Trabajo Fin de Máster
Ingeniería Industrial

Comparación de algoritmos de aprendizaje automático para clasificación de golpes de pádel

Autor: Guillermo Cartes Domínguez
Tutores: Daniel Gutiérrez Reina
Alejandro Tapia Córdoba
Evelia Franco Álvarez

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster
Ingeniería Industrial

Comparación de algoritmos de aprendizaje automático para clasificación de golpes de pádel

Autor:

Guillermo Cartes Domínguez

Tutores:

Daniel Gutiérrez Reina

Profesor Contratado Doctor

Alejandro Tapia Córdoba

Profesor Contratado Doctor

Evelia Franco Álvarez

Profesor Contratado Doctor

Dpto. Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster: Comparación de algoritmos de aprendizaje automático para clasificación de golpes de pádel

Autor: Guillermo Cartes Domínguez

Tutor: Daniel Gutiérrez Reina
Alejandro Tapia Córdoba
Evelia Franco Álvarez

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis amigos

A mis profesores

Agradecimientos

En primer lugar, agradecer a Daniel Gutiérrez Reina haberme dado la oportunidad de realizar un proyecto tan interesante junto con un gran equipo que me ha permitido adentrarme en el mundo de la inteligencia artificial de una forma excitante y práctica. Así pues, agradecer a Evelia Franco Álvarez y Alejandro Tapia Córdoba haber querido ser partícipes de mi trabajo final de máster.

Por otro lado, la creación de la base de datos ha sido posible gracias a la colaboración desinteresada de numerosos jugadores de pádel, a los que agradezco su entusiasmo y entrega en cada golpe realizado. Gracias a Guillermo Castilla Gómez, Juan Manuel Garrido Bogado, Alejandro Moral Lozano, Carlotta Casali Vannicelli, Antonio Alfonso Morales Santiago, Marta Morales Rodríguez, Gonzalo Guadalix Arribas, Luka Payras Mirkine, Guillermo Avilés López, Cristina Blanco Pérez, Lucía Morales Rodríguez y Pablo Megino Cañaveras.

Por último, este trabajo concluye seis intensos años en la Escuela Técnica Superior de Ingeniería de Sevilla. Gracias a mis compañeros, por hacer más amenas las horas en la escuela y por haberse convertido muchos de ellos en amigos. Gracias infinitas a mis padres, Rafael y Mari Ángeles, por hacer de mí la persona que soy hoy; y gracias a mi hermano Rafa y a Marta, por ser apoyos incondicionales.

Guillermo Cartes Domínguez

Sevilla, 2021

Resumen

El procesamiento de datos en la práctica deportiva es un hecho cada vez más presente a todos los niveles, desde profesionales en búsqueda de herramientas que permitan mejorar su rendimiento, hasta principiantes motivados por la cuantificación de su actividad física.

En este trabajo se lleva a cabo una comparación entre algunos de los principales algoritmos de aprendizaje automático con el fin de clasificar golpes de pádel. Estos son: redes neuronales, redes neuronales convolucionales 1D, árbol de decisión, K vecinos más próximos y máquinas de vector soporte.

Antes de llevar a cabo una clasificación de golpes de pádel se necesita una base de datos suficientemente representativa que recoja numerosos ejemplos de la realización de estos golpes. Dado que no existe en la literatura un conjunto de datos similar, se procede a la creación de este, para lo que se desarrolla un sistema de recogida de datos basado en un dispositivo electrónico que cuenta con una unidad de medida inercial (IMU).

Por último, se estudia el comportamiento de cada algoritmo con el conjunto de datos creado.

Abstract

The presence of data processing in sport is growing at all levels, from professionals looking for new tools that allow them to improve their performance to beginners who want to quantify their physical activity.

A comparison between some of the main machine learning algorithms that aim to classify padel shots is carried out. These algorithms are neural networks, convolutional neural networks 1D, decision tree, K nearest neighbors, and support vector machine.

Before classification takes place, a sufficiently representative dataset containing numerous examples of the movement of padel strokes is needed. Due to the lack of a similar dataset in literature, this dataset is created, for which a data collection system based on an electronic device containing an inertial measurement unit (IMU) is developed.

Finally, the behaviour of all algorithms with the created dataset is analysed.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xvii**

Índice de Figuras **xix**

Notación **xxiii**

1 Introducción **1**
1.1 Descripción del problema *2*
1.2 Objetivos *2*

2 Estado del arte **3**
2.1 Aprendizaje automático aplicado a la clasificación en deportes de raqueta *3*
2.2 Estudio de mercado de dispositivos electrónicos aplicados en la recolección de datos *5*

3 Clasificación de golpes y sistema desarrollado **7**
3.1 Problema de clasificación de golpes *8*
3.2 Dispositivo para toma de muestras *8*
3.2.1 Raspberry Pi 8
3.2.2 IMU 8
3.2.3 Plataforma 9
3.2.4 Batería externa 9
3.2.5 VNC Viewer 9
3.2.6 Python 10
3.3 Sistema desarrollado *10*

4 Conjunto de datos **11**
4.1 Caracterización del conjunto de datos. *11*
4.2 Proceso de toma de datos *14*
4.2.1 El deportista 14
4.2.2 Responsable de lanzar las bolas 14
4.2.3 Responsable del registro de datos 15
4.3 Detector de golpes *15*
4.3.1 Condición de detección 16
4.3.2 Duración del golpe 17
4.3.3 Resultados 17
4.4 Creación del conjunto de datos *20*

5 Metodología: Algoritmos de aprendizaje automático **23**
5.1 Introducción a la clasificación. Conceptos previos *23*
5.1.1 Modelo matemático 23
5.1.2 Conjunto de datos 23
5.1.3 Aprendizaje supervisado y no supervisado 23
5.1.4 Parámetros e hiperparámetros 24

5.1.5 Overfitting 24
5.1.6 Regularización 24
5.1.7 Librerías para aprendizaje automático 24
5.2 Redes neuronales densamente conectadas *24*
5.2.1 Perceptrón 25
5.2.2 Proceso de aprendizaje 27
5.2.3 Hiperparámetros 29
5.2.4 Overfitting y regularización 30
5.3 Redes neuronales convolucionales 1D *30*
5.3.1 Hiperparámetros 32
5.4 Árbol de decisión *32*
5.4.1 Estructura 32
5.4.2 Función de impureza 33
5.4.3 Hiperparámetros 34
5.5 K Vecinos más próximos (KNN) *34*
5.6 Máquinas de vector soporte (SVM) *35*
5.6.1 Problema multiclase 36
5.6.2 Hiperparámetros 36

6 Resultados **37**
6.1 Redes neuronales densamente conectadas *39*
6.1.1 Red neuronal densa con serie temporal 39
6.1.2 Red neuronal densa con feature engineering 43
6.1.3 Resumen de resultados de redes neuronales densamente conectadas 46
6.2 Redes neuronales convolucionales 1D *47*
6.2.1 Clasificador con una capa convolucional y una capa densa 47
6.2.2 Clasificador con una capa convolucional y dos capas densas 49
6.2.3 Clasificador con dos capas convolucionales y una capa densa 51
6.2.4 Clasificador con dos capas convolucionales y dos capas densas 53
6.2.5 Resumen de resultados de redes neuronales convolucionales 1D 55
6.3 Árbol de decisión *56*
6.3.1 Árbol de decisión con serie temporal 56
6.3.2 Árbol de decisión con feature engineering 57
6.3.3 Resumen de resultados de árbol de decisión 58
6.4 K Vecinos más próximos (KNN) *59*
6.4.1 KNN con serie temporal 59
6.4.2 KNN con feature engineering 61
6.4.3 Resumen de resultados KNN 63
6.5 Máquinas de vector soporte (SVM) *63*
6.5.1 SVM con serie temporal 63
6.5.2 SVM con feature engineering 66
6.5.3 Resumen de resultados SVM 68
6.6 Resumen final de los resultados *69*

7 Conclusiones y trabajos futuros **71**
7.1 Conclusiones *71*
7.2 Trabajos futuros *71*

Referencias **75**

ÍNDICE DE TABLAS

Tabla 1. Estudio de mercado de IMUs 5
Tabla 2. Deportistas del conjunto de datos. 13
Tabla 3. Matriz de confusión del detector de golpes. 17
Tabla 4. Conjunto de datos final 22
Tabla 5. Configuraciones red neuronal densa. Serie temporal. Clasificador con dos capas. 40
Tabla 6. Configuraciones red densa. Serie temporal. Clasificador con tres capas. 42
Tabla 7. Configuraciones red neuronal densa. Feature engineering. Clasificador con dos capas. 43
Tabla 8. Configuraciones red neuronal densa. Feature engineering. Clasificador con tres capas. 45
Tabla 9. Configuraciones CNN1D. Clasificador con una capa convolucional y una capa densa. 47
Tabla 10. Configuraciones CNN1D. Clasificador con una capa convolucional y dos capas densas. 49
Tabla 11. Configuraciones CNN1D. Clasificador con dos capas convolucionales y una capa densa. 51
Tabla 12. Configuraciones CNN1D. Clasificador con dos capas convolucionales y dos capas densas. 53
Tabla 13. Configuraciones árbol de decisión con serie temporal. 56
Tabla 14. Configuraciones árbol de decisión con feature engineering. 57
Tabla 15. Algoritmo KNN con serie temporal. Accuracy en función de *k*. 59
Tabla 16. Algoritmo KNN con feature engineering. Accuracy en función de *k*. 61
Tabla 17. Algoritmo SVM con serie temporal. Accuracy (%) en función de kernel y C. 64
Tabla 18. Algoritmo SVM con feature engineering. Accuracy (%) en función de kernel y C. 66
Tabla 19. Resumen de resultados. 69

ÍNDICE DE FIGURAS

Figura 1. Pista de pádel. 7
Figura 2. Palas y pelota de pádel. 8
Figura 3. Ejes del Sense Hat. 9
Figura 4. Raspberry Pi + Sense Hat + Plataforma. 9
Figura 5. Sistema de toma de datos. 10
Figura 6. Golpes en la base de datos por tipo de golpe. 14
Figura 7. Deportista durante una sesión de toma de datos. 15
Figura 8. Diagrama de flujo del detector de golpes 16
Figura 9. Detección de golpes durante una prueba. 18
Figura 10. Ejemplo de golpe de derecha. 19
Figura 11. Ejemplo de golpe de revés. 19
Figura 12. Comparación de un golpe de derecha con un golpe de revés. 20
Figura 13. Diagrama de flujo del creador del dataset. 21
Figura 14. Inteligencia Artificial, Machine Learning y Deep Learning. 25
Figura 15. Diagrama de un perceptrón con 5 señales de entrada. 25
Figura 16. División de clases. Perceptrón. 26
Figura 17. Función de activación en escalón (roja) vs sigmoid (azul). 27
Figura 18. Red neuronal densamente conectada. Ejemplo con dos capas. 27
Figura 19. Funciones de activación a) Linear, b) Sigmoid, c) Tanh, d) ReLU 28
Figura 20. Forwardpropagation, backwardpropagation y loss. Disponible en [26]. 29
Figura 21. Aplicación de dropout a una red reuronal. Disponible en [28]. 30
Figura 22. Operación de convolución. 31
Figura 23. Ejemplo de kernel para golpes de pádel. 31
Figura 24. Árbol de decisión de 7 clases. 33
Figura 25. Ejemplo algoritmo KNN. 35
Figura 26. Ejemplo algoritmo SVM. 36
Figura 27. Serie temporal vs feature engineering. 38
Figura 28. Red neuronal densa con serie temporal. Clasificador con dos capas. Accuracy para mejores hiperparámetros. 40
Figura 29. Red neuronal densa con serie temporal. Clasificador con dos capas. Matriz de confusión. 41
Figura 30. Red densa con serie temporal. Clasificador con tres capas. Accuracy para mejores hiperparámetros. 42
Figura 31. Red neuronal densa con serie temporal. Clasificador con 3 capas. Matriz de confusión. 43
Figura 32. Red neuronal densa con feature engineering. Clasificador con dos capas. Accuracy para mejores hiperparámetros. 44
Figura 33. Red neuronal densa con feature engineering. Clasificador con dos capas. Matriz de confusión.

44
Figura 34. Red neuronal densa con feature engineering. Clasificador con tres capas. Accuracy para mejores hiperparámetros. 45
Figura 35. Red neuronal densa con feature engineering. Clasificador con tres capas. Matriz de confusión. 46
Figura 36. Resumen de la accuracy para los clasificadores de red neuronal densa. 47
Figura 37. CNN1D. Clasificador con una capa convolucional y una capa densa. Accuracy para mejores hiperparámetros. 48
Figura 38. CNN1D. Matriz de confusión para una capa convolucional y una capa densa. 49
Figura 39. CNN1D. Clasificador con una capa convolucional y dos capas densas. Accuracy para mejores hiperparámetros. 50
Figura 40. CNN1D. Matriz de confusión para una capa convolucional y dos capas densas. 51
Figura 41. CNN1D. Clasificador con dos capas convolucionales y una capa densa. Accuracy para mejores hiperparámetros. 52
Figura 42. CNN1D. Matriz de confusión para dos capas convolucionales y una capa densa. 53
Figura 43. CNN1D. Clasificador con dos capas convolucionales y dos capas densas. Accuracy para mejores hiperparámetros. 54
Figura 44. CNN1D. Matriz de confusión para dos capas convolucionales y dos capas densas. 55
Figura 45. Resumen de la accuracy para los clasificadores CNN1D. 55
Figura 46. Árbol de decisión con serie temporal. Accuracy para mejores hiperparámetros. 56
Figura 47. Árbol de decisión con serie temporal. Matriz de confusión. 57
Figura 48. Árbol de decisión con feature engineering. Accuracy para mejores hiperparámetros. 58
Figura 49. Árbol de decisión con feature engineering. Matriz de confusión. 58
Figura 50. Resumen de la accuracy para los clasificadores de árbol de decisión. 59
Figura 51. Algoritmo KNN con serie temporal. Diagrama de bigotes en función de *k*. 60
Figura 52. Algoritmo KNN con serie temporal. Matriz de confusión para $k = 1$. 60
Figura 53. Algoritmo KNN con serie temporal. Accuracy para $k = 1$. 61
Figura 54. Algoritmo KNN con feature engineering. Diagrama de bigotes en función de k. 62
Figura 55. Algoritmo KNN con feature engineering. Matriz de confusión para $k = 1$. 62
Figura 56. Algoritmo KNN con feature engineering. Accuracy para $k = 1$. 63
Figura 57. Algoritmo KNN. Resultados de la serie temporal vs feature engineering para $k = 1$. 63
Figura 58. Algoritmo SVM con serie temporal. Diagrama de bigotes en función de kernel y C. 64
Figura 59. Algoritmo SVM con serie temporal. Matriz de confusión para kernel radial con $C = 10$. 65
Figura 60. Algoritmo SVM con serie temporal. Accuracy para kernel radial y $C = 10$. 65
Figura 61. Algoritmo SVM con feature engineering. Diagrama de bigotes en función de kernel y C. 66
Figura 62. Algoritmo SVM con feature engineering. Matriz de confusión para kernel radial con $C = 100$. 67
Figura 63. Algoritmo SVM con feature engineering. Accuracy para kernel radial y $C = 100$. 67
Figura 64. Algoritmo SVM. Resumen de los resultados. 68
Figura 65. Algoritmo SVM. Resultados de la serie temporal vs feature engineering para mejor hiperparametrización. 68

Figura 66. Accuracy para la mejor hiperparametrización de cada algoritmo. 69
Figura 67. Ejemplo de la técnica window warping para el aumento de datos. Disponible en [40]. 72
Figura 68. Nuevo diseño del dispositivo de toma de datos. 73
Figura 69. Encapsulamiento del sensor BNO055. 73
Figura 70. Sistema para una predicción en tiempo real. 74

Notación

MEMS	Microelectromechanical Systems
IoT	Internet of Things
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
IMU	Inertial Measurement Unit
DOF	Degrees of Freedom
CNN	Convolutional Neural Network
KNN	K Nearest Neighbors
SVM	Support Vector Machines
LSTM	Long Short-Term Memory
ODR	Output Data Rate
BDO	Bit Data Output
HAR	Human Activity Recognition
BLE	Bluetooth Low Energy
CSV	Comma Separated Values

1 INTRODUCCIÓN

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

- Albert Einstein -

En los últimos años, el uso de sistemas microelectromecánicos (MEMS) ha vivido una gran expansión en multitud de sectores, a pesar de que esta tecnología se venía desarrollando desde varias décadas atrás. Su origen tiene lugar con la revolución del silicio, gracias a dos importantes invenciones en 1959: el chip de circuito integrado y el Metal Oxide Semiconductor Field Effect Transistor (MOSFET), que condujo a la miniaturización de la electrónica, tal y como predijo la ley de Moore [1]. La fabricación masiva de estos sistemas debido a la popularidad de su uso en multitud de dispositivos inteligentes ha provocado una bajada drástica de su precio, facilitando así su expansión en todos los ámbitos [2].

Un uso muy conocido de estos sistemas MEMS son los wearables, dispositivos de tamaño reducido que se colocan en el cuerpo humano e interactúan con él y otros dispositivos para una finalidad concreta. Estos dispositivos, que pueden ser pulseras, relojes o anillos, entre otros, recogen información del cuerpo que puede ser procesada por ellos mismos o por otro dispositivo al que estén conectados de forma remota. Esta interconexión digital de objetos a través de la red es lo que se conoce como el internet de las cosas (IoT, por sus siglas en inglés), donde cada objeto (no necesariamente wearables) puede mandar y recibir información sin necesidad de intervención humana. Los wearables que han conseguido más popularidad son aquellos que cuantifican la actividad física y realizan un seguimiento biométrico. Un ejemplo son las pulseras y relojes inteligentes, los cuales permiten realizar un seguimiento continuo de los movimientos y actividades que realiza la persona que lo utiliza. Es común ver que los usuarios de esta tecnología cuantifican diariamente los pasos realizados, la distancia recorrida o las horas de sueño. Para poner en magnitud la popularidad de estos dispositivos, según un estudio de mercado del CCS Insight [3], el pasado año 2020 se vendieron 193 millones de dispositivos en todo el mundo, con un valor de mercado de 24.000 millones de dólares. Además, para 2021 se estima un crecimiento del 24%, alcanzando los 239 millones de unidades de venta [3]. Estas pulseras de actividad también se usan para seguimiento biométrico; ya que pueden realizar, entre otras cosas, electrocardiogramas o mediciones de cantidad de oxígeno en sangre, habiéndose popularizado este último con la llegada del coronavirus SARS-CoV-2. También son útiles para el seguimiento de personas dependientes, donde la rápida detección de caídas o accidentes puede jugar un papel fundamental en una situación de emergencia.

Ahora bien, ¿para qué usar estos dispositivos en la práctica deportiva a todos los niveles? La respuesta es sencilla: dan acceso a millones de datos. Los datos son la base de la información y con la información se construye el conocimiento [4]. Hoy en día existe la posibilidad de procesar cantidades enormes de datos; un fenómeno que está transformando los deportes profesionales en prácticamente todos los niveles, desde los aficionados en sus casas hasta los deportistas en el terreno de juego, pasando por cuerpos técnicos, médicos y clubes. Esta tecnología permite realizar diversas acciones como comparar el rendimiento de los jugadores o recopilar las estadísticas de los partidos [5]. Cualquier dato generado en el terreno de juego es susceptible de ser analizado y estudiado. Por ejemplo, en un partido de tenis de una hora y media de duración se generan unos 70.000 registros, y durante un partido de fútbol se pueden llegar a capturar alrededor de ocho millones de datos [6]. Resulta evidente que esta ingente cantidad de información solo puede ser analizada mediante las nuevas tecnologías. Si

bien el pionero en usar el Big Data fue el béisbol en los años 70, donde se empezó a usar los registros históricos de los jugadores de béisbol de las grandes ligas americanas [6], son variadas y numerosas las aplicaciones que han surgido posteriormente adaptadas a otros contextos. En el caso del baloncesto, por ejemplo, los datos han demostrado en la NBA que son más eficientes los jugadores más bajos y versátiles que los más altos que defiendan el aro, ya que es más productivo anotar triples que canastas de dos puntos, aún suponiendo una peor defensa en el propio aro [6]. En el fútbol, el Big Data aporta un gran valor a todos los clubes del mundo, ya que usan esta tecnología para mejorar sus estrategias de juego, fichar a nuevos jugadores, etc. Por otro lado, el tratamiento de datos también puede ayudar a evitar lesiones. Un estudio realizado por Global Sports Salaries Survey 2015 calcula que los clubes pierden entre el 10 y el 30 por ciento de sus plantillas por lesiones. Con el uso de la tecnología se puede identificar cualquier deficiencia en el estado físico del atleta y prevenir así la lesión [5].

En el caso de un deporte en pleno auge mundial como es el pádel, es previsible que el análisis de datos juegue un papel importante. Las investigaciones realizadas en pádel se han centrado -como se verá más adelante- en el estudio de distancias recorridas, velocidad, tipo de desplazamiento de los jugadores, zonas de juego o golpes ganadores. El golpeo de la bola se puede considerar el elemento fundamental de este deporte, por lo que el acceso a su información adquiere una especial relevancia. ¿Qué golpes se realizan durante un entrenamiento o un partido?, ¿en qué cantidad y proporción?, ¿es un patrón que se repite en cada jugador?, ¿qué golpes se dan en las victorias?, ¿qué golpes se dan en las derrotas?, ¿cómo se puede mejorar la técnica del golpe? Las respuestas a estas preguntas tienen una gran importancia, puesto que pueden ayudar a mejorar el rendimiento de los deportistas y contribuir así a su éxito deportivo. Así, el presente estudio pretende desarrollar un sistema, mediante aprendizaje automático, que sea la base para dar respuesta a algunas de las preguntas anteriores.

1.1 Descripción del problema

Las pulseras de actividad y relojes inteligentes son capaces de monitorizar un gran número de actividades deportivas: caminar, correr, nadar o montar en bici, entre otros; pero no permiten tener un seguimiento exhaustivo dentro de cada deporte. Es posible, por ejemplo, cuantificar el tiempo que el usuario juega a pádel, los pasos realizados durante la actividad o la distancia recorrida, pero no ofrecen información sobre qué golpes se ha realizado durante el entrenamiento o la competición.

La tarea de clasificación de un conjunto de datos necesita, en primer lugar, una base de datos suficientemente representativa que recoja una gran cantidad de ejemplos de los datos que se quieran clasificar. Por otro lado, se necesita un algoritmo capaz de crear un modelo matemático que clasifique estos datos.

Si bien existen diferentes bases de datos de libre acceso que recogen gran variedad de movimientos humanos cotidianos, tales como caminar, saltar o subir y bajar escaleras, no existe una base de datos accesible que recoja las características dinámicas de los diferentes golpes que se pueden realizar jugando a pádel.

1.2 Objetivos

El objetivo principal del presente estudio es la comparación de modelos de aprendizaje automático para la clasificación de golpes de pádel. No obstante, existen una serie de objetivos previos necesarios para hacer posible el objetivo final. Los objetivos se pueden desglosar de la siguiente forma:

- Creación de un prototipo de sistema electrónico que permita recolectar datos de los golpes de pádel de una forma sencilla.
- Creación de una base de datos de golpes de pádel para el entrenamiento, validación y testeo de los clasificadores que se desarrollen.
- Comparación y evaluación de los distintos algoritmos de aprendizaje automático para determinar el más preciso en la clasificación de golpes de pádel.

2 ESTADO DEL ARTE

2.1 Aprendizaje automático aplicado a la clasificación en deportes de raqueta

El reconocimiento de actividades humanas (HAR, de sus siglas en inglés) se ha expandido en los últimos años gracias al desarrollo de técnicas de aprendizaje automático y al uso de dispositivos con sensores integrados que permiten la captura de información durante el desarrollo normal de las actividades.

La clasificación de golpes de pádel puede ser considerada como una clasificación de movimientos de mano, ya que en cada golpe el deportista realiza un movimiento de mano significativamente distinto. Hay numerosos sensores disponibles como acelerómetros, giroscopios, magnetómetros, electrocardiogramas, barómetros, sensores de temperatura, humedad, etc. En ocasiones, algunos autores usan cámaras para el reconocimiento de actividades, como es el caso del estudio [7]. Sin embargo, para la clasificación en deportes de raqueta lo más común es utilizar conjuntamente acelerómetro y giroscopio, que miden aceleraciones lineales y velocidades angulares, respectivamente.

En [8], Tabrizi, Pashazadeh y Jabani presentaron un estudio comparativo de golpes de tenis de mesa usando Deep Learning. En cuanto a la recogida de muestras, usaron un sensor BNO055 de 9 grados de libertad (DOF, por sus siglas en inglés) montado sobre la misma pala, con una frecuencia de muestreo de 50 Hz y 70 Hz muestras por golpe. Recogieron datos de 16 participantes divididos en dos grupos: principiantes y profesionales. Del grupo de profesionales se tomaron 1080 golpes, mientras que del grupo de principiantes 648, sin hacer distinciones de género, edad o altura. De los 1728 golpes tomados, 158 fueron excluidos por estar incompletos, por lo que la base de datos recogía 1570 golpes. De los 1570 golpes, 740 eran de tipo básico, 421 de topsping[1] y 409 de push stroke[2]. El 75% de los golpes fueron seleccionados para el conjunto de entrenamiento, el 20% para validación, y el 5% restante para test, siendo la selección aleatoria. Se llevó a cabo una evaluación de tres modelos de machine learning: LSTM, CNN 2D y SVM, donde concluyeron que el primero de ellos tenía mayor precisión y robustez que los otros dos.

Benages, Buldain y Orrite [9] realizaron un estudio sobre detección de actividades de tenis. En este estudio se usó un SensorTag CC2650STK de Texas Instruments con tecnología Bluetooth Low Energy, usando una tasa de muestreo de 20 Hz. Usaron dos sensores, uno colocado en la muñeca y otro en la cintura del deportista, lo que hace un total de 12 DOF (6+6). A diferencia del estudio [8], en [9] se realiza una clasificación de actividades, no solo de golpes, por lo que la recolección de datos se realiza con ventanas de 20 muestras con una superposición del 90%. Llevaron a cabo tres clasificadores distintos: uno de identificación de golpes de tenis o actividad normal, otro de clasificación de actividades y el último sobre clasificación de golpes. Se tuvieron en cuenta 7 actividades comunes dentro de la pista: andar, correr, saltar, agacharse, levantarse, estar parado, estar sentado, levantarse y sentarse. En cuanto a los golpes, se tuvieron en cuenta cuatro golpes diferentes: derecha, revés, globo y volea. Se recogieron datos de 8 personas, 4 hombres y 4 mujeres, de los cuales 7 eran diestros y una mujer zurda. Los resultados de la clasificación de golpes fueron un 99.25% de acierto para golpes de personas que habían sido vistas por el modelo y un 96.51% de acierto para golpes de personas que no habían sido previamente vistas por el modelo.

En [10], Kos, Ženko, Vlaj y Kramberger presentaron un estudio sobre detección y clasificación de golpes de tenis. Este se llevó a cabo con un sensor colocado en la muñeca usando los 6 DOF, con una tasa de muestreo de 1000 Hz. La recolección de datos se realizó con 3 personas de diferentes niveles de juego, donde se recogieron un total de 147 golpes de tres tipos: servicio, derecha y revés. Para la detección de golpes, usaron un umbral en la derivada del giroscopio: cuando se supera dicho umbral, establecido experimentalmente, se consideraba que

[1] Un golpe de Topsping es aquel en el que la pelota gira hacia delante mientras se mueve.
[2] En este caso la pelota gira hacia detrás mientras se mueve.

se había detectado un golpe. En cuanto a la clasificación, usaron un algoritmo simple observando previamente las diferencias entre los tres golpes estudiados, de forma que la clasificación de cada golpe se reduce al cumplimiento de dos condiciones en los ejes del giroscopio: se identifica el eje que ha alcanzado el máximo y el eje que ha alcanzado el mínimo. Si dichos ejes coinciden con los observados en cada tipo de golpe, el golpe es clasificado como uno de ese tipo. Se obtiene así un acierto del 98.1%.

Whiteside, Cant, Connolly y Reid realizaron una clasificación de golpes de tenis mediante Machine Learning [11]. Usaron datos de 19 deportistas (8 mujeres y 11 hombres) para clasificar 9 golpes distintos, recogiendo 28.582 golpes, con una IMU colocada en la muñeca del deportista y una tasa de muestreo de 500 Hz. Estudiaron el comportamiento de seis modelos de machine learning, consiguiendo un acierto del 93.2%.

Otro ejemplo de clasificación de golpes de tenis se puede encontrar en [12], realizada por Ebner y Findling. Usaron dos sensores XSens MTw-38A70G20, uno colocado en la muñeca y otro en la raqueta, con un rango de ± 1200 dps en el giroscopio y ± 16 g para el acelerómetro; una tasa de muestreo de 100 Hz y una duración del golpe de un segundo. Para llevar a cabo la detección de golpes, se usa un umbral en la derivada del acelerómetro, de $\pm 15 m/s^3$, consiguiendo detectar el 98% de los golpes, pero con algunos falsos positivos. A diferencia del estudio realizado en [10], en este caso se busca el umbral en la derivada del acelerómetro y no en la del giroscopio. En este caso se clasifican 8 golpes diferentes, con un total de 2.000 golpes recogidos, con una precisión del 98.9 %.

En [13] se desarrolla un sistema de aprendizaje para mejorar las habilidades de los jugadores de bádminton principiantes. Dicho sistema proporciona las similitudes entre los diferentes golpes realizados, comparando el movimiento y la fuerza de cada golpe. Se utiliza una Myo armband, una especie de brazalete inteligente que se coloca en el antebrazo. Este estudio concluyó que aquellos alumnos que aprendían mediante este sistema conseguían mejorar significativamente más sus golpes que aquellos que elegían un sistema tradicional, especialmente en los golpes de remate y revés.

En cuanto a pádel, se pueden destacar los siguientes estudios:

Un análisis de la distancia recorrida en pádel en función del nivel de juego y el número de puntos por partido se lleva a cabo en [14]. Para su desarrollo, se utilizaron dos cámaras de vídeo cenitales colocadas en la parte superior del techo de la pista, a 9 metros del suelo, cada una enfocando a una parte de la pista. Se recogieron muestras de 108 jugadores federados, en 27 partidos realizando un total de 4.406 puntos. Los resultados mostraron que un jugador recorre de media unos 11 metros por punto y 2.900 metros por partido, dividiéndose un 51% en fase activa (tiempo de juego) y un 49% en fase pasiva (tiempo de descanso). Los jugadores se clasificaron en 3 niveles, y se concluyó que los jugadores de nivel medio recorren casi 400 metros más en la fase activa que los jugadores de nivel alto y casi 900 metros más que los jugadores de nivel bajo.

En [15] se presenta un análisis de los golpes finales del punto en pádel mediante un árbol de decisión. Se recogen 2.110 acciones de juego y se analizan: golpe, zona de la pista, eficacia, dirección, resultado y lado de juego. En este caso, se utilizó una cámara de video colocada a 1,5 metros de altura y 3 metros detrás de la pista. Se recogieron un total de 1.055 puntos de 9 partidos de pádel de primer nivel, con 36 jugadores diferentes. Los resultados mostraron que mantener posiciones cercanas a la red aumenta las probabilidades de victoria, siendo las secuencias de finalización más frecuentes las de fondo-volea y globo-remate. Además, utilizar trayectorias cruzadas en el penúltimo golpe aumenta las posibilidades de un error posterior de los rivales.

En [16] se lleva a cabo un análisis de los indicadores de rendimiento del juego de pádel, así como su influencia en los resultados de los partidos. Se recopilaron datos del World Padel Tour entre 2015 y 2019, 1.070 sets de 532 partidos, registrándose variables como el sexo, ronda, resultados de juego, efectividad del golpe y break points. Los resultados mostraron diferencias significativas entre ganadores y perdedores respecto al sexo en los break points.

En [17] se realiza una estimación de la posición del jugador basándose en imágenes, mediante el uso de redes neuronales convolucionales. Los resultados obtienen un 98% de acierto en la posición con una tolerancia de 30 cm para jugadores en la mitad inferior de la pista.

Por otro lado, en el mercado existe una aplicación llamada Babolat Play que ofrece un seguimiento de la práctica deportiva de tenis. Para ello, es necesario usar su propia raqueta, la cual integra la tecnología necesaria para el uso de la aplicación. En la aplicación se recoge una interfaz que, entre otras cosas, clasifica los golpes realizados

en cuatro tipos de golpes: derecha, revés, servicio y remate, además de dar información sobre los efectos de la bola. La raqueta es una raqueta estándar que integra un acelerómetro y un giroscopio, con una autonomía de 6 horas de juego, 150 horas de memoria interna y 300 gramos de peso [18].

2.2 Estudio de mercado de dispositivos electrónicos aplicados en la recolección de datos

En cuanto a la recogida de datos, en el mercado hay una gran variedad de sensores de diferentes prestaciones que podrían ser de utilidad en este estudio. Anteriormente, se ha visto que en [8] se usa el sensor BNO055 de Adafruit con una frecuencia de muestreo de 50Hz; en [9] y [19] el sensor CC2650STK de Texas Instruments, con tasas de muestreo de 20 y 10 Hz, respectivamente; y en [12] el sensor XSens MTw-38A70G20 a 100Hz.

En la Tabla 1, se lleva a cabo un estudio de mercado con diferentes sensores donde se recogen sus principales características.

Tabla 1. Estudio de mercado de IMUs

Dispositivo		**MPU9250 (SensorTag CC2650STK)**	**LSM9DS1 (Sense Hat)**	**LSM9DS1 (SparkFun)**	**Adafruit BNO055**	**LSM6DSL (Berry IMU v3)**	**GY-85**	**MPU6050 (GY-521)**	**MPU9250 (Waveshare 10 DOF IMU)**
Estudios anteriores		[9]			[8]				
Sensores adicionales a acelerómetro y giroscopio		Magnetómetro	Sense Hat	Magnetómetro	Ángulos de Euler	Magnetómetro, Temperatura, Presión, Altitud	Magnetómetro	Magnetómetro	Magnetómetro y Barómetro
ODR	**Accel**	Hasta 4KHz	Hasta 952Hz	Hasta 952Hz	100Hz (Fusion data)	Hasta 6,66kHz	Hasta 3,2kHz	Hasta 1kHz	Hasta 4kHz
	Gyros	Hasta 8KHz	Hasta 952Hz	Hasta 952Hz	100Hz (Fusion data)	Hasta 6,66kHz	Hasta 8kHz	Hasta 8kHz	Hasta 8kHz
Rango	**Accel**	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g	±2/±4/±8/ ±16 g
	Gyros	±250/±500/ ±1000/±2000 dps	±245/±500/ ±2000 dps	±245/±500/ ±2000 dps	±125dps to ±2000dps	±125/±250/ ±500/±1000/ ±2000 dps	Hasta 2000 dps	±250/±500/ ±1000/±2000 dps	±250/±500/ ±1000/±2000 dps
BDO	**Accel**	16 bit	16 bit	16 bit	14 bit	16 bit	13 bit	16 bit	16 bit
	Gyros	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit	16 bit
Precio		Descatalogado	32,25 €	$ 15,95	$ 34,95	$ 26	$ 15,69	7 €	$ 15,99

Se pueden observar numerosas similitudes entre los dispositivos. Con respecto a los sensores que incorporan, es importante destacar que para la clasificación de movimientos solo son necesarios los datos de acelerómetro y giroscopio, por lo que los sensores adicionales no serán un factor decisivo a la hora de elegir el dispositivo. Donde se encuentran más diferencias es en la tasa de muestreo. Teniendo en cuenta que en los estudios de clasificación de golpes vistos anteriormente la menor tasa de muestreo tenía lugar en [9], siendo de 20 Hz y con la que se obtenía hasta un 99,25% de acierto, se puede comprobar que todos los dispositivos serán válidos en este apartado. En cuanto a rango y DBO, apenas se encuentran diferencias entre dispositivos. En cuanto a precio, todos se encuentran relativamente cerca, estando el MPU9250 ya descatalogado.

No obstante, el factor decisivo para decantarse por un dispositivo u otro ha sido la disponibilidad inmediata de numerosas unidades del Sense Hat en el Departamento de Ingeniería Electrónica. Este dispositivo es muy

utilizado para enseñanza, ya que cuenta con numerosos sensores a los que se suma la facilidad de uso con la Raspberry Pi. Por tanto, el dispositivo elegido ha sido la IMU LSM9DS1, incorporada en el Sense Hat.

3 CLASIFICACIÓN DE GOLPES Y SISTEMA DESARROLLADO

"Si le das pescado a un hombre hambriento, lo nutres una jornada.
Si le enseñas a pescar, lo nutrirás toda la vida."

- Lao-tsé -

En esta sección se desarrolla un sistema que permite recolectar datos de golpes de pádel de jugadores reales para su posterior clasificación. Pero antes de ver con detalle cómo es el proceso de recogida de datos, es importante conocer en líneas generales en qué consiste este deporte. El pádel es un deporte de raqueta o pala que se juega por parejas en una pista como la que se muestra en la Figura 1, y que consiste en hacer botar la bola en el campo contrario, con un sistema de puntos similar al tenis, pero con la posibilidad de rebotar en las paredes. La pista tiene 20 metros de largo por 10 metros de ancho, cerrada totalmente con fondos de 4 metros de alto (los 3 primeros deben permitir el rebote de la bola) y con laterales de igual altura en el fondo, descendiendo de forma escalonada hacia el centro de la pista. Los 4 metros del lateral más cercanos al fondo también deben permitir el rebote. La pista se divide por la mitad con una red de aproximadamente 92 cm de altura. A ambos lados de ella, paralelas a la misma y a una distancia de 6,95 m están las líneas de servicio. La pala y la pelota con la que se juega a pádel se muestra en la Figura 2.

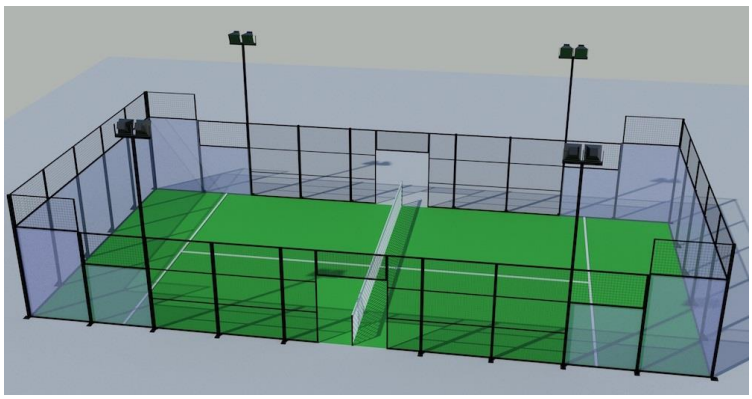

Figura 1. Pista de pádel.

Disponible en: http://manzasport.com/venta-pistas-de-padel [Consulta 08/11/2021]

Figura 2. Palas y pelota de pádel.

Disponible en: https://www.compramejor.es/mejores-palas-padel/ [Consulta 08/11/2021]

3.1 Problema de clasificación de golpes

Como se vio en el punto 2, la clasificación de golpes de pádel puede ser considerada como una clasificación de movimientos de mano. Por tanto, la clasificación de golpes consistirá en identificar qué movimiento ha realizado el deportista con la mano con la que coge su pala. Para ello se hace uso de dos sensores: un acelerómetro y un giroscopio, colocados en la muñeca del deportista, mediante los cuales se consiguen 6 grados de libertad: tres aceleraciones lineales y tres velocidades angulares.

Para llevar a cabo la clasificación, los algoritmos de machine learning necesitan una base de datos que recoja multitud de muestras de aquellos datos que se quieran clasificar, en este caso golpes de pádel. Por tanto, un objetivo previo a la clasificación es crear dicha base de datos. Para ello, se necesita un dispositivo que recoja los datos del golpe como son las velocidades y aceleraciones que experimenta la mano del deportista cuando realiza cada golpe, además de un sistema de detección de golpes que permita guardar cada golpe de forma estándar y eliminar aquella información que no es de interés. A continuación, se verán cada uno de los puntos anteriores.

3.2 Dispositivo para toma de muestras

El dispositivo para la toma de muestra consta de los siguientes componentes hardware y software:

3.2.1 Raspberry Pi

Una Raspberry Pi es un ordenador de placa simple de bajo coste desarrollado por la Raspberry Pi Foundation. Fue diseñado originalmente para la enseñanza de informática en las escuelas, aunque su potencial ha provocado que su uso se extienda por más ámbitos [20].

El modelo utilizado es una Raspberry Pi 4 que cuenta con 4GB de RAM.

3.2.2 IMU

Una unidad de medida inercial o IMU, por sus siglas en inglés, es un dispositivo electrónico que mide las velocidades y aceleraciones que sufre un objeto. Consta de un acelerómetro, que mide las aceleraciones lineales en los 3 ejes del espacio, y de un giroscopio, que mide las velocidades angulares en los 3 ejes. En algunos casos cuenta también con un magnetómetro, que mide el campo magnético en cada eje del espacio.

La IMU utilizada es la LSM9DS1, la cual está incorporada en el Sense Hat, un dispositivo que se conecta a la Raspberry Pi a través del GPIO.

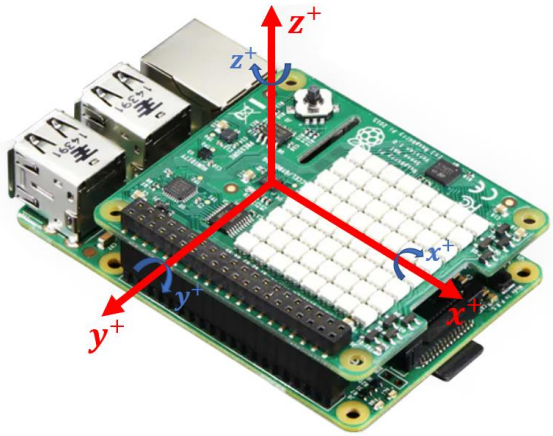

Figura 3. Ejes del Sense Hat.

3.2.3 Plataforma

Para poder colocar la Raspberry Pi a la muñeca, se ha fabricado una plataforma impresa en 3D sobre la que se conecta la placa mediante tornillos y se sujeta a la muñeca mediante velcros.

Figura 4. Raspberry Pi + Sense Hat + Plataforma.

El dispositivo tiene unas dimensiones de 99x68x24 mm y un peso de 103 gramos. Pese a la robustez del dispositivo, permite realizar los movimientos típicos de pádel sin dificultades, dado que se trata de un deporte donde la muñeca no suele tener mucho movimiento, como sí es el caso de otros deportes de raqueta como el tenis de mesa.

3.2.4 Batería externa

La alimentación de la Raspberry Pi se realiza mediante USB por una batería externa portátil, de las que se usan habitualmente para cargar los smartphones. La Raspberry Pi 4 se carga a 5V 3A, una configuración común en la telefonía móvil actual, por lo que hay en el mercado un amplio abanico de posibilidades.

3.2.5 VNC Viewer

VNC es un programa que permite tomar el control del ordenador servidor remotamente a través de un cliente multiplataforma. Por tanto, permite gobernar remotamente la Raspberry Pi desde cualquier ordenador. Para llevarlo a cabo, se establece una conexión Wifi a través de un smartphone, a la que se conectarán tanto el ordenador como la Raspberry. Esta conexión es importante ya que permite guardar los datos etiquetados correctamente.

3.2.6 Python

Python es un lenguaje de programación de código abierto, dinámico, interpretado y multiplataforma. Es administrado por la Python Software Foundation, y fue creado a finales de los ochenta por Guido van Rossum [21]. Es el tercer lenguaje más usado en el mundo, por detrás de C y Java, especialmente popular en el procesamiento de datos.

Este lenguaje se ha usado tanto en la programación del dispositivo de toma de muestras como en el algoritmo de clasificación. La multitud de librerías de las que Python dispone hace que sea un lenguaje muy indicado para este proyecto.

3.3 Sistema desarrollado

El dispositivo para toma de muestras cuenta, por tanto, con la IMU del Sense Hat (LSM9DS1) conectada a la Raspberry Pi. Ambos se dispondrán colocados en la muñeca del deportista para poder llevar a cabo la toma de muestras. La Raspberry Pi se alimenta vía USB mediante una batería externa portátil, que irá colocada en la cintura del deportista. Por último, la Raspberry se conecta gracias a VNC Viewer a un ordenador portátil de forma inalámbrica mediante una red Wifi, que es proporcionada por un smartphone. En la Figura 5 se muestra un esquema de esta configuración.

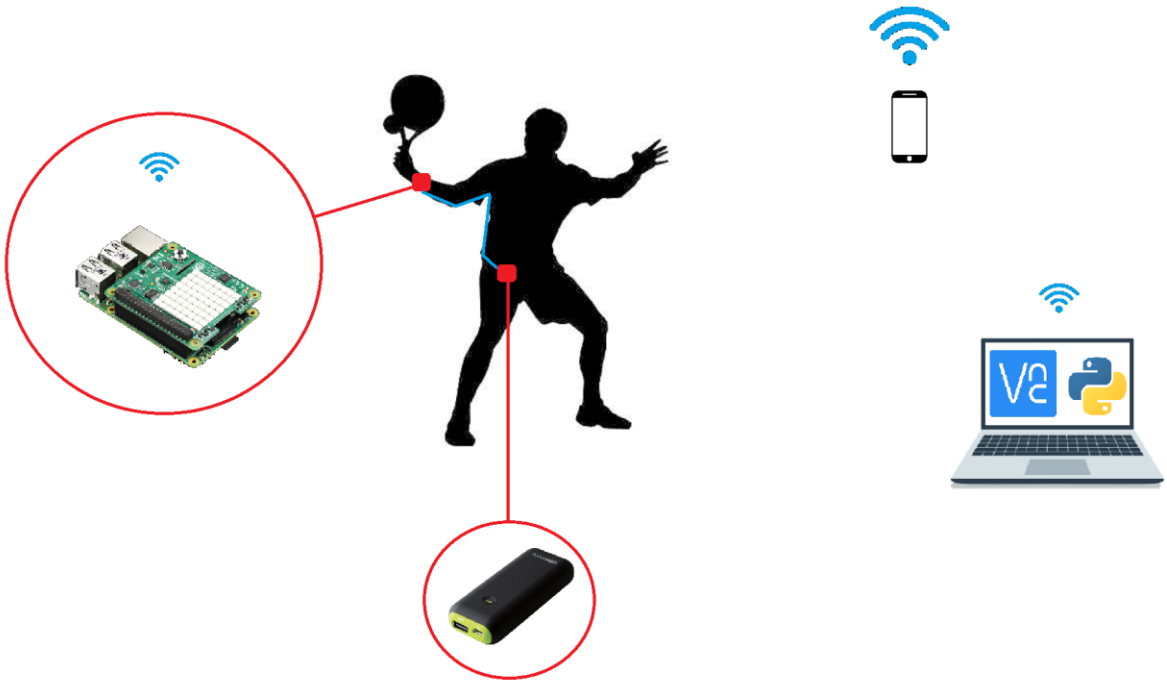

Figura 5. Sistema de toma de datos.

4 CONJUNTO DE DATOS

4.1 Caracterización del conjunto de datos.

Como punto de partida es importante determinar qué golpes se quieren clasificar. Existen multitud de golpes de pádel, los cuales se pueden realizar con diferentes efectos o dejando que la pelota toque la pared antes de devolverla. Para este estudio, se considerarán los golpes más comunes en base a la consulta de dos expertos, ambos entrenadores nacionales. Se tendrá en cuenta si se ha realizado el golpe antes o después de rebotar la pelota en la pared, pero no se considerará el efecto de la pelota en cada golpe. Así pues, se clasificará 13 golpes distintos:

0. ***Fondo derecha***
1. ***Fondo revés***
2. ***Derecha con pared***
3. ***Revés con pared***
4. ***Globo de derecha***
5. ***Globo de revés***
6. ***Globo de derecha con pared***
7. ***Globo de revés con pared***
8. ***Volea de derecha***
9. ***Volea de revés***
10. ***Bandeja***
11. ***Remate***
12. ***Saque***

A continuación, se explicará en qué consiste cada uno de los golpes anteriores. Cabe destacar que la numeración de cada golpe en el listado coincide con el número que lo representa en la base de datos, como se verá más adelante.

0. *Fondo derecha:* se golpea la pelota del mismo lado del brazo hábil del jugador, después de que la pelota bote en el suelo, habitualmente ligeramente por detrás de la línea de servicio.
1. *Fondo revés:* es el golpe opuesto al de derecha, ya que la pelota se golpea del lado contrario al brazo hábil del jugador. También se realiza después de botar la pelota, habitualmente ligeramente por detrás de la línea de servicio.
2. *Derecha con pared:* es un golpe que se realiza hacia delante después de que la pelota haya contactado con la pared del fondo de la pista, golpeando la bola del mismo lado del brazo hábil del jugador.
3. *Revés con pared*: es un golpe que se realiza hacia delante después de que la pelota haya contactado con la pared del fondo de la pista, golpeando la bola del lado contrario al brazo hábil del jugador.
4. *Globo de derecha:* es un golpe de defensa que busca lanzar la pelota con una altura suficiente como para que el jugador contrario que se encuentra cerca de la red no logre devolverla desde su posición y se vea obligado a retroceder. Se golpea la pelota del mismo lado del brazo hábil del jugador.
5. *Globo de revés:* es un golpe de defensa que busca lanzar la pelota con una altura suficiente como para que el jugador contrario que se encuentra cerca de la red no logre devolverla desde su posición y se vea obligado a retroceder. Se golpea la pelota del lado contrario al brazo hábil del jugador.
6. *Globo de derecha con pared:* es un golpe de defensa que busca elevar la pelota para que el jugador contrario retroceda, golpeando la pelota del mismo lado del brazo hábil del jugador. A diferencia del globo de derecha, en este caso se golpea la bola después de que haya contactado con la pared del fondo

de la pista.

7. *Globo de revés con pared:* es un golpe de defensa que busca elevar la pelota para que el jugador contrario retroceda, golpeando la pelota del lado contrario al brazo hábil del jugador. A diferencia del globo de revés, en este caso se golpea la bola después de que haya contactado con la pared del fondo de la pista.
8. *Volea de derecha:* es un golpe de ataque que se realiza cerca de la red y sin dejar botar la pelota. Busca mantener al contrario en el fondo de la pista y ganar el punto. La pelota se golpea del mismo lado del brazo hábil del jugador.
9. *Volea de revés:* es un golpe de ataque que se realiza cerca de la red y sin dejar botar la pelota. Busca mantener al contrario en el fondo de la pista y ganar el punto. La pelota se golpea del lado contrario al brazo hábil del jugador.
10. *Bandeja:* en un golpe a medio camino entre una volea y un remate que se realiza en media pista y busca no perder la iniciativa en la red. Se realiza del lado del brazo hábil del jugador.
11. *Remate:* golpe que se realiza extendiendo el brazo hacia arriba y golpeando la bola en el punto más alto. Se realiza del lado del brazo hábil del jugador con la intención de finalizar el punto.
12. *Saque:* es el golpe con el que se inicia el juego y es el único en el que el propio jugador se coloca la bola. El contacto con la bola debe ser a la altura de la cintura o por debajo, y siempre después de haber botado la bola. A diferencia que en el tenis no se busca ganar el punto directo, se pretende poner en dificultad al contrario para poder ganar la posición de ataque en la red. Se realiza del lado del brazo hábil del jugador.

Aunque existen otros golpes propios del pádel como son: la bajada de pared, la chiquita o el remate por tres, se consideran que los golpes anteriores son los más representativos y comunes en un partido de pádel. El trabajo propuesto se podría extender fácilmente para considerar otros golpes de pádel.

Conocidos los golpes que se recogerán en el conjunto de datos, se pasará a exponer cómo es exactamente. La base de datos ha sido realizada con 12 personas, de las que cabe destacar las siguientes características:

Tabla 2. Deportistas del conjunto de datos.

ID Deportista	Sexo	Nivel	Mano	Revés	Altura (cm)	Edad (años)	Número de golpes
1	Hombre	1	Diestro	1	178	37	109
2	Mujer	2	Diestro	1	157	34	157
3	Hombre	2	Diestro	1	187	23	136
4	Hombre	3	Diestro	1	175	47	313
5	Hombre	4	Diestro	1	176	38	269
6	Mujer	5	Diestro	1	170	28	231
7	Hombre	2	Diestro	1	173	50	183
8	Mujer	1	Diestro	1	164	23	131
9	Hombre	3	Diestro	1	185	30	181
10	Mujer	3	Diestro	2	167	25	236
11	Hombre	4	Diestro	1	184	32	163
12	Mujer	3	Diestro	1	164	22	219
						Total	2328

Algunas características como sexo, altura, edad o número de golpes no necesitan explicación. El resto serán explicadas a continuación.

El ***ID*** del deportista es un número único para cada jugador, que sirve para identificarlo.

Para identificar el ***nivel*** de juego de cada persona, se ha llevado a cabo la siguiente clasificación:

1. **Beginner-novice:** el deportista no está federado y lleva menos de un año jugando regularmente.
2. **Amateur:** el deportista no está federado y lleva más de un año jugando regularmente.
3. **Experienced intermediate level:** el deportista juega competiciones federadas en su comunidad y es categorizado como jugador nivel intermedio por dos entrenadores nacionales.
4. **Experienced high level:** el deportista juega competiciones federadas en su comunidad y es categorizado como jugador nivel alto por dos entrenadores nacionales.
5. **Professional:** el deportista es internacional, juega en el World Padel Tour (competición más importante del pádel).

En cuanto a la ***mano*** con la que juega el deportista, puede observarse que en esta toma de datos todos son diestros. Esto es debido a que la inmensa mayoría de los jugadores son diestros y no es fácil encontrar jugadores zurdos. No obstante, se verá en el apartado 7.2, como punto a tratar en trabajos futuros.

Respecto a ***revés***, identifica si el deportista realiza el golpe de revés usando una mano o las dos. Por tanto, tomará valores 1 o 2, respectivamente. Cabe destacar que en el pádel normalmente se realiza el golpe de revés a una mano, siendo el golpe de revés a dos manos característico de jugadores que se inician en el pádel después de haber practicado tenis.

En resumen, el dataset ha sido realizado por 12 personas (7 hombres y 5 mujeres) de diferentes niveles de juego (desde novato hasta profesional). Se han guardado un total de 2328 golpes de los 13 tipos de golpes que son objeto de estudio en este documento. La cantidad de golpes registrados de cada tipo es la siguiente:

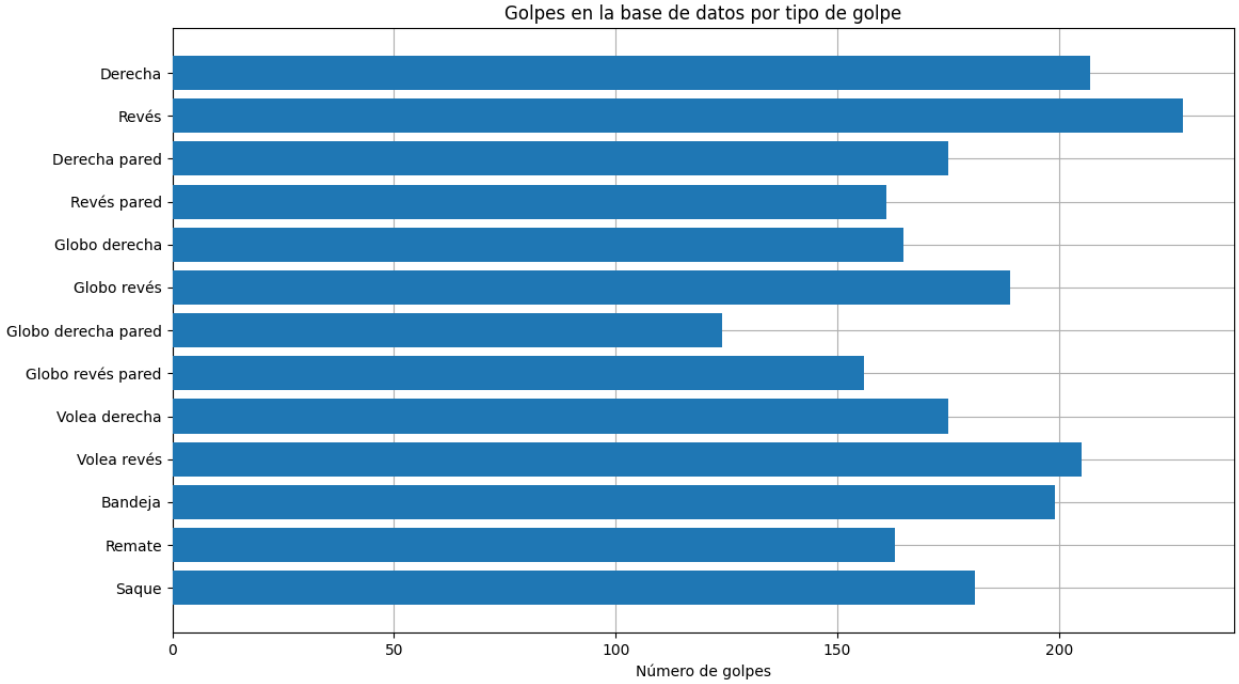

Figura 6. Golpes en la base de datos por tipo de golpe.

4.2 Proceso de toma de datos

Para llevar a cabo el proceso de toma de datos de una forma óptima serán necesarias tres personas: el deportista, una persona responsable de lanzar las bolas al deportista, y una tercera persona responsable de controlar la Raspberry Pi y grabar los datos. No obstante, también se podría llevar a cabo con 2 personas o incluso con una que haga todo, pero el proceso se volvería lento y tedioso. La mayoría de los golpes también han sido grabados en vídeo.

4.2.1 El deportista

En el punto 3.2 se vio el dispositivo para la toma de muestras. El deportista llevará la Raspberry Pi durante el desarrollo de la toma de muestras, junto con el Sense Hat y la batería. El único requisito a nivel técnico es que conozca y sepa ejecutar los golpes que realiza en las pruebas.

4.2.2 Responsable de lanzar las bolas

Esta persona se encarga de que al deportista le llegue la bola en unas condiciones aptas para realizar el golpe que se desee. Por tanto, el responsable de lanzar las bolas deberá conocer el deporte. El uso de material de entrenamiento como un carrito de bolas facilitarán enormemente su trabajo.

4.2.3 Responsable del registro de datos

Esta persona se encarga de guardar los datos de la prueba. Con el ordenador, manejará la Raspberry Pi y guardará los datos de acelerómetro y giroscopio. Además, se graba la prueba en vídeo para validar de forma visual lo que se ha realizado. Un trípode para la cámara/smartphone facilita este trabajo. Cabe destacar que el mismo smartphone que se usa para establecer una conexión wifi, se usa para grabar en vídeo la prueba.

En cuanto a la captura de datos, se llevará a cabo de la siguiente forma. Se divide el proceso en 13 pruebas, cada una de ellas identificada con el nombre del golpe que se quiere guardar. Cada prueba tendrá una duración establecida (normalmente se ha establecido en minuto y medio) y durante su desarrollo, el deportista solo podrá realizar el tipo de golpe que se está guardando. De dicha forma, todos los golpes estarán etiquetados, lo que se conoce como aprendizaje supervisado. A modo de ejemplo, si la prueba es de golpes de remate, durante la prueba el deportista solo realiza golpes de remate, y la prueba se guarda como "remate.csv". El periodo de muestreo es de 20 Hz.

Figura 7. Deportista durante una sesión de toma de datos.

4.3 Detector de golpes

Llegados a este punto, lo que se ha obtenido es un documento de texto plano que recoge los datos de acelerómetro y giroscopio con un periodo de 20 Hz, que se corresponde a un periodo de tiempo en el que se ha realizado una colección de algún tipo de golpe. De todos los datos recogidos, algunos se corresponderán con un golpe, mientras que otros se corresponderán al tiempo entre golpes en el que el deportista puede estar esperando la siguiente bola o moviéndose por la pista para colocarse correctamente. Por tanto, se necesita un sistema que detecte cuándo se ha realizado un golpe para que filtre los datos y elimine aquellos que no son de interés.

Una cuestión importante es qué es exactamente un golpe. ¿Es el golpe el instante de tiempo en el que se produce el impacto con la bola?, ¿es ese instante del impacto más un intervalo de tiempo posterior?, ¿o anterior? En este estudio se ha considerado que el **golpe es el intervalo de tiempo que engloba la preparación del golpeo, el impacto con la bola, y el movimiento inmediatamente posterior al impacto**. De esta forma habrá suficiente información para poder clasificar cada tipo de golpe. Ahora bien, ¿qué intervalo de tiempo es el adecuado?, o aún más importante, ¿qué diferencia ese intervalo de tiempo donde se produce el golpe al resto de datos de la prueba?

4.3.1 Condición de detección

En [10] se realizó un detector de golpes sencillo, que consistía en un umbral en la derivada del giroscopio que obtenía buenos resultados. En un principio, se optó por la misma solución, pero en determinadas circunstancias los resultados no resultaron satisfactorios. Dado que la idea de un umbral de detección resultó bastante interesante, ya que los 6 GDL sufrían cambios significativos al producirse un golpe, alcanzando valores máximos y mínimos tanto en acelerómetro como giroscopio, se optó por definir un umbral en cada GDL; de forma que, si alguno de los ejes de acelerómetro y giroscopio supera su umbral correspondiente en la misma muestra, se podría asumir que se ha producido un golpe. Pero los umbrales de acelerómetro y giroscopio se pueden superar en más de una ocasión durante un mismo golpe, por lo que se opta por una variable que desactive la búsqueda de un golpe mientras no se haya guardado el último golpe detectado; es decir, mientras la muestra actual pertenezca un intervalo de tiempo de un golpe ya detectado. El diagrama de flujo del detector de golpes sería el mostrado en la Figura 8.

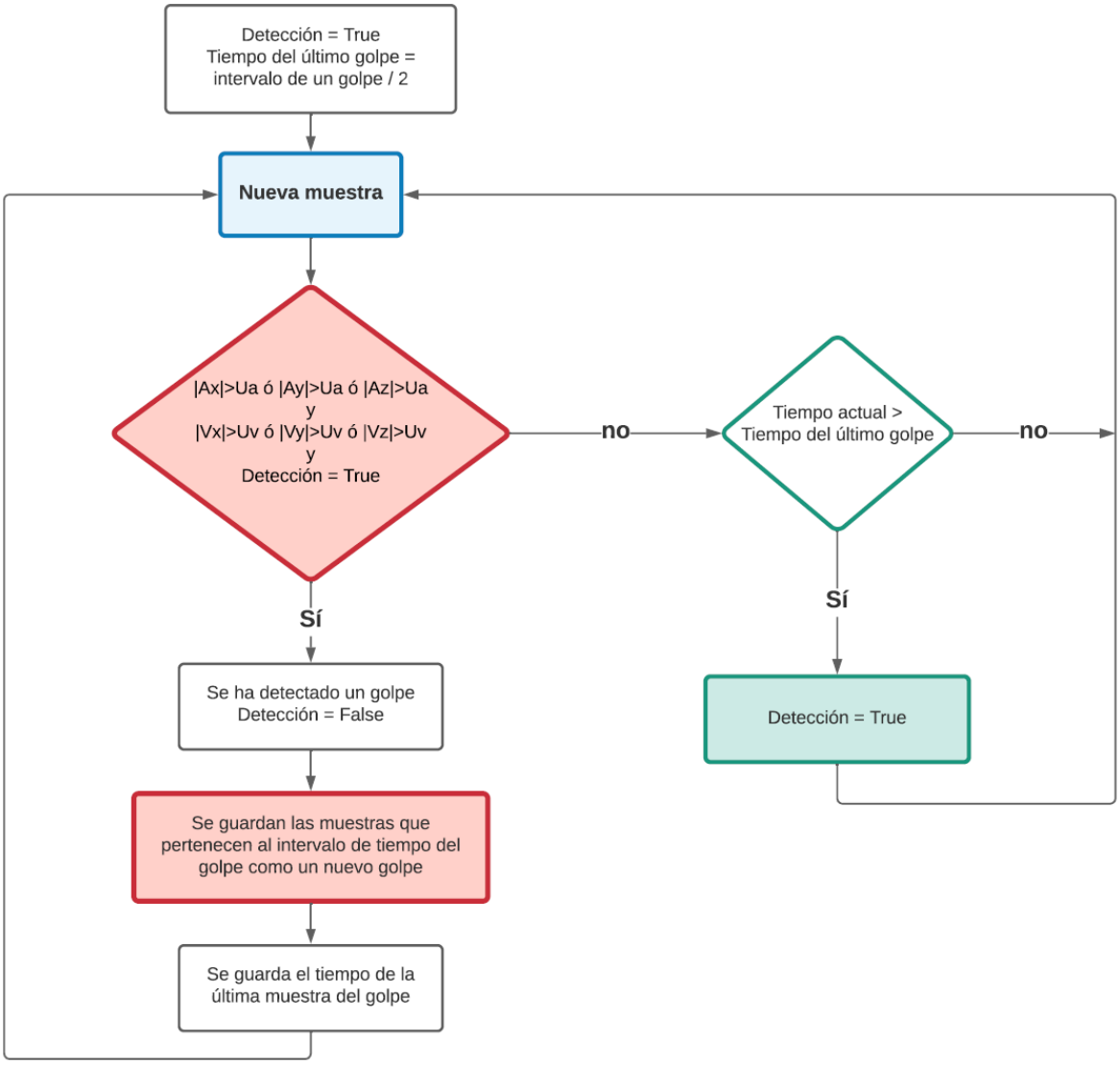

Figura 8. Diagrama de flujo del detector de golpes

En esta figura:

$$Ax = aceleración\ lineal\ en\ el\ eje\ x\ de\ la\ muestra\ actual$$
$$Ay = aceleración\ lineal\ en\ el\ eje\ y\ de\ la\ muestra\ actual$$
$$Az = aceleración\ lineal\ en\ el\ eje\ z\ de\ la\ muestra\ actual$$
$$Vx = velocidad\ angular\ en\ el\ eje\ x\ de\ la\ muestra\ actual$$
$$Vy = velocidad\ angular\ en\ el\ eje\ y\ de\ la\ muestra\ actual$$

$$Vz = velocidad\ angular\ en\ el\ eje\ z\ de\ la\ muestra\ actual$$
$$Ua = umbral\ de\ la\ aceleración$$
$$Uv = umbral\ de\ la\ velocidad$$

Los valores estimados para cada umbral son:

$$\boldsymbol{Ua = 3Gs}$$
$$\boldsymbol{Uv = 5rad/s}$$

4.3.2 Duración del golpe

En la literatura se pueden encontrar algunos ejemplos sobre qué duración es la adecuada para un golpe. En [8] se tomaron 70 muestras por golpe con una tasa de muestreo de 50 Hz, lo que supone una duración de 1.4 segundos por golpe. En [9] se tomaron 20 muestras con una tasa de 20 Hz, por lo que la duración era de un segundo. En [10] se cogieron 1000 muestras con una tasa de 1 kHz, siendo la duración también de un segundo. En [22] se lleva a cabo un estudio donde se observa que una tasa correcta para reconocer actividades humanas se encuentra entre los 20 y 30 Hz.

Fijar una duración del golpe mayor o menor es un tema subjetivo y que puede depender de la naturaleza del golpe. Por ejemplo, hay golpes que requieren una mayor preparación previa al impacto con la bola, como ocurre con el golpeo de bandeja, mientras que hay otros golpes que no se preparan tanto, como es el caso de una volea. Pero el objetivo en este apartado es buscar una duración uniforme de manera que todos los tipos de golpes queden registrados con la misma duración para poder luego clasificarlos. En un principio, se probó con una duración de un segundo como se mostraba en estudios anteriores, pero el clasificador visto en el apartado 4.3.1 no funcionaba correctamente, ya que identificaba más de un golpe cuando se realizaba uno solo, especialmente en los golpes de bandeja y remate. Por tanto, se estableció una duración de 2 segundos. Cabe destacar que cuando se detecta un golpe se guardan las muestras correspondientes a la duración del golpe, de forma que la detección queda justamente en el centro del intervalo.

4.3.3 Resultados

El detector de golpe fue probado con un total de 362 golpes, de los cuales 354 fueron detectados. El procedimiento de validación fue comparar los golpes realizados y detectados con el vídeo que se había grabado en cada prueba. Es un procedimiento lento pero eficaz. La matriz de confusión sería la siguiente:

Tabla 3. Matriz de confusión del detector de golpes.

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	354	8
	Negativos	0	-

El porcentaje de casos en los que el detector acierta, su exactitud, a la que habitualmente se le llama por su nombre en inglés (*accuracy*), será:

$$Accuracy = \frac{VP + VN}{VP + FP + VN + FN} = \frac{354}{354 + 0 + 8} = 97.79\% \quad (1)$$

Otros conceptos importantes son la precisión y la exhaustividad. La precisión (*precision*) mide la probabilidad de que el golpe detectado se corresponda con un golpe real, mientras que la exhaustividad (*recall*) mide la probabilidad de que un golpe real sea detectado.

$$precision = \frac{VP}{VP + FP} = \frac{354}{354 + 0} = 100\% \quad (2)$$

$$recall = \frac{VP}{VP + FN} = \frac{354}{354 + 8} = 97.79\% \quad (3)$$

Cabe destacar que el aspecto más crítico son los falsos positivos, que se deben minimizar a toda costa, ya que un falso positivo significa que se está metiendo en la base de datos un golpe que no es un golpe. Es decir, la precisión del detector debe ser lo más alta posible, como se aprecia en la ecuación (2). Para ello, los valores de los umbrales y de la duración del golpe son conservativos. Por otro lado, un falso negativo solo significa que se está dejando un golpe por el camino, lo que no afectaría al sistema si se tienen un número suficiente de muestras. No obstante, la accuracy del detector es bastante alta, de un 97.79%.

A modo de ejemplo, el detector de golpes tendría el siguiente comportamiento en una prueba. Se puede observar en la Figura 9 que durante la prueba se detectan 18 golpes (cada vez que el detector se pone a nivel alto). Sobre el segundo 50, hay un periodo de tiempo en el que el detector no detecta ningún golpe, pudiéndose observar en las representaciones de acelerómetro y giroscopio que no hay ningún valor pico (golpe) como sí sucede cuando el detector se activa.

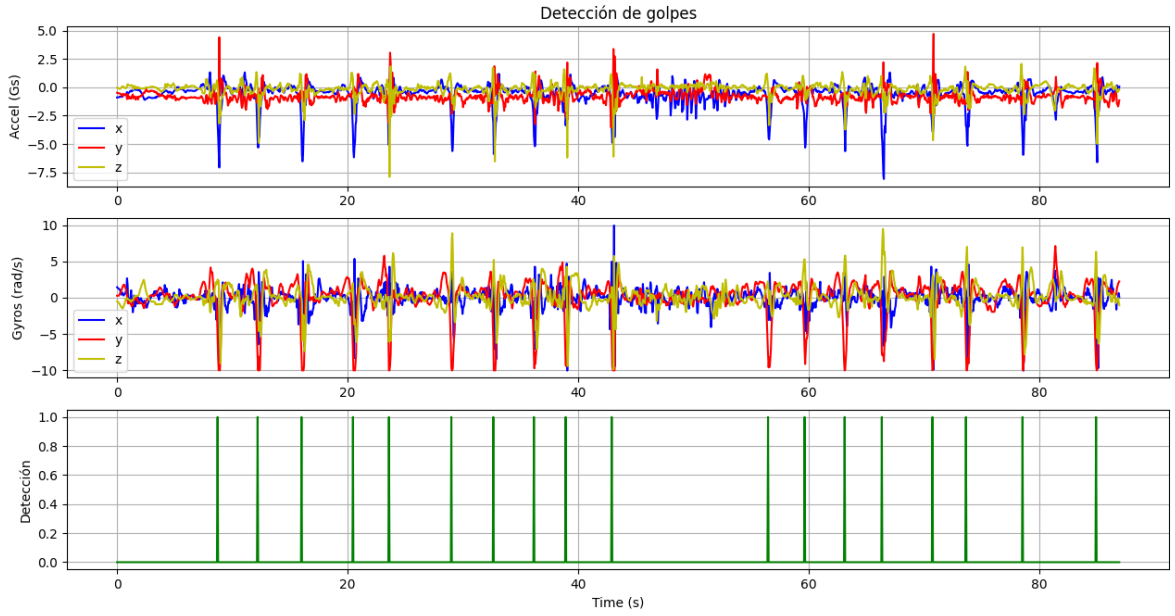

Figura 9. Detección de golpes durante una prueba.

La prueba anterior dura minuto y medio y se han detectado 18 golpes. Eso significa que de los 90 segundos que dura la prueba, apenas 36 segundos (18 golpes por 2 segundos por golpe) son de utilidad. He aquí la importancia

del detector.

Por último, se mostrarán algunos ejemplos de los golpes una vez detectados y aislados del resto de la información no relevante. En la Figura 10 se puede ver un golpe de derecha, mientras que en la Figura 11 se muestra un golpe de revés. Una mejor comparación de ambos tipos de golpes se muestra en la Figura 12, donde se observan comportamientos distintos en cada grado de libertad, siendo más evidentes en los datos del giroscopio.

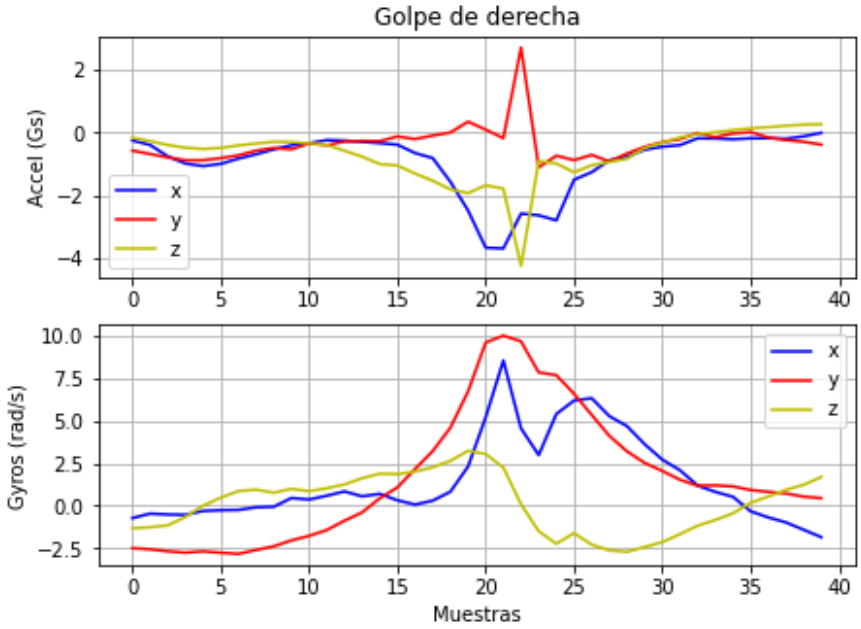

Figura 10. Ejemplo de golpe de derecha.

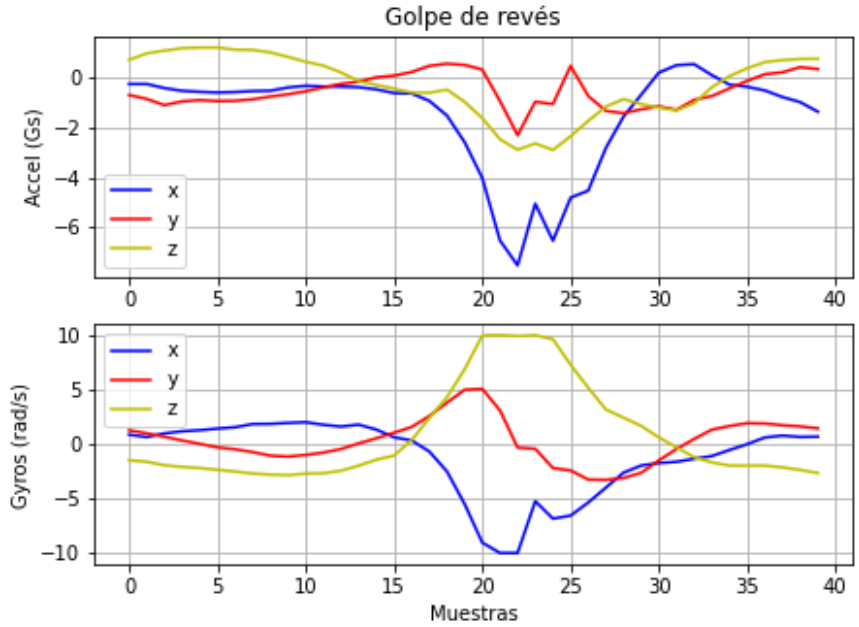

Figura 11. Ejemplo de golpe de revés.

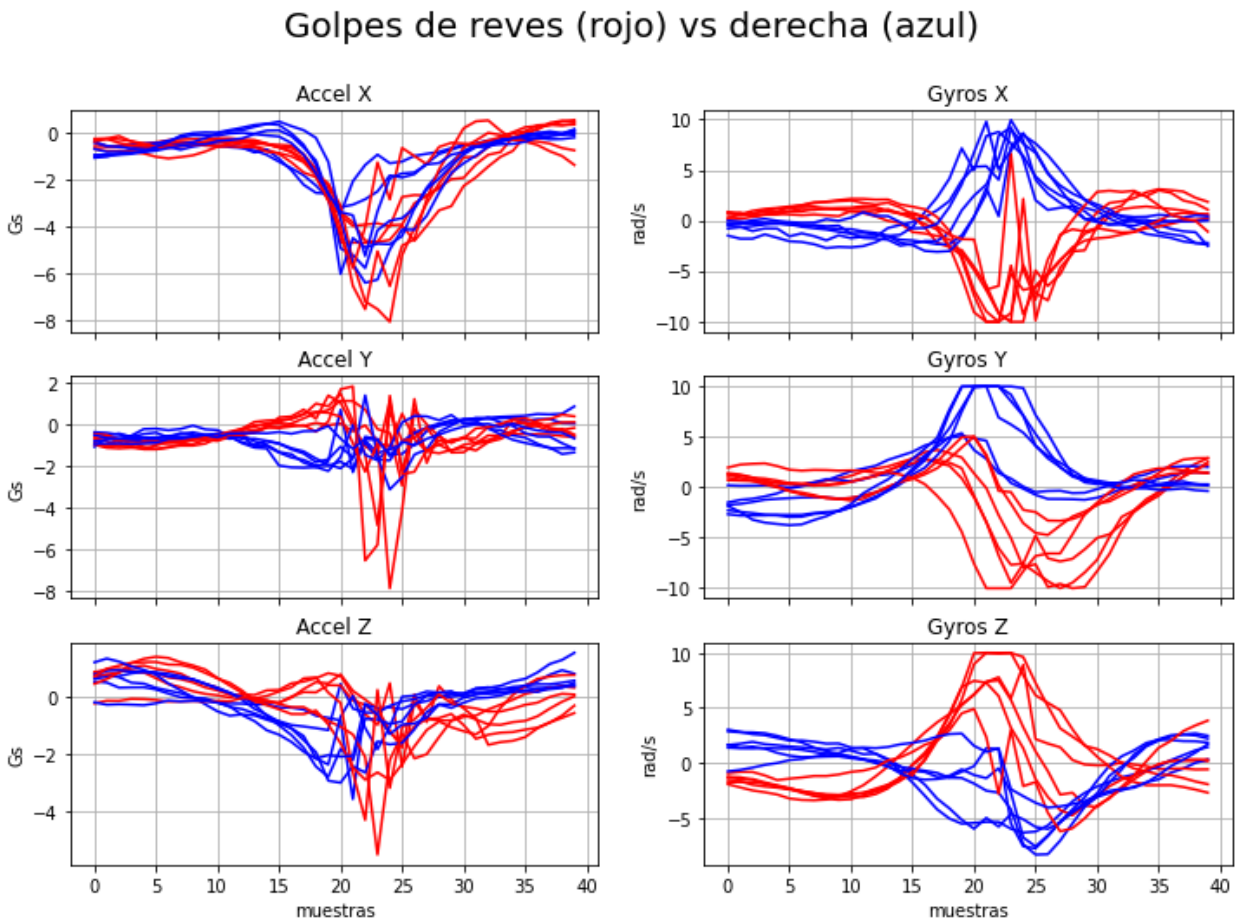

Figura 12. Comparación de un golpe de derecha con un golpe de revés.

4.4 Creación del conjunto de datos

Hasta ahora se ha conseguido extraer e identificar cada golpe por separado. Lo siguiente será recoger todos los golpes en un solo documento y almacenar toda la información relevante de cada golpe. Como es evidente, deberán guardarse las 40 muestras del golpe (2 segundos a 20 Hz), lo que hacen un total de 240 datos (6 grados de libertad por 40 muestras). Pero también deberán guardarse otros datos importantes como son: el tipo de golpe y las características de la persona que ha realizado el golpe, las cuales se vieron en la Tabla 2.

Para hacer el proceso fácilmente extendible, de forma que añadir nuevos golpes a la base de datos no suponga un problema, se ha propuesto el algoritmo de la Figura 13. La idea es que haya una carpeta principal donde se guarden todas las pruebas. Dentro de la carpeta principal, habrá más carpetas identificadas con el nombre de cada deportista que ha hecho las pruebas. Dentro de la carpeta de cada deportista, se ubicarán los documentos de texto plano en el que se recoge cada prueba realizada por el deportista, donde el nombre del documento identifica el tipo de golpe que se realiza. De esta forma, para crear el dataset simplemente habrá que recorrer todos los archivos de todas las carpetas, guardando cada golpe con sus características correspondientes, identificadas con los nombres del archivo y carpeta a los que pertenece el golpe. Por tanto, ampliar el dataset será tan sencillo como añadir nuevas carpetas con nuevos archivos a la carpeta principal. El diagrama de flujo se muestra en la Figura 13.

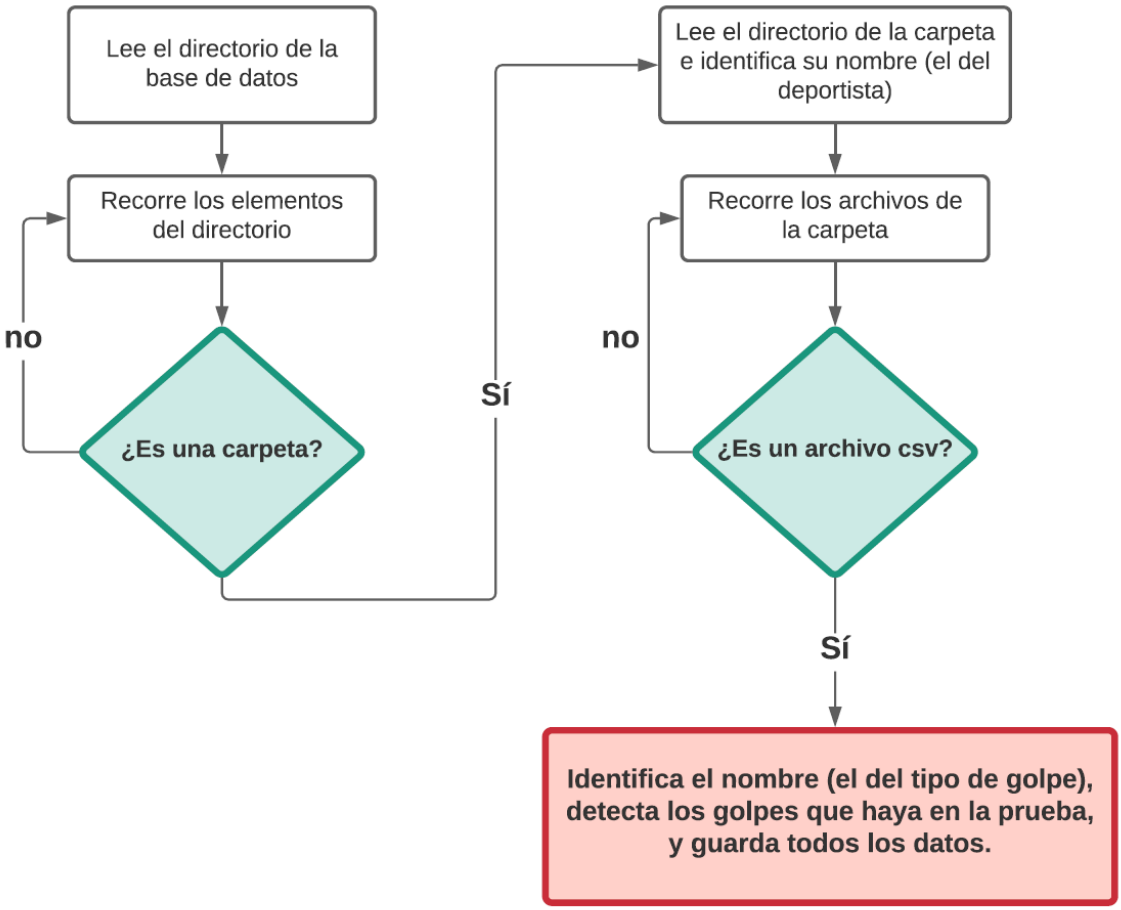

Figura 13. Diagrama de flujo del creador del dataset.

Finalmente, en la Tabla 4 se muestra la forma que tendrá el conjunto de datos final. Se trata de un DataFrame de 2329 filas (una por cada golpe más la cabecera del dataset) y 250 columnas. Las primeras 240 columnas se corresponden con las 40 muestras de cada grado de libertad, mientras que las 10 últimas filas son características del golpe. La columna 241 se corresponde con el *tipo de golpe*, que será el número entero que define al golpe en el apartado 4.1. El *número de golpe* se corresponde con el número que representa al golpe en la prueba del deportista, es decir, si es el primer golpe que se realiza en la prueba, el segundo, etc., por lo que también será un número entero. En la siguiente columna, el *tiempo de golpe* se corresponde con el tiempo (en segundos) en el que se ha realizado el golpe durante la prueba, suponiendo el instante inicial en el inicio de cada prueba de cada deportista. Las siguientes columnas son características que definen al deportista, las cuales fueron explicadas en la Tabla 2. El conjunto de datos está disponible en [23].

Tabla 4. Conjunto de datos final

Ax0…Ax 39	Ay0…Ay 39	Az0…Az 39	Vx0…Vx 39	Vy0…Vy 39	Vz0…Vz 39	Tipo de golpe	Número de golpe	Tiempo del golpe	Sexo	Nivel	Mano	Revés	Altura	Edad	Id
Golpe 1															
Golpe 2															
⋮															
Golpe 2328															

6 DOF × 40 muestras | **Características del golpe** | **Características del deportista**

5 METODOLOGÍA: ALGORITMOS DE APRENDIZAJE AUTOMÁTICO

5.1 Introducción a la clasificación. Conceptos previos

En los apartados siguientes serán tratados los diferentes algoritmos objeto de estudio de este proyecto, pero antes de entrar en detalle es importante conocer ciertos conceptos con el objetivo de comprender mejor el funcionamiento de estos. No obstante, cada algoritmo trabaja de una forma distinta y no todos adoptan cada uno de los conceptos que se mostrarán a continuación.

5.1.1 Modelo matemático

Los algoritmos de clasificación tienen el objetivo de reconocer las diferentes clases[3] que se van a clasificar y con ello hacer una predicción. Para hacer una estimación lo más certera posible, estos algoritmos crean un modelo matemático que les permite clasificar datos nuevos a los que nunca han tenido acceso. El modelo matemático y la forma de obtenerlo es característica de cada algoritmo.

El modelo matemático que crea cada algoritmo es, por tanto, la herramienta que hace posible la clasificación. Sin embargo, no todos los algoritmos necesitan crear un modelo matemático, aunque la mayoría de los que se estudiarán más adelante sí lo hacen.

5.1.2 Conjunto de datos

El aprendizaje automático requiere un conjunto de datos para entrenar el modelo matemático. Habitualmente los datos disponibles se dividen en tres conjuntos: datos de entrenamiento *(training)*, datos de validación *(validation)* y datos de testeo *(test)*. Los datos de entrenamiento son usados por el algoritmo para obtener los parámetros de su modelo matemático. Para adaptar correctamente el modelo a los datos de entrada, el programador puede cambiar los hiperparámetros del modelo (a diferencia de los parámetros que son elegidos por el modelo matemático, los hiperparámetros son variables del modelo accesibles al programador) y entrenar el algoritmo nuevamente. Los datos de validación tienen la finalidad de evaluar el modelo entrenado para ajustarlo hasta obtener unos resultados de validación aceptables. Por último, los datos de testeo se reservan para evaluar el modelo final, por lo que los datos de testeo son los únicos que no han influido en la creación del modelo.

5.1.3 Aprendizaje supervisado y no supervisado

El aprendizaje es supervisado cuando el conjunto de datos dispone de la clase a la que pertenece cada entrada, es decir, incluyen la solución deseada. Se dice entonces que los datos están etiquetados. Esta etiqueta permite al algoritmo ajustar su modelo comparando la salida generada por un dato con su etiqueta. Por otro lado, el aprendizaje es no supervisado cuando los datos no están etiquetados, por lo que no tiene información acerca del valor objetivo de cada dato. Estos algoritmos realizan tareas de agrupamiento con el objetivo de encontrar grupos similares en el conjunto de datos.

Todos los algoritmos que se presentarán en este estudio serán de aprendizaje supervisado. Así pues, el conjunto de datos debe contar con los datos etiquetados, como se mostró en el apartado 4.

[3] Una clase es un conjunto de elementos con características comunes.

5.1.4 Parámetros e hiperparámetros

En el apartado 5.1.2 se ha hablado tanto de parámetros como de hiperparámetros, sin entrar en más detalles. Generalmente, se considera parámetros a aquellas variables internas al modelo cuyo valor es estimado a partir de los datos de entrada, mientras que los hiperparámetros son las variables accesibles al programador que son externas al modelo y que no pueden ser calculadas con los datos de entrada.

5.1.5 Overfitting

El overfitting o sobreajuste es uno de los problemas más comunes en el aprendizaje automático. En general, el overfitting tiene lugar por dos motivos: el modelo utilizado es muy complejo y/o faltan datos de entrenamiento. Esto provoca que el modelo no sea preciso con nuevos datos de entrada que no haya visto anteriormente, haciendo predicciones incorrectas. El overfitting es, por tanto, un fenómeno no deseable. Para evitarlo, existen algunas técnicas como la regularización.

5.1.6 Regularización

La regularización consiste en un conjunto de estrategias orientadas a evitar el overfitting durante el entrenamiento del modelo. Aunque existen diferentes alternativas, todas persiguen el mismo objetivo. Por ejemplo, en las redes neuronales, la regularización se lleva a cabo mediante una capa *dropout*; en SVM la regularización se controla introduciendo una penalización en la función objetivo; y en el árbol de decisión se limita el tamaño del árbol. Todos los casos serán estudiados más adelante.

5.1.7 Librerías para aprendizaje automático

5.1.7.1 Tensor Flow - Keras

Tensor Flow es una plataforma de código abierto para aprendizaje automático. Keras es la API de alto nivel de Tensor Flow, y es una librería de código abierto que permite construir y entrenar modelos de aprendizaje profundo. Presenta tres ventajas clave: tiene una interfaz sencilla, es modular y configurable, y es fácil de extender [24]. En este estudio se usa la versión 2.3.0 de tensorflow y la versión 1.0.8 de keras.

5.1.7.2 Scikit-learn

Scikit-learn es una librería de libre acceso para aprendizaje automático en Python, construida sobre NumPy, SciPy y matplotlib, otras librerías conocidas de Python [25]. En este estudio se usa la versión 0.24.2.

Ambas plataformas simplifican la construcción de los algoritmos presentados en este estudio, por lo que suponen una herramienta esencial en la creación de los modelos. La versión utilizada de Python ha sido la versión 3.8 y la plataforma en la que se ha llevado a cabo la programación ha sido Spyder 4.2.5, en Anaconda.

5.2 Redes neuronales densamente conectadas

Antes de hablar de redes neuronales se debe hablar de Inteligencia Artificial, Machine Learning y Deep Learning. Una introducción a este ámbito se recoge en el libro de Jordi Torres [26], *Deep Learning. Introducción práctica con Keras*, que será usado de base en los siguientes párrafos.

La Inteligencia Artificial es un campo muy extenso que engloba muchas áreas relacionadas con el aprendizaje automático, que dotan a las máquinas de capacidades que pueden ser comparables a la inteligencia característica de los humanos. El Machine Learning, o aprendizaje automático, es un campo dentro de la Inteligencia Artificial que proporciona a los ordenadores la capacidad de aprender sin ser explícitamente programados para ello, es decir, que no han recibido reglas explícitas para lograr su tarea, sino que son capaces de realizarlas de forma automática. Por último, el Deep Learning, o aprendizaje profundo, es una subparte dentro del Machine Learning que permite crear modelos compuestos de múltiples capas mediante las cuales un conjunto de datos de entrada

genera una salida próxima a la esperada [26].

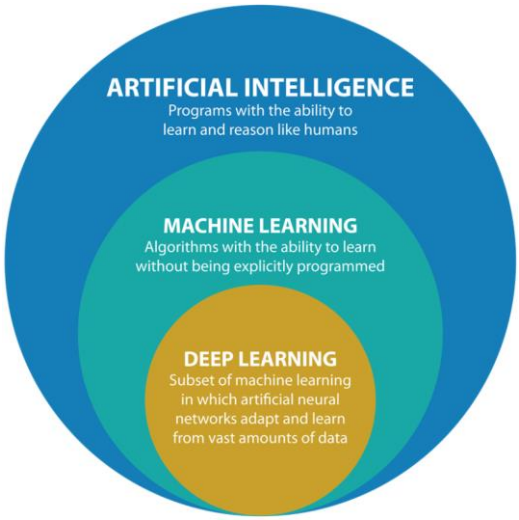

Figura 14. Inteligencia Artificial, Machine Learning y Deep Learning.

Disponible en: https://medium.com/@experiencIA18/diferencias-entre-la-inteligencia-artificial-y-el-machine-learning-f0448c503cd4 [Consulta 26/10/2021]

Una red neuronal es un caso concreto de estrategia de Deep Learning que se ha popularizado en los últimos años al conseguir grandes resultados en el área de visión por computador. Las redes neuronales son modelos computacionales formadas por neuronas conectadas entre sí, de forma que cada conexión está asociada a un peso que determina la importancia de dicha conexión al multiplicarse por el valor de entrada [26]. Para entender mejor el funcionamiento de una red neuronal, se comenzará por la red neuronal más simple: el perceptrón.

5.2.1 Perceptrón

El perceptrón es la forma más simple de red neuronal porque consta de una sola capa que contiene una sola neurona. El perceptrón recibe los valores de entrada, realiza una suma ponderada en función de unos pesos y el resultado lo introduce en una función de activación que genera el resultado final. Una representación gráfica del funcionamiento del perceptrón se muestra en la Figura 15.

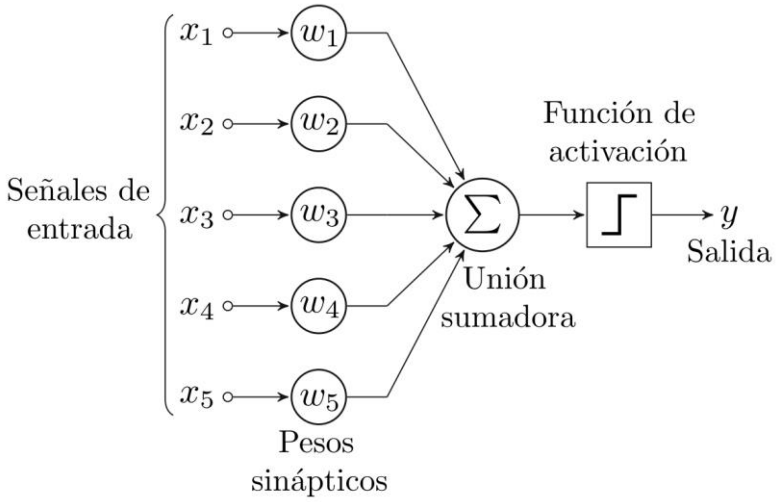

Figura 15. Diagrama de un perceptrón con 5 señales de entrada.

Disponible en: https://es.wikipedia.org/wiki/Perceptrón [Consulta 26/10/2021]

La salida $y(x)$ del perceptrón que realiza la predicción tiene la forma siguiente:

$$z = \sum_{i=1}^{p} \omega_i \cdot x_i + \theta \tag{4}$$

$$y(x) = \begin{cases} 1 & si \quad z \geq 0 \\ 0 & si \quad z < 0 \end{cases} \tag{5}$$

Donde,

$$y(x) = predicción\ del\ modelo$$
$$z = función\ sumadora$$
$$x_i = datos\ de\ entrada, \quad i = \{1,2,\dots,p\}$$
$$\omega_i = pesos\ sinápticos$$
$$\theta = sesgo$$

La ecuación (4) realiza una división del espacio de búsqueda en dos regiones, de forma que cada región representa a una clase. Por ejemplo, para una entrada de dos dimensiones, la división vendría dada por una línea recta, que se representa en la Figura 16. En esta figura, la recta z divide el espacio de búsqueda en dos regiones que representan a cada clase: una región representa los círculos rojos y la otra a los círculos azules. Si la entrada tuviese tres dimensiones, la división estaría definida por un plano en lugar de una recta.

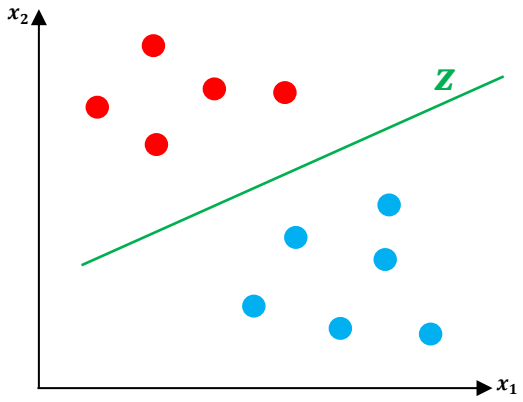

Figura 16. División de clases. Perceptrón.

El aprendizaje del perceptrón consiste, por tanto, en determinar los pesos sinápticos (ω_i) y el sesgo o umbral (θ) que provocan que z realice una división correcta de las clases. El proceso comienza con unos valores aleatorios que se van modificando en función de la diferencia entre los valores deseados y los calculados por la red [27].

La principal limitación del perceptrón es que solo es capaz de realizar una clasificación binaria. Para una clasificación multiclase habría que añadir al menos otra neurona.

En el caso anterior, la función de activación es una función escalón entre 0 y 1. Para cambiar la función de activación a, por ejemplo, una función sigmoid, bastaría con cambiar la expresión de $y(x)$, como se muestra en la ecuación (6).

$$y(x) = \frac{1}{1 + e^{-z}} \tag{6}$$

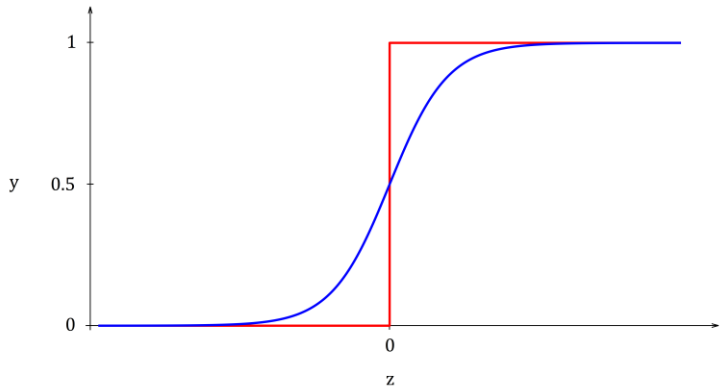

Figura 17. Función de activación en escalón (roja) vs sigmoid (azul).

5.2.2 Proceso de aprendizaje

El perceptrón es un caso muy simple porque tiene una sola capa con una sola neurona, pero lo habitual es encontrar redes con multitud de capas y neuronas. Una red neuronal está formada por neuronas conectadas entre ellas, donde a cada conexión se le asocia un peso, como se mostró en el apartado anterior para el caso particular del perceptrón. En la Figura 18 se representa gráficamente una de estas redes donde cada neurona de una capa se conecta con todas las neuronas de la capa siguiente, de ahí su nombre.

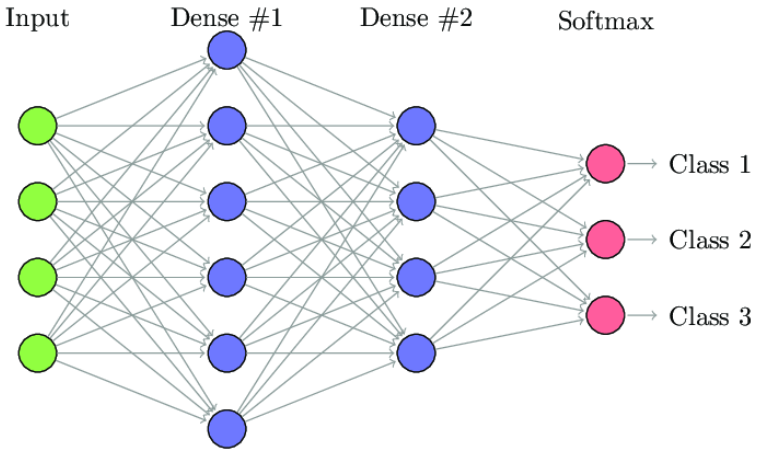

Figura 18. Red neuronal densamente conectada. Ejemplo con dos capas.

Disponible en: https://www.researchgate.net/figure/Example-of-fully-connected-neural-network_fig2_331525817 [Consulta 26/10/2021]

Antes de explicar con detalle el funcionamiento del proceso de aprendizaje de la red neuronal, es necesario introducir el concepto de función de activación. Cada neurona posee una ***función de activación*** que determina la salida de la neurona y que permite introducir la no linealidad en las capacidades de modelado de la red; es decir, permite propagar hacia delante la salida de una neurona. Las funciones de activación más comunes son [26]:

→ ***Linear:*** es una función identidad en la que la salida es igual que la entrada.

$$f(x) = x \tag{7}$$

→ ***Sigmoid:*** permite reducir valores extremos o atípicos en datos válidos. A partir de un rango casi infinito, la función obtiene probabilidades simples entre 0 y 1.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

→ ***Tanh:*** es similar a la función sigmoid, pero en un rango entre -1 y 1.

$$f(x) = \tanh(x) \tag{9}$$

→ ***Softmax:*** permite generalizar una regresión logística de forma que se pueden clasificar múltiples clases y no solo en binario. Por ello, es típica en la última capa de una clasificación multiclase. Garantiza que la suma de todas las neuronas de salida sea 1 y que la probabilidad más alta se use como predicción final.

$$y_i = \frac{e^i}{\sum e^i} \tag{10}$$

→ ***ReLU:*** permite activar un solo nodo si la salida supera un cierto umbral.

$$f(x) = \max\{0, x\} \tag{11}$$

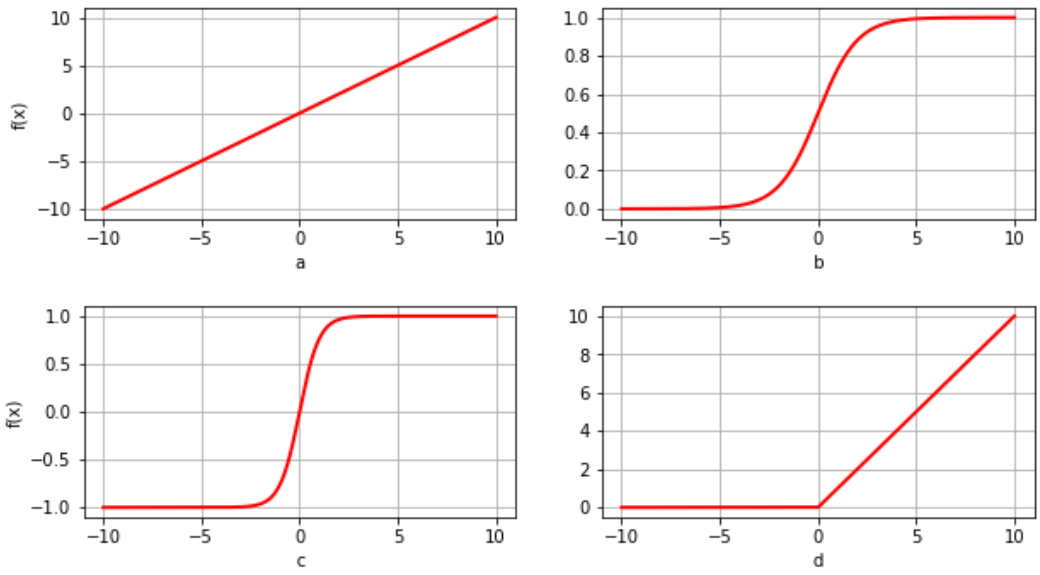

Figura 19. Funciones de activación a) Linear, b) Sigmoid, c) Tanh, d) ReLU

Volviendo al proceso de aprendizaje de la red, este proceso consiste en asignar unos valores a los parámetros de la red (pesos y sesgos) que permitan que la predicción del modelo sea aceptable. El aprendizaje puede ser interpretado como un proceso de ida y vuelta, donde hay tres conceptos importantes: ***forwardpropagation, function loss*** y ***backpropagation*** [26].

El *forwardpropagation* consiste en pasar el conjunto de datos de entrenamiento a través de toda la red para que

las neuronas apliquen sus transformaciones y la red pueda calcular sus predicciones. El paso de los datos por todas las neuronas de la red implica que cada neurona recibe la información de la capa anterior, aplica su transformación y la pasa a la capa siguiente. Cuando esta información haya cruzado toda la red, la capa final ha realizado su transformación y por tanto la predicción de la red.

La predicción de la red puede coincidir o no con la etiqueta de cada dato de entrenamiento. El error de las predicciones se calcula mediante la *function loss*, que compara la predicción con el resultado correcto (los datos poseen su etiqueta, ya que se trata de aprendizaje supervisado). El objetivo del aprendizaje será minimizar este error.

Para poder reducir el error, la estimación *loss* debe ser enviada al resto de la red para corregir así los pesos. Es decir, la información se propaga hacia atrás, lo que se conoce como *backpropagation*. Partiendo de la capa de salida, la estimación loss se propaga hacia todas las neuronas de la capa anterior que contribuyen directamente a la salida. No obstante, cada neurona solo recibe una fracción de la señal total de *loss*, en función de la contribución relativa que haya aportado a la salida con la que se ha calculado la *function loss*. El proceso se repite capa por capa hasta que todas las neuronas de la red reciban su información [26].

Gráficamente, el proceso es el siguiente:

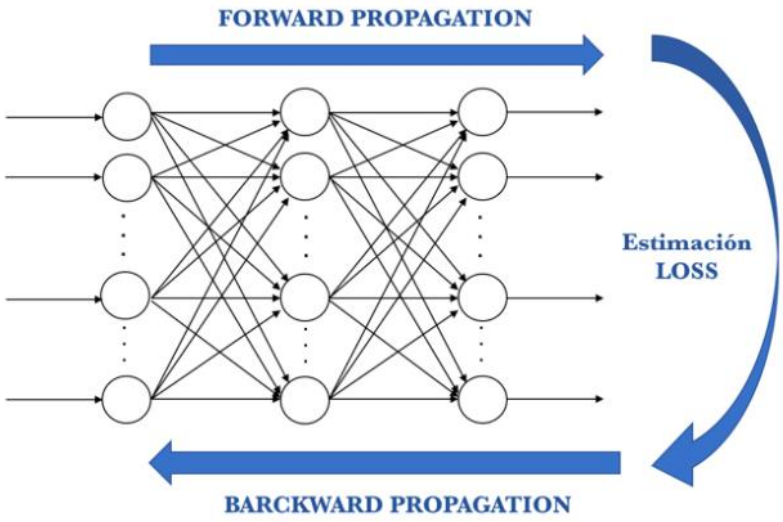

Figura 20. Forwardpropagation, backwardpropagation y loss. Disponible en [26].

Una vez se ha propagado hacia atrás el error cometido en la predicción, se pueden ajustar los pesos de las conexiones entre neuronas. Este ajuste debe realizarse teniendo en cuenta que el objetivo es minimizar el error lo máximo posible, para lo que se usa el *gradient descent*. Esta técnica usa la derivada (o gradiente) de la *function loss*, para ver cuánto debe cambiar el modelo para dirigir el error al mínimo global. En general, los datos se dividen en lotes (*batches*) que pasarán por la red un número determinado de veces (*epochs*). El gradiente se calcula con cada paso de un lote por la red.

5.2.3 Hiperparámetros

Como se mostró en el apartado 5.1.4, un parámetro es una variable interna del modelo cuyo valor se estima en el proceso de aprendizaje, mientras que un hiperparámetro es una variable externa al modelo y cuyo valor es proporcionado por el programador. Los hiperparámetros más comunes son el número de epochs, el batch size y el learning rate [26].

5.2.3.1 Número de epochs

Indica el número de veces en la que los datos de entrenamiento pasan por la red neuronal en el proceso de

entrenamiento. Por ejemplo, si existen 2000 datos y se consideran 10 epochs, los 2000 datos pasarán 10 veces por la red neuronal.

5.2.3.2 Batch size

Dado el gran tamaño de los datos de entrenamiento en aplicaciones típicas de Deep Learning, es habitual dividir estos en lotes más pequeños para ser procesados por la red. El batch size indica el tamaño del lote que pasará por la red en una iteración del entrenamiento antes de actualizar el gradiente.

5.2.3.3 Learning rate

Este hiperparámetro indica la rapidez con la que aprende el modelo, es decir, cómo actualiza los parámetros. Un learning rate grande puede ser bueno para acelerar el aprendizaje del modelo, pero podría provocar que el proceso converja a una solución subóptima o incluso que no alcance la convergencia. Por otro lado, un learning rate pequeño ofrece una mejor oportunidad de alcanzar el mínimo error, pero podría suponer un proceso de aprendizaje inaceptablemente lento.

5.2.4 Overfitting y regularización

Con el objetivo de evitar que el modelo se ajuste en exceso a los datos de entrenamiento, en las redes neuronales se puede incluir una capa *dropout*. El *dropout* consiste en desactivar arbitrariamente neuronas durante el entrenamiento para evitar que el modelo se haga excesivamente dependiente en un grupo pequeño de ellas, provocando overfitting. De esta forma, se le impide al modelo hacer demasiado importantes ciertas neuronas, ya que se le obliga a entrenar sin ellas. En la Figura 21 se representa gráficamente la aplicación de dropout a una red neuronal, donde se aprecia que una vez que se aplica, hay ciertas neuronas que no están conectadas al resto.

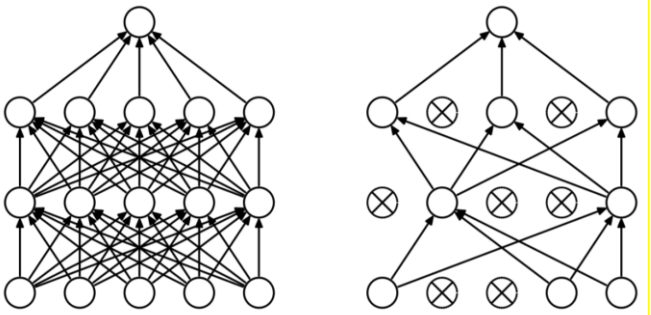

Figura 21. Aplicación de dropout a una red reuronal. Disponible en [28].

Por último, para poder implementar una red neuronal fácilmente en un ordenador y llevar a cabo la clasificación de golpes, se hace uso de Keras.

5.3 Redes neuronales convolucionales 1D

En el punto anterior se presentó la estructura de una red neuronal densamente conectada. La diferencia principal entre una capa densamente conectada y una capa especializada en la operación de convolución, o capa convolucional, es que la capa densa aprende patrones globales en su espacio global de entrada, mientras que la capa convolucional aprende patrones locales en pequeñas ventanas espaciales. Esto es así debido a que las redes neuronales convolucionales 1D hacen la suposición de que las entradas son series temporales, permitiendo codificar ciertas propiedades para reconocer elementos concretos en la serie temporal [26]. Por tanto, este tipo de redes son especialmente adecuadas para el análisis de secuencias temporales de sensores, como es el caso del acelerómetro y giroscopio.

En otras palabras, *en una capa convolucional las neuronas se conectan por pequeñas zonas localizadas, mientras que en una capa densamente conectada todas las neuronas de la primera capa se conectan con todas las de la segunda* [26]. Pero ¿cómo se conectan exactamente esas pequeñas zonas? La conexión se realiza mediante un filtro (al que se llama ***kernel***) que recorre la serie temporal, que no es más que un conjunto de pesos cuyo valor se estima en el proceso de aprendizaje. Por tanto, para cada valor de entrada, el kernel aplica sus pesos a una serie de elementos consecutivos y obtiene un valor de salida [29]. Dicho valor se corresponde con el producto escalar de la subsecuencia de la entrada y un vector kernel de pesos aprendidos de la misma longitud que la subsecuencia de la entrada. Para obtener el siguiente valor de la salida, se sigue el mismo procedimiento con el mismo valor de los pesos de kernel (los pesos se actualizan cuando cada lote pasa por la red), pero la subsecuencia de entrada se desplaza hacia delante para agregar un valor nuevo y eliminar el más antiguo. Este proceso se realiza hasta completar la serie temporal para cada elemento del lote (batch) [29]. En la Figura 22 se representa gráficamente el proceso para un ejemplo con un kernel de longitud 3. Para una subsecuencia de entrada o ventana de convolución, se le aplica el producto escalar del kernel y se obtiene un valor de salida. En la figura se muestran 3 subsecuencias distintas (verde, rojo y amarillo), donde la ventana de convolución se ha ido desplazando secuencialmente hacia la derecha.

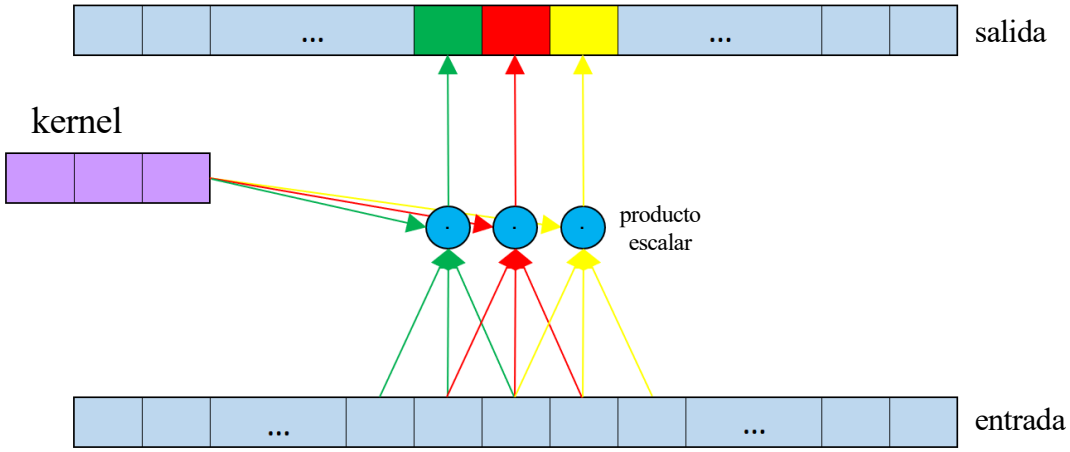

Figura 22. Operación de convolución.

Para el caso concreto de este proyecto, en la Figura 23 se representa la serie temporal del clasificador de golpes de pádel (tres velocidades y tres aceleraciones) y cómo el filtro kernel recorre la serie temporal con una ventana de tamaño fijo (en este caso tiene un valor de 3). Es decir, las neuronas se conectan en pequeñas zonas localizadas y no en todo el espacio global de entrada como lo hacían las redes neuronales densamente conectadas.

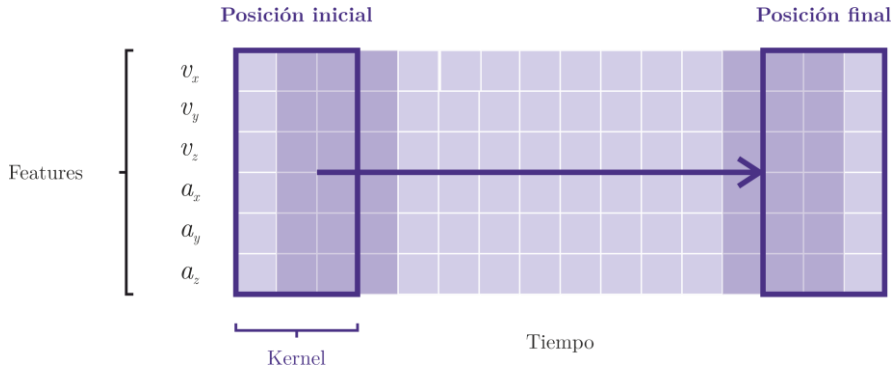

Figura 23. Ejemplo de kernel para golpes de pádel.

5.3.1 Hiperparámetros

Los principales hiperparámetros de una red neuronal convolucional 1D en Keras son [26] [30]:

5.3.1.1 Filters:

El número de *filtros* representa el número de características que se desea manejar para hacer la clasificación.

5.3.1.2 Kernel_size:

Es el tamaño de la ventana de convolución que recorre la serie temporal. Por ejemplo, en la Figura 23 el *kernel_size* tiene un valor de 3.

5.3.1.3 Padding:

Consiste en agregar ceros en los extremos de la serie temporal antes de hacer pasar la ventana por ella. Los valores que puede tomar son: *valid* (no hay padding), *same* (se añade tantos ceros como sea necesario para que la salida tenga la misma dimensión que la entrada) y *causal* (convolución causal, es decir, la salida $[t]$ no depende de la entrada $[t+1]$).

5.3.1.4 Stride:

Indica el número de pasos en que se mueve la ventana de convolución. Cuanto mayor sea, menor superposición tendrá la ventana.

5.3.1.5 Activation:

Función de activación que se emplea. En el apartado 5.2.2 se vieron las principales funciones de activación.

5.4 Árbol de decisión

Un árbol de decisión es un algoritmo de aprendizaje automático supervisado que construye un árbol durante el entrenamiento, que es el que se aplica a la hora de realizar la predicción. Dicho árbol contiene condicionales del tipo if-else anidados que permiten alcanzar el valor buscado. Construido el árbol, es posible conocer la clase de un nuevo objeto haciéndolo pasar por el árbol y respondiendo sucesivamente a las preguntas. Una de las grandes ventajas es que no son sensibles a la escala de los datos, lo que simplifica sustancialmente el preprocesamiento [31].

5.4.1 Estructura

Un árbol de decisión está compuesto por un nodo raíz, nodos de decisión y nodos terminales [32]. Estos se relacionan entre sí formando ramas (o subárboles) a través de flechas, como se detalla a continuación.

5.4.1.1 Nodo raíz

Representa a toda la población y se divide en dos o más conjuntos. El nodo raíz también es un nodo de decisión.

5.4.1.2 Nodo de decisión

Nodo que se divide en subnodos adicionales, donde hay que tomar una decisión.

5.4.1.3 Nodo terminal

Nodo donde termina el flujo y representa la clase a la que pertenece el objeto que haya llegado hasta él.

5.4.1.4 Rama o subárbol

Subsección del árbol de decisión.

5.4.1.5 Flechas

Nexos de unión entre nodos.

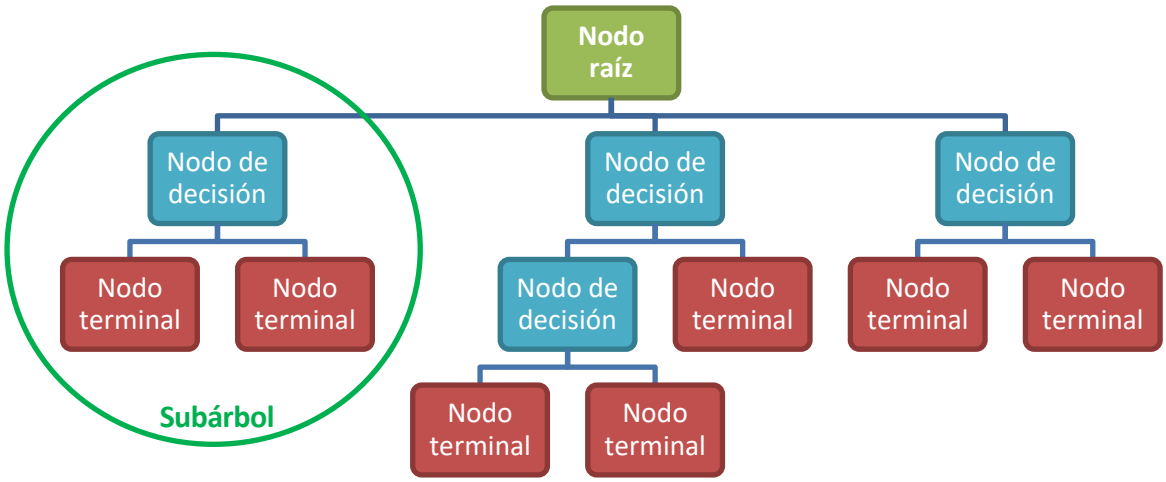

Figura 24. Árbol de decisión de 7 clases.

5.4.2 Función de impureza

En cada nodo de decisión existe un conjunto de elementos de diferentes clases. El criterio utilizado para tomar las decisiones es la minimización de una función de impureza que evalúa la calidad de la división realizada.

Para elegir la variable con la que se obtiene la mejor división se hace uso de tres funciones de impureza, que se describirán a continuación [31]. En un escenario con k clases y con p_k como fracción de elementos de la clase k en la muestra total:

5.4.2.1 Error de clasificación:

Mide la clase con mayor número de elementos. A menor error de clasificación, mayor pureza.

$$error\ de\ clasificación = 1 - \max(p_k) \tag{12}$$

5.4.2.2 Índice Gini:

Mide el grado de impureza de un nodo. A mayor índice Gini, menor pureza, por lo que interesa la variable con menor Gini ponderado.

$$Gini = 1 - \sum_k p_k^2 \tag{13}$$

5.4.2.3 Entropía:

Mide el desorden de un sistema. A menor entropía, mayor pureza.

$$H = -\sum_{k} p_k \cdot \log p_k \tag{14}$$

En general, un grado de impureza nulo significa que todos los elementos son de la misma clase.

5.4.3 Hiperparámetros

Por defecto, el algoritmo va a crear un árbol de decisión tan complejo como sea necesario, lo que se traduce en un estado de sobreentrenamiento. Para evitarlo, se establecen una serie de hiperparámetros en la implementación de Scikit-Learn [31]:

→ *max_depth:* profundidad del árbol.

→ *min_samples_split:* número mínimo de muestras para dividir un nodo.

→ *min_samples_leaf:* número mínimo de muestras por hoja (nodo terminal).

→ *max_leaf_nodes:* número máximo de hojas (número de clases que se quieren clasificar).

→ *min_impurity_split:* mínimo de impureza para dividir un nodo.

→ *criterion:* función de impureza a utilizar.

5.5 K Vecinos más próximos (KNN)

El método de los vecinos más próximos (K-Nearest Neighbors, KNN) es un algoritmo de aprendizaje supervisado que, a partir de un conjunto de datos iniciales, trata de clasificar correctamente las instancias nuevas en función de su similitud con las primeras. En particular, el algoritmo clasifica cada nuevo dato en el grupo correspondiente según el grupo al que pertenezcan sus *k* vecinos más próximos en el espacio de búsqueda [33].

El espacio se divide en regiones por localizaciones y etiquetas de los ejemplos del entrenamiento. Para conocer la distancia desde el punto del espacio que se desea clasificar a los objetos vecinos se usa, generalmente, la distancia euclídea. Dicha distancia se puede deducir a partir del teorema de Pitágoras. En un espacio euclídeo n-dimensional, la distancia euclidiana entre dos puntos $A = (a_1, a_2 \dots a_n)$ y $B = (b_1, b_2 \dots b_n)$ se define como:

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{15}$$

Donde,

$$d = distancia\ euclídea$$
$$A = (a_1, a_2 \dots a_n)$$
$$B = (b_1, b_2 \dots b_n)$$
$$n = nº\ de\ dimensiones\ del\ espacio\ euclídeo$$

La clase más común entre los *k* vecinos más cercanos a dicho punto del espacio será la clase que asignada. Una representación gráfica de este algoritmo se puede ver en la Figura 25, donde se pretende clasificar el punto verde. Con $k = 3$, el punto se clasifica como triángulo rojo ya que, de sus 3 vecinos más próximos, 2 son triángulos rojos y uno es cuadrado azul; por tanto, la clase más común entre sus *k* vecinos más cercanos es triángulo rojo. Por otro lado, con $k = 8$, el punto verde se clasifica de la clase cuadrado azul: hay 3 triángulos rojos y 5 cuadrados azules. Se puede deducir que, para un mismo conjunto de datos, el valor del hiperparámetro k determinará la clase del objeto, por lo que es de especial importancia determinar el valor óptimo de k.

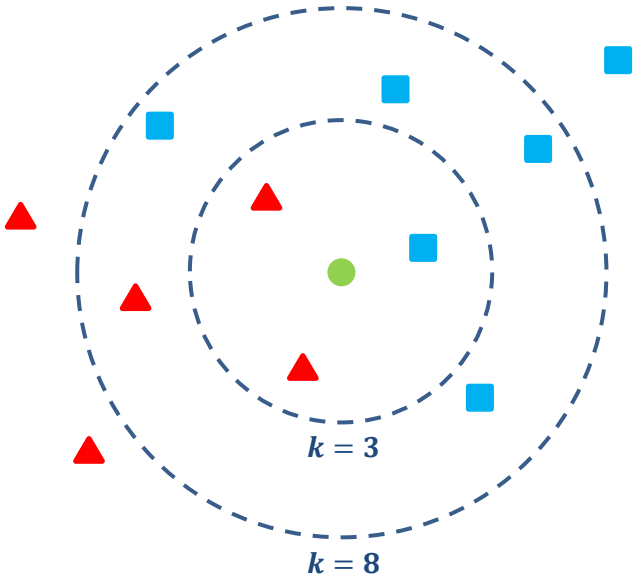

Figura 25. Ejemplo algoritmo KNN.

Puede darse el caso de que haya el mismo número de vecinos de clases distintas. Por este motivo, en clasificación binaria se evita elegir un valor par para k. En clasificación multiclase, como es el caso, esto puede ocurrir para cualquier valor de k (a excepción de k unitario), por lo que conviene definir un criterio para resolver estos conflictos. Dicho criterio puede ser escoger la clase del vecino más próximo, una clase aleatoria (entre las que ha habido empate), o la clase con una distancia media inferior, entre otros. La librería Scikit Learn, en caso de empate escoge el valor que aparezca antes en el conjunto de vecinos.

El algoritmo KNN se conoce como un método de aprendizaje vago (Lazy Learning), ya que no entrena un modelo, si no que simplemente compara el nuevo objeto con los que ya conoce y obtiene así una predicción. Pese a no entrenar un modelo, se trata de un proceso de clasificación lento, ya que cada nueva instancia se compara con todo el conjunto de datos existente.

5.6 Máquinas de vector soporte (SVM)

Las máquinas de vector soporte (Support Vector Machines, SVM) son un conjunto de algoritmos de aprendizaje supervisado que trata de encontrar un hiperplano de separación entre clases. La característica fundamental de las SVM es la búsqueda de una clasificación óptima que se realiza maximizando el margen de separación entre las clases, es decir, el hiperplano de separación entre clases equidista del ejemplo más cercano de cada clase [34].

En la Figura 26 se representa un ejemplo en dos dimensiones, donde se pretende clasificar dos clases: puntos rojos y azules. Dos posibles hiperplanos son H1 y H2, ya que ambos separan las dos clases; pero solo H1 lo hace de forma que maximiza el margen de separación. De hecho, el hiperplano H2 clasificaría al punto verde como azul, mientras que el hiperplano H1 lo clasificaría como rojo, que sería lo correcto. Los vectores soporte (representados en amarillo en la Figura 26) son los que definen la máxima separación del hiperplano que separa las clases. El principal inconveniente del clasificador SVM es su baja robustez, ya que es muy sensible a variaciones en los datos. Observando la Figura 8 se puede entender que la inclusión de un nuevo punto (rojo o azul) puede provocar un cambio drástico en el hiperplano H1.

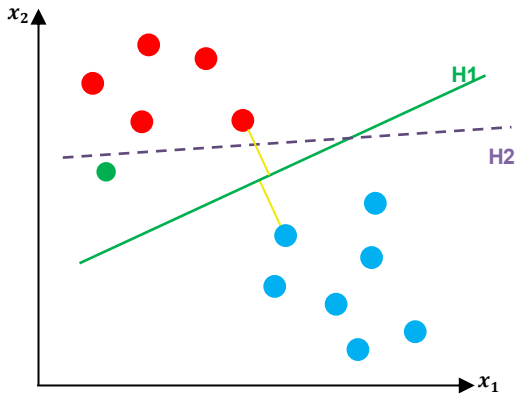

Figura 26. Ejemplo algoritmo SVM.

5.6.1 Problema multiclase

Para un problema multiclase, como es el caso del clasificador de golpes de pádel, se han creado numerosas estrategias para utilizar este algoritmo, tales como one-versus-one, one-versus-all o DAGSVM [35].

5.6.1.1 One-versus-one

Consiste en comparar todos los posibles pares de clases, por lo que para clasificar N clases se necesitan $N \cdot (N-1)/2$ SVMs. Para generar una predicción, se emplean todos los clasificadores, registrando el número de veces que se asigna la observación a cada clase. La clase más frecuente será la clase asignada.

5.6.1.2 One-versus-all

Consiste en comparar cada clase frente a las restantes, de forma que para clasificar N clases se necesitan N SVMs. Para generar una predicción, se emplean los N clasificadores y se asigna la clase a aquella predicción que resulte positiva.

5.6.1.3 DAGSVM

Es una mejora del método one-versus-one. Trata de reducir el tiempo de ejecución, eliminando comparaciones innecesarias empleando un grafo acíclico dirigido (DAG). Por ejemplo, sea un clasificador de cuatro clases (A, B, C, D) con 6 clasificadores para cada par (A-B, A-C, A-D, B-C, B-D, C-D); se inicia con el comparador A-B y resulta que es de clase A. En dicho caso, puesto que no es de clase B, se eliminan aquellas comparaciones que contienen la clase B.

5.6.2 Hiperparámetros

Algunos de los hiperparámetros más importantes para implementar una SVM en Scikit-Learn son [35] [36]:

→ *kernel:* las clases sklearn.svm.SVC y sklearn.svm.NuSVC permiten emplear un kernel lineal (linear), polinomial (poly), gaussiano de base radial (rbf) o sigmoide (sigmoid).

→ *C:* la clase sklearn.svm.SVC controla la regularización con el hiperparámetro C. La regularización consiste en generalizar el modelo para la mayoría de los casos, pese a que algunos pocos casos del entrenamiento no estén perfectamente clasificados. El valor de C es inversamente proporcional a la fuerza de la regularización. NuSVC no lleva a cabo la regularización, de forma que crea el modelo con el máximo número de vectores soporte permitidos.

→ *decision_function_shape*: define el método para multiclase: one-versus-one (ovo) o one-versus-rest (ovr).

6 RESULTADOS

"Haced bien hasta las más pequeñas cosas
sin trascendencia aparente."

- Juan Ramón Jiménez -

En este capítulo se mostrarán los resultados obtenidos con cada uno de los clasificadores implementados, así como el procedimiento seguido para el estudio de cada uno de ellos. Los códigos que se han desarrollado se encuentran disponibles en el repositorio de GitHub [23], y las versiones de las librerías utilizadas se mostraron en el apartado 5.1.7.

El conjunto de datos creado contiene 240 características que se corresponden con los valores de la serie temporal de cada grado de libertad (6 GDL por 40 muestras). Asimismo, posee la etiqueta del golpe y otras características relacionadas con el desarrollo de la prueba y el deportista, como se mostró en la Tabla 4. Dado que la clasificación de golpes de pádel se ha considerado como una clasificación de movimientos de mano, solo se hará uso de la serie temporal del golpe y su etiqueta, por lo que el clasificador no tendrá información de quién ha realizado el golpe o qué lugar ocupa el golpe en la prueba realizada. Así pues, la entrada a los algoritmos de aprendizaje automático serán las series temporales de los golpes. Estos datos en crudo pueden ser procesados antes de pasar por el algoritmo con lo que se conoce como "***feature engineering***", o ingeniería de características. Esta técnica consiste en extraer las características más relevantes de los datos para que sirvan de entrada al algoritmo en el entrenamiento del modelo. En este caso, las características extraídas son el valor máximo, mínimo y medio de la serie temporal de cada grado de libertad del golpe. De esta forma se reduce el tamaño del conjunto de datos, ya que para definir un golpe con la serie temporal se necesitan 240 datos, mientras que con feature engineering solo 18, siendo la etiqueta o label la misma en ambos casos. Por tanto, se estudiará el comportamiento de los clasificadores tanto con la serie temporal como con la técnica feature engineering. En la Figura 27 se muestra un esquema comparativo entre las dos entradas a los algoritmos.

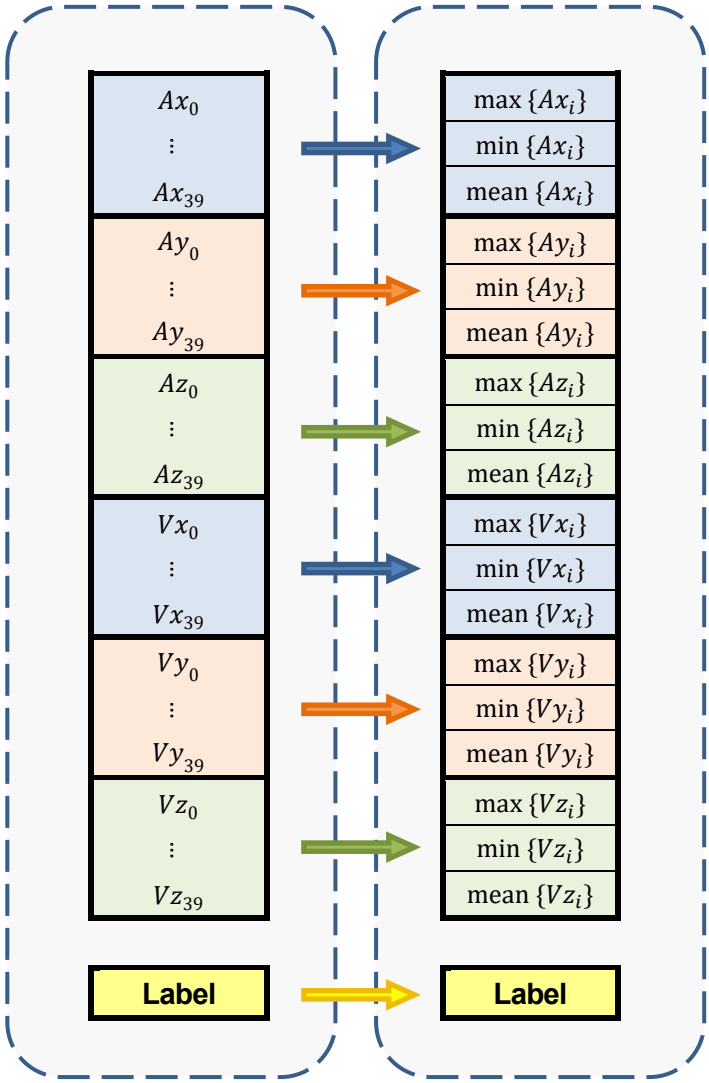

Figura 27. Serie temporal vs feature engineering.

El conjunto de datos de entrada ya sea la serie temporal o con feature engineering, será dividido en entrenamiento (64 %), validación (16 %) y testeo (20 %). En los casos en los que no se lleve a cabo la validación, el conjunto de datos solo se dividirá en entrenamiento (70 %) y testeo (30 %). La división de los datos se hará de forma balanceada y aleatoria, aunque con una *"aleatoriedad fija"*, es decir, aunque la división sea aleatoria, esta será siempre la misma para que todos los algoritmos cuenten con la misma entrada.

Para una justa comparación entre algoritmos, primero se realiza una hiperparametrización de cada uno de ellos con el fin de ajustar correctamente cada algoritmo al problema planteado. Esta hiperparametrización se realizará mediante una búsqueda en rejilla o "***grid search***", donde se establecen unos valores posibles para cada hiperparámetro, que serán evaluados para luego encontrar la mejor combinación entre ellos, es decir, aquella que consiga una mayor accuracy.

Para cada algoritmo, se presentará una tabla con los valores asignados a cada hiperparámetro con los que se

realiza la búsqueda de la mejor configuración. Se presentará también un diagrama de bigotes con los resultados tras ejecutar el código en repetidas ocasiones, ya sea con la misma hiperparametrización o con diferentes valores para los hiperparámetros. Por último, se mostrará la matriz de confusión para la mejor accuracy. Una matriz de confusión relaciona las predicciones realizadas por el clasificador con los valores reales (etiquetas) de los datos, por lo que permite ver de forma gráfica tanto los fallos como los aciertos del clasificador. Dado que la matriz de confusión permite ver los fallos que se han cometido, se comentará la naturaleza estos. Para poder referenciar a los golpes de una forma sencilla, se llamarán a estos por sus iniciales, de forma que:

D: fondo Derecha

R: fondo Revés

DP: Derecha con Pared

RP: revés con Pared

GD: Globo de Derecha

GR: Globo de Revés

GDP: Globo de Derecha con Pared

GRP: Globo de Revés con Pared

VD: Volea de Derecha

VR: Volea de Revés

B: Bandeja

RM: ReMate

S: Saque

Una vez conocido el procedimiento a seguir para el estudio de cada algoritmo, se mostrarán los resultados obtenidos en los siguientes apartados. El código que se ha realizado para cada algoritmo puede consultarse en el repositorio de GitHub disponible en [23].

6.1 Redes neuronales densamente conectadas

En el apartado 5.2 se habló del funcionamiento de las redes neuronales densamente conectadas. Este tipo de redes tienen una serie de hiperparámetros para ajustar la red a cada caso, como el número epochs, el batch size, o el número de filtros. Dado que las posibles combinaciones de los anteriores hiperparámetros pueden ser infinitas, se llevará a cabo una búsqueda en rejilla o "grid search". La división del conjunto de datos será de un 64 % para entrenamiento, un 16 % para validación y un 20 % para testeo. En los siguientes puntos se probarán diferentes configuraciones para los dos tipos de datos de entrada: serie temporal y feature engineering.

6.1.1 Red neuronal densa con serie temporal

Para este caso, la entrada al algoritmo será la serie temporal completa del golpe, es decir, las 40 muestras de cada uno de los 6 grados de libertad, que suponen 240 valores en total.

6.1.1.1 Clasificador con dos capas densamente conectadas

En primer lugar, se probará un clasificador con dos capas densamente conectadas. La segunda de ellas tendrá 13 filtros, coincidiendo con el número de clases que se van a clasificar. El número de filtros de la primera capa, el número de epochs y el tamaño de los lotes serán variables, de forma que se prueban diferentes configuraciones hasta obtener la que ofrece un mejor resultado. Puesto que las posibles configuraciones son infinitas, con el fin de disminuir el tiempo de cálculo se acota cada variable y se prueba un número finito de configuraciones con una búsqueda en rejilla:

Tabla 5. Configuraciones red neuronal densa. Serie temporal. Clasificador con dos capas.

Número de filtros de la primera capa	[50, 100, 200, 1000]
Epochs	[10, 40, 70]
Batch size	[30, 50, 70]

Entre las posibles configuraciones, el clasificador que obtuvo una mayor accuracy fue el siguiente:

```
Model: "sequential_352"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_352 (Flatten)        (None, 240)               0
_________________________________________________________________
dense_704 (Dense)            (None, 1000)              241000
_________________________________________________________________
dense_705 (Dense)            (None, 13)                13013
=================================================================
Total params: 254,013
Trainable params: 254,013
Non-trainable params: 0
```

Este modelo tiene una primera capa con 1000 filtros, seguido de una segunda con 13. La primera capa tiene como función de activación la función ReLU (Rectified Linear Unit), mientras que la función de la segunda es softmax. Para poder llevar a cabo una clasificación multiclase, la función de activación de la última capa debe ser softmax, ya que devuelve la distribución de probabilidad sobre clases de salida mutuamente excluyentes [26]. Por tanto, tanto en este como en los clasificadores de redes neuronales que se verán a continuación, la última capa será una capa densamente conectada con 13 filtros y función de activación softmax.

Cabe destacar que, antes de la primera capa, se dispone una capa de aplanamiento de las características llamada *flatten*. Esto es así ya que los datos se recogen de la base de datos en una matriz, pero las capas densas necesitan un vector como entrada.

En cuanto a los hiperparámetros relativos a epochs y batch size, se han tomado valores de 70 y 30, respectivamente, obteniéndose una accuracy de 89.14% ($\pm$ 0.52%), como se resume en la Figura 28.

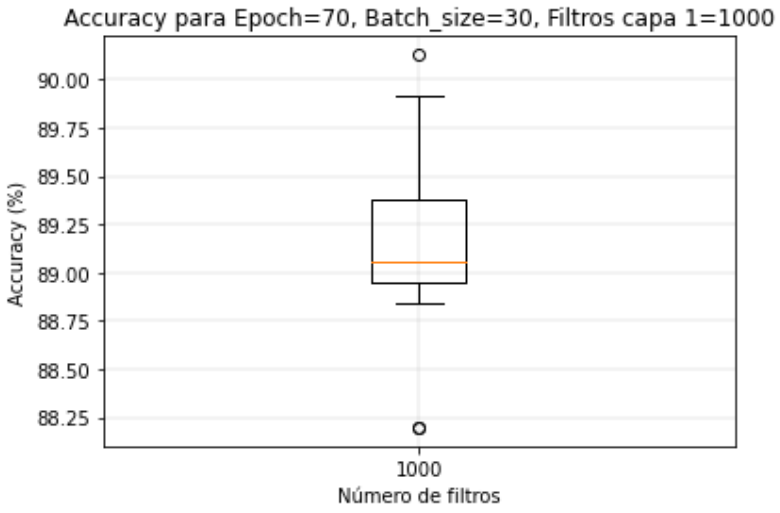

Figura 28. Red neuronal densa con serie temporal. Clasificador con dos capas. Accuracy para mejores hiperparámetros.

La matriz de confusión para el mejor de los casos de esta configuración se muestra en la Figura 29. Gracias a la matriz de confusión se pueden identificar las clases con las que más problemas tiene el algoritmo para su clasificación. Cada casilla de la matriz se identifica por dos valores: el valor que ha predicho el algoritmo (eje horizontal) y el valor real del golpe (eje vertical). Por tanto, una casilla que en el eje horizontal tenga RP y en el eje vertical R, se corresponderá con el número de golpes en los que el algoritmo ha identificado como un golpe de *Revés Pared* un golpe que realmente es de *fondo Revés* (todos los golpes se asociaron a sus iniciales al principio de este capítulo). Por ejemplo, si en dicha casilla hay un 3, significa que ha identificado como RP tres golpes que realmente son R. Por tanto, la accuracy del clasificador será mayor cuanto más altos sean los números de la diagonal principal de la matriz y más bajos sean el resto.

En la Figura 29, el error más común, es decir, el valor más alto fuera de la diagonal principal, es predecir un golpe GDP en lugar de un GD, y un golpe B en lugar de un RM, ambos con 4 predicciones erróneas. Le sigue confundir un golpe GDP con DP, DP con GDP y RP con GRP, todos con 3 errores. El algoritmo tiene, por tanto, **mayores problemas de clasificación con aquellos golpes cuya dinámica es más parecida**. Por ejemplo, el error más común en este caso es confundir un golpe GDP con GD. El movimiento que realiza la mano en ambos golpes es muy similar ya que ambos son golpes de globo y de derecha, donde la diferencia reside en que en el golpe con pared se golpea la pelota después de que esta haya tocado la pared del fondo de la pista, mientras que en el golpe sin pared se golpea la pelota sin dejar que toque la pared de fondo. El siguiente error más común es confundir un golpe B con RM, donde ambos golpes se realizan con la mano elevada: a la altura de la cabeza en el caso de la bandeja y más alta y con el brazo extendido en el caso del remate. Para el resto de los casos puede compararse el movimiento de los golpes atendiendo a la explicación realizada en el apartado 4.1.

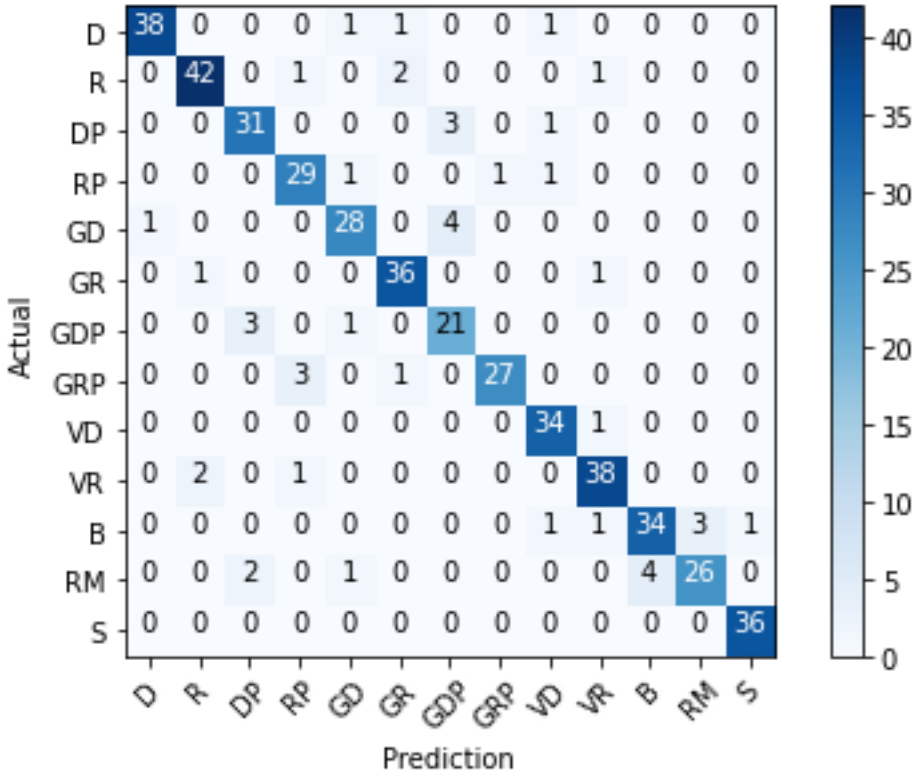

Figura 29. Red neuronal densa con serie temporal. Clasificador con dos capas. Matriz de confusión.

6.1.1.2 Clasificador con tres capas densamente conectadas

En este caso, se probará un clasificador con tres capas densamente conectadas. Como en el caso anterior, la

última capa tendrá 13 filtros, coincidiendo con el número de clases que se van a clasificar. Se probará una serie de configuraciones según los valores de la Tabla 6. El número de filtros de la segunda capa será la mitad del de la primera capa.

Tabla 6. Configuraciones red densa. Serie temporal. Clasificador con tres capas.

Número de filtros de la primera capa	[50, 100, 200, 1000]
Epochs	[10, 40, 70]
Batch size	[30, 50, 70]

El clasificador que alcanzó una mayor accuracy fue el siguiente:

```
Model: "sequential_506"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_506 (Flatten)        (None, 240)               0
_________________________________________________________________
dense_1161 (Dense)           (None, 1000)              241000
_________________________________________________________________
dense_1162 (Dense)           (None, 500)               500500
_________________________________________________________________
dense_1163 (Dense)           (None, 13)                6513
=================================================================
Total params: 748,013
Trainable params: 748,013
Non-trainable params: 0
```

Este modelo tiene una primera capa con 1000 filtros, seguido de una segunda con 500 y una tercera con 13. Como se comentó en el punto anterior, las capas densas van precedidas de una capa de aplanamiento de los datos; y la función de activación será ReLU, excepto para la última capa que será softmax. Los hiperparámetros epochs y batch size se han fijado en 70 y 30, respectivamente, obteniéndose una accuracy de 90.44% (± 0.57%), como se muestra en la Figura 30.

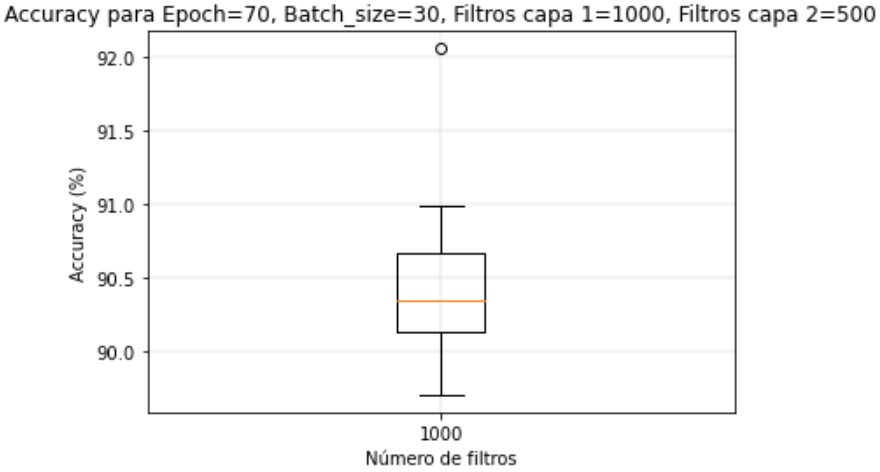

Figura 30. Red densa con serie temporal. Clasificador con tres capas. Accuracy para mejores hiperparámetros.

La matriz de confusión para el mejor de los casos de esta configuración se muestra en la Figura 31. En este caso, el error más común es confundir golpe GDP con GD, con 4 fallos. Confundir B con RM, RM con B y GDP con DP tienen todos 3 fallos. De nuevo, hay mayores problemas con aquellos golpes más similares.

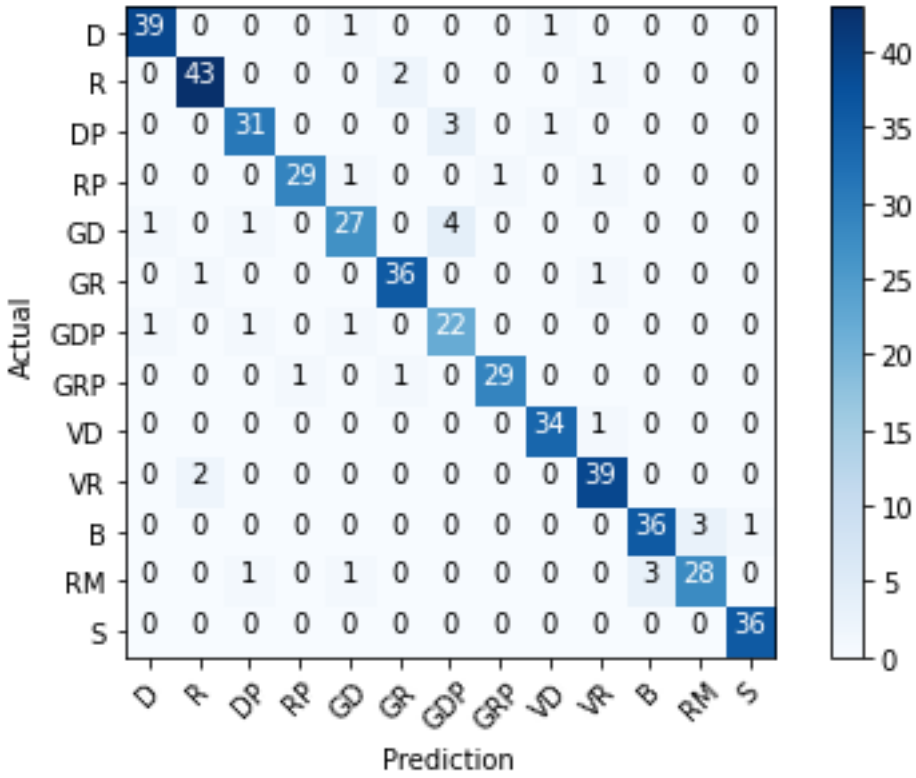

Figura 31. Red neuronal densa con serie temporal. Clasificador con 3 capas. Matriz de confusión.

6.1.2 Red neuronal densa con feature engineering

6.1.2.1 Clasificador con dos capas densamente conectadas

Como se vió anteriormente, la última capa tendrá 13 filtros con función de activación softmax. El número de filtros de la primera capa, el número de epochs y el tamaño de los lotes de las diferentes configuraciones se muestran en la tabla siguiente.

Tabla 7. Configuraciones red neuronal densa. Feature engineering. Clasificador con dos capas.

Número de filtros de la primera capa	[100, 500, 1000, 1500, 2000]
Epochs	[40, 70, 100, 200]
Batch size	[30, 50, 70]

La configuración con mayor accuracy fue de 2000 filtros en la primera capa, epoch de 200 y batch size 30. Consigue una accuracy de 79.99% (± 1.18%), como se muestra en la Figura 32. El clasificador es el siguiente:

```
Model: "sequential_379"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_379 (Flatten)        (None, 18)                0
_________________________________________________________________
dense_758 (Dense)            (None, 2000)              38000
```

```
dense_759 (Dense)            (None, 13)                26013
=================================================================
Total params: 64,013
Trainable params: 64,013
Non-trainable params: 0
```

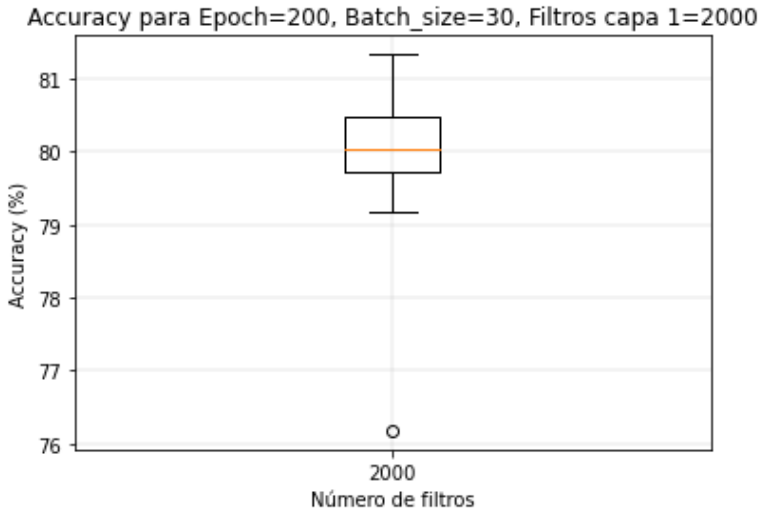

Figura 32. Red neuronal densa con feature engineering. Clasificador con dos capas. Accuracy para mejores hiperparámetros.

La matriz de confusión para el mejor de los casos de esta configuración se muestra en la Figura 33. En este caso, el error más común es confundir un golpe VR con R, con 7 fallos. Hay varios casos con 4 fallos, entre los que destacan VD con B. Estos golpes también tienen ciertas similitudes, siendo su principal diferencia una mayor altura de golpeo para el golpe de bandeja y una mayor preparación.

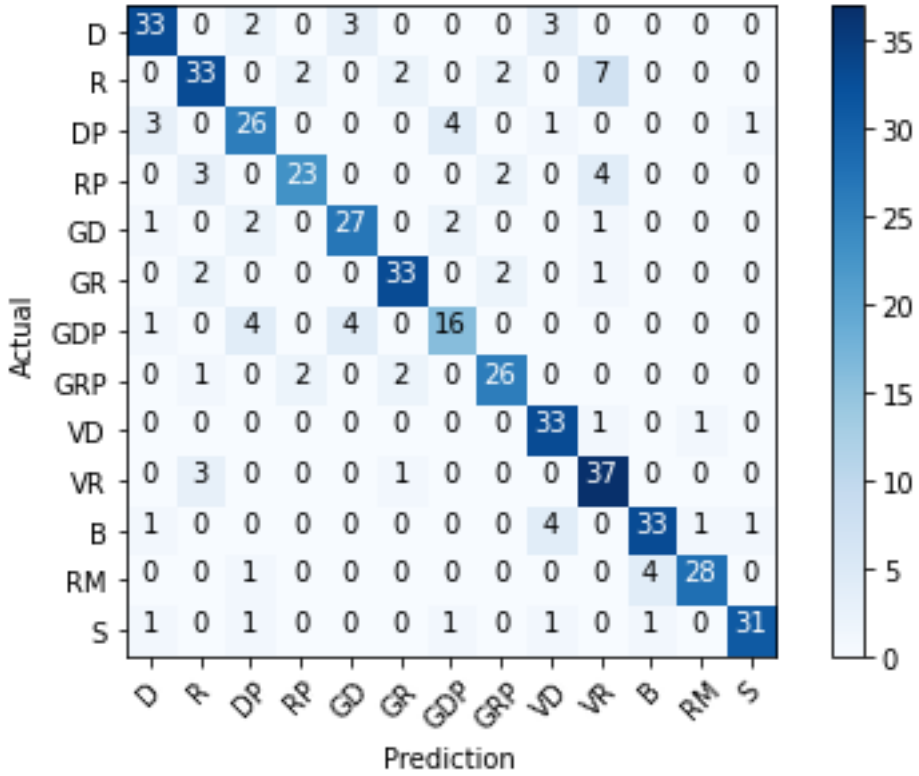

Figura 33. Red neuronal densa con feature engineering. Clasificador con dos capas. Matriz de confusión.

6.1.2.2 Clasificador con tres capas densamente conectadas

Para cada configuración, el número de filtros de la primera capa viene determinado en la Tabla 8. La segunda capa tendrá la mitad de los filtros de la primera, y la última capa tendrá 13. El número de epochs y el tamaño de los lotes de las diferentes configuraciones serán:

Tabla 8. Configuraciones red neuronal densa. Feature engineering. Clasificador con tres capas.

Número de filtros de la primera capa	[100, 500, 1000, 1500, 2000]
Epochs	[40, 70, 100, 200]
Batch size	[30, 50, 70]

Entre las posibles configuraciones, el clasificador con una mayor accuracy fue con 1000 filtros en la primera capa, 500 en la segunda, epoch de 200 y batch size 30. Alcanza una accuracy de 80.59% (± 1.06%), como se muestra en la Figura 34.

```
Model: "sequential_558"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_558 (Flatten)        (None, 18)                0
_________________________________________________________________
dense_1374 (Dense)           (None, 1000)              19000
_________________________________________________________________
dense_1375 (Dense)           (None, 500)               500500
_________________________________________________________________
dense_1376 (Dense)           (None, 13)                6513
=================================================================
Total params: 526,013
Trainable params: 526,013
Non-trainable params: 0
```

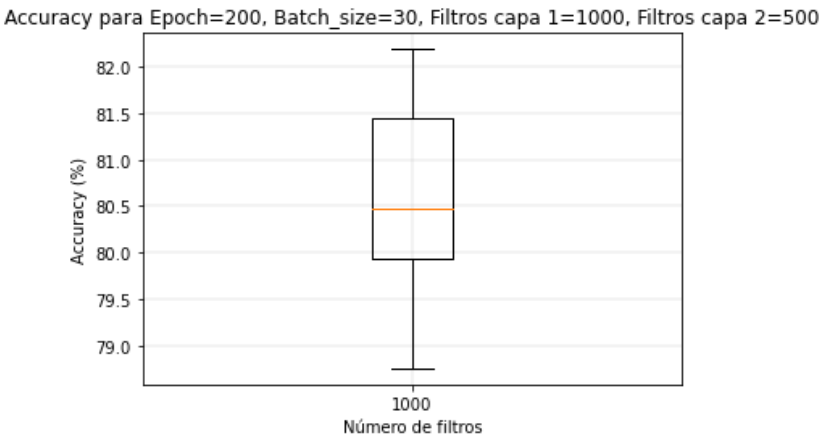

Figura 34. Red neuronal densa con feature engineering. Clasificador con tres capas. Accuracy para mejores hiperparámetros.

La matriz de confusión para el mejor de los casos de esta configuración se muestra en la Figura 35. El error más

común es confundir un golpe VD con B y B con RM, con 5 errores. Es decir, tiene más dificultades para diferenciar la bandeja de la volea de derecha y el remate, ya que la bandeja es un golpe que por sus características se encuentra entre los otros dos, como se explicó en el apartado 4.1.

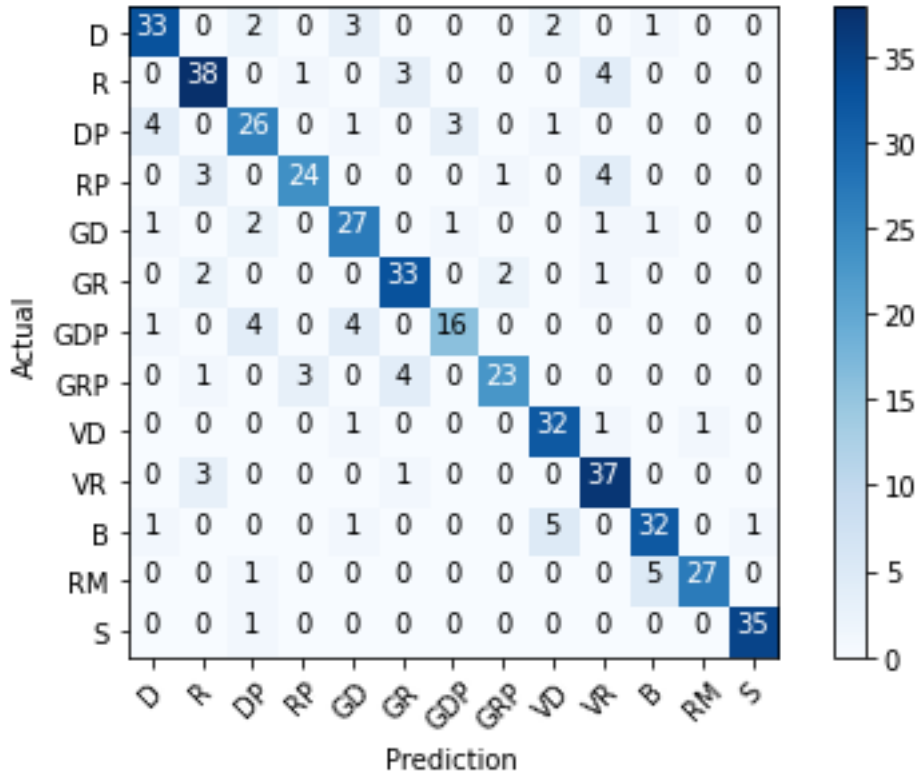

Figura 35. Red neuronal densa con feature engineering. Clasificador con tres capas. Matriz de confusión.

6.1.3 Resumen de resultados de redes neuronales densamente conectadas

En la Figura 36 se muestra un resumen de los resultados obtenidos para los clasificadores de red neuronal densa. Se observa que cuando la entrada es la serie temporal (ST en la figura) la respuesta del clasificador es considerablemente mejor que cuando la entrada es con la técnica feature engineering (FE en la figura). De hecho, el clasificador con feature engineering se sitúa en torno al 80 % de accuracy, mientras que el clasificador con serie temporal se sitúa cercano al 90%. La entrada con serie temporal supone, por tanto, una mejora de un 10 % de la accuracy respecto al clasificador con feature engineering. Para la red neuronal densa, la mejor hiperparametrización es aquella con tres capas, una primera capa con 1000 filtros, seguido de una segunda con 500 y una tercera con 13 (una por clase), con 70 epochs y 30 batch, que logra una accuracy máxima de 92.06%. La media es de 90.44% ($\pm$ 0.57%)

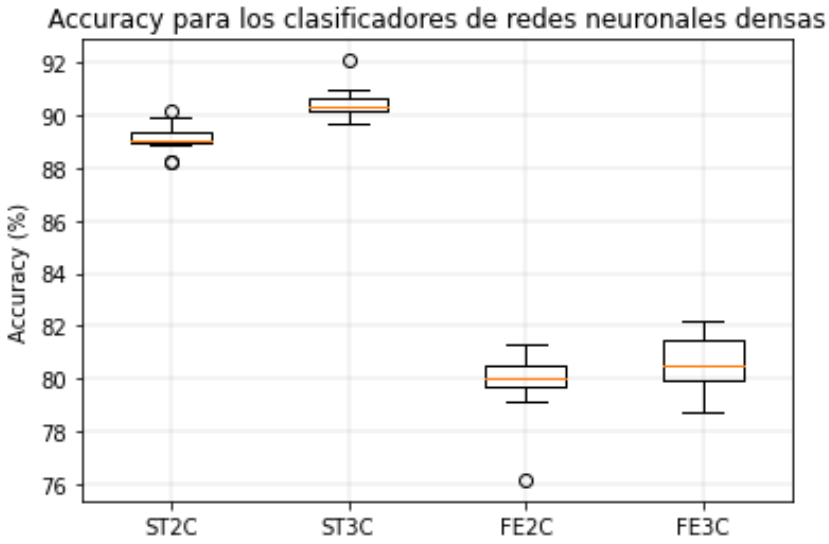

Figura 36. Resumen de la accuracy para los clasificadores de red neuronal densa.

6.2 Redes neuronales convolucionales 1D

Las redes densamente conectadas que han sido expuestas en el apartado anterior pueden ir precedidas de una o varias capas de redes neuronales convolucionales de 1D. Como se vió en el punto 5.3, este tipo de redes hacen la suposición de que los datos de entrada son series temporales por lo que, a priori, es un buen clasificador para el tipo de datos que se están clasificando. Puesto que se espera como entrada una serie temporal, este clasificador no se probará con la técnica feature engineering. Al igual que para el caso de las redes neuronales densas, el conjunto de datos se divide de un 64 % para entrenamiento, un 16 % para validación y un 20 % para testeo. La hiperparametrización se llevará a cabo mediante una búsqueda en rejilla. El código utilizado para este algoritmo se ha basado en el código que se expone en [37].

6.2.1 Clasificador con una capa convolucional y una capa densa

Este clasificador consiste en una primera capa convolucional seguida de una capa densamente conectada de softmax. Los diferentes valores probados en este clasificador para buscar la mejor configuración vienen mostrados en la siguiente tabla:

Tabla 9. Configuraciones CNN1D. Clasificador con una capa convolucional y una capa densa.

Número de filtros de la capa convolucional	[32, 64, 128, 256]
Kernel size	[3, 5, 8]
Epochs	[40, 70]
Batch size	[10, 30, 50, 70]

El clasificador con los mejores hiperparámetros es el siguiente:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 33, 128)           6272
_________________________________________________________________
dropout (Dropout)            (None, 33, 128)           0
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 16, 128)           0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 13)                26637
=================================================================
Total params: 32,909
Trainable params: 32,909
Non-trainable params: 0
```

Tiene una primera capa convolucional de 128 filtros, y un kernel de 8. A continuación, hay una capa *dropout* cuya funcionalidad ya se conoce y una capa *maxpooling*, que consiste en reducir las características recogidas por la capa convolucional para quedarse solo con las más importantes. Por último, tiene una capa *flatten* previa a la capa densamente conectada con 13 filtros y con función de activación softmax. Epoch tiene un valor de 70 y batch size de 50. La accuracy alcanzada es de 88.63% (± 1.24%), como se muestra en la Figura 37.

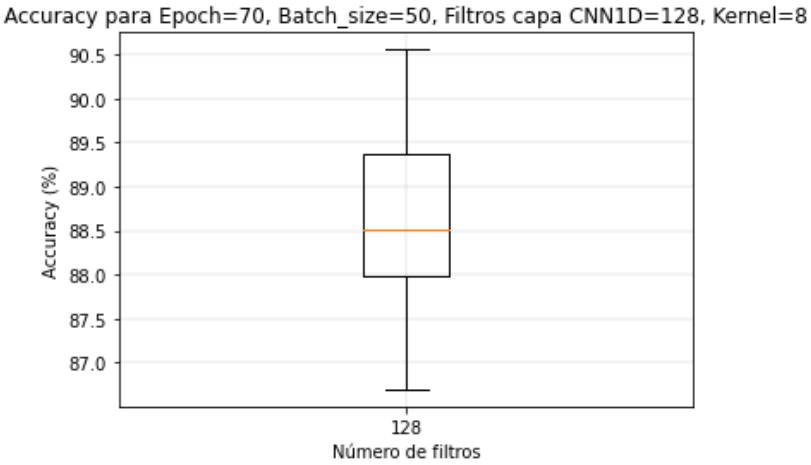

Figura 37. CNN1D. Clasificador con una capa convolucional y una capa densa. Accuracy para mejores hiperparámetros.

La matriz de confusión se muestra en la Figura 38. Se observa que los fallos más comunes son confundir un golpe DP con GDP y un golpe GDP con GD, ambos con 5 errores. El siguiente fallo más común es un golpe GR con R, con 4 errores.

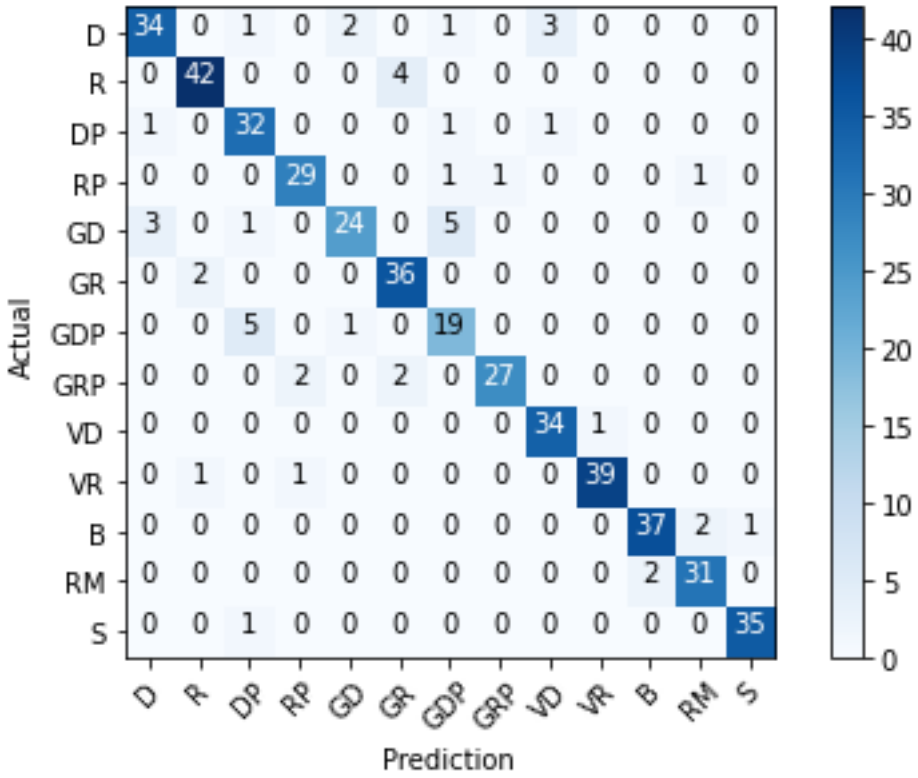

Figura 38. CNN1D. Matriz de confusión para una capa convolucional y una capa densa.

6.2.2 Clasificador con una capa convolucional y dos capas densas

Este clasificador tiene una primera capa convolucional seguida de dos capas densamente conectadas: una capa ReLU y otra final de softmax. Los diferentes valores probados en este clasificador se muestran en la siguiente tabla:

Tabla 10. Configuraciones CNN1D. Clasificador con una capa convolucional y dos capas densas.

Número de filtros de la capa convolucional	[32, 64, 128, 256]
Número de filtros de la primera capa densa	[100, 1000]
Kernel size	[3, 5, 8]
Epochs	[40, 70]
Batch size	[10, 30, 50, 70]

El clasificador con los mejores hiperparámetros es el siguiente:

```
Model: "sequential_441"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_639 (Conv1D)          (None, 33, 256)           12544
_________________________________________________________________
dropout_441 (Dropout)        (None, 33, 256)           0
_________________________________________________________________
max_pooling1d_441 (MaxPoolin (None, 16, 256)           0
_________________________________________________________________
flatten_441 (Flatten)        (None, 4096)              0
_________________________________________________________________
dense_634 (Dense)            (None, 1000)              4097000
_________________________________________________________________
dense_635 (Dense)            (None, 13)                13013
=================================================================
Total params: 4,122,557
Trainable params: 4,122,557
Non-trainable params: 0
```

La primera capa es convolucional de 256 filtros, con un kernel de 8. Le sigue una capa *dropout* y una capa *maxpooling*, cuyas funcionalidades se vieron en el punto anterior. A continuación, tiene una capa *flatten* previa a la capa densamente conectada con 1000 filtros y con función de activación ReLU, y otra capa densa de 13 filtros y con función de activación softmax. El epoch tiene un valor de 70 y el batch size de 70. La accuracy conseguida, como se muestra en la Figura 39, es de 91.28% (± 2.41%).

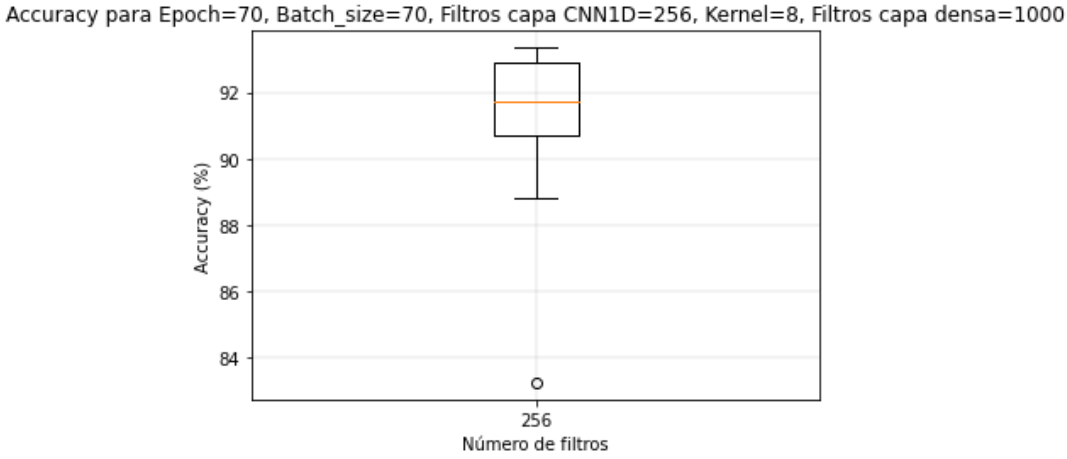

Figura 39. CNN1D. Clasificador con una capa convolucional y dos capas densas. Accuracy para mejores hiperparámetros.

La matriz de confusión se muestra en la Figura 40. Este clasificador supera el 90 % de acierto, lo que se refleja en la matriz de confusión con un gran número de ceros fuera de la diagonal principal. Los fallos más comunes son confundir un golpe D con DP y un golpe DP con GDP, ambos con 3 errores.

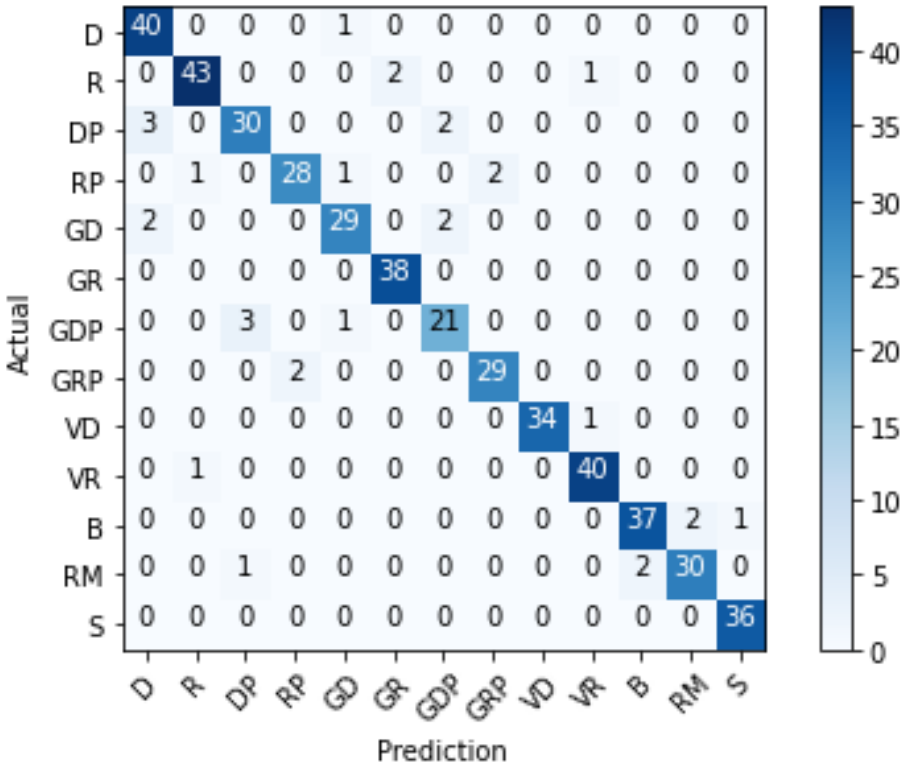

Figura 40. CNN1D. Matriz de confusión para una capa convolucional y dos capas densas.

6.2.3 Clasificador con dos capas convolucionales y una capa densa

En este caso, hay dos capas convolucionales y una última capa densa. Los diferentes valores probados en este clasificador se muestran en la siguiente tabla:

Tabla 11. Configuraciones CNN1D. Clasificador con dos capas convolucionales y una capa densa.

Número de filtros de la primera capa convolucional	[32, 64, 128, 256]
Kernel size	[3, 5, 8]
Epochs	[40, 70]
Batch size	[10, 30, 50, 70]

El clasificador con los mejores hiperparámetros es el siguiente:

```
Model: "sequential_405"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_601 (Conv1D)          (None, 33, 256)           12544
_________________________________________________________________
conv1d_602 (Conv1D)          (None, 26, 128)           262272
_________________________________________________________________
```

```
dropout_405 (Dropout)        (None, 26, 128)           0
_________________________________________________________________
max_pooling1d_405 (MaxPoolin (None, 13, 128)           0
_________________________________________________________________
flatten_405 (Flatten)        (None, 1664)              0
_________________________________________________________________
dense_564 (Dense)            (None, 13)                21645
=================================================================
Total params: 296,461
Trainable params: 296,461
Non-trainable params: 0
```

La primera capa convolucional cuenta con 256 filtros, mientras que la segunda tiene 128, ambas con un kernel de 8. Todos los ejemplos probados contaban con la mitad de los filtros en la segunda capa convolucional que en la primera. Les siguen una capa de *dropout* y otra de *maxpooling*, ya conocidas. A continuación, hay una capa *flatten* previa a la última capa, una densa de 13 filtros y con función de activación softmax. El epoch tiene un valor de 70 y el batch size de 70. La accuracy media conseguida es de 91.08% (± 1.01%), como se muestra en la Figura 41.

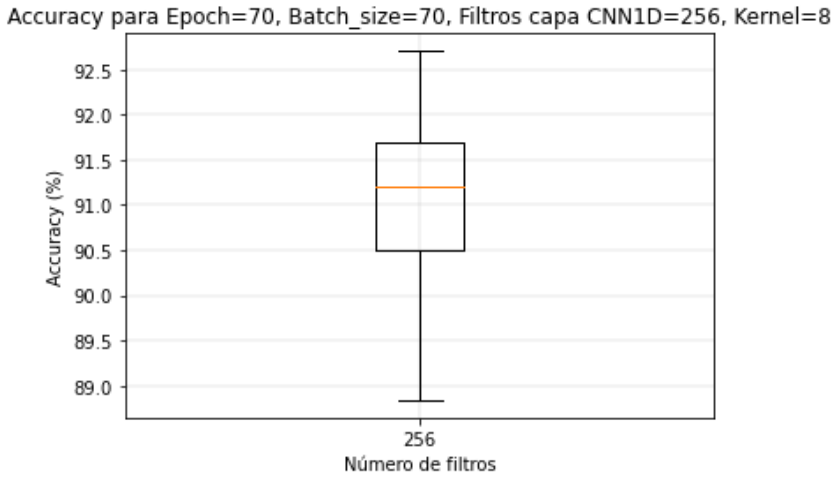

Figura 41. CNN1D. Clasificador con dos capas convolucionales y una capa densa. Accuracy para mejores hiperparámetros.

La matriz de confusión se muestra en la Figura 42. De nuevo se obtiene un gran número de ceros fuera de la diagonal principal. Los errores más comunes en este caso se corresponden con confundir un golpe DP con GDP, un golpe GDP con GD y un golpe B con RM, todos con 3 predicciones erróneas.

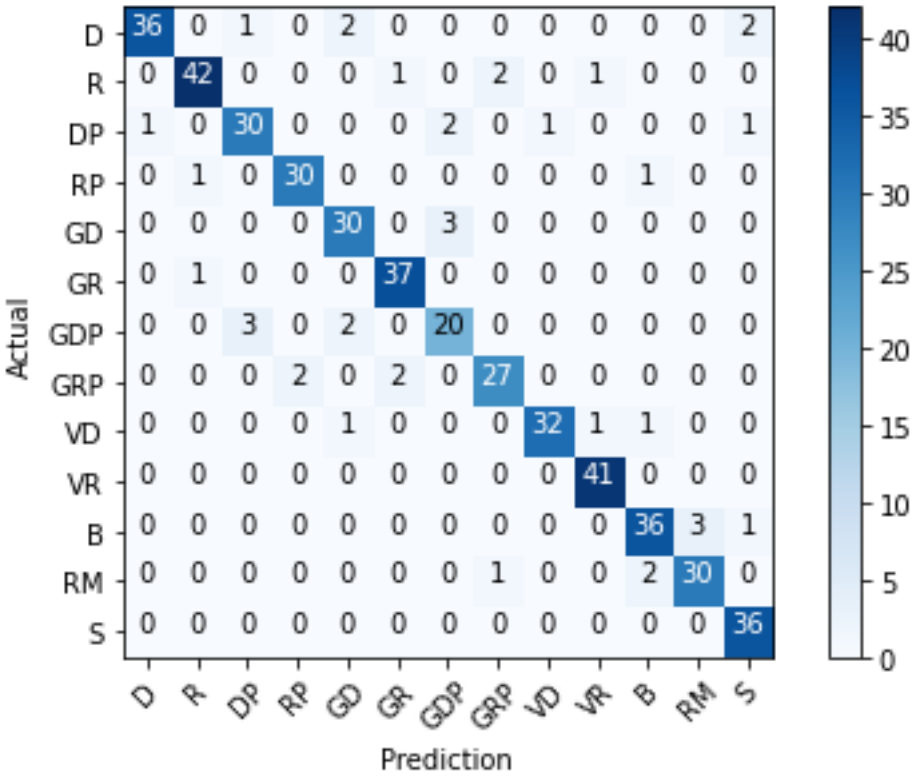

Figura 42. CNN1D. Matriz de confusión para dos capas convolucionales y una capa densa.

6.2.4 Clasificador con dos capas convolucionales y dos capas densas

Este clasificador cuenta con dos capas convolucionales y dos capas densas: una con función de activación ReLU y otra softmax. Los diferentes valores para las diferentes configuraciones se muestran en la siguiente tabla:

Tabla 12. Configuraciones CNN1D. Clasificador con dos capas convolucionales y dos capas densas.

Número de filtros de la primera capa convolucional	[32, 64, 128, 256]
Número de filtros de la primera capa densa	[100, 1000]
Kernel size	[3, 5, 8]
Epochs	[40, 70]
Batch size	[10, 30, 50, 70]

El clasificador con los mejores hiperparámetros es el siguiente:

```
Model: "sequential_50"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
conv1d_100 (Conv1D)          (None, 36, 128)           3968
_________________________________________________________________
conv1d_101 (Conv1D)          (None, 32, 64)            41024
_________________________________________________________________
dropout_50 (Dropout)         (None, 32, 64)            0
_________________________________________________________________
max_pooling1d_50 (MaxPooling (None, 16, 64)            0
_________________________________________________________________
flatten_50 (Flatten)         (None, 1024)              0
_________________________________________________________________
dense_100 (Dense)            (None, 1000)              1025000
_________________________________________________________________
dense_101 (Dense)            (None, 13)                13013
=================================================================
Total params: 1,083,005
Trainable params: 1,083,005
Non-trainable params: 0
```

La primera capa convolucional cuenta con 128 filtros, mientras que la segunda tiene 64, ambas con un kernel de 5. Todos los ejemplos probados contaban con la mitad de los filtros en la segunda capa convolucional que en la primera. Les siguen una capa de *dropout* y otra de *maxpooling*, ya conocidas. A continuación, hay una capa *flatten* previa a las capas densas. La primera de ellas cuenta con 1000 filtros y con función de activación ReLU, mientras que la segunda y última capa densa tiene 13 filtros y función de activación softmax. El epoch tiene un valor de 70 y el batch size de 70. La accuracy conseguida es de 90.32% (± 1.28%), como se muestra en la Figura 43.

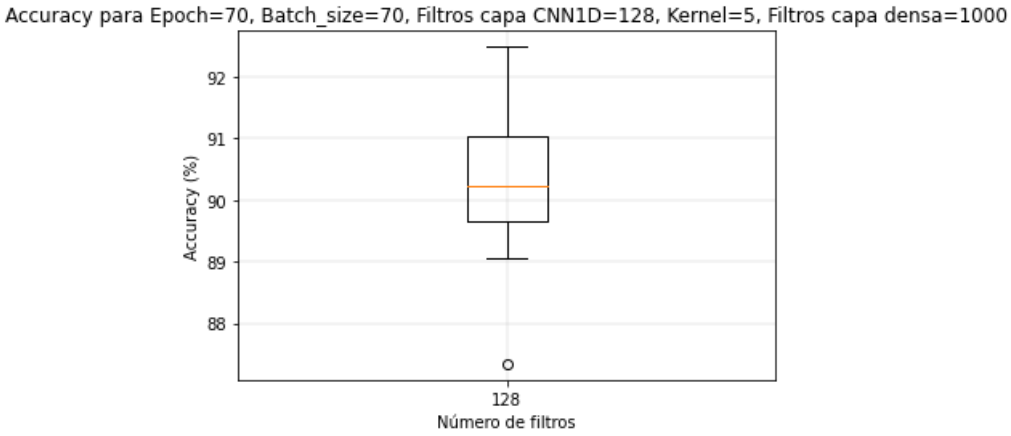

Figura 43. CNN1D. Clasificador con dos capas convolucionales y dos capas densas. Accuracy para mejores hiperparámetros.

La matriz de confusión se muestra en la Figura 44. Los errores más comunes se corresponden a confundir un golpe RM con B, con 4 errores. Le sigue confundir un golpe D con GD y un golpe GDP con GD, ambos con 3 errores.

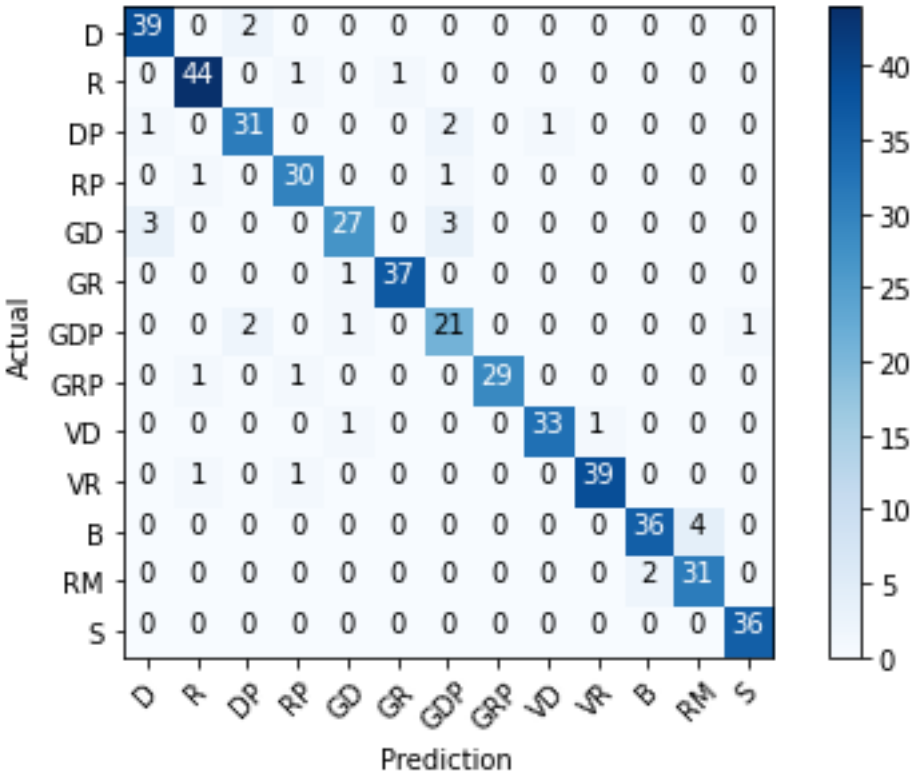

Figura 44. CNN1D. Matriz de confusión para dos capas convolucionales y dos capas densas.

6.2.5 Resumen de resultados de redes neuronales convolucionales 1D

En la Figura 45 se muestra un resumen de los resultados obtenidos para los clasificadores CNN1D. La configuración que alcanza una mayor accuracy es la de una capa convolucional y dos capas densas (1C+2D en la figura), logrando un 93.35 % para su mayor accuracy, con una media de 91.28 %. No obstante, la dispersión de este modelo es alta (± 2.41%). Por otro lado, la configuración 2C+1D reúne una alta accuracy con una menor dispersión, con un 91.08% (± 1.01%). Cabe destacar que las 4 opciones representadas están muy cerca entre sí, alrededor del 90% en todos los casos, que supone una respuesta considerablemente correcta de los modelos.

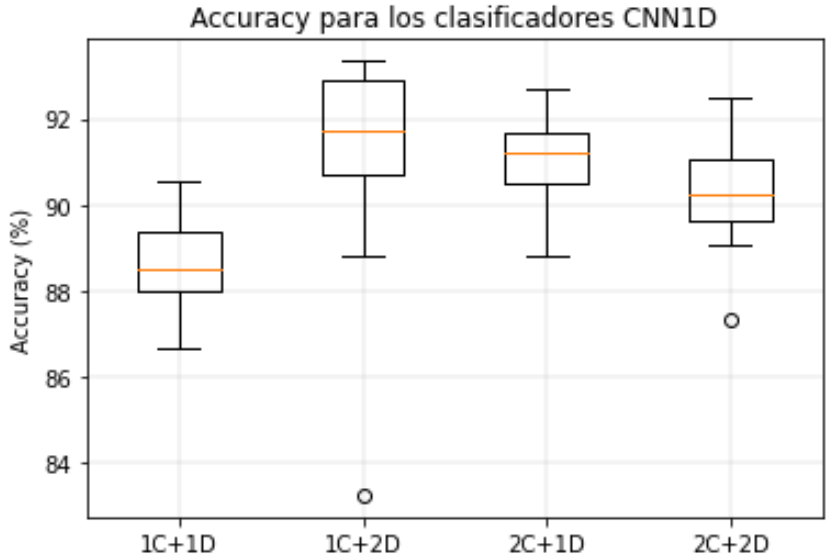

Figura 45. Resumen de la accuracy para los clasificadores CNN1D.

6.3 Árbol de decisión

6.3.1 Árbol de decisión con serie temporal

Para crear un clasificador mediante un árbol de decisión, existe un conjunto de hiperparámetros que se pueden modificar para adaptar la red y obtener así un árbol de decisión óptimo para cada caso. Para encontrar la combinación óptima, se llevará a cabo una búsqueda de rejilla con los valores mostrados en la Tabla 13. La entrada para este clasificador será la serie temporal de cada golpe, y la división realizada del conjunto de datos se corresponde a un 70 % para entrenamiento y un 30 % para testeo.

Tabla 13. Configuraciones árbol de decisión con serie temporal.

Max_depth	[1, 10, 20, 30, 40]
Min_samples_split	[2, 4, 8, 10, 20, 100]
Min_samples_leaf	[1, 2, 3, 4, 5, 6, 10]
Criterion	[entropy, gini]

La mejor configuración de hiperparámetros es la que tiene una profundidad del árbol de 40 (*max_depth*), 4 muestras como mínimo para dividir un nodo (*min_samples_split*), una muestra por hoja como mínimo (*min_samples_leaf*) y función de impureza *entropy*. El clasificador obtiene una accuracy de 60.59% (± 1.06%).

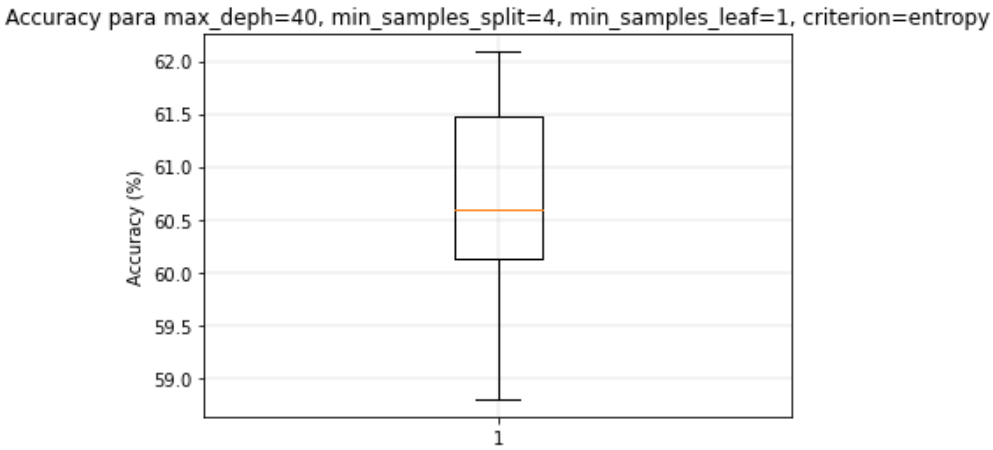

Figura 46. Árbol de decisión con serie temporal. Accuracy para mejores hiperparámetros.

La matriz de confusión para el mejor de los casos se muestra en la Figura 47. En este caso, los errores más comunes son confundir un golpe DP con D, con 18 fallos. Con 12 fallos se encuentran R con RP y R con VR.

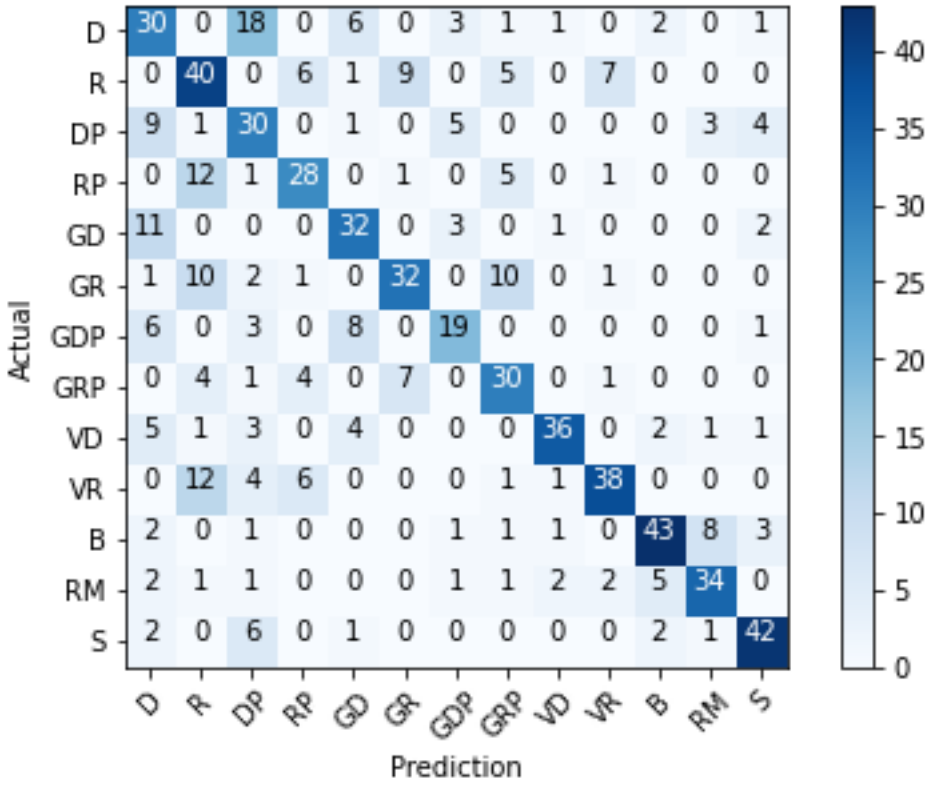

Figura 47. Árbol de decisión con serie temporal. Matriz de confusión.

6.3.2 Árbol de decisión con feature engineering

En este caso, la entrada al clasificador está determinada por la técnica feature engineering. Como se comentó anteriormente, se tendrá como entrada el valor máximo, mínimo y medio de la serie temporal de cada grado de libertad de cada golpe. La división realizada del conjunto de datos se corresponde a un 70 % para entrenamiento y un 30 % para testeo. Para encontrar la configuración óptima para el clasificador, se lleva a cabo una búsqueda en rejilla con los valores de la Tabla 14.

Tabla 14. Configuraciones árbol de decisión con feature engineering.

Max_depth	[1, 10, 20, 30, 40]
Min_samples_split	[2, 4, 8, 10, 20, 100]
Min_samples_leaf	[1, 2, 3, 4, 5, 6, 10]
Criterion	[entropy, gini]

La mejor configuración de hiperparámetros es la que tiene una profundidad del árbol de 10 (*max_depth*), 2 muestras como mínimo para dividir un nodo (*min_samples_split*), una muestra por hoja como mínimo (*min_samples_leaf*) y función de impureza *entropy*. El clasificador obtiene una accuracy de 60.76% (± 0.90%).

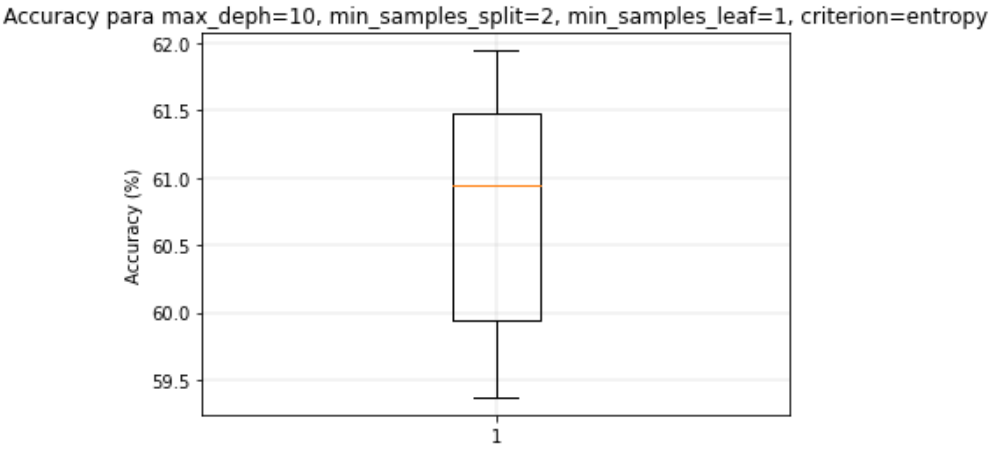

Figura 48. Árbol de decisión con feature engineering. Accuracy para mejores hiperparámetros.

La matriz de confusión para el mejor de los casos se muestra en la Figura 49. Para este caso, los errores más comunes se cometen al confundir un golpe R con RP y B con RM, ambos con 13 fallos. Les sigue R con VR, con 12 fallos.

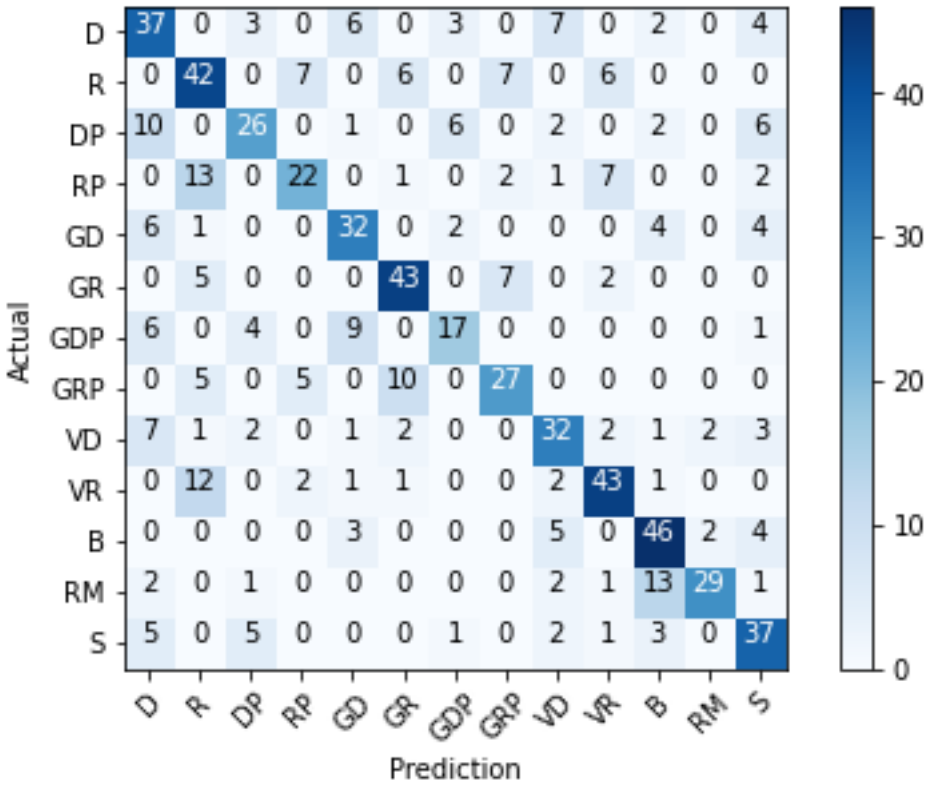

Figura 49. Árbol de decisión con feature engineering. Matriz de confusión.

6.3.3 Resumen de resultados de árbol de decisión

En la Figura 50 se muestra un resumen de los resultados obtenidos para los clasificadores de árbol de decisión. El clasificador con serie temporal obtiene una accuracy media de 60.59 %, frente a un 60.76 % del clasificador con feature engineering. Ambos valores medios están muy próximos, sin embargo, el clasificador con feature engineering tiene una menor dispersión. En cuanto al valor máximo, la entrada con la serie temporal logra una

mayor accuracy, alcanzando el 62.09 %.

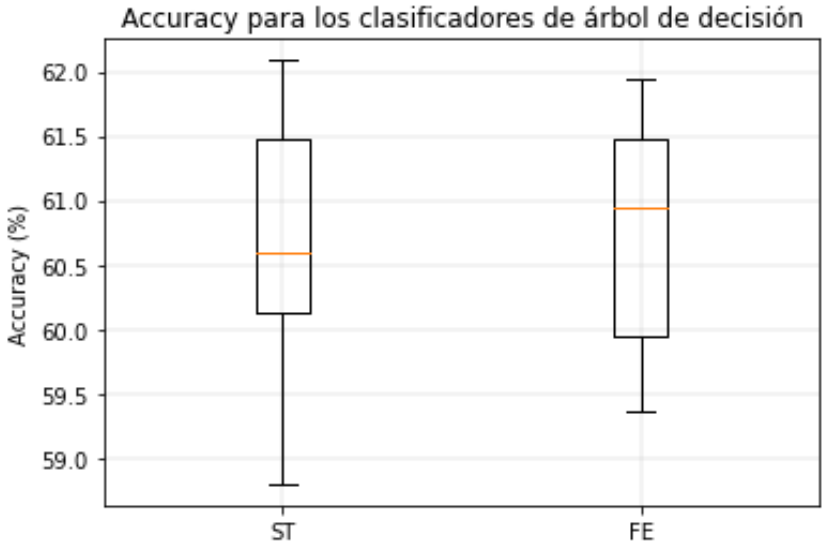

Figura 50. Resumen de la accuracy para los clasificadores de árbol de decisión.

6.4 K Vecinos más próximos (KNN)

6.4.1 KNN con serie temporal

En el punto 5.5 se vio que este clasificador tiene una variable k que controla el número de vecinos más próximos que se tienen en cuenta para la clasificación. En este caso, se estudiará el comportamiento del algoritmo KNN con los datos de la serie temporal. Se ha probado con diferentes valores para la variable k, incluyendo los valores pares puesto que no se trata de una clasificación binaria. Cabe recordar que, en caso de empate, Scikit Learn escoge el valor que aparezca antes en el conjunto de vecinos. Los resultados se resumen en la Tabla 15.

Tabla 15. Algoritmo KNN con serie temporal. Accuracy en función de k.

k	1	2	3	4	5	6	7	8	9
Accuracy (%)	84.12	79.97	80.97	79.26	78.83	78.11	78.25	77.11	76.25
k	10	11	12	13	14	15	20	25	30
Accuracy (%)	74.68	75.39	74.25	74.68	72.96	73.10	68.67	68.38	63.66

Se observa que la mayor accuracy se obtiene para $k = 1$, donde alcanza un 84.12%. Este es un caso especial dentro del algoritmo KNN, ya que solo considera el vecino más próximo en lugar de un conjunto de ellos. Para el resto de los valores, generalmente la accuracy desciende al aumentar el valor de k. Un diagrama de bigotes con los resultados de la tabla anterior se muestra en la Figura 51.

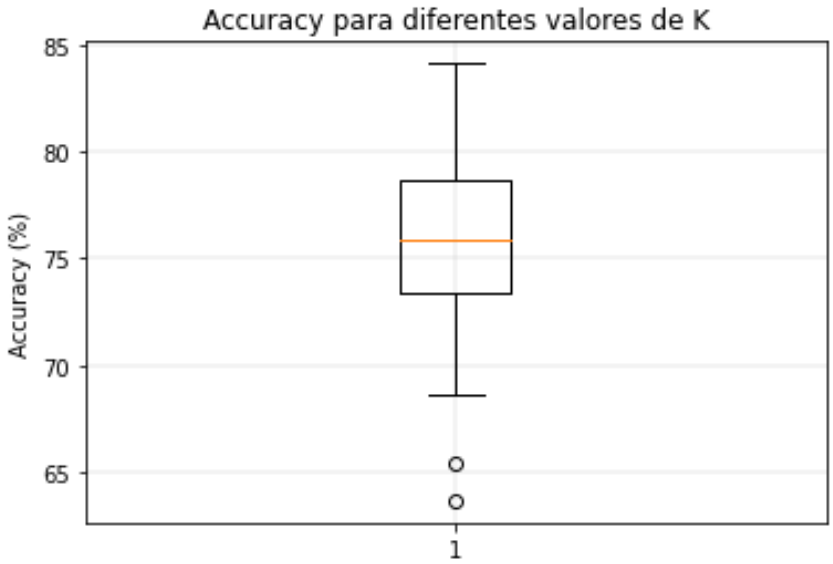

Figura 51. Algoritmo KNN con serie temporal. Diagrama de bigotes en función de *k*.

La matriz de confusión para $k = 1$ se muestra en la Figura 52. Para este caso, los errores más comunes es confundir un golpe GDP con DP, con 8 fallos, seguido de un golpe DP con D, GR con R y GRP con RP, estos últimos con 7 errores. Se observa que de nuevo los fallos más comunes se corresponden con los golpes más similares.

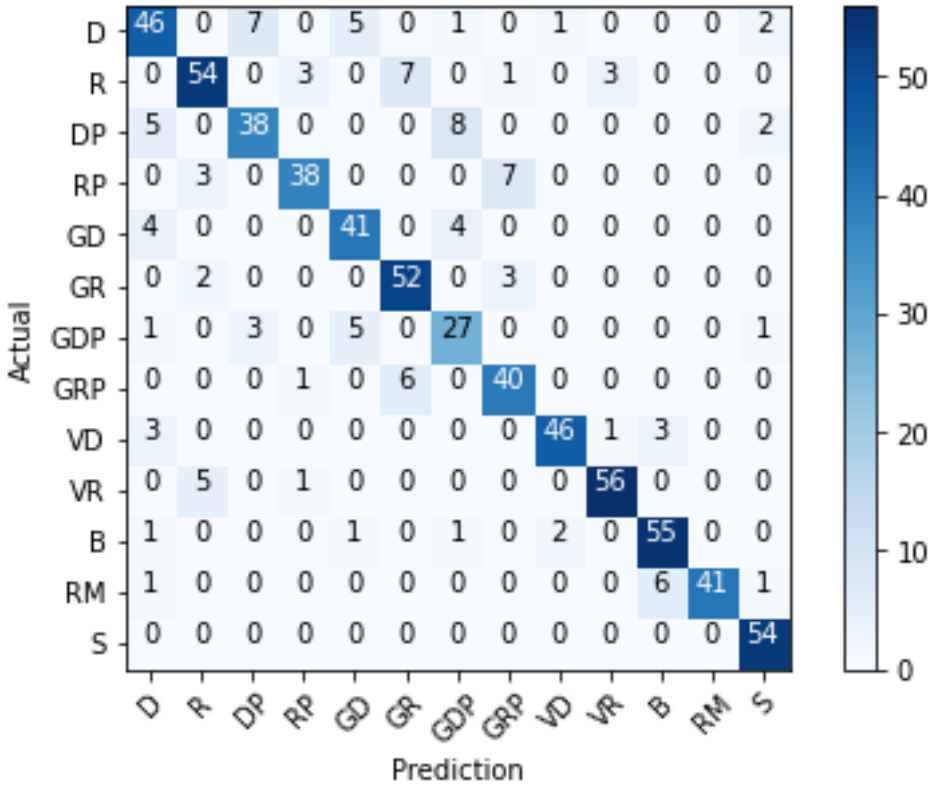

Figura 52. Algoritmo KNN con serie temporal. Matriz de confusión para $k = 1$.

Cabe destacar que los resultados de este algoritmo no tienen dispersión ya que el conjunto de datos se dividió en entrenamiento y testeo de una forma aleatoria pero fija (para que la división fuese siempre la misma). De esta forma, aún realizando infinitas veces la predicción, los resultados siempre serían los mismos, ya que se

estudiarían las mismas distancias en cada iteración. Si, por el contrario, se realizan divisiones distintas del conjunto de datos para la mejor configuración (serie temporal con $k = 1$), se obtiene una accuracy de 83.36% ($\pm$ 1.08%), como se muestra en la Figura 53.

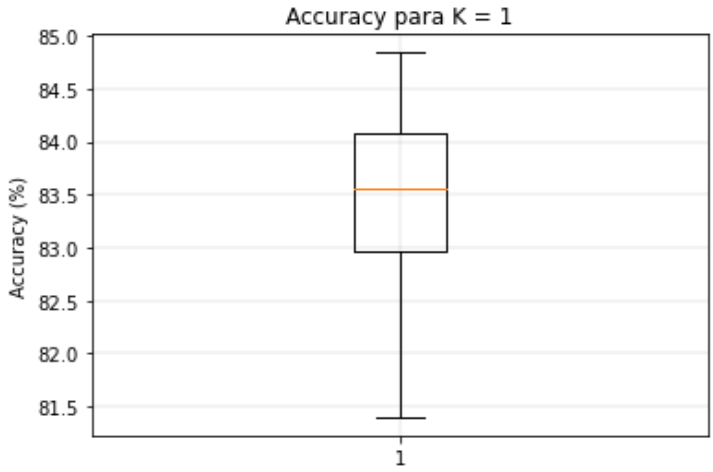

Figura 53. Algoritmo KNN con serie temporal. Accuracy para $k = 1$.

6.4.2 KNN con feature engineering

A diferencia del apartado anterior, el algoritmo no recibirá como entrada la serie temporal, sino los valores máximos, mínimos y medios de la serie temporal de cada grado de libertad. Se ha realizado la clasificación con diferentes valores para la variable *k*, obteniéndose los siguientes resultados.

Tabla 16. Algoritmo KNN con feature engineering. Accuracy en función de *k*.

k	1	2	3	4	5	6	7	8	9
Accuracy (%)	68.24	64.95	65.38	64.66	64.38	64.81	64.52	63.52	63.66
k	10	11	12	13	14	15	20	25	30
Accuracy (%)	63.52	63.23	63.23	63.52	61.52	62.37	60.66	58.80	58.94

Para este clasificador, al igual que pasaba con el clasificador para la serie temporal como entrada, la accuracy máxima se obtiene para $k = 1$, aunque en este caso obtiene solo un 68.24 %. Un diagrama de bigotes con los resultados de la tabla anterior se muestra en la Figura 54.

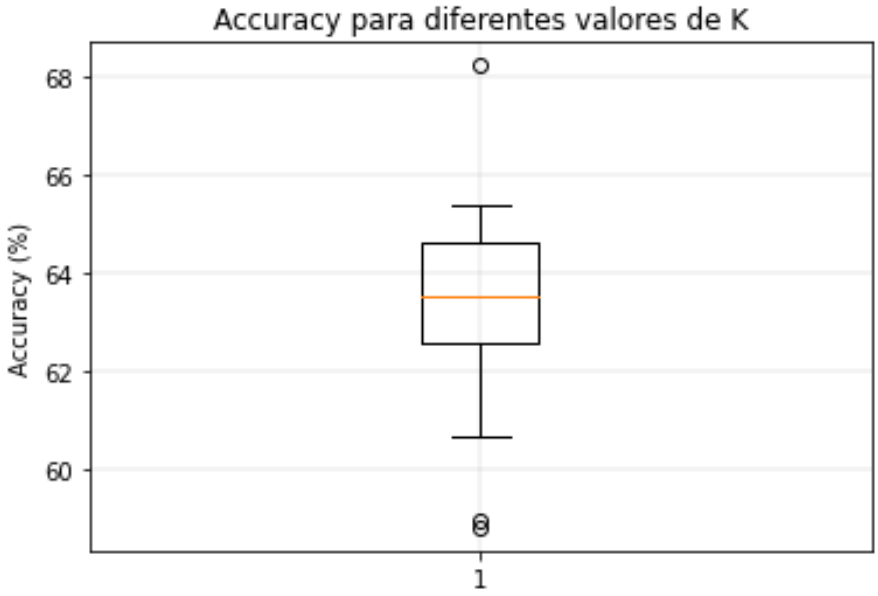

Figura 54. Algoritmo KNN con feature engineering. Diagrama de bigotes en función de k.

La matriz de confusión para $k = 1$ se muestra en la Figura 55. Para este caso, los errores más comunes tienen lugar con 10 fallos, entre los que se encuentra confundir un golpe D con DP, RP con R, y S con DP.

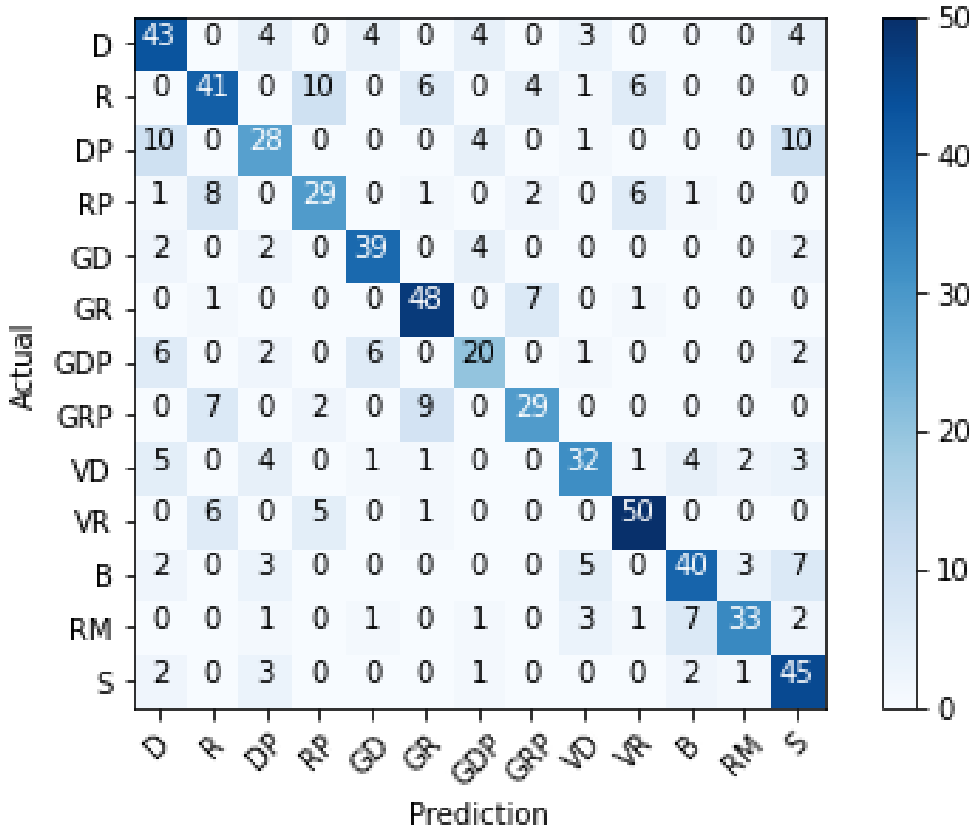

Figura 55. Algoritmo KNN con feature engineering. Matriz de confusión para $k = 1$.

Como se comentó en el apartado anterior, los resultados no tienen dispersión porque se han realizado con una división fija de los datos. Ejecutando repetidas veces el algoritmo con diferentes divisiones, se obtiene un

66.64% ($\pm$ 0.92%) de accuracy, como se muestra en la Figura 56.

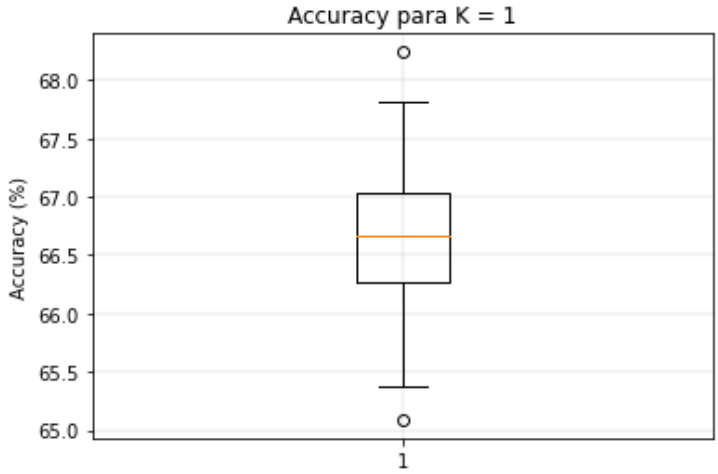

Figura 56. Algoritmo KNN con feature engineering. Accuracy para $k = 1$.

6.4.3 Resumen de resultados KNN

En la Figura 57 se muestra un resumen de los resultados del algoritmo KNN para la serie temporal y feature engineering, ambos con $k = 1$. Se observa que la serie temporal tiene mejores resultados, alcanzando un 83.63% de media. En el caso de la entrada con feature engineering, se queda en una media de 66.64%. En ambos casos, el mejor valor de k es la unidad, es decir, se realiza una mejor clasificación asignando la clase del vecino más próximo. El valor máximo de nuevo se obtiene para la serie temporal como entrada, alcanzando un 84.84%.

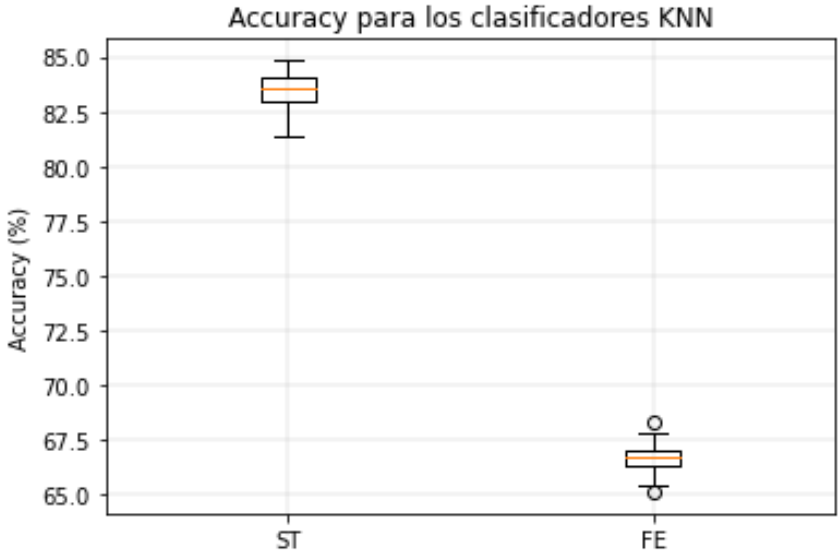

Figura 57. Algoritmo KNN. Resultados de la serie temporal vs feature engineering para $k = 1$.

6.5 Máquinas de vector soporte (SVM)

6.5.1 SVM con serie temporal

Las máquinas de vector soporte, como se vió en el apartado 5.6, cuentan una serie de hiperparámetros que se

pueden modificar para ajustar correctamente el modelo y conseguir una mejor respuesta. En este caso, se ha estudiado la accuracy obtenida en función del filtro kernel y la regularización. No se mostrarán los resultados en función de los métodos one-versus-one y one-versus-all, puesto que no muestran diferencias entre ambos. Los resultados obtenidos para el SVM con serie temporal se resumen en la Tabla 17.

Tabla 17. Algoritmo SVM con serie temporal. Accuracy (%) en función de kernel y C.

		Regularización (C)								
		0.01	0.1	0.5	1	2	10	12	20	100
Filtro Kernel	Lineal	81.40	79.83	80.40	80.40	80.40	80.40	80.40	80.40	80.40
	Polinómico	17.45	56.22	76.39	82.69	85.69	88.27	88.13	88.13	88.13
	Radial	17.17	55.94	79.40	85.84	86.98	90.41	90.27	89.99	90.41
	Sigmoide	18.60	41.63	62.80	65.09	62.66	55.22	53.79	54.51	52.22

Merece la pena recordar que el valor de C es inversamente proporcional a la fuerza de la regularización; es decir, cuanto mayor sea el valor de C, menor será la regularización, por lo que se generaliza menos el modelo.

Puede observarse que la mayor accuracy (90.41%) se obtiene para un kernel radial con regularización $C = 10$ y $C = 100$. Aunque con $C = 10$ y 100 se alcanza la misma accuracy, no se obtiene la misma matriz de confusión, ya que los fallos que cometen no son exactamente los mismos. Sin embargo, son muy parecidos ya que se cometen con los mismos golpes y en cantidades casi exactas, por lo que se mostrará la matriz de confusión para un solo caso. La matriz de confusión para $C = 10$ se muestra en la Figura 59. El filtro kernel polinómico se acerca bastante, consiguiendo 88.27 % para $C = 10$. El kernel lineal se mantiene prácticamente invariante a la regularización, cercano al 80 %. El kernel sigmoide tiene su mejor respuesta para una regularización unitaria, donde logra superar el 65 %. En la Figura 58 se muestra un diagrama de bigotes para los resultados de la tabla anterior, donde se puede constatar visualmente la importancia de una correcta hiperparametrización: en función de la hiperparametrización se obtiene una accuracy entre un 17.17% y un 90.41%, es decir, una diferencia de más de un 73% entre la mejor y la peor hiperparametrización.

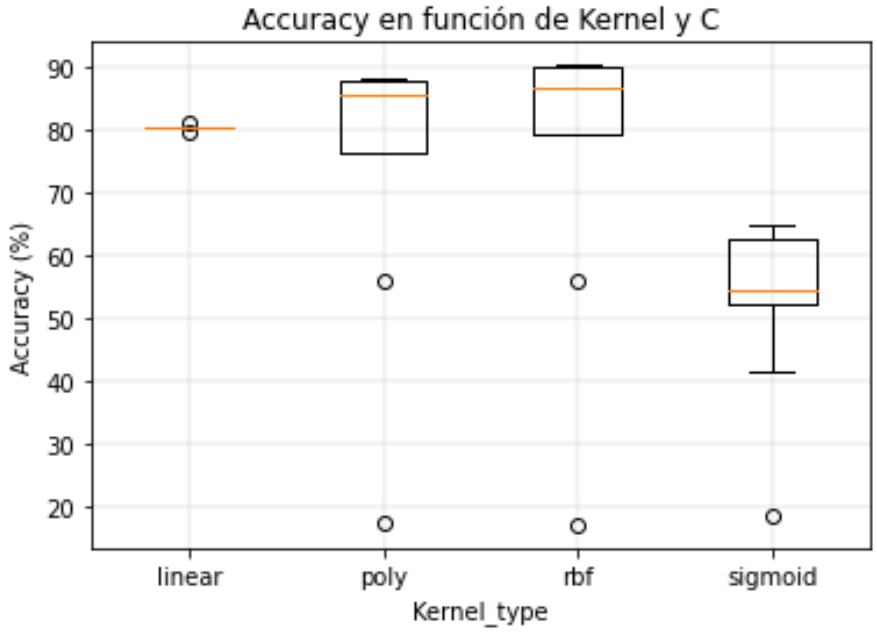

Figura 58. Algoritmo SVM con serie temporal. Diagrama de bigotes en función de kernel y C.

La matriz de confusión se muestra en la Figura 59. Vuelve a ocurrir que los errores más comunes tienen lugar entre aquellos golpes con dinámica similar. El error más común para este caso es confundir un golpe GDP con GD, con 10 fallos. Con 6 errores se encuentra confundir un golpe GR con R y B con RM.

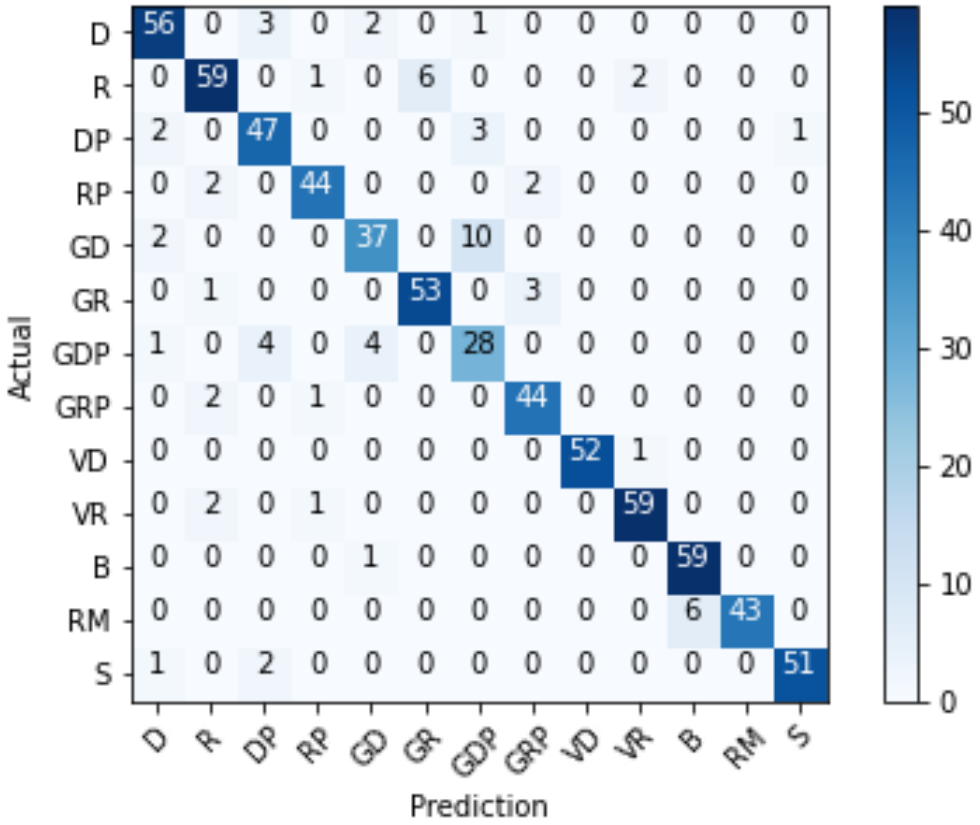

Figura 59. Algoritmo SVM con serie temporal. Matriz de confusión para kernel radial con $C = 10$.

Como ocurría en anteriormente, estos resultados no tienen dispersión ya que la división del conjunto de datos se dividió en entrenamiento y testeo de una forma aleatoria pero fija. Si se realizan divisiones distintas del conjunto de datos para la mejor configuración (serie temporal con kernel radial y $C = 10$), se obtiene una accuracy de 90.52% ($\pm$ 1.24%), como se muestra en la Figura 60.

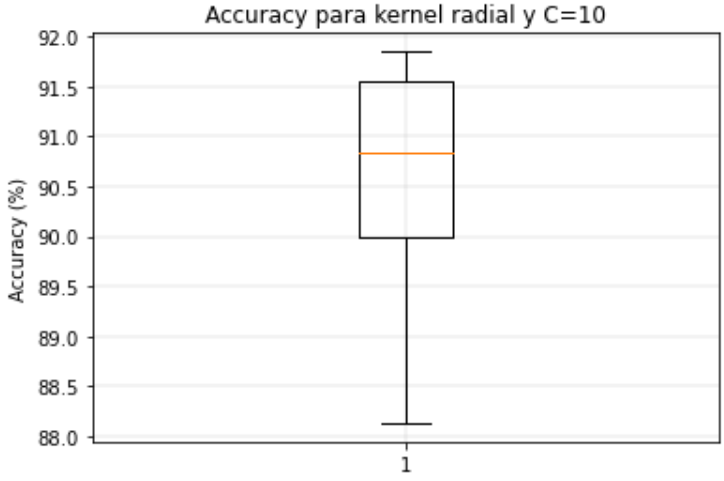

Figura 60. Algoritmo SVM con serie temporal. Accuracy para kernel radial y $C = 10$.

6.5.2 SVM con feature engineering

En este apartado se estudiarán los resultados para el SVM con la técnica feature engineering, que se resumen en la Tabla 18.

Tabla 18. Algoritmo SVM con feature engineering. Accuracy (%) en función de kernel y C.

		Regularización (C)								
		0.01	0.1	1	10	100	200	300	500	1000
Filtro Kernel	Lineal	62.66	68.81	70.82	69.96	69.67	70.10	70.10	69.38	69.53
	Polinómico	23.46	44.64	61.52	68.96	73.82	72.82	73.24	72.53	71.39
	Radial	9.73	40.77	61.37	68.53	74.25	73.96	72.82	72.96	71.24
	Sigmoide	9.73	18.45	31.62	29.47	23.46	23.75	22.89	22.75	24.18

Puede observarse que la mayor accuracy (74.25 %) se obtiene para un kernel radial con regularización $C = 100$. No obstante, al igual que ocurría para el SVM con serie temporal, se obtienen resultados muy similares con el filtro radial y polinómico. En este caso, el kernel radial consigue un 73.82% para $C = 100$. El kernel lineal es el más robusto respecto a la regularización, como ya ocurría en el SVM con serie temporal, superando el 70 %. El kernel sigmoide es de nuevo el que peores resultados obtiene, con un 31.62 % para una regularización también unitaria. En la Figura 61 se muestra un diagrama de bigotes para los resultados de la tabla anterior.

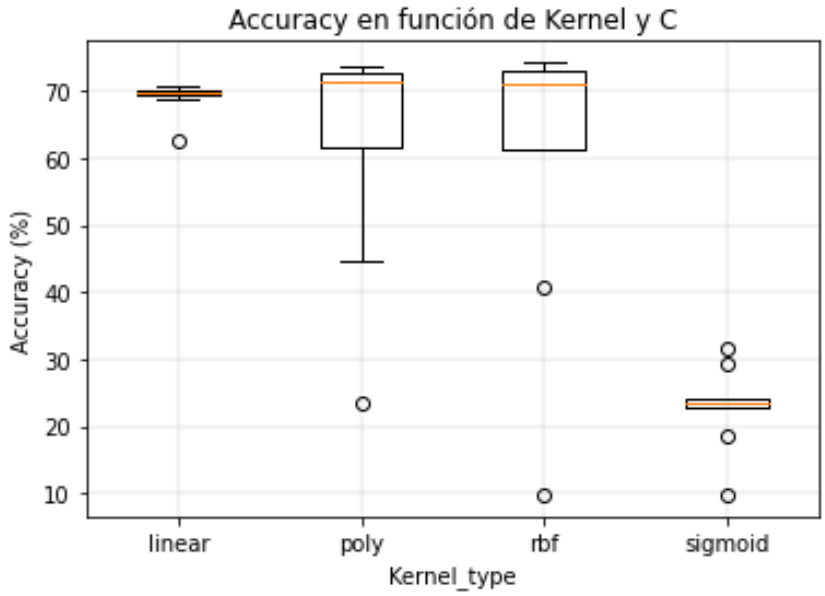

Figura 61. Algoritmo SVM con feature engineering. Diagrama de bigotes en función de kernel y C.

La matriz de confusión para esta configuración se muestra en la Figura 62. Para este caso, el error más común es confundir un golpe R con VR, con 13 fallos, seguido de GD con GDP, con 11 fallos. Se puede apreciar aquí que un descenso notable de la accuracy se corresponde con un aumento del número de fallos cometido por el algoritmo, como era de esperar. Lo que no cambia es el hecho de confundir con más frecuencia aquellos golpes que son más parecidos. No obstante, en este caso que tiene más fallos en la predicción, se aprecia más

confusiones en golpes que no son parecidos. Por ejemplo, se obtienen 3 fallos en golpe de S con B. Es evidente que estos dos golpes tienen muchas más diferencias que similitudes, pero es un error que, a pesar de existir, no es dominante, ya que es de los fallos menos comunes del clasificador.

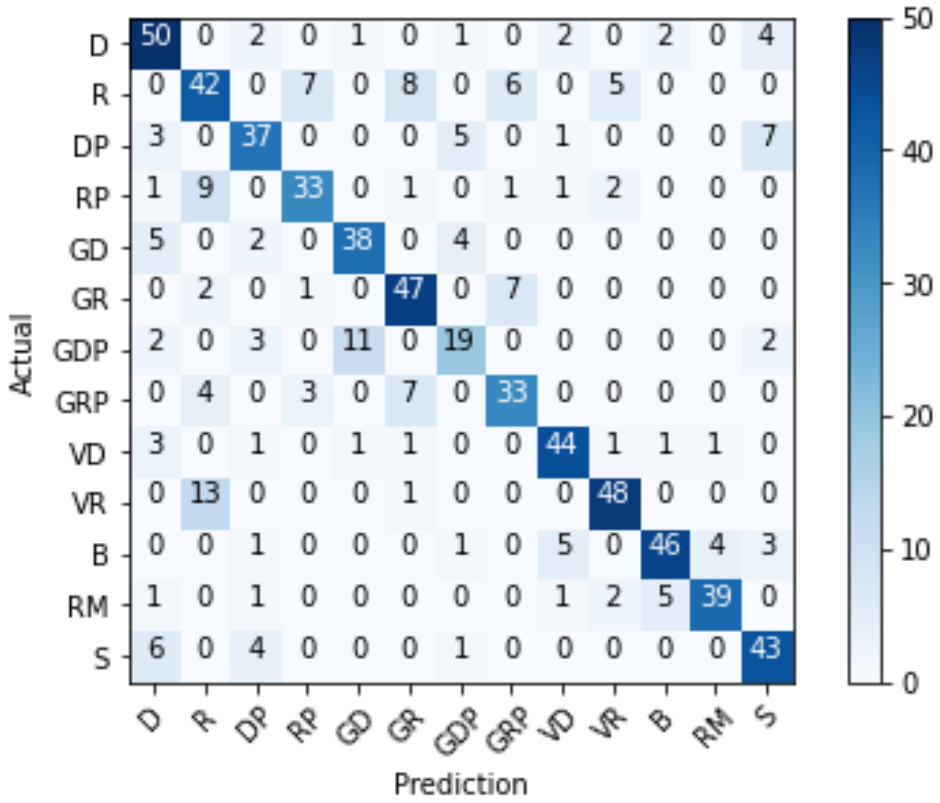

Figura 62. Algoritmo SVM con feature engineering. Matriz de confusión para kernel radial con $C = 100$.

Si se realiza una división diferente del conjunto de datos en cada ejecución del algoritmo, se alcanza una accuracy de 74.46% ($\pm$ 1.60%), representado en la Figura 63.

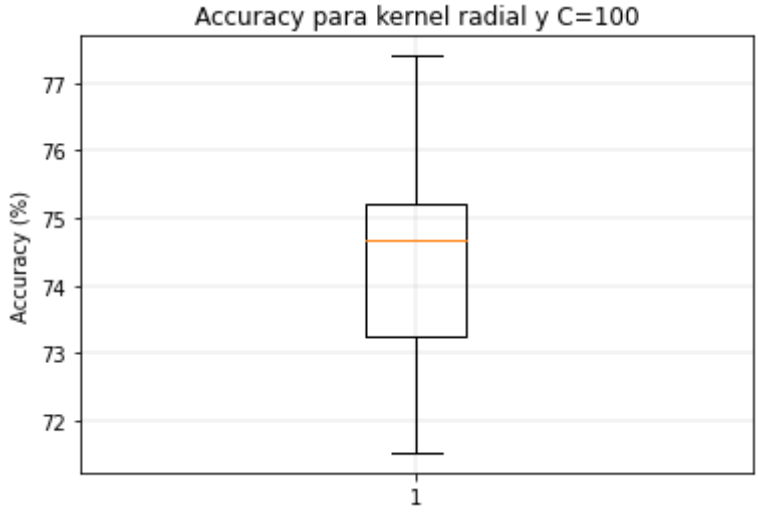

Figura 63. Algoritmo SVM con feature engineering. Accuracy para kernel radial y $C = 100$.

6.5.3 Resumen de resultados SVM

En la Figura 64 se muestra una recopilación de los resultados obtenidos para los clasificadores de SVM para diferentes valores de kernel y C. En primer lugar, se aprecia que cuando la entrada es con la serie temporal del golpe los resultados son considerablemente mejores que con feature engineering como entrada. Por otro lado, se observa que los diferentes kernel tienen comportamientos similares independientemente del tipo de entrada: el lineal es poco sensible a la regularización, el radial es el que obtiene mejores resultados seguido muy de cerca del polinómico, y el sigmoid siempre obtiene los peores resultados. La mejor configuración para el SVM sería un kernel radial con $C = 10$.

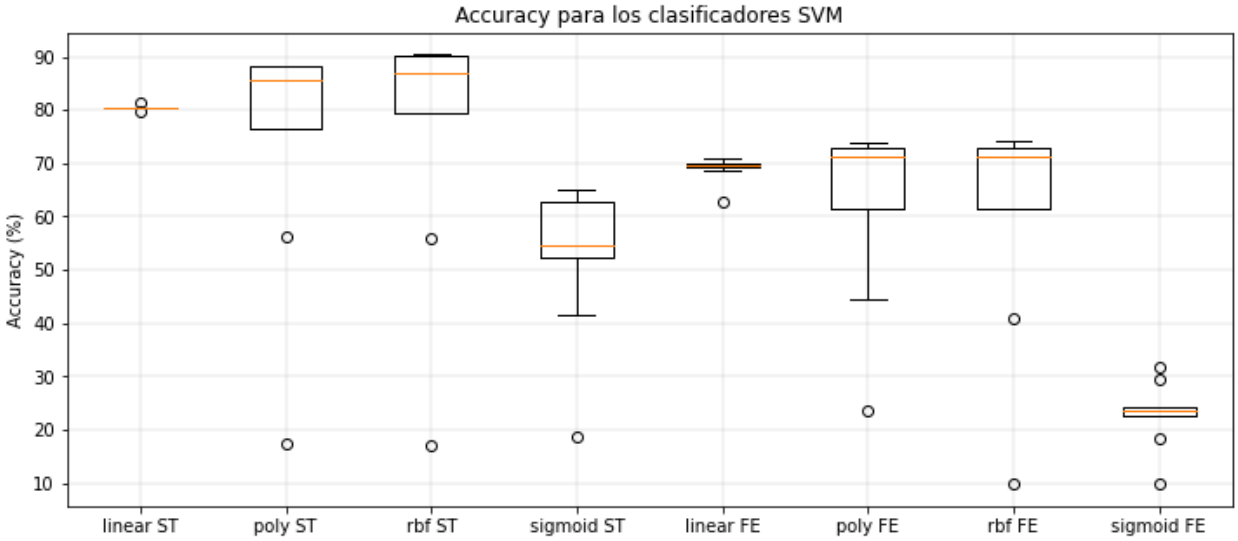

Figura 64. Algoritmo SVM. Resumen de los resultados.

Por último, se representa en la Figura 65 una comparación de los resultados entre la mejor hiperparametrización para serie temporal y feature engineering. Se observa que el valor máximo se obtiene para la serie temporal como entrada, con una accuracy de 91.85 %.

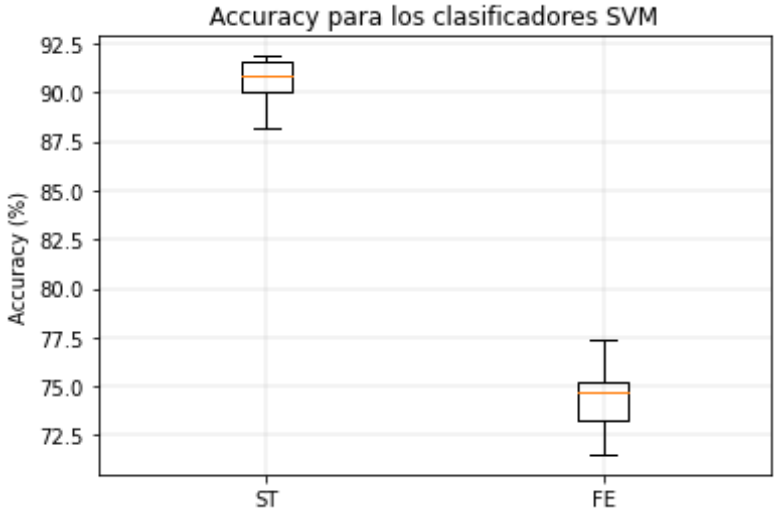

Figura 65. Algoritmo SVM. Resultados de la serie temporal vs feature engineering para mejor hiperparametrización.

6.6 Resumen final de los resultados

Por último, se realizará un resumen de los resultados de los apartados anteriores. En la Tabla 19 se muestra una recopilación de la accuracy obtenida para las diferentes configuraciones de los clasificadores estudiados. En la columna "Serie temporal" se muestra la media de la accuracy para la serie temporal como entrada, mientras que en la columna "Feature engineering" se muestra la media para una entrada con feature engineering. En la última columna, "Máximo", se muestra la máxima accuracy alcanzada por el algoritmo, independientemente del tipo de entrada con la que se haya obtenido. Por otro lado, en la Figura 66 se representa la accuracy para la mejor hiperparametrización de cada algoritmo.

Tabla 19. Resumen de resultados.

		Serie temporal	Feature engineering	Máximo
Redes Neuronales	Dos capas densas	89.14 %	79.99 %	90.13 %
	Tres capas densas	90.44 %	80.59 %	92.06 %
Redes Neuronales Convolucionales	Una capa convolucional y una capa densa	88.63 %		90.56 %
	Una capa convolucional y dos capas densas	91.28 %		93.35 %
	Dos capas convolucionales y una capa densa	91.08 %		92.70 %
	Dos capas convolucionales y dos capas densas	90.32 %		92.49 %
Árbol de decisión		60.59 %	60.76 %	62.09 %
KNN		83.36 %	66.64 %	84.84 %
SVM		90.52 %	74.46 %	91.85 %

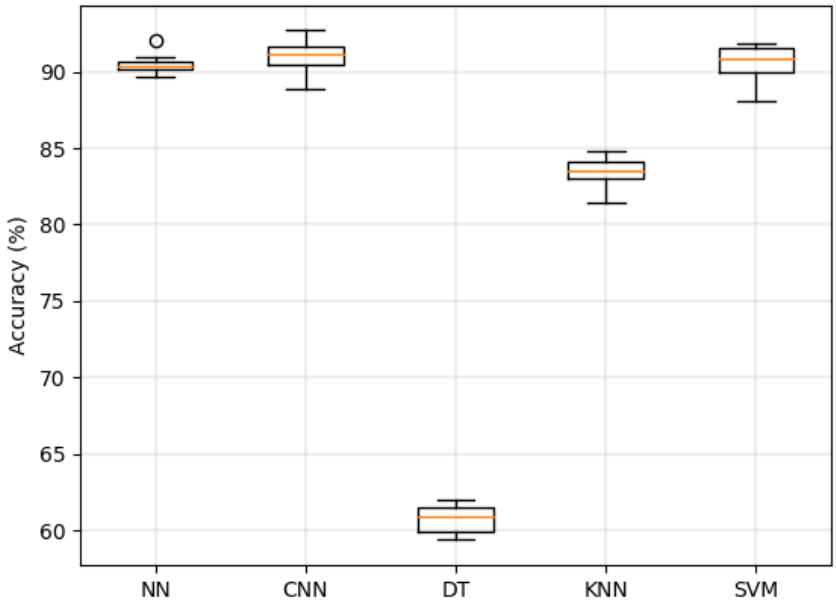

Figura 66. Accuracy para la mejor hiperparametrización de cada algoritmo.

En términos generales y atendiendo a los valores máximos de la accuracy, se observa que el clasificador con una mayor accuracy es el CNN1D, que alcanza un 93.35%. El siguiente clasificador con una mejor respuesta es la red neuronal densa con un 92.06% para su mejor configuración, seguido muy de cerca por el SVM con un 91.85% de accuracy máxima. La respuesta de estos tres clasificadores ante la clasificación de golpes de pádel es, por tanto, muy similar. Algo por debajo de los anteriores se sitúa el KNN, con un 84.84%. Por último, más distanciado se encuentra el árbol de decisión que solo alcanza un 62.09%. Cabe destacar que todos los valores máximos se corresponden con la serie temporal como entrada. En cuanto a los valores medios obtenidos, la mayor accuracy media la obtiene el CNN1D (91.28%), seguido por el SVM (90.52%) y la red neuronal densa (90.44%), ambos muy cercanos entre sí. El algoritmo KNN consigue una media de 83.36%, mientras el árbol de decisión obtiene un 60.76%.

Lo primero que llama la atención es que la red neuronal convolucional 1D no es claramente superior al resto, a pesar de ser un clasificador específicamente diseñado para recibir como entrada series temporales. De hecho, si se comparan los resultados de la red neuronal con dos capas densas con respecto a la CNN1D de una capa convolucional y dos capas densas (donde la única diferencia entre ambos es la existencia de una primera capa convolucional para el CNN1D) la mejora no es significativa: la red con dos capas densas alcanza un 89.14% de accuracy, mientras que cuando se le añade la capa convolucional apenas se incrementa hasta un 91.28%. Es decir, se mejora la accuracy en un 2.14% a costa de un mayor coste computacional. Si se compara este mismo clasificador CNN con la red densa con tres capas, donde ahora la diferencia es que la primera capa es convolucional en el primero y densamente conectada en el segundo, la diferencia es aún menor: 91.28% frente a 90.44%.

Siguiendo con las redes neuronales, se aprecia una ligera mejora introduciendo nuevas capas a la red. En el caso de la red neural densa, meter una tercera capa supone una mejora de un 1.3% para la red con serie temporal, y un 0.61% para feature engineering. En el caso de las redes convolucionales, añadir una segunda capa convolucional al clasificador de una capa convolucional y una capa densa, representa una mejora de un 2.45%. En cambio, añadir una segunda capa convolucional al clasificador con una capa convolucional y dos capas densas no supone ninguna mejoría, sino todo lo contrario, ya que la accuracy desciende casi un 1%. No obstante, se puede apreciar que las diferencias en la accuracy a la hora de añadir capas no son especialmente significativas, ya que en el mayor de los casos apenas supone una mejora del 2.5%.

Por otro lado, es destacable que el clasificador más simple, el KNN con $k = 1$ obtiene una accuracy bastante elevada, quedándo de media a menos de un 8% del clasificador con mayor accuracy.

Se puede observar también que la técnica de feature engineering no supone una mejora en ningún caso. Solo para el árbol de decisión el clasificador con feature engineering logra superar en media al clasificador con la serie temporal, pero dicha mejora es casi imperceptible y no se refleja en el valor máximo, donde de nuevo es el clasificador con serie temporal el que alcanza el máximo. Para el resto, el clasificador con feature engineering supone una pérdida media de hasta un 16% de accuracy respecto al clasificador con serie temporal. Esto podría deberse a que las características extraídas de la serie temporal para aplicar feature engineering (valor máximo, mínimo y medio) no sean las más representativas para realizar la clasificación de golpes de pádel, o simplemente a que el algoritmo tiene mejor respuesta a la serie temporal, ya que ofrece una mayor información. Un estudio más detallado sobre la aplicación de feature engineering a series temporales de golpes de pádel podría dar respuesta a si con la elección de otras características podría superar esta técnica los resultados obtenidos con la serie temporal.

Por último, el hecho de que la clasificación sea peor en aquellos golpes más parecidos es totalmente normal y cabía esperar que ocurriese así. En efecto, hay determinados casos que incluso para una persona sería difícil clasificar. Es evidente que diferenciar un golpe de derechas con un golpe de revés es sencillo, pero ¿cuál es la diferencia entre un golpe de remate en el que el brazo no se ha elevado lo suficiente y un golpe de bandeja que se haya realizado un poco más arriba de lo normal? Incluso dos personas experimentadas en el deporte podrían discrepar en clasificarlo de una clase u otra. Por tanto, que haya una mayor confusión en la clasificación de estos golpes es totalmente aceptable.

7 CONCLUSIONES Y TRABAJOS FUTUROS

7.1 Conclusiones

En primer lugar, se ha podido comprobar que es posible hacer una predicción fiable de golpes de pádel aún sin tener acceso a un extenso conjunto de datos, ya que se logra un acierto superior al 90% con hasta tres algoritmos diferentes, con una base de datos de solo 2.328 golpes para representar un amplio catálogo de clases: 13 tipos de golpes. Así pues, se constata el inmenso potencial de las técnicas de aprendizaje automático, más especialmente para la clasificación en deportes de raqueta.

Por otro lado, se ha comprobado que para realizar una correcta clasificación de series temporales no se necesitan algoritmos diseñados específicamente para clasificar este tipo de señales, ya que el uso de capas convolucionales de una dimensión apenas supone una mejora de un 1% respecto a otros algoritmos que no la utilizan.

En cuanto a los datos de entrada de los algoritmos, se ha comprobado que se realizan mejores predicciones haciendo uso de la serie temporal completa del golpe que utilizando la técnica de feature engineering. Además, se ha demostrado la importancia de una correcta hiperparametrización, puesto que un mismo algoritmo llega a diferir su respuesta en más de un 73% en función de los hiperparámetros elegidos.

Por otra parte, se ha verificado la afirmación realizada al principio del documento sobre considerar la clasificación de golpes de pádel como una clasificación de movimientos de mano. De igual forma, se ha constatado que el uso de un acelerómetro y giroscopio en la recogida de datos es suficiente para llevar a cabo dicha clasificación.

Por último, respecto a la creación del conjunto de datos, se ha constatado que es posible crear un dispositivo con recursos habituales en un departamento de ingeniería electrónica que permita recolectar los movimientos que realiza la mano de un jugador practicando pádel. De igual forma, es factible crear un sistema sencillo y eficaz para recoger todos esos datos e identificar los golpes con el objetivo de crear un conjunto de datos etiquetados.

7.2 Trabajos futuros

Hay varias líneas de estudio con las que seguir este proyecto. Principalmente se pueden dividir en dos grupos: aquellas destinadas a mejorar el conjunto de datos y aquellas destinadas a mejorar los algoritmos de clasificación. El primero de ellos es el que resulta más interesante, especialmente dada la inexistencia en la literatura de un conjunto de datos de golpes de pádel, en contraste con la amplia gama de clasificadores de series temporales disponibles, incluso de ejemplos concretos aplicados al problema de movimientos de muñeca para la práctica de deportes de raqueta. Por este motivo, la primera línea de trabajo propuesta a continuación se basa en la mejora del conjunto de datos de entrenamiento.

Las bases de datos que se usan para el entrenamiento de los modelos de inteligencia artificial suelen contar con cientos de miles de datos. La cantidad de datos necesarios para el entrenamiento de un modelo depende del problema en estudio y del algoritmo elegido, pero hay una premisa clara: cuantos más datos se pongan a disposición del algoritmo para el entrenamiento y más diversos sean estos, mejor será el modelo obtenido. No obstante, el acceso a estos datos lleva normalmente un coste -no necesariamente económico- asociado. Para el caso del dataset creado en este documento, en una prueba de una hora de duración se recogían unos 200 golpes. Esto requiere la disponibilidad de todas las partes necesarias para la prueba: el deportista, el responsable de lanzar las bolas y el responsable de la recogida de datos; además de los recursos materiales, tales como la pista y el equipamiento correspondiente. Por este motivo, no es sencillo disponer de tantos datos como serían

necesarios para un correcto entrenamiento de los modelos.

Como solución a este problema, existe una técnica llamada "***data augmentation***", o aumento de datos, que consiste en la aplicación de transformaciones aleatorias (pero realistas) a los datos con el objetivo de aumentar el tamaño y la diversidad del conjunto de entrenamiento [38]. Puesto que aumenta la diversidad de los datos, es muy útil para mejorar el overfitting.

Un ejemplo de aumento de datos de entrenamiento en series temporales se lleva a cabo en [39], donde Woo Oh y Jeong realizan un estudio sobre el aumento de datos para la detección de fallos en rodamientos a partir de las vibraciones de estos.

En [40], se lleva a cabo un estudio sobre el aumento de datos en series temporales usando redes neuronales convolucionales. Usan dos métodos para lel aumento de datos, los cuales son:

- ***Window Slicing:*** consiste en extraer segmentos de la serie temporal y hacer la clasificación en estos cortes. El tamaño del segmento es un parámetro de este método. En la etapa de entrenamiento, el algoritmo aprende usando estos segmentos; y en el test, cada segmento de la serie es clasificado por el algoritmo, de forma que la clase de la serie original será aquella con más votos entre sus segmentos. Esta técnica está inspirada en la visión computacional.
- ***Window Warping:*** esta técnica es más específica para series temporales y consiste en deformar un segmento de la serie seleccionado aleatoriamente acelerándolo o decelerándolo, como se muestra en la Figura 67.

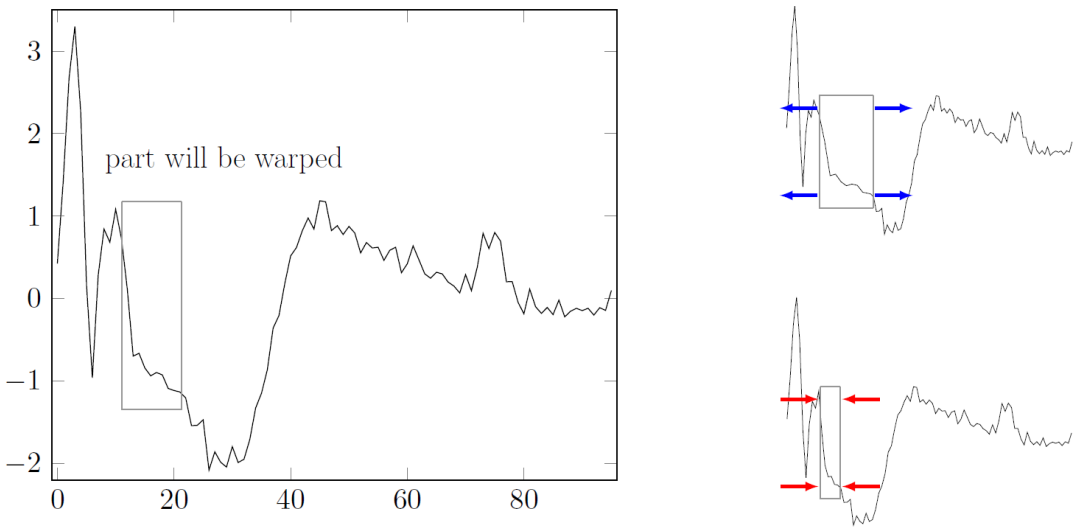

Figura 67. Ejemplo de la técnica window warping para el aumento de datos. Disponible en [40].

En este estudio, Le Guennec, Malinowski y Tavenard llegaron a la conclusión que la técnica del aumento de datos mejoraba el rendimiento de las redes neuronales convolucionales [40].

Por tanto, como trabajo futuro se propone el uso de la técnica data augmentation en el conjunto de datos realizado, estudiando el comportamiento de los distintos clasificadores con esta técnica respecto a la base de datos original.

Otro aspecto para mejorar la base de datos sería la ***diferenciación*** entre los distintos ***efectos*** de cada golpe. En la base de datos actual se han realizado los golpes con varios efectos, pero solamente se han etiquetado con el tipo de golpe. Sería posible realizar una diferenciación en los golpes actuales inspeccionando cada golpe en los vídeos realizados durante las pruebas, pero sería un proceso muy lento. Por tanto, se propone recoger nuevos datos en los que solo se realice un tipo de efecto en cada prueba, realizando tantas pruebas por cada golpe como posibles efectos tenga. De esta forma se podría tener nuevos datos etiquetados con el tipo de golpe y efecto de una forma más sencilla, a la vez que se aumenta la base de datos con golpes reales. De igual forma, aunque la base de datos es ya suficientemente representativa, se podría aumentar con nuevos tipos de golpes como la chiquita o el remate por tres.

En cuanto a los fallos que se cometen en la clasificación, podría estudiarse qué perfil de jugador presenta mayores dificultades a la hora de clasificar sus golpes. La actual base de datos representa desde jugadores principiantes hasta jugadores profesionales, por lo que las diferentes ejecuciones en función del nivel del jugador podrían dar lugar a diferentes resultados en la clasificación. Asimismo, los diferentes efectos que no se han etiquetado en la base de datos podrían ser el origen de los errores más comunes en la clasificación.

Por otro lado, un asunto importante en la captura de datos que es susceptible de mejora es el ***dispositivo de toma de datos***. El dispositivo actual, descrito en el apartado 3.2, tiene en la muñeca del deportista una plataforma con la Raspberry Pi y el Sense Hat. Lo ideal sería limitar la implementación en la muñeca al equipamiento indispensable, es decir, la IMU responsable de recoger los movimientos de la misma, de forma que el dispositivo interfiera lo menos posible en el libre movimiento del deportista, haciéndolo más cómodo para su uso en entrenamientos largos o competiciones. En este sentido, se ha diseñado una nueva versión del dispositivo, mostrado en la Figura 68, que supone una significativa reducción en peso y volumen con respecto al diseño anterior. Se trata de una IMU Adafruit BNO055 encapsulada para introducirla en una correa de Apple Watch, como se muestra en la Figura 69. Esta se conecta por cable a la Raspberry Pi, que puede alojarse cómodamente en la cintura del deportista junto a la batería. Tiene un peso de 37 gramos (sensor + cápsula + correa). Se propone por tanto el uso de este dispositivo para las nuevas recogidas de datos.

Figura 68. Nuevo diseño del dispositivo de toma de datos.

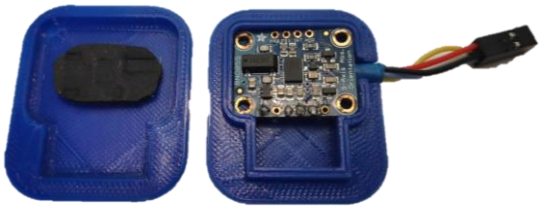

Figura 69. Encapsulamiento del sensor BNO055.

Aunque este dispositivo supone una mejora considerable respecto al primer diseño, tanto en volumen como en peso, es evidente que aún necesita mejoras. Respecto al volumen del dispositivo, hay numerosos jugadores que practican pádel con un Apple Watch, que tiene exactamente las mismas dimensiones. Por otro lado, este dispositivo tiene un peso de 37 gramos, frente a los 50 gramos de un Apple Watch, por lo que no se considera que sea un problema su uso en el terreno de juego. Sin embargo, este diseño se conecta por cable a la placa, lo que sí puede presentar molestias durante las pruebas. Por ende, sería interesante para futuros diseños incluir un dispositivo bluetooth de bajo consumo (BLE) que, además de solucionar el problema del cable, redujese el consumo de batería, haciendo el dispositivo más eficiente.

Otra línea de estudio destacable respecto al sensor sería examinar si la respuesta de los algoritmos es sensible a la ***frecuencia de muestreo*** de la IMU o a la ***duración del golpe*** establecida.

Por otra parte, en el conjunto de datos creado en este documento todos los deportistas tienen algo en común: todos ellos son diestros. Aunque la inmensa mayoría de los jugadores son diestros, hay una pequeña parte de ***jugadores zurdos*** que deberían tener representación en la base de datos. En trabajos futuros, se propone la inclusión de estos jugadores en la base de datos, así como técnicas de ***data augmentation*** que permitan utilizar el conjunto de datos existente de jugadores diestros para clasificar golpes de jugadores zurdos.

Por último, se podría implementar un modelo ya entrenado para ***predecir los golpes en tiempo real***. Este sistema tendría que leer los datos en ventanas de una duración fija (y mayor a la duración del golpe, que son 40 muestras) con solapamiento entre ellas, detectar los posibles golpes en cada ventana y pasar estos golpes al modelo ya entrenado para que realice la predicción. Un esquema del sistema se muestra en la Figura 70.

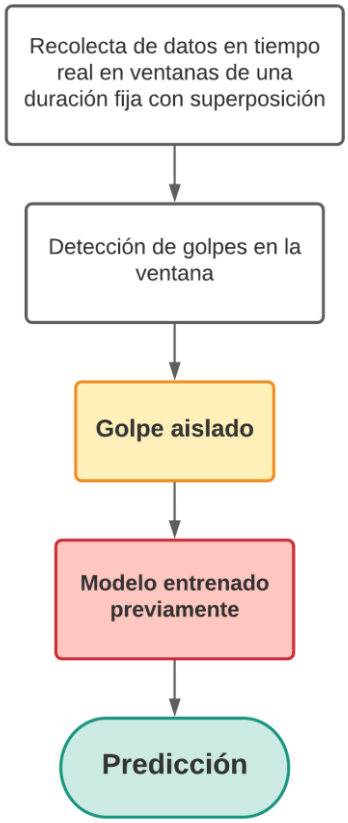

Figura 70. Sistema para una predicción en tiempo real.

REFERENCIAS

[1] Wikipedia, «Microelectromechanical Systems,» 2021 June 14. [En línea]. Available: https://en.wikipedia.org/wiki/Microelectromechanical_systems. [Último acceso: 13 07 2021].

[2] M. Kos y I. Kramberger, «A Wearable Device and System for Movement and Biometric Data Acquisition for Sports Applications,» *IEEE,* 2017.

[3] CCS Insight, «Healthy Outlook for Wearables,» 24 Febrero 2021. [En línea]. Available: https://www.ccsinsight.com/press/company-news/healthy-outlook-for-wearables-as-users-focus-on-fitness-and-well-being/. [Último acceso: 09 Julio 2021].

[4] J. Murua, «El valor de los datos en el deporte,» 2 Noviembre 2018. [En línea]. Available: https://economiaenchandal.com/2018/11/02/el-valor-de-los-datos-en-el-deporte/. [Último acceso: 13 Julio 2021].

[5] J. P. Da Silva, «La era del Big Data en el deporte,» Expansión, 23 Diciembre 2015. [En línea]. Available: https://www.expansion.com/economia-digital/2015/12/23/567ad9aaca47417e6a8b45b8.html. [Último acceso: 13 Julio 2021].

[6] Three Points, «Big Data aplicado a los deportes.,» 26 Mayo 2020. [En línea]. Available: https://www.threepoints.com/es/big-data-aplicado-a-los-deportes. [Último acceso: 13 Julio 2021].

[7] S. Ke, H. Thuc, Y. Lee, J. Hwang, J. Yoo y K. Choi, «A Review on Video-Based Human Activity Recognition,» *MPDI,* 2013.

[8] S. S. Tabrizi, S. Pashazadeh y V. Javani, «Comparative Study of Table Tennis Forehand Strokes Classification Using Deep Learning and SVM.,» *IEEE SENSORS JOURNAL,* vol. 20, nº 22, 2020.

[9] L. Benages Pardo, D. Buldain Pérez y C. Orrite Uruñuela, «Detection of Tennis Activities with Wearable Sensors,» *MPDI,* 2019.

[10] M. Kos, J. Ženko, D. Vlaj y I. Kramberger, «Tennis Stroke Detection and Classification Using Miniature IMU Device,» de *The 23rd International Conference of System, Signals and Image Processing*, Bratislava, 2016.

[11] D. Whiteside, O. Cant, M. Connolly y M. Reid, «Monitoring Hitting Load in Tennis Using Inertial Sensors and Machine Learning,» *International Journal of Sports Physiology and Performance,* 2017.

[12] C. J. Ebner y R. D. Findling, «Tennis Stroke Classification: ComparingWrist and Racket as IMU Sensor Position,» de *MOMM*, Munich, 2019.

[13] K.-C. Lin, C.-W. Wei, C.-L. Lai, I.-L. Cheng y N.-S. Chen, «Development of a badminton teaching system with wearable technology for improving students' badminton doubles skills,» *AECT,* 2021.

[14] J. Ramón-Llín, J. Guzmán, S. Llana, G. Vuckovic, D. Muñoz y B. Sánchez, «Análisis de la distancia

recorrida en pádel en función del nivel de juego y el número de puntos por partido.,» *FEADEF,* 2021.

[15] J. Ramón-Llín, J. Guzmán, D. Muñoz, R. Martínez-Gallego, A. Sánchez-Pay y B. Sánchez-Alcaraz, «Análisis secuencial de golpeos finales del punto en pádel mediante árbol decisional.,» *Revista internacional de medicina y ciencias de la actividad física y del deporte,* 2021.

[16] A. Escudero-Tena, B. Sánchez-Alcaraz, J. García-Rubio y S. Ibáñez, «Analysis of Game Performance Indicators during 2015–2019 World Padel Tour Seasons and Their Influence on Match Outcome,» *Int. J. Environ. Res. Public Health,* 2021.

[17] M. Javadiha, C. Andujar, E. Lacasa, A. Ric y A. Susin, «Estimating Player Positions from Padel High-Angle Videos: Accuracy Comparison of Recent Computer Vision Methods,» *MPDI,* 2021.

[18] B. Play, «Manual de Usuario». Barcelona.

[19] I. Aguilera Calle, D. García Moreno, V. Morante Pindado, A. Pidal Gallego, J. Arroyo Gallardo y M. F. Cárdenas Bonett, Reconocimiento de actividad mediante pulsera con sensores, Madrid, 2017.

[20] Wikipedia, «Raspberry Pi,» [En línea]. Available: https://es.wikipedia.org/wiki/Raspberry_Pi. [Último acceso: 09 Julio 2021].

[21] Wikipedia, «Python,» [En línea]. Available: https://es.wikipedia.org/wiki/Python. [Último acceso: 09 Julio 2021].

[22] E. K. Antonsson y R. W. Mann, «The frequency content of gait,» *Journal of Biomechanics,* vol. 18, nº 1, pp. 39-47, 1985.

[23] G. Cartes Domínguez, «Padel-Shot-Classification-and-Dataset,» 6 11 2021. [En línea]. Available: https://github.com/guillecartes/Padel-Shot-Classification-and-Dataset. [Último acceso: 6 11 2021].

[24] TensorFlow, «Keras,» 21 Septiembre 2020. [En línea]. Available: https://www.tensorflow.org/guide/keras?hl=es. [Último acceso: 09 Julio 2021].

[25] Scikit-Learn, «Scikit-Learn. Machine Learning in Python,» [En línea]. Available: https://scikit-learn.org/stable/. [Último acceso: 26 10 2021].

[26] J. Torres, Deep Learning. Introducción práctica con Keras, Barcelona: Watch This Space, 2018.

[27] D. Calvo, «Perceptrón – Red neuronal,» 2018 12 8. [En línea]. Available: https://www.diegocalvo.es/perceptron/. [Último acceso: 2021 10 26].

[28] R. S. Srinivasamurthy, «Understanding 1D Convolutional Neural Networks Using Multiclass Time-Varying Signals,» *All Theses,* 2018.

[29] ICHI.PRO, «Predicción y redes convolucionales temporales,» ICHI.PRO, 2021. [En línea]. Available: https://ichi.pro/es/prediccion-y-redes-convolucionales-temporales-102124506152164.

[30] Keras, «Conv1D layer,» [En línea]. Available: https://keras.io/api/layers/convolution_layers/convolution1d/. [Último acceso: 31 8 2021].

[31] InteractiveChaos, «Árbol de decisión,» [En línea]. Available: https://interactivechaos.com/es/manual/tutorial-de-machine-learning/arbol-de-decision. [Último acceso:

24 08 2021].

[32] Sitio Big Data, «Árbol de decisión en Machine Learning,» Wordpress, 14 12 2019. [En línea]. Available: https://sitiobigdata.com/2019/12/14/arbol-de-decision-en-machine-learning-parte-1/. [Último acceso: 24 08 2021].

[33] Merkle, «El algoritmo K-NN y su importancia en el modelado de datos,» Merkle, 1 9 2020. [En línea]. Available: https://www.merkleinc.com/es/es/blog/algoritmo-knn-modelado-datos. [Último acceso: 24 8 2021].

[34] E. Campo León, Introducción a las SVM., Zaragoza: Universidad de Zaragoza, 2016.

[35] J. Amat Rodrigo, «Máquinas de Vector Soporte (SVM) con Python,» cienciadedatos.net, Diciembre 2020. [En línea]. Available: https://www.cienciadedatos.net/documentos/py24-svm-python.html. [Último acceso: 30 08 2021].

[36] Scikit Learn, «sklearn.svm.SVC,» 2020. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Último acceso: 30 08 2021].

[37] J. Brownlee, «1D Convolutional Neural Network Models for Human Activity Recognition,» 28 8 2020. [En línea]. Available: https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/. [Último acceso: 14 5 2021].

[38] TensorFlow, «Aumento de datos,» TensorFlow, 05 10 2021. [En línea]. Available: https://www.tensorflow.org/tutorials/images/data_augmentation. [Último acceso: 21 10 2021].

[39] J. Woo Oh y J. Jeong, «Data augmentation for bearing fault detection with a light weight CNN,» *Procedia Computer Science,* nº 175, pp. 72-79, 2020.

[40] A. Le Guennec, S. Malinowski y R. Tavenard, «Data Augmentation for Time Series Classification using Convolutional Neural Networks,» *HAL,* 2016.