64.1 %
	Hielo	71.4 %	76.6 %	76.6 %	23.4 %	73.9 %
	Esfera	70.9 %	68.0 %	68.0 %	32.0 %	69.4 %
	Global	69.1 %	69.2 %	69.2 %	30.8 %	69.1 %
Random Forest	Arena	66.3 %	56.8 %	56.8 %	43.2 %	61.2 %
	Hielo	64.9 %	71.6 %	71.6 %	28.4 %	68.1 %
	Esfera	63.6 %	66.2 %	66.2 %	33.8 %	64.9 %
	Global	65.0 %	64.9 %	64.9 %	35.1 %	64.7 %
Red Neuronal	Arena	43.5 %	45.5 %	45.5 %	54.5 %	44.5 %
	Hielo	40.7 %	54.1 %	54.1 %	45.9 %	46.4 %
	Esfera	41.7 %	26.1 %	26.1 %	73.9 %	32.1 %
	Global	42.0 %	41.9 %	41.9 %	58.1 %	41.0 %

Tabla 6.5: Métricas de evaluación utilizando *features* temporales.

Modelo	Clase	Precisión	Recall	Accuracy	Tasa de error	F1 Score
Árbol de decisión	Arena	65.5 %	64.9 %	64.9 %	35.1 %	65.2 %
	Hielo	69.2 %	73.0 %	73.0 %	27.0 %	71.1 %
	Esfera	67.0 %	64.0 %	64.0 %	36.0 %	65.4 %
	Global	67.2 %	67.3 %	67.3 %	32.7 %	67.2 %
Análisis Discriminante	Arena	47.5 %	89.2 %	89.2 %	10.8 %	62.0 %
	Hielo	67.3 %	44.6 %	44.6 %	55.4 %	53.7 %
	Esfera	64.7 %	29.7 %	29.7 %	70.3 %	40.7 %
	Global	59.8 %	54.5 %	54.5 %	45.5 %	52.1 %
Naive Bayes	Arena	48.8 %	87.8 %	87.8 %	12.2 %	62.7 %
	Hielo	65.5 %	50.5 %	50.5 %	49.5 %	57.0 %
	Esfera	69.5 %	29.7 %	29.7 %	70.3 %	41.6 %
	Global	61.2 %	56.0 %	56.0 %	44.0 %	53.8 %
SVM	Arena	73.2 %	67.6 %	67.6 %	32.4 %	70.3 %
	Hielo	74.1 %	65.8 %	65.8 %	34.2 %	69.7 %
	Esfera	64.8 %	77.0 %	77.0 %	23.0 %	70.4 %
	Global	70.7 %	70.1 %	70.1 %	29.9 %	70.1 %
KNN	Arena	77.5 %	74.3 %	74.3 %	25.7 %	75.9 %
	Hielo	75.4 %	84.2 %	84.2 %	15.8 %	79.6 %
	Esfera	80.0 %	73.9 %	73.9 %	26.1 %	76.8 %
	Global	77.6 %	77.5 %	77.5 %	22.5 %	77.4 %
Random Forest	Arena	77.2 %	82.4 %	82.4 %	17.6 %	79.7 %
	Hielo	83.6 %	82.9 %	82.9 %	17.1 %	83.3 %
	Esfera	82.3 %	77.5 %	77.5 %	22.5 %	79.8 %
	Global	81.0 %	80.9 %	80.9 %	19.1 %	80.9 %
Red Neuronal	Arena	68.1 %	72.1 %	72.1 %	27.9 %	70.0 %
	Hielo	70.0 %	70.3 %	70.3 %	29.7 %	70.1 %
	Esfera	73.1 %	68.5 %	68.5 %	31.5 %	70.7 %
	Global	70.4 %	70.3 %	70.3 %	29.7 %	70.3 %

Tabla 6.6: Tiempos de ejecución utilizando *features* del contenido en frecuencia.

Modelo	Tiempo de ejecución [s]
Árbol de decisión	72
Análisis Discriminante	84
Naive Bayes	327
SVM	3526
KNN	141
Random Forest	414
Red Neuronal	610

Tabla 6.7: Tiempos de ejecución utilizando *features* temporales.

Modelo	**Tiempo de ejecución** [s]
Árbol de decisión	45
Análisis Discriminante	57
Naive Bayes	151
SVM	2400
KNN	72
Random Forest	894
Red Neuronal	6827

Por una parte, en el entrenamiento realizado tomando exclusivamente el contenido en frecuencia, se observa un empeoramiento de los resultados. El *accuracy* y *F1 score* de todos los modelos decae entorno a un 10 %, lo cual hace aumentar la tasa de error. Por otro lado, atendiendo a los resultados del entrenamiento con variables temporales, se observa también una disminución generalizada de *accuracy* y *F1 score* pero no tan abrupta, resultando la caída de un 5 % aproximadamente. Las tendencias entre modelos son similares al caso anterior: el que presenta mejores resultados sigue siendo el *Random Forest* seguido del KNN.

En cuanto a los tiempos de entrenamiento son similares al caso base y siguen la misma tendencia de forma general.

De los resultados obtenidos entre categorías, destacar dos valores anómalos para el caso del estudio en el dominio de la frecuencia: hay métricas de evaluación que proporcionan una exactitud del 100 % para ciertas clases del modelo SVM y se observa una gran reducción de *accuracy* y *F1 score* para el caso de la red neuronal.

El valor excepcional obtenido en el modelo SVM se explica observando su matriz de confusión en la Figura 6.1. El algoritmo clasifica de forma correcta todos los datos de arena y los datos que confunde de hielo y esfera metálica siempre los etiqueta como arena. A la vista de este hecho, pese a contar con valores muy buenos en alguna de las métricas no se puede considerar como un buen modelo.

En cuanto a la gran disminución de *accuracy* mostrada por la red neuronal, se explica observando su tiempo de evaluación. En el caso base el tiempo de evaluación de la red neuronal es de 3784 segundos, mientras que para el estudio con los *features* pertenecientes al dominio de la frecuencia es de tan solo 610 segundos. Esta disminución abrupta del tiempo de entrenamiento, induce a pensar que esté motivada por la caída del modelo en un mínimo local. Por tanto, la solución a la que se llega no es la óptima y se debería repetir el entrenamiento iniciando los parámetros del modelo en otros valores.

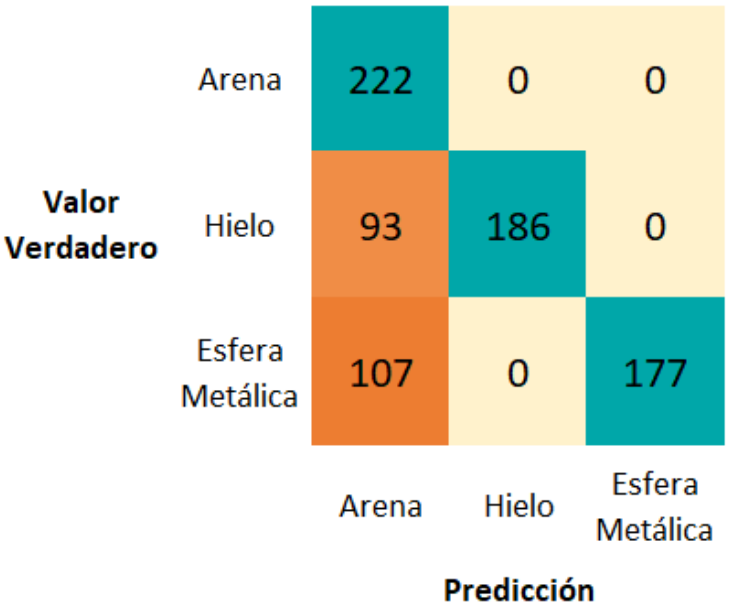

Figura 6.1: Matriz de confusión del modelo SVM utilizando el contenido en frecuencia.

6.2.3. Transformación de *features*

En este caso, se realiza una transformación y reducción de *features* con el método de PCA (*Principal Component Analysis*). Tras la realización del PCA, de las componentes principales extraídas por el análisis se decide coger aquellas que son suficientes para explicar un 95 % de la varianza de los datos. En el diagrama de Pareto mostrado en la Figura 6.2, se observa que se necesitan al menos 12 de estas componentes, quedando fuera del modelo las 3 restantes.

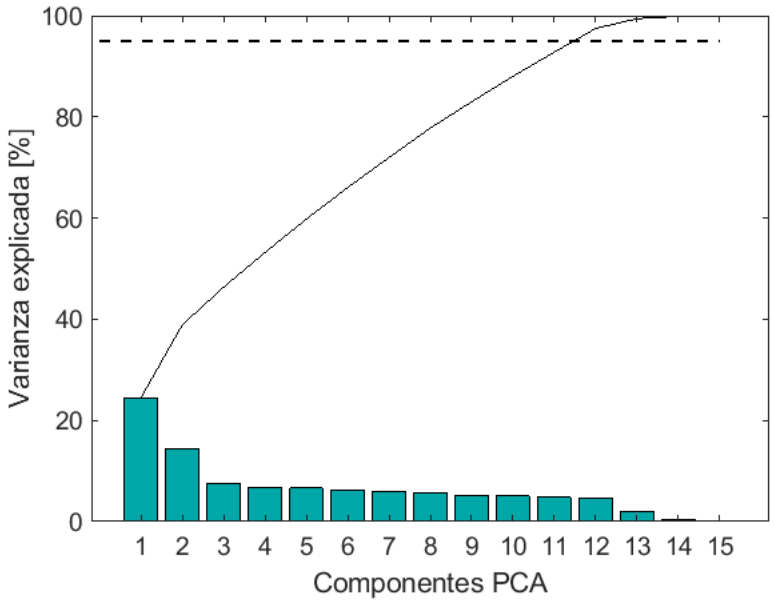

Figura 6.2: Diagrama de Pareto PCA.

Tabla 6.8: Métricas de evaluación caso con PCA.

Modelo	Clase	Precisión	Recall	Accuracy	Tasa de error	F1 Score
Árbol de decisión	Arena	59.2 %	58.1 %	58.1 %	41.9 %	58.6 %
	Hielo	60.1 %	59.0 %	59.0 %	41.0 %	59.5 %
	Esfera	56.5 %	58.6 %	58.6 %	41.4 %	57.5 %
	Global	58.6 %	58.6 %	58.6 %	41.4 %	58.6 %
Análisis Discriminante	Arena	42.5 %	99.1 %	99.1 %	0.9 %	59.5 %
	Hielo	65.3 %	22.1 %	22.1 %	77.9 %	33.0 %
	Esfera	43.8 %	14.4 %	14.4 %	85.6 %	21.7 %
	Global	50.5 %	45.2 %	45.2 %	54.8 %	38.1 %
Naive Bayes	Arena	44.6 %	77.0 %	77.0 %	23.0 %	56.5 %
	Hielo	59.7 %	50.0 %	50.0 %	50.0 %	54.4 %
	Esfera	55.7 %	24.3 %	24.3 %	75.7 %	33.9 %
	Global	53.3 %	50.5 %	50.5 %	49.5 %	48.3 %
SVM	Arena	45.5 %	77.9 %	77.9 %	22.1 %	57.5 %
	Hielo	72.0 %	46.4 %	46.4 %	53.6 %	56.4 %
	Esfera	72.0 %	46.4 %	46.4 %	53.6 %	56.4 %
	Global	63.2 %	56.9 %	56.9 %	43.1 %	56.8 %
KNN	Arena	63.0 %	78.4 %	78.4 %	21.6 %	69.9 %
	Hielo	78.0 %	70.3 %	70.3 %	29.7 %	73.9 %
	Esfera	69.5 %	59.5 %	59.5 %	40.5 %	64.1 %
	Global	70.2 %	69.4 %	69.4 %	30.6 %	69.3 %
Random Forest	Arena	67.1 %	62.6 %	62.6 %	37.4 %	64.8 %
	Hielo	70.0 %	69.4 %	69.4 %	30.6 %	69.7 %
	Esfera	64.4 %	69.4 %	69.4 %	30.6 %	66.8 %
	Global	67.2 %	67.1 %	67.1 %	32.9 %	67.1 %
Red Neuronal	Arena	45.5 %	92.8 %	92.8 %	7.2 %	61.0 %
	Hielo	63.6 %	46.4 %	46.4 %	53.6 %	53.6 %
	Esfera	56.9 %	13.1 %	13.1 %	86.9 %	21.2 %
	Global	55.3 %	50.8 %	50.8 %	49.2 %	45.3 %

Tabla 6.9: Tiempos de ejecución caso PCA.

Modelo	Tiempo de ejecución [s]
Árbol de decisión	59
Análisis Discriminante	74
Naive Bayes	101
SVM	5209
KNN	108
Random Forest	611
Red Neuronal	3327

De los resultados proporcionados por las métricas de evaluación y los tiempos de ejecución se observa una gran reducción de *accuracy* y *F1 score* sin una clara mejora en los tiempos. La tendencia es la misma que para el caso base pero con peores resultados.

Por ello, para los datos analizados, hacer un PCA no aporta beneficios al estudio sino lo contrario. La cantidad de datos no es muy elevada por lo que reducir el número de *features* prácticamente no influye en el tiempo de ejecución pero sí reduce la precisión de forma considerable. Además, en este caso solamente se han eliminado 3 *features*, correspondientes con las componentes que no son necesarias para explicar el 95 % de la variabilidad de los datos. Quizás para otros casos en los que se consiga una reducción mayor o exista un gran número de datos si que resulta útil.

6.2.4. Validación de datos

Se realizan dos ensayos para comprobar la influencia de la forma de validación de datos sobre los resultados. Se parte del caso base, en el cual se realiza una *K-fold cross validation* tomando $k = 10$ y se realizan otros dos entrenamientos de modelos. Por una parte, se lleva a cabo una *holdout validation* separando el 20 % de los datos para test y entrenando el algoritmo con el 80 % restante. Por otra parte, se deseaba hacer un LOOCV, el caso extremo de validación cruzada donde cada uno de los datos se utiliza como test realizando tantas iteraciones como numero de datos. Sin embargo, la app Classification Learner no permite llegar a este límite, por lo que se elige el número máximo de k permitido que es 50.

Los resultados de las métricas de evaluación se muestran en las tablas 6.10 y 6.12 y los tiempos en las tablas 6.11 y 6.13.

Para el caso de *holdout validation* a la vista de la tabla de métricas y tiempos se obtienen muy buenos resultados. Los valores de *accuracy* y *F1 score* son prácticamente iguales o incluso superiores (como en el caso del árbol de decisión y SVM) al caso base. Estos resultados se consiguen con unos tiempos de ejecución notablemente inferiores, especialmente para la red neuronal y SVM donde se consigue prácticamente el mismo resultado en un 10 % del tiempo.

Por contra, en el caso en el que se trata de aproximarse a un LOOCV los resultados no son tan buenos. Las métricas de evaluación son prácticamente similares a las del caso base. Pero, atendiendo a los tiempos de ejecución de los modelos, se observa cómo son notablemente superiores, llegando a tardar más de 1 hora para el entrenamiento del *Random Forest*, cuando el caso base lo hace en 5 minutos e incluso 11 y 6 horas para los modelos SVM y Red Neuronal. Por tanto, incrementar el número de evaluaciones del modelo con diferentes particiones de datos no merece la pena en este caso ya que los resultados obtenidos no son mejores y el tiempo de ejecución se incrementa de forma exponencial.

Tabla 6.10: Métricas de evaluación *holdout validation*.

Modelo	Clase	Precisión	Recall	Accuracy	Tasa de error	F1 Score
Árbol de decisión	Arena	72.6 %	73.0 %	73.0 %	27.0 %	72.8 %
	Hielo	76.1 %	74.8 %	74.8 %	25.2 %	75.5 %
	Esfera	72.0 %	73.0 %	73.0 %	27.0 %	72.5 %
	Global	73.6 %	73.6 %	73.6 %	26.4 %	73.6 %
Análisis Discriminante	Arena	47.8 %	68.9 %	68.9 %	31.1 %	56.5 %
	Hielo	63.9 %	45.5 %	45.5 %	54.5 %	53.2 %
	Esfera	48.9 %	41.4 %	41.4 %	58.6 %	44.9 %
	Global	53.6 %	52.0 %	52.0 %	48.0 %	51.5 %
Naive Bayes	Arena	47.6 %	86.0 %	86.0 %	14.0 %	61.3 %
	Hielo	68.0 %	45.9 %	45.9 %	54.1 %	54.8 %
	Esfera	67.8 %	35.1 %	35.1 %	64.9 %	46.3 %
	Global	61.2 %	55.7 %	55.7 %	44.3 %	54.1 %
SVM	Arena	77.6 %	73.4 %	73.4 %	26.6 %	75.5 %
	Hielo	78.9 %	82.4 %	82.4 %	17.6 %	80.6 %
	Esfera	73.2 %	73.9 %	73.9 %	26.1 %	73.5 %
	Global	76.6 %	76.6 %	76.6 %	23.4 %	76.5 %
KNN	Arena	71.8 %	89.6 %	89.6 %	10.4 %	79.8 %
	Hielo	77.7 %	81.5 %	81.5 %	18.5 %	79.6 %
	Esfera	88.5 %	62.2 %	62.2 %	37.8 %	73.0 %
	Global	79.3 %	77.8 %	77.8 %	22.2 %	77.4 %
Random Forest	Arena	75.1 %	85.6 %	85.6 %	14.4 %	80.0 %
	Hielo	88.7 %	81.1 %	81.1 %	18.9 %	84.7 %
	Esfera	82.9 %	78.4 %	78.4 %	21.6 %	80.6 %
	Global	82.2 %	81.7 %	81.7 %	18.3 %	81.8 %
Red Neuronal	Arena	71.7 %	68.5 %	68.5 %	31.5 %	70.0 %
	Hielo	73.4 %	77.0 %	77.0 %	23.0 %	75.2 %
	Esfera	71.5 %	71.2 %	71.2 %	28.8 %	71.3 %
	Global	72.2 %	72.2 %	72.2 %	27.8 %	72.2 %

Tabla 6.11: Tiempos de ejecución *holdout validation*.

Modelo	Tiempo de ejecución [s]
Árbol de decisión	18
Análisis Discriminante	52
Naive Bayes	78
SVM	205
KNN	35
Random Forest	173
Red Neuronal	318

Tabla 6.12: Métricas de evaluación *K-fold* con $k = 50$.

Modelo	Clase	Precisión	Recall	Accuracy	Tasa de error	F1 Score
Árbol de decisión	Arena	69.6 %	68.0 %	68.0 %	32.0 %	68.8 %
	Hielo	70.3 %	70.3 %	70.3 %	29.7 %	70.3 %
	Esfera	66.5 %	68.0 %	68.0 %	32.0 %	67.3 %
	Global	68.8 %	68.8 %	68.8 %	31.2 %	68.8 %
Análisis Discriminante	Arena	52.1 %	71.2 %	71.2 %	28.8 %	60.2 %
	Hielo	60.3 %	42.3 %	42.3 %	57.7 %	49.7 %
	Esfera	44.9 %	41.9 %	41.9 %	58.1 %	43.4 %
	Global	52.4 %	51.8 %	51.8 %	48.2 %	51.1 %
Naive Bayes	Arena	50.8 %	85.1 %	85.1 %	14.9 %	63.6 %
	Hielo	71.2 %	50.0 %	50.0 %	50.0 %	58.7 %
	Esfera	72.5 %	45.0 %	45.0 %	55.0 %	55.6 %
	Global	64.8 %	60.1 %	60.1 %	39.9 %	59.3 %
SVM	Arena	72.3 %	67.1 %	67.1 %	32.9 %	69.6 %
	Hielo	75.3 %	75.7 %	75.7 %	24.3 %	75.5 %
	Esfera	67.1 %	71.6 %	71.6 %	28.4 %	69.3 %
	Global	71.6 %	71.5 %	71.5 %	28.5 %	71.5 %
KNN	Arena	70.3 %	85.1 %	85.1 %	14.9 %	77.0 %
	Hielo	76.8 %	80.6 %	80.6 %	19.4 %	78.7 %
	Esfera	88.4 %	65.3 %	65.3 %	34.7 %	75.1 %
	Global	78.5 %	77.0 %	77.0 %	23.0 %	76.9 %
Random Forest	Arena	72.6 %	86.0 %	86.0 %	14.0 %	78.8 %
	Hielo	90.3 %	79.3 %	79.3 %	20.7 %	84.4 %
	Esfera	80.8 %	75.7 %	75.7 %	24.3 %	78.1 %
	Global	81.2 %	80.3 %	80.3 %	19.7 %	80.4 %
Red Neuronal	Arena	71.0 %	71.6 %	71.6 %	28.4 %	71.3 %
	Hielo	81.2 %	77.9 %	77.9 %	22.1 %	79.5 %
	Esfera	72.9 %	75.2 %	75.2 %	24.8 %	74.1 %
	Global	75.0 %	74.9 %	74.9 %	25.1 %	75.0 %

Tabla 6.13: Tiempos de ejecución *K-fold* con $k = 50$.

Modelo	Tiempo de ejecución [s]
Árbol de decisión	152
Análisis Discriminante	96
Naive Bayes	707
SVM	41650
KNN	325
Random Forest	4427
Red Neuronal	21988

De todos los casos estudiados, ninguno ha mejorado los valores de *accuracy* y *F1 score* del caso base. Sin embargo, en cuanto a tiempo de ejecución si que se ha llegado a modelos que lo han mejorado. Aunque, por contra, estos han obtenido peores métricas de evaluación. Otro caso han sido modelos que han requerido de una duración de entrenamiento muy superior sin lograr un aumento claro de la exactitud.

Con todo lo anterior, se extrae que el modelo que mejores resultados obtiene es el ***Random Forest*** con los parámetros del caso base, resultando la validación el parámetro menos influyente en los valores de *accuracy* y *F1 score* pero sí en el tiempo de ejecución y la elección de *features* el más influyente y crucial para el correcto ajuste del modelo a los datos.

Estos resultados concuerdan con lo expuesto en la Capítulo 2 donde se menciona que para tareas de clasificación, el algoritmo de *Machine Learning* más potente disponible actualmente es el *Random Forest*.

Capítulo 7

Conclusiones

Con el desarrollo del presente proyecto se ha conseguido obtener un algoritmo de clasificación para identificar distintos impactos sobre una placa plana de material compuesto de fibra de carbono.

Durante el estudio, se ha hecho un análisis de los fundamentos teóricos para comprender los diferentes ámbitos involucrados en el mismo. Con esa base, se han llevado a cabo los experimentos necesarios para la toma de datos. Tras el procesado de estos, se han entrenado varios modelos de inteligencia artificial con diferentes parámetros, llegando al óptimo buscado y extrayendo diversas conclusiones de lo aprendido:

- En base a los resultados obtenidos, se concluye que es posible identificar el tipo de impacto con el uso de emisiones acústicas y algoritmos de inteligencia artificial obteniendo buenos resultados en un periodo corto de tiempo.
- Realizar una correcta elección y tratamiento de los datos de entrada al modelo, es crítico en términos de eficiencia y eficacia.
- El correcto funcionamiento de un modelo depende no sólo de sus hiperparámetros, si no también de otros factores, que cobran mayor o menor relevancia en función del tipo de datos de entrada, lo cual se refleja en los diferentes casos de estudio.
- Se ha determinado el *Random Forest* como el modelo más efectivo.

A continuación, se proponen trabajos futuros que pueden resultar de interés tras la realización de este proyecto como son:

- Utilizar otra transformación de la señal de emisiones acústicas como puede ser la transformada Wavelet para la obtención de imágenes susceptibles de estudio con redes neuronales convolucionales.
- Empleo de algoritmos no supervisados para comparar su rendimiento.
- Evolución a ensayos estáticos o de fatiga de probetas de material compuesto para evaluar fallos reales de delaminacion, despegue o rotura de matriz y/o fibras.

- Extrapolación a cualquier tipo de material, estructura o componente que se desee monitorizar.
- Implementación de los algoritmos para la categorización de datos en tiempo real.

Bibliografía

[1] United States Government Accountability Office, *Status of FAA' s Actions to Oversee the Safety of Composite Airplanes*, sep. de 2011.

[2] R. Slayton y G. Spinardi, "Radical innovation in scaling up: Boeing' s dreamliner and the challenge of socio-technical transitions," *Technovation*, vol. 47, págs. 47-58, ene. de 2016, ISSN: 01664972. DOI: 10.1016/j.technovation.2015.08.004. dirección: https://linkinghub.elsevier.com/retrieve/pii/S0166497215000619 (visitado 17-07-2023).

[3] C. Muir, B. Swaminathan, A. S. Almansour et al., "Damage mechanism identification in composites via machine learning and acoustic emission," *npj Computational Materials*, vol. 7, n.º 1, pág. 95, 24 de jun. de 2021, ISSN: 2057-3960. DOI: 10.1038/s41524-021-00565-x. dirección: https://www.nature.com/articles/s41524-021-00565-x (visitado 17-07-2023).

[4] J. Wang, W. Zhou, X.-y. Ren, M.-m. Su y J. Liu, "A waveform-based clustering and machine learning method for damage mode identification in CFRP laminates," *Composite Structures*, vol. 312, pág. 116 875, mayo de 2023, ISSN: 02638223. DOI: 10.1016/j.compstruct.2023.116875. dirección: https://linkinghub.elsevier.com/retrieve/pii/S0263822323002192 (visitado 17-07-2023).

[5] S. Sikdar, D. Liu y A. Kundu, "Acoustic emission data based deep learning approach for classification and detection of damage-sources in a composite panel," *Composites Part B: Engineering*, vol. 228, pág. 109 450, ene. de 2022, ISSN: 13598368. DOI: 10.1016/j.compositesb.2021.109450. dirección: https://linkinghub.elsevier.com/retrieve/pii/S1359836821008179 (visitado 17-07-2023).

[6] F. Lissek, A. Haeger, V. Knoblauch, S. Hloch, F. Pude y M. Kaufeld, "Acoustic emission for interlaminar toughness testing of CFRP: Evaluation of the crack growth due to burst analysis," *Composites Part B: Engineering*, vol. 136, págs. 55-62, mar. de 2018, ISSN: 13598368. DOI: 10.1016/j.compositesb.2017.10.012. dirección: https://linkinghub.elsevier.com/retrieve/pii/S1359836817313720 (visitado 17-07-2023).

[7] M. Giordano, A. Calabro, C. Esposito, A. D'Amore y L. Nicolais, "An acoustic-emission characterization of the failure modes in polymer-composite materials," *Composites Science and Technology*, vol. 58, n.º 12, págs. 1923-1928, dic. de 1998, ISSN: 02663538. DOI: 10.1016/S0266-3538(98)00013-X. dirección: https:

//linkinghub.elsevier.com/retrieve/pii/S026635389800013X (visitado 17-07-2023).

[8] D. Aljets, A. Chong, S. Wilcox, K. Holford, R. Pullin y M. Eaton, "Classification of Delamination and Matrix Cracking in Carbon Fibre Composite Plates Using Acoustic Emission (AE)," en *Volume 1: 22nd Biennial Conference on Mechanical Vibration and Noise, Parts A and B*, San Diego, California, USA: ASMEDC, 1 de ene. de 2009, págs. 29-36, ISBN: 978-0-7918-4898-2. DOI: 10.1115/DETC2009-86414. dirección: https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings/IDETC-CIE2009/48982/29/343012 (visitado 23-05-2023).

[9] M. Šofer, J. Cienciala, M. Fusek, P. Pavlíček y R. Moravec, "Damage analysis of composite CFRP tubes using acoustic emission monitoring and pattern recognition approach," *Materials*, vol. 14, n.º 4, pág. 786, 7 de feb. de 2021, ISSN: 1996-1944. DOI: 10.3390/ma14040786. dirección: https://www.mdpi.com/1996-1944/14/4/786 (visitado 17-07-2023).

[10] J. P. McCrory, S. K. Al-Jumaili, D. Crivelli et al., "Damage classification in carbon fibre composites using acoustic emission: A comparison of three techniques," *Composites Part B: Engineering*, vol. 68, págs. 424-430, ene. de 2015, ISSN: 13598368. DOI: 10.1016/j.compositesb.2014.08.046. dirección: https://linkinghub.elsevier.com/retrieve/pii/S1359836814003849 (visitado 17-07-2023).

[11] F. E. Oz, N. Ersoy y S. V. Lomov, "Do high frequency acoustic emission events always represent fibre failure in CFRP laminates?" *Composites Part A: Applied Science and Manufacturing*, vol. 103, págs. 230-235, dic. de 2017, ISSN: 1359835X. DOI: 10.1016/j.compositesa.2017.10.013. dirección: https://linkinghub.elsevier.com/retrieve/pii/S1359835X17303718 (visitado 17-07-2023).

[12] R. S. Almeida, M. D. Magalhães, M. N. Karim, K. Tushtev y K. Rezwan, "Identifying damage mechanisms of composites by acoustic emission and supervised machine learning," *Materials & Design*, vol. 227, pág. 111 745, mar. de 2023, ISSN: 02641275. DOI: 10.1016/j.matdes.2023.111745. dirección: https://linkinghub.elsevier.com/retrieve/pii/S0264127523001600 (visitado 17-07-2023).

[13] C. Barile, G. Pappalettera, V. Paramsamy Kannan y C. Casavola, "A neural network framework for validating information–theoretics parameters in the applications of acoustic emission technique for mechanical characterization of materials," *Materials*, vol. 16, n.º 1, pág. 300, 28 de dic. de 2022, ISSN: 1996-1944. DOI: 10.3390/ma16010300. dirección: https://www.mdpi.com/1996-1944/16/1/300 (visitado 17-07-2023).

[14] H. A. Sawan, M. E. Walter y B. Marquette, "Unsupervised learning for classification of acoustic emission events from tensile and bending experiments with open-hole carbon fiber composite samples," *Composites Science and Technology*, vol. 107, págs. 89-97, feb. de 2015, ISSN: 02663538. DOI: 10.1016/j.compscitech.2014.12.003. dirección: https://linkinghub.elsevier.com/retrieve/pii/S0266353814004205 (visitado 17-07-2023).

[15] J. Albelda y E. Giner, *Análisis y diseño con materiales compuestos*, Book Title: Análisis y diseño con materiales compuestos Place: València Series: Apuntes., 2019.

[16] M. Wevers, "Listening to the sound of materials: Acoustic emission for the analysis of material behaviour," *NDT & E International*, vol. 30, n.º 2, págs. 99-106, abr. de 1997, ISSN: 09638695. DOI: 10.1016/S0963-8695(96)00051-5. dirección: https://linkinghub.elsevier.com/retrieve/pii/S0963869596000515 (visitado 24-05-2023).

[17] Z.-X. Zhang, "Stress waves," en *Rock Fracture and Blasting*, Elsevier, 2016, págs. 1-36, ISBN: 978-0-12-802688-5. DOI: 10.1016/B978-0-12-802688-5.00001-4. dirección: https://linkinghub.elsevier.com/retrieve/pii/B9780128026885000014 (visitado 24-05-2023).

[18] V. .-. systeme GmbH, *AE testing (AT) fundamentals - equipment - data analysis (overview)*. dirección: https://www.vallen.de (visitado 24-05-2023).

[19] N. H. H. K. Kyōkai, ed., *Practical acoustic emission testing*, OCLC: ocn947019440, Place of publication not identified: Springer, 2016, 130 págs., ISBN: 978-4-431-55071-6.

[20] V. S. GmbH, "AMSY-6 software operation manual," ago. de 2022.

[21] R. A. E. D. L. LENGUA. "inteligencia | Diccionario de la lengua española," «Diccionario de la lengua española» - Edición del Tricentenario. (), dirección: https://dle.rae.es/inteligencia (visitado 25-05-2023).

[22] Z.-H. Zhou, *Machine learning*, trad. por S. Liu. Singapore: Springer, 2021, 458 págs., ISBN: 9789811519673 9789811519666.

[23] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Second edition. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, 2019, 819 págs., ISBN: 978-1-4920-3264-9.

[24] S. Chandramouli, S. Dutt y A. Dāśa, *Machine Learning*, 1st edition. Pearson Education India, 2018, OCLC: 1138943535.

[25] F. Camastra y A. Vinciarelli, *Machine learning for audio, image and video analysis: theory and applications* (Advanced information and knowledge processing), 2nd edition. London Heidelberg New York Dordrecht: Springer, 2015, 561 págs., ISBN: 978-1-4471-6734-1 978-1-4471-6840-9.

[26] N. M. Seel, ed., *Encyclopedia of the sciences of learning*, Springer reference, New York: Springer, 2012, 7 págs., ISBN: 978-1-4419-1427-9 978-1-4419-1428-6 978-1-4419-5503-6.

[27] S. J. Pan y Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, n.º 10, págs. 1345-1359, oct. de 2010, ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191. dirección: http://ieeexplore.ieee.org/document/5288526/ (visitado 30-05-2023).

[28] I. The MathWorks. "Ayuda y documentación - MATLAB & Simulink - MathWorks España." (), dirección: https://es.mathworks.com/help/matlab/learn_matlab/help.html (visitado 26-06-2023).

[29] D. Soni. "Introduction to k-nearest-neighbors," Medium. (16 de jul. de 2019), dirección: https://towardsdatascience.com/introduction-to-k-nearest-neighbors-3b534bb11d26 (visitado 30-05-2023).

[30] J. Han y M. Kamber, *Data mining: concepts and techniques*, 3rd ed. Burlington, MA: Elsevier, 2012, 703 págs., ISBN: 978-0-12-381479-1.

[31] Y. Dodge, *The concise encyclopedia of statistics* (Springer reference), 1st. ed. New York: Springer, 2008, 616 págs., ISBN: 978-0-387-31742-7 978-0-387-32833-1 978-0-387-33828-6.

[32] T. Mladenova e I. Valova, "Comparative analysis between the traditional K-Nearest Neighbor and Modifications with Weight-Calculation," en *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey: IEEE, 20 de oct. de 2022, págs. 961-965, ISBN: 978-1-66547-013-1. DOI: 10.1109/ISMSIT56059.2022.9932693. dirección: https://ieeexplore.ieee.org/document/9932693/ (visitado 06-07-2023).

[33] I. Rish, "An empirical study of the naive Bayes classifier," en *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Issue: 22, vol. 3, 2001, págs. 41-46.

[34] C. Singla y C. Jindal, "Comparison of various classification models using machine learning to predict mobile phones price range," en *Convergence of Cloud with AI for Big Data Analytics*, D. B. Rawat, L. K. Awasthi, V. E. Balas, M. Kumar y J. K. Samriya, eds., 1.ª ed., Wiley, 2 de mayo de 2023, págs. 401-419, ISBN: 978-1-119-90488-5 978-1-119-90523-3. DOI: 10.1002/9781119905233.ch17. dirección: https://onlinelibrary.wiley.com/doi/10.1002/9781119905233.ch17 (visitado 13-06-2023).

[35] J. Hallinan, "Data mining for microbiologists," en *Methods in Microbiology*, vol. 39, Elsevier, 2012, págs. 27-79, ISBN: 978-0-08-099387-4. DOI: 10.1016/B978-0-08-099387-4.00002-8. dirección: https://linkinghub.elsevier.com/retrieve/pii/B9780080993874000028 (visitado 19-06-2023).

[36] I. The MathWorks. "Machine Learning with MATLAB." (), dirección: https://matlabacademy.mathworks.com/R2022b/portal.html?course=mlml#chapter=3&lesson=5&section=1 (visitado 19-06-2023).

[37] T. Hastie, R. Tibshirani y J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction* (Springer series in statistics), 2nd ed. New York, NY: Springer, 2009, 745 págs., ISBN: 978-0-387-84857-0 978-0-387-84858-7.

[38] C. Henley. "1.1: La neurona," LibreTexts Español. (30 de oct. de 2022), dirección: https://espanol.libretexts.org/Salud/Farmacolog%C3%ADa_y_Neurociencia/Fundamentos_de_la_Neurociencia_(Henley)/01%3A_Estructura_y_funci%C3%B3n_de_las_neuronas/1.01%3A_La_neurona (visitado 06-07-2023).

[39] E. Alpaydin, *Introduction to machine learning* (Adaptive computation and machine learning), Third edition. Cambridge, Massachusetts: The MIT Press, 2014, 613 págs., ISBN: 978-0-262-02818-9.

[40] R. Flórez López y J. M. Fernández Fernández, *Las redes neuronales artificiales: fundamentos teóricos y aplicaciones prácticas*, 1a. ed. Oleiros, La Coruña: Netbiblo, 2008, OCLC: 433872848, ISBN: 978-84-9745-246-5.

[41] V. Naranjo, *Deep Learning aplicado al análisis de señales e imágenes. Fundamentos de Redes Neuronales*, 12 de jul. de 2023.

[42] Z.-H. Zhou, *Ensemble methods: foundations and algorithms* (Machine Learning & Pattern Recognition Series). Boca Raton, Fla.: CRC Press, Taylor & Francis, 2012, 222 págs., ISBN: 978-1-4398-3005-5 978-1-4398-3003-1.

[43] V. S. GmbH, "Acoustic emission sensors and preamplifiers,"

[44] M. Teodorczyk, "Influence of Aggregate Gradation on the Longitudinal Wave Velocity Changes in Unloaded Concrete," *IOP Conference Series: Materials Science and Engineering*, vol. 245, pág. 032 084, oct. de 2017, ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/245/3/032084. dirección: https://iopscience.iop.org/article/10.1088/1757-899X/245/3/032084 (visitado 18-07-2023).

[45] D.-J. Jwo, W.-Y. Chang e I.-H. Wu, "Windowing techniques, the welch method for improvement of power spectrum estimation," *Computers, Materials & Continua*, vol. 67, n.º 3, págs. 3983-4003, 2021, ISSN: 1546-2226. DOI: 10.32604/cmc.2021.014752. dirección: https://www.techscience.com/cmc/v67n3/41590 (visitado 03-07-2023).

Parte II

ANEXOS

Anexo A

Código Fuente

A.1. Extracción de datos

ExtraccionDatos.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 9 11:03:46 2023

@author: crima15c

"""
#%% Borrar datos almacenados en las variables
d = dir()
for obj in d:
    if not obj.startswith('__'):
        del globals()[obj]
#%% #Extraer los datos de los archivos y pasarlos a un csv para su uso en MATLAB

#Librerías necesarias
from pathlib import Path
import vallenae as vae
import pandas as pd

#Ruta de los arch
HERE = Path(__file__).parent if "__file__" in locals() else Path.cwd()

#Función para extraer los datos
def conversion(name):

    #Directorio de los archivos
    name_tra = name + ".tradb"
    name_pri = name + ".pridb"
    TRADB = HERE / name  / name_tra  # Archivo de transitorios
    PRIDB = HERE / name / name_pri   # Archivo de datos

    #Leer archivos
    pridb = vae.io.PriDatabase(PRIDB)
    tradb = vae.io.TraDatabase(TRADB)

    #Extraer descriptores para cada hit
    df_hits = pridb.read_hits()

    #Extraer transitorio de cada hit
    trai=df_hits["trai"].to_numpy()
```

```
    amplitude=[]

    for trans in trai:

        amp, tiempo = tradb.read_wave(trans, time_axis=True) # Leemos el transitorio i-↩
            esimo

        amplitude.append(amp)

    amplitude=pd.DataFrame(amplitude)

    return df_hits, amplitude
#%%

#Extracción de datos de los archivos deseados

data_files = [ "Hielo","Esfera","Arena"]

for i, file_name in enumerate(data_files, start=1):
    data, trai = conversion(file_name)
    data.to_csv(f"data{i}.csv", index=False)
    trai.to_csv(f"trai{i}.csv", index=False)
```

A.2. Procesamiento de la señal

Procesamiento.m

```
%% Limpieza de variables, ventanas y consola

clc
clear all
close all

%% Importar archivos

% Arena --> 1; Hielo --> 2; Esfera Metálica --> 3;

testimport=[1 2 3]; %Introducir el número de test a utilizar

n=length(testimport);

dataEA = cell(1, n);

for i=1:n
    data = readtable(['data', num2str(i), '.csv']);
    dataEA{i}=data;
end

%% Eliminar datos que no proporcionan información útil (Tiempo, canal, param_id, rms, ↩
    trai)

datautil = cell(1, n);
for i=1:n
    a=removevars(dataEA{1,i},{'time','channel','param_id','rms','trai'});
    datautil{1,i}=a;
end

%% Renombrar datos

dataArena=datautil{1,1};
dataHielo=datautil{1,2};
dataEsfera=datautil{1,3};
```

```
%% Añadir los transitorios

%Importar datos tradb

trai = cell(1, n);
for i = 1:n
    trai{i} = readtable(['trai', num2str(i), '.csv']);
    trai{i} = removerows(trai{i}, 1); % Quitar la primera fila
    trai{i} = fillmissing(trai{i}, 'constant', 0); % Rellenar valores NaN
end

%% Eliminar los transitorios que no interesan

for i=1:n
traidata=(dataEA{i}.trai)'; %Transitorios que hay en data que nos interesan
traitrai=1:height(trai{i});  %Vector con el número de transitorio en trai
toNOdeletetrai=ismember(traitrai,traidata);
trai{i}=trai{i}(toNOdeletetrai,:);

end

%% Recortar los transitorios al de menor valor

Min_trai=min(min(width(trai{1}),width(trai{2})),width(trai{3}));

%Datos recortados
trai{1}=trai{1}(:,1:Min_trai);
trai{2}=trai{2}(:,1:Min_trai);
trai{3}=trai{3}(:,1:Min_trai);



%% P welch de las señales

%Señales de Arena

fmuestreo=2*500000;
n=width(trai{1});
nseg=5;
winlen=n/nseg;
window=kaiser(round(winlen),15);
numoverlap = round(0.5*winlen);

potenciasArena=[];
frecuenciasArena=[];

for i=1:height(trai{1})
    [p,freq]=pwelch(table2array(trai{1}(i,:)),window,numoverlap,2^12,fmuestreo);

    potenciasArena=[potenciasArena,p];
    frecuenciasArena=freq;

end

pwrArena = pow2db(potenciasArena);

% Señales de Hielo
fmuestreo=2*500000;
n=width(trai{2});
nseg=5;
winlen=n/nseg;
window=kaiser(round(winlen),15);
numoverlap = round(0.5*winlen);

potenciasHielo=[];
frecuenciasHielo=[];

for i=1:height(trai{2})
    [p,freq]=pwelch(table2array(trai{2}(i,:)),window,numoverlap,2^12,fmuestreo);
```

```
    potenciasHielo=[potenciasHielo,p];
    frecuenciasHielo=freq;

end

pwrHielo = pow2db(potenciasHielo);

% Señales de Esfera
fmuestreo=2*500000;
n=width(trai{3});
nseg=5;
winlen=n/nseg;
window=kaiser(round(winlen),15);
numoverlap = round(0.5*winlen);

potenciasEsfera=[];
frecuenciasEsfera=[];

for i=1:height(trai{3})
    [p,freq]=pwelch(table2array(trai{3}(i,:)),window,numoverlap,2^12,fmuestreo);

    potenciasEsfera=[potenciasEsfera,p];
    frecuenciasEsfera=freq;

end

pwrEsfera = pow2db(potenciasEsfera);

%% Recortar las señales al intervalo del sensor 100-450 kHz

toDeleteArena=frecuenciasArena<100000 | frecuenciasArena>450000;
frecuenciasArena(toDeleteArena,:)=[];
pwrArena(toDeleteArena, :)=[];

toDeleteEsfera=frecuenciasEsfera<100000 | frecuenciasEsfera>450000;
frecuenciasEsfera(toDeleteEsfera,:)=[];
pwrEsfera(toDeleteEsfera, :)=[];

toDeleteHielo=frecuenciasHielo<100000 | frecuenciasHielo>450000;
frecuenciasHielo(toDeleteHielo,:)=[];
pwrHielo(toDeleteHielo, :)=[];

%% Buscar picos de frecuencia Arena

picosArena=[];
localizArena=[];

%Encuentra los 10 primeros picos de frecuencia, sortstr y descend los ordena del mas alto↩
    al más bajo

for i=1:width(pwrArena)
    [peak,loc] = findpeaks(pwrArena(:,i),frecuenciasArena,"NPeaks",10,"SortStr","descend↩
        ");
    picosArena=[picosArena,peak];
    localizArena=[localizArena,loc];
end

%% Buscar picos de frecuencia Esfera

picosEsfera=[];
localizEsfera=[];

%Encuentra los 10 primeros picos de frecuencia, sortstr y descend los ordena del mas alto↩
    al más bajo

for i=1:width(pwrEsfera)
    [peak,loc] = findpeaks(pwrEsfera(:,i),frecuenciasEsfera,"NPeaks",10,"SortStr","↩
        descend");
```

```
    picosEsfera=[picosEsfera,peak];
    localizEsfera=[localizEsfera,loc];
end

%% Buscar picos de frecuencia Hielo

picosHielo=[];
localizHielo=[];

%Encuentra los 10 primeros picos de frecuencia, sortstr y descend los ordena del mas alto↩
     al más bajo

for i=1:width(pwrHielo)
    [peak,loc] = findpeaks(pwrHielo(:,i),frecuenciasHielo,"NPeaks",10,"SortStr","descend↩
        ");
    picosHielo=[picosHielo,peak];
    localizHielo=[localizHielo,loc];
end

%% Añadir la categoría de la localización de los picos
localizArena=localizArena';
localizEsfera=localizEsfera';
localizHielo=localizHielo';

id=ones(height(localizArena), 1);
localizArena=[localizArena,id];
localizArena=array2table(localizArena);
localizArena.Properties.VariableNames={'1','2','3','4','5','6','7','8','9','10','Class'};

id=2*ones(height(localizEsfera), 1);
localizEsfera=[localizEsfera,id];
localizEsfera=array2table(localizEsfera);
localizEsfera.Properties.VariableNames={'1','2','3','4','5','6','7','8','9','10','Class'↩
    };

id=3*ones(height(localizHielo), 1);
localizHielo=[localizHielo,id];
localizHielo=array2table(localizHielo);
localizHielo.Properties.VariableNames={'1','2','3','4','5','6','7','8','9','10','Class'};


%% Unir transitorios y datos

ArenaFin=[localizArena, dataArena];
EsferaFin=[localizEsfera, dataEsfera];
HieloFin=[localizHielo, dataHielo];

%% Recortar los datos al de menor valor

Min_data=min(min(height(ArenaFin),height(EsferaFin)),height(HieloFin));

ArenaFin=datasample(ArenaFin,Min_data);
EsferaFin=datasample(EsferaFin,Min_data);
HieloFin=datasample(HieloFin,Min_data);

%% Unir todos los datos en una variable
DATA=[ArenaFin ;  EsferaFin; HieloFin];
```

A.3. Funciones algoritmos

ArbolDecision.m

```
function [trainedClassifier, validationAccuracy] = ArbolDecision(trainingData)
```

```
% [trainedClassifier, validationAccuracy] = ArbolDecision(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = ArbolDecision(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 12:55:28


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 1486, ...
    'Surrogate', 'off', ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
```

```
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
    'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationTree = classificationTree;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
    yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
    by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map'')↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationTree, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

AnalisisDiscriminante.m

```
function [trainedClassifier, validationAccuracy] = AnalisisDiscriminante(trainingData)
% [trainedClassifier, validationAccuracy] = AnalisisDiscriminante(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
```

```
% T, enter:
%   [trainedClassifier, validationAccuracy] = AnalisisDiscriminante(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 13:02:16


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationDiscriminant = fitcdiscr(...
    predictors, ...
    response, ...
    'DiscrimType', 'linear', ...
    'Gamma', 0, ...
    'FillCoeffs', 'off', ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
discriminantPredictFcn = @(x) predict(classificationDiscriminant, x);
trainedClassifier.predictFcn = @(x) discriminantPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
     'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationDiscriminant = classificationDiscriminant;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
     yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
     by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map''))↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationDiscriminant, 'KFold', 10);
```

```
% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

KNN.m

```
function [trainedClassifier, validationAccuracy] = KNN(trainingData)
% [trainedClassifier, validationAccuracy] = KNN(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = KNN(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 13:00:47


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationKNN = fitcknn(...
```

```
    predictors, ...
    response, ...
    'Distance', 'Cosine', ...
    'Exponent', [], ...
    'NumNeighbors', 1325, ...
    'DistanceWeight', 'SquaredInverse', ...
    'Standardize', true, ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x) knnPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
    'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationKNN = classificationKNN;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
    yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
    by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map''))↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationKNN, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

NaiveBayes.m

```
function [trainedClassifier, validationAccuracy] = NaiveBayes(trainingData)
% [trainedClassifier, validationAccuracy] = NaiveBayes(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
```

```
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = NaiveBayes(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 12:58:36


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.

% Expand the Distribution Names per predictor
% Numerical predictors are assigned either Gaussian or Kernel distribution and ↩
    categorical predictors are assigned mvmn distribution
% Gaussian is replaced with Normal when passing to the fitcnb function
distributionNames =  repmat({'Kernel'}, 1, length(isCategoricalPredictor));
distributionNames(isCategoricalPredictor) = {'mvmn'};

if any(strcmp(distributionNames,'Kernel'))
    classificationNaiveBayes = fitcnb(...
        predictors, ...
        response, ...
        'Kernel', 'Normal', ...
        'Support', 'Unbounded', ...
        'DistributionNames', distributionNames, ...
        'ClassNames', [1; 2; 3]);
else
    classificationNaiveBayes = fitcnb(...
        predictors, ...
        response, ...
        'DistributionNames', distributionNames, ...
        'ClassNames', [1; 2; 3]);
end

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
naiveBayesPredictFcn = @(x) predict(classificationNaiveBayes, x);
```

```
trainedClassifier.predictFcn = @(x) naiveBayesPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
    'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationNaiveBayes = classificationNaiveBayes;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
    yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
    by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map'')↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationNaiveBayes, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

SVM.m

```
function [trainedClassifier, validationAccuracy] = SVM(trainingData)
% [trainedClassifier, validationAccuracy] = SVM(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
```

```
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = SVM(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 12:59:32


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
template = templateSVM(...
    'KernelFunction', 'gaussian', ...
    'PolynomialOrder', [], ...
    'KernelScale', 3.048193218644286, ...
    'BoxConstraint', 576.4890726074755, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsone', ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
     'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
     yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
     by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map'')↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
```

```
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

ANN.m

```
function [trainedClassifier, validationAccuracy] = ANN(trainingData)
% [trainedClassifier, validationAccuracy] = ANN(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = ANN(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 13:01:49


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
```

```
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationNeuralNetwork = fitcnet(...
    predictors, ...
    response, ...
    'LayerSizes', [297 27 112], ...
    'Activations', 'relu', ...
    'Lambda', 1.612290436090357e-06, ...
    'IterationLimit', 1000, ...
    'Standardize', true, ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
neuralNetworkPredictFcn = @(x) predict(classificationNeuralNetwork, x);
trainedClassifier.predictFcn = @(x) neuralNetworkPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
    'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationNeuralNetwork = classificationNeuralNetwork;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
    yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
    by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map'')↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationNeuralNetwork, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

RandomForest.m

```
function [trainedClassifier, validationAccuracy] = RandomForest(trainingData)
% [trainedClassifier, validationAccuracy] = RandomForest(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
```

```
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy as a
%       percentage. In the app, the Models pane displays this overall
%       accuracy score for each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = RandomForest(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Jul-2023 13:01:16


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
template = templateTree(...
    'MaxNumSplits', 248, ...
    'NumVariablesToSample', 'all');
classificationEnsemble = fitcensemble(...
    predictors, ...
    response, ...
    'Method', 'AdaBoostM2', ...
    'NumLearningCycles', 259, ...
    'Learners', template, ...
    'LearnRate', 0.5831878022176471, ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
```

```
trainedClassifier.RequiredVariables = {'1', '10', '2', '3', '4', '5', '6', '7', '8', '9',↩
    'amplitude', 'counts', 'duration', 'energy', 'rise_time'};
trainedClassifier.ClassificationEnsemble = classificationEnsemble;
trainedClassifier.About = 'This struct is a trained model exported from Classification ↩
    Learner R2022b.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n ↩
    yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this ↩
    struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the variables returned↩
    by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector, datatype) must ↩
    match the original training data. \nAdditional variables are ignored. \n \nFor more ↩
    information, see <a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map'')↩
    , ''appclassification_exportmodeltoworkspace'')">How to predict using an exported ↩
    model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'amplitude', '↩
    duration', 'energy', 'rise_time', 'counts'};
predictors = inputTable(:, predictorNames);
response = inputTable.Class;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, ↩
    false, false, false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationEnsemble, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

Anexo B

ODS Agenda 2030

B.1. Objetivos de desarrollo sostenible de la agenda 2030

Se incluye a continuación, una tabla que recoge la relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Tabla B.1: Relación del trabajo con los ODS.

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.	X			
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.			X	
ODS 13. Acción por el clima.				X
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X

Se determina que el trabajo tiene una relación alta con el objetivo 9 de industria, innovación e infraestructuras. El proyecto trata sobre emisiones acústicas y su caracterización mediante IA en materiales compuestos y tal y como se ha visto en el apartado

de introducción, el uso de materiales compuestos está en auge en industrias potentes como son la automovilística y aeronáutica. Además, el uso de inteligencia artificial para el tratamiento de datos es sin duda una innovación. Esta aplicación se une al aumento del uso de técnicas de IA experimentado en diversos ámbitos en la actualidad.

Por otro lado, se determina que tiene una relación baja con el objetivo 12 de producción y consumos responsables. El proyecto, de forma directa no se relaciona, pero puede sentar las bases de una producción responsable de materiales compuestos. Técnicas de caracterización como las utilizadas, podrían emplearse en la monitorización de fallos de las piezas, conllevando el alargamiento de su vida útil y por tanto, se vería controlada su producción.

No se ve relación alguna del trabajo con el resto de objetivos de desarrollo sostenible de la agenda 2030.

Parte III

PLIEGO DE CONDICIONES

Anexo C

Pliego de Condiciones

El desarrollo de todo proyecto esta obligado a cumplir con la regulación vigente. En este apartado, se habla sobre el marco legal y las especificaciones técnicas que incumben a este trabajo.

El objetivo del pliego de condiciones es comprobar que se cumplen las especificaciones técnicas y legales a la hora de ejecutar el TFM, como si de un contrato entre propiedad y contratista se tratase.

C.1. Obligaciones y derechos de los trabajadores

La Ley del estatuto de los trabajadores recogida en el Real Decreto 2/2025 del 23 de octubre recoge las obligaciones y derechos de los trabajadores. Se incluyen los requisitos que deben cumplir tanto los empleados como sus responsables.

C.2. Condiciones del puesto de trabajo

Por una parte, se deben cumplir las disposiciones que establece el BOE-A-1971-380 sobre seguridad e higiene en el trabajo, a fin de prevenir accidentes y enfermedades profesionales y de lograr las mejores condiciones de higiene y bienestar en los centros y puestos de trabajo en que dichas personas desarrollan sus actividades.

En este decreto se incluyen diversos aspectos:

- Seguridad estructural de los edificios en los que se lleva a cabo el trabajo.
- Condiciones mínimas del área de trabajo en cuanto a dimensiones.
- Iluminación adecuada del entorno de trabajo, a ser posible natural.
- Condiciones mínimas y máximas de ventilación, temperatura y humedad.
- Eliminación de ruidos, vibraciones y trepidaciones.

- Abastecimiento de agua potable.
- Requisitos relativos a las instalaciones eléctricas.
- Condiciones relativas al almacenamiento, manipulación y transporte de materias inflamables.
- Medidas de protección contra incendios.
- Medidas de emergencia e información sobre vías y salidas de evacuación.

En cuanto al desarrollo de actividades, la gran mayoría del tiempo se invierte en trabajo con equipos informáticos y redacción. Por ello, se debe cumplir con el Real Decreto 488/1997 de 14 de abril que establece las disposiciones mínimas de seguridad y de salud para la utilización por los trabajadores de equipos que incluyan pantallas de visualización. También se incluye una parte de ensayos experimentales, regulada por la Ley 31/1995, de 8 de noviembre, de prevención de riesgos laborales y el Real Decreto 1215/1997, de 18 de julio, por el que se establecen las disposiciones mínimas de seguridad y salud para la utilización por los trabajadores de los equipos de trabajo.

En cuanto a las condiciones del puesto de trabajo que establece el Real Decreto 488/1997, se proporciona información relativa a los siguientes aspectos:

- Ergonomía del puesto de trabajo
- Equipo de trabajo (Pantalla, teclado, superficie de trabajo y asiento)
- Entorno (Espacio, iluminación, reflejos y deslumbramientos, ruido, calor, emisiones y humedad)
- Interconexión ordenador/persona

Existen aspectos en los que se solapan ambas normativas, en esos casos se debe cumplir con lo establecido en ambas.

C.3. Condiciones del teletrabajo

La aparición de la COVID-19 y la pandemia que causó en 2020 potenció el trabajo a distancia. Esta situación se regula por el Real Decreto-Ley 28/2020 de 22 de septiembre, de trabajo a distancia.

Los aspectos más importantes que incluye son: la igualdad de derechos y condiciones entre empleados a distancia y en el lugar de trabajo; el derecho a la desconexión digital para garantizar el equilibrio entre vida laboral y personal; la obligación del empleador de cubrir los gastos relacionados con el trabajo remoto; la necesidad de un acuerdo escrito para establecer el trabajo a distancia y también se promueve la formación adecuada para empleados y empleadores sobre el uso de tecnologías de la información.

C.4. Condiciones y especificaciones técnicas

Conocimientos requeridos

El presente trabajo, en cuanto al ámbito académico, requiere por una parte de los conocimientos técnicos adquiridos durante el grado y posterior máster en materias como: análisis, diseño y fabricación con materiales compuestos; informática; ciencia de materiales; resistencia de materiales y mecánica entre otras. Además se ha requerido formación adicional en el campo de emisiones acústicas, procesamiento de señales, inteligencia artificial y programación.

Equipo informático

En cuanto a condiciones técnicas de hardware y software, al ser un trabajo didáctico no se ha requerido más de lo que se detalla a continuación. Sin embargo, en el caso de continuar con el ámbito de la inteligencia artificial, se requerirían condiciones técnicas de hardware y software más potentes capaces de albergar un gran número de datos y de entrenar modelos complejos.

El equipo informático empleado para la elaboración del proyecto ha sido un ordenador portátil MSI GF63 Thin 10SCXR-405XES. Se trata de un equipo personal, capaz de cumplir con los requerimientos del trabajo. Sus especificaciones son:

Tabla C.1: Especificaciones Equipo informático.

Modelo:	MSI GF63 Thin 10SCXR-405XES
Procesador:	Intel® Core i7-10750H+HM470
Almacenamiento:	512GB NVMe PCIe Gen3x4 SSD
Memoria:	DDR IV 16GB (2666MHz)
Controlador gráfico:	GeForce® GTX 1650 MAX Q, GDDR6 4GB
Sistema operativo:	Windows 10 Home

En cuanto a software, para llevar a cabo el trabajo se han utilizado los programas necesarios para realizar cada una de las tareas involucradas en el desarrollo del proyecto. Entre ellos se encuentran:

- LinWave acquisition: software de la empresa Vallen Systeme GmbH para la interacción con el equipo de emisiones acústicas y toma de datos.
- Vallen VisualAE: software de la empresa Vallen Systeme GmbH para la visualización de los datos de emisiones acústicas recogidos.
- Anaconda Navigator: software libre, gestor de paquetes en python. Incluye el entorno spyder donde se realiza la extracción de datos del formato otorgado por el programa de adquisición a un formato legible por MATLAB.

- MATLAB: Entorno de programación de MathWorks, se ha utilizado para el procesamiento de datos y la creación y entrenamiento de los modelos de IA con el uso de la aplicación Classification Learner.
- Microsoft 365®: Se ha hecho uso de programas como Powerpoint y Excel para la creación y edición de imágenes y gráficas.
- Overleaf: Herramienta de código libre para la redacción *online* de textos utilizando el formato LaTeX.
- Zotero: Aplicación de gestión de referencias bibliográficas de código abierto y gratuito, diseñada para organizar, almacenar y citar fuentes de información.
- Otros: Se han utilizado diversos navegadores para la búsqueda de información y lectores como Adobe Acrobat Reader para el análisis de textos.

Para el empleo de aquellos *softwares* que no son de código abierto se ha hecho uso de las licencias institucionales con las que cuenta la UPV.

Conexión a internet

El desarrollo del trabajo, ha implicado la búsqueda de información y tratamiento de datos, procesos que se realizan mayoritariamente *online*. Para ello, se ha utilizado la red disponible en la UPV y la conexión a escritorio remoto por VPN cuando no se estaba en el campus.

C.5. Control de calidad

Los materiales empleados para la realización de los ensayos deben cumplir las especificaciones dadas por el fabricante y satisfacer los mínimos de calidad requeridos por las normas UNE (Una Norma Española) para cada uno de los componentes.

C.6. Supervisión

El trabajo se realiza bajo la supervisión de dos profesores tutores de la UPV. Expertos en la ejecución de proyectos y trabajos de investigación de diversa temática similares al realizado. Su supervisión sirve de gran ayuda a la hora de realizar el seguimiento del trabajo, de resolver dudas y de proponer mejoras.

Parte IV

PRESUPUESTO

Anexo D

Presupuesto

En esta sección, se realiza una estimación de los costes involucrados en el desarrollo del proyecto. En el presupuesto se incluye tanto los costes del personal como el uso de hardware y software necesarios. A esto se añade el 21 % de IVA (Impuesto sobre el Valor Añadido) vigente en España.

D.1. Desglose de costes

Coste de personal

Los costes de personal incluyen los gastos del estudiante que esta completando sus estudios llevando a cabo el presente proyecto y de los profesores que lo tutorizan. Se considera que los tutores dedican parte de sus horas de trabajo al proyecto y por tanto, este forma parte de su sueldo.

Para asignar las retribuciones de cada uno de los miembros involucrados: al alumno se asigna el sueldo correspondiente a un técnico superior de investigación; a uno de los tutores, el salario de un profesor ayudante doctor y al otro la de un catedrático. En cuanto a las horas, se estiman en función de las horas de dedicación que requieren los 13.5 créditos ECTS que se otorgan al proyecto.

Tabla D.1: Desglose de costes de personal.

Concepto	**Tiempo**	**Coste unitario [€/h]**	**Importe [€]**
Autor	350	10.17	3559.50
Tutor Profesor	25	14.63	365.75
Tutor Catedrático	15	24.16	362.40
		Total	**4287.65**

Coste de equipo de emisiones acústicas

Para la adquisición de las emisiones acústicas se ha hecho uso de hardware y software específicos adquiridos para tal fin. Se estima su vida útil en unos 5 años, por lo que se

calcula la amortización equivalente.

Tabla D.2: Desglose de costes equipo de emisiones acústicas.

Concepto	**Periodo de amortización** [Meses]	**Coste total [€]**	**Importe [€]**
Linwave 1002	6	9513	951.3
PoE	6	328	32.8
Cable Ethernet	6	112	11.2
2 Sensores VS150M	6	572	57.2
2 Cable Sensor	6	112	11.2
Software Adquisición	6	3109	31.09
Formación	-	225	225
		Total	**1622.10**

Coste de equipo informático y licencias

En cuanto a hardware, se especifica el coste de los equipos utilizados teniendo en cuenta al igual que en caso anterior una vida útil de 5 años. De la parte de software, se detallan los programas utilizados que no son de código abierto. Existen diversos tipos de licencia, en función del desarrollador, por lo que se utiliza el coste equivalente mensual de cada una de ellas.

Tabla D.3: Desglose de costes hardware.

Equipo	**Periodo de amortización** [Meses]	**Coste total [€]**	**Importe [€]**
MSI GF63 Thin 10	6	778,85	77.89
Pantalla BenQ 24”	6	95.87	9.59
Teclado + Ratón	6	27	2.7
		Total	**90.17**

Tabla D.4: Desglose de costes software.

Programa	**Tiempo** [Meses]	**Coste unitario** [€/Mes]	**Importe [€]**
Windows 10 ©	6	12.08	72.5
MATLAB ®	6	71.67	430
Microsoft Office 365 ©	6	5.75	34.5
		Total	**537**

D.2. Coste total

Realizando la suma de todos los costes parciales, se establece el presupuesto global del proyecto.

Tabla D.5: Presupuesto global.

Concepto	**Importe [€]**
Personal	4287.65
Equipo EA	1622.10
Equipo Informático	90.17
Software	537
Subtotal	6536.92
IVA (21 %)	1372.75
Total	**7909.67**

Por tanto, el coste global del proyecto asciende a un total de:

SIETE MIL NOVECIENTOS NUEVE EUROS CON SESENTA Y SIETE CÉNTIMOS