



Universidad de Castilla-La Mancha
Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado
Grado en Ingeniería Informática
Tecnología específica de computación

Predicción de resultados de partidos en la NBA usando aprendizaje automático.

Jaime Parada López

Julio, 2023



TRABAJO FIN DE GRADO
Grado en Ingeniería Informática
Tecnología específica de computación

**Predicción de resultados de partidos en la NBA
usando aprendizaje automático.**

Autor: Jaime Parada López
Tutor: José Antonio Gámez Martín
Tutor: José Miguel Puerta Callejón

Julio, 2023

*A mis padres,
por creer siempre en mí*

Resumen

La *National Basketball Association*, también conocida como NBA, es mundialmente reconocida por ser una de las mejores y más competitivas ligas de baloncesto, por la que han pasado algunos de los jugadores con mayor renombre de la historia como pueden ser Michael Jordan, LeBron James o Kobe Bryant entre otros muchos. Asimismo, la NBA destaca por su alta influencia cultural, ya que, atrae a jugadores y aficionados de todo el mundo, y tiene una gran presencia en medios de comunicación y en múltiples negocios.

Debido a la gran naturaleza dinámica del juego, donde en cuestión de poco tiempo las estrategias y decisiones de un equipo pueden variar, y a la gran presencia de distintas variables que pueden influir en el resultado de un equipo, predecir el ganador de un partido de la NBA puede llegar a ser un gran desafío. Por ello, para poder llevar a cabo una predicción precisa del ganador de un partido de la NBA, tenemos que obtener numerosos datos históricos de la liga y aplicar novedosas técnicas de aprendizaje automático.

Actualmente, existen varias asociaciones que ofrecen datos, análisis y predicciones sobre la NBA, como son la famosa *ESPN* o el reconocido *FiveThirtyEight*, las cuales pronostican resultados de partidos o calculan el rendimiento que los equipos tendrán durante la temporada entre otras muchas estadísticas que llegan a ofrecer.

En el presente Trabajo de Fin de Grado se va a realizar un proceso de minería de datos, mediante el cual podremos realizar predicciones de partidos de NBA empleando técnicas de aprendizaje automático. Además, se realizará una aplicación web donde podremos predecir los partidos correspondientes a una fecha determinada. Este proyecto se llevará a cabo siguiendo la metodología *Cross Industry Standard Process for Data Mining* (CRISP-DM), la cual es un marco de trabajo para guiar y estructurar el desarrollo de este TFG.

Agradecimientos

En primer lugar, tengo que agradecerle a mis padres su constante apoyo y confianza en mí. Gracias a ellos soy una mejor persona y he logrado todos los objetivos que me he propuesto, además de que me han brindado todas las oportunidades con las que he podido progresar a lo largo de mi vida.

Me gustaría continuar agradeciendo todo el trabajo y atención que me han ofrecido mis tutores de este trabajo, José Antonio Gámez Martín y José Miguel Puerta Callejón, los cuales me han guiado durante todo el camino de este trabajo y me han proporcionado ayuda siempre que la he necesitado.

Quiero continuar agradeciéndole a mis amigos que siempre hayan permanecido a mi lado y me hayan sacado una sonrisa en los malos días, sin ellos no estaría donde estoy ni sería quien soy. En especial, quiero agradecerle a Pablo Palacios su fundamental ayuda, no solo en este trabajo, sino a lo largo de los cuatro años de carrera, quien ha sido la persona que me ha acompañado día a día y con la que he podido crecer tanto como ingeniero como persona.

En última instancia, agradezco a mí mismo todo el trabajo duro y constancia que he tenido a lo largo de todos estos años. '*Strive for greatness*' ha sido el lema que me ha motivado a nunca darme por vencido y dar siempre el máximo en cada momento.

Índice general

1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Competencias	2
1.4 Estructura del trabajo	3
2 Estado Del Arte	5
2.1 Introducción	5
2.2 Funcionamiento de la NBA	5
2.3 Aprendizaje automático	9
2.3.1 Árboles de decisión	10
2.3.2 Hoeffding Trees	11
2.3.3 Multiclasificadores (ensembles)	12
2.3.4 Bagging	13
2.3.5 Random Forest	14
2.3.6 Boosting	15
2.3.7 Gradient Boosting	16
2.3.8 Histogram Gradient Boosting	18
2.3.9 Redes neuronales	20
3 Desarrollo del proyecto	23
3.1 Comprensión de los datos y creación de la base de datos	23
3.2 Análisis exploratorio de los datos	26
3.3 Preprocesamiento de los datos	35

3.4 Creación de los modelos	36
3.4.1 <i>Introducción</i>	36
3.4.2 <i>Selección de modelos estáticos</i>	38
3.4.3 <i>Selección de modelos dinámicos</i>	52
3.4.4 <i>Construcción y validación del modelo final</i>	71
3.4.5 <i>Conclusiones y futuras mejoras</i>	72
4 Desarrollo de la aplicación web	75
4.1 Introducción	75
4.2 Diseño de la aplicación web	75
4.3 <i>Flask</i>	75
4.4 Arquitectura del proyecto	76
4.5 Aplicación web	77
4.6 Conclusiones y futuras mejoras	80
5 Conclusiones del proyecto	83
5.1 Introducción	83
5.2 Conclusiones	83
5.3 Mejoras futuras	84
5.4 Competencias desarrolladas.....	84
A Anexo	87
A.1 <i>Cross Industry Standard Process for Data Mining (CRISP-DM)</i>	87
A.2 Planificación del proyecto	89
A.3 Bases de datos	91
A.4 Recursos software	99
A.5 Recursos hardware.....	101
Referencia bibliográfica	107

Índice de figuras

2.1	Divisiones de la NBA	6
2.2	Cuadro de Playoffs NBA	7
2.3	Estadísticas en un partido entre <i>Los Angeles Lakers</i> y <i>Memphis Grizzlies</i> . .	8
2.4	Estadísticas de un jugador	8
2.5	Árbol de decisión	11
2.6	Algoritmo <i>bagging</i>	14
2.7	Algoritmo <i>boosting</i>	17
2.8	Ilustración gráfica <i>boosting</i>	18
2.9	Algoritmo <i>Gradient boosting</i>	19
2.10	Algoritmo <i>Histogram Gradient boosting</i>	19
2.11	Neurona artificial	21
2.12	Red neuronal	22
3.1	Porcentaje de victorias del equipo local	27
3.2	Victorias hasta cierto encuentro por parte del equipo local	28
3.3	Gráfica con las 20 variables predictoras con mayor ganancia de información	29
3.4	Matriz de correlación de las 20 variables con mayor ganancia de información	31
3.5	Victoria de los equipos teniendo en cuenta los mejores jugadores	32
3.6	Porcentaje de victorias del equipo local	32
3.7	Valoración por partido hasta el encuentro considerado por parte de los mejores jugadores del equipo local	33
3.8	Gráfico de densidad de las variables <i>HOME_TEAM_WINS</i> y <i>W_PCT_H</i> . .	34
3.9	Gráfico de densidad de las variables <i>HOME_TEAM_WINS</i> y <i>PTS_1_E1</i> . .	35
3.10	Gráfico de entrenamiento por semanas	38
3.11	Árbol de decisión	41
3.12	Gráfico de las variables más significativas en modelo <i>bagging</i>	43
3.13	Gráfico de las variables más significativas en modelo <i>random forest</i>	45
3.14	Gráfico de las variables más significativas en modelo <i>adaptative boosting</i> . .	47
3.15	Gráfico de las variables más significativas en modelo <i>gradient boosting</i> . .	49
3.16	Árbol de decisión con técnicas de olvido	60

3.17 Árbol de decisión con técnicas de olvido prediciendo por partidos	61
4.1 Estructura del proyecto	76
4.2 Interfaz aplicación	77
4.3 Menú desplegable para elegir un modelo	78
4.4 Introducción de fecha errónea en la aplicación	79
4.5 No selección de un modelo antes de predecir	79
4.6 Predicciones realizadas en la aplicación	80
A.1 Ciclo de vida CRISP-DM	88
A.2 Diagrama de planificación	91

Índice de tablas

3.1	Resultados árboles de decisión	40
3.2	Resultados <i>bagging</i>	43
3.3	Resultados <i>random forest</i>	45
3.4	Resultados <i>adaptative boosting</i>	47
3.5	Resultados <i>gradient boosting</i>	48
3.6	Resultados <i>histogram gradient boosting</i>	50
3.7	Resultados redes neuronales	52
3.8	Resultados <i>Hoeffding trees</i>	56
3.9	Resultados árboles de decisión con técnicas de olvido	58
3.10	Resultados árboles de decisión con técnicas de olvido prediciendo por partidos	59
3.11	Resultados <i>bagging</i> con técnicas de olvido	63
3.12	Resultados <i>random forest</i> con técnicas de olvido	64
3.13	Resultados <i>adaptative boosting</i> con técnicas de olvido	65
3.14	Resultados <i>gradient boosting</i> con técnicas de olvido	67
3.15	Resultados <i>histogram gradient boosting</i> con técnicas de olvido	68
3.16	Resultados de <i>ensemble</i> dinámico	69
3.17	Resultados de modelo con combinación de clasificadores con técnicas de olvido	70
3.18	Construcción y validación del modelo final	72
A.1	Base de datos <i>bbdd_NBA.csv</i>	92
A.2	Base de datos <i>games_details.csv</i>	93
A.3	Base de datos <i>players.csv</i>	94
A.4	Base de datos <i>ranking.csv</i>	94
A.5	Base de datos <i>teams.csv</i>	95
A.6	Base de datos <i>bbdd_NBA.csv</i>	99

Índice de algoritmos

3.1 Pseudocódigo de estrategias de olvido	53
3.2 Pseudocódigo de <i>ensembles</i> dinámicos	54

Índice de listados de código

3.1 Ejemplo de código de <i>Hoeffding Trees</i> en Python	55
3.2 Ejemplo de código de los clasificadores para modelo de combinación de clasificadores con técnicas de olvido en Python	70

1. Introducción

A lo largo de la historia del baloncesto, hemos podido comprobar cómo se han ido produciendo resultados que al principio parecían improbables y, a priori, imposibles de predecir. Un claro ejemplo de competición donde se producen resultados claramente inesperados es la National Basketball League, más conocida como NBA, considerada la mejor competición de baloncesto del mundo.

Conforme pasan los años, se van analizando de diferentes maneras los jugadores y equipos con novedosas estadísticas como puede ser, el *true shooting percentage* o el *player efficiency rating*, las cuales miden la eficiencia de un jugador al lanzar el balón y el rendimiento por minuto de un jugador respectivamente.

Aun teniendo múltiples estadísticas de jugadores, equipos e, incluso, entrenadores, se siguen rompiendo récords y se siguen realizando actuaciones históricas que son, a priori, prácticamente impredecibles. Un claro ejemplo fue la remontada de los *Cleveland Cavaliers* frente a los *Golden State Warriors*, cuando en el año 2016 perdían en las finales la serie 1-3, siendo una serie al mejor de 7 partidos. En ese momento, ningún equipo había remontado un 1-3 en unas finales de la NBA, el dato era que de las 32 veces que ocurrió, siempre había ganado el equipo con ventaja, pero ese año se produjo la excepción.

Por lo tanto, que la NBA sea una liga muy impredecible puede afectar considerablemente en todos aquellos millones personas que apuestan en partidos de esta competición. En 2019, según la American Gaming Association, se estimó que 500 millones de dólares fueron apostados en el extranjero, es decir, fuera de territorio estadounidense y, además, también se estimó que el gasto medio anual per cápita en apuestas de la NBA era de 111.49 dólares [47].

1.1. Motivación

Actualmente, se utilizan distintas técnicas de aprendizaje automático para múltiples usos en el mundo del deporte, como pueden ser la prevención de lesiones [50] o el estudio del

rendimiento de los deportistas profesionales [21], entre muchas otras áreas. La motivación de este trabajo de fin de grado proviene de poder de ofrecer a este mundo tan amplio, un estudio y desarrollo detallado relacionado con la predicción de resultados deportivos, en este caso con respecto a la NBA.

1.2. Objetivos

El principal objetivo de este trabajo de fin de grado es la predicción del equipo ganador en partidos de la NBA a partir de una base de datos, respecto a los equipos y los jugadores desde la temporada 2003-04 hasta mitad de la temporada 2022-23. Además, hay que destacar que se realizará una aplicación para tener una interfaz donde constatar resultados de diferentes partidos a partir del programa realizado. Por lo tanto, consideraremos los siguientes objetivos:

- Realizar un estudio y análisis de los datos proporcionados por las correspondientes bases de datos.
- Extracción y almacenamiento de los datos más significativos, los cuales combinaremos en una única base de datos.
- Identificar los algoritmos de predicción, las estrategias de validación y selección del modelo óptimo.
- Incorporación del modelo de aprendizaje automático en una aplicación para predecir el ganador de un partido correspondiente.

1.3. Competencias

Se trabajarán principalmente (en distinto grado) las siguientes competencias específicas de la tecnología de computación:

- Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.
- Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos de computación, percepción y actuación en ambientes o entornos inteligentes.
- Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

1.4. Estructura del trabajo

El presente trabajo se dividirá en cinco capítulos:

1. **Introducción.** En este capítulo inicial se presentará el trabajo actual y comentarán los objetivos, competencias y estructura del proyecto (Capítulo 1).
2. **Estado del arte.** En este capítulo se explicará el funcionamiento de la NBA y las soluciones actuales para la predicción de este tipo de problemas (Capítulo 2).
3. **Desarrollo del proyecto.** Esta tercera parte del proyecto se enfoca en el ciclo de vida completo de minería de datos utilizando una metodología CRISP-DM (Anexo A.1). Esto implicará realizar lo siguiente:
 - 3.1 Preparación de los datos (Capítulo 3.1).
 - 3.2 Análisis exploratorio de los datos (Capítulo 3.2).
 - 3.3 Preprocesamiento de datos (Capítulo 3.3).
 - 3.4 Desarrollo de distintos modelos para predecir (Capítulo 3.4).
4. **Desarrollo de la aplicación.** A lo largo de este cuarto capítulo se detallará el proceso de desarrollo de una aplicación web y las respectivas tecnologías utilizadas (Capítulo 4).
5. **Conclusiones.** Este es el capítulo final donde se tratarán las conclusiones y posibles perfeccionamientos futuros sobre el proyecto (Capítulo 5).

2. Estado Del Arte

2.1. Introducción

A lo largo de este capítulo se va a describir tanto el propio funcionamiento de la NBA como el de las técnicas actuales para poder realizar predicciones con aprendizaje automático en problemas parecidos al actual.

2.2. Funcionamiento de la NBA

La *National Basketball League*, más conocida como NBA, es una liga de baloncesto profesional que se juega, por lo común, en Estados Unidos y que fue inaugurada el 6 de junio de 1946.

La NBA está formada por 30 equipos, los cuales 15 pertenecen a la conferencia este y los otros 15 a la conferencia oeste. Además, cada conferencia se divide a su vez en 3 divisiones, la conferencia oeste se divide en noroeste, suroeste y pacífico, y la conferencia este se divide en atlántico, central y sureste [34].

Por lo tanto, la distribución de los equipos quedaría de la siguiente manera:

- Conferencia oeste:

- Noroeste: Denver Nuggets, Minnesota Timberwolves, Oklahoma City Thunder, Portland Trail Blazers, Utah Jazz.
- Pacífica: Phoenix Suns, Los Ángeles Clippers, Sacramento Kings, Los Ángeles Lakers, Golden State Warriors.
- Suroeste: Houston Rockets, San Antonio Spurs, New Orleans Pelicans, Memphis Grizzlies, Dallas Mavericks.

- Conferencia este:
 - Atlántica: Toronto Raptors, New York Knicks, Brooklyn Nets, Philadelphia 76ers, Boston Celtics.
 - Central: Cleveland Cavaliers, Indiana Pacers, Detroit Pistons, Milwaukee Bucks, Chicago Bulls.
 - Sureste: Miami Heat, Orlando Magic, Washington Wizards, Charlotte Hornets, Atlanta Hawks.

En la Figura 2.1 podemos observar la distribución de los equipos de la NBA citados previamente.

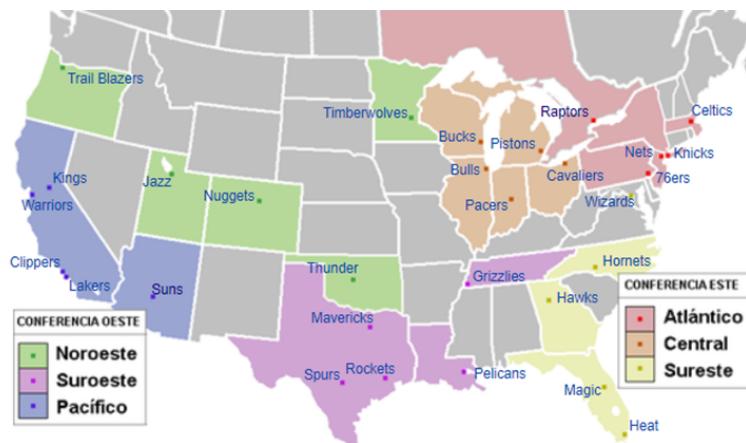


Figura 2.1: Divisiones de la NBA[38].

La temporada regular comienza en el mes de octubre y termina en abril, durante la cual, cada equipo juega 82 partidos. Y durante el mes de febrero se produce un parón donde se celebra en un finde semana el famoso *All-Star*, donde participan los mejores jugadores de la liga para realizar diferentes concursos y el partido final de las estrellas.

Tras ese periodo de temporada regular se realiza el torneo llamado *play-in*, implementado en estos últimos años, el cual determinará quién termina como séptimo y octavo, este torneo se producirá en cada conferencia entre los equipos que hayan quedado entre el séptimo y el décimo. Se realiza un partido entre el noveno y décimo, quien gane pasa al siguiente partido y luego se realiza un partido entre el octavo y séptimo, quien gane se clasifica como séptimo y la octava plaza se disputa entre el ganador del primer partido entre el noveno y décimo y del perdedor del segundo entre el octavo y séptimo.

Una vez tenemos todos los equipos en su correspondiente posición, empiezan los famosos playoffs, de los cuales saldrá el campeón de la temporada. Se enfrentará el primero contra el octavo, el segundo contra el séptimo, el tercero contra el sexto y el cuarto contra el quinto,

de cada conferencia. Finalmente, lucharán por el título el campeón de cada conferencia [39]. La Figura 2.2 muestra claramente como es la estructura de los playoffs de la NBA.



Figura 2.2: Cuadro de playoffs NBA [39].

Los partidos duran 48 minutos, es decir, 12 minutos cada cuarto, y que cada posesión dura 24 segundos, en los cuales cada equipo tiene 12 jugadores activos en el correspondiente encuentro de los 15 máximos que contiene cada equipo. El equipo ganador será el equipo que más puntos tenga durante el periodo completo del partido, si empatan a puntos se añadirán 5 minutos extra y, por lo tanto, jugarán una prórroga.

En cada partido se lleva la cuenta de los puntos, rebotes, asistencias, robos, tapones, tiros de campo acertados, tiros de campo fallados, porcentaje de tiros de campo, triples acertados, triples fallados, porcentaje de triples, tiros libres acertados, tiros libres fallados, porcentaje de tiros libres, pérdidas y faltas de cada equipo e individual entre otras muchas estadísticas.

Con respecto a las estadísticas de los jugadores, también se recoge el número de minutos jugados, pero es importante hablar de la estadística del +/-, la cual estima la diferencia para cada jugador entre los puntos aportados por el equipo estando él en pista y los puntos encajados por el equipo estando él en pista. Aunque esta estadística puede engañar, ya que, un jugador de rol en una muy buena racha en un determinado momento del partido por parte de sus compañeros, estaría indicando que tiene un amplio número de +/- cuando puede ser que no esté aportando mucho a su equipo. Por ello, se calcula una valoración que permite medir de mejor manera el rendimiento de un jugador en un partido, la cual es la suma de los aspectos positivos del jugador, como los puntos, rebotes, asistencias, robos, tapones y faltas recibidas y se le restan las acciones negativas como los tiros de campo fallados, los tiros libres fallados, pérdidas y faltas personales.

Como podemos ver, en la Figura 2.3 tenemos un ejemplo de qué datos se recogen a nivel de equipo en un partido y en la Figura 2.4 un ejemplo de los datos que se recogen sobre un jugador.

ESTADÍSTICAS DE EQUIPO		
42/89	Tiros de campo	49/92
47.2	Tiros de campo %	53.3
13/36	Triples	16/37
36.1	Triples %	43.2
15/16	Tiros libres	14/16
93.8	Tiros libres %	87.5
34	Total de rebotes	45
6	Rebotes ofensivos	10
28	Rebotes defensivos	35
24	Asistencias	25
2	Bloqueos	11
6	Bolas robadas	7
12	Bolas perdidas	14
56	Puntos en la pintura	56
19	Faltas personales	18

Figura 2.3: Estadísticas en un partido entre *Los Angeles Lakers* y *Memphis Grizzlies* [33].

NAME	DATE	TM	OPP	MIN	PTS	REB	AST	STL	BLK	TOV	FGM	FGA
 LeBron James	4/16/2023	 LAL	@  MEM	34	21	11	5	2	3	5	8	16
				FG%	3PM	3PA	3P%	FTM	FTA	FT%	PF	+/
				50.0	3	8	37.5	2	4	50.0	1	+7

Figura 2.4: Estadísticas de un jugador [46].

Cada jugador puede cometer un máximo de 5 faltas, a la sexta se le expulsará. Además, existen faltas de un carácter más grave que las normales, como puede ser la falta técnica que otorga al rival un tiro libre o la falta flagrante que le otorga al rival dos tiros libres y posesión.

2.3. Aprendizaje automático

El aprendizaje automático es un campo de la inteligencia artificial que se centra en desarrollar sistemas que aprenden, o mejoran el rendimiento, en función de los datos que consumen. [35].

El aprendizaje automático se puede dividir en varios tipos: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado y aprendizaje por refuerzo [20]. En este TFG nos centramos en el aprendizaje automático supervisado, que es el que tendremos que tener en cuenta a la hora de realizar predicciones, en particular, nos centramos en el caso de que la variable objetivo a predecir es discreta, también conocido como, clasificación supervisada.

El aprendizaje automático supervisado se puede separar en dos tipos de problemas: [27]

- Clasificación. Utiliza un algoritmo para asignar con precisión datos de prueba a categorías específicas.
- Regresión. Se utiliza para comprender la relación entre variables dependientes e independientes.

A continuación se van a explicar algunos de los términos de alta significancia que se van a citar a lo largo de este proyecto, ya que, es muy importante conocer el significado del error debido a algunos términos como el sesgo, el sobreajuste o la varianza.

- **Error debido a la varianza.** Se produce cuando el modelo aprende patrones espurios debido al ruido de los datos. Por ello, la existencia de ruido en el conjunto de datos de entrenamiento afecta en gran medida, ya que los ejemplos nuevos que se encuentren cerca de ejemplos ruidosos se predirán de forma incorrecta. Este error suele reducirse aumentando el número de ejemplos del conjunto de datos de entrenamiento o usando modelos más simples [48].
- **Error debido al sesgo.** Se debe a las suposiciones incorrectas del modelo, estas suposiciones erróneas suele reducirse usando modelos más complejos o una mayor dimensionalidad de los datos [48].
- **Error debido al sobreajuste.** Se produce cuando un modelo proporciona predicciones precisas para los datos de entrenamiento, pero no para los datos nuevos. Un modelo sobreajustado puede proporcionar predicciones inexactas y no puede funcionar bien para todos los tipos de datos nuevos, esto ocurre cuando un modelo es demasiado complejo, el conjunto de datos es demasiado pequeño y contiene datos ruidosos [18].

A continuación, vamos a describir diferentes algoritmos, con los cuales vamos a realizar los modelos correspondientes posteriormente y así, realizar las respectivas predicciones.

2.3.1. Árboles de decisión

Los árboles de decisión son un método de aprendizaje supervisado que se utiliza para la clasificación y la regresión [42], donde el objetivo de los árboles es predecir el valor, mientras que, el objetivo del algoritmo de aprendizaje es obtener el árbol a partir de los datos. Los árboles de decisión se pueden dividir en árboles de clasificación y árboles de regresión.

Una de las principales ventajas a tratar es la interpretabilidad de los modelos que se obtengan, ya que, la estructura de los árboles de decisión es de fácil comprensión. Mediante la observación de un árbol de decisión podemos identificar las reglas que se producen desde el nodo raíz hasta el nodo hoja, podemos distinguir cuáles son las variables que cobran más importancia en el modelo y, además, se pueden contemplar gráficamente mediante el uso de algunos software de visualización de gráficos, como el que usaremos durante el desarrollo de nuestro código, *Graphviz* [4]. Asimismo, destacan por su bajo coste en el proceso de predicción debido a su rapidez en tiempo de ejecución y su capacidad para manejar, de manera implícita, tanto variables categóricas como numéricas [48].

Sin embargo, existen algunos inconvenientes que se tienen que tratar, como puede ser su tendencia a la creación de modelos demasiados complejos que no suelen generalizar ante conjuntos de datos no visualizados por el algoritmo de aprendizaje, por lo que, en muchas ocasiones tienden a sobreajustar. De igual manera, existe la desventaja de que los modelos son inestables frente a modificaciones en el conjunto de datos de entrenamiento [48].

En los árboles de decisión se tiene una estructura donde los nodos internos representan las características del conjunto y las ramas representan las reglas de decisión y cada nodo hoja representa el resultado [30].

Los árboles tienen dos tipos de nodos:

- **Nodo de decisión.** Son nodos que se utilizan para decidir por qué rama se procesará el registro a predecir y tienen múltiples ramas.
- **Nodo hoja.** Son nodos finales y contienen el valor de la variable objetivo que se devolverá para los registros que lo alcancen.

En la Figura 2.5 podemos ver un claro ejemplo de árbol de decisión, el cual tiene 4 nodos de decisión, donde uno de ellos es raíz, que es desde donde comienza el árbol, y 5 nodos hoja. Cada nodo hoja equivale a una regla si leemos desde el árbol desde la raíz a él.

Para poder formar un árbol de decisión hay que saber cómo se elige el mejor atributo en cada nodo. Normalmente, se recurre a medidas estadísticas o de teoría de la información, como por ejemplo la entropía, la ganancia de información o el índice Gini [28].

Además, hay que destacar que la implementación de los árboles de decisión se ha realizado mediante la utilización de la librería *scikit-learn* y del módulo *sklearn.tree*, el cual incluye un clasificador de árboles de decisión, *DecisionTreeClassifier* [9].

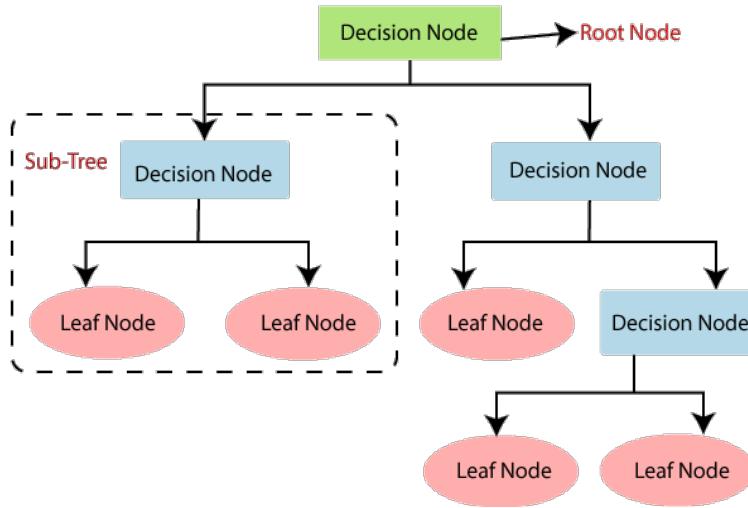


Figura 2.5: Árbol de decisión [30].

2.3.2. *Hoeffding Trees*

Un *Hoeffding Tree* es un árbol de decisión incremental que es capaz de aprender de los flujos de datos, el cual explota la idea de que una pequeña muestra puede ser suficiente para elegir un atributo de división óptimo. Esta idea está respaldada matemáticamente por el límite aditivo de Chernoff [16]. Esto permite que los árboles se construyan y actualicen de manera eficiente a medida que llegan nuevos datos.

Por lo tanto, la idea principal de los *hoeffding Trees* es que retrasan la decisión de particionar un nodo hasta que el número de registros que ha llegado a ese nodo supera un determinado umbral, mediante dichos criterios como pueden ser límite de hoeffding o límite de Chernoff. Además, se demuestra que el árbol resultante es similar al que se obtendría si se hubieran tenido todos los datos desde el principio.

Este tipo de árboles tienen algunas ventajas como puede ser su aprendizaje adaptativo, el cual se produce a medida que van llegando los nuevos datos o su alta efectividad a la hora de manejar un volumen muy grande de datos, ya que, no hace falta procesar todos los datos debido a que se va actualizando de manera incremental.

Sin embargo, los *hoeffding trees* también tienen algunas desventajas, como por ejemplo, el hecho de que pueden ser más sensibles al ruido o que pueden conllevar a una mayor dificultad el manejo de atributos con una gran cantidad de valores únicos.

Posteriormente, en la realización de modelos con los *hoeffding Trees* se podrá observar cómo se realiza un método utilizado para predecir las etiquetas, donde se utilizarán modelos tipo *Naive Bayes*. Por lo que, a continuación se va a detallar en qué consisten estos métodos.

Los modelos Naive Bayes ofrecen una forma fácil de construir ciertos modelos con un comportamiento bueno, ya que se basan en que las variables predictoras no dependen unas de otras. Las técnicas Naive Bayes necesitan una forma de calcular la probabilidad posterior de que prevenga un cierto evento, dadas algunas probabilidades de eventos anteriores. La fórmula básica que se utiliza en el modelo Naive Bayes es que se detalla a continuación [6].

$$P(\text{Clase} | \text{Características}) = (P(\text{Clase}) * P(\text{Características} | \text{Clase})) / P(\text{Características})$$

Donde:

- **P(Clase | Características)**. Probabilidad de que la instancia pertenezca a una clase dada las características observadas.
- **P(Clase)**. Probabilidad previa de la clase.
- **P(Características | Clase)**. Probabilidad condicional de observar las características dada la clase.
- **P(Características)**. Probabilidad de observar las características.

Naive Bayes tiene algunas ventajas como su rapidez y sencillez a la hora de predecir, pero también tiene alguna desventaja como que asume que todas las características son independientes entre sí, de modo que nunca podrá aprender la relación existente entre ellas.

Para concluir, es importante comentar que se ha utilizado el módulo *skmultiflow.trees*, el cual contiene el clasificador utilizado *HoeffdingTreeClassifier* [15].

2.3.3. Multiclasificadores (*ensembles*)

Los *ensembles* son métodos que aprenden de un conjunto de clasificadores del conjunto de datos y mediante los cuales se fusionan varios modelos diferentes y producir un único resultado.

En estos modelos se realiza un proceso para clasificar nuevas instancias en tres pasos:

1. Aprender n modelos a partir de los datos, los cuales constituyen el *ensemble*.
2. La instancia es procesada por cada modelo del *ensemble*.
3. Los resultados obtenidos son combinados para obtener un único clasificador.

Es importante considerar que haya independencia entre los clasificadores y que, además, los modelos base se complementen entre sí y así se pueda obtener un resultado mejor [36]. Cabe destacar, que con los *ensembles*, generalmente, se conseguirá mejorar el resultado que se pueda obtener con un único clasificador. Hay varios tipos de algoritmos utilizados como pueden ser *Bagging*, *Boosting*, *Random forest* o *Gradient Boosting*. Los cuales vamos a ir viendo su funcionamiento y su relación con la predicción con aprendizaje automático.

2.3.4. *Bagging*

Bagging es un algoritmo cuya estrategia es utilizar una función de aprendizaje y obtener modelos bien diversos entre sí para reducir el error obtenido mediante varianza. Básicamente, son clasificadores que individualmente tengan poder de generalización y predictivo, pero que al mismo tiempo estén lo menos correlacionado entre sí. Por ello, se utilizan diversas técnicas de muestreo y aleatorización de los modelos [48].

Dado un conjunto de datos D con n instancias, se repite t veces:

- Obtener una muestra D_t de tamaño n mediante muestreo aleatorio con reemplazo de D .
- Aprender un clasificador M_t de D_t .

Seguidamente, se clasifica un nuevo caso X con $\{M_1, \dots, M_t\}$. Por lo tanto, sea y_t la salida obtenida de M_t , la salida del *ensemble* M^* aplica votación por mayoría, por lo que, $M^*(X) = mode\{y_1, \dots, y_t\}$. Para un caso con predicción numérica, donde hubiera que realizar un modelo con regresión sería $M^*(X) = mean\{y_1, \dots, y_t\}$.

Una de las principales ventajas de este algoritmo es que suele mejorar la precisión, ya que combina las predicciones de varios modelos entonces, se reduce el error de predicción. Además, esta técnica funciona porque reduce, tal y como se ha citado anteriormente, la varianza y, además, el sobreajuste. Asimismo, es un algoritmo útil para conjuntos de datos ruidosos y, donde también, se reduce el sesgo [23].

Cabe destacar, que lo ideal es encontrar un equilibrio entre el sesgo y la varianza para obtener un modelo que generalice bien a nuevos datos.

Una de las principales desventajas es que, *bagging*, tiene un alto coste computacional en comparación a otros métodos, ya que, requiere entrenar múltiples modelos en conjuntos de datos *bootstrap*. Además, este tipo de técnicas son menos interpretables que, por ejemplo, los árboles de decisión.

La implementación de los correspondientes modelos utilizando *bagging* se ha llevado a cabo mediante la utilización del módulo *sklearn.ensemble*, el cual incluye el clasificador *BaggingClassifier* [11].

En la Figura 2.6 podemos observar cómo sería un modelo con un *ensemble bagging*.

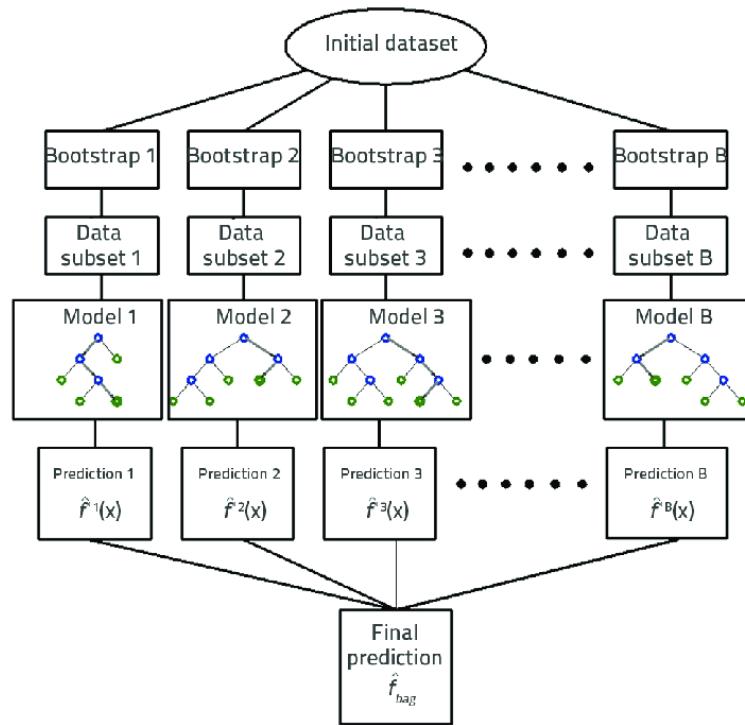


Figura 2.6: Algoritmo *bagging* [31].

2.3.5. Random Forest

Random forest es una técnica de aprendizaje automático supervisado, en el cual se realiza una clasificación aleatoria de bosques, donde se crean múltiples árboles de decisión utilizando diferentes subconjuntos aleatorios de datos y características. Las predicciones se obtienen del cálculo de la predicción de cada árbol y realizando votación por mayoría [43].

El *ensemble random forest*, busca reducir el error obtenido mediante varianza, pero adicionalmente integra otras técnicas de aleatorización en el aprendizaje de los árboles de decisión para ampliar la generalización del ensemble. Concretamente, utiliza una muestra aleatoria de los atributos a la hora de seleccionar cada punto óptimo de corte para reducir la varianza de cada uno de los estimadores del ensemble, a costa de incrementar ligeramente el sesgo [48].

También, es importante destacar, que debido la aleatoriedad en la construcción de cada árbol, se reduce la probabilidad de que el modelo sobreajuste y que, es un método muy robusto contra el ruido, donde no se aplica poda, aunque se puede considerar la profundidad de los árboles.

La técnica mediante la cual el árbol aprende de forma aleatorizada es la llamada agregación de *bootstrap*, donde selecciona aleatoriamente muestras con reemplazo, lo que significa que una misma muestra puede aparecer múltiples veces en una muestra de entrenamiento [29].

Una vez seleccionado nuestra muestra de entrenamiento aleatoria, el árbol se irá construyendo realizando particiones en función de una elección aleatoria de una característica.

Seguidamente, se realiza el anterior proceso repetidas veces hasta obtener el número de árboles de decisión que queramos y, posteriormente, se realiza la predicción mediante votación por mayoría, la cual significa que se aplica la mayoría de votos de los árboles individuales.

La realización de los modelos correspondientes a *random forest* se ha llevado a cabo mediante la utilización del módulo *sklearn.ensemble*, el cual incluye el clasificador *RandomForestClassifier* [14].

Para finalizar, hay que enfatizar en el hecho de que este método produce árboles más largos que algunos algoritmos como C4.5, etc. Y que además, el tiempo para hacer crecer el árbol es reducido, ya que, solo un subconjunto de variables es evaluada en cada nodo interno.

2.3.6. *Boosting*

Boosting es un *ensemble*, el cual trata de reducir el sesgo guiando la función de aprendizaje para que incida en aquellos ejemplos que sean más difíciles de generalizar. Por lo que, se trata de establecer un problema de optimización de funciones de coste. Por ello, los clasificadores ideales que se pueden utilizar en este tipo de técnicas son clasificadores con un poder de generalización y parámetros no muy complejos ni sobreajustados (*weak learners*). Esto otorga al algoritmo espacio para optimizar dichos parámetros y teóricamente transformar el clasificador base en un *strong learner* [48].

Una de las principales ventajas de *boosting* es la capacidad de mejorar en gran medida la precisión de los modelos de clasificación débiles al combinar múltiples clasificadores, debido a que asocia un mayor peso a los ejemplos de entrenamiento que son difíciles de clasificar. Sin embargo, en este tipo de técnicas, a veces se tiende sobreajustar instancias mal clasificadas [48].

En el método de *boosting* se asocian unos pesos a las instancias del conjunto de datos D y el aprendizaje se organiza en una secuencia de modelos, donde el clasificador C_t aprende teniendo en cuenta los errores de los anteriores clasificadores C_1, \dots, C_{t-a} , como por ejemplo, árboles de decisión con una profundidad limitada.

Cada modelo M_T aprende con los pesos asociados a las instancias. Después de que este modelo aprenda, su importancia se calcula en función del error, y los pesos son modificados, incrementando para aquellas instancias mal clasificadas y disminuyendo para aquellas instancias que clasifican mejor. Por lo tanto, el modelo M_{t+1} aprende usando los pesos actualizados y, por consiguiente, presta más atención a las instancias mal clasificadas con anterioridad.

Finalmente, el *ensemble* M^* combina los votos de cada clasificador, pero dándole unos pesos en función de la importancia de cada uno.

Cabe destacar, que *boosting* suele obtener mejores resultados que *bagging*, pero que también suele sobreajustar para aquellas instancias mal clasificadas.

Al igual que en los anteriores *ensembles*, se utiliza el módulo `sklearn.ensemble`, el cual contiene el clasificador que se está tratando en esta sección, *AdaBoostClassifier* [10].

En la Figura 2.7 podemos ver cómo sería el funcionamiento del algoritmo basado en *boosting* y en la Figura 2.8 observamos cómo funciona realmente *boosting* y cómo le da importancia a las instancias que clasifican peor.

2.3.7. Gradient Boosting

Gradient Boosting es una técnica de aprendizaje automático por el cual se entrena muchos modelos de manera gradual, aditiva y secuencial. La principal diferencia entre *boosting* y *Gradient boosting* es la manera en el que los algoritmos identifican las deficiencias de los modelos débiles. *Gradient boosting* identifica las deficiencias mediante el uso de gradientes en la función de pérdida [44].

En *gradient boosting* tanto para problemas de clasificación como de regresión, el ensemble está formado por árboles de regresión, que en cada paso intentan predecir el error cometido, de forma que se va modulando una clasificación inicial muy simple como por ejemplo, la media o la moda. Además, en cada iteración todos los modelos anteriores aprendidos son usados y todos los modelos aprendidos tienen la misma importancia.

Gradient boosting genera modelos precisos que tienen buena escalabilidad, por lo tanto, pueden aplicarse a conjuntos de datos con un elevado número de observaciones [41]. Sin embargo, es una técnica donde se puede sobreajustar en algunas ocasiones, sobre todo si no configuramos de la manera correcta los hiperparámetros correspondientes.

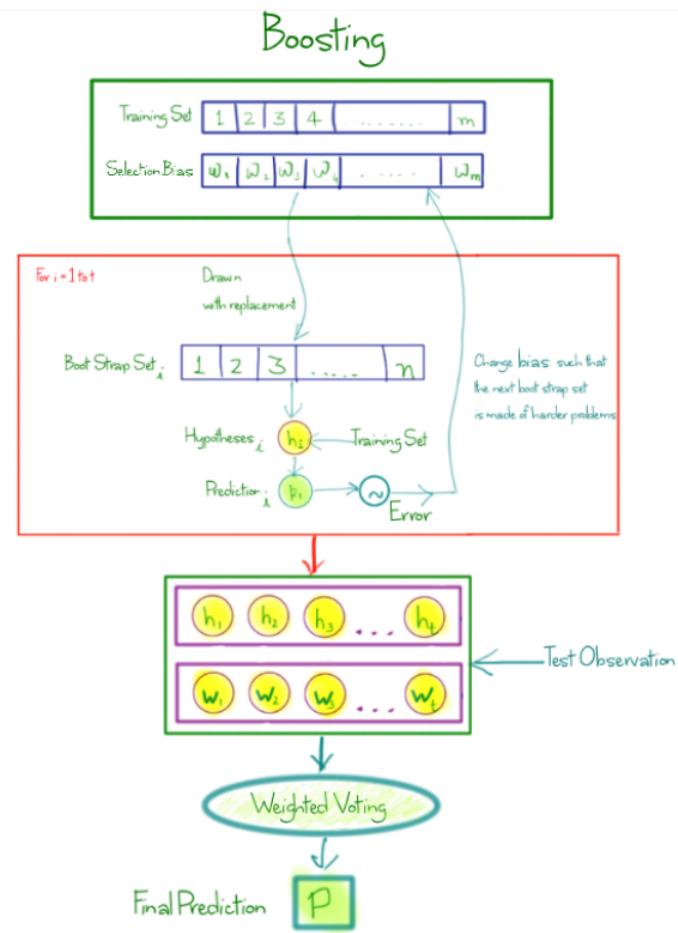


Figura 2.7: Algoritmo *boosting* [23].

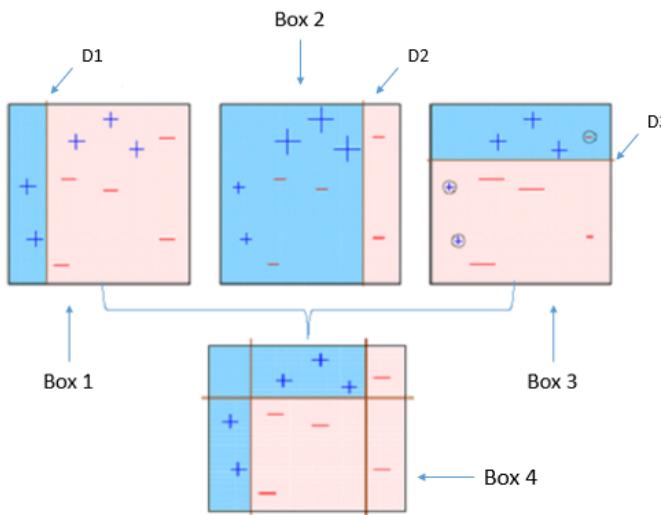


Figura 2.8: Ilustración gráfica *boosting* [40].

Al igual que en los anteriores *ensembles* se utiliza el módulo `sklearn.ensemble`, pero en este caso el clasificador con el que se implementarán los modelos será *GradientBoostingClassifier* [12].

En la Figura 2.9 podemos observar el funcionamiento del algoritmo de *gradient boosting*. Cabe destacar que, al contrario que *boosting* en el que pesan los ejemplos a partir de los errores, aquí lo que se hace es intentar predecir los residuos que minimizan el error actual, que viene dado por el *ensemble* construido hasta el momento, no solo por el último árbol considerado.

2.3.8. Histogram Gradient Boosting

Histogram gradient boosting es una optimización del anterior algoritmo, *gradient boosting*, y en donde se discretiza el conjunto de datos de entrada para reducir el número de puntos de corte a considerar en la construcción de los árboles de decisión. De esta manera, no tiene que considerar cada valor distinto de las variables predictoras continuas como punto de corte al construir los árboles de decisión, lo que le permite reducir en varios órdenes de magnitud el tiempo de entrenamiento e inferencia [48]. Además, merece la pena destacar que, es capaz de tratar valores perdidos de manera implícita sin necesidad de realizar una imputación previa [48].

El módulo, `sklearn.ensemble`, utilizado en anteriores algoritmos, será el que utilizaremos para la implementación de los modelos de *Histogram gradient boosting*, el cual contiene *HistGradientBoostingClassifier* [13].

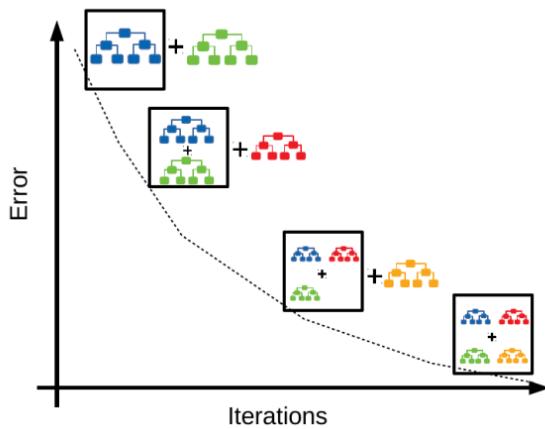


Figura 2.9: Algoritmo *Gradient boosting* [23]

En la Figura 2.10, podemos observar cómo se construye el correspondiente histograma en este tipo de algoritmos.

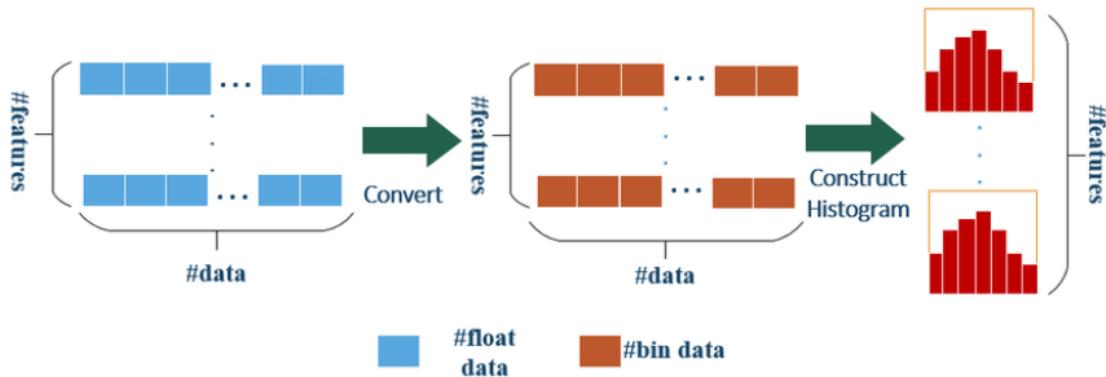


Figura 2.10: Algoritmo *Histogram Gradient boosting* [5]

2.3.9. Redes neuronales

Las redes neuronales son una técnica de inteligencia artificial mediante la cual las computadoras aprenden a procesar los datos de una manera que está inspirada en la forma en que lo hace el cerebro humano [19].

Las redes neuronales están compuestas por las neuronas artificiales, las cuales están interconectadas entre sí formando capas y transmitiendo señales a través de conexiones ponderadas. Estas neuronas reciben uno o varios valores de entradas y genera un valor de salida utilizando un proceso de dos pasos, los cuales son los siguientes [24].

1. Calcula una suma ponderada de los valores de entrada.
2. Procesa el valor de resultante usando una función de activación.

Una de las principales ventajas de las redes neuronales que son capaces de aprender patrones complejos y tienen una alta adaptabilidad y flexibilidad, ya que pueden extraer características importantes del conjunto de datos y se pueden adaptar a distintos tipos de problemas, como de clasificación o de regresión, entre muchos otros.

Sin embargo, a pesar de que tienen un gran número de ventajas, también pueden presentar algunas desventajas, como es el gran tiempo de entrenamiento que puede conllevar realizar un modelo con redes neuronales o, también en algunos casos, donde si no se controla de manera adecuada, se puede llegar a sobreajustar.

En la Figura 2.11 se puede observar un ejemplo de lo que sería una neuronal artificial.

Una red neuronal está formada por varias capas de neuronas artificiales que están divididas en una capa de entrada, cero o más capas ocultas y una capa de salida. En la Figura 2.12 se puede visualizar una representación de una red neuronal que tiene una capa de entrada, una capa oculta y una capa de salida [24].

Las redes neuronales son utilizadas en problemas de clasificación de aprendizaje automático debido a su capacidad para aprender patrones complejos y extraer características relevantes de los datos.

Además, hay varios tipos de redes neuronales, algunas de las más importantes son las que se citarán a continuación [19].

- **Redes neuronales prealimentadas (FNN).** Procesan los datos en una dirección, desde el nodo de entrada hasta el nodo de salida, y mejoran las predicciones a lo largo del tiempo.
- **Redes neuronales convolucionales (CNN).** Realizan funciones matemáticas específicas, sobre todo, para clasificar imágenes porque pueden extraer sus características más relevantes.

- **Redes Neuronales Recurrentes (RNN).** Trabajan con datos secuenciales y capturan sus dependencias temporales en mediante conexiones recurrentes.

Se ha empleado la librería *TensorFlow* para la construcción de las redes neuronales en los correspondientes modelos.

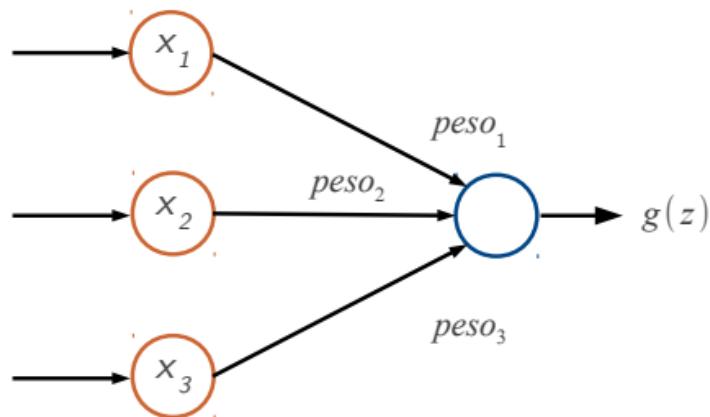


Figura 2.11: Neurona artificial [24]

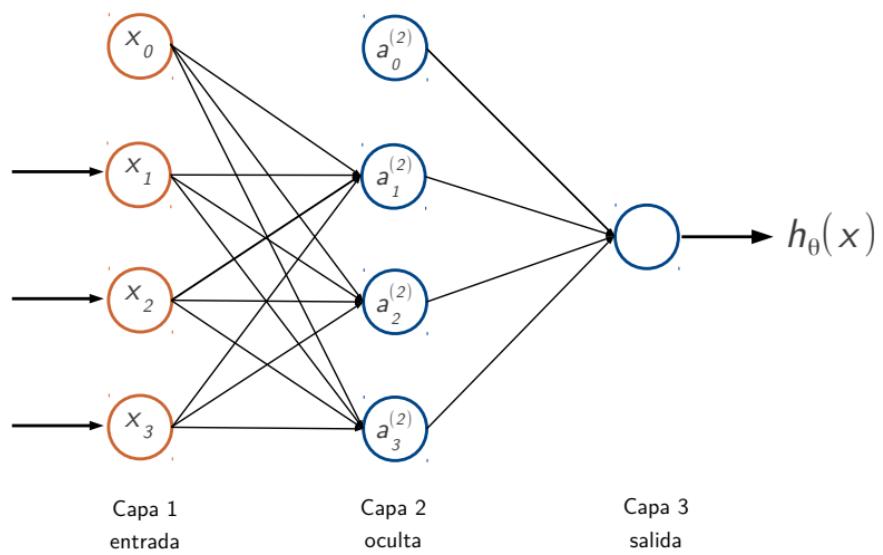


Figura 2.12: Red neuronal [24]

3. Desarrollo del proyecto

En este capítulo se va a indicar el desarrollo del proyecto siguiendo las fases de la metodología CRISP-DM, desde la comprensión de los datos y la creación de la base de datos hasta el desarrollo de los modelos para la predicción, pasando por un correspondiente análisis exploratorio y preprocesamiento de datos. Todas estas fases están citadas y explicadas de forma más detallada en el Anexo [A.1](#).

3.1. Comprensión de los datos y creación de la base de datos

En la fase de la comprensión de los datos primero se adquirirán los datos de las correspondientes bases de datos, donde las describiremos con sus correspondientes variables. Además, unificaremos estas bases de datos iniciales en una única y crearemos nuevas variables que nos servirán para el desarrollo posterior de los modelos.

Para comenzar, hay que destacar la procedencia de los datos, la cual es de la plataforma [Kaggle](#) (*NBA games data*), y de donde obtendremos diversas estadísticas tanto de equipos como de jugadores, además de los correspondientes ids y nombres de cada uno de los jugadores y equipos.

La base de datos se compone de 5 archivos, los cuales son:

- *games.csv*. En esta base de datos tenemos 21 columnas que representan las variables de los partidos entre dos equipos y tiene, 26650 filas, donde cada fila representa un partido entre dos equipos. Por lo tanto, en esta base de datos tenemos, 26650 partidos con 21 variables cada uno.
- *games_details.csv*. En esta base de datos tenemos 29 columnas y 668627 filas, las cuales indicarán las estadísticas de un jugador en un partido determinado.
- *players.csv*. En esta base de datos tenemos 4 columnas y 7228 filas, donde cada fila representará un jugador de la NBA.

-
- *ranking.csv*. En esta base de datos tenemos 13 columnas y 210342 filas, donde cada fila indica la información y el récord del equipo correspondiente en una fecha determinada.
 - *teams.csv*. En esta base de datos tenemos 14 columnas y 30 filas, las cuales representan y describen los equipos que hay en la NBA.

La información detallada de las variables en cada uno de estos archivos se muestra detalladamente en el Anexo [A.3](#).

Una vez hemos comprendido las 5 bases de datos que tenemos y el funcionamiento de cada una de ellas con sus correspondientes variables, podemos crear una única base de datos con la que trabajaremos posteriormente.

Para comenzar, tenemos de considerar que, tanto la base de datos de *games.csv* como la de *games_details.csv*, son las que más tenemos que tener en cuenta a la hora de la creación de nuestra base de datos, ya que, son las que mayor información nos proporcionan acerca de los partidos. Sin embargo, nos encontramos con un problema en la base de datos *games.csv*, ya que, cada fila indica los datos de un partido, por lo que, ya sabemos qué equipo va a ganar. Por lo tanto, se ha considerado que es más correcto que en cada fila estén los promedios de cada variable hasta la fecha de los dos equipos que se enfrenten en cierta jornada. Además, aparte de usar la base de datos *games.csv*, obtendremos de *games_details.csv* los 3 mejores jugadores con sus promedios a considerar en cada jornada de ambos equipos.

Las variables que obtendremos para cada uno de estos jugadores son las siguientes, donde $X1$ será un 1, si es el mejor del equipo, un 2, si es el segundo mejor del equipo o un 3, si es el tercer mejor, y $X2$ será 1, si es el equipo local o 2, si es el equipo visitante.

- *PLAYER_ID_X1_EX2*. Id del jugador correspondiente.
- *PLAYER_NAME_X1_EX2*. Nombre del jugador correspondiente.
- *PTS_X1_EX2*. Puntos por partido del jugador correspondiente hasta cierta fecha en esa temporada.
- *AST_X1_EX2*. Asistencias por partido del jugador correspondiente hasta cierta fecha en esa temporada.
- *REB_X1_EX2*. Rebotes por partido del jugador correspondiente hasta cierta fecha de esa temporada.
- *VAL_X1_EX2*. Valoración por partido del jugador correspondiente hasta cierta fecha de esa temporada.
- *COMMENT_JX1_EX2*. Indicará un 1 si jugó el anterior partido o un 0 si no jugó.

Las anteriores variables se detallan, junto a las demás de la correspondiente base de datos, en la Tabla [A.6](#), la cual está contenida en el Anexo [A.3](#).

Primeramente, vamos a tratar de utilizar de manera correcta la base de datos *games.csv*, por lo que, vamos a tratar de eliminar las columnas *GAME_STATUS_TEXT*, *TEAM_ID_home* y *TEAM_ID_away* porque no son relevantes. Seguidamente, vamos a eliminar las filas con valores desconocidos (codificados como *NaN*), aunque en esta base de datos es un porcentaje muy pequeño, por lo que no tendrá significancia eliminarlas. A continuación, ordenamos por fecha las filas y quitamos los partidos que son de pretemporada para solo tener los de temporada regular y playoffs.

A posteriori, en vez tener los puntos, rebotes, asistencias, porcentajes, etc. vamos a extraer los promedios de cada equipo, como local y visitante, de cada una de esas variables, por ejemplo, si nos situásemos en la jornada 20 de la temporada 2018-19 y *Los Angeles Lakers* jugasen de local, se obtendrían los promedios de cuando juegan como local *Los Angeles Lakers* hasta esa jornada. Estos promedios los podremos calcular debido a que, en la base de datos *games.csv*, tenemos los puntos, rebotes, asistencias y porcentajes que se producen en cada partido por parte de ambos equipos, por lo que, recorreremos todas las temporadas de un equipo en específico y dentro de cada una de las temporadas recorreremos todas las filas, que serán los partidos, donde podemos ir obteniendo los promedios a medida que vamos avanzando. Con respecto al primer partido, como no tenemos partidos anteriores, indicaremos los promedios de la temporada anterior. Por lo tanto, el anterior proceso lo repetiremos para todos los equipos.

Posteriormente, vamos a tratar la base de datos *games_details.csv*, la cual utilizaremos para extraer los 3 mejores jugadores de cada equipo hasta cierta fecha. Para comenzar con este proceso, vamos a crear una nueva variable llamada valoración, la cual servirá como métrica para saber qué jugadores ofrecen mejor o peor rendimiento que otros en función de sus principales estadísticas. Esta valoración será la suma de los puntos, rebotes, asistencias, robos y tapones, a la que se le restará los tiros de campo fallados, tiros libres fallados, pérdidas y faltas personales. Los 3 jugadores de cada equipo con mejor valoración de promedio son los que consideraremos mejores en cada jornada. Estas variables son informativas y relevantes a la hora de predecir partidos debido a que, la NBA es una liga donde las estrellas de los equipos son un factor determinante a la hora de ganar partidos. Además, tal y como se mostrará más adelante en la Sección 3.2, se podrá ver cómo estas variables tienen un potencial discriminador debido a que se consigue una gran ganancia de información para la clase respecto a cada una de ellas.

A continuación, haremos un join, con la clave de la id de los partidos, de la base de datos que hemos creado primero, donde se obtenían los promedios de cada equipo hasta el momento en esa temporada, y de la base de datos con los 3 mejores jugadores de cada equipo en cada jornada.

Una vez tengamos los promedios de los equipos y los mejores jugadores de cada uno de ellos, vamos a introducir datos de la base de datos de *ranking.csv*. El contenido que vamos a añadir a nuestra base de datos es el récord correspondiente a cada equipo en cada jornada, ya que, en la base de datos tenemos el ranking de los equipos en cada jornada, por lo que se

va a extraer esos datos y se van a introducir junto a los promedios de equipos y los mejores jugadores.

A continuación, vamos a extraer más contenido de la base de datos *games_details.csv*, de la cual vamos a poder saber para cada jugador si el anterior partido jugó o no, ya sea por lesión, decisiones del entrenador, etc. En la base de datos, cada jugador puede tener un comentario o tener un valor *Nan* en la columna *COMMENT*. Entonces, en nuestra base de datos solo nos quedaremos con, si jugó o no, independientemente de la razón, por lo tanto, la variable indicará un 1 si jugó el anterior partido sin problema o un 0 si no jugó.

Para finalizar con esta sección, se van a detallar algunas de las características de la base de datos definitiva con la que posteriormente se trabajará.

- Nuestra base de datos tiene 68 variables y 24910 registros.
- Se han construido 54 variables nuevas a partir de la información contenida en *games.csv* y *games_details.csv*, ya que, tal y como se ha comentado anteriormente, todos los promedios obtenidos, tanto de equipos como de jugadores, han sido extraídos de estas dos bases de datos.
- Una característica muy significativa es que la base de datos es el carácter temporal de la BBDD, ya que los partidos se juegan en una fecha concreta y para simular el proceso real, no podemos utilizar información posterior de una fecha para predecir partidos previos a dicha fecha.
- Al margen de los promedios calculados, también es importante decir que se almacenarán fechas, id del partido, id y nombre de ambos equipos, la temporada en la que se produce el partido y la variable objetivo *HOME_TEAM_WINS*, la cual es la que se está tratando de predecir. Esta variable indicará un 1 si el equipo local gana o un 0 si pierde.
- Indicaremos los promedios de la anterior temporada en el primer partido de cada equipo en cada temporada, excepto en la primera temporada, ya que no tenemos datos previos, por lo que en estas celdas indicaremos el valor de esa columna con un 0 en vez de con un *Nan* para que así, no aparezcan problemas a la hora de extraer las gráficas correspondientes en la Sección 3.2.

El contenido detallado de todas las bases de datos citadas previamente se podrá consultar en las correspondientes tablas del Anexo A.3.

3.2. Análisis exploratorio de los datos

En primer lugar, hay que describir el contenido de la base de datos que hemos creado previamente, tal y como hemos descrito en la sección anterior. La base de datos con la que vamos a trabajar tiene 68 columnas y 24910 filas. Para ver detalladamente el contenido de la

base de datos podemos observar la Tabla A.6, la cual está contenida en el Anexo A.3.

Para comenzar, vamos a observar algunas de las variables más importantes mediante gráficas univariadas. Una variable de interés es *HOME_TEAM_WINS*, donde podemos observar en la Figura 3.6, que el 59.1% de los equipos locales ganan sus respectivos partidos, mientras que el 40.9% los pierden.

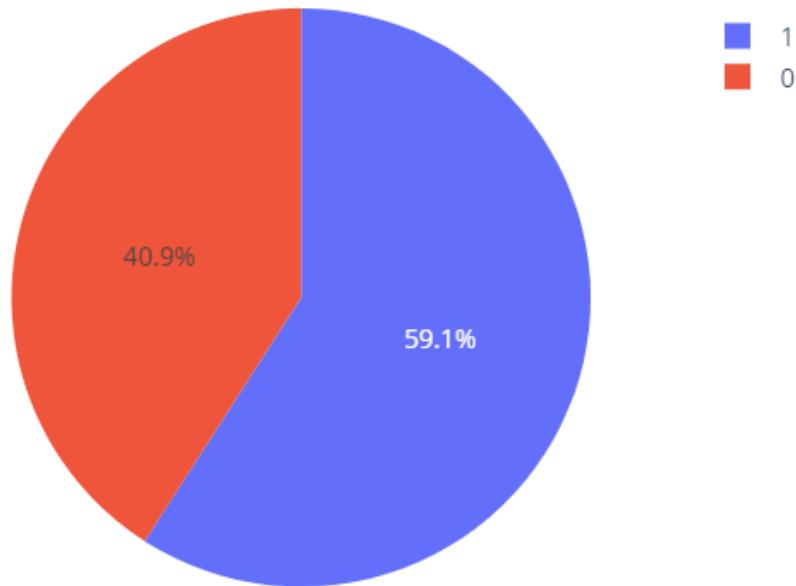


Figura 3.1: Porcentaje de victorias del equipo local.

Seguidamente, vamos a observar la variable *W_H*, la cual representa el número de victorias que lleva el equipo local hasta la jornada considerada. Tal y como podemos ver la Figura 3.2, todos empiezan en 0 victorias, luego vemos que la mitad llegan al menos a 30 victorias, pero solo una minoría llega a 60 o incluso 70. Hay que destacar, que en la NBA se juegan 82 partidos de temporada regular, donde solo 2 equipos en la historia de la NBA han llegado a tener más de 70 victorias [45]. Además, en la Figura 3.2 podemos comprobar que sigue una distribución asimétrica positiva, donde observamos que conforme el número de victorias aumenta, el número de equipos con dicho número de victorias disminuye.

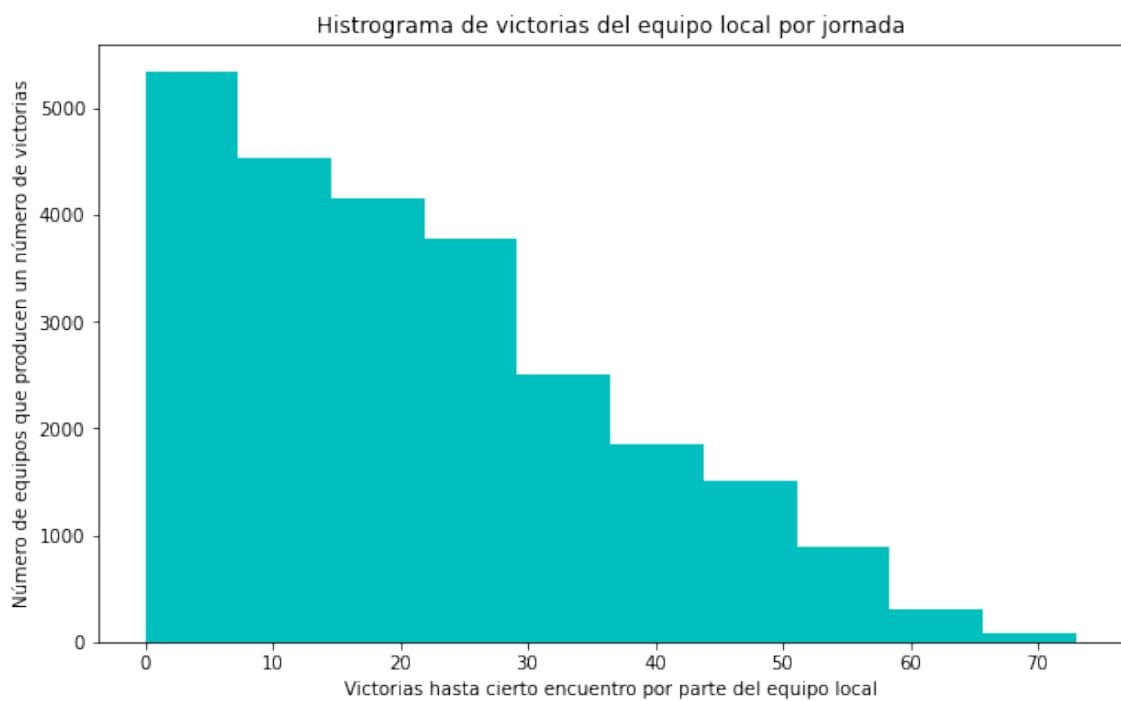


Figura 3.2: Victorias hasta cierto encuentro por parte del equipo local.

Una de las cuestiones que se consideró importante realizar en el análisis exploratorio fue calcular la ganancia de información de las variables predictoras respecto a la clase. Debido a que tenemos 67 variables predictoras, la Figura 3.3 solo mostrará las 20 con mayor ganancia de información, ya que son las que consideraremos más significantes. Por lo tanto, tal y como podemos observar, las variables que más ganancia de información tienen respecto a la clase son los puntos por partido de cada equipo, como local o visitante, el porcentaje de victorias de cada equipo, y los puntos y valoraciones por partido de los mejores jugadores de cada equipo. Hay que resaltar el hecho de que los promedios respecto a los equipos y las estadísticas de los jugadores, son promedios hasta antes del partido descrito, ya que, si no tendríamos ya el resultado.

Tal y como se puede observar en la Figura 3.3, la mayoría de las variables más significativas son variables que hemos construido previamente en la Sección 3.1, por ejemplo, las variables que recogen los promedios de puntos por partido de un equipo, como local o visitante, observamos que tienen una gran ganancia de información respecto a la que consiguen las demás. Otro aspecto a destacar, es la ganancia de información que obtienen las variables construidas que tratan las estadísticas de los jugadores, ya que, como hemos comentado anteriormente, en la NBA cobran una gran importancia las actuaciones individuales de los jugadores y son muy relevantes a la hora de hacer ganar a un equipo, por lo tanto, podemos ver como la valoración y los puntos, por partido, de los mejores jugadores de ambos equipos tienen una ganancia de información elevada.

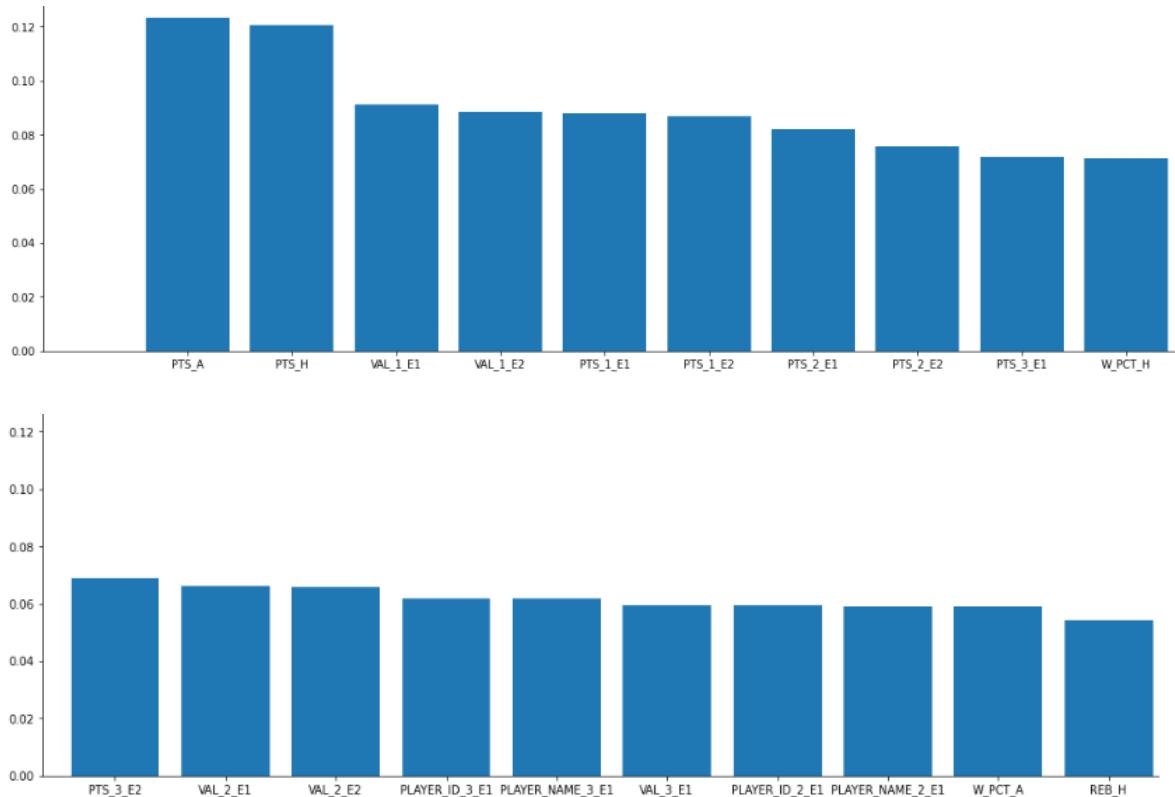


Figura 3.3: Gráfica con las 20 variables predictoras con mayor ganancia de información.

Posteriormente, se ha realizado la matriz de correlación de las 20 variables con mayor ganancia de información calculadas previamente. En esta matriz se mide el grado de relación lineal entre cada par de variables y estos valores de correlación se pueden ubicar entre -1 y +1. Si las dos variables tienden a aumentar o disminuir al mismo tiempo, el valor de correlación es positivo.

En la Figura 3.4 podemos observar la alta correlación entre algunos pares de variables, donde se puede contemplar que hay mucha correlación entre las variables que registran los promedios de puntos y valoración de los correspondientes jugadores, esto es debido a que la puntuación afecta significativamente al cálculo de la respectiva valoración.

Además, la valoración por partido de los jugadores, también tiene una alta correlación con respecto a los puntos por partido que produce su equipo, esto es debido a que, si los tres mejores jugadores obtienen una alta valoración ayudarán a su equipo a obtener más puntos,

ya no solo anotando puntos, sino que también con otro tipo de actuaciones como asistiendo, reboteando o con algún tipo de actuaciones defensivas, como robando el balón o taponando tiros del equipo rival, las cuales acaben en una buena transición ofensiva.

Asimismo, es importante fijarse en que las valoraciones por partido de los jugadores de un equipo están correlacionadas entre sí, de lo que podemos entender que, cuando un jugador juega bien y obtiene una buena valoración, generalmente, los demás también aportan y se pueden ver involucrados en jugadas donde la valoración de ambos se incremente.

A continuación, vamos a realizar algunas gráficas bivariadas para tener en cuenta no solo una variable en cada gráfico.

Para comenzar vamos a tener en cuenta las variables que recogen la valoración de los mejores jugadores, y la variable a predecir *HOME_TEAM_WINS*. Teniendo en cuenta estas variables vamos a extraer cuándo el equipo local gana teniendo en cuenta la suma de los 3 mejores jugadores de cada equipo. Por lo tanto, en la Figura 3.5 podemos observar que:

- El 33.6% de los equipos locales con mejor valoración en sus jugadores que el visitante acaba ganando.
- El 16.1% de los equipos locales con mejor valoración en sus jugadores que el visitante acaba perdiendo. Por lo que, podemos concluir que generalmente cuando el equipo local tiene mejor equipo no suele perder.
- El 24.9% de los equipos visitantes con mejor valoración en sus jugadores que el local acaba ganando.
- El 25.4% de los equipos visitantes con mejor valoración en sus jugadores que el local acaba perdiendo.

Por lo que, como podemos observar que tenga mejor equipo, el equipo local es más significativo para los locales porque tienen más probabilidades de ganar. Sin embargo, que tenga mejor equipo el visitante no es tan significante como en los equipos locales, ya que, la probabilidad es muy similar para cuando ganan como para cuando pierden con mejor equipo que el local.

A continuación, vamos a observar cómo el promedio de puntos por partido ha ido aumentando conforme pasan los años, tanto por parte del equipo local como del equipo visitante. A su vez, vemos que el equipo local tiene ligeramente mayor promedio de puntos que el equipo visitante, lo cual es razonable tener en cuenta con respecto al porcentaje de victorias, ya que, si los equipos locales promedian más puntos por partido que los visitantes entonces, generalmente los locales ganarán más partidos que los visitantes, tal y como hemos podido ver anteriormente en la Figura 3.6. Esta superioridad en los puntos por partido por temporadas por parte de los equipos locales es algo que podemos observar en la Figura 3.6.

3. Desarrollo del proyecto

PTS_A	1	0.55	0.27	0.48	0.19	0.36	0.15	0.28	0.14	0.023
PTS_H	0.55	1	0.47	0.24	0.36	0.16	0.26	0.13	0.24	0.3
VAL_1_E1	0.27	0.47	1	0.14	0.66	0.1	0.34	0.074	0.16	0.41
VAL_1_E2	0.48	0.24	0.14	1	0.1	0.66	0.089	0.34	0.077	0.016
PTS_1_E1	0.19	0.36	0.66	0.1	1	0.077	0.073	0.047	-0.0058	0.24
PTS_1_E2	0.36	0.16	0.1	0.66	0.077	1	0.065	0.077	0.054	0.0063
PTS_2_E1	0.15	0.26	0.34	0.089	0.073	0.065	1	0.037	0.028	0.18
PTS_2_E2	0.28	0.13	0.074	0.34	0.047	0.077	0.037	1	0.054	0.019
PTS_3_E1	0.14	0.24	0.16	0.077	-0.0058	0.054	0.028	0.054	1	0.14
W_PCT_H	0.023	0.3	0.41	0.016	0.24	0.0063	0.18	0.019	0.14	1
PTS_3_E2	0.22	0.12	0.076	0.16	0.056	-0.013	0.05	0.031	0.031	0.011
VAL_2_E1	0.29	0.5	0.55	0.16	0.35	0.1	0.57	0.079	0.32	0.38
VAL_2_E2	0.52	0.26	0.15	0.55	0.097	0.34	0.083	0.58	0.082	0.028
PLAYER_ID_3_E1	0.43	0.35	0.11	0.22	0.064	0.14	0.018	0.1	0.12	-0.062
VAL_3_E1	0.28	0.49	0.39	0.15	0.23	0.096	0.34	0.093	0.54	0.35
PLAYER_ID_2_E1	0.44	0.35	0.1	0.23	0.017	0.15	0.064	0.11	0.11	-0.07
W_PCT_A	0.24	0.047	0.018	0.41	0.017	0.23	0.027	0.18	0.0017	0.15
REB_H	0.34	0.53	0.21	0.12	0.12	0.082	0.091	0.067	0.092	0.2
	PTS_A	PTS_H	VAL_1_E1	VAL_1_E2	PTS_1_E1	PTS_1_E2	PTS_2_E1	PTS_2_E2	PTS_3_E1	W_PCT_H

PTS_A	0.22	0.29	0.52	0.43	0.28	0.44	0.24	0.34		
PTS_H	0.12	0.5	0.26	0.35	0.49	0.35	0.047	0.53		
VAL_1_E1	0.076	0.55	0.15	0.11	0.39	0.1	0.018	0.21		
VAL_1_E2	0.16	0.16	0.55	0.22	0.15	0.23	0.41	0.12		
PTS_1_E1	0.056	0.35	0.097	0.064	0.23	0.017	0.017	0.12		
PTS_1_E2	-0.013	0.1	0.34	0.14	0.096	0.15	0.23	0.082		
PTS_2_E1	0.05	0.57	0.083	0.018	0.34	0.064	0.027	0.091		
PTS_2_E2	0.031	0.079	0.58	0.1	0.093	0.11	0.18	0.067		
PTS_3_E1	0.031	0.32	0.082	0.12	0.54	0.11	0.0017	0.092		
W_PCT_H	0.011	0.38	0.028	-0.062	0.35	-0.07	0.15	0.2		
PTS_3_E2	1	0.081	0.32	0.13	0.083	0.12	0.13	0.061		
VAL_2_E1	0.081	1	0.16	0.14	0.67	0.15	0.023	0.2		
VAL_2_E2	0.32	0.16	1	0.24	0.17	0.25	0.37	0.15		
PLAYER_ID_3_E1	0.13	0.14	0.24	1	0.19	0.5	-0.0087	0.2		
VAL_3_E1	0.083	0.67	0.17	0.19	1	0.18	0.022	0.2		
PLAYER_ID_2_E1	0.12	0.15	0.25	0.5	0.18	1	0.00047	0.21		
W_PCT_A	0.13	0.023	0.37	-0.0087	0.022	0.00047	1	0.035		
REB_H	0.061	0.2	0.15	0.2	0.2	0.21	0.035	1		
	PTS_3_E2	VAL_2_E1	VAL_2_E2	PLAYER_ID_3_E1	VAL_3_E1	PLAYER_ID_2_E1	W_PCT_A	REB_H		

Figura 3.4: Matriz de correlación de las 20 variables con mayor ganancia de información.

Valoración de los mejores jugadores y victoria del equipo correspondiente

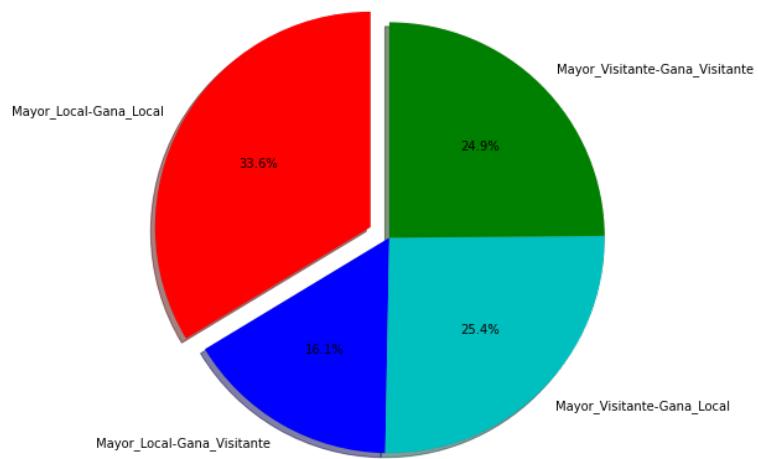


Figura 3.5: Victoria de los equipos teniendo en cuenta los mejores jugadores.

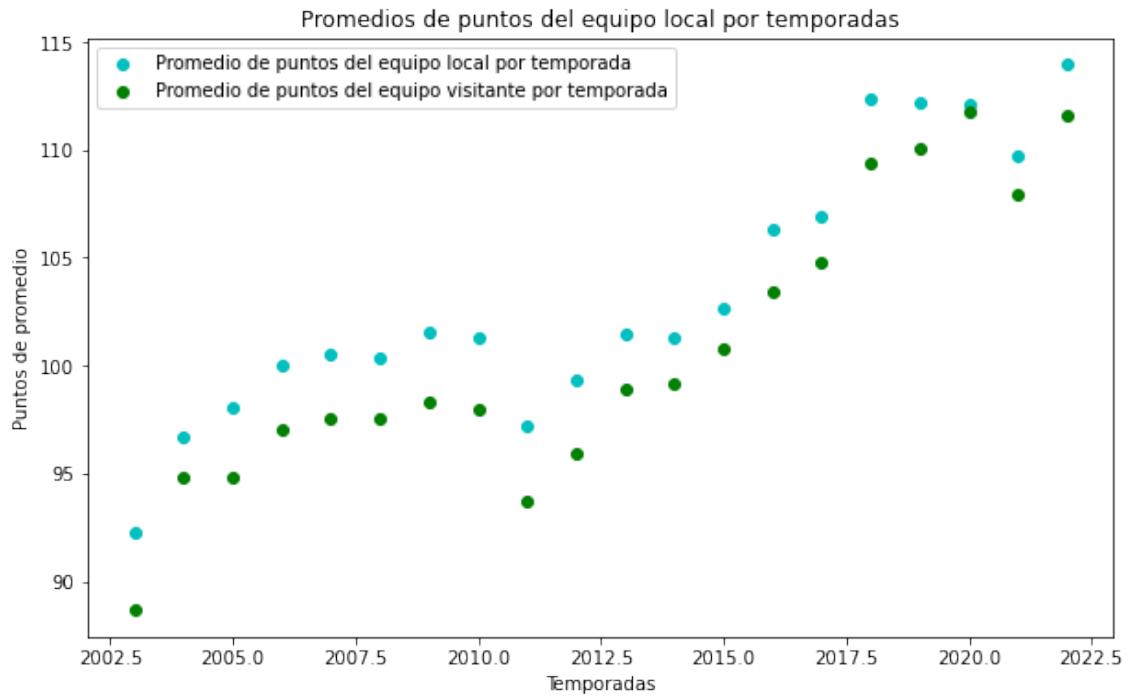


Figura 3.6: Porcentaje de victorias del equipo local.

Otras de las variables a tener en cuenta son las que representan a los mejores jugadores de cada equipo respecto a su valoración por partido hasta la jornada considerada. En la Figura 3.7 podemos observar un gráfico de densidad para las variables VAL_1_E1 , VAL_2_E1 y VAL_3_E1 , las cuales representan a los mejores jugadores del equipo local, donde obtenemos que la valoración suele variar:

- Para VAL_1_E1 entre 15 y 25 en su mayoría, aunque hay algunos pocos casos donde baja de 15 y sube de 25.
- Para VAL_2_E1 entre 12 y 20 en su mayoría, aunque hay algunos pocos casos donde baja de 12 y sube de 20.
- Para VAL_3_E1 entre 10 y 17 en su mayoría, aunque hay algunos pocos casos donde baja de 12 y sube de 20.

Por lo tanto, observamos que, tal y como comentamos en la Sección 3.1 a la hora de crear estas variables, los mejores jugadores serán los que mayor valoración por partido obtengan.

Aunque solo hemos mostrado los gráficos de densidad para los mejores jugadores del equipo local, los mejores jugadores del equipo visitante tienen unas gráficas similares. Además, estas variables son de una alta importancia, ya que, en la NBA, los jugadores estrellas con mejores estadísticas amplían de manera significante las probabilidades de ganar [17].

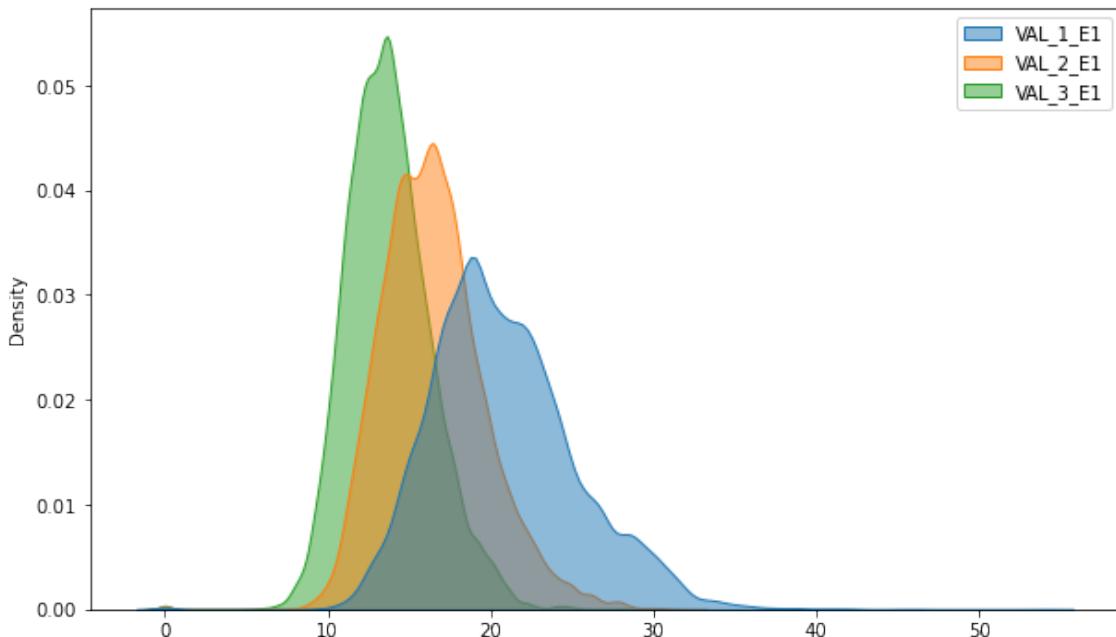


Figura 3.7: Matriz de correlación de las 20 variables con mayor ganancia de información.

Para concluir, vamos a extraer algunas gráficas de densidad bivariadas, como por ejemplo la Figura 3.8, en donde podemos observar que esta variable separa bien los valores de *HOME_TEAM_WINS* cortando en el punto 0.5, por lo que, podemos concluir que la variable *W_PCT_H* será una variable significativa. Podemos fijarnos que en el punto 0.0 hay pico pronunciado, esto se debe a las primeras jornadas de cada temporada, donde no podemos obtener promedios anteriores a dichas jornadas.

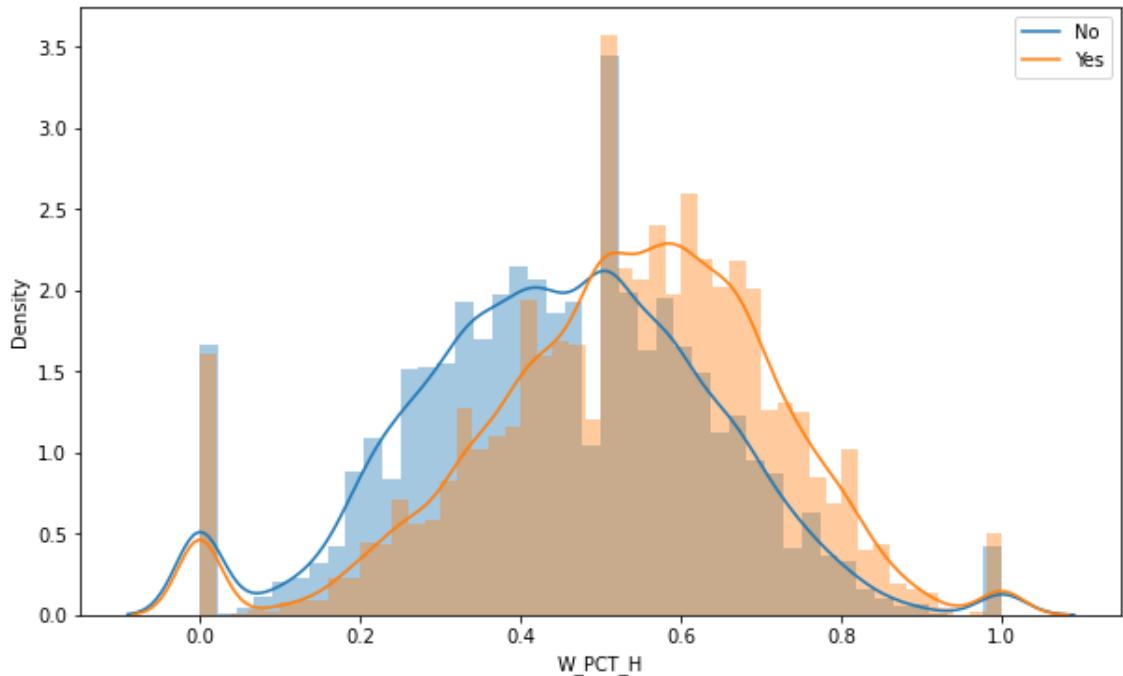


Figura 3.8: Gráfico de densidad de las variables *HOME_TEAM_WINS* y *W_PCT_H*.

A continuación, podemos ver en la Figura 3.9 un gráfico de densidad de las variables *PTS_1_E1* y *HOME_TEAM_WINS*, donde podemos ver que corta sobre el punto 23 y donde observamos como el equipo local tiende a ganar cuando su mejor jugador anota más de 23 puntos por partido aproximadamente. Además, se puede observar una distribución aproximadamente normal debido a que la curva es simétrica y que la mayor concentración de los datos se encuentra en el centro de la distribución. Viendo la Figura 3.9 podemos concluir que esta variable es significativa.

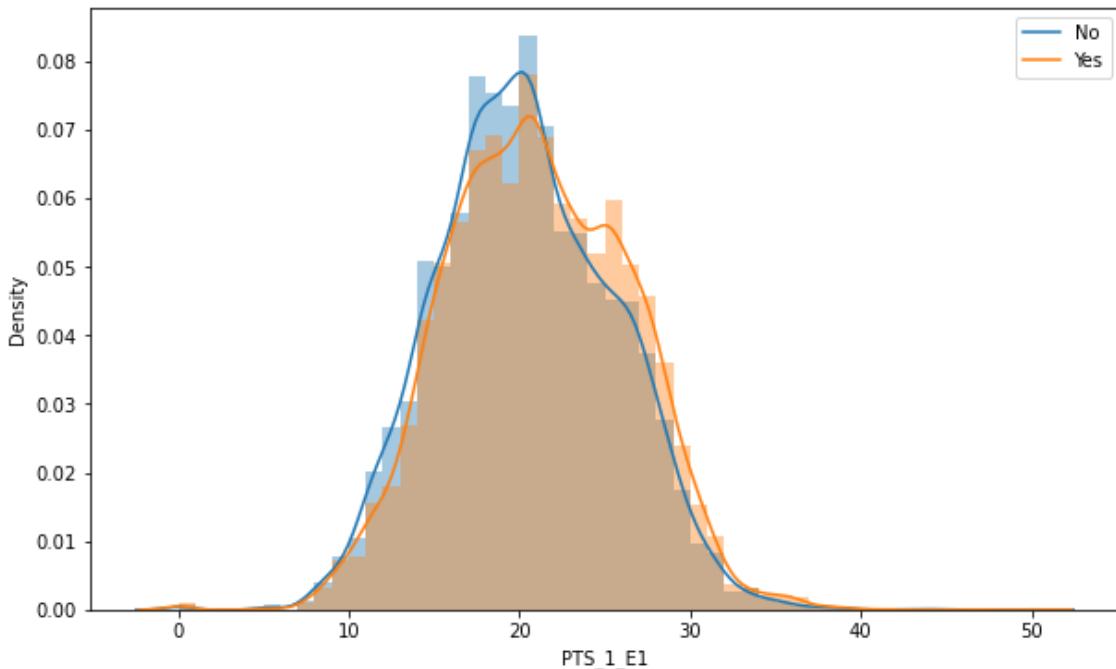


Figura 3.9: Gráfico de densidad de las variables *HOME_TEAM_WINS* y *PTS_1_E1*.

3.3. Preprocesamiento de los datos

El preprocesamiento de datos es un paso muy importante en un proyecto de minería de datos, donde los datos se transforman o codifican para llevarlos a un estado el cual la máquina pueda analizarlos fácilmente [37].

Se estima que el preprocesamiento de datos y la Sección 3.1, comprensión de los datos y creación de la base de datos, ocupa al menos, un 80% del esfuerzo dedicado a un proyecto. Además, si hay demasiados datos redundantes, ruidosos o irrelevantes, el modelo aprendido se va a ajustar a este tipo de datos en lugar de a la información relevante, obteniendo resultados inesperados.

El preprocesamiento de datos contiene las siguientes tareas:

- Limpieza de datos. Se rellenan los valores que faltan, se detectan y se quitan los valores atípicos y los datos con ruido [8].
- Integración de datos. Se basa en combinar múltiples tablas o registros para crear nuevos registros o valores [25].

-
- Transformación de datos. Se normalizan los datos para reducir el ruido y las dimensiones.
 - Reducción de datos. Consiste en decidir qué datos deben ser utilizados para el análisis y si el criterio que se sigue incluye la relevancia con respecto a los objetivos [25].

En nuestro proyecto se aplican muchas de estas tareas. Por ejemplo, realizaremos limpieza de datos poniendo los valores NaN a 0 en los promedios de los primeros partidos de la base de datos, ya que, al ser los primeros partidos, no tenemos estadísticas previas, y que, así, podamos realizar los modelos posteriores sin que aparezcan excepciones. También, los valores de algunas variables categóricas se codificaron.

Tal y como hemos visto en la sección anterior, realizamos integración de datos para crear nuevos registros en nuestra base de datos final.

Asimismo, se realizó una reducción de datos eliminando las variables que no eran relevantes para el modelado, como pueden ser por ejemplo la id del partido, el nombre del equipo, nombre del mejor jugador, etc. Dejando una base de datos con solo variables numéricas y que sean significativas (tengan poder discriminativo) en nuestros futuros modelos.

Para concluir con el preprocesamiento de datos, realizaremos una normalización de la base de datos, donde los valores varían entre 0 y 1. Este proceso se hace debido a que es necesaria para que algunos algoritmos modelen los datos correctamente, evita problemas y, generalmente, los algoritmos de aprendizaje automático funcionan mejor.

3.4. Creación de los modelos

3.4.1. Introducción

Una vez se haya realizado el correspondiente preprocesamiento de datos, tal y como hemos detallado en la sección anterior, tenemos que realizar la siguiente fase de la metodología CRISP-DM, el modelado.

En esta fase es donde desarrollaremos algoritmos y aplicaremos técnicas de minería de datos para problemas de clasificación y, además, ajustaremos lo mejor posible los parámetros para obtener los clasificadores óptimos.

Los modelos que realizaremos serán de los siguientes dos tipos:

- **Estáticos.** Modelos que se construyen utilizando un conjunto fijo de datos y parámetros, y no se actualizan con nuevos datos a medida que se recopilan. Estos modelos son

útiles cuando los datos son estáticos y los tenemos disponibles a la hora de entrenar, si no se deberían entrenar muchos modelos estáticos para simular un proceso dinámico y, entonces se podría obtener una gran penalización computacional.

- **Dinámicos.** Modelos que son más flexibles y precisos en entornos donde los datos llegan secuencialmente con el tiempo o incluso pueden cambiar de tendencia con el tiempo, como puede ser nuestro problema donde las temporadas más recientes cobran más importancia que las más antiguas. Estos modelos son ideales para problemas que requieren respuestas en tiempo real y necesitan adaptarse a condiciones cambiantes. Algunos modelos incrementales como los *hoeffding trees*, aprenden conforme llegan los datos y algunas estrategias consiguen descubrir cuando ha habido un cambio y se van descartando algunos modelos.

Previamente a la creación de los modelos, definiremos cuáles serán las variables predictoras, las cuales serán las que utilizaremos para predecir la variable objetivo *HOME_TEAM_WINS*. En nuestro problema descartaremos de este conjunto de variables predictoras los nombres de los equipos y jugadores, sus correspondientes ids, y la fecha y el id del partido correspondiente.

Dentro de estos dos tipos de modelos, aplicaremos distintas técnicas de minería de datos para clasificación, como pueden ser árboles de decisión o *ensembles* como *Random Forest*, *Bagging*, *Boosting* o *Gradient Boosting*, entre muchas otras técnicas.

Además, cabe destacar que, se va a probar a entrenar y predecir con diferentes subconjuntos de entrenamiento y prueba, los cuales pueden representar una semana o un partido. Por lo tanto, es importante, darle importancia tanto a la precisión con la que el algoritmo predice, como al tiempo de ejecución que esto conlleva.

Antes de comenzar a describir y desarrollar los distintos modelos que se realizaron, vamos a definir los 2 escenarios que se consideraron para entrenar y predecir. Los cuales son:

- Predicción por **partidos**. En este tipo de algoritmo, tomaremos como conjunto de entrenamiento todos los partidos anteriores al partido que vayamos a predecir. Este método es eficaz, ya que, tiene en cuenta partidos de la temporada actual en la que nos situemos, sin embargo, puede conllevar un tiempo de ejecución elevado, por lo que, solo se utilizará este escenario con árboles de decisión con estrategias de olvido debido a que con los demás métodos el tiempo de ejecución es muy elevado.
- Predicción por **semanas**. Este será el método que utilizaremos en todos los modelos y el que se consideró el más eficaz, teniendo en cuenta el tiempo de ejecución y la precisión con la que se predice. El algoritmo consiste en predecir una semana y entrenar con todas las semanas anteriores a esa semana y, por consiguiente, se reduce el tiempo de ejecución con respecto al método comentado anteriormente y, además, la precisión en estos modelos no se ve muy afectada. En la imagen 3.10 se puede observar cómo funciona el entrenamiento y la predicción por semanas.

Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	...	Semana _n
Entrenas hasta semana 1	Predigo semana 2					
Entrenas hasta semana 2		Predigo semana 3				
Entrenas hasta semana 3			Predigo semana 4			
Entrenas hasta semana 4				Predigo semana 5		
Entrenas hasta semana n-1						Predigo semana n

Figura 3.10: Gráfico de entrenamiento por semanas.

Es importante resaltar que, se empieza a entrenar y predecir a partir de la décima temporada, en nuestro caso es la temporada 2013, para así tener un historial suficiente del que poder aprender. Se estimó que no era necesario predecir las 20 temporadas para determinar la precisión de los modelos.

Además, es de gran importancia resaltar el hecho de que se va a entrenar y predecir en todos los modelos por semanas, ya que se pudo observar que no afectaba significativamente a la precisión de los algoritmos, pero, sin embargo, sí se conseguía reducir considerablemente el tiempo de ejecución.

Para concluir, hay que detallar que en la selección de los modelos donde se prueban distintas configuraciones de hiperparámetros, se va a validar desde la temporada 2013 a la temporada 2020 y, posteriormente, en la construcción y validación del modelo final se obtendrá un modelo con los mejores hiperparámetros obtenidos en la etapa de selección de modelos para cada algoritmo, entrenando y prediciendo con las dos últimas temporadas, es decir, las temporadas 2021 y 2022.

3.4.2. Selección de modelos estáticos

Los modelos estáticos son aquellos en donde se entrena y se predice de la misma manera durante todo el proceso, es decir, se entrena y se predice tal y como se ha podido ver en la imagen 3.10.

Este tipo de modelos puede ser poco eficaz frente a problemas donde los datos futuros puedan cambiar, esto puede producir que, la precisión con la que el algoritmo predice puede verse considerablemente afectada de manera negativa.

Sin embargo, estos modelos suelen ser más rápidos que los modelos dinámicos, ya que, no precisan de modificaciones ni alteraciones durante la ejecución del correspondiente algoritmo.

Las técnicas que se aplicarán en este tipo de modelos serán árboles de decisión y algunos *ensembles* como *bagging (bootstrap aggregating)*, *random forest*, *boosting (adaptative boosting)* y *gradient boosting*, y redes neuronales.

Árboles de decisión

Aunque se podrían probar una inmensa cantidad de combinaciones de hiperparámetros, hemos decidido limitar esas combinaciones a un espacio determinado de búsqueda. Se ha decidido delimitar ese espacio debido a que si utilizamos un conjunto demasiado grande de hiperparámetros el tiempo de ejecución puede llegar a ser muy alto. Los hiperparámetros que se han decidido seleccionar son los siguientes.

- **Criterio de división (*criterion*)**. Este hiperparámetro determina cómo se medirá la impureza de un *split*, cuyo valor puede ser “*gini*”, “*entropía*” o “*log_loss*”. El valor predeterminado será “*gini*” [22]. En nuestros modelos probaremos tanto con “*gini*” como con “*entropía*”.
- **Máximo número de nodos hoja (*max_leaf_nodes*)**. Este hiperparámetro hará crecer el árbol con un número máximo de nodos hoja, donde los mejores nodos se definen como una reducción relativa de la impureza [9]. El valor de este hiperparámetro por defecto es de ”*None*”, por lo que, se utilizaría un número ilimitado de nodos hoja. En los siguientes modelos se decidió probar con los valores 2, 30, 60 y 100.
- **Profundidad máxima (*max_depth*)**. Este hiperparámetro indica el número máximo de profundidad, donde el valor predeterminado será ”*None*”, es decir, habrá un número ilimitado de nodos hoja. En los modelos realizados se utilizaron los valores 25, 50 y ”*None*”.

Con los anteriores hiperparámetros podemos obtener un conjunto de combinaciones posibles, donde tendremos 24 árboles distintos, de los cuales, una vez entrenados obtenemos los siguientes resultados.

Resultados de los modelos usando árboles de decisión

Los resultados correspondientes a los 8 mejores modelos con *score* realizados con árboles de decisión, ordenados de mayor a menor tasa de acierto, son los que podemos visualizar en la tabla 3.1.

Tal y como podemos observar en la tabla 3.1, los mejores modelos son los que, generalmente, tienen 30 como valor para *max_leaf_nodes*, además, podemos observar cómo utilizando la entropía como criterio de división obtenemos un *score* ligeramente mejor que utilizando *gini*, sin embargo, el tiempo de ejecución de los modelos utilizando *gini* es menor que cuando utilizamos la entropía.

Hay que destacar, que para el desarrollo de los siguientes modelos donde tengamos que utilizar árboles de decisión como estimador base, utilizaremos como criterio de división *gini*, debido a la reducción del tiempo de ejecución que ofrecía y a que, se observó que obtenía un promedio de *score* mejor.

criterion	max_leaf_nodes	max_depth	score	Tiempo
<i>entropy</i>	30	25	0.6404	138.31s
<i>entropy</i>	30	50	0.6403	138.63s
<i>entropy</i>	30	<i>None</i>	0.6403	139.46s
<i>gini</i>	30	25	0.6364	118.76s
<i>gini</i>	30	50	0.6364	118.91s
<i>gini</i>	30	<i>None</i>	0.6364	118.92s
<i>gini</i>	60	<i>None</i>	0.6347	129.05s
<i>gini</i>	60	50	0.6346	128.23s

Tabla 3.1: Resultados árboles de decisión

Además, tal y como podemos ver en la Figura 3.11, hemos representado visualmente un árbol de decisión con criterio de división *entropy*, 30 como máximo número de nodos hoja y 25 de profundidad, del cual obteníamos un *score* de 0.6404 en 138.31 segundos, tal y como se ha podido ver anteriormente.

En la Figura 3.11 podemos ver como el árbol elige en reiteradas ocasiones las variables *W_PCT_H* y *W_PCT_A* en los nodos superiores, por lo tanto, podemos indicar que las variables que registran el porcentaje de victorias de los equipos locales y visitantes son muy significantes a la hora de predecir. Asimismo, tal y como vimos en la Sección 3.2, estas dos variables no están muy correlacionadas entre sí, pero cuando obtuvimos las 20 variables con mayor ganancia de información, sí que ambas pertenecían a este conjunto, por lo que, se puede afirmar que son variables informativas.

Seguidamente, en los nodos inferiores podemos ver otras variables que también son significativas, aunque en menor medida que las anteriores, como pueden ser las valoraciones por partido de ciertos jugadores, las victorias hasta la fecha correspondiente de los equipos o, incluso, en un nodo las asistencias por partido del mejor jugador del equipo visitante.

3. Desarrollo del proyecto

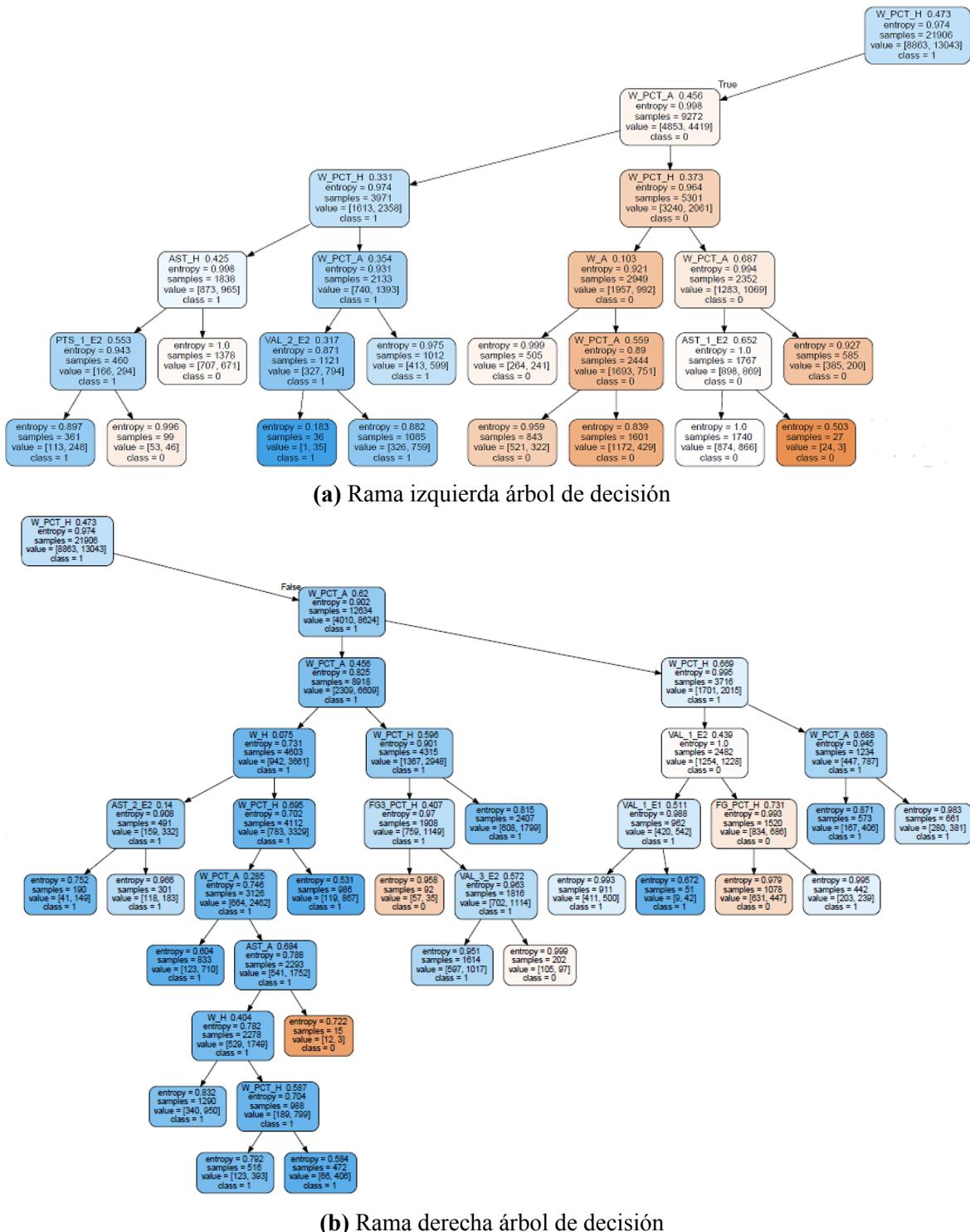


Figura 3.11: Árbol de decisión.

Bagging (Bootstrap aggregating)

Al igual que en el modelo anterior, podemos obtener un amplio conjunto de combinaciones de hiperparámetros, pero se va a delimitar ese espacio de búsqueda. Los hiperparámetros que vamos a seleccionar son los siguientes.

- **Estimador base (*base_estimator*)**. Este hiperparámetro será un estimador base, el cual será un conjunto de árboles de decisión. Estos árboles de decisión tendrán a su vez, diferentes hiperparámetros, los cuales fueron limitados en relación con el rendimiento de los anteriores árboles de decisión.
 - **Criterio de división (*criterion*)**. El cual será “*gini*”.
 - **Máximo número de nodos hoja (*max_leaf_nodes*)**. Se consideró que sería 30.
 - **Profundidad máxima (*max_depth*)**. La profundidad máxima será 25, 50 o *None*.
- **Número de estimadores (*n_estimators*)**. Representa el número de estimadores base en el *ensemble*. El valor predeterminado de este hiperparámetro es 10, en nuestro caso probaremos con 20 y 50 estimadores base [11].

Dados los anteriores hiperparámetros obtenemos un conjunto de 6 combinaciones posibles, es decir, obtenemos 6 modelos con bagging.

Resultados de modelos utilizando *bagging (Bootstrap aggregating)*

Los resultados correspondientes a los 6 modelos realizados, ordenados de mayor a menor precisión, son los que podemos visualizar en la tabla 3.2.

En la Tabla 3.2 podemos observar como los modelos en los que utilizamos 50 estimadores base obtienen un *score* ligeramente mejor que los que usan solo 20 aunque, sin embargo, estos modelos llevan un mayor tiempo de ejecución.

Por lo tanto, hay que tener en cuenta tanto la precisión del algoritmo como el tiempo de ejecución, ya que, la diferencia de precisión entre los modelos es prácticamente de centésimas mientras que, el tiempo de ejecución sí que aumenta considerablemente. Se puede concluir que con el aumento del valor de los estimadores base obtendremos resultados ligeramente mejores.

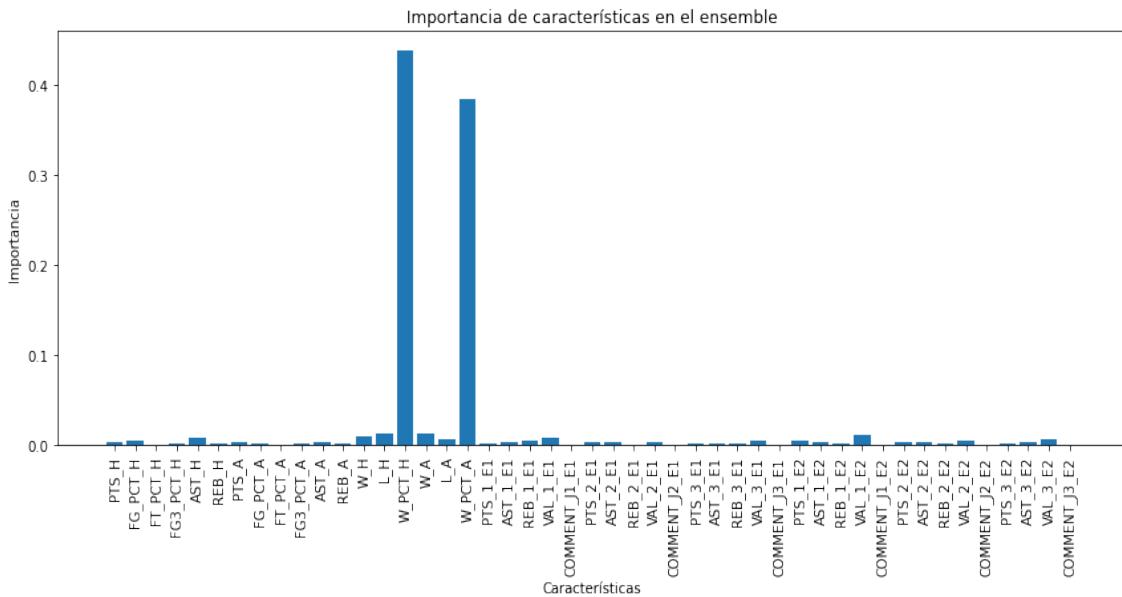
Debido al buen rendimiento que proporcionó los valores de *gini* y 30 en las variables *criterion* y *max_leaf_nodes* respectivamente, se decidió realizar el modelo con un estimador base que contenga esos dos valores y se fuera alternando en la profundidad máxima del árbol.

base_estimator			n_estimators	score	Tiempo
criterion	max_leaf_nodes	max_depth			
<i>gini</i>	30	None	20	0.6473	854.77s
<i>gini</i>	30	25	20	0.6466	2035.28s
<i>gini</i>	30	50	50	0.6452	1975.73s
<i>gini</i>	30	50	None	0.6450	1989.25s
<i>gini</i>	30	25	20	0.6448	864.32s
<i>gini</i>	30	50	20	0.6446	810.02s

Tabla 3.2: Resultados bagging

Tal y como podemos ver en la Figura 3.12, las variables más significativas del mejor modelo obtenido, son las que indican el porcentaje de victorias que tiene cada equipo en cada jornada, es decir, las variables *W_PCT_H* y *W_PCT_A*. Estas variables coinciden con las que, en la anterior sección, el árbol de decisión elegía en los nodos superiores y consideraba más significativas.

Asimismo, las variables que registran las victorias, las derrotas y las que indican la valoración del mejor jugador de cada equipo tienen mayor importancia que las demás, sin embargo, tienen una importancia mucho menor a las registradas con los porcentajes de victorias.


Figura 3.12: Gráfico de las variables más significativas en modelo bagging.

Random forest

Tal y como hemos llevado a cabo durante la realización de los modelos anteriores, se va a obtener un conjunto de combinaciones de diferentes hiperparámetros. Los correspondientes hiperparámetros son los siguientes.

- **Criterio de división (criterion).** Esta característica ya ha sido citado anteriormente, el cual tiene el valor de "gini" por defecto. En nuestro modelo tenemos dos posibles opciones, "gini" y "entropy".
- **Máximo número de nodos hoja (max_leaf_nodes).** Su valor predeterminado es *None*, aunque se considerará el valor de 30.
- **Número máximo de características (max_features).** Indica el número máximo de características que se consideran al observar la mejor decisión al dividir. Esta variable tomará los valores de "sqrt" y "log2", los cuales realizan la raíz cuadrada y el logaritmo en base de dos del número de máxima de características posible respectivamente. El valor por defecto es "sqrt".
- **Número de estimadores (n_estimators).** Representa el número de estimadores en el *ensemble*. El valor predeterminado de este hiperparámetro es 100, en nuestro caso probaremos con 20 y 50 estimadores.

Por lo tanto, en este modelo tendremos un conjunto de 12 combinaciones posibles de los que, a continuación, mostraremos tanto su precisión como su tiempo de ejecución.

Resultados de modelos utilizando *random forest*

Los mejores resultados que se obtuvieron de los modelos correspondientes al *ensemble random forest* son los que podemos visualizar en la Tabla 3.3.

Tal y como podemos observar en la Tabla 3.3, obtenemos modelos con mayor tasa de acierto si aumentamos el número de estimadores, ya que, los modelos con 50 y 100 estimadores tienden a obtener mejores resultados que en los que se usan 20 estimadores, pero, a su vez, los modelos con mayor número de estimadores tienen un mayor tiempo de ejecución. Pese a esta última observación, la diferencia de precisión entre modelos es muy pequeña, pero el tiempo de ejecución sí que puede llegar a ser significativo si aumentásemos, aún más, el número de estimadores.

criterion	max_leaf_nodes	max_features	n_estimators	score	Tiempo
<i>gini</i>	30	<i>sqrt</i>	20	0.6400	173.88s
<i>gini</i>	30	<i>sqrt</i>	50	0.6391	338.57s
<i>gini</i>	30	<i>sqrt</i>	100	0.6385	613.33s
<i>gini</i>	30	<i>log2</i>	50	0.6384	317.43s
<i>entropy</i>	30	<i>sqrt</i>	20	0.6382	204.33s
<i>gini</i>	30	<i>log2</i>	100	0.6380	559.71s
<i>entropy</i>	30	<i>sqrt</i>	100	0.6379	793.97s
<i>entropy</i>	30	<i>log2</i>	100	0.6368	693.73s

Tabla 3.3: Resultados *random forest*

En la Figura 3.13 podemos observar las variables que cobran más importancia que, al igual que en modelos anteriores, son las que indican el porcentaje de victorias de ambos equipos, sin embargo, a diferencia de los modelos anteriores, podemos ver cómo en este modelo se le da una significancia mayor a algunas variables que antes no se le daba, como puede ser la característica que indica el porcentaje de tiros acertados por partido del equipo local, como local, o incluso, la valoración del mejor jugador de ambos equipos, además de las victorias y derrotas de ambos equipos.

Por lo tanto, podemos observar un cambio de tendencia a la hora de considerar más o menos importantes a ciertas características respecto a los anteriores modelos, donde, prácticamente, solo se tenían en cuenta, o en mayor medida, las variables *W_PCT_H* y *W_PCT_A*.

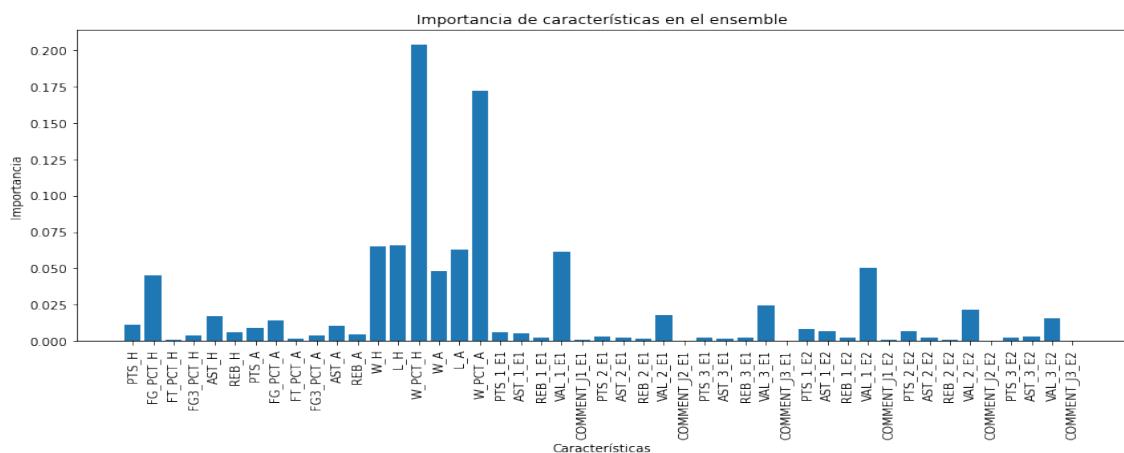


Figura 3.13: Gráfico de las variables más significativas en modelo *random forest*.

Adaptative boosting

A continuación, se van a comentar los diferentes hiperparámetros que se configurarán a la hora de crear los correspondientes modelos. En los siguientes modelos, vamos a modificar los mismos hiperparámetros que se trataron en el *ensemble bagging*, por lo que, como ya está explicado anteriormente qué significa cada hiperparámetro, nos limitaremos a indicar los valores que tendrán cada uno de ellos, los cuales son los siguientes.

- **Estimador base (*base_estimator*)**. Hiperparámetros de los estimadores base:
 - **Máximo número de nodos hoja (*max_leaf_nodes*)**. Se consideró que sería 30.
 - **Profundidad máxima (*max_depth*)**. La profundidad máxima será 1,2 y 3.
- **Tasa de aprendizaje (*learning_rate*)**. Este hiperparámetro representa cómo de rápido se actualizan los pesos en respuesta al error calculado durante el entrenamiento. Una tasa de aprendizaje baja hará que se aprenda más despacio, mientras que una alta puede hacer que los parámetros no se ajusten de manera correcta al conjunto de datos. El valor por defecto es 1.0, pero en nuestro caso probaremos con 0.95 y 1.0.
- **Número de estimadores (*n_estimators*)**. Probaremos con 20, 50 y 100 estimadores.

Por lo tanto, tal y como podemos comprobar, observando los hiperparámetros anteriores, podemos obtener 12 combinaciones posibles.

Resultados de modelos utilizando *adaptative boosting*

Los resultados, ordenados por *score* obtenido, que corresponden al conjunto de los 8 mejores modelos realizados, se pueden visualizar en la Tabla 3.4.

Al observar la Tabla 3.4, podemos percatarnos de cómo los modelos donde proporcionamos un mayor número de *n_estimators*, indican un leve aumento de la precisión frente a los modelos con menos número de estimadores. Además, los modelos con menos estimadores llevan un tiempo de ejecución menor que los propuestos con un número mayor de estimadores.

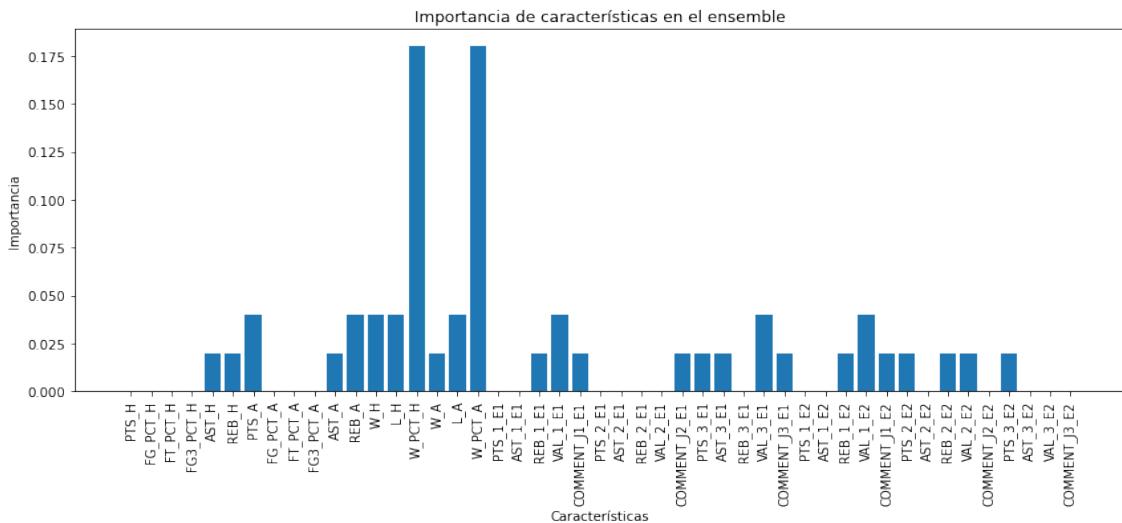
Asimismo, podemos observar cómo los modelos donde la profundidad es menor ofrece tanto mejor precisión como menor tiempo de ejecución si comparáramos modelos con el mismo número de estimadores base y distintas profundidades máximas.

<i>base_estimator</i>		<i>learning_rate</i>	<i>n_estimators</i>	<i>score</i>	Tiempo
<i>max_leaf_nodes</i>	<i>max_depth</i>				
30	1	0.95	50	0.6457	626.63s
30	1	1.0	50	0.6452	637.14s
30	1	0.95	20	0.6403	293.90s
30	1	1.0	20	0.6400	294.94s
30	2	0.95	20	0.6393	493.39s
30	2	1.0	20	0.6380	493.19s
30	3	0.95	20	0.6368	653.67s
30	3	1.0	20	0.6359	655.54s

Tabla 3.4: Resultados *adaptive boosting*

En la Figura 3.14 podemos visualizar cómo, el mejor modelo obtenido le da mucha importancia a variables que no coinciden con las que mayor ganancia de información proporcionaban, las cuales fueron obtenidas en la Sección 3.2.

Es cierto que, como se puede apreciar en la Figura 3.14, las variables que más importancia reciben coinciden con las que se consideran más relevantes en los anteriores *ensembles*. Sin embargo, a diferencia del anterior modelo, le da un poco menos de importancia a las variables que registran las victorias y derrotas de los equipos, mientras que, a las variables que registran las estadísticas individuales del mejor jugador de ambos equipos le da más importancia.


Figura 3.14: Gráfico de las variables más significativas en modelo *adaptive boosting*.

Gradient boosting

A continuación, se van a detallar los hiperparámetros que vamos a configurar en los modelos correspondientes. Se podrán formar un total de 12 combinaciones posibles. Los hiperparámetros con los que se va a trabajar son los siguientes.

- **Número de estimadores (*n_estimators*)**. Número de estimadores en el ensemble, donde el valor predeterminado de este hiperparámetro es 100. En nuestro caso probaremos con 20 y 50.
- **Profundidad máxima (*max_depth*)**. Profundidad máxima de los estimadores individuales, esta profundidad limita el número de nodos en el árbol. El valor por defecto es 3, pero se probará con una profundidad máxima de 1, 2 y 3.
- **Tasa de aprendizaje (*learning_rate*)**. El valor predeterminado es 0.1, sin embargo, en nuestro caso probaremos con 0.05 y 0.1.

Resultados de modelos utilizando *gradient boosting*

En la Tabla 3.5 podemos visualizar los 8 mejores modelos, en función de la precisión de cada uno de ellos, y ordenados de mayor a menor por el *score* correspondiente.

Como se puede apreciar en la Tabla 3.5, los modelos donde el valor de la profundidad es 3 y el número de estimadores es mayor, son en donde se obtiene mejor precisión. Además, cabe mencionar, que se obtuvieron tasas de acierto superiores en aquellos modelos cuya profundidad era 2 o 3, a diferencia de aquellos modelos cuya máxima profundidad era 1, donde se consiguieron resultados ligeramente peores.

<i>max_depth</i>	<i>learning_rate</i>	<i>n_estimators</i>	<i>score</i>	Tiempo
3	0.1	50	0.6509	1422.84s
3	0.05	50	0.6450	1413.44s
2	0.1	50	0.6449	974.06s
3	0.1	20	0.6432	611.43s
2	0.05	50	0.6414	983.22s
2	0.1	20	0.6408	439.66s
1	0.1	50	0.6349	529.86s
3	0.05	20	0.6300	635.36s

Tabla 3.5: Resultados *gradient boosting*

Del mismo modo que en las anteriores secciones, se va a extraer la importancia que el mejor modelo le da a las distintas características que se tienen. En la Figura 3.15 podemos visualizar una gráfica donde se indican cuáles son las variables más importantes en este modelo.

Al igual que las anteriores gráficas extraídas de sus correspondientes *ensembles*, las variables más significativas son las que indican el porcentaje de victorias de los equipos, mientras que, a su vez le da importancia a algunas otras variables como las victorias y derrotas de los equipos, y la valoración de algunos de los mejores jugadores de ambos equipos.

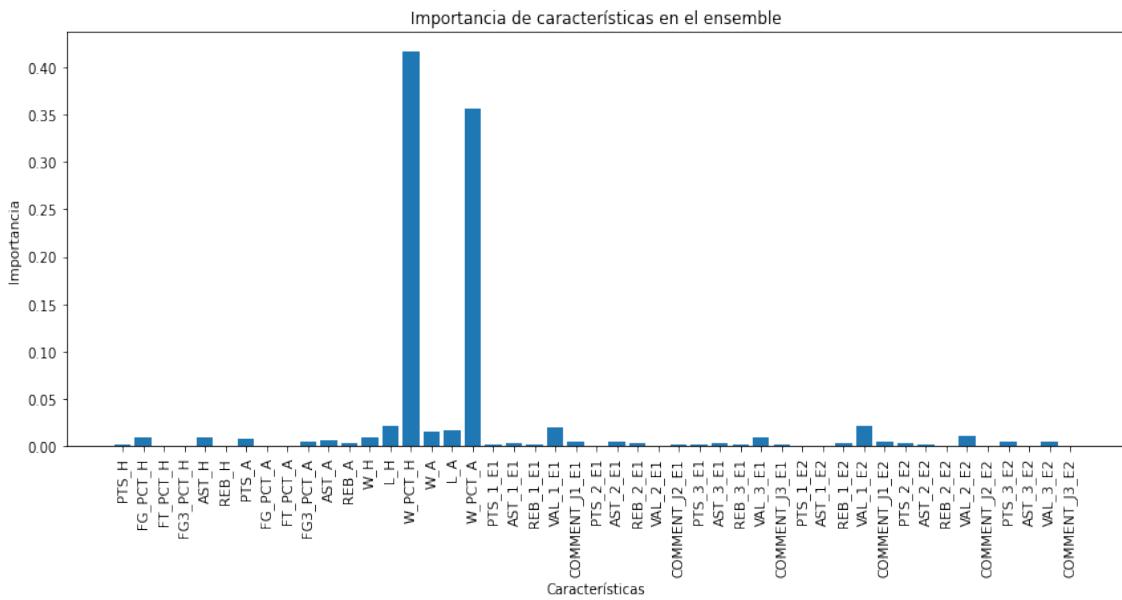


Figura 3.15: Gráfico de las variables más significativas en modelo *gradient boosting*.

Histogram gradient boosting

El clasificador se puede implementar mediante la configuración de diferentes hiperparámetros, los cuales son:

- **Número máximo de iteraciones (*max_iter*)**. Representa el número máximo de iteraciones del proceso de *boosting*, en nuestro caso será el máximo número de árboles para clasificación binaria, cuyo valor será 20 y 50, mientras que, el valor por defecto es 100.
- **Tasa de aprendizaje (*learning_rate*)**. La tasa de aprendizaje, la cual se explicó en la anterior sección, tendrá un valor predeterminado de 0.1, pero se probaran los valores 0.1, 0.03 y 0.05.

-
- **Máximo número de nodos hoja (*max_leaf_nodes*)**. El número máximo predeterminado de nodos hoja es 31, en el caso de nuestros modelos utilizaremos los valores 30 y 60.

Resultados de modelos utilizando *histogram gradient boosting*

A continuación, podemos observar la Tabla 3.6, de la cual podemos percibir cómo obtenemos modelos con una buena tasa de acierto, con valores muy cercanos a los obtenidos en el algoritmo de *gradient boosting*, pero obteniéndolos en un tiempo mucho menor. Los resultados mostrados en la siguiente tabla, serán los mejores 8 modelos obtenidos de las 12 posibles combinaciones.

Tal y como podemos, los modelos donde indicamos mayor número máximo de iteraciones, es en los cuales se obtiene un ligero aumento del *score* correspondiente. Además, tal y como se ha comentado durante esta sección, una de las características a destacar de este tipo de algoritmos era que el tiempo de ejecución era mucho menor que el obtenido en los modelos realizados con *gradient boosting*. Por último, hay que resaltar el hecho de que se obtienen mejores resultados en los modelos donde consideramos el valor de 0.03 y 0.05 como tasas de aprendizaje.

<i>max_leaf_nodes</i>	<i>learning_rate</i>	<i>max_iter</i>	<i>score</i>	Tiempo
30	0.05	50	0.6488	378.11s
60	0.05	50	0.6468	821.22s
60	0.03	50	0.6450	601.72s
30	0.03	50	0.6414	452.78s
30	0.05	20	0.6330	277.30s
60	0.05	20	0.6328	409.30s
60	0.03	20	0.6216	430.25s
30	0.03	20	0.6192	245.00s

Tabla 3.6: Resultados *histogram gradient boosting*

Redes neuronales

Los modelos que se realizaron consisten en una red neuronal secuencial con 5 capas densas completamente conectadas, 4 de ellas son capas ocultas, las cuales contienen 64 unidades y utilizan la función de activación '*relu*' y la última capa, es la capa de salida, la cual tiene una sola unidad y utiliza la función de activación '*sigmoid*'.

El modelo se compila con el optimizador '*Adam*' y la función de pérdida '*binary_crossentropy*'. Además, se tendrán en cuenta como métricas de evaluación la precisión binaria y el área bajo la curva, también conocido como AUC.

Cabe destacar que, la métrica de evaluación área bajo la curva (AUC) es una herramienta que se utiliza para medir el acierto en la predicción de eventos binarios, es decir, eventos que bien ocurren o no ocurren. La curva ROC indica la relación entre la tasa de verdaderos positivos, es decir, el número de elementos identificados correctamente como positivos del total de positivos verdaderos, y la tasa de falsos positivos, es decir, cuando se indica como algo positivo algo que no lo es, para diferentes niveles del umbral [3].

Resultados de modelos utilizando redes neuronales

A continuación, se van a mostrar los resultados obtenidos con el modelo que se ha detallado anteriormente, pero entrenando con un valor de 32 para la variable *batch_size* y con unos valores de 10, 20 y 30 *epochs*.

Los *epochs* serán el número de épocas que se realizan para entrenar el modelo, es decir, en cada predicción se entrenará un número correspondiente. También es de alta importancia detallar qué es el *batch_size*, también conocido como tamaño de lote, el cual divide los datos de entrenamiento en lotes de tamaño 32 en nuestro caso y los pesos del modelo se actualizan posteriormente al procesamiento de cada lote.

Además, una vez obtenidas las predicciones se redondeará los valores de las predicciones a números enteros, ya que, las redes neuronales nos devuelven valores continuos. A las predicciones con valores redondeados se le llamará posteriormente en los resultados, *score*.

En la Tabla 3.7 se van a mostrar los resultados que corresponden a dichos modelos. Además de mostrar el *score*, el tiempo de ejecución y el número de épocas de cada modelo, también se mostrará, los valores obtenidos en el último paso de entrenamiento del valor de la función de pérdida, la precisión binaria obtenida y el valor correspondiente al AUC.

Tal y como se puede apreciar, con un mayor número de *epochs* se obtienen mejores resultados tanto para la función de pérdida, como para el *binary_accuracy*, como para el AUC. Sin embargo, la precisión del modelo no depende del número de épocas realizadas, ya que, podemos observar como con 10 épocas se obtiene un resultado ligeramente mejor que con 20, aunque con 30 épocas también se obtiene mejor tasa de acierto que empleando 20.

Además, se puede percibir que cuanto mayor número de épocas indiquemos, mayor será el tiempo de ejecución del modelo.

<i>loss</i>	<i>binary_accuracy</i>	AUC	<i>epochs</i>	<i>score</i>	Tiempo
0.1076	0.9564	0.9923	10	0.5801	2328.75s
0.0839	0.9671	0.9951	30	0.5774	9852.57s
0.1241	0.9511	0.9899	20	0.5747	6708.68s

Tabla 3.7: Resultados redes neuronales

3.4.3. Selección de modelos dinámicos

Los modelos dinámicos son aquellos que se van actualizando conforme se obtienen nuevas observaciones a lo largo del tiempo. Principalmente, se utilizan para comprender cómo evolucionan las variables en un sistema a lo largo del tiempo y, así, capturar sus cambios de comportamiento.

Este tipo de modelos nos ayudarán a obtener una mejor precisión, ya que, en nuestro problema, los equipos tienden a cambiar bastante a lo largo de los años y, un equipo que puede no ganar nada, a los 5 años puede estar ganando en gran cantidad.

Este tipo de modelos será más preciso que en el caso de los modelos estáticos, sin embargo, el tiempo de ejecución será mayor.

Durante esta sección se detallarán muchos de los algoritmos de clasificación de los que hablamos en los modelos estáticos, pero también se tratarán nuevos algoritmos como es el caso de los *hoeffding trees*.

Para la realización de modelos dinámicos que tengan en cuenta el posible cambio de tendencia, usaremos técnicas de olvido donde implementaremos algoritmos de clasificación como árboles de decisión, *hoeffding trees*, *bagging*, *random forest*, *adaptative boosting*, *gradient boosting* o *Histogram Gradient Boosting*. A continuación se va a detallar en que consisten las estrategias de olvido.

1. Entrenamos y predecimos con un clasificador desde el principio.
2. A partir del punto que se quiera, se empieza a entrenar y predecir con otro nuevo clasificador.

3. Seguidamente en cada predicción se va a comparar la tasa de acierto del primero conforme el segundo.
4. Cuando se perciba un cambio de tendencia el cual definiremos como, por ejemplo, que la tasa de acierto del segundo clasificador sea mayor que la del primero tres veces seguidas, entonces cuando esto se cumpla el primer clasificador pasará a ser el segundo y el segundo pasará a crearse un nuevo clasificador.
5. Se repiten el proceso anterior hasta el final del proceso de entrenamiento y predicción.

En el Algoritmo 3.1, se muestra el pseudocódigo de cómo funciona las estrategias de olvido en nuestro proyecto.

Además de realizar técnicas de olvido de la anterior manera, también realizaremos este tipo de técnicas con *ensembles* con un límite de clasificadores, también denominados *ensembles* dinámicos, donde realizaremos de manera manual un *ensemble* el cual contendrá un número máximo de árboles de decisión. Una vez se supere dicho límite de árboles se irán eliminando los más antiguos.

En el Algoritmo 3.2, se muestra el pseudocódigo de cómo funcionan los *ensembles* dinámicos en nuestro proyecto.

```

1: clasefador1 ← nuevoClasificador
2: clasefador2 ← nuevoClasificador
3: semanaClasificador2 ← 10
4: para j desde semana1 hasta semanan hacer
5:   Entrenar hasta j-1 con clasefador1
6:   Predecir semana j con clasefador1
7:   Guardar la tasa de acierto del clasefador1 como tasa1
8:   si j > semanaClasificador2 entonces
9:     Entrenar hasta j-1 con clasefador2
10:    Predecir semana j con clasefador2
11:    Guardar la tasa de acierto del clasefador2 como tasa2
12:    si tasa2 > tasa1 entonces
13:      contador ← contador + 1
14:    si no
15:      contador ← 0
16:    si contador = 3 entonces
17:      clasefador1 ← clasefador2
18:      clasefador2 ← nuevoClasificador
19:      contador ← 0
20:      tasa1 ← tasa2

```

Algoritmo 3.1: Pseudocódigo de estrategias de olvido

```

1: maxTrees  $\leftarrow 15$ 
2: ensemble  $\leftarrow []$ 
3: para j desde semana1 hasta semanan hacer
4:   clasificador  $\leftarrow$  nuevoClasificador
5:   Entrenar hasta j-1 con clasificador
6:   Agregamos clasificador a ensemble
7:   longitud  $\leftarrow$  Longitud de ensemble
8:   si longitud  $>$  maxTrees entonces
      Eliminar el elemento más antiguo de la lista ensemble
10:  Predecir mediante votación por mayoría la semana j, con los clasificadores de la lista
ensemble

```

Algoritmo 3.2: Pseudocódigo de *ensembles* dinámicos

Este tipo de modelos son muy útiles de realizar cuando tenemos situaciones que requieren de adaptabilidad en el tiempo, como es el caso de nuestro problema donde, generalmente, los datos más cercanos en el tiempo proporcionan más información que los más antiguos.

Hoeffding Trees

A continuación se van a especificar los diferentes hiperparámetros que vamos a modificar para la posterior realización de los correspondientes modelos.

- **Criterio de división (*split_criterion*)**. Se refiere al criterio de división en cada nodo. El valor por defecto es '*info_gain*', pero en nuestro caso probaremos con '*gini*', '*info_gain*' y '*hellinger*'.
- **Umbral de empate de la mejor división (*tie_threshold*)**. Indica el umbral por debajo del cual una división se verá obligada a desempatar. El valor predeterminado es 0.05, mientras que en los siguientes modelos se probará con 0.05, 0.03 y 0.01.
- **Predicción de nodos hoja (*leaf_prediction*)**. Indica el método utilizado para predecir las etiquetas en los nodos hoja y, donde el valor predeterminado será '*nba*' (*Naive Bayes Adaptive*), pero en nuestros modelos consideraremos los valores '*mc*' (*Majority Class*), '*nb*' (*Naive Bayes*) y '*nba*' (*Naive Bayes Adaptive*).

Con los anteriores hiperparámetros podremos obtener un conjunto de 27 combinaciones posibles, de los cuales comentaremos los resultados posteriormente.

En el Código 3.1 podemos observar un ejemplo de código de cómo construir un *Hoeffding Tree*, el cual lo proporciona la propia documentación oficial de la biblioteca de *scikit-multiflow*. Tal y como podemos ver, se configura el clasificador correspondiente y se van leyendo las muestras a partir de un *data stream*, el cual va a hacer que el tiempo de ejecución se reduzca considerablemente, en comparación a si utilizásemos un archivo .csv. A su vez, podemos observar un bucle, el cual tiene que cumplir la condición de que el número de muestras procesadas sea menor al número máximo de muestras y haya más muestras disponibles en el flujo de datos. También es importante destacar el método *partial_fit*, el cual va a ir actualizando el modelo correspondiente de forma incremental.

```
# Imports
from skmultiflow.data import SEAGenerator
from skmultiflow.trees import HoeffdingTreeClassifier

# Setting up a data stream
stream = SEAGenerator(random_state=1)

# Setup Hoeffding Tree estimator
ht = HoeffdingTreeClassifier()

# Setup variables to control loop and track performance
n_samples = 0
correct_cnt = 0
max_samples = 200

# Train the estimator with the samples provided by the data stream
while n_samples < max_samples and stream.has_more_samples():
    X, y = stream.next_sample()
    y_pred = ht.predict(X)
    if y[0] == y_pred[0]:
        correct_cnt += 1
    ht = ht.partial_fit(X, y)
    n_samples += 1

# Display results
print('{} samples analyzed.'.format(n_samples))
print('Hoeffding Tree accuracy: {}'.format(correct_cnt / n_samples))
```

Código 3.1: Ejemplo de código de *Hoeffding Trees* en Python

Resultados de los modelos usando *Hoeffding Trees*

A continuación, se van a comentar los mejores 10 modelos obtenidos a partir de las combinaciones realizadas. Se van a ordenar de mayor a menor por *score* obtenido, en caso de empate el que mejor tiempo de ejecución ofrezca.

Tal y como podemos visualizar en la tabla 3.8, los modelos que ofrecen mejor rendimiento son aquellos que tienen '*nba*' (*Naive Bayes Adaptive*) como valor para la variable *leaf_prediction* y, a su vez, podemos percatarnos de que los modelos con los valores 0.03 y 0.05 para la variable *tie_threshold* ofrecen una precisión mayor que los que usan 0.01.

Uno de los puntos más importantes a mencionar es el bajo tiempo de ejecución que estas técnicas necesitan para entrenar, ya que, como hemos visto en anteriores modelos, se necesitaban tiempos de muchos minutos para entrenar y validar, mientras que, estas técnicas lo consigue en cuestión de segundos. Esto se debe a su capacidad de aprender incrementalmente, tal y como se comentó anteriormente.

Aunque estos modelos sean un poco menos precisos en comparación a algunos vistos anteriormente, es importante tener en cuenta su bajo tiempo de ejecución con respecto a los tiempos que ofrecen los modelos obtenidos hasta el momento.

<i>split_criterion</i>	<i>tie_threshold</i>	<i>leaf_prediction</i>	<i>score</i>	Tiempo
<i>gini</i>	0.05	<i>nba</i>	0.6281	15.57s
<i>info_gain</i>	0.05	<i>nba</i>	0.6279	16.73s
<i>gini</i>	0.03	<i>nba</i>	0.6267	15.23s
<i>info_gain</i>	0.03	<i>nba</i>	0.6266	15.48s
<i>hellinger</i>	0.03	<i>nba</i>	0.6228	14.93s
<i>hellinger</i>	0.01	<i>nba</i>	0.6228	14.96s
<i>hellinger</i>	0.05	<i>nba</i>	0.6206	15.49s
<i>gini</i>	0.01	<i>nba</i>	0.6136	18.45s
<i>info_gain</i>	0.01	<i>nba</i>	0.6136	18.87s
<i>hellinger</i>	0.01	<i>mc</i>	0.6124	6.70s

Tabla 3.8: Resultados *Hoeffding trees*

Árboles de decisión con técnicas de olvido

En la Sección 2.3.1, se detalló en qué consistía el funcionamiento de los árboles de decisión por lo que, en esta sección solo se citarán los hiperparámetros que se van a configurar y qué resultados se han obtenido para poder comparar con respecto a la tasa de acierto y el tiempo de los modelos estáticos.

Los hiperparámetros configurados en este tipo de modelos serán los mismos que los escogidos en la anterior sección, en los modelos estáticos. Los cuales son los siguientes.

- **Criterio de división (criterion).** Se utilizarán los valores de "gini" y "entropy".
- **Máximo número de nodos hoja (max_leaf_nodes).** En este tipo de modelos se decidió usar los valores 2, 30, 60 y 100.
- **Profundidad máxima (max_depth).** Los valores que se emplean son 25, 50 y "None".

Al igual que en los modelos estáticos, obtenemos 24 árboles diferentes y, también es importante destacar que, para estos modelos se va a utilizar el mismo módulo y clasificador que el utilizado en los métodos estáticos.

Resultados de los modelos usando árboles de decisión con técnicas de olvido

En la Tabla 3.9 podemos observar los resultados de los 8 mejores árboles obtenidos utilizando técnicas de olvido con árboles de decisión, en función de su precisión y tiempo de ejecución.

De la misma manera que en los modelos estáticos, los modelos que ofrecen mejores resultados en función del *score* obtenido son los que tienen un valor de 30 para la variable *max_leaf_nodes*.

Si comparamos con los modelos estáticos, podemos observar que obtenemos una mejor tasa de acierto utilizando modelos dinámicos, llegando a obtener, en muchos casos, un aumento de 0.02, pero, sin embargo, el tiempo de ejecución para obtener dichos resultados es mayor, llegando a ser el doble o incluso el triple en algunos casos.

Tal y como se realizó en la sección de los modelos estáticos, hemos representado un árbol de decisión para observar qué características escoge antes. En este caso se ha querido representado el árbol que mejor rendimiento ofrece, en términos de tasa de acierto, el cual es el que tiene como criterio de división entropía, como máximo número de nodos hoja el valor 30 y 25 como máxima profundidad. La representación de este árbol de decisión con técnicas de olvido lo podemos visualizar en la Figura 3.16.

<i>criterion</i>	<i>max_leaf_nodes</i>	<i>max_depth</i>	<i>score</i>	Tiempo
<i>entropy</i>	30	25	0.6578	515.47s
<i>entropy</i>	30	<i>None</i>	0.6577	505.41s
<i>entropy</i>	30	50	0.6576	514.45s
<i>gini</i>	30	<i>None</i>	0.6513	481.56s
<i>gini</i>	30	25	0.6510	481.70s
<i>gini</i>	30	50	0.6509	481.14s
<i>entropy</i>	60	25	0.6508	531.47s
<i>entropy</i>	60	<i>None</i>	0.6505	532.61s

Tabla 3.9: Resultados árboles de decisión con técnicas de olvido

De igual manera que en los modelos estáticos, el árbol tiende a escoger en muchas ocasiones las variables que registran el porcentaje de victorias de los equipos, pero, sin embargo, también escoge otras variables en los niveles más cercanos al nodo raíz, como por ejemplo, los jugadores con mejor valoración por partido o los partidos perdidos por el equipo visitante. Asimismo, es importante resaltar el hecho de que aparecen otras variables como puede ser la que indica el porcentaje de triples del equipo local, esta variable puede llegar a ser importante, debido a que, actualmente el número de triples por partido por parte de los equipos ha aumentado y, anotar un alto porcentaje de triples puede llegar a ser significativo a la hora de ganar un partido.

Tal y como se ha comentado anteriormente, las predicciones realizadas se hacen sobre las semanas, es decir, se entrena sobre un conjunto de semanas y se predice la semana que corresponda, sin embargo, en esta sección se decidió a utilizar el mejor árbol de decisión obtenido con técnicas de olvido y predecir por partidos. Esta manera de predecir se llevó a cabo para saber hasta cuánto era capaz el algoritmo de predecir correctamente. No obstante, se asume que el tiempo de ejecución del algoritmo iba a ser muy superior al obtenido cuando se ha predicho con semanas. Los resultados obtenidos fueron los que se muestran en la Tabla 3.10.

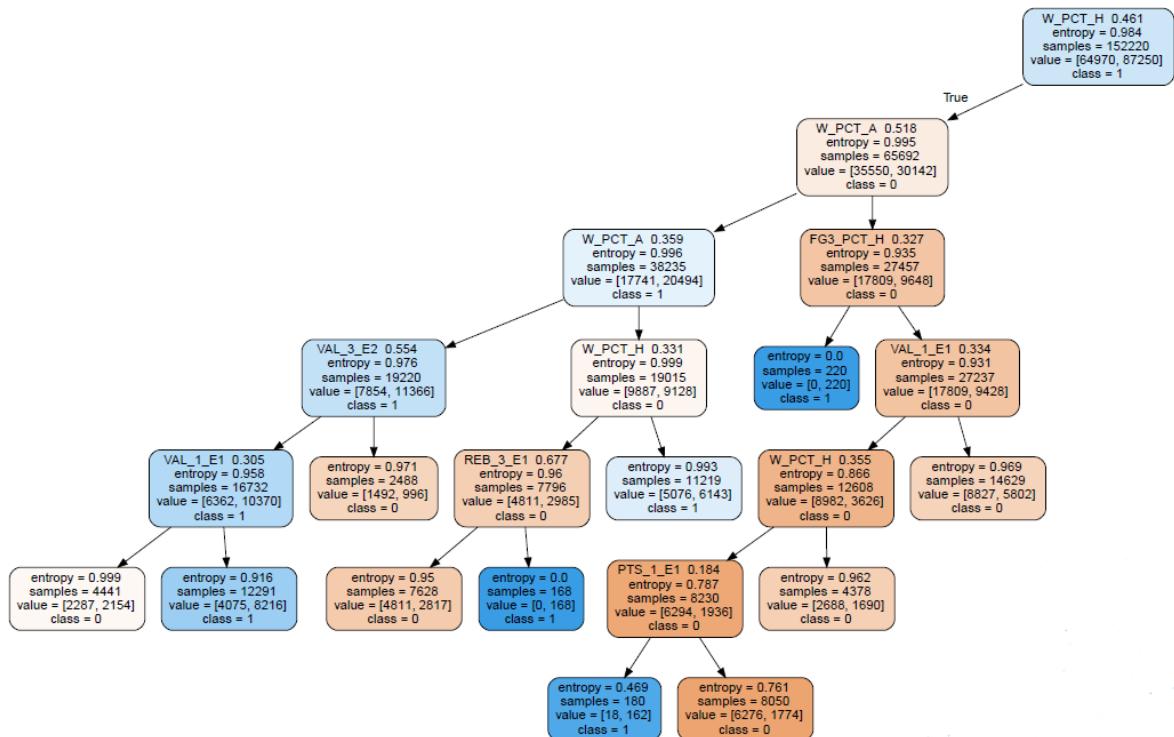
Como se puede observar, se ha conseguido aumentar hasta en un 8% el *score* obtenido por el mejor árbol de decisión, prediciendo por semanas, pero, sin embargo, el tiempo de ejecución ha aumentado bastante, ya que, en el anterior modelo, el tiempo de ejecución es de 11 minutos aproximadamente, mientras que, prediciendo por partidos, el tiempo de ejecución es de 1 hora y 30 minutos aproximadamente. Por lo que, se puede apreciar que el algoritmo predice de manera más efectiva, pero en un tiempo altamente superior.

<i>criterion</i>	<i>max_leaf_nodes</i>	<i>max_depth</i>	<i>score</i>	Tiempo
<i>entropy</i>	30	25	0.7486	4497.1s

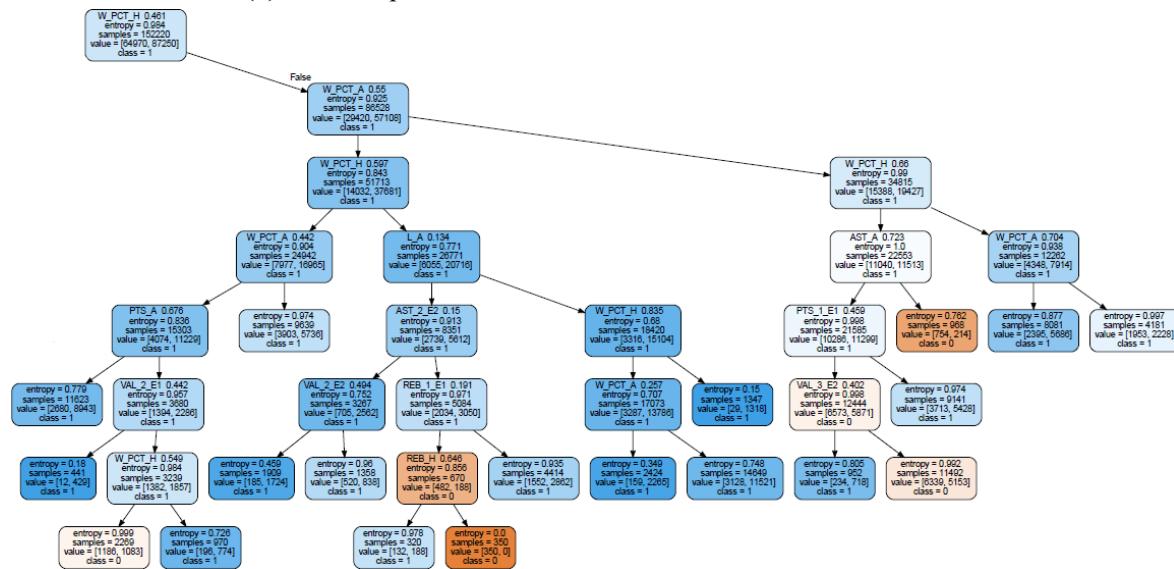
Tabla 3.10: Resultados árboles de decisión con técnicas de olvido prediciendo por partidos

De la misma manera que con el mejor árbol de decisión anterior, se decidió visualizar el correspondiente árbol de decisión con técnicas de olvido, pero prediciendo por partidos. Dicho árbol lo podemos observar en la Figura 3.17, el cual escoge en las profundidades más cercanas a la raíz, las variables que almacenan el porcentaje de victorias de los equipos, de la misma manera que en el anterior modelo con árboles de decisión con técnicas de olvido prediciendo por semanas. Sin embargo, se puede visualizar como en la rama izquierda del árbol se tienen muchos menos nodos hojas y la rama derecha es más grande, donde se pueden ver algunas variables como las asistencias del mejor jugador del equipo local, las asistencias del segundo mejor jugador del equipo visitante o los puntos del segundo mejor jugador del equipo local entre muchas otras variables.

Por lo tanto, para finalizar esta sección, se puede concluir que entrenar y predecir partido a partido es más preciso pero a la vez más costoso en términos de tiempo de ejecución en comparación con entrenar y predecir por semanas.

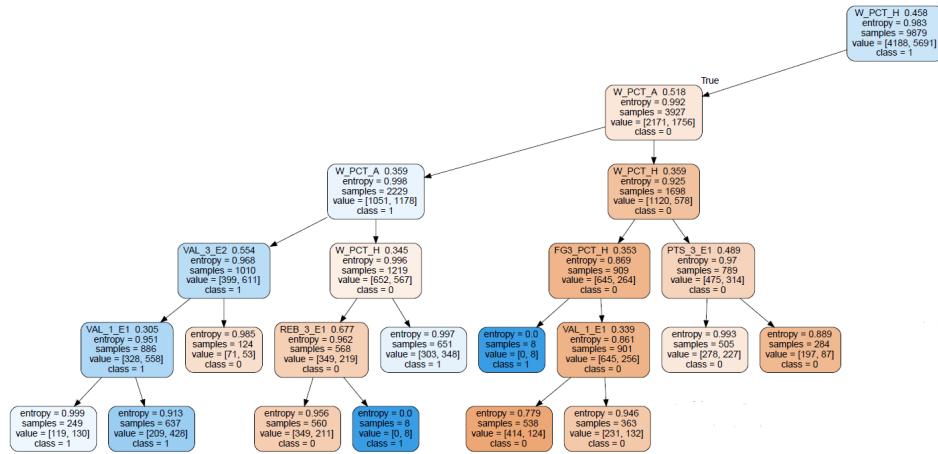


(a) Rama izquierda árbol de decisión con técnicas de olvido

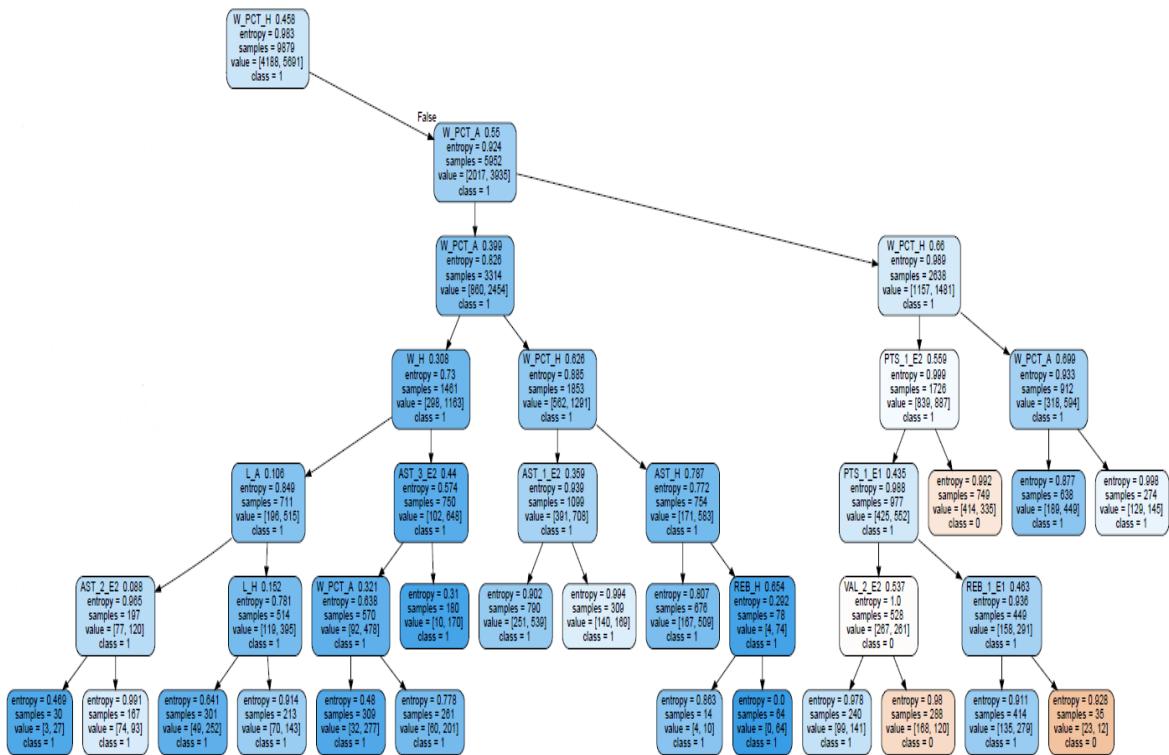


(b) Rama derecha árbol de decisión con técnicas de olvido

Figura 3.16: Árbol de decisión con técnicas de olvido



(a) Rama izquierda del árbol de decisión con técnicas de olvido prediciendo por partidos



(b) Rama derecha del árbol de decisión con técnicas de olvido prediciendo por partidos

Figura 3.17: Árbol de decisión con técnicas de olvido prediciendo por partidos

Bagging con técnicas de olvido

Bagging es un *ensemble*, el cual ya se define en la Sección 2.3.4, por lo tanto, durante esta sección, al igual que en la anterior, se van a citar los hiperparámetros configurados y vamos a comparar con los modelos estáticos los resultados obtenidos.

A continuación, se van a indicar los hiperparámetros escogidos para la realización posterior de los correspondientes modelos. Estos hiperparámetros serán los mismos que en el caso de los modelos estáticos para, así, poder realizar una comparativa entre ambos.

- **Estimador base (*base_estimator*)**. Este hiperparámetro será un estimador base. Estos estimadores tendrán a su vez, los siguientes hiperparámetros.
 - **Criterio de división (*criterion*)**. El cual será "*gini*".
 - **Máximo número de nodos hoja (*max_leaf_nodes*)**. Se consideró que sería 30.
 - **Profundidad máxima (*max_depth*)**. La profundidad máxima será 25, 50 o None.
- **Número de estimadores (*n_estimators*)**. En nuestros modelos probaremos con 20 y 50.

Como tenemos los mismos hiperparámetros que en los modelos estáticos, se van a obtener el mismo número de combinaciones posibles, es decir, 6 combinaciones posibles, de los cuales vamos a comentar sus resultados a continuación.

Resultados de modelos utilizando *bagging (Bootstrap aggregating)* con técnicas de olvido

Seguidamente, se van a indicar los resultados de los modelos realizados utilizando *bagging* con técnicas de olvido, los cuales podemos observar en la Tabla 3.11.

En los modelos estáticos se pudo observar que los modelos que ofrecían mejor rendimiento eran, generalmente, los que tenían mayor número de estimadores, no obstante, en estos modelos eso es algo que no está presente, ya que, tal y como podemos ver en la anterior tabla, dos de los mejores tres modelos que se obtienen, son aquellos que tienen 20 estimadores.

Al utilizar modelos con técnicas de olvido observamos que obtenemos modelos un poco más precisos, pero, a pesar de ello, el tiempo de ejecución suele ser mayor, como por ejemplo, un clasificador *bagging* con criterio '*gini*', 30 como número máximo de nodos hoja, 25 de profundidad y 20 estimadores base, en modelos estáticos conlleva un tiempo de ejecución de 864.32 segundos, mientras que en los modelos dinámicos con técnicas de olvido es de 2839.13 segundos, es decir, 3 veces más. Por eso, es muy importante no solo tener presente la exactitud con la que predice el algoritmo sino también cuánto tiempo conlleva entrenar y predecir dicho modelo.

<i>base_estimator</i>			<i>n_estimators</i>	<i>score</i>	Tiempo
<i>criterion</i>	<i>max_leaf_nodes</i>	<i>max_depth</i>			
<i>gini</i>	30	<i>None</i>	20	0.6575	2762.19s
<i>gini</i>	30	<i>None</i>	50	0.6567	6740.00s
<i>gini</i>	30	50	20	0.6561	2881.24s
<i>gini</i>	30	25	50	0.6561	6585.29s
<i>gini</i>	30	25	20	0.6557	2839.13s
<i>gini</i>	30	50	50	0.6557	2839.13s

Tabla 3.11: Resultados *bagging* con técnicas de olvido

Random forest con técnicas de olvido

Random forest es un algoritmo que se detalló y estudió en la Sección 2.3.5, por lo que, a lo largo de esta sección se va a comentar los hiperparámetros correspondientes y se va a comentar similitudes y diferencias de resultados respecto a los modelos estáticos.

Seguidamente, se van a citar los hiperparámetros que se configuran para la realización de los posteriores modelos, los cuales van a ser los mismos que en el caso de los modelos estáticos y, así, poder realizar una comparación más precisa. Dichos parámetros son los que indican a continuación.

- **Criterio de división (*criterion*)**. Se escogen los valores de "*gini*" y "*entropy*".
- **Máximo número de nodos hoja (*max_leaf_nodes*)**. Utilizaremos, al igual que en anteriores modelos, el valor de 30 para esta variable.
- **Número máximo de características (*max_features*)**. Se tomarán los valores de "*sqrt*" y "*log2*".
- **Número de estimadores (*n_estimators*)**. Se prueba con 20, 50 y 100 estimadores.

Por lo tanto, obtenemos un conjunto de 12 combinaciones posibles para la realización de los posteriores modelos.

Resultados de modelos utilizando *random forest* con técnicas de olvido

En la Tabla 3.12 podemos visualizar los resultados obtenidos a la hora de realizar los modelos utilizando *random forest* con técnicas de olvido.

Podemos observar que los mejores resultados son ofrecidos por modelos cuyo valor es *sqrt* para la variable *max_features*. Además, es relevante resaltar que se obtiene un *score* ligeramente mayor que si utilizásemos modelos estáticos, pero, de igual manera, podemos observar unos tiempos de ejecución muy altos comparados con los modelos estáticos.

criterion	max_leaf_nodes	max_features	n_estimators	score	Tiempo
<i>gini</i>	30	<i>sqrt</i>	20	0.6531	688.59s
<i>gini</i>	30	<i>sqrt</i>	100	0.6513	2170.76s
<i>entropy</i>	30	<i>sqrt</i>	50	0.6505	1369.13s
<i>entropy</i>	30	<i>log2</i>	20	0.6491	628.34s
<i>entropy</i>	30	<i>sqrt</i>	20	0.6490	685.92s
<i>gini</i>	30	<i>log2</i>	50	0.6483	1078.03s
<i>entropy</i>	30	<i>sqrt</i>	100	0.6482	2451.19s
<i>entropy</i>	30	<i>log2</i>	100	0.6477	2072.51s

Tabla 3.12: Resultados *random forest* con técnicas de olvido

Adaptative boosting con técnicas de olvido

Adaptative boosting es un *ensemble*, el cual fue descrito en detalle en la Sección 2.3.6, por lo que, esta sección, al igual que los anteriores modelos dinámicos, se centrará en indicar los hiperparámetros utilizados y de realizar un análisis comparativo entre este modelo y su correspondiente modelo estático.

Los hiperparámetros propios de este modelo dinámico utilizando *Adaptative boosting* con técnicas de olvido son los citados a continuación.

- **Estimador base (*base_estimator*)**. Hiperparámetros de los estimadores base:
 - **Criterio de división (*criterion*)**. El cual será "*gini*".
 - **Máximo número de nodos hoja (*max_leaf_nodes*)**. Se consideró que sería 30.
 - **Profundidad máxima (*max_depth*)**. La profundidad máxima será 25, 50 o *None*.
- **Número de estimadores (*n_estimators*)**. Probaremos con 20 y 50 estimadores base.

Se obtendrán, de la misma manera que el modelo estático correspondiente, un conjunto de 12 combinaciones posibles.

Resultados de modelos utilizando *adaptative boosting* con técnicas de olvido

A continuación, se van a mostrar los 8 mejores modelos obtenidos, empleando *adaptative boosting* con técnicas de olvido.

Como se puede apreciar en la Tabla 3.13, al igual que en los modelos estáticos, aquellos modelos que tienen 1 como máxima profundidad obtienen mejores resultados que los que utilizan 2 o 3, además de predecir con mayor exactitud, también lo consiguen en menos tiempo. Asimismo, es fundamental comentar otra similitud respecto de los modelos estáticos, la cual es que obtienen mejor tasa de acierto los modelos que tienen mayor número de estimadores base.

Sin embargo, también se pueden visualizar algunas distinciones respecto a los modelos estáticos. Una de las principales que el tiempo que tardan en entrenar y predecir es ampliamente mayor en los modelos dinámicos, sin embargo, se consigue mejorar ligeramente la exactitud con la que los modelos predicen.

<i>base_estimator</i>		<i>learning_rate</i>	<i>n_estimators</i>	<i>score</i>	Tiempo
<i>max_leaf_nodes</i>	<i>max_depth</i>				
30	1	0.95	50	0.6591	2558.06s
30	1	1.0	50	0.6584	2344.75s
30	2	0.95	20	0.6528	1759.71s
30	1	1.0	20	0.6516	1120.60s
30	1	0.95	20	0.6512	1189.85s
30	2	1.0	20	0.6507	1746.32s
30	3	0.95	20	0.6506	2369.84s
30	3	1.0	20	0.6488	2363.55s

Tabla 3.13: Resultados *adaptative boosting* con técnicas de olvido

Gradient boosting con técnicas de olvido

Gradient boosting es una técnica de aprendizaje automático que vimos en detalle a lo largo de la Sección 2.3.7, por lo que, a lo largo de esta sección se citarán los hiperparámetros que se configuran y se estudiarán detalladamente los resultados obtenidos.

Seguidamente, se van a comentar los hiperparámetros que se van a seleccionar para la posterior realización de los modelos. Además, estos hiperparámetros serán los mismos que los utilizados en los modelos estáticos para así poder realizar un estudio comparativo justo.

- **Número de estimadores (*n_estimators*)**. En nuestro caso se prueba con 20 y 50 estimadores.
- **Profundidad máxima (*max_depth*)**. Se utilizará una profundidad máxima de 1, 2 y 3.
- **Tasa de aprendizaje (*learning_rate*)**. Se aplicarán los valores de 0.05 y 0.1.

Utilizando estos valores obtendremos un conjunto de 12 combinaciones posibles, de las cuales se van a mencionar cuáles ofrecen mejor rendimiento.

Resultados de modelos utilizando *gradient boosting* con técnicas de olvido

A continuación, en la Tabla 3.14 se van a mostrar los 8 mejores modelos obtenidos, teniendo en cuenta el *score* conseguido, de los 12 que se realizaron.

Tal y como podemos observar los modelos que ofrecen un mejor rendimiento son aquellos donde, tanto la máxima profundidad como el número de estimadores tienen un valor alto, ya que, se puede apreciar como los modelos donde la máxima profundidad es 1 y el número de estimadores base es 20 ofrecen una precisión menor, pero, a su vez, un tiempo de ejecución más bajo.

Observando los resultados que ofrece *gradient boosting* podemos realizando una comparativa justa debido a que hemos seleccionado los mismos hiperparámetros en ambos modelos.

Se puede apreciar como utilizando técnicas de olvido conseguimos mejor precisión respecto a los modelos estáticos, pero el tiempo que conlleva entrenar y validar es mucho mayor.

Es importante resaltar el hecho de que los mejores modelos obtenidos en ambos tipos de algoritmos han sido obtenidos con valores muy similares para los mismos hiperparámetros, por lo que se puede comprender de que el valor de esos hiperparámetros son significativos a la hora de realizar un modelo con una tasa de acierto alta.

<i>max_depth</i>	<i>learning_rate</i>	<i>n_estimators</i>	<i>score</i>	Tiempo
3	0.1	50	0.6597	4869.00s
2	0.1	50	0.6562	3425.16s
3	0.03	50	0.6544	4899.03s
3	0.1	20	0.6533	2148.41s
2	0.1	20	0.6524	1552.68s
2	0.05	50	0.6521	3440.76s
1	0.1	50	0.6472	1878.68s
3	0.05	20	0.6407	2108.55s

Tabla 3.14: Resultados *gradient boosting* con técnicas de olvido

***Histogram gradient boosting* con técnicas de olvido**

Al igual que en las anteriores secciones donde se han comentado algoritmos utilizando técnicas de olvido, se van a citar los hiperparámetros correspondientes y los resultados conseguidos por los mejores modelos. El algoritmo descrito y explicado en detalle se puede encontrar en la Sección 2.3.8.

A continuación, se van a comentar cuáles son los hiperparámetros que se configuraran seguidamente en la realización de los modelos con técnicas de olvido. Estos hiperparámetros son los mismos que se utilizaron en los modelos estáticos para así poder hacer un análisis comparativo de manera justa.

- **Número máximo de iteraciones (*max_iter*)**. Utilizaremos los valores 20 y 50.
- **Tasa de aprendizaje (*learning_rate*)**. Se indicarán los valores de 0.01, 0.03 y 0.05.
- **Máximo número de nodos hoja (*max_leaf_nodes*)**. Se emplean los valores 30 y 60.

Con estos valores obtenemos un conjunto de 12 posibles modelos a realizar y, de los cuales mostraremos posteriormente sus tasas de acierto y tiempos de ejecución conseguidos.

Resultados de modelos utilizando *Histogram gradient boosting* con técnicas de olvido

A continuación, se van a mostrar los resultados obtenidos de los 8 mejores modelos ordenados de mayor a menor por *score* obtenido. Dichos resultados los podremos visualizar en la Tabla 3.15.

Visualizando los resultados que ofrece *histogram gradient boosting* con y sin técnicas de olvido, podemos ver claramente que ambos tipo de modelos se obtienen mejores tasas de acierto cuando el número de iteraciones y la tasa de aprendizaje tienen unos valores de 50, y 0.03 y 0.05 respectivamente.

Además, de la misma manera que en las anteriores comparaciones entre estos tipos de modelos, los modelos con técnicas de olvido producen mejores resultados con respecto a la tasa de acierto, pero el tiempo de ejecución es mucho mayor.

<i>max_leaf_nodes</i>	<i>learning_rate</i>	<i>max_iter</i>	<i>score</i>	Tiempo
30	0.05	50	0.6580	933.99s
60	0.05	50	0.6543	1345.24s
30	0.03	50	0.6542	880.47s
60	0.03	50	0.6530	1357.82s
30	0.05	20	0.6505	619.33s
60	0.05	20	0.6449	776.25s
60	0.03	20	0.6365	776.62s
30	0.03	20	0.6349	580.69s

Tabla 3.15: Resultados *histogram gradient boosting* con técnicas de olvido

Ensembles dinámicos

En este tipo de *ensembles* se utilizarán árboles de decisión para formar el *ensemble* correspondiente. Estos árboles de decisión se construyen, al igual que se ha hecho en las anteriores secciones, configurando una serie de hiperparámetros, los cuales son los que se citan a continuación.

- **Criterio de división (*criterion*)**. En el cual se consideran los valores de "gini" y "entropy".
- **Máximo número de nodos hoja (*max_leaf_nodes*)**. Se consideró que sería 2, 30, 60 y 100.
- **Número máximo de características (*max_features*)**. Se tomarán los valores de "sqrt", "log2" y "auto".

Con estos hiperparámetros se forman un conjunto de 24 modelos diferentes, de los cuales se va a comentar los resultados que ofrecen a continuación.

Resultados de modelos utilizando *ensembles* dinámicos

Seguidamente, se van a mostrar los resultados de los 5 mejores modelos obtenidos utilizando este tipo de modelo en la Tabla 3.16.

Tal y como se puede apreciar en la tabla correspondiente, los modelos que mejor precisión ofrecen son aquellos que mayor valor tienen en la variable *max_leaf_nodes*. Este tipo de modelos ofrecen mejor *score* que aquellos que tienen, por ejemplo, un valor de 2 y, además, el tiempo de ejecución es muy similar en todos los modelos realizados.

criterion	max_leaf_nodes	max_features	score	Tiempo
<i>entropy</i>	100	<i>sqrt</i>	0.6392	205.56s
<i>gini</i>	60	<i>sqrt</i>	0.6390	202.52s
<i>gini</i>	100	<i>auto</i>	0.6385	201.66s
<i>gini</i>	100	<i>sqrt</i>	0.6383	201.17s
<i>gini</i>	60	<i>auto</i>	0.6381	200.55s

Tabla 3.16: Resultados de *ensemble* dinámico

Combinación de clasificadores con técnicas de olvido.

A lo largo de esta sección se va a detallar el proceso que se lleva a cabo para diseñar un nuevo modelo, el cual consiste en utilizar técnicas de olvido, pero con tres clasificadores, los cuales antes de realizar cada predicción harán votación por mayoría. El proceso es el mismo que se realizaba anteriormente con técnicas de olvido, solo que en vez de un solo tipo de clasificador en este algoritmo se tendrán tres tipos de clasificadores.

Al igual que en los anteriores modelos se realizarán las predicciones por semanas, además los tres tipos de clasificadores elegidos fueron un árbol de decisión, un *histogram gradient boosting* y un *random forest*.

Los hiperparámetros elegidos para estos clasificadores serán aquellos con lo que mejor precisión se obtenía en sus correspondientes secciones. En el Código 3.2 se puede observar cómo se han declarado los 3 tipos de clasificadores.

```
DecisionTreeClassifier(criterion='entropy', max_leaf_nodes=30,
max_depth=25)
HistGradientBoostingClassifier(learning_rate=0.05, max_iter=50,
max_leaf_nodes=30)
RandomForestClassifier(n_estimators=20, criterion='gini',
max_features='sqrt', max_leaf_nodes=30)
```

Código 3.2: Ejemplo de código de los clasificadores para modelo de combinación de clasificadores con técnicas de olvido en Python

Una vez declarados los 3 tipos de clasificadores con los que se va a construir el correspondiente modelo, es momento de comentar los resultados.

Resultado del modelo utilizando la combinación de clasificadores con técnicas de olvido

A continuación, se va a mostrar y comentar el resultado obtenido utilizando la combinación de clasificadores con técnicas de olvido. Dicho resultado lo podemos contemplar en la Tabla 3.17.

Tal y como se puede observar, se obtienen resultados ligeramente mejores que si se utilizase únicamente un solo clasificador, sin embargo, el tiempo de ejecución de entrenamiento y predicción es altamente mayor que empleando un solo clasificador.

<i>Clasificador1</i>	<i>Clasificador2</i>	<i>Clasificador3</i>	<i>score</i>	<i>Tiempo</i>
Árbol de decisión	<i>Histogram gradient boosting</i>	<i>Random forest</i>	0.6566	2813.15s

Tabla 3.17: Resultados de modelo con combinación de clasificadores con técnicas de olvido

3.4.4. Construcción y validación del modelo final

En esta sección se obtendrá una comparativa indicando el *accuracy* y tiempo del mejor modelo seleccionado para cada algoritmo. Estos modelos se obtendrán prediciendo sobre un conjunto diferente al que se haya usado para la anterior selección de los hiperparámetros, el cual es en nuestro problema, las dos últimas temporadas. Los resultados de dichos modelos se pueden observar en la tabla 3.18, de manera ordenada por *accuracy* obtenido. Mientras no se especifique nada en la columna de estrategia de predicción, la predicción se realizará siempre por semanas.

En la tabla 3.18 se tiene una columna donde se indica el algoritmo correspondiente, la estrategia de predicción para describir si es un modelo dinámico o estático, si es dinámico especificaremos si se utilizan estrategias de olvido y, su correspondiente *accuracy* y tiempo de ejecución.

Tal y como se puede comprobar, el modelo que mejor *accuracy* ofrece es en el que utilizamos un algoritmo de *gradient boosting* utilizando estrategias de olvido. Este tipo de algoritmo ofrece una precisión de 0.6280 en 353.54 segundos.

Hay que destacar, que utilizando un algoritmo prediciendo por partidos como el que se utilizaba con árboles de decisión, se obtienen mejores resultados que prediciendo por semanas, en concreto se obtiene un *accuracy* de 0.7649 en 932.87 segundos, pero no se ha decidido realizar con esta técnica las predicciones debido a que es mucho más costoso computacionalmente, por lo que, tampoco se incluirá en la tabla correspondiente y será solamente una comparativa relativa a las semanas.

Algoritmo	Estrategia de predicción	Accuracy	Tiempo
Gradient boosting	Estrategia de olvido	0.6280	353.54s
Histogram gradient boosting	Estrategia de olvido	0.6254	219.75s
Votación por mayoría con 3 clasificadores	Estrategia de olvido	0.6232	315.76s
Random forest	Modelo estático	0.6228	53.94
Bagging	Estrategia de olvido	0.6228	213.37s
Histogram gradient boosting	Modelo estático	0.6217	174.07s
Gradient boosting	Modelo estático	0.6216	371.67s
Adaptative boosting	Estrategia de olvido	0.6211	183.72s

Algoritmo	Estrategia de predicción	<i>Accuracy</i>	Tiempo
Bagging	Modelo estático	0.6188	524.36s
Adaptative boosting	Modelo estático	0.6183	191.39s
Árbol de decisión	Estrategia de olvido	0.6172	37.78s
Ensemble dinámico	Estrategia de olvido	0.6155	31.87s
Random forest	Estrategia de olvido	0.6133	57.31s
Árbol de decisión	Modelo estático	0.6117	46.62s
Redes neuronales	Modelo estático	0.6005	713.78s
Hoeffding tree	Modelo dinámico	0.5950	1.8401s

Tabla 3.18: Construcción y validación del modelo final

3.4.5. Conclusiones y futuras mejoras

Posteriormente a la realización de los anteriores modelos, se pueden extraer algunas conclusiones de los modelos obtenidos y se pueden especificar algunas proyectadas mejoras.

Primeramente, es de gran importancia destacar que la realización de un buen análisis exploratorio y su posterior preprocesamiento de datos es de gran ayuda para la obtención de unos muy buenos resultados.

Seguidamente, se tiene que comentar la significancia de entrenar y validar, en este proyecto, por semanas, ya que, si hubiésemos entrenado y validado jornada a jornada se hubiera visto un aumento en la precisión de los algoritmos, pero el tiempo de ejecución hubiese aumentado considerablemente, lo que habría conducido a la realización de menor modelos y la elección de menos modelos con un número menor de hiperparámetros.

Además, uno de los aspectos claves ha sido la realización de modelos dinámicos, como es el caso de los modelos con técnicas de olvido, ya que, debido a la obligación de entrenar y predecir de manera secuencial en el tiempo, estas técnicas se adaptan en el tiempo y consiguen una mayor precisión a la hora de predecir.

La elección de un modelo u otro depende de las necesidades el usuario, es decir, si se prefiere un modelo que entrene rápidamente a pesar de perder una pequeña cantidad de información, quizás se prefiera utilizar modelos estáticos prediciendo por semanas o el uso del clasificador *hoeffding tree*, sin embargo, si se prefiere utilizar un modelo más preciso aunque

el tiempo de ejecución fuera mayor, tal vez se prefiera el uso de modelos con técnicas de olvido y prediciendo partido a partido para evitar la menor pérdida de información posible a la hora de entrenar.

Para concluir con esta sección, se pueden tener algunas mejoras en el futuro, como puede ser la selección, aún más detallada, de las características con la que el modelo entrenará o la elección de un número mayor de hiperparámetros. Asimismo, se probará en mayor medida la predicción por partidos para poder obtener un algoritmo con la mayor predicción posible.

4. Desarrollo de la aplicación web

4.1. Introducción

Una vez se han llevado a cabo los correspondientes modelos y se ha podido concluir en cuáles son los más convenientes a utilizar, tanto por precisión como por tiempo de ejecución, se requiere realizar la última parte de la metodología CRISP-DM, la cual vimos en el Anexo A.1, también conocida como despliegue, donde desarrollaremos una aplicación web.

Por lo tanto, durante esta sección se va a detallar el diseño de la aplicación, el *framework* utilizado, la principal funcionalidad de la que se dispone y los lenguajes utilizados para su desarrollo.

4.2. Diseño de la aplicación web

La aplicación web consistirá principalmente en ofrecer al usuario una interfaz donde podrá introducir una fecha correspondiente para que se le proporcione las predicciones de la fecha introducida. La aplicación indicará, en un espacio determinado, los partidos y que dos equipos se enfrentan y, seguidamente, cuál de ambos ganará.

4.3. *Flask*

El *framework* que utilizaremos para el desarrollo de la aplicación web será *Flask*, el cual nos permite crear de una manera muy sencilla aplicaciones web con Python.

Es muy recomendado usar *Flask* debido a una serie de ventajas que proporciona como por ejemplo, el desarrollo de una aplicación web de una forma ágil. Además, incluye un servidor

web de desarrollo para ir probando los resultados y, un depurador y soporte integrado para pruebas unitarias. Asimismo, cabe destacar el hecho de que es un software de código abierto [32].

4.4. Arquitectura del proyecto

En esta sección se va a detallar tanto la estructura que contiene el proyecto, como el funcionamiento de los archivos que lleva consigo, los cuales vamos a detallar a lo largo de esta sección. Además, podemos observar la estructura del proyecto en la Figura 4.1.

Para comenzar, tal y como se puede observar en la Figura 4.1, la aplicación se compone de dos archivos de tipo Python, los cuales son:

- ***train.py***. Este es el archivo donde se pre-entrenan los correspondientes modelos y, una vez entrenados, se almacenan en la carpeta llamada ”*models*”, la cual contiene los modelos estáticos y dinámicos. Los modelos se almacenan debido a la utilización del módulo *pickle*, ya que, este módulo serializa objetos de Python y los almacena en un archivo, el cual tendrá una extensión *.pkl* [7].
- ***app.py***. Este archivo contiene el desarrollo de la aplicación, donde usaremos *Flask* y cargaremos el modelo previamente entrenado. Asimismo, se gestionarán las solicitudes web entrantes y enviar respuestas al usuario.

A su vez, tendremos los archivos ***index.html*** y ***style.css***, los cuales son el que estructura y despliega una página web y sus contenidos, y el que definirá el diseño de la página respectivamente. El archivo HTML hace que la información regrese a las vistas ya procesadas y esta devuelva la información hacia *Flask*.

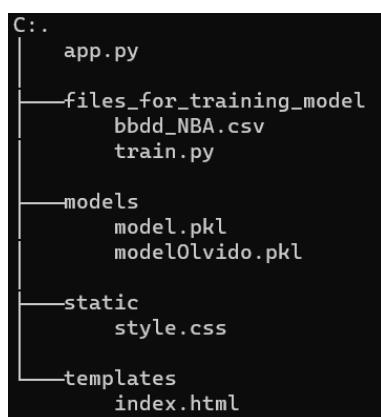


Figura 4.1: Estructura del proyecto.

4.5. Aplicación web

Una vez se ha descrito tanto la estructura del proyecto como las funcionalidades de los archivos que lo componen, se va a explicar el funcionamiento de la aplicación web.

Para comenzar, se debe comentar el hecho de que el entorno gráfico es bastante simple, donde tenemos en la parte central de la pantalla un recuadro, el cual nos indica que podemos predecir resultados de la NBA ingresando una fecha y seleccionando el tipo de modelo con el que queremos predecir. Dicha fecha tiene que ser introducida en el formato que se indica el cual es el año primero, después el mes y seguidamente el día (Año-Mes-Día) y, además, es obligatorio elegir antes de presionar el botón de predecir un tipo de modelo.

Tal y como podemos ver en la Figura 4.2, tenemos un recuadro que nos indica que podemos introducir la fecha correspondiente en el formato idóneo.

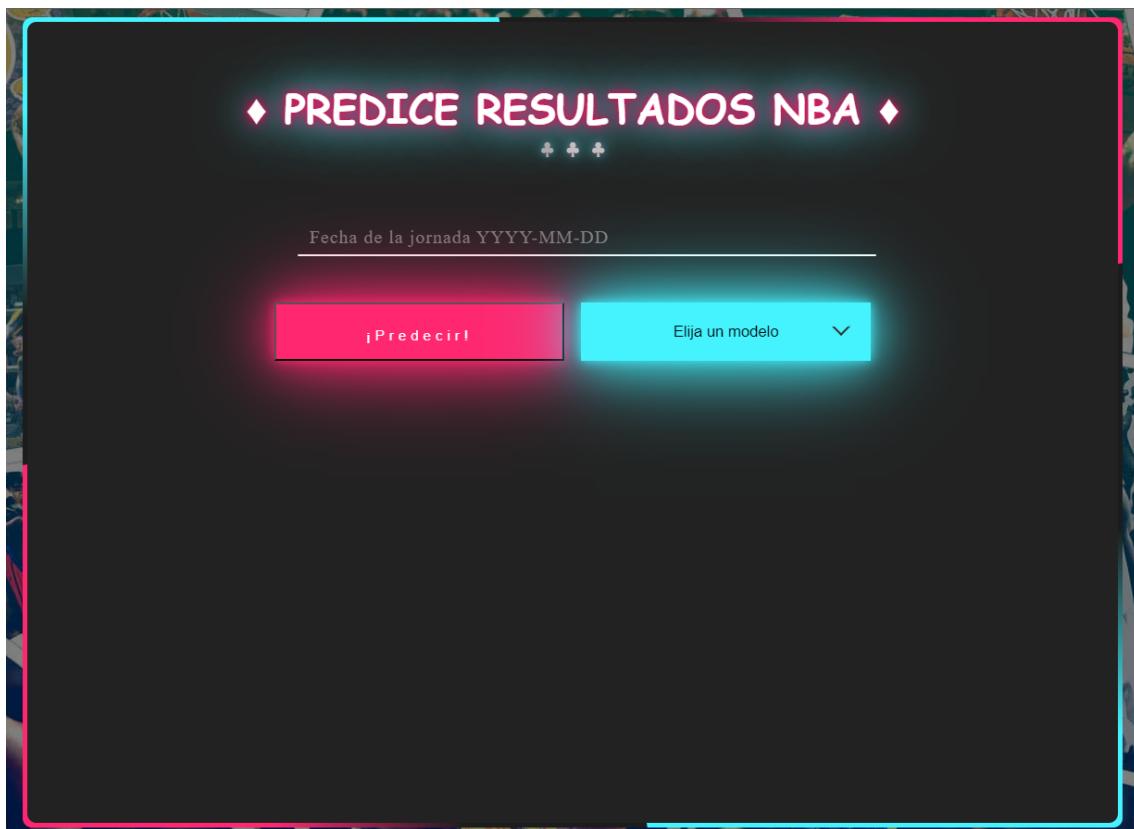


Figura 4.2: Interfaz aplicación.

Además, tal y como se ha comentado anteriormente, tenemos que presionar el botón que nos indica que elegimos un modelo, esto le indicará al programa con qué tipo de modelo quiere predecir, si con uno estático o con uno dinámico. En la Figura 4.3 se puede observar qué sucede cuando desplegamos el menú desplegable.

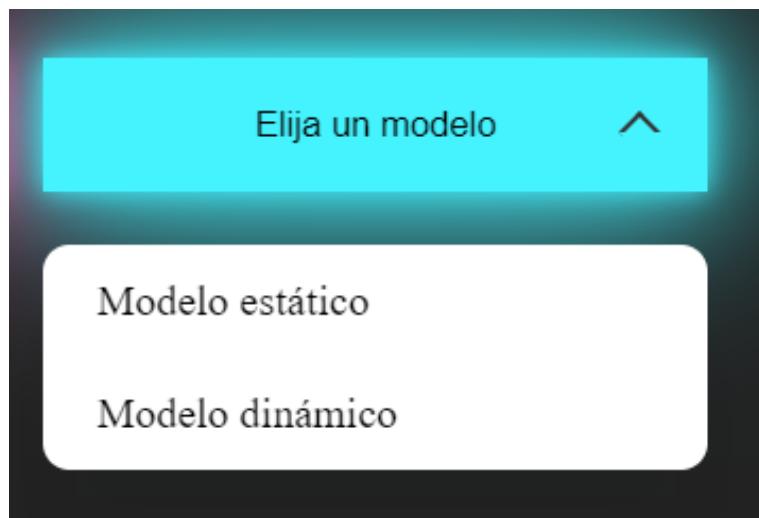


Figura 4.3: Menú desplegable para elegir un modelo.

A continuación, se van a detallar qué ocurre si no elegimos un modelo o introducimos una fecha errónea o no disponible. Debido a que la aplicación fallaba y se bloqueaba cuando no se introducía una fecha de manera correcta o no se elegía un modelo anteriormente, se propuso que en ese tipo de casos se avisara al usuario por qué estaba fallando para que lo corrigiese e introdujera de manera correcta los datos. Por lo tanto, pueden ocurrir las dos siguientes posibilidades.

1. **Introducir una fecha errónea.** En este caso, se produce un error si introducimos una fecha fuera del rango permitido para predecir, ya que, se entrena desde la temporada 2003-04 hasta la temporada 2021-22 y se predecirán los datos disponibles de la temporada 2022-23, que van desde la fecha '2022-10-18' hasta '2022-12-22'. Asimismo, se produce un fallo si se introduce una fecha en un formato diferente al que se indica en el propio programa o si se introduce cualquier otra cosa que no sea una fecha. En la Figura 4.4 podemos ver que si no cumplimos cualquiera de las anteriores condiciones, se indicará en un comentario que no hay partidos disponibles en esas fechas o que no se ha introducido la fecha de manera correcta.
2. **No seleccionar modelo.** Este fallo aparecerá si se intenta predecir sin elegir previamente un modelo de los disponibles. El desplegable correspondiente permitirá elegir entre unos tipos de modelos y antes de predecir tendremos que seleccionar uno. En la

Figura 4.5 se muestra que si no cumplimos esta condición se le indica al usuario que tiene que elegir un modelo previamente.

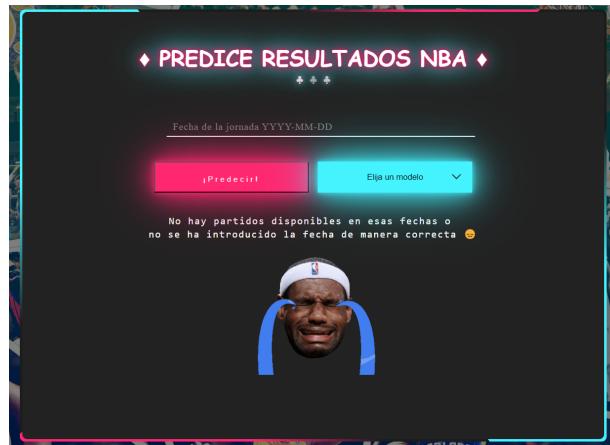


Figura 4.4: Introducción de fecha errónea en la aplicación.

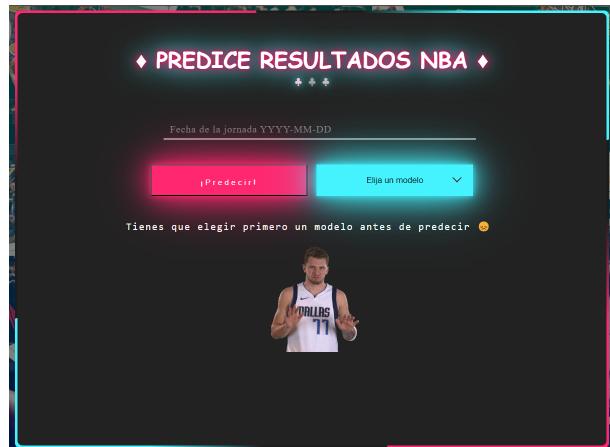


Figura 4.5: No se selecciona un modelo antes de predecir.

Una vez se hayan introducido y seleccionado los datos anteriores de manera correcta se puede presionar el botón de predecir, el cual nos mostrará los partidos correspondientes a dicha fecha con sus predicciones.

Las predicciones van acompañadas de la probabilidad con la que predice dicho resultado y, aunque esto no sería posible implementarlo para el despliegue de la aplicación, se indica

con un *tic* verde o una equis si se acertó o si se falló en la predicción respectivamente. En la Figura 4.6 se puede observar un ejemplo de cómo se muestran los resultados.

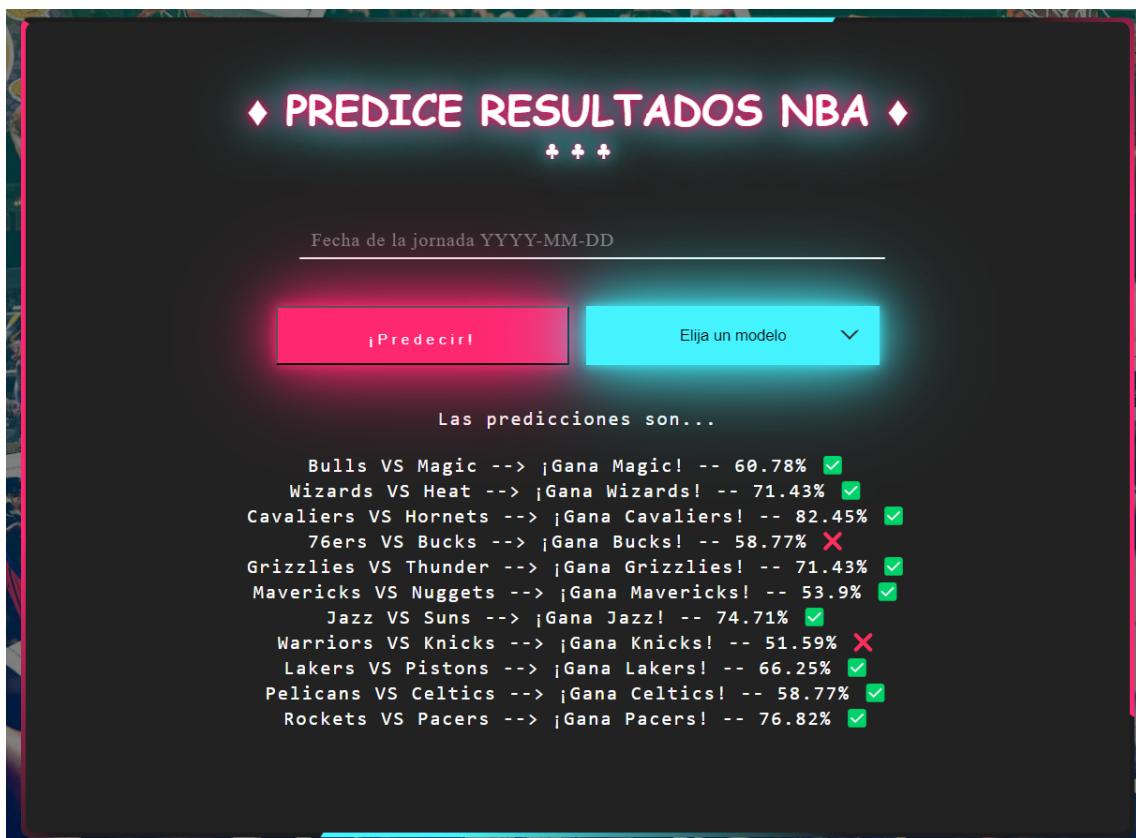


Figura 4.6: Predicciones realizadas en la aplicación.

4.6. Conclusiones y futuras mejoras

Una vez terminada la anterior sección donde se ha detallado en profundidad el funcionamiento de la aplicación web, se van a citar las correspondientes conclusiones y, a su vez, algunas mejoras que se podrán realizar en el futuro.

Como conclusiones se puede decir que se han obtenido los objetivos propuestos, donde se ha diseñado e implementado una aplicación web para poner en explotación los modelos aprendidos. La creación de dicha aplicación ha sido posible gracias al *framework* utilizado, también conocido como *Flask*.

Algunas de las posibles mejoras que se pueden implementar en el futuro pueden ser la posibilidad de predecir solamente un partido y que dichos partidos se complementen no solo con la predicción y probabilidad, sino que también con una gráfica que registre las estadísticas más importantes de los equipos. Además, se podría implementar la capacidad de elegir entre un número mayor de modelos para que así el usuario tenga más fuentes para obtener las predicciones y probabilidades.

5. Conclusiones del proyecto

5.1. Introducción

En este capítulo se van a tratar las conclusiones finales de este Trabajo Fin de Grado y, aunque anteriormente, en la Sección 3.4, ya se han detallado algunas conclusiones y futuras mejoras, en este capítulo se van a detallar en mayor medida, además de tratar otros temas y no solo las conclusiones relacionadas con los modelos.

5.2. Conclusiones

Para comenzar, se debe comentar que el objetivo de este Trabajo Fin de Grado era la realización de un buen proyecto de minería de datos, donde se realizaría la predicción de resultados de NBA utilizando técnicas de aprendizaje automático. Además, se propuso la realización de una aplicación para que en un futuro cualquier usuario la pudiese usar para realizar futuras predicciones.

Siguiendo la metodología CRIPS-DM, se realiza el proyecto de tal manera que, se obtienen distintos modelos, unos más precisos que otros y unos más rápidos que otros a la hora de entrenar. La realización de estos modelos fue interesante para realizar una comparativa entre los hiperparámetros elegidos para cada modelo y las técnicas realizadas con dichos modelos. Se realizó la elección de dos modelos, los cuales se implementarían en la aplicación pre-entrenados para poder realizar predicciones de una temporada pudiendo elegir entre ambos tipos de modelos.

Dicha aplicación se implementó para mostrar cómo de efectivos eran los modelos realizados y para enseñar cómo se puede predecir en tiempo real utilizando unos modelos que se

han entrenado previamente.

Por lo tanto, se puede concluir que, aunque la NBA sea una liga donde se suelen producir resultados muy impredecibles y asombrosos, se puede predecir con cierto porcentaje de efectividad los partidos con técnicas de aprendizaje automático.

5.3. Mejoras futuras

En esta sección se van a tratar algunas de las posibles mejoras del proyecto, desde la creación de la base de datos hasta la realización de la aplicación web, pasando por la creación de los modelos.

Para comenzar, se podría realizar la creación de algunas más variables, como pueden ser el récord de los últimos 10 partidos, el récord contra el equipo rival, etc.

Seguidamente, a la hora de realizar los modelos, tal y como se ha comentado en la Sección 3.4, se podría hacer un análisis más exhaustivo de la elección de los hiperparámetros y la realización de modelos entrenando y prediciendo, no solo por semanas, sino por partidos, jornadas, meses etc.

Para finalizar, se podrían implementar algunas mejoras en la aplicación como la visualización de ciertos gráficos cuando se predijesen los partidos o si predijésemos un solo partido comparar en una gráfica las estadísticas de ambos equipos hasta la fecha que se quiera predecir en esa temporada.

5.4. Competencias desarrolladas

En esta sección se van a detallar cómo se han desarrollado las diferentes competencias propias de la intensificación de computación, las cuales se citaron en la Sección 1.3.

- Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Esta competencia ha sido abordada en el Capítulo 4, donde se ha desarrollado una aplicación web a la que se le han incluido dos modelos, los cuales han sido seleccionados anteriormente mediante un análisis comparativo de los modelos.

- Capacidad para adquirir, obtener, formalizar y representar el conocimiento humano en una forma computable para la resolución de problemas mediante un sistema informático en cualquier ámbito de aplicación, particularmente los relacionados con aspectos

de computación, percepción y actuación en ambientes o entornos inteligentes.

Esta capacidad ha sido tratada en la Sección 3.3, donde se ha realizado el correspondiente preprocesamiento de datos.

- Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.

En la Sección 3.4 se ha llevado a cabo el desarrollo de modelos basados en árboles de decisión, *ensembles* y redes neuronales empleando técnicas y algoritmos de aprendizaje automático.

A. Anexo

A.1. *Cross Industry Standard Process for Data Mining (CRISP-DM)*

Cross Industry Standard Process for Data Mining, también conocida como CRISP-DM, es una metodología que implementa todas las tareas necesarias en un proyecto de minería de datos [49].

CRISP-DM incluye descripciones de las fases normales de un proyecto, las tareas necesarias en cada fase y una explicación de las relaciones entre las tareas y, a su vez, ofrece un resumen del ciclo vital de minería de datos [26].

El ciclo vital del modelo contiene seis fases con flechas que indican las dependencias más importantes y frecuentes entre fases, aunque la secuencia de las fases no es estricta y a menudo se requieren vueltas atrás para refinar los procesos realizados [26]. Las fases del ciclo de vida son:

1. **Comprensión del negocio.** Es la primera fase donde debe comenzar un proyecto de minería de datos donde se enfoca en la comprensión de los objetivos de proyecto, seguidamente este conocimiento de los datos se convierte en un problema de minería de datos y en un plan preliminar diseñado para alcanzar los objetivos [1].
2. **Comprensión de los datos.** En esta segunda fase se realiza una recolección y exploración inicial de los datos. Esta fase es muy importante, ya que, nos ayuda a descubrir conocimiento sobre los datos previamente al modelado.
3. **Preparación de los datos.** Esta fase contempla la selección, limpieza y generación de conjuntos de datos. Básicamente, cubren las actividades básicas para construir el conjunto final de datos.

4. **Modelado.** Durante esta fase se seleccionan algoritmos de modelo y los mejores parámetros correspondientes para obtener posteriormente buenos resultados. A partir de estos algoritmos y técnicas de modelado con sus parámetros se construyen los modelos.
5. **Evaluación.** Esta fase se realizará después de haber realizado algunos modelos, de los cuales realizaremos una evaluación de los resultados obtenidos en función de nuestros objetivos.
6. **Despliegue.** En esta fase se lleva a cabo la implementación, monitorización y mantenimiento de los modelos [49], y donde el conocimiento obtenido tendrá que organizarse y presentarse para que el cliente pueda usarlo [1].

En la Figura A.1 podemos ver un ciclo de vida completo de la metodología de *Cross Industry Standard Process for Data Mining* (CRISP-DM).

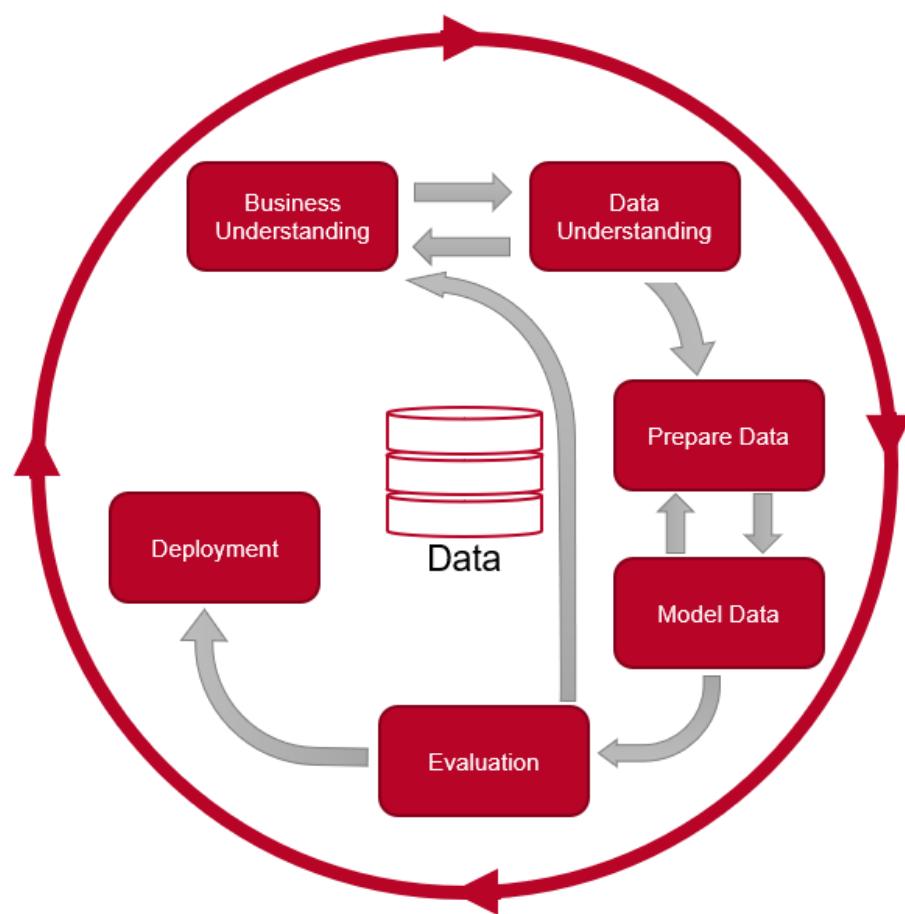


Figura A.1: Ciclo de vida CRISP-DM [2].

A.2. Planificación del proyecto

A lo largo de esta sección se va a comentar cómo se ha planificado el trabajo y la metodología que se ha podido seguir durante el desarrollo del mismo.

En la Figura A.2 se puede visualizar un diagrama donde se detalla cuánto tiempo ha llevado cada paso de la metodología CRIPS-DM. Previamente, se realizará un repaso de qué es cada fase para así comprender de mejor manera el diagrama.

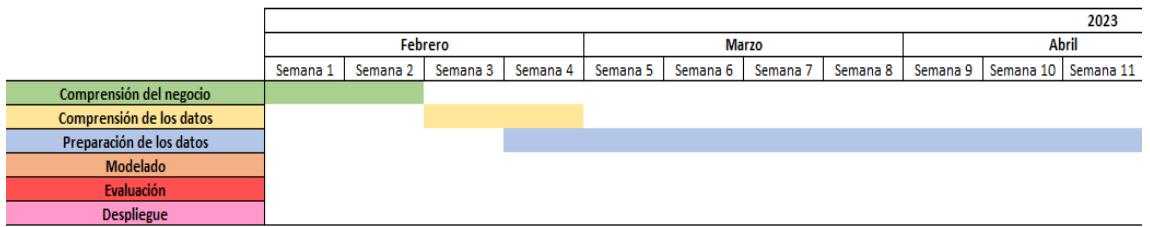
1. **Comprensión del negocio.** Esta fase se detalla en el Capítulo 1 y es la primera fase donde debe comenzar un proyecto de minería de datos donde se enfoca en la comprensión de los objetivos de proyecto, seguidamente este conocimiento de los datos se convierte en un problema de minería de datos y en un plan preliminar diseñado para alcanzar los objetivos.
2. **Comprensión de los datos.** Esta fase corresponde al Capítulo 3, la cual realiza una recolección y exploración inicial de los datos.
3. **Preparación de los datos.** Esta fase corresponde al Capítulo 3, la cual contempla la selección, limpieza y generación de conjuntos de datos.
4. **Modelado.** Esta fase se puede observar en el Capítulo 3. Durante esta fase se seleccionan algoritmos de modelo y los mejores parámetros correspondientes para obtener posteriormente buenos resultados. A partir de estos algoritmos y técnicas de modelado con sus parámetros se construyen los modelos.
5. **Evaluación.** Esta fase se puede visualizar en el Capítulo 3. Realizaremos una evaluación de los resultados obtenidos en función de nuestros objetivos.
6. **Despliegue.** Esta fase corresponde al Capítulo 4, en la cual se lleva a cabo la implementación, monitorización y mantenimiento de los modelos.

El desarrollo del trabajo se ha dividido en una serie de tareas, las cuales se han ido realizando en una serie de *sprints*, una vez terminado cada *sprint* se ha llevado a cabo una reunión con los tutores para poder detallar las tareas del siguiente *sprint*.

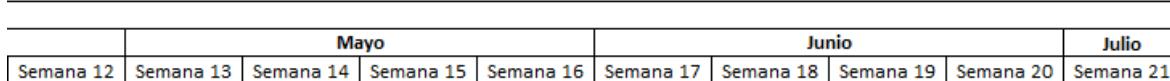
Además, tal y como se ha comentado en la Sección A.1, se seguirá la metodología CRIPD-DM, por lo que, posteriormente a la visualización de la Figura A.2, se van a detallar las tareas que se ha ido realizando en los correspondientes *sprints*.

1. **Creación de una única base de datos.** En este sprint se quiso unificar las 5 bases de datos que se tenían en una sola y, además, la creación de nuevas variables que serían significativas para las posteriores predicciones.
2. **Realización del análisis exploratorio de datos.** Se examina y se comprenden los datos antes de realizar ningún modelo. Se visualizarán diferentes gráficas para estudiar las distintas variables del conjunto de datos.

-
3. **Preprocesamiento de datos.** Se realiza una selección de las características significativas para realizar los modelos posteriormente y, además, se normaliza toda la base de datos para que se tenga una escala similar.
 4. **Realizar modelo sencillo estático.** Se lleva a cabo un modelo simple que entrena por temporadas y predice una temporada entera, lo cual es poco efectivo, ya que, de una temporada a otra puede haber cambios que hagan que el equipo gane más en la siguiente temporada.
 5. **Realización de modelos más precisos estáticos.** Se efectuaron modelos más precisos que el anterior como modelos prediciendo por partidos, jornadas y semanas. Finalmente, se decidió utilizar para obtener los resultados el de semanas por la reducción del tiempo de ejecución.
 6. **Realizar estudio de los modelos estáticos.** Se obtuvo los resultados correspondientes a los modelos estáticos y se obtuvieron las características más importantes que consideraba cada clasificador.
 7. **Realización de los modelos dinámicos.** Se llevó a cabo la realización de modelos dinámicos para la obtención de un *accuracy* mayor.
 8. **Realizar estudio de los modelos dinámicos** Se obtuvo los resultados correspondientes a los modelos dinámicos y se compararon los resultados con los modelos estáticos.
 9. **Desarrollo de la aplicación web.** Se llevó una aplicación web, donde se podía elegir una jornada y se mostraban al usuario las predicciones correspondientes a esa jornada.
 10. **Últimas mejoras.** Finalmente, se añaden algunas mejoras a la aplicación para que muestre las probabilidades de las predicciones y muestre si se acertó o no en la predicción. Además, se realizó un modelo más en el que se empleaban 3 clasificadores y se obtenían las predicciones mediante votación por mayoría en cada una de ellas.



(a) Planificación de la comprensión del negocio, de la comprensión de los datos y la preparación de los datos.



(b) Planificación del modelado, de la evaluación y el despliegue.

Figura A.2: Diagrama de planificación.

A.3. Bases de datos

En esta sección vamos a observar el contenido de las distintas bases de datos que utilizamos durante este proyecto. La Tabla A.1 especifica el contenido de la base de datos games.csv, la Tabla A.2 el contenido de la base de datos games_details.csv, la Tabla A.3 el contenido de la base de datos players.csv, la Tabla A.4 el contenido de la base de datos ranking.csv y la Tabla A.5 el contenido de la base de datos teams.csv. Finalmente, cabe destacar que la Tabla A.6 representará la base de datos que hemos creado a partir de las anteriores y la cual utilizamos en la realización de nuestros modelos.

Variable	Descripción
GAME_DATE_EST	Fecha en la que se produce el encuentro.
GAME_ID	Id del partido.
GAME_STATUS_TEXT	Indica si ha terminado el partido, en nuestro caso todos los partidos han terminado, por lo que esta variable siempre tendrá valor 1.
Continúa en la siguiente página	

Tabla A.1 – continuación de la página anterior

Variable	Descripción
<i>HOME_TEAM_ID</i>	Id del equipo local.
<i>VISITOR_TEAM_ID</i>	Id del equipo visitante.
<i>SEASON</i>	Temporada en la que se produce el encuentro.
<i>TEAM_ID_home</i>	Id del equipo local, por lo que, tendrá un valor duplicado respecto a la variable HOME_TEAM_ID.
<i>PTS_home</i>	Puntos que metió el equipo local en el encuentro.
<i>FG_PCT_home</i>	Porcentaje de tiros de campo anotados del equipo local en el encuentro.
<i>FT_PCT_home</i>	Porcentaje de tiros libres acertados del equipo local en el encuentro.
<i>FG3_PCT_home</i>	Porcentaje de triples acertados del equipo local en el encuentro.
<i>AST_home</i>	Asistencias que hizo el equipo local en el encuentro.
<i>REB_home</i>	Rebotes que cogió el equipo local en el encuentro.
<i>TEAM_ID_away</i>	Id del equipo visitante, por lo que, tendrá un valor duplicado respecto a la variable VISITOR_TEAM_ID.
<i>PTS_away</i>	Puntos que metió el equipo visitante en el encuentro.
<i>FG_PCT_away</i>	Porcentaje de tiros de campo anotados del equipo visitante en el encuentro.
<i>FT_PCT_away</i>	Porcentaje de tiros libres acertados del equipo visitante en el encuentro.
<i>FG3_PCT_away</i>	Porcentaje de triples acertados del equipo visitante en el encuentro.
<i>AST_away</i>	Asistencias que hizo el equipo visitante en el encuentro.
<i>REB_away</i>	Rebotes que cogió el equipo visitante en el encuentro.
<i>HOME_TEAM_WINS</i>	Indica si el equipo local ganó o no, si es un 1 entonces significa que ganó, sin embargo, si es un 0, entonces el equipo local perdió.

Tabla A.1: Base de datos *bbdd_NBA.csv*

Variable	Descripción
<i>GAME_ID</i>	Id del partido.
Continúa en la siguiente página	

Tabla A.2 – continuación de la página anterior

Variable	Descripción
<i>TEAM_ID</i>	Id del equipo.
<i>TEAM_ABBREVIATION</i>	Abreviación del nombre del equipo.
<i>TEAM_CITY</i>	Ciudad a la que pertenece el equipo.
<i>PLAYER_ID</i>	Id del jugador.
<i>PLAYER_NAME</i>	Nombre del jugador.
<i>NICKNAME</i>	Apodo del jugador.
<i>START_POSITION</i>	Indica si es titular y su posición, si es titular pondrá una abreviatura con la posición en la que juega G (base/escolta), F (alero/ala-pívot) o C (pívot), si no lo es, la celda será null.
<i>COMMENT</i>	Comentario acerca de si el jugador jugó el partido.
<i>MIN</i>	Minutos que jugó el jugador en el encuentro.
<i>FGM</i>	Tiros metidos por el jugador en el encuentro.
<i>FGA</i>	Tiros intentados por el jugador en el encuentro.
<i>FG_PCT</i>	Porcentajes de tiros anotados por el jugador en el encuentro.
<i>FG3M</i>	Triples anotados por el jugador en el encuentro.
<i>FG3A</i>	Triples intentados por el jugador en el encuentro.
<i>FG3_PCT</i>	Porcentajes de triples anotados por el jugador en el encuentro.
<i>FTM</i>	Tiros libres anotados por el jugador en el encuentro.
<i>FTA</i>	Tiros libres intentados por el jugador en el encuentro.
<i>FT_PCT</i>	Porcentaje de tiros libres anotados por el jugador en el encuentro.
<i>OREB</i>	Rebotes ofensivos obtenidos por el jugador en el encuentro.
<i>DREB</i>	Rebotes defensivos obtenidos por el jugador en el encuentro.
<i>REB</i>	Rebotes totales obtenidos por el jugador en el encuentro.
<i>AST</i>	Asistencias dadas por el jugador en el encuentro.
<i>STL</i>	Robos realizados por el jugador en el encuentro.
<i>BLK</i>	Tapones realizados por el jugador en el encuentro.
<i>TO</i>	Pérdidas realizadas por el jugador en el encuentro.
<i>PF</i>	Faltas realizadas por el jugador en el encuentro.
<i>PTS</i>	Puntos hechos por el jugador en el encuentro.
<i>PLUS_MINUS</i>	Diferencia, mientras el jugador está en pista, entre los puntos aportados por el equipo y los puntos encajados por el equipo.

Tabla A.2: Base de datos *games_details.csv*

Variable	Descripción
<i>PLAYER_NAME</i>	Nombre del jugador, donde habrá 1749 valores únicos.
<i>TEAM_ID</i>	Id del equipo.
<i>PLAYER_ID</i>	Id del jugador.
<i>SEASON</i>	Temporada en la que juega el jugador.

Tabla A.3: Base de datos *players.csv*

Variable	Descripción
<i>TEAM_ID</i>	Id del equipo.
<i>LEAGUE_ID</i>	Id de la liga, siempre tendrá un valor de 0.
<i>SEASON_ID</i>	Temporada dada la fecha.
<i>STANDINGSDATE</i>	Fecha donde se recogen los datos.
<i>CONFERENCE</i>	Conferencia en la que está el equipo correspondiente.
<i>TEAM</i>	Nombre del equipo correspondiente.
<i>G</i>	Número de partidos jugados hasta la fecha.
<i>W</i>	Número de partidos ganados hasta la fecha.
<i>L</i>	Número de partidos perdidos hasta la fecha.
<i>W_PCT</i>	Porcentaje de partidos ganados hasta la fecha.
<i>HOME_RECORD</i>	Récord en casa del equipo hasta la fecha.
<i>ROAD_RECORD</i>	Récord fuera de casa del equipo hasta la fecha.
<i>RETURNTOPLAY</i>	Indica qué equipos tenían que ir a jugar a la famosa "burbuja" en la temporada 2019-2020, debido al parón por la pandemia mundial del COVID-19. Durante este periodo jugaron 8 partidos y los playoffs, y permanecieron en el recinto de <i>Walt Disney World</i> en Orlando, como zona de aislamiento.

Tabla A.4: Base de datos *ranking.csv*

Variable	Descripción
<i>LEAGUE_ID</i>	Id de la liga, donde todas las filas serán 0.
<i>TEAM_ID</i>	Id del equipo.
<i>MIN_YEAR</i>	Año en el que se fundó el equipo.
<i>MAX_YEAR</i>	Año hasta donde llega la existencia del equipo.
<i>ABBREVIATION</i>	Abreviación del nombre del equipo.
Continúa en la siguiente página	

Tabla A.5 – continuación de la página anterior

Variable	Descripción
<i>NICKNAME</i>	Apodo del equipo.
<i>YEARFOUNDED</i>	Año en el que se fundó el equipo, lo cual es una columna duplicada con respecto a MIN_YEAR.
<i>CITY</i>	Ciudad a la que pertenece el equipo.
<i>ARENA</i>	Nombre del estadio del equipo.
<i>ARENACAPACITY</i>	Capacidad del estadio del equipo correspondiente.
<i>OWNER</i>	Propietario del equipo.
<i>GENERALMANAGER</i>	Gerente general del equipo.
<i>HEADCOACH</i>	Entrenador principal del equipo.
<i>DLEAGUEAFFILIATION</i>	Nombre del equipo de la G-League al que están afiliados. La G-League es una liga menor de baloncesto promocionada por la NBA.

Tabla A.5: Base de datos *teams.csv*

Variable	Descripción
<i>GAME_DATE_EST</i>	Fecha en la que se produce el partido.
<i>GAME_ID</i>	Id del partido.
<i>HOME_TEAM_ID</i>	Id del equipo local.
<i>HOME_TEAM_NAME</i>	Nombre del equipo local.
<i>VISITOR_TEAM_ID</i>	Id del equipo visitante.
<i>VISITOR_TEAM_NAME</i>	Nombre del equipo visitante.
<i>SEASON</i>	Temporada en la que se produce el partido.
<i>PTS_H</i>	Puntos por partido del equipo local, como local, hasta la fecha durante esa temporada. Si el partido correspondiente es el primer partido de ese equipo en esa temporada, entonces esta variable representaría los puntos por partido como local de la temporada pasada. Es importante saber que esta última condición la aplicaremos a las siguientes variables que corresponden a los promedios de los equipos.
<i>FG_PCT_H</i>	Porcentaje de tiros del equipo local hasta la fecha durante esa temporada como local.
<i>FG3_PCT_H</i>	Porcentaje de triples del equipo local hasta la fecha durante esa temporada como local.
<i>AST_H</i>	Asistencias por partido del equipo local hasta la fecha durante esa temporada como local.
Continúa en la siguiente página	

Tabla A.6 – continuación de la página anterior

Variable	Descripción
<i>REB_H</i>	Rebotes por partido del equipo local hasta la fecha durante esa temporada como local.
<i>PTS_A</i>	Puntos por partido del equipo visitante, como visitante, hasta la fecha durante esa temporada. Si el partido correspondiente es el primer partido de ese equipo en esa temporada, entonces esta variable representaría los puntos por partido como visitante de la temporada pasada. Es importante saber que esta última condición la aplicaremos a las siguientes variables que corresponden a los promedios de los equipos.
<i>FG_PCT_A</i>	Porcentaje de tiros del equipo visitante hasta la fecha durante esa temporada como visitante.
<i>FG3_PCT_A</i>	Porcentaje de triples del equipo visitante hasta la fecha durante esa temporada como visitante.
<i>AST_A</i>	Asistencias por partido del equipo visitante hasta la fecha durante esa temporada como visitante.
<i>REB_A</i>	Rebotes por partido del equipo visitante hasta la fecha durante esa temporada como visitante.
<i>W_H</i>	Números de partidos ganados por parte del equipo local hasta la fecha en esa temporada.
<i>L_H</i>	Números de partidos perdidos por parte del equipo local hasta la fecha en esa temporada.
<i>W_PCT_H</i>	Porcentaje de victorias por parte del equipo local hasta la fecha en esa temporada.
<i>W_A</i>	Números de partidos ganados por parte del equipo visitante hasta la fecha en esa temporada.
<i>L_A</i>	Números de partidos perdidos por parte del equipo visitante hasta la fecha en esa temporada.
<i>W_PCT_A</i>	Porcentaje de victorias por parte del equipo visitante hasta la fecha en esa temporada.
<i>PLAYER_ID_I_EI</i>	Id del mejor jugador del equipo local.
<i>PLAYER_NAME_I_EI</i>	Nombre del mejor jugador del equipo local.
<i>PTS_I_EI</i>	Puntos por partido hasta el momento en esa temporada del mejor jugador del equipo local.
<i>AST_I_EI</i>	Asistencias por partido hasta el momento en esa temporada del mejor jugador del equipo local.
<i>REB_I_EI</i>	Rebotes por partido hasta el momento en esa temporada del mejor jugador del equipo local.

Continúa en la siguiente página

Tabla A.6 – continuación de la página anterior

Variable	Descripción
<i>VAL_1_E1</i>	Valoración por partido hasta el momento en esa temporada del mejor jugador del equipo local.
<i>COMMENT_J1_E1</i>	Indica un 1 si jugó el anterior partido o un 0 si se perdió el anterior partido, con esto consideraremos que puede ser que sea duda para jugar el partido actual y que puede afectar en el resultado.
<i>PLAYER_ID_2_E1</i>	Id del segundo mejor jugador del equipo local.
<i>PLAYER_NAME_2_E1</i>	Nombre del segundo mejor jugador del equipo local.
<i>PTS_2_E1</i>	Puntos por partido hasta el momento en esa temporada del segundo mejor jugador del equipo local.
<i>AST_2_E1</i>	Asistencias por partido hasta el momento en esa temporada del segundo mejor jugador del equipo local.
<i>REB_2_E1</i>	Rebotes por partido hasta el momento en esa temporada del segundo mejor jugador del equipo local.
<i>VAL_2_E1</i>	Valoración por partido hasta el momento en esa temporada del segundo mejor jugador del equipo local.
<i>COMMENT_J2_E1</i>	Indica un 1 si jugó el anterior partido o un 0 si se perdió el anterior partido, con esto consideraremos que puede ser que sea duda para jugar el partido actual y que puede afectar en el resultado.
<i>PLAYER_ID_3_E1</i>	Id del tercer mejor jugador del equipo local.
<i>PLAYER_NAME_3_E1</i>	Nombre del tercer mejor jugador del equipo local.
<i>PTS_3_E1</i>	Puntos por partido hasta el momento en esa temporada del tercer mejor jugador del equipo local.
<i>AST_3_E1</i>	Asistencias por partido hasta el momento en esa temporada del tercer mejor jugador del equipo local.
<i>REB_3_E1</i>	Rebotes por partido hasta el momento en esa temporada del tercer mejor jugador del equipo local.
<i>VAL_3_E1</i>	Valoración por partido hasta el momento en esa temporada del tercer mejor jugador del equipo local.
<i>COMMENT_J3_E1</i>	Indica un 1 si jugó el anterior partido o un 0 si se perdió el anterior partido, con esto consideraremos que puede ser que sea duda para jugar el partido actual y que puede afectar en el resultado.
<i>PLAYER_ID_1_E2</i>	Id del mejor jugador del equipo visitante.
Continúa en la siguiente página	

Tabla A.6 – continuación de la página anterior

Variable	Descripción
<i>PLAYER_NAME_1_E2</i>	Nombre del mejor jugador del equipo visitante.
<i>PTS_1_E2</i>	Puntos por partido hasta el momento en esa temporada del mejor jugador del equipo visitante.
<i>AST_1_E2</i>	Asistencias por partido hasta el momento en esa temporada del mejor jugador del equipo visitante.
<i>REB_1_E2</i>	Rebotes por partido hasta el momento en esa temporada del mejor jugador del equipo visitante.
<i>VAL_1_E2</i>	Valoración por partido hasta el momento en esa temporada del mejor jugador del equipo visitante.
<i>COMMENT_J1_E2</i>	Indica un 1 si jugó el anterior partido o un 0 si se perdió el anterior partido, con esto consideraremos que puede ser que sea duda para jugar el partido actual y que puede afectar en el resultado.
<i>PLAYER_ID_2_E2</i>	Id del segundo mejor jugador del equipo visitante.
<i>PLAYER_NAME_2_E2</i>	Nombre del segundo mejor jugador del equipo visitante.
<i>PTS_2_E2</i>	Puntos por partido hasta el momento en esa temporada del segundo mejor jugador del equipo visitante.
<i>AST_2_E2</i>	Asistencias por partido hasta el momento en esa temporada del segundo mejor jugador del equipo visitante.
<i>REB_2_E2</i>	Rebotes por partido hasta el momento en esa temporada del segundo mejor jugador del equipo visitante.
<i>VAL_2_E2</i>	Valoración por partido hasta el momento en esa temporada del segundo mejor jugador del equipo visitante.
<i>COMMENT_J2_E2</i>	Indica un 1 si jugó el anterior partido o un 0 si se perdió el anterior partido, con esto consideraremos que puede ser que sea duda para jugar el partido actual y que puede afectar en el resultado.
<i>PLAYER_ID_3_E2</i>	Id del tercer mejor jugador del equipo visitante.
<i>PLAYER_NAME_3_E2</i>	Nombre del tercer mejor jugador del equipo visitante.
<i>PTS_3_E2</i>	Puntos por partido hasta el momento en esa temporada del tercer mejor jugador del equipo visitante.
<i>AST_3_E2</i>	Asistencias por partido hasta el momento en esa temporada del tercer mejor jugador del equipo visitante.

Continúa en la siguiente página

Tabla A.6 – continuación de la página anterior

Variable	Descripción
<i>REB_3_E2</i>	Rebotes por partido hasta el momento en esa temporada del tercer mejor jugador del equipo visitante.
<i>VAL_3_E2</i>	Valoración por partido hasta el momento en esa temporada del tercer mejor jugador del equipo visitante.
<i>COMMENT_J3_E2</i>	Indica un 1 si jugó el anterior partido o un 0 si se perdió el anterior partido, con esto consideraremos que puede ser que sea duda para jugar el partido actual y que puede afectar en el resultado.
<i>HOME_TEAM_WINS</i>	Representa la variable a predecir e indica 1 si el equipo local ganó o 0 si no ganó el encuentro.

Tabla A.6: Base de datos *bbdd_NBA.csv*

A.4. Recursos software

Durante la realización de este proyecto, se han utilizado distintos recursos software, los cuales se detallarán a continuación ordenados alfabéticamente.

- **Flask.** *Framework* que permite crear aplicaciones web de forma rápida, el cual está escrito en el lenguaje de programación Python. En nuestro proyecto lo utilizaremos para crear nuestra aplicación web.
- **Graphviz.** Biblioteca de visualización de gráficos que utiliza el lenguaje *DOT* para generar gráficos automáticamente, la cual se utiliza para la visualización de los árboles de decisión en la creación de los modelos.
- **Iertools** Biblioteca estándar de Python que proporciona una serie de herramientas y funciones para trabajar con iteradores. Es utilizada para crear todas las combinaciones posibles con los correspondientes hiperparámetros en la creación de los modelos.
- **Jupyter notebooks.** Aplicación web de código abierto que permite crear y compartir documentos interactivos que contienen código, visualizaciones y texto. Utilizado para la realización de todo el código del TFG.
- **Keras.** Biblioteca de *Deep Learning* de alto nivel escrita en Python, la cual es utilizada para la realización de los modelos estáticos con redes neuronales.
- **Matplotlib.** Biblioteca de visualización de datos en Python, la cual es utilizada extraer las gráficas en el análisis exploratorio de datos y las gráficas que muestran la importancia de las características en los modelos realizados.

-
- **Microsoft Excel.** Aplicación de hojas de cálculo, que se utiliza para la realización del diagrama en la planificación del proyecto, en la Sección [A.2](#).
 - **NumPy.** Biblioteca de Python utilizada principalmente para el cálculo numérico, la cual es empleada en la creación de la base de datos y en la realización de los modelos.
 - **Overleaf.** Plataforma en línea que permite la edición y colaboración en tiempo real de documentos LaTeX, la cual es utilizada para la realización de este documento.
 - **Pandas.** Biblioteca de software de código abierto para el lenguaje de programación Python. Esta biblioteca fue utilizada para el tratamiento y manipulación de datos, además, de emplear estructuras de datos eficientes como los *DataFrames*.
 - **Plotly.** Biblioteca de visualización interactiva en Python que permite crear gráficos y visualizaciones de datos, utilizada en el análisis exploratorio de datos.
 - **Python.** Lenguaje de programación de alto nivel, interpretado y de propósito general, el cual es utilizado durante la realización de la base de datos, el análisis exploratorio de datos, creación y evaluación de los modelos y creación de la aplicación web.
 - **Scikit-learn.** Biblioteca de aprendizaje automático utilizada para la creación de los modelos correspondientes.
 - **Scikit-multiflow.** Biblioteca de aprendizaje automático en Python diseñada para el aprendizaje en continuo y el procesamiento de datos en flujo, la cual es utilizada para la realización de los *hoeffding trees* en la realización de los modelos.
 - **SciPy.** Biblioteca de Python para la computación científica y el análisis de datos, la cual es utilizada para calcular la ganancia de información en el análisis exploratorio de datos.
 - **Seaborn.** Biblioteca de visualización de datos en Python utilizada en el análisis exploratorio de datos.
 - **TensorFlow.** Biblioteca de código abierto utilizada para el aprendizaje automático y la inteligencia artificial. Esta biblioteca es empleada para la realización de los modelos con redes neuronales.
 - **Time.** Es una biblioteca en Python que proporciona funciones para trabajar con el tiempo, la cual es empleada para el cálculo del tiempo de ejecución de los modelos.
 - **Visual Studio Code.** Software creado por Microsoft que se utiliza para abrir, ver y editar archivos de código fuente, el cual utilizaremos en nuestros archivos Python.
 - **Windows 10.** Sistema operativo desarrollado por Microsoft donde se ha ejecutado todo el proyecto.

A.5. Recursos hardware

A lo largo de este trabajo de fin de grado se han utilizado una serie de recursos hardware, los cuales son los citados a continuación.

- **CPU.** Intel(R) Core(TM) i7-8750H a 2.20GHz.
- **Tarjeta gráfica.** NVIDIA GeForce GTX 1050 Ti.
- **RAM.** 16,0 GB.

Referencia bibliográfica

- [1] Crisp-dm: La metodología para poner orden en los proyectos. , <https://www.sngular.com/es/data-science-crisp-dm-metodologia/>. Online; accedido 20-05-2023.
- [2] The data science process (crisp-dm). , year=2020, <https://michael-fuchs-python.netlify.app/2020/08/21/the-data-science-process-crisp-dm/>. Online; accedido 20-05-2023.
- [3] Evaluación de indicadores adelantados mediante el área auc. https://www.bis.org/publ/qtrpdf/r_qt1403z_es.htm. Online; accedido 29-06-2023.
- [4] Graphviz. <https://graphviz.org/>. Online; accedido 05-06-2023.
- [5] Histogram algorithm of lightgbm. https://www.researchgate.net/figure/Histogram-algorithm-of-LightGBM_fig2_350848994. Online; accedido 19-06-2023.
- [6] Modelos naive bayes: Precisión e independencia. <https://theblackboxlab.com/2022/03/30/modelos-naive-bayes/>. Online; accedido 05-07-2023.
- [7] pickle — python object serialization. <https://docs.python.org/3/library/pickle.html>. Online; accedido 20-06-2023.
- [8] Preparación de datos para el aprendizaje automático mejorado. , <https://learn.microsoft.com/es-es/azure/architecture/data-science-process/prepare-data>. Online; accedido 22-05-2023.
- [9] Referencia de la api. https://scikit-learn-org.translate.goog/stable/modules/classes.html?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sc#module-sklearn.tree. Online; accedido 05-06-2023.
- [10] sklearn.ensemble.adaboostclasificador. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>. Online; accedido 10-06-2023.

-
- [11] sklearn.ensemble.baggingclassifier. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html>. Online; accedido 08-06-2023.
 - [12] sklearn.ensemble.gradientboostingclassifier. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>. Online; accedido 12-06-2023.
 - [13] sklearn.ensemble.histgradientboostingclassifier. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html>. Online; accedido 14-06-2023.
 - [14] sklearn.ensemble.randomforestclassifier. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. Online; accedido 10-06-2023.
 - [15] skmultiflow.trees.hoeffdingtreeclassifier. <https://scikit-multiflow.readthedocs.io/en/stable/api/generated/skmultiflow.trees.HoeffdingTreeClassifier.html>. Online; accedido 23-06-2023.
 - [16] What is hoeffding tree algorithm? <https://www.tutorialspoint.com/what-is-hoeffding-tree-algorithm>. Online; accedido 19-06-2023.
 - [17] What's more important to nba team success, depth or star power? <https://bleacherreport.com/articles/1281737-whats-more-important-to-nba-team-success-depth-or-star-power>. Online; accedido 20-05-2023.
 - [18] ¿qué es el sobreajuste? <https://aws.amazon.com/es/what-is/overfitting/>. Online; accedido 08-06-2023.
 - [19] ¿qué es una red neuronal? <https://aws.amazon.com/es/what-is/neural-network/>. Online; accedido 28-06-2023.
 - [20] ¿cómo aprenden las máquinas? machine learning y sus diferentes tipos. , <https://datos.gob.es/es/blog/como-aprenden-las-maquinias-machine-learning-y-sus-diferentes-tipos>, 2020. Online; accedido 02-06-2023.
 - [21] Claudia Carpintero Borrajo et al. Estudio del rendimiento en ciclistas mediante técnicas del aprendizaje automático. <https://repositorio.unican.es/xmlui/bitstream/handle/10902/23576/CarpinteroBorrajoClaudia-TFG-Matematicas.pdf?sequence=1>, note = "Online; accedido 03-05-2023.", 2021.
 - [22] Frank Ceballos. Scikit-learn decision trees explained. <https://towardsdatascience.com/scikit-learn-decision-trees-explained-803f3812290d>. Online; accedido 08-06-2023.
-

- [23] Luis de la Ossa y José Antonio Gámez. Lesson 6. combining classifiers - ensembles. Asignatura de minería de datos. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática. Universidad de Castilla-La Mancha, 2023. Online; accedido 03-06-2023.
- [24] Luis de la Ossa y José Antonio Gámez. Lesson 9: Neural networks (and deep learning). Asignatura de minería de datos. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática. Universidad de Castilla-La Mancha, 2023. Online; accedido 28-06-2023.
- [25] Francisco Herrera and JR Cano. Técnicas de reducción de datos en kdd. el uso de algoritmos evolutivos para la selección de instancias. *Actas del I Seminario sobre Sistemas Inteligentes (SSI_06)*, pages 165–181, 2006. Online; accedido 22-05-2023.
- [26] IBM. Conceptos básicos de ayuda de crisp-dm. , year=2021, <https://www.ibm.com/docs/es/spss-modeler/saas?topic=dm-crisp-help-overview>. Online; accedido 20-05-2023.
- [27] IBM. Cómo funciona el aprendizaje supervisado. <https://www.ibm.com/es-es/topics/supervised-learning>. Online; accedido 22-05-2023.
- [28] IBM. Decision trees. <https://www.ibm.com/topics/decision-trees>. Online; accedido 22-05-2023.
- [29] IBM. ¿qué es bagging? <https://www.ibm.com/es-es/topics/bagging>. Online; accedido 03-06-2023.
- [30] Javatpoint. Algoritmo de clasificación del árbol de decisión. <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>. Online; accedido 22-05-2023.
- [31] Miljan Kovacevic. Bootstrap aggregation (bagging) in regression tree ensembles. https://www.researchgate.net/figure/Bootstrap-aggregation-bagging-in-regression-tree-ensembles_fig1_349624596. Online; accedido 20-05-2023.
- [32] José Domingo Muñoz. Qué es flask. <https://openwebinars.net/blog/que-es-flask/#ventajas-de-usar-un-framework>. Online; accedido 20-06-2023.
- [33] NBA. Estadísticas de ambos equipos en un partido. https://www.google.com/search?q=lakers+nba+grizzlies+game+1+playoffs+2023&rlz=1C1CHBD_esES1030ES1030&sxsrf=APwXEdcU5WDYUgAJjwbnji_jyvLBG0iXqg. Online; accedido 14-05-2023.
- [34] NBA. About the nba. <https://www.nba.com/news/about>, 2023. Online; accedido 22-05-2023.

-
- [35] Oracle. ¿qué es el aprendizaje automático? <https://www.oracle.com/es/artificial-intelligence/machine-learning/what-is-machine-learning/>. Online; accedido 22-05-2023.
- [36] Nikunj C. Oza. Ensemble data mining methods. <https://ntrs.nasa.gov/api/citations/20060015642/downloads/20060015642.pdf>. Online; accedido 20-05-2023.
- [37] Pranjal Pandey. Data preprocessing: Concepts. , <https://towardsdatascience.com/data-preprocessing-concepts-fa946d11c825>. Online; accedido 22-05-2023.
- [38] Paco Pérez. Archivo:nba conferences divisions new.png. https://es.wikipedia.org/wiki/Archivo:NBA_Conferences_Divisions_new.png, 2008. Online; accedido 8-05-2023.
- [39] Sergio Rabinal. ¿qué es, cómo funciona y cuándo se juega el play-in para clasificarse a los nba playoffs 2023? <https://www.sportingnews.com/es/nba/news/que-es-como-funciona-cuando-se-juega-play-in-nba-playoffs/1le5v71gecg2z16eekw790uq32>, 2019. Online; accedido 22-05-2023.
- [40] Sunil Ray. Quick introduction to boosting algorithms in machine learning. <https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/>. Online; accedido 20-05-2023.
- [41] Joaquín Amat Rodrigo. Gradient boosting con python. https://www.cienciadedatos.net/documentos/py09_gradient_boosting_python. Online; accedido 12-06-2023.
- [42] Scikit-learn. Decision trees. <https://scikit-learn.org/stable/modules/tree.html>. Online; accedido 22-05-2023.
- [43] Adán Shafi. Clasificación aleatoria de bosques con scikit-learn random forest classification with scikit-learn. <https://www.datacamp.com/tutorial/random-forests-classifier-python1>. Online; accedido 10-06-2023.
- [44] Harshdeep Singh. Understanding gradient boosting machines. <https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab>. Online; accedido 22-05-2023.
- [45] Statmuse. Best records by a team in a season in nba history. <https://www.statmuse.com/nba/ask/best-records-by-a-team-in-a-season-in-nba-history>. Online; accedido 20-05-2023.

- [46] Statmuse. Stats lebron james vs the grizzlies in the first game 2023 playoffs. <https://www.statmuse.com/nba/ask/stats-lebron-james-vs-the-grizzlies-in-the-first-game-2023-playoffs>, 2023. Online; accedido 15-05-2023.
- [47] Harry Volarevic. What does the growth of the nba betting industry mean for our game? <https://www.basketballnetwork.net/latest-news/what-does-the-growth-of-the-nba-betting-industry-mean-for-our-game>, 2021. Online; accedido 03-05-2023.
- [48] Juan Carlos Alfaro Jiménez y José Antonio Gámez Martín. Práctica 2: Aprendizaje y selección de modelos de clasificación. Asignatura de minería de datos. Grado en Ingeniería Informática. Escuela Superior de Ingeniería Informática. Universidad de Castilla-La Mancha, 2023. Online; accedido 05-06-2023.
- [49] Daniel Álvarez Gil. Metodología crisp-dm. , year=2021, <https://www.adictosaltrabajo.com/2021/01/14/metodologia-crisp-dm/>. Online; accedido 20-05-2023.
- [50] Álvaro Ibañez. Datos y aprendizaje automático para prevenir y evitar lesiones en deportes de contacto. <https://tecvolucion.com/datos-aprendizaje-automatico-deportes-contacto/>, 2019. Online; accedido 22-05-2023.