ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Estudio de técnicas de preprocesado y clasificación para predicción del comportamiento del ganado

(Study on the techniques of preprocessing and clasification for behaviour prediction of livestock)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Andrés Martínez Lozano

Septiembre - 2024

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Andrés Martínez Lozano
Director del TFG: Pablo Pedro Sánchez Espeso
Título: "Estudio de las técnicas de preprocesado y clasificación en la predicción del comportamiento del ganado"
Title: "Study on the techniques of preprocessing and clasification in the behaviour prediction of livestock"

Presentado a examen el día:

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre):
Secretario (Apellidos, Nombre):
Vocal (Apellidos, Nombre):

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del TFG
(sólo si es distinto del Secretario)

Vº Bº del subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Agradecimientos

A mis padres, por su confianza y apoyo, sin ellos no habría sido capaz de enfrentarme a mis miedos y volver a estudiar. A ellos, por los pequeños gestos diarios que acaban siendo un mundo.

A mi tutor, Pablo, cuyo entusiasmo es contagioso, por transmitirnos a todos que encontrar un problema es sólo el primer paso para encontrar una solución.

A mis compañeros de laboratorio, por la ayuda, los cafés de por la mañana y hacer el verano más entretenido.

A Aurora, por estar.

Resumen

En este trabajo se estudia el impacto de ciertas características de las etapas de preprocesado y clasificación en un proyecto de *Machine Learning* dentro del contexto de reconocimiento de la actividad animal (*Animal Activity Recognition, AAR)* con sensores inerciales. Para ello se realiza un estudio del estado del arte, con objeto de determinar los retos, técnicas y aspectos a tener en cuenta en AAR, concluyendo con la identificación de las características a analizar y los modelos a utilizar en este trabajo. A continuación, se han implementado diversos programas en Python, que permiten explorar diferentes combinaciones de características y entrenar diversos modelos a partir de los datos contenidos en un conjunto de datos (*dataset)* público. Finalmente se lleva a cabo una evaluación de dicha exploración en base a varias métricas, para determinar qué características han sido clave en el proceso. También se ha analizado el impacto que tienen dichas características en el uso de sensores de bajo consumo con aceleradores para *Machine Learning*, claves en el futuro uso comercial de estas tecnologías.

Palabras clave: Agricultura, cabra, oveja, etograma, comportamiento, *AAR* (*Animal Activity Recognition*), sensores inerciales, acelerómetro, *LSM6DOSX*, *Machine Learning*, *Random Forest*, *dataset*, algoritmo, clasificador, modelo, predicción, preprocesado, precisión.

Abstract

This document presents an analysis of the effect of certain features from the steps of preprocessing and classifying in a Machine Learning project in the context of Animal Activty Recognition with inertial sensors. To achieve this, it begins by finding in the literature how a project is organized and then the features to analyze are selected. Pyhon's scripts are created in order to implement the different combinations of these features and to train several models with data within a public dataset. Finally, the models are evaluated based on some metrics to figure out which features have been key in the process.

Keywords: Agriculture, goat, sheep, ethogram, animal behavior, *AAR* (*Animal Activity Recognition*), inertial sensors, accelerometer, *LSM6DOSX*, Machine Learning, Random Forest, dataset, algorithm, classifier, model, prediction, preprocessing, accuracy.

Índice

Índice de figuras

Índice de tablas

Índice de ecuaciones

Acrónimos

AAR: *Animal Activity Recognition*

ML: *Machine Learning*

RF: *Random Forest*

NB: *Naive Bayes*

GNB: *Gaussian Naive Bayes*

BNB: *Bernoulli Naive Bayes*

KNN: *K Nearest Neighbors*

SVM: *Support Vector Machines*

IMU: *Inertial Measurement Unit*

DL: *Deep Learning*

PLF: *Precision Livestock Farming*

CSV: *Comma-Separated Values*

OBDA: *Overall Body Dynamic Acceleration*

VeBDA: *Vectorial Dynamic Body Acceleration*

SMA: *Signal Magnitude Area*

PCA: *Principal Component Analysis*

CART: *Classification And Regression Tree*

XGBoost: *Extreme Gradient Boosting*

LDA: *Linear Discriminant Analysis*

QDA: *Quadratic Discriminant Analysis*

LSVM: *Linear Support Vector Machines*

ANN: *Artificial Neural Network*

Capítulo 1: Introducción

1.1 Motivación

Autómata, derivado de la palabra griega "αὐτόματος" ("*automatos"*, algo que se mueve por sí mismo, algo espontáneo [1]), es un término que actualmente hace referencia a un dispositivo mecánico o electrónico que realiza ciertas funciones sin precisar de una constante supervisión humana. Automatizar un proceso es, por tanto, lograr que este se produzca sin la intervención directa de mano de obra o con una cantidad de recursos humanos reducida.

Artesano, del italiano "*artigiano*" (ejercer un arte mecánico[2]), en cambio, hace referencia a la "elaboración de objetos a mano con un sello personal, estético o de tradición cultural"[3], es decir, a la presencia ineludible de un ser humano en un proceso de fabricación. Como se observa, la artesanía y la automatización son antitéticas, y fue la primera la que, salvo por las excepciones permitidas por poleas, palancas y rampas, reinó en los sectores productivos hasta el final de la Edad Moderna, siendo el más importante de ellos el agrícola.

Dicho sector empezó a experimentar cambios a partir del S. XVI, que desembocaron en un aumento de la productividad que permitió un mayor crecimiento demográfico y la liberación de mano de obra [4], al abandonar los métodos artesanales. Sin embargo, no será hasta la implantación de los avances industriales cuando en el sector agrícola-ganadero empieza a dispararse la producción, a la vez que se hunde la fuerza de trabajo humano requerida para mantenerla. Por lo tanto, la industrialización y el incremento de la tecnificación no han sido nunca ajenos al sector agrícola y ganadero en las revoluciones industriales previas. Cabe preguntarse pues, si en los años de la "*Industria 4.0*" existirán cambios significativos similares que automaticen aún más los procesos, permitiendo producir más con menor coste humano. En este sentido, la continua mejora de las prestaciones de los sistemas electrónicos, cada vez más potentes y pequeños, aportan plataformas en las que implementar nuevas tecnologías y aplicaciones, como el reconocimiento automático de las actividades que realizan los animales o AAR (*Animal Activity Recognition*).

AAR es un área de conocimiento en desarrollo y con gran potencial, que se beneficia de las constantes mejoras tanto en microelectrónica y sensores como en ciencia de datos y ML (*Machine Learning*). Además, es un campo de estudio amplio que incluye a cualquier tipo de animal, ya sea doméstico, salvaje o de explotación ganadera intensiva o extensiva. Su interés nace del hecho de que el comportamiento es un indicador fundamental en la evaluación del estado de salud de los animales [5], [6], [7], [8] así como de su relación con el entorno [5], [6]. Conociendo cómo debe ser el comportamiento habitual de un animal se pueden detectar patrones anómalos que indiquen si está sufriendo estrés de alguna clase o si padece alguna enfermedad. También se puede conocer es estado psicológico, si tiene algún problema físico que le produzca dolor (como la cojera), si va a parir o incluso si la ingesta nutricional no ha sido la adecuada [5], [7], [8], [9]. En definitiva, AAR sirve para recabar información acerca del grado de bienestar animal del sujeto observado.

El bienestar animal es un concepto relacionado con la veterinaria y la bioética que pone de relieve las necesidades de los animales, especialmente las de aquellos que mantienen una estrecha relación con el ser humano. Desde un punto de vista ético, y cada vez más desde un punto de vista legal, se considera que cualquier animal debe poder disfrutar de "cinco libertades", a saber: ausencia de hambre y sed; ausencia de incomodidad; ausencia de dolor, lesiones y enfermedades; ausencia de miedo y angustia; y tener la libertad para expresar un comportamiento normal [10]. Al margen de las consideraciones éticas, promover el bienestar de los animales genera beneficios para la propia industria, como la mejora de la eficacia productiva o el incremento de calidad del producto final [11]. Por ello, la automatización del reconocimiento de la actividad animal resulta fundamental como herramienta para la consecución de estos objetivos.

Por otro lado, evaluando la actividad se pueden detectar problemas ambientales en el ecosistema [6], [12], como fuego [13], [14], caza furtiva [15], [16], fragmentación del hábitat y eventos relacionados con el cambio climático o propagación de enfermedades [17]. Todo esto contribuye, en el caso de la fauna salvaje, a incrementar el conocimiento que se tiene del medio ambiente, los procesos naturales y el impacto del cambio climático antropogénico. En el caso de la fauna doméstica, AAR ayuda en la prevención de enfermedades, permitiendo un tratamiento temprano [18], [19]. En el sector rural, AAR permite reducir el impacto sobre el medio ambiente de la actividad ganadera (por ejemplo, previendo pautas de alimentación [20]) además de evitar la erosión [21], darle un mejor uso al terreno [22], y aumentar la productividad de los animales y el entorno [23], [24].

Sin embargo, la inspección visual del comportamiento de los animales salvajes o domésticos plantea diversos problemas. Por un lado, la presencia de un ser humano puede alterar el comportamiento de los animales que se pretenden observar, al resultar un método invasivo [25]. Por otro lado, no todos los animales son accesibles para realizar este tipo de observaciones ya sea porque están en emplazamientos de difícil acceso [26] o porque rehúyen el contacto humano, como casi cualquier animal salvaje. Finalmente, es una labor que puede generar resultados poco precisos o inexactos [27] además de ser muy costosa desde un punto de vista de la carga de trabajo puesto que implica la observación durante horas de un grupo reducido de animales. Cuando el número de sujetos es elevado o las condiciones de observación son malas, se vuelve una tarea completamente inviable [12].

Por estas razones, la automatización en este ámbito resulta tan atrayente, siendo un campo de investigación en auge. Con el desarrollo de la ganadería de precisión (PLF, *Precision Livestock Farming*), el uso de GPS en animales, las redes de comunicación de baja potencia y área amplia (LoRaWAN), los sensores de medición inercial (IMU, *Inertial Measurement Unit)* y las técnicas de ML se pueden conseguir todos los beneficios de la evaluación de la actividad del ganado sin padecer los problemas de la observación directa. Sin embargo, aún quedan muchos retos por superar para el uso práctico de AAR. La adquisición de datos para el entrenamiento de algoritmos, la gestión del consumo de los equipos de medida, predicción y transmisión, las diferencias inherentes entre especies o la variedad de terrenos y ecosistemas son algunos de los retos que hacen que todavía se esté lejos de soluciones integrales, adaptables y comerciales de técnicas de AAR.

1.2 Objetivos

El principal objetivo de este proyecto, que estudia técnicas de ML para el análisis de la actividad animal con sensores inerciales, es averiguar qué parámetros del preprocesado y de la creación de un modelo predictivo son los mejores para predecir el comportamiento de un conjunto de cabras. Para ello se han identificado los siguientes objetivos específicos:

- Adquirir una visión holística de AAR mediante el estudio de la literatura existente, sintetizando toda la información recabada de forma que pueda servir como compendio y guía para futuros proyectos.
- Aplicar todas las combinaciones de elementos del preprocesado posibles, así como un afinado de hiper parámetros para construir un modelo de referencia con RF (*Random Forest*) para cada combinación.
- Usar los modelos de referencia como guía para la evaluación del resto de clasificadores empleados, así como para analizar el efecto de las diferentes partes del preprocesado en el resultado final.
- Evaluar la genericidad del modelo probando a inferir la actividad de animales diferentes a los usados durante el entrenamiento. Se pretende analizar el impacto del problema de la generalización del dominio ("*domain generalization"*) empleando datos de animales taxonómicamente cercanos: cabras y ovejas, de la subfamilia *Caprinae*.
- Analizar la viabilidad comercial de los modelos de referencia con mejor puntuación de precisión, comparando sus resultados con los conseguidos mediante la simulación de las características del módulo comercial "LSM6DSOX" de STMicroelectronics.

1.3 Estructura del documento

Este documento está desglosado en cinco capítulos, además de esta introducción, que abordan los siguientes aspectos:

- En el **capítulo 2** se analizan los antecedentes del trabajo que se ha realizado. Se hace un repaso del estado del arte y se identifican las diversas tecnologías utilizadas en AAR, constituyendo la base teórica del proyecto. En particular, se ha analizado cómo han configurado otros autores el preprocesado de datos, qué clasificadores de ML han sido más empleados y las razones para ello atendiendo a las singularidades de cada trabajo. También se reseñan las herramientas empleadas para el análisis del rendimiento tras la fase de inferencia.
- En el **capítulo 3** se presenta la metodología que se ha seguido para alcanzar los objetivos de este trabajo. Se parte de ciertas hipótesis identificadas en el estado del arte para describir, a continuación, las etapas que integran el desarrollo de aplicaciones de AAR con ML en este proyecto. Se exponen las razones que han llevado a incluirlas, se analizan

las limitaciones de la aproximación y se relaciona la misma con los retos de este campo de investigación.

- En el **capítulo 4** se describe la implementación de la metodología de principio a fin: desde las particularidades del *dataset* (conjunto de datos) que se va a emplear hasta el cálculo de métricas para la evaluación de los modelos desarrollados, pasando por el tratamiento de los datos y la selección de características para el entrenamiento.
- En el **capítulo 5** se presentan y comentan los resultados obtenidos con la metodología del capítulo anterior, esto es, los valores de cada una de las métricas para cada uno de los modelos resultantes del proceso. Dichos valores se representan gráficamente en función de cada uno de los parámetros, para mayor claridad expositiva.
- Finalmente, en el **capítulo 6** se extraen las conclusiones del trabajo realizado, se comentan las dificultades encontradas, los conocimientos adquiridos y las posibles líneas de investigación futuras.

Capítulo 2: Antecedentes

En este capítulo se presenta un análisis del estado del arte, con el fin de conocer cuáles han sido las soluciones adoptadas anteriormente a los retos que plantea el reconocimiento de la actividad animal, definiendo el nivel de partida teórico del trabajo.

Por razones de claridad expositiva, la información extraída de la bibliografía consultada será presentada en este capítulo en diferentes apartados, a saber: estructura de un desarrollo para AAR, animales empleados en AAR, etograma o conjunto de comportamientos a estudiar, tipos de sensores y su posición en el animal, etiquetado de los datos, procesado de los datos en bruto, algoritmos empleados, evaluación de las predicciones y retos de la AAR. Esta división expositiva está fundamentada en la forma de estructurar este tipo de trabajos como se verá en el siguiente apartado.

2.1 Estructura de un desarrollo de AAR

Lo primero es analizar cómo está estructurado un desarrollo de este tipo, qué fases tiene y cómo son cada una de ellas, antes de entrar a estudiar más en detalle sus singularidades.

Aunque los distintos autores difieran entre ellos a la hora de nombrar y agrupar las fases de un desarrollo para AAR con sensores y ML, el campo de estudio está lo suficientemente avanzado y maduro como para que ya exista un cierto consenso.

Una forma de comprobarlo es acudiendo a los artículos de revisión, que condensan la información del estado del arte mediante el análisis sistemático de un número elevado de artículos del campo de conocimiento. En este caso, los artículos de revisión consultados muestran que, aunque haya dos tendencias a la hora de agrupar las actividades, la estructura subyacente es la misma. Así, algunos trabajos dividen el proceso en siete etapas diferenciadas mientras que otros la reducen únicamente a tres.

Este último es el caso de *Riaboff et al.* [28], [29], que en sus dos artículos de revisión establecen que un desarrollo para AAR con IMUs y ML incluye tres fases: una fase inicial de recolección de datos, una intermedia donde los datos recogidos se procesan y acondicionan para el clasificador y una tercera, la de desarrollo del modelo de clasificación.

En la primera fase, recolección de datos, los trabajos [28], [29] definen tres procedimientos: la grabación de la señal del sensor inercial, la anotación del comportamiento del animal y la combinación de la señal con las anotaciones. Las dos primeras se realizan en paralelo y la tercera es realmente la combinación de sus resultados.

La segunda fase, el preprocesado, la descomponen a su vez en una primera etapa de observación de los datos y una segunda de procesado de estos. La primera consiste en analizar cómo son los datos recogidos. La segunda etapa es en la que se opera con los datos, con el fin de extraer más información y prepararlos para el clasificador. En ella se filtra el ruido, se segmenta la serie temporal, se extraen nuevas características…

En la última fase, desarrollo del modelo, los trabajos reseñados incluyen el escalado de los datos, la división del *dataset* en subconjuntos de entrenamiento y test, el entrenamiento del modelo y su validación con el subconjunto de test.

La perspectiva presentada (*Riaboff* [28], [29]), en el fondo no difiere demasiado de lo establecido en la revisión de *Kleanthous el al* [30]. En dicha referencia se identifican siete partes principales en vez de tres, siendo la estructura subyacente similar al definir prácticamente las mismas actividades en el mismo orden:

1. Adquisición de los datos
2. Etiquetado de los datos
3. Preprocesado
4. Selección del tamaño de ventana y extracción de características
5. Selección de características o reducción de dimensionalidad
6. Desarrollo del modelo
7. Evaluación del modelo

Quizás la diferencia más significativa entre trabajos es que el primero (*Riaboff el al* [28], [29]) no menciona explícitamente las técnicas de selección de características o reducción de dimensionalidad y sí la posibilidad de trabajar con series temporales adicionales [29]. Sea como fuere, independientemente de cómo se agrupen las actividades, lo que cada uno recoge en cada apartado es muy similar a lo que recoge el otro.

En el trabajo de *Mao et al.* [31], de AAR con DL (*Deep Learning*), se emplean ambas formas de estructurar este tipo de desarrollo. Además, se hace referencia a la revisión de *Kleanthous* [30] y se menciona la selección de características, pero se limita a identificar únicamente tres fases cuando representan gráficamente el proceso.

El resto de los artículos consultados, en mayor o menor medida, siguen este tipo de esquemas a la hora de presentar las fases del proceso. En algunos casos omiten algunos pasos por no ser relevantes para la explicación y en otros casos, no los implementan. Sin embargo, no hay ninguna referencia de las estudiadas que añada nada nuevo a lo presentado en los artículos de revisión mencionados.

2.2 Animales empleados en la AAR

Como se ha comentado anteriormente, las técnicas de AAR se emplean tanto para monitorizar la actividad de los animales domésticos y el ganado, como la de los animales salvajes. El rango de especies diferentes sobre las que se ha experimentado es ciertamente amplio, desde vacas hasta rinocerontes [32]. Este proyecto se ha desarrollado con el objetivo de ayudar fundamentalmente a la monitorización del ganado en el contexto de la ganadería de precisión o PLF. Por ese motivo únicamente se van a tener en cuenta en este análisis los artículos que hayan empleado ganado, ignorando los orientados a fauna salvaje y mascotas.

La elección del animal a monitorizar es algo que cambia totalmente algunas etapas del desarrollo de técnicas de AAR. Por ejemplo, las etapas iniciales del desarrollo, relacionadas con la adquisición de datos (elección de los sensores, su ubicación, actividades que se quieren monitorizar…), son muy dependientes de esta elección, ya que ni todos los animales se comportan igual, ni tienen las mismas dimensiones. Algunos investigadores encuentran que incluso puede haber diferencias significativas entre individuos de la misma variedad dentro una subespecie [33], [34]. Otras investigaciones concluyen que, para una buena generalización del modelo, se deberían emplear animales de una misma variedad pero con diferentes características físicas y etológicas (animales de diferentes tamaños, edades, sexos…) [29]. Algunos incluso trabajan con animales en diferentes localizaciones [8].

Según la revisión de *Mao et al.* [31], los animales domesticados que principalmente se han usado en trabajos de AAR son: las vacas, que son el animal más frecuente debido a su peso económico en el sector y su gran tamaño, lo que facilita el proceso de AAR; las ovejas, que tienen también un gran impacto económico por la producción de lana y cuyo descenso de actividad está estrechamente relacionado a problemas de salud; los caballos, cuya disminución voluntaria de la ingesta de alimentos es un signo relacionado con infecciones; las gallinas, para estudiar sus patrones de comportamiento en relación con la puesta de huevos o el desarrollo de enfermedades; y los cerdos, muy importantes también para la industria alimentaria y cuyos niveles de actividad son críticos para el conocimiento de su estado de salud.

Riaboff et al. [29] hicieron un análisis exhaustivo del uso de rumiantes en el campo de la AAR. En él detectaron que además de vacas y ovejas, había un porcentaje pequeño (8% frente al 64% de vacas) de los estudios enfocados a cabras. En dicho estudio recogen también las variedades diferentes de cada especie empleadas en los artículos revisados, así como el número de animales empleados en cada uno para la adquisición de datos. Esto resulta importante puesto que, como señalan las conclusiones del artículo [29], un mayor número de ejemplares ayuda a desarrollar modelos cuya capacidad de generalizar es mayor. Así se evita el *overfitting*, que aparece cuando un modelo es demasiado dependiente de los datos de entrenamiento, siendo incapaz de inferir correctamente con datos nuevos.

El estudio que se ha realizado de la bibliografía se ha limitado a las tres especies mencionadas en el párrafo anterior: vacas, ovejas y cabras. En los trabajos consultados que emplean ganado vacuno destaca el hecho de que suelen emplear un número alto de individuos dentro de rebaños relativamente grandes. En un caso, 10 vacas siendo monitorizadas dentro de un rebaño de 71 piezas pastando al aire libre día y noche [28]. En otro, 13 terneras con sensores de un total de 20, en este caso estabuladas [35]. En un tercero, emplean 30 vacas lecheras también estabuladas [12]. Todos estos estudios fueron realizados con la variedad Holstein o con variedades mestizas similares. El único estudio que se sale un poco de la norma es el de *Robert et al.* [19], que separan a los terneros en tres grupos de 5 animales y en cada grupo analiza la actividad de un individuo.

En los estudios realizados con ovejas hay más variedad: algunos autores emplean ovejas merinas o variedades mestizas monitorizando 5 en un rebaño de 10 [7], [36] o 12 de 39 posibles [37]. *Kleanthous et al.* en sus artículos de 2019 y 2020 [38], [39] emplea entre 7 y 8 ovejas de la variedad de las Islas Hébridas, aunque sin analizarlas dentro de un rebaño. Otros investigadores

como *Marais et al.* [40] indican que han empleado 5 ovejas para la AAR, pero no indican ni su raza ni si estaban integradas dentro de un rebaño más amplio.

En cuanto a los estudios que emplean cabras, contrastan con los de las vacas porque usan un número bastante más bajo de individuos y con razas mucho menos homogéneas. Así, hay estudios que han utilizado simultáneamente 3 cabras pigmeas y 2 cabras salvajes [34]. Otro solamente estudia 3 cabras de la variedad *Saanen* [41]. Un tercer estudio trabaja con 2 animales dentro de un rebaño de 24 cabezas de la variedad mestiza *Thüringer-Toteburg* y simultáneamente con 1 espécimen de *Jabal Akhdar* en un rebaño de 70 de ellas [8].

Por último, hay que destacar un par de estudios que analizan conjuntamente ovejas y cabras. *Kamminga et al.* [33] extraen datos de 4 cabras y 2 ovejas y se propone crear un modelo predictivo que se sobreponga a las diferencias inherentes entre especies experimentando con diferentes formas de entrenamiento. El dataset que desarrollaron y que proporcionan como código abierto ha sido empleado por otros trabajos de la bibliografía [42] y es el elegido en este proyecto como el caso práctico.

2.3 Etograma

Un etograma es el conjunto de comportamientos que se van a estudiar, anotar o predecir. Como se ha mencionado en el apartado anterior, depende en gran medida del tipo de animal con el que se vaya a trabajar, pues no todos realizan las mismas actividades. La elección del conjunto de actividades depende también en gran medida del objetivo del estudio y es aquí en donde existen muchas aproximaciones diferentes.

Algunos trabajos, por ejemplo, sólo diferencian entre actividad e inactividad. En cambio, otros eligen etogramas reducidos con las cuatro o cinco actividades más habituales del animal y el resto las clasifican como “desconocido” u “otras”. Esta aproximación es una solución de compromiso que permite identificar mucho de lo que hacen los animales sin complicar demasiado el estudio con actividades poco frecuentes [7], [28], [33], [34], [38], [39], [40], [42]. Algunos artículos simplifican aún más el problema diferenciado sólo tres actividades: reposar, estar levantado o caminar [7], [19], [41].

También ocurre lo contrario, *Carslake et al.* [35] construyen un etograma más complejo diferenciando dos tipos de posturas y siete comportamientos diferentes. En esta línea, otros autores se interesan no sólo por las actividades sino también por las transiciones entre ellas [12] [36]. Esto es algo que generalmente no se tiene en cuenta y añade dificultad adicional a la predicción.

Fogarty et al. [37] hacen una distinción interesante al establecer tres etogramas diferentes con las mismas observaciones. Por un lado, identifican cuatro comportamientos en un etograma (pastar, reposar, estar de pie y caminar) que en otro etograma clasifican en función de su actividad (activo o inactivo) y en un tercero en función de la postura (erguido o postrado).

Las actividades más comúnmente predichas para los rumiantes, según el análisis de *Riaboff et al.* [29] son las siguientes:

1. Alimentarse: comer, pastar, forrajear o morder arbustos
2. Rumiar
3. Descansar o estar inactivo
4. Moverse: caminar, correr o buscar
5. Estar de pie
6. Estar tumbados
7. Transiciones entre posturas como levantarse o tumbarse
8. Otras

En el apartado "otras" se incluyen actividades bastante específicas y muy poco investigadas por ser menos comunes y, por ello, difíciles de registrar. Ejemplos de estas actividades son rascarse, luchar o cojear.

2.4 Tipos de sensores y su posición en el animal

Los sensores son los elementos electrónicos que transforman ciertas magnitudes físicas en magnitudes eléctricas, para ser posteriormente analizadas. En el campo de la AAR se emplean especialmente IMUs, sensores inerciales, aunque algunos estudios introducen de forma complementaria otros diferentes. En función de qué actividades quieran usarse para entrenar al modelo de ML, se habrá de contemplar el uso de unos u otros sensores, el número de ellos y también su ubicación en el cuerpo del animal a analizar.

En la bibliografía consultada se encuentra un claro predominio del acelerómetro triaxial, presente en todos los artículos consultados. Es este un sensor que mide la aceleración a la que está sometido un cuerpo cualquiera. Esta aceleración es tanto estática (aceleración de la gravedad) como dinámica (la provocada por el movimiento) [7] y puede proporcionar mucha información acerca de la actividad del animal. *Kleanthous et al.* [30] en su artículo de revisión de 2022 afirmaban que el uso de un acelerómetro es suficiente para identificar actividades tales como caminar, pastar, rascarse o permanecer tumbado con gran porcentaje de acierto. Defendían que la adición de sensores complementarios no parecía incrementar sustancialmente la precisión. Diversos autores consultados utilizan solamente este tipo de sensor para la toma de medidas y son capaces de predecir con bastante grado de acierto, lo que parece sustentar esta idea [7], [19], [28], [34], [36], [37], [39], [40].

También aparece el uso del giroscopio tri axial, un sensor inercial capaz de medir la orientación y la velocidad angular, que suele acompañar al acelerómetro. Autores como *Carslake et al.* [35] hacen uso de esta combinación con buenos resultados. Según su estudio, de todas las características extraídas con ambos tipos de sensores, entre las diez que más información aportan a la clasificación, prácticamente la mitad de ellas están relacionadas con medidas del giroscopio. Por eso [35] concluye que la inclusión de este sensor es clave para una buena precisión. Sin embargo, en otros estudios al analizar los resultados de añadirlo, matizan que, si bien es cierto que la precisión es más alta, el incremento del porcentaje de precisión es

bastante pequeño. Es decir, el acelerómetro parece ser suficiente para una buena clasificación [34], [38], [41].

Otro sensor bastante utilizado es el magnetómetro tri axial (o brújula digital), un sensor que detecta los cambios en el campo magnético en un lugar concreto, por lo que también es capaz de medir rotaciones. Los artículos consultados que lo utilizan lo usan junto a acelerómetros y giroscopios [33], [34], [41], [42] y, en general, no especifican si la precisión aumenta gracias a su uso o no. Sí se comenta en un caso que al hacer selección de las cinco características más importantes, una de ellas pertenece a un magnetómetro [42]. *Sakai et al.* [41] señalan que los resultados de hacer una predicción usando tan sólo la información del magnetómetro son muy similares, en cuanto a precisión, a los de usar solamente un acelerómetro por lo que serían intercambiables. *Riaboff et al.* [29] señalan que algunos artículos sí ven una mejora en la inferencia mientras que otros consideran que la mejora no es suficiente para justificar el consumo de energía de un dispositivo adicional, así que concluyen que es necesita más investigación al respecto.

Otros sensores que aparecen en la bibliografía son los de temperatura, humedad, y presión, pero su uso parece muy limitado. El único trabajo que comenta algo acerca de su uso señala que aunque en el *dataset* original aparecían medidas de dichos sensores ([33]) no fueron usados sus datos en sus modelos [42].

Otro aspecto clave relacionado con el sensado para AAR es el posicionamiento de los dispositivos en el animal, algo estrechamente relacionado con el conjunto de actividades a reconocer. A priori parece lógico pensar que, para observar varios comportamientos con los sensores inerciales, las ubicaciones de estos deban ser diferentes. Sin embargo, si se requiere una medición de un abanico de actividades amplio en el que varias partes del cuerpo del animal estén involucradas ¿cuál es la mejor solución?

La bibliografía consultada parece indicar que la posición más habitual de los sensores es el cuello [7], [8], [12], [28], [33], [34], [35], [36], [39], [40], [42]. Esto es algo que concuerda con lo observado en el artículo de revisión [29], que también indica que en esa posición es fundamentalmente usada para diferenciar actividad de inactividad, además de múltiples comportamientos de manera simultánea [30]. Entre los investigadores que usan collares para ubicar los sensores destaca el trabajo de *Meijers et al.* [34] donde se centran en buscar características independientes de la orientación del sensor. Como se ve en la Figura 1, en dicho trabajo se colocan seis módulos de sensores inerciales en diferentes lugares del collar del animal con diferentes orientaciones. De esta forma exploran un problema poco tratado en AAR, como es el del desplazamiento del collar respecto a su posición original durante la toma de datos.

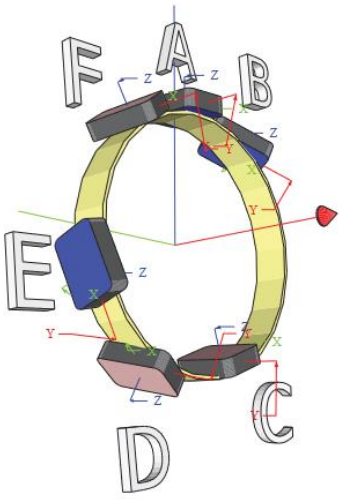

Figura 1: Posiciones y orientaciones de los sensores en el collar. Fuente: Meijers et al. - 2018

Otra posición bastante habitual es la pata del animal [29], [36] especialmente cuando se investiga la cojera de los animales o las diferentes formas de moverse: andar, trotar y galopar [30]. También ha sido usada para reconocer un conjunto de actividades más general: estar de pie, estar tumbado y caminar, pero con menos éxito [19].

En tercer lugar, por frecuencia de uso, *Riaboff et al.* [29] identifican a la oreja del animal, lugar común también en la bibliografía consultada [7], [36], [37]. Esta posición es utilizada para un reconocimiento general de la actividad, habiéndose empleado para distinguir entre animales con cojera y sin ella, clasificar la postura o reconocer si el individuo pasta, se tumba, está de pie o camina [30]. Aparte de dar buenos resultados cuando se trata de clasificar múltiples clases, esta ubicación para un sensor tiene la ventaja de que puede acoplarse al crotal o etiqueta que tiene el ganado para identificar a cada individuo [7].

Otras posiciones habituales son: la mandíbula [36], que da muy buenos resultados para el análisis de la alimentación [30]; el pecho entre el final del cuello y el comienzo de las patas delanteras; o la grupa, en la región escapular (Figura 2), en estos último casos empleando arneses [8], [29], [38], [41].

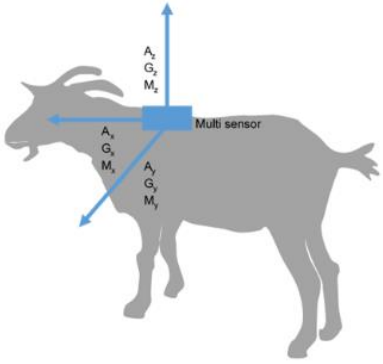

Figura 2: Sensor en la grupa del animal. Fuente: Sakai et al. - 2019

2.5 Etiquetado de los datos

En un proyecto de AAR, una vez seleccionados los animales que van a portar los sensores (IMUs) y elegido dónde se van a colocar en el cuerpo del animal, queda por definir la estrategia del etiquetado de datos. Esta estrategia define la forma en la que se va a asignar un comportamiento observado a una serie de medidas capturadas por los sensores.

Habitualmente la forma de proceder consiste en dejar a los animales en un entorno seminatural relativamente controlado, con los sensores registrando sus movimientos. Dichos sensores suelen estar sincronizados temporalmente con las grabaciones de vídeo de la actividad del animal que se realizan o bien desde puntos fijos o bien con cámara en mano siguiendo al animal desde una distancia que permita observar sin obstáculos la actividad del sujeto del experimento. Cuando las cámaras están en una posición fija, se debe tener visión de todo el recinto donde los animales desarrollen sus actividades, así como una forma de identificar a los individuos que estén participando en él. Un modo recurrente para identificarles es mediante el uso de colores diferentes en los collares o en los *espráis* de pintura que se les aplica sobre el lomo [30]. De esta forma se puede asociar cada sensor con la actividad que realiza su portador en la grabación de video.

Una vez alcanzadas las horas necesarias de grabación y de toma de datos, el siguiente paso es cotejarlos. Para ello, se extrae la información de los sensores y cámaras, y se asocian los intervalos temporales de medidas de los sensores con las diferentes actividades observadas en los videos. En la literatura consultada se ha observado que cada autor aborda este problema de una forma ligeramente diferente a los demás. En general esto se puede hacer manualmente [35], [40], con software propio creado *ad hoc* por el propio grupo de investigación [33] o mediante software diseñado por terceros [8], [38], [43], sea comercial [7], [36] o no [29], [30]

2.6 Procesado de los datos en bruto

Una vez que se ha conseguido elaborar un conjunto de datos, con las medidas de los sensores y las etiquetas de las actividades observadas, lo más habitual es procesarlo o acondicionarlo para facilitar obtener resultados satisfactorios en la fase de inferencia. La secuencia de medidas en bruto que se ha obtenido representa una única serie temporal que ofrece una información valiosa pero limitada. Por eso en la etapa de preprocesado se llevan a cabo una serie de pasos destinados a aprovechar lo máximo posible los datos recogidos [28], [29] y a darle un mejor contexto al clasificador del problema [44].

Resulta importante señalar que, si bien todos los trabajos de la bibliografía consultada realizan preprocesado de los datos en algún punto, cada uno lo hace de manera distinta. Es por ello por lo que la fase de preprocesado es la más heterogénea de todas las etapas de desarrollo de técnicas de AAR: cada autor elige en función de su experiencia, capacidades, limitaciones técnicas, intereses o contexto del trabajo cómo preprocesar sus datos. Todas las técnicas que se describen a continuación se han usado dentro de esta etapa de AAR en uno o varios artículos, aunque pocos de ellos las han empleado todas en el mismo trabajo. Es por ello por lo que a

continuación se presenta un análisis de las estrategias de preprocesado en los artículos de revisión, añadiendo a continuación otras aproximaciones incluidas en artículos específicos.

De la bibliografía consultada no se ha encontrado ningún análisis más detallado que el de *"Predicting livestock behaviour using accelerometers: A systematic review of processing techniques for ruminant behaviour prediction from raw accelerometer data"* [29], que cubre ampliamente lo tratado por el resto. Por ello, se emplea su forma de estructurar la presentación del preprocesado, dividiéndolo en cuatro apartados: limpieza de ruido y filtrado, cálculo de series adicionales, segmentación y cálculo de características.

Limpieza de ruido y filtrado:

El primer paso, después de observar cómo es el conjunto de datos disponible, es corregir los defectos que pueda haber en él debido a una mala transmisión de los datos [12] o a las diferentes frecuencias de muestreo de los sensores. Uno de los errores más comunes que suelen encontrarse es la ausencia de datos en algunas muestras (por ejemplo, que falte el valor de la aceleración en uno de los 3 ejes en los que se mide). Cuando esto sucede hay varias aproximaciones para la gestión de datos ausentes como eliminar todas las muestras a las que les falte una medida [12], [39], realizar una interpolación entre la medida anterior y la posterior [42] o escribir el valor medio [45] entre otras.

A continuación se aplica algún tipo de filtrado para la eliminación de *outliers* [45] o muestras que estadísticamente destacan por encima del resto y que suelen ser ruido producido por el sensor al, por ejemplo, moverse en su enganche en el animal. Para ello se pueden emplear diversas técnicas, desde umbrales estadísticos a algoritmos de ML [46].

También es recurrente el empleo de filtros clásicos para la eliminación de ruido. Por ejemplo, *Riaboff et al.* [28] emplean varios filtros paso bajo, uno a 5 Hz y otro a 10 Hz [28], [40], así como un filtro paso alto a 0.3 Hz de tipología *Buttersworth* de 6º orden. A veces se emplean también filtros para eliminar información que no interesa, como el efecto de la aceleración de la gravedad [28].

Por último, algunos trabajos introducen una fase de sobre-muestreo (añadir muestras a las etiquetas menos habituales) o infra muestreo (eliminar muestras de los comportamientos más repetidos). Estas técnicas tienen por objetivo equilibrar *datasets* desequilibrados, es decir, conjuntos de datos en los cuales algunas actividades son mucho más comunes que otras, algo que puede provocar pérdida de precisión o sesgos en el clasificador según cual se utilice. Los resultados y conclusiones acerca del uso de filtros son variados [37], [41].

Cálculo de series adicionales:

El conjunto de datos que se ha obtenido de los animales es, en sí mismo, una serie temporal. Sin embargo, con esos mismos datos se pueden calcular otras series temporales adicionales que complementen y amplíen la información recogida por la original. En la mayor parte de los casos se usa una combinación de varias series [29] para obtener la serie adicional.

Entre dichas series temporales, destacan las que son independientes de la orientación del sensor. Al margen de cómo se las denomine en la literatura, dichas series modelan el módulo de la aceleración y la calculan como la raíz cuadrada de la suma de las componentes de la aceleración en cada eje al cuadrado [19], [28], [34], [35], [39]. En este aspecto destaca el trabajo de *Meijers et al.* [34] que se centra en cómo usar esta serie temporal para la selección de características inmunes al movimiento del collar donde están integrados los sensores[1] [34].

Existen otras series adicionales que aportan información acerca del gasto de energía del animal, su aceleración dinámica o su inclinación, OBDA o VeDBA, entre otras [41]. También existen algunas relacionadas con bandas de frecuencia específicas [29].

Segmentación:

Consiste en agrupar una cantidad de muestras de la serie temporal de modo que constituyan la unidad mínima de análisis. Dependiendo del artículo, dicho conjunto de muestras se denomina ventana, intervalo o época. Por lo tanto, la ventana es la "**unidad estadística fundamental** en los análisis subsiguientes" [29]. Por ello, la segmentación es un paso crítico puesto que el resto del proceso (selección de características, entrenamiento del modelo…) dependerá de las elecciones que se hagan en este punto. Esto es así porque se divide una serie temporal continua en periodos con patrones de actividad concretos que se analizarán sin tener en cuenta (en principio) la dependencia temporal entre ellos. Las decisiones que se toman durante la fase de segmentación son fundamentalmente dos: el tamaño de ventana y la superposición entre ventanas. Ambas decisiones tienen un impacto directo en la precisión del algoritmo pero también en el uso de los recursos *hardware* [30].

El tamaño de ventana es la duración del segmento, medido en segundos, y se corresponde con un número concreto de las muestras del conjunto de datos. La elección del tamaño depende de la frecuencia de muestreo de los sensores y, especialmente, de qué actividades se quieran reconocer y de su duración. Por ello, tiene que ser una duración lo suficientemente grande como para poder resultar significativa en la predicción, pero también lo suficientemente pequeña como para no incluir varios comportamientos diferentes [30]. Esta decisión afecta directamente al rendimiento del modelo por lo se observa en los trabajos estudiados una gran variedad en las elecciones de tamaño: desde ventanas de menos de un segundo a otras de más de un minuto, siendo lo más habitual los tamaños de entre 1 y 10 segundos [7], [12], [29], [38], [39], [40]

Por otro lado, la superposición es el porcentaje de muestras que se repiten entre dos ventanas consecutivas [29]. Lo más habitual es que sea el 0% [7], [36] aunque también se han utilizado solapamientos de 25%, 50% [34], [35], [40], 75% o 90% [28], especialmente den aplicaciones que precisan capturar de forma más precisa las transiciones entre actividades [30], [36]. Incluir superposición entre ventanas puede ayudar a mejorar la eficacia de un clasificador al incrementar la cantidad de datos disponibles. Sin embargo, a mayor porcentaje de solapamiento mayor será también el número de operaciones y el consumo de energía [28].

[1] Véase apartado 2.4

Diversos autores han trabajado en la selección de un tamaño de ventana y una superposición óptima para sus proyectos, probando con varios valores y evaluando los resultados. El análisis de los resultados obtenidos produce resultados dispares. *Barwick et al.* [36] observaron que no había una diferencia significativa en el rendimiento entre ventanas de 3, 5 y 10 segundos. *Carslake et al.* realizan un análisis similar estudiando ventanas de entre 1 y 10 segundos con superposición del 50% variando la frecuencia de muestreo de los sensores. Encuentran que existe una solución de compromiso entre los 10 y 20 Hz que proporciona buenos resultados al adaptar el tamaño de ventana y que mantiene el consumo energético bajo. *Fogarty et al.* [37] prueban con tamaños de 5, 10 y 30 segundos pero sólo si cada ventana contiene una sola actividad, en caso contrario las descartan. Concluyen que en general las ventanas más grandes proporcionan mejores resultados, pero que debe seguirse investigando al respecto.

Cálculo de características:

Las características son los elementos de información que se extraen de los datos de la serie temporal (conocidos como variables [47]). Son los resultados de una serie de operaciones que se realizan con las variables por cada ventana y ayudan al clasificador a encontrar relaciones mejor. Es una etapa del preprocesado también muy abundante en la literatura, muy heterogénea y que únicamente no está presente en los proyectos en los que se emplean clasificadores de DL [29]. *Riaboff et al.* hacen dos clasificaciones de las características, por una lado en función de su dominio (temporales, frecuenciales o *wavelet*) y por otro en función del tipo de información que ofrecen (intensidad del movimiento, orientación, la forma de la señal y la descripción física del movimiento) [28], [29].

Características del dominio del tiempo: *Figo et al.* [44], que hacen el análisis más extenso, las clasifican en dos subgrupos, uno de funciones estadísticas y matemáticas donde incluyen la media, la mediana, la varianza, la desviación estándar, el máximo, el mínimo, el rango entre máximo y mínimo, la correlación, la integración… y otro grupo donde incluyen a las que denomina “otras funciones” que son la velocidad angular, los cruces por cero, la SMA, SVM entre muchas otras.

Características del dominio frecuencial: A través de la Transformada de Fourier u otras como la *Wavelet* se extrae información acerca de las señales de los sensores y su naturaleza periódica [44]. Algunas de ellas son la componente de DC, la energía espectral, la entropía espectral o la frecuencia pico [30], [35], [40]. Los estudios que las han empleado destacan que, a pesar de su utilidad (más robustas contra el ruido), destacan por su complejidad de cálculo. Esto implica un mayor consumo de energía [29], [44] y en algunos casos, no incluirlas no supone mucha pérdida de precisión [34].

Otras características: no pertenecen a los dominios anteriores y son muy poco abundantes en la literatura. Algunas de ellas son la distancia Euclidiana, la distancia de Leveshtein o la DTW (*Dynamic Time Warping*) [44]

Selección de características y reducción de dimensionalidad:

Después de haber extraído características de las series temporales de forma sistemática, queda analizar si toda esa información nueva es realmente relevante para el clasificador a emplear. Además, hay que tener en cuenta que un exceso de información puede ser demoledor para ciertos algoritmos de ML, incrementando enormemente su tiempo de entrenamiento. Para minimizar este problema, se emplean métodos de reducción de dimensionalidad y de selección de características, que pueden dividirse en métodos de filtrado, *wrappers* y soluciones híbridas y embebidas [30]

Métodos de filtrado: Analizan la cantidad de información (ganancia de información, ratio de ganancia…), las relaciones estadísticas (PCA, test de Chi Cuadrado [34], puntuación de Fisher [40]…) [39], [42], la consistencia o la puntuación de similitud (algoritmos como *Relief* o *ReliefF* [34], [35]) entre características de un espacio multidimensional. Estos métodos funcionan mejor que los *wrappers* y los métodos híbridos por ser más rápidos e independientes del clasificador. Sin embargo fallan al analizar las dependencias entre características [30].

Wrappers: Utilizan un entrenamiento recursivo del clasificador, empleando conjuntos diferentes de características y analizando cuales dan el mejor resultado. A esta categoría pertenecen los algoritmos de tipo *"greedy"* (*Forward Selection*[34]*, Backward Elimination* [38]) u otros más complejos como Boruta [42]. Estas técnicas son costosas desde el punto de vista computacional [30].

Híbridos y embebidos: Están a medio camino entre *Wrappers* y los métodos de filtrado. Son muy variados y generalmente utilizan varios algoritmos como SVM, CART o XGBoost [30]. Por ejemplo, *Barwick et al.* implementan un método basado en RF con medición de la importancia de las características basadas en un índice *Gini* , que también emplean *Fogarty et al.*[7], [36], [37].

2.7 Algoritmos empleados

En la bibliografía consultada, para AAR se emplean mayormente algoritmos de ML pertenecientes al subgrupo que clasifica mediante un aprendizaje supervisado. Esto quiere decir que son algoritmos que necesitan un conjunto de datos etiquetados[2] para poder ser entrenados. Los algoritmos de aprendizaje no supervisado descubren por sí mismos las relaciones subyacentes en los datos de entrada y los agrupan con etiquetas. En ocasiones se usa una combinación de ambos tipos de algoritmos como método de reducción de la dimensionalidad [30] o de selección de las características más relevantes para la predicción[3].

Los artículos de revisión consultados muestran la amplitud de este campo de investigación, al presentar análisis bastante extensos de los algoritmos empleados en el estado del arte. *Kleanthous et al.* [30] en su artículo de 2022 reconocen el uso de al menos 14 clasificadores diferentes, donde los más comunes son LDA (*Linear Discriminant Analysis)*, RF

[2] Véase apartado 2.5
[3] Véase apartado 2.6

(*Random Forest*), QDA (*Quadratic Discriminant Analysis*), K-NN (*K-Nearest Neighbors)*, CART (*Classification And Regression Trees*), LSVM (*Linear Support Vector Machines*), ANN (*Artificial Nerural Network)* y NB (*Naive Bayes*).

Riaboff et al. clasifican los algoritmos usados en AAR en varios grupos. El más relevante, al que llama *"Supervised Machine Learning",* lo forman lo que se podrían denominar algoritmos de ML clásicos: los ya comentados LDA, QDA, SVM, K-NN y NB a los que añade el DT (*Decision Trees*) que es la base de RF. Mencionan también un dato relevante y es que el afinado de los hiper parámetros que configuran los algoritmos de AAR, se suele hacer a través de la *Grid Search*.

Por relevancia, pone en segundo lugar a los métodos de clasificación denominados "*Ensemble"*. Estos son métodos constituidos por conjuntos de algoritmos clásicos cuyo ejemplo más paradigmático es *Random Forest*, pero donde también menciona a XGB (*Extreme Gradient Boosting*) y a ADAboost [29].

Por último, cabe mencionar el grupo de los algoritmos de DL, que, aunque no son tan habituales en su revisión como otros, resultan relevantes para este proyecto por sus particularidades y su potencial. Sobre estos aspectos también encontramos información en [31] donde se presentan algoritmos como los MLP, que son FFNN (*Fully connected Feedforward Neural Networks*), o las CNNs (*Convolutional Neural Networks*). También enumeran algunos clasificadores dentro de las RNNs (*Recurrent Neural Network*) como LSTM o GRU y varios modelos híbridos como CNN-LSTM y CNN-GRU [31].

Entre los artículos de la literatura revisada hay varios que emplean diferentes clasificadores con el mismo conjunto de datos, a fin de establecer comparaciones entre ellos, de forma similar al trabajo que se expondrá más adelante en este documento.

En "*Behaviour classification of extensively grazed sheep using machine learning*" de *Fogarty et al.* [37], se analizan cuatro clasificadores diferentes con el objetivo de reconocer cuatro actividades: pastar, estar tumbado, estar de pie y caminar. La estrategia de desarrollo de AAR adoptada en [37] hace que, por un lado, todas las características extraídas sean usadas para entrenar modelos basados en SVM y CART. Por otro lado, para los modelos basados en LDA y QDA, se realizará antes una selección de las tres características más importantes mediante un algoritmo de *Random Forest* (reducción de dimensionalidad). El trabajo también pretende analizar qué tamaños de ventanas son óptimos para una predicción. Los resultados presentados en el artículo son dispares. Por un lado, para predecir las cuatro actividades, SVM es el algoritmo que mejor funciona y lo hace con ventanas de 10 segundos, pero no pasa del 76.9% de precisión. Una de las razones es que clasifica muy mal la actividad de caminar y eso empeora la media. Por otro lado, para discernir si el animal está activo o no, CART con ventanas de 30 segundos es lo que mejor resultado proporciona (98.1%), según [37]. Por último, para [37], LDA con ventanas de 30 segundos es la mejor combinación para predecir la postura (90.6%) .

Kamminga et al. en su artículo *"Generic online animal activity recognition on collar tags"* [33] comparan los resultados de siete clasificadores diferentes, DT, SVM, KNN, LDA, NB, NN y DNN. No solamente analizan la precisión de la predicción de cada uno de ellos, sino que también extraen conclusiones del análisis del uso de memoria y tiempo de CPU dedicado al

entrenamiento y la inferencia. Sus resultados muestran que en la fase de entrenamiento los que resultan más costosos en términos de memoria y CPU son los dos tipos de algoritmos de DL y los que menos recursos requieren son NB y KNN. En la fase de inferencia en cambio, el uso de memoria es mínimo entre DT, NB, NN y DNN y los algoritmos que menos tiempo de CPU consumen son NB, DNN y NN. Como su estudio está realizado con la idea de minimizar el consumo de energía en el dispositivo (computación *on-edge*) el mejor en ese caso resulta ser NB. Al analizar la precisión y la *F-score (métricas que se estudiaran en el siguiente apartado)*, todos los clasificadores dan buenos resultados, destacando KNN en ambas métricas y seguido de los algoritmos de DL y NB. Por todo esto, concluye que DNN es el modelo más prometedor por su bajo uso de recursos en la fase de inferencia y los buenos resultados de la predicción.

En *"Machine Learning Techniques for Classification of Livestock Behavior", Kleanthous et al.* usan los mismos datos [42] que en el caso anterior con la diferencia de que emplea *Boruta* para seleccionar las características más relevantes. A continuación, entrenan cuatro clasificadores diferentes: MLP (*Multi-Layer Peceptron*), RF, XGB y KNN. Los resultados obtenidos son también sobresalientes, con precisiones superando el 92% y destacando XGB, que supera o iguala los resultados del resto de clasificadores en todas las métricas, seguido por RF. El artículo concluye que los resultados son tan buenos porque las características extraídas por *Boruta* son realmente relevantes y diferencian entre las cuatro actividades a reconocer.

Usando también el mismo conjunto de datos, *Meijers et al.* aplican un filtro Chi Cuadrado y realizan una *Forward Selection* de características, empleando para ello dos algoritmos diferentes, DT y NB [34]. Para ello se generan dos subgrupos diferenciados por el algoritmo de selección y, posteriormente, se evalúan los resultados de entrenar seis clasificadores diferentes: DT, NN, SVM, NB, LDA y KNN. El mencionado trabajo concluye que, si la selección se hace correctamente, a partir de 3 características la precisión no aumenta significativamente al añadir nuevas características. En [34] también se llega a la conclusión de que NB es mejor que DT para la selección de características. Puesto que NB es más sensible a ruido y a características irrelevantes, es más estricto a la hora de escoger aquellas que maximicen el resultado final. El clasificador que más precisión consigue es la red neuronal, NN.

Sakai et al. [41] emplean dos clasificadores distintos, como KNN y DT, y compara sus resultados empleando datos de giroscopios y magnetómetros con y sin infra-muestreo. Con ello pretenden evaluar cuales son los beneficios de balancear los datos. Encuentran que a KNN no le beneficia esa estrategia por ser un algoritmo muy dependiente de la cantidad de datos y al infra-muestrear se pierde información. En cambio, al clasificador DT sí le ayuda ya que padece de cierta tendencia a producir *overfit* y, al evitar que haya actividades con más muestras que otras, se evita que acabe estando sesgado hacia las más habituales.

En el resto de los trabajos consultados se emplean algoritmos como QDA ([7], [36]), RF ([38], [39]), DT ([19], [28]) o ADABoost [35] entre otros, con diferentes objetivos y resultados variados.

2.8 Evaluación de las predicciones

Al final de todo el proceso queda evaluarlo, para ver si las elecciones que se han ido tomando y que han culminado en el desarrollo de un modelo de ML han sido las correctas.

El proceso de validación del modelo puede realizarse de diversas formas, partiendo de la base de que, para evaluar el comportamiento de un modelo, se deben usar datos que **no** hayan sido empleados para el entrenamiento de este [48]. Por ello se analiza la capacidad de inferencia y generalización, estudiando los resultados de un modelo cuando se le suministran datos que no ha visto nunca. Para ello se comparan las actividades predichas con las que se observaron para dichos datos y se obtiene el porcentaje de acierto.

Existen diferentes modos de abordar la división del *dataset* para la validación. Una de las técnicas más sencillas es la división de todo el conjunto de datos en dos de tamaño desigual: uno de entrenamiento (70% aproximadamente de todas las muestras) y otro de prueba (aproximadamente 30%). Otro método habitual es usar *Cross-Validation* [12], [28], [33], [34], [35], [39], [40], [41], [42], consistente en dividir el conjunto entero en K subgrupos, usar para el entrenamiento K-1 de ellos, validar con el subgrupo restante, repetir este procedimiento K veces y obtener la media de las K iteraciones [29]. Los valores más habituales para K son 3, 5 y 10.

Similar a Cross-Validation existe LOOV (Leave-One-Out-Validation), también bastante empleado y que puede ser una forma más adecuada cuando existe cierto grado de variabilidad entre los sujetos del experimento [7], [28], [37], [38].

La forma en la que se separan los datos para generar subgrupos que usar en la validación varía entre autores: puede ser aleatoria, dependiente del tiempo o dependiente del animal, siendo estos dos últimos bastante menos frecuentes [29]. En la separación aleatoria, se emplea con frecuencia una división estratificada, que fuerza a que las muestras aleatorias elegidas sigan la distribución de clases del *dataset* original [28], [48].

Para evaluar las predicciones se usan una serie variada de métricas procedentes de los campos de la estadística, la medicina y el análisis de poblaciones. Las más habituales son las que derivan directamente de la matriz de confusión, y que, hasta cierto punto, resultan intuitivas: precisión (*accuracy,* Ecuación 1*)*, sensibilidad (*sensitivity*), especificidad (*specificity*) y exactitud *(precision/recall)*. Además, con bastante frecuencia se emplean también la *F-score* (Ecuación 2Ecuación 2), el coeficiente Kappa de Cohen o el Área bajo la curva (AUC, *Area Under the Curve*) [29]. La *F-score* se suele emplear en lugar de las métricas sensibilidad y exactitud pues consigue aunar en una sola medida la relación de ambas con la precisión.

El uso de estas métricas no es homogéneo y varía con el trabajo consultado; algunos artículos sólo usan la precisión para evaluar el modelo [8], [40], otros emplean adicionalmente diferentes combinaciones del resto de métricas mencionadas para caracterizar mejor a sus modelos [7], [12], [33], [34], [35], [36], [37], [38], [39], [41], [42]. También existen diferentes aproximaciones a la hora de tratar problemas multiclase (reconocimiento de más de dos actividades). Algunos trabajos optan por ofrecer métricas que promedian los resultados de todas las actividades predichas y otros las analizan individualmente. Lo recomendable es adoptar ambas [29].

$$Precisión = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

Ecuación 1: Precisión

$$F_{score} = \frac{2T_p}{2T_p + F_p + F_n}$$

Ecuación 2: F-score

2.9 Retos de la AAR

Como se ha expuesto brevemente antes en el apartado 1.1, existen una serie de dificultades asociadas al uso de IMUs en animales con el fin de predecir su comportamiento. Cómo adquirir los datos provenientes de los animales y asociar cada muestra a un comportamiento observado para entrenar un algoritmo; cómo gestionar las limitaciones de consumo, potencia, capacidad de procesado o de uso de memoria de los equipos de medida, de transmisión o de tratamiento de dichos datos; cómo adaptar los equipos y los procesos a animales diferentes en zonas geográficas u orográficas distintas; cómo ponderar si la solución elegida realmente es válida son, a grandes rasgos, algunas de esas dificultades aunque, por supuesto, a medida que se va al detalle aparecen muchas más. Además, en cada fase del desarrollo de técnicas de AAR la tipología de los retos es totalmente diferente a la del resto de fases, desde problemas mecánicos con las sujeciones de los IMUs[4] a problemas de sensibilidad del modelo desarrollado, pasando por todas las dificultades relativas a la gestión de los recursos *hardware*.

Por eso, en este apartado se exponen tan sólo los retos relacionados con los objetivos del trabajo y que vienen bastante bien condensados en el artículo [31]. En él, el autor reflexiona acerca de ellos en el contexto del uso de herramientas de DL en AAR, siendo sus conclusiones extrapolables en gran medida a cualquier proyecto con ML.

En el trabajo [31] se divide el desarrollo de técnicas de AAR en tres partes diferenciadas, se analizan los retos que se presentan y se proponen cauces de acción para afrontarlos. En la fase de adquisición de datos de los animales, [31] señala dos dificultades: la gestión de la energía y también la escasez de anotaciones que está relacionada con la necesidad de muchos algoritmos de recibir grandes cantidades de información para el entrenamiento. En el apartado de desarrollo del modelo destacan el desequilibrio de clases y la similitud entre actividades, dos problemáticas muy presentes en el desarrollo de esta investigación como se verá más adelante en el documento. Por último, al hacer la inferencia, *Mao* et al. [31] vuelven a incidir en la eficiencia energética y exponen también las dificultades implícitas de la generalización del dominio. Esto algo que también se verá más adelante en este trabajo al experimentar qué sucede cuando a un modelo entrenado con los datos de un tipo de animal se le pide que haga la inferencia con datos de otro ligeramente distinto.

Las soluciones presentadas por la comunidad científica a estos retos han sido muy variadas y han ido evolucionando con el paso de los años y el avance de la técnica. Algunas de ellas se verán implementadas en los siguientes apartados.

[4] Véase apartado 2.4

Capítulo 3: Metodología

A partir del estudio del estado del arte realizado en el capítulo anterior, se define en esta sección cual debe ser la estructura genérica de un proyecto de desarrollo de aplicaciones de AAR. Además de identificar las fases de dicho proceso, se definen las acciones a realizar en cada fase y se analiza el impacto que algunas acciones relevantes pueden tener en el resultado final (por ejemplo, como afecta la selección del etograma a la técnica AAR). Como el objetivo principal de este trabajo es determinar cuáles son los parámetros óptimos para desarrollar una aplicación de AAR, el siguiente paso es determinar cuáles son las etapas del proceso de desarrollo que deben ser estudiadas y que parámetros deben ser explorados para obtener una solución óptima. Además de justificar la selección fases y parámetros a definir, se analizan las razones de no seleccionar otras etapas y parámetros, así como las limitaciones que esta decisión produce. Por último, se define como deben ser explorados los parámetros o grados de libertad identificados para obtener una solución óptima.

3.1 Análisis de antecedentes.

En el capítulo anterior se ha realizado un estudio detallado de las diferentes opciones que se han propuesto para lleva a cabo un desarrollo orientado al reconocimiento automático de la actividad animal mediante el uso de herramientas de ML. Queda probado que la AAR puede abordarse de muchas formas diferentes, y que hay que adaptar el procedimiento a la casuística y tipología de cada caso. Sin perjuicio de que algunos artículos analizados en el capítulo anterior no incluyan todos los pasos del proceso, todos utilizan las siguientes fases:

1.- Captura de datos: etapa en la que a partir de las especificaciones del proyecto de desarrollo se toman ciertas decisiones, como el tipo de animal a analizar, los sensores y dispositivos que se van a usar para la captura de datos, dónde van a estar ubicados, qué procedimiento se va a usar para la anotación de la actividad y qué formato va a tener el *dataset* (Figura 3).

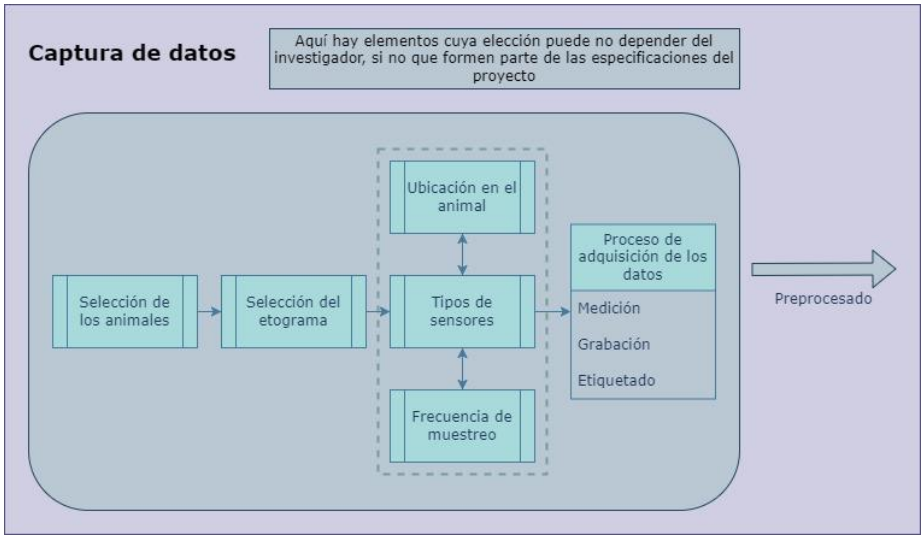

Figura 3: Diagrama de la fase de captura de datos

2.- Preprocesamiento de los datos: etapa en la que se tratan y procesan los datos capturados con el fin de que el clasificador obtenga resultados óptimos. Se suele comenzar eliminando ruido y gestionando los valores ausentes en la serie temporal. Luego algunos trabajos añaden series adicionales, donde la más común es el módulo de la aceleración, para acto seguido, segmentarlas en ventanas con cierto grado de superposición. De la información de las series temporales contenidas en cada ventana se extraen nuevas características de diversos tipos. Finalmente se incluye el paso de la selección de características, que emplea diferentes técnicas para reducir la dimensionalidad del *dataset* y mantener la información más significativa, lo que ahorra bastante tiempo durante el entrenamiento de los clasificadores (Figura 4 y Figura 5).

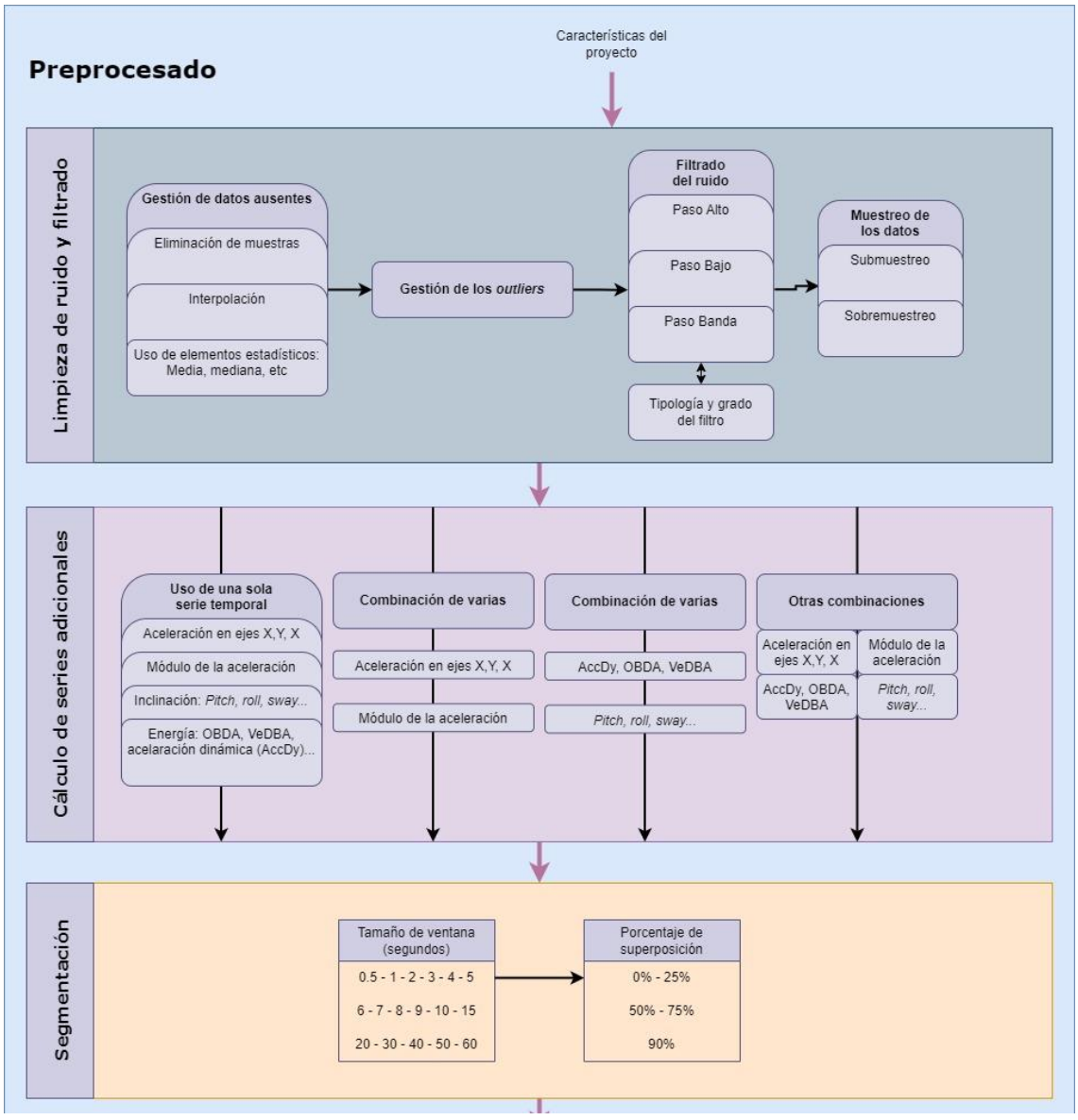

Figura 4: Diagrama de la fase de preprocesado (I)

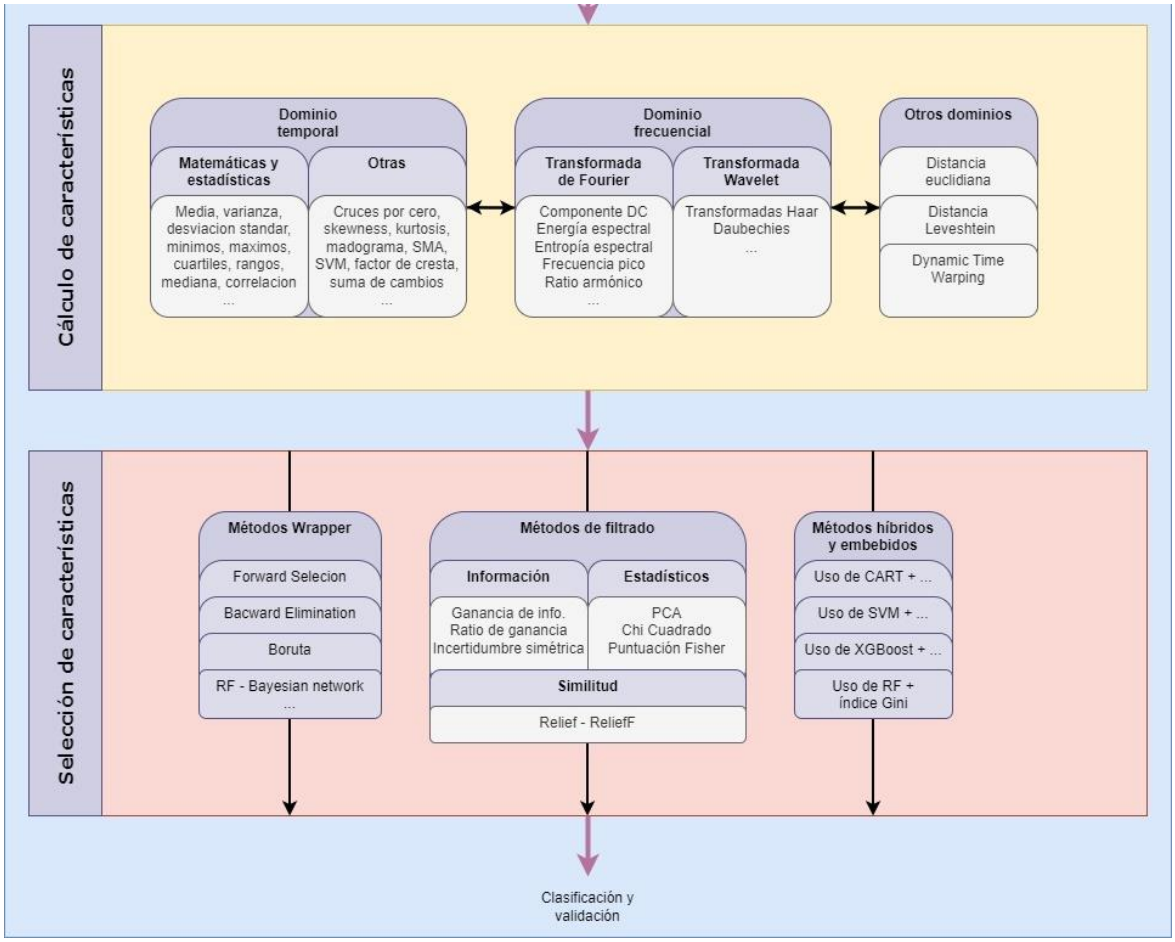

Figura 5: Diagrama de la fase de preprocesado (II)

3.- Clasificación y evaluación de las predicciones: se aplica un escalado al *dataset* si el clasificador lo requiere y se dividen los datos de diferentes formas según cómo se vayan a hacer el entrenamiento y la evaluación. Tras la elección del clasificador y el afinado de sus hiper-parámetros, se entrena el clasificador con el subconjunto de datos de entrenamiento. A continuación, se evalúa del modelo y se generan las métricas correspondientes para determinar su precisión y calidad. Si el resultado es satisfactorio, el modelo está listo para su implementación y despliegue. En caso contrario se podría repetir el procedimiento desde la etapa de preprocesado o desde la de clasificación, cambiando las fases que se consideren pertinentes (Figura 6).

Como conclusión, en este trabajo se va a utilizar un proceso de desarrollo de aplicaciones de AAR en 3 fases. Por lo observado en la literatura, una estructura rica en detalles, con etapas diferenciadas es clave para obtener una mayor compresión del problema y la casuística que lo rodea. También permite abordar soluciones más específicas que ofrezcan mejores resultados.

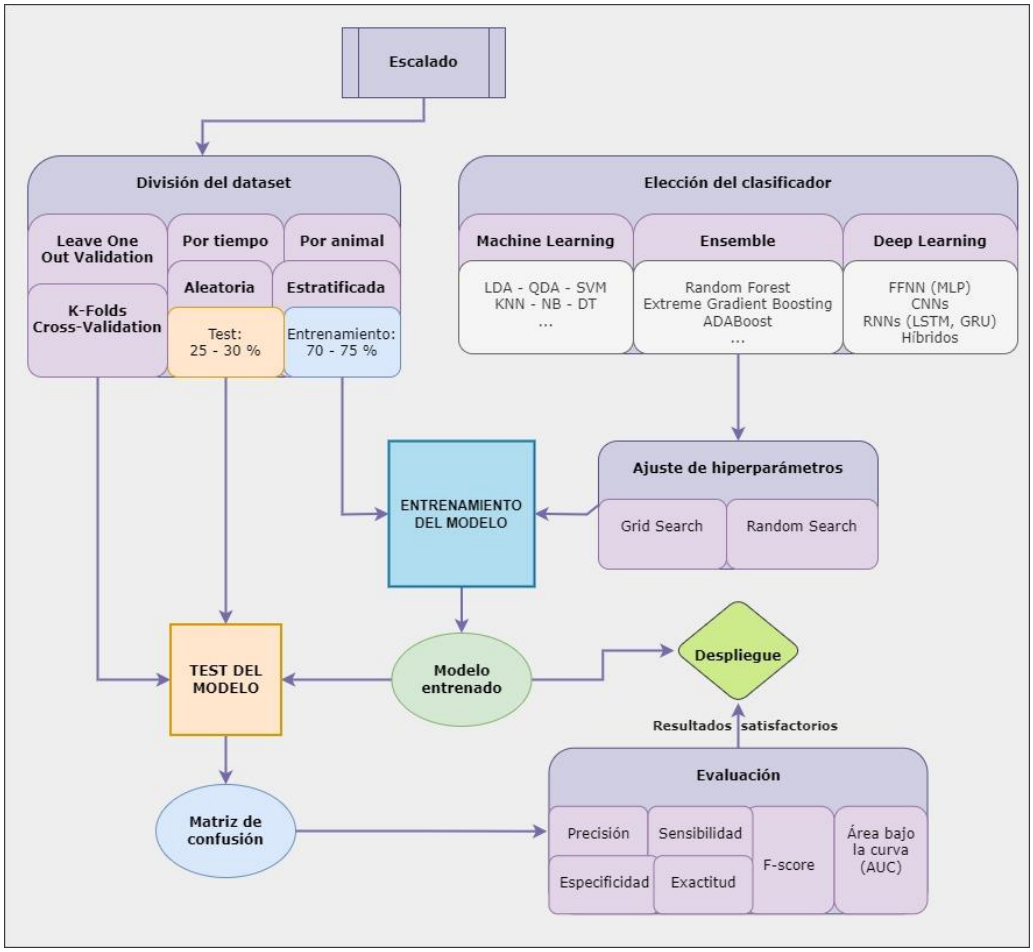

Figura 6: Diagrama de la fase de clasificación y evaluación

En cuanto a la **elección de los animales**, del análisis bibliográfico llama la atención la heterogeneidad y la escasez de estudios que se centren en las cabras. En cualquier caso, se extrae la idea fundamental de que la elección de la especie afecta y condiciona profundamente al resto del trabajo. Aunque la estructura subyacente se mantenga, en cada una de las fases las elecciones y decisiones que se adopten pueden ser radicalmente diferentes entre animales, incluso a pesar de que sean similares físicamente y de la misma familia, como es el caso de cabras y ovejas, los comportamientos pueden diferir en gran medida.

Por otro lado, parece interesante señalar que cuanto más heterogéneo sea el grupo de análisis dentro de una variedad de animales, más robusto será el modelo final al poder clasificar mejor las mismas actividades de animales con características físicas diversas. Esto es especialmente interesante de cara a crear modelos que generalicen bien y eviten los problemas de deriva de concepto o de deriva de datos.

La **elección del etograma** también resulta un aspecto clave del proceso. Hay que tener en cuenta que escoger un etograma variado, que abarque prácticamente toda la actividad realizada por el animal, en vez de usar un etograma más limitado, no tiene por qué ser una buena elección. Esto es debido a que un etograma más grande requiere de una cantidad de datos mucho mayor, dado que no todas las actividades son realizadas por un sujeto de forma homogénea a lo largo

del día, por lo que habrá algunas de las que apenas se recaben datos. Para clasificar bien se necesita mucha información sobre cada una de las etiquetas; si una de ellas es poco habitual y no se tienen suficientes datos, la probabilidad de que el modelo la prediga mal es bastante alta, reduciendo la precisión media. Esta problemática está muy relacionada con los retos de la escasez de anotaciones y el desequilibrio de clases que se han comentado anteriormente[5]. Por esta razón, al definir el conjunto de actividades que se quieren monitorizar hay que tener en cuenta la frecuencia con la que el animal las realiza.

Por otro lado, a partir del estudio de artículos orientados al **uso de sensores**, es posible concluir que existen suficientes evidencias como para afirmar que el uso de un acelerómetro es suficiente para abordar eficientemente el reconocimiento automático de la actividad animal. Emplear giroscopios o magnetómetros adicionales puede mejorar ligeramente la precisión de las predicciones, pero su inclusión debe estar supeditada a los requisitos de consumo energético del proyecto de desarrollo. Si las restricciones son muy exigentes, prescindir de ellos no debería comportar un problema en la predicción si el resto de las decisiones de diseño son adecuadas (como la extracción y selección de características). También cabe señalar que, en general, para el reconocimiento de actividades de un etograma generalista, el mejor lugar para colocar el sensor y el más empleado es el cuello del animal. Para reconocer entre conjuntos de actividades más específicos y limitados (patrones de alimentación, patrones de movimiento…) habría que valorar otras ubicaciones para los sensores, como las patas o la cabeza, por ejemplo.

El **etiquetado de los datos**, el procedimiento por el cual se cotejan las medidas extraídas de los animales con las anotaciones hechas acerca de su comportamiento, es quizás la parte más "artesanal" y menos automatizada de todo el proceso. Esta falta de automatización empieza por la propia forma de realizar las anotaciones y observar la actividad del animal mediante grabaciones de video que deben ser procesadas manualmente. La forma ideal de etiquetar los datos resulta muy dependiente de las características del entorno en el cual están los sujetos a observar, por lo que las soluciones requieren de bastante coste y esfuerzo.

En cuanto a las características del **preprocesado**, queda patente en el estudio de los antecedentes que no existe una forma de proceder que sea óptima para cualquier proyecto de desarrollo, si no que depende de muchos factores. Es una fase que se presta mucho a la investigación y a la exploración de nuevas soluciones. De forma similar, la etapa de **elección del algoritmo** y el afinado de los hiper parámetros es totalmente dependiente de las características de la investigación y de las decisiones que se hayan tomado en etapas previas. Cada clasificador destaca en áreas diferentes y su uso está supeditado a la evaluación de sus fortalezas y debilidades, con lo que tampoco se puede afirmar con rotundidad que uno sea mejor que los demás. Sin embargo, sí que queda patente que para la **evaluación de los resultados** el uso de sólo un indicador, como es el de la precisión, resulta insuficiente. Es muy recomendable incluir más indicadores que ofrezcan una visión completa y enriquezcan el análisis, como se ha hecho en este trabajo.

[5] Véase capítulo 2.9

3.2 Definición de la metodología

En el apartado anterior, se ha definido el procedimiento que hay que adoptar para desarrollar un proyecto de AAR mediante herramientas de ML. Se ha mostrado también qué partes lo componen y cuáles son los elementos más recurrentes en ellas. El siguiente paso es definir la metodología que permita encontrar la configuración optima de dicho procedimiento. Para ello, es necesario identificar los grados de libertad que se pueden usar para guiar el problema a la mejor solución posible. Observando las figuras del apartado anterior se puede llegar fácilmente a la conclusión de que los grados de libertad que se tiene en proyectos de esta naturaleza es muy alto. Además, en todos los niveles se tienen que tomar decisiones y de muchas de ellas no se puede evaluar el resultado hasta el final de todo el proceso. Aunque las combinaciones de parámetros son múltiples, sin embargo, se pueden agrupar en seis grupos independientes, en seis grados de libertad (Figura 7)

Figura 7: Grados de libertad de un proyecto de AAR y ML

Cada uno de los grados de libertad de la imagen anterior está compuesto a su vez de otros muchos, pero sirve para ilustrar cómo se va a enfocar este trabajo de este proyecto de manera más sencilla. De los seis grados de libertad mostrados en la figura anterior solamente se van a

utilizar cuatro: el etograma, la configuración del preprocesado, las características obtenidas de los datos y el clasificador.

La selección del tipo de sensor y su ubicación están fuera del ámbito de análisis de este trabajo debido a varias razones. Por un lado, dichas decisiones forman parte de la etapa de captura de datos y encontrar una manera óptima de abordar dicha cuestión constituye en sí mismo un problema específico. De hecho, en el grupo de investigación en donde se desarrolla este proyecto ya ha habido trabajos previos enfocado específicamente a esta cuestión [43]. Por otro lado, carece de sentido intentar generalizar la mencionada etapa puesto que depende completamente del contexto en el que se desarrolla el proyecto. Como se ha comentado antes, las soluciones adoptadas para la captura de datos son prácticamente artesanales, dependiendo de la variedad de los animales, de la época del año, de la orografía y del material del que se disponga para las grabaciones, entre otros muchos factores.

Es por este cúmulo de razones por las que se parte de un *dataset* ya confeccionado, al igual que los trabajos [34], [42]. El dataset utilizado es público y fue desarrollado por un grupo de investigación de la universidad de Trente, dirigido por *Jacob W. Kamminga* [33]. Para los entrenamientos de modelos que se realicen en este trabajo, se emplearán únicamente los datos asociados a las cabras, no a las ovejas. Sin embargo, sí se emplearán para evaluar la genericidad de dichos modelos como se explicará más adelante.

El proceso que se ha definido para obtener una técnica de AAR eficiente se puede dividir en dos apartados. En el primero, se busca la estrategia de preprocesado óptima. Para ello se parte del *dataset* mencionado y se aplican cambios y transformaciones a sus datos, generando múltiples *datasets* intermedios diferentes. En el segundo apartado se trabaja con los clasificadores. A partir de cada uno de estos *datasets* intermedios se entrenará un modelo diferente por cada uno de los clasificadores elegidos y se evaluarán comparativamente los resultados de todos ellos.

Creación del modelo de referencia para cada combinación del preprocesado.

En este trabajo, uno de los objetivos planteados es realizar, en base a un clasificador *Random Forest*, un modelo de referencia para cada combinación de los elementos posibles de preprocesado. Este modelo de referencia servirá para evaluar la calidad de otros clasificadores con el mismo preprocesado. Para ello, se van a seleccionar una serie de características relevantes con las que crear distintos conjuntos de valores de entrada para el clasificador. Como se observa en la Figura 7, en el caso del preprocesado esto implica poder variar valores tales como tamaño de ventana, porcentaje de superposición y frecuencia de muestreo. Sin embargo, como se parte de un *dataset* ya creado, no se puede modificar la frecuencia de muestreo, algo que también suele suceder cuando se parte de un modelo concreto de sensor comercial pues no todos soportan el cambio de dicha frecuencia. En este proyecto, la frecuencia de muestreo se ha fijado en 200 Hz.

En la bibliografía se definen los tamaños de ventana y la superposición con los que se van a hacer pruebas. Como existe una gran variedad de tamaños de ventana, se eligen ventanas de 1, 2, 3 y 10 segundos. Se escogen estos tamaños porque se desea analizar cómo evolucionan la

precisión y la *F*-score ante incrementos pequeños del tamaño de ventana, de uno en un segundos. La ventana de 10 segundos se emplea en cambio para comparar si los resultados observados con ventanas de tamaños menores son extrapolables, es decir, si las tendencias que se hayan podido observar, se mantienen y aplican para tamaños superiores. Al muestrear a 200 Hz estas ventanas equivalen a 200, 400, 600 y 2000 muestras consecutivas. Respecto al grado de superposición se probará con valores del 0, 25, 50 y 75%.

Por otro lado, se observa que en la Figura 4 y en la Figura 5 existen bastantes más aspectos relevantes para tener en cuenta durante el preprocesado de datos a parte de la superposición y el tamaño de ventana que se presentan en la Figura 7. En el subapartado 3.3 de este mismo capítulo se explicará por qué algunos no han sido tenidos en cuenta y con otros no se han probado diferentes configuraciones.

Otro de los grados libertad identificados como aspecto a explorar es la selección del etograma. Aunque el *dataset* seleccionado tenga un etograma definido, en el que recogen hasta nueve actividades (estar tumbado, estar de pie, pastar, luchar, agitarse, rascarse, caminar, trotar y correr), se ha optado por trabajar con dos conjuntos diferentes de actividades, el original y otro que incluye únicamente las cuatro actividades más comunes. De esta forma se puede analizar qué sucede cuanto el conjunto de datos está desequilibrado.

Existen tres razones por la cuales se elige un etograma con cuatro actividades en vez de uno de tres o de cinco. En primer lugar, porque en la bibliografía consultada es más habitual encontrar conjuntos con ese número. En segundo lugar, porque la distribución de muestras es similar; no es homogénea, pero existe menos desequilibrio entre las cuatro más comunes que entre las cinco, ya que la quinta se distancia más de las tres primeras que la cuarta. Y, por último, porque a pesar de que las tres actividades más comunes están muy equilibradas, resulta un etograma un poco pobre y existe el riesgo de que no refleje bien los cambios en el preprocesado si la clasificación siempre da buenos resultados.

En tercer y último lugar, está la extracción de las características. La cantidad de características que se pueden extraer de una secuencia temporal es muy alta, con lo que resultaría inviable evaluar el impacto sobre el resultado final de cada una de las que se utilicen. Por ello se usan dos conjuntos de características del dominio temporal, inspirados por la división que hace *Figo et al.*[44] uno donde se recogen las características estadísticas y matemáticas más sencillas (media, mediana, cuartiles, etc.) y otro donde se recogen las “otras” (asimetría, curtosis, SMA, etc.). Primero se probará sólo con el subconjunto de las más sencillas y luego con ambos simultáneamente.

La razón por la cual se opta por esta división es simple. Por un lado, se busca experimentar qué precisión ofrecen las características más básicas y sencillas de calcular como son la media, la mediana, la varianza, la desviación estándar, el valor mínimo, el valor máximo, el rango entre mínimo y máximo, el cuartil 25, el cuartil 75 y el rango entre los dos cuartiles. Las grandes ventajas que tienen estas características son su sencillez y que no requieren mucha potencia de cálculo, ni para el entrenamiento ni para la inferencia.

La adición de un segundo conjunto de características menos evidentes obedece al interés por buscar aquellas que den mejores resultados y maximicen la precisión. No son tan sencillas

de calcular como las anteriores, pero aun así no necesitan tampoco muchos recursos. En este trabajo se han empleado las siguientes: curtosis, asimetría, entropía, energía, cruces por cero, SMA, variación de movimiento, media de cambios en valor absoluto, la correlación entre cada pareja de ejes y el valor RMS del módulo.

Cabe señalar que no todas esas características se han calculado sobre la misma serie temporal. Antes de la extracción de características se generó la serie adicional con el módulo de la aceleración, para evitar problemas con los cambios de orientación. Por otro lado, hay características que no se utilizan en el clasificador. A pesar de utilizar las variables originales para el cálculo de la SMA, la variación de movimiento, las correlaciones entre ejes y la energía, no se emplean para el clasificador pues se eliminan del *dataset* tras hacer la extracción de las características mencionadas. La razón tras esto está relacionada con la búsqueda de soluciones independientes de la rotación de los sensores, como en [34] pues el empleo de dichas variables podría entorpecer la búsqueda de un modelo de referencia que pueda ser generalizable.

Como resultado del análisis anterior, se obtienen a partir del *dataset* original, 64[6] (Tabla 1) *datasests* con diferente preprocesado. Dichos *datasests* se utilizan para entrenar el modelo de referencia. En primer lugar, se realizará un escalado de los datos, para acto seguido eliminar las características menos relevantes con *Backward Elimination*, afinar los hiper-parámetros del *Random Forest* y finalmente entrenar el modelo con cada uno de los 64 *datasets* generados previamente. Del por qué se emplean estos elementos se hablará en el siguiente apartado.

La razón por la cual se utiliza como referencia un algoritmo del tipo *Random Forest* es su buen equilibrio entre consumo, complejidad y resultados. RF es un tipo de clasificador recurrente en la literatura, capaz de afinarse mucho mediante los hiper-parámetros y que presenta un punto medio entre los algoritmos complejos y precisos, y los que son muy simples y menos precisos.

Etograma	Tamaño de ventana	Superposición	Características
Original	1 s	0 %	Solo estadísticas
	2 s	25 %	
Cuatro más comunes	3 s	50 %	Estadísticas + otras
	10 s	75 %	

Tabla 1: Combinaciones para el preprocesado

Tras la creación del modelo se procederá a su evaluación, calculando la matriz de confusión y de ahí se obtendrán las métricas de precisión, sensibilidad, exactitud y F1, que servirán de referencia de ahora en adelante.

[6] 2 etogramas * 4 tamaños de ventana * 4 porcentajes de superposición * dos conjuntos de características

Evaluación de clasificadores

Una vez se ha obtenido un modelo *ideal* con el que comparar, se utilizan los 64 *datasets* preprocesados para entrenar y evaluar otros algoritmos como *Gaussian Naive Bayes* (GNB), *Bernoulli Naive Bayes* (BNB), *K Nearest Neighbors* (KNN) y *Support Vector Machine* (SVM).

Los modelos NB se escogen por ser sencillos y rápidos, tal y como demuestran *Kamminga et al.* [33]. Se emplean dos versiones diferentes porque cada una asume de forma diferente cómo va a ser la distribución de los datos. KNN se elige como algoritmo para el análisis debido a que es también más sencillo que RF, pero se diferencia de NB en que realmente no "predice" si no que "recuerda" sus datos de entrenamiento. Por último, se elige una arquitectura SVM por ser un clasificador más complejo que RF a la hora de entrenar y que es conocido por operar muy bien en determinados tipos de problemas de ML.

El procedimiento es el mismo que en el caso de la creación del modelo de referencia: escalado de los datos, selección de características, afinado de hiper parámetros para cada uno de los modelos, entrenamiento del clasificador y evaluación de cada uno de los 64 resultados.

La selección tiene como objetivo eliminar todas las características menos las cinco más relevantes. De esta manera se aligera el proceso de entrenamiento y se gana conocimiento acerca de la naturaleza del problema, destacando qué características son más significativas para la discriminación de clases. Se emplea *Backward Elimination* como método de selección de características debido a que, a diferencia de *Forward Selection,* tiende a tener más en consideración las relaciones subyacentes entre características, a pesar de ser ambos algoritmos del tipo *Greedy Selection*.

El afinado de hiper parámetros se realiza mediante *Grid Search*, es decir, dándole al clasificador una malla de posibles parámetros para su configuración, entre los que tendrá que elegir la mejor combinación posible de ellos. Se emplea para mejorar los resultados del algoritmo, aunque a veces se recomienda no aplicarlo para evitar el *overfitting* derivado de ajustarlo mucho a unos datos de entrada concretos.

Genericidad del modelo

Uno de los retos que se han mencionado previamente en este documento es la generalización del dominio (*"domain generalization"*), que es "la habilidad de los modelos para generalizar bien ante dominios nuevos" [31]. Estos dominios nuevos pueden ser animales de variedades diferentes, de otras especies, o sometidos a otras condiciones ambientales. En el caso que ocupa a este trabajo el dominio empleado se corresponde con las cabras del *dataset* de *Kamminga et al.* [33]. Para analizar qué tan genéricos pueden resultar los modelos entrenados con estos datos se evaluará su capacidad de predecir la actividad de ovejas de la misma subfamilia que las cabras (*caprinae*). Los datos de estas ovejas están también presentes en el *dataset,* aunque no se hayan empleado anteriormente.

Para poder emplear esta nueva información en la predicción se la debe preprocesar antes, al igual que se ha hecho con los datos de las cabras. Se siguen los mismos pasos, pero en vez de emplear 64 configuraciones diferentes en base a las posibilidades que ofrecen el tamaño de

ventana, la superposición, los conjuntos de características o los etogramas, se emplean solamente 32. Se decide prescindir para este análisis de la variable de las características, empleando solamente la combinación de todas, debido al tiempo excesivo que conllevaría realizar otra vez una tanda entera de preprocesado. Se considera que para evaluar la genericidad de los modelos entrenados es suficiente con comparar los resultados de los modelos de referencia con los producidos por los 32 preprocesados de las ovejas.

Viabilidad del modelo de referencia

Los modelos de referencia que se han desarrollado para cada una de las combinaciones del preprocesado son hasta cierto punto ideales, en tanto en cuanto se han destinado abundantes recursos para dotarles de una capacidad predictiva alta. Algunos llegan a ocupar un par de gigas de memoria una vez exportados debido al alto número de árboles de decisión y la gran profundidad que tienen los que conforman cada uno de los modelos *Random Forest* producidos.

Esto resulta inviable para una implementación práctica ya que las necesidades de procesado, consumo energético y espacio de memoria de modelos tan grandes y exhaustivos descartan su despliegue en campo en dispositivos comerciales. Como modelos de referencia funcionan muy bien porque sirven de punto de comparación, son el objetivo al que se quiere llegar, pero no son prácticos porque no se atienen a las limitaciones que impondría casi cualquier proyecto realista de AAR.

Por esta razón se busca evaluar cuánta diferencia existe entre estos modelos "ideales" y modelos empleados en soluciones comerciales presentes en el mercado. De esta manera se analiza su viabilidad comercial. Para ello se tomará como referencia de modelo *Random Forest* implementado en el módulo "LSM6DSOX" de *STMicroelectronics*. Se emulará su funcionamiento atendiendo a sus características y limitaciones técnicas: un número máximo de 8 árboles de decisión, 256 nodos máximo entre todos los nodos y 16 posibles etiquetas como tope.

Finalmente se comparan las métricas obtenidas de los resultados de este modelo "realista" con las de los modelos de referencia en la implementación del próximo capítulo.

3.3 Limitaciones

Se ha comentado ya cómo se va a organizar el trabajo para alcanzar los objetivos propuestos y por qué se han tomado ciertas decisiones diseñando la metodología. En este apartado se comentarán las limitaciones a las que se enfrenta este estudio, qué etapas no se han incluido, cuales se mantienen fijas para todas las combinaciones de preprocesado y por qué.

Como se ha comentado, a grandes rasgos la implementación consta de dos partes: una en la que se ejecutan las variaciones del preprocesado generando nuevos *datasets* y otra en la que se emplean dichos *datasets* para el entrenamiento y evaluación de modelos de ML. Para la creación de los modelos de referencia, la evaluación de los clasificadores, el análisis de genericidad y el de viabilidad se han producido 96 *datasets* diferentes, entrenado 384 modelos

y realizado 416 inferencias[7]. La limitación más importante son los recursos necesarios para el procesado y, por tanto, el tiempo necesario para llevar a cabo todo lo mencionado. Para desarrollar el proyecto, se ha contado con un ordenador personal portátil, así como acceso a uno de los servidores más potentes del grupo de ingeniería microelectrónica de TEISA (Fénix) y se ha empleado la computación en la nube a través del servicio de *Google Colab.*

Aunque no todos los preprocesados ni todos los entrenamientos de clasificadores hayan durado lo mismo, se ha dedicado un tiempo considerable a todas estas actividades. Esta es la principal razón por la cual el análisis se ha limitado "solamente" a variar el tamaño de ventana, la superposición, el etograma, las características empleadas y los clasificadores. Incluir más variabilidad hubiera sido interesante y es una de las líneas de investigación a futuro, pero hubiera resultado irrealizable puesto que el tiempo aumenta geométricamente con cada grado de libertad.

Los elementos que se han mantenido constantes durante todo el proceso han sido: la gestión de datos ausentes, la gestión de outliers, la combinación de las series temporales módulo y aceleración en los tres ejes, y el ajuste de hiper parámetros (Figura 4, Figura 5 y Figura 6). La razón es la comentada previamente (tiempo y capacidad de cómputo). Además, introducir variabilidad en los dos primeros casos no se considera importante puesto que sólo ayuda al tratamiento del ruido del *dataset* y se ha partido de uno ya procesado. Por ese mismo motivo no se incluye la etapa de filtrado. Estos aspectos hubieran tenido más relevancia si los datos hubieran formado parte de un *dataset* propio, es decir, si el trabajo hubiera estado más centrado en la etapa de adquisición de datos.

Por otro lado, el muestreo de los datos que se indica en la Figura 4, hubiera sido un aspecto interesante a explorar como medida para evitar la escasez de anotaciones (sobre muestreo) y para tratar con el desequilibrio de clases (submuestreo). Sin embargo, esto habría complicado el proyecto y producido avances poco relevantes, porque para tratar con esos problemas ya se usa la variación del etograma.

El porcentaje de superposición al 90% fue descartado en su uso para el análisis después de unas pruebas iniciales, donde se comprobó que para ciertas configuraciones el tiempo consumido en el preprocesado y en la creación del modelo era inasumible. A pesar de ello, se reconoce que los resultados de las pocas pruebas que se ejecutaron fueron halagüeños, con precisiones altas. Se consideró inasumible aun así debido a que el único modelo que se llegó a entrenar con esta configuración tardó más de un día y acabó ocupando 2 GB.

Hubiera resultado interesante también analizar la adición de algunas características frecuenciales. Desde el punto de vista de alguien que se está formando resultaba muy atrayente la posibilidad de implementarlas, aunque complicaban mucho el estudio. Además, en el contexto de la computación *on-edge* estas características conllevan demasiado coste computacional y algunos autores no las recomiendan para estas aplicaciones mientas que las del dominio temporal son muy recomendadas, independientemente de cuál sea. [44].

[7] 64 *datasets* intermedios de cabras y 32 de ovejas son 96 en total. 64 *datasets* empleados para 6 clasificadores diferentes hacen 384 modelos y 416 inferencias empleando los *datasets* de ovejas

Capítulo 4: Implementación

En este capítulo se describe como se ha llevado a la práctica la metodología presentada en el capítulo anterior. Se empieza describiendo la forma de organizar el trabajo de acuerdo con lo mencionado en el Capítulo 3 y las herramientas que se han utilizado para ello. Posteriormente se describe someramente el *dataset* del que se parte antes de pasar a explicar cómo ha sido tratado en la fase de preprocesado, mediante la descripción del código. Seguidamente se reseñan las partes significativas del código escrito para la selección de características, entrenamiento de los diferentes modelos y su evaluación. Para concluir se incluye una pequeña explicación acerca de la emulación del módulo comercial LSM6DSOX, empleado para analizar la viabilidad de los modelos de referencia.

4.1 Organización del trabajo

Estructurar un trabajo como este, del que se prevé que se van a obtener muchos datos que hay que analizar, no es sencillo y resulta fundamental. Desde el principio se optó por la modularidad. Por ello, se invirtió cierto tiempo en descubrir cómo se podría dividir en piezas más pequeñas el proceso entero (de este esfuerzo salen las imágenes del Capítulo 3.1) y se invirtió otro tanto en hacerlo efectivo.

De este tiempo y esfuerzo invertidos se obtuvo la división antes comentada[8] en dos fases: una de procesado de los datos y otra dedicada al entrenamiento, inferencia y evaluación. Estas dos fases se corresponden con dos tandas de programas (*scripts* en Python) diferentes. La primera carga el *dataset*, realiza todas las operaciones necesarias con él y lo devuelve listo para ser entrenado, guardándose en formato CSV en la carpeta que le corresponda. A este conjunto de datos se le ha denominado "*dataset* de preprocesado". La segunda fase carga un "*dataset* de preprocesado" y opera con él, de forma que se seleccionan sus características y se usa para entrenar el modelo elegido. A continuación, se exporta el modelo entrenado junto a la matriz de confusión, las métricas y las características seleccionadas. Esta forma de trabajar permite una ejecución en "*pipeline"*: mientras se está realizando un preprocesado en un dispositivo se puede estar ejecutando en paralelo en otro dispositivo un entrenamiento utilizando el preprocesado hecho anteriormente, lo cual permite reducir el tiempo de cómputo del proceso.

Para gestionar el proceso descrito anteriormente se ideó un sistema de nombres para los archivos que se iban generando, de tal forma que el propio descriptor del archivo indicara lo que contenía. La generación de estos nombres se realizó a través de una hoja de Excel en la que la elección de una serie de características del preprocesado acababa formando parte del nombre del archivo/carpeta. De esta forma, cada parte del preprocesado añadida a la configuración del archivo añadía un código concreto de entre tres y cuatro caracteres al nombre. Por ejemplo, para un archivo que empleara el etograma original, con un tamaño de ventana de 10 segundos, una superposición del 50% y los dos conjuntos de características, el nombre del archivo sería:

[8] Véase inicio del Capítulo 3.2

kam10ws50ovtestot

Donde “kam” identifica el dataset original (*Kamminga*), “10ws” define el tamaño de ventana (*Window Size*), “50ov” indica el grado de superposición (*overlap*), “tes” apunta a las características temporales estadísticas y “tot” incluye las características temporales llamadas “otras” (Tabla 2).

KAM: Etograma de nueve clases	**CUS**: Etograma de cuatro clases	**TES**: conjunto simple de características	**TESTOT**: conjunto completo de características	**Xws**: tamaño de ventana X	**Xov**: porcentaje de superposición X

Tabla 2: Códigos utilizados en los nombres de ficheros

Utilizando la misma estrategia se eligió el nombre de los 64 *Jupyter notebooks* que iban a generar los 64 *datasets* diferentes utilizando las 64 combinaciones posibles de las variables del preprocesado seleccionadas en el Capítulo 3 (tamaño de ventana, superposición, etograma y “otras” características). Cada uno de ellos es similar a los demás, pues solamente varían entre ellos las líneas que procesan los datos y almacenan los resultados.

Por otro lado, se crea un archivo de *script* en Python por cada uno de los clasificadores que se van a emplear. En ellos se realiza la carga de un *dataset* de preprocesado, se realiza el escalado de los datos, se extraen las 5 características más importantes, se afinan los hiper parámetros del clasificador, se entrena, se evalúa su rendimiento y por último se exportan los resultados. Entre los archivos de uno u otro clasificador sólo varía la llamada al clasificador y el ajuste de sus parámetros, ya que cada uno tiene características diferentes. El resto de los elementos se mantienen constantes.

Finalmente, los datos obtenidos de cada evaluación (“dataset de preprocesado X” con “clasificador Y”) se incluyen en hojas de Excel. Estos datos se pueden consultar en el material complementario de este trabajo.

En los siguientes apartados se explicará con más detalle el código implementado, utilizando como ejemplo alguno de los *scripts* realizados e indicando cuales son las partes que varían entre archivos.

4.2 El *dataset*

Se parte de un grupo de *datasets* de acceso público, desarrollados para el artículo de *Kamminga* [33], en el que se registran los datos de cuatro cabras y dos ovejas capturados por sensores ubicados en el collar.

Cada collar tiene los sensores colocados en un lugar diferente y todos los collares pueden rotar sobre sí mismos con los movimientos del animal. Se integran en cada collar un acelerómetro triaxial estándar, un acelerómetro triaxial de alta intensidad, un giroscopio triaxial, un magnetómetro triaxial, un sensor de temperatura y otro de presión.

En cuanto a las etiquetas, el dataset identifica nueve clases de comportamiento en el etograma (estar tumbado, estar de pie, pastar, luchar, agitarse, rascarse, caminar, trotar y correr), como se ha comentado anteriormente.

Al margen de las medidas y las etiquetas, cada uno de los *datasets* incluye la identificación del animal, el número de segmento y el recuento del tiempo. Los segmentos delimitan medidas que se han tomado de forma continua y que se corresponden con una sola actividad. Son de tamaño variable y no se asegura la continuidad temporal entre segmentos consecutivos.

4.3 Fase de preprocesado

En esta fase se importa el *dataset* mencionado en el punto anterior, se opera con él y se exporta en formato CSV, danto lugar a lo que se ha decidido denominar *dataset* de preprocesado. El nombre de dicho *dataset* indica la combinación de elementos de preprocesado que se han empleado en su generación, como se ha comentado previamente.

Para realizar el preprocesado se emplean tanto *scripts* de Python como *Jupyter Notebooks* indistintamente. En este proyecto se han usado Python 3.11.5 y el gestor de paquetes Anaconda, en su versión 24.5.0. Los principales paquetes instalados son SciKitLearn (1.2.2), NumPy (1.24.3), Pandas (1.5.3), SciPy (1.10.1) y MatPlotLib (3.7.1).

En el proceso de ejemplo que se va a describir a continuación se emplea el archivo “cus3ws25ovtestot”, por ser uno bastante completo. Lo descrito en esta sección es válido para el resto de las acciones de preprocesado, con ligeros cambios relacionados sus variables.

Inicio:

Se comienza asignando a una variable el nombre del archivo de preprocesado, para emplearlo a la hora de crear carpetas y guardar los archivos relacionados. Se continúa cargando todos los archivos CSV asociados a las cabras y se juntándolos en una única variable de tipo *dataframe*. Se usan herramientas de visualización de datos para tener una perspectiva general de las muestras capturadas y se guarda esa información como archivos Excel en la carpeta creada a tal efecto para la combinación específica de preprocesado.

Posteriormente se implementa la **selección del etograma**. Este es uno de los pasos que dependen de la configuración y cambia entre archivos. En una parte del código no se modifica nada porque se mantiene el número de clases considerada en el *dataset* original. En la otra mitad se prescinde de todas las etiquetas, salvo las cuatro más relevantes, que se corresponden con los comportamientos de caminar, estar de pie, estar tumbado y pastar (Figura 8)

```
# Se eliminan todas las etiquetas menos las cuatro más abundantes: walking, standing, lying y grazing:
dataset = dataset[(dataset["label"] == "walking") |
                  (dataset["label"] == "standing") |
                  (dataset["label"] == "lying") |
                  (dataset["label"] == "grazing")]

```

Figura 8: Selección del etograma

Preprocesado:

A continuación, se eliminan todas las columnas del *dataset* que carecen de interés para el análisis que se va a realizar, permaneciendo solamente las columnas correspondientes a las medidas del acelerómetro triaxial, identificación de segmento y etiquetas.

Si se detecta que en el *dataset* existen valores de tipo *"inf"* (falta dato) se transforman en valores "*NaN"* (indefinido) para a continuación dividir el conjunto de datos en dos partes: una que incluya sólo con los valores numéricos y otra con los valores de texto. De esta forma se puede aplicar un método sobre la columna numérica para sustituir los valores "*NaN*" por la interpolación del dato anterior de la columna con el dato posterior (Figura 9)

```
# Reemplazo de inf por NaN
dataset.replace([np.inf, -np.inf], np.nan, inplace=True)

# Separación de los datos numéricos de los de texto
dataset_num = dataset.select_dtypes(include=[np.number])  # Numerico
dataset_tex = dataset.select_dtypes(include=[object])     # Texto

# Interpolación, limite = 2 para los casos en los que haya dos NaN seguidos
dataset_num = dataset_num.interpolate(method = 'linear', axis = 1, limit = 2)
```

Figura 9: Gestión de datos ausentes

A continuación se eliminan los *outliers* empleando para ello una herramienta de *SciKitLearn* basada en *Random Forest* denominada *IsolationForest* [46]. Se declara el método, y se le aplica al subconjunto de datos numéricos de forma que genera un array de la misma longitud que el subconjunto, pero compuesto de "1" y "-1". Estos números indican si en cada posición se ha detectado un *outlier* (-1) o no (1). Se usa el array para eliminar de los dos subconjuntos de datos (numéricos y de texto) las muestras correspondientes a las posiciones donde se encuentren los "-1". Tras eso, se reinician los índices y se vuelven a unir los subconjuntos en un único *dataframe* (Figura 10)

```
# Gestión de outliers:
isolation_forest = IsolationForest(random_state=42)
dataset_outlier = isolation_forest.fit_predict(dataset_num)
# Para quitar los outliers del dataset se eliminan filas allá donde haya un -1:
dataset_num = dataset_num.iloc[dataset_outlier == 1]
dataset_tex = dataset_tex.iloc[dataset_outlier == 1]
# Se reordenan los indices:
dataset_num.reset_index(inplace=True, drop=True)
dataset_tex.reset_index(inplace=True, drop=True)
# Se juntan otra vez:
dataset = pd.concat([dataset_num, dataset_tex], axis = 1)
```

Figura 10: Gestión de outliers

Por último, se añaden las series temporales. Como se ha comentado en el capítulo de metodología sólo se añade el módulo de la aceleración, por las limitaciones de recursos para incluir más variabilidad. En este punto, si solamente se va a usar el conjunto más básico de características, se borran las columnas correspondientes a las medidas de los ejes. Si no es así, se mantienen hasta la extracción de características.

Segmentación:

Para segmentar el *dataset* según las indicaciones (ventana de 3 segundos y superposición del 25%) primero habrá que definir las **variables de segmentación**, , el **tamaño de ventana** y la **superposición** (Figura 11).

A continuación, se decide borrar aquellos segmentos cuyo tamaño sea inferior al tamaño de ventana ya que utilizarlos podría suponer un problema de distorsión de datos. Se empieza contando el número de muestras por segmento, luego se seleccionan y extraen aquellos más pequeños que el tamaño de ventana. Se compara este *dataframe* extraído con el *dataset* original generando un *array* de valores booleanos que indique las posiciones de muestras presentes en ambas estructuras de datos. Se añade dicho *array* al *dataset* original y se procede a reescribirlo con los valores *False* solamente, es decir, con los valores que no estaban presentes en ambas partes de la comparación anterior. Como los segmentos se han desordenado se ordenan por valor y finalmente se reinicia el índice (Figura 11).

```
# Segmentación: Variables
num_samples = 600  # Tamaño de ventana
ov = 0.25          # Superposición
np.random.seed(42)

# Eliminación de segmentos inferiores a num_samples ---
# Recuento de las muestras por cada segmento:
cuenta = dataset["segment_ID"].value_counts().
        reset_index().rename(columns={"index": "segment_ID", "segment_ID": "count"})
# Selección de segmentos más pequeños que el tamaño de ventana:
cuenta_infws = cuenta.loc[cuenta['count'] < num_samples].copy()
# Se buscan esos segmentos en el dataset:
cond = dataset["segment_ID"].isin(cuenta_infws["segment_ID"])
np.count_nonzero(cond == True)
# Se añade el array de booleanos al dataset:
dataset['Inf_WS'] = cond.tolist()
# Eliminación de las muestras cuyo segmento es pequeño:
dataset = dataset[dataset["Inf_WS"] == False]
dataset = dataset.drop(["Inf_WS"], axis = 1)
# Los segmentos se han desordenado, se ordenan por valor:
dataset.sort_values(by=["segment_ID"], inplace = True)
# Se reinicia el índice:
dataset.reset_index(inplace=True, drop=True)
```

Figura 11: Variables de segmentación y borrado de segmentos de tamaño inferior al tamaño de ventana

Con el fin de poder recorrer el *dataset* entero para poder hacer la segmentación, se modifica la numeración de los segmentos para que sean consecutivos (a pesar de estar ordenados su numeración no es consecutiva). Para hacer esto primero se obtiene la cantidad de segmentos actualmente presentes contando los elementos únicos. Luego se contabiliza el número de elementos de cada uno y se ordenan por índice para proceder a renombrar los segmentos. Para ello se emplea un bucle que se repite tantas veces como el número de segmentos presentes en *dataset* y que sustituye el número de segmento del conjunto de muestras que comparten el mismo *segment_ID* por uno nuevo que indique la ordinalidad (Figura 12).

```
# Renumeración de los segmentos ---
# Cantidad de elementos diferentes:
array_segmen = dataset["segment_ID"].unique()
max_elemen = len(array_segmen)
# Contabilización y reordenamiento:
len_segmen = dataset["segment_ID"].value_counts()
len_segmen = len_segmen.sort_index(axis = 'index', ascending = True, inplace = False)
len_segmen = len_segmen.reset_index()
len_segmen = len_segmen['segment_ID'].squeeze()
# Renumeración:
for i in range (0, max_elemen, 1):
    dataset.loc[dataset["segment_ID"] == array_segmen[i], "segment_ID"] = i
```

Figura 12: Renumeración de los segmentos

Una vez que los segmentos de tamaño inferior al tamaño de ventana han sido eliminados y que los restantes han sido ordenados se pretende eliminar muestras de cada uno de ellos de forma aleatoria, para que contengan un número de muestras múltiplo del tamaño de ventana. Con esta estrategia se pretende conseguir que cada segmento se pueda dividir en un número entero de ventanas, quitando muestras aleatoriamente. Se elige hacer esto porque descartar siempre las últimas muestras que no encajen de cada segmento podría sesgar los datos haciendo que, por ejemplo, se ignorasen bastantes muestras pertenecientes a periodos de transición entre actividades. Para implementarlo se empieza creando un *array* vacío para ir guardando en él el número de ventanas que entran en cada segmento. A continuación, un bucle recorre el conjunto de datos analizando para cada valor de segmento si su número de muestras coincide con un múltiplo del tamaño de ventanas o no. Si coincide, se guarda en el *array* mencionado antes el número de ventanas, si no, elimina el número de muestras aleatorias necesario para que coincida y luego guarda el número de ventanas que contiene dicho segmento (Figura 13).

```
num_windows = np.empty(max_elemen, dtype=int)

for i in range (0, max_elemen, 1):
    dataset['index'] = dataset.index

    # Condicional para eliminar muestras aleatoriamente
    # Se resta al tamaño del segmento el numero de muestras, pues inicialmente
    # en todos los segmentos cabrá una ventana, y el restante se divide entre lo que
    # se desplaza la ventana (el tamaño de la ventana menos el nº de muestras que solapan)
    if (len_segmen[i] - num_samples) % (num_samples * (1 - ov)) == 0:
        num_windows[i] = 1 + int((len_segmen[i] - num_samples) / (num_samples * (1 - ov)))
    else:
        remove_n = (len_segmen[i] - num_samples) % (int(num_samples * (1 - ov)))  # Numero de muestras a quitar
        drop_index = np.random.choice(dataset.loc[dataset["segment_ID"] == i, "index"], remove_n, replace = False)
        dataset = dataset.drop(drop_index, axis = 0)
        num_windows[i] = 1 + int((len_segmen[i] - num_samples) / (num_samples * (1 - ov)))
        dataset = dataset.drop(['index'], axis=1)
        dataset.reset_index(inplace=True, drop=True)
```

Figura 13: Borrado aleatorio de muestras por cada segmento

Después de hacer esto, se obtiene un *dataset* del tamaño adecuado para realizar la extracción de características. La modificación introducida anteriormente garantiza que cada etiqueta presente en el *dataset* tenga una cantidad de muestras múltiplo del tamaño de

ventana, por lo que al extraer las características de cada ventana no se mezclan las medidas de diferentes clases.

Extracción de características y exportado:

Antes de nada, se añaden a la variable *daframe* del *dataset* las columnas vacías correspondientes a todas las **características** que se van a calcular.

Para el cálculo de características en sí, se empieza creando una variable *dataframe* vacío con la cantidad de columnas del *dataset* que se está utilizando. A continuación, se ejecuta un bucle que recorre todo el *dataset* y que avanza por cada iteración el número de muestras no solapadas entre ventanas consecutivas (por ejemplo, 150 muestras si el tamaño es 200 y el porcentaje de superposición del 25%).

En cada iteración se hacen diversas operaciones, empezando por crear una fila vacía para guardar en ella los resultados y al final añadirla al *dataframe* vacío mencionado antes. Luego se extraen las medidas correspondientes al número de muestras definidas para la ventana en cada caso. Se extraen las de una serie temporal (el módulo) si sólo se va a emplear el conjunto de características básico, y se extraen de dos series (variables X, Y, Z además del módulo) si se van a usar los dos conjuntos. De cara a analizar la variación de movimiento se extraen los mismos datos, pero en una muestra anterior (*"_prev"*) y como los segmentos no son consecutivos, en la primera muestra de cada ventana se inserta el valor medio de la misma (se correspondería con la posición -1) (Figura 14)

```
# Calculo de las características

total_time = 0.0
tam_dataset = len(dataset)

new_dataset = pd.DataFrame(columns = dataset.columns)

for i in range (0, tam_dataset, int(num_samples * (1 - ov))):
    start = time.time()
    row_df = pd.DataFrame(columns = dataset.columns, index = range(1))  # Se crea con una fila vacia

    # Parte del dataset con la que se opera en cada iteración
    rows_iter = dataset["Mod"].loc[(dataset["index"] >= i) & (dataset["index"] < i+num_samples)].reset_index(drop=True)
    rows_prev = pd.concat([pd.DataFrame([rows_iter.mean()]), dataset["Mod"].loc[(dataset["index"] >= i)
                                    & (dataset["index"] < i+num_samples-1)]], ignore_index = True)
    rows_iter_x = dataset["ax"].loc[(dataset["index"] >= i) & (dataset["index"] < i+num_samples)].reset_index(drop=True)
    rows_iter_y = dataset["ay"].loc[(dataset["index"] >= i) & (dataset["index"] < i+num_samples)].reset_index(drop=True)
    rows_iter_z = dataset["az"].loc[(dataset["index"] >= i) & (dataset["index"] < i+num_samples)].reset_index(drop=True)
    rows_prev_x = pd.concat([pd.DataFrame([rows_iter_x.mean()]), dataset["ax"].loc[(dataset["index"] >= i)
                                    & (dataset["index"] < i+num_samples-1)]], ignore_index = True)
    rows_prev_y = pd.concat([pd.DataFrame([rows_iter_y.mean()]), dataset["ay"].loc[(dataset["index"] >= i)
                                    & (dataset["index"] < i+num_samples-1)]], ignore_index = True)
    rows_prev_z = pd.concat([pd.DataFrame([rows_iter_z.mean()]), dataset["az"].loc[(dataset["index"] >= i)
                                    & (dataset["index"] < i+num_samples-1)]], ignore_index = True)
```

Figura 14: Extracción de características (I)

En cada iteración la fila vacía que se ha creado se rellena o bien con los cálculos realizados o bien con una copia de los datos de la primera muestra de la ventana sobre la que se está trabajando (Figura 15).

```
    row_df["segment_ID"] = dataset.loc[dataset["index"] == i, 'segment_ID'].iat[0]
    row_df["ax"] = dataset.loc[dataset["index"] == i, 'ax'].iat[0]
    row_df["ay"] = dataset.loc[dataset["index"] == i, 'ay'].iat[0]
    row_df["az"] = dataset.loc[dataset["index"] == i, 'az'].iat[0]
    row_df["label"] = dataset.loc[dataset["index"] == i, "label"].iat[0]
    row_df["Mod"] = dataset.loc[dataset["index"] == i, "Mod"].iat[0]
    row_df["mean"] = rows_iter.mean()
    row_df["std"] = rows_iter.std()
    row_df["v_min"] = rows_iter.min()
    row_df["v_max"] = rows_iter.max()
    row_df["minmax"] = row_df["v_max"] - row_df["v_min"]
    row_df["median"] = rows_iter.median()
    row_df["var"] = rows_iter.var()
    row_df["q25"] = rows_iter.quantile(0.25)
    row_df["q75"] = rows_iter.quantile(0.75)
    row_df["q2575"] = row_df["q75"] - row_df["q25"]
    row_df["index"] = dataset.loc[dataset["index"] == i, "index"].iat[0]

    row_df["energy"] = ((rows_iter_x**2) + (rows_iter_y**2) + (rows_iter_z**2)).sum()
    row_df["entropy"] = entropy(round(10*rows_iter))
    row_df["kurtosis"] = kurtosis(rows_iter, axis = 0)
    row_df["skewness"] = skew(rows_iter, axis = 0)

    zc_x = len(np.where(np.diff(np.sign(rows_iter_x)))[0])
    zc_y = len(np.where(np.diff(np.sign(rows_iter_y)))[0])
    zc_z = len(np.where(np.diff(np.sign(rows_iter_z)))[0])
    row_df["zcr"] = ((zc_x)+(zc_y)+(zc_z)) / num_samples
```

Figura 15: Extracción de características (II)

Para la mayoría de las características que se extraen, *SciKitLearn* incluye un método para calcularlas fácilmente. Esto no ocurre para la energía, la *SMA*, la variación de movimiento (“*mov_var”*), el valor RMS del módulo (“*rms_sv”*), las correlaciones entre pares de ejes (*“par_corr_ [XY, XZ, YZ]”)*, los cruces por cero (“*zcr”*) o la media de cambios en valor absoluto (“*mean_abs_changes”*). En estos casos se tiene que calcular explícitamente como se ve en la Figura 15 y en la Figura 16 siguiendo las fórmulas de trabajos como el de *Kleanthous et al.* [30].

```
    row_df["sma"] = (rows_iter_x.abs().sum() + rows_iter_y.abs().sum() + rows_iter_z.abs().sum()) / num_samples
    row_df["rms_sv"] = np.sqrt(((rows_iter**2).sum()) / num_samples)
    row_df["mov_var"] = (((rows_iter_x - rows_prev_x).abs().sum()) + ((rows_iter_y - rows_prev_y).abs().sum())
                         + ((rows_iter_z - rows_prev_z).abs().sum())) / num_samples
    row_df["par_corr_xy"] = np.cov(rows_iter_x, rows_iter_y, rowvar = False)[0,1] / (rows_iter_x.std() * rows_iter_y.std())
    row_df["par_corr_xz"] = np.cov(rows_iter_x, rows_iter_z, rowvar = False)[0,1] / (rows_iter_x.std() * rows_iter_z.std())
    row_df["par_corr_yz"] = np.cov(rows_iter_y, rows_iter_z, rowvar = False)[0,1] / (rows_iter_y.std() * rows_iter_z.std())
    row_df["mean_abs_changes"] = ((rows_iter - rows_prev).abs().sum()) / num_samples

    actual_segment_row = row_df["segment_ID"].iat[0]
    actual_num_window = new_dataset.loc[new_dataset["segment_ID"] == actual_segment_row, "segment_ID"].count()

    if (actual_num_window < num_windows[actual_segment_row]):
        new_dataset = pd.concat([new_dataset, row_df], ignore_index = True)
    del row_df

    end = time.time()
    total_time += end - start
    print('Index:', i, 'Porcentaje:', (100*i/tam_dataset), '%', 'Tiempo transcurrido:', total_time, end='\r')

dataset.drop(columns = ['ax', 'ay', 'az'], inplace = True)
```

Figura 16: Extracción de características (III)

En la Figura 16 también se describe el proceso de concatenación de la fila que se creó para esta iteración. En él se obtiene el valor del segmento de la fila que se acaba de calcular, y se cuenta el número de veces que aparece en el *dataframe*. De esta manera se puede contabilizar si ha excedido el tamaño de ventanas calculado para dicho segmento en la Figura 13. Si ese fuera

el caso, la nueva fila no se añadiría pues sería un error. Finalmente se borran las variables correspondientes a los ejes X, Y y Z.

Antes de exportar el resultado de todo el procesado, al *dataset* resultante se le quitan las columnas que no se van a emplear para la clasificación y se le reinicia el índice. Se exporta el resultado en formato de archivo de valores separados por comas (*CSV*) y en libro de Excel.

4.4 Fase de clasificación y evaluación

En esta fase se importan los diferentes *datasets* generados en la fase anterior y se les emplea para entrenar y evaluar el rendimiento de cada uno de los clasificadores propuestos. Como en este caso entre los diferentes *scripts* sólo cambia la fase de afinado de los hiper parámetros se describirá primero la parte del código que es común a todos y posteriormente se analizará la parte específica de cada clasificador. En ella se reseñará brevemente la elección de los hiper parámetros en base a la documentación de cada clasificador que proporciona *SciKitLearn*.

Estructura común a todos los clasificadores:

La parte común a todos los *scripts* de entrenamiento y evaluación está compuesta por el escalado, la selección de las cinco características más importantes y la evaluación del modelo.

Después de cargar el *dataset* de preprocesado correspondiente se efectúa un ajuste de las clases presentes en el mismo (Figura 17). El ajuste consiste en descartar aquellas actividades de las que haya menos de 20 muestras. Esto es debido a que al utilizar una división estratificada del *dataset* para generar un subconjunto de entrenamiento y otro de *prueba*, si el de test cuenta con menos de 5 ejemplos de una actividad los métodos de evaluación de *SciKitLearn* generan un aviso y la precisión se ve muy afectada. Como el mínimo es de 5 características y el subconjunto de prueba es el 25% del total, la forma de garantizar que siempre haya más de dicho número es que en el *dataset* de preprocesado tenga como mínimo 20 o más datos.

```
# Se analiza el numero de muestras que tiene cada clase
num_samples_label = dataset["label"].value_counts(sort = False)
array_label = dataset["label"].unique()  # Array de etiquetas

for i in range (0, len(num_samples_label)):
    if (num_samples_label.iloc[i] <= 20):
        dataset.drop(dataset.loc[dataset["label"] ==
        array_label[i], "index"], axis = 0, inplace = True)

dataset.drop(["index"], axis = 1, inplace = True)
dataset.reset_index(inplace=True, drop=True)
dataset["label"].value_counts(sort = False)
```

Figura 17: Ajuste previo

Posteriormente, se implementa el escalado. Para ello se separan las etiquetas de los datos numéricos, se aplica el escalado a estos datos a través del método *StandardScaler* y se vuelven a juntar con las etiquetas

```
std_scale = StandardScaler()
# Se separan las etiquetas y los numeros de segmento del dataset
dataset_scale = dataset.drop(["segment_ID", "label"], axis=1)
array_scale = std_scale.fit_transform(dataset_scale)
dataset_no_scale = dataset[['label']].copy()

# Se juntan los datos escalados con las ID de segmento y las etiquetas
dataset_scale = pd.DataFrame(array_scale, columns=dataset_scale.columns)
dataset = pd.concat([dataset_no_scale, dataset_scale], axis=1)
```

Figura 18: Escalado de los datos

Tras esto, se lleva a cabo la división estratificada del *dataset* en dos partes: el *subset* de entrenamiento (75%) y el de prueba (25%). A continuación, se llama al modelo del clasificador y se dejan los hiper parámetros por defecto, como se ve en la Figura 19, ya que de momento sólo se va a emplear para la selección de características utilizando la técnica "*Backward Elimination"*. Como se observa en la figura, se está empleando de ejemplo un clasificador SVM. Se carga el método de selección y se le aplica el método "*fit_transform"* para que cargue los datos numéricos del entrenamiento por un lado y las etiquetas por otro y pueda así determinar qué 5 características son las más relevantes. Una vez determinadas, se eliminan del *dataset* de entrenamiento y posteriormente se hará lo mismo con el de prueba.

```
dataset_train, dataset_test = train_test_split(dataset, test_size=0.25,
                                               stratify = dataset["label"],
                                               random_state=42)

lsvc = LinearSVC(penalty='l2', loss='squared_hinge',
                 dual='auto', tol=0.0001, C=1.0,
                 multi_class='ovr', fit_intercept=True,
                 class_weight=None,  random_state= 42,
                 max_iter=1000) # Los hiperparametros se adaptaran luego

# Metodo de seleccion
sfs = SequentialFeatureSelector(lsvc, n_features_to_select=5, direction='backward')
sfs.fit_transform(dataset_train.drop(["label"], axis=1), dataset_train['label'])
features = sfs.get_support()

dataset_train_label = dataset_train['label']. copy()
dataset_train = dataset_train.drop(['label'], axis=1)
dataset_train = dataset_train.loc[:, features]
dataset_train = pd.concat([dataset_train, dataset_train_label], axis=1)
```

Figura 19: Proceso de selección de características

El último apartado común a todos los *scripts* es el de evaluación (Figura 20), donde se emplean diferentes técnicas y métodos del paquete de *SciKitLearn.* Primero se realiza una validación cruzada con 5 desdobles sobre el *dataset* de entrenamiento para observar qué tan diferentes son los resultados de precisión entre unos datos que el clasificador conoce y otros que no. De esta forma, si se observa que al hacer el test la precisión decae mucho, se puede concluir que el modelo está provocando *overfit*. Después de la validación cruzada se eliminan las características irrelevantes del *subset* de prueba y se realiza la inferencia pasándole al modelo desarrollado el mencionado *subset*. Se imprimen tanto el porcentaje de acierto como el número de muestras clasificadas.

Para finalizar la evaluación, se usan dos métodos de *SciKitLearn*: uno para generar una matriz de confusión con los resultados del test y otro para extraer automáticamente las métricas de exactitud, sensibilidad y *F-score,* como se ve en la Figura 20.

```
# Cross - Validation 5 folds
score_cv = cross_val_score(lsvc, dataset_train, dataset_train_label, cv=5)
print("Accuracy CV: ", score_cv*100, "Acc media: ", 100*np.sum(score_cv)/5)

# TEST:
dataset_test_label = dataset_test['label']. copy()
dataset_test = dataset_test.drop(['label'], axis=1)
dataset_test = dataset_test.loc[:, features]

label_predict_test = lsvc.predict(dataset_test) # Predicción
print("Accuracy valores entrenamiento:",
      accuracy_score(label_predict_test, dataset_test_label)*100)

# Precisión y muestras bien clasificadas:
pre_nob =  round(accuracy_score(dataset_test_label, label_predict_test)*100, 2)
print("Numero de muestras bien clasificadas (TP & TN):",
      accuracy_score(dataset_test_label, label_predict_test, normalize = False))

# Matriz de confusión
labels = dataset_train_label.unique()
conf_matrix = multilabel_confusion_matrix(label_predict_test,
                                          dataset_test_label, labels = labels).ravel()

pre, rec, f1, n_occur = precision_recall_fscore_support(label_predict_test,
                                                         dataset_test_label, average = 'macro')
exa_nob = round(pre*100, 2)
sen_nob = round(rec*100, 2)
f1_nob = round(f1*100, 2)
print("\nMACRO\nExactitud:", pre*100, "Sensibilidad:", rec*100, "F1score:", f1*100)

print(classification_report(label_predict_test, dataset_test_label, labels = labels))
```

Figura 20: Evaluación

Una vez evaluado el resultado del modelo, queda exportar tanto el propio modelo como los datos que se han generado (Figura 21). El modelo se exporta en dos formatos diferentes, *joblib* y *picke* por razones de compatibilidad. Finalmente, se exportan a archivos Excel tanto las métricas relevantes como la matriz de confusión generada al evaluar el

Random Forest Ideal o modelo de referencia:

En la Figura 22 se observa el proceso de afinado de los hiper parámetros del clasificador mediante el método de búsqueda de *GridSearch* y el entrenamiento con el *subset* correspondiente, ya sin las características poco relevantes, como se ha comentado anteriormente.

```
# PATH
name = file_name + file_ext + ".pkl"
name2 = file_name + file_ext + ".joblib"
path_pickle = os.path.join(path, name)
path_joblib = os.path.join(path, name2)

# Se exporta la matriz de confusión
path_xslx = file_name + file_ext + '.xlsx'
path_xslx = os.path.join(path, path_xslx)
conf_matrix = np.reshape(conf_matrix, (len(labels), 4))
conf_matrix = pd.DataFrame(conf_matrix, columns=['TN', 'FP', 'FN', 'TP'],
                           index = labels)
conf_matrix.to_excel(path_xslx)

# Se exportan los resultados
resul_col = ['features', 'pre_nob', 'exa_nob', 'sen_nob', 'f1_nob']
string_features = ', '.join([dataset_train.columns[0], dataset_train.columns[1],
                             dataset_train.columns[2],dataset_train.columns[3],
                             dataset_train.columns[4]])


resultados = pd.DataFrame(columns = resul_col, data = [[string_features,
                                                        pre_nob, exa_nob,
                                                        sen_nob, f1_nob]])
path_resul = file_name + file_ext + '_resultados.xlsx'
path_resul = os.path.join(path, path_resul)
resultados.to_excel(path_resul)

# Se exporta el modelo
pickle.dump(lsvc, open(path_pickle, "wb"))  # "name_file+name_ext+.pkl"
joblib.dump(lsvc, path_joblib)  # "name_file+name_ext+.joblib"
```

Figura 21: Exportado de los resultados

Los parámetros del afinado de RF no parecen muchos en comparación con los de otros clasificadores como SVM. Sin embargo, no son necesarios más porque los esenciales son: el número de estimadores (*"n_estimators"*) y el criterio empleado para analizar la calidad de las separaciones. Otros parámetros como la profundidad máxima o el número mínimo de muestras por hoja se han dejado en su valor por defecto porque no son restrictivos sino más bien lo contrario. El número de estimadores es realmente el número de árboles de decisión que puede integrar el clasificador. Por defecto son 100, pero se ha querido probar con diferentes posibilidades, mayores y menores (Figura 22). También se ha probado con todos los criterios posibles incluidos en el paquete de *SciKitLearn* [49].

```
# Se separan las etiquetas
dataset_train_label = dataset_train['label'].copy()
dataset_train = dataset_train.drop(['label'], axis=1)

# Se definen los hiperparametros:
hparam = {'n_estimators': [10, 50, 100, 250, 500],
            'criterion' : ('gini', 'entropy', 'log_loss'),
             'random_state' : [42]}

clf = GridSearchCV(rfc, hparam)
clf.fit(dataset_train, dataset_train_label)

# Se entrena  con los mejores parametros:
rfc.set_params(**clf.best_params_)
rfc.fit(dataset_train, dataset_train_label)
```

Figura 22: Random Forest ideal

Naive Bayes:

Se emplean dos clasificadores diferentes del paquete de *SciKitLearn* para evaluar el funcionamiento de *Naive Bayes:* el que asume que la distribución de los datos es gaussiana (GNB) [50] y el que asume que es una distribución de Bernoulli (BNB) [51]. Dado que a priori no se conoce cómo es la distribución de los datos por ventana, para estos clasificadores **no** se realiza el ajuste de hiper parámetros. Afinarlos requeriría tener un conocimiento apriorístico que no se tiene ni se puede tener en un caso como este, con lo que se dejan los valores por defecto.

Estos algoritmos son más sencillos que RF, de ellos se espera que den resultados aceptables con bastante menos tiempo de entrenamiento.

K Nearest Neighbors:

En la Figura 23 se aprecia el procedimiento de afinado de los hiper parámetros antes de usar el modelo para la predicción. De todos los parámetros en los que se introduce variabilidad el más representativo es el del número de vecinos (“*n_neighbors”*) que es el que da el nombre al algoritmo. La cifra elegida constituye la cantidad de muestras cercanas que participan en la votación para decidir la clase de la muestra nueva, a más vecinos, mayor debería ser a priori la precisión. El siguiente parámetro, los pesos (“*weighs”*), es la forma en la que se pondera la contribución de cada vecino, si es uniforme todos los votos valen lo mismo, en cambio, si depende de la distancia los vecinos más cercanos tendrán más peso en la decisión. Por último, el parámetro “*p”* es el parámetro de la métrica *Minkowski*, por defecto a 2, pero que cuando se pone a 1 equivale a la *distancia Manhattan.*

```
# Se separan las etiquetas
dataset_train_label = dataset_train['label'].copy()
dataset_train = dataset_train.drop(['label'], axis=1)

# Se afinan los hiperparametros
hparam = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 10],
          'weighs' : ('uniform', 'distance'),
          'algorithm' : ['auto'],
          'p': [1, 2]}

clf = GridSearchCV(knn, hparam)
clf.fit(dataset_train, dataset_train_label)

# Se tiene ya el clasificador con los mejores parametros
knn.set_params(**clf.best_params_)
knn.fit(dataset_train, dataset_train_label)

# Se predice el resultado
label_predict = knn.predict(dataset_train)
```

Figura 23: K Nearest Neighbors

Este algoritmo también es en principio más sencillo que RF puesto que mide distancias y clasifica según cercanía a cada clase. Es un tipo de clasificador no generalizable puesto que realmente lo único que hace es recordar sus datos de entrenamiento. A pesar de no ser

paramétrico suele funcionar bien donde el umbral de decisión es irregular, por lo que se espera que proporcione resultados mejores que NB con un tiempo de entrenamiento un poco mayor. Suele funcionar bien para mucha cantidad de datos [41].

Support Vector Machines:

En el caso de SVM, *SciKitLearn* dispone de varios clasificadores: *SVC, NuSVC* o *LSVC.* Los dos primeros son similares entre sí mientras que el tercero usa un kernel lineal con lo que su entrenamiento y uso es menos costoso. En el contexto de la AAR es muy importante la gestión de la energía como ya se ha comentado, con lo que se opta por emplear *LSVC* como forma de comprobar qué tal operan los algoritmos SVM con el conjunto de datos del que se dispone. En la Figura 24 se observa que se hace un afinado de múltiples parámetros.

El parámetro "*loss"* hace referencia a la función de pérdidas o función que se emplea para evaluar el resultado. Dicho parámetro varía entre la función por defecto de SVM y su versión al cuadrado. El siguiente parámetro es la tolerancia (*"tol"*) que no es más que la condición de parada. Cuanto más pequeña es la tolerancia, más estricto es el algoritmo, con la posibilidad de no converger. "*C*" es el parámetro de regularización. "*Multi_class"* hace referencia a la forma en la que se gestiona el problema de la multi clase (cuando hay más de dos clases posibles) donde "*ovr"* implementa una estrategia de uno contra todos. La otra opción intenta optimizar todas las clases a la vez. Por último, "*max_iter*" es el número máximo de iteraciones que puede ejecutar el algoritmo si no alcanza el criterio de parada.

```
# Se separan las etiquetas
dataset_train_label = dataset_train['label'].copy()
dataset_train = dataset_train.drop(['label'], axis=1)

hparam = {'loss': ['hinge', 'squared_hinge'],
          'tol' : [0.001, 0.0005, 0.0001],
          'C' : [0.5, 1.0, 1.5],
          'multi_class': ['ovr', 'crammer_singer'],
          'max_iter': [250, 500, 1000, 1250, 1500]}

clf = GridSearchCV(lsvc, hparam)
clf.fit(dataset_train, dataset_train_label)

# Se tiene ya el clasificador con los mejores parametros
lsvc.set_params(**clf.best_params_)
lsvc.fit(dataset_train, dataset_train_label)
print("Hparams: ", clf.best_params_)

# Se predice el resultado
label_predict = lsvc.predict(dataset_train)
```

Figura 24: Support Vector Machine

SVM, aun empleando su versión con *kernel* lineal, se espera que necesite más tiempo de entrenamiento que RF y que ofrezca unos resultados similares o incluso superiores en algunos casos, dado que funciona muy bien cuando existen varias clases [52].

4.5 Módulo LSM6DSOX

Para emular las características de este sensor se ha tenido que desarrollar un código específico. La principal razón es que su acelerador para ML está limitado a 8 árboles de decisión y 256 nodos, que deben estar repartidos entre los mencionados árboles. De esta manera nunca se puede tener una cantidad de árboles cuya suma de nodos exceda los 256. Esto complica un poco el afinado de hiper parámetros, pues hay que introducir de alguna forma la restricción comentada y no es posible incluir una fórmula en un elemento que se va a interpretar literalmente, como es una etiqueta para un parámetro.

Se ha optado por una solución a medida, se definen ocho conjuntos de hiper parámetros donde siempre se respete la condición estipulada (Figura 25). Como se observa, "*max_depth*" es el parámetro que identifica la cantidad de nodos máxima posible por cada árbol. "*Criterion*" hace referencia a la medida empleada para medir la calidad de la separación.

```
# Se separan las etiquetas
dataset_train_label = dataset_train['label'].copy()
dataset_train = dataset_train.drop(['label'], axis=1)

# Se deja el GridSearch en standby
# Se definen los hiperparametros:
hparam1 = {'n_estimators':[1],'criterion':('gini','entropy','log_loss'),'max_depth':[256],'random_state':[42]}
hparam2 = {'n_estimators':[2],'criterion':('gini','entropy','log_loss'),'max_depth':[128],'random_state':[42]}
hparam3 = {'n_estimators':[3],'criterion':('gini','entropy','log_loss'),'max_depth':[85],'random_state':[42]}
hparam4 = {'n_estimators':[4],'criterion':('gini','entropy','log_loss'),'max_depth':[64],'random_state':[42]}
hparam5 = {'n_estimators':[5],'criterion':('gini','entropy','log_loss'),'max_depth':[51],'random_state':[42]}
hparam6 = {'n_estimators':[6],'criterion':('gini','entropy','log_loss'),'max_depth':[42],'random_state':[42]}
hparam7 = {'n_estimators':[7],'criterion':('gini','entropy','log_loss'),'max_depth':[36],'random_state':[42]}
hparam8 = {'n_estimators':[8],'criterion':('gini','entropy','log_loss'),'max_depth':[32],'random_state':[42]}
```

Figura 25: Hiper parámetros para el RF de LSM6DSOX

Tras la definición de las mallas se ejecuta una búsqueda sobre cada una de ellas y se imprimen los resultados de la mejor puntuación de cada combinación. Posteriormente se introduce toda esta información en *arrays* de forma que para cada caso se elijan los hiper parámetros de forma automática (Figura 26).

```
# Array de hparams
clf_array = np.array([clf1, clf2, clf3, clf4, clf5, clf6, clf7, clf8])
# Array de las mejores puntuaciones
clf_score_array = np.array([clf1.best_score_, clf2.best_score_,
                            clf3.best_score_, clf4.best_score_,
                            clf5.best_score_, clf6.best_score_,
                            clf7.best_score_, clf8.best_score_])

# Se selecciona la mejor de todas las puntuaciones
best_score = clf_score_array.max()
# Se selecciona el nombre de la posicion donde está la mejor puntuacion
best_clf = clf_array[np.where(clf_score_array == best_score)[0]]

# Se tiene ya el clasificador con los mejores parametros
rfc.set_params(**best_clf[0].best_params_)
rfc.fit(dataset_train, dataset_train_label)

# Se predice el resultado
label_predict = rfc.predict(dataset_train)
```

Figura 26: Elección automática de hiper parámetros

Capítulo 5: Resultados

En este capítulo se presentan los resultados obtenidos al aplicar la metodología descrita en el Capítulo 3 que ha sido implementada en el 4. Entre cuadernos *Jupyter* para los preprocesados y *scripts* de *Python* para el entrenamiento de los clasificadores, se han generado un total de 460 archivos de código con un peso medio de 75 KB. A partir del *dataset* original, de ocho millones de muestras, se han creado 96 *datasets* intermedios útiles. Unidos a los modelos que se han ido exportando tras evaluar su rendimiento suman un tamaño total de 28 GB de datos producidos.

Los datos que se han ido generando se han recopilado de forma íntegra y tabulado en el material complementario de este proyecto para que sean accesibles. En este capítulo, se les representará mediante gráficos que recojan la información fundamental y que servirán de apoyo para entender las ideas que se expongan en capítulo siguiente a este, el de las conclusiones. Por no saturar al lector con información redundante, se prescinde de la representación de los resultados obtenidos para la sensibilidad y la exactitud, ya que la *F-score* abarca en cierta forma a estas métricas.

Este capítulo se estructura en 5 partes. En la primera se muestran los resultados de cada clasificador mediante gráficas que analizan el impacto del tamaño de ventana y de la superposición en función del etograma y las características utilizadas. A continuación, se aborda el problema de la genericidad de forma similar ,exponiendo los resultados de la inferencia para datos de ovejas. Después se evalúa la viabilidad de los modelos de referencia comparándolos con la simulación del módulo LSM6DSOX. Por último, se hace una revisión de las características más empleadas y se ofrece una perspectiva general de todos los modelos.

5.1 *Random Forest* ideal

Para el etograma completo (KAM), de nueve actividades, y el conjunto de características básico (TES) (Figura 27), los modelos de referencia elaborados con RF demuestran una progresión positiva en la precisión tanto en función del tamaño de ventana como en función del solape. En la gráfica 27(A) se aprecia un continuo crecimiento de la precisión, llegando hasta el 93% en el mejor caso. En cambio, respecto a la *F-score*, los resultados son en general mediocres salvo en el caso de la ventana de diez segundos, que son muy buenos (27(C) y 27(D)).

En la Figura 28, al incluir el set completo de características (TESTOT), se aprecia que a pesar de que se mantenga la tendencia alcista de la precisión en función del tamaño de ventana y de la superposición, los resultados son de media un poco inferiores. Pese a ello, la configuración de preprocesado con mejor resultado, supera por un 1% a la mejor del caso anterior, llegando a un 94%. El comportamiento de la *F-score* no cambia mucho con respecto a lo comentado en el apartado anterior.

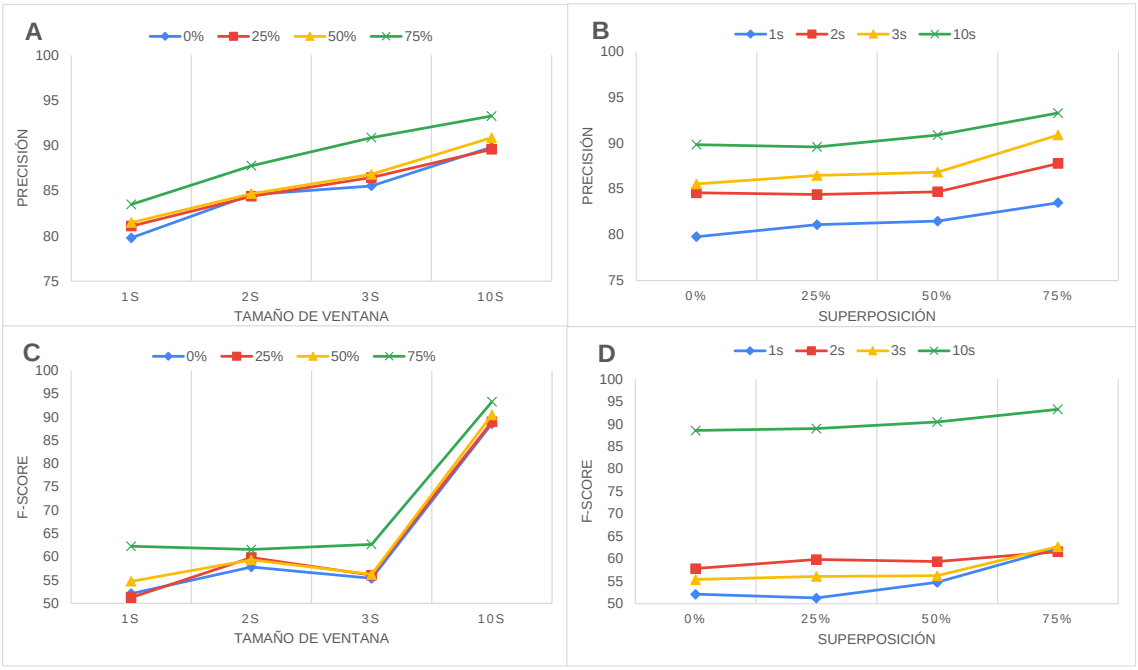

Figura 27: RF Ideal. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TES

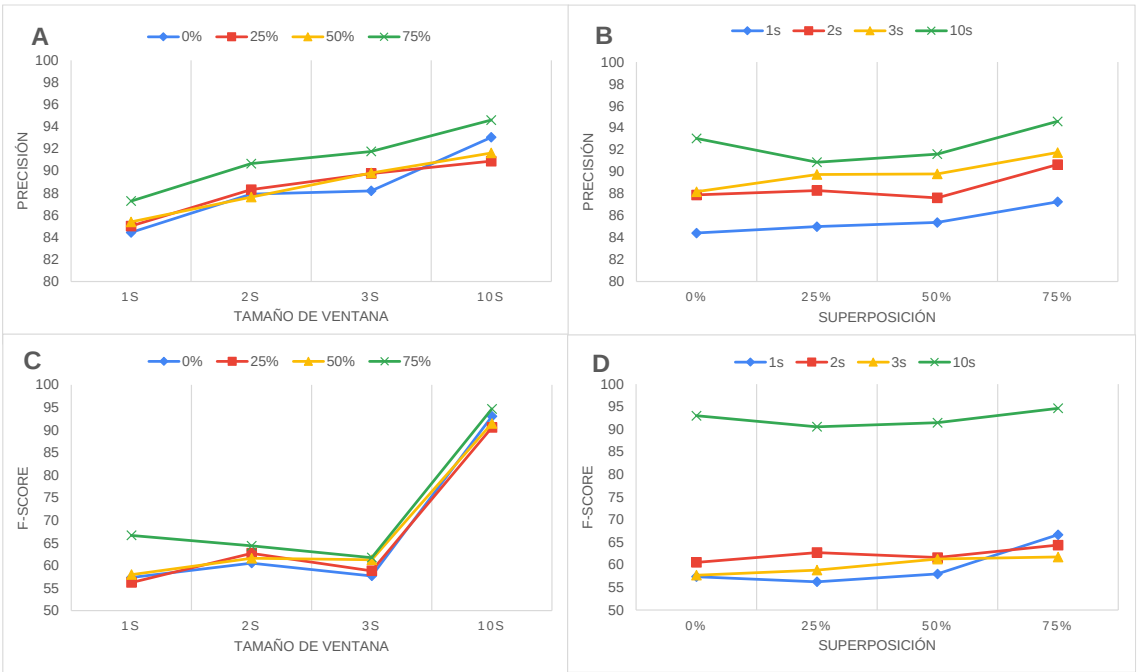

Figura 28: RF Ideal. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TESTOT

Al usar el etograma y el conjunto de características reducido (CUS y TES) (Figura 29) se observan los mismos patrones que en la Figura 27: la precisión aumenta con el tamaño de ventana y con el solape. La precisión media se incrementa muy poco, siendo el mejor resultado de un 94%. En cambio, la *F-score* (29(C) y 29(D)) se ve incrementada en valor medio un 17% aproximadamente debido al cambio de etograma y llega a puntuaciones bastante altas (93%).

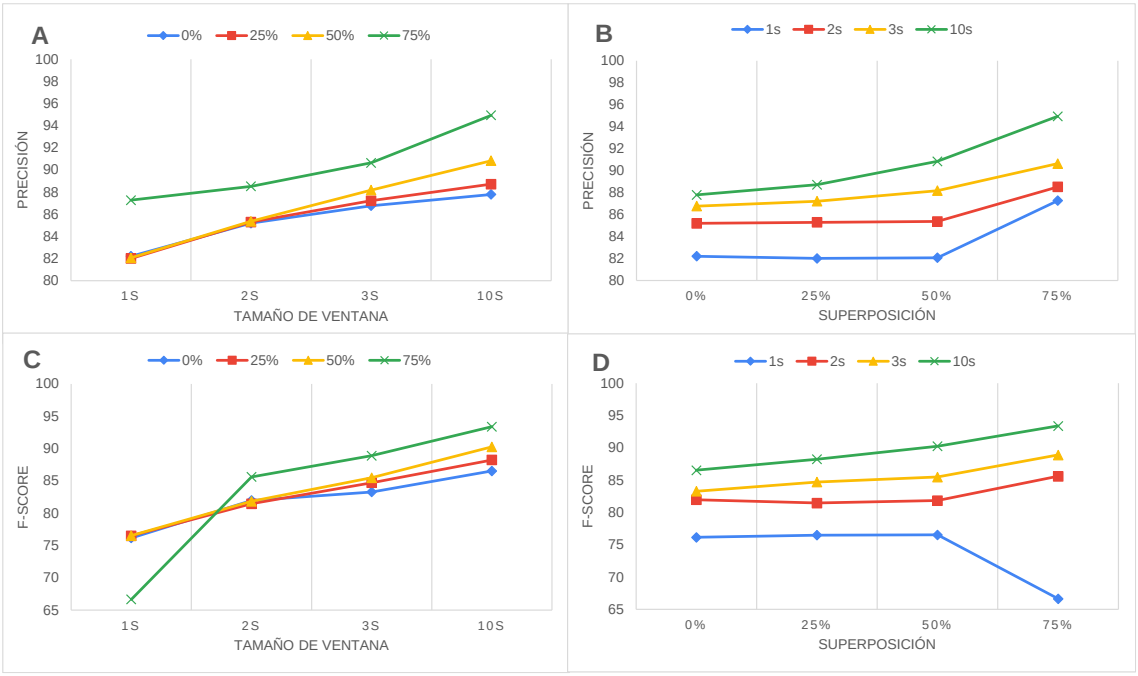

Figura 29: RF Ideal. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TES

La Figura 30 es similar a la Figura 29 en cuanto a las tendencias de los datos (incremento de la precisión y la *F-score* tanto con el tamaño de ventana como con el solape). El uso de un conjunto más amplio de características para la etapa de selección en este caso afecta al incremento global de ambas métricas. El preprocesado que mejor resultados da es la combinación de ventanas de diez segundos con superposición del 75%, llegando al **95.81%.**

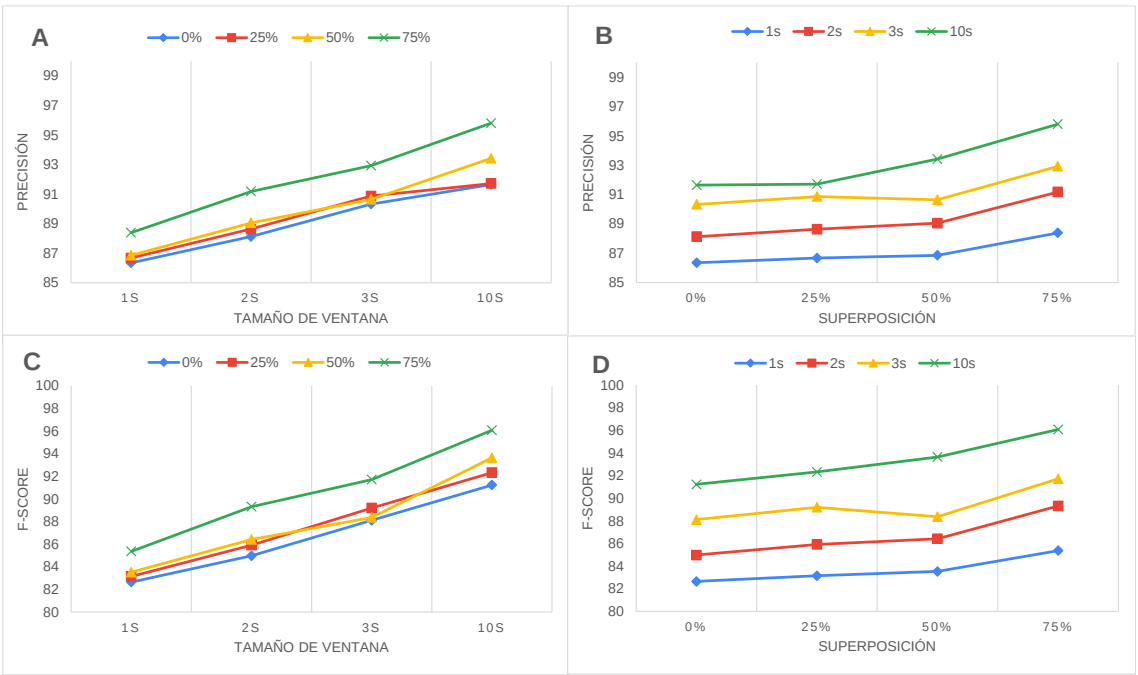

Figura 30: RF Ideal. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TESTOT

5.2 *Naive Bayes*

5.2.1 *Gaussian Naive Bayes*

En la Figura 31 se observa el análisis de los resultados para el etograma original y el conjunto reducido de estadísticas. Se observa en el gráfico 31(A) que la precisión aumenta hasta un 13% al emplear ventanas más grandes, independientemente de la superposición que se aplique. En el 31(B) se aprecia lo mismo, la precisión no varía prácticamente para los diferentes porcentajes. En cuanto a la *F-score*, se evidencia que el mejor resultado es para las ventanas de 10 segundos según lo señalado en 31(C) y 31(D).

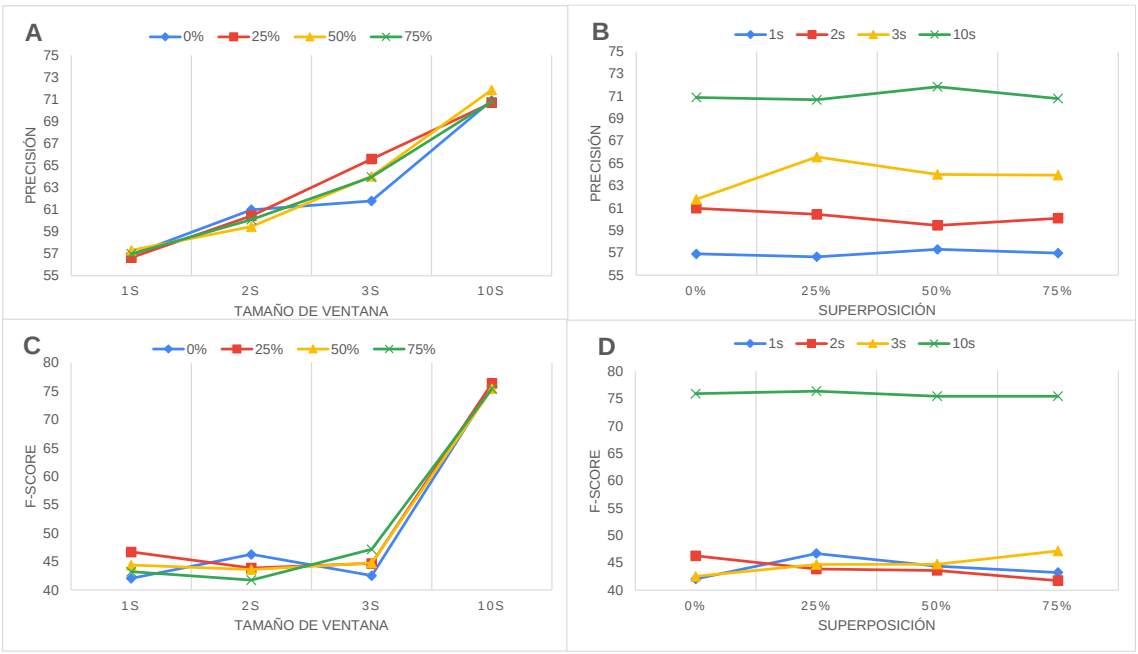

Figura 31: GNB. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TES

De la Figura 32 destaca el aumento de precisión en el gráfico A, de media un 16% superior. También llama la atención que, a diferencia del caso anterior, la precisión parece decrecer un poco con el aumento de la superposición, pero sólo para las ventanas de 10 segundos.

En la Figura 33 se ven los resultados de emplear un etograma de cuatro actividades y el conjunto reducido de características. La precisión es similar o peor que en la Figura 31, sin embargo, la *F-score* mejora respecto a ella siendo mejor para ventanas más grandes.

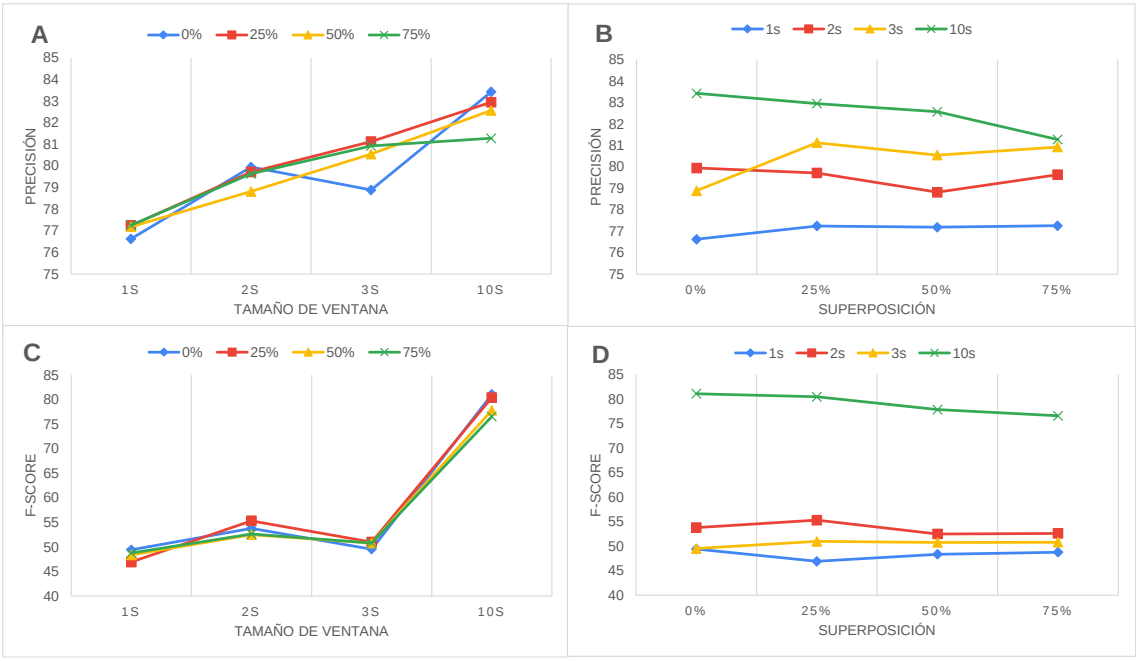

Figura 32: GNB. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TESTOT

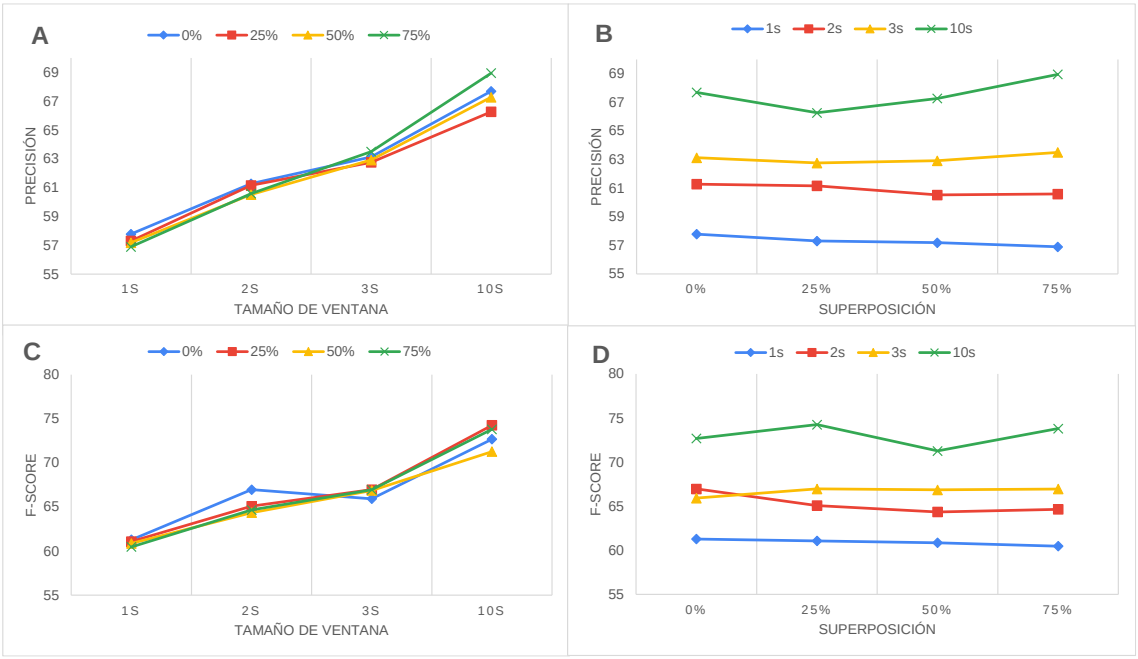

Figura 33: GNB. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TES

La Figura 34 representa unos resultados en cuanto a precisión muy similares a los de la Figura 32, sin embargo, destaca porque la *F-score* es sensiblemente superior para todos los tamaños de ventana menos para la de diez segundos.

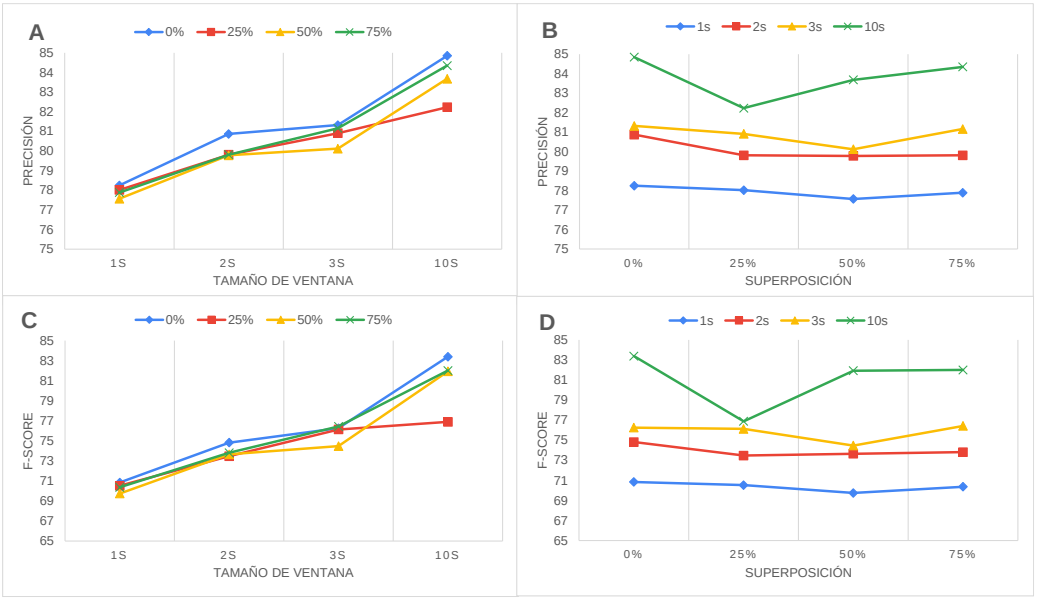

Figura 34: GNB. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TESTOT

Comparando los resultados con los de los modelos de referencia se detecta una diferencia muy importante en la precisión. Es además una diferencia heterogénea ya que de media ronda el 23% para preprocesados con el conjunto de características reducido y el 9% para los que emplean el conjunto completo. La inclusión de más características parece vital para la inferencia.

5.2.2 *Bernoulli Naive Bayes*

Esta implementación contrasta con la anterior. Mientras que a grandes rasgos se cumple que la precisión mejora conforme al aumento del tamaño de ventana y la superposición, llama la atención que no suceda igual con las características: se nota mejora entre la Figura 35 y la Figura 36 al usar el conjunto completo, pero no tan grande como en el caso de GNB.

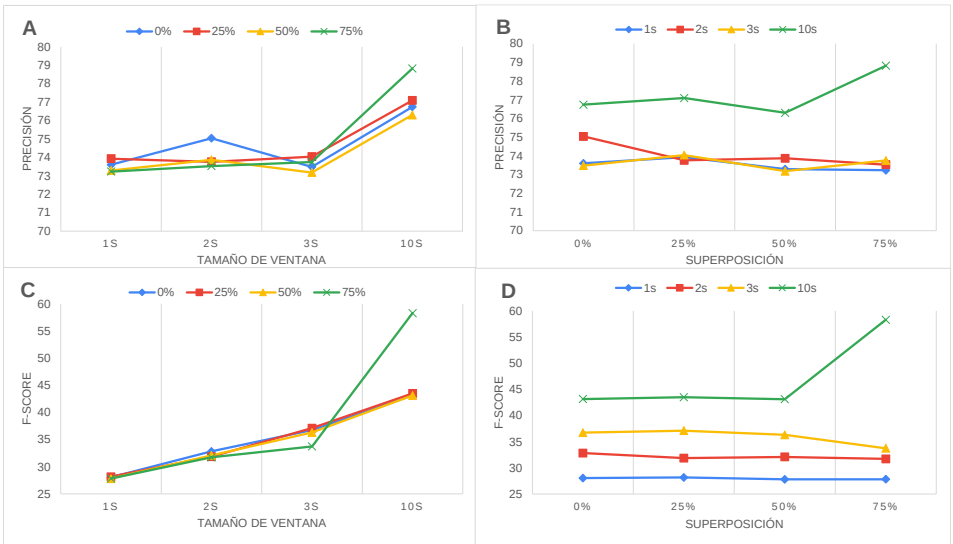

Figura 35: BNB. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TES

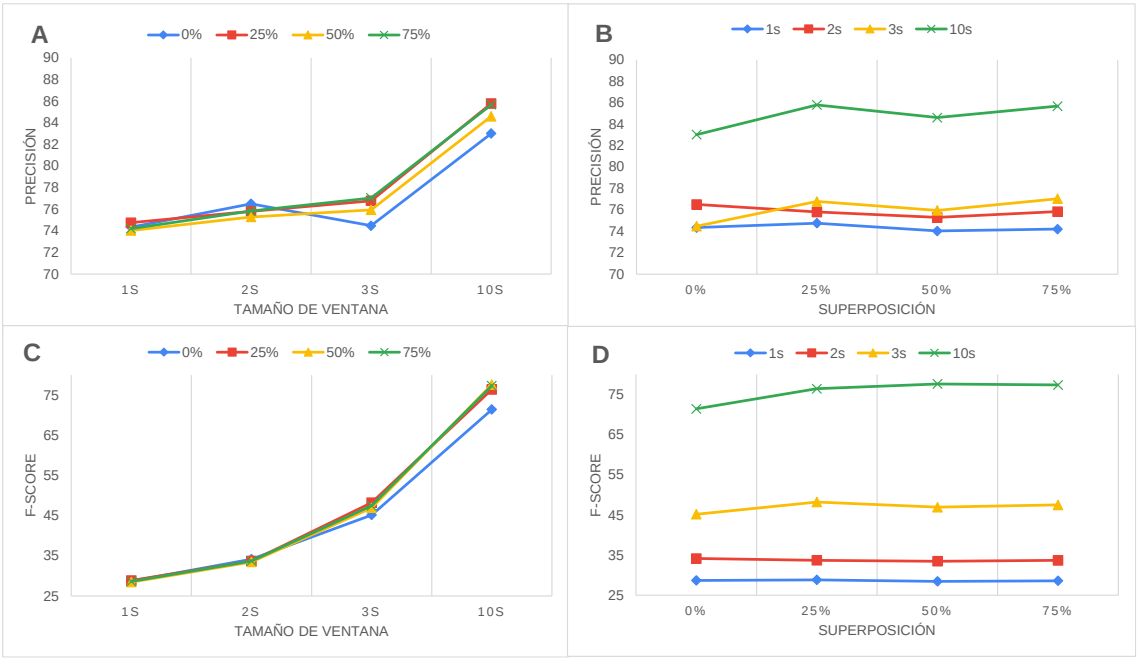

Figura 36: BNB. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TESTOT

Además, se diferencia del modelo anterior en cómo es la *F-score*, con muy malos resultados en general para los tamaños de ventana pequeños. Al emplear el etograma reducido de cuatro actividades (Figura 37 y Figura 38) dicha puntuación mejora algo en las ventanas de uno, dos y tres segundos, aunque para la de diez segundos se mantenga igual.

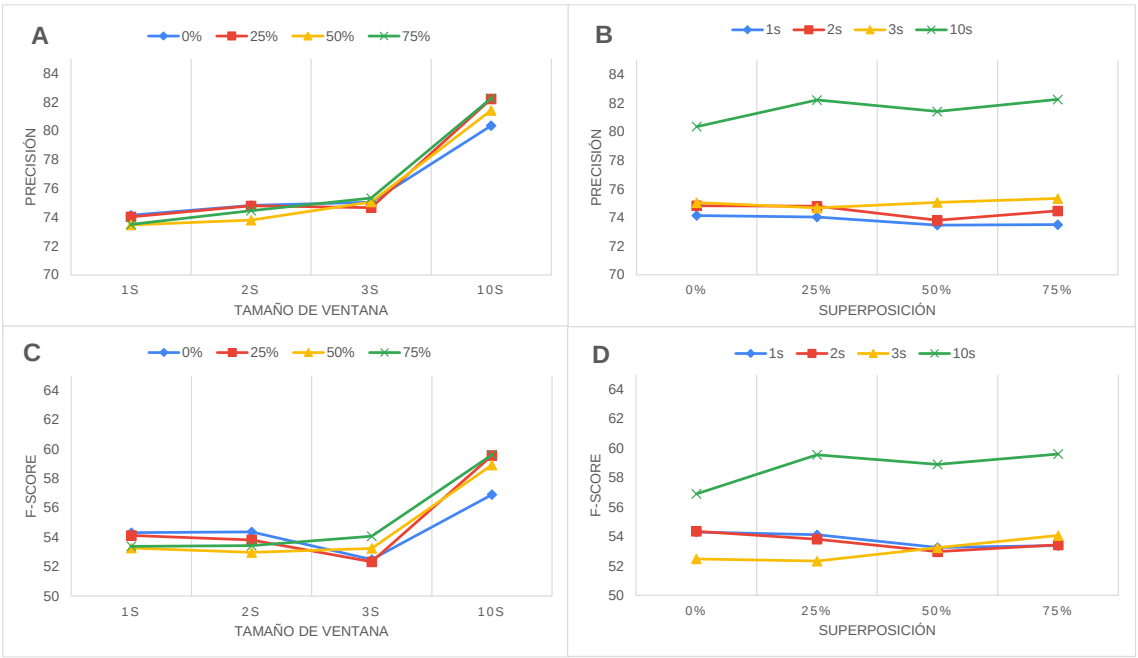

Figura 37: BNB. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TES

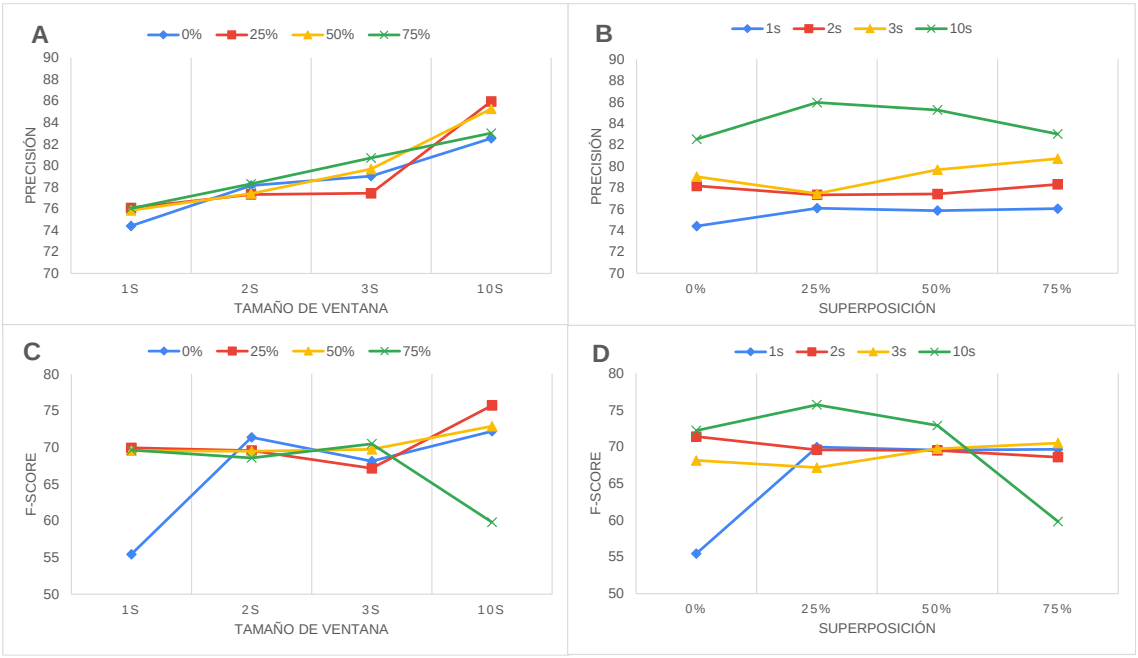

Figura 38: BNB. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TESTOT

A diferencia de GNB, la comparación de los resultados con los modelos ideales arroja bastante homogeneidad. Independientemente del conjunto de actividades y de las características que se elijan, la diferencia en precisión siempre ronda el 11% de media. El mejor caso de BNB es peor que el mejor de GNB, sin embargo, los peores resultados de la implementación con *Bernoulli* son bastante mejores que los peores de la implementación Gaussiana.

5.3 *K Nearest Neighbors*

Los resultados de ejecutar este algoritmo para los *datasets* generados con un etograma compuesto por nueve actividades y las características más sencillas se muestran en la Figura 39. En los gráficos A y B de dicha figura se aprecia que existe una relación evidente entre la precisión y el tamaño de ventana, puesto que a medida que la ventana crece, la precisión pasa del 80% al 91%. Esto no sucede con la superposición pues la precisión se mantiene para cada porcentaje de solape. Algo similar sucede con la *F-score,* aumenta considerablemente para las ventanas más grandes, pasando a estar alrededor del 90%, lo cual es un resultado muy bueno porque indica que existen muy pocos falsos positivos y falsos negativos.

Al añadir nuevas características al análisis (Figura 40), los patrones se mantienen similares. El único cambio significativo es el aumento de la precisión media un 3% y el de la *F-score* un 2%. Se alcanza la precisión más alta de todos los experimentos con KNN, 95.14%.

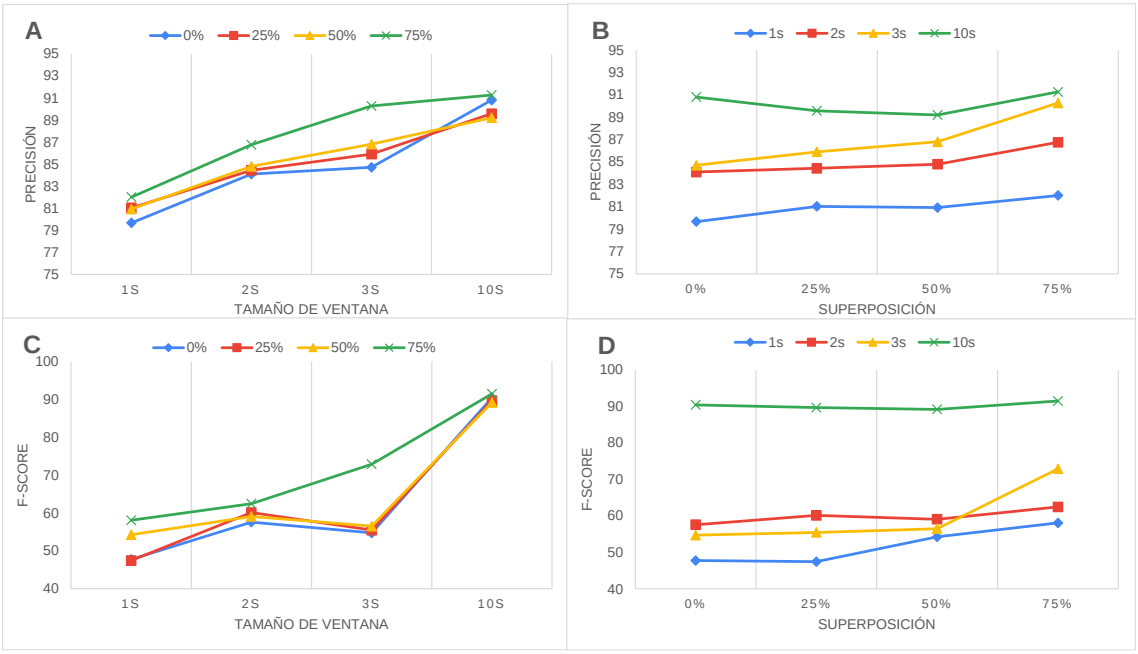

Figura 39: KNN. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TES

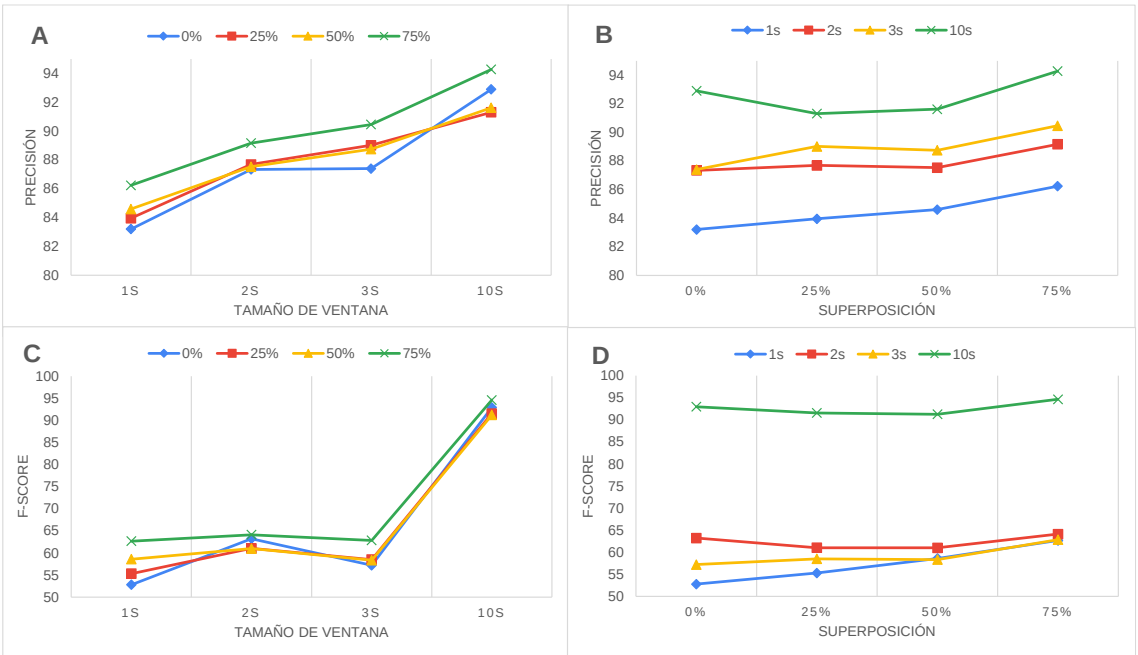

Figura 40: KNN. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TESTOT

Sin embargo, si se reduce el etograma a cuatro actividades y se vuelve a emplear el conjunto reducido de características (Figura 41) se observa un patrón diferente en la evolución de la precisión en función de la superposición: existe cierta mejora con el incremento de la superposición. A pesar de esto, la precisión media es menor que en el caso anterior (Figura 40) y muy parecida a la mostrada en la Figura 39. También se observa una mejora sustancial en la *F-score*, siendo el resultado más bajo de 75.5% cuando en los casos anteriores rondaban el 50%.

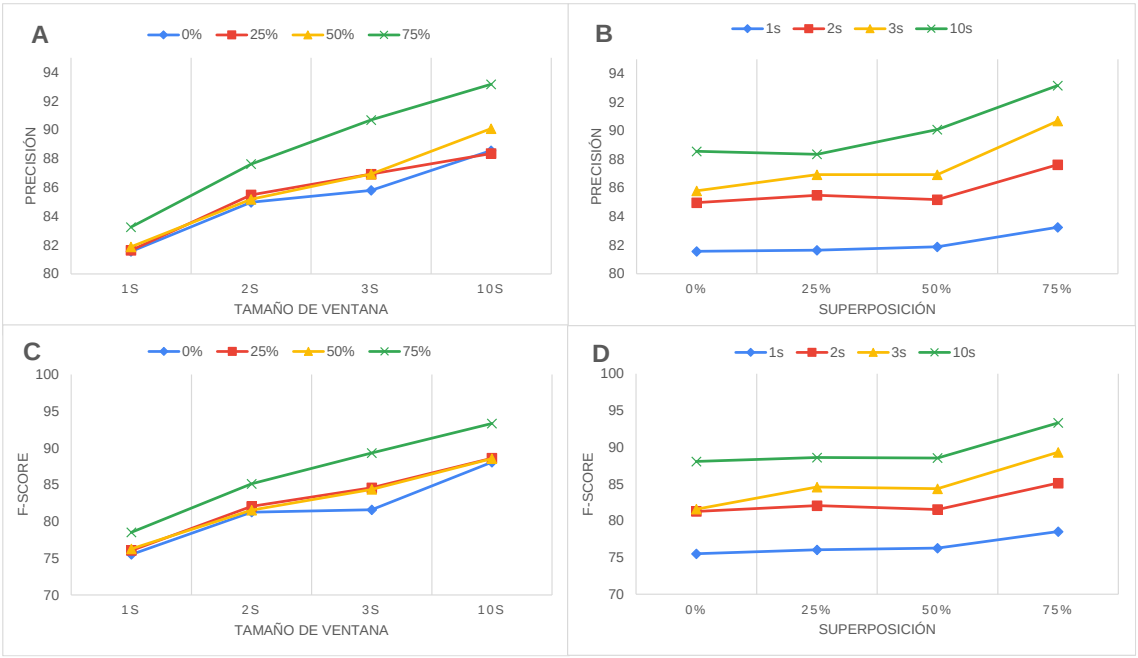

Figura 41: KNN. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TES

En la Figura 42 se hace más patente lo comentado en el caso anterior, existe una progresión positiva entre la precisión y la superposición que se hace evidente también para la *F-score*. Estas configuraciones ofrecen los valores medios más altos de precisión y *F-score.*

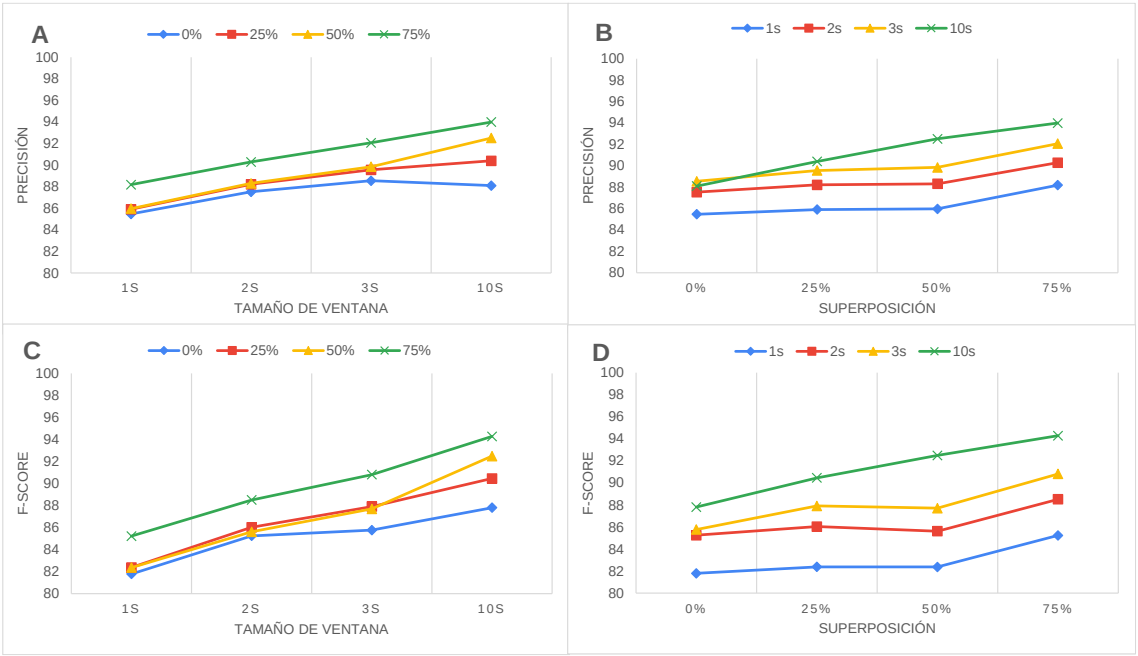

Figura 42: KNN. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TESTOT

En comparación con los modelos de referencia de *RF*, se alcanzan precisiones muy similares, habitualmente un 0.6% inferiores, salvo para la combinación de etograma reducido y todas las características, que la brecha llega hasta el 1% de media. En cuanto a *F-score*, la

diferencia es aún menor, siendo KNN incluso capaz de mejorar los resultados de los modelos de referencia en algunos casos. Hay que tener en cuenta para valorar estos resultados que el algoritmo KNN es mucho más rápido en el entrenamiento que los RF ideales y que los modelos generados ocupan bastante menos.

5.4 *Support Vector Machines*

Tras la ejecución del entrenamiento del clasificador SVM se evalúan sus resultados al igual que en los apartados anteriores. En Figura 43 se observa la misma tendencia en cuanto a la precisión del resto de clasificadores, aumenta conforme aumenta el tamaño de ventana, aunque en este caso se observe (A) un aplanamiento para las superposiciones de 50 y 75% en el tamaño de ventana superior. Por otro lado, se aprecia en el gráfico B un comportamiento que sólo se había visto en un caso anteriormente. Al igual que en la Figura 32, para el clasificador GNB, la precisión tiene una tendencia inversamente proporcional al incremento de la superposición, pero sólo para la ventana de mayor tamaño. La *F-score* sólo cosecha buenos resultados en esta ventana como se aprecia en los gráficos C y D.

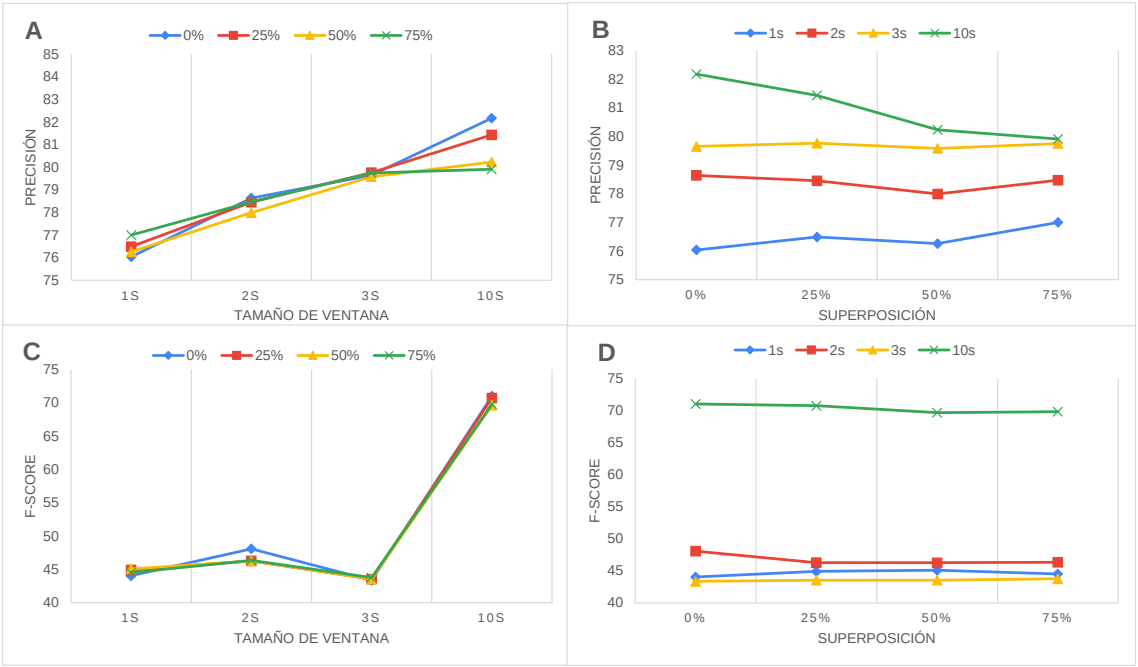

Figura 43: SVM. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TES

En la Figura 44, al igual que en la anterior se dan los mismos patrones, y destaca aún más si cabe, que el mejor resultado de precisión sea para la ventana de diez segundos sin utilizar ningún tipo de solape. En este caso alcanza la puntuación más alta de todos los entrenamientos, 87.33%. Se aprecia la diferencia de incluir más características ya que la precisión aumenta de media un 3%. La *F-score* permanece sin cambios destacables.

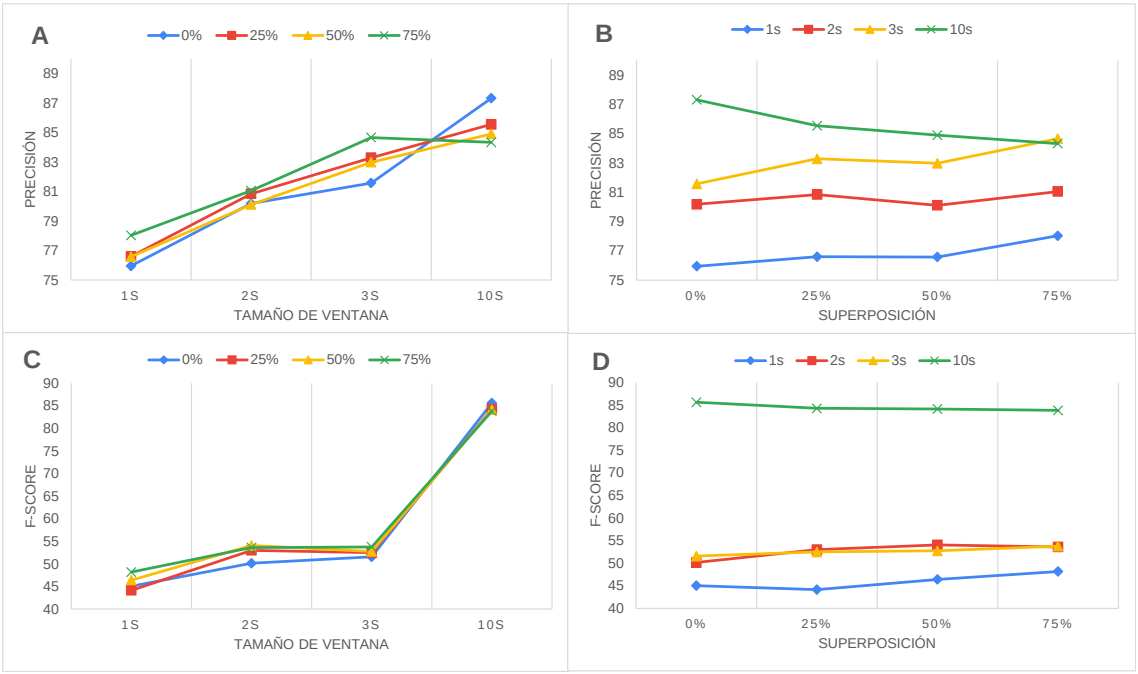

Figura 44: SVM. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TESTOT

Al reducir el etograma a cuatro actividades, se aprecia una mejora relevante en la *F-score*, llegando a aumentar casi un 20% en el caso de las ventanas de 1 segundo. Pese a esta mejora media, en las ventanas de 10 segundos no se aprecia apenas cambio, dando la sensación de que, por el tipo de clasificador y datos, es el máximo que se puede alcanzar en esta métrica con el conjunto reducido de características (alrededor del 70%). La precisión media se mantiene estancada respecto a la Figura 43 y disminuye un poco respecto a la Figura 44.

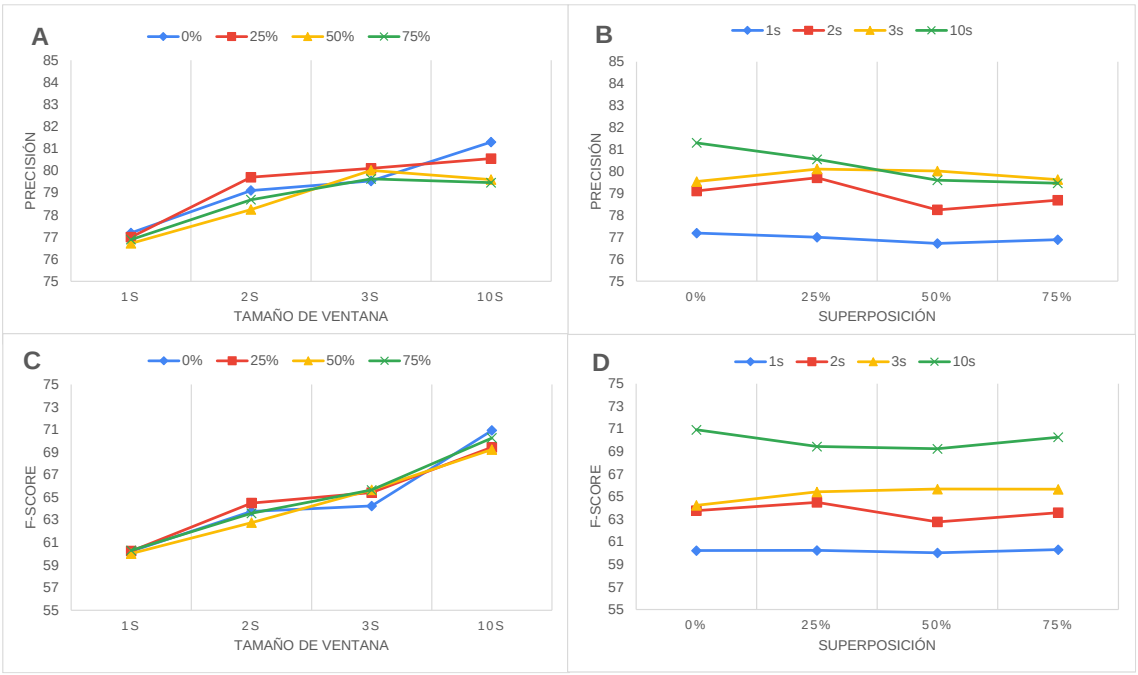

Figura 45: SVM. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TES

Pese a la reducción del etograma y la inclusión de más características, la precisión como se aprecia en la Figura 46 no varía apenas con respecto a los casos anteriores. Mejora un 1.2% de media respecto a lo visto en la Figura 44, pero sin alcanzar en ningún momento una precisión superior a la máxima de dicho caso. Aquí no parece darse la situación en la que la precisión disminuya con el porcentaje de solape. En cambio, sí se nota un incremento general de la *F-score* con la inclusión de nuevas métricas. Esto es llamativo porque hasta ahora esta métrica estaba íntimamente ligada a la elección del etograma, y aquí parece depender más de las características.

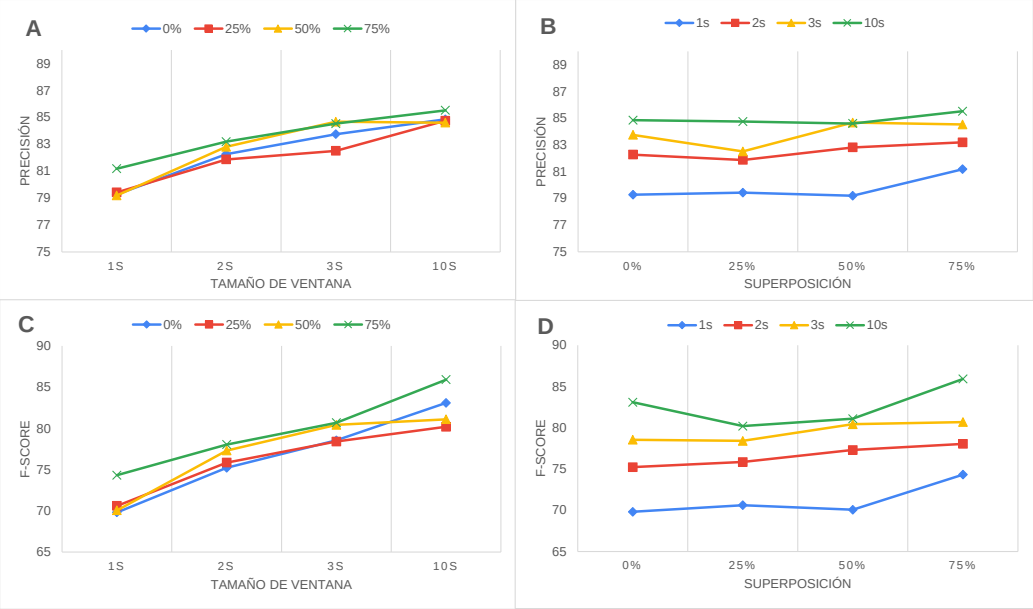

Figura 46: SVM. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TESTOT

Si se analiza el tiempo que se han tardado en entrenar estos clasificadores y las expectativas depositados en ellos tras el análisis de la literatura, resulta un tanto decepcionante que sus resultados estén de media siempre un 8% por debajo de los modelos de referencia en cuanto a precisión y un 15% en cuanto a *F-score*.

5.5 Ovejas

En la Figura 47 se observan los resultados de la inferencia para datos de ovejas usando un modelo entrenado con datos de cabras donde se emplea el etograma del *dataset* original. Son resultados aparentemente dispares en los que la precisión orbita entre el 40 y el 75% y la *F-score* no excede en ningún momento el 50%. Hay que tener en cuenta que los autores del *dataset* reconocen hasta nueve actividades diferentes para las cabras, pero las ovejas tan sólo realizan la mitad de ellas. Este hecho ayuda a explicar lo baja que es la métrica de *F-score*.

Estos efectos también se observan en la Figura 48. Aunque la precisión llegue a ser aceptable en algunos puntos superando el 80% de aciertos, la *F-score* se mantiene siempre por debajo del 50%.

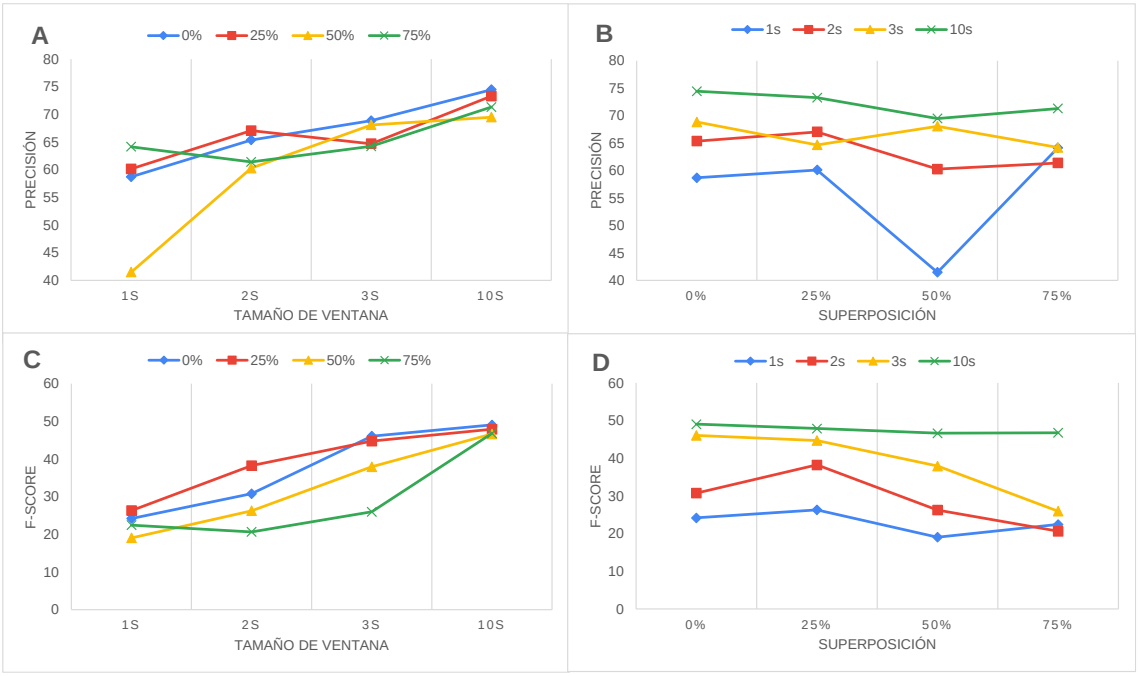

Figura 47: Ovejas. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TESTOT

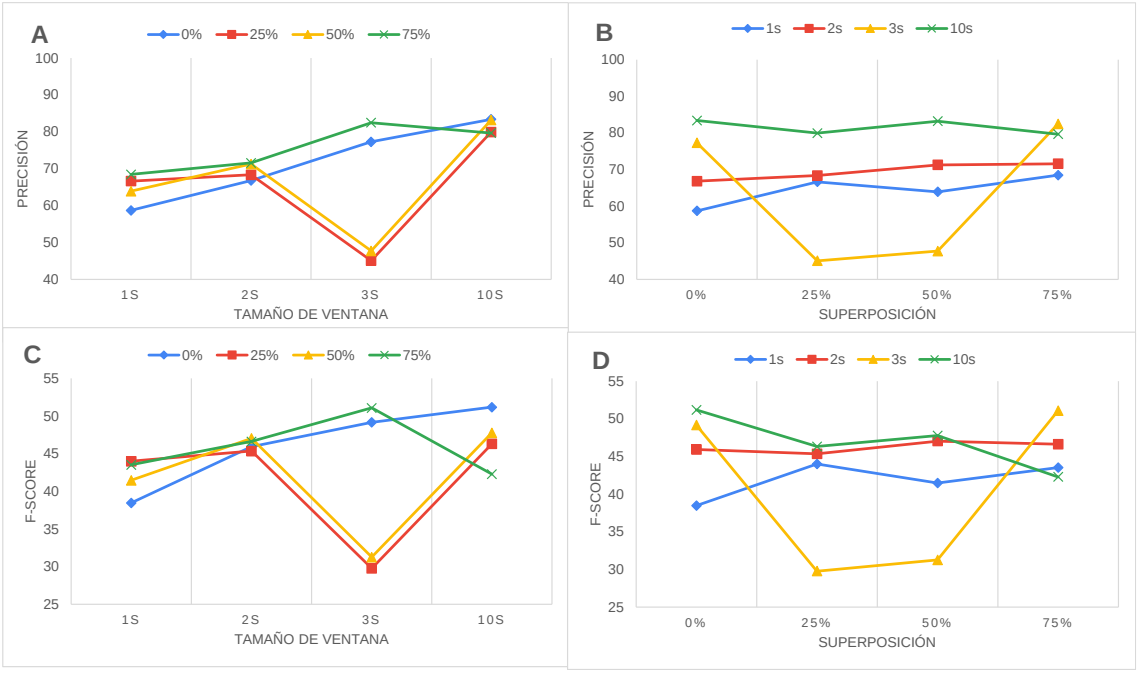

Figura 48: Ovejas. Precisión y F-score en función del tamaño de ventana y el solape para CUS – TESTOT

Los resultados de media son un 22% inferiores a los del modelo de referencia para la precisión y de un 40% para la *F-score.* La diferencia entre precisión y *F-score*, así como la variabilidad de los resultados indican que existen muchos falsos positivos y falsos negativos. Esto puede no lastrar a la precisión si el *dataset* de entrada está desequilibrado, de tal forma que muchas muestras se clasifican bien porque una actividad es mucho más habitual que las demás, que por el contrario siempre se clasifican mal.

5.6 LSM6DSOX

En este apartado se analizan los resultados de la simulación del módulo LSM6DSOX. Como se ha explicado previamente, se han obtenido a través del modelado de un algoritmo RF usando para ello los parámetros descritos por el fabricante. Sin embargo, hay que señalar antes de pasar a la exposición de los resultados que, si bien el clasificador se ha emulado satisfactoriamente, el preprocesado de los datos que se ha ejecutado en este trabajo no es exactamente igual al que implementa el circuito comercial mencionado. Para analizar y valorar los resultados obtenidos ha de tenerse en cuenta este hecho.

Al observar los datos y representarlos gráficamente como se ha hecho hasta ahora se observa que existe una relación clara entre los resultados de la emulación y los resultados de los modelos ideales que se han entrenado antes. Si se compara la Figura 49 con la Figura 27 se observa que la distribución de los datos sobre las gráficas sigue los mismos patrones. La única diferencia reside en las puntuaciones de precisión y *F-score* donde el modelo ideal supera a la emulación por 1.5% y 0.7% de media respectivamente. Este hecho se repite en el resto de las configuraciones de etograma y características, con lo que se prescinde de representar sus resultados gráficamente por no añadir información redundante.

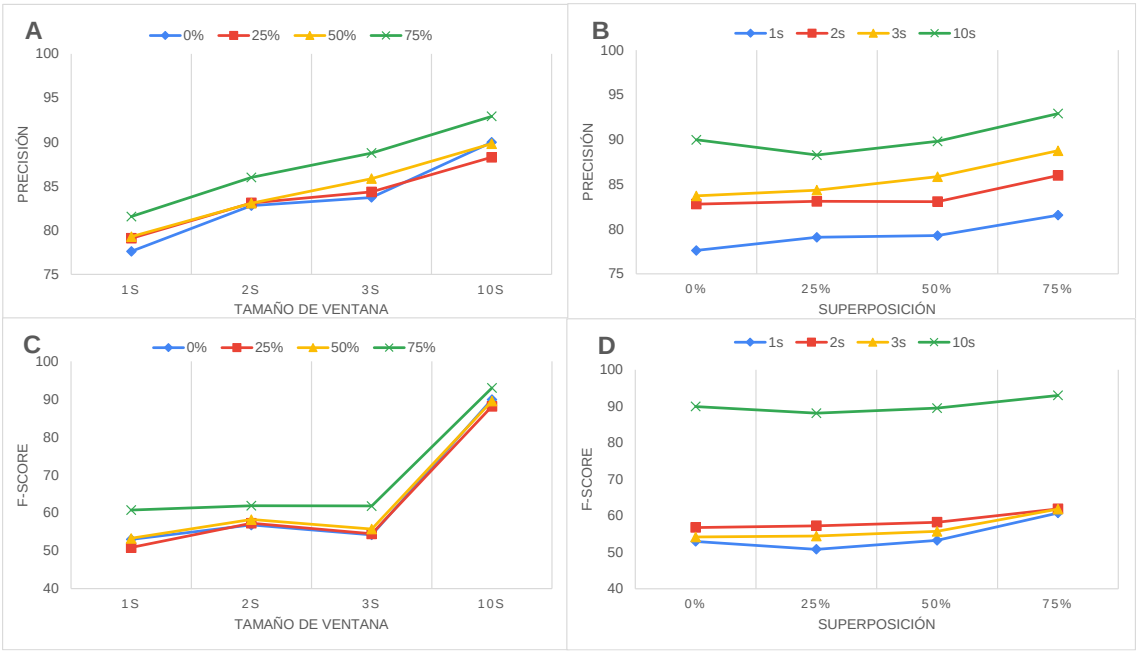

Figura 49: LSM6DSOX. Precisión y F-score en función del tamaño de ventana y el solape para KAM – TES

La emulación del módulo LSM6DSOX llega a tener una precisión máxima del 94.21% con una *F-score* del 94.15% para la configuración correspondiente al etograma reducido, el tamaño de ventana de 10 segundos, la superposición del 75% y el conjunto completo de características. Son resultados muy buenos para un modelo mucho más sencillo que el ideal, cuyo tiempo de entrenamiento es muy bajo en comparación.

5.7 Características más usadas

Además de las métricas descritas en apartados anteriores, se han registrado las características que *Backward Elimination* ha escogido como las más representativas en cada caso y para cada clasificador. En la Tabla 3 se identifica cada código empleado en los gráficos con la característica a la que corresponde.

Leyenda de las características empleadas

mean: Media	**v_min:** Valor mínimo	**v_max:** Valor máximo	**minmax:** Rango entre min. y max.	**std:** Desviación estándar	**par_corr_xy**: Correlación entre ejes X e Y	**energy:** Energía
entropy: Entropía	**q25:** Cuartil 25%	**q75:** Cuartil 75%	**q2575:** Rango entre cuartiles	**mov_var:** variación de movimiento	**par_corr_xz:** Correlación entre ejes X y Z	**skewness:** Asimetría
kurtosis: Curtosis	**mean_abs_changes**: media de cambios valor abs.	**rms_sv:** Valor RMS de la señal	**sma:** Área magnitud de la señal	**zcr:** Ratio de cruces por cero	**par_corr_yz:** Correlación entre ejes Y y Z	**median:** Mediana

Tabla 3: Leyenda de las características empleadas

En la Figura 50 se representan las características más usadas en los modelos que se han elaborado. Llama la atención que las más empleadas netamente sean características tan sencillas como el cuartil 25, el rango entre cuartiles o el valor máximo, que ofrecen información acerca de rangos estadísticos. Entre las más usadas están la mediana, la media y la varianza que describen mejor la distribución de las muestras dentro de una ventana.

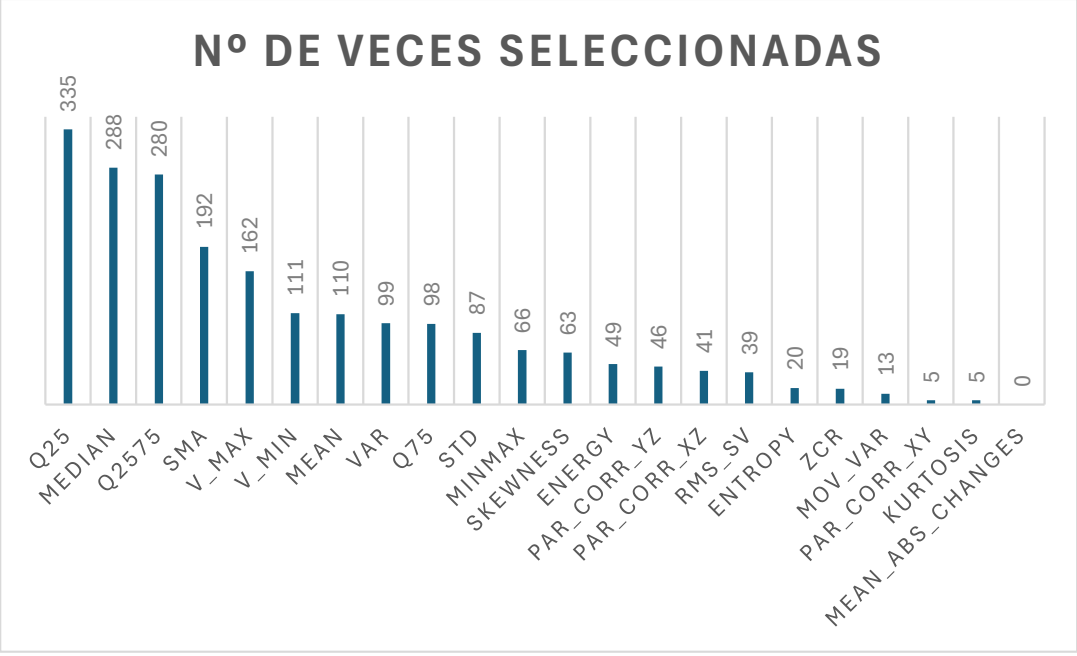

Figura 50: Características más empleadas

De las características adicionales sólo está entre las cinco más usadas SMA, que está relacionada indirectamente con la energía de la ventana. La asimetría y la curtosis también parecen relevantes en algunos casos y aportan información acerca de la distribución de las muestras. El resto de las características del conjunto de "*otras*" apenas han sido utilizadas.

Hay que tener en cuenta que estas características sólo tienen la posibilidad de aparecer en la mitad de los resultados que las otras, por eso se representa en la Figura 51 cómo se distribuyen porcentualmente.

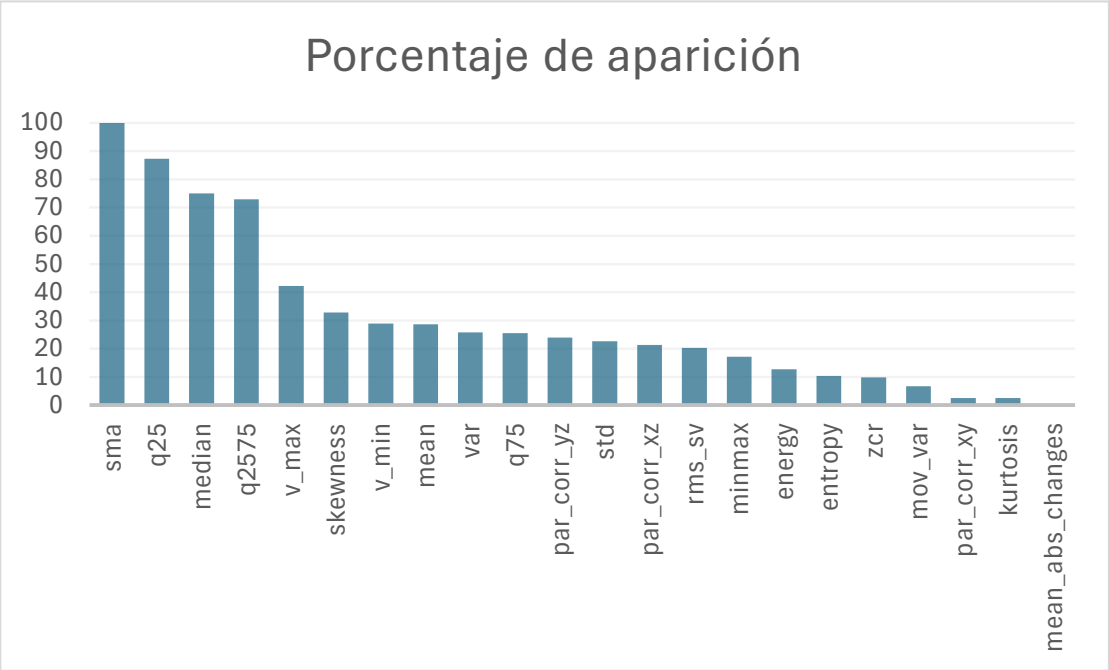

Figura 51: Porcentaje de selección de cada característica

Signal Magnitude Area tiene un porcentaje del 100%, esto significa que el seleccionador de características siempre que ha podido elegirla lo ha hecho con lo que se estima que ofrece mucha información útil para diferenciar entre clases a la hora de hacer la inferencia.

5.8 Vista general de los clasificadores

Hasta ahora se ha ofrecido una visión pormenorizada del rendimiento de los clasificadores atendiendo fundamentalmente a variaciones de parámetros tales como el tamaño de ventana y la superposición. En menor medida se ha comentado también el efecto del etograma en función de si es el completo (KAM) o si es el reducido (CUS), así como el de los conjuntos de características empleados: (TES) para el conjunto reducido y TESTOT para el completo.

Para poner un poco en perspectiva lo analizado, se muestra en la Figura 52 una comparación entre los diferentes clasificadores usados en el trabajo. Obsérvese cómo KNN y el RF entrenado para emular el comportamiento del módulo LSM6DSOX (RRFC) se aproximan mucho a la precisión del modelo ideal (RFC). Llama también la atención que todos los clasificadores pierden de media cierta capacidad de acierto al volver a emplear el conjunto reducido de características, como se ve en la sección CUS-TES.

Los puntos que se aprecian con forma de "+" son las evaluaciones de las ovejas, definitivamente muy por debajo del resto.

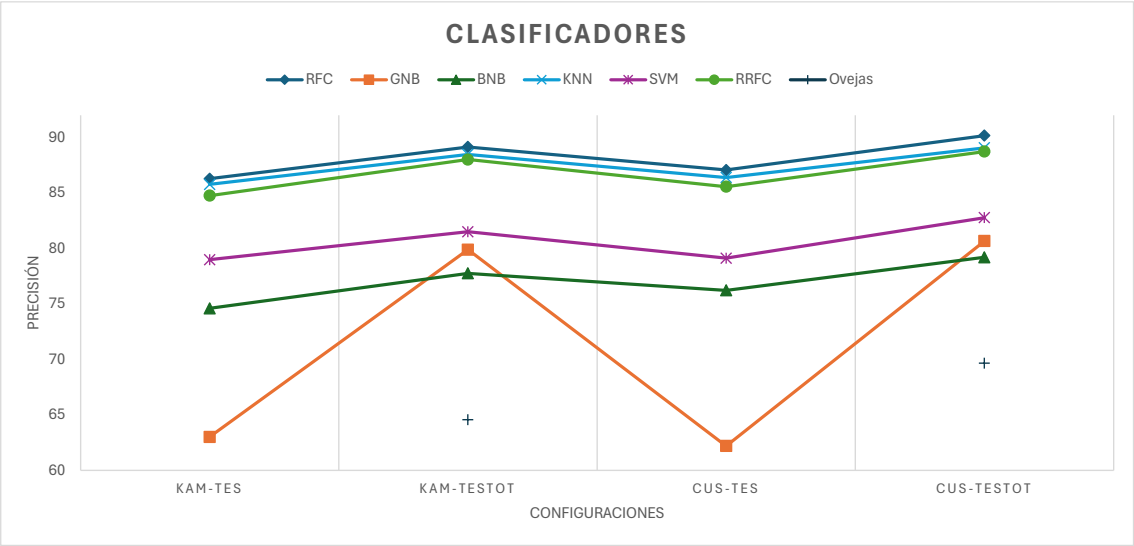

Figura 52: Precisión media de los clasificadores en función de su etograma y características

En la Figura 53 se puede observar la importancia del equilibrio entre clases. Los clasificadores al acceder a un *dataset* con menos actividades, pero más balanceadas entre ellas evitan caer en sesgos al ser todas las actividades igual de importantes para la precisión. De esta manera se reducen los falsos positivos y negativos y, por tanto, se incrementa la *F-score*. Cabe mencionar como en el caso anterior, la sorprendente similitud entre los modelos de referencia basados en RF, los modelos que implementan el algoritmo KNN y la emulación del módulo LSM6DSOX.

Hay que reseñar también la paupérrima puntuación de la inferencia realizada con datos de ovejas, que permanece de media siempre por debajo del 50%. También es interesante observar cómo el algoritmo BNB que conforme a la métrica de precisión funcionaba mejor que GNB, en este apartado es con diferencia el peor de todos. Esto pone de relieve la necesidad de hacer una evaluación exhaustiva utilizando diferentes indicadores siempre que sea posible, ya que, a la vista de esto, GNB sería bastante mejor opción para la AAR que BNB.

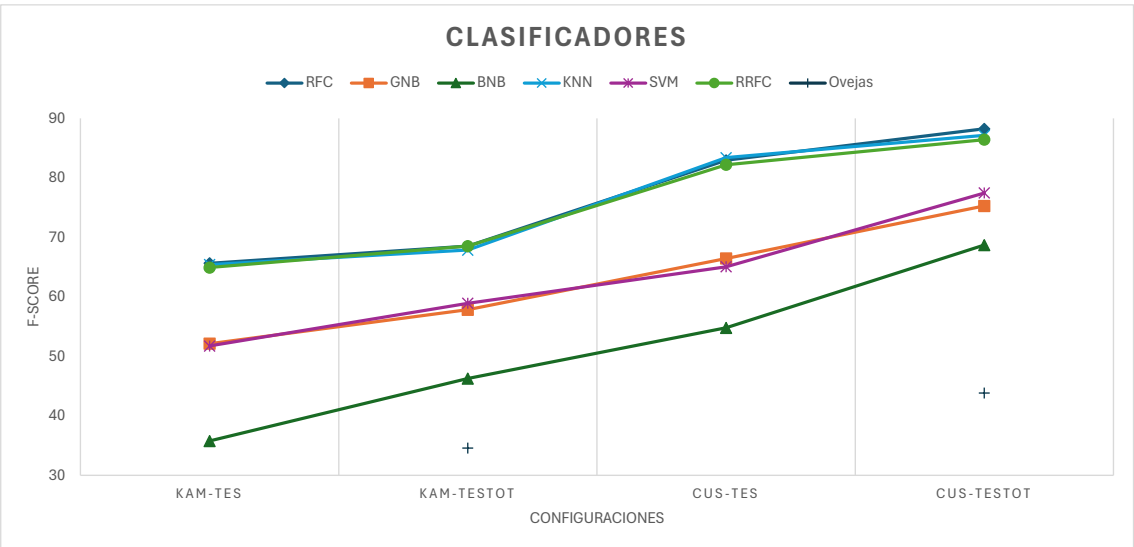

Figura 53: F-score media de los clasificadores en función de su etograma y características

Capítulo 6: Conclusión

En el capítulo final de este trabajo se presentan las conclusiones a las que se ha llegado en vista de los resultados expuestos en el capítulo anterior y los objetivos definidos para este trabajo: hallar una estructura común en la literatura para organizar un proyecto de desarrollo de AAR con herramientas de ML; encontrar las mejores combinaciones de elementos de preprocesado para elaborar un modelo de referencia en base a un *dataset* concreto; ver qué clasificadores ofrecen mejores resultados para dicho *dataset*; y observar qué tan generalizable y viable es el modelo de referencia elaborado.

Finalmente se indican posibles líneas de investigación futuras en base a lo observado durante la realización de este trabajo.

6.1 Conclusiones

Tras la revisión de la literatura se ha encontrado que, pese a que todas las aproximaciones analizadas difieren entre ellas en algún aspecto en concreto, existe cierta estructura subyacente común a todas ellas. Este esquema común parte de la captura de datos en animales, posteriormente integra una fase de adecuación de los datos llamada preprocesado y finaliza con el entrenamiento de un modelo y su evaluación.

En todos estos trabajos previos, se toman ciertas decisiones en cada fase. En este proyecto se ha realizado una evaluación de las que se han considerado más relevantes:

- La selección del etograma resulta fundamental para evitar los problemas de desequilibrio entre clases. Este desequilibrio quizás no se note tanto en la precisión, sin embargo, resulta patente en métricas como la sensibilidad, la exactitud o la *F-score* ya que recogen los falsos positivos y falsos negativos. Los modelos entrenados con *datasets* reales tienden a sesgar sus resultados hacia las actividades más frecuentemente observadas. Por esto, no son capaces de identificar correctamente las clases que, aun siendo igualmente importantes, son menos frecuentes. A la hora de seleccionar un etograma se debe llegar a una solución de compromiso que tenga en cuenta cuantas actividades se necesitan predecir realmente y qué porcentaje mínimo de precisión se necesita para cada una de ellas.
- La elección del tamaño de ventana ha demostrado ser en este trabajo, de extrema importancia en la evaluación de la precisión. Se ha visto que para todos los clasificadores el incremento de la precisión está directamente relacionada con un incremento del número de muestras por ventana. La que mejor resultados ofrece es la ventana de diez segundos y como la tendencia observada es ascendente, sería interesante comprobar qué sucede al incrementar aún más el tamaño.
- El porcentaje de solape entre ventanas ha resultado ser una variable menos relevante de lo que se esperaba en un principio. Su influencia en la precisión y la *F-score* es limitada pues no se ha observado que varíen significativamente aislando esta variable, con la única excepción de KNN. Superponer ventanas es un recurso muy habitual cuando existe

escasez de anotaciones y cuando los datos a analizar presentan muchas transiciones entre actividades. Sin embargo, el *dataset* del que se ha partido parecía ser lo suficientemente grande como para no depender de esto.

- La extracción de características también ha demostrado tener un peso importante en los resultados, con una mejora clara de la precisión para casi todos los clasificadores y con un incremento de la *F-score* en algunos casos. Sin embargo, la extracción de características puede ser un proceso bastante costoso computacionalmente con lo que la selección de las más relevantes resulta también vital de cara a la gestión del consumo energético y la complejidad de cómputo. En este proyecto, las características más comunes han sido cuatro medidas estadísticas básicas y otra medida menos utilizada (SMA) que, sin embargo, ha demostrado ser muy valiosa, ya que siempre que ha estado disponible ha sido elegida.

Habiendo desarrollado una colección de modelos de referencia con *Random Forests*, se ha evaluado comparativamente el desempeño de otros algoritmos presentes en la bibliografía:

- Se han evaluado dos implementaciones diferentes de *Naive Bayes*, que asumen distribuciones de datos diferentes: gaussiana (GNB) y de Bernoulli (BNB). GNB parece depender fuertemente de las características seleccionadas, lo que obliga a realizar una selección cuidadosa de las mismas para ser utilizada. BNB, por el contrario, tiene un desempeño mucho menos dependiente de las características, pero ofrece una *F-score* bastante mala con el dataset seleccionado, lo que impediría usarlo en aplicaciones comerciales. Ambos algoritmos son mucho más rápidos en el entrenamiento que *RF* y también computan la inferencia en menos tiempo, por lo que al menos GNB podría ser un buen candidato para aplicaciones donde el consumo de energía sea un factor clave.
- *K Nearest Neighbors* es un algoritmo que recuerda sus datos de entrenamiento durante la inferencia y define umbrales de decisión en base a las muestras más cercanas. Su proceso de entrenamiento e inferencia es muy rápido en comparación con los modelos de referencia. Por eso sorprende su buen funcionamiento, destacando especialmente en la *F-score*, lo que implica que es muy capaz de distinguir entre clases diferentes. El aumento de la precisión y la *F-score* con respecto al porcentaje de superposición es una anomalía en todo lo analizado y posiblemente se deba a que una mayor cantidad de muestras permite definir mejor unos umbrales precisos. A pesar del poco tiempo que requiere para entrenar e inferir, parece plantear dos problemas. Por un lado, la evolución de los resultados conforme al tamaño de ventana lleva a pensar que, aunque funcione bien, tiene una capacidad limitada para mejorar ya que como se ve en la Figura 52, no escala tan bien como RF. Por otro lado, es un algoritmo muy dependiente de los datos de entrenamiento. Por ello habría que valorar en campo su desempeño ante unas condiciones ligeramente cambiantes. En conclusión, antes de afirmar que es un algoritmo excelente, como parece apreciarse por los resultados de este estudio, habría que analizar si es capaz de mantener unas métricas tan buenas al producirse cambios en el entorno (*concept drift).*

- Los resultados ofrecidos por SVM no son nada buenos para un clasificador que consume tantos recursos en el entrenamiento. Teniendo esto en cuenta, no sería una opción para un proyecto de desarrollo de AAR ya que su costoso procesado no es compensado con unas buenas métricas. Quizás las configuraciones de preprocesado empleadas no son las adecuadas para entrenar a este algoritmo, que tal vez necesite ventanas que no se superpongan como dan a entender los resultados previamente mostrados.

En la evaluación de la genericidad de los modelos de referencia basados en *Random Forest* se descubre que los datos de entrenamiento que se han empleado para crear el modelo no son lo suficientemente genéricos como para predecir en base a ellos el comportamiento de ovejas. Pese a que sean animales de tamaños similares y con algunas pautas de comportamiento compartidas por ser rumiantes, el modelo entrenado con datos de cabras no generaliza bien. Los resultados en cuanto a precisión no son tan malos como podrían ser. Sin embargo, la *F-score* no deja lugar a duda: existen muchos falsos positivos y falsos negativos, con lo cual hay varias actividades que no se están clasificando adecuadamente. Por esto se concluye que los modelos ideales no son genéricos, si no que están circunscritos al dominio de su entrenamiento. Sería interesante repetir este experimento mezclando los datos de ambos animales como hacen los propios autores del *dataset* [33].

Los modelos de referencia que se han desarrollado como punto de comparación no son viables para ser implementados en dispositivos con ciertas restricciones en cuanto a consumo de energía, capacidad de procesado y memoria. Esto no es tanto porque sean modelos muy complejos o exigentes, sino porque, como se ve en los resultados, una versión mucho más sencilla del mismo clasificador ofrece resultados muy parecidos. Los modelos ideales cuentan con 100, 150 y hasta 500 árboles de decisión, mientras que el modelo que se ha simulado con las características técnicas del LSM6DSOX es perfectamente solvente con solo 8 árboles como máximo. Esto pone de manifiesto que en este campo de investigación se dan ejemplos del Principio de Pareto y que el algoritmo *Random Forest* incluso en sus versiones más sencillas es un clasificador de gran potencial.

6.2 Futuras líneas de investigación

Durante la realización de este trabajo se ha percibido que existen varias posibles líneas de investigación a futuro. Algunas están relacionadas con asuntos que se han quedado en el tintero, pues por la limitación de tiempo, recursos y espacio no han podido investigarse en el presente trabajo. Algunas de ellas son:

- Evaluar el impacto de tamaños de ventana más grandes, algo habitual en la literatura.

- El empleo de otras frecuencias de muestreo, otros tipos de filtros, más características, series temporales diferentes u otras técnicas de reducción de la dimensionalidad, con el fin de evaluar su impacto en los resultados finales de un clasificador.
- El uso de un *dataset* propio, con datos recogidos en campo, con el fin de poder establecer comparaciones con otros *datasets*, evaluar los clasificadores con datos reales y poder proporcionar datos públicos con los que contribuir a la ciencia en abierto.
- El cómputo en software en el módulo LSM6DSOX de la métrica SMA, ya que no es una característica que calcule en su preprocesado. Así se podría verificar, si como se ha concluido en este trabajo, es una característica muy relevante en AAR.

Por otro lado, existen líneas de investigación abiertas por otros investigadores que son tan profundas como prometedoras. Las más cercanas e interesantes a lo tratado aquí versan sobre:

- La computación *on*-edge. El análisis del rendimiento energético de los sistemas de AAR, así como el coste computacional y el uso de memoria de diferentes clasificadores y elementos del preprocesado. La finalidad es encontrar soluciones comerciales con buen rendimiento, bajo consumo y adaptables a diferentes contextos.
- El uso de Deep Learning y redes neuronales para el análisis del comportamiento de individuos y de poblaciones enteras de animales.

Bibliografía

[1] «autómata». Accedido: 26 de agosto de 2024. [En línea]. Disponible en: https://definiciona.com/automata/

[2] «Significado de artesano». Accedido: 27 de agosto de 2024. [En línea]. Disponible en: https://significado.net/artesano/

[3] «artesano», *Wikcionario, el diccionario libre*. 26 de agosto de 2024. Accedido: 27 de agosto de 2024. [En línea]. Disponible en: https://es.wiktionary.org/w/index.php?title=artesano&oldid=5567478

[4] «Historia de la agricultura», *Wikipedia, la enciclopedia libre*. 12 de junio de 2024. Accedido: 27 de agosto de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Historia_de_la_agricultura&oldid=160701677

[5] E. S. Fogarty, D. L. Swain, G. Cronin, y M. Trotter, «Autonomous on-animal sensors in sheep research: A systematic review», *Comput. Electron. Agric.*, vol. 150, pp. 245-256, jul. 2018, doi: 10.1016/j.compag.2018.04.017.

[6] J. K. Petersen, *Understanding Surveillance Technologies: Spy Devices, Their Origins & Applications*. Boca Raton: CRC Press, 2000. doi: 10.1201/9781420038811.

[7] J. Barwick, D. W. Lamb, R. Dobos, M. Welch, y M. Trotter, «Categorising sheep activity using a tri-axial accelerometer», *Comput. Electron. Agric.*, vol. 145, pp. 289-297, feb. 2018, doi: 10.1016/j.compag.2018.01.007.

[8] M. Moreau, S. Siebert, A. Buerkert, y E. Schlecht, «Use of a tri-axial accelerometer for automated recording and classification of goats' grazing behaviour», *Appl. Anim. Behav. Sci.*, vol. 119, n.º 3, pp. 158-170, jul. 2009, doi: 10.1016/j.applanim.2009.04.008.

[9] K. M. McLennan *et al.*, «Technical note: Validation of an automatic recording system to assess behavioural activity level in sheep (*Ovis aries*)», *Small Rumin. Res.*, vol. 127, pp. 92-96, jun. 2015, doi: 10.1016/j.smallrumres.2015.04.002.

[10] «Bienestar animal - Temas especiales», Manual de veterinaria de MSD. Accedido: 3 de septiembre de 2024. [En línea]. Disponible en: https://www.msdvetmanual.com/es/temas-especiales/bienestar-animal/bienestar-animal

[11] «Bienestar de los animales», *Wikipedia, la enciclopedia libre*. 30 de junio de 2024. Accedido: 3 de septiembre de 2024. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Bienestar_de_los_animales&oldid=161040535

[12] P. Martiskainen, M. Järvinen, J.-P. Skön, J. Tiirikainen, M. Kolehmainen, y J. Mononen, «Cow behaviour pattern recognition using a three-dimensional accelerometer and support vector machines», *Appl. Anim. Behav. Sci.*, vol. 119, n.º 1, pp. 32-38, jun. 2009, doi: 10.1016/j.applanim.2009.03.005.

[13] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, y M. Srivastava, «Using mobile phones to determine transportation modes», *ACM Trans Sen Netw*, vol. 6, n.º 2, p. 13:1-13:27, mar. 2010, doi: 10.1145/1689239.1689243.

[14] «Sensors | Free Full-Text | Animals as Mobile Biological Sensors for Forest Fire Detection». Accedido: 28 de agosto de 2024. [En línea]. Disponible en: https://www.mdpi.com/1424-8220/7/12/3084

[15] J. Banzi, «A Sensor Based Anti-Poaching System in Tanzania National Parks», *Int. J. Sci. Technol. Res.*, vol. Volume 4, abr. 2014.

[16] L. Roux y S. Petrus, «A prototype animal borne behaviour monitoring system», Stellenbosch : Stellenbosch University, 2016. Accedido: 5 de marzo de 2024. [En línea]. Disponible en: http://hdl.handle.net/10019.1/98463
[17] «An emerging movement ecology paradigm | PNAS». Accedido: 28 de agosto de 2024. [En línea]. Disponible en: https://www.pnas.org/doi/full/10.1073/pnas.0808918105
[18] S. A. Schoenig *et al.*, «Ambulatory instrumentation suitable for long-term monitoring of cattle health», *Conf. Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. IEEE Eng. Med. Biol. Soc. Annu. Conf.*, vol. 2004, pp. 2379-2382, 2004, doi: 10.1109/IEMBS.2004.1403689.
[19] B. Robert, B. J. White, D. G. Renter, y R. L. Larson, «Evaluation of three-dimensional accelerometers to monitor and classify behavior patterns in cattle», *Comput. Electron. Agric.*, vol. 67, n.º 1, pp. 80-84, jun. 2009, doi: 10.1016/j.compag.2009.03.002.
[20] D. R. Lockyer y R. A. Champion, «Methane production by sheep in relation to temporal changes in grazing behaviour», *Agric. Ecosyst. Environ.*, vol. 86, n.º 3, pp. 237-246, sep. 2001, doi: 10.1016/S0167-8809(00)00289-9.
[21] S. M. Rutter, «13 - Advanced livestock management solutions», en *Advances in Sheep Welfare*, D. M. Ferguson, C. Lee, y A. Fisher, Eds., en Woodhead Publishing Series in Food Science, Technology and Nutrition. , Woodhead Publishing, 2017, pp. 245-261. doi: 10.1016/B978-0-08-100718-1.00013-3.
[22] B. E. Norton, M. Barnes, y R. Teague, «Grazing Management Can Improve Livestock Distribution: Increasing accessible forage and effective grazing capacity», *Rangelands*, vol. 35, n.º 5, pp. 45-51, oct. 2013, doi: 10.2111/RANGELANDS-D-13-00016.1.
[23] D. Smith *et al.*, «Bag of Class Posteriors, a new multivariate time series classifier applied to animal behaviour identification», *Expert Syst. Appl.*, vol. 42, n.º 7, pp. 3774-3784, may 2015, doi: 10.1016/j.eswa.2014.11.033.
[24] J. B. Roelofs, F. J. C. M. van Eerdenburg, N. M. Soede, y B. Kemp, «Pedometer readings for estrous detection and as predictor for time of ovulation in dairy cattle», *Theriogenology*, vol. 64, n.º 8, pp. 1690-1703, nov. 2005, doi: 10.1016/j.theriogenology.2005.04.004.
[25] T. Ishiwata, R. Kilgour, K. Uetake, Y. Eguchi, y T. Tanaka, «Choice of attractive conditions by beef cattle in a Y-maze just after release from restraint», *J. Anim. Sci.*, vol. 85, pp. 1080-5, abr. 2007, doi: 10.2527/jas.2006-405.
[26] R. C. Dobos, D. B. Taylor, M. G. Trotter, B. E. McCorkell, D. A. Schneider, y G. N. Hinch, «Characterising activities of free-ranging Merino ewes before, during and after lambing from GNSS data», *Small Rumin. Res.*, vol. 131, pp. 12-16, oct. 2015, doi: 10.1016/j.smallrumres.2015.06.017.
[27] E. D. Ungar y S. M. Rutter, «Classifying cattle jaw movements: Comparing IGER Behaviour Recorder and acoustic techniques», *Appl. Anim. Behav. Sci.*, vol. 98, n.º 1, pp. 11-27, jun. 2006, doi: 10.1016/j.applanim.2005.08.011.
[28] L. Riaboff *et al.*, «Evaluation of pre-processing methods for the prediction of cattle behaviour from accelerometer data», ago. 2019, doi: 10.1016/j.compag.2019.104961.
[29] L. Riaboff, L. Shalloo, A. F. Smeaton, S. Couvreur, A. Madouasse, y M. T. Keane, «Predicting livestock behaviour using accelerometers: A systematic review of processing techniques for ruminant behaviour prediction from raw accelerometer data», *Comput. Electron. Agric.*, vol. 192, 2022, doi: 10.1016/j.compag.2021.106610.
[30] N. Kleanthous, A. J. Hussain, W. Khan, J. Sneddon, A. Al-Shamma'a, y P. Liatsis, «A survey of machine learning approaches in animal behaviour», *Neurocomputing*, vol. 491, pp. 442-463, jun. 2022, doi: 10.1016/j.neucom.2021.10.126.
[31] A. Mao, E. Huang, X. Wang, y K. Liu, «Deep learning-based animal activity recognition with wearable sensors: Overview, challenges, and future directions», *Comput. Electron. Agric.*, vol. 211, p. 108043, ago. 2023, doi: 10.1016/j.compag.2023.108043.

[32] S. Le Roux, R. Wolhuter, y T. Niesler, *An Overview of Automatic Behaviour Classification for Animal-Borne Sensor Applications in South Africa*. 2017, p. 19. doi: 10.1145/3132711.3132716.
[33] J. Kamminga, H. Bisby, D. Le, N. Meratnia, y P. Havinga, *Generic online animal activity recognition on collar tags*. 2017, p. 606. doi: 10.1145/3123024.3124407.
[34] J. P. Meijers, H. Bisby, N. Meratnia, y P. J. M. Havinga, «Robust Sensor-Orientation-Independent Feature Selection for Animal Activity Recognition on Collar Tags», *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, n.º 1, p. 15:1-15:27, mar. 2018, doi: 10.1145/3191747.
[35] C. Carslake, J. A. Vázquez-Diosdado, y J. Kaler, «Machine Learning Algorithms to Classify and Quantify Multiple Behaviours in Dairy Calves Using a Sensor: Moving beyond Classification in Precision Livestock», *Sensors*, vol. 21, n.º 1, Art. n.º 1, ene. 2021, doi: 10.3390/s21010088.
[36] J. Barwick, D. W. Lamb, R. Dobos, M. Welch, D. Schneider, y M. Trotter, «Identifying Sheep Activity from Tri-Axial Acceleration Signals Using a Moving Window Classification Model», *Remote Sens.*, vol. 12, n.º 4, Art. n.º 4, ene. 2020, doi: 10.3390/rs12040646.
[37] E. S. Fogarty, D. L. Swain, G. M. Cronin, L. E. Moraes, y M. Trotter, «Behaviour classification of extensively grazed sheep using machine learning», *Comput. Electron. Agric.*, vol. 169, p. 105175, feb. 2020, doi: 10.1016/j.compag.2019.105175.
[38] N. Kleanthous, A. Hussain, A. Mason, y J. Sneddon, «Data Science Approaches for the Analysis of Animal Behaviours», en *Intelligent Computing Methodologies*, D.-S. Huang, Z.-K. Huang, y A. Hussain, Eds., en Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 411-422. doi: 10.1007/978-3-030-26766-7_38.
[39] N. Kleanthous, A. Hussain, W. Khan, J. Sneddon, y A. Mason, «Feature Extraction and Random Forest to Identify Sheep Behavior from Accelerometer Data», en *Intelligent Computing Methodologies*, D.-S. Huang y P. Premaratne, Eds., en Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 408-419. doi: 10.1007/978-3-030-60796-8_35.
[40] J. Marais, R. Wolhuter, T. Niesler, y S. Le Roux, *Automatic classification of sheep behaviour using 3-axis accelerometer data*. 2014.
[41] K. Sakai, K. Oishi, M. Miwa, H. Kumagai, y H. Hirooka, «Behavior classification of goats using 9-axis multi sensors: The effect of imbalanced datasets on classification performance», *Comput. Electron. Agric.*, vol. 166, p. 105027, nov. 2019, doi: 10.1016/j.compag.2019.105027.
[42] N. Kleanthous *et al.*, «Machine Learning Techniques for Classification of Livestock Behavior», en *Neural Information Processing*, L. Cheng, A. C. S. Leung, y S. Ozawa, Eds., en Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 304-315. doi: 10.1007/978-3-030-04212-7_26.
[43] M. Cordovilla Serrano, P. P. Sánchez Espeso, y U. de Cantabria, «Técnicas de sensado y análisis del comportamiento del ganado en explotaciones extensivas: Sensing and analysis techniques of livestock behavior in extensive farming», 2022. Accedido: 7 de marzo de 2024. [En línea]. Disponible en: https://go.exlibris.link/VjPDcD7M
[44] D. Figo, P. Diniz, D. Ferreira, y J. Cardoso, «Preprocessing techniques for context recognition from accelerometer data», *Pers. Ubiquitous Comput.*, vol. 14, pp. 645-662, oct. 2010, doi: 10.1007/s00779-010-0293-9.
[45] «Real-World Machine Learning», Manning Publications. Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://www.manning.com/books/real-world-machine-learning

[46] «IsolationForest», scikit-learn. Accedido: 1 de septiembre de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/generated/sklearn.ensemble.IsolationForest.html
[47] I. Guyon y A. Elisseeff, «An Introduction of Variable and Feature Selection», *J Mach. Learn. Res. Spec. Issue Var. Feature Sel.*, vol. 3, pp. 1157-1182, ene. 2003, doi: 10.1162/153244303322753616.
[48] A. Géron, «Hands-On Machine Learning with Scikit-Learn and TensorFlow».
[49] «RandomForestClassifier», scikit-learn. Accedido: 6 de septiembre de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[50] «GaussianNB», scikit-learn. Accedido: 7 de septiembre de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[51] «BernoulliNB», scikit-learn. Accedido: 7 de septiembre de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html
[52] «1.4. Support Vector Machines», scikit-learn. Accedido: 7 de septiembre de 2024. [En línea]. Disponible en: https://scikit-learn/stable/modules/svm.html