Universidad Europea

UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

MÁSTER UNIVERSITARIO EN ANALISIS DE DATOS MASIVOS (BIG DATA)

TRABAJO FIN DE MÁSTER

Impacto de la frecuencia de muestreo en la precisión de modelos de clasificación para reconocimiento de actividad humana

BRYAN JESÚS LÓPEZ GONZÁLEZ

Dirigido por

Dr. Sergio Iglesias-Pérez

CURSO 2024-2025

TÍTULO: Impacto de la frecuencia de muestreo en la precisión de modelos de clasificación para reconocimiento de actividad humana

AUTOR: Bryan Jesús López González

TITULACIÓN: MÁSTER UNIVERSITARIO EN ANALISIS DE DATOS MASIVOS (BIG DATA)

DIRECTOR/ES DEL PROYECTO: Dr. Sergio Iglesias-Pérez

FECHA: Septiembre de 2025

Resumen

Este trabajo presenta el diseño e implementación de un pipeline reproducible en Python para la evaluación de modelos de clasificación de actividades humanas a partir de señales inerciales. Se utilizaron clasificadores *ensemble* basados en árboles de decisión y una red neuronal convolucional, comparando enfoques que emplean características estadísticas frente al uso de la señal cruda. Se implementó un proceso de reducción progresiva de la frecuencia de muestreo para analizar el impacto en métricas como *accuracy* y *f1-score*. Los resultados muestran que las representaciones estadísticas ofrecen mayor robustez ante disminución de datos, mientras que el uso de la señal cruda puede simplificar la inferencia pero es más sensible a la pérdida de resolución temporal. Este estudio proporciona criterios para seleccionar configuraciones que equilibren rendimiento y eficiencia en escenarios con restricciones computacionales, como sistemas embebidos o dispositivos IoT.

Palabras clave: Reconocimiento de actividades humanas, sensores inerciales, *machine learning*, redes neuronales convolucionales, frecuencia de muestreo.

Abstract

This work presents the design and implementation of a reproducible Python pipeline for evaluating human activity classification models from inertial sensor data. Decision tree–based ensemble classifiers and a convolutional neural network were used, comparing approaches that employ statistical features versus raw signal processing. A progressive sampling rate reduction process was implemented to assess the impact on metrics such as accuracy and f1-score. Results show that statistical representations provide greater robustness under reduced data scenarios, while raw signal processing can simplify inference but is more sensitive to temporal resolution loss. This study offers criteria for selecting configurations that balance performance and efficiency in computationally constrained environments, such as embedded systems or IoT devices.

Keywords: Human activity recognition, inertial sensors, machine learning, convolutional neural networks, sampling rate.

AGRADECIMIENTOS

A mis padres: Betsy y David, y a mi novia Violeta. Por creer en mí, apoyarme y siempre motivarme a dar más de mí mismo.

"No te preocupes por lo que otros dicen de ti. Preocúpate por lo que tú piensas de ti mismo"

Marco Aurelio

Tabla Resumen

	DATOS
Nombre y apellidos:	Bryan Jesús López González
Título del proyecto:	Impacto de la frecuencia de muestreo en la precisión de modelos de clasificación para reconocimiento de actividad humana
Directores del proyecto:	Sergio Iglesias-Pérez
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto:	NO
El proyecto ha consistido en el desarrollo de una investigación o innovación:	SI
Objetivo general del proyecto:	Estudiar el efecto de la reducción de la frecuencia de muestreo en la precisión de modelos de clasificación para reconocimiento de actividad humana, mediante un flujo de trabajo reproducible que permita evaluar distintos algoritmos de forma estructurada.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. Introducción

El crecimiento exponencial del Internet de las Cosas (IoT) ha transformado la manera en que interactuamos con la tecnología en nuestra vida cotidiana. Se espera que para el año 2030, el número de dispositivos IoT conectados alcance los 40 mil millones a nivel mundial [1]. Esta proliferación de dispositivos ha impulsado el desarrollo de aplicaciones en diversos sectores, desde la automatización industrial hasta la salud y el bienestar personal. En este contexto, el reconocimiento de actividad humana (HAR, por sus siglas en inglés) se ha consolidado como una técnica esencial para monitorear y analizar comportamientos humanos mediante sensores incorporados en dispositivos móviles y relojes inteligentes. Las aplicaciones de HAR abarcan desde el seguimiento de la actividad física en dispositivos de fitness hasta la supervisión de pacientes en entornos de atención médica [2]. Con el objetivo de evitar las latencias y consumos energéticos asociados a la conectividad con recursos en la nube, los cuales exhiben mayor capacidad de cómputo, se busca implementar estos clasificadores dentro de los mismos dispositivos que tienen los sensores, sin embargo, la implementación eficiente de estas aplicaciones en dispositivos con recursos limitados, especialmente en el caso de los microcontroladores, presenta desafíos significativos [3].

Uno de los principales retos en este ámbito es la necesidad de procesar grandes volúmenes de datos generados por sensores a altas frecuencias de muestreo, lo que puede comprometer la autonomía y la capacidad de procesamiento en sistemas embebidos [4]. Reducir la frecuencia de muestreo podría aliviar estas limitaciones, pero es crucial entender cómo esta reducción afecta la precisión de los modelos de clasificación utilizados en HAR [5]. Este proyecto surge de la experiencia previa del autor en sistemas embebidos, aplicaciones de IoT y Edge Computing. Durante su formación de pregrado, desarrolló un clasificador KNN en MATLAB para detectar actividades humanas básicas y durante su ejercicio profesional, ha desplegado redes neuronales en microcontroladores de 32 bits para diversas aplicaciones. Estas experiencias, combinadas con el interés en optimizar el rendimiento de modelos de HAR en dispositivos con recursos limitados, motiva esta investigación.

Uno de los desafíos más significativos en el reconocimiento de actividades humanas (HAR) es la alta variabilidad intrínseca de las señales capturadas por sensores portátiles. Factores como la heterogeneidad de los usuarios, las diferencias en la ejecución de una misma actividad y las condiciones de uso de los dispositivos introducen una notable variación en los patrones registrados. A esto se suma el reto de gestionar la naturaleza secuencial y no estacionaria de los datos de sensores, lo que complica la segmentación temporal y la extracción de características relevantes. Otro obstáculo clave es la escasez de generalización de los modelos: un clasificador que funciona bien en un grupo de usuarios o en un contexto específico puede perder precisión al trasladarse a escenarios diferentes. Además, las limitaciones computacionales de los dispositivos portátiles restringen la complejidad de los algoritmos que pueden implementarse de manera eficiente. Como resultado, el HAR enfrenta una tensión constante entre la necesidad de precisión y la eficiencia en entornos con recursos restringidos [6].

En los últimos años, la investigación en reconocimiento de actividades humanas ha evolucionado desde un enfoque centralizado en la nube hacia arquitecturas distribuidas en el borde (*edge computing*). Esta transición responde a la necesidad de minimizar latencias críticas y reducir el consumo energético asociado al envío continuo de datos, permitiendo que los modelos se ejecuten directamente en dispositivos cercanos a la fuente de datos. Dicho cambio no solo impacta la eficiencia técnica, sino que también mejora la escalabilidad del IoT, pues evita la saturación de infraestructuras centralizadas. En este sentido, el **procesamiento en local** adquiere una relevancia central dentro del reconocimiento de actividades humanas. A diferencia del enfoque basado en la nube, que requiere transmitir continuamente grandes volúmenes de datos a servidores remotos, la ejecución de los algoritmos directamente en los dispositivos reduce drásticamente la latencia y permite respuestas en tiempo real, algo crítico en aplicaciones como la detección de caídas, la monitorización de pacientes o la seguridad en entornos industriales. Además, el procesamiento en local incrementa la **privacidad de los usuarios**, ya que la información sensible de movimiento o salud no necesita salir del dispositivo. Otro beneficio clave es la **eficiencia energética**, pues la transmisión inalámbrica de datos suele consumir más energía que el cálculo interno en un microcontrolador optimizado. De esta forma, el procesamiento embebido no solo garantiza un funcionamiento más autónomo y sostenible, sino que también facilita la escalabilidad del IoT al reducir la dependencia de infraestructuras externas [7].

Como ya ha sido mencionado, La eficiencia energética también emerge como un factor determinante en el diseño de sistemas HAR en IoT. Estudios recientes muestran que la transmisión inalámbrica de datos consume considerablemente más energía que el cómputo local, lo cual afecta directamente la autonomía de dispositivos portátiles o embebidos. En este contexto, procesar en el propio microcontrolador (o cualquier arquitectura embebida) no solo reduce el gasto energético, sino que también contribuye a la sostenibilidad de las soluciones, permitiendo una mayor autonomía y menor necesidad de recarga en aplicaciones críticas. Otro aspecto clave que motiva el procesamiento en local es la preservación de la privacidad. Los sensores inerciales capturan información íntimamente ligada a los hábitos, rutinas y, en ocasiones, a la salud de los usuarios. Transmitir estos datos a la nube implica riesgos de seguridad y de cumplimiento normativo, especialmente en ámbitos como la salud digital. Ejecutar los algoritmos de reconocimiento en el propio dispositivo evita exponer continuamente estos datos a redes externas, incrementando la confianza y seguridad de las soluciones IoT.

El mercado de dispositivos wearables está experimentando un crecimiento vertiginoso impulsado tanto por consumo general como por aplicaciones médicas. Los smartwatches, gafas inteligentes y rastreadores de fitness están liderando esta tendencia, al tiempo que el sector salud demanda dispositivos capaces de monitorear funciones corporales de forma discreta y continua. Este apetito tecnológico se enfrenta a una clara restricción: el tamaño físico del dispositivo limita drásticamente la capacidad de la batería. Dado que los sensores MEMS (como acelerómetros y giróscopos) generan una gran cantidad de datos a alta frecuencia, su operación constante se traduce en un notable consumo energético. Por lo tanto, los microcontroladores (MCUs) sin duda constituyen el subsistema electrónico más importante dentro del wearable, pues deben ser compactos, eficientes en energía y suficientemente versátiles para gestionar

adquisición de señal, procesamiento y comunicación inalámbrica . Esta realidad técnica resalta por qué optimizar procesos como la frecuencia de muestreo es crítico: al reducir la tasa a la que los sensores recolectan datos, es posible disminuir el volumen de procesamiento y transmisión, aliviando así la carga energética y prolongando la autonomía del dispositivo sin sacrificar su funcionalidad [8].

La literatura sobre HAR ha mostrado una tendencia a utilizar datasets con frecuencias de muestreo relativamente altas (50–200 Hz), procesados mayoritariamente en servidores o GPUs, con el fin de capturar microvariaciones del movimiento y maximizar el rendimiento de los modelos. Sin embargo, estas configuraciones no siempre reflejan las restricciones de los dispositivos embebidos, donde el consumo energético y la autonomía son factores críticos. De hecho, la reducción de la frecuencia de muestreo implica un compromiso entre eficiencia y precisión, pues si bien disminuye el volumen de datos y el gasto energético, también puede degradar la capacidad de los modelos para diferenciar actividades de mayor complejidad motriz, como subir o bajar escaleras. A pesar de ello, existen pocos estudios que analicen de manera sistemática cómo se comportan los modelos de HAR bajo una reducción progresiva de la frecuencia en entornos de cómputo limitado, lo que constituye una oportunidad de investigación relevante para el ámbito de IoT y sistemas embebidos [9].

El objetivo principal de este trabajo es estudiar el efecto de la reducción de la frecuencia de muestreo en la precisión de modelos de clasificación para reconocimiento de actividad humana. Para ello, buscando implementar un flujo de trabajo reproducible que permita evaluar distintos algoritmos de forma estructurada, facilitando su posible despliegue futuro en entornos de ejecución más escalables. Los resultados de esta investigación podrían contribuir al diseño de soluciones más eficientes y sostenibles en aplicaciones de HAR, especialmente en escenarios donde los recursos computacionales y energéticos son limitados. Para el desarrollo de este trabajo, se hará uso del dataset *Heterogeneity Human Activity Recognition* del *UC Irvine Machine Learning Repository*, el cual se compone de aproximadamente 44 millones de muestras [10].

Este nivel de magnitud convierte al dataset en un caso representativo de *Big Data*, en el que las técnicas de procesamiento tradicionales pueden resultar insuficientes tanto en tiempo como en consumo de memoria. En este sentido, se optó por la utilización de **Dask**, una herramienta de procesamiento paralelo en Python diseñada para manejar grandes volúmenes de datos distribuidos en múltiples particiones. El uso de Dask no solo permite realizar operaciones sobre el conjunto completo sin necesidad de cargarlo en memoria de manera secuencial, sino que también proporciona un ecosistema compatible con bibliotecas como *scikit-learn* y *pandas*, facilitando la transición hacia un pipeline de análisis escalable. De esta forma, la metodología implementada en este trabajo no solo aborda la pregunta de investigación, sino que también establece una infraestructura flexible y eficiente que puede ser replicada o extendida en proyectos futuros de reconocimiento de actividad humana sobre datasets masivos.

Capítulo 2. Objetivos

En este capítulo se presentan el objetivo general del proyecto y los objetivos específicos planteados de manera que, la consecución de estos últimos, comprenden el alcance planteado del primero. También se incluye un apartado que condensa los potenciales beneficios de este proyecto desde el punto de vista académico y práctico.

2.1 Objetivos generales

El objetivo general de este trabajo consiste en analizar el impacto que tiene la reducción de la frecuencia de muestreo sobre la precisión de distintos modelos de clasificación para reconocimiento de actividad humana (HAR). Con este fin, se plantea el diseño de un flujo de trabajo reproducible que permita aplicar diferentes algoritmos de manera estructurada, evaluando su desempeño en escenarios donde la frecuencia de muestreo se reduce en el dataset original. Este enfoque no solo busca medir la degradación de las métricas de los modelos frente a variaciones en la resolución temporal de los datos, sino también identificar estrategias que permitan un equilibrio entre la eficiencia energética, la autonomía de los dispositivos y la calidad de la clasificación. De esta manera, el objetivo general no se limita a un análisis experimental, sino que pretende aportar lineamientos prácticos que favorezcan el despliegue de sistemas HAR en contextos con restricciones computacionales, como los propios del IoT y la computación en el borde.

2.2 Objetivos específicos

1. **Implementar diferentes algoritmos de clasificación sobre el conjunto de datos definido.**

   Este objetivo se centra en la selección y entrenamiento de modelos representativos de distintas aproximaciones al aprendizaje automático, incluyendo técnicas basadas en árboles de decisión (como Random Forest y Gradient Boosting) y arquitecturas neuronales (como redes convolucionales). La diversidad de modelos permite obtener una visión comparativa y evaluar la robustez de cada enfoque frente a la reducción de la frecuencia de muestreo.

2. **Diseñar un proceso de reducción progresiva de la frecuencia de muestreo de los datos.**

   Para simular escenarios de bajo consumo y optimización en sistemas embebidos, se propone un procedimiento controlado que reduzca sistemáticamente la tasa de muestreo de las señales de los sensores. Este diseño permitirá establecer un marco experimental donde se observe cómo el nivel de granularidad temporal influye en el rendimiento de los modelos.

3. **Evaluar cuantitativamente la variación de la precisión de los modelos en función de la frecuencia de muestreo.**

   Este objetivo persigue medir con rigurosidad cómo métricas como accuracy, precision, recall y F1-score cambian a medida que disminuye la cantidad de datos disponibles por segundo. La evaluación cuantitativa será la base para identificar tendencias y determinar umbrales críticos en los que la reducción de la frecuencia impacta significativamente la clasificación.

4. **Identificar configuraciones que mantengan un equilibrio entre rendimiento y eficiencia.**

   A partir de los resultados obtenidos, se buscarán configuraciones de muestreo y modelos que representen compromisos aceptables entre la calidad del reconocimiento y el consumo de recursos. Este objetivo es clave para proponer soluciones aplicables en sistemas con recursos limitados, como los wearables y dispositivos IoT.

5. **Diseñar un pipeline reproducible que permita seleccionar y evaluar modelos de clasificación bajo diferentes frecuencias de muestreo, con una estructura que facilite su posible despliegue futuro en entornos de ejecución más escalables.**

   Este objetivo se orienta hacia la construcción de una metodología robusta y modular, capaz de ser reutilizada o extendida en futuros trabajos. La reproducibilidad no solo aporta rigor científico, sino que también favorece la transferencia tecnológica hacia entornos industriales o de investigación.

6. **Discutir la aplicabilidad de los resultados en escenarios con restricciones computacionales, como sistemas embebidos o dispositivos IoT.**

   Finalmente, se busca extrapolar los hallazgos experimentales hacia contextos reales de aplicación. Este análisis permitirá reflexionar sobre las limitaciones prácticas y las oportunidades que brinda la reducción de frecuencia en el diseño de sistemas HAR embebidos, aportando recomendaciones para futuros desarrollos.

2.3 Beneficios del proyecto

Este proyecto aporta beneficios tanto en el ámbito académico como en el práctico. Desde la perspectiva académica, ofrece una mejor comprensión del impacto que tiene la reducción de la frecuencia de muestreo en el rendimiento de modelos de reconocimiento de actividad humana, un aspecto que no suele ser el foco principal en la mayoría de las investigaciones sobre HAR. Además, la implementación de un flujo de trabajo reproducible constituye un aporte metodológico que facilita la comparación y validación de resultados en estudios posteriores.

En el plano práctico, los hallazgos de este trabajo pueden servir como referencia para el diseño de sistemas más eficientes en recursos, lo cual es especialmente relevante en dispositivos embebidos o portátiles donde la autonomía energética y la capacidad de cómputo son limitadas. De esta forma, los resultados no solo orientan sobre la precisión alcanzable bajo diferentes

configuraciones, sino que también proporcionan lineamientos para equilibrar rendimiento y eficiencia en escenarios reales de aplicación.

De forma complementaria, este proyecto también se alinea con diversos Objetivos de Desarrollo Sostenible (ODS) propuestos por la ONU, especialmente en lo relacionado con la innovación tecnológica y la sostenibilidad energética . Por ejemplo, al optimizar el uso de datos y reducir el consumo energético en dispositivos embebidos, la investigación contribuye directamente al ODS 7: Energía Asequible y No Contaminante [11], promoviendo sistemas más eficientes y con menor huella energética. Asimismo, la creación de soluciones tecnológicas reproducibles y escalables para el reconocimiento de actividades humanas (HAR) se relaciona con el ODS 9: Industria, Innovación e Infraestructura [12], al fomentar el desarrollo de herramientas digitales avanzadas para entornos con recursos limitados. Finalmente, las aplicaciones potenciales en salud y monitoreo remoto apoyan el ODS 3: Salud y Bienestar [13], mostrando cómo la tecnología puede impactar de manera positiva en la calidad de vida y la sostenibilidad de las comunidades modernas.

Capítulo 3. Marco Teórico

En esta sección, como marco teórico del trabajo realizado, se presentan diversas definiciones, tecnologías y conceptos necesarios para el entendimiento completo del desarrollo. Empezando por el reconocimiento de actividades humanas (HAR) como área del conocimiento, la presentación del dataset HHAR y las características de los dispositivos que contienen los sensores inerciales utilizados para captar datos. El procesamiento distribuido con Dask, en donde además de presentar este framework, se realiza una comparativa entre este y una tecnología más popular: Apache Spark, con el objetivo de justificar la elección de Dask. En ventaneo y extracción de características se presentan las generalidades de esta estrategia para procesar datos de series temporales. En Modelos de Clasificación se presentan los modelos entrenados durante el desarrollo del proyecto y en Evaluación del Modelo se presentan las métricas utilizadas para evaluar el rendimiento de los modelos. Finalmente, se presenta el estado del arte, en donde se revisan trabajos similares buscando generar una contextualización y justificación del trabajo realizado.

3.1 Reconocimiento de Actividades Humanas (HAR)

El reconocimiento de actividades humanas (Human Activity Recognition, HAR) es un campo de investigación que busca la identificación y/o monitoreo de movimientos y comportamientos de una persona a partir de datos sensoriales. En contextos como la salud, la asistencia a personas mayores, el deporte o la vigilancia, HAR permite desarrollar sistemas inteligentes que infieren actividades como caminar, correr, estar de pie o subir escaleras. HAR es un área importante de investigación en computación ubicua, su implementación se ha facilitado gracias a la miniaturización de sensores como acelerómetros y giróscopos, integrados comúnmente en teléfonos móviles y wearable, los cuales habilitan la recolección de datos de actividad humana de forma poco intrusiva, barata y conveniente [14].

3.2 Sensores Inerciales y el Dataset HHAR

La realización de este TFM se enfoca en sensores inerciales (acelerómetros y giroscopios), estos capturan información sobre el movimiento del dispositivo en los tres ejes espaciales (X, Y, Z). En este trabajo se utiliza, puntualmente, el conjunto de datos HHAR (Heterogeneity Human Activity Recognition), diseñado para evaluar algoritmos de reconocimiento de actividades humanas en contextos del mundo real. Este dataset contiene mediciones de aceleración y velocidad angular obtenidas desde dispositivos móviles, específicamente, 8 smartphones (2 Samsung Galaxy S3 mini, 2 Samsung Galaxy S3, 2 LG Nexus 4 y 2 Samsung Galaxy S+) y 4 smartwatches (2 LG G Watch y 2 Samsung Galaxy Gear), utilizados por 9 usuarios identificados de la ‘a’ a la ‘i’ mientras realizaban distintas actividades. **Las actividades registradas incluyen caminar, estar de pie, sentarse, subir escaleras, bajar escaleras y montar bicicleta**. Las mediciones fueron tomadas a la máxima frecuencia de muestreo permitida por cada dispositivo, lo que introduce variabilidad

en las tasas de muestreo [10]. Esta heterogeneidad en dispositivos, frecuencias de muestreo y usuarios añade realismo y complejidad al problema.

3.3 Procesamiento Distribuido con Dask

Dada la magnitud del dataset, se emplea Dask, una librería de Python que permite el procesamiento distribuido de grandes volúmenes de datos de forma similar a pandas. Dask permite operar sobre particiones, paralelizando tareas como el filtrado, transformación y extracción de características (feature extraction) y demás operaciones que se exponen en la API de Pandas. La diferencia es que con Dask, se puede trabajar de manera distribuida, utilizando todos los núcleos del computador e incluso, de más computadores en caso de contar con un cluster [15].

3.3.1 Dask vs Spark

Spark es, probablemente, el framework más popular para realizar procesamiento distribuido de datos. Cabe cuestionarse, ¿Por qué en la realización de este trabajo se escoge una librería menos popular como Dask? A continuación se presenta una tabla que compara ciertas generalidades entre las dos librerías según la documentación de Dask [16].

Característica	**Dask**	**Spark**
Lenguaje base	Escrito en Python	Escrito en Scala, con soporte para Python y R
Ecosistema	Es un componente del ecosistema Python, extiende librerías como Numpy, Pandas y Scikit-learn	Proyecto grande, se integra bien con otros proyectos de Apache
Año de inicio	2010	2014
Diseño	En su nivel fundamental, está basado en un task scheduler	Basado en el paradigma Map-Shuffle-Reduce
Escalamiento	Desde uno hasta miles de nodos	Desde uno hasta miles de nodos

Tabla 1. Comparativa entre Dask y Spark

En cuanto a velocidad de procesamiento, existen fuentes que evidencian que Dask es más rápido que Spark [17], sin embargo, el criterio final de decisión no está basado en esta ventaja, está basado en el contexto de este proyecto: Si bien Apache Spark es una tecnología ampliamente adoptada en la industria, respaldada por una gran comunidad de desarrolladores y empresas que contribuyen activamente a su evolución y relevancia, se deduce de la investigación realizada

para la ejecución de este trabajo, que su elección está generalmente motivada por requisitos de integración con sistemas existentes, entornos de producción a gran escala o despliegues en clústeres complejos. Spark se ha convertido en una herramienta estándar en entornos empresariales, y dominarla constituye una habilidad demandada en el ámbito profesional. Sin embargo, en el contexto particular de este trabajo, no se plantea como objetivo la integración con sistemas empresariales existentes ni la puesta en producción de una solución basada en una infraestructura distribuida excesivamente compleja. El enfoque está centrado en el desarrollo de un pipeline de procesamiento distribuido eficaz y comprensible, que permita explorar, transformar y modelar datos de sensores inerciales de forma eficiente y reproducible.

Teniendo en cuenta esta perspectiva, se opta por utilizar Dask como framework de procesamiento distribuido, ya que ofrece una interfaz más simple, cercana y muy familiar al ecosistema Python. Dask permite ejecutar cargas distribuidas incluso en entornos locales sin necesidad de infraestructura adicional. Esta elección responde a un criterio práctico: maximizar la productividad y centrarse en los aspectos metodológicos del análisis de datos y del aprendizaje automático, sin verse condicionado por la complejidad operativa de Spark, además de aprovechar la oportunidad para estudiar Dask como tecnología con el objetivo de expandir conocimiento. Cabe destacar que esta decisión también permite aprovechar al máximo los recursos disponibles, al tiempo que se mantienen buenas prácticas de escalabilidad y paralelización del procesamiento de datos.

3.4 Ventaneo y Extracción de Características

El procesamiento de las series temporales se puede realizar mediante segmentación por ventanas deslizantes (sliding windows). Cada ventana agrupa una porción de la señal a la que se le aplican funciones estadísticas (media, desviación estándar, mínimo, máximo) sobre cada eje. Este enfoque transforma el problema de series temporales en uno de clasificación multivariada por ventanas, lo cual simplifica el modelado y entrenamiento de clasificadores clásicos [18] Este es uno de los acercamientos que se han tomado en la realización de este trabajo.

3.5 Modelos de Clasificación

Se exploran distintos enfoques de clasificación supervisada:

- **Random Forest Trees**: Ensemble de árboles de decisión con buena capacidad de generalización y bajo riesgo de sobreajuste.
- **XGBoost**: Variante eficiente de Gradient Boosting, robusta a datos ruidosos y con buena capacidad explicativa.
- **Histogram-based Gradient Boosting Classification Tree:** Variante optimizada del algoritmo de Gradient Boosting que agrupa los valores continuos en histogramas antes

de construir los árboles, esto permite una mayor eficiencia computacional y escalabilidad sin perder capacidad predictiva.

- **Redes Neuronales Convolucionales (CNNs)**: Aplicadas sobre las series crudas segmentadas, capturan patrones locales de movimiento. Utilizadas comúnmente en procesamiento de señales inerciales.

3.6 Evaluación del Modelo

La evaluación de los modelos se lleva a cabo utilizando un train/test split estratificado por usuario, lo que asegura que los conjuntos de entrenamiento y prueba mantengan proporciones representativas de datos de cada participante. Este procedimiento es especialmente relevante en problemas de reconocimiento de actividad humana (HAR), ya que garantiza que la validación de los modelos refleje la variabilidad natural existente entre distintos usuarios y evita sesgos en la evaluación. En cuanto a las métricas utilizadas, se consideran las clásicamente empleadas en clasificación supervisada: precisión (precision), recuperación (recall), F1-score y accuracy. Sin embargo, el análisis de resultados se enfoca principalmente en dos métricas: accuracy y F1-score, seleccionadas por su relevancia en el contexto de este trabajo.

Accuracy

corresponde a la proporción de predicciones correctas respecto al total de muestras evaluadas. Dada la relativa homogeneidad de clases en el dataset empleado, esta métrica se considera informativa y útil para reflejar el desempeño global de los modelos, pues un sesgo de clase no representa un riesgo tan elevado como en escenarios con fuerte desbalanceo.

F1-score

se define como la media armónica entre precisión y recall. A diferencia del accuracy, el F1-score no solo cuantifica cuántas predicciones fueron correctas, sino que también captura la capacidad del modelo para minimizar falsos positivos y falsos negativos de manera conjunta. Esto lo convierte en una medida más integral del desempeño, especialmente en actividades cuya clasificación puede ser más ambigua.

De esta manera, el análisis de resultados combina una visión global del rendimiento de los modelos, aportada por el accuracy, con una evaluación más equilibrada y robusta, proporcionada por el F1-score, lo que permite obtener conclusiones más completas respecto al impacto de la reducción de la frecuencia de muestreo en el reconocimiento de actividades humanas.

3.7 Estado del Arte

El Reconocimiento de Actividades Humanas (HAR) se ha convertido en un área clave dentro de la computación ubicua [14], con aplicaciones relevantes en entornos de vida asistida, rehabilitación, detección de caídas, análisis del comportamiento humano y sistemas inteligentes para el hogar. El objetivo principal de HAR es identificar automáticamente actividades como caminar, estar de pie, correr o subir escaleras, utilizando datos sensoriales recolectados a través de múltiples fuentes, entre las que destacan los sensores inerciales integrados en dispositivos móviles y wearables.

Inicialmente, las soluciones para HAR dependían de sistemas de visión por computador o datos obtenidos de redes sociales, sin embargo, los sensores inerciales, acelerómetros y giroscopio se han posicionado como herramientas ideales debido a su disponibilidad, bajo coste, no intrusividad y ubicuidad. Los smartphones y smartwatches modernos integran múltiples sensores capaces de capturar datos fisiológicos y de movimiento con alta fidelidad, lo que permite realizar un seguimiento continuo de las actividades diarias sin requerir infraestructura adicional.

Los sensores móviles permiten recolectar datos multimodales con aplicaciones en contextos reales, ofreciendo ventajas claras frente a soluciones basadas en visión, como el respeto a la privacidad, la independencia del entorno y la escalabilidad. No obstante, esta naturaleza heterogénea: diferencias entre dispositivos, frecuencias de muestreo, posiciones de uso o patrones individuales de movimiento, presenta un reto importante en términos de generalización y robustez de los modelos.

Históricamente, los sistemas HAR han dependido de técnicas de ingeniería de características manuales (handcrafted features), donde se extraen atributos estadísticos y de frecuencia (media, desviación estándar, energía, entre otros) de segmentos temporales definidos por ventanas deslizantes. Aunque estas técnicas son eficientes computacionalmente y relativamente simples de interpretar, presentan limitaciones a la hora de modelar actividades complejas, especialmente ante datos no estacionarios o de alta dimensionalidad. Además, la selección de características suele ser dependiente del dominio de aplicación, dificultando la transferibilidad entre contextos.

Frente a estas limitaciones, el aprendizaje profundo ha emergido como una solución prometedora al permitir la extracción automática de representaciones jerárquicas y discriminativas directamente desde los datos brutos. Modelos como las redes neuronales convolucionales (CNN), recurrentes (RNN) y autoencoders han demostrado un rendimiento superior en tareas de clasificación de actividades humanas, especialmente cuando se combinan con estrategias de fusión multimodal de sensores para capturar tanto variaciones espaciales como temporales.

A pesar de los avances, existen aún múltiples retos de investigación abiertos:

- La necesidad de modelos robustos frente a la variabilidad entre usuarios (intra- e inter-clase).

- El diseño de arquitecturas capaces de capturar estructuras jerárquicas y temporales en los datos.
- El problema del desbalance de clases en datasets reales.
- El entrenamiento de modelos eficaces con datos no etiquetados, aprovechando el IoT y el crowdsourcing.
- El despliegue eficiente en dispositivos con recursos limitados (inferencia en tiempo real y dentro de los mismos dispositivos).

Un estudio evaluó el impacto de distintas frecuencias de muestreo en el rendimiento de modelos de aprendizaje automático para HAR. En su experimento, 30 participantes sanos llevaron sensores inerciales de nueve ejes en distintas ubicaciones del cuerpo (incluyendo la muñeca no dominante y el pecho) mientras realizaban nueve actividades comunes. Al comparar frecuencias de muestreo que iban desde 100 Hz hasta 1 Hz, **se observó que reducir la frecuencia hasta 10 Hz no afectaba significativamente la precisión de reconocimiento, mientras que una reducción extrema a 1 Hz degradaba el desempeño** [19] especialmente para actividades con patrones más sutiles como el cepillado dental. Estos resultados sugieren que es posible optimizar el consumo energético y el volumen de datos, factores clave para la miniaturización y autonomía de los dispositivo, sin sacrificar precisión, siempre que se mantenga una frecuencia mínima adecuada. Este tipo de análisis es particularmente relevante en el contexto de aplicaciones clínicas, donde el monitoreo continuo y eficiente es fundamental.

En casos específicos como en la detección de caídas, una aplicación de HAR de gran relevancia, especialmente para personas de la tercera edad, el uso eficiente de los recursos computacionales y energéticos es fundamental. Un estudio investigó cómo la **frecuencia de muestreo** afecta directamente la efectividad de los sistemas de detección de caídas. En dicho estudio, se entrenó una red neuronal convolucional (CNN) capaz de distinguir directamente caídas de actividades convencionales utilizando señales de aceleración cruda capturadas por sensores portátiles. A diferencia de trabajos previos, la evaluación se llevó a cabo sobre un amplio conjunto de repositorios públicos de trazas de referencia. Los resultados mostraron que reducir progresivamente la frecuencia de muestreo no necesariamente compromete la capacidad de detección, encontrando que una frecuencia de **20 Hz es suficiente para mantener un rendimiento óptimo** del sistema . Este hallazgo es particularmente valioso, ya que permite el diseño de dispositivos de bajo consumo energético y menor complejidad, sin sacrificar la precisión necesaria en contextos críticos como la atención a adultos mayores.

El uso de sensores inerciales en el análisis de carrera (running) ha ganado relevancia en los últimos años, particularmente para la obtención de parámetros cinemáticos, espaciotemporales y cinéticos. Un estudio evaluó sistemáticamente cómo la frecuencia de muestreo afecta la precisión de estos parámetros durante carrera con apoyo de talón. En el estudio, sensores inerciales midieron datos a 1000 Hz, y posteriormente se simularon frecuencias menores (500, 333, 250, 200 y 100 Hz) mediante reducción en post-procesado. Los resultados mostraron que para medir con precisión parámetros cinéticos como la aceleración máxima del talón, se requieren al menos 500 Hz, mientras que otros parámetros como la longitud de zancada se

estiman adecuadamente desde 333 Hz. En general, se concluyó que una frecuencia mínima de 200 Hz es necesaria para preservar la fiabilidad de los cálculos de aceleración tibial máxima, duración de zancada y variables cinemáticas [20].

Por otra parte, en cuanto a sistemas distribuidos, es importante contextualizar que el crecimiento de los volúmenes de datos ha impulsado la necesidad de herramientas que permitan paralelizar y distribuir cargas de trabajo de forma eficiente. En el ámbito del cómputo científico y la analítica de datos, la programación paralela y el *multiprocessing* han emergido como estrategias fundamentales para abordar problemas numéricos complejos. La idea central de estos enfoques es descomponer grandes problemas en subtareas más pequeñas que pueden ejecutarse de forma simultánea en arquitecturas multiprocesador y multinúcleo, reduciendo significativamente el tiempo total de cómputo. Python, históricamente limitado por el *Global Interpreter Lock (GIL)*, ha evolucionado en los últimos años hacia un ecosistema robusto de librerías y paquetes que permiten aprovechar plenamente las capacidades de paralelización. Esta transformación ha consolidado al lenguaje como una herramienta no solo para el desarrollo rápido de prototipos, sino también para aplicaciones de alto rendimiento en dominios como multimedia, detección de ataques, supercomputación y algoritmos evolutivos. En este contexto, el surgimiento de bibliotecas específicas para procesamiento paralelo en Python representa un paso clave hacia la democratización del acceso a técnicas de cómputo intensivo en diversos campos de investigación y desarrollo [21].

Dentro de algunas librerías de Python disponibles para procesamiento distribuido, se pueden encontrar opciones que van desde utilidades estándar incluidas en el propio lenguaje hasta frameworks más avanzados diseñados para entornos de alto rendimiento. Por ejemplo, la librería **multiprocessing** permite ejecutar múltiples procesos en paralelo aprovechando máquinas multiprocesador, evitando la limitación del *Global Interpreter Lock* (GIL) y ofreciendo una API sencilla para la creación y coordinación de procesos. **JMetalPy**, por su parte, está orientada a la resolución de problemas de optimización multiobjetivo, pero también integra capacidades de paralelización en sistemas multicore y clústeres, mostrando cómo la computación paralela puede extenderse a áreas más específicas de la inteligencia artificial. Otro caso es **Parsl**, una librería que facilita la definición de flujos de trabajo paralelos y asíncronos a partir de funciones de Python, orientada a la ejecución en arquitecturas que van desde procesadores multinúcleo hasta supercomputadores [21].

Asimismo, **Ray** se presenta como un framework de propósito general para el cómputo paralelo y distribuido, con un modelo basado en actores y tareas que se coordina mediante un planificador distribuido tolerante a fallos. En escenarios de computación en la nube, proyectos como **PyWren** destacan por ejecutar código Python como funciones *serverless*, dividiendo las cargas de trabajo en acciones concurrentes sobre infraestructuras de almacenamiento de objetos. Finalmente, conectores como **PyNetLogo** permiten la interacción entre simulaciones en NetLogo y entornos Python, habilitando análisis globales con procesamiento paralelo. En conjunto, estas librerías ilustran la diversidad de enfoques que Python ofrece actualmente para abordar el reto del procesamiento paralelo y distribuido [21].

En cuanto a la evolución de herramientas para el procesamiento paralelo en Python, es importante destacar que bibliotecas como multiprocessing se introdujeron en la librería estándar desde la versión 2.6 de Python en 2008, proporcionando una forma básica de crear procesos independientes y superar parcialmente las limitaciones del Global Interpreter Lock (GIL) [22]. Estas librerías sentaron un precedente para el desarrollo de soluciones más avanzadas, pero estaban orientadas principalmente a entornos de memoria compartida y multiprocesadores locales. Por otro lado, Dask apareció de manera más reciente, específicamente en el año 2014, esta librería se ha consolidado como una herramienta diseñada específicamente para Big Data y flujos distribuidos sobre múltiples nodos o clusters [23]. A diferencia de las aproximaciones previas, Dask ofrece un ecosistema nativo para integrar procesamiento distribuido con bibliotecas científicas ampliamente utilizadas como NumPy, Pandas y scikit-learn, lo cual lo convierte en una solución más alineada con las necesidades actuales de análisis de grandes volúmenes de datos. Esta evolución histórica justifica su adopción en este trabajo, ya que Dask permite escalar los experimentos de reconocimiento de actividad humana sobre datasets masivos, superando las limitaciones de las bibliotecas más antiguas y proporcionando una plataforma moderna, flexible y eficiente.

La herramienta Dask se ha consolidado como una de las soluciones más relevantes dentro del ecosistema Python para abordar problemas de gran escala en ciencia de datos. Dask fue concebido con la finalidad de extender las bibliotecas científicas más utilizadas en Python, como NumPy, Pandas y scikit-learn, hacia escenarios donde los volúmenes de datos superan las capacidades de memoria de un único computador o requieren paralelización intensiva. Su arquitectura se fundamenta en dos elementos esenciales: por un lado, colecciones de alto nivel que replican las interfaces de las bibliotecas tradicionales, pero dividen la información en bloques o particiones, y por otro, un grafo dinámico de tareas que describe las operaciones a ejecutar y permite programarlas de manera paralela y eficiente mediante un scheduler. Esta combinación de algoritmos bloqueados con programación de tareas dota a Dask de gran versatilidad, al poder gestionar cargas de trabajo tanto regulares como irregulares [24].

Un aspecto crucial que lo diferencia de aproximaciones previas es su capacidad para operar de forma eficiente tanto en un único nodo, el caso de este proyecto, como en un clúster distribuido. En un entorno local, Dask aprovecha múltiples núcleos de procesamiento a través de hilos o procesos, manejando datos mayores a la memoria mediante cálculos *out-of-core*. En cambio, en un despliegue distribuido, el mismo grafo lógico se distribuye entre múltiples trabajadores coordinados por un scheduler centralizado, posibilitando escalar la ejecución desde un portátil hasta un supercomputador sin modificar el código subyacente. Esta flexibilidad convierte a Dask en una herramienta idónea para la investigación y la industria, ya que adapta el mismo flujo de trabajo a infraestructuras con distintas capacidades [24].

Asimismo, su diseño ofrece ventajas significativas frente a otros marcos de Big Data. A diferencia de Spark, que exige adoptar un modelo específico de programación, Dask mantiene compatibilidad con las bibliotecas tradicionales de Python, permitiendo a científicos e ingenieros de datos migrar sus proyectos sin abandonar las herramientas ya conocidas. Además, habilita cálculos fuera de memoria y soporta tanto cargas masivas por lotes como grafos de tareas más

dinámicos, adecuándose a diferentes tipos de aplicaciones. En este sentido, Dask no solo representa un avance en el ámbito de la computación distribuida en Python, sino que también abre la puerta a un análisis escalable y reproducible en problemas como el reconocimiento de actividades humanas, donde los datasets suelen ser masivos y heterogéneos [24].

En términos arquitectónicos, Dask se estructura alrededor de un modelo de ejecución basado en grafos dinámicos de tareas (*dynamic task graphs*). Cada operación que el usuario realiza sobre colecciones como *Dask Array* o *Dask DataFrame* se traduce en un conjunto de tareas pequeñas interconectadas, que describen dependencias explícitas de entrada y salida. Estas tareas son gestionadas por un *scheduler* que planifica su ejecución en paralelo, distribuyendo la carga de manera eficiente en los recursos disponibles. Este diseño permite a Dask adaptarse tanto a cálculos altamente regulares —como operaciones matriciales— como a flujos de trabajo más irregulares, característicos del procesamiento de datos reales [24].

El sistema distingue entre dos modos de despliegue. En entornos de un solo nodo, Dask aprovecha el paralelismo mediante *threads* o procesos, combinando programación concurrente con estrategias de cómputo *out-of-core* para manejar datasets que exceden la memoria RAM. En configuraciones distribuidas, en cambio, se emplea un *scheduler centralizado* que coordina múltiples *workers*, cada uno con su propia memoria y recursos de cómputo. La comunicación entre *scheduler* y *workers* se realiza mediante protocolos ligeros y asincrónicos, lo que minimiza la sobrecarga en la transferencia de datos y permite escalar de manera lineal a medida que se agregan nodos. De esta forma, el mismo grafo de tareas puede ejecutarse indistintamente en un portátil o en un clúster de supercomputación, sin necesidad de modificar el código del usuario [24].

Otro aspecto relevante es que el diseño modular de Dask separa claramente las colecciones de alto nivel del motor de ejecución. Mientras que las colecciones exponen una interfaz familiar para usuarios de NumPy, Pandas o scikit-learn, el motor de *scheduling* opera de manera independiente, optimizando el orden de ejecución, detectando paralelismo inherente y gestionando la comunicación entre nodos. Esta separación no solo facilita la extensibilidad, sino que también permite a Dask adaptarse a distintos escenarios de hardware, desde servidores multicore hasta clusters heterogéneos. En consecuencia, Dask logra un balance entre flexibilidad y eficiencia, consolidándose como una herramienta idónea para aplicaciones que requieren procesar grandes volúmenes de datos con tiempos de respuesta competitivos [24].

Capítulo 4. Metodología

En este capítulo se presenta la planificación del proyecto, donde, a partir de conceptos y marcos de referencia de la Gestión de Proyectos, se lleva a cabo el desglose de los hitos principales. Asimismo, se incluye un diagrama de Gantt con las estimaciones de tiempo correspondientes, considerando un horizonte de 10 semanas académicas. Finalmente, en la sección de Procedimiento, se describirá en detalle el desarrollo realizado para el cumplimiento de los objetivos definidos en el proyecto.

4.1 Planificación del proyecto

En la etapa donde se plantea el proyecto, se definieron los siguientes hitos para lograr su consecución exitosa:

1. **Revisión bibliográfica sobre reconocimiento de actividad humana, clasificación de series temporales y técnicas de submuestreo.**

   La revisión de literatura constituye la fase inicial de planificación del proyecto y se alinea con la mejor práctica de gestión del conocimiento. Su alcance implica no solo recopilar estudios previos, sino identificar vacíos en la investigación y delimitar el estado del arte en HAR. En términos de gestión de proyectos, este hito asegura que los entregables posteriores estén fundamentados en evidencia y evita reprocesos derivados de desconocer investigaciones clave.

2. **Exploración y análisis preliminar del conjunto de datos, considerando la posibilidad de incluir datasets complementarios si resultan pertinentes.**

   Esta etapa se corresponde con la iniciación técnica del proyecto y sigue principios de control de calidad temprana. El objetivo es evaluar la idoneidad del dataset. Desde la gestión de proyectos, realizar este análisis preliminar asegura que las fases siguientes se construyan sobre insumos confiables, minimizando el riesgo de desviaciones. Es un ejemplo de cómo aplicar la gestión de riesgos a nivel de recursos, al anticipar limitaciones de los datos disponibles.

3. **Implementación de clasificadores base utilizando el dataset principal y, en caso necesario, pruebas exploratorias con otros conjuntos de datos.**

   La implementación de clasificadores iniciales corresponde a la fase de ejecución y permite generar una línea base (baseline) para las comparaciones posteriores. Este hito es fundamental porque proporciona un criterio de referencia que servirá como punto de control.

4. **Diseño y aplicación de un esquema de reducción progresiva de la frecuencia de muestreo adaptable a distintos datasets.**

   Este hito es de suma importancia, pues establece el proceso central de la investigación. Su implementación se documentó de forma reproducible, garantizando la trazabilidad

de cada decisión técnica. Desde la óptica de gestión de proyectos, este paso se traduce en la creación de un proceso estandarizado que puede reutilizarse en futuros trabajos, cumpliendo con buenas prácticas de estandarización y transferencia de conocimiento.

5. **Evaluación y comparación del rendimiento de los modelos de clasificación bajo diferentes frecuencias de muestreo.**

   La evaluación constituye un proceso de control y monitoreo del proyecto. Aquí se aplicaron las métricas de calidad (accuracy, F1-score, etc.) para medir el desempeño de los modelos bajo distintas condiciones. Este hito es clave porque asegura que el producto intermedio cumple con los criterios de aceptación definidos.

6. **Análisis detallado de resultados, identificación de patrones y elaboración de conclusiones generales aplicables a distintos contextos.**

   Finalmente, esta etapa se sitúa en la fase de cierre del proyecto, en la que se consolidan los hallazgos, se identifican patrones y se documentan las lecciones aprendidas. Desde las mejores prácticas de gestión, este hito garantiza que el conocimiento generado sea transferible, escalable y aplicable a otros contextos. A nivel práctico, permite también hacer recomendaciones para proyectos futuros, cumpliendo con la gestión del conocimiento organizacional y la mejora continua.

Esta planeación fue ejecutada con una estimación de 10 semanas de desarrollo en donde podemos ver la distribución de hitos a lo largo de las semanas en el siguiente diagrama de Gantt.

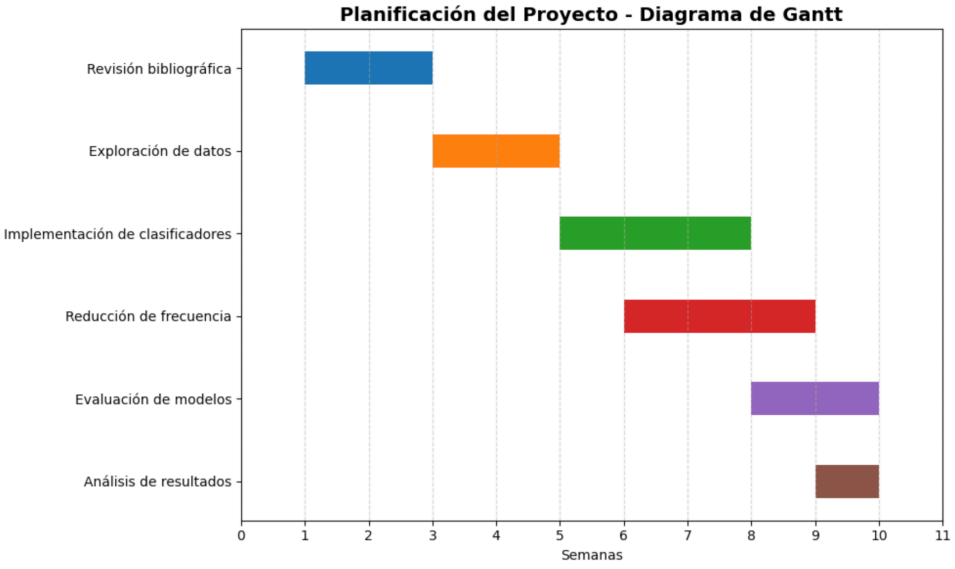

Figura 1. Diagrama de Gantt del proyecto

4.2 Procedimiento

En este apartado se expone el procedimiento seguido en el desarrollo del proyecto, el cual se vincula de manera directa con los objetivos específicos y con la planificación presentada en la sección anterior. En la primera sección, se ofrece una descripción detallada del dataset HHAR, con el fin de contextualizar sus características, variables y condiciones de adquisición de datos. A continuación, en la segunda sección, se presentan los pasos realizados para el estudio inicial del dataset, las limitaciones identificadas y las decisiones metodológicas adoptadas. En la tercera sección se aborda la implementación de un pipeline parametrizable en Python para el entrenamiento de los modelos, destacando su parametrización. Posteriormente, en la cuarta sección, se profundiza en los modelos de machine learning seleccionados, incluyendo los criterios de elección y su aplicación dentro del proyecto. Finalmente, en la quinta sección, se describen las estrategias aplicadas para reducir el volumen de datos del dataset original, con el objetivo de simular condiciones de submuestreo y evaluar su impacto en el rendimiento de los modelos.

4.2.1 Dataset Heterogeneity Human Activity Recognition (HHAR)

El primer paso realizado al escoger el dataset a usar en el proyecto, Heterogeneity Human Activity Recognition (HHAR) [10], fue estudiar su naturaleza, sus variables y detalles que entreguen el contexto necesario para poder trabajar con este mismo. Este dataset fue donado al repositorio UC Irvine en el 2015, especialmente diseñado para evaluar algoritmos de reconocimiento de actividades humanas en contextos del mundo real, considerando explícitamente la heterogeneidad en dispositivos, sensores y escenarios típicos encontrados en aplicaciones reales. Este dataset, multivariado y de naturaleza temporal (time-series), contiene un total de 43.930.257 registros recolectados por acelerómetros y giroscopios, registrando actividades humanas predefinidas Las actividades capturadas en este dataset incluyen seis clases específicas: ‘Biking’ (bicicleta), ‘Sitting’ (sentado), ‘Standing’ (de pie), ‘Walking’ (caminando), ‘Stair Up’ (subiendo escaleras) y ‘Stair Down’ (bajando escaleras). Dichas actividades se registraron sin un orden específico, buscando reflejar condiciones realistas y variadas.

Los datos se obtuvieron usando diferentes modelos de dispositivos móviles, destacando especialmente 4 smartwatches (2 LG watches y 2 Samsung Galaxy Gears) y 8 smartphones (2 Samsung Galaxy S3 mini, 2 Samsung Galaxy S3, 2 LG Nexus 4 y 2 Samsung Galaxy S+). En total, participaron 9 usuarios distintos que realizaron actividades en escenarios cuidadosamente diseñados para representar condiciones realistas y variadas: rutas alternativas para actividades como andar en bicicleta o caminar, y diferentes escaleras para registrar actividades como subir o bajar escaleras. Además del conjunto principal de datos orientado a reconocer actividades dinámicas, HHAR incluye un subconjunto de datos denominado “Still experiment dataset”, compuesto por registros estáticos del acelerómetro en dispositivos colocados en seis diferentes orientaciones físicas (sobre la parte trasera, inferior, frontal, izquierda, derecha y superior). Esta submuestra involucra dispositivos adicionales, sumando en total 31 smartphones, 4 smartwatches y 1 tablet, representando 13 modelos diferentes y variantes de sistemas operativos (Android e iOS). La presencia de este conjunto estático permite explorar la influencia

de la orientación y el ruido del sensor en la precisión de los modelos de reconocimiento de actividades, sin embargo, a efectos del trabajo realizado, no se tuvo en cuenta este subconjunto de datos.

Cada registro del dataset cuenta con información detallada: la marca temporal (Arrival_Time, Creation_Time), lecturas de acelerómetros y giroscopios en los ejes espaciales (X, Y, Z), identificadores del usuario, modelo y dispositivo, así como la etiqueta de la actividad realizada (ground truth o gt). Es relevante señalar que el dataset HHAR posee valores faltantes, lo que implica que una fase significativa del preprocesamiento debe considerar estrategias efectivas para gestionar dichos datos, garantizando así la calidad y confiabilidad de los análisis y modelos derivados.

4.2.2 Análisis Exploratorio de los Datos

En esta fase del proyecto, ya entendiendo el dataset, se realizan diferentes observaciones sobre los datos, utilizando un Jupyter Notebook. Es en este punto del proyecto se escogió la librería Dask como herramienta de procesamiento distribuido, dado el gran volumen de datos (43,930,257 instancias), facilitando así el procesamiento y análisis eficiente del dataset completo. Dask permite trabajar con estructuras similares a las de Pandas, pero de forma distribuida, lo cual es crucial para manejar grandes volúmenes de información sin saturar la memoria RAM del equipo.

Durante el análisis exploratorio inicial, se identificó una limitación clave del dataset HHAR, relacionada con las etiquetas temporales: los registros del giroscopio y el acelerómetro provienen de archivos CSV distintos, cada uno con etiquetas temporales propias tomadas en instantes ligeramente diferentes. Esto implica que sincronizar ambos sensores para modelar de manera conjunta resultaba complejo y potencialmente introducía imprecisiones. Por esta razón, **se decidió trabajar únicamente con los datos del acelerómetro de teléfonos** bajo la hipótesis de que esta información por sí sola sería suficiente para el entrenamiento y análisis, además de simplificar el proceso al evitar problemas derivados de la falta de sincronización exacta entre ambos sensores. Otro aspecto observado durante esta fase exploratoria es la presencia de dos tipos diferentes de etiquetas temporales en cada fila del dataset original: *"creation_time"* y *"arrival_time".* Después de una evaluación cuidadosa, se optó por usar únicamente la etiqueta "*creation_time*" en los análisis posteriores, ya que esta representa el instante real en el que se generaron los datos.

Para cargar los datos, se implementaron funciones específicas utilizando Dask, que permiten seleccionar únicamente las columnas de interés del dataset original. Se definieron explícitamente los tipos de datos (dtypes) para cada columna (int64 para tiempos, float32 para lecturas de sensores y variables categóricas para actividades y usuarios), optimizando así el rendimiento computacional. Las columnas seleccionadas fueron las siguientes: *creation_time* (renombrada como "time"), ejes X, Y, Z del acelerómetro, etiqueta de actividad *gt* y el identificador del usuario *user.* Además, se añadió una columna adicional para identificar claramente el tipo de sensor y dispositivo utilizados en cada registro. Posteriormente, se procedió a crear un índice basado en la etiqueta temporal seleccionada *time*, ordenando los

datos cronológicamente. Se verificó explícitamente la cantidad de particiones generadas y su coherencia para asegurar la eficiencia en las operaciones posteriores, obteniendo así particiones homogéneas y ordenadas de manera ascendente en el tiempo. Adicionalmente, se realizó una gráfica con las frecuencias de cada actividad en el dataset, para poder conocer si estaba balanceado en todas sus clases. El resultado fue positivo, y se pudo verificar que es un dataset bastante homogéneo en su distribución.

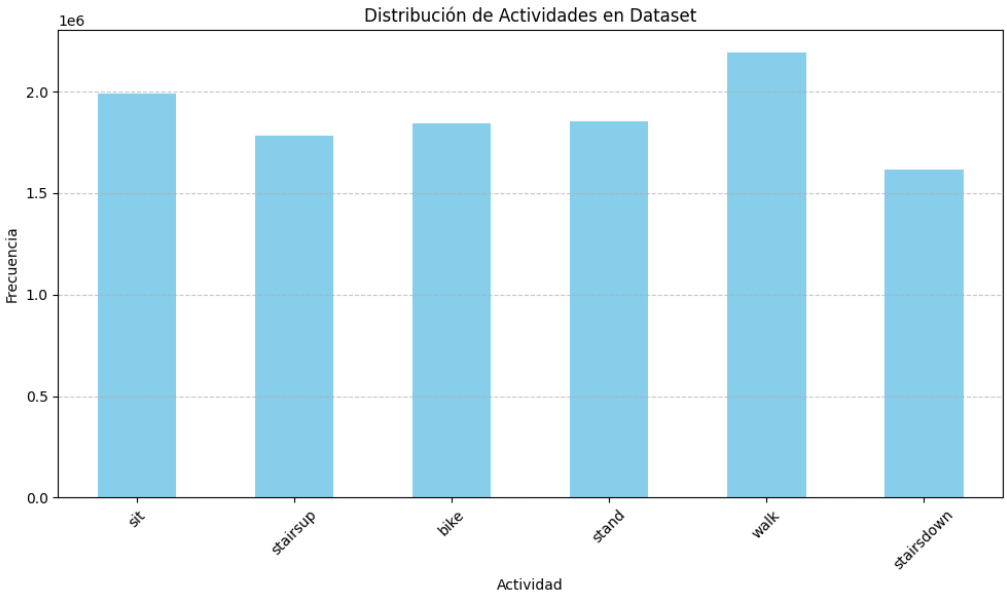

Figura 2. Distribución de actividades en el dataset

El paso final, fue identificar la cantidad de columnas con valores faltantes y eliminarlas del dataset. Del análisis se obtuvo que 1.783.200 filas no tenían clasificación de actividad (Columna *gt*). Finalmente se presenta en la siguiente tabla las primeras 10 filas del dataset.

time	X	Y	Z	activity	userid	sensor_tag
450772396000	-2.758086	1.838724	9.500074	stand	g	acc_phone
450782283000	-2.758086	1.685497	9.500074	stand	g	acc_phone
450802669000	-2.758086	1.838724	9.346847	stand	g	acc_phone
450822719000	-2.604859	1.838724	9.19362	stand	g	acc_phone
450831996000	-2.604859	1.685497	9.19362	stand	g	acc_phone
450852687000	-2.604859	1.838724	9.040393	stand	g	acc_phone

450862087000	-2.758086	1.838724	9.19362	stand	g	acc_phone
450882656000	-2.604859	1.838724	9.346847	stand	g	acc_phone
450902675000	-2.758086	1.838724	9.19362	stand	g	acc_phone
450912166000	-2.604859	1.838724	9.19362	stand	g	acc_phone

Tabla 2. Primeras 10 filas del dataset

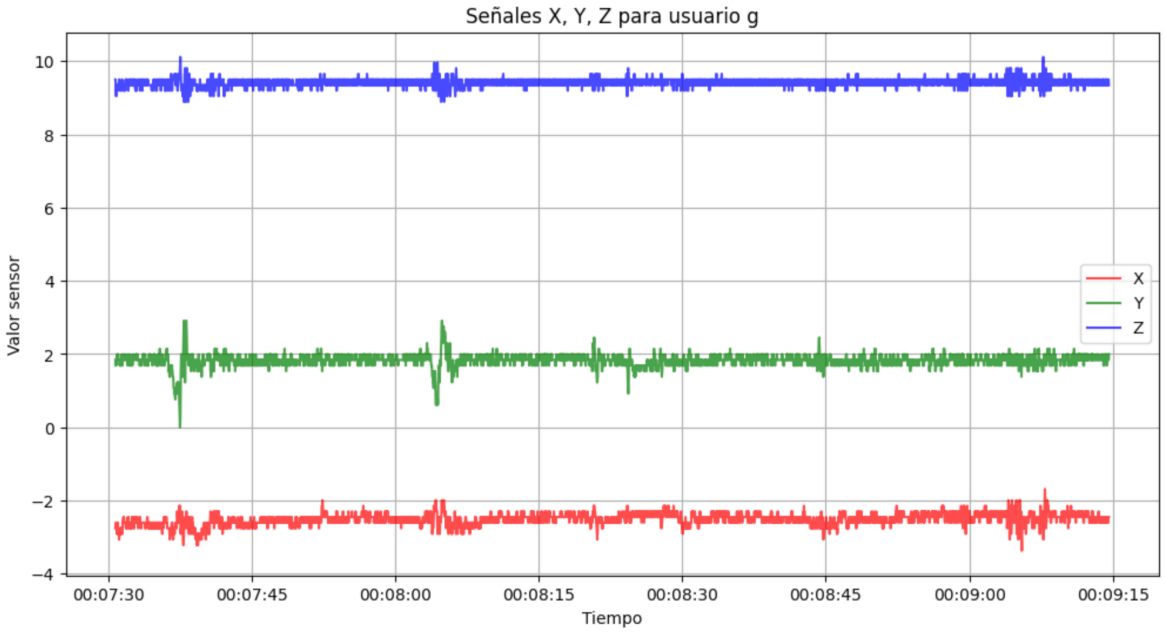

Figura 3. Gráfica de los ejes X,Y,Z del acelerómetro

4.2.3 Desarrollo de programa en Python

Gran parte de la experimentación inicial con el dataset HHAR y la librería Dask se realizó en Jupyter Notebook, lo cual facilitó el análisis exploratorio y el desarrollo iterativo del código. Sin embargo, una vez alcanzada una estructura sólida para el entrenamiento de modelos y con resultados preliminares disponibles, se tomó la decisión de desarrollar un programa modular que actuara como un **pipeline** automatizado y parametrizable. Este pipeline fue diseñado con el objetivo de sistematizar la experimentación, permitiendo ejecutar entrenamientos con diferentes configuraciones de manera reproducible y eficiente. Para lograrlo, se reutilizó gran parte del código construido previamente, pero se reorganizó en un entorno más estructurado, eliminando la parte exploratoria y enfocándose exclusivamente en el flujo de procesamiento, modelado y evaluación. Dicho programa puede encontrarse en este repositorio.

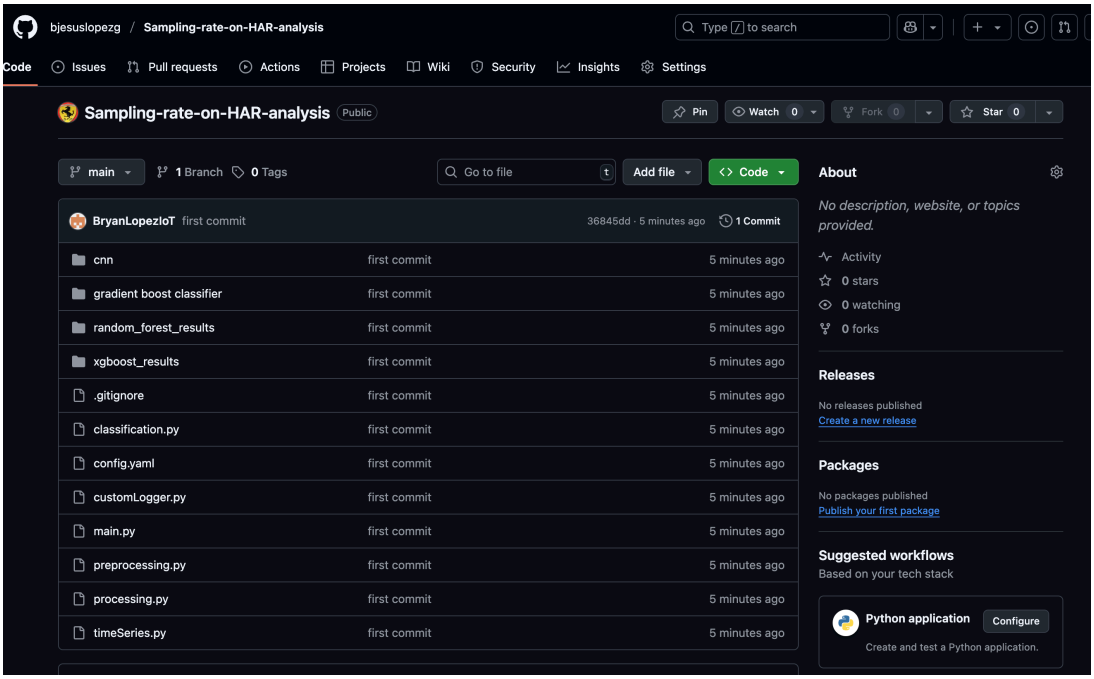

Figura 4. Vista del repositorio creado y sus ficheros

El sistema recibe sus parámetros desde un archivo de configuración en formato YAML, en el cual se especifican aspectos como el modelo de clasificación a utilizar, el nivel de submuestreo, el tipo de representación de las ventanas, y otros hiperparámetros relevantes. A partir de esta configuración, el programa ejecuta automáticamente el preprocesamiento, entrena el modelo, y guarda tanto los resultados como un log detallado de la ejecución en un archivo de texto de reporte. Este programa permitió una optimización del flujo de trabajo y dotar al proyecto de observabilidad y trazabilidad con el archivo de reporte. A continuación, se presentan las características fundamentales de cada componente de dicho programa:

- **main.py**
    - Archivo principal que orquesta la ejecución completa del pipeline.
    - Carga el archivo de configuración config.yaml.
    - Inicializa el cliente Dask para procesamiento distribuido.
    - Llama a las funciones de preprocesamiento, extracción de características y entrenamiento de modelos según el modelo especificado.
    - Registra logs y resultados en un archivo .txt.
    -
- **config.yaml**
    - Archivo de configuración que permite parametrizar la ejecución del pipeline.

- Define el dataset, tipo de sensor, tamaño de ventana, modelo a entrenar, y parámetros de entrenamiento (epochs, batch size…)
- Facilita la ejecución repetible de experimentos con diferentes configuraciones.

- **preprocessing.py**
  - Contiene funciones para cargar y limpiar los datos desde archivos CSV.
  - Implementa lógica de preprocesamiento como:
    - Renombrar columnas.
    - Establecer índices temporales.
    - Eliminar valores nulos.
    - Aplicar submuestreo programático por índice (usando *row_id*).
- **processing.py**
  - Extrae las características de las señales en ventanas deslizantes.
  - Define dos flujos distintos:
    - Extracción de estadísticas (media, std, min, max) para modelos de clasificación tabular.
    - Segmentación de señales crudas (matrices 512×3) para modelos de series temporales.
  - Expone funciones para trabajar con Dask DataFrames.
- **classification.py**
  - Entrena y evalúa modelos clásicos de clasificación:
    - RandomForestClassifier.
    - XGBoostClassifier.
    - HistGradientBoostingClassifier.
  - Utiliza train_test_split con estratificación por usuario.
  - Imprime métricas de rendimiento con classification_report.
- **timeSeries.py**
  - Define y entrena modelos para series temporales:
  - Prepara tensores (numpy) a partir de las ventanas.
  - Usa Keras para definir y entrenar las redes neuronales.

- **customLogger.py**
    - Define la clase Tee para redirigir simultáneamente la salida estándar (stdout y stderr) a la consola y a un archivo de log. Esto con el objetivo de ver tanto en consola como en el archivo txt, los logs de ejecución.
    - Permite registrar todos los mensajes e informes de ejecución de forma persistente.

4.2.4 Entrenamiento de modelos de Machine Learning

Para poder entrenar los modelos, se tomaron dos acercamientos distintos que tienen en común el uso de ventanas deslizantes de longitud fija. Ambas estrategias utilizan un tamaño de ventana de 512 muestras, con un solapamiento del 50%, es decir, un paso de 256 muestras entre una ventana y la siguiente.

Primera estrategia

se adoptó una perspectiva clásica de ingeniería de características. Para cada ventana extraída de la señal, se calcularon estadísticas descriptivas por eje del acelerómetro: media, desviación estándar, mínimo y máximo. Esto permitió transformar la serie temporal en un conjunto de vectores de características que resumen la forma de la señal en cada ventana. Además, se asoció a cada ventana una etiqueta de actividad (mediante la moda dentro de la ventana) y un identificador de usuario. Este enfoque permitió construir un dataset clásico de aprendizaje supervisado, compatible con modelos de clasificación basados en árboles de decisión como XGBoost, Random Forest y Histogram-based Gradient Boosting Trees. Estas técnicas fueron seleccionadas por su robustez, interpretabilidad y buen desempeño sobre conjuntos de datos tabulares.

Segunda estrategia

se decidió mantener la representación cruda de la señal temporal. Para cada ventana, se extrajo directamente la matriz tridimensional de forma (512, 3), correspondiente a las lecturas de los tres ejes del acelerómetro. A esta representación se le añadió la etiqueta de actividad y el usuario correspondiente. Esta configuración es más adecuada para modelos capaces de capturar patrones espaciales y temporales directamente sobre los datos, como lo son las redes neuronales convolucionales (CNN). Este tipo de arquitectura permite aprender automáticamente características relevantes sin necesidad de definir manualmente atributos estadísticos, por lo que resulta particularmente útil en contextos donde el comportamiento dinámico de la señal es esencial. Ambos enfoques fueron implementados mediante funciones que procesan cada partición del dataset usando estructuras tipo ventana con desplazamiento. Esta separación de estrategias no solo permitió comparar el desempeño entre modelos clásicos y neuronales, sino también evaluar cómo el preprocesamiento influye en la capacidad de generalización y la precisión del reconocimiento de actividades.

Usando la **primera estrategia** con los modelos ya mencionados (XGBoost, Random Forest y Histogram-based Gradient Boosting Trees), el componente de *classification* cuenta con la función *train_model*. Esta función implementa la etapa de entrenamiento supervisado para

estos modelos. Su objetivo es recibir como entrada el conjunto de características (*features*) previamente extraído del dataset HHAR y entrenar un modelo específico en función de la configuración seleccionada en el archivo YAML. Internamente, la función realiza un preprocesamiento de las etiquetas mediante codificación con LabelEncoder para transformar las etiquetas categóricas en valores enteros, formato que es necesario para la mayoría de los clasificadores. Posteriormente, se dividen los datos en conjuntos de entrenamiento y prueba utilizando una partición estratificada por usuario, garantizando que la distribución de las clases se mantenga balanceada entre ambos conjuntos. Una vez entrenado el modelo, se evalúa su rendimiento utilizando métricas estándar como precisión, recall y f1-score, generando un reporte completo.

Hiperparámetro	**Valor**
n_estimators	200
n_jobs	-1
random_state	42

Tabla 3. Hiperparámetros seleccionados para Random Forest Classifier.

Hiperparámetro	**Valor**
objective	“multi:softmax”
num_class	Número de actividades presentes en el dataset
n_estimators	200
max_depth	6
learning_rate	0.1
n_jobs	-1
random_state	42
use_label_encoder	False
eval_metric	mlogloss

Tabla 4. Hiperparámetros seleccionados para XG-Boost

Hiperparámetro	Valor
Learning_rate	0.1
max_depth	5
random_state	42

Tabla 5. Hiperparámetros seleccionados para Histogram-based Gradient Boosting

La selección de hiperparámetros para los modelos de clasificación se realizó teniendo en cuenta un compromiso entre rendimiento y costo computacional, en Tabla 3, Tabla 4 y Tabla 5 se pueden apreciar los valores de ciertos hiperparámetros escogidos para cada modelo (no se listan los que asigna por defecto la API de Scikit-Learn). En el caso de **Random Forest**, se utilizó un número de estimadores *(n_estimators*) de 200 para asegurar estabilidad en la predicción sin incurrir en sobrecostos de tiempo de cómputo innecesarios. En el modelo **XGBoost**, se configuró la profundidad máxima de los árboles (*max_depth*) en 6 y la tasa de aprendizaje (*learning_rate*) en 0.1, valores recomendados para problemas multiclase de complejidad media como HAR (*Human Activity Recognition*) [25]. Además, se desactivó el codificador interno (*use_label_encoder=False*) dado que las etiquetas se preprocesan externamente con *LabelEncoder*, lo que permite un mayor control. Finalmente, para **Gradient Boosting (HistGradientBoostingClassifier)** se optó por una profundidad máxima (*max_depth*) de 3 y una tasa de aprendizaje de 0.1, buscando evitar sobreajuste y mantener tiempos de entrenamiento reducidos. Estos hiperparámetros fueron elegidos en base a experimentación previa, haciendo especial énfasis en buscar buenos resultados con tiempos de entrenamiento reducidos, no necesariamente teniendo un enfoque en maximizar las métricas, puesto que el objetivo principal del proyecto es estudiar el efecto de la reducción de la frecuencia de muestreo sobre un clasificador base (Con el 100% de las muestras).

El módulo *timeSeries* implementa la lógica de entrenamiento de modelos de aprendizaje profundo diseñados para el procesamiento de datos secuenciales, específicamente redes neuronales convolucionales (CNN) adaptadas a series temporales multivariadas, este modulo está asociado a el procesamiento definido en la **segunda estrategia**. La función *train_time_series_model* recibe como entrada un conjunto de ventanas temporales (*windows*) que contienen las señales de acelerómetros o giroscopios segmentadas en intervalos fijos. Cada ventana se organiza como un tensor tridimensional con la forma (n_muestras, pasos_de_tiempo, canales), donde los canales corresponden a los ejes X, Y y Z de la señal. Estas ventanas están asociadas a etiquetas de actividad y de usuario, que son preprocesadas utilizando *LabelEncoder* para convertir las clases categóricas en índices enteros. Esto es necesario ya que la capa de salida del modelo CNN utiliza una activación *softmax*, la cual espera clases numéricas consecutivas. Además, la división entre conjuntos de entrenamiento y prueba

se realiza manteniendo la estratificación por usuario para garantizar que las clases estén equilibradas en ambos subconjuntos y evitar sesgos.

El modelo CNN definido en la función make_cnn está compuesto por dos bloques de capas convolucionales Conv1D, cada una con 64 filtros y tamaño de kernel igual a 5. Estas capas aprenden filtros que capturan patrones locales en la señal temporal (por ejemplo, oscilaciones o picos característicos de ciertos movimientos) aplicando la activación ReLU para introducir no linealidad. El parámetro padding='same' asegura que la salida conserve la misma longitud que la entrada, lo cual es útil para mantener la información temporal a lo largo de la red. Después de cada convolución, una capa de MaxPooling1D con tamaño de ventana 2 reduce la dimensionalidad y ayuda a que el modelo aprenda características más generales, además de disminuir el riesgo de sobreajuste. Posteriormente, la salida es aplanada (Flatten) y pasa por una capa densa (Dense) con 128 neuronas y activación ReLU, que combina las características extraídas en un espacio más abstracto. Se incorpora una capa de *Dropout* con tasa de 0.3 para regularizar el modelo y reducir el sobreajuste. Finalmente, la capa de salida es una Dense con número de neuronas igual al número de clases (n_classes) y activación *softmax*, lo que permite obtener probabilidades para cada categoría de actividad [26].

4.2.5 Reducción programática de la frecuencia de muestreo

Después de encontrar una combinación adecuada de hiperparámetros para entrenar los modelos de clasificación sobre el dataset base, se procedió a diseñar una estrategia para evaluar el impacto de reducir la frecuencia de muestreo. Es importante destacar que, como se especifica en la documentación del dataset HHAR, las señales fueron registradas a la máxima frecuencia permitida por cada dispositivo, lo que genera un escenario de heterogeneidad difícil de controlar directamente. Durante esta etapa se evidenció que las diferencias de tiempo entre muestras consecutivas no eran constantes, lo cual fue corroborado mediante inspecciones directas de los datos. Esto imposibilita la aplicación directa de un proceso basado en delta temporal para reducir la frecuencia de muestreo, ya que no existe un valor confiable y homogéneo de frecuencia base. Aunque sería posible estimar una frecuencia promedio por dispositivo, esto introduciría ambigüedades.

Por lo tanto, se optó por una solución pragmática: se asignó a cada fila del dataset un identificador numérico incremental (row_id), equivalente a un contador de muestras. Este enfoque permite definir un factor de submuestreo como un número entero que indica cada cuántas filas se conserva una muestra. Para implementar esto, se utilizó una función que calcula el número de registros por partición y genera los offsets acumulados para crear una columna *row_id* coherente a lo largo de todo el dataset. Luego, se aplica un filtro que selecciona únicamente aquellas filas cuyo *row_id* es divisible por el valor de submuestreo deseado. Finalmente, se reestablece el índice temporal original (time) para mantener la compatibilidad con etapas posteriores del procesamiento. Las gráficas presentes en Figura 3 y Figura 5, muestran los primeros 45 segundos de señal para el usuario g, sin embargo, la primera muestra

las señales como vienen en el dataset, la segunda muestra las señales submuestreadas con un salto de diez muestras.

time	X	Y	Z	activity	user id	sensor_tag	row_id
450772396000	-2.758086	1.838724	9.500074	stand	g	acc_phone	0
450782283000	-2.758086	1.685497	9.500074	stand	g	acc_phone	1
450802669000	-2.758086	1.838724	9.346847	stand	g	acc_phone	2
450822719000	-2.604859	1.838724	9.19362	stand	g	acc_phone	3
450831996000	-2.604859	1.685497	9.19362	stand	g	acc_phone	4
450852687000	-2.604859	1.838724	9.040393	stand	g	acc_phone	5
450862087000	-2.758086	1.838724	9.19362	stand	g	acc_phone	6
450882656000	-2.604859	1.838724	9.346847	stand	g	acc_phone	7
450902675000	-2.758086	1.838724	9.19362	stand	g	acc_phone	8
450912166000	-2.604859	1.838724	9.19362	stand	g	acc_phone	9

Tabla 6. Primeras 10 filas del dataset con la columna contabilizadora row_id

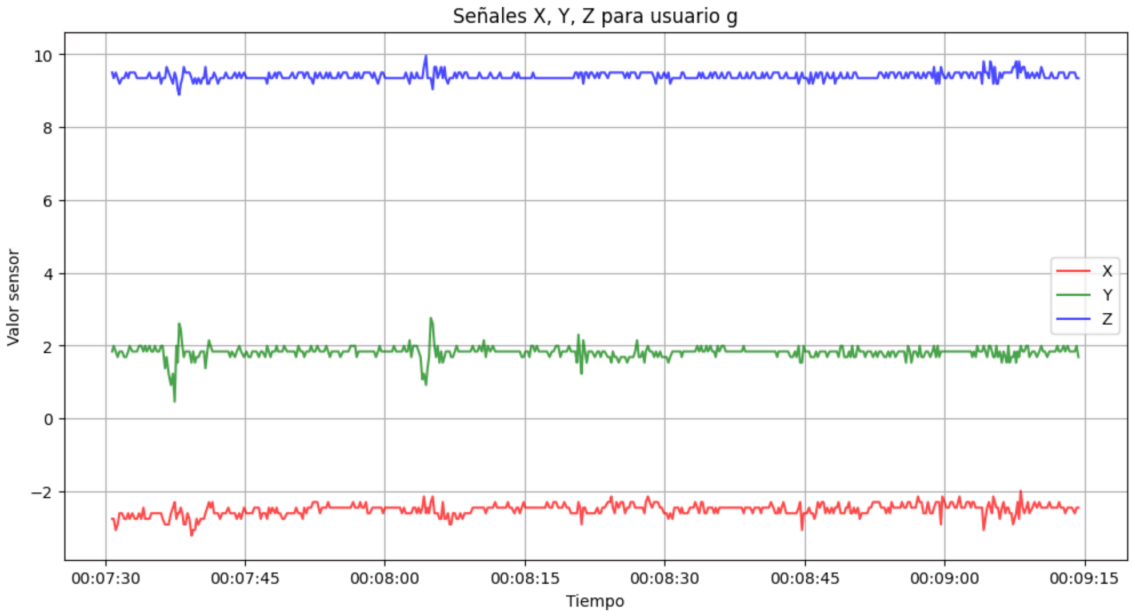

Figura 5. Gráfica de los ejes X,Y,Z del acelerómetro submuestrado.

Capítulo 5. Resultados

Con el programa Python desarrollado, se entrenaron los modelos escogidos y se obtuvieron sus métricas de rendimiento, para este propósito, el archivo yaml cuenta con un parámetro denominado *downsample_stride*, el cuál define la cantidad de muestras a saltar durante la carga de datos. Los experimentos fueron realizados con valores para este parámetro de 0, 2, 5, 10, 20 y 50, lo cual corresponde respectivamente a retenciones del 100%, 50%, 20%, 10% y 2% de las filas del dataset inicial.

En los **Anexos** se incluyen las tablas completas para cada modelo y cada partición de datos definida, con todas las métricas de evaluación: *f1-score*, *recall*, *precision* y *accuracy.* En la presente sección de **Resultados** se mostrará una versión resumida, centrada únicamente en el *F1-Score* y el *Accuracy*. Esta decisión responde a que el *F1-Score* ofrece una visión más integral al combinar *recall* y *precision* en una sola medida, mientras que el *Accuracy* resulta pertinente dada la distribución relativamente homogénea de clases en el dataset.

5.1 XG-Boost

Se presentan las métricas para el modelo XG-Boost en la tabla Tabla 7.

Porcentaje de datos	F1-Score						Accuracy
	Bike	Sit	Stairsdown	Stairsup	Stand	Walk	
100%	0.90	0.97	0.85	0.86	0.94	0.91	0.91
50%	0.92	0.98	0.87	0.88	0.96	0.94	0.93
20%	0.86	0.94	0.81	0.80	0.89	0.88	0.87
10%	0.87	0.94	0.75	0.77	0.91	0.90	0.87
5%	0.87	0.93	0.51	0.71	0.88	0.88	0.82
2%	0.70	0.92	0.21	0.46	0.76	0.69	0.69

Tabla 7. F1-Score y Accuracy para diferentes frecuencias de muestreo del modelo XG-Boost

5.2 Random Forest Tree

Se presentan las métricas para el modelo Random Forest Tree en la Tabla 8.

Porcentaje de datos	F1-Score						Accuracy
	Bike	Sit	Stairsdown	Stairsup	Stand	Walk	
100%	0.91	0.97	0.86	0.87	0.94	0.92	0.92
50%	0.92	0.98	0.88	0.89	0.96	0.94	0.93
20%	0.88	0.94	0.80	0.84	0.91	0.91	0.88
10%	0.86	0.93	0.77	0.78	0.93	0.91	0.87
5%	0.83	0.90	0.56	0.68	0.89	0.90	0.81
2%	0.61	0.90	0.34	0.55	0.84	0.83	0.73

Tabla 8. F1-Score y Accuracy para diferentes frecuencias de muestreo del modelo Random Forest Tree

5.3 Histogram-based Gradient Boosting Classification Tree

Se presentan las métricas para el modelo Histogram-based Gradient Boosting Classification Tree en la tabla Tabla 9.

Porcentaje de datos	F1-Score						Accuracy
	Bike	Sit	Stairsdown	Stairsup	Stand	Walk	
100%	0.88	0.97	0.83	0.83	0.93	0.90	0.89
50%	0.93	0.98	0.86	0.88	0.95	0.93	0.92
20%	0.87	0.95	0.82	0.80	0.88	0.90	0.87
10%	0.86	0.94	0.74	0.76	0.94	0.91	0.86
5%	0.79	0.91	0.51	0.65	0.88	0.90	0.79
2%	0.70	0.88	0.28	0.72	0.73	0.80	0.75

Tabla 9. F1-Score y Accuracy para diferentes frecuencias de muestreo del modelo Histogram-based Gradient Boosting

5.4 Redes Neuronales Convolucionales (CNN)

Se presentan las métricas para las Redes Neuronales Convolucionales en la tabla Tabla 10.

Porcentaje de datos	F1-Score						Accuracy
	Bike	Sit	Stairsdown	Stairsup	Stand	Walk	
100%	0.87	0.95	0.79	0.83	0.90	0.87	0.87
50%	0.85	0.94	0.84	0.85	0.89	0.91	0.88
20%	0.76	0.89	0.73	0.74	0.79	0.85	0.80
10%	0.81	0.92	0.78	0.65	0.88	0.79	0.81
5%	0.63	0.83	0.48	0.51	0.77	0.75	0.67
2%	0.60	0.82	0.11	0.49	0.76	0.58	0.63

Tabla 10. F1-Score y Accuracy para diferentes frecuencias de muestreo de Redes Neuronales Convolucionales

Capítulo 6. Discusión de Resultados

A partir de las tablas presentadas en el capítulo anterior, se optó por representar los valores de F1-Score y Accuracy mediante gráficos de líneas, con el propósito de ofrecer una visualización más clara que facilite su análisis y discusión de los resultados.

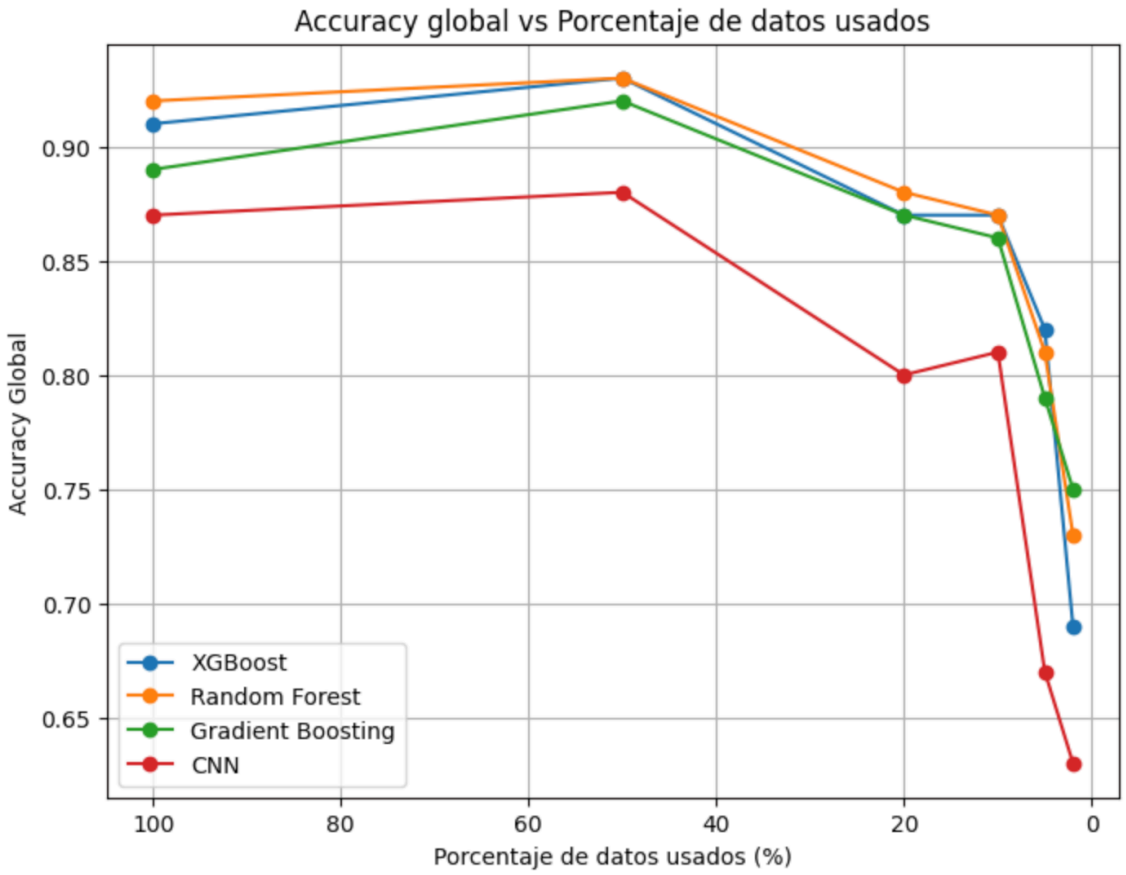

Figura 6. Accuracy Global vs Porcentaje de datos de usados

Como ya ha sido mencionado, teniendo en cuenta la relativa homogeneidad del dataset, se podría considerar la exactitud (accuracy) como una métrica de interés a estudiar, ya que permite evaluar de manera directa la proporción de predicciones correctas sobre el total de muestras. Al analizar el comportamiento de los cuatro modelos implementados (XGBoost, Random Forest, Gradient Boosting y CNN) frente a diferentes porcentajes de datos disponibles (100%, 50%, 20%, 10%, 5% y 2%), se observa una tendencia clara: los modelos basados en ensemble (XGBoost, Random Forest y Gradient Boosting) presentan un mejor desempeño y mayor estabilidad al reducir la cantidad de datos de entrenamiento, mientras que la red neuronal convolucional (CNN) experimenta una caída más pronunciada en su exactitud.

En el caso de Random Forest y XGBoost, los resultados muestran valores de accuracy muy similares y altos, manteniéndose por encima de 0.87 incluso con solo el 10% de los datos, lo que evidencia su capacidad de generalización y robustez frente a reducciones de tamaño de dataset. Gradient Boosting presenta un comportamiento levemente inferior, pero con una tendencia estable hasta el 10%, decreciendo más rápidamente al llegar a 5% y 2% de datos. La CNN, por su parte, muestra un mayor impacto ante la reducción de datos, comenzando con 0.87 en el 100% de datos y descendiendo a 0.63 con solo el 2%, lo que confirma la sensibilidad de los modelos de aprendizaje profundo al tamaño del conjunto de entrenamiento y su mayor dependencia de grandes volúmenes de datos para mantener un rendimiento competitivo.

El deterioro más pronunciado en el rendimiento de la CNN ante la reducción de datos podría explicarse por la naturaleza misma del modelo y la representación utilizada. Mientras que los modelos de *ensemble* trabajan con un conjunto reducido de características estadísticas extraídas de las ventanas temporales (medias, desviaciones, valores mínimos y máximos por eje), lo que condensa la información en un espacio de menor dimensionalidad y reduce la varianza del modelo, la CNN procesa la señal completa como serie temporal tridimensional. Esto implica que, a diferencia de los modelos que trabajan sobre medidas agregadas, la CNN necesita aprender patrones espaciales y temporales más complejos directamente de los datos crudos, lo que requiere un volumen de datos sustancial para generalizar adecuadamente.

En escenarios con porcentajes reducidos de datos, la red convolucional dispone de menos ejemplos para aprender estas representaciones de alto nivel, lo que provoca un ajuste insuficiente de los filtros y capas densas. Al no existir un resumen estadístico que atenúe la variabilidad propia de cada ventana, la CNN es más susceptible al ruido y a variaciones individuales en las señales. Esto genera una disminución del rendimiento que se hace especialmente evidente en los escenarios de 5% y 2% de datos. En contraste, los modelos basados en características estadísticas ya parten de un conjunto de entradas más robustas y compactas, lo que les permite mantener su rendimiento de manera más estable incluso en condiciones de datos limitados.

A pesar de que la técnica de resumir estadísticamente la señal ha demostrado ser altamente efectiva para el reconocimiento de actividades, y especialmente más robusta frente a reducciones en la frecuencia de muestreo, es importante considerar sus implicaciones en escenarios de inferencia en tiempo real. El cálculo de estadísticas por ventanas requiere ejecutar operaciones agregadas sobre múltiples muestras, lo cual implica un procesamiento adicional previo a la etapa de clasificación. Este **overhead computacional** puede ser aceptable en entornos con recursos holgados o cuando el procesamiento se realiza de manera diferida (*offline*).

Sin embargo, en sistemas embebidos o dispositivos IoT donde la inferencia debe realizarse de manera continua y con latencias mínimas, dicho procesamiento intermedio, como la extracción manual de características, puede convertirse en un cuello de botella. En contraste, trabajar directamente con la señal en bruto permite que modelos de **deep learning** (como CNN) aprendan representaciones discriminativas directamente desde la secuencia temporal, reduciendo la latencia y resultando en pipelines más simples. Esta aproximación es especialmente útil en escenarios de alto rendimiento embebido: se ha demostrado que técnicas de aprendizaje profundo permiten extraer características automáticamente de datos crudos, lo que mejora la capacidad para trabajar con datos ruidosos o heterogéneos en tiempo real [27] . Por otra parte, arquitecturas **near-sensor o in-sensor** (con CNNs integradas cerca o dentro del sensor) minimizan la transferencia de datos, reducen el consumo energético y aceleran la inferencia local [28]. Además, en aplicaciones IoT que requieren decisiones instantáneas, los modelos basados en **deep learning** han demostrado ofrecer un alto rendimiento en precisión y velocidad, superando a métodos convencionales de preprocesamiento seguido de clasificación . Todo esto explica por qué utilizar la señal cruda sigue siendo una estrategia valiosa en escenarios donde la rapidez de decisión y la simplicidad arquitectónica son prioritarias [29].

Es interesante observar que, en todos los casos, el *accuracy* incrementa cuando se utiliza el 50% de los datos en lugar del 100%. Este comportamiento podría explicarse por la presencia de ruido y variabilidad en el conjunto completo: al entrenar con todos los datos disponibles, los modelos pueden estar expuestos a ejemplos atípicos, inconsistentes o poco representativos, lo que deteriora su capacidad de generalización. En cambio, al reducir la muestra a un 50%, se disminuye la probabilidad de incluir este ruido, y el modelo logra concentrarse en patrones más claros y consistentes [30].

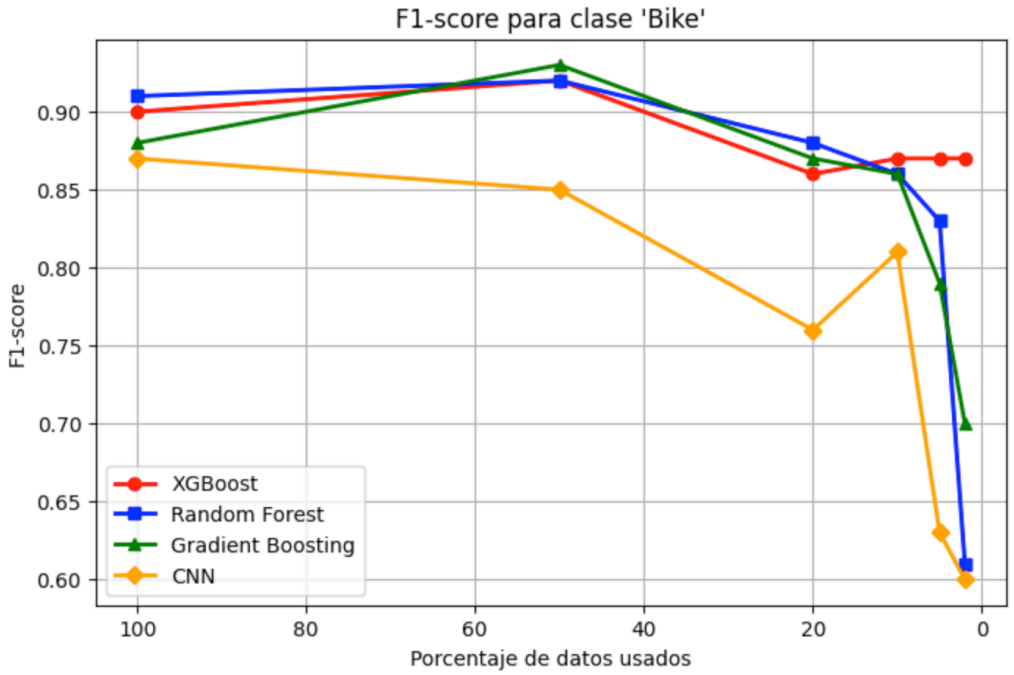

Figura 7. F1 Score para la clase Bike

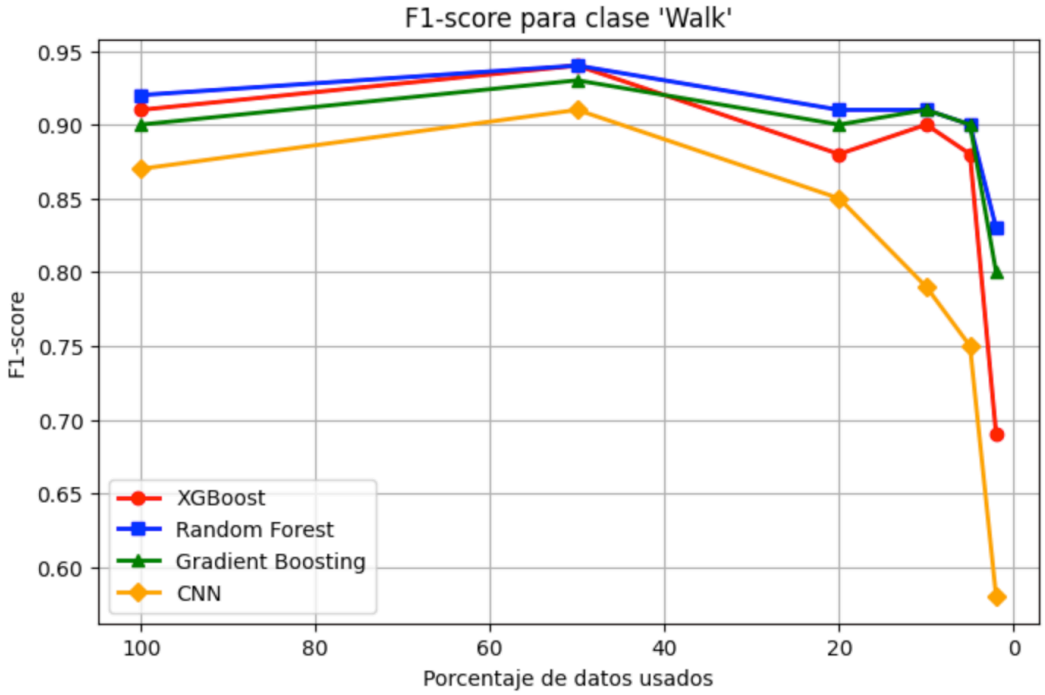

Figura 8. F1-Score para la clase Walk

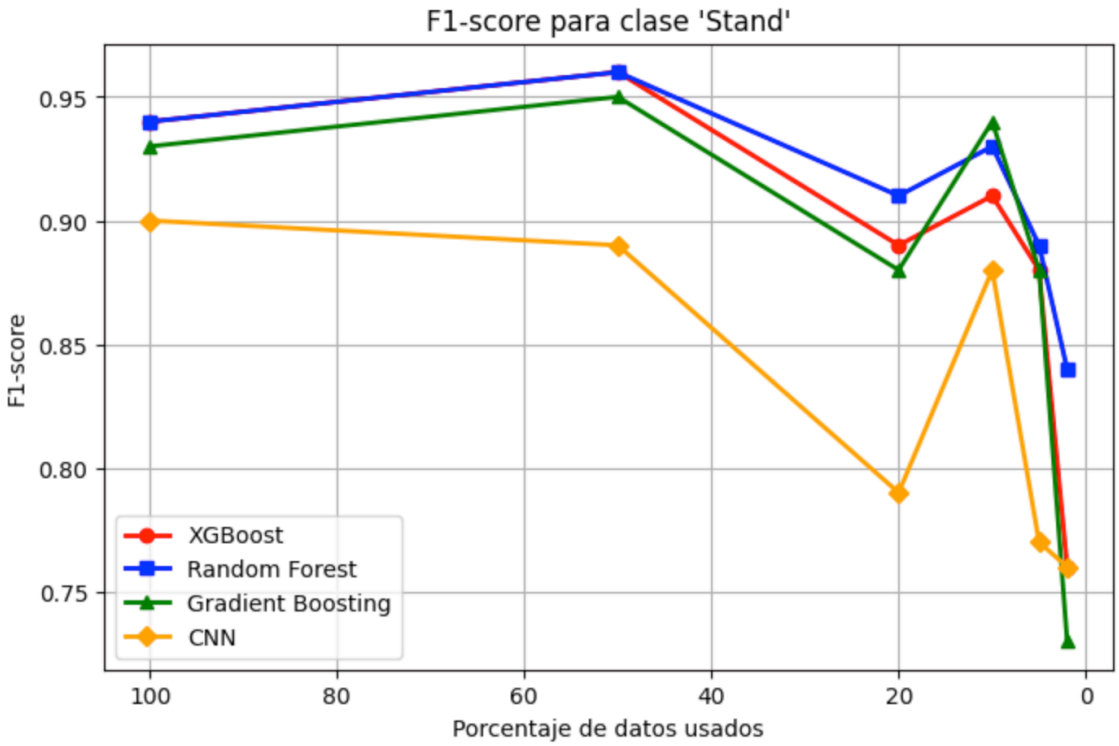

Figura 9. F1-Score para la clase Stand

En el análisis de las métricas F1-score por clase se observa un patrón claro: las actividades que implican mayor movimiento, como *bike* y *walk*, presentan una degradación más marcada en su desempeño a medida que disminuye la frecuencia de muestreo con respecto a la clase *stand*. Este comportamiento se explica porque dichas actividades dependen de patrones temporales de mediana o alta frecuencia, donde la pérdida de resolución temporal compromete la capacidad del modelo para capturar correctamente los cambios rápidos de la señal. En contraste, actividades de baja dinámica como *stand* muestran una mayor estabilidad en el F1-score incluso con reducciones significativas de datos, ya que la información relevante para su clasificación está asociada a patrones más estáticos y menos sensibles a la disminución en la tasa de muestreo. Este resultado pone de manifiesto, como ya se había discutido en el estado del arte, la importancia de ajustar la frecuencia de adquisición de datos según la naturaleza de la actividad que se pretende detectar, **especialmente en escenarios de optimización de recursos en sistemas embebidos**.

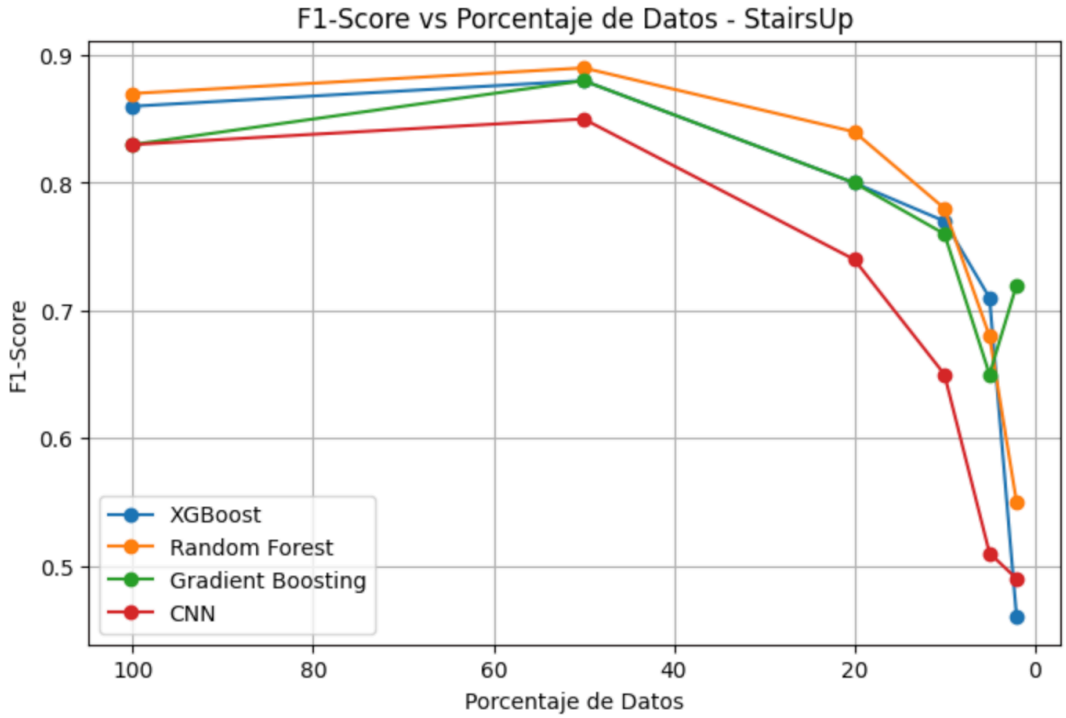

Figura 10. F1-Score para la clase StairsUp

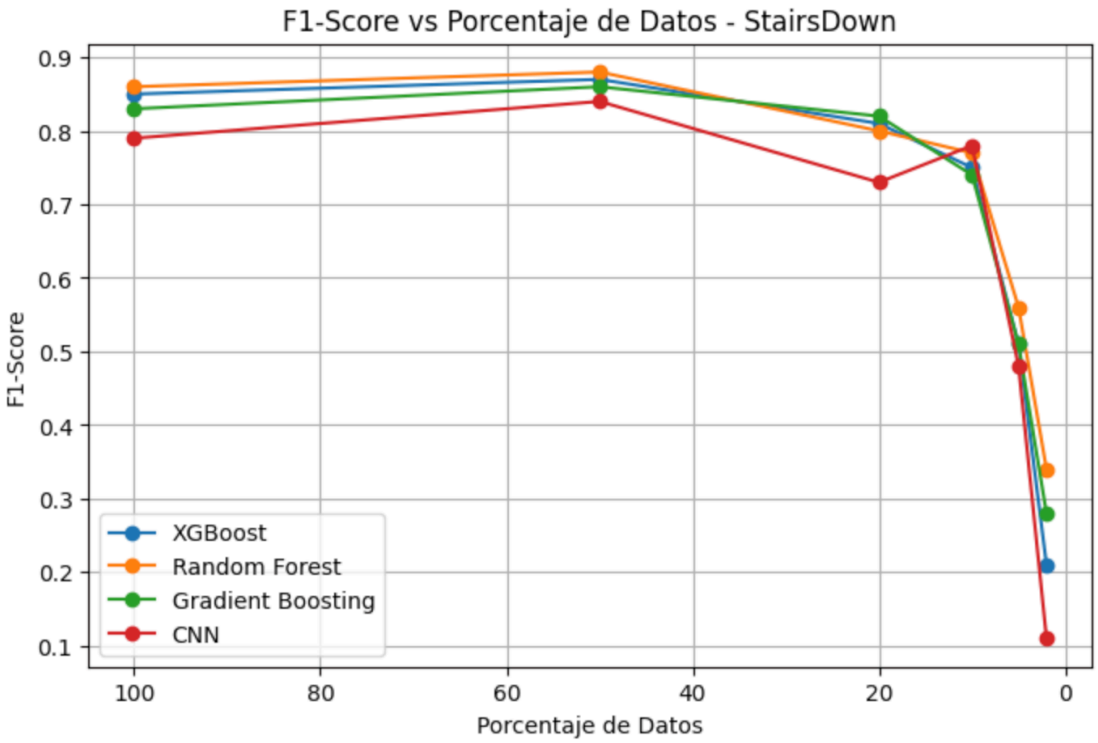

Figura 11. F1-Score para la clase StairsDown

Se observa que para actividades como *stairsup* y *stairsdown*, el deterioro del F1-score con la reducción del porcentaje de datos es mucho más pronunciado que para actividades más estáticas. Esto es consistente con la naturaleza más compleja y variable de estas actividades, que involucran movimientos más irregulares y patrones menos consistentes en la señal. Los modelos basados en árboles (XGBoost, Random Forest y Gradient Boosting) mantienen un rendimiento aceptable hasta reducciones moderadas, pero sufren caídas bruscas a partir de 10%-5% de datos. La CNN muestra una degradación más progresiva pero más acusada en valores bajos, lo que podría reflejar su dependencia de patrones temporales completos que se ven afectados por menor densidad de datos. Esto sugiere que para actividades dinámicas y con variabilidad intrínseca alta, la preservación de la frecuencia de muestreo o el uso de técnicas de aumento de datos es crítica para mantener el rendimiento de clasificación.

La actividad *stairsdown* presenta una degradación más pronunciada en el **f1-score** a medida que se reduce el porcentaje de datos, en comparación con *stairsup*. Esto puede explicarse a partir de características biomecánicas propias del descenso de escaleras: diversos estudios han mostrado que al bajar, los patrones de fuerza y aceleración son más irregulares y variables entre individuos. Por ejemplo, se ha reportado que durante el descenso el primer metatarso recibe mayores fuerzas de impacto en comparación con el ascenso. Asimismo, investigaciones en cinemática y dinámica de la marcha han demostrado que la velocidad articular de cadera y rodilla es distinta en descenso, y que las fuerzas de reacción vertical del suelo son considerablemente mayores al bajar escaleras en la fase de apoyo inicial [31], [32]. En escenarios de muestreo reducido, estas microvariaciones y picos de fuerza se pierden, dificultando la clasificación automática. En contraste, *stairsup* tiende a exhibir un patrón más uniforme y predecible, caracterizado por un movimiento más controlado y repetitivo, lo cual lo hace más robusto frente a reducciones en la frecuencia de muestreo y la resolución temporal.

Conclusiones

En el presente trabajo se desarrolló e implementó un pipeline reproducible en Python para la evaluación de distintos modelos de clasificación aplicados a un conjunto de datos de actividad humana proveniente de sensores inerciales. En línea con el primer objetivo específico, se entrenaron modelos ensemble basados en árboles de decisión (XGBoost, Random Forest y Gradient Boosting) y una red neuronal convolucional (CNN), cada uno con enfoques diferenciados de preprocesamiento: los modelos ensemble trabajaron con representaciones estadísticas extraídas de las ventanas temporales, mientras que la CNN procesó directamente la señal cruda.

Cumpliendo el segundo y tercer objetivo, se diseñó e integró un mecanismo programático para la reducción progresiva de la frecuencia de muestreo en la etapa de carga de datos, lo que permitió evaluar sistemáticamente el impacto en el rendimiento de cada modelo. Los resultados mostraron que, aunque todos los modelos presentan una degradación de métricas como accuracy y f1-score con la reducción de datos, el efecto es más pronunciado en la CNN y en actividades de alta variabilidad cinemática, como bike y stairsdown. Esto respalda la hipótesis de que los modelos basados en características estadísticas son más resilientes frente a pérdidas de resolución temporal, dado que resumen la señal y reducen la dependencia de microvariaciones temporales.

En cuanto al cuarto objetivo, se identificaron configuraciones que logran un equilibrio entre rendimiento y eficiencia. Por ejemplo, modelos ensemble entrenados con datos al 20 % de la frecuencia original conservaron un rendimiento superior al 85 % de accuracy, lo que supone un ahorro significativo en volumen de datos y coste computacional sin comprometer drásticamente la calidad de clasificación.

En relación al quinto objetivo, la arquitectura del pipeline permite reproducir experimentos fácilmente modificando un archivo de configuración YAML, con soporte para ajustar el tipo de modelo, parámetros de extracción de características y porcentaje de datos empleados. Este diseño modular y parametrizable favorece su integración en entornos de ejecución más escalables o distribuidos, como clústeres Dask.

Finalmente, y alineado con el sexto objetivo, se discutió la aplicabilidad de los resultados en contextos con restricciones de recursos, como dispositivos IoT o sistemas embebidos. Si bien el uso de representaciones estadísticas ofrece una mayor robustez ante reducción de muestreo, implica un coste adicional de preprocesamiento. En cambio, el procesamiento de la señal cruda, como en la CNN, puede simplificar la etapa de inferencia a costa de requerir más datos y potencia de cálculo en el entrenamiento. La elección entre uno u otro enfoque debe considerar el escenario de despliegue, las limitaciones de hardware y la criticidad del tiempo de respuesta.

Capítulo 7. Referencias

[1] S. Sinha, “Number of connected IoT devices growing 13% to 18.8 billion.” Accessed: May 28, 2025. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[2] A. Mukherjee *et al.*, “Edge-based Human Activity Recognition System for Smart Healthcare,” *Journal of The Institution of Engineers (India): Series B*, vol. 103, no. 3, pp. 809–815, Jun. 2022, doi: 10.1007/S40031-021-00663-W/METRICS.

[3] M. A. Hossen and P. E. Abas, “Machine Learning for Human Activity Recognition: State-of-the-Art Techniques and Emerging Trends,” *J Imaging*, vol. 11, no. 3, p. 91, Mar. 2025, doi: 10.3390/JIMAGING11030091/S1.

[4] M. T. Vi, D. N. Tran, V. T. Thuong, N. N. Linh, and D. T. Tran, “Efficient Real-Time Devices Based on Accelerometer Using Machine Learning for HAR on Low-Performance Microcontrollers,” *Computers, Materials and Continua*, vol. 81, no. 1, pp. 1729–1756, Oct. 2024, doi: 10.32604/CMC.2024.055511.

[5] A. Khan, N. Hammerla, S. Mellor, and T. Plötz, “Optimising sampling rates for accelerometer-based human activity recognition,” *Pattern Recognit Lett*, vol. 73, pp. 33–40, Apr. 2016, doi: 10.1016/J.PATREC.2016.01.001.

[6] T. S. Qureshi, M. H. Shahid, A. A. Farhan, and S. Alamri, “A systematic literature review on human activity recognition using smart devices: advances, challenges, and future directions,” *Artif Intell Rev*, vol. 58, no. 9, pp. 1–57, Sep. 2025, doi: 10.1007/S10462-025-11275-X/FIGURES/11.

[7] F. C. Andriulo, M. Fiore, M. Mongiello, E. Traversa, and V. Zizzo, “Edge Computing and Cloud Computing for Internet of Things: A Review,” *Informatics 2024, Vol. 11, Page 71*, vol. 11, no. 4, p. 71, Sep. 2024, doi: 10.3390/INFORMATICS11040071.

[8] “Embedded - The basics of designing wearable electronics with microcontrollers.” Accessed: Aug. 15, 2025. [Online]. Available: https://www.embedded.com/the-basics-of-designing-wearable-electronics-with-microcontrollers/

[9] C. Contoli, V. Freschi, and E. Lattanzi, “Energy-aware human activity recognition for wearable devices: A comprehensive review,” *Pervasive Mob Comput*, vol. 104, p. 101976, Nov. 2024, doi: 10.1016/J.PMCJ.2024.101976.

[10] “Heterogeneity Activity Recognition - UCI Machine Learning Repository.” Accessed: May 30, 2025. [Online]. Available: https://archive.ics.uci.edu/dataset/344/heterogeneity+activity+recognition

[11] “Energía - Desarrollo Sostenible.” Accessed: Sep. 11, 2025. [Online]. Available: https://www.un.org/sustainabledevelopment/es/energy/

[12] “Infraestructura - Desarrollo Sostenible.” Accessed: Sep. 11, 2025. [Online]. Available: https://www.un.org/sustainabledevelopment/es/infrastructure/

[13] "Salud - Desarrollo Sostenible." Accessed: Sep. 11, 2025. [Online]. Available: https://www.un.org/sustainabledevelopment/es/health/

[14] H. F. Nweke, Y. W. Teh, M. A. Al-garadi, and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Syst Appl*, vol. 105, pp. 233–261, Sep. 2018, doi: 10.1016/J.ESWA.2018.03.056.

[15] "Dask | Scale the Python tools you love." Accessed: Jun. 27, 2025. [Online]. Available: https://www.dask.org/

[16] "Comparison to Spark — Dask documentation." Accessed: Jun. 27, 2025. [Online]. Available: https://docs.dask.org/en/stable/spark.html

[17] "Dask vs. Spark — Coiled documentation." Accessed: Jun. 27, 2025. [Online]. Available: https://docs.coiled.io/blog/spark-vs-dask.html

[18] "Sliding Window Technique — reduce the complexity of your algorithm | by Data Overload | Medium." Accessed: Jun. 27, 2025. [Online]. Available: https://medium.com/@data-overload/sliding-window-technique-reduce-the-complexity-of-your-algorithm-5badb2cf432f

[19] T. Yamane, M. Kimura, and M. Morita, "Effects of Sampling Frequency on Human Activity Recognition with Machine Learning Aiming at Clinical Applications," *Sensors (Basel)*, vol. 25, no. 12, p. 3780, Jun. 2025, doi: 10.3390/S25123780.

[20] C. Mitschke, F. Zaumseil, and T. L. Milani, "The influence of inertial sensor sampling frequency on the accuracy of measurement parameters in rearfoot running," *Comput Methods Biomech Biomed Engin*, vol. 20, no. 14, pp. 1502–1511, Oct. 2017, doi: 10.1080/10255842.2017.1382482,.

[21] Z. A. Aziz, D. Naseradeen Abdulqader, A. B. Sallow, and H. Khalid Omer, "Python Parallel Processing and Multiprocessing: A Review," *Academic Journal of Nawroz University (AJNU)*, vol. 10, no. 3, p. 2021, doi: 10.25007/ajnu.v10n3a1145.

[22] "Multiprocessing Pool PEP and History - Super Fast Python." Accessed: Aug. 29, 2025. [Online]. Available: https://superfastpython.com/multiprocessing-pool-pep/

[23] "Dask | Encyclopedia MDPI." Accessed: Aug. 29, 2025. [Online]. Available: https://encyclopedia.pub/entry/30459

[24] M. Rocklin, "Dask: Parallel Computation with Blocked algorithms and Task Scheduling," *PROC. OF THE 14th PYTHON IN SCIENCE CONF*, 2015, Accessed: Aug. 29, 2025. [Online]. Available: https://www.youtube.com/watch?v=1kkFZ4P-XHg

[25] "furkankocatepe/Activity-Recognition-with-XGBoost: Human Activity Recognition System Using IoT Data." Accessed: Aug. 04, 2025. [Online]. Available: https://github.com/furkankocatepe/Activity-Recognition-with-XGBoost

[26] "1D Convolutional Neural Network Models for Human Activity Recognition - MachineLearningMastery.com." Accessed: Aug. 31, 2025. [Online]. Available: https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

[27] A. Cortesi *et al.*, "Integration of Deep Learning into the IoT: A Survey of Techniques and Challenges for Real-World Applications," *Electronics 2023, Vol. 12, Page 4925*, vol. 12, no. 24, p. 4925, Dec. 2023, doi: 10.3390/ELECTRONICS12244925.

[28] W. Fabre, K. Haroun, V. Lorrain, M. Lepecq, and G. Sicard, "From Near-Sensor to In-Sensor: A State-of-the-Art Review of Embedded AI Vision Systems," *Sensors 2024, Vol. 24, Page 5446*, vol. 24, no. 16, p. 5446, Aug. 2024, doi: 10.3390/S24165446.

[29] P. A. Mary, A. Sharma, S. Dekka, V. D. Gowda, and M. Singh, "Deep Learning Approaches for Real-Time Data Analytics in IoT Sensor Networks," *Lecture Notes in Networks and Systems*, vol. 1007 LNNS, pp. 239–248, 2024, doi: 10.1007/978-981-97-5146-4_21/FIGURES/5.

[30] Y. Villuendas-Rey, C. C. Tusell-Rey, and O. Camacho-Nieto, "Simultaneous Instance and Attribute Selection for Noise Filtering," *Applied Sciences 2024, Vol. 14, Page 8459*, vol. 14, no. 18, p. 8459, Sep. 2024, doi: 10.3390/APP14188459.

[31] L. S. Adiputra, S. Parasuraman, M. K. A. A. Khan, and I. Elamvazuthi, "Bio mechanics of Desending and Ascending Walk," *Procedia Comput Sci*, vol. 76, pp. 264–269, Jan. 2015, doi: 10.1016/J.PROCS.2015.12.285.

[32] A. N. Amirudin, S. Parasuraman, A. Kadirvel, M. K. A. Ahmed Khan, and I. Elamvazuthi, "Biomechanics of Hip, Knee and Ankle Joint Loading during Ascent and Descent Walking," *Procedia Comput Sci*, vol. 42, no. C, pp. 336–344, Jan. 2014, doi: 10.1016/J.PROCS.2014.11.071.

Capítulo 8. Anexos

En esta sección se presentan con mayor detalle las métricas resultantes para cada modelo entrenado. Sirviendo como extensión de los resultados presentados en el Capítulo 5 y Capítulo 6.

8.1 XG-Boost

8.1.1 100% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.89	0.91	0.90	
sit	0.96	0.97	0.97	
stairsdown	0.87	0.83	0.85	
stairsup	0.85	0.87	0.86	
stand	0.95	0.93	0.94	
walk	0.91	0.92	0.91	
Total				0.91

Tabla 11. Resultados con el 100% de los datos para XG-Boost

8.1.2 50% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.93	0.91	0.92	
sit	0.98	0.98	0.98	
stairsdown	0.87	0.86	0.87	
stairsup	0.89	0.88	0.88	
stand	0.97	0.94	0.96	

walk	0.92	0.95	0.94	
Total				0.93

Tabla 12. Resultados con el 50% de los datos para XG-Boost

8.1.3 20% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.88	0.85	0.86	
sit	0.92	0.95	0.94	
stairsdown	0.83	0.80	0.81	
stairsup	0.78	0.81	0.80	
stand	0.90	0.88	0.89	
walk	0.88	0.89	0.88	
Total				0.87

Tabla 13. Resultados con el 20% de los datos para XG-Boost

8.1.4 10% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.82	0.91	0.87	
sit	0.93	0.94	0.94	
stairsdown	0.70	0.82	0.75	
stairsup	0.83	0.72	0.77	
stand	0.95	0.88	0.91	
walk	0.91	0.89	0.90	

Total				0.87

Tabla 14. Resultados con el 10% de los datos para XG-Boost

8.1.5 5% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.90	0.84	0.87	
sit	0.92	0.94	0.93	
stairsdown	0.49	0.53	0.51	
stairsup	0.69	0.73	0.71	
stand	0.91	0.84	0.88	
walk	0.88	0.88	0.88	
Total				0.82

Tabla 15. Resultados con el 5% de los datos para XG-Boost

8.1.6 2% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.72	0.68	0.70	
sit	0.86	0.98	0.92	
stairsdown	0.20	0.21	0.21	
stairsup	0.42	0.51	0.46	
stand	0.82	0.70	0.76	
walk	0.75	0.64	0.69	

Total				0.69

Tabla 16. Resultados con el 2% de los datos para XG-Boost

8.2 Random Forest Tree

8.2.1 100% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.90	0.92	0.91	
sit	0.97	0.98	0.97	
stairsdown	0.87	0.85	0.86	
stairsup	0.86	0.88	0.87	
stand	0.97	0.92	0.94	
walk	0.92	0.93	0.92	
Total				0.92

Tabla 17. Resultados con el 100% de los datos para Random Forest Trees

8.2.2 50% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.92	0.92	0.92	
sit	0.98	0.99	0.98	
stairsdown	0.89	0.86	0.88	
stairsup	0.88	0.89	0.89	

stand	0.97	0.95	0.96	
walk	0.93	0.95	0.94	
Total				0.93

Tabla 18. Resultados con el 50% de los datos para Random Forest Trees

8.2.3 20% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.88	0.89	0.88	
sit	0.96	0.92	0.94	
stairsdown	0.79	0.81	0.80	
stairsup	0.83	0.84	0.84	
stand	0.93	0.89	0.91	
walk	0.90	0.93	0.91	
Total				0.88

Tabla 19. Resultados con el 20% de los datos para Random Forest Trees.

8.2.4 10% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.83	0.90	0.86	
sit	0.94	0.92	0.93	
stairsdown	0.73	0.82	0.77	
stairsup	0.83	0.74	0.78	

stand	0.94	0.91	0.93	
walk	0.91	0.90	0.91	
Total				0.87

Tabla 20. Resultados con el 10% de los datos para Random Forest Trees

8.2.5 5% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.86	0.80	0.83	
sit	0.90	0.90	0.90	
stairsdown	0.60	0.52	0.56	
stairsup	0.64	0.73	0.68	
stand	0.91	0.88	0.89	
walk	0.87	0.93	0.90	
Total				0.81

Tabla 21. Resultados con el 5% de los datos para Random Forest Trees

8.2.6 2% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.73	0.52	0.61	
sit	0.87	0.92	0.90	
stairsdown	0.43	0.29	0.34	
stairsup	0.45	0.69	0.55	

stand	0.86	0.83	0.84	
walk	0.82	0.84	0.83	
Total				0.73

Tabla 22. Resultados con el 2% de los datos para Random Forest Trees

8.3 Histogram-based Gradient Boosting Classification Tree:

8.3.1 100% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.88	0.89	0.88	
sit	0.97	0.97	0.97	
stairsdown	0.83	0.83	0.83	
stairsup	0.83	0.84	0.83	
stand	0.95	0.91	0.93	
walk	0.90	0.90	0.90	
Total				0.89

Tabla 23. Resultados con el 100% de los datos para el modelo Gradient Boosting Classifier

8.3.2 50% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.94	0.91	0.93	
sit	0.98	0.98	0.98	

stairsdown	0.86	0.87	0.86	
stairsup	0.87	0.88	0.88	
stand	0.96	0.94	0.95	
walk	0.92	0.94	0.93	
Total				0.92

Tabla 24. Resultados con el 50% de los datos para el modelo Gradient Boosting Classifier

8.3.3 20% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.89	0.85	0.87	
sit	0.94	0.95	0.95	
stairsdown	0.79	0.85	0.82	
stairsup	0.80	0.81	0.80	
stand	0.89	0.87	0.88	
walk	0.91	0.89	0.90	
Total				0.87

Tabla 25. Resultados con el 20% de los datos para el modelo Gradient Boosting Classifier

8.3.4 10% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.81	0.90	0.86	
sit	0.96	0.92	0.94	

stairsdown	0.73	0.75	0.74	
stairsup	0.76	0.76	0.76	
stand	0.96	0.91	0.94	
walk	0.92	0.90	0.91	
Total				0.86

Tabla 26. Resultados con el 10% de los datos para el modelo Gradient Boosting Classifier

8.3.5 5% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.78	0.80	0.79	
sit	0.89	0.93	0.91	
stairsdown	0.55	0.48	0.51	
stairsup	0.61	0.69	0.65	
stand	0.92	0.83	0.88	
walk	0.90	0.90	0.90	
Total				0.79

Tabla 27. Resultados con el 5% de los datos para el modelo Gradient Boosting Classifier

8.3.6 2% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.69	0.71	0.70	
sit	0.91	0.86	0.88	

stairsdown	0.29	0.27	0.28	
stairsup	0.68	0.76	0.72	
stand	0.71	0.75	0.73	
walk	0.83	0.77	0.80	
Total				0.75

Tabla 28. Resultados con el 2% de los datos para el modelo Gradient Boosting Classifier

8.4 Redes Neuronales Convolucionales (CNN)

8.4.1 100% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.91	0.83	0.87	
sit	0.94	0.97	0.95	
stairsdown	0.83	0.75	0.79	
stairsup	0.84	0.82	0.83	
stand	0.91	0.88	0.90	
walk	0.81	0.95	0.87	
Total				0.87

Tabla 29. Resultados con el 100% de los datos para el modelo CNN

8.4.2 50% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**

bike	0.86	0.83	0.85	
sit	0.93	0.96	0.94	
stairsdown	0.84	0.85	0.84	
stairsup	0.86	0.83	0.85	
stand	0.86	0.91	0.89	
walk	0.92	0.89	0.91	
Total				0.88

Tabla 30. Resultados con el 50% de los datos para el modelo CNN

8.4.3 20% de los datos

Clase	**Precision**	**Recall**	**F1-Score**	**Accuracy**
bike	0.80	0.72	0.76	
sit	0.85	0.93	0.89	
stairsdown	0.77	0.69	0.73	
stairsup	0.74	0.73	0.74	
stand	0.83	0.75	0.79	
walk	0.80	0.91	0.85	
Total				0.80

Tabla 31. Resultados con el 20% de los datos para el modelo CNN

8.4.4 10% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.87	0.75	0.81	
sit	0.93	0.91	0.92	
stairsdown	0.81	0.74	0.78	
stairsup	0.69	0.61	0.65	
stand	0.93	0.84	0.88	
walk	0.68	0.94	0.79	
Total				0.81

Tabla 32. Resultados con el 10% de los datos para el modelo CNN

8.4.5 5% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.55	0.74	0.63	
sit	0.93	0.74	0.83	
stairsdown	0.49	0.47	0.48	
stairsup	0.49	0.53	0.51	
stand	0.83	0.73	0.77	
walk	0.76	0.74	0.75	
Total				0.67

Tabla 33. Resultados con el 5% de los datos para el modelo CNN

8.4.6 2% de los datos

Clase	Precision	Recall	F1-Score	Accuracy
bike	0.70	0.52	0.60	
sit	0.77	0.88	0.82	
stairsdown	0.33	0.06	0.11	
stairsup	0.38	0.68	0.49	
stand	0.75	0.77	0.76	
walk	0.65	0.53	0.58	
Total				0.63

Tabla 34. Resultados con el 2% de los datos para el modelo CNN

[PÁGINA INTENCIONADAMENTE EN BLANCO]