Análisis automático de los lanzamientos de penalti en el ámbito del balonmano

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Diego Castellano Caballero

TUTORIZADO POR:
David Sebastián Freire Obregón
Modesto Fernando Castrillón Santana

Fecha Julio / 2023

Agradecimientos

En este apartado, en primer lugar, me gustaría agradecer la labor de mis tutores, David Sebastián Freire Obregón y Modesto Fernando Castrillón Santana, los cuales me han acompañado a lo largo de este trabajo y su ayuda ha sido indispensable para la completitud de este.

En segundo lugar, quiero agradecer a todos los compañeros que han pasado conmigo este largo camino. Su apoyo y presencia estos han hecho que estos 4 años se me hayan pasado volando.

Por últimO, quiero agradecer de corazón el apoyo incondicional de mi familia, sin ellos nada de esto hubiera sido posible. Ellos han hecho que sea la persona que soy hoy en día por lo que estaré eternamente agradecido.

Resumen

Esta propuesta de trabajo de fin de título se centra en el análisis del movimiento así como el reconocimiento de acciones dentro de una competición deportiva como lo es el balonmano. A pesar de encontrarse entre los deportes más populares y practicados, no cuenta con bases de datos de una de acción de gran relevancia en este deporte como lo es el lanzamiento de penalti o 7 metros. El estudio de estas acciones son de gran interés ya que se puede evaluar tanto al lanzador en función del lanzamiento del balón, así como el portero que pretende evitar que el balón entre en la portería.

Para el desarrollo de este trabajo se requiere de la recopilación de una extensa base de datos de lanzamientos de 7 metros, a la que posteriormente se le aplicará un modelo de inferencia encargado de analizar las características del movimiento y determinar aspectos descriptivos de la acción.

Este proyecto busca contribuir al conocimiento científico de este bonito deporte, pudiéndose utilizar el modelo resultante de esta propuesta en futuras investigaciones y mejoras de esta práctica deportiva.

Abstract

This final project proposal focuses on the analysis of movement as well as the recognition of actions within a sporting competition such as handball. Despite being among the most popular and practiced sports, there are no databases on one of the most relevant actions in this sport, such as the penalty shot. The study of these actions is of great interest as it is possible to evaluate both the shooter in terms of the ball's release, as well as the goalkeeper who tries to prevent the ball from entering the goal.

The development of this work requires the compilation of an extensive database of penalti shots, to which an inference model will be applied to analyse the characteristics of the movement and determine descriptive aspects of the action.

This project seeks to contribute to the scientific knowledge of this beautiful sport, being able to use the model resulting from this proposal in future research and improvements of this sport.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

> "Si sólo piensas en ganar, es más fácil que pierdas. Es mejor pensar en cómo ganar"
>
> Valero Rivera, entrenador español de balonmano

Las competiciones deportivas han experimentado grandes avances en las últimas dos décadas gracias a la aplicaciones de la inteligencia artificial (IA) en estas. Esto se debe a que la gran cantidad de datos (estadísticas de partidos, datos biomecánicos, rendimiento de los jugadores, vídeos e información visual de seguimiento, etc.) que se producen en los eventos deportivos pueden llegar a ser utilizados por los métodos y algoritmos de aprendizaje automático, o *machine learning*, con un amplio árbol de finalidades [5].

La aplicación de dichos métodos ha demostrado muchos beneficios en distintos aspectos deportivos. Entre estos beneficios se encuentran la mejora del rendimiento deportivo, la elaboración y selección de tácticas, la prevención de lesiones, la ayuda a la toma de decisiones en el arbitraje, la anotación de estadísticas y la selección de nuevos talentos entre otros. [35]. En definitiva, la aparición de la IA en el ámbito deportivo ha supuesto una transformación al como se entienden y practican los deportes.

Esta propuesta, entre todos aquellos deportes que pueden o ya se benefician de las posibilidades ofrecidas por el *machine learning*, se centra en el balonmano. El balonmano, o *handball* en inglés, es un deporte de contacto que, como su nombre indica, se practica haciendo uso de un balón dominado con las manos. En este participan dos equipos de 7 jugadores cada uno, 6 jugadores de campo y un portero o guardameta; que compiten por la posesión de la pelota e introducir esta última en la portería rival un mayor número de veces que el equipo contrario.

Al igual que en otros deportes de pelota como el fútbol o el *waterpolo*, le pena máxima es el penalti. El lanzamiento de penalti o 7 metros en el caso del balonmano, se ordena en cualquier parte del campo de juego si se frustra un ocasión manifiesta de gol por parte de un defensor [25]. Como su nombre indica, se trata de un tiro a portería desde la línea de 7 metros, y en este intervienen el lanzador, cuyo objetivo es meter el balón en la portería; y el portero, que trata de evitar el gol del rival.

Los penaltis en el balonmano son acciones de gran interés para analizar, el resultado de este lanzamiento (gol, parada y fallo) depende tanto del lanzamiento del balón por parte del lanzador; como el movimiento del guardameta con el objetivo de parar la pelota. En la literatura, aunque se pueden encontrar algunos artículos científicos en los que se apliquen técnicas de aprendizaje automático en el balonmano, no se disponen de estudios e investigaciones acerca de tiros de 7 metros en este deporte.

Es por ello que surge este trabajo, en el que, tras la recopilación de una extensa base de datos de vídeos de acciones de 7 metros y su posteriores anotaciones, se aplicarán técnicas de *machine learning* con el objetivo de predecir el resultado de la acción dados ciertos parámetros. Además, se trabajará en clasificadores que, dado el vídeo de un penalti, traten de determinar el resultado de dicho lanzamiento.

1.1. Motivación

Como ya se ha mencionado anteriormente, no se disponen de estudios e investigaciones acerca del empleo de técnicas de aprendizaje automático en acciones de lanzamientos de penalti en el balonmano, lo que explica la necesidad de la recopilación de datos acerca de estos últimos. En la literatura se pueden encontrar interesantes artículos relacionados con distintos aspectos del balonmano, pero ninguno centrado en la acción propuesta.

El balonmano es una de las prácticas deportivas con un mayor número de practicantes a nivel mundial, y se trata de un deporte complejo en el que se dan lugar a muchas acciones propias de este último (golpes francos, diferentes tipos de lanzamiento en suspensión o en apoyo, fintas, etc.). Entre estas se encuentra la acción del lanzamiento de 7 metros, la cual, por norma general se efectúa repetidas veces a lo largo de un encuentro. Además, en partidos en los que el empate no es una opción y se transcurre el tiempo de la prórroga, se decide el resultado mediante una tanda de penaltis.

En un lanzamiento de penalti intervienen un jugador que actúa como lanzador, el cual puede realizar amagos o lanzar de muchas formas posibles; y otro jugador que funciona como portero, que realizando diversos movimientos y desplazamientos, busca evitar que la pelota lanzada entre en su portería.

Entonces, el hecho de que se trate de una situación muy frecuente en este deporte, y que sea una acción en la que dos jugadores pueden realizar una gran variedad de posibles acciones, convierten el estudio y análisis de los lanzamientos de 7 metros mediante inteligencia artificial en una interesante propuesta a desarrollar como Trabajo de Fin de Grado. En esta última se juntan el interés por las técnicas de aprendizaje automático y la visión por computador con la pasión por el esta práctica deportiva.

1.2. Objetivos iniciales

Los objetivos iniciales planteados son el desarrollo de una modelo de inferencia capaz de predecir los resultados de lanzamientos de penaltis según datos que caracterizan dichas acciones; así como obtener un clasificador de vídeos capaz de determinar si un 7 metros resulta en gol, fallo o parada del guardameta. Para lograr dichos objetivos, se deberán cubrir los siguientes aspectos:

- La recopilación de un conjunto extenso de vídeos de lanzamientos de penalti, entre los que hayan muestras de cada tipo de posible resultado de la acción (gol, fallo o parada).
- Edición y recorte de los vídeos recopilados.
- Anotación manual de las características de los vídeos.
- Estudio de las herramientas, tecnologías y modelos a emplear en el desarrollo de la propuesta.
- Entrenamiento de los modelos de inferencia.
- Inferencia y validación.
- Análisis de los resultados.

1.3. Aportaciones

Este trabajo busca aportar a la comprensión y avance del campo de la IA en el ámbito deportivo, y especialmente, al balonmano. Una de las principales contribuciones es la creación de una base de datos de diferentes lanzamientos de 7 metros, la cual puede servir de punto de partida para posteriores estudios e investigaciones en esta área, ya que actualmente no existen conjuntos de datos del estilo disponibles en la literatura.

Mediante el uso de múltiples clasificadores para predecir el resultado de los tiros de penalti en función de un conjunto específico de características para cada tiro, permitirá analizar y determinar cuáles de estas características (brazo del lanzador, tiro en bote, el movimiento del portero, etc.) un mayor peso a la hora de resultar en gol, fallo o parada.

En cuanto a la parte de clasificación de los vídeos, los modelos de inferencia desarrollados en esta propuesta pueden llegar a ser de ayuda en futuras investigaciones, que permitan llegar a automatizar completamente los procesos de anotación de estadísticas en cuanto a los 7 metros en el transcurso de un partido de balonmano.

1.4. Competencias

En esta sección se expondrán las competencias específicas cubiertas en este Trabajo de Fin de Título. Entre estas competencias se encuentran:

- ***TFG:*** *Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas*

- ***CI15:*** *Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.*
  Esta competencia es cubierta por la naturaleza propia de la propuesta, la cual conlleva el el estudio y empleo de métodos de *machine learning* para el desarrollo de los modelos de inferencia.

- ***CI16:*** *Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería del software.*
  El desarrollo de este trabajo ha seguido una metodología iterativa, empezando con la recopilación de vídeos para conformar la base de datos; y terminando con el análisis de los resultados obtenidos por los modelos aplicados.

1.5. Planificación del trabajo

En la siguiente tabla se puede ver las planificación inicial de los hitos a conseguir en el desarrollo de la propuesta planteada. En esta tabla se indicarán las fases que conforman al trabajo a realizar, su duración estimada y las tareas asociadas a dicha fase.

Cuadro 1.1: Planificación del trabajo

Fases	**Duración estimada**	**Tareas**
Estudio previo y análisis	30 horas	Tarea 1.1: Estudio de las herramientas para la descarga y corte de vídeos
		Tarea 1.2: Familiarización con el proceso de anotación manual
		Tarea 1.3. Familiarización con clasificadores
Diseño, desarrollo e implementación	210 horas	Tarea 2.1: Descarga de vídeos
		Tarea 2.2: Corte de vídeos
		Tarea 2.3: Anotación de vídeos
		Tarea 2.4: Diseño de una batería de clasificadores.
Validación y evaluación de resultados	30 horas	Tarea 3.1: Diseño de experimentos de validación
		Tarea 3.2: Evaluación de resultados
Documentación y presentación	30 horas	Tarea 4.1: Redacción de la memoria del TFG
		Tarea 4.2: Realización de la presentación
		Tarea 4.3: Ensayos presentación

En el siguiente diagrama se puede ver la división del trabajo a realizar para llevar a cabo la propuesta, dividiendo este último en dos fases, una de desarrollo y otra de evaluación:

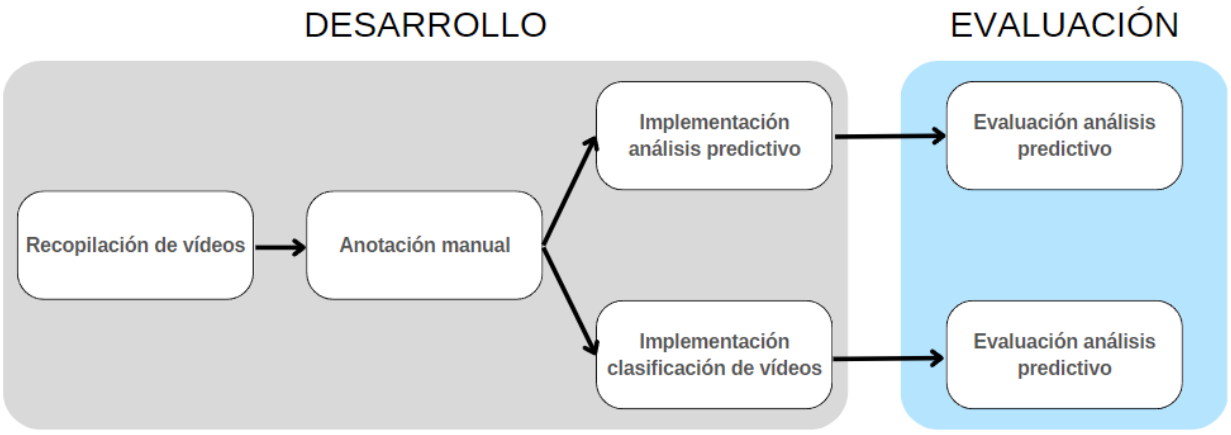


Ilustración 1.1: Flujo de trabajo del proyecto

Capítulo 2

Estado del arte

En esta sección se expondrán los métodos y técnicas de aprendizaje automático empleados en el desarrollo de este trabajo. Para ello, en primer lugar se realizará una pequeña introducción al aprendizaje automático o *machine learning*.

Como este trabajo se compone de dos partes, una de análisis predictivo (*predictive analytics*) del resultado del penalti en base a las características de este, y otra de clasificación de vídeos (*video classification*) para determinar el éxito o fracaso del lanzamiento. Se hará un división entre estás a la hora de describir los aspectos relevantes en cuanto a estas.

2.1. Aprendizaje automático o *machine learning*

Un pionero en el campo de los juegos artificiales y la inteligencia artificial como Arthur L. Samuel, en los inicios de esta última definió la rama del aprendizaje automático o *machine learning* como "campo de estudio que de la inteligencia artificial que confiere a los ordenadores la capacidad de aprender sin ser programados explícitamente". Este último se ocupa del desarrollo de algoritmos y modelos estadísticos que permitan a los ordenadores aprender de grandes cantidades de datos y analizarlos para identificar patrones, hacer predicciones o decisiones y resolver problemas complejos [11]. En el aprendizaje automático se distinguen:

- ✔ **Aprendizaje supervisado**
  En este tipo de *machine learning*, los datos de entrada son etiquetados con sus respectivas clases, indicando el resultado esperado para cada instancia de datos. En una primera fase de entrenamiento, el modelo buscará patrones o relaciones entre los distintos datos, comparando iterativamente los resultados obtenidos con el propio modelo y los resultados

reales. De esta manera, mediante algoritmos y técnicas específicas, se ajusta la función de salida del modelo en cada iteración.
Tras el entrenamiento, se procede a la fase de inferencia en el que se prueba el modelo en un conjunto de datos de entrada no utilizados en el entrenamiento a modo de evaluar el rendimiento del modelo en datos no vistos previamente. Este aprendizaje puede ser dividido en:

- **Clasificación:** empleado en problemas cuyos resultados son valores discretos. Los modelos de clasificación pueden ser binarios, cuando existen dos resultados o clases posibles; o multiclases.
- **Regresión:** se utilizan este tipo de modelos en los casos en los que se producen valores reales como salida.

✔ **Aprendizaje no supervisado**
A diferencia del aprendizaje supervisado, el no supervisado busca encontrar patrones en un conjunto de datos de entrada no etiquetados. El objetivo de los modelos de este tipo de aprendizaje es la extracción de relaciones, tendencias, anomalías o agrupaciones en los datos. Se puede dividir en:

- **Agrupamiento o *clustering*:** los algoritmos de *clustering* son utilizados para agrupar datos en función de su similitud.
- **Reducción de dimensionalidad:** este tipo de algoritmos busca disminuir el conjunto de datos original mediante procedimientos matemáticos y estadísticos.

✔ **Aprendizaje semi-supervisado**
Combina elementos tanto del aprendizaje supervisado como del no supervisado, de manera que, los modelos de este tipo, tienen datos de entrada etiquetados y no etiquetados. Este es útil cuando se manejan grandes cantidades de datos y el proceso de etiquetado pueda llegar a resultar costoso. Entonces, se entrena un algoritmo supervisado sobre los datos etiquetados, y se emplea este modelo para etiquetar el resto de los datos.

✔ **Aprendizaje por refuerzo**
Este enfoque de aprendizaje consiste en el entrenamiento de un agente, capaz de percibir e interpretar el entorno, a través de un proceso de prueba y error. Durante dicho entrenamiento, se recompensan las acciones o comportamientos deseados y se penalizan los que no.

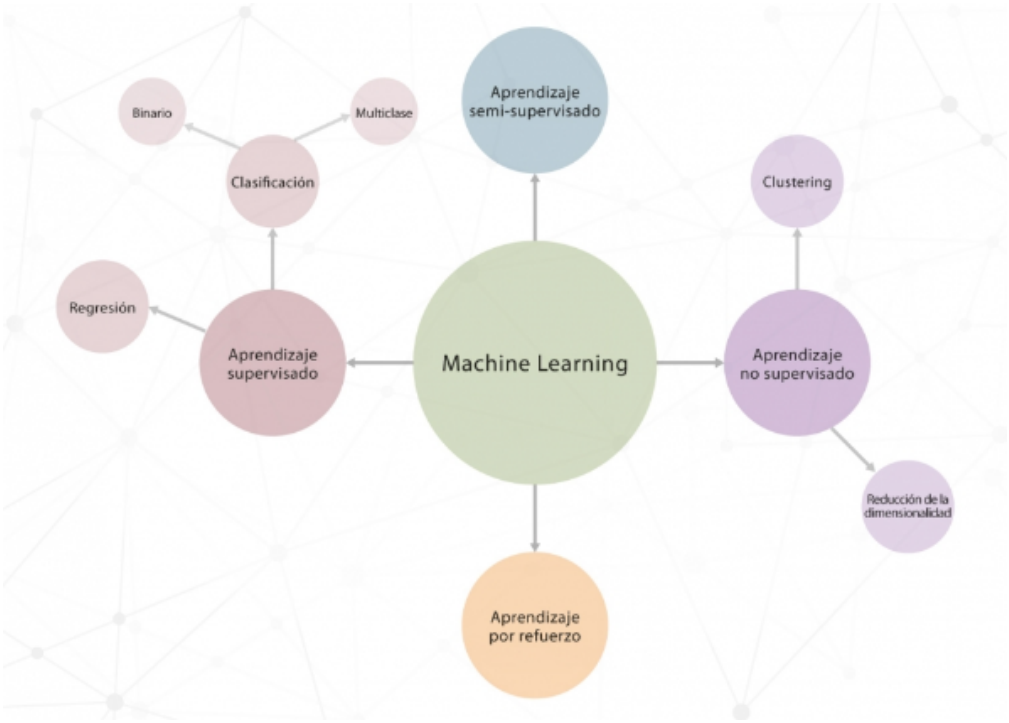


Ilustración 2.1: Tipos de aprendizaje automático [4]

2.2. *Análisis predictivo*

Haciendo uso de modelos matemáticos y estadísticos capaces de aprender de datos históricos, la disciplina del análisis predictivo o *predictive analytics* en inglés tiene como finalidad realizar predicciones precisas sobre nuevos datos. Para ello se necesita encontrar patrones, relaciones y tendencias en los datos existentes; y ahí es donde aparece el aprendizaje automático o *machine learning*.

En este apartado se hará un barrido por los diferentes clasificadores empleados en esta propuesta para predecir el resultado de un lanzamiento dados un conjunto de datos acerca de este. Además, se introduce el concepto de selección de variables el cual es usado en el proyecto.

2.2.1. Clasificadores lineales

Son algoritmos sencillos pero potentes que, basándose en la combinación lineal de las características de un objeto, toman la decisión de clasificar este último en un caso u otro. Estos clasificadores tratan de encontrar un hiperplano que, combinando las características de una instancia ponderadas por determinados coeficientes, separe los objetos en sus respectivas cla-

ses dentro del espacio de características. En la ilustración **2.2** se puede apreciar, en el caso de un problema binario (dos clases), una función lineal o hiperplano que separa la muestras de manera que las se encuentran en un lado de este último son clasificadas como un clase, mientras que las demás muestras son identificadas como la otra clase.

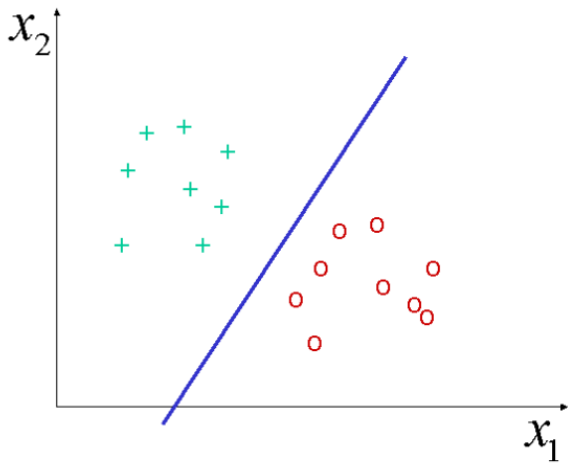


Ilustración 2.2: Espacio de características clasificador lineal [37]

Las entradas de este tipo de clasificadores son vectores conformados por las características de cada objeto que son llamados *vectores de características*. Los coeficientes son ajustados a medida que avanza el proceso de entrenamiento.
Dado un vector de características $\vec{x}$ y un vector de coeficientes $\vec{w}$, la salida del clasificador viene dada por:

$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_j\right)$$

Ilustración 2.3: Salida clasificador linear [38]

2.2.1.1. Regresión logística

La regresión logística o *logistic regression* es un algoritmo estadístico que emplea la salida de la función lineal de regresión (valores continuos) como entrada y clasifica los objetos de manera discreta mediante un función logística, también conocida como función sigmoidea. Gracias a esta última, se transforman los valores reales de entrada por valores entre 0 y 1 [12]. La función logística o sigmoidea se define:

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

Ilustración 2.4: Ecuación función sigmoidea

donde x es un número real dado, en el caso de la aplicación de este algoritmo, por la salida de la función lineal que emplea el vector de características ponderadas por coeficientes determinados. La representación gráfica de la función logística o sigmoidea se muestra a continuación:

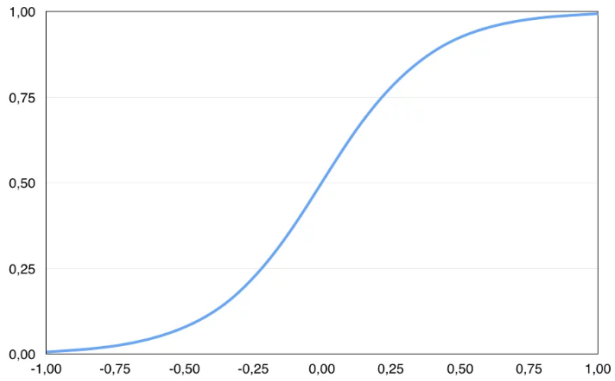


Ilustración 2.5: Representación gráfica función sigmoidea [27]

Como se puede apreciar, el algoritmo de regresión logística permite clasificar únicamente entre dos clases, es decir, problemas binarios. No obstante, existen técnicas que permiten el empleo de este modelo lineal en problemas en los que se tienen más de dos resultados posibles como lo es la regresión logística multinomial. A diferencia de la regresión lógistica binaria, se emplea una función softmax, la cual es una generalización de la función sigmoide. La función softmax para K clases se define como:

$$\text{Softmax } \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Ilustración 2.6: Ecuación función softmax

donde z^i representa la combinación lineal de las características y los pesos asociados con la clase i.

2.2.1.2. Naive Bayes

Este clasificador se basa en la aplicación de los teoremas de Bayes y recibe el apelativo de ingenuo ya que se fundamenta en la suposición de independencia entre las características que definen una clase. Dicha suposición significa que se asume que la presencia o falta de de una característica no afecta al resto de las características, lo que resulta en un algoritmo veloz pero no aplicable a todos los casos.

Además, para el aplicativo de los teoremas de Bayes, se parte de la idea que todas las características tienen el mismo peso en la salida del algoritmo. La expresión del Teorema de Bayes desde la que parten los clasificadores de Naive Bayes es la siguiente:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Ilustración 2.7: Teorema de Bayes en clasificador Naive Bayes [2]

donde y representa la clase de la que se está calculando la probabilidad y X es el vector de características. Utilizando la anterior expresión, el clasificador Naive Bayes calcula la probabilidad de que una instancia pertenezca a una clase dada la presencia de ciertas características.

2.2.1.3. SVM con función kernel lineal

En primer lugar, se hará una introducción a las *Support Vector Machines* (SVMs). Los algoritmos de SVMs pueden ser empleados tanto en problemas de clasificación lineal como no lineales ya que se tratan de modelos fácilmente adaptables y eficientes que pueden manejar tanto datos de alta dimensión como relaciones no lineales.

La finalidad de los clasificadores SVMs es encontrar el punto (1 dimensión), línea (2 dimensiones), plano (3 dimensiones) o hiperplano (más de 3 dimensiones) óptimo capaz de separar las instancias de un problema dentro del espacio N-dimensional donde N es el número de características. Se entiende como óptimo el hiperplano o límite de decisión cuya distancia con los objetos más cercanos de las diferentes clases es máxima [13].

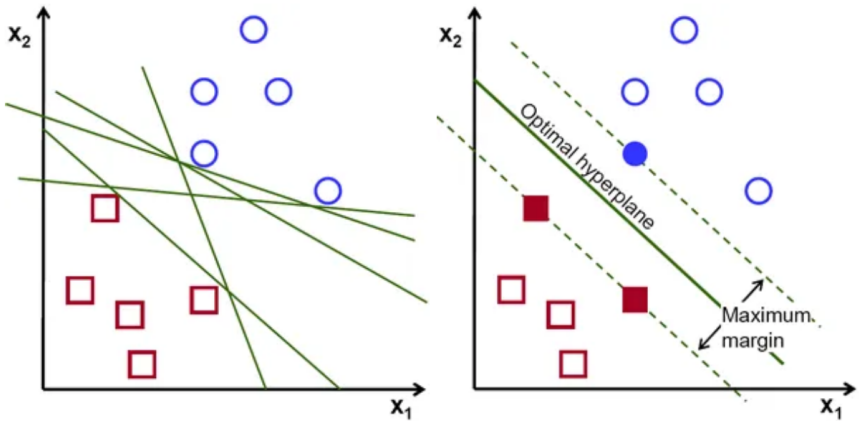


Ilustración 2.8: Hiperplano óptimo en SVM [8]

Los vectores de soporte se definen como los puntos más cercanos al hiperplano, los cuales tienen un rol fundamental en la posición y orientación del hiperplano; y la distancia que mantienen estos últimos con el hiperplano se denomina margen.

En la realidad, rara vez se consigue un hiperplano que consiga la separación perfecta de los datos sin incurrir en sobreajuste u *overfitting* (resultados no extrapolables a otros datos diferentes a los usados en el entrenamiento). Por ello, durante el entrenamiento, las SVMs emplean un parámetro, denominado C, el cual controla el equilibrio entre los margenes máximos y los errores de clasificación. De esta manera, se permite la creación de un margen blando (*soft margin*) que permite algunos errores de clasificación a la vez que los penaliza [39]. Un valor mayor de C permite un margen menor pero reduce el error en el conjunto de entrenamiento, mientras que un valor menor de C permite un margen mayor pero puede aumentar el error de clasificación en el conjunto de entrenamiento.

Los algoritmos SVM al igual que otros planteados para problemas binarios, pueden ser usados en problemas multiclase haciendo uso de dos enfoques:

- ✔ One-vs-All: se entrena un clasificador para cada clase, tomando el resto de clases como clase negativa. Entonces, para determinar la clase de un dato pasado como entrada, se emplean todos estos clasificadores y se escoge la clase con mayor puntuación.

- ✔ One-vs-One: se entrena un clasificador para cada par de clases posibles. En el caso de que el problema tenga K clases posibles, se entrenarán $\frac{K \cdot (K-1)}{2}$ clasificadores. Para la clasificación se sigue una estrategia de votación entre la salida de todos los clasificadores.

En las SVMs, para mapear los datos en entrada en el espacio de características se emplea una función matemática llamada *kernel*, permitiendo dividir el espacio de características don-

de los puntos de los datos no son linealmente separables. Para ello, la función kernel permite mapear las instancias de un espacio de características original a un espacio de características de mayor dimensión.

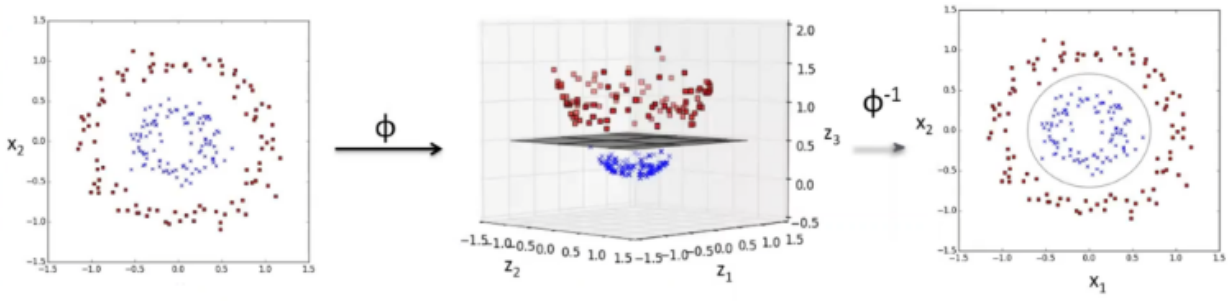


Ilustración 2.9: Transformación de dimensionalidad funciones kernel [21]

En problemas donde los datos pueden ser separados linealmente, se puede emplear una SVM con un kernel lineal para encontrar un hiperplano óptimo. Supongamos que se tiene un problema binario de clasificación donde las clases son +1 y -1. Entonces, cualquier hiperplano que divide ambas clases puede ser expresado como:

$$\mathbf{w}^{\mathsf{T}}\mathbf{x} - b = 0$$

Ilustración 2.10: Expresión hiperplano SVM lineal

donde w se trata del vector normal al hiperplano. La representación gráfica de dicho hiperplano usando una SVM lineal es la siguiente:

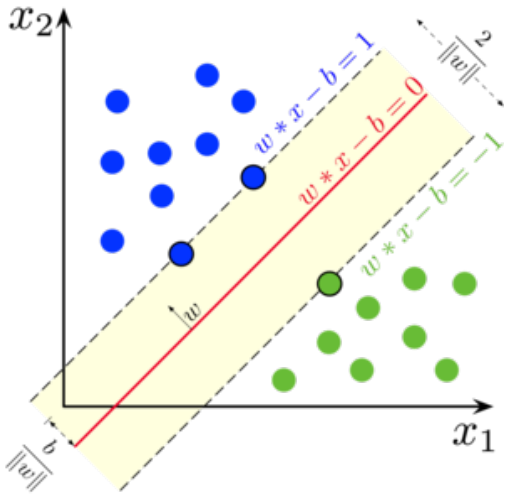


Ilustración 2.11: Representación hiperplano SVM lineal [42]

2.2.2. Clasificadores no lineales

Los algoritmos de este tipo son utilizados en problemas donde los datos no son separables linealmente dentro del espacio de características. Estos algoritmos ofrecen mejores resultados en problemas más complejos que los clasificadores lineales al ser capaces de encontrar relaciones no lineales entre las características de entrada y las etiquetas correspondientes [10].

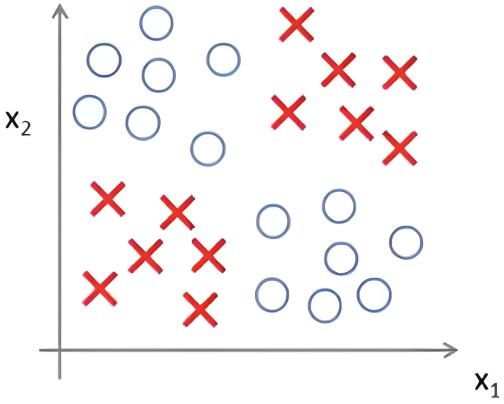


Ilustración 2.12: Problema no lineal

2.2.2.1. K-vecinos más cercanos

El algoritmo de K-vecinos más cercanos o *K-nearest Neighbours* (K-NN) se basa en la idea de que las instancias similares suelen pertenecer a la misma clase. Entonces, usa la proximidad entre los objetos para hacer decisiones o clasificaciones, de manera que, para la clasificación

de una instancia, se tienen como referencia las instancias más próximas o vecinas.

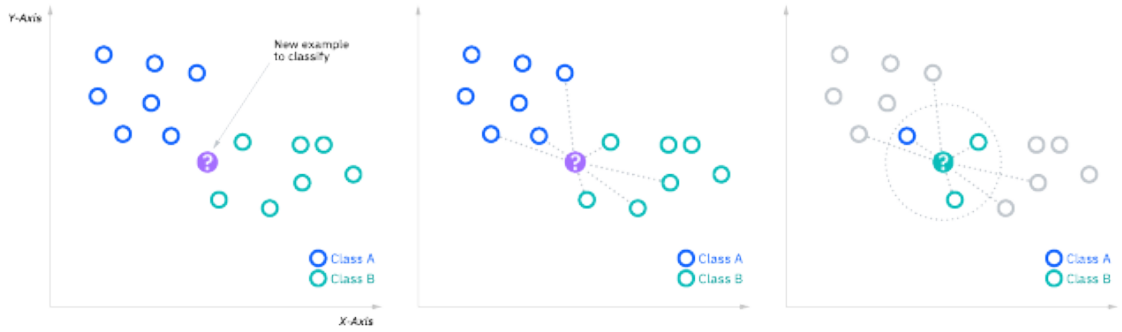


Ilustración 2.13: Representación gráfica algoritmo K-nearest Neighbours [18]

K-NN, para la tarea de identificar los K vecinos más cercanos necesita determinar la distancia de un punto con los vecinos, para lo cual se emplena distintas métricas (distancia euclídea, distancia Manhattan, distancia Minkowski, etc.); definir el parámetro K, el cual indica el número de vecinos a usar en el algoritmo. La decisión de qué valor de K usar tiene un papel crucial en los resultados del clasificador, dependiendo esta de los datos de entrada. En problemas donde se tienen muchos valores atípicos (outliers), valores altos de K funcionan mejor. Además, se recomienda que k sea impar para evitar empates en el proceso de clasificación [18].

2.2.2.2. SVM con función kernel de base radial

Este clasificador no lineal se basa en las máquinas de vectores de soporte (SVM) introducidas en el apartado ***2.2.1.3****.SVM con función kernel lineal* y hace uso de una función kernel de base radial o *Radial Basis Function* (RBF). Esta función mide la similitud (mayor proximidad) entre dos puntos de datos en el espacio de características de acuerdo a la distancia euclídea entre ellos. Este kernel puede ser representado matemáticamente:

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

Ilustración 2.14: Kernel RBF [31]

donde $\|X_1 - X_2\|^2$ es la distancia euclídea entre los puntos. Teniendo en cuenta que $\gamma = 1/2\sigma^2$:

$$K(X_1, X_2) = \exp(-\gamma \|X_1 - X_2\|^2)$$

Ilustración 2.15: Kernel RBF introduciendo gamma(γ)

Con valores de gamma (γ) mayores resultan límites de decisión más complejos que mejoran los resultados sobre los datos de entrenamiento; mientras que con valores de gamma inferiores, los límites de decisión son más flexibles lo cual puede hacer que el clasificador obtenga mejores resultados con nuevos datos de entrada [24].

2.2.2.3. Árbol de decisión

Este clasificador no lineal se fundamenta en una estructura de árbol en la que cada nodo interno (nodo de decisión) representa una pregunta de clasificación acerca de una característica de entrada y los nodos terminales indican la salida del algoritmo, es decir, la clase asociada por este último al conjunto de datos de entrada.

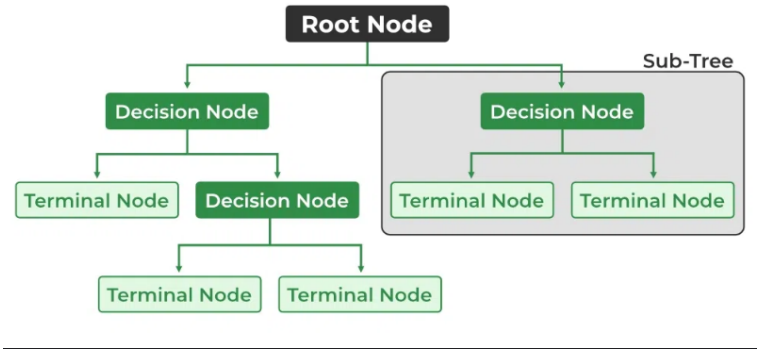


Ilustración 2.16: Árbol de decisión [9]

En cuanto a la construcción del árbol de decisión, en un principio todas las instancias de entrenamiento se encuentran contenidas en el nodo raíz. En cada iteración se escoge la

característica que mejor separe las instancias, evaluando la homogeneidad los subconjuntos de datos separados utilizando la entropía (medida del grado de aleatoriedad o incertidumbre) o el índice de Gini (puntuación que evalúa la precisión de una división). Este proceso se lleva a cabo de forma iterativa para cada nodo hasta el punto en que se cumple un criterio de parada, como alcanzar una profundidad máxima o un número mínimo de instancias en los nodos hoja [9].

2.2.2.4. Multi-layer Perceptron

El Perceptrón Multicapa o *Multi-Layer Perceptron* (MLP) se trata de un tipo de red neuronal que se emplea con frecuencia en problemas de clasificación más desafiantes. Este tipo de clasificador es capaz de encontrar relaciones y patrones complejas en datos que no pueden ser separados linealmente.

Un perceptrón es uno de los componentes fundamentales en el campo de las redes neuronales. Se basa en un modelo matemático que genera una salida sumando varias entradas (valores de las características) ponderadas y aplicando un función de activación.

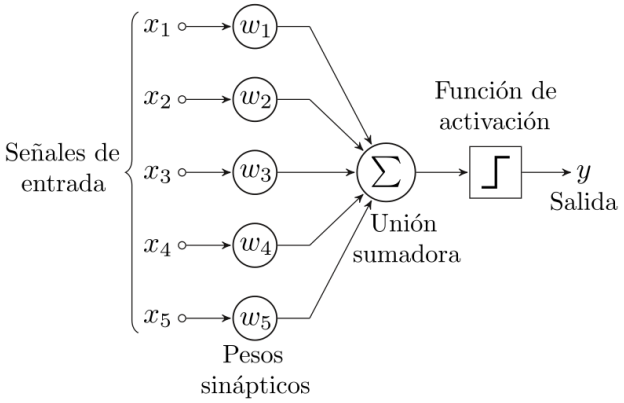


Ilustración 2.17: Perceptrón simple [41]

Una capa, en el contexto de las redes neuronales, se refiere a un conjunto de unidades que trabajan conjuntamente en el procesamiento de la información de entrada. En un perceptrón simple como el que se puede ver en la ilustración **2.17**, solo hay dos capas, la de entrada y la de salida.

Los MLPs, como su nombre indica, están conformados por múltiples capas. Las capas de este tipo de perceptrones que se encuentran entre la capa de entrada y la de salida son denominadas capas ocultas. En esta configuración, las neuronas o nodos de una capa están

conectada con todas las de la capa anterior y la siguiente. Cada conexión entre nodos lleva asociado un peso, el cual establece la importancia relativa de la señal que pasa por esa conexión.

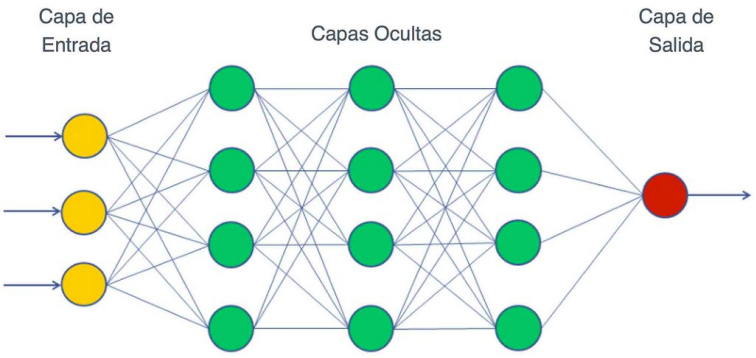


Ilustración 2.18: Perceptrón multicapa [14]

En el proceso de entrenamiento, el modelo ajusta los pesos de las conexiones para intentar mejorar los resultados de la clasificación. Para minimizar la diferencia entre las predicciones del modelo y las etiquetas reales del conjunto de entrenamiento, se busca encontrar la combinación de pesos ideal. Los algoritmos de aprendizaje que cambian gradualmente los pesos para reducir el error de predicción, como el algoritmo de descenso por el gradiente, llevan a cabo esta optimización.

Las neuronas o nodos de las capas ocultas y la capa de salida utilizan funciones de activación, las cuales se tratan de funciones no lineales que deciden si una neurona se active o no. Entre las funciones de activación más comunes se encuentran la función sigmoide, tangente hiperbólica (Tanh), ReLU (*Rectified Linear Unit*) y softmax. Estas funciones son aplicadas a la suma ponderada de las entradas y sus pesos en cada neurona, generando la salida de la neurona.

2.2.3. Selección de características

La selección de características o *feature selection* es una técnica utilizada en el aprendizaje automático para encontrar y elegir un subconjunto con las características más relevantes de un conjunto de datos [40]. El objetivo principal de la selección de características es mejorar el rendimiento del modelo mediante la reducción de la dimensionalidad del conjunto de datos y la eliminación de características sin importancia, redundantes o ruidosas que podrían afectar la precisión del modelo. Entre los beneficios aportados por esta técnica:

+ Reducción del tiempo de entrenamiento: cuando el conjunto de datos es grande y complejo, menos características significan un entrenamiento de modelos más rápido.

+ Mejora de la precisión: la precisión del modelo se puede mejorar eliminando características poco importantes, redundantes o ruidosas.

+ Mayor interpretabilidad: el modelo resultante es más simple de interpretar y comprender.

Los algoritmos de selección de características pueden ser divididos en:

✔ **Métodos de filtro**: clasifican las características de acuerdo con medidas estadísticas, como la correlación y la información mutua. Aunque computacionalmente son efectivos, los métodos de filtro no tienen en cuenta la interacción entre las características.

✔ **Métodos envolventes**: estos métodos, para evaluar rendimiento de varios subconjuntos de características, emplean un algoritmo de aprendizaje automático. Agregan o eliminan características de forma iterativa mientras supervisan el rendimiento del modelo para encontrar el mejor subconjunto de características. Aunque los métodos de envoltura requieren mucho cómputo, pueden capturar cómo interactúan las características.

✔ **Métodos integrados**: combinan el proceso de entrenamiento del modelo con la selección de características. Penalizan los coeficientes de características irrelevantes y eligen las relevantes utilizando técnicas de regularización como la regresión de Lasso y Ridge. Loss métodos integrados son computacionalmente eficientes y recogen la interacción entre características.

Encontrar el equilibrio ideal entre la reducción de la dimensionalidad y la precisión del modelo es fundamental ya que la selección de características puede provocar ocasionalmente la pérdida de información importante.

2.3. Clasificación de vídeos

La clasificación de vídeos en el campo del *machine learning*, a diferencia de la clasificación de imágenes estáticas, tiene en cuenta las relaciones espacio-temporales de la secuencia de fotogramas o *frames* en el tiempo. La gran cantidad de datos que se manejan en clasificaciones de este tipo requieren del uso de técnicas propias del aprendizaje profundo o *deep learning*.

El *deep learning* se trata de una subdisciplina del *machine learning* que tiene como finalidad el desarrollo de algoritmos y modelos de inteligencia artificial que se inspiran en cómo procesa el cerebro humano la información. Estas técnicas emplean estructuras de redes neuronales profundas compuestas por varias capas de neuronas artificiales que trabajan conjuntamente.

2.3.1. *Convolutional Neural Networks* (CNN)

En tareas de visión por computador se emplean comúnmente redes neuronales convolucionales o *Convolutional Neural Networks* (CNN). Este modelo de redes funciona especialmente bien en el procesamiento de datos con estructura de cuadrícula, como las imágenes que son representadas digitalmente como una matriz (blanco/negro) o conjunto de matrices de datos (RGB).

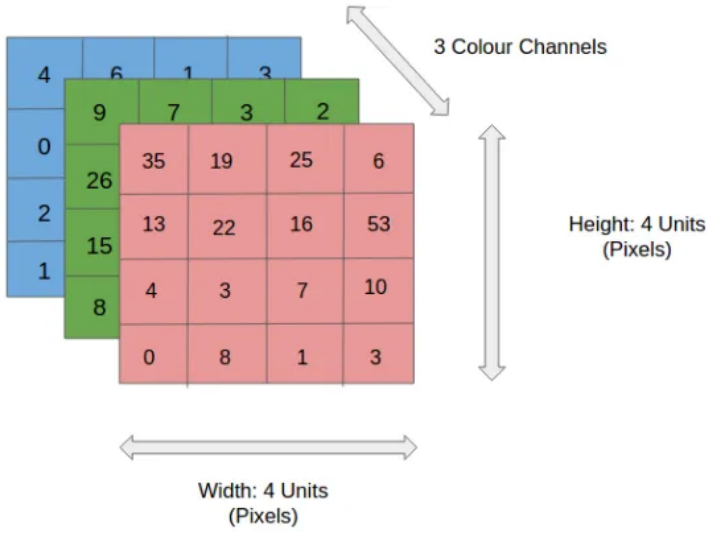


Ilustración 2.19: Representación digital imagen RGB 4x4 [28]

Las CNNs se trata de redes que se componen por tres tipos de capas principales [19]:

✔ **Capa de convolución**: en este tipo de capas es donde se realizan los cálculos para detectar las características en una imagen. Para ello, hace uso de la técnica de convolución, la cual consiste en realizar el producto escalar entre un grupo de píxeles adyacentes y un filtro o kernel. Este último se trata de una matriz bidimensional de pesos encargada de detectar características relevantes. El filtro es desplazado y aplicado por toda la imagen generando un mapa de características, mapa de activación o característica convolucionada.

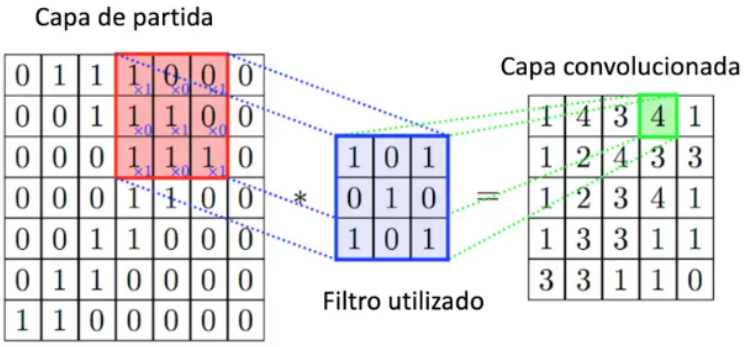


Ilustración 2.20: Convolución

La CNN puede tener varias capas de convolución, creando una jerarquía de características para detectar patrones complejos, como partes de un objeto. Esto permite que la red detecte características de nivel inferior y las combine para representar patrones de nivel superior, creando una jerarquía de características en la red.

✔ **Capa de agrupamiento o *pooling***: también conocida como capa de submuestreo, ayuda a reducir el número de parámetros de entrada y reducir así la dimensión. Al igual que la convolución, el agrupamiento hace uso de filtros o kernel exceptuando que esto no llevan pesos.
Existen dos tipos principales de agrupamiento: el agrupamiento máximo o *max pooling*, seleccionando el valor máximo en el campo receptivo; y el agrupamiento medio o *average pooling*, calculando el valor promedio.

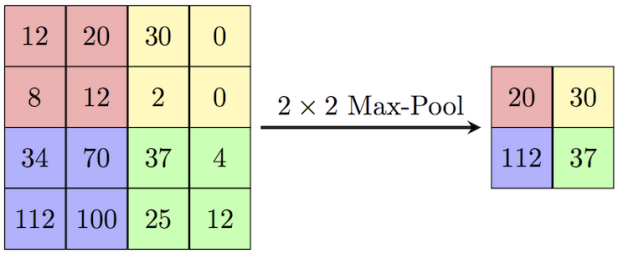


Ilustración 2.21: Ejemplo de Max Pooling [7]

Aunque se pierde información, la agrupación mejora la eficiencia, reduce la complejidad y evita el sobreajuste en la CNN.

✔ **Capa totalmente conectada o *fully-connected***: realiza la tarea de clasificación utilizando características extraídas por las capas anteriores.

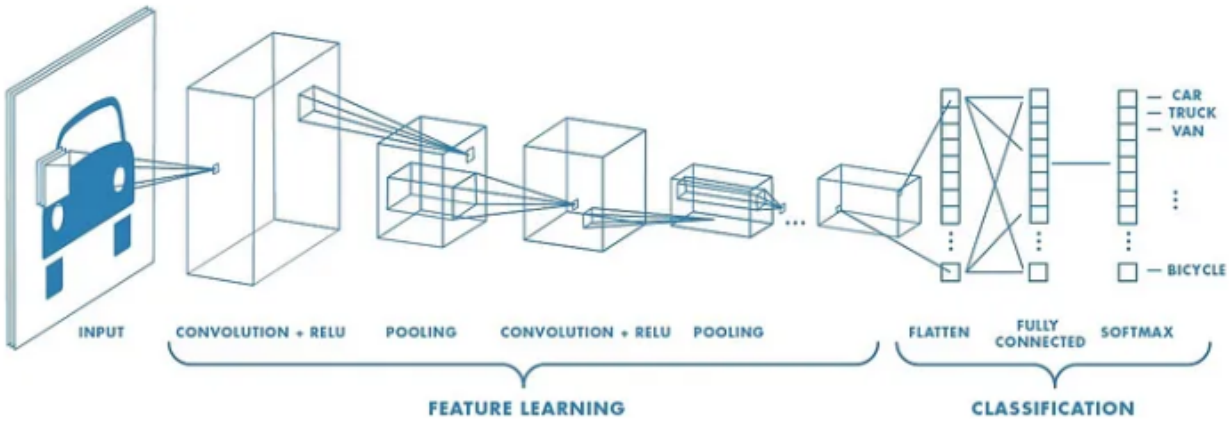


Ilustración 2.22: Estructura de una CNN [28]

La red explicada anteriormente se trata de una red neuronal convolucional 2D (2D CNN). Estas redes han demostrado ser muy eficientes en cuanto imágenes, pero presentan ciertas limitaciones en la clasificación de vídeos ya que no son capaces de analizar relaciones temporales entre la secuencia de fotogramas.

2.3.2. 3D Convolutional Neural Networks (3D CNNs)

Las redes neuronales convolucionales 3D (3D CNNs) aparecen para solucionar las limitaciones expuestas anteriormente que presentan las 2D CNNs en la clasificación de vídeos. Estas redes emplean la convolución 3D, en la que el filtro o kernel 3D es aplicado al cubo tridimensional formado por la secuencia de fotogramas adyacentes. [20]. De esta manera, se pueden extraer características espacio-temporales de un vídeo o imagen 3D.

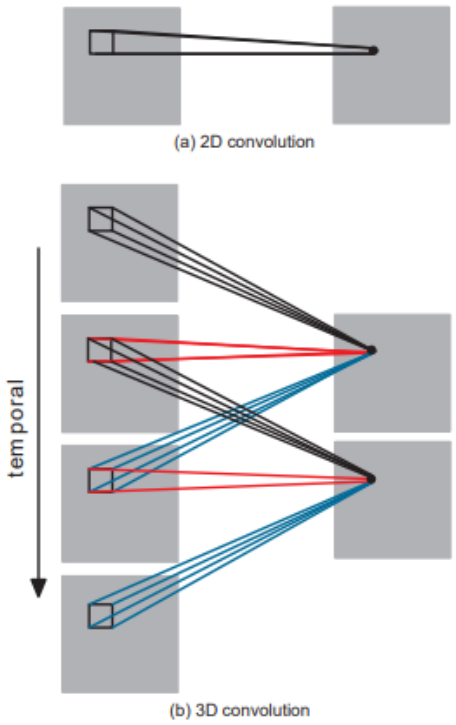


Ilustración 2.23: Comparación convolución 2D y 3D [20]

Debido al aumento de la dimensionalidad (tiempo) y a que hay más parámetros de entrada (varios *frames* a la vez), entrenar una red neuronal de este tipo puede resultar costoso desde el punto de vista computacional.

2.3.3. (2+1)D CNN

Se trata de una variante de las 3D CNNs que se fundamenta en la idea de descomponer la convolución 3D en dos pasos: una convolución 2D, que representa el espacio; y una convolución 1D, que se corresponde con el tiempo. Esta arquitectura, a pesar de no cambiar el número de parámetros respecto a red neuronal convolucional 3D, ofrece un costo computacional menor al aplicar filtros menos complejos además de que la separación de componentes espaciales y temporales facilita la optimización [36].

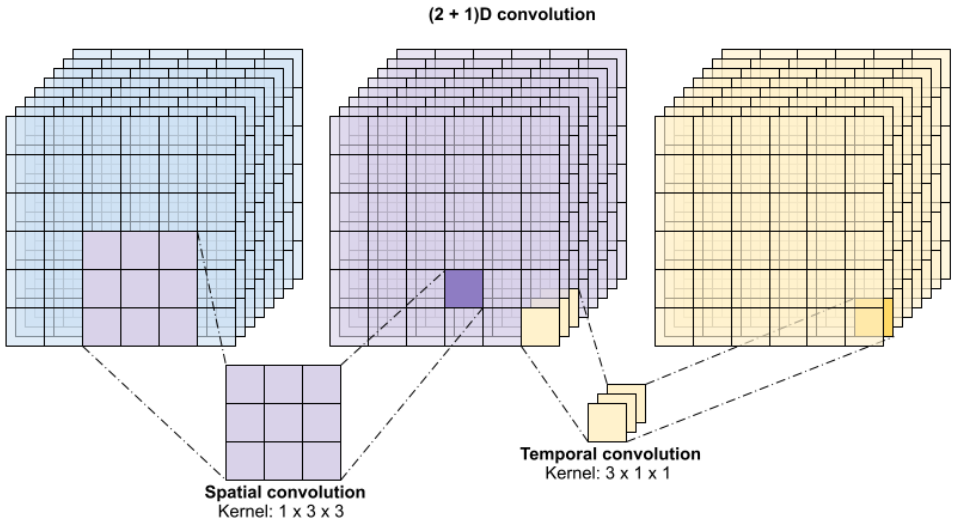


Ilustración 2.24: Convolución (2+1)D [34]

2.3.4. Inflated 3D CNN

La red neuronal convolucional Inflated 3D (I3D) es una arquitectura que se inspira en la redes convolucionales 2D, convirtiéndolas temporalmente redes 3D. Para ello, la I3D comienza con una arquitectura 2D, a la cual *infla* todos los filtros, tanto de las capas convolucionales como las de agrupamiento, para dotarlos con una dimensión adicional, el tiempo. Esto permite que este modelo obtenga un rendimiento similar al de las 3D CNNs tradicionales con un menor número de parámetros y costo computacional. La arquitectura propuesta por sus creadores, Joao Carreira y Andrew Zisserman [1] es la siguiente:

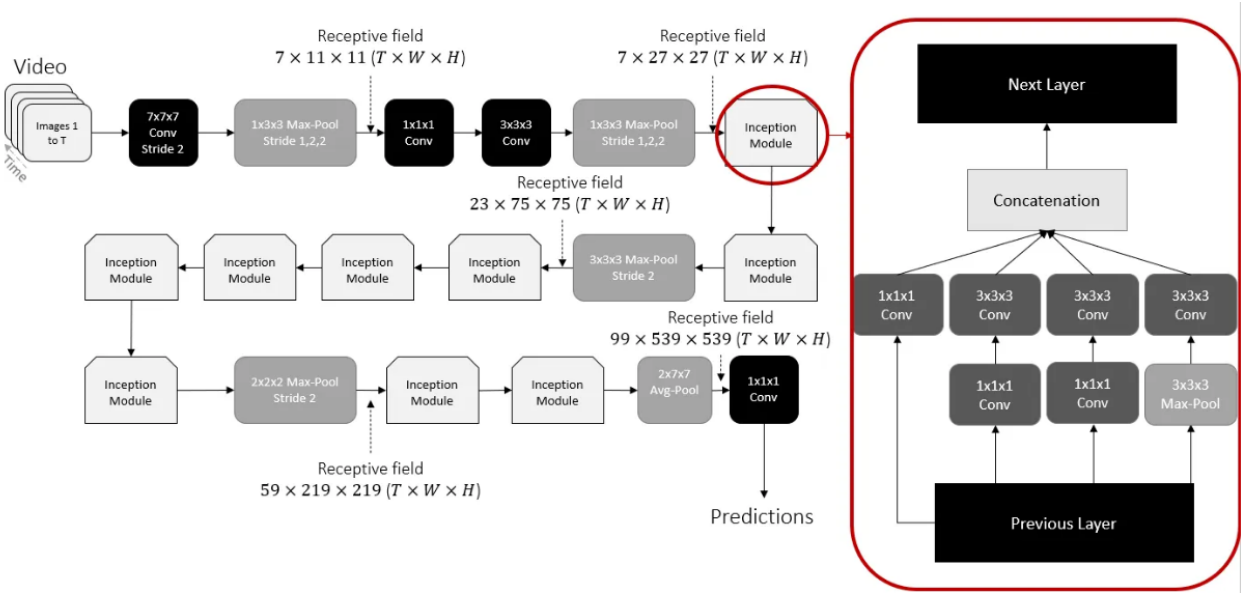


Ilustración 2.25: Arquitectura I3D [29]

2.3.5. Modelos MoViNet

MoViNet es una familia de modelos eficaces de clasificación de vídeo basados en arquitecturas 3D CNN que son capaces de ejecutarse en dispositivos móviles. Estos modelos solucionan el problema que presentan las 3D CNNs tradicionales al mejorar la eficiencia en el punto de vista computacional y de memoria, pudiendo ser utilizados en tareas de reconocimiento de acciones en vídeos en directo [23].

Para conseguir estas mejoras, los creadores de estos modelos han empleado las siguientes técnicas:

✔ MoViNets emplea una búsqueda de arquitectura neural para diseñar una amplia gama

de arquitecturas 3D CNN eficientes y diversas. Esta técnica ayuda a encontrar modelos con un equilibrio óptimo entre precisión y eficiencia computacional.

- ✔ Técnica de Stream Buffer: esta técnica desvincula la memoria de la duración del clip de vídeo, lo que permite a las 3D CNNs emplear secuencias de vídeo de longitud arbitraria tanto para el entrenamiento como para la inferencia con una costo de memoria constante.

- ✔ Se propone una técnica de ensamblado simple para mejorar aún más la precisión del modelo sin sacrificar la eficiencia.

Los modelos MoViNets han demostrado un gran rendimiento en un gran conjunto de datos (aproximadamente 650.000) como lo es Kinetics 600, el cual cuenta con un total de 600 clases que representan acciones humanas. En la siguiente tabla se puede ver el rendimiento de los modelos base de MoViNet, los cuales no son aprpiados para la inferencia de vídeos en directo, en dicho conjunto de datos:

Cuadro 2.1: Rendimiento modelos MoViNet base en Kinetics 600 [23]

Model Name	**Top-1 Accuracy**	**Top-5 Accuracy**	**Input Shape**
MoViNet-A0-Base	72.28	90.92	50 x 172 x 172
MoViNet-A1-Base	76.69	93.40	50 x 172 x 172
MoViNet-A2-Base	78.62	94.17	50 x 224 x 224
MoViNet-A3-Base	81.79	95.67	120 x 256 x 256
MoViNet-A4-Base	83.48	96.16	80 x 290 x 290
MoViNet-A5-Base	84.27	96.39	120 x 320 x 320

donde la precisión "Top-1"se refiere a la precisión al clasificar el vídeo correctamente con la etiqueta de la predicción más probable, mientras que "Top-5.es la precisión al encontrarse la clase correcta entre las 5 predicciones más probables.

Capítulo 3

Desarrollo

En esta capítulo se explicará el proceso de desarrollo de los clasificadores, tanto los empleados en la predicción de resultados como los de clasificación de vídeos; así como todo aquello empleado en dicho proceso. Este proceso ha seguido una metodología de desarrollo en cascada en la que las etapas del proceso se realizan secuencialmente, no pasando a la siguiente etapa sin haber cumplido los objetivos de la etapa actual.

En la primera sección del capítulo, llamada ***3.1**.Herramientas utilizadas*, se hará una breve introducción a los recursos hardware y software utilizados en el proyecto. Además, se expondrán las principales bibliotecas necesarias en el desarrollo de los modelos.

Luego, se describirá el proceso seguido para la recopilación de los datos necesarios para conformar la base de datos en la sección ***3.2**.Recopilación de vídeos*.

Una vez recopilados los vídeos, se necesitará la anotación manual de las características de cada vídeo, cuyo proceso será descrito en la sección ***3.3**.Anotación de los datos*. Cabe destacar que dicha anotación manual será necesaria únicamente para los modelos de análisis predictivo, ya que, como se explicara posteriormente, los nombres de los archivos de los vídeos a clasificar contendrán la etiqueta del resultado de la acción.

Ya habiendo conformado la base de datos, se dividirá la implementación de los modelos en dos secciones: ***3.4**.Implementación modelos de predicción*, donde se explicará cómo se ha realizado el análisis predictivo con varios algoritmos; y ***3.5**.Implementación clasificadores de vídeo*, apartado en el que se describirá el proceso de aplicación de distintos modelos de clasificación de vídeos en la propuesta planteada.

En el siguiente diagrama, extraído de la ilustración **1.1**, se puede apreciar la división del

proceso de desarrollo en los distintos módulos:

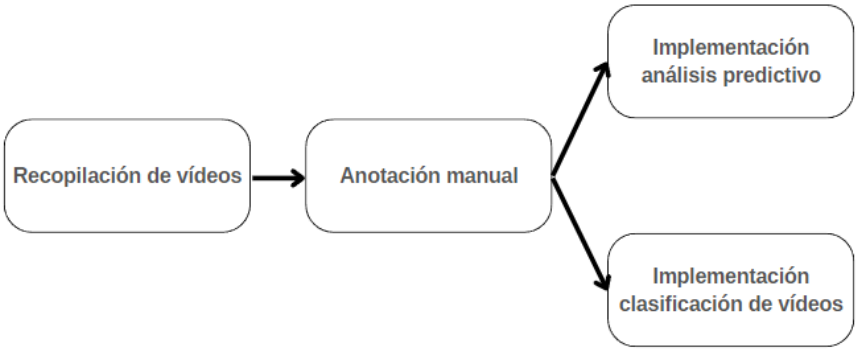


Ilustración 3.1: Flujo del proceso de desarrollo

3.1. Herramientas utilizadas

3.1.1. Recursos hardware

Para la realización de todas las tareas propuestas en este Trabajo de Fin de Título se ha utilizado un ordenador de sobremesa el cual consta con procesador Intel Core de 9ª generación de 8 núcleos y 32 GB de memoria RAM. Destacar que este sistema cuenta con una tarjeta gráfica NVIDIA GeForce RTX 2070 SUPER con 8 GB de VRAM, la cual ha sido utilizada en las tareas de cómputo más demandantes relacionadas con la clasificación de vídeo.

3.1.2. Recursos software

3.1.2.1. Herramienta de edición de vídeos: Clipchamp

Para extraer los fragmentos de vídeos o *clips* de las acciones de lanzamientos de penalti se ha empleado la herramienta software de edición de vídeos Clipchamp. Su elección se explica por la facilidad que ofrece para recortar *clips* de un vídeo importado, pudiendo hacer recortes en fotogramas específicos de manera sencilla.

Ilustración 3.2: Logo de Clipchamp [Clipchamp]

3.1.2.2. Herramienta de anotación de vídeos: Excel

En cuanto a las características de cada vídeo de lanzamiento de 7 metros, estas han sido anotadas en un documento Excel, ya que su manejo se hace familiar y facilita la organización y visualización de los datos, haciendo más amena la tarea de anotación manual de características.

Además, Excel permite exportar sus hojas de cálculos en diferentes formatos, siendo uno de estos el formato de valores separados por comas o en inglés *comma-separated values* (CSV), el cual ha sido el empleado para extraer los datos en la implementación del análisis predictivo.

Ilustración 3.3: Logo de Excel [Excel]

3.1.2.3. Lenguaje de programación: Python

Como lenguaje de programación de este proyecto se ha utilizado Python. Este se trata de un lenguaje de alto nivel e interpretado el cual se ha convertido en uno de los lenguajes más populares y versátiles en el desarrollo de software. Esto se explica por su simplicidad, legibilidad y la posibilidad que ofrece de desarrollar aplicaciones de todo tipo.

La elección de este lenguaje se explica por la amplia gama de bibliotecas que ofrece para tareas de aprendizaje automático y visión por computador, además de su facilidad de uso y experiencia con este.

Ilustración 3.4: Logo de Python [Python]

3.1.2.4. Entorno de trabajo: Google Colab

Google Colab, o también conocido como "Colaboratory", se trata de una herramienta en la nube gratuita que permite escribir y ejecutar código arbitrario de Python en el navegador. Para ello, Colab ofrece entornos de programación que se basan en Jupyter Notebooks que ofrecen recursos computacionales como CPUs, TPUs (Unidades de Procesamiento Tensorial) y GPUs (Unidades de Procesamiento Gráfico) de forma gratuita.

No obstante, los recursos ofrecidos no son ilimitados. Estos son recursos compartidos por todos los usuarios, de manera que son asignados según la disponibilidad y demanda de estos. La cantidad de recursos disponibles pueden llegar a ser un problema en tareas que requieren grandes conjuntos de datos o modelos de aprendizaje automático complejos [Google].

Durante el desarrollo de los modelos de clasificación de vídeos, se ha afrontado un problema con la mencionada limitación de recursos, de ahí que se decidiera usar los recursos hardware del sistema, creando una conexión local entre el Colab y este último. Para ello, se ha seguido las instrucciones que ofrece Google para conectarse a un entorno local.

En el caso de los modelos de predicción, al no demandar tantos recursos computacionales, se ha hecho uso de los recursos de Colab. Para acceder al archivo de formato ".csv" que contiene los datos de entrada, se ha aprovechado la posibilidad que ofrece Colaboratory para acceder al Google Drive personal, donde se ha almacenado dicho archivo.

Ilustración 3.5: Logo de Google Colab [Google Colab]

3.1.2.5. Principales bibliotecas utilizadas

Scikit-learn En el análisis predictivo se hecho uso de la biblioteca Sckit-learn. Esta es una popular biblioteca de código abierto para Python que se emplea en tareas de aprendizaje automático.

Scikit-learn ofrece una amplia gama de herramientas y algoritmos, incluyendo clasificadores lineales y no lineales, así como métodos de selección de características. Esto hace que esta biblioteca sea ideal para el desarrollo de esta propuesta.

Ilustración 3.6: Logo de Scikit-learn [scikit-learn]

TensorFlow Esta biblioteca se ha empleado en la parte de clasificación de vídeos. TensorFlow es de código abierto y ha sido desarrollada por Google para tareas de *machine learning* y especialmente de modelado de redes neuronales.

Una de las características que diferencian a esta biblioteca es la utilización de ”tensores´´ en las operaciones. Estos se tratan de vectores o matrices de N-dimensiones capaces de representar todo tipo de datos. Gracias a estos, los datos pueden ser manejados y procesados de manera eficiente.

TensorFlow ofrece una amplia variedad de herramientas para la construcción y entrenamiento de modelos de aprendizaje profundo de gran complejidad. Además, es capaz de aprovechar el potencial de las CPUs y GPUs al ejecutar operaciones matemáticas de una forma optimizada, lo que acelera los procesos de entrenamiento y evaluación.

Ilustración 3.7: Logo de TensorFlow [32]

Keras Keras es una biblioteca de código abierto de alto nivel escrita en Python diseñada con el objetivo de proporcionar una API simple y fácil para el desarrollo y entrenamiento de modelos de aprendizaje profundo.

Todas las funcionalidades que ofrece esta biblioteca han sido completamente integradas en TensorFlow y pueden utilizarse de manera nativa desde esta última, facilitando más aún el desarrollo de sistemas de *deep learning*.

Ilustración 3.8: Logo de Keras [Keras]

3.2. Recopilación de datos

La primera fase del proyecto ha sido la búsqueda y descarga de vídeos que contuvieran acciones de lanzamientos de penalti de los cuales extraer fragmentos con dichas acciones. El proceso llevado a cabo para cada vídeo ha seguido los siguientes pasos:

3.2.1. Búsqueda y descarga del vídeo

Dicha búsqueda se ha realizado en la plataforma YouTube, en la que se pueden encontrar muchos vídeos relacionados con el deporte del balonmano. Entre estos vídeos, los más útiles para extraer acciones de penalti han sido aquellas recopilaciones de momentos destacados de partidos de balonmano. Para realizar esta tarea, se ha hecho uso de una hoja de cálculos de Excel para anotar el nombre de cada vídeo, la URL de este, y el número de penaltis extraíbles, diferenciando entre gol, fallo y parada.

Una vez encontrado el vídeo y comprobado que este no había sido descargado anteriormente haciendo uso del documento en Excel, se procede a su descarga. Esta se ha realizado con la herramienta en la nube de Wave.video la cual permite la descarga de vídeos de la plataforma YouTube en formato MP4. Los vídeos empleados en esta propuesta han sido bajados con una resolución de 1280 x 720 píxeles.

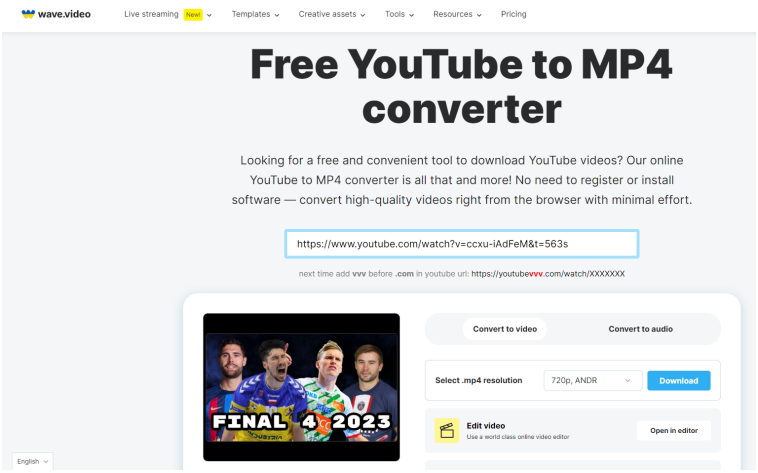
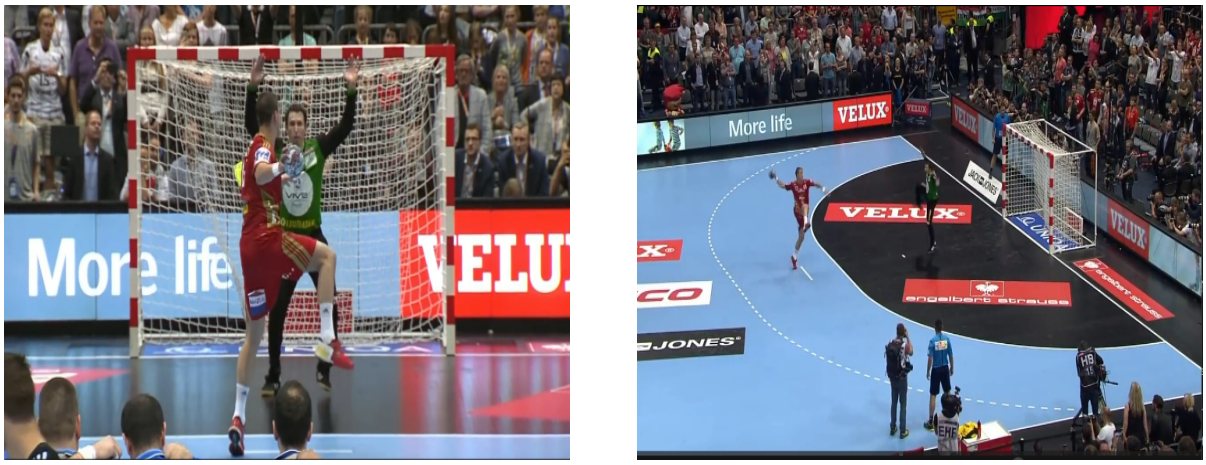


Ilustración 3.9: YouTube converter Wave.video

3.2.2. Recorte de fragmentos de vídeo

Ya teniendo el vídeo almacenado en el ordenador personal, resta extraer los *clips* con los lanzamientos de 7 metros contenidos en el vídeo descargado. Para ello, se importa este último a la herramienta de edición Clipchamp. Hay que destacar que, en cuanto la posición de la cámara, se pueden distinguir dos tipos: trasera, encontrándose la cámara a espaldas del lanzador; y diagonal, capturando la acción diagonalmente.



(a) Cámara trasera (b) Cámara diagonal

Ilustración 3.10: Dos tipos de fragmento de vídeo según la posición de la cámara

Se ha seguido una nomenclatura específica para distinguir entre las distintas acciones en la carpeta donde se han almacenado los fragmentos de vídeos que conforman la base de datos

creada. Esta esta conformada por las siguientes partes separadas por "_":

- ✔ Resultado de la acción, siendo las posibilidades "Gol", "Fallo" y "Parada".
- ✔ Número del vídeo, siendo el primero "v1", el segundo "v2" y así sucesivamente.
- ✔ Número del lanzamiento de penalti dentro del vídeo, siendo el primero "p1", el segundo "p2" y así sucesivamente, restableciéndose al pasar al siguiente vídeo.
- ✔ Posición de la cámara, asignándole "back" a los archivos de vídeo con la cámara en la parte trasera como en la ilustración **3.10a**; y "diag" cuando la cámara es diagonal como en la ilustración **3.10b**.

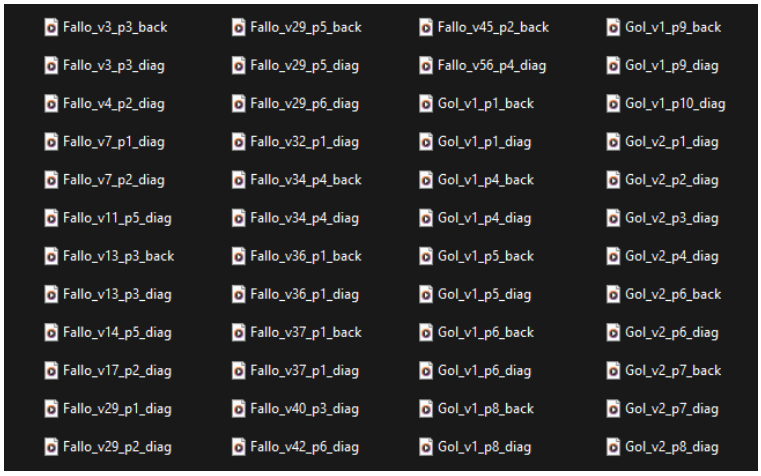


Ilustración 3.11: Ejemplo de nomenclatura de los fragmentos de vídeo recopilados

Al finalizar el proceso explicado en esta sección, se han llegado a recopilar un total de 569 lanzamientos, de los cuales:

- 387 resultan en gol, que representan el 68.01 % del total de vídeos recopilados.
- 156 son paradas del guardameta, que suponen un 27.42 % del total.
- 26 son fallos del lanzador, 4.57 % del total.

3.3. Anotación de los datos

Para el desarrollo de los modelos de análisis predictivo, se requiere el etiquetado de las características de los vídeos recopilados para el entrenamiento e inferencia de estos últimos. Para llevar a cabo esta tarea se ha hecho uso de una hoja de cálculos de Excel, en la cual, para cada característica de la acción se han empleado valores numéricos a modo de agilizar el trabajo. El atributo a predecir por los modelos es el resultado del lanzamiento, en donde se representará el gol con un 0, la parada con un 1 y el fallo con un 2.

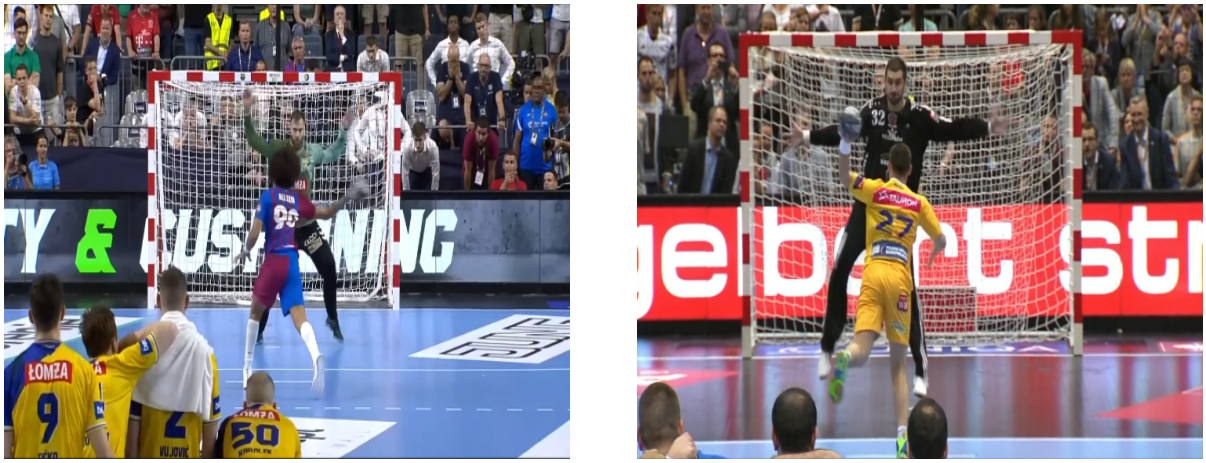
Como ya se ha mencionado anteriormente, el éxito o fracaso de un tiro de 7 metros depende de muchos factores, los cuales vienen determinados por las acciones realizadas tanto por el lanzador como por el portero. Entonces, en esta anotación manual, se distinguirán tres tipos de características en un lanzamiento de penalti:

3.3.1. Características relacionadas con el lanzador

El lanzador es aquel jugador que, con el objetivo de marcar gol, lanza el balón a la portería desde una distancia de 7 metros en este caso. Este lanzamiento puede ser efectuado de varias maneras, las cuales dependerán de:

- ✔ **Brazo del lanzador**: según con qué brazo realice el lanzamiento, un lanzador puede ser:
  - Diestro: representado en la anotación por un 0.
  - Zurdo: representado en la anotación por un 1.
- ✔ **Número de amagos**: en balonmano, al igual que en otros deportes de pelota, un amago se trata de un engaño del atacante con el que intenta hacer creer al defensor o portero de que va a realizar un pase o lanzamiento. En el caso de un penalti, el lanzador busca descolocar al portero para tener más opciones de meter gol. Hay que destacar que, una vez que el arbitro haya hecho sonar su silbato, el lanzador cuenta con 3 segundos para tirar, por lo que el número de amagos está limitado. La anotación de esta característica se ha escrito el número directamente, habiendo lanzamientos con ninguno (0), uno (1) o dos (2) amagos.
- ✔ **Altura del brazo**: según desde donde lance el lanzador, influye en la trayectoria y el ángulo del lanzamiento, así como en la velocidad del balón. Entonces, se distinguen dos tipos de lanzamientos:

- Brazo a la altura del hombro, anotados con un 0.
- Brazo por encima de la altura del hombro, anotados con un 1.



(a) Brazo a la altura del hombro (b) Brazo por encima de la altura del hombro

Ilustración 3.12: Dos tipos de lanzamiento según la altura del brazo del lanzador

3.3.2. Características relacionadas con el portero

El portero o guardameta es el encargado de evitar que el balón lanzado a su portería se introduzca en esta. A diferencia del lanzador, el portero tiene una mayor capacidad de movimiento. Su éxito en la parada depende de varios factores importantes, tales como:

✔ **Posición respecto a la portería**: en el reglamento de este deporte se especifica que el portero se puede encontrar adelantado en un penalti hasta 4 metros desde la portería, distancia señalizada en el terreno de juego por una pequeña línea. Estar adelantado permite al portero cubrir un espacio mayor y reducir así las posibilidades del lanzador, de ahí a que la mayoría de los guardametas decidan adelantarse lo máximo posible para detener un lanzamiento de 7 metros. En la anotación realizada, se han diferenciado tres posiciones del portero respecto a la portería:

- Adelantado hasta la línea de 4 metros, representado por un 0.
- Adelantado entre la línea de portería y la línea de 4 metros, representado por un 1.
- No adelantado, es decir, que se sitúa en la línea de portería, representado por un 2.

✔ **Lado al que se desplaza**: el portero tiene la opción de moverse hacia la izquierda, hacia la derecha o quedarse en el centro para intentar detener el balón. Es importante

destacar que se ha tenido en cuenta la perspectiva del portero a la hora de anotar su desplazamiento.

✔ **Acción realizada**: en cuanto a esta característica, se tienen como opciones:

- El portero se va completamente al suelo, lo que suele ser necesario en muchas ocasiones para detener lanzamientos en bote o abajo. Esta acción es anotada con un 2.
- El portero salta para detener el balón. Dentro de esta categoría se encuentra el famoso "espagat", acción en la que el portero salta abriendo las piernas y manos en el aire. Es anotada con un 1.
- El portero no realiza ninguna de las dos acciones anteriores, anotado con un 0.

(a) Portero completamente al suelo

(b) Portero realiza un "espagat"

Ilustración 3.13: Acciones del portero

✔ **Toca el balón**: se ha considerado si el portero ha tocado el balón durante el lanzamiento. Si el portero lo ha tocado (0), generalmente esto resulta en una parada exitosa, aunque en algunas ocasiones el balón puede terminar entrando en la portería. Por otro lado, si el portero no ha tocado el balón (1), no existe la posibilidad de parada como resultado.

✔ **Posición del brazo izquierdo/derecho**: se han tenido en cuenta ambos brazos por separado como características. Las posibles posiciones del brazo izquierdo/derecho del portero diferenciadas en el proceso de anotación han sido las siguientes:

- Brazo por debajo del hombro (0).

- Brazo a la altura del hombro (1).
- Brazo por encima del hombro (2).
- Brazo cubriendo el lado contrario (3). Esta situación se da en casos en los que el portero desee tapar completamente un lado, posicionando sus dos brazos en ese lado.

✔ **Posición de la pierna izquierda/derecha**: al igual que con los brazos, se han realizado anotaciones para ambas pierna. Se distinguen las siguientes posiciones:

- Pierna a la altura del suelo (0).
- Pierna a la altura de la cadera (1).
- Pierna a la altura de la cabeza (2).

3.3.3. Características relacionadas con el balón

Dentro de estas características se encuentran:

✔ **Bote**: se anotará un 0 en el caso de que el balón bote antes de entrar en la portería o de la parada del portero. En caso contrario, se anotará un 1.

✔ **Portería**: en esta característica, se etiquetará el lugar de la portería por el cual entra el balón, o en el caso de parada por parte del portero o fallo del lanzador, se etiquetará el lugar al que iba dirigido el balón. Para ello, se ha dividido en una matriz 3 x 3 la portería, siendo los valores anotados los de la siguiente ilustración:

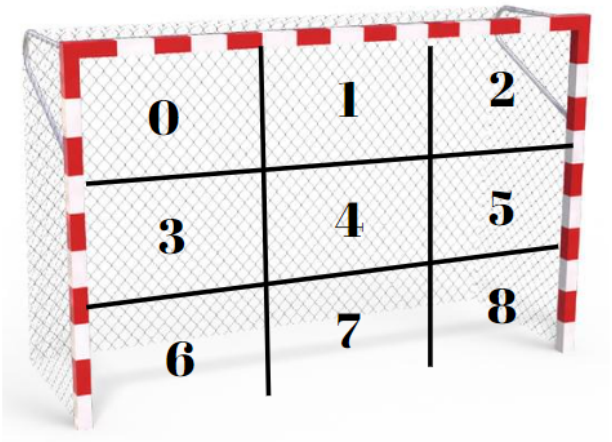


Ilustración 3.14: División de la portería para la anotación

3.4. Implementación modelos de análisis predictivo

En esta sección se describirán los pasos seguidos durante el desarrollo de los modelos de inferencia respectivos a la predicción del resultado basándose en los rasgos del lanzamiento. Todo el desarrollo que se expondrá a continuación se ha realizado en un Jupyter Notebook dentro de la herramienta Google Colab, haciendo uso de los recursos alojados que esta ofrece.

3.4.1. Configuración del entorno de trabajo

En primer lugar, una vez conectado al entorno de ejecución alojado en Colab, se descargan las dependencias necesarias, que en este caso solo se necesitará bajarse la biblioteca Scikit-learn. Tras dicha descarga, se importan todas aquellas librerías y utilidades necesarias.

3.4.2. Carga de los datos

Los datos necesarios para el entrenamiento e evaluación de los modelos desarrollados son importados desde un archivo con formato ".csv", en el cual las características de cada instancia se encuentran separadas por comas. Este archivo se encuentra almacenado en un directorio de Google Drive, por lo que se aprovecha la utilidad disponible en Colaborary que permite montar el sistema de archivos de Drive en el entorno de ejecución.

Una vez declarada la ruta donde se ubica el archivo con las anotaciones, se utiliza la librería Pandas para leer este último y preparar el conjunto de datos. Pandas es una biblioteca de Python que proporciona herramientas de manipulación y análisis de datos. Gracias a esta última, se lee el archivo de anotaciones y se crea un objeto DataFrame, que se trata de una estructura de datos bidimensional con columnas y filas etiquetadas, facilitando así el manejo de los datos. En la siguiente imagen se puede apreciar la información relevante del DataFrame creado:

```
RangeIndex: 569 entries, 0 to 568
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Brazo lanzador            569 non-null    int64
 1   Amagos                    569 non-null    int64
 2   Altura brazo lanzador     569 non-null    int64
 3   Portero adelantado        569 non-null    int64
 4   Accion portero            569 non-null    int64
 5   Pierna derecha portero    569 non-null    int64
 6   Pierna izquierda portero  569 non-null    int64
 7   Brazo derecho portero     569 non-null    int64
 8   Brazo izquierdo portero   569 non-null    int64
 9   Lado portero              569 non-null    int64
 10  Porteria                  569 non-null    int64
 11  Bote                      569 non-null    int64
 12  Resultado                 569 non-null    int64
dtypes: int64(13)
memory usage: 57.9 KB
```

Ilustración 3.15: Información relevante del conjunto de datos creado

Se puede ver en la ilustración **3.15** que el tipo de todas las características es "int64", lo que se explica con el uso de valores enteros en la anotación descrita en la sección ***3.3.****Anotación de los datos*. Se aprecia que el número total de características tenidas en cuenta es de 13.

	Brazo lanzador	Amagos	Altura brazo lanzador	Portero adelantado	Accion portero	Pierna derecha portero	Pierna izquierda portero	Brazo derecho portero	Brazo izquierdo portero	Lado portero	Porteria	Bote	Resultado
0	0	0	1	0	0	1	0	2	2	1	8	1	0
1	0	1	1	0	0	1	0	2	2	1	8	1	0
2	1	1	1	0	1	2	0	2	0	1	0	1	1
3	1	1	1	0	1	2	0	2	0	1	0	1	1
4	1	1	1	0	0	0	1	1	1	1	0	1	1

Ilustración 3.16: Primeras 5 instancias del conjunto de datos

3.4.3. Creación de clasificadores

Se exponen a continuación los detalles de la implementación de los clasificadores de la librería Scikit-learn escogidos para esta tarea. Todos ellos pertenecen al aprendizaje automático supervisado, una elección justificada por la naturaleza del problema, el cual implica una clasificación, y por la presencia de etiquetas en los datos.

Como ya se ha comentado en la sección **3.3.Anotación de los datos**, se cuentan con tres tipos de características diferenciadas: relacionadas con el lanzador, con el portero y con el balón. Teniendo esto en cuenta, se han utilizado los clasificadores que se expondrán en esta sección en tres conjuntos de datos distintos:

1. Características relacionadas con el lanzador + características relacionadas con el balón: 5 características que son el brazo del lanzador, el número de amagos, la altura del brazo

del lanzador, por donde entra o va dirigido el balón en la portería y el bote.

2. Características relacionadas con el portero + características relacionadas con el balón: 9 características que son la posición del portero, la acción de este, la posición de sus brazos y piernas, el lado al que se desplaza, por donde entra o va dirigido el balón en la portería y el bote.

3. Todas las características: un total de 12 características.

Esta segmentación en tres conjuntos de datos se justifica por la búsqueda de determinar qué jugador tiene mayor influencia en el posible resultado de la acción.

Se plantea la hipótesis que los clasificadores entrenados con el primero de los conjuntos presentados sea el que menor exactitud consigan, seguidos por los entrenados con el segundo conjunto. Esto se debe a que el portero tiene a su disposición un amplia gama de movimientos posibles mientras que el lanzador se encuentra más limitado en este aspecto. Además, el portero, en la mayoría de las acciones actúa reaccionando al lanzamiento, muchas veces tirando de sus propios reflejos, por lo que en estos casos cuenta con información de este último, pudiendo llegar o no a detener el tiro. Se supone que los clasificadores que disponen de toda la información de la acción sean más precisos.

Teniendo en cuenta la división de los datos expuestas, se han utilizado varios de los clasificadores implementados en la biblioteca Scikit-learn. En específico, los seleccionados para esta tarea de análisis predictivo son todos aquellos que fueron explicados en la sección **2.2.Análisis predictivo** perteneciente al capítulo del estado del arte.

Cabe destacar que algunos de los algoritmos seguidos por los clasificadores están implementados para problemas binarios. En estos casos, Scikit-learn ofrece la posibilidad de emplear estos últimos en casos de que hayan más de dos clases posibles de realizar las estrategias "One-vs-All" y "One-vs-One" introducidas con los algoritmos SVM en la sección **2.2.1.3**. En estos casos, se han probado ambos enfoques, siendo el primero de ellos el escogido ya que se han obtenido mejores resultados con este.

La discretización de los valores de las características conlleva que todos estos se encuentren un rango específico y escala común, lo que facilita el procesamiento de estos por parte de los algoritmos de aprendizaje automático. Por lo tanto, no se necesita aplicar técnicas de preprocesamiento de los datos como la estandarización.

Ahora se comentarán los aspectos más relevantes de los clasificadores entrenados, indicando la función de la biblioteca empleada para su creación así como los parámetros utilizados:

3.4.3.1. Clasificadores lineales

✔ **Regresión Logística Multinomial**

En el caso del algoritmo de regresión logística, al tratar con más de dos posibles resultados, ha sido necesario usar la variante multinomial de dicho algoritmo. Para ello, se establece en el parámetro *multi_class* el valor "multinomial". El resto de parámetros se han utilizado los valores por defecto. En la siguiente imagen se puede ver la creación de este clasificador.

```
LogisticRegression(multi_class="multinomial")
```

Ilustración 3.17: Creación de clasificador Regresión Logística Multinomial

✔ **Naive Bayes Multinomial**

En la figura **3.18** se puede apreciar la creación de este modelo. En este caso, se utiliza la versión multinomial de este algoritmo, la cual es adecuada para problemas con múltiples clases y es especialmente útil cuando se trabaja con valores discretos en las características de los datos.

```
MultinomialNB()
```

Ilustración 3.18: Creación de clasificador Naive Bayes Multinomial

✔ **SVM con función kernel lineal**

En este modelo, al tratarse de un algoritmo de clasificación binaria, se necesita aplicar la estrategia "One-vs-All". Para ello, se emplea la clase de Scikit-learn llamada *OneVsRestClassifier()*, la cual implementa dicho enfoque para problemas multiclase. El clasificador es pasado como parámetro a esta clase, y este último es creado con:

```
LinearSVC(C=3, max_iter=10000)
```

Ilustración 3.19: Creación de clasificador SVM con función kernel lineal

donde el parámetro c indica el valor del parámetro C el cual controla el equilibrio entre los margenes máximos y los errores de clasificación. Tras probar varios valores de C, se ha seleccionado un valor de C=3 pues ofrece mayor rendimiento. Además, se ha establecido el número máximo de iteraciones en el entrenamiento del modelo en 10000.

3.4.3.2. Clasificadores no lineales

✔ **SVM con función kernel de base radial**

Se plantea el mismo problema que con el algoritmo SVM con función kernel lineal. Entonces, se usa de nuevo el enfoque "One-vs-All". La creación de este clasificado puede ser vista en la siguiente imagen:

```
SVC(kernel='rbf', gamma=1.5, C=3)
```

Ilustración 3.20: Creación de clasificador SVM con función kernel de base radial

donde el parámetro *gamma* se refiere al valor de γ que se puede apreciar en la expresión de la función de base radial en la figura **2.15**. Se le ha asignado el valor 1.5 a este parámetro y el valor 3 al parámetro C.

✔ **K-vecinos más cercanos**

En este clasificador únicamente se establece el número K de vecinos más cercanos. Para evitar empates, se ha seleccionado un valor impar, el cual es 15. Se han probado varios valores de K, y el que mejor rendimiento proporcionó fue este último. Además, se ha comprobado que la aplicación de la estrategia "One-vs-All" mejora la exactitud de este clasificador. Gracias a este enfoque, el clasificador e adapta individualmente a cada clase, lo que puede resultar en que instancias de una clase específica más difícil de distinguir sean clasificadas correctamente.

```
KNeighborsClassifier(15)
```

Ilustración 3.21: Creación de clasificador K-vecinos más cercanos

✔ **Árbol de decisión**

Al igual que con el anterior clasificador, "One-vs-All" mejora el rendimiento de este.

```
DecisionTreeClassifier()
```

Ilustración 3.22: Creación de clasificador árbol de decisión

3.4.3.3. Redes neuronales

Gracias a la biblioteca Scikit-learn se pueden crear clasificadores basados en redes neuronales como lo son los MLPs, de los cuales se habla en la sección **2.2.2.4**. Para ello, se emplea la clase *MLPClassifier()* estableciendo los siguientes parámetros:

- *solver* = "lbfgs"
  Este parámetro se refiere al método de optimización empleado en el entrenamiento, el cual es el algoritmo encargado de ajustar los pesos de las neuronas de la red durante dicho proceso. Entre todos lo métodos de optimización implementados y ofrecidos por la clase del clasificador, se ha seleccionado L-BFGS ya que se obtienen mejores resultados con este. Este último rinde mejor y más rápido que otros en conjuntos de datos relativamente pequeños como el que se utiliza en este caso.

- *alpha* = 0.5
  Para evitar el sobreajuste u *overfitting* se utiliza la técnica de regularización L2 o regularización sobre regresión de Ridge, cuya fuerza o intensidad está controlada por el parámetro α(alpha). En este caso se utiliza una regularización moderada.

- *activation* = "relu"
  La función de activación por defecto en la clase *MLPClassifier()* es ReLu (*Rectified Linear Unit*), la cual se trata de una función simple pero eficiente cuya fórmula matemática es $f(x) = \text{máx}(0, x)$. Esta anula los valores negativos y los positivos los mantiene.

- *random_state* = 1
  Se establece un valor arbitrario en la semilla aleatoria generadora de los pesos iniciales del clasificador, controlando así la aleatoriedad del modelo.

- *early_stopping* = True
  Al activar este parámetro, se reserva un 10 % de los datos de entrenamiento como conjunto de validación al finalizar cada época o iteración. Si la puntuación obtenida por el clasificador en estos datos de validación no mejora en las últimas 10 épocas, la ejecución se detiene, lo que evita un entrenamiento innecesario.

- *max_iter* = 100000
  Ya que se activa la parada anticipada, se pone un número elevado de épocas.

- *hidden_layer_sizes*
  Con este parámetro se puede especificar el número de capas ocultas de la red así como las neuronas que conforman cada una de estas. Se han probado 4 distintas combinaciones

de capas ocultas:

- [10, 10]: 2 capas con 10 neuronas cada una.
- [50, 20]: 2 capas con 50 neuronas la primera y 10 la segunda.
- [50, 20, 10]: 3 capas con 50 neuronas la primera, 20 la segunda y 10 la tercera.
- [100, 50, 20, 10]: 4 capas con 100 neuronas la primera, 50 la segunda, 20 la tercera y 10 la última.

```
MLPClassifier(
            solver="lbfgs",
            alpha=0.5,
            random_state=1,
            max_iter=100000,
            early_stopping=True,
            hidden_layer_sizes=[100, 50, 20, 10],
        ),
```

Ilustración 3.23: Creación de clasificador MLP

3.4.4. Entrenamiento de los clasificadores

Antes de proceder al entrenamiento de los modelos, es necesario definir tanto el conjunto de datos de entrenamiento como el conjunto de datos de validación para evaluar el rendimiento de los modelos entrenados. Scikit-learn ofrece la utilidad de segmentar el conjunto de datos completo en los dos grupos, entrenamiento y validación. Para ello, para cada uno de los 3 datasets explicados en el inicio del apartado anterior, se utiliza la función *train_test_split()*, con la que se destinan el 80 % de las instancias al entrenamiento y el 20 % restante a la validación.

```
features_lanzador = ['Brazo lanzador', 'Amagos', 'Altura brazo lanzador', 'Porteria', 'Bote', 'Resultado']
dataset_lanzador = dataset[features_lanzador]

X = dataset_lanzador[features_lanzador[:-1]]
y = dataset_lanzador['Resultado']

X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
```

Ilustración 3.24: Segmentación en los grupos de entrenamiento y test en Scikit-learn del conjunto de datos características del lanzador

donde la “X” corresponde a los valores de las características de entrada de los clasificadores

mientras que la "y" representa las etiquetas de salida correspondiente a cada dato de entrada. Se especifica con *test_size*=0.2 que el 20 % de los datos de entrada serán del conjunto de evaluación.

Las clases de Python con las que se han creado todos los clasificadores expuestos anteriormente cuentan con un método denominado *fit()*. Este último es utilizado para entrenar el modelo con los datos de entrenamiento creados, que en este caso se encuentran almacenados en las variables *X_train* e *y_train* que son creadas en la ilustración **3.24**. En la figura **3.25** se puede ver que ambas variables son pasadas como parámetros, donde *X_train* es el conjunto de variables de entrada e *y_train* los valores de salida de cada dato de *X_train*.

```
clf.fit(X_train, y_train)
```

Ilustración 3.25: Entrenamiento de los clasificadores mediante la función *fit()*

Una vez terminado el entrenamiento del clasificador, el objeto que contiene a este último contará con los parámetros del modelo ajustados a los datos de entrenamiento según el algoritmo empleado en este proceso. En otras palabras, el clasificador habrá "aprendido" y estará listo para hacer predicciones sobre los datos de validación, los cuales no han sido vistos por el modelo.

3.4.5. Selección de características

Además de la implementación de los clasificadores mencionados anteriormente, se ha realizado un proceso de selección de características para determinar cuales de las características más importantes de cada uno de los tres conjuntos de datos, entendiendo como importancia el grado de influencia en los resultados de clasificación. Para ello, Scikit-learn tiene herramientas para hacer una selección entre las distintas características en aras de mejorar la precisión de los modelos o aumentar su rendimiento en datasets con muchas variables.

Se utiliza una técnica de selección de características que se basa en el análisis estadístico univariante de cada característica con respecto a la variable objetivo, en este caso el resultado de la acción. Con esta última se calcula una puntuación de la relevancia de cada variable en la predicción. En la figura **3.26** se puede ver la implementación de dicha técnica, donde "k" es el número de características seleccionadas.

Al aplicar esta técnica con los datos de entrenamiento, se pueden obtener las puntuaciones de cada una de las características. Teniendo esto en cuenta, el criterio tomado para la selección

del número "k" ha sido escoger todas aquellas características que superasen un puntuaje superior a 1. Estas puntuaciones se verán en el apartado ***4.2****.Resultados análisis predictivo*. Entonces, los valores de "k":

- ✔ Características relacionadas con el lanzador + balón: k = 3
- ✔ Características relacionadas con el portero + características balón: k = 6
- ✔ Todas las características: k = 7

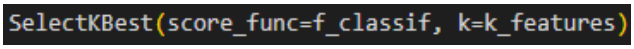


Ilustración 3.26: Selección de características con *SelectKBest()*

Se puede ver en la anterior imagen anterior que la función que determina la importancia de las características (*score_func*) es "f_classif". Esta última funciona en base a una prueba de análisis de varianza (ANOVA), la cual se utiliza para comparar las medias de varios atributos y determinar si hay diferencias significativas entre ellos.

3.5. Implementación clasificadores de vídeo

De la misma manera que con la implementación de los modelos de análisis predictivo, se describirán los detalles más relevantes del cómo se han creado y entrenado los modelos de clasificación de vídeos. Se usa de nuevo Google Colab, pero en este caso, conectado a un entorno de ejecución local. Para llevar a cabo este desarrollo, se ha tomado como punto de partida el tutorial de TensorFlow titulado "Video classification with a 3D convolutional neural network" [33].

Se han necesitado aplicar algunos cambios necesarios a las funciones definidas en el anterior tutorial para utilizar los vídeos recopilados en esta propuesta como datos de entrada. Este último implementa el modelo de red neuronal convolucional (2+1)D, y en el trabajo desarrollado, se ha incluido el modelo Inflated 3D CNN. A continuación, se mostrarán los pasos seguidos en el entrenamiento de estos dos modelos.

3.5.1. Configuración del entorno de trabajo

Antes de comenzar con la creación de los clasificadores, se deberán descargar e importar todas aquellas dependencias necesarias, entre las que se encuentran la librería TensorFlow, remotezip (usada para descargar contenidos de un archivo ZIP desde un servidor web) y OpenCv (biblioteca de visión artificial)

3.5.2. Carga de los datos

Al igual que en el tutorial de TensorFlow, los videos serán descargados desde un archivo ZIP alojado, en este caso, en un servidor alojado personal. En la figura **3.27** se puede ver la función encargada de esta tarea, con la cual se crean tres subconjuntos (uno de entrenamiento, uno de validación y otro de prueba) con las rutas de los vídeos descargados. Hay que destacar que el método de segmentación en estos tres subconjuntos ha sido modificado respecto al método original de TensorFlow, de manera que dicha división resulta en:

- 80 % de los vídeos se usarán en el entrenamiento.

- 10 % se usarán en la validación.

- 10 % de los vídeos destinados a la evaluación.

```
URL = 'http://dcastellanoo.com/penalti_videos.zip'
download_dir = pathlib.Path('./penalti_subset/')
subset_paths = download_subset(URL,
                        splits = {"train": 0.8, "val": 0.5, "test": 1},
                        download_dir = download_dir)
```

Ilustración 3.27: Descarga del conjunto de datos con las rutas de los vídeos

Los tres subconjuntos creados contienen las rutas de los vídeos, pero lo que necesitan los clasificadores basados en CNNs son imágenes, por lo que es necesario extraer los fotogramas de cada vídeo. Para ello, se hace uso de una clase generadora implementada en el tutorial seguido llamada *FrameGenerator()*. Esta última recibe como parámetros el subconjunto con las rutas de los vídeos y el número de *frames* a extraer de cada vídeo.

En la ilustración **3.28** se puede apreciar la función encargada de sacar "n" fotogramas del vídeo almacenado en la ruta proporcionado como parámetro. Esta ha sido ajustada para seleccionar y extraer los fotogramas a lo largo de toda la duración del vídeo. El parámetro *n_frames* se establecerá en 20 para cada vídeo, las dimensiones de cada fotograma será 224 X 224.

```
def frames_from_video_file(video_path, n_frames, output_size = (224,224)):
  """
    Creates frames from each video file present for each category.

    Args:
      video_path: File path to the video.
      n_frames: Number of frames to be created per video file.
      output_size: Pixel size of the output frame image.

    Return:
      An NumPy array of frames in the shape of (n_frames, height, width, channels).
  """
  # Read each video frame by frame
  result = []
  src = cv2.VideoCapture(str(video_path))

  video_length = src.get(cv2.CAP_PROP_FRAME_COUNT)

  frame_step = int(video_length / n_frames)

  src.set(cv2.CAP_PROP_POS_FRAMES, 0)
  # ret is a boolean indicating whether read was successful, frame is the image itself
  ret, frame = src.read()
  result.append(format_frames(frame, output_size))

  for _ in range(n_frames - 1):
    for _ in range(frame_step):
      ret, frame = src.read()
    if ret:
      frame = format_frames(frame, output_size)
      result.append(frame)
    else:
      result.append(np.zeros_like(result[0]))
  src.release()
  result = np.array(result)[..., [2, 1, 0]]

  return result
```

Ilustración 3.28: Función para crear los *frames* de cada vídeo

Entonces, para crear los conjuntos de datos se emplean las utilidades de TensorFlow con el generador de *frames* introducido anteriormente, lo cual se puede ver en la ilustración **3.29**

```
n_frames = 20
batch_size = 8

output_signature = (tf.TensorSpec(shape = (None, None, None, 3), dtype = tf.float32),
                    tf.TensorSpec(shape = (), dtype = tf.int16))

train_ds = tf.data.Dataset.from_generator(FrameGenerator(subset_paths['train'], n_frames, training=True),
                                          output_signature = output_signature)

train_ds = train_ds.batch(batch_size)

val_ds = tf.data.Dataset.from_generator(FrameGenerator(subset_paths['val'], n_frames),
                                        output_signature = output_signature)
val_ds = val_ds.batch(batch_size)

test_ds = tf.data.Dataset.from_generator(FrameGenerator(subset_paths['test'], n_frames),
                                         output_signature = output_signature)

test_ds = test_ds.batch(batch_size)
```

Ilustración 3.29: Creación de los conjuntos de datos de entrada de los clasificadores

En la ilustración **3.29** se puede ver que se especifica el tamaño de lote o *batch size* tendrá un valor de 8 en los tres datasets, lo que significa que en cada época de los procesos de entrenamiento, validación y *test* se utilizan 8 secuencias de 20 fotogramas. El parámetro *output_signature* indica la estructura y tipo de los tensores generadas en cada iteración.

3.5.3. Creación de los clasificadores

Una vez se han definido los conjuntos de datos de entrenamiento, validación y prueba, se procede a la implementación de los clasificadores (2+1)D e I3D. La biblioteca TensorFlow proporciona herramientas y utilidades para la creación de redes neuronales convolucionales, pudiendo crear modelos añadiendo las distintas capas (convolucionales, agrupamiento, etc.) de manera secuencial.

A continuación se exponen los clasificadores empleados:

- ✔ **(2+1)D CNN**
  Se utiliza en este caso el modelo implementado en el tutorial [33] de TensorFlow, el cual se trata de una (2+1)D ResNet18, lo que significa que esta CNN utiliza bloques residuales para facilitar el entrenamiento. En la figura **3.30** se puede ver la implementación de un bloque residual, el cual fue introducido con la arquitectura ResNet [17] y tiene como objetivo resolver el problema del posible estancamiento de la red neuronal durante el entrenamiento debido al desvanecimiento del gradiente de la función de perdida. Los

bloques residuales introducen un "salto"con el cual el gradiente se puede propague entre dos capas sin pasar por capas intermedias.

```
class ResidualMain(keras.layers.Layer):
  """
    Residual block of the model with convolution, layer normalization, and the
    activation function, ReLU.
  """
  def __init__(self, filters, kernel_size):
    super().__init__()
    self.seq = keras.Sequential([
        Conv2Plus1D(filters=filters,
                    kernel_size=kernel_size,
                    padding='same'),
        layers.LayerNormalization(),
        layers.ReLU(),
        Conv2Plus1D(filters=filters,
                    kernel_size=kernel_size,
                    padding='same'),
        layers.LayerNormalization()
    ])

  def call(self, x):
    return self.seq(x)
```

Ilustración 3.30: Implementación de un bloque residual

La convolución (2+1), presentada en la sección **2.3.3**.*(2+1)D CNN*, es implementada en la clase *Conv2Plus1D()* la cual se puede ver en la ilustración **3.31**. Al iniciar esta clase, se necesita un kernel tridimensional donde las dimensiones son (tiempo x ancho x alto). Teniendo esto en cuenta, la convolución implementada emplea una primera convolución 3D espacial donde se toman las dimensiones espaciales con un kernel (1 x ancho x alto); y otra convolución 3D temporal donde se toma el tiempo (tiempo x 1 x 1).

```
class Conv2Plus1D(keras.layers.Layer):
  def __init__(self, filters, kernel_size, padding):
    """
      A sequence of convolutional layers that first apply the convolution operation over the
      spatial dimensions, and then the temporal dimension.
    """
    super().__init__()
    self.seq = keras.Sequential([
        # Spatial decomposition
        layers.Conv3D(filters=filters,
                      kernel_size=(1, kernel_size[1], kernel_size[2]),
                      padding=padding),
        # Temporal decomposition
        layers.Conv3D(filters=filters,
                      kernel_size=(kernel_size[0], 1, 1),
                      padding=padding)
        ])

  def call(self, x):
    return self.seq(x)
```

Ilustración 3.31: Implementación de la convolución (2+1)D

En la ilustración **3.32** se puede apreciar el diagrama con las capas que conforman el modelo de inferencia en la (2+1)D CNN así como los tensores de entrada y de salida de estas. Para extraer características a diferentes escalas, se hace un redimensionamiento de los vídeos con *ResizeVideo()*.

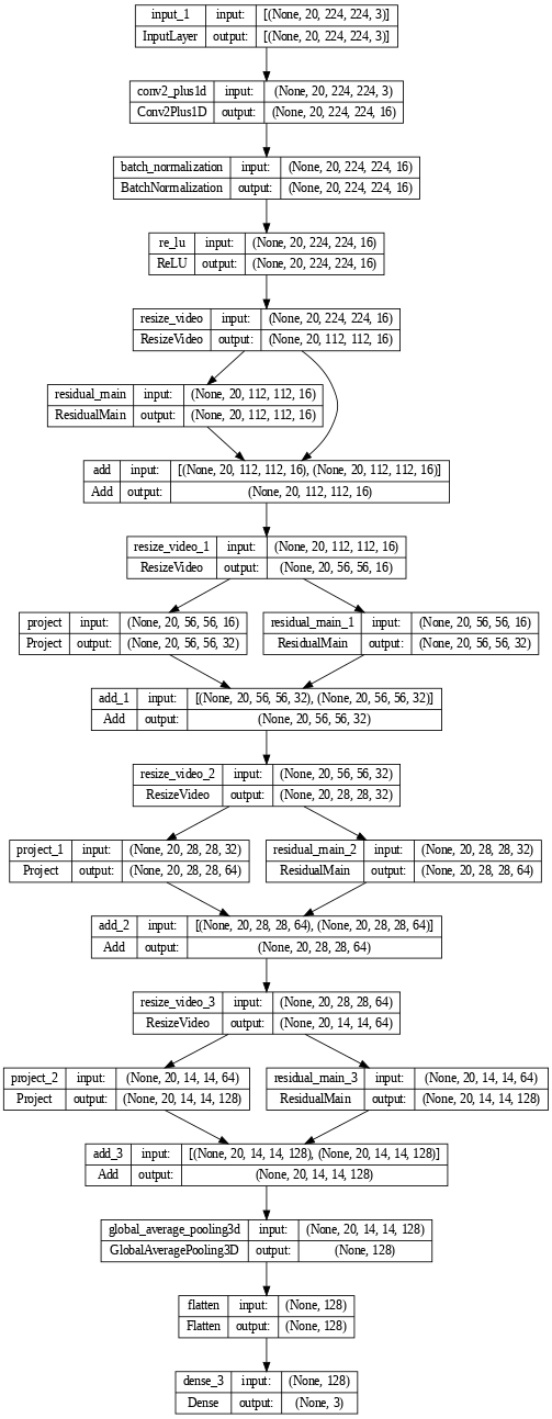


Ilustración 3.32: Diagrama del modelo de inferencia basándose en (2+1)D CNN

✔ **I3D CNN**

Este modelo ha sido implementación propia usando como guía el código fuente de este modelo [1] disponible públicamente en GitHub. En la figura **3.33** se enseña la implementación de la arquitectura I3D, arquitectura que se muestra en la imagen **2.25**.

```
class I3D(keras.layers.Layer):
  def __init__(self):
    super().__init__()

    self.conv3d_a1 = layers.Conv3D(filters=32, kernel_size=(7, 7, 7), strides=(2, 2, 2), padding='valid', activation='relu')
    self.maxpool3d_a = layers.MaxPool3D(pool_size=(1, 3, 3), strides=(1, 2, 2), padding='same')

    self.conv3d_b1 = layers.Conv3D(filters=64, kernel_size=(1, 1, 1), padding='same', activation='relu')
    self.conv3d_b2 = layers.Conv3D(filters=64, kernel_size=(3, 3, 3), padding='same', activation='relu')
    self.maxpool3d_b = layers.MaxPool3D(pool_size=(1, 3, 3), strides=(1, 2, 2), padding='same')

    self.inceptionmodule_c1 = InceptionModule(64, 96, 128, 16, 32, 32)
    self.inceptionmodule_c2 = InceptionModule(128, 128, 192, 32, 96, 64)
    self.maxpool3d_c = layers.MaxPool3D(pool_size=(3, 3, 3), strides=(2, 2, 2), padding='same')

    self.inceptionmodule_d1 = InceptionModule(192, 96, 208, 16, 48, 64)
    self.inceptionmodule_d2 = InceptionModule(160, 112, 224, 24, 64, 64)
    self.inceptionmodule_d3 = InceptionModule(128, 128, 256, 24, 64, 64)
    self.inceptionmodule_d4 = InceptionModule(112, 144, 288, 32, 64, 64)
    self.inceptionmodule_d5 = InceptionModule(256, 160, 320, 32, 128, 128)
    self.maxpool3d_d = layers.MaxPool3D(pool_size=(2, 2, 2), strides=(2, 2, 2), padding='same')

    self.inceptionmodule_e1 = InceptionModule(256, 160, 320, 32, 128, 128)
    self.inceptionmodule_e2 = InceptionModule(384, 192, 384, 48, 128, 128)

    self.avgpool3d_f = layers.AveragePooling3D(pool_size=(2, 7, 7), strides=(1, 1, 1), padding='valid')

  def call(self, inputs):
    x = self.conv3d_a1(inputs)
    x = self.maxpool3d_a(x)

    x = self.conv3d_b1(x)
    x = self.conv3d_b2(x)
    x = self.maxpool3d_b(x)

    x = self.inceptionmodule_c1(x)
    x = self.inceptionmodule_c2(x)
    x = self.maxpool3d_c(x)

    x = self.inceptionmodule_d1(x)
    x = self.inceptionmodule_d2(x)
    x = self.inceptionmodule_d3(x)
    x = self.inceptionmodule_d4(x)
    x = self.inceptionmodule_d5(x)
    x = self.maxpool3d_d(x)

    x = self.inceptionmodule_e1(x)
    x = self.inceptionmodule_e2(x)
    x = self.avgpool3d_f(x)

    return x
```

Ilustración 3.33: Implementación de la arquitectura Inflated 3D CNN

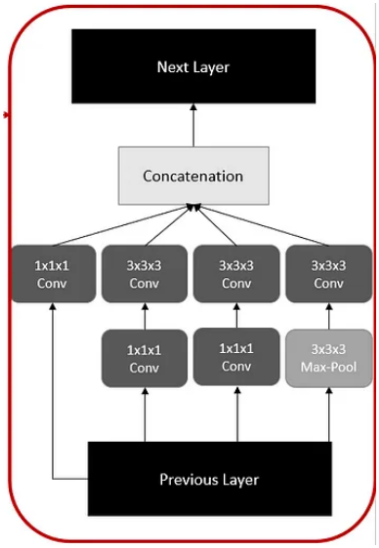


Ilustración 3.34: Estructura *Inception Module*

Este modelo emplea una estructura diseñada para la extracción de características a múltiples niveles llamada *Inception Module*. Este fue originalmente introducido en redes convolucionales 2D y ha sido adaptada para trabajar en el procesamiento de vídeos. En este módulo se concatenan las salidas de varias capas con diferentes tamaños de kernel en un solo tensor de salida. La estructura de capas de dicho módulo se muestra en la figura **3.34**. En la ilustración **3.35** se puede apreciar la implementación del *Inception Module*

```
class InceptionModule(keras.layers.Layer):
  def __init__(self, f1, f2, f3, f4, f5, f6):
    super().__init__()

    self.branch1 = layers.Conv3D(f1, kernel_size=(1, 1, 1), padding='same', activation='relu')

    self.branch2 = tf.keras.Sequential([
        layers.Conv3D(f2, kernel_size=(1, 1, 1), padding='same', activation='relu'),
        layers.Conv3D(f3, kernel_size=(3, 3, 3), padding='same', activation='relu')
    ])

    self.branch3 = tf.keras.Sequential([
        layers.Conv3D(f4, kernel_size=(1, 1, 1), padding='same', activation='relu'),
        layers.Conv3D(f5, kernel_size=(3, 3, 3), padding='same', activation='relu')
    ])

    self.branch4 = tf.keras.Sequential([
        layers.MaxPool3D(pool_size=(3, 3, 3), strides=(1, 1, 1), padding='same'),
        layers.Conv3D(f6, kernel_size=(1, 1, 1), padding='same', activation='relu')
    ])

  def call(self, inputs):
      branch1_output = self.branch1(inputs)
      branch2_output = self.branch2(inputs)
      branch3_output = self.branch3(inputs)
      branch4_output = self.branch4(inputs)

      output = tf.keras.layers.concatenate([branch1_output, branch2_output, branch3_output, branch4_output], axis=-1)
      return output
```

Ilustración 3.35: Clase *InceptionModule()*

3.5.4. Entrenamiento de los modelos

Ya construido el modelo, el siguiente paso es la compilación de este último. En dicho proceso se asigna la función de pérdida, el algoritmo de optimización y las métricas con las que evaluar el modelo.

En este caso, se ha usado como función de pérdida la entropía cruzada categórica dispersa o *Sparse Categorical Crossentropy*, la cual es ampliamente utilizada en problemas multiclase. Como optimizador, se ha seleccionado *Adaptive Moment Estimation* o Adam con una tasa de aprendizaje de 0.01. Este último es muy utilizado por su eficiencia y eficacia en el ajuste de pesos durante el proceso de entrenamiento. Por último, como métrica de evaluación se ha escogido la precisión, definida en la sección **4.1**. En la figura **3.36** se puede ver la función que lleva a cabo la compilación del modelo.

```
model.compile(loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = keras.optimizers.Adam(learning_rate = 0.01),
              metrics = ['accuracy'])
```

Ilustración 3.36: Compilación de modelo clasificador de vídeos

Solo resta entrenar el modelo, para lo cual se ejecuta el método *fit()*. A este último se le pasan como parámetros el conjunto de entrenamiento, el número de épocas o iteraciones de entrenamiento y una función encargada de parar el entrenamiento llegada cierta condición.

```
es = EarlyStopping(monitor='accuracy', mode='max', verbose=1, patience=8, restore_best_weights=True)

history = model.fit(x = train_ds,
                    epochs = 50,
                    validation_data = val_ds,
                    callbacks = [es])
```

Ilustración 3.37: Entrenamiento de modelo clasificador de vídeos

Como se puede ver en la ilustración **3.37**, el número de épocas se establece en 50, pudiendo terminar el entrenamiento si la precisión sobre los datos de entrenamiento no mejora en las últimas 8 iteraciones. Este condición es añadida a al entrenamiento con la función *EarlyStopping()* de la librería Keras.

Capítulo 4

Evaluación de resultados

En este capítulo se mostrarán y valorarán los resultados obtenidos en las distintas tareas trabajadas en este trabajo. Para ello, en primer lugar se introducirán las métricas empleadas en la evaluación de los modelos.

4.1. Métricas de evaluación de resultados

Existen muchas métricas que permiten cuantificar y comparar el rendimiento de los modelos de inferencia desarrollados. Las utilizadas en la evaluación de los resultados de esta propuesta son métricas asociadas a la clasificación de los datos de entrada, que se definirán en el caso de un problema multiclase:

- ✔ Verdadero positivo o *True Positive* (TP): número de muestras clasificadas correctamente en la clase i por el modelo.
- ✔ Verdadero negativo o *True Negative* (TN): número de muestras que no pertenecen a la clase i y que fueron clasificadas o predichas en otra clase distinta a i.
- ✔ Falso positivo o *False Positive* (FP): número de muestras clasificadas o predichas incorrectamente en la clase i.
- ✔ Falso negativo o *False Negative* (FN): número de muestras que pertenecen a la clase i y que fueron clasificadas o predichas en la clase i.

Teniendo en cuenta los conceptos anteriores, se define la métrica **precisión** o ***precision***. La precisión para una clase i proporciona la medida de cuántas de las clasificaciones/predic-

ciones positivas son correctas. Su fórmula para una clase i:

$$\text{Precisión para la clase } i = \frac{TP}{TP + FP}$$

Se define también el ***recall*** o textbfexhaustividad, que, para una clase i, mide el número de verdaderos positivos sobre el total de entre verdaderos positivos y falsos negativos. Entonces, la fórmula del *recall*:

$$\textit{Recall} \text{ para la clase } i = \frac{TP}{TP + FN}$$

La métrica de ***F1-Score*** es una medida que combina los valores de la precisión y el *recall*. Esta métrica es útil para representar el balance entre la precisión y el *recall*. Su fórmula es la siguiente:

$$\textit{F1-Score} \text{ para la clase } i = 2 \cdot \frac{\textit{Precisión} \cdot \textit{Recall}}{\textit{Precisión} + \textit{Recall}}$$

Como última medida, se encuentra la **exactitud** o *accuracy* total del modelo. Representa la proporción de muestras clasificadas correctamente sobre el total de muestras del conjunto de datos. La exactitud en problemas en los que el número de instancias de cada clase están desbalanceados puede resultar engañosa, ya que en puede darse el caso de que el modelo obtenga una exactitud alta pues clasifica todas las muestras con la clase que más instancias tenga.

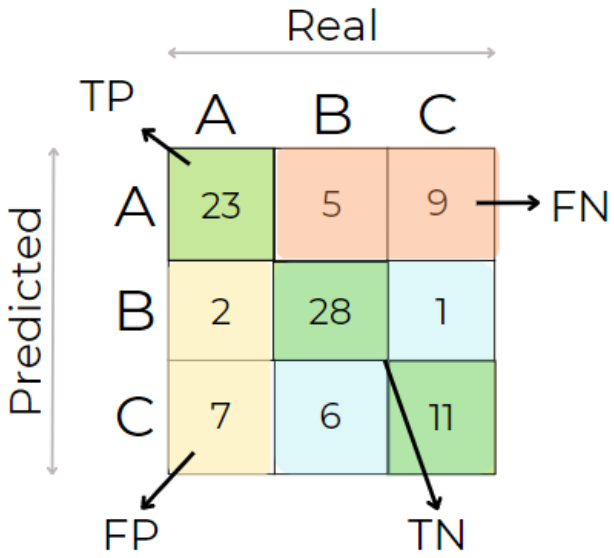


Ilustración 4.1: Ejemplo de matriz de confusión

Por último, se introduce la matriz de confusión, la cual se emplea con frecuencia en la evaluación de modelos de inferencia. Se trata de una matriz cuadrada en la que las filas representan las clases reales y las columnas las clases predichas por el clasificador. En esta, los valores de la diagonal representan las clasificaciones correctas o verdaderos positivos. En la ilustración **4.1** se puede ver un ejemplo de una matriz de confusión, donde se puede ver coloreadas en verde las celdas de verdaderos positivos. Se pueden apreciar también en dicha imagen distinguidos los valores TP, TN, FN y FP en el caso de la clase A.

4.2. Resultados análisis predictivo

A continuación se mostrarán los resultados de todos los clasificadores empleados en la tarea de predicción del resultado de un lanzamiento. Se procederá dividiendo esta sección entre los tres subconjuntos de datos alternando las características usadas en el entrenamiento; y en cada subconjunto se separarán los clasificadores según de qué tipo sean (lineal, no lineal y basados en redes neuronales), realizando una comparativa entre los que mejores resultados ofrecen de cada tipo.

En las sucesivas tablas con las métricas de evaluación de los clasificadores, se destacarán con color verde los mejores resultados dentro de cada comparativa. Como regla general, se considerarán como mejores o con mejor rendimiento aquellos clasificadores tengan un mayor número de métricas destacadas. Además, se tomará la métrica "F1-Score" se tomará como la más relevante y la exactitud como la de menos importancia.

El conjunto de datos empleado para la evaluación de los clasificadores utilizados en esta tarea consta con 68 muestras cuyo resultado es "Gol", 43 son etiquetadas como "Parada" y solamente 3 acciones como "Fallo".

Además, verán las puntuaciones en el proceso de selección de características así como los resultados obtenidos por el mejor modelo de inferencia, utilizando las características con una importancia mayor a 1.

4.2.1. Características relacionadas con el lanzador + balón

4.2.1.1. Clasificadores lineales

Cuadro 4.1: Métricas de los clasificadores lineales (características lanzador + balón)

	Regresión Logística	**Naive Bayes**	**SVM lineal**
Accuracy	59.65 %	59.65 %	60.53 %
Precision (Gol)	60.36 %	59.64 %	60.71 %
Recall (Gol)	98.53 %	100.0 %	100.0 %
F1-Score (Gol)	74.86 %	74.72 %	75.56 %
Precision (Parada)	33.33 %	0.0 %	50.0 %
Recall (Parada)	2.32 %	0.0 %	23.3 %
F1-Score (Parada)	4.34 %	0.0 %	4.44 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %

Con las métricas que se pueden ver en la tabla **4.1**, se puede inferir que las instancias del problema de clasificación abordado no son separables de manera lineal. La clase de "Fallo" no es clasificada correctamente por ninguno de los clasificadores.

Entre los clasificadores lineales entrenados y evaluados con las características del lanzador y del balón, el mejor es el que emplea el algoritmo SVM con función kernel lineal.

4.2.1.2. Clasificadores no lineales

Cuadro 4.2: Métricas de los clasificadores no lineales (características lanzador + balón)

	RBF SVM	**K-Nearest Neighbours**	**Decission Tree**
Accuracy	71.93 %	67.54 %	62.28 %
Precision (Gol)	70.45 %	73.17 %	63.0 %
Recall (Gol)	91.18 %	88.23 %	92.64 %
F1-Score (Gol)	79.49 %	80.0 %	75.0 %
Precision (Parada)	76.92 %	68.0 %	57.14 %
Recall (Parada)	46.51 %	39.53 %	18.60 %
F1-Score (Parada)	57.97 %	50.0 %	28.07 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %

Los clasificadores no lineales, mostrados en el cuadro **4.2**, ofrecen mejores resultados que los lineales. No obstante, no son capaces de clasificar correctamente ninguno de los lanza-

mientos fallidos.

Se ha seleccionado como mejor clasificador no lineal al que utiliza SVM con función kernel de base radial ya que es el que, fijándonos en el *F1-Score* para la clase "Parada", mejor clasifica esta última con gran diferencia. Se considera la diferencia entre las puntuaciones para la clase "Gol" entre RBF SVM y K-vecinos más cercanos como despreciable.

4.2.1.3. Clasificadores basados en redes neuronales

Cuadro 4.3: Métricas de los clasificadores MLPs (características lanzador + balón)

	MLP (10, 10)	**MLP (50, 20)**	**MLP (50, 20, 10)**	**MLP (100, 50, 20, 10)**
Accuracy	62.28 %	64.91 %	64.91 %	64.04 %
Precision (Gol)	63.0 %	64.95 %	64.58 %	64,58 %
Recall (Gol)	92.65 %	92.64 %	91.18 %	91.18 %
F1-Score (Gol)	75.0 %	76.36 %	75.61 %	75.61 %
Precision (Parada)	57.14 %	64.70 %	66.67 %	61.11 %
Recall (Parada)	18.60 %	25.58 %	27.90 %	25.58 %
F1-Score (Parada)	28.07 %	36.67 %	29.34 %	36.07 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %	0.0 %

Al igual que el resto de clasificadores, los basados en redes neuronales obtienen resultados nulos para el resultado "Fallo".

Valorando *F1-Score* como la métrica con mayor relevancia, se toma como mejor clasificador basado en red neuronal multicapa aquel que tiene dos capas ocultas, la primera con 50 neuronas y la segunda con 20.

4.2.1.4. Comparativa entre los mejores clasificadores

Cuadro 4.4: Comparativa entre los mejores clasificadores (características lanzador + balón)

	SVM lineal	**RBF SVM**	**MLP (50, 20)**
Accuracy	60.53 %	71.93 %	64.91 %
Precision (Gol)	60.71 %	70.45 %	64.95 %
Recall (Gol)	100.0 %	91.18 %	92.64 %
F1-Score (Gol)	75.56 %	79.49 %	76.36 %
Precision (Parada)	50.0 %	76.92 %	64.70 %
Recall (Parada)	23.3 %	46.51 %	25.58 %
F1-Score (Parada)	4.44 %	57.97 %	36.67 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %

En la tabla **4.4** se muestra una comparativa entre los mejores clasificadores de cada tipo. Todos ellos coinciden en no clasificar ninguna muestra como "Fallo", lo cual se explica en parte por la falta de acciones pertenecientes a dicho resultado.

El mejor clasificador entre los tres mostrados en el cuadro **4.4** es el no lineal que implementa el algoritmo SVM con función kernel de base radial. En la figura **4.2** se puede ver la matriz de confusión de dicho clasificador para el conjunto de datos de evaluación. En esta matriz se puede ver como el clasificador únicamente predice las muestras entre "Gol" y "Parada".

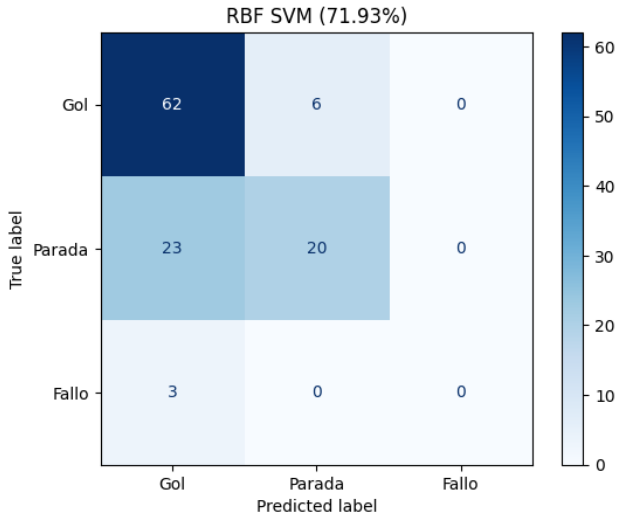


Ilustración 4.2: Matriz de confusión clasificador RBF SVM (características lanzador + balón)

4.2.1.5. Selección de características

Tras utilizar la función *SelectKBest()* de la librería Scikit-learn, las puntuaciones del grado de relevancia de las características relacionadas con el lanzador y el balón se muestran en la siguiente tabla:

Cuadro 4.5: Importancia de las características (características lanzador + balón)

Posición	**Característica**	**Importancia**
1	Bote	33.7058
2	Balón en la portería	2.7281
3	Brazo lanzador	1.3119
4	Amagos	0.6679
5	Altura brazo lanzador	0.1343

Se puede ver como las características que más influyen en el resultado de una acción, según el análisis univariante llevado a cabo por *SelectKBest()*, son el bote del balón, por donde entra o va dirigido el balón a la portería y el brazo del lanzador. Destaca la diferencia de importancia entre la característica del bote respecto a las demás. Esto se debe a que, en los casos de parada del portero, el balón no suele botar antes de que este último efectúe la parada.

A continuación se verán los resultados de aplicar el clasificador RBF SVM con las 3 características más importantes, donde se puede apreciar una pérdida sustancial del rendimiento del clasificador.

Cuadro 4.6: Métricas del clasificador RBF SVM con 3 características (características lanzador + balón)

	RBF SVM con 3 características
Accuracy	62.28 %
Precision (Gol)	61.68 %
Recall (Gol)	97.06 %
F1-Score (Gol)	75.43 %
Precision (Parada)	71.43 %
Recall (Parada)	11.63 %
F1-Score (Parada)	20.0 %
Precision (Fallo)	0.0 %
Recall (Fallo)	0.0 %
F1-Score (Fallo)	0.0 %

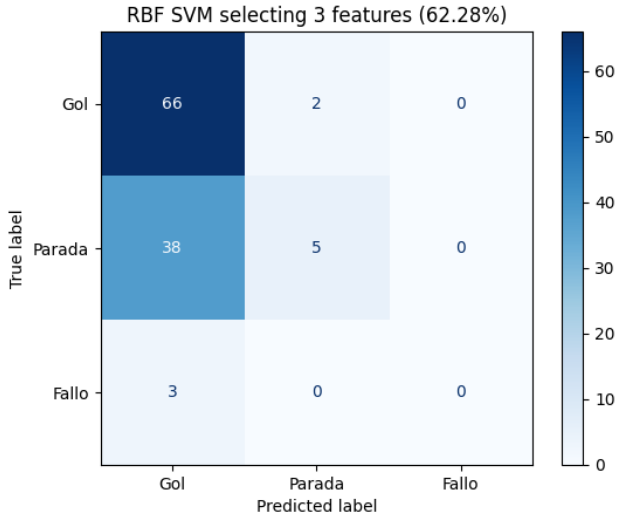


Ilustración 4.3: Matriz de confusión clasificador RBF SVM con 3 características(características lanzador + balón)

4.2.2. Características relacionadas con el portero + balón

4.2.2.1. Clasificadores lineales

Cuadro 4.7: Métricas de los clasificadores lineales (características portero + balón)

	Regresión Logística	**Naive Bayes**	**SVM lineal**
Accuracy	64.04 %	60.53 %	60.53 %
Precision (Gol)	63.36 %	60.18 %	60.95 %
Recall (Gol)	92.65 %	100.0 %	94.12 %
F1-Score (Gol)	74.45 %	75.14 %	73.99 %
Precision (Parada)	66.67 %	100.0 %	55.56 %
Recall (Parada)	23.26 %	2.33 %	11.63 %
F1-Score (Parada)	34.48 %	4.55 %	19.23 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %

En la tabla **4.7** se puede apreciar el bajo rendimiento de los clasificadores lineales en el caso del subconjunto de datos con las características del portero y las del balón. Entre estos, el que mejores resultados proporciona es el que implementa el algoritmo de regresión logística multinomial.

4.2.2.2. Clasificadores no lineales

Cuadro 4.8: Métricas de los clasificadores no lineales (características portero + balón)

	RBF SVM	**K-Nearest Neighbours**	**Decission Tree**
Accuracy	68.42 %	62.28 %	63.16 %
Precision (Gol)	67.71 %	63.64 %	76.19 %
Recall (Gol)	95.59 %	92.65 %	70.59 %
F1-Score (Gol)	79.27 %	75.45 %	73.28 %
Precision (Parada)	72.22 %	53.33 %	62.16 %
Recall (Parada)	30.23 %	18.60 %	53.49 %
F1-Score (Parada)	42.62 %	27.59 %	57.50 %
Precision (Fallo)	0.0 %	0.0 %	7.14 %
Recall (Fallo)	0.0 %	0.0 %	33.33 %
F1-Score (Fallo)	0.0 %	0.0 %	11.76 %

Se puede ver que los clasificadores no lineales resultan ser mejores que los lineales en este problema. Vemos que el modelo que utiliza el algoritmo de árbol de decisión es el primero en clasificar al menos una muestra de la clase “Fallo” correctamente, y este modelo es el seleccionado como el mejor entre los no lineales.

4.2.2.3. Clasificadores basados en redes neuronales

Cuadro 4.9: Métricas de los clasificadores MLPs (características portero + balón)

	MLP (10, 10)	**MLP (50, 20)**	**MLP (50, 20, 10)**	**MLP (100, 50, 20, 10)**
Accuracy	72.81 %	72.81 %	77.19 %	71.05 %
Precision (Gol)	75.32 %	77.03 %	79.22 %	74.03 %
Recall (Gol)	85.29 %	83.82 %	89.71 %	83.82 %
F1-Score (Gol)	80.0 %	80.28 %	84.14 %	78.62 %
Precision (Parada)	69.44 %	70.27 %	77.14 %	68.57 %
Recall (Parada)	58.14 %	60.47 %	62.79 %	55.81 %
F1-Score (Parada)	63.29 %	65.0 %	69.23 %	61.53 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %	0.0 %

Por último en cuanto a los clasificadores implementados para este subconjunto de datos, se encuentran los basados en perceptrones multicapa. Los resultados de estos pueden ser vistos en la tabla **4.9**, donde el MLP con tres capas ocultas destaca sobre el resto.

4.2.2.4. Comparativa entre los mejores clasificadores

Cuadro 4.10: Comparativa entre los mejores clasificadores (características portero + balón)

	Regresión Logística	**Decission Tree**	**MLP (50, 20, 10)**
Accuracy	64.04 %	63.16 %	77.19 %
Precision (Gol)	63.36 %	76.19 %	79.22 %
Recall (Gol)	92.65 %	70.59 %	89.71 %
F1-Score (Gol)	74.45 %	73.28 %	84.14 %
Precision (Parada)	66.67 %	62.16 %	77.14 %
Recall (Parada)	23.26 %	53.49 %	62.79 %
F1-Score (Parada)	34.48 %	57.50 %	69.23 %
Precision (Fallo)	0.0 %	7.14 %	0.0 %
Recall (Fallo)	0.0 %	33.33 %	0.0 %
F1-Score (Fallo)	0.0 %	11.76 %	0.0 %

A pesar de que el clasificador no lineal de árbol de decisión es capaz de clasificar al menos una ocurrencia de la clase “Fallo”, se opta por el modelo basado en una red neuronal multicapa al superar con diferencia en el resto de métricas al no lineal.

Se muestra en la ilustración **4.4** la matriz de confusión del clasificador MLP con tres capas en el caso del subconjunto de datos de características relacionadas con el portero y el balón. En este caso, el modelo clasifica incorrectamente dos muestras como “Fallo”.

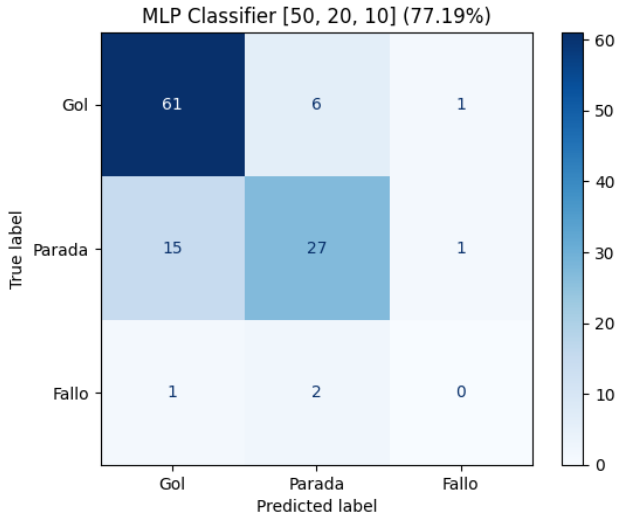


Ilustración 4.4: Matriz de confusión clasificador (MLP 50, 20, 10) (características portero + balón)

4.2.2.5. Selección de características

Cuadro 4.11: Importancia de las características (características portero + balón)

Posición	Característica	Importancia
1	Bote	33.7058
2	Pierna derecha portero	4.0990
3	Balón en la portería	2.7281
4	Pierna izquierda portero	2.4895
5	Brazo derecho portero	2.4724
6	Acción portero	1.8964
7	Brazo izquierdo portero	0.9204
8	Lado portero	0.6259
9	Portero adelantado	0.1948

En la tabla **4.11** se muestran las importancia de las características tras aplicar la función *SelectKBest()* a este subconjunto. Son 6 características las que sobrepasan una puntuación de 1, y estas serán las seleccionadas para entrenar y evaluar el modelo MLP con tres capas cuyos resultados se pueden ver a continuación:

Cuadro 4.12: Métricas del clasificador MLP (50, 20, 10) con 6 características (características portero + balón)

	MLP (50, 20, 10) con 6 características
Accuracy	61.40 %
Precision (Gol)	64.77 %
Recall (Gol)	83.82 %
F1-Score (Gol)	73.07 %
Precision (Parada)	52.0 %
Recall (Parada)	30.23 %
F1-Score (Parada)	38.24 %
Precision (Fallo)	0.0 %
Recall (Fallo)	0.0 %
F1-Score (Fallo)	0.0 %

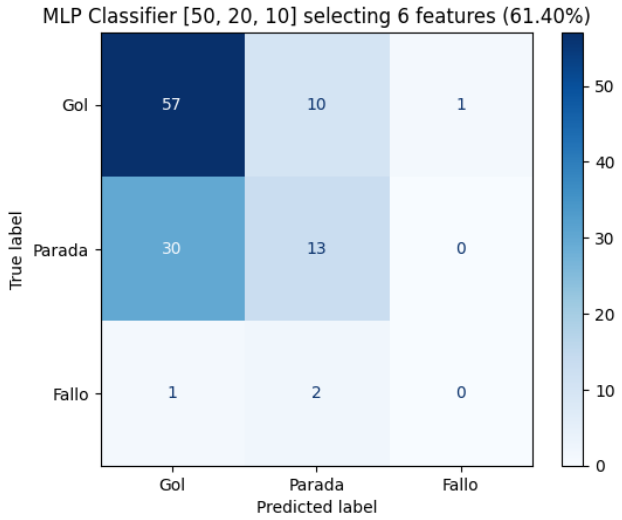


Ilustración 4.5: Matriz de confusión clasificador MLP (50, 20, 10) con 6 características(características portero + balón)

Se aprecia como este último clasificador, empleando las 6 características más importantes según la función *SelectKBest()*, se pierde casi un 16 % de exactitud, lo que indica que el método escogido para la selección de características puede no llegar a ser el mejor.

4.2.3. Todas las características

4.2.3.1. Clasificadores lineales

Cuadro 4.13: Métricas de los clasificadores lineales (todas las características)

	Regresión Logística	**Naive Bayes**	**SVM lineal**
Accuracy	64.04 %	59.65 %	61.40 %
Precision (Gol)	64.29 %	59.65 %	62.63 %
Recall (Gol)	92.65 %	100.0 %	91.18 %
F1-Score (Gol)	75.90 %	74.73 %	74.25 %
Precision (Parada)	62.50 %	0.0 %	53.33 %
Recall (Parada)	23.26 %	0.0 %	18.60 %
F1-Score (Parada)	33.90 %	0.0 %	27.59 %
Precision (Fallo)	0.0 %	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %	0.0 %

Utilizando el conjunto de datos con todas las características, los clasificadores lineales obtienen resultados similares a los obtenidos con los otros dos conjuntos de datos, indicando que las muestras no son separables linealmente. Al igual que con el anterior subconjuntos (características portero y balón), el mejor modelo es el de regresión logística multinomial.

4.2.3.2. Clasificadores no lineales

Cuadro 4.14: Métricas de los clasificadores no lineales (todas las características)

	RBF SVM	**K-Nearest Neighbours**	**Decission Tree**
Accuracy	71.93 %	62.28 %	64.04 %
Precision (Gol)	68.0 %	62.75 %	79.37 %
Recall (Gol)	100.0 %	94.12 %	73.53 %
F1-Score (Gol)	80.95 %	75.29 %	76.34 %
Precision (Parada)	100.0 %	58.33 %	61.11 %
Recall (Parada)	30.23 %	16.28 %	51.16 %
F1-Score (Parada)	46.43 %	25.45 %	55.70 %
Precision (Fallo)	100.0 %	0.0 %	6.67 %
Recall (Fallo)	33.33 %	0.0 %	33.33 %
F1-Score (Fallo)	50.0 %	0.0 %	11.11 %

En el cuadro **4.14** se puede apreciar que, en cuanto a la predicción de resultados en el conjunto de datos original, el clasificador no lineal que mejor resultados ofrece es el que utiliza el algoritmo SVM con función kernel de base radial. Este último es el que mejores métricas ha obtenido en cuanto a la clase “Fallo” hasta ahora.

4.2.3.3. Clasificadores basados en redes neuronales

Cuadro 4.15: Métricas de los clasificadores MLPs (todas las características)

	MLP (10, 10)	**MLP (50, 20)**	**MLP (50, 20, 10)**	**MLP (100, 50, 20, 10)**
Accuracy	70.18 %	81.58 %	78.95 %	82.46 %
Precision (Gol)	72.73 %	80.25 %	77.78 %	82.28 %
Recall (Gol)	82.35 %	95.59 %	92.65 %	95.59 %
F1-Score (Gol)	77.24 %	87.25 %	84.56 %	88.44 %
Precision (Parada)	65.71 %	84.38 %	83.87 %	84.85 %
Recall (Parada)	53.49 %	62.79 %	60.47 %	65.12 %
F1-Score (Parada)	58.97 %	72.0 %	70.27 %	73.68 %
Precision (Fallo)	50.0 %	100.0 %	50.0 %	50.0 %
Recall (Fallo)	33.33 %	33.33 %	33.33 %	33.33 %
F1-Score (Fallo)	40.0 %	50.0 %	40.0 %	40.0 %

En el caso de los clasificadores basados en redes neuronales para el conjunto total de datos, se puede ver en la tabla *4.15* como dos de estos superan el 80 % de exactitud. Se ha seleccionado el modelo MLP de mayor complejidad (4 capas ocultas) como el mejor de las cuatro configuraciones debido a que es la que mejores métricas proporciona en el cómputo global.

No obstante, hay que destacar lo cerca que se queda el clasificador MLP con dos capas ocultas de 50 y 20 neuronas de los resultados del modelo MLP de 4 capas. Esta red neuronal multicapa, al contar con menor número de capas y de neuronas, su entrenamiento e inferencia tiene una duración menor.

4.2.3.4. Comparativa entre los mejores clasificadores

Cuadro 4.16: Comparativa entre los mejores clasificadores (todas las características)

	Regresión Logística	**RBF SVM**	**MLP (100, 50, 20, 10)**
Accuracy	64.04 %	71.93 %	82.46 %
Precision (Gol)	64.29 %	68.0 %	82.28 %
Recall (Gol)	92.65 %	100.0 %	95.59 %
F1-Score (Gol)	75.90 %	80.95 %	88.44 %
Precision (Parada)	62.50 %	100.0 %	84.85 %
Recall (Parada)	23.26 %	30.23 %	65.12 %
F1-Score (Parada)	33.90 %	46.43 %	73.68 %
Precision (Fallo)	0.0 %	100.0 %	50.0 %
Recall (Fallo)	0.0 %	33.33 %	33.33 %
F1-Score (Fallo)	0.0 %	50.0 %	40.0 %

Se puede apreciar en el cuadro **4.16**, entre los mejores clasificadores de cada tipo, destaca el que se basa en perceptrón multicapa con 4 capas ocultas. A continuación se muestra la matriz de confusión obtenida en el conjunto de evaluación por este último modelo.

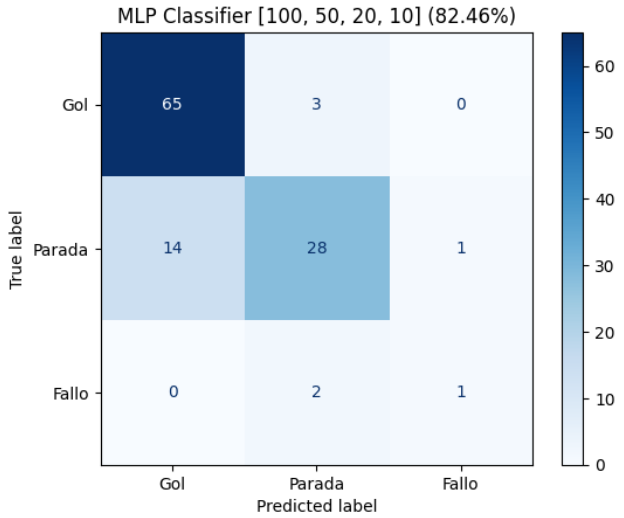


Ilustración 4.6: Matriz de confusión clasificador (MLP 100, 50, 20, 10) (todas las características)

Observando la figura **4.6**, se puede ver que es la matriz de confusión que mejores resultados ofrece entre las presentadas hasta ahora. En la diagonal de esta matriz se concentra el mayor número de muestras, lo que es un buen indicador de que el modelo de inferencia empleado presenta un buen desempeño en el conjunto de evaluación.

4.2.3.5. Selección de características

Cuadro 4.17: Importancia de las características (todas las características)

Posición	Característica	Importancia
1	Bote	33.7058
2	Pierna derecha portero	4.0990
3	Balón en la portería	2.7281
4	Pierna izquierda portero	2.4895
5	Brazo derecho portero	2.4724
6	Acción portero	1.8964
7	Brazo lanzador	1.3119
8	Brazo izquierdo portero	0.9204
9	Amagos	0.6679
10	Lado portero	0.6259
11	Portero adelantado	0.1948
12	Altura brazo lanzador	0.1343

Del cuadro **4.17** se puede extraer que, según las métricas estadísticas univariantes de la función *SelectKBest()*, las características del lanzador quedan en un segundo plano en cuanto al grado de relevancia en la predicción del resultado. Son 7 las características que superan una puntuación de 1, entre las cuales solo se encuentra una relacionada con el lanzador. Se utilizan dichas características para el entrenamiento y evaluación de un modelo basado en un MLP de 4 capas ocultas.

Cuadro 4.18: Métricas del clasificador MLP (100, 50, 20, 10) con 7 características (todas las características)

	MLP (100, 50, 20, 10) con 7 características
Accuracy	74.56 %
Precision (Gol)	74.70 %
Recall (Gol)	91.18 %
F1-Score (Gol)	82.12 %
Precision (Parada)	79.31 %
Recall (Parada)	53.49 %
F1-Score (Parada)	63.89 %
Precision (Fallo)	0.0 %
Recall (Fallo)	0.0 %
F1-Score (Fallo)	0.0 %

Se puede ver en los resultados de la tabla **4.18** un pérdida sustancial del rendimiento del clasificador al utilizar únicamente la 7 características más importantes, lo que denota nuevamente que método de selección de características a lo mejor no es el más óptimo.

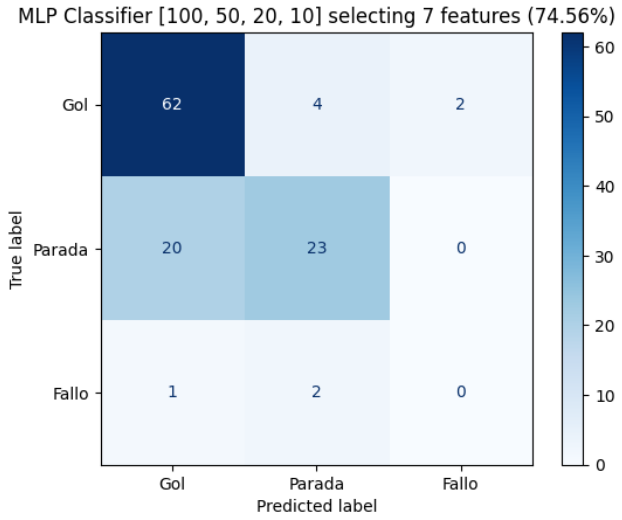


Ilustración 4.7: Matriz de confusión clasificador MLP (100, 50, 20, 10) con 7 características (todas las características)

4.3. Resultados clasificación de vídeo

A continuación se mostrarán los resultados obtenidos con los dos modelos de inferencia basados en CNNs entrenados en esta propuesta. Al igual que con los resultados del análisis predictivo, se utilizarán las métricas *precision*, *recall* y *F1-Score* para medir el rendimiento de ambos clasificadores.

Cuadro 4.19: Métricas del clasificadores de vídeo

	(2+1)D CNN	**I3D CNN**
Accuracy	67.24 %	67.24 %
Precision (Gol)	67.24 %	67.24 %
Recall (Gol)	100.0 %	100.0 %
F1-Score (Gol)	80.41 %	80.41 %
Precision (Parada)	0.0 %	0.0 %
Recall (Parada)	0.0 %	0.0 %
F1-Score (Parada)	0.0 %	0.0 %
Precision (Fallo)	0.0 %	0.0 %
Recall (Fallo)	0.0 %	0.0 %
F1-Score (Fallo)	0.0 %	0.0 %

En la tabla **4.19** se pueden ver los resultados tanto del modelo basado en la red neuronal convolucional (2+1)D como el que implementa la red Inflated 3D. Como se puede ver, los modelos no han sido capaces de clasificar correctamente las acciones, ya que todas estas son clasificadas por ambos modelos en la clase "Gol", que resulta ser la clase con mayor número de muestras. Esto último se puede ver en la matriz de confusión de la ilustración **4.8**, en la cual todas los vídeos se predicen como goles. Esta matriz es la misma para ambos modelos.

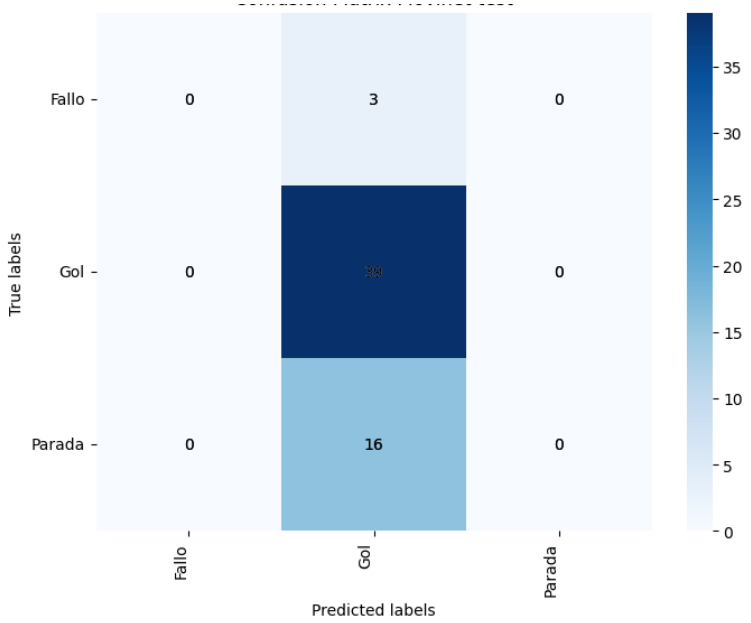


Ilustración 4.8: Matriz de confusión clasificadores de vídeo

Los resultados negativos obtenidos por estos modelos sugieren que las características del problema requieren otra manera de manejar los vídeos, otro tipo de modelos de clasificación de vídeos o que se ha llevado a cabo una incorrecta implementación de los clasificadores presentados.

Capítulo 5

Conclusiones y trabajo futuro

En el presente trabajo de fin de título se han trabajado en las dos tareas propuestas: análisis predictivo y clasificación de vídeos en el contexto de un lanzamiento de penalti en el balonmano. Para ello, se han cumplimentado los objetivos planteados inicialmente:

- ✔ La recopilación de un conjunto de datos de vídeos de tiros de 7 metros, llegando a reunir un total de 569 vídeos de los cuales 387 son goles, 156 son paradas y 26 son fallos. Estos fueron obtenidos tras la extracción de las acciones de penalti de vídeos de partidos completos o sus resúmenes con momentos destacados.

- ✔ Se ha llevado a cabo un proceso de anotación manual de la características relevantes de cada acción de lanzamiento de 7 metros recopilada, diferenciando 12 características distintas sin contar el resultado. Además, se han dividido dichas características en 3 tipos, la relacionadas con el lanzador, con el balón y con el portero.

- ✔ Tanto para la tarea de análisis predictivo como la de clasificación de vídeos, se han investigado distintos modelos y algoritmos, decidiendo los clasificadores ofrecidos por la biblioteca Scikit-learn en el primer de los casos; y los modelos clasificación de vídeos y reconocimiento de acciones (2+1)D CNN e I3D CNN.

- ✔ Se han creado dos documentos de Google Colab en los que se ha trabajado en la implementación y entrenamiento de los modelos propuestos.

- ✔ Por último, se ha realizado un análisis de los resultados obtenidos haciendo uso de métricas frecuentemente utilizadas en la evaluación de modelos de inferencia como los desarrollados en la propuesta.

A continuación se adjuntan los enlaces a los Jupyter Notebooks de Google Colab en los que se ha trabajado:

✔ Google Colab del análisis predictivo

✔ Google Colab de la clasificación de vídeo

En relación al análisis predictivo desarrollado, ,la hipótesis planteada en el apartado **3.4.3** se ha confirmado, tal y como se puede ver en el cuadro **5.1**. En esta última se planteaba que utilizando únicamente las características relacionadas con el lanzador del penalti y el balón, un clasificador obtendría un peor rendimiento que otro que emplease las características relacionadas con el portero y el balón.

Cuadro 5.1: Comparativa de los resultados entre los datos

	Lanzador+balón	**Portero+balón**	**Todas las características**
Accuracy	71.93 %	77.19 %	82.46 %
Precision (Gol)	70.45 %	79.22 %	82.28 %
Recall (Gol)	91.18 %	89.71 %	95.59 %
F1-Score (Gol)	79.49 %	84.14 %	88.44 %
Precision (Parada)	76.92 %	77.14 %	84.85 %
Recall (Parada)	46.51 %	62.79 %	65.12 %
F1-Score (Parada)	57.97 %	69.23 %	73.68 %
Precision (Fallo)	0.0 %	0.0 %	50.0 %
Recall (Fallo)	0.0 %	0.0 %	33.33 %
F1-Score (Fallo)	0.0 %	0.0 %	40.0 %

En el proceso de evaluación de los modelos utilizados para predecir el resultado de un lanzamiento, se ha evidenciado que los clasificadores lineales han obtenido pésimos resultados para los tres subconjuntos empleados, lo que indica que el problema de clasificación presentado es no lineal. Exceptuando el subconjunto con las características del lanzador y el balón, los clasificadores basados en redes neuronales multicapa ofrecen un mejor rendimiento que el resto, aunque estos requieren más tiempo computacional.

Además, en esta primera parte de la propuesta, se ha realizado un proceso de selección de características para determinar cuáles de las anotadas tienen mayor relevancia en la predicción del resultado de la acción. Se ha observado que las características relacionadas con el portero tienen mayor importancia en comparación a las relacionadas con el lanzador. Esto último sugiere que las acciones realizadas por el guardameta influyen de manera más significativa que las realizadas por el lanzador.

Por otra lado, en la parte de este trabajo que busca la clasificación de vídeos de penalti

en sus correspondientes resultados, los resultados obtenidos no fueron para nada lo esperado. Ambos modelos de redes convolucionales desarrollados clasificaron todos los vídeos como goles. Como se mencionó anteriormente, esto puede deberse a varios factores, entre los que se encuentran la posible incorrecta implementación de los modelos, a necesidad de utilizar otro tipo de modelos para las características del problema o posibles errores en el manejo de los datos de entrada, como una mala resolución utilizada o extracción de fotogramas.

Se propone que la principal razón del pésimo rendimiento de los modelos de inferencia sea la naturaleza de las características de la acción del lanzamiento. Todo lanzamiento, sea gol, parada o fallo, comienza de la misma manera. Los que determinan el resultado de la acción son los fotogramas finales del vídeo donde se puede ver el balón dentro de la portería (gol), el balón rebotado por el portero (parada) o el balón fuera de los límites de la portería. Esto último puede dificultar que los clasificadores determinen correctamente el resultado del penalti.

Como trabajo futuro se propone probar nuevas configuraciones en cuanto a los modelos ya implementados, como el uso de dimensiones distintas a las empleadas, utilizar un mayor número de fotogramas por vídeo, etc. Además, se podrían probar otros modelos de clasificación de vídeos con el objetivo de conseguir mejores resultados que los obtenidos con I3D CNN y (2+1)D CNN. En el caso que con nuevas configuraciones y modelo no mejorara el rendimiento, sería interesante la implementación de clasificadores de imágenes capaces de detectar que un lanzamiento resulte en gol mediante determinar que el balón se encuentre dentro de la portería.

A pesar de los resultados obtenidos, se espera que el trabajo realizado pueda sentar las bases de futuros trabajos e investigaciones relacionadas con el deporte del balonmano. Asimismo, el conjunto de vídeos recopilados en esta propuesta así como sus anotaciones pueden ser utilidad en futuros estudios y análisis en esta área.

Bibliografía

[1] Carreira, J. and Zisserman, A. (2017). Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733.

[2] Chauhan, N. S. (2022). Naïve bayes algorithm: Everything you need to know. https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html.

[Clipchamp] Clipchamp. Herramienta de edición de vídeos clipchamp. https://clipchamp.com/es/.

[4] datos.gob.es (2020). ¿cómo aprenden las máquinas? machine learning y sus diferentes tipos. https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos.

[5] Dia, M. (2022). Ai in sports. https://www.thats-ai.org/en-GB/units/ai-in-sports.

[Excel] Excel. Herramienta de anotación de vídeos excel. https://www.microsoft.com/es-es/microsoft-365/excel.

[7] Fernández, A. (2023). Qué son y cómo crear una red neuronal convolucional con keras. https://anderfernandez.com/blog/que-es-una-red-neuronal-convolucional-y-como-crearlaen-keras/#Creando-una-red-neuronal-convolucional.

[8] Gandhi, R. (2018). Support vector machine — introduction to machine learning algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[9] GeeksforGeeks (2023a). Decision tree. https://www.geeksforgeeks.org/decision-tree/.

[10] GeeksforGeeks (2023b). Getting started with classification. https://www.geeksforgeeks.org/getting-started-with-classification/?ref=lbp.

[11] GeeksforGeeks (2023c). An introduction to machine learning. https://www.geeksforgeeks.org/introduction-machine-learning/.

[12] GeeksforGeeks (2023d). Logistic regression in machine learning. https://www.geeksforgeeks.org/understanding-logistic-regression/.

[13] GeeksforGeeks (2023e). Support vector machine (svm) algorithm. https://www.geeksforgeeks.org/support-vector-machine-algorithm/.

[14] González, L. (2023). ¿qué es el perceptrón? perceptrón simple y multicapa. https://aprendeia.com/que-es-el-perceptron-simple-y-multicapa/.

[Google] Google. Google colaboratory. https://research.google.com/colaboratory/faq.html.

[Google Colab] Google Colab. Entorno de trabajo google colab. https://colab.research.google.com.

[17] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[18] IBM (2023a). What is the k-nearest neighbors algorithm? https://www.ibm.com/topics/knn?mhsrc=ibmsearch_a&mhq=What%20is%20the%20k-nearest%20neighbors%20algorithm%26quest%3B.

[19] IBM (2023b). ¿qué son las redes neuronales convolucionales? https://www.ibm.com/es-es/topics/convolutional-neural-networks.

[20] Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231.

[21] KeepCoding (2022). Importancia de los kernels para los svm. https://keepcoding.io/blog/importancia-de-los-kernels-para-los-svm/.

[Keras] Keras. Biblioteca keras. https://keras.io.

[23] Kondratyuk, D., Yuan, L., Li, Y., Zhang, L., Tan, M., Brown, M., and Gong, B. (2021). Movinets: Mobile video networks for efficient video recognition. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16015–16025.

[24] Kumar, A. (2023). Svm rbf kernel parameters: Python examples. https://vitalflux.com/svm-rbf-kernel-parameters-code-sample/.

[25] Marca (2020). Enciclopedia deportiva: balonmano. https://especiales.marca.com/juegos-olimpicos/balonmano.html.

[Python] Python. Lenguaje de programación python. https://www.python.org.

[27] Rodríguez, D. (2018). La regresión logística. https://www.analyticslane.com/2018/07/23/la-regresion-logistica/.

[28] Saha, S. (2018). A comprehensive guide to convolutional neural networks — the eli5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[29] Schiappa, M. (2020). Understanding the backbone of video classification: The i3d architecture. https://towardsdatascience.com/understanding-the-backbone-of-video-classification-the-i3d-architecture-d4011391692.

[scikit-learn] scikit-learn. Biblioteca scikit-learn. https://scikit-learn.org/stable/.

[31] Sreenivasa, S. (2020). Radial basis function (rbf) kernel: The go-to kernel. https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a.

[32] TensorFlow. Biblioteca tensorflow. https://www.tensorflow.org.

[33] TensorFlow. Video classification with a 3d convolutional neural network. https://www.tensorflow.org/tutorials/video/video_classification.

[34] Tensorflow (2023). Video classification with a 3d convolutional neural network. https://www.tensorflow.org/tutorials/video/video_classification.

[35] Tokio School (2022). La inteligencia artificial en el deporte. https://www.tokioschool.com/noticias/inteligencia-artificial-deporte/.

[36] Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M. (2018). A closer look at spatiotemporal convolutions for action recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6450–6459.

[37] Tresp, V. (2016). Linear classification. https://www.dbs.ifi.lmu.de/Lehre/MaschLernen/SS2016/Skript/LinearClassifiers2016.pdf.

[38] Wikipedia (2019). Clasificador lineal. https://es.wikipedia.org/wiki/Clasificador_lineal.

[39] Wikipedia (2022). Máquinas de vectores de soporte. https://es.wikipedia.org/wiki/MÃąquinas_de_vectores_de_soporte#Soft_margin:_Errores_de_entrenamiento.

[40] Wikipedia (2023a). Feature selection. https://en.wikipedia.org/wiki/Feature_selection.

[41] Wikipedia (2023b). Perceptrón. https://es.wikipedia.org/wiki/PerceptrÃşn.

[42] Wikipedia (2023c). Support vector machine. https://en.wikipedia.org/wiki/Support_vector_machine.