Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Aplicación de Estrategias y Modelos de Inteligencia Artificial en el Juego de Azar Blackjack en Entornos Web

Autor: Daniel Díaz Buzón

Tutor: Francisco José Fernández Jiménez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

Aplicación de Estrategias y Modelos de Inteligencia Artificial en el Juego de Azar Blackjack en Entornos Web

Autor:
Daniel Díaz Buzón

Tutor:
Francisco José Fernández Jiménez
Profesor Colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado: Aplicación de Estrategias y Modelos de Inteligencia Artificial en el Juego de Azar Blackjack en Entornos Web

Autor: Daniel Díaz Buzón
Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Antes de comenzar veo necesario la dedicación de unas palabras a aquellas personas que han sido un gran apoyo y sin las cuáles probablemente no estaría aquí escribiendo esto.

Primeramente, agradecer a mis padres y hermano el que hayan estado ahí, apoyando de la manera que ellos veían posible o que yo les dejaba. Celebrando cada aprobado y animándome en cada suspenso.

Me gustaría continuar con mis compañeros, los cuales considero amigos. Ellos han logrado que días que prometían ser eternos y deprimentes, como entrar de noche para hacer prácticas y acabar de noche de estudiar hayan sido amenos y divertidos.

Para concluir, mencionar a mi tutor Francisco José Fernández Jiménez y otros docentes que han hecho seguir alimentando esa curiosidad y fascinación por la tecnología que me hizo emprender este camino de las Telecomunicaciones.

Daniel Díaz Buzón

Sevilla, 2024

Resumen

Este Trabajo Fin de Grado (TFG) irá centrado en la creación de un programa para poder practicar el juego de cartas Blackjack con la ayuda de la Inteligencia Artificial (IA). El objetivo principal será poder unir en un mismo programa la matemática y probabilidad de dicho juego con la IA de una manera visual, creativa y divertida.

La segunda sección abordará el Estado del Arte en el que abordaremos los siguientes puntos divididos por subsecciones. La primera hará mención al Blackjack, haciendo un análisis de su historia, las reglas fundamentales en vigor en la actualidad, las diversas estrategias que se pueden emplear y una revisión exhaustiva del estudio de probabilidad realizado sobre este juego. La segunda abarcará a la Inteligencia Artificial donde haremos un repaso de su creación, historia y la evolución que ha sufrido hasta la actualidad, haciendo énfasis en la Interfaz de Programación de Aplicaciones (API) que se usará para dicho TFG. Y la última abordará las herramientas que he usado para la creación de este proyecto, desde lenguaje de programación hasta librerías de Inteligencia Artificial.

La sección siguiente abarcará todo el tema del diseño e implementación del código subdividido en dos apartados: el Frontend, el Backend y la Inteligencia Artificial. Cada apartado estará a su vez dividido en pequeños apartados para poder describir de manera más exhaustiva todos los diseños, funciones y entrenamientos realizados.

En la sección de "Pruebas y Experimentos", se aborda la validación del código propuesto. Se presentan pruebas unitarias, funcionales y de integración que evaluarán cada función, su efectividad y rendimiento en diferentes escenarios.

Por último pero no menos importante se incluirán dos secciones adicionales. La primera se centrará en detallar las diversas complicaciones enfrentadas durante la elaboración de este Trabajo Fin de Grado, mientras que la siguiente se enfocará en discutir las conclusiones derivadas de su finalización y posibles líneas futuras de trabajo.

Este TFG se hará desde un enfoque matemático para poner en práctica todo lo aprendido en la carrera. Los resultados que espero obtener son conocimientos de creación y entrenamiento de modelos IA y como esta pueda rivalizar con un modelo matemático estudiado y creado para el Blackjack.

Abstract

This Final Degree Project (TFG) will be focused on the creation of a program to practice the card game Blackjack with the help of Artificial Intelligence (IA). The main objective will be to be able to unite in the same program the mathematics and probability of this game with the IA in a visual, creative and fun way.

The second section will deal with the State of the Art in which we will address the following points divided by subsections. The first one will mention Blackjack, analysing its history, the fundamental rules currently in force, the different strategies that can be used and an exhaustive review of the probability study carried out on this game. The second will cover Artificial Intelligence where we will review its creation, history and the evolution it has undergone up to the present day, with emphasis on the Application Programming Interface (API) that will be used for this TFG. And the last one will deal with the tools I have used for the creation of this project, from programming language to Artificial Intelligence libraries.

The following section will cover all the design and implementation of the code, divided into two sections: the Frontend, the Backend and Artificial Intelligence. Each section will be further divided into smaller sections in order to be able to describe more exhaustively all the designs, functions and training carried out.

In the "Tests and Experiments" section, the validation of the proposed code is addressed. Unit, functional and integration tests are presented that will evaluate each function, its effectiveness and performance in different scenarios.

Last but not least, two additional sections will be included. The first one will focus on detailing the various complications faced during the elaboration of this Final Degree Project, while the next one will focus on discussing the conclusions derived from its completion and possible future lines of work.

This TFG will be done from a mathematical approach to put into practice everything I have learned in my degree. The results I hope to obtain are knowledge of creation and training of AI models and how it can rival a mathematical model studied and created for Blackjack.

Índice

1 Introducción

"La tecnología es la maestría del hombre sobre la naturaleza. El azar es la naturaleza aún no dominada."

JOHN H.HOLLAND

En los últimos años, hemos presenciado el surgimiento de la Inteligencia Artificial en medio de una pandemia, ya que antes era algo desconocido por la mayoría. Sin embargo, no fue hasta la creación y la amplia difusión del modelo conversacional más sofisticado de ésta, ChatGPT ([54]), que empezamos a tomar plena conciencia de las oportunidades (o desafíos) que esta nos presentaría.

Como ingeniero de telecomunicaciones, reconozco el potencial que podemos obtener de la Inteligencia Artificial ejerciendo un uso correcto de ella. Además, vi una gran posibilidad de aplicarla al juego (sin ánimo de lucro). Por ello fue por lo que surgió la idea de realizar este TFG, unir dos de mis pasiones: jugar a juegos y la tecnología.

Realicé un estudio previo de posibles juegos a los que añadir un modelo de Inteligencia Artificial. En una primera instancia pensé en el famoso juego del póker pero no suscitó en mí el suficiente interés debido a preferencias de jugabilidad, por ello me acabé decantando por el Blackjack.

El aspecto fundamental de este trabajo será la parte de backend, dónde se abordara distintos algoritmos que imiten las estrategias actuales y la creación y entrenamiento del modelo de Inteligencia Artificial.

El presente trabajo se realizará teniendo en cuenta siempre las buenas prácticas y recomendaciones aprendidas en la carrera para la depuración y diseño de un código, así como la correcta creación y escritura de la memoria. Además, se pretende en última instancia no fomentar la adicción al juego ya que dicho trabajo se ha realizado con fines educativos y creativos no con fines lucrativos.

1.1 Alcance

El foco de este proyecto se concentra en el desarrollo de una aplicación programada en Python, aprovechando sus frameworks y librerías, con el propósito de construir un modelo de Inteligencia Artificial diseñado para perfeccionar las habilidades en el juego del Blackjack.

La automatización de diferentes estrategias se limitará a las más conocidas, además de a estudios probabilísticos realizados con el fin de obtener los mejores resultados posibles. El proceso de entrenamiento de la Inteligencia Artificial seguirá el método de prueba y error, respetando las reglas fundamentales del juego.

El objetivo principal de esta herramienta consistirá en brindar apoyo a jugadores de diferentes niveles, facilitándoles la práctica y mejora en el juego.

Adicionalmente, se contempla la implementación de varios niveles en la Inteligencia Artificial (incrementando su entrenamiento) para permitir al usuario personalizar la dificultad. Esto proporcionará una experiencia más adaptativa, permitiendo a los jugadores abordar desafíos personalizados que se alineen con sus metas de aprendizaje. También, se añadirá un jugador NPC (Non Playable Character) con acceso al estudio probabilístico el cual sabrá en cada momento qué jugada hacer, actuando como un desafío final para el usuario.

En resumen, la meta de este proyecto se limita a la creación de una herramienta respaldada por Inteligencia Artificial, destinada a que jugadores de diversos perfiles practiquen y perfeccionen sus habilidades en el Blackjack. Es esencial destacar que no se busca fomentar la adicción al juego ni el uso indebido de esta herramienta.

1.2 Objetivos

Los principales objetivos que se quieren alcanzar con este trabajo son:

- Realizar un estudio acerca de las reglas, estrategias y probabilidades del Blackjack para comprender su funcionamiento.
- Realizar un estudio acerca de la creación y entrenamiento de un modelo de Inteligencia Artificial, usando las librerías de Python.
- Aprovechar los conocimientos recopilados durante esos estudios para elaborar una biblioteca de funciones en Python que permitan recrear el juego del Blackjack, además de poder crear y entrenar mi propio modelo de Inteligencia Artificial.
- Crear una Interfaz Gráfica de Usuario (GUI) para proporcionar una experiencia más amigable y agradable al usuario.
- Enlazar dicho GUI con la biblioteca de funciones de forma que, al interactuar el usuario, esto se traduzca en llamadas a las propias funciones.

1.3 Estructura de la memoria

La presente memoria está organizada de manera coherente para proporcionar una comprensión integral del estudio realizado y sus resultados. A continuación, se describe la estructura general de los capítulos que componen este documento:

Sección 2: Estado del Arte

En esta sección se abordarán los elementos fundamentales del Trabajo de Fin de Grado (TFG), centrándose en el Blackjack, la Inteligencia Artificial y las herramientas usadas para crear el proyecto. Cada uno de estos temas se dividirá en secciones específicas. En el apartado dedicado al Blackjack, se explorarán aspectos como su trasfondo histórico, reglas, estrategias y análisis probabilísticos. En cuanto a la Inteligencia Artificial, se examinará su evolución histórica y se detallará el modelo utilizado en el desarrollo de este proyecto. Por último, se verá una sección dedicada al lenguaje de programación utilizado y librerías de éste para lograr los objetivos.

Sección 3: Diseño e Implementación

Se presenta el diseño e implementación del código dividido en dos grandes secciones. La primera sección tratará el diseño del proyecto, esta explicación se dividirá en 3 apartados: Diseño del Backend, Diseño del Frontend y Diseño de la Inteligencia Artificial. Para finalizar, la segunda sección se centrará en la implementación de dicho diseño, esta también se verá en diferentes apartados para realizar un enfoque mayor en cada uno, de igual manera que con el diseño, se verá primero el Backend, posteriormente el Frontend y terminaremos con la Inteligencia Artificial.

Sección 4: Pruebas y Experimentos

En esta sección se detallan las pruebas realizadas para validar la efectividad y la precisión de las funciones. Se dividen en cuatro categorías: pruebas unitarias, pruebas funcionales, pruebas de

integración y una última que tratará de las pruebas realizadas a la Inteligencia Artificial. Cada prueba se presenta en secciones dedicadas, destacando los casos de prueba utilizados y los resultados obtenidos.

Sección 5: Dificultades Encontradas

Se explicarán todos los problemas encontrados en la elaboración de este trabajo así como de las soluciones usadas para solventarlas en caso de haberlo conseguido.

Sección 6: Conclusiones

Tras todo el estudio realizado en este proyecto se detallan las conclusiones más importantes y las áreas de mejora en caso de que algún otro alumno decida continuar desarrollándolo.

2 Estado del Arte

En el ámbito de la investigación y el desarrollo tecnológico, la convergencia del juego y la Inteligencia Artificial es un terreno fértil para explorar nuevas fronteras. En este estado del arte, navegaremos por la historia, sus reglas, sus complejas estrategias y los análisis probabilísticos del juego del Blackjack. Además, explicaremos también el ascenso y éxito de la Inteligencia Artificial y cómo podríamos aplicarla a dicho juego, usando como base la empresa de OpenAI, una de las más conocidas [55], con la ayuda del lenguaje de programación Python.

2.1 Blackjack

2.1.1 Historia

De entre todos los juegos de casino, el Blackjack siempre ha estado considerado uno de los más representativos y glamurosos, en gran medida por su naturaleza excitante e imprevisible. Su historia es dilatada e interesante, como veremos a continuación.

Las bases del juego se remontan a la época de la antigua Roma (tallando los números en piedra) pero los primeros registros escritos y verificables de este juego aparecen en la novela "*Rinconete y Cortadillo*", de Miguel de Cervantes, escrita a principios del siglo XVII dónde se hacía referencia a un juego llamado "*La Veintiuna*". Este entretenimiento bien podría ser el precursor del Blackjack, ya que su objetivo era llegar a ese valor mediante las cartas. Sin embargo, diversos historiadores atribuyen el origen de este juego de cartas a la Francia de finales del XVII y principios del XVIII; concretamente, por la presencia en los salones de la aristocracia del "*Vingt-et-un*".

Lo cierto es que Francia fue la encargada de expandir el Blackjack por el mundo (mediante los colonos llegó a los Estados Unidos). El juego, en los Estados Unidos, conoció la fama con rapidez, llegando, a principios del siglo XX, a una ciudad de nueva creación en medio del desierto: Las Vegas. Aquí, se produjeron algunos cambios en el juego, lo que daría lugar a la variante conocida como Blackjack Americano. A partir del 2000, aparecerían los primeros casinos en línea.

¿Por qué se llama Blackjack a el juego?

Para promocionar este juego, los primeros casinos americanos decidieron ofrecer una bonificación especial: un pago de 10 a 1 cuando el jugador ligaba un As de picas y una J negra. Esta opción se hizo popular durante la famosa fiebre del oro del Klondike (región canadiense al este de Alaska). El término "*blackjack*" se refiere a la esfalerita, un mineral de sulfuro de zinc, que solía indicar la proximidad de vetas de oro y plata. Por tanto, según esta teoría, la J negra precedía a una mano ganadora.

Blackjack como icono de los juegos de casino en la cultura popular

Por la forma en la que se juega al Blackjack, su naturaleza y sus reglas, además de su historia, este juego se ha convertido en un auténtico icono dentro de los juegos de casino e, incluso, ha traspasado

este espacio para convertirse en parte de la cultura popular. En la actualidad, ya forma parte de un reducido grupo de juegos que ha inspirado películas, canciones, literatura y otras expresiones del arte.

Desde las páginas de los libros de James Bond hasta las composiciones de Sheryl Crow, e incluso en clásicos cinematográficos como "*Rain Man*" y "*21: Blackjack*", el juego de Blackjack y las reflexiones de quienes participan en él han dejado una huella significativa en la cultura popular. En ambas películas, el juego de Blackjack se emplea como un medio para narrar historias donde la inteligencia y la estrategia ocupan un lugar central.

Tampoco faltan episodios científicos relacionados con el Blackjack, como la historia del renombrado matemático y estadístico Edward O. Thorp. En 1962, Thorp escribió el libro "*Beat the Dealer*", detallando sus técnicas para vencer a los casinos mediante el conteo de cartas. Este libro inspiró a generaciones de jugadores a aprender a contar cartas, lo que se tradujo en cambios en las reglas del juego para hacer más difícil el uso de esta técnica.

En general, si la queremos simplificar al máximo, la estrategia óptima sugiere plantarse cuando tienes una puntuación de 17 o superior, pedir carta si está entre 12 y 16 y la banca tiene una puntuación baja y, por último, pedir carta si está por debajo de 12. A raíz del éxito de la estrategia óptima, un grupo de estudiantes del MIT decidió analizar con más detenimiento la estrategia del juego.

El equipo de blackjack del MIT estaba formado por alumnos del Massachusetts Institute of Technology, Harvard y otras universidades de primer nivel. Durante años, este equipo estudió y perfeccionó técnicas y estrategias relacionadas con el recuento de cartas para derrotar a los casinos de todo el mundo. El grupo operó con éxito desde 1979 hasta comienzos del siglo XXI. Además de las clásicas técnicas relacionadas con el recuento de cartas, también perfeccionó otras relacionadas con el seguimiento de los ases. De esta manera, los jugadores obtenían una ventaja competitiva de entre el 2 % y el 4 %.

A lo largo de la historia, el Blackjack ha sido una forma de entretenimiento, pero con el tiempo ha evolucionado, dando lugar a competiciones que brindan la oportunidad de participar a aquellos entusiastas con una mayor destreza en el juego. Este cambio significativo ocurrió con la llegada de los casinos, donde el Blackjack desempeñó un papel destacado, dando origen a los primeros torneos y partidas de carácter profesional.

A día de hoy, encontramos diversas modalidades que han aguantado el paso del tiempo y otras tantas que han aparecido en los últimos años, gracias a las posibilidades que brinda la tecnología que ha llevado al Blackjack hasta el entorno digital, las cuales son "*Blackjack Online*" y "*Blackjack Live*" ambas usando la tecnología actual aunque de maneras diferentes. No se profundizarán en ellas puesto que no son el objetivo principal de este trabajo.

Si se quiere profundizar en la historia y evolución que ha ido teniendo el Blackjack a lo largo de la historia, visitar [56] o [57].

2.1.2 Reglas Básicas

En el blackjack, la principal premisa es que el oponente es la casa, independientemente del número de jugadores en la mesa. Cada jugador se enfrenta directamente al crupier. Por lo tanto, el verdadero objetivo no es alcanzar una puntuación de 21, como podría pensarse debido a la denominación del juego o al hecho de que conseguir 21 resulta en una victoria. Más bien, el objetivo es vencer a la casa, ya sea forzándola a pasarse de 21 o a obtener una puntuación menor que la del jugador.

Para ello se utilizará desde 1 sola baraja hasta un máximo de 8 barajas (en la modalidad clásica utilizan 8 y en la americana 6).

Valor de las cartas y puntuaciones

Las cartas que van del 2 al 10 obtienen el valor que su propio número indica, mientras que las figuras (J, Q y K) valen 10 puntos. Existe una carta “especial” que sería el as ya que esta puede tener un valor de 1 u 11, en función de lo que más convenga al jugador.

La máxima puntuación posible sería conseguir el 21 con únicamente dos cartas (As + 10 o As + figura), esta mano es conocida como "blackjack". El blackjack es una mano casi imbatible, ya que la banca la puede igualar logrando otro blackjack. Sin embargo, el 21 se puede conseguir también con suma de más de dos cartas pero perdería ese valor distintivo del "blackjack". Un dato importante es que, si bien todas las apuestas ganadoras tienen una relación de pagos de 1 a 1 (la banca da 1 ficha por cada ficha apostada por el jugador), el blackjack se paga 3 a 2 (la banca da 3 fichas al jugador por cada 2 fichas apostadas).

Vamos a centrarnos en la mesa para ver cómo se jugaría.

El crupier reparte dos cartas a cada jugador y dos a sí mismo, de estas últimas una estaría boca arriba y una boca abajo, esta última sólo la podrá ver al final de la ronda, cuando todos los jugadores de la mesa han terminado su turno. Después, cada jugador tiene las siguientes opciones (para el proyecto, sólo estará disponible pedir carta y plantarse):

1. **Apostar**: el jugador antes de recibir sus cartas deberá realizar una primera apuesta que se encuentre entre la mínima y la máxima de la mesa.
2. **Pedir carta**: si tiene una puntuación baja y quiere mejorarla. Una vez recibida sus cartas, puede pedir más cartas, llevando cuidado de no superar los 21.
3. **Plantarse**: si el jugador considera que la puntuación que obtiene con sus cartas ya es suficiente.
4. **Dividir**: se puede dividir la mano (aunque no es obligatorio) si el jugador ha recibido dos cartas iguales. Cuando el jugador decide dividir su mano, su apuesta se duplica, ya que en ese momento juega con dos manos distintas.
5. **Doblar**: si la mano del jugador es muy prometedora (es decir, entre 9, 10 u 11), este puede doblar su apuesta inicial antes de recibir otra carta (y ***sólo podrá recibir una más***).
6. **Apuesta de Seguro**: si el crupier con la carta visible recibida tiene la oportunidad de lograr blackjack, preguntará a los jugadores si quieren apostar a que logrará el blackjack. Una vez se decidan todos los jugadores, el crupier tendrá la oportunidad de observar su carta previo al inicio de la partida. Ésta opción de apuesta de seguro sólo está disponible en el Blackjack Americano, en el resto de variables de Blackjack el crupier no verá su carta hasta su turno.
7. **Rendirse**: esto sólo es posible en el Blackjack Americano y hará que perdamos la mitad de nuestra apuesta.

Cuando la ronda ha pedido carta o ha decidido no pedir, la banca completará su mano. En caso de empate, la banca ganará a los jugadores.

Diferencia entre cartas duras y blandas

Una de las principales señas de identidad del Blackjack es el hecho de que el As pueda valer 1 u 11. Por eso, cuando recibimos las dos cartas iniciales y ninguna de ellas es un As, se conoce como "*mano dura*"; mientras que, si el As está entre ellas, será una "*mano blanda*", ya que el valor del As dependerá de lo que más nos interese. Si pedimos más cartas, la mano puede seguir siendo "*blanda*" o "*dura*".

Esta peculiaridad del juego no afecta, únicamente, a la libertad del jugador para tomar ciertas decisiones. También influye a las jugadas de la banca, según sea la modalidad. Por ejemplo, en el Blackjack Clásico, deberá pedir con 16 o menos o con un 17 suave; mientras que, en la versión americana, deberá plantarse con un 17 suave.

Las reglas de la banca

La principal regla es que el crupier está obligado a pedir carta hasta obtener un valor mínimo de 17, es decir, puede darse dos casos:

1. El jugador tiene una mano que suma 14 y el crupier una mano de suma 15, se podría pensar que se ha perdido automáticamente pero como el crupier debe pedir carta hasta un mínimo de 17 puede darse la posibilidad de que éste se pase y ganemos.
2. El jugador tiene una mano que suma 18 y el crupier 17. En esta situación el jugador gana dado que el crupier está obligado a plantarse si tiene un valor mínimo de 17.

Hay casinos en los que existe una regla que dice que el ***crupier tiene que plantarse con un 17 fuerte*** (es decir, si el valor se obtiene de un 10 + 7), pero, ***si se trata de un 17 suave*** (es decir, As + 6), ***puede pedir carta***.

A continuación, veremos también los premios que suelen estar estipulados en los casinos cuando uno se sienta a jugar:

- **1 a 1**: una ficha por cada apostada si ganamos a la banca.
- **3 a 2**: tres fichas por cada dos apostadas si ganamos a la banca haciendo ***blackjack*** (siempre y cuando la banca no lo obtenga también en cuyo caso perderíamos). En el Blackjack Americano de Las Vegas se paga con **6 a 5**.
- **2 a 1**: dos fichas por cada apostada si ganamos una apuesta de seguro.

Five Card Charlie Rule

Esta regla indica que si un jugador llega a pedir tres cartas, es decir, su mano tiene un total de 5 cartas y sigue estando por debajo de 21, se proclama automáticamente como ganador en esa ronda.

Realmente no es muy conocida en el ámbito de los juegos de Blackjack presenciales debido a que tiene una probabilidad bastante baja de que suceda, también es algo que favorece al jugador. Aunque no sea algo que suceda con la suficiente frecuencia como para que resulte en una gran ventaja para el jugador, no tengo constancia de que en juegos presenciales se aplique.

Sin embargo, si es cierto que en algunos juegos de Blackjack online se han llegado a aplicar esta regla o variaciones de ésta (6-Card, 7-Card) y como este proyecto abarca una simulación de partidas en formato online, se aplicará.

Para saber más sobre las reglas del Blackjack en otro tipo de modalidades o directamente ver cuáles han sido mis fuentes de información, recomiendo visitar [59].

2.1.3 Estrategias y Análisis Probabilísticos

2.1.3.1 Estudio Matemático del Blackjack

Para analizar el blackjack a nivel matemático debemos saber que se juega con la baraja francesa, la cual posee 52 cartas, donde:

- 4 son Ases (tienen valor de 1 u 11).
- 16 son 10 o figuras (tienen valor de 10).
- Las 32 restantes son números del 2 al 9.

Por lo que podemos sacar algunas conclusiones:

- $\frac{4}{13}$ de la baraja tienen un valor de 10.
- $\frac{1}{13}$ son Ases. Por tanto, ¿cuál es la probabilidad de obtener un blackjack? Para lograrlo necesitamos un As de los cuatro posibles y una carta con valor 10 de las 16 existentes, calculemos dicha probabilidad:
  - Necesitamos sacar una combinación de 2 cartas de las 52 posibles, es decir, combinaciones de 52 elementos cogidos de 2 en 2:

$$\frac{4*16}{C(52,2)} = \frac{64}{1326} = 4.8\,\% \tag{2.1}$$

Probabilidad de pasarnos de 21

Para calcular la probabilidad de pasarnos de 21, debemos calcular la probabilidad de pasarnos con cualquier mano, y para eso necesitamos ir mano por mano. Imaginemos que nuestra mano suma 12, al pedir una carta más, se nos plantean todos estos casos, ilustrados en Figura 2.1: Como

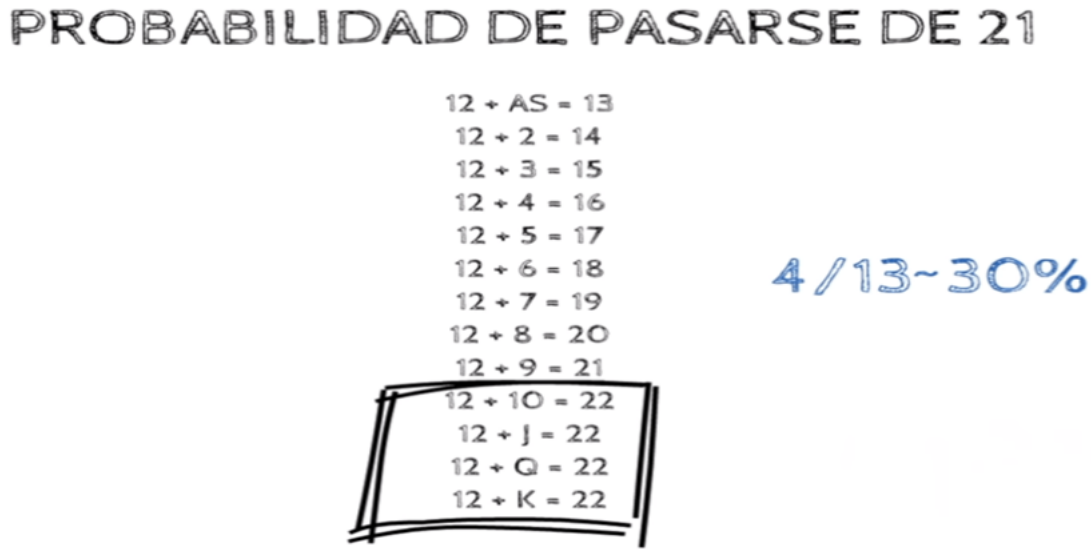


Figura 2.1 Probabilidad de pasarnos de 21 teniendo un 12. Fuente: [45].

vemos a priori hay ***4 casos de 13*** donde superamos 21. Esto es aproximadamente un 30 %, lo que significa que con 12 sólo en el 30 % de las veces que pidamos carta nos pasaremos de 21.

El cálculo de esta probabilidad es una simplificación porque realmente tendríamos que tener en cuenta que las cartas que conforman el 12 varían ligeramente la probabilidad de la carta que hemos pedido. Si calculamos lo mismo para todas las manos posibles (y de manera exacta) obtenemos las siguientes probabilidades que se muestran en la Figura 2.2, dónde a la izquierda tendremos el valor actual de las cartas y a la derecha la probabilidad que tendríamos de pasarnos de 21 al pedir carta:

Figura 2.2 Probabilidad de pasarnos de 21. Fuente: [45].

Probabilidad de que el Crupier se pase de 21

Como acabamos de ver, si no queremos perder la mayoría de veces debemos pedir carta, siempre y cuando nuestra mano sume menos de 14. La pregunta aquí radicaría entonces en si poniéndonos un límite de 14 tenemos opciones de ganar o no.

Si tenemos en cuenta que la banca siempre pide carta hasta alcanzar la suma de 17 o más (si es Blackjack Americano ese límite puede variar), plantarse con un valor menor de 17 solo te haría ganador en caso de que la banca se pasará de 21. Es decir, si el crupier no se pasa de 21, su mano siempre va a tener un valor mínimo de 17 por lo que aquellos que se plantaron con menos de 17 perderán su apuesta.

Pues bien, según el estudio matemático realizado en [80], las ***probabilidades de que el crupier se pase*** en función de la carta visible que tiene, son las ilustradas en la Figura 2.3: Si nos fijamos,

Figura 2.3 Probabilidad de que el Crupier se pase de 21. Fuente: [45].

las cartas con un valor bajo dan una mayor probabilidad de que se pase al crupier pero en ningún caso dicha probabilidad supera el 50%. Haciendo el promedio de todas llegamos a la conclusión de que ***el crupier se pasa en un 28.35% de las veces*** ($\frac{368.67}{13}$) lo que significa que en el ***71.65%*** de las veces que te plantes con menos de 17 vas a perder. Además, hay que tener en cuenta que el

crupier siempre esperará a que el jugador termine su jugada antes de proceder a jugar él lo que le otorga una ventaja, pero ¿cuánta?.

Veamos, si el jugador ***se pasa de 21***, pierde su apuesta, ***independientemente de lo que haga el crupier***. Inclusive si el crupier se pasase también de 21 ya que el jugador se eliminó primero. Eso hace que la banca gane en un escenario que a priori sería un empate, y esto ocurre el ***7.9% de las veces que se juega***. Eso es aproximadamente un 8 % de ventaja sobre el jugador que finalmente se ve reducido a un ***5.6%*** si tenemos en cuenta que el blackjack se paga en una relación de 3-2. Esto es lo que produce ***la principal ventaja del casino***, la victoria en los "empates".

2.1.3.2 Estrategia Básica

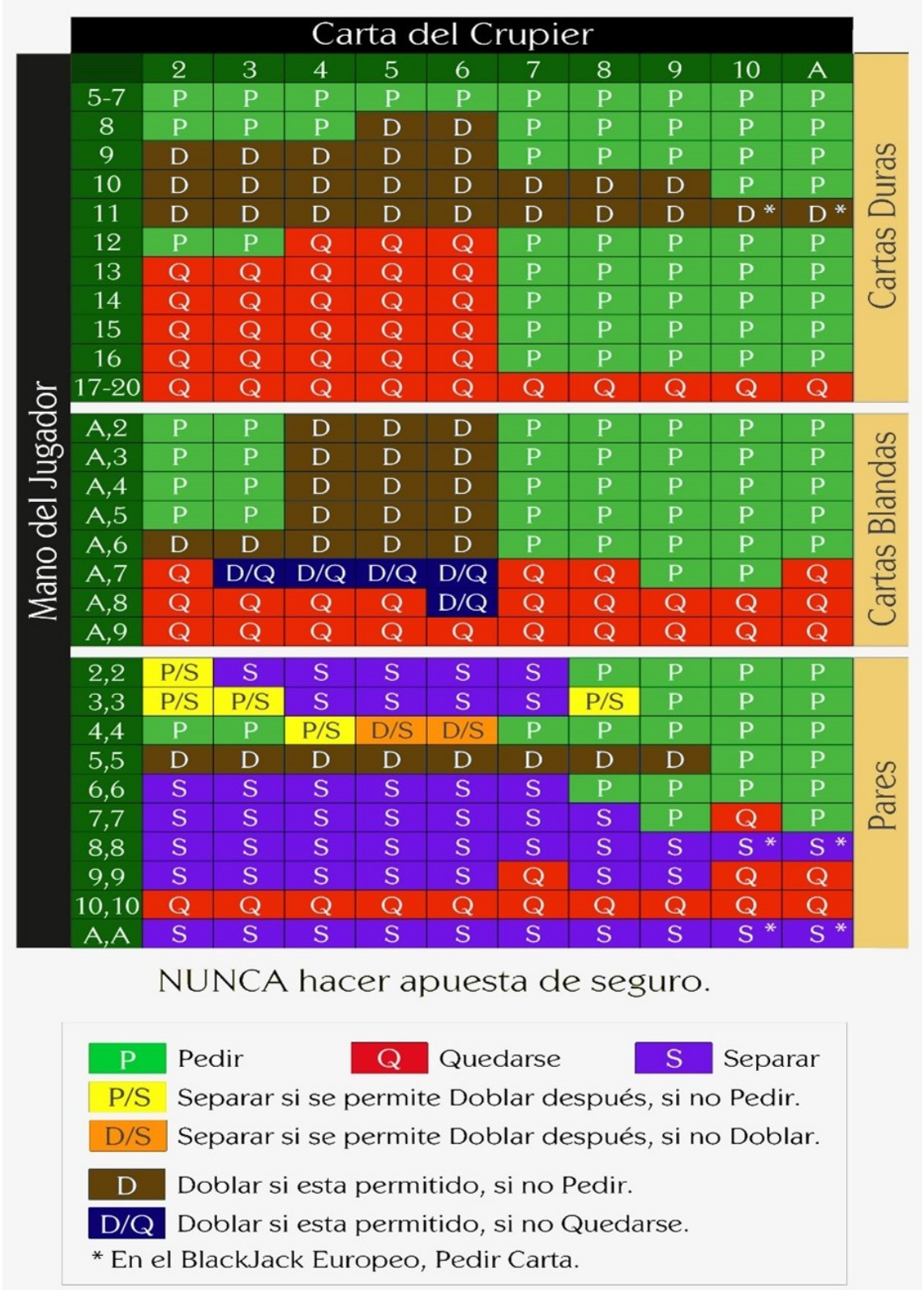
Esta estrategia se basa en ***optimizar las decisiones del jugador*** de tal manera que la acción que elija sea la óptima en cuanto a ganancia de entre todas las acciones disponibles a tomar. Esto nos brinda a priori la ***máxima ganancia posible a largo plazo***.

Se logra creando un modelo de probabilidad que, basado en la mano del jugador y considerando la carta del crupier, determina el momento óptimo para tomar una decisión específica, ya sea en manos duras o blandas, como se ha explicado anteriormente.

En términos matemáticos, hablamos de que se está utilizando una función recursiva junto con valores fijos y una tabla de probabilidades para las diferentes sumas que puede tener el crupier. Esta función calcula la ganancia esperada al quedarse con las cartas actuales o al pedir una carta más, considerando todas las posibles cartas que el crupier podría obtener. Esto da como resultado una tabla que indica con precisión y fundamentos matemáticos qué acción es más beneficiosa en cualquier situación entre el jugador y el crupier. Esta metodología proporciona pautas que se aplican a la Estrategia Básica para optimizarla.

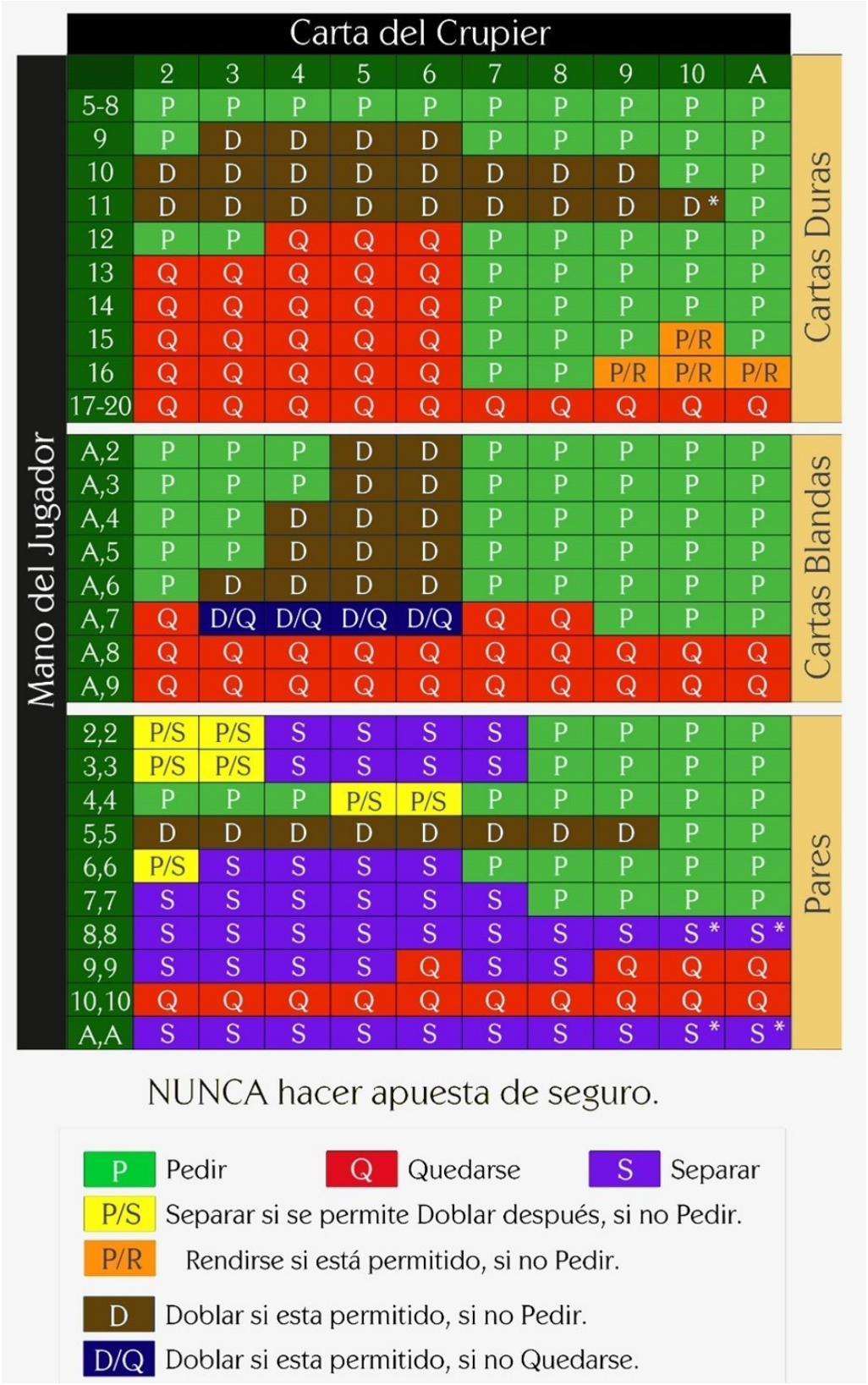
Dichas pautas se muestran en las siguientes figuras: Figura 2.4 y Figura 2.5, ambas imágenes se han obtenido de la siguiente página [45]. Como detalle hay que fijarse que en las manos "***duras***" ***a partir de 17*** nos da instrucciones de plantarnos, lo que significa que independientemente de la carta del crupier, las matemáticas nos dicen que la ganancia esperada será siempre más alta si nos plantamos (razón del límite de las manos "duras" a 17). Además, también nos fijamos que el jugador debería de doblar o dividir sus cartas más a menudo cuando la mano del crupier es "***blanda***" que si fuese "***dura***".

Para acabar con estas imágenes, nos podremos fijar que nunca se debe apostar a seguro ya que es ínfimamente probable que el crupier saque blackjack cuando tiene un As como primera carta.



	2	3	4	5	6	7	8	9	10	A
5-7	P	P	P	P	P	P	P	P	P	P
8	P	P	P	D	D	P	P	P	P	P
9	D	D	D	D	D	P	P	P	P	P
10	D	D	D	D	D	D	D	D	P	P
11	D	D	D	D	D	D	D	D	D *	D *
12	P	P	Q	Q	Q	P	P	P	P	P
13	Q	Q	Q	Q	Q	P	P	P	P	P
14	Q	Q	Q	Q	Q	P	P	P	P	P
15	Q	Q	Q	Q	Q	P	P	P	P	P
16	Q	Q	Q	Q	Q	P	P	P	P	P
17-20	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
A,2	P	P	D	D	D	P	P	P	P	P
A,3	P	P	D	D	D	P	P	P	P	P
A,4	P	P	D	D	D	P	P	P	P	P
A,5	P	P	D	D	D	P	P	P	P	P
A,6	D	D	D	D	D	P	P	P	P	P
A,7	Q	D/Q	D/Q	D/Q	D/Q	Q	Q	P	P	Q
A,8	Q	Q	Q	Q	D/Q	Q	Q	Q	Q	Q
A,9	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
2,2	P/S	S	S	S	S	S	P	P	P	P
3,3	P/S	P/S	S	S	S	S	P/S	P	P	P
4,4	P	P	P/S	D/S	D/S	P	P	P	P	P
5,5	D	D	D	D	D	D	D	D	P	P
6,6	S	S	S	S	S	S	P	P	P	P
7,7	S	S	S	S	S	S	S	P	Q	P
8,8	S	S	S	S	S	S	S	S	S *	S *
9,9	S	S	S	S	S	Q	S	S	Q	Q
10,10	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
A,A	S	S	S	S	S	S	S	S	S *	S *

Figura 2.4 Estrategia Básica del Blackjack para 1 Mazo. Fuente: [45].



	2	3	4	5	6	7	8	9	10	A
5-8	P	P	P	P	P	P	P	P	P	P
9	P	D	D	D	D	P	P	P	P	P
10	D	D	D	D	D	D	D	D	P	P
11	D	D	D	D	D	D	D	D	D *	P
12	P	P	Q	Q	Q	P	P	P	P	P
13	Q	Q	Q	Q	Q	P	P	P	P	P
14	Q	Q	Q	Q	Q	P	P	P	P	P
15	Q	Q	Q	Q	Q	P	P	P	P/R	P
16	Q	Q	Q	Q	Q	P	P	P/R	P/R	P/R
17-20	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
A,2	P	P	P	D	D	P	P	P	P	P
A,3	P	P	P	D	D	P	P	P	P	P
A,4	P	P	D	D	D	P	P	P	P	P
A,5	P	P	D	D	D	P	P	P	P	P
A,6	P	D	D	D	D	P	P	P	P	P
A,7	Q	D/Q	D/Q	D/Q	D/Q	Q	Q	P	P	P
A,8	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
A,9	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
2,2	P/S	P/S	S	S	S	S	P	P	P	P
3,3	P/S	P/S	S	S	S	S	P	P	P	P
4,4	P	P	P	P/S	P/S	P	P	P	P	P
5,5	D	D	D	D	D	D	D	D	P	P
6,6	P/S	S	S	S	S	P	P	P	P	P
7,7	S	S	S	S	S	S	P	P	P	P
8,8	S	S	S	S	S	S	S	S	S *	S *
9,9	S	S	S	S	Q	S	S	Q	Q	Q
10,10	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
A,A	S	S	S	S	S	S	S	S	S *	S *

Figura 2.5 Estrategia Básica del Blackjack para Varios Mazos. Fuente: [45].

A continuación, en la Tabla 2.1, procederemos a mostrar la rentabilidad esperada de la primera ronda con la Estrategia Básica Óptima, para las 4 variantes de reglas para la separación de parejas, donde el ***DAS***[1] y la reseparación (***resplit***[2]) están permitidas. En la Tabla 2.1, tenemos representados los rendimientos esperados al aplicar cada tipo de variante (indicada en la columna superior), dónde tener un número negativo significa pérdidas económicas esperadas y un número positivo, ganancias económicas esperadas. A su vez, los valores de los rendimientos están calculados para distintos números de mazos. En la Figura 2.6 tenemos representados en formato de líneas rectas

Tabla 2.1 Estrategia Básica Óptima: Rendimiento Esperado del jugador frente al Número de Barajas. Fuente: [78].

Nº Mazos	No DAS, no resplit	No DAS, resplit	DAS, no resplit	DAS, resplit	Incremento de rendición
∞	-0.6902	-0.6510	-0.5704	-0.5904	+0.0932
8	-0.6901	-0.5730	-0.4877	-0.4310	+0.0825
7	-0.5974	-0.5617	-0.4758	-0.4197	+0.0810
6	-0.5819	-0.5468	-0.4599	-0.4047	+0.0790
5	-0.5601	-0.5258	-0.4378	-0.3837	+0.0764
4	-0.5274	-0.4943	-0.4046	-0.3523	+0.0726
3	-0.4731	-0.4420	-0.3495	-0.3000	+0.0662
2	-0.3621	-0.3349	-0.2368	-0.1930	+0.0553
1	-0.0147	+0.0018	+0.1143	+0.1419	+0.0236
Pendiente	0.336	0.323	0.344	0.325	

los resultados de la Tabla 2.1, expliquemos entonces el contenido de dicha figura. La gráfica lo que representa es el rendimiento esperado por el jugador frente al número inverso de barajas, por tanto tenemos en el eje X el número inverso de barajas, llamado con 1/D (D=número de mazos) y en el eje Y, el rendimiento esperado. Tenemos 4 líneas que representarán cada una a cada variable explicada en la tabla, aplicando un mapeo según los valores obtenidos, sería:

- Recta verde => DAS, no resplit (es la mayor pendiente que tenemos).
- Recta amarilla => No DAS, no resplit.
- Recta roja => DAS, resplit.
- Recta azul => No DAS, resplit.

Entonces, echando un vistazo a la Figura 2.6 podríamos sacar las conclusiones de qué aplicar DAS sin resplit nos proporcionaría un impacto positivo más directo conforme menor número de mazos se juega en la partida y no aplicar DAS pero si el resplit, el que peor impacto nos generaría.

Para unas conclusiones en más detalle, se recomienda al lector visitar el Capítulo 7.2.2 del libro citado en la bibliografía: [78].

Por lo tanto, la Estrategia Básica Óptima en un juego de un solo mazo tiene probabilidades casi iguales, ligeramente positivas o negativas dependiendo de las reglas para las parejas divididas.

Resumiendo, con esta estrategia pasaríamos del ***5.6%*** anterior a un ***porcentaje inferior al 1%*** para la banca que oscilará dependiendo de las reglas de la mesa y de las barajas que se utilicen. Tenemos que entender que aunque se trate de ***una estrategia óptima no implica que necesariamente sea una estrategia ganadora***, pues el balance de nuestra ventaja pese a ser pequeño sigue siendo negativo. Y esto se debe principalmente a que las cartas altas del crupier disminuyen en

[1] DAS: Doblar después de dividir, se refiere a qué una vez separas tus cartas iguales puedas doblar tu apuesta.

[2] Resplit: volver a separar tus cartas después de una separación, puede darse el caso que una vez separas tus cartas en una de ellas vuelva a caer la misma carta, entonces podrías volver a separar.

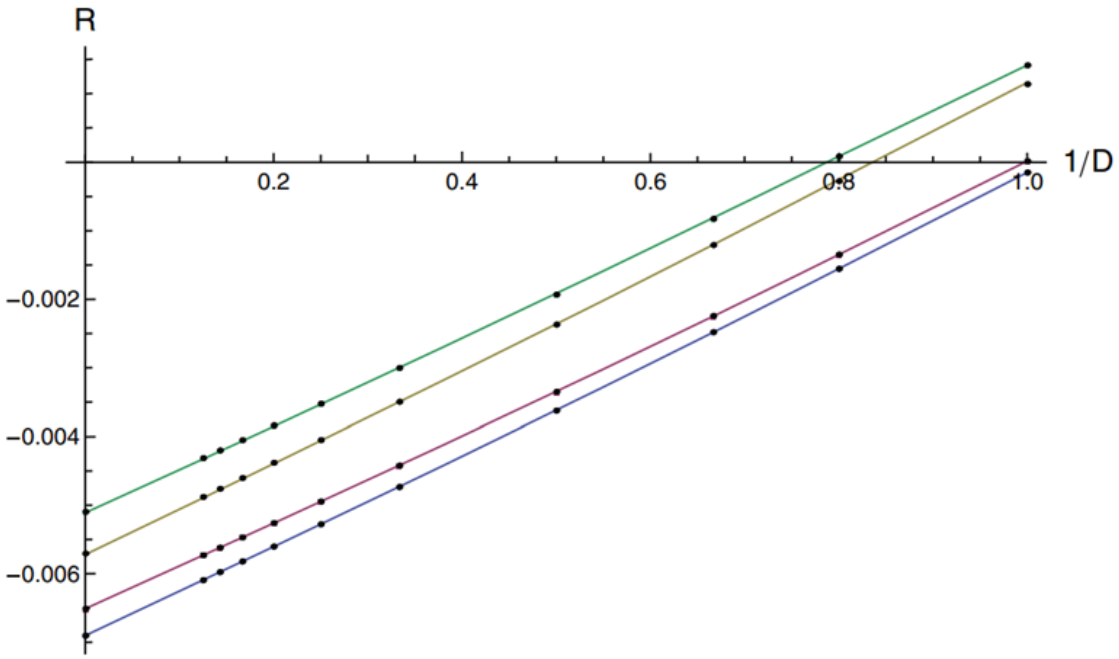


Figura 2.6 Rendimiento Esperado: Ronda Barajada frente al Número Inverso de Barajas. Fuente: [78].

mayor medida nuestra ventaja que las cartas bajas, por lo que aunque tomemos la mejor decisión, no ganaremos necesariamente, aunque si jugaremos de la mejor manera posible.

Si se quiere profundizar viendo un marco analítico y el retorno esperado desde un punto más matemático como el estudio de las fórmulas usadas, recomiendo visitar el Capítulo 7.1 del libro [78] para el juego con un número elevado de mazos y el Capítulo 7.2 del libro [78] para el juego con un número pequeño de mazos.

2.1.3.3 Conteo de Cartas

Hasta ahora se han señalado varios elementos de información que ayudan al jugador a tomar decisiones. El valor total de las cartas de su mano y el valor de la carta descubierta del repartidor juntos conducen a la Estrategia Básica.

La información más importante que aún no se ha utilizado son los valores de las cartas repartidas en rondas anteriores. Si se pudieran rastrear, el jugador tendría una idea de la composición de la baraja restante y de si es más o menos favorable para él. Explotar la información contenida en las cartas repartidas en rondas anteriores es la base del llamado "conteo de cartas".

El efecto que subyace a la importancia del recuento de cartas es que la composición de la baraja fluctúa a medida que se reparten las cartas. Si una baraja ha agrupado los cincos principalmente en la primera mitad, la segunda mitad ofrecerá una probabilidad inferior a la media de sacar otro. Además, las desviaciones de las probabilidades respecto a la media aumentarán a medida que se repartan más cartas.

El éxito del conteo de cartas en el blackjack se debe al hecho de que el rendimiento esperado del jugador en una mano determinada depende sensiblemente de las probabilidades de esa mano. Un jugador que puede calibrar los cambios en el retorno o rendimiento esperado, puede entonces ajustar el tamaño de su apuesta en consecuencia.

Vamos a tratar dos cuestiones importantes para desarrollar un método práctico de recuento de cartas:

1. Decidir qué indicadores de la composición de la baraja son significativos y están al alcance de nuestras capacidades. Por lo tanto, el reto reside en encontrar un esquema simple que nos permita conservar el mayor poder predictivo posible (afortunadamente, con un registro se puede conseguir).
2. Identificar una regla para ajustar las apuestas. Aunque es necesario aumentar las apuestas durante los períodos de rentabilidad favorable, la clave estará en encontrar una estrategia óptima para cuántas fichas apostar, para que nos permita equilibrar la recompensa con el riesgo.

Recuentos Lineales

Aunque nadie sea capaz de contar el valor de cada carta por separado y simultáneamente, es bastante factible rastrear una combinación lineal de esos números. Aquí es donde introducimos el concepto de "***vector de recuento***", un conjunto de diez números indexados del 1 al 10. Llamaremos "elemento" a cada número del conjunto, y su índice corresponde al valor de la carta.

En la siguiente tabla (Tabla 2.2), veremos todos los vectores de recuento existentes (al menos los más reconocidos), ordenados por complejidad: En esta tabla tenemos en la fila superior una lista de número que identifican el tipo de cartas, o más bien, el valor que tienen en el Blackjack (por ello existe una columna sola para el As y una para el 10 la cual abarcaría 10, J, Q y K). Los números de las filas siguientes de la tabla son los puntos asignados a la carta de la fila superior.

Los distintos niveles de complejidad se diferencian por los colores de sus celdas, empezando con el amarillo-verdoso como nivel 1, azul-verdoso como nivel 2, verde nivel 3, naranja nivel 4 y fucsia nivel 5 (dentro de celdas de mismos colores no están ordenados cómo tal).

A continuación, introducimos el "***recuento corriente***". En cada barajada, el recuento corriente se reinicia a 0. Si la primera carta repartida a partir de entonces tiene valor v, el recuento corriente se incrementa con el valor que tiene el elemento x del vector de recuento. Si la segunda carta repartida tiene valor k, el recuento corriente se incrementa de nuevo en el elemento k. Con cada carta siguiente, la cuenta incrementa en el elemento del vector de recuento correspondiente al valor de la carta repartida, cómo ejemplo tendríamos el algoritmo ***Hi-Lo*** donde la cuenta se incrementa en 1 si la carta es "alta" o se decrementa en 1 si la carta es "baja" (esto se verá más en profundidad después en el apartado propio de el algoritmo).

Por último, introducimos lo que la mayoría denomina el "***recuento real***", o recuento por baraja. El recuento real es el recuento corriente dividido por el número de barajas que quedan en la baraja. Aunque estimar esta proporción mentalmente, sobre la marcha, es obviamente una complicación añadida y una molestia, no es tan difícil y es imprescindible para calibrar el rendimiento esperado.

Afortunadamente, no es necesario conocer con exactitud el número de barajas restantes; basta con una estimación aproximada. Es de utilidad saber la siguiente información: un jugador de media roba 2.7 cartas al jugar su mano, mientras que el Crupier roba de media 2.8 cartas. Por lo tanto, una mesa con un solo jugador consume 5.5 cartas por ronda de media, o unas 9,5 rondas por cada baraja de 52 cartas.

El jugador puede simplemente llevar la cuenta del número de rondas jugadas desde la última barajada para aproximarse al número de barajas que quedan por repartir. En ejemplo:

- Una partida de seis mazos contiene cartas suficientes de media para unas 57 rondas, por lo que 19 rondas después de barajar corresponden a una baraja con unos cuatro mazos restantes.

Aún más sencillo, algunos jugadores experimentados calculan las barajas restantes simplemente mirando la altura de la pila de descarte.

Elegir un Vector de Recuento

Actualmente existen una gran cantidad de vectores de recuento con distintos niveles de dificultad, de los cuales se han mostrado los más reconocidos en la Tabla 2.2.

Tabla 2.2 Sistemas de conteos de cartas más populares en orden de complejidad. Fuente: [78] y [24].

NOMBRE	ACRÓNIMO	2s	3s	4s	5s	6s	7s	8s	9s	10s	As
Complete Point Count	**High-Low / Hi-Lo**	1	1	1	1	1	0	0	0	-1	-1
Revere Plus-Minus	**Revere +/-**	1	1	1	1	1	0	0	-1	-1	0
Knock-Out	**K-O**	1	1	1	1	1	1	0	0	-1	-1
Silver Fox		1	1	1	1	1	1	0	-1	-1	-1
Canfield Expert		0	1	1	1	1	1	0	-1	-1	0
Hi-Optimum I	**Hi-Opt I**	0	1	1	1	1	0	0	0	-1	0
Uston Advanced Plus/Minus	**Uston APM**	0	1	1	1	1	1	0	0	-1	-1
Keep It Simple 2	**KISS 2**	0/1	1	1	1	1	0	0	0	-1	0
Keep It Simple 3	**KISS 3**	0/1	1	1	1	1	1	0	0	-1	-1
Red Seven		1	1	1	1	1	0/1	0	0	-1	-1
Advanced Omega II		1	1	2	2	2	1	0	-1	-2	0
Canfield Master		1	1	2	2	2	1	0	-1	-2	0
Zen Count		1	1	2	2	2	1	0	0	-2	-1
High Optimum II	**Hi-Opt II**	1	1	2	2	1	1	0	0	-2	0
Unbalanced Zen II	**UPZ II**	1	2	2	2	2	1	0	0	-2	-1
Mentor		1	2	2	2	2	1	0	-1	-2	-1
Revere Point Count	**RPC**	1	2	2	2	2	1	0	-1	-2	-2
Uston Advanced Point Count	**Uston APC**	1	2	2	3	2	2	1	-1	-3	0
Uston Strongest & Simplest	**Uston SS**	2	2	2	3	2	1	0	-1	-2	-2
Revere 14 Count		2	2	3	4	2	1	0	-2	-3	0
Revere Advanced Point Count	**RAPC**	2	3	3	4	3	2	0	-1	-3	-4
Wong Halves		0.5	1	1	1.5	1	0.5	0	-0.5	-1	-1

De todos los vectores de recuento lineales de la tabla, los más usados suelen ser los de complejidad nivel 1 debido al equilibrio de sencillez y rendimiento que obtienes con ellos. Por tanto, los que trataremos más a fondo en este apartado serán los denominados “Hi-Lo” y “Hi-Opt I”, aunque también los compararemos con otros más complejos de la tabla para comprobar que tampoco difieren mucho en los rendimientos esperados como jugador. Pero primero, realizaremos un ejemplo de cómo se llevaría el recuento con ambos vectores:

1. Considere la observación de las cartas 3, 10, 5 y 7 repartidas sucesivamente después de barajar.

2. El ***recuento corriente*** que llevaríamos sería +1 -1 +1 +0 = +1. En ambos esquemas se asigna +1 a los valores de cartas de 3 a 6, y -1 al valor 10; los valores 7 a 9 se ignoran.

Ambos vectores de recuento sólo difieren en si se presta atención o no a los valores 1 (el As) y 2. Dado que el esquema "Hi-Opt I" permite al jugador ignorar más valores al mantener su vector de recuento, es claramente más sencillo que "Hi-Lo". Este último, sin embargo, resulta ser una aproximación más exacta para calibrar el rendimiento esperado.

El "***betting correlator***" (***BC***) se utiliza como una importante medida para evaluar la eficacia del vector de recuento aproximado. El vector de recuento óptimo tiene un ***BC*** de exactamente 1, y todo vector de recuento aproximado tendrá un ***BC*** inferior a uno. Cuánto más se acerque el ***BC*** a uno, mejor será la aproximación.

En la Tabla 2.3 veremos una lista de distintos vectores de recuento aproximado y sus respectivos BC, también mostraremos los valores que toman en sus vectores de recuento, como se ha visto con "Hi-Lo" y "Hi-Opt I": Como se puede observar en la tabla, "***Hi-Lo***" y "***Hi-Opt***" son los que peor

Tabla 2.3 Otros vectores de recuento y sus BC comparados al óptimo. Fuente: [78].

Valor Carta \ Vector Recuento	2	3	4	5	6	7	8	9	10	As	BC
Optimum (vb)	+0.82	+0.94	+1.21	+1.52	+0.98	+0.57	-0.06	-0.42	-1.07	-1.28	1.000
Ultimate	+5	+6	+8	+11	+6	+4	0	-3	-7	-9	0.998
Halves I	+1	+1	+1	+1.5	+1	+0.5	0	-0.5	-1	-1.5	0.994
Halves II	+0.5	+1	+1	+1.5	+1	+0.5	0	-0.5	-1	-1	0.992
Revere	+0.5	+1	+1	+1	+1	+0.5	0	0	-1	-1	0.975
Hi-Lo	+1	+1	+1	+1	+1	0	0	0	-1	-1	0.967
Hi-Opt	0	+1	+1	+1	+1	0	0	0	-1	-1	0.874

BC poseen. Sin embargo, también poseen los vectores de recuento más simples. Por lo tanto, se puede afirmar con confianza que "***Hi-Lo***" se destaca como uno de los mejores debido al equilibrio que logra entre la facilidad de uso y la aproximación al óptimo. Por tanto, al ser el vector "***Hi-Lo***" el que mejor equilibrio muestra y además ser uno de los principales objetivos a añadir en el programa como algoritmo de conteo de cartas, lo veremos más en profundidad a continuación.

Sistema Hi-Lo

En este vector de recuento la idea es ***controlar las cartas altas de la baraja***, que son las únicas que pueden conformar el ***blackjack***, de tal manera que si sabemos cuántas cartas altas han aparecido sabemos si quedan muchas o pocas por aparecer, y así podemos saber qué probabilidades tenemos de obtener el ***blackjack***.

Para conocer esa ventaja necesitamos tener un control sobre las cartas que van apareciendo y para ello usaremos los datos mostrados en la Tabla 2.3 y asociaremos de dicha manera los valores a las cartas que vayan apareciendo.

Dado que en este sistema existe la misma proporción de cartas altas y bajas, cinco en cada rango, una cuenta positiva o negativa nos va a indicar directamente cuando quedan o no más cartas altas en la baraja. De la siguiente manera:

- Si nuestra cuenta es ***negativa*** es porque habremos contado más cartas altas, y quedarán por tanto ***menos cartas altas por aparecer***, algo que no nos interesa.
- Si nuestra cuenta es ***positiva*** es porque habremos contado más cartas bajas, y quedarán por tanto ***más cartas altas por aparecer***, algo que nos interesa.

Por tanto, nuestro objetivo sería ***acumular en la cuenta el número positivo más alto posible*** que nos estuviera ofreciendo las mayores probabilidades de lograr un ***blackjack*** (o al menos lograr 21 sin pasarnos). Si conseguimos llevar a cabo con éxito este sistema podríamos modificar la estrategia

básica a nuestro interés, mostradas en la Figura 2.4 y la Figura 2.5, para doblar la apuesta y obtener más ganancias.

Hay que tener en cuenta además que como las cartas bajas y altas están equilibradas en número, una vez se finalice la baraja la cuenta total debe de dar 0 obligatoriamente.

En el caso de jugar con ***varias barajas*** hay que tener en cuenta que no es lo mismo una suma con 4 barajas pendientes que la misma suma con 1. Por eso, ***la suma actual tiene que ser dividida entre el número de barajas que se cree que quedan por salir, y esta será la cuenta real*** (procedimiento comentado en el apartado 2.1.3.3).

Pese a que podemos tener una ventaja con este sistema todo esto sucede tras ***miles y miles de manos*** y nos da una ***ventaja de décimas*** de porcentaje.

Para finalizar este apartado mencionaremos una serie de medidas que han acabado tomando los casinos para contrarrestar a los contadores de cartas. Veamos un resumen de cuáles y cómo han afectado al juego:

1. ***Aumentar el número de barajas***: esto complica llevar la cuenta bastante más como hemos visto anteriormente ya que nuestra cuenta habría que dividirla entre el número de barajas restantes para obtener el recuento real.
2. ***Introducción de mezcladores y elementos automáticos***: usados para mezclar las cartas antes de que termine el juego, obligando al jugador contador de cartas a tener que reiniciar su cuenta y por tanto perder toda la ventaja que había logrado.

Si se quiere conocer el marco analítico del conteo de cartas, los retornos esperados o como optimizar los vectores de recuento recomiendo leer el Capítulo 8 del libro [78]. O si por otro lado se quiere saber cómo jugar las distintas estrategias utilizando el conteo de cartas, habría que hacer una lectura del Capítulo 10 del libro [78].

2.1.3.4 Síntesis y Observaciones

Los apartados anteriores han ilustrado ampliamente su argumento de que el blackjack es un juego que da al jugador muchas opciones y, como resultado, puede optimizarse exhaustivamente. Por tanto, el mensaje más importante que se puede extraer es que la forma de jugar al blackjack dependerá de por qué se está jugando.

Para el jugador recreativo, que simplemente va a pasárselo bien sin querer realizar un esfuerzo mental muy excesivo, recomendaría la estrategia básica sin recuento. Una virtud especial es su sencillez, por tanto, es fácil de aprender (o reaprender) y de utilizar en la mesa.

Para el jugador competitivo, que va a ganar y no le importa realizar un esfuerzo mental, recomendaría que aprendiese a hacer seguimiento de las cartas repartidas y ajustar el tamaño de su apuesta en consecuencia. Básicamente, aprender a contar cartas y apostar en relación a la cuenta que lleve, de esta manera puede llegar a sacarle ventaja al casino (al menos en sentido estadístico).

El procedimiento de recuento de cartas, en resumen, requiere de una serie de pasos que se han visto ya en profundidad en su respectivo apartado (2.1.3.3).

Sin embargo, utilizar una estrategia con un rendimiento positivo no significa necesariamente que el jugador vaya a ganar en una sesión determinada. El rendimiento es una medida del resultado medio de un gran número de rondas. En cualquier número moderado de rondas, las desviaciones de esa media pueden ser grandes. En la Figura 2.7 tenemos representadas dos imágenes, para cada imagen tenemos dos curvas, la que está más a la izquierda (es decir, su centro está más a la izquierda) representa al jugador no contador de cartas y la que está más a la derecha (es decir, su centro está más a la derecha) representa al jugador que cuenta cartas.

Para las distribuciones de resultados de la Figura 2.7a muestran que es probable que cada jugador gane o pierda hasta 20 unidades o más. Aunque los picos (y las medias) de las distribuciones son diferentes (-2 vs. +4), las distribuciones en sí son tal que la distinción entre ellas es apenas

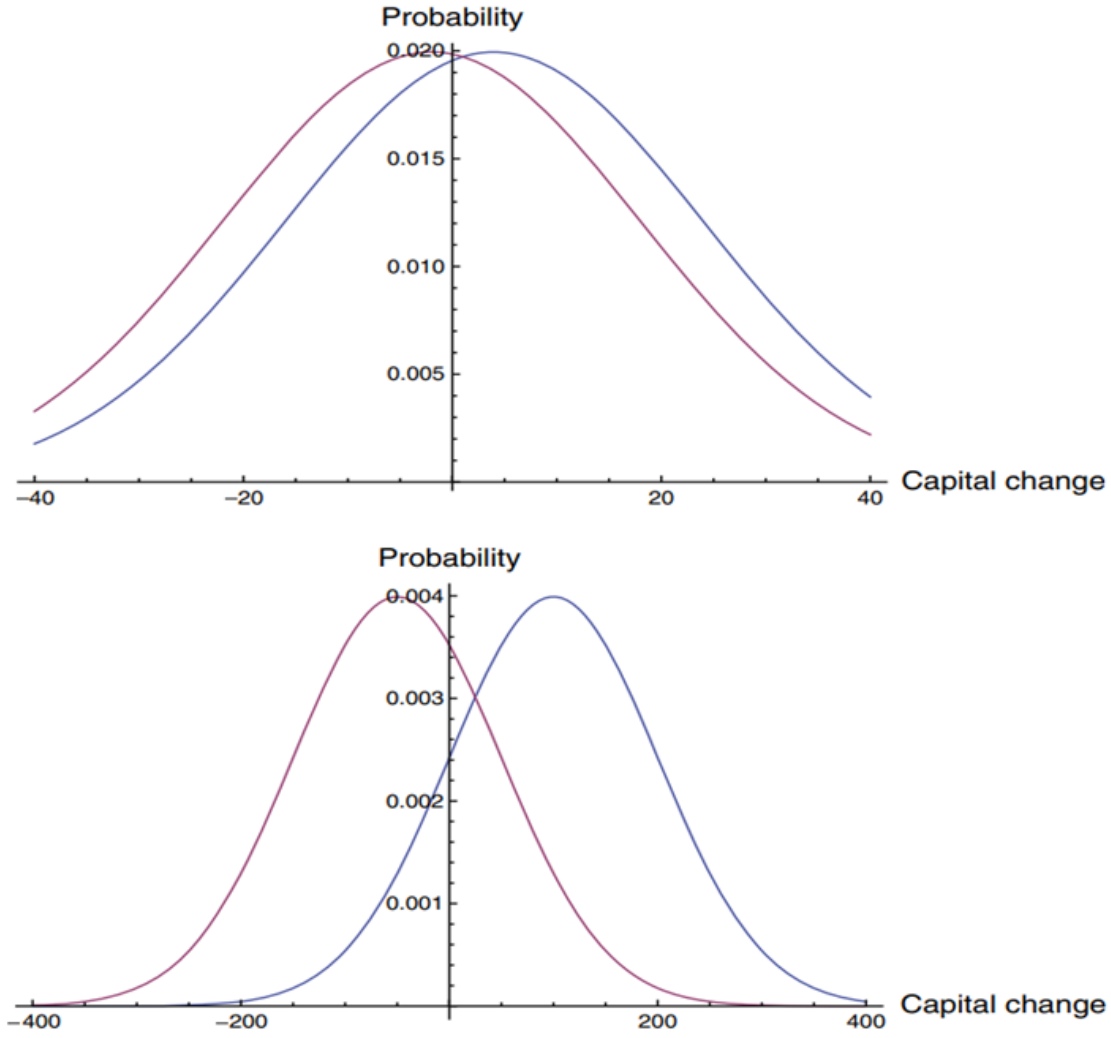


Figura 2.7 Distribución de los Resultados. Figura de arriba (a): partida tras 400 rondas jugadas. Figura de abajo (b): partida tras 10000 rondas jugadas. Fuente: [78].

perceptible. El contador de cartas competitivo pierde en el 44 % de estas sesiones; el no contador gana en el 46 % de ellas.

Un observador que estuviera siguiendo los resultados de los dos jugadores durante esa sesión tendría dificultades para determinar quién estaba jugando con qué estrategia. Identificar al contador de cartas como el jugador con el mejor resultado sería correcto sólo el 57 % de las veces, apenas un poco mejor que una probabilidad de cara/cruz. Además, el observador no sería capaz de distinguir a los dos con mucha certeza hasta después de muchas sesiones. Por ejemplo, después de 10000 rondas, las distribuciones están claramente separadas, como se muestra en la Figura 2.7b; en ese punto, el observador podría asignar estrategias a los jugadores (identificar como el contador de cartas al jugador con mejor rendimiento) con un 82 % de precisión. Aun así, el jugador que cuente cartas de manera óptima no tiene una garantía sustancial de ganar realmente a un jugador que no cuenta cartas, hasta después de miles de rondas; y ni así, la certeza está lejos de ser total.

Sin embargo, cuantas más rondas se jueguen (ya sea en una sesión o en varias sesiones), mayor es la posibilidad de arruinarse. Pero la ruina no convierte la estrategia ganadora con un rendimiento positivo de un jugador competitivo en una estrategia perdedora.

El ***factor de reducción del rendimiento*** debido a la ruina depende de tres parámetros, los cuáles ya se han visto anteriormente más en profundidad por lo que ahora sólo se hará un breve resumen

de ellos:

1. La ***cobertura***, es decir, la relación entre el capital y la apuesta típica, la cual puede controlar más fácilmente el jugador. La probabilidad de ruina se puede reducir aumentando la cobertura (una apuesta alta hace que la ruina sea insignificante). Por el contrario, un jugador recreativo cuyo presupuesto limitado permite sólo una apuesta pequeña, debería reducir sus apuestas, en particular eligiendo una mesa con la apuesta mínima más baja posible.
2. El ***número de rondas jugadas***. En una sesión corta con al menos una cobertura moderada, la ruina es bastante improbable. A medida que se juegan más rondas, el riesgo de ruina aumenta. En casi todas las circunstancias realistas, la ruina será un suceso ocasional que debe aceptarse.

Toda esta información ha sido sacada y mostrada de una manera resumida del Capítulo 6 del libro [78].

2.1.4 Los "Tells" del Crupier

Empezaremos por definir el concepto de "tell" en los juegos de azar antes de explorar cómo se aplican al juego de Blackjack. Los "tell" son señales inadvertidas que los jugadores proporcionan, como manías o actos reflejos, que pueden revelar información sobre la calidad de sus manos o su estrategia. Estas pistas pueden ser sutiles pero pueden ser utilizadas por otros jugadores para inferir la fuerza de la mano de un oponente y tomar decisiones estratégicas en consecuencia.

Los "tell" en el Blackjack

En el caso del blackjack, no estamos jugando contra otros jugadores sino contra la banca, representada por el crupier, que aunque no tiene un gran margen de maniobra (se rige a las reglas estipuladas por el casino), puede que en ciertos casos nos ofrezca algún “tell” respecto a sus cartas.

Partamos de la base de que, en la mayoría de los casos, los crupieres de blackjack ni siquiera miran su carta descubierta hasta que todos los jugadores de la mesa han realizado su acción, por lo que, en estos casos, no nos servirá de mucho detectar un “tell” del crupier.

Del mismo modo, detectar este tipo de información es un trabajo que requiere no sólo mucho tiempo en la mesa para estar seguro de su validez, sino también una sensibilidad muy personal que sólo se perfecciona tras largas sesiones y mucha experiencia.

Si este no es el caso, es mejor abstenerse de aplicarlo y continuar con el juego clásico en la mesa, hasta que se vuelva automático y nos permita prestar atención también a otros detalles. Por lo tanto, si tenemos suficiente experiencia, será crucial detectar algunos indicios específicos del crupier mientras miramos su carta.

¿Qué "tell" puede mostrar un Crupier

Muy a menudo todos estamos acostumbrados a seguir determinados patrones en nuestras acciones. Lo mismo le ocurrirá al Crupier, que tiene que lidiar con una acción particularmente repetitiva y mecánica, tanto que desarrolla (quizás incluso a su pesar) hábito que de alguna manera podemos intentar conocer.

En concreto, tenemos dos grandes posibilidades a analizar en el comportamiento del crupier: la del ***“timing”*** con el que mira su carta y la de la posición en las que la coloca.

El "tell" del momento de mirar la carta cubierta ("timing")

La propia estructura de las cartas hace que haya algunos valores más fáciles de leer que otros. El **10**, por ejemplo, es sin duda el más inmediato, ya que basta con echar un vistazo desde cualquier ángulo para ver el doble dígito impreso en la carta.

Más o menos lo mismo se aplica a las distintas figuras, ya que no tiene que reconocer necesariamente su valor, sino que todas valen un diez. En estos dos casos, es probable que el crupier haga un movimiento muy fugaz y rápido para ver su carta cubierta.

Por el contrario, sabemos que hay cartas que son un poco más complicadas de reconocer. El ejemplo más llamativo (especialmente en blackjack donde los valores son cruciales) es el del as y

el 4, que al levantar sólo el borde de la carta suelen ser muy similares. Pero también ocurre con el 2 o el 5, así como con el 3 o el 8. En todos estos casos, es probable que al crupier no le baste con echar un vistazo rápido a la esquina de la carta, dado además que su posición es desfavorable en términos de inclinación con respecto a la mesa.

Eso sí, todos los crupieres experimentados se habrán formado una rutina de hacer las mismas jugadas a la misma velocidad precisamente para no dar este tipo de información pero, en cualquier caso, puede haber crupieres aún sin esta rutina o con menos experiencia como para dejar escapar un pequeño "tell" en este sentido.

En cualquier caso, por lo tanto, intentaremos observar atentamente el comportamiento del crupier en la fase de «lectura» de sus cartas, quizás verificando de vez en cuando si nuestra información resulta ser correcta a medida que avanza la mano (incluso cuando no estamos directamente en juego).

El "tell" de la posición de las cartas

En efecto, algunos crupieres, sobre todo los que no tienen mucha experiencia, pueden tener la tentación de colocar sus cartas de una manera determinada en función de si van a recibir o no otras cartas en su turno. Se puede observar, por ejemplo, que colocan sus cartas justo a su derecha, con el fin de dejar más espacio para colocar una o más cartas adicionales (es decir, probablemente tienen una carta baja y esperan sacar de la baraja en su turno).

A la inversa, colocarlos más hacia el centro podría indicar un punto ya establecido sin necesidad de añadir más cartas. Sin embargo, se trata de un «tell» muy complejo y raro de encontrar, precisamente porque una de las primeras cosas que aprenden los crupieres es a colocar siempre las cartas de la misma manera para evitar dar este tipo de información. Pero, precisamente, la inexperiencia, el cansancio o simples malas rutinas llevan a detectar también este comportamiento.

Para profundizar en esta información, visitar [58].

2.1.5 Misticismo en las posiciones

Partamos de un supuesto puramente matemático: la posición en la mesa no afecta en absoluto a la probabilidad estadística de obtener un punto u otro. Así que, si estamos jugando simplemente por diversión o al límite aplicando una estrategia básica, no importa realmente qué posición elijamos.

Elegir el lugar en primera posición

Ahora bien, si pensamos un poco, podríamos decir en primer lugar que hay un asiento que probablemente hay que evitar, concretamente el de la primera posición. Como se ha dicho, no cambia nada desde el punto de vista de las posibilidades de ganar, pero sin duda nos encontraremos en cada mano siempre siendo los primeros en hablar, teniendo así menos tiempo que todos los demás en la mesa para pensar en nuestras elecciones.

Este problema es relativo si jugamos por diversión o con una estrategia básica ya asimilada, pero una consideración a tener en cuenta si tal vez somos nuevos en el juego (disponemos de un poco más de tiempo) y sobre todo si, por el contrario, somos contadores de cartas. En ese caso, la elección debería necesariamente recaer en otra posición, la de "*tercera base*".

El mejor puesto: La *tercera base*

El lugar que en la jerga se denomina "*tercera base*" no es otro que el situado a la derecha del Crupier, es decir, el del jugador que será el último en actuar en cada mano antes que la propia banca.

Una posición ventajosa, como se ha dicho, aunque solo sea para disponer de ese poco más de tiempo para decidir nuestra acción (al ser los últimos en hablar), pero aún más rentable si somos de esos jugadores que, de alguna manera, están contando cartas.

En ese caso, de hecho, la ventaja es doble. Por un lado, conoceremos todas las cartas que han salido durante la mano y podremos actualizar el recuento con algo más de información; por otro, la propia posición es logísticamente útil para poder llevar a cabo un recuento preciso sin perder

de vista toda la mesa y sin ningún movimiento particular de la cabeza (lo que suele disparar las alarmas de quienes nos observan).

Sin embargo, también hay que tener en cuenta otra cuestión sobre esta posición. También se suele pensar que es crucial para que toda la mesa gane la mano ya que, al ser el último en jugar antes de la banca, podría o bien sacar la carta correcta que luego haría que la banca se pase, o por el contrario cometer el error decisivo que haga que otros jugadores pierdan.

Esta percepción es obviamente errónea desde un punto de vista estadístico, ya que realmente no hay correlación entre las cosas. Pero como mucha gente no piensa así, y para evitar presiones de más, será mejor evitar sentarse en esa posición si sólo queremos disfrutar de nuestra sesión sin ninguna presión adicional.

Para saber más, visitar [60].

2.2 Inteligencia Artificial

2.2.1 ¿Qué es la Inteligencia Artificial?

Aunque los primeros referentes históricos se remontan a los trabajos de Alan Turing en los años 30 del siglo XX, es en el año 1950 cuando Turing, [79], publica un artículo con el título «Computing machinery and intelligence» en la revista Mind, el cuál podemos visitarlo en [75], donde se hace la pregunta: "¿Pueden las máquinas pensar?". Los fundamentos teóricos de la IA se encuentran en el experimento que propone en dicho artículo y que pasó a denominarse Test de Turing, y mediante cuya superación por una máquina se podía considerar que sería capaz de pasar por un humano en una charla ciega. Este test sigue estando vigente en la actualidad y es motivo de estudios e investigaciones continuas.

Sin embargo, numerosos investigadores e historiadores consideran que el punto de partida de la moderna inteligencia artificial fue el año 1956, cuando los padres de la inteligencia artificial moderna, John McCarty, Marvin Misky y Claude Shannon acuñaron formalmente el término durante la conferencia de Darmouth, como: «la ciencia e ingenio de hacer máquinas inteligentes, especialmente programas de cálculo inteligente».

Es decir, que en propiedad lo que denominamos como "Inteligencia Artificial" son en realidad un conjunto de técnicas, metodologías, herramientas, procesos, etc. que tienen en común lo siguiente:

- Se basan en datos procesables mediante métodos matemáticos.
- Su propósito es escoger la mejor acción posible para conseguir un objetivo determinado, como podemos observar en la Figura 2.8.

Es evidente que el proceso clave es el "***razonamiento***" o procesado de la información que conduce a la toma de decisiones. En este sentido, dichas metodologías se pueden agrupar en dos:

- ***Toma de decisiones*** ("razonamiento"): estas metodologías se caracterizan por transformar la información (entrada de datos) en un modelo conceptual de conocimiento y usar este marco conceptual para hacer inferencias de un punto de partida a otro de llegada ("razonamiento") y determinar la mejor opción posible.
- ***Aprendizaje***: este grupo de técnicas permiten al sistema aprender a resolver problemas que no están claramente definidos o cuya solución no puede ser expresada mediante reglas concretas. Por ejemplo, todas aquellas que tienen que ver con el lenguaje o con la predicción de comportamientos. Las metodologías de aprendizaje a su vez se pueden subdividir en aprendizaje supervisado o no supervisado, los cuales veremos en profundidad en futuros apartados.

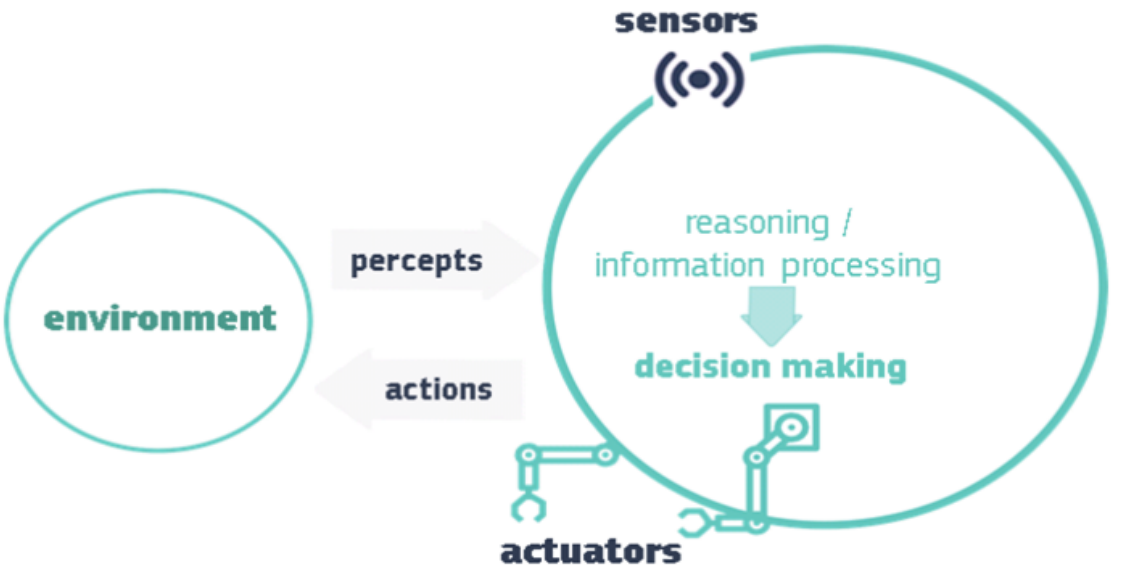


Figura 2.8 Representación esquemática de un sistema de Inteligencia Artificial. Fuente: [14] (página 4).

Es evidente que la mayoría de sistema de IA necesitan ingentes cantidades de datos para procesar las inferencias necesarias para resolver los problemas que se plantean. Por ello, es absolutamente crítico asegurarse de que la calidad de los datos de partida sea óptima y tener en cuenta los siguientes criterios:

- ***Integridad*** de los datos: grado de conservación de los datos originales.
- ***Representatividad***: ¿los datos recogidos representan el problema que se quiere resolver?, ¿hay sesgos en la recogida de datos?.
- ***Unicidad***: ¿son los únicos datos que describen el mismo problema?.
- ***Exactitud***: ¿los datos recogidos describen las variables del problema con exactitud?.

2.2.1.1 Pros y Contras del Uso de la IA

En una publicación, el Parlamento Europeo identifica una serie de oportunidades y desafíos que resumen muy bien la situación actual de las aplicaciones de la Inteligencia Artificial, las cuales veremos de manera resumida en Tabla 2.4, para profundizar recomiendo visitar dicha publicación [21].

Tabla 2.4 Pros y Contras del Uso de la IA. Fuente: [21].

Ventajas	Desventajas
Beneficios de la IA para los ciudadanos.	Infrautilización o uso excesivo de la IA.
Oportunidades para las empresas.	Responsabilidad y Retos de Transparencia.
Oportunidades para los servicios públicos.	Amenazas a los derechos fundamentales y a la democracia.
Refuerzo de la democracia.	Impacto en el empleo.
IA, seguridad y protección.	Riesgos en la seguridad y protección.

2.2.1.2 Paradigmas de Aprendizaje

El ser humano está en constante aprendizaje desde sus primeros años de vida. Gracias a este proceso cada individuo va conformando su propia experiencia y adaptándola a su realidad para usar

conocimientos, habilidades, destrezas o conductas aprendidas en el momento en que lo necesite. Si nos fiásemos de la intuición colectiva y preguntásemos a un grupo de no expertos, probablemente obtendríamos fundamentalmente tres respuestas relacionadas con las conclusiones extraídas sobre el aprendizaje a lo largo de la historia, las cuales serían:

- Aprendemos a través de la interacción con otros humanos, por ejemplo en el colegio o tutorizado por los padres.
- Aprendemos a través de la observación de la realidad.
- Aprendemos a partir de la experiencia y estímulos, ya sean positivos o negativos, de sus acciones previas.

Probablemente todas estas afirmaciones son ciertas, y el aprendizaje se derive de una combinación de todos estos factores. Sin embargo, ¿Cómo aprenden las máquinas? O lo que es lo mismo, ¿cómo aprenden los algoritmos de aprendizaje automático? Para entrenar algoritmos de machine learning, normalmente citamos tres paradigmas de aprendizaje, que están íntimamente relacionados con las tres respuestas anteriores: aprendizaje supervisado, aprendizaje no supervisado, y aprendizaje reforzado. Saber cómo funcionan estos paradigmas es fundamental para moverse en el mundo del machine learning

Aprendizaje Supervisado

Cuando hablamos de aprendizaje supervisado nos estamos refiriendo al proceso de encontrar la relación entre unas variables de entrada y unas de salida. Este aprendizaje surge de enseñar a los algoritmos cuál es el resultado que quieres obtener para una determinada entrada, es decir, de enseñarles ejemplos. Para saber más recomiendo visitar las siguientes páginas [31] y [46]

Aprendizaje No Supervisado

El aprendizaje no supervisado es el paradigma que busca producir conocimiento únicamente de los datos que se proporcionan como entrada, sin explicarle al sistema en ningún momento qué resultado queremos obtener. Si podíamos asimilar el aprendizaje supervisado al aprendizaje realizado por los humanos en clase, o tutorizados por sus padres, el aprendizaje no supervisado es el equivalente al aprendizaje que realizan los humanos a partir de sus observaciones de la realidad, sin ninguna supervisión ajena. Para saber más recomiendo visitar las siguientes páginas [30] y [50].

Aprendizaje Reforzado

A medio camino entre el aprendizaje supervisado y el no supervisado se encuentra el aprendizaje reforzado o por refuerzo ('reinforcement learning' en inglés), el cual sería el análogo en el mundo del machine learning al aprendizaje mediante estímulos. El Reinforcement Learning se basa en hacer aprender a la máquina basándose en un esquema de "premios y castigos", como el perro de Pavlov, para saber más visitar [62]. Su principal particularidad es que es capaz de funcionar sin grandes cantidades de datos de entrenamiento.

Los inicios del aprendizaje reforzado se remontan a los años 50, cuando el matemático estadounidense Claude Shannon creó un ratón artificial, llamado Theseus, se puede ver un vídeo demostrativo en [41], que a través de prueba y error logró aprender a atravesar un laberinto. Utilizando la misma aproximación, en el año 2013 los investigadores de Deep Mind (Google) crearon un algoritmo capaz de aprender a jugar y superar cualquier juego de Atari desde cero sin tener ningún conocimiento previo de las mecánicas del juego, conocido como agent57, para saber más [63]. El aprendizaje reforzado es una de las áreas más excitantes y prometedoras del machine learning en la actualidad. Para saber más recomiendo visitar las siguientes páginas [4] y [47].

2.2.2 Machine Learning

En términos generales, se considera que un agente aprende cuando mejora su rendimiento en una tarea específica con la experiencia. En el contexto del aprendizaje automático, un programa de software analiza datos, construye un modelo basado en ellos y lo valida con nuevas observaciones

para confirmar las hipótesis del modelo. Desde una perspectiva científica, el aprendizaje automático busca codificar la lógica del método científico en software, permitiendo que programas realicen procesos inductivos para adquirir conocimiento.

El machine learning no es simplemente una disciplina dentro de la inteligencia artificial, sino un componente esencial que transforma todo el campo científico al relacionarse con diversas aplicaciones y conectarlas entre sí. Esto permite que algoritmos dedicados a tareas como visión por computadora, reconocimiento de patrones y procesamiento de lenguaje natural puedan desempeñarse en esas áreas debido a la programación específica. Por lo tanto, el machine learning se integra con otras disciplinas de la IA como un nuevo enfoque para construir modelos.

Los vehículos autónomos de Google, Tesla y otros fabricantes se entrenan siguiendo este enfoque, observando y aprendiendo de los conductores humanos en una gran variedad de situaciones en la carretera. Este cambio de paradigma, donde no se programa simplemente un algoritmo para conducir un automóvil, sino que se enseña a un algoritmo a hacerlo, ha sido un factor clave en el rápido crecimiento de la inteligencia artificial en la última década.

Dentro del Machine Learning se ha vuelto bastante usado el método Monte Carlo, el cuál es una técnica matemática para estimar los resultados de sucesos inciertos basándose en la generación de múltiples resultados posibles utilizando distribuciones de probabilidad y valores aleatorios. Suele usarse para predicciones a largo plazo debido a su precisión. A medida que aumenta el número de entradas, el número de predicciones también crece, lo que le permite proyectar los resultados más lejos en el tiempo con más precisión. Funciona mediante la configuración de un modelo predictivo, la especificación de distribuciones de probabilidad para las variables involucradas y la ejecución repetida de simulaciones, permitiéndole generar un rango de posibles resultados. Para saber más sobre el método Monte Carlo, visitar [32] o bien el artículo [35], además de poder un caso de uso de dicho método en este vídeo [67].

Como hemos visto en el apartado anterior, para poder entrenar los algoritmos de machine learning existen tres tipos de aprendizaje. A continuación, veremos en profundidad los algoritmos referentes al Aprendizaje Supervisado, por tanto, recordemos brevemente a qué se refería dicho aprendizaje. El aprendizaje supervisado se define por el uso de conjuntos de datos etiquetados (observaciones y respuestas) que se utilizan para entrenar algoritmos en la realización de dos tareas fundamentales:

- ***Clasificación***: : el objetivo es agrupar o clasificar datos en clases o categorías comunes. El clasificador emplea su conocimiento del mundo (datos de ejemplo) para construir un modelo que le permite asignar etiquetas de clase a los datos de entrada.
- ***Predicción*** (o ***regresión***): el objetivo es encontrar la relación subyacente entre las variables dependientes e independientes que componen los datos de ejemplo. Dicha relación entre entidades constituye el modelo observacional en el que se basan las predicciones.

La separación de los algoritmos en ambos grupos se puede observar en la Figura 2.9.

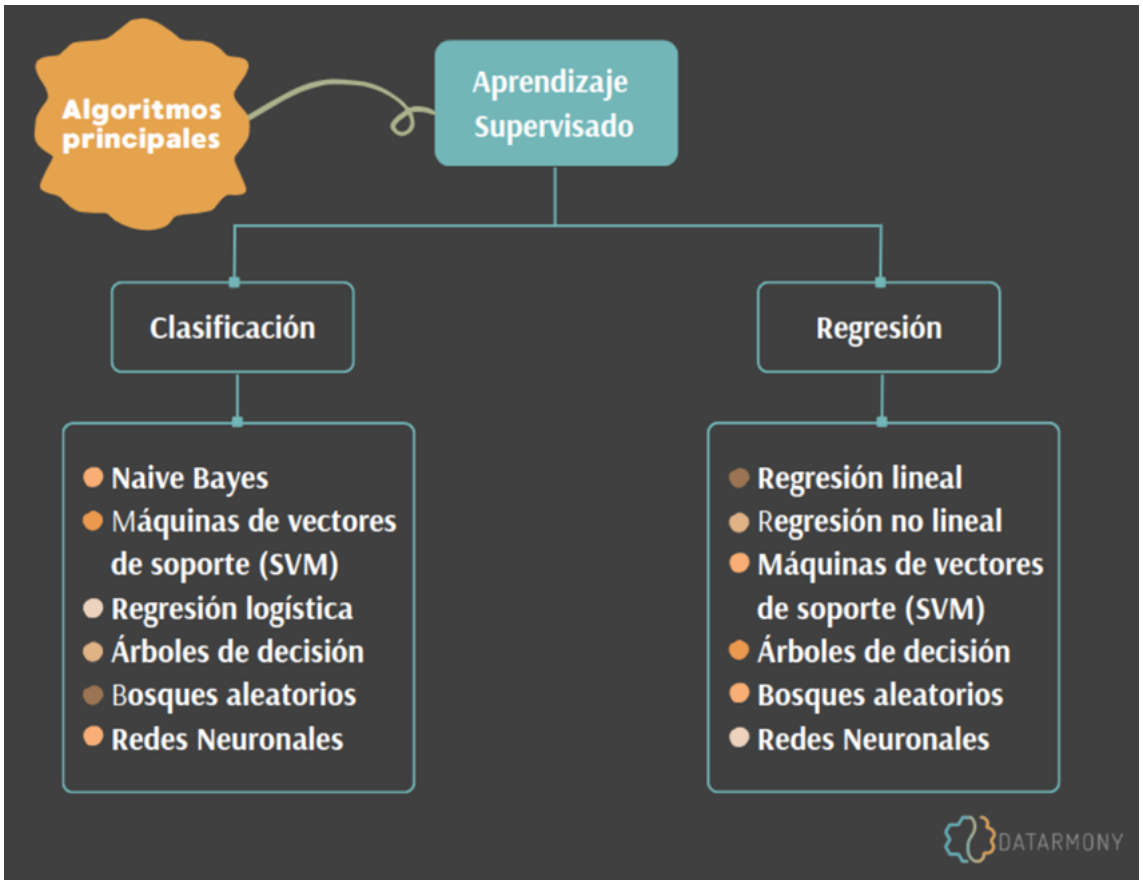


Figura 2.9 Principales algoritmos del Aprendizaje Supervisado dividido por grupos. Fuente: DATARMONY.

2.2.2.1 Regresión Lineal

En estadística, la regresión lineal es un conjunto de herramientas matemáticas que permiten modelar la relación entre una variable dependiente y una o más variables explicativas X asumiendo una dependencia lineal entre ambas. Un modelo que se construye a partir de una única variable explicativa se denomina de regresión lineal simple, y si se configura en base a más de una variable independiente de regresión lineal múltiple.

Ejecutar un análisis de regresión lineal requiere que la variable dependiente sea continua. Por supuesto, la condición básica que impone un modelo de regresión lineal es que la relación esperada entre las variables de estudio sea lineal, de otro modo la capacidad predictiva del modelo será nula.

Los algoritmos de regresión lineal se utilizan generalmente para estimar valores reales a partir de variables continuas. Mediante un modelo de regresión lineal estimaremos la relación entre la variable objetivo y la o las variables dependiente/s a partir del ajuste de los datos a una línea recta o una combinación lineal de factores. Para el caso más sencillo (una sola variable dependiente) dicha relación se expresa matemáticamente como:

$$y = a + X * b \tag{2.2}$$

Donde a y b son los coeficientes (***intersección*** y ***pendiente***) que producen el mejor ajuste de los datos de acuerdo a un criterio o procedimiento matemático. La ***intersección*** representa la posición en la que la recta cruza el eje vertical o valor esperado del resultado cuando el valor del predictor es cero y la ***pendiente*** indica la inclinación de la recta y representa el valor de cambio medio en el resultado; a mayor magnitud de la pendiente mayor tasa de cambio.

Pero, ¿cómo seleccionamos qué parámetros producen el mejor ajuste a los datos? Aquí entra la función de coste o función de pérdida (***loss function***), que utilizamos para definir y medir el error de un modelo. En el caso que nos ocupa la función de coste habitual se conoce como error cuadrático medio (***MSE***). El ***MSE*** calcula la diferencia media al cuadrado entre los valores reales y las predicciones para cada observación. El resultado es un valor numérico que representa el coste o puntuación asociada al conjunto actual de pesos. El objetivo es encontrar el valor mínimo del ***MSE*** que optimiza la precisión del modelo de regresión lineal.

Dos términos importantes de la regresión lineal serían:

- ***Factor de confusión***: se refiere a una variable independiente que (en los estudios no aleatorios) se asocia, no sólo con la variable dependiente, sino también con otras variables independientes. La presencia de factores de confusión puede distorsionar los resultados del modelo.
- ***Reajuste***: técnica estadística para eliminar la influencia de uno o más factores de confusión.

Si se quiere profundizar en el algoritmo de la regresión lineal recomiendo visitar la bibliografía, en concreto, [27].

2.2.2.2 Regresión Logística

La regresión logística constituye en una generalización de la regresión lineal que extiende su funcionamiento para aceptar variables de respuesta categóricas y obtener predicciones para variables dependientes binarias o multiclase. Entonces, utilizaremos la regresión logística para construir modelos en los que el resultado esperado, la predicción, sea una etiqueta de clase (positivo/negativo), esto es, a la hora de realizar tareas de clasificación binaria (***binary classification***).

En realidad, el algoritmo de regresión logística lo que nos permite es estimar la probabilidad de que una instancia pertenezca a una clase particular; cuando nuestra variable resultado es binaria estamos tratando de predecir la probabilidad de que tome un valor en [0, 1]. En una regresión logística se asume una relación lineal con el logaritmo natural de las probabilidades para el resultado:

$$y = a + b * X = ln(\frac{p}{1-p}) = \frac{1}{1+e^{-\Theta x}} \tag{2.3}$$

En lugar de modelar una relación lineal entre la variable independiente y la probabilidad del resultado, que producirá probabilidades fuera del rango [0, 1], la regresión logística establece una relación sigmoidea (con forma de S, ver Figura 2.10) entre la variable independiente y la probabilidad de que el resultado sea 0 o 1, que genera valores adecuadamente restringidos. en [0, 1].

La función de coste que nos permita encontrar los valores óptimos de los parámetros del modelo será aquella que castigue (asigne el menor peso) a las predicciones que arrojan un valor igual a 1 cuando el valor verdadero sea 0, y viceversa.

Para saber más sobre el algoritmo de regresión logística recomiendo visitar la bibliografía, en concreto, [28].

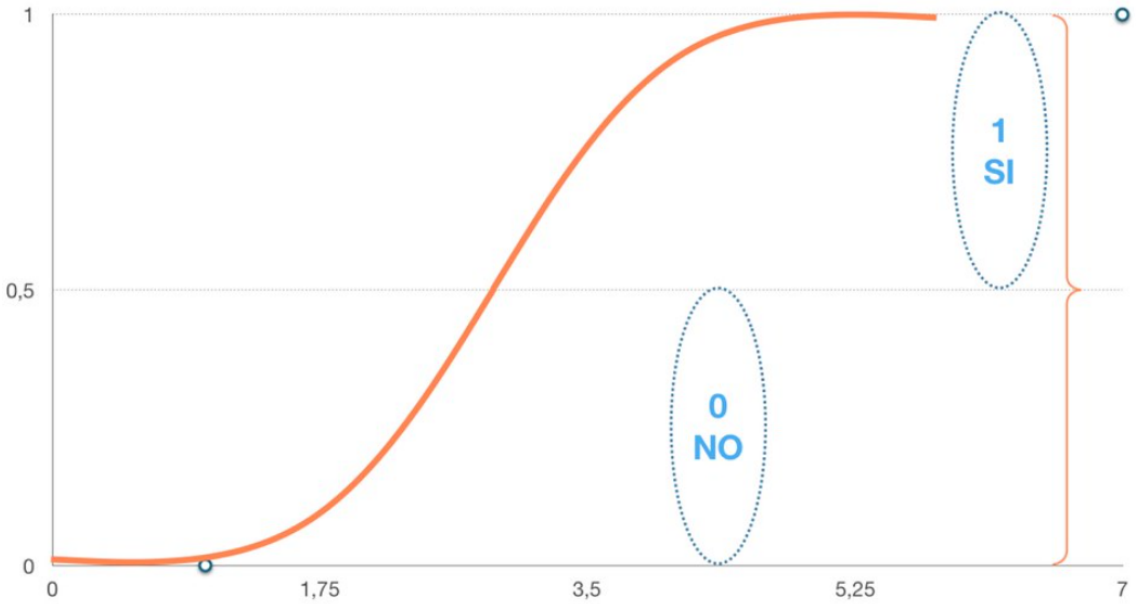


Figura 2.10 Curva sigmoidea que modela la regresión logística. Fuente: [25].

2.2.2.3 Máquinas de Vectores de Soporte (SVM)

Las ***SVM*** resultan particularmente adecuadas para llevar a cabo tareas de clasificación sobre conjuntos de datos complejos de mediano o pequeño tamaño, las máquinas ***SVM*** son algoritmos de aprendizaje automático versátiles que se pueden emplear tanto para construir modelos de clasificación lineal y no lineal, como para resolver problemas de regresión e incluso en la detección de valores atípicos en los datos.

Un modelo ***SVM*** es, básicamente, una representación de diferentes clases en un hiperplano que se define en un espacio multidimensional. La máquina ***SVM*** es la que genera dicho hiperplano de forma iterativa buscando el conjunto de parámetros óptimos que minimizan el error asociado. El objetivo es dividir los conjuntos de datos en clases y encontrar el hiperplano marginal máximo o ***MMH*** que define la mejor separación geométrica entre las clases (ver Figura 2.11). En un espacio bidimensional este hiperplano es una línea que divide un plano en dos secciones y en las que cada clase se encuentra a un lado.

En la Figura 2.11 aparecen tres conceptos que serían los siguientes:

- ***Vectores de soporte***: Posición espacial de datos más cercano al hiperplano. Estos vectores ayudan a definir la línea de separación entre los dos conjuntos.
- ***Hiperplano***: Es el plano o espacio de decisión que divide un conjunto de elementos que pertenecen a clases distintas.
- ***Margen***: Distancia entre dos líneas trazadas sobre los puntos de datos de diferentes clases que se hayan más próximos, es equidistante a los vectores de soporte y ortogonal a la frontera. El tamaño del margen es un indicador del poder de separación del modelo, a mayor margen mejor separación entre clases.

Para profundizar, recomiendo visitar [44] o [49] de la bibliografía.

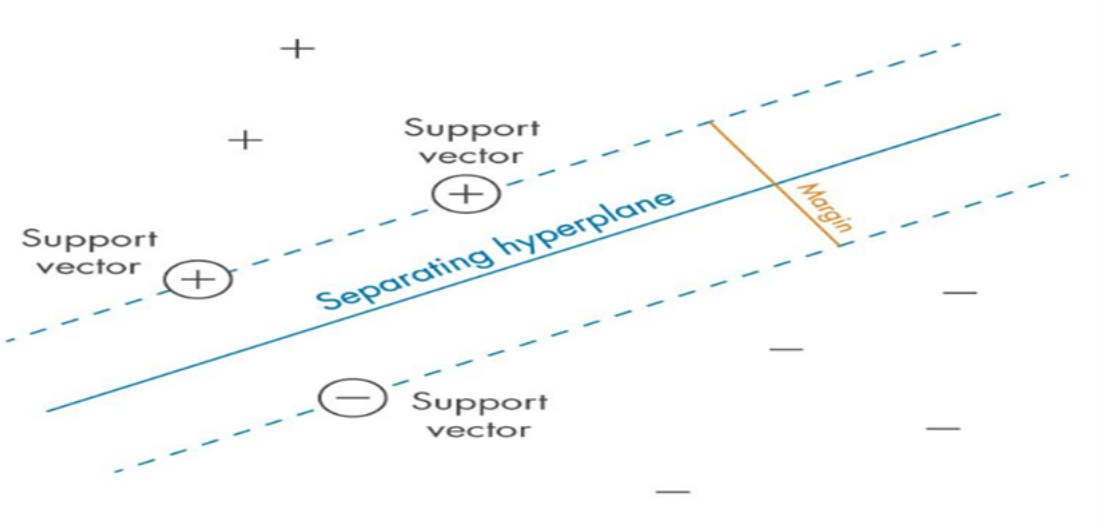


Figura 2.11 Ejemplificación del margen que intenta optimizar las SVM. Fuente: [49].

2.2.2.4 Vecinos más cercanos

Los algoritmos basados en Vecinos más cercanos o ***k-NN*** (k-Nearest Neighbors) se pueden emplear para trabajar tanto en problemas de regresión como en tareas de clasificación. En un problema de clasificación, el algoritmo ***k-NN*** selecciona la clase correcta utilizando como métrica de similitud la distancia entre un grupo de puntos en los datos de prueba y todos los puntos que contiene el conjunto de datos completo: un dato se asocia a un cierto grupo si tiene k vecinos más cerca de aquel que de otro grupo. Por tanto, podemos considerar que un ***k-NN*** es simplemente un método de búsqueda que clasifica observaciones explorando su entorno.

Esta búsqueda se realiza según el siguiente procedimiento; para cada elemento en los datos de prueba, se calcula la distancia a los k elementos más cercanos de cada clase, el elemento actual se etiqueta de acuerdo a la clase para la que la distancia promedio de sus k elementos es menor. La representación mostrada en la Figura 2.12 ayuda a la comprensión de este procedimiento.

La función de distancia euclidiana es la más popular y es efectiva siempre que trabajemos con datos de dimensionalidad reducida. Dado un punto de consulta (q) y un punto de datos (p) esta distancia se define como:

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2.4}$$

Si se quiere profundizar en el algoritmo de los vecinos más cercanos o k-NN recomiendo visitar la bibliografía, en concreto, [29].

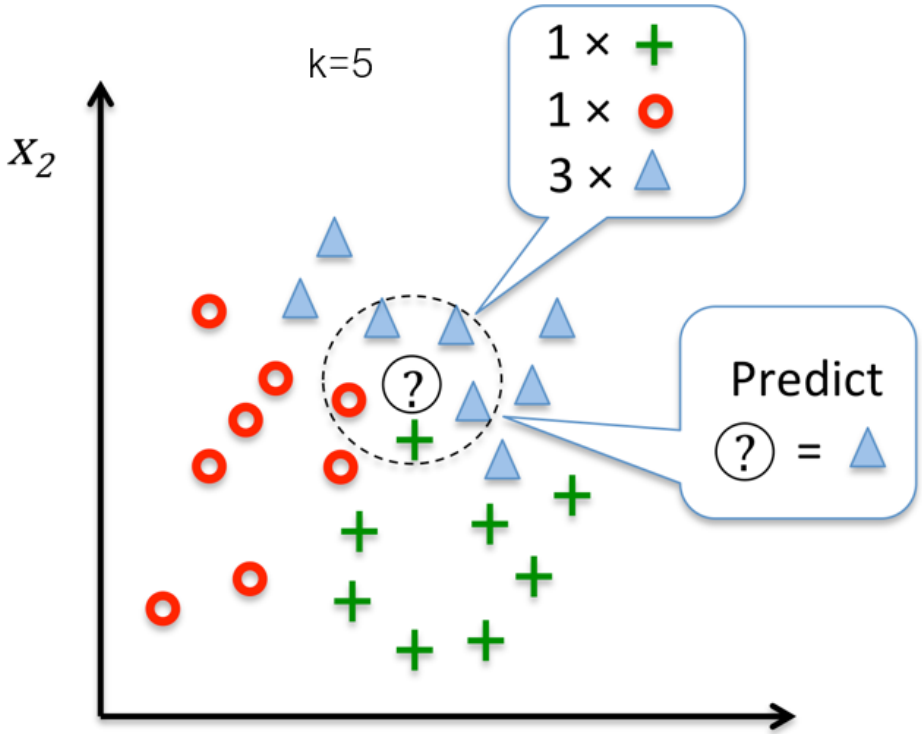


Figura 2.12 Ejemplificación del algoritmo k-NN. Fuente: [40].

2.2.2.5 Árboles de decisión

Un árbol de decisión es un algoritmo de aprendizaje supervisado que podemos emplear tanto en problemas de clasificación como de regresión. Estos son un tipo de algoritmos potentes y versátiles, capaces de adaptarse a conjuntos de datos de origen y formas diversas. Además son modelos flexibles en los que el número de parámetros no va a crecer necesariamente al añadir nuevas características, y que pueden dar como resultado una predicción categórica o una predicción numérica.

Los modelos basados en árboles de decisión se construyen evaluando recursivamente diferentes características y utilizando en cada nodo la característica que produce la separación óptima entre subgrupos de datos.

Pero, ¿cómo podemos medir cuál de los test consigue la menor incertidumbre en la clasificación? Como hemos mencionado antes, los árboles de decisión se construyen dividiendo de manera recursiva la muestra de entrenamiento en base a las características que mejor operan en una tarea específica. La medida del desempeño en esta tarea, o criterio de minimización, nos la proporcionarán métricas dedicadas como el ***índice de Gini*** (para saber más [36]) o el ***valor de entropía***.

Por otra parte, podemos conceptualizar la ***entropía*** como una métrica que nos indica el desorden de las características respecto al grupo objetivo, de modo que la división óptima sería aquella en la que se elige la característica que minimiza la ***entropía***. El valor máximo de ***entropía*** se obtiene cuando la probabilidad de que un elemento pertenezca a una de las dos clases evaluadas en el nodo es idéntica. Un nodo es puro cuando la ***entropía*** alcanza su valor mínimo, el cero.

Además de las métricas o criterios de evaluación, los parámetros más comunes que afinar durante el entrenamiento de un modelo basado en árboles de decisión son:

- ***max_depth***: La profundidad máxima que pueden alcanzar los nodos del árbol de decisión, esto puede ayudar a evitar el sobreajuste pero también podría tener el efecto contrario.
- ***min_samples_leaf***: La cantidad mínima de ejemplos que debe tener un nodo hoja.

- ***min_samples_split***: La cantidad mínima de ejemplos que debe tener un nodo de decisión para realizar una división, si la cantidad no es suficiente el nodo permanece como nodo hoja.

Una de las cualidades de los algoritmos basados en árboles de decisión es su eficacia a la hora de mapear relaciones no lineales. Sin embargo, el entrenamiento de un árbol de decisión pueden terminar generando ramas con reglas muy estrictas aplicadas sobre grupos de datos muy reducidos esto puede afectar gravemente a la capacidad predictiva del modelo cuando lo enfrentemos a datos no vistos, introduciendo problemas de sobreajuste con bastante facilidad.

Para saber más sobre el algoritmo de los árboles de decisión recomiendo visitar la bibliografía, en concreto, [34].

2.2.2.6 Bosques Aleatorios

Un bosque aleatorio es un algoritmo que combina una colección de árboles de decisión que se entrenan sobre subconjuntos (seleccionados de modo aleatorio) de la muestra de entrenamiento y del grupo de variables explicativas. En un bosque aleatorio, las predicciones se obtienen a partir de la media de las predicciones realizadas por los árboles de decisión que los componen (ver Figura 2.13). Los bosques aleatorios son menos propensos al sobreajuste que los árboles de decisión porque consiguen evitar que un árbol aislado aprenda todas las instancias y variables explicativas del problema, al restringir su exposición a un subconjunto del dataset completo. Cada árbol individual del bosque aleatorio proporcionará una predicción de clase, pero sólo la clase con más votos se convierte en la predicción del modelo. Además de ser útiles como clasificadores, los bosques

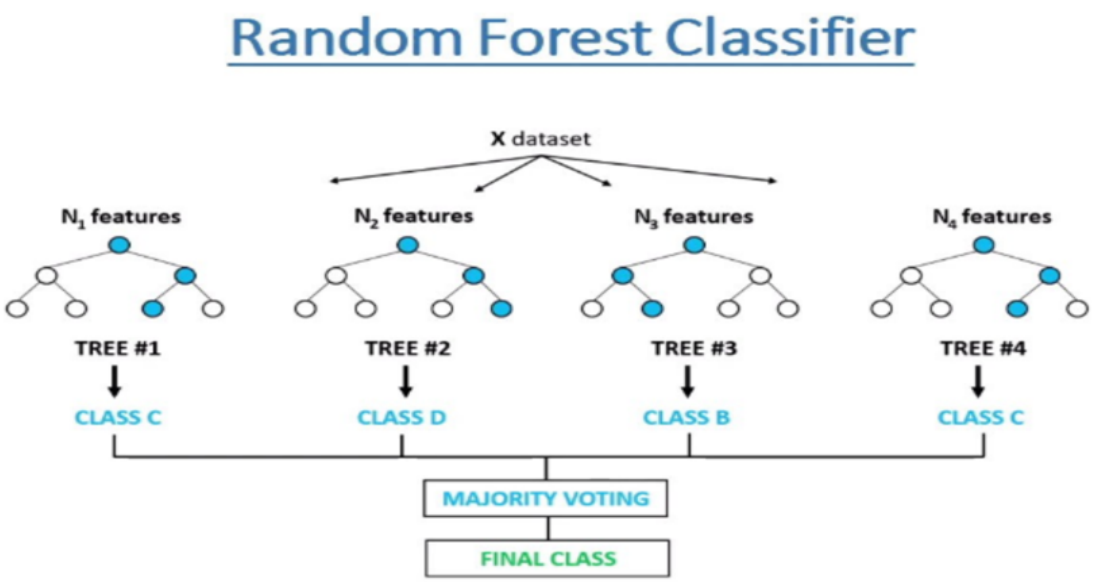


Figura 2.13 Ejemplificación del algoritmo de árboles de decisión. Fuente: [12].

aleatorios pueden llevar a cabo tareas de regresión. Pero, mientras que en la clasificación la salida está determinada por la moda (etiqueta más repetida), en la regresión la predicción del modelo es el resultado de calcular el valor medio de las predicciones que producen los árboles individuales.

Para profundizar, recomiendo visitar el apartado [33] de la bibliografía.

2.2.2.7 Algoritmo Q-Learning

Este algoritmo consiste en actualizar iterativamente una tabla que llamaremos Q-Table con la ayuda de una función llamada Q-función, que estima la utilidad de tomar una determinada acción en un estado dado. El agente explora el entorno, toma acciones, observa las recompensas resultantes y actualiza sus estimaciones de la Q-función cómo podemos ver en la 2.14. Para actualizar los valores de la Q-table con la función Q, se utiliza la ecuación de Bellman y la política de recompensas. La

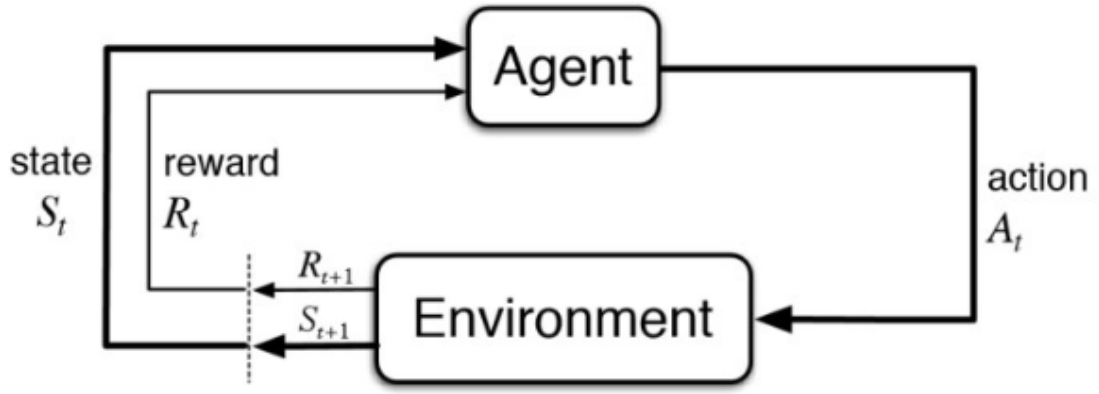


Figura 2.14 Ejemplificación Agente-Entorno del algoritmo Q-Learning. Fuente: [11].

función Q la podemos observar en la Ecuación 2.5 pero previo a ello vamos a explicar qué es la política de recompensas, básicamente es asignar una recompensa positiva cuando gana la partida pero asignar una recompensa negativa cuando pierde la partida.

$$Q'(s,a) = Q(s,a) + \alpha[R + (\lambda \cdot maxQ(s',a')) - Q(s,a)] \tag{2.5}$$

donde:

- $Q'(s,a)$: valor que obtenemos y actualizaremos en la tabla
- $Q(s,a)$: es el valor de la función Q en el estado y acción actual
- α: es el ratio de aprendizaje del agente, donde un valor más alto dará mayor importancia a las nuevas recompensas.
- R: es la recompensa inmediata después de realizar la acción a en el estado s.
- λ: es la tasa de descuento que nos permite medir cómo valoramos las recompensas futuras respecto a las inmediatas, mayor valor de λ implica mayor valor a las recompensas futuras.
- $maxQ(s',a')$: es el valor máximo esperado, es decir, el valor que obtendríamos si tomamos la acción óptima para el nuevo estado.

Por último, explicaremos cómo se lleva la actualización de los valores en la tabla ayudándonos de la 2.15: ***Paso 1: Inicializar la Q-table***

Será una tabla de n-columnas (= número de acciones) y m-filas (= número de estados), al inicio cada hueco de la tabla tendrá valor 0.

Pasos 2 y 3: Elegir y Realizar una acción

Esta combinación de pasos se realiza por un tiempo indefinido. Elegiremos una acción (a) en el estado (s) basado en la Q-Table. Pero, como se mencionó anteriormente, cuando el episodio comienza inicialmente, cada valor de Q es 0.

Para elegir la acción usaremos la "***Estrategia Codicia de Épsilon (e-greedy police)***", consiste en que mediante un valor ε (elegido por nosotros) se escogerá una acción totalmente al azar y a medida que nuestro agente va explorando el entorno, irá aprendiendo y mejorando su estrategia (disminuyendo así el valor de ε). Para saber más sobre esta estrategia recomiendo visitar [5].

Pasos 4 y 5: Evaluar la recompensa y actualizar la tabla

Ahora hemos tomado ya una acción, llegado a un nuevo estado y observado el resultado y recompensa. Sólo nos faltaría actualizar la tabla y para ello usaremos la función Q que se ha explicado anteriormente en 2.5.

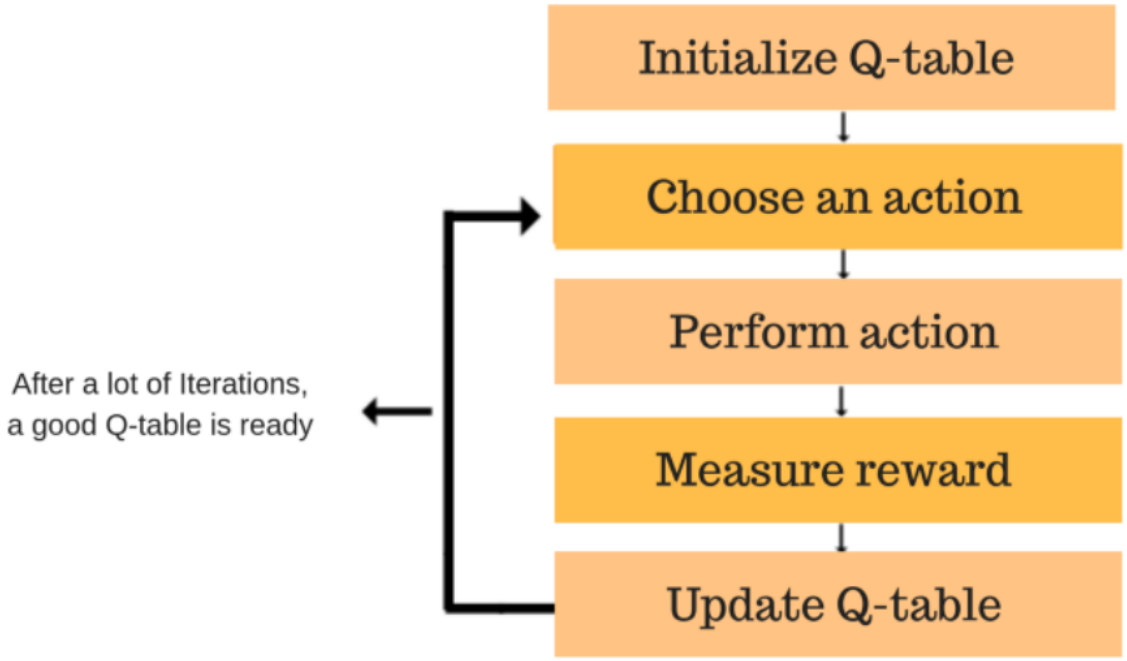


Figura 2.15 Proceso de la actualización de los valores de la Q-Table. Fuente: [23].

Para saber más sobre el algoritmo de Q-Learning, visitar [23] o también [5].

2.2.3 Deep Learning

2.2.3.1 ¿Qué es?

Dentro del Machine Learning nos encontramos con múltiples técnicas para llevar a cabo diferentes tareas como hemos visto en el apartado anterior. Sin embargo, si uno de estos grupos de modelos ha dado fama al aprendizaje automático por su flexibilidad y sus múltiples aplicaciones, este es el de las redes neuronales.

Las redes neuronales son cascadas de capas con unidades de procesamiento no lineal (llamadas neuronas) que tienen la capacidad de extraer y transformar información de unas variables de entrada para producir una salida deseada. Estas redes están inspiradas en la arquitectura del cerebro humano y están constituidas por múltiples capas de neuronas en las que cada capa usa la salida de la capa anterior como entrada (de ahí el nombre de "redes").

Lo interesante de las redes neuronales es que pueden aprender de forma jerarquizada. Las primeras capas aprenden conceptos muy concretos, directamente relacionados con la información de entrada, mientras que las capas posteriores utilizan la información aprendida previamente para desarrollar conceptos más abstractos. Esto hace que cuantas más capas se añadan, el nivel de progresión de la red neuronal en su nivel de conocimiento sea cada vez más interesante. A estas redes neuronales con múltiples capas se las conoce como redes neuronales "profundas", dando origen al popular concepto de aprendizaje profundo o "deep learning". Una vez que se ha entrenado una red neuronal profunda, esta puede controlar más ambigüedad que una red superficial.

El principal problema de las redes neuronales profundas es que necesitan gran cantidad de datos para ser entrenadas. Cuanto más generalizable y abstracto sea el conocimiento que queremos obtener, más datos necesitaremos, haciendo de las redes neuronales grandes devoradoras de datos. Sin embargo, con la llegada de la era de la información, disponemos de cada vez más datos, lo cual está haciendo que las redes neuronales se incorporen a más y más campos. Esta acumulación masiva de datos en todos los ámbitos, junto con su análisis de forma masiva, ha dado lugar al concepto de "Big Data".

Si se quiere repasar con la ayuda de un mapa conceptual todos estos conceptos vistos aquí como la inteligencia artificial, machine learning, deep learning/redes neuronales o Big Data. Recomiendo ver el siguiente vídeo [18].

2.2.3.2 El perceptrón

El aprendizaje profundo en su forma más simple es una evolución del algoritmo del ***perceptrón***, entrenado con un optimizador basado en gradientes. Al igual que una neurona biológica, el algoritmo del ***perceptrón*** actúa como una neurona artificial, con múltiples entradas y pesos asociados con cada entrada, cada uno de los cuales produce una salida.

Para una serie de entradas, digamos, características de una entidad de cualquier tipo, el ***perceptrón*** es capaz de clasificarlo en una de dos categorías. La fórmula básica del ***perceptrón*** es la siguiente (también lo podemos ver ilustrado en la siguiente Figura 2.16):

$$y(x_1,...,x_n) = f(w_1x_1 + ... + w_nx_n) \tag{2.6}$$

Como mencionamos anteriormente, el perceptrón es un clasificador binario, y por tanto el objetivo de la función f será devolver 0 si las características de la entrada se corresponden con una de las clases, y 1 si las características se corresponden con la otra. La función de los pesos w_i es la de hacer que las características que sean más importantes sean las que más contribuyan a la salida. Como la función $(w_1x_1 + ... + w_nx_n)$ es lineal, lo que está haciendo el ***perceptrón*** en realidad es

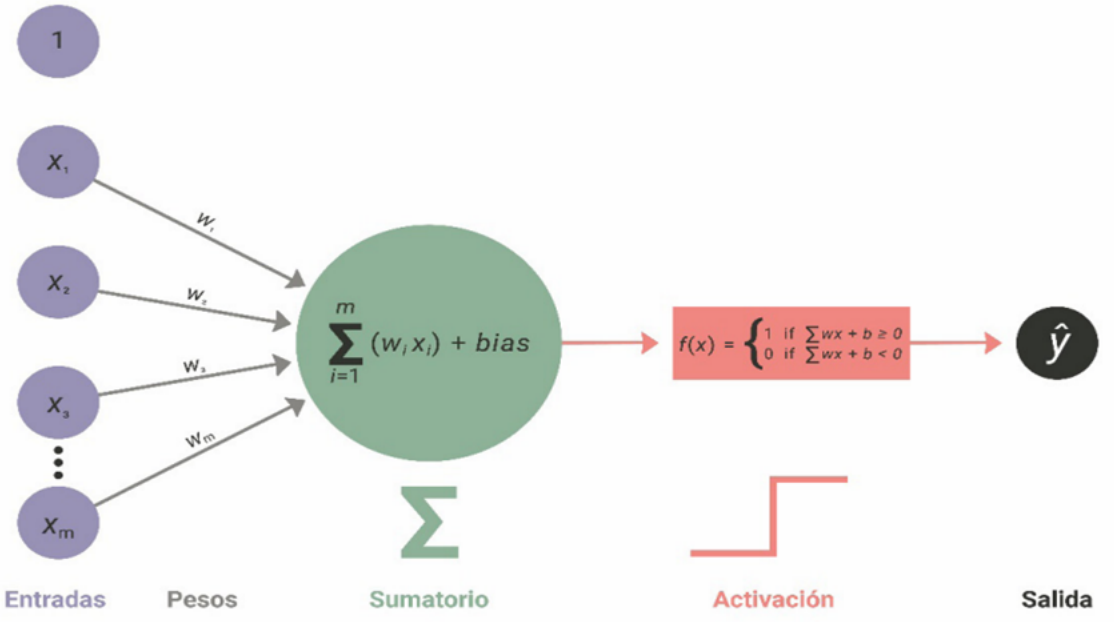


Figura 2.16 Diagrama de una neurona artificial. Fuente: [1].

aprender a dibujar un hiperplano que permita separar las dos clases como se ve en la Figura 2.17. La primera limitación y la más evidente es que el perceptrón solo permitía clasificar en dos clases, y por lo tanto, no podía solucionar problemas de clasificación más complejos con múltiples salidas. La segunda es que incluso cuando sólo hay dos clases, estas pueden no ser separables de forma lineal, como puedes ver en la Figura 2.18. ¿Se puede trazar una línea que separe los puntos azules y naranjas en estos ejemplos?. Para profundizar conocimientos sobre el perceptrón recomiendo visitar el apartado [6] de la bibliografía.

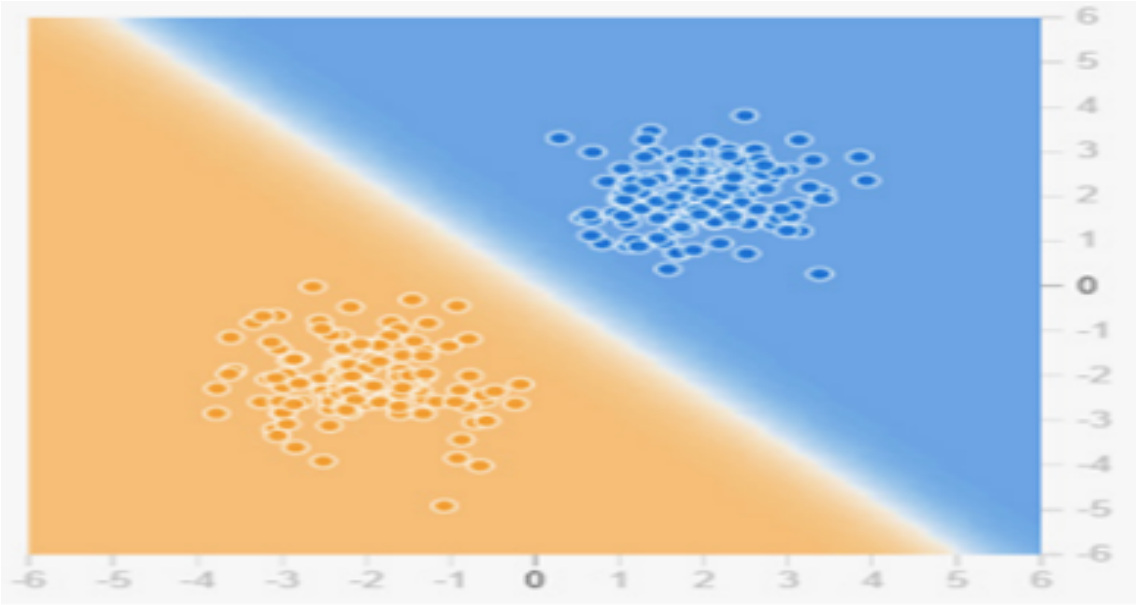


Figura 2.17 Ejemplo de línea de separación construida por un perceptrón (hiperplano). Fuente: [74].

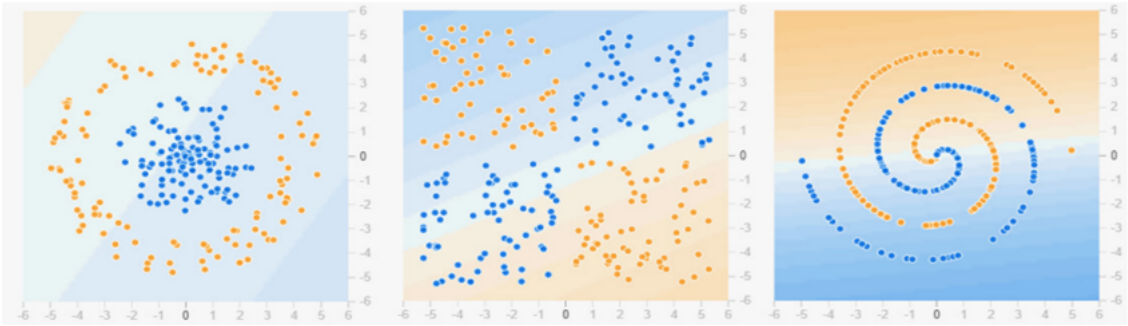


Figura 2.18 Ejemplos de problemas de clasificación no binarios. Fuente: [74].

2.2.4 Redes Neuronales

2.2.4.1 ¿Cómo entrenarlas?

Por desgracia, la inmensa mayoría de los problemas complejos que nos encontraremos en el mundo real son problemas no lineales que no pueden resolverse con este sencillo algoritmo lineal. Para intentar solucionar esto, las redes neuronales unen varias neuronas en una red. Un añadido importante es que a la salida de la neurona se la aplica una función no lineal llamada ***función de activación***. La función de activación decide si una neurona artificial debe activarse o no, ayudando a la red neuronal a aprender patrones complejos en los datos y también a normalizar la salida de cada neurona. Existen varias posibilidades para la función de activación, siendo las más comunes las siguientes:

- ***Función sigmoidea***: La activación sigmoidea genera un valor en (0, 1), que es útil para realizar cálculos que deben interpretarse como probabilidades. Además, normaliza los datos y también es útil para crear salidas probabilísticas y construir funciones de pérdida derivadas de modelos de máxima verosimilitud.
- ***Tangente Hiperbólica***: La función tanh es preferible a la sigmoidea cuando se desea que las salidas de los cálculos sean tanto positivas como negativas.

- ***Función de Activación Lineal Rectificada***: La función de activación lineal rectificada o ***RELU***. La salida de la ***RELU*** es igual a la entrada si el valor de la entrada es positivo, pero si recibe una entrada negativa, la función devuelve la salida cero. Es la función de activación más utilizada en la actualidad por sus ventajas matemáticas y por su aceleración del aprendizaje.

Esta nueva función de activación no lineal permite que el valor de salida sea una combinación ponderada no lineal de sus entradas, creando así características no lineales utilizadas por la siguiente capa. Es importante entender que una función de tipo escalón como la utilizada en el perceptrón para la salida o una función lineal no sirve para poder obtener la solución de nuestro problema planteado de problemas no lineales (Figura 2.19). Cuantas más capas añadimos a una red neuronal, más

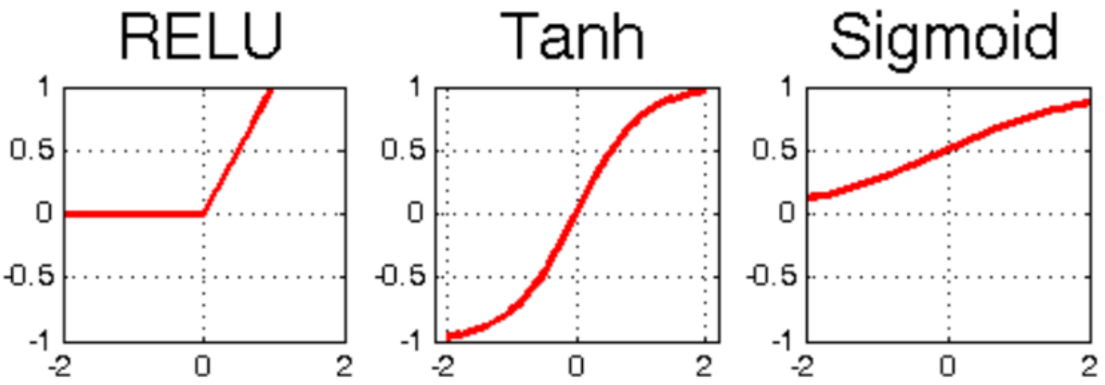


Figura 2.19 Funciones de Activación. Fuente: [26].

compleja puede ser la tarea de clasificación que puede llevar a cabo. Este concepto es el que da nombre al aprendizaje profundo o "deep learning". La característica distintiva del aprendizaje profundo es su aplicación a problemas que antes no eran factibles con los métodos tradicionales y las redes neuronales más pequeñas. Las redes más profundas permiten que se aprendan más capas de abstracciones jerárquicas para los datos de entrada, por lo que se vuelven capaces de aprender funciones de orden superior en dominios más complejos. Un ejemplo de estas redes neuronales "profundas" lo podemos encontrar en la Figura 2.20.

Para saber más sobre redes neuronales, invito a visitar [10] de la bibliografía.

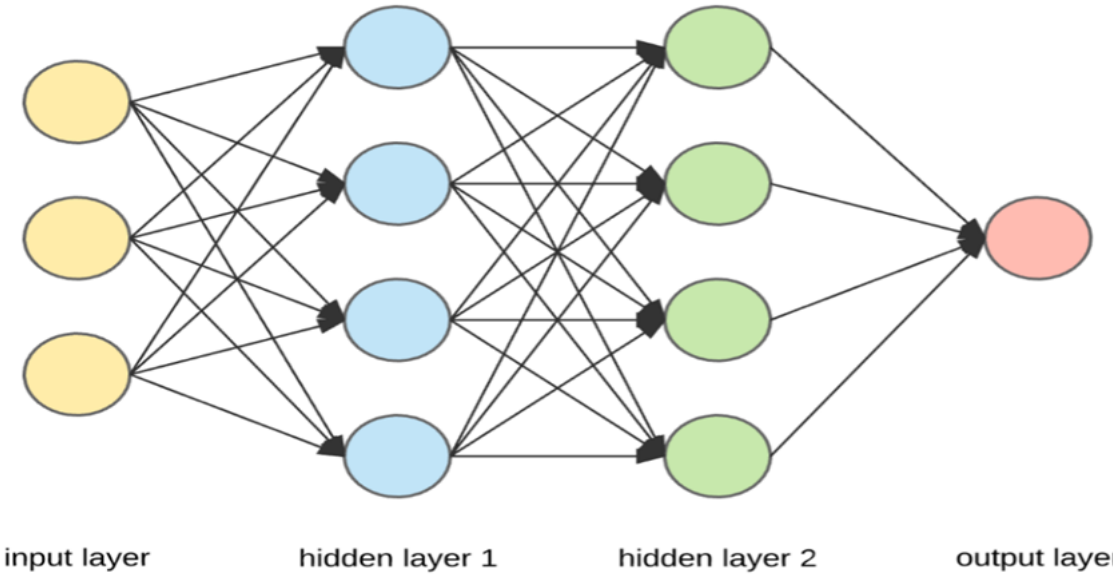


Figura 2.20 Ejemplos de una red neuronal "profunda". Fuente: [76].

2.2.4.2 Entrenamiento de una Red Neuronal (Aprendizaje Supervisado).

Hasta ahora hemos definido cómo construir una red neuronal; sin embargo para que esta sea realmente útil es necesario entrenarla. Como ya vimos anteriormente con el perceptrón, entrenar una red neuronal consiste en encontrar el conjunto de pesos w_i para cada neurona que hagan que la red sea eficiente en la tarea deseada. Volvamos brevemente al perceptrón para ilustrar este problema:

$$y(x_1,x_2) = g(w_0 + w_1x_1 + w_2x_2) \tag{2.7}$$

Las soluciones obtenibles con el entrenamiento pueden tener una buena precisión y separar de forma efectiva las clases. Sin embargo, ¿cómo podemos orientar a nuestro algoritmo para que encuentre la mejor solución? Aquí entra en juego la "***función de coste***", una función matemática que modele el error en la clasificación para así poder minimizarlo.

Hay muchas funciones de coste y cada una tiene sus casos de uso dependiendo de si se trata de un problema de regresión o de clasificación. Lo que necesitamos saber es que es una función $f(y,y')$ que mide la diferencia entre los valores predichos por nuestro algoritmo (y) y los valores teóricos provenientes del set de entrenamiento (y'). De esta forma, el problema de entrenar a nuestra neurona para resolver un problema de clasificación o regresión pasa por minimizar el error $f(y,y')$. En nuestro ejemplo, evaluar la variación de $f(y,y')$ con respecto a los cambios de w_1 es relativamente sencillo tanto de forma matemática como de forma computacional. De forma matemática, los cambios de f con respecto a w_1 se evaluarían simplemente utilizando la teoría de derivadas parciales. Siendo $z = w_1x_1 + w_2x_2$:

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial y} * \frac{\partial y}{\partial z} * \frac{\partial z}{\partial w_1} \tag{2.8}$$

De forma computacional, bastaría con introducir un pequeño cambio en w_1, y recalcular los resultados de la red neuronal. Conceptualmente, si con el cambio hace que se reduzca la ***función de coste*** y que los resultados de la clasificación ***mejoren***, seguiríamos modificando w_1 en la ***misma dirección***, mientras que si ***empeoran***, retrocederíamos al valor anterior y nos moveríamos en la ***dirección contraria***. Este proceso continuaría hasta encontrar un valor óptimo para el parámetro. Cierto es que la salida de la neurona no depende solo de w_1, sino que lo hace también con respecto a w_2. Para expresar la dependencia de la salida con respecto a ambas variables, utilizamos el vector:

$$\nabla f = \left(\frac{df}{dw_1}, \frac{df}{dw_2}, ..., \frac{df}{dw_n} \right) \tag{2.9}$$

que matemáticamente se conoce como ***vector gradiente***. El objetivo del entrenamiento de una red neuronal es ***minimizar la función de coste***, lo que se traduciría en encontrar un conjunto de parámetros tal que $\nabla f = 0$. Estos conjuntos de parámetros señalan puntos en los que la ***función de coste*** está en un mínimo o en un máximo local o parcial; por ejemplo, cualquiera de los puntos señalados en la Figura 2.21:

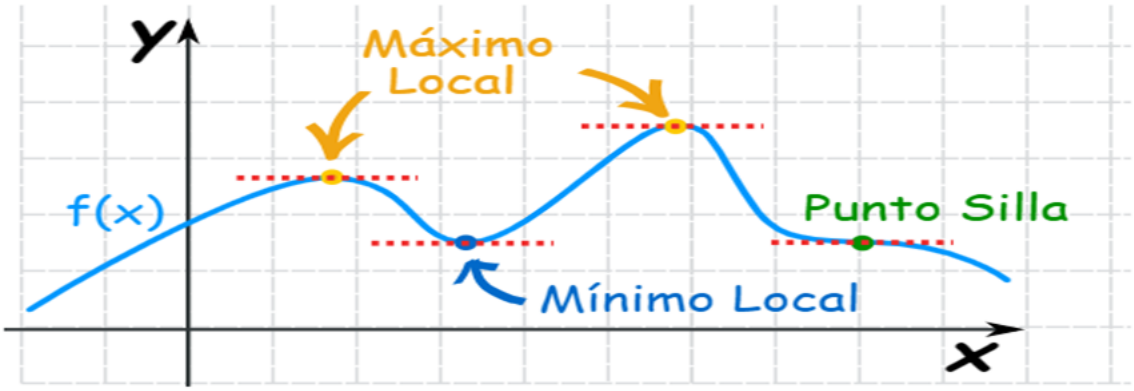


Figura 2.21 Máximos y mínimos de una función de coste. Fuente: [42].

Uno de los puntos marcados en la Figura 2.21 es un ***punto de silla***, un punto con derivada cero pero que no es un mínimo ni un máximo. Esto no supone un problema grave, ya que un paso adicional nos llevará a zonas con gradiente más bajo. Es importante entender que nuestro algoritmo solo conoce el gradiente en el entorno en el que lo calculemos, por lo tanto si nos encontramos con un mínimo local, no es posible saber si en la función ∇f existen mínimos menores. Es importante entender que, aunque el punto mínimo local cumple la condición $\nabla f = 0$, solo el ***mínimo global*** cumple con nuestra condición de ***minimizar la función de coste*** (es el error mínimo posible).

Existe una dificultad añadida, y es que si nuestra red neuronal es demasiado grande, con capas interconectadas entre ellas, la variación de un parámetro en una de las primeras capas debe ser propagada a través de las salidas de su neurona a las capas subsiguientes, y de estas a las subsiguientes, por lo que además del problema anterior se nos suma el cálculo del gradiente y se vuelve extremadamente costoso. Si nos fijamos en la Figura 2.22, veremos rápidamente que los cambios en w van a producir cambios en las entradas de todas las neuronas conectadas, y, por tanto, se deben evaluar también los cambios en los parámetros de estas para saber cómo los cambios de w influyen en Y_P.

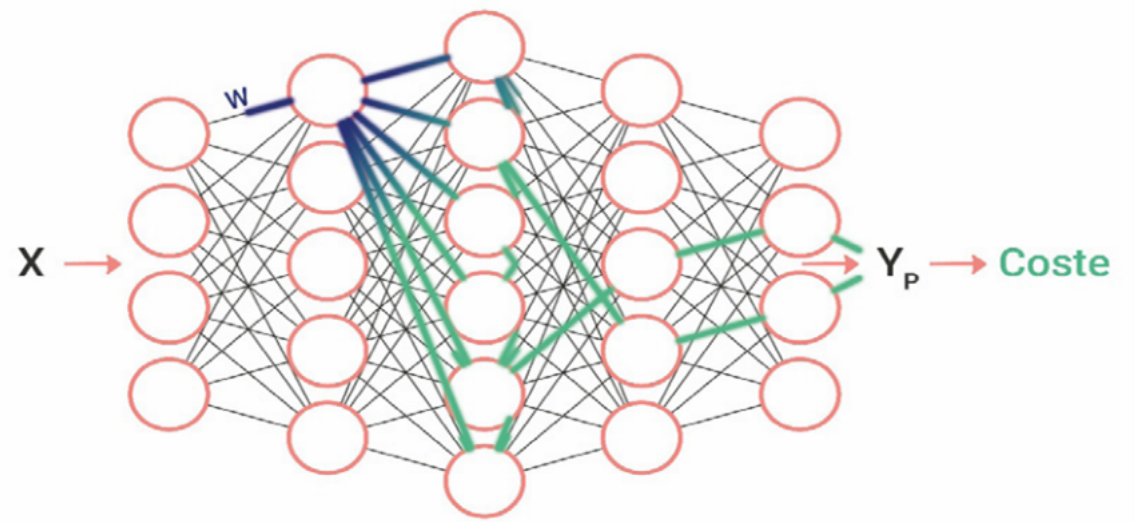


Figura 2.22 Propagación del error hacia delante. Fuente: DotCSV.

2.2.4.3 Algoritmos de Entrenamiento

Descenso del Gradiente

El descenso del gradiente es un algoritmo de optimización iterativa para encontrar un mínimo en una función diferenciable. La idea detrás del algoritmo de descenso del gradiente es la siguiente: cuando inicializamos nuestra red neuronal, normalmente lo hacemos con un conjunto aleatorio de valores de los pesos w_i, esto significa empezar en un punto aleatorio de la superficie. A partir de este punto, la idea del algoritmo de descenso del gradiente consiste en calcular el gradiente ∇f, y comenzar a moverse a través de la superficie en pequeños pasos en la dirección contraría, es decir, hacia donde la pendiente de descenso es más rápida. Es importante entender que la forma de la superficie es desconocida, ya que tanto las redes neuronales como las funciones de coste son normalmente matemáticamente complejas.

Para profundizar conocimientos sobre este algoritmo recomiendo visitar [16] o bien visualizar [17], ambos disponibles en la bibliografía del proyecto.

Backpropagation

El algoritmo de descenso del gradiente soluciona el problema de encontrar los mínimos de la función de coste, pero como señalamos anteriormente, otro problema añadido era el cálculo del propio gradiente, debido a la dependencia de la salida de los parámetros de todas las capas. Esta es la última pieza del puzzle, y es el problema que busca resolver el algoritmo de ***backpropagation***.

El objetivo de cualquier algoritmo de aprendizaje supervisado es encontrar la función que mejor mapee un conjunto de entradas a su salida correcta, representada por un conjunto de pesos w_i. Como expusimos anteriormente, para evaluar el cambio en la salida cuando variamos el parámetro w_0 en la primera capa, debemos propagar hacia adelante los cambios a través de toda la red, para observar cómo estos influyen en la salida. En el algoritmo de ***backpropagation***, el error en la salida se propaga hacia atrás hacia todas las neuronas de la siguiente capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, dependiendo de los pesos de sus entradas. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total (Figura 2.23).

Al igual que para el algoritmo del descenso del gradiente, para profundizar en este caso en el algoritmo de backpropagation. Recomiendo visitar [9] o bien visualizar [19] disponibles en la bibliografía del proyecto.

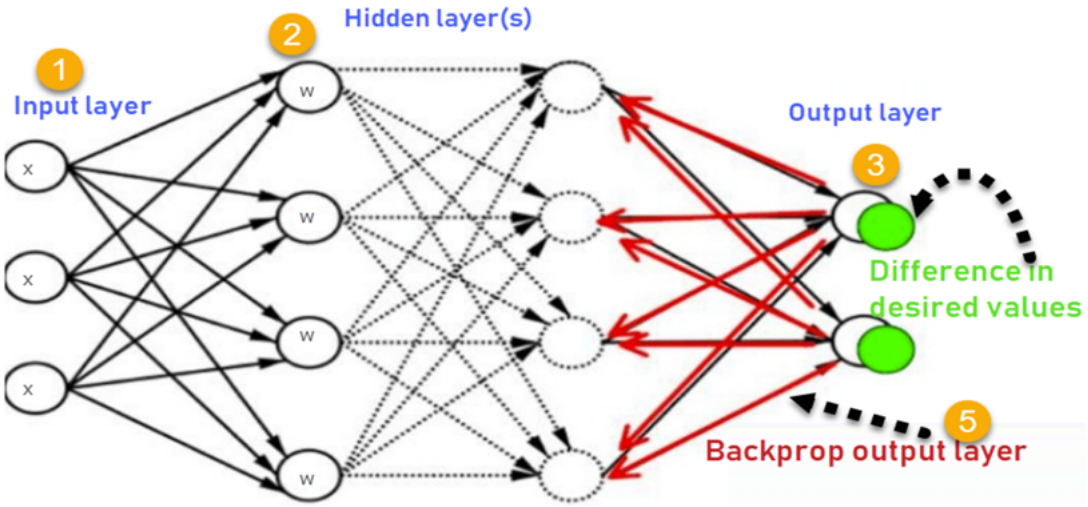


Figura 2.23 Concepto del algoritmo backpropagation. Fuente: [65].

2.2.4.4 Tipos de Redes Neuronales

A medida que las redes neuronales crecen en tamaño, también lo hace la expresividad de los desarrolladores, que llevan a cabo distintos tipos de redes neuronales especializadas en ciertos tipos de tareas, generalmente inspirándose en lo que sabemos de los procesos de aprendizaje humano en distintas tareas e intentando representar estos procesos en sus arquitecturas.

A día de hoy existen muchos tipos de redes neuronales, cada una con sus fortalezas únicas. Algunas de las arquitecturas más relevantes son las siguientes:

- ***Perceptrones multicapa***: Son redes que se utilizan para clasificar datos que no se pueden separar linealmente. Es un tipo de red neuronal artificial que está completamente conectada. Esto se debe a que cada nodo de una capa está conectado a cada nodo de la siguiente capa.

- ***Redes Neuronales Convolucionales***: Una red neuronal convolucional (CNN) es una variación de la red neuronal convencional que contiene una o más capas convolucionales. La capa convolucional realiza una operación convolucional en la entrada, lo que reduce el tamaño de las entradas. De esta forma la red puede ser mucho más profunda pero con muchos menos parámetros. Debido a esta capacidad, las redes neuronales convolucionales muestran resultados muy efectivos en el reconocimiento de imágenes y vídeo (lo que las hace realmente útiles en el campo de la medicina) y algunas tareas del procesamiento del lenguaje natural como el análisis semántico y la detección de paráfrasis. Para profundizar ver el siguiente vídeo [20].

- ***Redes Neuronales Recurrentes***: Una red neuronal recurrente (RNN) es un tipo de red neuronal en la que la salida de una capa en particular se guarda y se retroalimenta a la entrada. Esto le permite exhibir un comportamiento dinámico temporal. Las RNN pueden usar su estado interno (memoria) para procesar secuencias de entradas de longitud variable. Esto las hace aplicables a tareas como el reconocimiento de escritura a mano conectado y no segmentado o el reconocimiento de voz. La primera capa se forma con el producto de la suma de pesos y características. Sin embargo, en las capas posteriores, comienza el proceso recurrente de la red neuronal. De cada paso de tiempo al siguiente, cada nodo recordará alguna información que tenía en el paso de tiempo anterior. La red neuronal comienza con la propagación frontal como de costumbre, pero recuerda la información que puede necesitar usar más adelante.

- ***Redes Generativas Antagónicas***: Las Redes Generativas Antagónicas (GAN), son una clase de algoritmos de inteligencia artificial que se utilizan en el aprendizaje no supervisado, implementadas por un sistema de dos redes neuronales que compiten mutuamente. Las redes generativas antagónicas se utilizan para la generación de texto y, sobre todo, imágenes. La idea es simple, una de las redes, llamada el ***generador***, se entrena para generar una imagen nueva para un subset. Por otro lado, la otra red, llamada ***discriminador***, se entrena para distinguir imágenes que pertenecen al subset de aquellas que no. El objetivo del entrenamiento en una ***GAN*** es conseguir que la red generativa genere imágenes lo suficientemente creíbles como para engañar a la red discriminativa, haciéndola creer que verdaderamente pertenecen al subset (Figura 2.24).

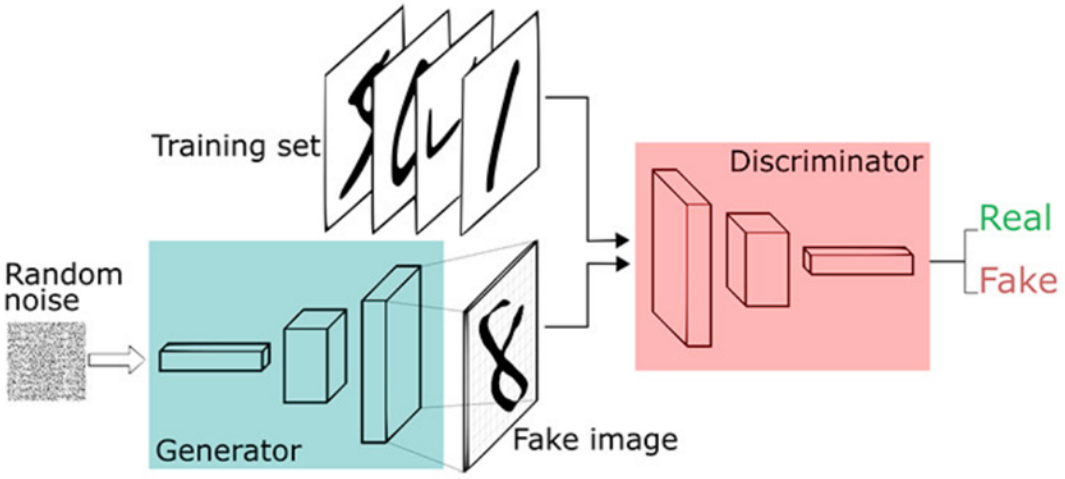


Figura 2.24 Concepto de GAN. Fuente: [51].

Para saber más sobre los tipos de redes neuronales recomiendo visitar [70].

2.2.5 El Paradigma de Aprendizaje. Selección de Características y Optimización de los Modelos

2.2.5.1 Conjuntos de entrenamiento, validación y prueba (training / validation / test)

La experiencia que el algoritmo utiliza para aprender reside en las observaciones o datos que componen el llamado conjunto de entrenamiento (***training set***). En los problemas de aprendizaje supervisado, cada observación consta de una variable respuesta observada y una o más variables explicativas observadas. Por otra parte, el conjunto de validación (***validation set***) es una colección similar de observaciones que se utiliza para evaluar el rendimiento del modelo utilizando una o varias métricas de rendimiento. Es importante que no se incluyan observaciones del conjunto de entrenamiento en el de validación, ya que en caso contrario no podremos saber con certeza si nuestro algoritmo ha aprendido a generalizar a partir del conjunto de entrenamiento o simplemente lo ha memorizado.

Como regla general se utiliza entre el 70 % y el 80 % de los datos de los que dispongamos para utilizarlos en el entrenamiento y el resto se reserva para evaluar el rendimiento del modelo. Es decir, se realiza una división 80 %-20 % o 70 %-30 % entre entrenamiento y validación.

Otro enfoque de consenso, y que recomiendo, consiste en dividir los datos en tres conjuntos independientes: entrenamiento, validación y prueba (test set). Este último deberá contener datos que nunca entran al modelo mientras se afina, si no que se reservan para elegir los hiperparámetros (revisaremos este concepto en una sección posterior). El conjunto de datos de prueba sólo se utiliza una vez que el modelo está completamente entrenado (utilizando los conjuntos de entrenamiento

y validación), como se observa en la Figura 2.25. La ***validación cruzada*** (***cross-validation***) es

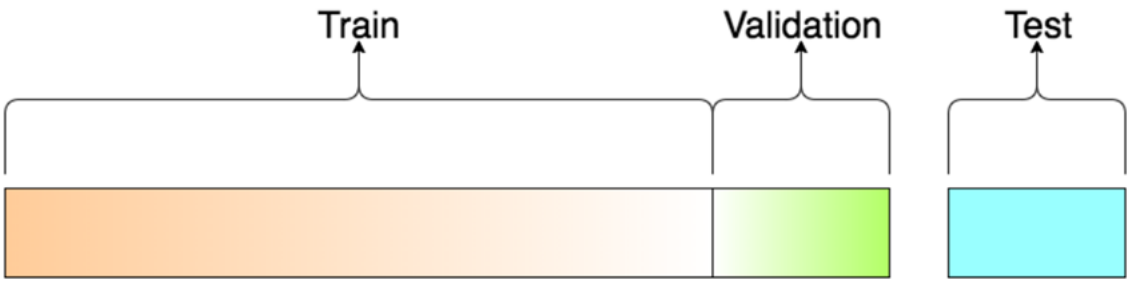


Figura 2.25 División recomendada del conjunto de datos. Fuente: [68].

otro método que se utiliza con frecuencia para barajar los datos que formarán los conjuntos de entrenamiento y prueba. Aquí, en lugar de crear una única división del conjunto global de datos, se aplican múltiples divisiones que producen diferentes subconjuntos o particiones. Para saber más sobre la ***validación cruzada*** recomiendo visitar [52] o [69].

Por otro lado, si se quiere profundizar conocimientos sobre los conjuntos de entrenamiento, visitar [71].

2.2.5.2 Problemas de sobreajuste y subajuste (overfitting and underfitting)

Los problemas de ***sobreajuste*** (***overfitting***) y de ***subajuste*** (***underfitting***) son problemas centrales en el ámbito del aprendizaje automático, ya que tienen un impacto directo en la fiabilidad de los modelos construidos, pudiendo estropear drásticamente su rendimiento cuando se les pone a prueba en entornos reales.

Denominamos ***sobreajuste*** al efecto de memorización del conjunto de entrenamiento. Esta sobreadaptación se produce cuando el modelo empieza a aprender las fluctuaciones aleatorias, anomalías y ruido que están presentes en el conjunto de datos que se le entrega. El resultado es que el modelo sobreajustado consigue describir perfectamente los datos con los que ha sido entrenado pero falla estrepitosamente cuando lo enfrentamos a datos no vistos.

Dos factores generales que pueden dar origen a los problemas de sobreajuste son el ***tiempo de entrenamiento*** y la ***complejidad del modelo***.

Finalmente, el problema del ***subajuste*** se produce cuando nuestro algoritmo es incapaz de modelar las tendencias presentes en los datos, dando como resultado un mal desempeño tanto en el conjunto de datos de entrenamiento como en el de prueba. Este problema surge cuando nuestra implementación es demasiado simple como para conseguir modelizar correctamente los datos. El remedio más directo es probar con otro algoritmo de aprendizaje automático. Se puede ver ambos errores en la Figura 2.26.

Por supuesto, producir un modelo que ajuste adecuadamente y consiga una correcta generalización de los datos debe ser nuestro objetivo, y será uno los principales retos a los que tendremos que enfrentarnos en un proyecto de aprendizaje automático.

Para saber más sobre los problemas de sobreajuste y subajuste, visitar [39] disponible en la bibliografía.

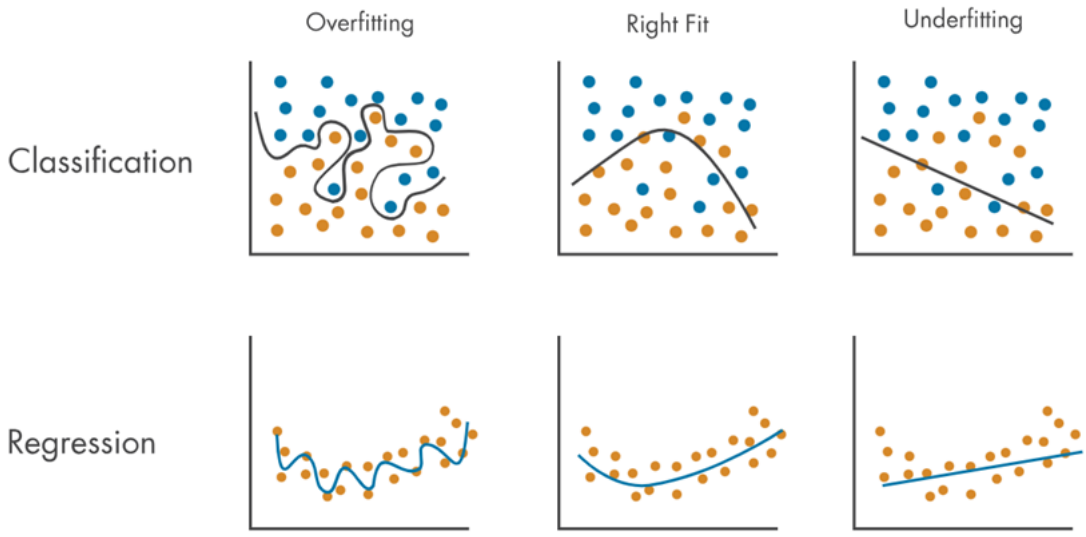


Figura 2.26 Ejemplificación de los problemas de sobreajuste y subajuste en ambos tipos de algoritmos del aprendizaje supervisado. Izquierda: sobreajuste. Centro: ajuste correcto. Derecha: subajuste. Fuente: [48].

En la detección rápida de los problemas de ajustes incorrectos nuestra mejor aliada es una buena métrica de rendimiento. Veremos a continuación cuáles son las más habituales.

2.2.5.3 Métricas de Desempeño

Podemos utilizar diferentes métricas para determinar si nuestro sistema de aprendizaje automático consigue realizar su tarea con eficacia. Muchas de estas métricas se basan en la medida del número de errores que comete nuestro modelo en sus predicciones, y que se asientan en los sesgos y varianza presentes en los datos. Un modelo con un sesgo elevado producirá errores similares para un cierto ***input*** independientemente del conjunto de entrenamiento utilizado; el modelo sesga sus propias suposiciones sobre la relación real al suponer que las relaciones que ha observado en los datos de entrenamiento son generales. Un modelo con una alta varianza se ajusta en exceso a los datos de entrenamiento, mientras que un modelo con un alto sesgo se ajusta por debajo de los datos de entrenamiento.

Ahora procederemos a ver algunas de las métricas más usadas, las cuales se verán de una resumida por ello si se quiere profundizar será necesario visitar [69] de la bibliografía.

Exactitud, Precisión y Exhaustividad

Como hemos comentado, los sistemas de aprendizaje automático deben evaluarse utilizando medidas de rendimiento que representen los costes asociados a los errores en el mundo real. Para entender mejor esta afirmación planteamos un ejemplo que describe el uso de una medida de rendimiento que resulta apropiada para la tarea general, pero no para su aplicación específica. Consideremos un problema de clasificación en el que un sistema de aprendizaje automático analiza señales en una red de telecomunicaciones para predecir la calidad de la conexión. La ***precisión***, definida aquí como la fracción de señales clasificadas correctamente, parece una medida muy intuitiva para valorar el desempeño del programa. Sin embargo, aunque esta ***precisión*** nos permite medir el rendimiento, no nos da información sobre el porcentaje de conexiones de baja calidad que el sistema clasifica como buenas, así como tampoco nos informa de cuántas conexiones buenas se identificaron como malas. Puede que en algunas aplicaciones los costes asociados a cualquiera de estos tipos de error sean equivalentes, pero está claro que tener una conexión de baja calidad tiene consecuencias más graves que identificar erróneamente una buena como mala.

Por tanto, es fundamental medir cada uno de los posibles resultados de la predicción para obtener diferentes perspectivas acerca del rendimiento de nuestro clasificador. En este sentido, cuando el

sistema clasifique correctamente una señal como de baja calidad, diremos que hemos obtenido un ***verdadero positivo***. Cuando se clasifique incorrectamente una señal de buena calidad como de baja calidad, estaremos ante un ***falso positivo***. Del mismo modo, un ***falso negativo*** es una predicción incorrecta de que la señal es buena, y un ***verdadero negativo*** es una predicción correcta de que una señal es buena. Estos cuatro resultados pueden utilizarse para calcular varias medidas comunes de rendimiento de la clasificación, como la ***exactitud*** (***accuracy***), la ***precisión*** (***precision***) y la ***exhaustividad*** (***recall***).

Obtenemos el valor de ***exactitud*** de nuestro modelo utilizando la fórmula siguiente:

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.10}$$

Calculamos el valor de ***precisión*** según:

$$P = \frac{TP}{TP+FP} \tag{2.11}$$

Y la ***exhaustividad*** la obtendríamos como:

$$R = \frac{TP}{TP+FN} \tag{2.12}$$

Donde:

- ***TP***: número de verdaderos positivos.
- ***FP***: número de falsos positivos.
- ***TN***: número de verdaderos negativos.
- ***FN***: número de falsos negativos.

Las medidas de ***precisión*** y ***exhaustividad*** nos pueden indicar que un clasificador alcanza una ***exactitud*** muy buena y sin embargo no consigue detectar la mayoría de señales de baja calidad.

Matriz de Confusión

Una matriz de confusión también nos permite describir el rendimiento de un modelo de clasificación resumiendo en un único cuadro las métricas relevantes. Vemos que no es más que un resumen de los resultados de las predicciones en un problema de clasificación. El número de predicciones correctas e incorrectas se recoge con valores de recuento y se desglosa por cada clase. La principal ventaja de esta representación es que nos revela de manera directa la forma en la que nuestro modelo de clasificación se confunde en sus predicciones. Nos informa de los errores que está cometiendo el clasificador y, lo que es más importante, de los tipos de errores en los que incurre. Un ejemplo de una matriz de confusión podría ser la de la Figura 2.27. Además, si se quiere profundizar más en los conocimientos que aquí se mencionan recomiendo visitar [2] de la bibliografía.

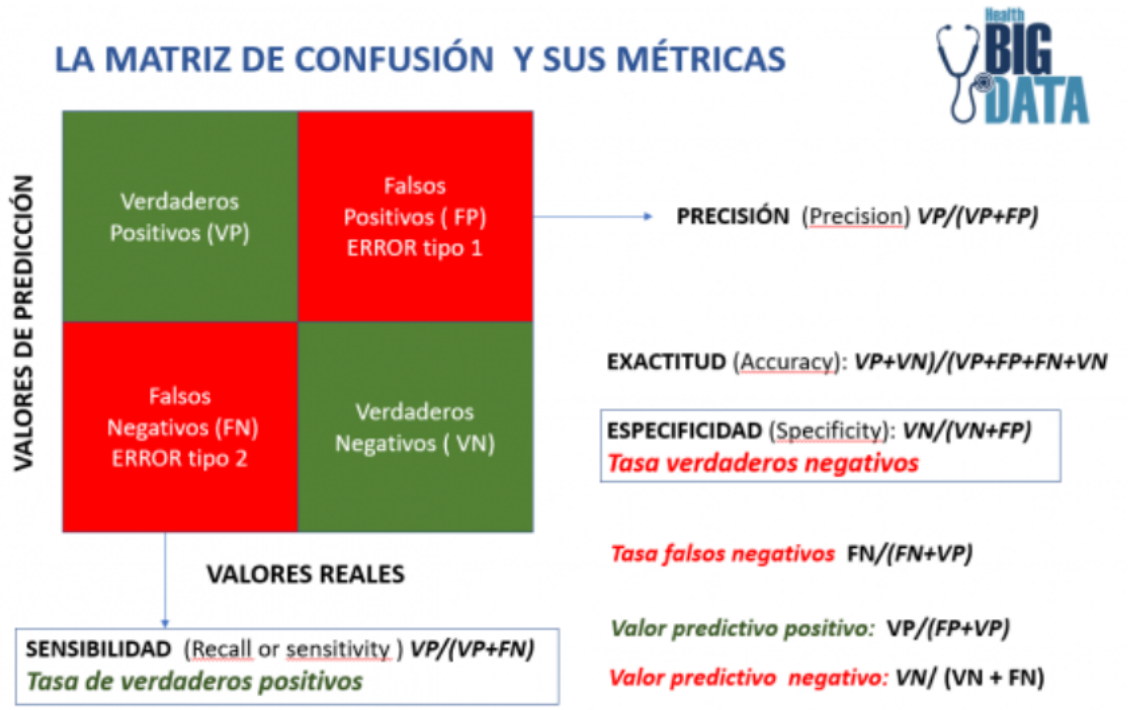


Figura 2.27 Matriz de confusión y sus métricas. Fuente: [2].

Curva ROC

Las curvas ***ROC*** (Receiver Operator Curve) son una forma de analizar visualmente la precisión en un problema de clasificación. Dado un cierto test o prueba diagnóstica, una curva ***ROC*** representa la relación entre la tasa de verdaderos positivos (sensibilidad) y la tasa de verdaderos negativos (especificidad) para los diferentes valores de corte (cut-off) que podemos definir dado el resultado numérico del parámetro que examina el test diagnóstico en cuestión. Para saber más, visitar [66] de la bibliografía.

2.2.5.4 Métricas de similitud

Este método suele usarse más en el ámbito de análisis de imágenes para poder identificar regiones o anomalías en ellas, o también para las cadenas de texto (parecido a cómo funciona ChatGPT). Por lo tanto, el uso de métricas rigurosas para evaluar la precisión y la sensibilidad de algoritmos de segmentación automática es de gran importancia.

Para profundizar en este tema y ver las distintas métricas de similitud más usadas actualmente, recomiendo visitar [37].

Ajuste de hiperparámetros

Como hemos visto, un modelo de aprendizaje automático consiste básicamente en un algoritmo matemático en el que una serie de parámetros se optimizan a partir de un conjunto de datos. Durante el proceso entrenamiento ajustamos los parámetros del modelo.

Pero en cualquier proyecto de aprendizaje automático encontraremos otro tipo de parámetros, conocidos como ***hiperparámetros***, que no pueden aprenderse directamente a partir del proceso de entrenamiento habitual. Estas cantidades, expresan propiedades importantes del modelo, como su complejidad o la rapidez con la que debe aprender, y deben ser fijadas antes de que comience el proceso de entrenamiento propiamente dicho. Aunque, normalmente, un ***hiperparámetro*** va a tener un efecto conocido en el modelo.

Algunos ejemplos de ***hiperparámetros*** básicos son los siguientes:

- El factor de penalización en un clasificador de regresión logística.
- La profundidad máxima de un árbol de decisión.

- La tasa de aprendizaje de entrenamiento (learning rate) en una red neuronal.
- Los hiperparámetros C y sigma para las máquinas de vectores de apoyo.
- El valor de k en un modelo de vecinos más próximos.

Vemos como, a la hora de construir nuestro modelo ideal de aprendizaje automático, podremos elegir entre diferentes opciones de diseño para definir su ***arquitectura***. Inicialmente lo habitual es que desconozcamos cuál es la arquitectura óptima, por lo que tendremos que explorar un espectro más o menos amplio de posibilidades. La filosofía del aprendizaje automático nos sugiere que encarguemos a la máquina la tarea de realizar dicha exploración y que se encargue de seleccionar los detalles (***hiperparámetros***) que configuren la arquitectura óptima de forma automática, denominado ***ajuste u optimización de hiperparámetros***.

Se pueden utilizar diferentes algoritmos de optimización, aunque dos de los métodos más sencillos y comunes son la ***búsqueda aleatoria*** (***random search***) y la ***búsqueda en cuadrícula*** (***grid search***).

La ***búsqueda en cuadrícula*** es ideal para comprobar combinaciones que se sabe que funcionan bien en general. La ***búsqueda aleatoria*** es la opción de referencia para descubrir y obtener combinaciones de ***hiperparámetros*** que no podríamos intuir fácilmente, a cambio requiere tiempo de ejecución más prologado.

Si se quiere profundizar más en los ajustes de hiperparámetros o bien en los dos algoritmos mencionados levemente aquí en el proyecto, se recomienda visitar [3] disponible en la bibliografía al final de éste.

2.2.5.5 Herramientas existentes en la programación

En este apartado se hará un breve repaso por las diferentes opciones que hay hoy en día para poder llevar la inteligencia artificial en la programación, en este caso, para Python. Las herramientas descritas aquí no solo serán librerías para poder implementar la IA en tu código, sino también para poder hacer tu código más eficiente y libre de errores y contribuir a mejorar la productividad y la calidad del software desarrollado. A continuación, se describen algunas de las herramientas más destacadas:

- ***TensorFlow y Keras***: son principalmente bibliotecas para el desarrollo de modelos de aprendizaje automático, también proporcionan herramientas que facilitan la programación en Python. Estas bibliotecas brindan estructuras prediseñadas y optimizadas para construir, entrenar y desplegar modelos de IA, simplificando tareas complejas que de otro modo requerirían un gran esfuerzo de codificación.
- ***Scikit-learn***: es una biblioteca popular de Python para el aprendizaje automático. Hay una amplia gama de algoritmos disponibles para tareas como la clasificación, la regresión, el agrupamiento y la reducción de dimensionalidad. Ella es ideal para principiantes y expertos en el aprendizaje automático porque es simple y consistente.
- ***PyTorch***: es una biblioteca de aprendizaje automático de código abierto desarrollada por Facebook's AI Research Lab. Debido a su flexibilidad y facilidad de uso, es especialmente popular en el ámbito de la investigación. PyTorch facilita la investigación y el desarrollo de aplicaciones de IA al proporcionar herramientas poderosas para construir y entrenar modelos de redes neuronales.

Para finalizar este apartado sobre las herramientas, indicar que en el código ha sido implementada la librería de TensorFlow para la inclusión de la Inteligencia Artificial en el código.

2.2.5.6 Monitorización

Hemos visto la importancia que tiene realizar un seguimiento de los parámetros que expresan el desempeño para encontrar el mejor modelo para nuestros datos. Para tener éxito en esta tarea es

fundamental ***monitorizar*** los valores de las métricas habituales (pérdida, precisión, coeficiente *DICE*, ...) y estudiar su convergencia realizando una serie de ***experimentos*** en los que variemos los valores que fijan los hiperparámetros de interés. En cada iteración nos interesará guardar toda información relevante que nos permita evaluar los resultados y poder obtener conclusiones sólidas.

En esta sección comentaremos una selección de las principales herramientas que nos facilitan esta tarea monitorización. No se va a profundizar a explicar cada una de esas herramientas, simplemente se indicará la referencia para que el lector pueda profundizar en ellas si lo desea.

NEPTUNE

Es como un almacén de metadatos derivados de un proyecto de desarrollo de IA. Esta herramienta nos permite supervisar, visualizar y comparar miles de modelos de aprendizaje automático en un solo lugar. Ofrece integraciones con más de 25 herramientas y librerías, incluyendo múltiples herramientas de entrenamiento de modelos y de optimización de hiperparámetros.

Veamos un resumen de las principales características que posee, si se quiere saber más sobre esta herramienta, recomiendo visitar [53].

- Posibilidad de registrar y mostrar múltiples tipos de metadatos.
- Flexibilidad para organizar los metadatos utilizados en desarrollo y en producción.
- Interfaz web de fácil navegación que permite comparar experimentos y crear cuadros de mando personalizados.

WEIGHT & BIASES (WANDB)

La plataforma Weight & Biases nos permite llevar a cabo el seguimiento de nuestros experimentos, y también gestionar el versionado del conjuntos de datos empleado y el diseño de los modelos.

Sus principales características serían las siguientes:

- Panel de control interactivo fácilmente configurable para organizar y visualizar los resultados del entrenamiento.
- Búsqueda de hiperparámetros y optimización de modelos con W&B Sweeps.
- Difusión y reduplicación de los conjuntos de datos registrados.
- Compatibilidad con diferentes entornos de trabajo y librerías, incluyendo:
  - Keras, visitar: [38].
  - PyTorch, visitar: [64].
  - TensorFlow, visitar: [73].
  - Fastai, visitar: [22].
  - Scikit-learn, visitar: [43].

Se invita al lector que si quiere saber más sobre esta plataforma visite: [7].

COMET

Es una plataforma destinada a facilitar el seguimiento, y optimización de experimentos y modelos de aprendizaje automático desde su composición inicial hasta la puesta en servicio. Esta herramienta nos permite registrar los conjuntos de datos empleados, cambios realizados en el código y el historial de los experimentos realizados.

Las principales características que posee COMET, son:

- Tabla de experimentos totalmente personalizable a través de su interfaz de usuario.
- Amplias funciones para realizar comparativas del código, hiperparámetros, métricas de evaluación, etc.

- Módulos dedicados que permiten identificar fácilmente posibles problemas en el conjunto de datos.

Para saber más sobre COMET, visitar [13].

TENSORBOARD

Es un kit de herramientas de visualización para TensorFlow ([73]) que permite analizar los resultados de la ejecución del proceso de entrenamiento de nuestros modelos. Es una herramienta de código abierto que ofrece funcionalidades básicas que abarcan todo el flujo de trabajo de un proyecto de aprendizaje automático. Aunque cuenta con una amplia comunidad de usuarios que puede facilitar la resolución de problemas, ofrece una experiencia de trabajo más adecuada para el usuario individual.

Aparte de lo mencionado, también posee las siguientes características:

- Seguir y visualizar métricas tales como la pérdida y la exactitud.
- Visualizar el grafo de nuestro modelo de aprendizaje profundo.
- Ver histogramas de pesos, sesgos y su evolución temporal.
- Mostrar imágenes, texto y datos de audio.

Para profundizar más sobre todas las capacidades que ofrece TensorBoard, recomiendo visitar: [72].

2.3 Lenguaje de Programación Python

En esta sección se explicará la elección de realizar este proyecto en Python y la adición del framework Django. Además, para este último se profundizará más para poder detallas aspectos importantes que se verán en el próximo capítulo de Diseño e Implementación.

Se ha seleccionado Python como base para este proyecto debido a mi familiaridad con este lenguaje y su amplio abanico de librerías, las cuales facilitan gran parte del diseño y desarrollo. Además, dado que se quería implementar una interfaz gráfica para la participación del usuario y se decidió desarrollar una página web, mi conocimiento previo en Django fue un factor decisivo en la toma de esta decisión.

Visto brevemente la razón por el uso de Python, comenzaré con la explicación de su framework Django.

2.3.1 Framework Django

Django es un framework de alto nivel para el desarrollo web en Python que fomenta un desarrollo rápido y un diseño limpio.

Explicar previo a las características de Django que al crear un proyecto, se crea una carpeta principal con el nombre del proyecto, posteriormente al iniciar el proyecto que se querrá crear, en mi caso sería el proyecto del blackjack (Django esto lo llama como aplicación). Por tanto, se nos quedarían dos carpetas en las cuáles tendremos algún archivo repetido cómo podría ser "***urls.py***" pero esto no provoca ningún conflicto ya que cómo se verá luego en el Apartado 2.3.1.2, en el archivo de "***urls.py***" de la carpeta principal del proyecto de Django se incluirán todas las URL que creemos en el archivo de la carpeta de nuestro proyecto.

A continuación, describiré algunas de las características y herramientas de Django más destacadas.

2.3.1.1 Bases de Datos y Django

La gestión de los datos es una parte crucial en las aplicaciones web, y Django ofrece una integración robusta con bases de datos (BBDD) a través de su sistema de ***mapeo objeto-relacional*** (ORM, por sus siglas en inglés), gracias al ORM, los desarrolladores pueden usar código Python para interactuar con la base de datos.

Django soporta múltiples sistemas de gestión de bases de datos (DBMS) como PostgreSQL, MySQL, SQLite y Oracle. La configuración de la base de datos se especifica en el archivo ***settings.py*** del proyecto, donde se define el motor de base de datos a utilizar, junto con otros detalles.

A continuación, se mostrarán otras características que posee Django para hacer la integración con las BBDD:

- ***Migraciones***: es el sistema que usa Django para gestionar los cambios en la estructura de la base de datos.
- ***Seguridad***: el ORM de Django incluye protecciones integradas contra los ataques más comunes a bases de datos (como puede ser inyección SQL).
- ***Mantenibilidad***: gracias a la abstracción que proporciona el ORM de Django y el sistema de las migraciones, permiten tener una gestión más cómoda de la base de datos, lo que provoca una mantenimiento más a largo plazo.
- ***Transacciones y Atomicidad***: Django soporta transacciones de base de datos, permitiendo a los desarrolladores agrupar operaciones en bloques atómicos que se completan completamente o se revierten en caso de error.

Vista la integración que nos otorga Django con las BBDD, veamos cómo funciona el enrutamiento que tiene Django con la página web a crear.

2.3.1.2 Sistema de Enrutamiento y URL

El sistema de enrutamiento y gestión de URL en Django es una de las características clave que permite a los desarrolladores definir cómo las solicitudes HTTP son manejadas por la aplicación web. Este sistema facilita la creación de rutas limpias y lógicas que son fáciles de entender y mantener.

Definición de una URL

Las rutas de URL se definen en el archivo "***urls.py***". Este archivo contiene patrones de URL que se asocian con vistas específicas, que son funciones o clases que procesan las solicitudes y devuelven respuestas. Los patrones de URL se definen utilizando expresiones regulares o la sintaxis de ***path***, lo que permite una gran flexibilidad en la definición de rutas.

Espacio de nombre de una URL

Django permite agrupar URL utilizando espacio de nombre (agrupando URL bajo un mismo nombre), lo que es especialmente útil para aplicaciones grandes o proyectos con múltiples aplicaciones. Los espacios de nombre ayudan a evitar conflictos de nombres y a organizar mejor las rutas.

Un ejemplo completo de la definición de una URL y un espacio de nombre, lo podemos encontrar en el siguiente apartado de código (Código 2.1):

Código 2.1 Definición de una URL.

```
# En la carpeta de authors (equivale a mi carpeta blackjack), archivo urls.py
from django.urls import path
from . import views

app_name = 'authors' # declaración del espacio de nombre
```

```
urlpatterns = [
    path('', views.indesx, name='index'),
    path('detail/<int:id>/', views.detail, name='detail'),
]

# En el archivo urls.py de la carpeta principal (equivale a mi carpeta project)
from django.urls import include, path

urlpatterns = [
    path('authors/', include('authors.urls')), #incluimos las URL declaradas
        antes
]
```

Aquí tendríamos lo siguiente:

- La definición del namespace, que sería "authors".
- Las URL de la carpeta "authors", que muestra:
  - Una URL hacia una página principal, asociada a la vista index.
  - Una URL hacia una página que muestra detalles, a la que se le pasará un parámetro que es un identificador (dicho parámetro estará indicado en la definición de la vista asociada, en este caso, detail).
- También se muestra la definición de una URL en el archivo principal del proyecto de Django, que se encarga de incluir el archivo de ***urls.py*** de la carpeta "authors". Esto quedará más aclarado cuando veamos el Apartado 3.1.2.

En resumen, el sistema de enrutamiento y URL de Django facilita la definición de rutas claras y lógicas dentro de una aplicación web. La capacidad de capturar parámetros de URL, utilizar vistas basadas en clases y organizar URL con namespaces, proporciona un control sobre cómo se manejan las solicitudes HTTP, mejorando tanto la estructura como la mantenibilidad del código.

Explicado el sistema de enrutamiento que proporciona Django, se pasará a explicar la arquitectura que sigue, profundizando en cada una de sus partes.

2.3.1.3 Arquitectura Modelo-Vista-Plantilla

Este patrón de diseño es similar al modelo Modelo-Vista-Controlador (MVC). Aquí se separa la lógica de la aplicación (Modelos, archivo ***models.py***) de la interfaz de usuario (Vistas, archivo ***views.py***) y la lógica de presentación (Plantillas, carpeta ***templates***), lo que facilita la gestión y el mantenimiento del código. Los Modelos definen la estructura de la base de datos, las Vistas controlan la lógica de la aplicación y las Plantillas determinan cómo se presenta la información al usuario.

Modelos de Django Los modelos en Django son una parte fundamental del framework y desempeñan un papel crucial en la definición y gestión de la estructura de datos de una aplicación web. En esencia, un modelo en Django es una clase de Python que representa una tabla en la base de datos. Cada atributo de la clase corresponde a un campo de la tabla. Los modelos de Django proporcionan una forma intuitiva y eficiente de trabajar con bases de datos sin tener que escribir SQL directamente.

Veamos las características principales que poseen:

- ***ORM***: cómo hemos visto ya, Django proporciona un ORM que permite interactuar con las bases de datos utilizando los objetos de Python, pues esos objetos son los modelos. Por tanto, gracias a los modelos podemos interactuar directamente con la base de datos sin necesidad de escribir código SQL.

- ***Tipos de Campos***: Django proporciona una amplia variedad de tipos de campos que se pueden utilizar para definir los atributos de los modelos, tales como CharField, IntegerField, DateField, ForeignKey, entre otros. Cada tipo de campo tiene opciones específicas que permiten personalizar su comportamiento y validación.
- ***Relaciones entre Modelos***: Los modelos de Django soportan varios tipos de relaciones entre tablas, incluyendo relaciones uno a uno (OneToOneField), uno a muchos (ForeignKey) y muchos a muchos (ManyToManyField). Estas relaciones permiten estructurar datos complejos y modelar interacciones entre diferentes entidades.
- ***Migraciones***: como se ha visto, las migraciones se usan para mantener la base de datos actualizada, es por eso que es una buena práctica (y bastante común) realizar migraciones cada vez que se modifica algún modelo.
- ***Métodos de Modelo***: Además de los atributos, los modelos de Django pueden incluir métodos personalizados que encapsulan lógica relacionada con los datos. Esto permite definir comportamientos específicos y operaciones comunes que se pueden realizar sobre los objetos del modelo.

Explicadas las principales características que poseen los modelos de Django, mostremos un código (Código 2.2) que abarque todas ellas de manera que queden más claras:

Código 2.2 Definición de un modelo.

```
# En el archivo models.py de la carpeta authors (equivale a mi carpeta
    blackjack)
class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    published_date = models.DateField()
    isbn = models.CharField(max_length=13, unique=True)

    def __str__(self):
        return self.title

    def was_published_recently(self):
        return self.published_date >= tiemzone.now() - datetime.timedelta(days
            =1)
```

Por despejar dudas con los tipos de campos, tendríamos:

- ***CharField***: un campo de texto, formato de cadena de caracteres (string), dónde se le puede indicar por ejemplo la longitud máxima permitida que tendrá, en este caso, el título. También, se le puede indicar que ese campo sea único, convirtiéndolo en el identificador de ese modelo/tabla de la base de datos.
- ***ForeignKey***: campo que será la clave externa del modelo "Author", se estaría indicando que 1 "author" posee muchos "book". Además, Django también nos ofrece la posibilidad de decidir qué va a ocurrir con los objetos del modelo "book" que pertenezcan a un objeto del modelo "author" que sea borrado de la base de datos, aquí se mantiene la lógica que tiene SQL y por tanto, con ***CASCADE*** se indica que también se borrarán de la base de datos.
- ***DateField***: campo de tipo fecha, en este caso, sólo indica la fecha en formato dd/mm/yyyy.

Explicada la parte de los modelos, pasaremos a describir la parte de las vistas del patrón de diseño que posee Django.

Vistas

Las vistas en Django son una parte esencial del framework que determinan qué datos se envían al navegador y cómo se presentan. En términos simples, una vista en Django es una función o una clase que toma una solicitud web y devuelve una respuesta web. Las vistas actúan como el punto intermedio entre los modelos (la lógica de datos) y las plantillas (la lógica de presentación). Existen dos tipos:

- ***Basadas en Funciones*** (FBV), que reciben un objeto HttpRequest y devuelven un HttpResponse. Son realmente útiles para tareas simples y directas.
- ***Basadas en Clases*** (CBV), son clases de Python que proporcionan una manera más estructurada y orientada a objetos de manejar las solicitudes HTTP. Permiten reutilizar código y mejorar la organización de la lógica de vistas, especialmente en aplicaciones grandes y complejas.

A continuación, veamos como funcionan:

1. Reciben la solicitud HTTP cuando un usuario accede a una URL (dicha URL está asociada a la vista en el archivo "***urls.py***").
2. Procesa la solicitud, lo que puede incluir la recuperación de datos de la base de datos, la validación de formularios, etc.
3. Genera una respuesta, generalmente utilizando una plantilla para renderizar los datos en HTML. La respuesta se devuelve como un objeto HttpResponse.

Dentro de la generación del objeto HttpResponse que genera la vista como respuesta hacia una plantilla, existen varias maneras de devolver dicho objeto, veamos las más importantes:

- ***HttpResponse***: devuelve directamente el objeto HttpResponse.
- ***JsonResponse***: para devolver respuestas en formato JSON más fácilmente.
- ***render***: el más utilizado, sirve para envolver una plantilla en un objeto HttpResponse y así cargar la plantilla en la página web.
- ***redirect***: usado para redirigir a una URL o vista en concreto.
- También existen maneras de enviar objetos HttpResponse con códigos de error cómo podrían ser un 404, 400, 403, etc.

Veamos un ejemplo muy básico de una vista para afianzar un poco lo explicado (Código 2.3):

Código 2.3 Definición de una vista.

```
# En el archivo views.py de la carpeta authors (equivale a mi carpeta blackjack
    )
from django.shortcuts import render
from .models import Book

def book_list(request):
    books = Book.objects.all() # se recuperan todos los libros
    # se renderizará la plantilla book_list.html para mostrar todos los libros
        obtenidos
    return render(request, 'books/book_list.html', {'books': books}
```

Aquí tendríamos que pasaríamos todos los objetos del modelo "Book" a la plantilla de "book_list.html" con el uso de ***render***.

Terminada la parte de las vistas, daremos paso a la última parte de la arquitectura que posee Django, explicando las plantillas.

Plantillas

Las plantillas permiten separar la lógica de presentación de la lógica de datos. El sistema de plantillas de Django permite a los desarrolladores crear documentos dinámicos y reutilizables que combinan contenido estático y dinámico, facilitando la generación de HTML de manera eficiente. Veamos cuáles son las características más importantes:

- ***Lenguaje de Plantillas***: es un lenguaje propio que ofrece Django para las plantillas, este lenguaje proporciona un conjunto de etiquetas, filtros y variables que se pueden utilizar para insertar datos dinámicos en las plantillas y controlar la lógica de presentación. Las variables que son utilizables han sido previamente usando la vista mediante un diccionario (que suele ser llamado "context").
- ***Herencia de Plantillas***: permite definir una estructura base común y extenderla en plantillas secundarias. Esto facilita la creación de un diseño coherente en toda la aplicación sin repetir código.

Un ejemplo de una plantilla de Django podría ser el siguiente código(Código 2.4):

Código 2.4 Definición de una plantilla.

```
# En el archivo book_list.html
# se extiende una plantilla base que posee todos los estilos de la página web
{% extends "base.html" %}

# lenguaje de plantillas de Django para incluir el título de la página web
{% block title %}Book List{% endblock %}

# lenguaje de plantillas de Django para incluir el código que vendrá a
    continuación en la plantilla base.html
{% block content %}
    <h2>Book List</h2>
    <ul>
        {% for book in books %} # recorremos todos los libros obtenidos en la
            vista
            <li>{{ book.title }} by {{ book.author.name }}</li>
        {% endfor %}
    </ul>
{% endblock %}
```

En esta figura, se ve el uso de las herencias de plantillas, heredando de una plantilla llamada "base.html", también tenemos el uso de las variables que obtuvimos en una vista para poder mostrar todos los objetos del modelo "Book" en la plantilla.

Si se quiere profundizar más en todo lo que ofrece Django, recomiendo visitar las siguientes referencias: [15] y [77] disponibles en la bibliografía.

3 Diseño e Implementación

En este capítulo se hará una explicación de cómo se ha desarrollado todo el código para llevar a cabo el funcionamiento del proyecto. Se hará una división en dos apartados:

- ***Diseño***: en este primer apartado se explicará toda la parte del diseño del proyecto, donde tendremos:
  - Explicación de la Arquitectura General que posee el proyecto.
  - Formato y Diseño que tendrá la parte de Backend del proyecto, incluyendo una explicación de la base de datos a utilizar y el diagrama de clases que tendríamos.
  - Formato y Diseño que tendrá la parte de Frontend, incluyendo la estructura de los archivos y plantillas que se usen.
  - Diseño de los modelos de Inteligencia Artificial, qué datos se usarán para el entrenamiento de los modelos (que se verán en el apartado de Implementación).
- ***Implementación***: para este apartado, se hará una explicación general (y detallada cuando sea necesaria) de la implementación del diseño, es decir, todo el código que posee el proyecto (al menos lo más importante). Por tanto, este apartado se dividirá a su vez en tres bloques:
  - ***Backend***: explicación de cómo se ha implementado todo el funcionamiento del proyecto, las diferentes clases creadas y funciones usadas. Así como también el uso de las diferentes librerías de Python para lograr el desempeño deseado.
  - ***Frontend***: explicación de cómo se ha implementado la página web en la que se alojará el proyecto, utilizando como base el framework Django de Python con la adición de HTML, CSS, Javascript y el uso de Ajax con la ayuda de la librería HTMX.
  - ***Inteligencia Artificial***: será una ampliación de lo visto en el diseño, aquí se explicarán los modelos creados y por qué.

3.1 Diseño

Cómo se ha comentado, aquí se verá el diseño del proyecto desde la arquitectura que se tiene (incluyendo la estructura de los archivos que se posee) hasta el formato que se va a seguir para la parte del Backend y Frontend.

Comencemos viendo el apartado de la arquitectura general.

3.1.1 Arquitectura General

Procedemos a explicar la arquitectura general que tiene el proyecto, para ello echemos un vistazo a la Figura 3.1. En ella podemos observar que tendremos una máquina que hará las funciones de

cliente y otra máquina que hará de servidor. Si bien para el alcance de este TFG nos quedaremos con una aplicación web que corra nuestro programa alojando el servidor en mi propia máquina. Echando un vistazo a la figura (Figura 3.1), nos encontramos que en la máquina servidor se aloja

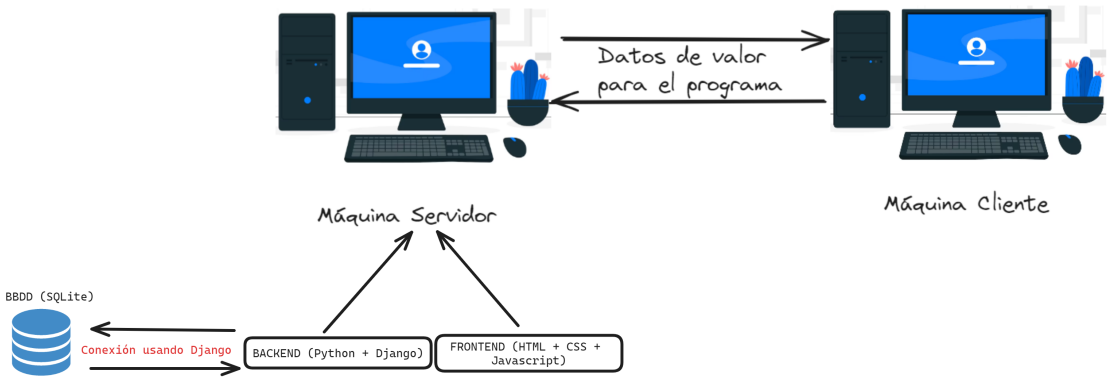


Figura 3.1 Arquitectura General del Proyecto.

el frontend para la página web, el backend para manejar los datos y aparte, una base de datos para guardar datos.

Vista en primera instancia lo que es la arquitectura general del proyecto, se pasará a continuación a mostrar la estructura de directorios del proyecto para después poder dar inicio a una explicación más profunda del diseño de los componentes vistos en la arquitectura.

3.1.2 Estructura de Directorios

Como se ha mencionado, aquí se verá la estructura de directorios que tiene el proyecto y se dará posteriormente una explicación de por qué se usa esa estructura. Con la creación de un proyecto de Django, se forma la siguiente estructura:

```
/
|-- blackjack
|-- project
|-- db.sqlite3
|-- manage.py
```

Donde nos encontraremos con dos carpetas:

- La carpeta del proyecto, ***blackjack***.
- La carpeta principal del proyecto que proporciona Django, ***project***, esta carpeta es creada automáticamente por Django al iniciar el proyecto para poder guardar los archivos relacionados con las configuraciones y el sistema de enrutamiento base.
- La base de datos que utilizaremos, ***db.sqlite3***, en este caso estamos usando SQLite.
- Un script que proporciona Django para poder interactuar con el proyecto, ***manage.py***.

Se pasará ahora a ver la carpeta que proporciona Django con la creación del proyecto, ***project***, que contiene lo siguiente:

```
/
|-- project
|   |-- pycache
```

```
|   |-- __init__.py
|   |-- asgi.py
|   |-- settings.py
|   |-- urls.py
|   |-- wsgi.py
```

Aquí lo que nos encontraremos son archivos que tienen que ver con la configuración principal del proyecto o configuraciones adicionales para posibles migraciones de servicio:

- ***settings.py***: archivo donde se guarda toda la configuración de Django para el proyecto, aquí se harán modificaciones relevantes con la base de datos.
- ***urls.py***: archivo de las URL principales del proyecto, para el alcance de este TFG el único uso que tiene es incluir el archivo de la carpeta del proyecto (***blackjack***), esto se verá más en profundidad en el apartado de la Implementación (Apartado 3.2.1.5).
- ***asgi.py***: archivo dónde están las configuraciones para una posible migración a ASGI (Asynchronous Server Gateway Interface).
- ***wsgi.py***: archivo dónde están las configuraciones para una posible migración a WSGI (Web Server Gateway Interface).

Por último, se verá la estructura que posee la carpeta ***blackjack***:

```
/
|-- blackjack
|   |-- pycache
|   |-- datasets
|   |-- migrations
|   |-- modelos
|   |-- static
|   |-- templates
|   |-- __iinit__.py
|   |-- admin.py
|   |-- apps.py
|   |-- create_q.py
|   |-- forms.py
|   |-- models.py
|   |-- process_data.py
|   |-- table_strategies.py
|   |-- tests.py
|   |-- urls.py
|   |-- utils.py
|   |-- views.py
```

Ahora se procederá a explicar que es lo que se tiene en esta carpeta ***blackjack***:

- Carpeta ***datasets***: se guardarán todos los sets de datos que se vayan creando para el entrenamiento del modelo de IA.
- Carpeta ***migrations***: tendremos un resumen de las migraciones a la base de datos que se hayan hecho.
- Carpeta ***modelos***: se almacenan tanto los modelos de Inteligencia Artificial que se creen como las Tablas Q ya creadas.

- Carpeta ***static***: es donde se guardan todas las imágenes que se utilizan en la página web.
- Carpeta ***templates***: aquí, como se explicó en el Estado del Arte (Apartado 2.3.1.3), tendremos almacenadas todas las plantillas que son usadas para la creación y diseño de la página web.
- Archivo ***admin.py***: configuración del portal de administrador que proporciona Django con la creación del proyecto.
- Archivo ***apps.py***: configuraciones más específicas que se aplican sobre el proyecto.
- Archivo ***forms.py***: creación de las clases de los formularios para la página web.
- Archivo ***models.py***: donde se crean los modelos de Django (más información en el Apartado 2.3.1.3).
- Archivo ***urls.py***: definición de todas las URL de la aplicación.
- Archivo ***utils.py***: es donde tendremos definidas la gran mayoría de las funciones usadas en el proyecto.
- Archivo ***views.py***: donde se definen las vistas de Django para el proyecto (más información en el Apartado 2.3.1.3).
- Archivos varios para el backend y frontend del proyecto, se verán en sus respectivos apartados en la sección de Implementación.

Habiendo abarcado toda la estructura de directorios del proyecto (en ciertas carpetas y archivos se profundizará más en futuros apartados), pasaremos a dar una explicación del diseño adoptado para la parte del backend.

3.1.3 Diseño del Backend

El objetivo principal que se quiere lograr con el diseño de la parte del backend es poder crear el juego del blackjack y a su vez se puedan crear diferentes modelos de Inteligencia Artificial que actúen como jugadores para la partida. Para poder alcanzar este objetivo ha sido necesario un diseño previo de cómo afrontar las diferentes necesidades que debía abastecer esta parte.

Entonces, sabiendo el objetivo que se tiene, se pasará a explicar los diferentes componentes que se han creado y utilizado para poder lograrlo. Cabe recalcar que aquí sólo se explicará diseño y todo el código usado se verá posteriormente en el apartado de Implementación.

3.1.3.1 Modelos

Se han creado dos modelos de Django, uno llamado Deck que hará las funciones del mazo de cartas con el que se jugará la partida y otro llamado Player para la creación de los jugadores. Veamos dichos modelos en un diagrama de clases para mostrar la relación que existe entre ambos (Figura 3.2):

El modelo Deck no estará conectado con la base de datos y el formato que manejará para las cartas será: "valor-palo". El valor de la carta estará representado por un solo carácter y abarcará todos los valores de la baraja francesa: "A, 2, ..., J, Q, K". Para el palo, se indicará con una letra en mayúsculas que serán: "H, D, C, S", que se corresponde con la primera letra de cada palo en inglés: "Hearts, Diamonds, Clubs, Spades".

El modelo Player, por el contrario, sí estará conectado con la base de datos. Que esté conectado con la base de datos es simplemente para poder mantener la información de los jugadores, de esta manera, si se decide cerrar la página web, los jugadores que se crearon se mantendrán guardados.

Por tanto, nos encontramos con que el ciclo de vida de un objeto de la clase Player (un jugador) sería el siguiente: se crea, se guarda en la base de datos y finaliza cuando se dé la orden de borrarlo. Sin embargo, eso no es algo óptimo ya que se saturaría en algún punto la base de datos de

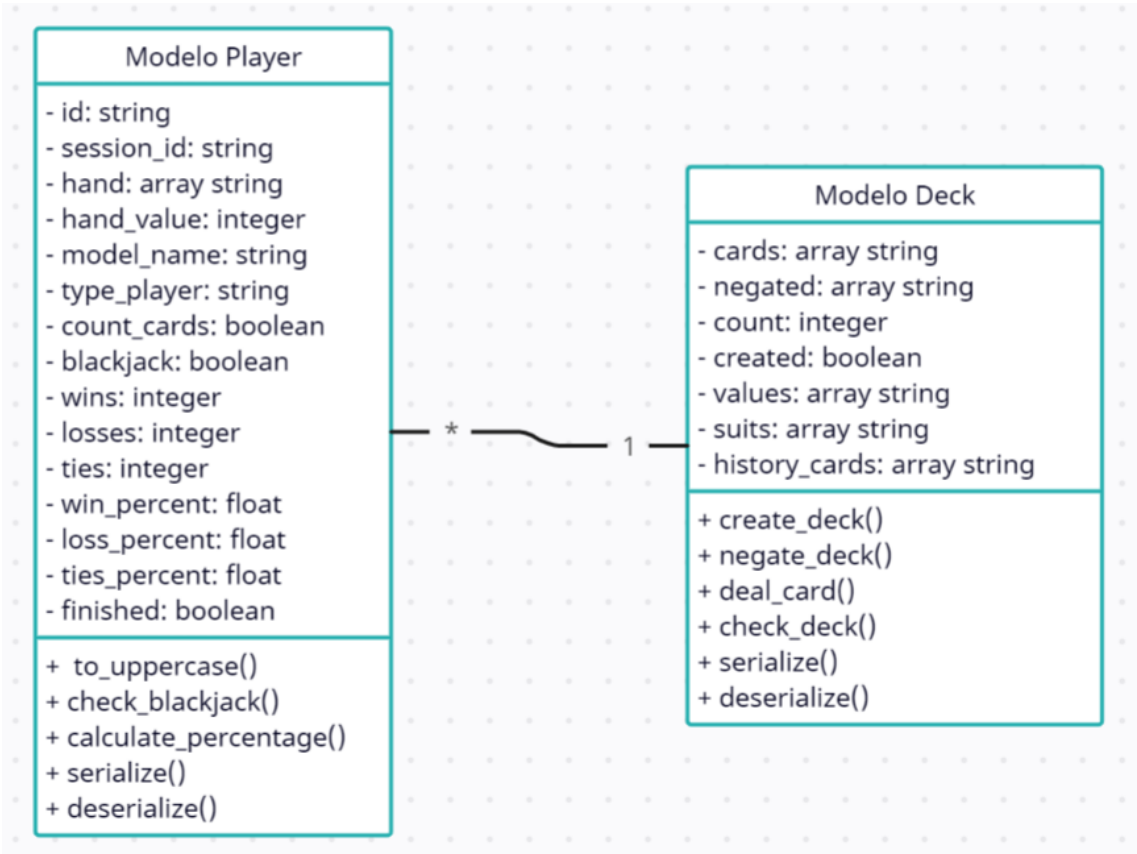


Figura 3.2 Diagrama de Clases.

tantos jugadores creados, por ello se le pondrá una cuenta atrás para que al pasar cierto tiempo (se considerará de inicio 7 días) se borre de la base de datos y con esto se cierre su ciclo de vida.

Dentro del modelo Player, existe un campo llamado "***type_player***" que será usado para indicar el tipo de jugador que se quiere crear, a elegir entre las siguientes variantes:

- ***IA***: se cargará un modelo de Inteligencia Artificial a elección por el usuario, incluyendo si se quiere que cuente cartas o no.
- ***Q-Learning***: se cargará una Tabla Q a elección por el usuario, a elegir entre 4 tablas donde difieren el número de partidas simuladas.
- ***NPC*** (Non Playable Character): jugador al que se le podrá configurar que cuente cartas (aplicando el algoritmo visto en el Apartado 2.1.3.3) o que tenga acceso a una tabla de estrategias para tomar la decisión.

Además de esos tres tipos de jugadores, existen también un tipo "***USER***" y un tipo "***DEALER***", que hacen referencia al usuario que juega la partida y al crupier de la partida. Estos dos tipos de jugadores no tendrán un diseño ni una lógica muy complicada, de todos modos se verán más adelante. Ahora pasaremos a explicar el diseño de los jugadores de tipo: AI, Q-LEARNING y NPC.

Tipo IA

En el jugador de Inteligencia Artificial se va a cargar un modelo para que se encargue de tomar las decisiones en la ronda y para ello será necesario crearlo y entrenarlo. Sin embargo, previo a la creación y entrenamiento del modelo hay que crear los datos con los que se entrenará.

En esta parte es donde se va a ver el diseño de los datos de entrenamiento que toma el modelo de Inteligencia Artificial. El set de datos que se usa para ello se guardará en dos archivos con dos terminaciones distintas, veámoslas:

- Terminación ***.data***: en este archivo se van a guardar los datos que se generan, éstos datos han sido decididos por mí tras varios entrenamientos con diferentes combinaciones y los que han salido mejor representados han sido:
  - Valor de la mano del jugador, es necesario que el jugador IA tome las decisiones en razón al valor de su mano.
  - Si la mano del jugador posee un AS que es evaluable como 11.
  - Valor de la carta visible del crupier, al igual que con la valoración de su mano, es de vital importancia saber qué carta tiene el crupier.
  - En caso de que el modelo a entrenar sea pensando en la opción de contar cartas, será necesario pasar también un historial de las cartas que han ido saliendo. Este cuarto parámetro, por tanto, sólo aparece cuando el set de datos sea pensado para el conteo de cartas.
- Terminación ***.tags***: este archivo guarda la decisión que se toma en cada ronda, llamadas etiquetas. Estas etiquetas siguen la siguiente lógica:
  - Si se ha pasado de 21, quiere decir que no tendría que haber pedido la última carta. Guardamos la etiqueta 's'== Quedarse.
  - Si la mano del jugador no supera 21 pero la del crupier sí, quiere decir que hizo bien en no pedir una carta más. Guardamos la etiqueta 's'== Quedarse.
  - Si la mano del jugador no supera la puntuación de la mano del crupier, quiere decir que debería de haber pedido carta. Guardamos la etiqueta 'h'== Pedir carta.
  - Si la mano del jugador supera la puntuación de la mano del crupier, quiere decir que hizo bien en no pedir una última carta. Guardamos la etiqueta 's'== Quedarse.
  - Si la mano del jugador empata en cuanto a puntuación a la mano del crupier, quiere decir que debería de haber pedido una carta más para intentar ganar. Guardamos la etiqueta 'h'== Pedir carta.

Visto que se guarda en los archivos, hace falta explicar cómo se van a generar estos datos y la respuesta es simple, se simularán tantas partidas como se quieran entre un jugador que toma decisiones al azar (básicamente sería un cara/cruz) contra el crupier. La única interacción que tengo ahí como desarrollador es de aplicar la lógica de las etiquetas y guardar los datos necesarios de la ronda en los archivos explicados previamente.

Un ejemplo de ambos archivos explicados, lo podríamos encontrar en las siguientes imágenes (Figura 3.3). Explicada la lógica utilizada para la creación de los sets de datos para el modelo que

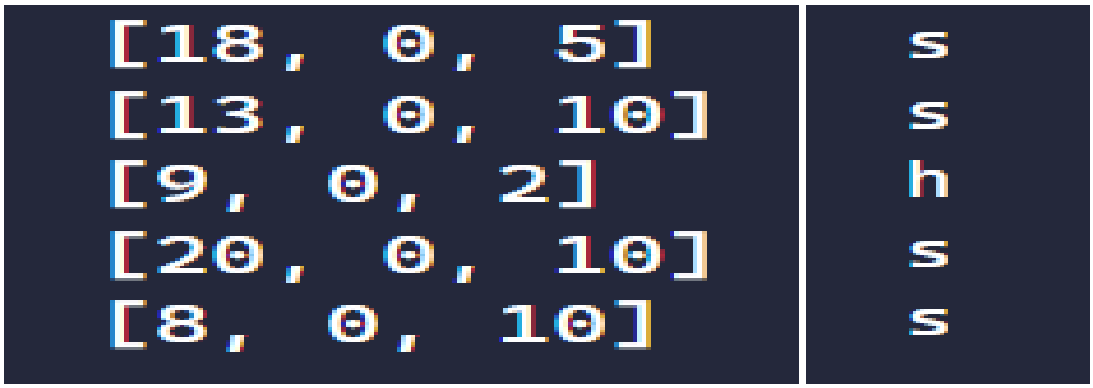


Figura 3.3 Archivos de sets de datos. Izquierda: archivo.data ; Derecha: archivo.tags.

usará el jugador IA, se pasará a ver la parte del diseño para el jugador Q-Learning.

Tipo Q-Learning

El siguiente tipo de jugador es el basado en el algoritmo de Q-Learning el cuál toma las decisiones usando el método Monte Carlo, ambos conceptos han sido vistos ya en el Estado del Arte (Apartado 2.2.2.7 y Apartado 2.2.2). Por tanto, procederemos a explicar la lógica para la creación de la Tabla Q.

Sabiendo la lógica que sigue el algoritmo Q-Learning, veamos el diseño que se ha seguido para ello, comenzando por las variables usadas:

- ***Variable*** ε: es la manera de indicar la aleatoriedad con la que queremos que se tomen las decisiones. Veamos como afecta esto a la Q:
  - ***Valor cercano a 1***: hará selecciones aleatorias con mayor frecuencia esto puede ser válido en las primeras etapas para que explore rápidamente todo el espacio de posibilidades.
  - ***Valor cercano a 0***: hará selecciones aleatorias con menor frecuencia, fijándose mucho más en los conocimientos que ha ido adquiriendo a lo largo de las simulaciones.
- ***Variable*** α: es la manera en la que indicamos la tasa de aprendizaje, es decir, con cuál frecuencia iremos actualizando los valores de la tabla Q. Veamos como afecta el valor que tenga al aprendizaje de la Q:
  - ***Valor cercano a 1***: más rápido se adaptará porque hará cambios más a menudo pero puede provocar problemas de inestabilidad.
  - ***Valor cercano a 0***: actualizará los valores más lentamente asegurándose una convergencia segura y estable pero demorará mucho más tiempo para aprender y actualizar los valores.
- ***Variable*** γ: es el factor de descuento, esto indica como valoramos las recompensas futuras con respecto a las cercanas. Veamos como influye los valores que tome en esas decisiones:
  - ***Valor cercano a 1***: toma con igual importancia las recompensas futuras con respecto a las inmediatas, esto es útil en problemas donde las acciones pasadas tienen peso en el futuro.
  - ***Valor cercano a 0***: ignora prácticamente las recompensas futuras, poniendo toda su atención en las recompensas inmediatas, por tanto no jugará con vistas a futuro sino con vistas al presente.

Vistas las variables, pasemos con la lógica de la creación de la Tabla Q:

- El estado que tendrá la tabla estará formado por:
  - Valoración en formato entero de la mano del jugador.
  - Valoración en formato entero de la carta visible del crupier.
  - Valoración en formato entero de si el jugador posee un AS en su mano evaluable como 11.
- El espectro de acciones que tiene es:
  - Pedir carta, representado como "h".
  - Quedarse, representado como "s".
- Las recompensas que puede obtener el jugador serán:
  - Si gana, obtiene una recompensa positiva con valor 1.

  - Si pierde, obtiene una recompensa negativa con valor -1.
  - Si empata, no obtiene ninguna recompensa, esto será representado con un 0.

Explicadas la lógica que sigue para la creación, incluyendo las variables, falta informar de que dicha creación se hará con un script en Python externo al fichero principal, por tanto, la tabla Q se guardará en un archivo y será cargada en el jugador para que tome las decisiones.

Dicho archivo será guardado haciendo uso de la librería ***pickle*** de Python, se ha elegido esta librería por la posibilidad que otorga de serializar (y posteriormente deserializar) los objetos de Python, permitiendo así su persistencia en archivos.

Descrito el diseño del jugador de tipo Q-Learning, se dará paso al jugador tipo NPC.

Tipo NPC

Para el jugador NPC se han creado dos opciones:

- Contador de cartas, usando el algoritmo Hi-Lo.
- No contador de cartas, usando una tabla de estrategias.

Comencemos viendo el diseño del jugador que sabe contar cartas, este diseño es muy simple ya que únicamente será necesario indicarle la cuenta que lleva (que está asociada al deck) y se aplicará la lógica del algoritmo Hi-Lo para actualizar el valor de la cuenta. Con el valor de la cuenta actualizada, se aplica una lógica de decisión (ha sido evaluada por mí, después de jugar cientos de partidas y ayudándome de la tabla de Estrategia Básica: Figura 2.5), esta lógica es la siguiente:

- Si el valor de la mano del jugador es menor de 11, siempre pedirá carta.
- Si el valor de la mano del jugador se encuentra entre 12 y 16, comprobamos la cuenta:
  - Si su valor es inferior a 0, quiere decir que han salido más cartas altas que bajas, y por tanto, pediremos carta, porque será menos probable pasarse de 21.
  - Si su valor es superior a 0, quiere decir que habrán salido más cartas bajas que altas, y por tanto, no pediremos carta, porque aumentan las posibilidades de pasarse de 21.
- Si el valor de la mano del jugador es superior a 16, no se pedirá carta porque será más probable pasarse de 21 que quedarse en 21 o no superarlo.

Explicado el diseño del jugador NPC que cuenta cartas, pasaremos a describir el diseño del que no cuenta cartas. Para el diseño de la tabla de estrategias se ha tomado como base la Figura 2.5 y se le han aplicado una serie de cambios:

- Se ha eliminado la opción de Separar debido a las dificultades que esto provocaba en el código.
- Se ha eliminado las opciones de Doblar y Rendirse debido a que el ámbito de este TFG no incluían las apuestas para evitar futuros problemas por adicción al juego. Como estas opciones iban directamente relacionadas con las apuestas, se eliminan.
- Todas las opciones eliminadas se han modificado para que la estrategia a aplicar sea pedir carta.

Después, para la creación de la tabla se ha usado un script guardado en un archivo de Python que se verá más a fondo en la sección de Implementación (si se quiere pasar directamente a verlo, visitar el Apartado 3.2.1.4).

Vistas las modificaciones que se aplican, veamos el diseño de éstas:

- Se crearán tres tablas (una para cada posibilidad de ronda):
  1. El jugador tiene como mano dos cartas diferentes, deberá buscar en la tabla que llamaremos "***table***". Cuando el programa requiera buscar en esta tabla, lo hará usando: "***normal***".
  2. El jugador tiene en su mano una carta que es un AS, deberá buscar en la tabla que llamaremos "***table_with_as***". Cuando el programa requiera buscar en esta tabla, lo hará usando: "***as***".
  3. El jugador tiene en su mano las dos cartas iguales, deberá buscar en la tabla que llamaremos "***table_with_pairs***". Cuando el programa requiera buscar en esta tabla, lo hará usando: "***equals***".
- Para cada tabla será necesario crear una lista con las diferentes opciones que tenemos en las filas, y a su vez crearemos una única lista que será las diferentes posibilidades que tenemos en las columnas.

Una vez descrito el diseño de las tablas que se han creado para que el jugador NPC tome las decisiones, se da por concluido la explicación del diseño de los modelos y por tanto se dará por concluido el apartado de diseño del Backend (la parte de descripción de las vistas y el enrutamiento que se sigue, se ven en la sección de Implementación ya que se considera que el diseño fue visto lo suficiente en el Estado del Arte: Apartado 2.3.1.3 y Apartado 2.3.1.2).

3.1.4 Diseño del Frontend

El objetivo principal del Frontend es poder crear una página web agradable para la vista y sencilla de utilizar para el futuro usuario. Esto se va a lograr haciendo uso de las plantillas de Django, HTML, CSS, Javascript y HTMX. Todo esto se irá viendo en este apartado poco a poco.

3.1.4.1 Plantillas de Django y HTML

En este apartado veremos la estructura de las plantillas que se ha elegido para el proyecto así cómo el código HTML usado. Antes de comenzar con la explicación de cómo se ha diseñado todo el frontend, mostraré una imagen del concepto que se quiere aplicar para que se pueda hacer una comparación posterior con los resultados, la imagen en cuestión será la Figura 3.4.
Lo que se quiere diseñar entonces es una página principal en la que se muestren las dos opciones que ofrece el proyecto y que al elegir, se redirija a otra página para poder realizar las acciones. Sabiendo ya un boceto de lo que se quiere diseñar, procedamos a ver cómo se ha logrado.

Primeramente se ha creado una plantilla que se llamará ***base.html*** y se usará como la base, valga la redundancia, para la página web. De esta plantilla heredarán el resto, exceptuando la pantalla del juego, como se puede ver la diferenciación en la imagen (Figura 3.4). El código que hace referencia a la plantilla ***base.html*** se puede encontrar en el anexo, concretamente en Código A.1.

En la plantilla ***base.html***, nos encontramos con varias cosas, recalquemos las más importantes:

- El uso de ***{% load static %}***, esto es usado por Django para poder permitir la inclusión de imágenes que se ubiquen en la carpeta ***static*** (vista en la estructura de directorios).
- El uso de ***{% block content %}{% endblock %}***, esto es lenguaje de plantillas de Django, es una de las partes que permiten la herencia entre plantillas.
- Funciones Javascript usadas para realizar distintas comprobaciones o adiciones de comportamiento en el lado del cliente.

Explicada la plantilla ***base.html***, de las que todas las plantillas heredarán menos las plantillas dedicadas al juego, la herencia se logra mediante lo siguiente:

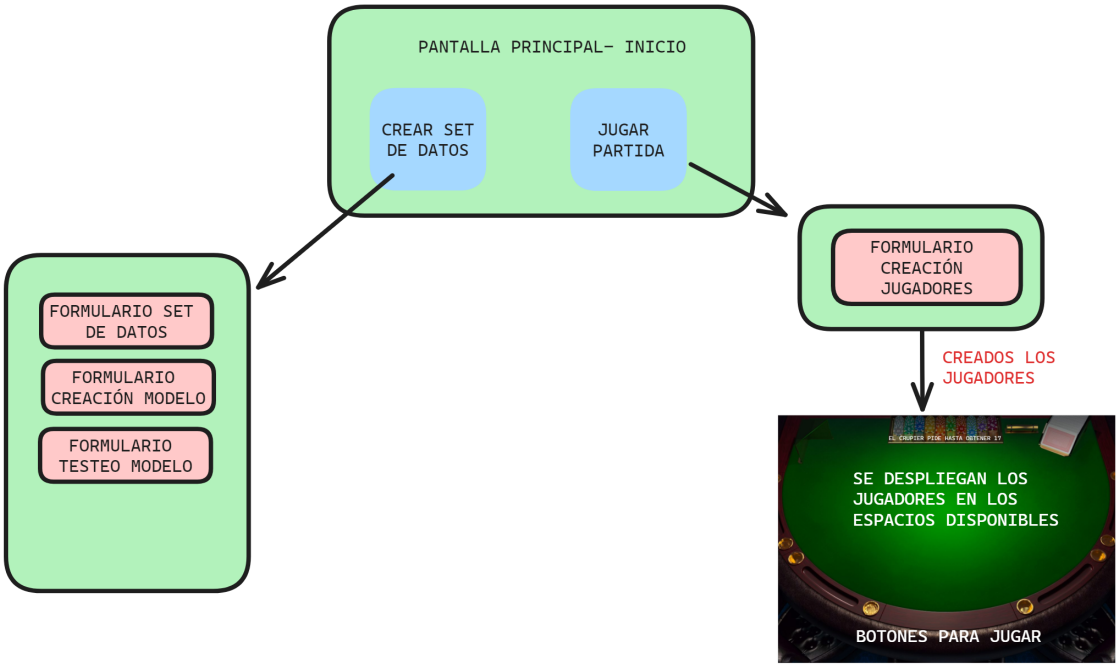


Figura 3.4 Boceto de la página web.

1. Uso de ***{% extends base.html%}***, lenguaje de plantilla de Django que indica que la plantilla donde aparezca esto está heredando de la plantilla base.
2. Indicar dentro de ***{% block content%}{% endblock%}***, todo el código de la plantilla para que se muestre siguiendo los estilos de la plantilla ***base.html*** y además pueda hacer uso de las funciones Javascript allí creadas.

Visto cómo se realiza la herencia, se explicará el diseño de las páginas mostradas en el boceto de la Figura 3.4:

- ***Pantalla Principal***, en esta plantilla se añadirán dos elementos que indiquen las opciones disponibles del proyecto.
- ***Pantalla Creación de Datos***, en esta plantilla se indicarán tres formularios distintos:
  - Formulario para crear un set de datos.
  - Formulario para crear un modelo de Inteligencia Artificial.
  - Formulario para testear un modelo de Inteligencia Artificial.
- ***Pantalla Creación Jugadores***, esta plantilla tendrá un formulario para crear los jugadores y una tabla donde se mostrarán todos los jugadores que se vayan creando.
- ***Pantalla de Juego***, esta plantilla será la única que no heredará de la plantilla ***base.html*** ya vista. Sin embargo, heredará de otra plantilla, llamada ***game_base.html*** la cual sigue exactamente el mismo diseño que la anterior mencionada pero con algunos cambios, cómo se puede apreciar en el anexo: Código A.5.

Explicado el diseño de la página web, se explicará la librería usada para poder realizar peticiones AJAX.

3.1.4.2 Librería HTMX

Esta librería ha sido usada para poder mejorar ciertos aspectos del diseño de la pantalla de juego.

HTMX está pensado para implementarlo directamente en botones o enlaces de HTML (elementos ***button*** y ***a***) y de esta manera evitar el código Javascript para realizar las peticiones AJAX, el cual puede resultar engorroso en algunos puntos del proyecto. El principal uso que se le ha dado en este proyecto ha sido la actualización dinámica de las manos de los jugadores en la partida.

Esta librería ofrece varios métodos para usar dependiendo del resultado que se quiera obtener, en este caso, los métodos utilizados han sido los siguientes:

- ***HX-POST***: esto lo que nos permite es hacer una petición tipo POST a la URL que indiquemos.
- ***HX-VALS***: esto nos da la opción de enviar un valor en la petición POST para que de esta manera podamos en la vista correspondiente comprobar la decisión del usuario.
- ***HX-TARGET***: esto indica qué parte de la página queremos que se actualice con la información nueva que generemos (en este caso, añadir una carta), al querer que se actualicen sólo las cartas del usuario, indicamos su lugar.
- ***HX-SWAP***: este parámetro sirve para indicar cómo queremos que se haga la actualización, según el valor indicado, el cambio se haría en el mismo objeto.

Por último, antes de pasar a explicar el diseño de la Inteligencia Artificial, indicar que en el Código B.1 se muestra el archivo donde se tienen creados todos los estilos que se aplican en toda la página web.

3.1.5 Diseño de la Inteligencia Artificial

En este apartado se explicará el diseño del modelo de Inteligencia Artificial, cómo se ha creado y por qué se ha hecho de esta manera. Comenzar indicando que para el proyecto ha sido usada la librería de TensorFlow (vista previamente en el Apartado 2.2.5.5).

Dentro de la librería TensorFlow, se usará la opción de Keras como herramienta para la creación del modelo. En Keras se ofrece un modelo secuencial, el cual se escogerá como base del nuestro, debido a que el blackjack es un juego cuyos sucesos ocurren por secuencias (reparto de cartas, decisión y resultados).

Entonces, teniendo como base el modelo secuencial había que pasar a diseñar el interior del modelo, es decir, añadir capas y neuronas. Pero esto no podía hacerse así como así, ha habido un estudio previo y pruebas varias hasta alcanzar un modelo que alcanzase unos requisitos que tenía como mínimos, dichos requisitos son los siguientes:

- Que las funciones de pérdida tanto para el set de entrenamiento como en el set de validación tuviesen el menor valor posible.
- Que la precisión obtenida tanto para el set de entrenamiento como en el set de validación tuviesen el mayor valor posible.

Con los requisitos ya claros, faltaba saber qué tipo de capas añadir al modelo para que los lograra. Dentro de Keras existen muchos tipos de capas: Densa (capa donde todas las neuronas están totalmente conectadas), de aplanamiento (pensado para convertir sets de datos de más de 1 dimensión a 1), de Dropout, Convolucionales, Activación, etc. Teniendo en conocimiento ya algunas de las capas disponibles, veamos el diseño decidido para los modelos:

1. ***Modelo no Contador de Cartas***, este modelo será el que reciba como set de datos de entrenamiento aquél que no tenía un historial de cartas. Estará formado por lo siguiente:
   a) Una primera capa que sirve para indicar el tamaño que tiene el set de datos.
   b) Dos capas Densas a las que se le aplica una capa de activación "***RELU***".

c) Una capa de Dropout con valor de 0,5. Esto lo que hace es indicarle al modelo que tiene que ir "encendiendo" y "apagando" el 50 % de las neuronas, para que al final del entrenamiento, la distribución de los pesos en cada neurona sea óptima.

d) Una última capa Densa, que tendrá dos neuronas y una función de activación "***softmax***", de esta manera los resultados de las predicciones del modelo son 1 (pedir carta) o 0 (quedarse).

2. ***Modelo Contador de Cartas***, este modelo, al contrario que el no contador de cartas, si recibirá en su set de entrenamientos el historial de las cartas que van saliendo. Estará formado por:

a) Una primera capa que sirve para indicar el tamaño que tiene el set de datos.

b) Una capa Densa a la que se le aplica una regularización en los pesos para evitar posibles sobreajustes en los pesos.

c) Una capa de Dropout con valor de 0,3.

d) Una capa Densa a la que se le aplica una regularización en los pesos para evitar posibles sobreajustes en los pesos.

e) Una capa de Dropout con valor de 0,5.

f) Una capa Densa a la que se le aplica una regularización en los pesos para evitar posibles sobreajustes en los pesos.

g) Una última capa Densa, que tendrá dos neuronas y una función de activación "***softmax***", de esta manera los resultados de las predicciones del modelo son 1 (pedir carta) o 0 (quedarse).

Visto el diseño, falta explicar cómo se le indica que está bien y que está mal para que aprenda el modelo. Esto se logra con los archivos de etiquetas ya que en ellos se guarda la acción correcta que debe tomar, entonces tomará una predicción y la comprobará con la etiqueta correcta, así sabe si la decisión tomada es correcta o no. En caso de que se equivoqué, ajustará los pesos de la red neuronal para que no vuelva a ocurrir, esto lo irá midiendo según la función de pérdida y la métrica de exactitud. Todo esto, ha sido explicado previamente en la sección de Aprendizaje Supervisado del Estado del Arte (para recordarlo, Apartado 2.2.1.2).

Aclarar que no se profundizará mucho más en los modelos de Inteligencia Artificial debido a que se quiere preservar la privacidad del código, se ha explicado de tal manera para que se entienda el diseño que se ha realizado sin la necesidad de tener que indicar valores de alta importancia cómo podrían ser los números de neuronas o el tamaño de las entradas de los sets de datos.

Explicado el diseño de los modelos de Inteligencia Artificial, se da por concluida la sección dedicada al diseño y por tanto, se pasará a detallar la Implementación en código realizada para lograr dicho diseño.

3.2 Implementación

En esta sección se explicará todo el código utilizado para lograr el diseño explicado anteriormente. Se verá de una manera sencilla y concisa para evitar el atascamiento y/o aburrimiento del lector entre tanta línea de código.

Se seguirá el mismo orden mostrado al principio de este capítulo, comenzando con el Backend del proyecto.

3.2.1 Backend

En este apartado se verá toda la implementación de los archivos referentes al Backend, que serán:

- El archivo de los modelos, llamado ***models.py***.
- El archivo de creación de la Tabla Q, llamado ***create_q.py***.
- El archivo de creación de la tabla de estrategias para el NPC, llamado ***table_strategies.py***.
- El archivo de funciones, llamado ***utils.py***.
- El archivo de las vistas, llamado ***views.py***.

Comenzaremos viendo el archivo de los modelos.

3.2.1.1 Archivo models.py

En este archivo se crearán los diferentes modelos que sean necesarios para el desarrollo del proyecto, los cuáles son el modelo ***Deck*** y el modelo ***Player***.

Modelo Deck

Es una clase llamada ***Deck*** que hereda de los modelos de Django (ya vimos lo que eran los modelos en el Estado del Arte), aquí es dónde se hará todo lo relacionado con la baraja de cartas, veremos a continuación las distintas funciones y atributos que tiene la clase.

Constructor

En esta función tenemos creados los atributos principales que serán necesarios para poder usar el mazo, podemos verlos más en profundidad en el Código 3.1:

Código 3.1 Constructor de la clase Deck.

```
def __init__(self):
    super(Deck, self).__init__()
    self.cards = []
    self.negated = [] # lista que nos servirá para llevar el conteo de cartas
    self.count = 0 # cuenta que sería el valor que se lleva para un conteo de
        cartas
    self.created = False # booleano que nos indica cuando se ha creado un nuevo
         mazo
    self.values = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q',
         'K'] # lista con los valores de las cartas
    self.suits = ['H', 'D', 'C', 'S'] # palos de las cartas (Hearts, Diamonds,
        Clubs, Spades)
    self.create_deck()
```

Cómo podemos ver, tenemos:

- "***cards***": lista dónde tendremos todas las cartas.
- "***negated***": lista que tendrá una lista de todas las cartas que han ido saliendo a lo largo de la partida.
- "***count***": variable entera usada para llevar la cuenta que usará el algoritmo Hi-Lo.
- "***created***": variable booleana que nos servirá para indicar cuando se ha creado un nuevo mazo.
- "***values***": lista con todos los valores que existen en el mazo de cartas.
- "***suits***: lista con todos los palos que existen en la baraja francesa.

Por último, hacemos una llamada a la función "***create_deck***" para crear el mazo, esta función se verá ahora.

Crear Deck

En esta función inicializamos los atributos de "***cards***", "***negated***" y "***count***" a los valores vacíos de cada tipo, también será necesario crear una lista para llevar el historial de los valores de las cartas (útil para el algoritmo Hi-Lo) y para acabar rellenar la lista de "***cards***". El funcionamiento de cómo se rellena el mazo lo vemos a continuación en Código 3.2:

Código 3.2 Rellenar el mazo.

```
def create_deck(self):
    self.cards = [] # lista de las cartas de la baraja
    self.negated = [] # lista que nos servirá para llevar una lista que muestra
        las cartas que han salido
    self.count = 0 # inicializamos a 0 otra vez el valor de la cuenta
    self.history_cards = [] # lista que lleva el historial de los valores de
       las cartas que han salido
    for value in self.values: # recorremos los 13 valores
        for suit in self.suits: # con los 4 palos
            for _ in range(6): # creamos 6 barajas
                self.cards.append(f'{value}-{suit}') # añadimos las cartas
```

Aquí lo que estamos haciendo es recorrer todos los valores disponibles en un mazo, después para cada valor, recorremos los cuatro palos y por último, lo recorremos un total de 6 veces, para lograr un set de 6 mazos, en vez de tener sólo 1. De esta manera estaríamos asemejándonos a la realidad.

Función que rellena una lista con el historial de cartas

En esta función lo que hacemos es recorrer la lista llamada "***negate***" (que como veremos después, se va rellenando con cada carta que se elimina de la lista "***cards***"). Para ello será necesario crear una lista de 312 ceros y asignar un valor numérico a las cartas correspondientes al AS y figuras:

Código 3.3 Asignar valores numéricos.

```
if card == "A":
    card = "1"
elif card == "J":
    card = "11"
elif card == "Q":
    card = "12"
elif card == "K":
    card = "13"
```

Una vez tenemos todos los valores numéricos de las cartas, será necesario calcular el índice que le corresponde a la carta en el total del mazo, por ejemplo: A los ASES le pertenecen los índices del 1 al 4 (o del 0 al 3 si hablamos con enumeración de Python). Para llevar esto a cabo:

Código 3.4 Calcular índice en la baraja.

```
value = int(card)
index = 4 * (value-1)
```

Cómo vemos, pasamos el valor de la carta a un entero (por ello se hacia la asignación numérica) y para hallar el índice con el que rellenaremos la lista se le resta 1 al valor (debido a que Python comienza sus cuentas con 0 en vez de 1) y multiplicamos por 4, porque cómo los palos de las cartas en el blackjack no importan, entonces se multiplica por la constante 4 que es el número de palos de la baraja. Una vez tenemos como calcular el índice de la carta, será necesario indicar en la lista que cartas han ido saliendo, para ello haremos lo mostrado en el Código 3.5:

Código 3.5 Rellenar la lista.

```
if final_list[index] == 0:
    final_list[index] = 1
elif final_list[index+1] == 0:
    final_list[index+1] = 1
elif final_list[index+2] == 0:
    final_list[index+2] = 1
elif final_list[index+3] == 0:
    final_list[index+3] = 1
```

Comprobamos si el hueco en la lista final correspondiente al índice de la carta ha salido ya o no (si ha salido tiene valor 1 y si no, valor 0), en caso de que no haya salido indicamos que ha salido cambiando su valor a 1. De esta manera vamos rellenando una lista con las posiciones de todas las cartas que han salido (esto será utilizado para el modelo de IA que cuenta cartas, usando esta función). Cabe recalcar que con esas líneas sólo estaríamos abarcando hasta el índice 52 de la baraja (51 en Python), para poder abarcar las 312 cartas con las que se estarían jugando se repetirían exactamente las mismas líneas pero las variables que se les suman cambian de la siguiente manera:

- Números del 0 al 3 para indicar todas las posiciones referentes a la primera baraja.
- Números del 52 al 55 para indicar todas las posiciones referentes a la segunda baraja, que iría desde la carta 53 (52 según Python) hasta la carta 104 (103 según Python).
- Números del 104 al 107 para hacer referencia todas las posiciones de la tercera baraja.
- Números del 156 al 159 para las posiciones de la cuarta baraja.
- Números del 208 al 211 para la quinta baraja.
- Números del 260 al 263 para la sexta baraja.

De esta manera, estaríamos abarcando las 312 cartas y sus diferentes posiciones. Por tanto, si salieran 4 ASES y posteriormente volviera a salir otro AS, se ubicaría en las posiciones de la lista referente a la segunda baraja. Indicar que la idea de esta función la he sacado de un usuario de GitHub llamado "Justin Bodnar" ([8]).

Robar Carta del Mazo

En esta función aplicamos la lógica de robar una carta aleatoriamente del mazo y cómo afecta esto a los distintos atributos de nuestra clase, veamos más a detalle la función en la Código 3.6:

Código 3.6 Robar Carta del Mazo.

```
def deal_card(self):
    self.check_deck() # comprobamos que el deck no esté vacío
    card = random.choice(self.cards) # elige un elemento al azar de la baraja
    self.cards.remove(card) # la elimina de la baraja
    self.negated.append(card) # la añade a la lista de "negated" para hacer el
        conteo
    self.history_cards.append(card) # la añade al historial de valores de
        cartas que han salido
    return card # esto es un string
```

Aquí, lo primero que hacemos es una llamada a una función que comprueba si el mazo está vacío o no de la siguiente manera:

Código 3.7 Comprobación del Mazo.

```
def check_deck(self):
    if len(self.cards) == 0:
        self.create_deck()
        self.created = True # indicamos que el mazo se ha creado
```

En caso de que el mazo estuviera vacío lo que haríamos sería volver a crearlo con la llamada a la función "***create_deck***" e indicar que se ha creado uno nuevo, actualizando el valor de "***created***" a True. Una vez que realizamos la comprobación y vemos que sigue teniendo cartas, procedemos a tomar un valor de carta al azar haciendo uso de la librería "***random***", posteriormente eliminamos dicha elección de la lista de "***cards***", la añadimos a las listas de "***negate***" y "***history_cards***" (usadas para llevar el conteo de las cartas que van saliendo) y por último devolvemos usando "***return***" dicha carta.

Para finalizar con esta clase, veremos un par de funciones encargadas de serializar y deserializar los valores que nos interesen, para poder guardarlos en la sesión.

Funciones de serialización y deserialización

Código 3.8 Función serialize.

```
def serialize(self):
    return {
        'cards': self.cards,
        'count': self.count,
        'negated': self.negated,
        'history_cards': self.history_cards,
        'created': self.created,
    }
```

Aquí lo único que estamos haciendo es convertir todos los datos en un diccionario para poder almacenarlo en la sesión que manejamos con Django, por tanto, lo que realizamos en la función encargada de deserializar es obtener el diccionario y recuperar los valores, almacenándolos como propiedades de una instancia que se almacenará en la variable deck que estemos usando.

Clase Player

Es una clase llamada "***Player***" que al igual que con la clase "***Deck***", hereda de los modelos de Django y haremos todo lo relacionado con los jugadores.

Constructor

En esta función crearemos todos los atributos que necesitemos y a su vez, los iniciaremos a los valores que recibamos. Veámoslo más a fondo en el Código 3.9:

Código 3.9 Atributos Player.

```
session_id = models.CharField(max_length=255, default='') # Identificador de
    sesión del jugador para hacerlos independientes
hand = models.JSONField(default=list, blank=True) # Almacenamos las cartas como
     una lista de strings.
hand_value = models.IntegerField(default=0, validators=[MinValueValidator(0)])
    # Valor total de la mano.
model_name = models.CharField(default="Desconocido", max_length=100) # Nombre
    del modelo.
type_player = models.CharField(default="Desconocido", max_length=50) # Tipo de
    jugador.
```

```
count_cards = models.BooleanField(default=False) # Indica si el jugador cuenta
    cartas.
blackjack = models.BooleanField(default=False) # Indica si el jugador ha
    obtenido blackjack.
wins = models.IntegerField(default=0, validators=[MinValueValidator(0)]) # Nú
    mero de victorias.
losses = models.IntegerField(default=0, validators=[MinValueValidator(0)]) # Nú
    mero de derrotas.
ties = models.IntegerField(default=0, validators=[MinValueValidator(0)]) # Nú
    mero de empates.
win_percent = models.FloatField(default=0) # Porcentaje de victorias
loss_percent = models.FloatField(default=0) # Porcentaje de derrotas
ties_percent = models.FloatField(default=0) # Porcentaje de empates
finished = models.BooleanField(default=False) # Indica si el jugador ha
    terminado su turno
created_at = models.DateTimeField(default=timezone.now) # Campo que guardará la
     fecha de creación del jugador
```

Detallar la importancia del atributo ***session_id***, será el encargado de poder diferenciar los jugadores creados en distintas sesiones, lo que nos brinda la capacidad de jugar diferentes usuarios a la vez. Se verá posteriormente qué valor y cómo se le da.

Procedemos viendo la función usada para saber si el jugador ha logrado un blackjack y de esa manera actualizar el valor de su atributo "***blackjack***" a "***True***".

Comprobar Blackjack

En esta función se comprueba si el jugador logra blackjack o no, para ello será necesario escribir las siguientes líneas de código:

Código 3.10 Comprobación Blackjack Jugador.

```
def check_blackjack(self):
    if self.hand_value == 21 and len(self.hand) == 2:
        self.blackjack = True
        return True
    else:
        self.blackjack = False
        return False
```

Por último, veremos la función llamada "***calculate_percentage***".

Calcular Porcentajes

En esta función lo que se hace es calcular el porcentaje de victorias y derrotas que tiene el jugador con el transcurso de cada partida. Para ello nos ayudaremos de los atributos de "***wins***" y "***losses***" creados previamente. Se puede ver la función en el Código 3.11:

Código 3.11 Calcular Porcentajes Victoria/Derrota/Empate.

```
def calculate_percentage(self):
    games = self.wins + self.losses + self.ties # calculamos el número total de
         partidas que ha jugado
    win_percent = float(self.wins/games)*100
    loss_percent = float(self.losses/games)*100
    ties_percent = float(self.ties/games)*100
    self.win_percent = round(win_percent, 2)
    self.loss_percent = round(loss_percent, 2)
    self.ties_percent = round(ties_percent, 2)
```

Además en esta clase también tenemos funciones para serializar y deserializar a los jugadores pero siguen la misma idea que para la clase Deck sólo que cambiando los valores que se quieran almacenar.

3.2.1.2 Archivo "create_q.py"

En este archivo tenemos lo necesario para poder crear diferentes tablas Q las cuáles han aprendido usando el algoritmo ***Q-Learning*** y el método ***Montecarlo***.

Para poder ejecutar este archivo y que funcione sin problemas será necesario incluir:

- La clase Deck del archivo "***models.py***" para poder usar un mazo de cartas.
- Las librerías de "***numpy***", "***random***", "***collections***" y "***pickle***". Veamos un poco el por qué de el uso de cada librería:
  - "***numpy***": será necesaria usarla porque para el manejo de los distintos "***arrays***" para la tabla Q piden el formato "***numpy***" para asegurar que todos los elementos de ese "***array***" sean del mismo tipo.
  - "***random***": para generar un valor aleatorio cuando sea necesario.
  - "***collections***": de esta librería sólo será necesario importar "***defaultdict***", el cuál será usado para crear la Q como tal. El hecho de que se cree la Q como un "***defaultdict***" en vez de un simple "***dict***" radica en evitar problemas con faltas de valores asociadas a las "***keys***".
  - "***pickle***": se usa para guardar la Tabla Q en un archivo ya que nos permite serializar los objetos de Python, cómo se explicó en el: Apartado 3.1.3.1.

Una vez visto las importaciones necesarias, pasemos a ver las funciones que permiten obtener como resultado las tablas Q con un número de partidas al gusto. Comenzaremos viendo la función encargada de aplicar el método Montecarlo.

Generar la siguiente acción

En esta función, como hemos dicho, se aplica el método Montecarlo para averiguar la probabilidad con la que se tomará un tipo de decisión, si al azar o según lo aprendido. Para eso, será necesario tener una probabilidad de pedir carta, una probabilidad de quedarse y un parámetro, llamado "ε" que mida la aleatoriedad con la que se toma la decisión. Toda esta parte han sido explicadas previamente en los: Apartado 3.1.3.1, Apartado 2.2.2.7 y Apartado 2.2.2.

Sabiendo esto, veamos el código usado para implementar la manera en la que se genera la acción (Código 3.12):

Código 3.12 Generar Siguiente Acción.

```
def gen_next_action(state, e, Q):
    hit_prob = Q[state][0] # la probabilidad de pedir carta la calculamos
        usando la función Q, indicando que la acción es 0
    stay_prob = Q[state][1] # la probabilidad de quedarse la calculamos usando
        la función Q, indicando que la acción es 1

    if np.random.uniform(0, 1) < e: # si un número aleatorio entre 0 o 1 es
        menor que la epsilon
        return random.choice([0, 1]) # elegimos una acción al azar
    else:
        # elegimos la mejor acción que haya aprendido la Q, teniendo en cuenta
            que => 0 === Pedir carta ; 1 === Quedarse
        return 0 if hit_prob > stay_prob else 1
```

Lo que nos encontramos es que para un mismo estado, en la Tabla Q se han guardado dos probabilidades, la de pedir carta y la de quedarse. Por tanto, para poder recuperar dichos valores será necesario indicar para ese estado, qué acción implica pedir carta, que será el 0, y que acción implica quedarse, que será el 1. Lo siguiente será comprobar si un valor aleatorio es menor que nuestra variable ε ya que en dicho caso la elección será tomada al azar y en caso contrario, la acción será tomada según la probabilidad más alta que tenga la Tabla Q guardada para ese estado.

A continuación, veremos como se crean los estados que forman parte de la tabla Q.

Crear los valores del estado

En esta función se creará el estado con el que se asocia la acción, los cuales forman parte de la tabla Q.

El estado estará formado por:

- Valor de la mano del jugador, en este caso, el jugador "Q".
- Valor de la carta visible del crupier.
- Si el AS que tiene en la mano puede ser usado como 11 o no.

Para comprobar si el AS puede tomar un valor de 11 o no, haremos uso de la función has_usable_ace que se verá luego (si se quiere saber su funcionamiento ahora, visitar el Apartado 3.2.1.6).

Actualizar valores tabla Q

Para poder actualizar los valores en la tabla Q será necesario recoger la información referente al estado y acción que tomo con ese estado y la recompensa que obtuvo al hacerlo. Una vez recopilados esos valores, los aplicaremos a la Ecuación 2.5, dónde si la llevamos al formato de código, tendremos el Código 3.13:

Código 3.13 Actualizar valores tabla Q.

```
def setQ(Q, currentEpisode, gamma, alpha):
    for t in range(len(currentEpisode)):
        rewards = currentEpisode[t:,2]
        # creamos una lista para la tasa de descuento con los valores de gamma
            incrementándose
        discountRate = [gamma**i for i in range(1,len(rewards)+1)]
        # descontamos las recompensas desde la partida t+1 en adelante
        updatedReward = rewards*discountRate
        # sumamos todas las recompensas descontadas para igualar el retorno al
            paso de t
        Gt = np.sum(updatedReward)
        # Calculamos el valor actual del par estado, acción en la tabla Q
        Q[currentEpisode[t][0]][currentEpisode[t][1]] += alpha *(Gt - Q[
            currentEpisode[t][0]][currentEpisode[t][1]])
    return Q
```

Veamos un poco lo que se hace en el código:

1. Recuperamos todas las recompensas que ha ido recibiendo, las cuales se ubican en la última posición de la lista "***currentEpisode***". Por tanto, con el uso de "***t:***", se indica que coja todas las recompensas desde el índice "t" en adelante.
2. Creamos la lista de tasas de descuento que vamos a tener en cada recompensa, aplicando el valor de "***gamma***".
3. Aplicamos la tasa de descuentos a las recompensas para posteriormente hallar el valor total sumándolas.

4. Por último, aplicamos Ecuación 2.5 dónde Gt haría referencia a la parte de $R+(\gamma \cdot maxQ(s',a')$ ya que es el total de todas las recompensas a las que se ha aplicado la tasa de descuento.

Continuemos con la función en la que se ubica toda la lógica necesaria para simular las partidas del blackjack y crear las tablas Q deseadas.

Simular las partidas y creación de las tablas Q

En esta función tenemos creadas las diferentes variables que serán necesarias para simular las partidas. Comencemos viendo cuáles son los valores que se han dado a las variables, mencionadas en el Apartado 3.1.3.1:

- ***Variable*** ε, tendrá valor de 0,02. De esta manera indicamos que queremos una aleatoriedad baja, es decir, priorice las decisiones que ha aprendido la Tabla Q.
- ***Variable*** α, tendrá valor de 0,01. De esta manera indicamos que queremos que la Tabla Q vaya aprendiendo de una manera lenta pero segura, este parámetro es clave para decidir el número de partidas con el que se creará la Tabla Q.
- ***Variable*** γ, tendrá valor de 1. De esta manera indicamos que queremos que el jugador Q-Learning tome con igual importancia las recompensas futuras y las recompensas inmediatas.

Posteriormente creamos la Q como un diccionario que tendrá estado-acción como los pares clave-valor. Además será necesario crear una lista para guardar lo ocurrido en la partida que llamaremos "***currentEpisode***", lo que guardaremos en esta lista será el estado actual, la acción tomada y la recompensa obtenida. Por último, crearemos variables para llevar unas estadísticas de las partidas para poder ver los resultados que ha obtenido la Q de victorias y derrotas.

Una vez creadas todas las variables, comenzamos a implementar la lógica del juego para la creación de las tablas Q. Iniciaremos recorriendo mediante un bucle *for* el número de partidas a simular. Después, inicializaremos las manos del jugador y crupier para poder asignarles cartas, esto se hará mediante lo mostrado en Código 3.14:

Código 3.14 Inicialización y asignación de las manos.

```
ais_hand =[] # mano de la IA
dealers_hand = [] # mano del dealer

for i in range(4): # rellenamos las manos de los jugadores
    if i in [0, 2]:
        ais_hand.append(deck.deal_card())
    else:
        dealers_hand.append(deck.deal_card())
```

Con las cartas ya asignadas a cada jugador, necesitamos comprobar si ha habido algún blackjack, para ello sólo será necesario comprobar si alguna de las manos ha logrado ya 21. Además de las comprobaciones será necesario asignar los valores a nuestras variables que afectan a la Q:

Código 3.15 Modificación estado, acción y recompensa.

```
current_state = create_state_values(ais_hand, dealers_hand) # creamos el valor
    del estado actual
action = 0 # la acción sería quedarse
reward = 1 # si logra blackjack, la recompensa es positiva
wins += 1 # si logra blackjack, incrementamos el número de victorias
```

En caso de que no se logre blackjack, se entra en la "segunda fase" del juego, donde se decide si pedir carta o no, esto ha sido implementado según el Código 3.16:

Código 3.16 Lógica de pedir/quedarse.

```
while True:
    current_state = create_state_values(ais_hand, dealers_hand) # creamos el
        valor del estado actual
    action = gen_next_action(current_state, e, Q) # generamos la acción que
        tomara la IA
    num_hits = 0 # variables para llevar el conteo de veces que pide carta
    if action == 1: # si la acción elegida es pedir carta
        num_hits += 1 # incrementamos en 1 las veces que pide carta
        ais_hand.append(deck.deal_card())
        # comprobamos que no se pase de 21 o si ha pedido 3 cartas y no ha
            perdido
        if hand_value(ais_hand) > 21 or num_hits == 3 and hand_value(ais_hand)
            <=21:
            break
    else: # si la acción elegida es quedarse
        # pasamos al turno del crupier
        while hand_value(dealers_hand) < 17: # mientras que tenga menos de 17,
            pide carta
            dealers_hand.append(deck.deal_card())
        break
```

Como se comprueba, se ejecutará siempre el hecho de tomar la decisión de pedir carta o no pero se llevará una cuenta de las veces que se pide para poder aplicar la "***Five Card Charlie Rule***", dicha regla ha sido explicada ya en el Estado del Arte (si se quiere recordar, visitar el Apartado 2.1.2).

Una vez salimos del bucle de decidir, habrá que comprobar quién es el ganador, para ello se hacen las siguientes comprobaciones:

- Si el jugador se pasa de 21, gana el crupier (recompensa negativa).
- Si el crupier se pasa de 21 y el jugador no, gana el jugador (recompensa positiva).
- Si ninguno se pasa de 21, el que tenga mayor valor gana.

Una vez terminada cada partida hará falta actualizar los valores de la tabla Q con los resultados de la partida, para ello necesitaremos las siguientes líneas de código:

Código 3.17 Actualización Q.

```
currentEpisode.append((current_state, action, reward))
# actualizamos los movimientos en la tabla Q y vaciamos el set de movimientos
currentEpisode = np.array(currentEpisode, dtype="object") # transformamos el
    array a un np.array para poder trabajar con él en la Q
Q = setQ(Q, currentEpisode, gamma, alpha)
currentEpisode = [] # la lista que contiene los movimiento de la partida
```

Por último, una vez terminada la simulación de todas las partidas, guardaremos la tabla Q en un archivo con el que recuperar todo el aprendizaje hecho y usarlo en nuestro código principal. Para ello:

Código 3.18 Guardar tabla Q.

```
# almacenamos la Q en un archivo
with open("blackjack/modelos/Tabla_Q/Q_table_"+str(games)+".pickle", "wb") as
    file:
    pickle.dump(Q, file)
```

Aclarar que las partidas no las juega ningún jugador real, sino que las juega un jugador tipo Q-Learning desde 0. Con esto claro, se crean 4 Tablas Q en las que varían únicamente el número de partidas (decidido tras varias pruebas y teniendo en cuenta el factor que supone la variable α). La aplicación de estos números de partidas ha sido de la siguiente forma (Código 3.19):

Código 3.19 Inicialización de la ejecución del archivo.

```
if __name__ == "__main__":
    deck = Deck()

    games = [250000, 500000, 750000, 1000000] # número de partidas que se jugar
        á
    for game in games:
        wins, losses = play(deck, game)

        print(f"Partidas Jugadas: {game}")
        print(f"Victorias Totales: {wins} => Porcentaje: {round((wins/int(game)
            )*100, 2)}%")
        print(f"Derrotas Totales: {losses} => Porcentaje: {round((losses/int(
            game))*100, 2)}%\n")
```

Simulamos desde 250000 partidas a 1000000 para que pueda aprender y mejorar, una vez ejecutado el archivo se puede percibir que los resultados son bastante parejos (visible en la Figura 3.5) sin notar una gran diferencia en el porcentaje de victorias lo cuál habla de manera positiva del modelo ya que es capaz de mantener su porcentaje a pesar de simular una gran cantidad de partidas.

```
Partidas Jugadas: 250000
Victorias Totales: 101539 ⇒ Porcentaje: 40.62%
Derrotas Totales: 132962 ⇒ Porcentaje: 53.18%

Partidas Jugadas: 500000
Victorias Totales: 203335 ⇒ Porcentaje: 40.67%
Derrotas Totales: 265554 ⇒ Porcentaje: 53.11%

Partidas Jugadas: 750000
Victorias Totales: 305811 ⇒ Porcentaje: 40.77%
Derrotas Totales: 397892 ⇒ Porcentaje: 53.05%

Partidas Jugadas: 1000000
Victorias Totales: 407668 ⇒ Porcentaje: 40.77%
Derrotas Totales: 530969 ⇒ Porcentaje: 53.1%
```

Figura 3.5 Ejecución archivo create_q.py.

3.2.1.3 Archivo process_data.py

En este archivo lo que hacemos es procesar los datos que generamos para el entrenamiento del modelo IA ya que queremos que sean una única lista de tipo entero, los datos han sido vistos ya en el Apartado 3.1.3.1.

Primeramente abrimos los archivos donde tenemos guardados los datos y las etiquetas creadas con:

Código 3.20 Apertura y lectura de los archivos.

```
data = open( "datasets/" + str(file_name) + ".data").readlines()
tags = open( "datasets/" + str(file_name) + ".tags").readlines()
```

Una vez hemos recopilado en las variables los datos y etiquetas comenzamos el procesamiento. Iniciamos con los datos para ello haremos uso del Código 3.21:

Código 3.21 Procesamiento de los datos.

```
for line in data:
    # Eliminamos el salto de línea y los corchetes de la línea
    line = line.strip().strip('[]')
    # Dividimos la línea en elementos individuales
    elements = line.split(', ')

    # Reemplazamos las cartas específicas y convierte los números a enteros
    processed_line = []
    for element in elements:
        element = element.strip("'") # Eliminamos las comillas simples si están
            presentes
        processed_line.append(int(element))

    data_clean.append(processed_line)
```

Comenzamos por el archivo que guarda todos los datos recopilados para el entrenamiento del modelo de Inteligencia Artificial, cómo se quiere tener una única lista que almacene todos los datos, será necesario eliminar los saltos de línea y los corchetes que encierran las posibles listas que existan en el fichero. Después dividiremos cada línea por comas e iremos transformando cada elemento a un entero y añadiéndolos a una lista que hará referencia a la línea ya transformada. Por último, esta línea la añadimos a una lista dónde almacenaremos todos los datos ya procesados.

Continuamos con las etiquetas ahora, descrito en el Código 3.22:

Código 3.22 Procesamiento de las etiquetas.

```
for tag in tags:
    tag = tag[:tag.index('\n')] # tomamos sólo las letras de cada línea
    if tag == "h":
        tags_clean = tags_clean + [ 1.0 ] # si la etiqueta es "h" => Pedir
    else:
        tags_clean = tags_clean + [ 0.0 ] # si la etiqueta es "s" => Quedarse
```

Esta vez es mucho más simple, comprobamos si la etiqueta es una "h" o una "s" y dependiendo de su valor añadimos una lista con el valor 1 o 0 respectivamente.

Lo último que hacemos en este archivo es devolver el tamaño del set de datos y la lista de los datos y etiquetas ya procesadas para su uso en el código principal.

3.2.1.4 Creación de las tablas de estrategia

En este archivo lo que se hace es llevar a código la tabla vista en la Figura 2.5. Cómo se ve en la figura, las filas tienen diferentes nombres según en el caso en el que nos encontremos pero las columnas tienen siempre el mismo nombre ya que hace referencia a la carta visible del crupier.

La forma de rellenar las tablas con los valores, la encontramos en el Código 3.23. Lo único que varía son los valores que dependerán del tipo de tabla que queramos rellenar, vistas en el Apartado 3.1.3.1.

Código 3.23 Rellenado de la tabla.

```
for row_name in name_rows:
    row = [row_name]
    for col_name in name_cols:
        # Verificar la condición para determinar el valor de la celda
        if row_name in [5, 6, 7, 8, 9, 10, 11]:
            value = 'h' # P == Pedir carta.
        elif row_name == 12:
            if (col_name in [4, 5, 6]):
                value = 's' # Q == No pedir carta.
            else:
                value = 'h'
        elif row_name in [13, 14, 15, 16]:
            if (col_name in [2, 3, 4, 5, 6]):
                value = 's'
            else:
                value = 'h'
        else:
            value = 's'
        row.append(value)
    table.append(row)
```

3.2.1.5 Archivo urls.py

Indicar que toda la lógica que posee el sistema de enrutamiento que ofrece Django ha sido visto ya en el Estado del Arte, concretamente en el: Apartado 2.3.1.2.

Este archivo existe dos veces en el proyecto, una vez en la carpeta principal del proyecto, llamada "***project***" y otro en la carpeta "***blackjack***", veremos primero el archivo ubicado en la carpeta "***project***" ya que es más simple.

Archivo urls.py carpeta project

Aquí indicaremos dos rutas, la del portal de administrador e incluiremos nuestras propias urls. Echemos un vistazo al archivo en el Código 3.24:

Código 3.24 Archivo urls.py "padre".

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('', include('blackjack.urls')),
    path('admin/', admin.site.urls),
]
```

Cómo se ve, incluimos nuestro archivo con el uso de "***path***" e "***include***", indicando cuál es el archivo urls que queremos cargar, en este caso es el de la aplicación llamada "***blackjack***".

Archivo urls.py carpeta blackjack

En este archivo es dónde ya están todas las rutas que tienen uso en nuestra aplicación, se sigue un estándar que se basa en indicar la URL que se muestra en la página web, la función que se

encargará de recibir la información de las peticiones a esa página y un nombre que podamos usar en el código para hacer referencia a esa URL:

Código 3.25 Archivo urls.py de la carpeta blackjack.

```
app_name = "blackjack"
urlpatterns = [
    path("", views.index, name="index"),
    # url que te lleva a la creación de los sets de datos
    path("datasets/", views.dataset, name="dataset"),
    # url que indica que ha sido exitosa la creación del modelo de Inteligencia
        Artificial
    path("models/", views.model, name="model"),
    # url que muestra el testado de los modelos IA
    path("models/tests", views.test_model, name="test_model"),
    # url que muestra el formulario para crear los jugadores
    path("players/", views.get_info_players, name="create_players"),
    # url que muestra la tabla con todos los jugadores creados para editarlos o
        eliminarlos
    path("players/edit/", views.edit_players, name="edit_players"),
    # url que muestra la partida
    path("game/", views.game, name="game"),
    # url a la que hace la solicitud AJAX
    path("game/select_action/<int:player_id>/", views.select_action, name="
        select_action"),
]
```

Cómo se muestra, también se pueden indicar parámetros que irán en la url con los que trabajaremos. La función se indica haciendo uso de "views.funcion" ya que dichas funciones existen en un archivo "***views.py***" que se verá a profundidad luego.

Continuando con la explicación de la parte de backend del proyecto, procederemos a ver un archivo llamado "***utils.py***".

3.2.1.6 Archivo utils.py

En este archivo tendremos las declaraciones de la gran mayoría de funciones que se usan en más de un archivo del proyecto o son de un uso general, cómo podría ser calcular el valor de la mano.

Comencemos viendo cada función importante que se encuentra en este archivo:

Función hand_value

En esta función calcularemos el valor de la mano, para ello nos ayudaremos del siguiente código (Código 3.26):

Código 3.26 Función hand_value.

```
def hand_value(hand):
    hand_value = 0
    has_ace = False
    for card in hand:
        value = card.split('-')[0]
        if value == 'A': # si la carta es un As
            has_ace = True # indico que tengo un As
            card_value = 1 # le doy como valor 1
        elif value in ['J', 'Q', 'K']: # si la carta es 10 o figura
            card_value = 10 # su valor es 10
        else:
```

```
            card_value = int(value) # si no es un As o una figura, su valor es
                el número en sí de la carta
        hand_value += card_value # sumo el valor de la carta al valor de la
            mano

    # En caso de que la mano tenga una carta que sea un As y el valor de su
        mano sea menor o igual que 11, entonces el As pasará a valer 11.
    if has_ace and hand_value <= 11:
        hand_value += 10

    return int(hand_value)
```

Por cada carta que tenga la mano, dividiremos la carta por el guión ya que su formato es "valor-palo" entonces nos interesa saber su valor y una vez obtenido su valor aplicaremos la lógica de valoración del blackjack (teniendo en cuenta si tenemos un AS para darle valor de 11 o 1).

Función has_usable_ace

En esta función nos centramos en comprobar si en la mano, tenemos un AS y si puede ser tomado con valor de 11 o no. Su uso principal radica en los sets de datos que se crean para los modelos de IA y está conformada por el Código 3.27:

Código 3.27 Función has_usable_ace.

```
def has_usable_ace(hand):
    value_hand, ace, ace_usable = 0, False, 0 # iniciamos la variable del valor
         de la mano, si tenemos un as en la mano y si es usable como 11
    for card in hand: # recorro la mano carta por carta
        value = card.split('-')[0]
        value_hand += min(10, int(value) if value not in ['J', 'Q', 'K', 'A']
            else 11) # calculo el valor mínimo que puede tomar la mano
        ace |= (value == 'A') # si tengo un AS => True ; si no tengo un AS =>
            False

    if ace and (value_hand + 1) <= 21: # si tengo un AS y el valor de la mano
        es menor que 21
        ace_usable = 1 # indico que puedo tomar el AS con valor = 11
    else:
        ace_usable = 0 # en caso contrario, le indico que no es posible
    return int(ace_usable)
```

En esta función, recorremos la mano carta por carta, obtenemos su valor y vamos obteniendo el valor de la mano tomando el mínimo entre 10 (el valor de las figuras) y el valor de la carta (a no ser que sea una figura o AS, que tomaría 11). Una vez hemos calculado el valor de su mano, es necesario saber si tenemos un AS en la mano o no, para ello hacemos un OR comprobando si el valor de la carta es un AS. Por último, si podemos tomarlo como 11 o no, es decir, si sumando 1 al valor actual de la mano es menor de 21, en cuyo caso devolvemos un 1.

Función find_action_in_table

En esta función haremos la lógica para encontrar en la tabla la acción que debe tomar el jugador NPC dependiendo de su mano y la carta visible del crupier. Para ello, será necesario averiguar en qué fila y columna buscar, esto lo lograremos mediante el Código 3.28:

Código 3.28 Función find_action_in_table.

```
def find_action_in_table(hand, card, type):
    if type == "equals": # si la búsqueda es en la tabla de cartas iguales
        row_index = name_rows_pair.index(hand[0].split('-')[0]) # buscamos el
            valor del índice de la fila según la mano del NPC
        col_index = name_cols.index(card.split('-')[0]) # buscamos el valor del
             índice de la columna según la carta visible del Crupier
        table_value = table_with_pairs[row_index+1][col_index+1] # obtenemos el
             valor de la tabla según el índice de la fila y columna
    elif type == "as": # si la búsqueda es en la tabla con un AS en la mano
        if hand[0].split('-')[0] == "A":
            row_index = name_rows_as.index(hand[1].split('-')[0]) # buscamos el
                valor del índice de la fila según la mano del NPC
        else:
            row_index = name_rows_as.index(hand[0].split('-')[0]) # buscamos el
                valor del índice de la fila según la mano del NPC
        col_index = name_cols.index(card.split('-')[0]) # buscamos el valor del
             índice de la columna según la carta visible del Crupier
        table_value = table_with_as[row_index+1][col_index+1] # obtenemos el
            valor de la tabla según el índice de la fila y columna
    else: # la búsqueda será en la tabla normal
        row_index = name_rows.index(hand_value(hand)) # buscamos el valor del í
            ndice de la fila según la mano del NPC
        col_index = name_cols.index(card.split('-')[0]) # buscamos el valor del
             índice de la columna según la carta visible del Crupier
        table_value = table[row_index+1][col_index+1] # obtenemos el valor de
            la tabla según el índice de la fila y columna

    return table_value # devolvemos la acción que debe de tomar el crupier
```

Como podemos observar, habrá que diferenciar el tipo de tabla según una cadena de texto que recibamos por parámetros. Después para poder obtener el valor que se encuentra en la fila y columna seleccionada será necesario buscar en el índice+1 debido a dos cosas:

1. En la tabla creada, la esquina superior izquierda no tiene ningún valor (que hace referencia al índice [0][0] de una tabla) y además, hay que tomar en cuenta los valores de las filas y columnas que son usados para indicar los nombres que ayudan a la diferenciación de los casos.
2. La enumeración que sigue Python es comenzando en el 0.

Por tanto, si yo quiero saber el valor de la acción que debería de tomar en la tabla para el caso de la primera fila y primera columna, debería de buscar en el valor de la tabla [2][2], en vez del que se podría suponer como [1][1].

Funciones para obtener los archivos

Aquí veremos tres funciones creadas de una manera similar cuyo objetivo es obtener una lista con valores únicos de los distintos archivos que se vayan creando tanto de los sets de datos como de los modelos de IA y los archivos de la Tabla Q ya creados y entrenados. Veamos una de las funciones para explicarla y hacernos una idea de cómo se han hecho el resto (Código 3.29):

Código 3.29 Obtención archivos del sistema.

```
def get_model_files():
    model_files = []
    unique_files = set() # creamos un set para guardar todos los archivos sin
        tener valores repetidos
    folder_path = os.path.join(os.path.dirname(__file__), 'modelos/red_neuronal
        /') # creamos la ruta dónde estás los archivos guardados
    files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(
        folder_path, f))] # recuperamos todos los archivos en un formato lista
    for file in files: # recorremos todos los archivos creados mediante un for
        file_name, _ = os.path.splitext(file) # nos quedamos solamente con el
            nombre del archivo
        file_name = file_name.split('_count')[0] # le quitamos la parte de
            count ya que nos da igual
        unique_files.add(file_name) # añadimos el nombre del archivo a nuestro
            set
    model_files = [(file, file) for file in unique_files]
    return sorted(model_files) # devolvemos el set de archivos ordenados
```

Iremos paso a paso explicándola:

- Crearemos la lista que devolveremos con los datos. Además, también crearemos un set que será el que nos servirá de "lista auxiliar" dónde tendremos todos los valores únicos.
- Procederemos indicando la ruta dónde se ubican los archivos que queremos obtener.
- Recuperamos todos los archivos que se encuentran en esa ruta y los guardamos en una lista.
- Posteriormente, recorremos esa lista archivo a archivo para quedarnos sólo con su nombre. En el caso de los sets de datos y de los modelos de IA, nos quedamos sólo con el nombre que indicó el usuario sin especificaciones de si cuenta cartas o no (ya que eso se indicará en el formulario posterior de creación de jugadores).
- A continuación, guardamos esos archivos en el set para por último, rellenar nuestra lista de archivos con una tupla que sea el nombre del archivo (esto está hecho así de cara a mostrarlo en los formularios de creación de jugadores), que devolveremos de forma ordenada para ser mostrada en la página web.

Cómo parte final del backend, veremos el archivo "***views.py***".

3.2.1.7 Archivo views.py

La explicación de este archivo la dividiré en tres partes:

1. Las diferentes importaciones que se hacen, tanto de archivos como de librerías.
2. Las vistas/funciones que son usadas para la página web, es decir, las funciones que aparecían en el Apartado 3.2.1.5.
3. Las funciones que son llamadas por las vistas para llevar a cabo ciertas creaciones, ediciones, etc. Las comúnmente reconocidas como funciones ***CRUD***.

Por tanto, comencemos viendo qué importaciones se hacen en este archivo y la explicación de porqué (al menos en las más importantes).

Importaciones

En esta parte también podríamos hacer una subdivisión en tres partes:

1. Importaciones del framework Django.

2. Importaciones de librerías de Python.
3. Importaciones de archivos/funciones del propio directorio.

Importaciones de Django

Aquí nos encontramos con las siguientes:

Código 3.30 Importaciones de Django.

```
from django.shortcuts import render, redirect
from django.views.decorators.csrf import csrf_exempt
from django.http import JsonResponse
from django.template.loader import render_to_string
from django.utils import timezone
```

En la primera línea lo que nos encontramos son atajos de Django para bien cargar una plantilla HTML en una URL (***render***) o bien redirigir el flujo hacia otra página web (***redirect***).

Continuando, tendremos un decorador de Django. Los decoradores son etiquetas que se le aplican a las vistas para aplicar o evitar ciertas reglas, por ejemplo existe el decorador "login_required" que hace que esa vista sólo pueda ser usada si ha habido un inicio de sesión en la página web. En nuestro caso, el decorador lo que hace es eximir a la vista que recibe una petición tipo POST de qué dicha petición lleve el crsf_token de Django, esto se verá más en profundidad cuando sea usado tanto en la vista como en la plantilla.

Importaciones librerías de Python

Aquí veremos las diferentes librerías que son usadas en el proyecto:

Código 3.31 Importaciones de Python.

```
import uuid
import random
import pickle
import numpy as np
import tensorflow as tf
from collections import defaultdict
from datetime import timedelta
```

Comencemos explicando las menos conocidas, cómo son:

- ***UUID***: esta librería es usada para generar un identificador único y aleatorio para poder diferenciar a los jugadores que se creen en las diferentes sesiones de los diferentes navegadores.
- ***PICKLE***: esta librería se usa para poder serializar las tablas Q y guardarlas en archivos que usarán los jugadores Q-Learning.

Por último, veremos las diferentes importaciones de los archivos.

Importaciones archivos de programa

En este apartado, veremos simplemente por encima que archivos son y por qué:

Código 3.32 Importaciones de archivos.

```
from .models import Deck, Player # modelos del jugador y del deck
from .forms import DatasetForm, ModelForm, TestModelForm, PlayerForm #
    formularios usados para el frontend
from .process_data import * # archivo para procesar los datos generados en el
    set
```

```
from .utils import * # archivo dónde se guardan funciones útiles para todo el
    programa
from .create_np_zeros_dict import create_np_zeros_dict
```

Importamos los modelos, para poder crear diferentes objetos que hereden de éstos, los formularios para poder recibir la información de la página web y crear los objetos. Todas las funciones del archivo "***utils.py***", el archivo usado para procesar los datos del set de datos creado para el entrenamiento de los modelos de IA y un archivo que es el encargado de crear la Q en la que cargaremos el archivo decidido por el jugador.

Vistas todas las importaciones, pasaremos a ver todas las vistas de Django que han sido necesarias para el proyecto.

Vistas

La explicación de las vistas de Django ha sido realizada ya en el Estado del Arte, si se quiere volver a recordar, visitar el Apartado 2.3.1.3.

Vista inicial => index

Esta es la vista inicial, la encargada de cargar la página principal del proyecto. Aquí se hará la comprobación de si el ciclo de vida de los jugadores ha llegado a su final o no. Además será la encargada de renderizar la plantilla inicial, ***index.html*** (Código A.2), cómo podemos ver en el siguiente código (Código 3.33):

Código 3.33 Vista index.

```
def index(request):
    threshold_date = timezone.now() - timedelta(days=7) # creamos una fecha de
        umbral igual a 7 días atrás
    Player.objects.filter(created_at__lt=threshold_date).delete() # borramos
        todos los jugadores cuya fecha de creación sea anterior
    return render(request, "blackjack/index.html")
```

Vista dataset

Esta vista será la encargada de recibir la información de la url "datasets/" que vimos y para ello tendremos el Código 3.34:

Código 3.34 Vista dataset.

```
def dataset(request):
    form = DatasetForm(request.POST) # creamos el formulario del set de datos
    data_files = get_dataset_files() # obtenemos el nombre de los archivos de
        los sets de datos ya creados
    model_files = get_model_files()
    context = { # creamos un contexto de archivos que queremos que reciba la
        plantilla
        'form': form,
        'data_files': data_files,
        'model_files': model_files,
    }
    if request.method == "POST": # comprobamos que la request que recibimos sea
         del tipo POST
        if form.is_valid(): # comprobamos si es válido el formulario y si lo es
             recibimos los datos de los campos del formulario
            games = form.cleaned_data["games"] # campo del número de partidas
            file_name = form.cleaned_data["file_name"] # campo del nombre del
                archivo
```

```
            file_name_count = file_name + "_count"
            create_dataset(int(games), file_name, False) # creamos un set de
                datos para el modelo no contador de cartas
            create_dataset(int(games), file_name_count, True) # creamos un set
                de datos para el modelo contador de cartas
            context['dataset_success'] = True # indicamos que ha habido éxito en
                 la creación del set de datos
            return render(request, 'blackjack/dataset.html', context)
    return render(request, 'blackjack/dataset.html', context)
```

Recibirá como parámetro un objeto con los datos de la petición HTTP, para poder obtener de ahí todos los valores necesarios. Comenzaremos inicializando un formulario del tipo Dataset para recoger todos los valores, después recuperaremos todos los archivos de datos con el uso de la función "***get_dataset_files***" que ya hemos mencionado y haremos lo mismo con los modelos. Además, toda esa información será pasada por contexto. Una vez creadas todas las variables necesarias, comprobamos si el objeto HTTP recibido es de tipo POST y en ese caso, si el formulario que recibimos y cuyos datos se rellenan con los datos obtenidos del objeto HTTP, es válido. Viendo que el formulario es válido, sólo nos queda obtener los valores de cada variable, llamar a la función encargada de crear el dataset y pasar por contexto que la creación ha sido exitosa para terminar usando un ***return render*** a la plantilla necesaria, que sería la de "***datasets.html***" (Código A.3).

A continuación, las vistas que siguen son para crear el modelo de Inteligencia Artificial y para testearlo después. Como estas vistas siguen exactamente el mismo razonamiento no se verán y se pasará directamente a la vista que se usa para crear a los distintos jugadores que participarán en la partida.

Vista Creación Jugadores => get_info_players

En esta vista la idea es recibir la información de la página web referente al formulario para crear un jugador, para ello haremos uso del Código 3.35:

Código 3.35 Vista get_info_players.

```
def get_info_players(request):
    if 'player_session_id' not in request.session:
        # Creo un identificador único perteneciente a la sesión del navegador
        request.session['player_session_id'] = str(uuid.uuid4())
    # Asigno ese identificador a todos los jugadores que se vayan creando
    session_id = request.session['player_session_id']
    # Creamos el jugador crupier y el jugador usuario, comprobamos si no
        existen
    if not Player.objects.filter(session_id=session_id, type_player="user").
        exists() and not Player.objects.filter(session_id=session_id,
        type_player="dealer").exists():
        # si no existen los jugadores usuario y crupier, los creamos
        Player.objects.create(session_id=session_id, type_player="dealer")
        Player.objects.create(session_id=session_id, type_player="user")
    form = PlayerForm(request.POST) # creamos el formulario del jugador
    ai_files = get_model_files() # obtenemos los nombres de los modelos
    q_files = get_q_files() # obtenemos los nombres de los ficheros de la tabla
         Q
    context = {
        'form': form,
        'ai_files': ai_files,
        'q_files': q_files,
        'players': Player.objects.filter(session_id=session_id).exclude(
            type_player="dealer"),
```

```
    }
    if request.method == "POST": # comprobamos que la request que recibimos sea
         del tipo POST
        if form.is_valid(): # comprobamos si es válido el formulario
            create_player(request, form) # creamos el jugador pasándole el
                formulario como parámetro para obtener los valores
            context["players_created"] = len(Player.objects.filter(session_id=
                session_id).exclude(type_player="user").exclude(type_player="
                dealer"))
            context["players"] = Player.objects.filter(session_id=session_id).
                exclude(type_player="dealer")
            return render(request, 'blackjack/create_players.html', context)
    return render(request, 'blackjack/create_players.html', context)
```

Recibiremos al igual un objeto con los datos de la petición HTTP como parámetro para mantener un flujo y lo primero que se hará es comprobar si en la sesión del objeto de la petición HTTP nos viene el parámetro "***player_session_id***", esto lo hacemos para saber si estamos ante una nueva sesión de juego o estamos en la misma. En caso de que no venga, crearemos un identificador único con el uso de la librería ***uuid*** y lo asignaremos a la sesión y a los jugadores creados en dicha sesión. Continuaremos comprobando si en esa sesión existen ya jugadores ***USER*** y ***DEALER*** para que en caso contrario crearlos con la ayuda de Django (y que guardaremos en la base de datos).

Con los usuarios y el identificador creados lo que sigue es crear el formulario de los jugadores que se rellenará con la información del objeto HTTP de tipo POST, recuperar los archivos de los modelos de Inteligencia Artificial y de la Tabla Q. Además, junto a toda esta información que pasaremos por contexto, tendremos también una lista con todos los jugadores creados, para ello volveremos a hacer uso de Django y sus sentencias de SQL, en este caso queremos recuperar todos los jugadores creados que no sean del tipo ***DEALER***.

Después de crear el contexto, comprobamos si el objeto de petición HTTP es tipo POST y si el formulario es válido para que así podamos crear el jugador haciendo uso de la función vista en el Apartado 3.2.1.8, a la que le pasaremos por parámetro el objeto de la petición HTTP y el formulario. Una vez creado el jugador, actualizamos el contexto de los jugadores y añadimos también al contexto la variable "***players_created***" que hace referencia al tamaño de la lista de jugadores que se han ido creando, excluyendo al usuario y al crupier porque el límite de creación de jugadores establecido es sin tomar en cuenta a éstos jugadores.

Vista para editar jugadores => edit_players

El inicio de esta vista es muy semejante a la anterior:

- Recibimos como parámetro el objeto de la petición HTTP.
- Creamos el formulario de jugadores, obtenemos los archivos de los modelos de IA y de las Tablas Q.
- Lo añadimos todo al contexto. Incluido los jugadores ya creados.

La diferencia principal que encontramos en esta vista es que está pensada bien para editar al jugador o bien para borrarlo, para ello haremos uso del Código 3.36:

Código 3.36 Vista edit_players.

```
def edit_players(request):
    ai_files = get_model_files() # obtenemos los nombres de los modelos
    q_files = get_q_files() # obtenemos los nombres de los ficheros de la tabla
         Q
    context = {
        'ai_files': ai_files,
```

```
        'q_files': q_files,
        'players': Player.objects.filter(session_id=request.session['
            player_session_id']).exclude(type_player="dealer"),
        'size_before': len(Player.objects.filter(session_id=request.session['
            player_session_id']).exclude(type_player="dealer").exclude(
            type_player="user")),
    }
    if request.method == "POST":
        form = PlayerForm(request.POST) # creamos el formulario del jugador
        context['form'] = form
        if request.POST.get("edit_player_id"): # si queremos editar el jugador
            player_id = request.POST.get("edit_player_id") # obtenemos el id del
                 jugador a editar
            if form.is_valid(): # comprobamos que el formulario sea válido
                edit_player(request, form, player_id) # editamos el jugador
                context["players"] = Player.objects.filter(session_id=request.
                    session['player_session_id']).exclude(type_player="dealer")
                context["players_created"] = len(Player.objects.filter(
                    session_id=request.session['player_session_id']).exclude(
                    type_player="dealer"))
                return render(request, 'blackjack/create_players.html', context)
            else:
                print(form.errors)
        if request.POST.get("delete_player_id"): # si queremos eliminar el
            jugador
            player_id = request.POST.get("delete_player_id") # obtenemos el id
                del jugador a borrar
            delete_player(request, player_id) # borramos el jugador
            context["players"] = Player.objects.filter(session_id=request.
                session['player_session_id']).exclude(type_player="dealer")
            context["players_created"] = len(Player.objects.filter(session_id=
                request.session['player_session_id']).exclude(type_player="
                dealer"))
            return render(request, 'blackjack/create_players.html', context)
        if form.is_valid(): # si volvemos a recibir info para crear un jugador
            create_player(request, form) # creamos el jugador pasándole el
                formulario como parámetro para obtener los valores
            context["players_created"] = len(Player.objects.filter(session_id=
                request.session['player_session_id']).exclude(type_player="
                dealer"))
            context["players"] = Player.objects.filter(session_id=request.
                session['player_session_id']).exclude(type_player="dealer")
            return render(request, 'blackjack/create_players.html', context)
        else:
                print(form.errors)
    return render(request, 'blackjack/create_players.html', context)
```

Una vez sabemos que estamos ante una petición de tipo POST será necesario comprobar si se quiere eliminar o editar el jugador y para ello vemos si lo que recibimos es el identificador en un ***input*** de HTML cuyo nombre sea "***edit_player_id***" o "***delete_player_id***":

- Si recibimos para editar el jugador, obtenemos el identificador, comprobamos si el formulario es válido y en dicho caso, llamamos a la función encargada de editar el jugador (Apartado 3.2.1.8), en caso de no ser válido el formulario se imprimiría los fallos por la terminal para mi control (ya que con las comprobaciones tanto en el lado del cliente como del servidor

es complicado que se llegue a producir dicho fallo). Por último, actualizaríamos los valores del contexto y renderizaríamos la plantilla.

- Si recibimos para eliminar el jugador, obtenemos el identificador y llamamos a la función encargada de borrarlo. Por último, actualizamos los valores del contexto y renderizamos la plantilla deseada.

Por último, cabe la posibilidad de que después de eliminar o borrar un jugador, se quiera crear uno nuevo otra vez y por ello damos la posibilidades con la comprobación de si el formulario es válido (destacar el orden de los bucles condicionales, para que las acciones tomadas sean correctas) y en dicho caso, aplicar los mismos pasos que vimos en la vista de creación de jugadores anteriormente.

A continuación, entramos ya en las vistas que toman la información de la partida y aplican toda la lógica detrás del juego en la página web.

Vista game

Esta es la primera vista y la inicial para la partida, aquí aplicaremos la lógica del reparto inicial de cartas y comprobaciones de blackjack. Además, también se hará una comprobación de si se reinicia como tal la partida o es sólo el inicio de una ronda más.

Esta vista es una de las dos que posee el decorador (necesario debido a que con la petición AJAX que se realiza con HTMX, no se incorpora el CSRF) que vimos en las importaciones, para hacer que funcione debe de usarse previo a la declaración de la vista y delante hay que indicar un "@", como vemos en el Código 3.37:

Código 3.37 Vista game => declaración.

```
@csrf_exempt
def game(request):
    players = Player.objects.filter(session_id=request.session['
        player_session_id']).exclude(type_player="dealer") # recuperamos todos
        los jugadores creados menos el crupier
    # comprobamos si ya hay un mazo creado para la sesión
    if 'deck' not in request.session: # si no hay un mazo creado
        deck = Deck() # lo creamos y lo guardamos en la sesión
        request.session['deck'] = deck.serialize()
    else: # si ya hay un mazo creado, lo recuperamos y actualizamos su valor de
         creación a False
        deck_data = request.session['deck']
        deck = Deck.deserialize(deck_data)
        deck.created = False

    if request.method == "POST" and request.POST.get("reboot") == "true": # si
        recibimos que quiere reiniciar la partida
        Player.objects.filter(session_id=request.session['player_session_id']).
            update(
            wins=0, losses=0, ties=0, win_percent=0, loss_percent=0,
                ties_percent=0) # reiniciamos los valores de victorias y
                derrotas
        # Ponemos un orden aleatorio a los usuario
        players = list(players)
        random.shuffle(players)
        # Actualizamos el mazo en nuestra sesión
        deck = Deck()
        request.session['deck'] = deck.serialize()
```

Además, en este mismo extracto del código podemos ver cómo hacemos la comprobación de si se quiere reiniciar la partida. En dicho caso, lo que se hace es filtrar todos los objetos jugadores que

no sean el crupier y actualizarle los valores relacionado con las victorias (número y porcentaje) a 0. Posteriormente, entramos ya a la utilidad general de la vista recuperando todos los jugadores que no sean el crupier y creando el deck para jugar la partida.

Una vez tenemos todos los jugadores y el deck, recuperamos al crupier y limpiamos las manos de todos los jugadores. Con las manos vacías de todos los jugadores, realizamos el reparto de las cartas como vemos en el Código 3.38:

Código 3.38 Vista game => reparto.

```
for _ in range(2): # repartimos las cartas a los jugadores
    for player in players:
        player.hand.append(deck.deal_card())
        player.save()
    dealer.hand.append(deck.deal_card())
```

Se reparte de manera que se asemeje a la realidad, una carta a todos primero, después al crupier y se repite el orden una segunda vez. Con las cartas ya repartidas, será necesario comprobar si ha habido algún blackjack, esto lo haremos por partes como vemos en el siguiente código:

Código 3.39 Vista game => comprobar blackjack.

```
if hand_value(dealer.hand) == 21:
        dealer.blackjack = True
    else:
        dealer.blackjack = False
    dealer.save()

# Actualizamos los valores de las manos de los jugadores
for player in players:
    player.hand_value = hand_value(player.hand)
    if player.check_blackjack(): # comprobamos si el jugador ha logrado un
        blackjack
        player.blackjack = True
        player.finished = True
        if player.type_player == "q":
            with open("blackjack/modelos/Tabla_Q/"+str(player.model_name)+".
                pickle", "rb") as file:
                Q = pickle.load(file) # cargamos la Q elegida
            current_state = create_state_values(player.hand, dealer.hand) #
                creamos el valor del estado actual
            action_q = 1 # generamos la acción que debe ejecutar la IA que deber
                ía de ser quedarse === 1
            reward = 1 # si el jugador Q tiene blackjack, indicamos que su
                recompensa es 1 === gana
            # añadimos los movimientos a la lista
            currentEpisode.append((current_state, action_q, reward))
            # actualizamos los movimientos en la tabla Q y vaciamos el set de
                movimientos
            currentEpisode = np.array(currentEpisode, dtype="object") #
                transformamos el array a un np.array para poder trabajar con él
                en la Q
            Q = setQ(Q, currentEpisode, gamma, alpha)
            currentEpisode = [] # la lista que contiene los movimiento de la
                partida
    else:
```

```
        player.blackjack = False
        player.finished = False
    player.save()
```

Para comprobar el blackjack del crupier simplemente será necesario saber si la valoración de sus cartas es 21 o no. En el caso de los jugadores, seguiría la misma lógica pero lo haríamos ayudándonos de sus propiedades. En caso de tener el blackjack alguno de los jugadores, actualizamos su campo de blackjack y de turno finalizado a verdadero y sólo en el caso de que el jugador sea tipo Q-Learning tendríamos que actualizar la tabla con la victoria.

Una vez sabemos qué jugadores han logrado blackjack. Después, será necesario saber quién es el jugador que va primero (hay que tener en cuenta que no debe de tener blackjack) y quién es el jugador que va después de él, a su vez también es necesario saber las posiciones de dichos jugadores. Para ello, nos ayudaremos del Código 3.40:

Código 3.40 Vista game => Averiguar posiciones y orden de los turnos.

```
if len(players) > 1:
    next_player = players[1]
    next_slot = 2 # posición del siguiente jugador
else:
    next_player = dealer
    next_slot = None

my_slot = 1 # posición del jugador actual
for i, player in enumerate(players):
    if not player.blackjack:
        first_player = players[i]
        my_slot = i+1
        break
```

Para averiguar la posición y quién va, recorreremos la lista de los jugadores precedida con ***enumerate*** para saber exactamente el índice del jugador que va, con ello comprobamos que no tenga blackjack y el primero que no lo tenga ese será el primero en ir y su posición será su índice más 1 (debido a la enumeración de Python).

Por último, tendremos que guardar ciertos valores en la sesión para que no se pierdan en el navegador cuando vamos avanzando entre las vistas, estos valores serán:

- El deck, para mantener un mismo deck. Para guardarlo en la sesión nos ha hecho falta usar su propiedad de serialización que vimos en el archivo de ***models.py***.
- La lista de los jugadores, para mantener el mismo orden. Para guardar en la sesión a los jugadores, haremos lo mismo que con el deck y usaremos la propiedad de serialización. Además, los jugadores estarán guardados en la sesión y en la base de datos para que cuando se cierre la sesión, dichos jugadores se mantengan y así se pueda recuperar la partida. De esta manera, se tendrá como un "backup" de la sesión.
- El primer y segundo jugador que van a parte de sus posiciones en la mesa. Es necesario saber qué jugadores van para indicarlos en la pantalla y a su vez, también es necesario saber la posición que ocupan en la partida para actualizar únicamente esa parte de la pantalla.

Con todos los valores guardados ya en la sesión, será necesario pasar también dichos valores por contexto para cargarlos en la plantilla. Para finalizar el apartado de las vistas, veremos la explicación de la vista encargada de aplicar la lógica de cada turno en la partida.

Vista para aplicar la lógica => select_action

Esta vista, al igual que la anterior, también posee el decorador. Esta vista es la encargada de recibir todas las peticiones AJAX que se hagan a la página usando HTMX. Además, tendrá también como argumentos a recibir el identificador del jugador cuyo turno es. Por tanto, lo primero que será necesario hacer es recuperar todos los valores de la sesión que pasamos en la anterior función (con el uso de la propiedad de deserialización cuando toque) y crear las variables necesarias para el desarrollo de ésta.

A continuación, con todas las variables ya creadas, comprobaremos que la petición sea de tipo POST y en dicho caso, obtendremos qué jugador es el que va a raíz de su identificador y para ello haremos uso de la función vista en el Apartado 3.2.1.8. Recuperado el jugador, hace falta saber que tipo de jugador es para aplicar una u otra lógica.

Para ello, comprobaremos que no tenga blackjack ni que sea del tipo ***DEALER***, si cumple esas dos condiciones, comprobaremos su tipo. Sabiendo su tipo, aplicamos su lógica:

- ***Jugador IA***: cargamos el modelo de IA elegido para ese modelo y dependiendo de si cuenta cartas o no, crearemos un set de datos con el que irá prediciendo la acción.
- ***Jugador Q-Learning***: cargamos la Tabla Q elegida y aplicamos la lógica vista en el Apartado 3.2.1.2.
- ***Jugador NPC***: dependiendo de si cuenta cartas o no, su lógica se aplicará siguiendo el algoritmo Hi-Lo o siguiendo la tabla de estrategias.

Aquí como podemos comprobar, dependiendo del tipo se aplicaría una lógica u otra pero hay algo común para todos, y es averiguar quién es el jugador que va justamente después de ellos en la partida, esto se realizará mediante el Código 3.41:

Código 3.41 Vista select_action => Averiguar siguiente jugador.

```
# sacamos el índice del jugador actual, para ubicarnos en el orden de la
    partida
try:
    current_index = players.index(player)
    my_slot = current_index + 1 # indicamos cuál es el slot dónde están
        ubicadas las cartas de este jugador
    next_index = (current_index + 1) % len(players)
    while True: # comprobamos de quién es el siguiente turno
        if next_index == 0: # si volvemos al inicio, el siguiente jugador es el
             crupier
            potential_next_player = dealer
            break
        else:
            potential_next_player = players[next_index] # obtenemos quién sería
                el siguiente jugador
            if potential_next_player.blackjack: # si el siguiente jugador ha
                logrado blackjack, saltamos al siguiente jugador
                next_index = (next_index + 1) % len(players) # sumamos 1 al í
                    ndice y volvemos a comprobar
                if next_index == 0: # si volvemos al inicio, el siguiente
                    jugador es el dealer
                    potential_next_player = dealer
                    break
                else: # si no hemos vuelto al inicio, volvemos a comprobar si el
                     siguiente jugador ha logrado blackjack
                    continue
```

```
            else:
                break
    next_slot = next_index + 1 # indicamos cuál es el slot dónde están ubicadas
        las cartas del siguiente jugador
except ValueError:
    print("ESTE JUGADOR NO PERTENECE A LA LISTA DE JUGADORES")
```

Comenzaríamos averiguando qué jugador es el actual (ya que puede haber más de un jugador del mismo tipo) y su posición (hace referencia a slot y next_slot en el código) en la mesa. Con ello, el siguiente índice simplemente es sumarle 1 al actual pero como queremos que no sobrepase el límite, le calculamos el módulo sobre la longitud de la lista de jugadores. Sabiendo cuál es el siguiente índice, falta comprobar quién es y si tiene que jugar para lograrlo. Lo que haremos será recorrer toda la lista y si el jugador ha terminado o tiene blackjack, pasamos al siguiente índice y en caso de que dicho índice sea 0, el siguiente jugador sería el crupier.

Visto cómo se aplicaría la lógica de cada turno, queda ver al crupier que simplemente sería añadir una carta a su mano hasta que esta sume un mínimo de 17. Pero además, también hay que tener en cuenta que después de su turno hay que mostrar las valoraciones e iniciar una siguiente ronda o en su defecto reiniciar la partida.

Primeramente comprobamos que la petición sea de tipo POST (esto ya había sido comprobado desde el inicio pero para indicarlo) ya que en caso de que la petición fuese de tipo GET lo que haríamos sería recargar la página. Sabiendo que es tipo POST, el orden sería primero empezar con el turno del crupier y como se ha explicado antes, pide hasta llegar a 17. Por último, si la petición de tipo POST viene con un mensaje pidiendo que se muestren los resultados:

1. Se recorren todos los jugadores.
2. Se comprueba si han logrado blackjack o no.
3. Se comprueba si el crupier o jugador se han pasado de 21.
4. Si ninguno de los dos se han pasado de 21, se comprueba quién tiene una mayor puntuación.

Con la explicación de esta vista, terminamos el apartado de todas las vistas del archivo y proseguiremos con la explicación de las funciones que son usadas por las vistas para poder realizar las ***CRUD*** (Create, Read, Update, Delete).

3.2.1.8 Funciones "CRUD"

En este apartado se verán las funciones cuyo uso están en la creación de los jugadores, edición y borrado de estos. Empecemos viendo una a una.

Función obtener jugador => get_player

Aquí recibiremos cómo parámetro el identificador del jugador que se quiere obtener, después será necesario saber si existe algún jugador con ese identificador que pertenezca y tenga la misma sesión en la que nos encontramos. Esto se logrará con el siguiente código:

Código 3.42 Función get_player.

```
def get_player(request, player_id):
    if Player.objects.filter(session_id=request.session['player_session_id'],
        id=player_id).exists():
        player = Player.objects.get(session_id=request.session['
            player_session_id'], id=player_id)
        return player
```

Por tanto, si existe dicho jugador, recuperamos el objeto del jugador y lo devolvemos. Visto ya cómo recuperamos el jugador, pasaremos a explicar la función que usamos para aplicar la acción elegida por el jugador usuario en la partida.

Función aplicar acción => apply_action

Para esta función recibiremos como parámetros la petición, el jugador que toca y el deck para poder robar la carta, esto se logrará con el Código 3.43:

Código 3.43 Función apply_action.

```
def apply_action(request, player, deck):
    # si elige pedir carta y no ha terminado el turno
    if (request.POST.get('action') == 'hit' and not player.finished):
        player.hand.append(deck.deal_card()) # añadimos una carta a la mano del
            jugador
        player.hand_value = hand_value(player.hand) # añadimos el valor de la
            mano actual del jugador
        # comprobamos que no se pase de 21 o si ha pedido 3 cartas y no ha
            perdido
        if player.hand_value >= 21:
            player.finished = True
            return player
        elif len(player.hand) == 5:
            player.finished = True
            return player
        else:
            player.finished = False
            return player
    # si la acción decidida es quedarse, se termina nuestro turno y pasamos al
        siguiente
    elif request.POST.get('action') == "stand":
        player.finished = True
        return player
    return player
```

Simplemente se comprobará que el jugador no haya terminado su turno y que haya decidido pedir carta, si eso se cumple, añadimos carta y aumentamos el contador. Después, se verificará que no se cumple ninguna condición que haya terminar el turno (como superar 21 o lograr cumplir la FCCR). Por último, en caso de que el jugador eligiese terminar el turno, actualizamos su campo de verificación del turno terminado y devolveremos el jugador.

Vistas las dos funciones encargadas de obtener al jugador y aplicar la lógica para el usuario, ahora pasaremos a ver las funciones referentes a crear el jugador, editarlo y borrarlo.

Función crear jugador => create_player

En esta función recibimos el objeto de la petición HTTP y el formulario como parámetros, para poder obtener los valores que el usuario haya elegido para crear al jugador. Veamos el código encargado de lograr esto:

Código 3.44 Función create_player.

```
def create_player(request, form):
    if request.POST.get("type_player") and form.cleaned_data["type_player"]:
        type_player = form.cleaned_data["type_player"] # obtenemos el campo del
            tipo de jugador
    if request.POST.get("ai_files") and form.cleaned_data["ai_files"]:
        ai_files = form.cleaned_data["ai_files"] # obtenemos el campo del
            nombre del modelo de IA
    if request.POST.get("q_files") and form.cleaned_data["q_files"]:
```

```
        q_files = form.cleaned_data["q_files"] # obtenemos el campo del nombre
            del fichero de la tabla Q
    if request.POST.get("count_cards") and form.cleaned_data["count_cards"]:
        count_cards = True # si recibimos un valor por la petición POST para
            count_cards indicamos que es True su valor
    else:
        count_cards = False # si no recibimos ningún valor, indicamos que el
            valor es False
    # hacemos las comprobaciones de las posibles combinaciones de jugadores
    if type_player == "ai" and count_cards: # si el jugador es IA y contador de
         cartas
        model_name = ai_files+"_count"
    elif type_player == "ai" and not count_cards: # si el jugador es IA y no
        contador de cartas
        model_name = ai_files
    elif type_player == "q": # si el jugador es tipo Q-Learning
        model_name = q_files
    else: # si el jugador es NPC
        if count_cards: # si el NPC cuenta cartas
            model_name = "hi_lo" # el nombre de su modelo será HI-LO haciendo
                referencia al algoritmo de contar cartas
        else: # si el NPC no cuenta cartas
            model_name = "table" # el nombre de su modelo será table, haciendo
                referencia a qué accede a una tabla de estrategias
    # creamos el jugador
    new_player = Player(session_id=request.session['player_session_id'],
        model_name=model_name, type_player=type_player, count_cards=count_cards
        , created_at=timezone.now)
    new_player.save() # guardamos el jugador
    print("Jugador creado correctamente")
```

Para comprobar si nos viene un valor o no en el formulario, hace falta ver que el campo del formulario esté relleno y venga en el valor del objeto de la petición HTTP. Entonces, si se cumple, se lo asignamos a una variable. Por último, hará falta saber qué tipo de jugador es el elegido y su modelo, es por eso que se ve si el tipo de jugador es la IA y cuenta cartas, será el modelo contador de cartas y así con el resto de tipo de jugadores.

Con todos los valores ya recuperados y sabidos el tipo de jugador, creamos el jugador haciendo uso de Django y lo almacenaremos en la base de datos. Para crearlo simplemente asignaremos valores a cada atributo que queramos rellenar y lo guardaremos haciendo uso de la propiedad ***save*** de los modelos de Django.

Vista la función que crea el jugador, continuaremos con la función que edita el jugador.

Función editar jugador => edit_player

Aquí recibiremos al igual que en la función de crear el jugador, el objeto de la petición HTTP y el formulario pero a parte también se recibirá como parámetro el identificador del jugador que se quiera editar.

Comenzaríamos de igual manera, comprobando y recuperando todos los valores del formulario pero a la hora de modificar el jugador se realizaría según el Código 3.45:

Código 3.45 Función edit_player.

```
if Player.objects.filter(session_id=request.session['player_session_id'], id=
    player_id).exists():
        Player.objects.filter(session_id=request.session['player_session_id'],
            id=player_id).update(model_name=model_name, type_player=type_player,
             count_cards=count_cards)
```

Verificamos si el jugador con ese identificador existe en nuestra sesión y en dicho caso, filtraríamos por la sesión y su identificador para actualizar todos los campos que son abarcados en la creación.

Para acabar con las funciones ***CRUD*** del jugador, veremos la función usada para borrar al jugador de la base de datos.

Función borrar jugador => delete_player

Al igual que en la función para la edición del jugador, recibimos como parámetros el objeto de la petición HTTP y el identificador del jugador pero con la diferencia de que aquí no hará falta un formulario.

Sabiendo ya lo que debe recibir como parámetros la función, el código necesario para borrarlo se asemeja también al visto para editarlo:

Código 3.46 Función delete_player.

```
def delete_player(request, player_id):
    if Player.objects.filter(session_id=request.session['player_session_id'],
        id=player_id).exists():
        Player.objects.filter(session_id=request.session['player_session_id'],
            id=player_id).delete()
        print("Jugador borrado correctamente")
    else:
        print("El jugador con ese identificador no existe")
```

Comprobamos que exista el jugador con ese identificador en nuestra sesión y si existe filtramos en los jugadores por ese identificador y sesión y lo borramos con la ayuda del método ***delete*** de los modelos de Django.

Para acabar con el apartado de las funciones, veremos la encargada de aplicar la lógica del algoritmo Hi-Lo para contar las cartas.

Función Algoritmo Hi-Lo => hi_lo

En esta función recibiremos como parámetros las cartas que han salido en la partida y la cuenta del deck que llevará el jugador NPC que sabe contar cartas, estos valores se los pasará el NPC a esta función cuando la llame ya que el objetivo de esta función es actualizar la cuenta y para ello nos ayudaremos del Código 3.47:

Código 3.47 Función hi_lo.

```
def hi_lo(cards, count):
    # recorremos todas las cartas
    for card in cards:
        if card in ["2", "3", "4", "5", "6"]: # si la carta está dentro de las
            "bajas", sumamos 1
            count += 1
        elif card in ["A", "10", "J", "Q", "K"]: # si la carta está dentro de
            las "altas", restamos 1
            count -= 1
    return count # devolvemos la cuenta
```

Por tanto, lo que haremos será recorrer todas las cartas con un **for** y si pertenece al grupo de las llamadas cartas bajas, sumaremos 1 a la cuenta y si por el contrario pertenece al grupo de las cartas altas, restaremos uno. De esta manera, tendremos siempre actualizada la cuenta real de la baraja.

Explicada la parte del Backend, proseguiremos con la explicación del Frontend y esto lo haremos mostrando previamente como se ve la página web para posteriormente explicar el código que he escrito para llegar a ese funcionamiento y estilo.

3.2.2 Frontend

En esta sección veremos toda la parte de estilos y funcionalidad en el lado del cliente que se ha implementado en el proyecto. Por facilitar la explicación y mantener una guía, se dividirá en tres apartados:

1. El archivo donde están creadas las clases de los formularios de Django, llamado ***forms.py***.
2. La pantalla inicial.
3. El apartado de los sets de datos.
4. El apartado de la partida.

Todo el código del proyecto referente al apartado del frontend estará incluido en los anexos.

3.2.2.1 Archivo forms.py

En este archivo crearemos las clases que tendrán los parámetros/campos necesarios para las creaciones de las diferentes opciones que propondremos en la página. Comencemos viendo las clases usadas para el apartado de los sets de datos.

Creación Dataset, creación modelo IA y testeo

Lo necesario para la creación del set de datos son los campos que podemos ver en este extracto de código:

Código 3.48 Clase Formulario Dataset.

```
class DatasetForm(forms.Form):
    games = forms.CharField(required=True, max_length=10)
    file_name = forms.CharField(required=True, max_length=100)
```

Esta clase heredará de los formularios de Django.

Continuemos viendo la clase usada para la creación de los modelos de IA. Sigue el mismo estándar pero variando los campos:

Código 3.49 Clase Formulario Modelo IA.

```
class ModelForm(forms.Form):
    from .utils import get_dataset_files
    model_name = forms.CharField(required=True, max_length=100)
    training_size = forms.NumberInput()
    num_epochs = forms.NumberInput()
    file_name = forms.ChoiceField(choices=get_dataset_files(), required=True)
```

Aquí cabe destacar la importación de una función y cómo es usada, su importación se verá más adelante en el Apartado 3.2.1.6. Es la manera que tenemos en Django de informar que queremos un campo de opciones (estilo ***select*** de HTML) e indicamos de esta manera qué opciones son las permitidas. Por último, veremos la clase para el formulario encargado de recibir la información del testeo del modelo:

Código 3.50 Clase Testeo Modelo IA.

```
class TestModelForm(forms.Form):
    from .utils import get_model_files
    file_model = forms.ChoiceField(choices=get_model_files(), required=True)
    num_games = forms.NumberInput()
```

Igual que en el caso del formulario para el modelo pero intercambiamos la función para en vez de obtener los datasets creados, obtengamos los modelos creados.

Creación/Edición Jugadores

En esta clase, crearemos los campos necesarios para poder crear un jugador para la partida, veámoslo más a fondo en el Código 3.51:

Código 3.51 Clase Formulario Jugador.

```
class PlayerForm(forms.Form):
    type_player_choices = [
        ('ai', 'Modelo de Inteligencia Artificial'),
        ('q', 'Algoritmo Q-Learning'),
        ('npc', 'Jugador NPC (Non Playable Character)'),
    ]
    from .utils import get_model_files, get_q_files
    ai_files = forms.ChoiceField(choices=get_model_files(), required=True)
    q_files = forms.ChoiceField(choices=get_q_files(), required=True)
    count_cards = forms.BooleanField(required=False)
    type_player = forms.ChoiceField(choices=type_player_choices, required=True)
```

De igual manera, para los campos de archivos de la Tabla Q y para los archivos del modelo de IA usamos funciones creadas que se verán más adelante. Sin embargo, para los tipos de jugadores hemos usado otra de las maneras existentes en Django de indicar las opciones permitidas, mediante un diccionario donde la clave es el valor de la opción del select de HTML y el valor, el texto que muestra la opción.

3.2.2.2 Pantalla Inicial

En esta parte de la página web tendremos desplegado dos recuadros con las diferentes opciones posibles que brinda la página web: crear sets de datos y modelos de Inteligencia Artificial y jugar al blackjack. Esto será presentado en la página web, según la Figura 3.6:

Figura 3.6 Pantalla Inicial de la página web.

Además, cómo se puede ver, hay una pequeña descripción de lo que nos encontramos en cada parte.

Explicada la pantalla principal, pasaremos a ver el apartado de la creación de un set de datos, modelos de Inteligencia Artificial y su posterior testeo.

3.2.3 Creación de un set de datos

Aquí tendremos una pantalla (manteniendo el mismo estilo y fondo) dónde aparecerán tres formularios como si fueran un acordeón, es decir, habría que pulsar dónde se quiera para que se despliegue su formulario correspondiente además de tener un botón para poder volver a la pantalla inicial. Lo descrito lo podremos encontrar en la Figura 3.7.

En esta parte del formulario de creación del set de datos, tendremos campos con mensajes de ayuda para guiar al usuario en su utilización. Además, se ha incluido una pequeña banda que ayuda a mantener un poco la guía de lo que sucede y es que al crear el set de datos muestra un mensaje de éxito. Para lograr su creación se ha usado los formularios que viene con HTML.

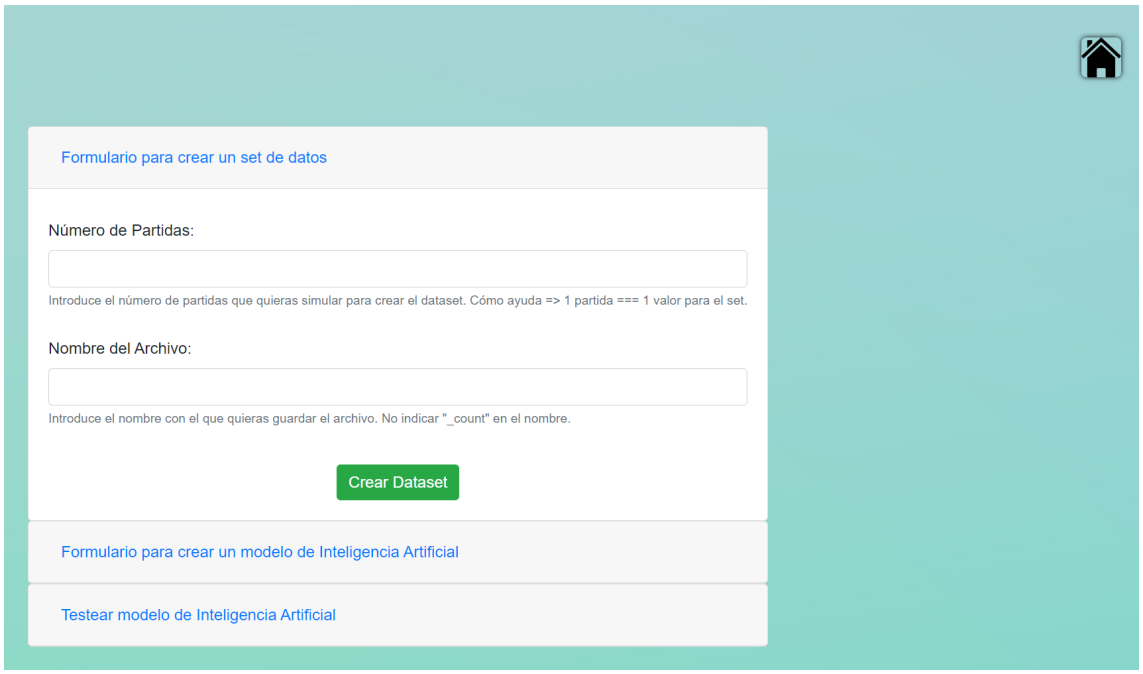


Figura 3.7 Pantalla creación de sets de datos.

3.2.3.1 Creación del modelo de Inteligencia Artificial

Una vez hemos creado el set de datos, nos iremos al apartado de formulario del modelo de Inteligencia Artificial para poder crearlo. Al igual que con el set de datos, al crearlo mostrará un texto de éxito como podemos ver en la Figura 3.8.

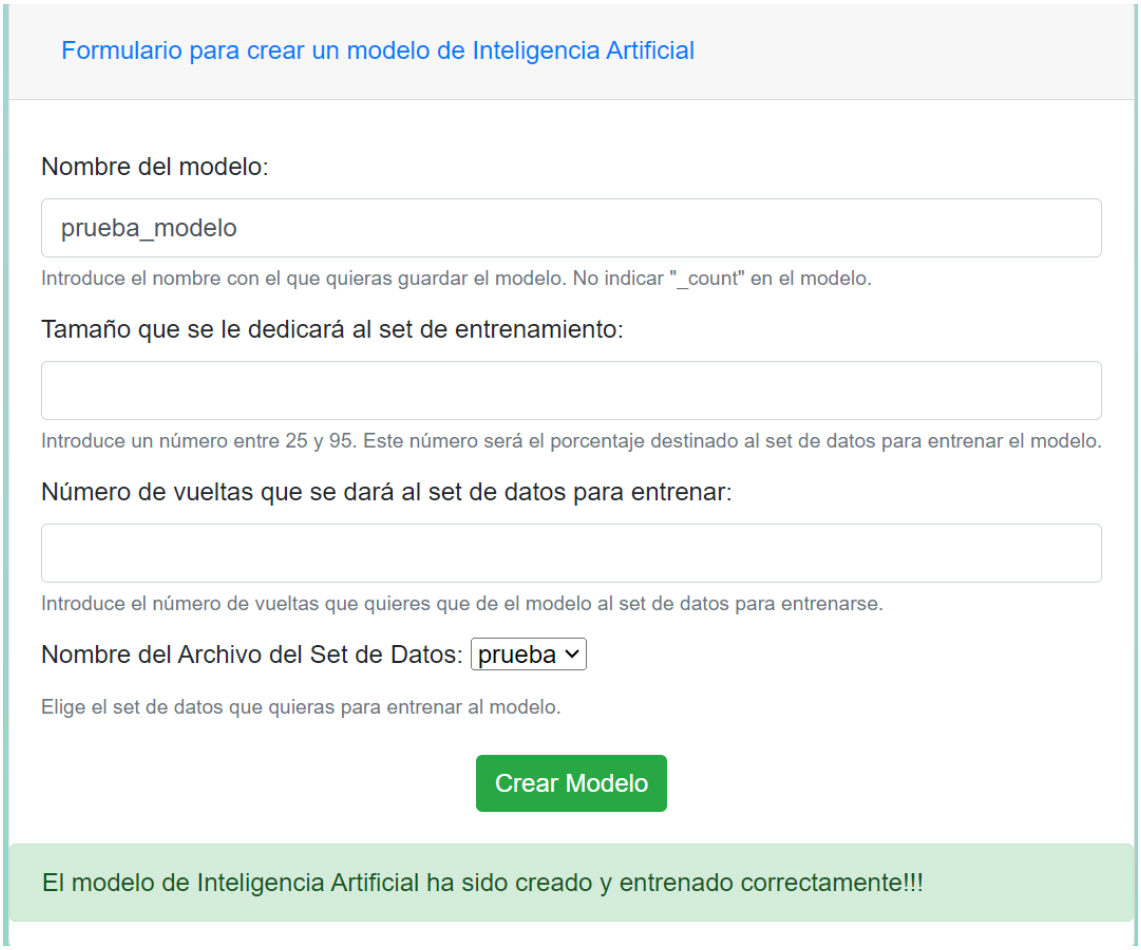


Figura 3.8 Formulario creación modelo de Inteligencia Artificial.

Aquí, destacaremos la parte del desplegable con el nombre del los archivos creados ya de set de datos ya que está hecha con la ayuda de Django. Haciendo un pequeño zoom del Código A.3 a esta parte, tendríamos el siguiente extracto:

Código 3.52 Desplegable archivos de los sets de datos.

```
<select id="id_file_name" class="form-select" name="file_name" required>
    {% for file, _ in data_files %}
        <option value="{{ file }}">{{ file }}</option>
    {% endfor %}
</select>
```

Como se ve, se ha hecho con el select que proporciona HTML pero para las diferentes opciones se ha implementado un *for* con la ayuda de Django, para poder recorrer así todos los archivos que obtuvimos de la función y pasamos por contexto, tal y como explicamos en la función vista en el Apartado 3.2.1.6.

A continuación, pasaremos a ver la parte del formulario para poder testear los modelos de Inteligencia Artificial y cómo se muestran los resultados de dicho testeo.

3.2.3.2 Testeo modelos de Inteligencia Artificial

Para este formulario, sólo nos será necesario indicar el número de partidas que queremos que se simule y el modelo que participará en las partidas (se testean tanto el modelo contador de cartas como el no contador), este formulario se ve como en la siguiente imagen:

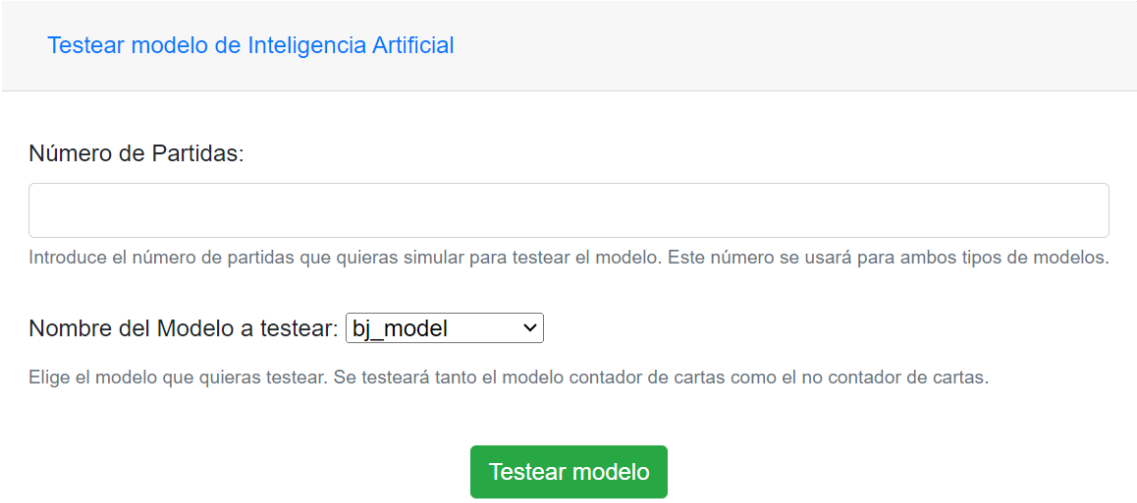


Figura 3.9 Formulario testeo del modelo de Inteligencia Artificial.

El desplegable de los modelos es igual al visto para los sets de datos. Testeado el modelo, nos saldrá un modal de Bootstrap con las estadísticas del testeo: con el número total de partidas que gana, pierde y empata con sus respectivos porcentajes.

Como queremos que se produzca un pequeño retraso a la hora de mostrar los resultados, nos ayudaremos de una función de Javascript que mostramos en el Código 3.53:

Código 3.53 Función para mostrar el modal de las estadísticas.

```
$(document).ready(function() {
    {% if show_modal %}
        setTimeout(function() {
            $('#statisticsModal').modal('show');
        }, 1000);
    {% endif %}
});
```

Explicada toda la sección de la página dedicada a la creación de los sets de datos y modelos de Inteligencia Artificial (incluido su testeo). Procederemos a ir viendo el apartado de jugar. Para empezar con este apartado es clave ver el formulario para crear jugadores que participen en la partida.

3.2.4 Crear Jugadores

Una vez entramos en esta parte de la página web, nos aparecerá un formulario para poder crear jugadores a nuestra izquierda y una tabla con todos los jugadores creados (comienza con sólo el usuario), como podemos ver en la Figura 3.10:

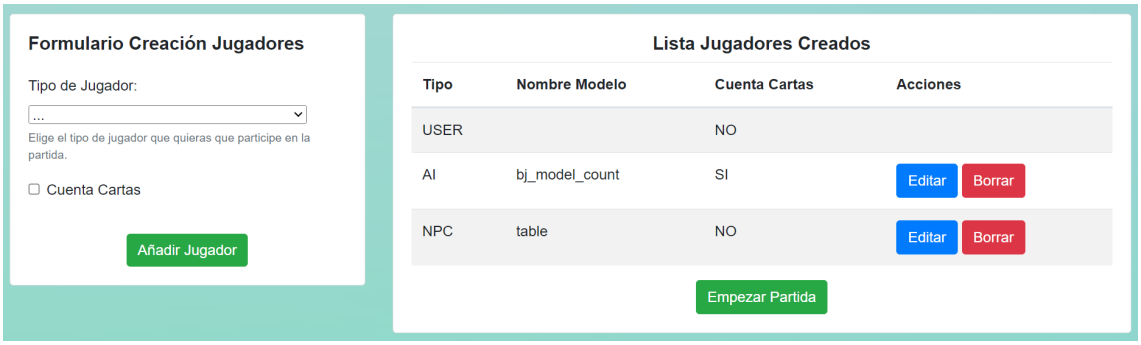


Figura 3.10 Pantalla de creación de jugadores.

Entonces podremos ir creando jugadores hasta un máximo de 5 (ya que no cuenta el usuario como tal) y si se quisiera editar algún jugador o eliminarlo directamente, se brinda la posibilidad en la tabla al pulsar la opción deseada. Para esto, con tal de hacerlo más enfocado en el jugador, nos saldrá un modal (para ambas opciones) con un formulario para poder editar al jugador o con un mensaje de aviso en caso de querer eliminarlo.

Pasemos a explicar el código que nos permite crear dicho formulario. Se usará igual que para los otros vistos, las etiquetas de HTML pero se ha añadido una funcionalidad extra para poder ir ocultando o mostrando las opciones de los archivos según el tipo de jugador deseado, como se muestra en las dos siguientes figuras (Figura 3.11). Ese cambio dinámico en el formulario se ha

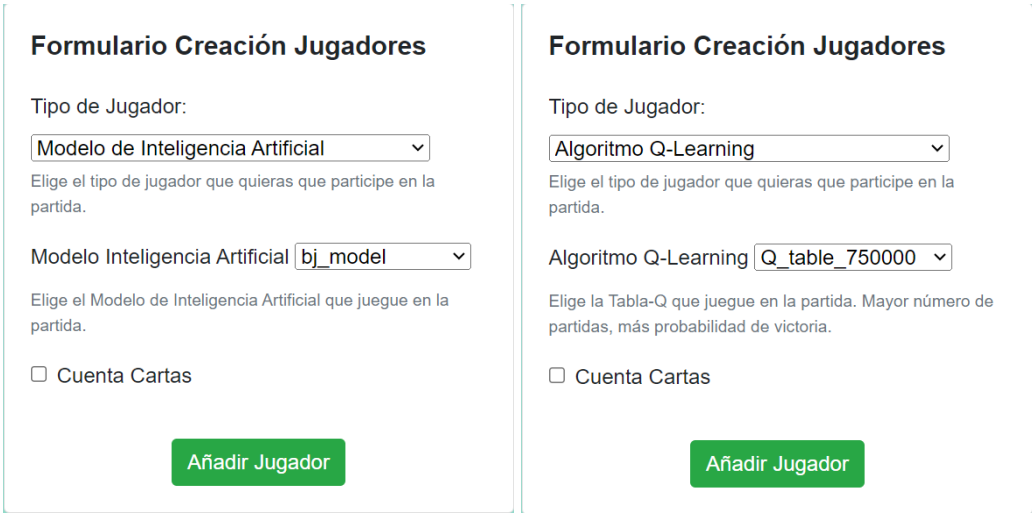


Figura 3.11 Formulario Creación Jugador.

realizado con la ayuda de una función en Javascript, llamada ***toggleField*** que encontramos en el Código A.1. Esta función está encargada de comprobar que tipo de jugador es elegido, si es tipo IA, mostrará los archivos de los modelos de IA y si es tipo Q pues mostrará los archivos de la Tabla Q, en caso de que el tipo de jugador sea NPC ese campo se ocultará ya que no existe archivo de selección para el NPC.

Visto ya la creación de los jugadores para la partida, procederemos a ver la partida ya como tal.

3.2.5 Jugar la partida

Comenzaremos explicando que en esta parte se ha usado una nueva plantilla llamada ***game_base*** para poder cambiar el fondo y así implantar una mesa de blackjack, esta plantilla se puede ver en el Código A.5.

3.2.5.1 Pantalla principal

Una vez tenemos el código base, pasaríamos a incluir y dar estilos a la página principal, como el reparto de cartas y ubicación de distintos botones para acabar obteniendo el resultado que vemos en la Figura 3.12:

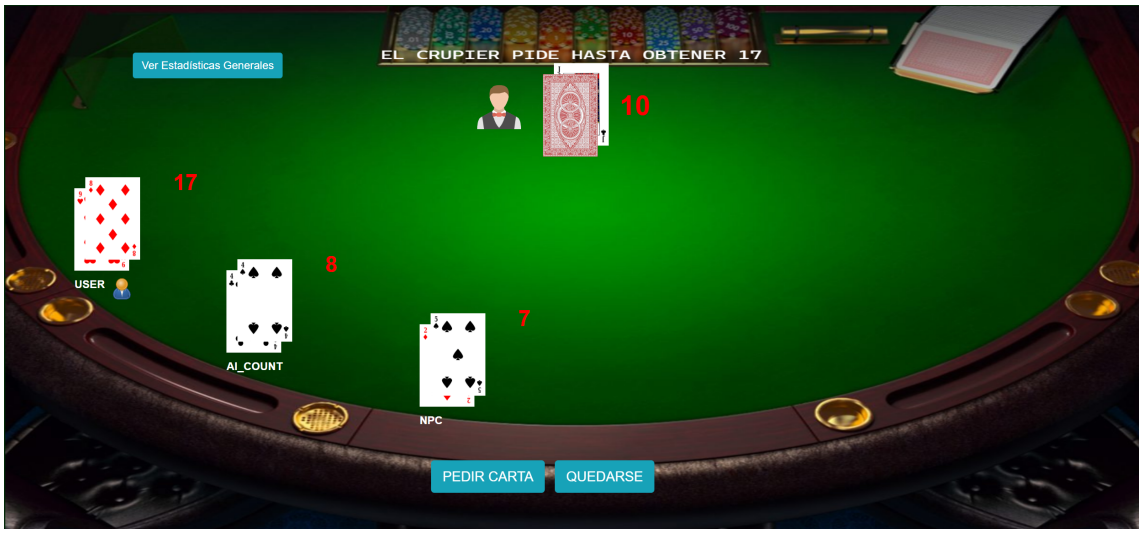


Figura 3.12 Pantalla inicial de Jugar Partida.

En esta pantalla lo que nos encontramos es un fondo que simula un tapete de blackjack donde se muestran todas las cartas de los usuarios (identificando a cada uno por su tipo concreto), arriba centrado las cartas del crupier. Además, también tendremos en la esquina superior izquierda un botón para mostrar las estadísticas de la partida. Por último, tendremos en la parte baja de manera central un botón que indica de quién es el turno y al pulsarlo comenzará la lógica de cada jugador, a excepción del usuario que ya habría que hacerlo pulsando los botones de pedir y quedarse.

Todo este código está presente en el Código A.6, dónde dentro de este presenciamos diferentes *includes* de Django para poder incluir diferentes plantillas que iremos viendo por partes.

Después de ver esta pantalla principal, pasaríamos a jugar la partida utilizando para ello los botones que hemos visto en la parte baja y al terminar la partida, se nos mostrará el botón para ver los resultados que explicaremos en el apartado del crupier. Una vez visto los resultados, la partida se acabaría y entonces aparecería el siguiente botón (Figura 3.13):

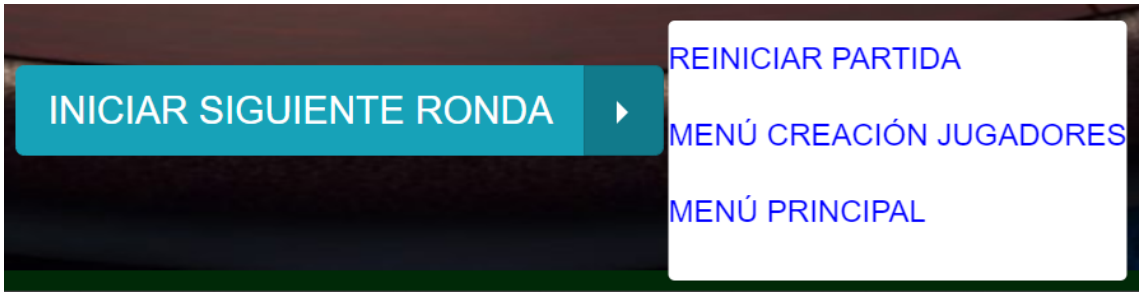


Figura 3.13 Menú disponible al finalizar una ronda de la partida.

Cómo podemos observar, dicho botón nos ofrece las siguientes opciones:

- Inicialmente, jugar una ronda más donde mantendríamos todas las estadísticas y el orden de los jugadores, simplemente limpiamos las manos de los jugadores y repartimos nuevas

cartas.

- Si pulsamos la flecha que aparece a la derecha del botón, se nos despliegan tres nuevas opciones:
  - Empezar una nueva partida, dónde se limpiarían las estadísticas de cada jugador, se variaría el orden de los jugadores y se repartirían nuevas cartas.
  - Volver al menú de creación de los jugadores, por si queremos añadir, editar o eliminar los jugadores.
  - Volver al menú principal.

Todo esta parte del menú, lo encontramos en el Código A.11. Vista ya la parte de la pantalla principal y sus botones, pasaremos a ver primeramente el apartado de las cartas del crupier y finalmente la de los jugadores.

3.2.5.2 Crupier

Este apartado referencia al Código A.8. Aquí, a parte de la lógica para mostrar las cartas del crupier teniendo en cuenta que al principio sólo se deben de mostrar 1 carta y la otra boca abajo, vemos que existe un botón con el indicativo de "VER RESULTADOS" y es que al pulsarlo se nos mostrará un pequeño texto informativo con el desenlace de la partida, mostrando ganadores y perdedores (Figura 3.14: Finalizada la parte del crupier, pasemos a explicar la parte de los jugadores.

Figura 3.14 Resultados de la ronda.

3.2.5.3 Jugadores

Aquí es dónde se ubican todos los jugadores y sus cartas, cómo vemos en el Código A.7. La lógica detrás de todo este código, es recorrer todos los jugadores y ubicarle en una posición dependiendo de su índice en la lista. Además, como queremos dinamismo en la página web, es necesario subdividir cada apartado del jugador para recargar sólo esa plantilla y es por eso la comprobación de qué tipo de jugador es en la plantilla para incluir una u otra.

Plantilla Jugador USER

En esta plantilla es donde se hará el reparto de las cartas del jugador usuario, por tanto, también será necesario indicar sus botones de acción como hemos visto en la pantalla principal y aparece en el Código A.9 (se puede apreciar la inclusión de una imagen para diferenciar claramente al usuario del resto).

A destacar en esta plantilla son los botones y la aplicación de una función Javascript para ocultarlo al pulsarlo. Además, también vemos el uso de la librería HTMX en Python para poder realizar las peticiones AJAX, expliquemos esto de HTMX más a fondo con la ayuda del Código 3.54:

Código 3.54 Botones del usuario.

```
<button type="button" hx-post="{% url 'blackjack:select_action' current_player.
    id %}" hx-vals='{"action": "hit"}' hx-target="#player-slot-{{ my_slot }}"
    class="btn btn-info btn-lg">PEDIR CARTA</button>
<!-- Hacemos la diferenciación de si el siguiente usuario es dealer o no -->
{% if next_player.type_player == "dealer" %}
    <button type="button" hx-post="{% url 'blackjack:select_action' dealer.id
        %}" hx-vals='{"action": "stand"}' onclick="hideParentDiv(this)" hx-
        target="#dealer-slot" hx-swap="innerHTML" class="btn btn-info btn-lg
        margin-left-10">
        QUEDARSE
    </button>
{% else %}
    <button type="button" hx-post="{% url 'blackjack:select_action' next_player
        .id %}" hx-vals='{"action": "stand"}' onclick="hideParentDiv(this)" hx-
        target="#player-slot-{{ next_slot }}" hx-swap="innerHTML" class="btn
        btn-info btn-lg margin-left-10">
        QUEDARSE
    </button>
{% endif %}
```

El funcionamiento que se le da a los botones de quedarse para el usuario es de dar fin a su propio turno pero a su vez dar comienzo al del siguiente jugador, es por ello que se le pasa el identificador del siguiente jugador (o del crupier en caso de ser el usuario el último).

Visto ya la parte del usuario, finalizaremos el apartado del Frontend con la plantilla usada para aplicar el reparto de cartas y los botones de funcionamiento.

Plantilla Jugadores No USER

Aquí se va a ver en profundidad la plantilla del jugador que es tipo IA pero las plantillas para los jugadores tipo NPC y tipo Q-Learning son exactamente iguales no se verán. Es necesario la separación dicha separación por plantillas para los jugadores para que no haya conflictos con las peticiones AJAX. Por tanto el código que hace referencia a esta plantilla es el Código A.10.

Esta plantilla es muy semejante de igual manera a la del usuario, por lo que no se verá en profundidad los botones ya que siguen la misma lógica. Sin embargo, sí explicaremos el porqué de la existencia de una función ***onclick*** en los botones. Todas las funciones Javascript se llaman de manera distinta pero su intención es la misma, ocultar el botón al ser pulsado, se pueden observar en la plantilla del código Código A.5 pero haremos un zoom a una de las funciones (Código 3.55):

Código 3.55 Función Javascript para ocultar el botón.

```
function hideButtonAI(button) {
    if (button.style.display != "none") {
        button.style.display = "none";
    } else {
        button.removeAttribute("style");
    }
}
```

En la función simplemente comprobamos que el botón que llama a la función no tenga el estilo de ***none*** para ocultarse, en caso de no tenerlo se lo añadimos para ocultarlo y en caso de tenerlo se borra el atributo de ***style*** para que se muestre dicho botón.

Explicada toda la parte del Frontend, se procederá a dar la explicación de la parte de Inteligencia Artificial del proyecto: las funciones usadas, modelos creados y su lógica detrás de ellos, así como

de la creación y lógica de la creación de los sets de datos que se usarán para el entreno y predicción de los modelos.

3.3 Inteligencia Artificial

En este apartado se verá la explicación de las funciones y el porqué del uso de los distintos modelos de IA.

3.3.1 Generar set de datos

Comencemos viendo la función encargada de generar el set de datos que se usará para entrenar al modelo de IA. Esto se implementa en una función que recibe como parámetros:

- Número de partidas que queremos simular para generar los datos (sigue una proporción de 1 partida = 1 dato y 1 etiqueta).
- Nombre de archivo con el que queremos que se guarde el set de datos para poder recuperarlo en un futuro para entrenar el modelo.
- Si queremos crear el set de datos para un modelo que cuenta cartas o no.

Una vez sabemos los parámetros que recibe, comenzaríamos la función creando un mazo con el que jugar y posteriormente recorriendo el número de partidas (que serían jugadas por un NPC que toma decisiones aleatoriamente). Después, por cada partida el procedimiento será parecido al que vimos en el Apartado 3.2.1.2 ya que necesitamos simular las partidas pero con diferencias que veremos a continuación.

Tendremos un par de listas para poder guardar los datos y las etiquetas, la decisión será tomada al azar con el uso de la librería "***random***" indicándole que el banco de opciones será "h"==Pedir y "s"==Quedarse. Veamos esto en código:

Código 3.56 Generación decisión y aplicación de ésta.

```
num_hits = 0 # iniciamos a 0 el número de veces que se ha pedido carta para
    implementar así "Five-Card Charlie Rule"
while True:
    choice = random.choice(actions) # elegimos una acción al azar
    # si la acción es pedir carta
    if choice == "h":
        num_hits += 1
        # añadimos la carta
        players_hand.append(deck.deal_card())
        # comprobamos si al pedir carta el jugador se pasa de 21
        if hand_value(players_hand) > 21 or num_hits == 3 and hand_value(
            players_hand) < 21:
            break
    else:
        break
```

Implementado ya el turno del jugador, pasaríamos al turno del crupier que como se ha visto ya tendrá que pedir cartas hasta que tenga mínimo 17 de valoración. Una vez el turno de ambos jugadores terminan, haríamos las comprobaciones de quién es el ganador y a su vez añadiríamos los valores correspondientes a las listas de los datos y etiquetas (por temas de mantener privado partes de mi código, esta lógica usada no se mostrará).

Con las listas de datos y etiquetas rellenas, procederíamos a la escritura en los archivos de dichos valores mediante el Código 3.57:

Código 3.57 Escritura de los datos en los archivos.

```
try:
    # comprobamos que el tamaño de los datos y etiquetas son iguales
    if len(data) != len(tags):
        print("ERROR!!!!")
        print(data+"\n")
        print(tags+"\n")

    with open("blackjack/datasets/" + str(name) + ".data", "a") as data_file:
        for line in data:
            # escribimos los datos obtenidos de la partida jugada en el archivo
                de datos
            data_file.write(str(line) + "\n")

    with open("blackjack/datasets/" + str(name) + ".tags", "a") as tags_file:
        for tag in tags:
            # escribimos las etiquetas obtenidas de la partida jugada en el
                archivo de etiquetas
            tags_file.write(str(tag) + "\n")
except Exception as e:
    print(e) # comprobamos qué ha fallado
```

Para la escritura de los archivos nos ayudaremos del uso de "***open***", la aplicación del "***with***" en vez de hacer únicamente "***open***" es para no tener que preocuparme de cerrar el archivo después de escribir las líneas. Además, se añade el uso de la sentencia ***try*** para que en caso de que haya fallos en la escritura, imprimir el fallo.

Vista ya la función encargada de generar los sets de datos, pasemos a la función encargada de crear el modelo de IA que usará esos sets para su entrenamiento.

3.3.1.1 Creación del modelo IA

En esta función recibiremos como parámetros lo siguiente:

- Nombre del modelo con el que queremos que se guarde.
- El tamaño en porcentaje que le queremos dedicar del set de datos total a entrenar el modelo con respecto al que se usará para validar el modelo.
- Número de épocas con el que se quiere entrenar al modelo (esto hace referencia al número de vueltas que dará a todo el set de entrenamiento).
- Nombre de archivo donde tenemos guardados los sets de datos.
- Si cuenta cartas o no el modelo de IA a crear.

Lo primero a hacer en la función será procesar los datos para poder usarlos, para ello haremos uso de la función que vimos en el: Apartado 3.2.1.3. Después tendremos que hacer la división del set en la parte de entrenamiento y validación (visible en Código 3.58):

Código 3.58 División de los sets en entrenamiento y validación.

```
#tam es el tamaño total que tiene el set de datos
#size es el porcentaje que se quiere dedicar de dicho tamaño al set de
    entrenamiento
size_train = int(tam*size)

# creamos los sets de entrenamiento y validación para los datos y etiquetas
```

```
train_data = np.array(data_clean[1:size_train])
train_tags = np.array(tags_clean[1:size_train])
test_data = np.array(data_clean[size_train:])
test_tags = np.array(tags_clean[size_train:])
```

Con los sets de datos y etiquetas ya divididos en los referentes al entrenamiento y la validación, procederemos a la creación del modelo teniendo en cuenta si es contador de cartas o no.

El diseño de los modelos de Inteligencia Artificial han sido vistos ya previamente en la sección de Diseño (Apartado 3.1.5). Veremos cómo se ha implementado el diseño en el Código 3.59:

Código 3.59 Compilación y Entrenamiento.

```
optimizer = tf.keras.optimizers.Adam() # creamos el optimizer del modelo por
    separado
# compilamos el modelo indicando el optimizer=Adam, loss=CrossEntropy entre los
     datos y las etiquetas y como métrica de medida, la exactitud
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
# entrenamos el modelo
model.fit(train_data, train_tags, validation_data=(test_data, test_tags),
    epochs=num_epochs)
```

Creamos el optimizador por separado para poder obtener después su configuración y guardarla por separado, en este caso se usa el optimizador "***Adam***" por los siguiente motivos (si se quiere profundizar más en este optimizador, visitar [61]):

- Es un optimizador que combina el ***descenso del gradiente*** y el "***Root Mean Square Propagation***".
- La unión de ambos nos proporciona las siguientes ventajas en el optimizador:
  - Utilizando promedios de los diferentes momentos del gradiente permite una adaptación a la tasa de aprendizaje conforme más se entrena, reduciendo a su vez las posibles oscilaciones en los pesos que se apliquen a las neuronas.
  - Gracias al uso del "***Root Mean Square Propagation***", tiene una menor dependencia a la escala del gradiente de manera que ante variaciones abruptas de éste, se vea menos penalizado que el propio algoritmo del ***descenso del gradiente***.
  - Al tener una rápida adaptación a la tasa de aprendizaje provoca una mayor convergencia en un menor transcurso de tiempo lo que nos facilita que entrenamientos con un número alto de épocas conlleve menos tiempo.

Para entrenar el modelo simplemente utilizaremos los sets de entrenamiento y de validación y como número de épocas el parámetro que recibamos. En el archivo .tags descrito previamente (Apartado 3.1.3.1), se indican las acciones correctas (las incorrectas serán las opuestas) y mediante el entrenamiento acabará aprendiendo a tomar dichas decisiones (aplicando ajuste de pesos con cada época).

Una vez hemos visto cómo y por qué se han creado estos modelos, veamos como se guarda lo necesario del modelo para el uso futuro de los modelos. Primero hay que guardar el propio modelo en un fichero con terminación ".h5", una vez guardado el modelo es necesario guardar también el optimizador usado con terminación ".pkl". La terminación ".h5" es para indicar la extensión del archivo que es "HDF5", dicha extensión es muy útil para manejar grandes cantidades de datos de manera eficiente. La terminación ".pkl" proviene de haber guardado el optimizador haciendo uso de la librería "***pickle***", la cual ha sido explicada ya.

Veamos entonces la manera de guardar el modelo y el optimizador:

Código 3.60 Guardar modelo y optimizador.

```
# guardamos el modelo en la carpeta modelos
model.save("modelos/red_neuronal/" + str(model_name) + ".h5")
# guardamos la configuración del optimizer en un archivo de config
optimizer_config = optimizer.get_config()
with open('modelos/configuration_optimizers/' + str(model_name) + '.pkl', 'wb')
     as f:
    pickle.dump(optimizer_config, f)
```

Como se ve, obtenemos la configuración que ha obtenido el optimizador con "***get_config***" y para su guardado se hace uso de la librería "***pickle***", está librería es usada para poder serializar y distinguir de manera única esa configuración.

Por último veremos la función encargada de testear los modelos que se hayan creado, de manera que podamos ver su desempeño previo al uso de éstos en las partidas.

3.3.2 Testeo de los modelos

Para ejecutar esta función será necesario que se le pasen como parámetros los siguientes valores:

- El nombre del modelo de IA que se quiera testear.
- El número de partidas que se quieren simular para testear el modelo.
- Si el modelo a testear cuenta cartas o no.
- Un mazo de cartas para testear el modelo.

Una vez se tiene claro qué hay que pasarle como parámetros, iniciamos las estadísticas para poder valorar al modelo como son: victorias, derrotas y empates. Con las estadísticas iniciadas, tendremos que recuperar el modelo y su configuración para poder testearlo, para ello haremos uso del siguiente código:

Código 3.61 Cargar modelo y optimizador.

```
# cargamos el modelo de IA ya entrenado
model = tf.keras.models.load_model("modelos/red_neuronal/" + str(model_name) +
    ".h5")
with open('modelos/configuration_optimizers/' + str(model_name) + '.pkl', 'rb')
     as f:
    optimizer_config = pickle.load(f)
optimizer = tf.keras.optimizers.Adam.from_config(optimizer_config)
```

Como se comento anteriormente, al guardar su configuración usando la librería "***pickle***" será necesario también recuperarla usando la misma librería y para cargarla indicamos que el optimizador es "Adam" y usamos la propiedad "***from_config***". Con el modelo y su configurador recuperado ya, procederemos a empezar las simulaciones de las partidas.

Comenzamos recorriendo el número de partidas con un bucle for e iniciamos las manos del modelo y del crupier, con las manos iniciales hacemos la comprobación de si alguien ha obtenido blackjack y si nadie lo logra, entramos a la lógica de cada turno.

Primero, comprobamos si el modelo es contador de cartas o no para crear el set de datos que usará como predicción de la manera correcta. Posteriormente, el modelo predecirá y se aplicará la decisión tomada. Finalmente, se daría paso al turno del crupier qué simplemente pedirá carta hasta que tenga como valoración un mínimo de 17.

4 Pruebas

Como en todo proyecto en el que se desarrolla un programa con varios elementos, se hace imprescindible diseñar y realizar un conjunto de pruebas. En concreto para este proyecto, se han realizado una serie de pruebas unitarias, de integración y de funcionalidad que se detallarán a continuación.

Comenzaremos con las pruebas unitarias, en este tipo de pruebas no se verán las vistas de Django ya que estás funciones son usadas para tener una conexión con la página web y por tanto las pruebas que más aplicarían a estas funciones serían las de funcionalidad y las de integración.

4.1 Pruebas Unitarias

Aquí se buscan realizar pruebas para comprobar que las funciones no tenga fallos y funcionen de manera correcta, como podrían ser ver si reciben los parámetros que necesita y realiza la funcionalidad cómo debe. Para usar la misma técnica que en el apartado de Diseño e Implementación, iremos por los distintos archivos que posee el proyecto y dentro de cada archivo, las funciones que existan.

4.1.1 Archivo models.py

4.1.1.1 Modelo Deck

Para el modelo Deck tenemos distintas propiedades ya vistas en el (Apartado 3.2.1.1) y aquí veremos las pruebas usadas y las respuestas esperadas.

Creación del Deck

Para comprobar el funcionamiento correcto de esta propiedad, tendríamos que crear un objeto que fuese un deck y con imprimir las cartas del modelo Deck veríamos que se ha creado de manera correcta el mazo:

Código 4.1 Comprobar creación del deck.

```
deck = Deck()
print(deck.cards)
```

Negación del Deck

En esta propiedad si recordamos, lo que hacíamos era rellenar una lista con unos conforme iban saliendo cartas para que de esta manera el modelo de IA puediese contar las cartas. Por tanto, la manera de comprobar su funcionamiento es simple:

1. Indicamos un punto de parada en el momento de repartir las cartas.
2. Añadimos la variable "***final_list***" al inspector de variables del modo de depuración que ofrece Visual Studio Code.

3. Vamos sacando cartas del mazo poco a poco e iremos viendo como la variable "***final_list***" va llenándose de unos.

Con esos simples pasos ya tendríamos realizada la comprobación con un resultado exitoso pero puede surgir la siguiente duda: ¿Puede darse el caso de que se rellene un hueco de una carta, por ejemplo 7-C cuando realmente ha salido el 7-D?¿No afecta de manera negativa a la función?

La respuesta a la primera pregunta es Sí, sí puede que se rellene un hueco de una carta que no haya salido, pero esto no afecta negativamente a la función o al conteo de cartas ya que el único interés es saber qué cartas salen, siendo más concretos, el valor de dichas cartas. Por tanto, nos da igual que se rellene el hueco de un 7-D cuando salió un 7-C mientras que el hueco que se rellene sea el de un 7, el funcionamiento será el esperado, entonces la respuesta a la segunda pregunta sería No.

Robar una carta

Para la comprobación de esta función, sólo habría que llamarla varias veces y comprobar que los valores son completamente aleatorios dentro de la lista de posibilidades que tiene (que son 52 cartas pero las cuales están repetidas 6 veces cada una).

Chequear el tamaño del mazo

En este caso, esta función está pensada para crear un mazo nuevo cada vez que se vacía. Por tanto, para comprobar su funcionamiento, se ha hecho de la siguiente maneras:

1. Colocar un punto de parada cuando se pida carta.
2. Dentro del modo de depuración, en el inspector de variables, añadir el mazo y el tamaño de éste (con el uso de *len(deck.cards)*).
3. Simular partidas, para comprobar que se va vaciando el deck y cuando quede poco, prestar mucha atención a que se llama a la función y si el deck se vuelve a llenar.

Funciones de serialización y deserialización

Para verificar estas funciones, habría que entrar en terreno de la página web ya, porque su uso está pensado para mantener un mismo deck a través de la navegación por la página web. De todas maneras, se puede comprobar por separado:

- Para la función encargada de serializar, hacemos una llamada a la función en el deck e imprimir dicha llamada para ver el resultado que obtenemos.
- Para la función encargada de deserializar, asociamos a una variable la llamada a esa función a la que le pasamos por parámetros el deck serializado. Al imprimir dicha variable, obtendríamos un nuevo objeto deck.

4.1.1.2 Modelo Player

Al igual que con el modelo Deck, veremos las pruebas realizadas a las distintas propiedades del modelo que han sido ya descritas y explicadas en el punto anterior.

Convertir a mayúsculas el tipo

La prueba a esta función es muy simple ya que sólo hace falta crear un jugador de un tipo concreto e imprimir la llamada a esa función para observar que lo que devuelve es dicho tipo en mayúsculas.

Para esta función, lo que haremos será otorgarle diferentes valores a las manos del jugador y llamarla para cada una, simplemente tendremos que ver si devuelve True cuando tiene un blackjack el jugador (AS + 10/J/Q/K), esto estará en la Código 4.2:

Código 4.2 Comprobar blackjack jugador.

```
player.hand = ['A-C', 'K-D'] => player.hand_value = 21 => player.
    check_blackjack = True
player.hand = ['9-D', '3-C'] => player.hand_value = 12 => player.
    check_blackjack = False
```

Función que calcula los porcentajes

Aquí, para verificar su funcionamiento tendríamos que añadir un número de victorias, derrotas o empates y comprobar cuál sería el resultado correcto con la ayuda de una calculadora o para simplificar, hacerlo que sumen 10 para que los porcentajes sean directos.

Funciones de serialización y deserialización

Para la comprobación de estas funciones, seguiríamos los mismos pasos que los descritos en el modelo Deck (que por ahorrar lectura, no se volverán a explicar).

4.1.2 Archivo process_data.py

En este archivo, si recordamos lo explicado en el Apartado 3.2.1.1, se realizaba una pequeña transformación a los sets de datos que generábamos para convertirlos todos los valores a enteros de valor 1 o 0. Por tanto, la manera para comprobar su funcionamiento sería llamar a la función pasándole sets de datos y ver si existe algún cambio. Como se muestra en la Figura 4.1, donde tendríamos el set de datos previo a la llamada a la función y cómo quedan después de ser procesados:

```
Archivo de datos: ['[18, 0, 5]\n', '[13, 0, 10]\n', '[9, 0, 2]\n', '[20, 0, 10]\n', '[8, 0, 10]\n']
Archivo de etiquetas: ['s\n', 's\n', 'h\n', 's\n', 's\n']
Archivo de datos procesado: [[18, 0, 5], [13, 0, 10], [9, 0, 2], [20, 0, 10], [8, 0, 10]]
Archivo de etiquetas procesado: [0.0, 0.0, 1.0, 0.0, 0.0]
Tamaño del set de datos: 5
```

Figura 4.1 Sets de datos previo al procesamiento (ARRIBA) y post procesamiento (ABAJO).

4.1.3 Archivo utils.py

En este archivo, cómo hemos explicado ya, están una serie de funciones que son usadas generalmente en el proyecto. Entonces, iremos viendo qué pruebas unitarias hemos estado realizando para asegurarnos del funcionamiento.

4.1.3.1 Función hand_value

Para comprobar el funcionamiento, lo que haremos será pasarles distintas manos y comprobar que nos devuelve el valor correcto, así como también toma en consideración el tener un AS y le aplica un valor al AS dependiendo de la situación en la que nos encontremos. Las pruebas realizadas se pueden ver en el Código 4.3:

Código 4.3 Comprobar valor mano jugador.

```
player.hand = ['7-H', 'Q-S'] => player.hand_value = 17
player.hand = ['3-D', '5-C', '9-H', 'K-S'] => player.hand_value = 27
player.hand = ['A-S', '4-H', '6-D'] => player.hand_value = 21
player.hand = ['2-C', 'J-C', 'K-S'] => player.hand_value = 22
```

4.1.3.2 Función card_value

Esta función es igual a la anterior pero pensada para una única carta, la del crupier al inicio de la ronda, por tanto no veo necesario volver a explicar qué pruebas han sido necesarias hacer para comprobar su funcionamiento. La única diferencia en este caso es que cuando la carta visible es un AS, el valor devuelto será 11.

4.1.3.3 Función has_usable_ace

Para comprobar esta función lo que se hará es pasarle distintas manos que tengan AS y ver si devuelve que es usable o no el AS, que un AS sea usable lo tomamos como que pueda valer 11 y no 1. En el Código 4.4, se verá las pruebas usadas:

Código 4.4 Comprobar AS usable como 11.

```
player.hand = ['7-H', 'Q-S', 'A-C'] => player.has_usable_ace = 0 # Equivale a
    False
player.hand = ['3-D', '5-C', 'A-S'] => player.has_usable_ace = 1 # Equivale a
    True
player.hand = ['A-S', '4-H', '6-D'] => player.has_usable_ace = 1 # Equivale a
    True
player.hand = ['A-C', 'J-C', 'K-S'] => player.has_usable_ace = 0 # Equivale a
    False
```

4.1.3.4 Función get_percentage

Aquí, la idea de esta función es devolver el porcentaje de una estadística, por tanto las pruebas a realizar será pasarle por argumento dos diferentes números, el primer número será las partidas de esa estadística y el otro número será el total de partidas jugadas, cómo es lógico el primer número siempre será menor o igual al segundo. Entonces para verificar su funcionamiento, se pasan diferentes números y se compara el valor devuelto por la función con los cálculos hechos con una calculadora y ver si coinciden.

4.1.3.5 Función clean_hand

Para verificar que funciona correctamente, le pasaremos jugadores y el crupier cuyas manos tienen cartas ya añadidas y tendremos que ver si después de llamar a la función siguen teniendo cartas en la mano o no ya que el trabajo de esta función es eliminar las cartas de la mano de cada jugador y el crupier. En el Código 4.5 se muestran las pruebas que he realizado:

Código 4.5 Comprobar función clean_hand.

```
# Previo a la llamada a clean_hand
player1.hand = ['7-H', 'Q-S', 'A-C']
player2.hand = ['3-D', '5-C', 'A-S']
player3.hand = ['A-S', '4-H', '6-D']
player4.hand = ['A-C', 'J-C', 'K-S']
# Llamada a la función clean_hand
players = [player1, player2, player3, player4] => clean_hand(players)
# Posterior a la llamada a la función clean_hand
player1.hand = []
player2.hand = []
player3.hand = []
player4.hand = []
```

4.1.3.6 Función find_action_in_table

En esta función que trata simplemente de buscar la acción que habría que tomar buscando en una tabla, la manera de verificar que funciona es pasarle distintos casos de ejemplo, los cuáles estaban formados por: la mano del jugador que llama la función, la carta visible del crupier y el tipo de tabla donde se quería buscar (que teníamos 3, un AS en mano, cartas iguales o ninguna de las anteriores). Por tanto, pasamos diferentes ejemplos y comprobamos con la tabla en la que debe de buscar al lado si el valor que nos devuelve es el correcto o no.

La tabla donde busca la acción la función es la que nos encontramos en la Figura 2.5 pero con algunos cambios (Apartado 3.1.3.1), por tanto para evitar posibles conflictos, mostraré pruebas realizadas cuyo resultado sea el mismo que deberíamos obtener en dicha tabla, los resultados aparecerán en el Código 4.6:

Código 4.6 Comprobar función find_action_in_table.

```
# PRUEBA 1 - TABLA VALORES IGUALES:
player.hand = ['7-H', '7-S'], dealer.hand[0] = ['8-D']
# según la fig02-05, debe pedir carta
find_action_in_table(player.hand, dealer.hand[0], 'equals') => action = 'h'
# PRUEBA 2 - TABLA VALORES DISTINTOS:
player.hand = ['9-D', 'K-C'], dealer.hand[0] = ['3-D']
# según la fig02-05, debe quedarse
find_action_in_table(player.hand, dealer.hand[0], 'normal') => action = 's'
# PRUEBA 3 - TABLA PARA MANOS CON AS
player.hand = ['A-S', '4-H'], dealer.hand[0] = ['K-S']
# según la fig02-05, debe pedir carta
find_action_in_table(player.hand, dealer.hand[0], 'as') => action = 'h'
```

4.1.3.7 Función gen_next_action

Para las comprobaciones de esta función será necesario volver a entrar en el modo de de depuración:

- Indicaremos un punto de parada antes de que se ejecute esta acción.
- Añadimos al inspector de variables las siguientes:
  - La probabilidad de pedir carta, que es llamada "***hit_prob***".
  - La probabilidad de quedarse, que es llamada "***stay_prob***".
- Después habrá que esperar a que el valor aleatorio que se genera sea mayor que el valor de ε que hemos indicado en el código y cuando se dé dicha ocasión, tendremos que observar qué valor es mayor, si la probabilidad de pedir carta o la de quedarse ya que la opción elegida será de mayor probabilidad.

4.1.3.8 Función set_Q

Aquí, poder verificar esta función se vuelve un poco más complicado debido a qué la finalidad de esta función es actualizar valores en la Tabla Q mediante una ecuación, vista en el Apartado 3.2.1.2. Por tanto, para hacer las comprobaciones correspondientes sería necesario lo siguiente:

- Indicar un punto de parada antes de actualizar el valor de la Tabla Q.
- Añadir al inspector de variables las siguientes:
  - ***Rewards***, para llevar un conteo de las recompensas.
  - ***discountRate***, para saber la tasa de descuento total que se debe aplicar a las recompensas.
  - La Tabla Q, para comprobar si se realiza correctamente el cálculo.
- Una vez se tienen añadidas las variables, vamos ejecutando paso a paso las líneas y comprobando si los resultados coinciden con nuestros cálculos.

4.1.3.9 Función get_dataset_files

Para comprobar el funcionamiento correcto de esta función será necesario simplemente ejecutarla y ver si lo devuelto es el contenido de la carpeta de la que queremos saber sus archivos. Además, cabe aclarar que las pruebas a esta función son exactamente iguales que las declaradas después, que serían ***get_model_files*** y ***get_q_files***. En la Figura 4.2 encontraremos las pruebas a las 3 funciones.

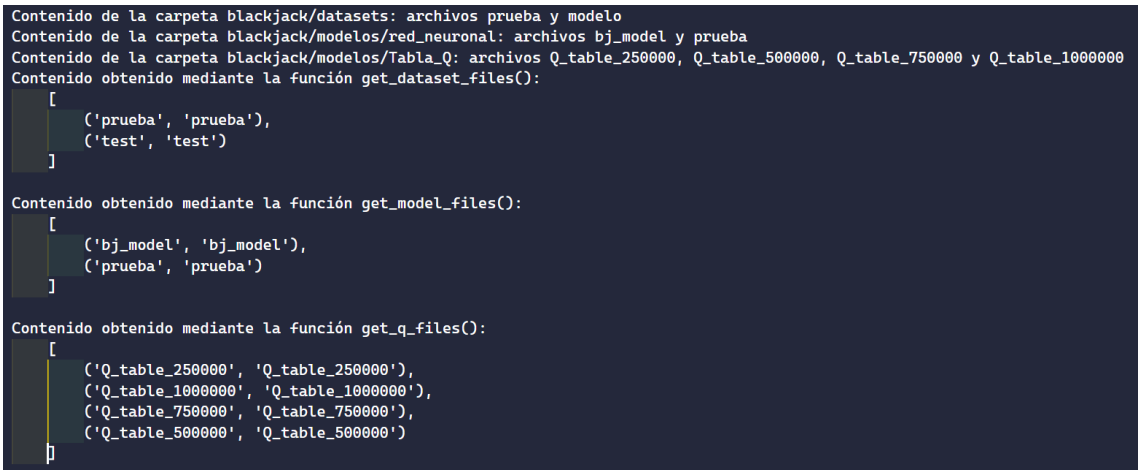


Figura 4.2 Comprobación funciones para obtener los archivos de los datos, modelos y Tablas Q.

Finalizada las explicaciones de las pruebas unitarias para las funciones que encontramos en el archivo ***utils.py***. A continuación, se explicará las que se ubican en el archivo ***views.py***.

4.1.4 Archivo views.py

4.1.4.1 Función hi_lo

Esta función está pensada para llevar la cuenta del mazo y por tanto, para probar su funcionamiento será necesario llamar a esta función pasándole un número de cartas al azar y la cuenta que estaríamos llevando. Un ejemplo de las pruebas realizas serían las que nos encontramos en el Código 4.7:

Código 4.7 Comprobar algoritmo Hi-Lo.

```
# PRUEBA 1 - COMENZANDO LA CUENTA EN 0
cards = ['7-H', '7-S', '9-H', 'K-S', 'A-S', '4-H', '8-S', 'A-H']
# resultado esperado empezando la cuenta en 0 => 0+0+0-1-1+1-1 = -2
count = 0 => hi_lo(cards, count) => count = -2
# PRUEBA 2 - MANTENER LA MISMA CUENTA Y APLICAR OTRO SET DE CARTAS
cards = ['4-H', '4-S', '10-H', 'K-S', 'A-S', '4-H', '2-S', '5-H']
# resultado esperado empezando la cuenta en -2 => -2 + (+1+1-1-1-1+1+1+1) = 0
count = -2 => hi_lo(cards, count) => count = 0
```

4.1.4.2 Función para crear el set de datos

Aquí, para comprobar su funcionamiento lo que haremos será ejecutarla pasándole distintos valores que requiere la función y verificar la creación del set de datos. Además, la escritura en los archivos está hecha dentro de un ***try - except***, por tanto se tiene en constancia si es posible algún fallo, la detección inmediata de él.

4.1.4.3 Función para crear el modelo IA

En esta función, tendremos un parámetro más conflictivo que en la llamada para la creación del set de datos y es el parámetro del tamaño que se le quiere dedicar del set de datos a entrenamiento, ese valor debe ser tomado como un porcentaje. Por tanto, será necesario una previa comprobación de que dicho tamaño que se le pasa por parámetro se encuentre entre 0 y 100. Además, puede darse el caso que por una equivocación se pase un número de épocas para entrenar el modelo negativo, lo que se hace en este caso es informar de ello y avisar de que ese número se va a convertir a positivo y se ejecutaría correctamente la función.

4.1.4.4 Función para testear el modelo IA

El parámetro clave de esta función es el nombre del modelo y para esto se hacen comprobaciones de intentar cargar el modelo y en caso de fallo se devuelve una cadena de texto informando de la imposibilidad de cargar el modelo. Por tanto, las pruebas que se han hecho para esta función son:

- Hacer una ejecución de la función pasando parámetros correctos y ver el resultado.
- Hacer una ejecución de la función pasando como nombre de modelo un valor completamente incorrecto y ver que salta el error y no se ejecuta.

Terminadas la explicación de las pruebas unitarias, comenzaremos con las pruebas de integración.

4.2 Pruebas Funcionales

Estas pruebas se realizarán para comprobar que la página web funciona correctamente, de esta manera verificaremos que los elementos que tendrán interacción con el usuario están trabajando correctamente. A continuación, nombraré una serie de acciones que he realizado:

1. Arrancar el servidor y comprobar que la página web carga correctamente.
2. Pinchar en el botón para la creación de un set de datos y ver que redirige a la página deseada. Dentro de esta parte de la página:
    - Ver que funciona correctamente todos los botones para ocultar y mostrar los distintos formularios.
    - Comprobar que el botón de volver al menú principal lo hace correctamente.
3. Pinchar en el botón para jugar una partida y ver que redirige también a la página que debe. Dentro de esta parte, tendremos:
    - Comprobar que la ocultación de los archivos a selección para la creación de los jugadores funcionan cuando no se eligen los tipos correctos (si elijo jugador IA se muestra archivos de modelos de IA y no de Tabla Q).
    - Ver que se crean los jugadores y que se muestran en la tabla correctamente.
    - Verificar que los botones de edición y borrado de jugadores, muestran un modal con un formulario y uno de confirmación, respectivamente.
    - Asegurarnos de que los botones para comenzar/continuar las partidas funcionan correctamente.
4. Dentro de la mesa para jugar la partida, comprobar que todos los botones de pedir carta/-quedarse e iniciar los turnos funcionan de manera óptima. Además, el botón para mostrar las estadísticas realiza su función correctamente y que los botones para iniciar una ronda nueva y el desplegable con el resto de funciones también hacen su función de manera correcta.

Finalizada la explicación de las pruebas funcionales para comprobar que la página web está actuando de la manera esperada. Terminaremos este capítulo de pruebas explicando las pruebas de integración para ver que el código se ha integrado correctamente con la página web y está funcionando todo de acuerdo a lo deseado.

4.3 Pruebas de Integración

Aquí se buscan realizar pruebas para comprobar que la página web esté integrada de manera óptima con el código para que podamos obtener los mismos resultados que hemos visto en las pruebas unitarias con las funciones. Por tanto, iremos viendo las pruebas realizadas en las vistas para lograr dicho objetivo.

4.3.1 Vista dataset

Para las comprobaciones en esta función, hay que saber que es la encargada de recibir la información del formulario de la creación de los sets de datos, entonces sabiendo esto:

- Probar directamente a crear set de datos para ver que salta error debido a la obligatoriedad de rellenar los campos del formulario. Esto se repetirá en todas las vistas que tengan que ver con formularios, por tanto se omitirán en las siguientes para evitar redundancia.
- Rellenar correctamente los campos y verificar que el set de datos se ha creado correctamente (tanto para el modelo no contador de cartas como el contador). La verificación se realiza viendo si se ha creado el archivo que almacena el set de datos o si salta la "alerta" de qué ha sido creado correctamente.
- Probar a introducir un número menor de 0 para el campo de número de partidas y ver que no deja enviar el formulario. Tanto si se intenta rellenar directamente por el teclado como al usar las flechas del campo del formulario.

Además, hace falta comprobar que al enviar correctamente todos los datos del formulario, se reciben y se interpretan óptimamente para la ejecución de la función (para ello nos ayudaremos del modo de depuración). Con el envío, se tiene que verificar que nos mantenemos en la misma página y se muestra la alerta de la creación exitosa.

4.3.2 Vista model

Aquí se harán las pruebas para comprobar que se puede crear el modelo desde la página web sin fallos. Para ello, tendremos que probar que los campos del formulario no presentan fallos, entonces tendremos los siguientes pasos que servirán cómo pruebas para verificar el funcionamiento correcto de la integración de la página con el código:

- Rellenaremos correctamente todos los valores y comprobaremos que se crea correctamente los modelos, es decir, se ha creado un archivo con el nombre del modelo y además salta la "alerta" de que se ha creado correctamente.
- Intentamos indicar un número que no se encuentre entre 25 y 95 para el porcentaje a dedicar del set de datos para entrenamiento. Esto dará en un error debido a las limitaciones indicadas en el campo HTML.
- Intentaremos indicar un número menor a 5 para el número de épocas para entrenar el modelo e igual que con el campo de porcentaje, nos dará un error debido a los límites indicados en el campo.
- Comprobar que todos los archivos que se muestran en el desplegable de los sets de datos, son todos los que están creados y ubicados en la carpeta.

Al crear el modelo, al igual que cuando se crea el set de datos, debemos mantenernos en la misma página y debe de aparecer la alerta de creación exitosa. Además, se comprobará con el uso del modo de depuración que se reciben e interpretan correctamente los valores introducidos en el formulario para la creación del modelo.

4.3.3 Vista test_model

Para comprobar la integración de la página web con esta vista, tendremos que comprobar si el formulario envía los datos correctamente. Para ello, haremos las comprobaciones básicas de intentar enviar sin rellenar los valores y también intentar enviarlo con un número negativo de partidas, en ambos casos dará error el envío. Además, tenemos que ver que el desplegable con los nombres de los modelos de IA se muestran correctamente y todos los creados.

Después del envío correcto, verificaremos que recibimos los datos y son interpretados de manera óptima (para ello nos ayudaremos del modo de depuración). También, una vez enviado los datos del formulario, hará falta verificar que salta un modal con un imagen de carga para que al terminar el testeo del modelo se cambie por una tabla con las estadísticas de la partida.

Visto el apartado de los sets de datos y modelos de Inteligencia Artificial, pasaremos a ver las pruebas realizadas al apartado de jugar, comenzando con la creación/edición/borrado de los jugadores.

4.3.4 Vista get_info_players

En las comprobaciones de esta función, la principal es ver que aparecen el formulario para crear jugadores y la tabla con el jugador USER ya creado.

Verificado esto, empezaremos con la comprobación del formulario dónde la única manera de crear jugadores es indicar una opción válida en el desplegable del tipo de jugador. Además, también será necesario comprobar que al indicar tipo de jugador "Modelo de Inteligencia Artificial" aparecerá un desplegable con los modelos ya creados y si por el contrario se elige "Algoritmo Q-Learning", dicho desplegable se cambiará y mostrará los archivos de las Tablas Q ya creadas y en caso de elegir el tipo "NPC", desaparece dicho desplegable. Entonces, comprobado esto, comenzamos con la creación de distintos jugadores y ver que aparecen en la tabla (dentro de esto será necesario que con el modo de depuración se obtienen los valores de manera correcta para poder crear los jugadores). Además, con cada creación de jugador, se debe mantener en la misma página web.

A parte del formulario, existen más funcionalidades en esta parte de la página web que han sido integradas cómo la posibilidad de editar/eliminar a un jugador, esto se comprobará con lo siguiente:

- Pulsamos el botón para editar un jugador y debe aparecer un modal con un formulario exactamente igual que el de creación para poder editar el jugador. Por tanto, al ser un formulario, las pruebas de su comprobación ya han sido vistas.
- Pulsamos el botón para eliminar un jugador y debe aparecer un modal con un mensaje de confirmación para evitar posibles errores. Además, que al pulsar Sí en la ventana se elimine el jugador y si se pulsa No, no se elimine.

Por último, hará falta verificar la funcionalidad de los dos botones para empezar la partida. Al pulsar cualquier botón, tendrá que aparecer una ventana de confirmación para asegurarnos de que la elección ha sido correcta.

4.3.5 Jugar la partida

Para iniciar las comprobaciones en estas vistas será necesario saber si el botón para ver las estadísticas generales funciona correctamente, ya que al pulsarlo deberá de saltar un modal con una tabla

mostrando las estadísticas de cada jugador. El funcionamiento correcto de este botón nos permitirá verificar si la decisión tomada de empezar una nueva partida o continuarla es ejecutada a la perfección ya que si se decide empezar una nueva partida, todas las estadísticas deben de estar iniciadas a 0 y en el caso de continuarla, las estadísticas se mantendrán las antiguas de los jugadores.

También, con una mera observación en la pantalla sabremos si la función ha sido integrada de manera correcta en la página web porque veremos a los distintos jugadores repartidos por la mesa con sus dos cartas repartidas (incluyendo una visible del crupier y otra oculta).

Después de las primeras comprobaciones, hace falta ver que se puede jugar sin problemas la partida. Por ello, se harán las siguientes comprobaciones:

- Los botones de inicio/final de turno funcionan correctamente, para ello será necesario comprobar que recibimos el identificador del jugador cuyo turno comienza para iniciar su lógica de juego. Después de pulsar en el botón, observar que el botón se oculta y cambia por otro que indica el turno del siguiente jugador y al pulsarlo lo inicia.
- Si el turno es del usuario, comprobar que funcionan los botones de pedir carta y quedarse, es decir, al pedir carta se añade una carta a la mano del jugador y esto se ve reflejado en la pantalla y al quedarse se pasa al siguiente turno. Además, también hará falta verificar que si el usuario se pasa de 21 o alcanza 21, sus botones se ocultan para mostrar uno indicando el inicio del siguiente jugador. Al igual que con el resto de jugadores, la integración con el código se realiza comprobando si el identificador pertenece al usuario y que acción ha elegido tomar.
- Otra comprobación necesaria es ver que si un jugador ha logrado blackjack, nunca será indicado en el orden de los turnos ya que no es necesario que juegue.

Con la finalización del turno de todos los jugadores incluido el crupier, se tiene que verificar que se muestra un mensaje de texto con los resultados de la ronda actual. Además, también deberá de aparecer un botón en la parte baja mostrando la opción de iniciar otra ronda (al ser pulsado, debe de comenzar otra ronda, su funcionamiento es similar a la opción de continuar partida que se ha comentado antes en la creación de jugadores) y al lado de dicho botón tendrá que aparecer una flecha que al ser pulsada desplegará un pequeño menú con tres opciones que deberán de funcionar correctamente:

1. Iniciar una nueva partida, esta opción al ser pulsada nos deberá de recargar la página y actualizar las estadísticas de todos los jugadores a 0.
2. Volver al menú de creación de jugadores, este al ser pulsado nos debe de volver a la pantalla donde creamos los jugadores.
3. Volver al menú principal, aquí cuando se pulse nos debe de enviar a la página inicial, donde mostrábamos las distintas opciones que proporcionaba este proyecto.

Explicadas las pruebas de integración, pasaremos a ver las pruebas y comprobaciones realizadas al modelo de Inteligencia Artificial para ver qué resultados he obtenido y si funciona correctamente.

4.4 Pruebas al Modelo de Inteligencia Artificial

Las pruebas principales que se podrían hacer para saber si el modelo funciona correctamente o no, sería comprobar si este sufre de sobreajuste (para recordar que era el sobreajuste, visitar el Apartado 2.2.5.2).

La manera que he usado para comprobar si mi modelo sufría de sobreajuste ha sido el uso de la herramienta TensorBoard, entrenando mi modelo durante 200 épocas, los resultados brindados han sido los siguientes.

4.4.1 Modelo Contador de Cartas

Veamos las gráficas de la función de pérdidas y precisión del entrenamiento del modelo:

- Gráfica de pérdidas => Figura 4.3.
- Gráfica de precisión => Figura 4.4.

En todas las gráficas que se muestren, la curva naranja hace referencia a los resultados del entrenamiento y la curva azul a los resultados de los valores para validación del modelo, es decir, comprobar cómo de bien predice. En esta figura se puede observar que no se produce ningún sobreajuste,

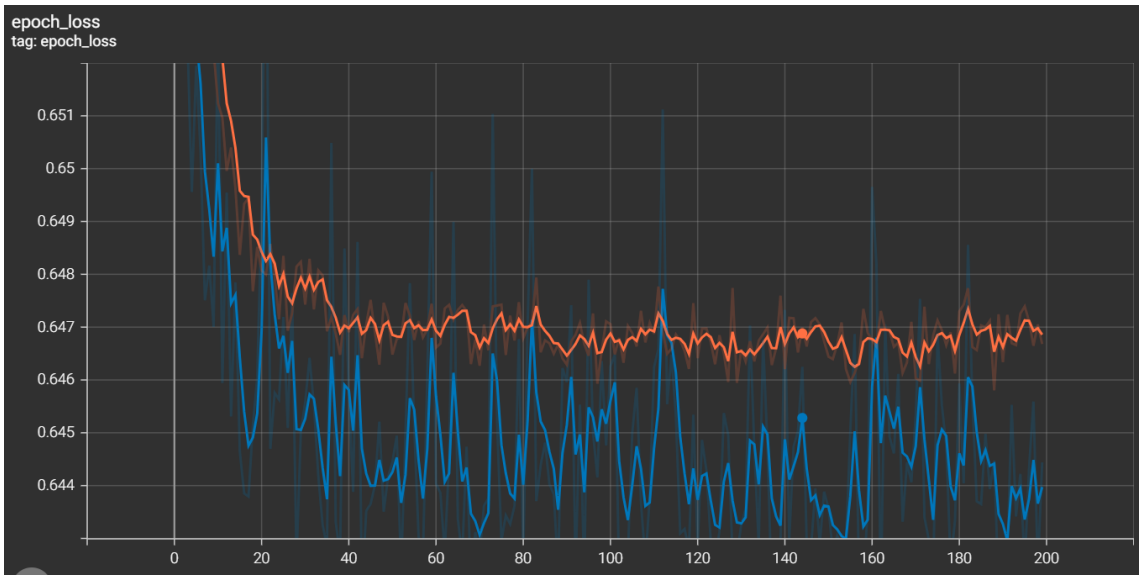


Figura 4.3 Gráfica de la función de pérdidas del modelo contador de cartas.

ya que en ningún momento se produce una bajada significativa en la curva del entrenamiento y a su vez una subida en la curva de validación. Además, se ve que ambas curvas llegan a mantenerse en unos valores de entre 0,644 y 0,647.

En la gráfica de precisión del modelo, podríamos encontrarnos algo parecido, no se ve un sobreajuste a primera vista pero si se nota bastantes fluctuaciones en la curva que representa la validación pero al final se suele mantener en un mismo rango de valores, por lo que podríamos considerarlo estable, estos rangos de valores son alrededor del $0{,}62 + -0{,}02$. En el caso de la curva de entrenamiento, se observa que aprende de manera correcta y estable.

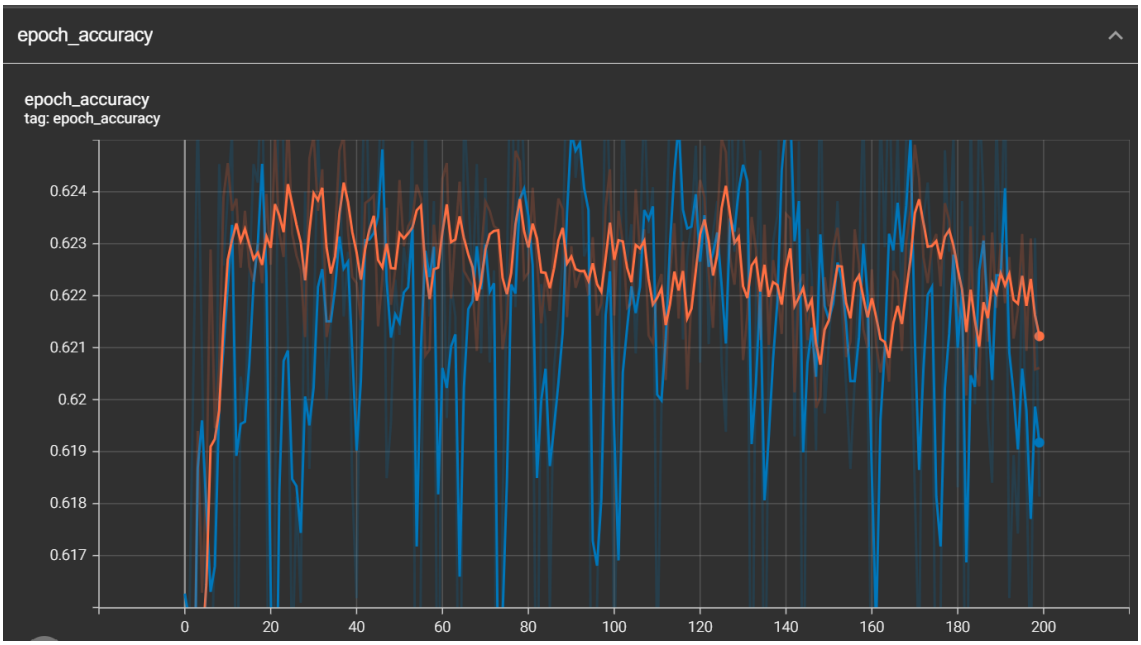


Figura 4.4 Gráfica de la precisión del modelo contador de cartas.

Viendo que en el entrenamiento los resultados arrojados son favorables, faltarían hacer comprobaciones de sus predicciones. Al considerarse un juego de azar de casino, su porcentaje de victorias rara vez podría alcanzar un 50 %, entonces se considera exitoso si se mantiene superior a un 35 % de victorias (cuantas más partidas se simulen, más probable será superar esas cotas). Los resultados se podrán comprobar en la Figura 4.7.

4.4.2 Modelo No Contador de Cartas

Al igual que con el modelo contador de cartas, se han sacado las gráficas referentes a la función de pérdidas y a la precisión para el entrenamiento de este modelo:

- Gráfica de pérdidas => Figura 4.5.
- Gráfica de precisión => Figura 4.6.

Aquí se mantiene la diferenciación en las curvas: curva naranja representa los datos de entrenamiento y curva azul representa los datos para validar el modelo.

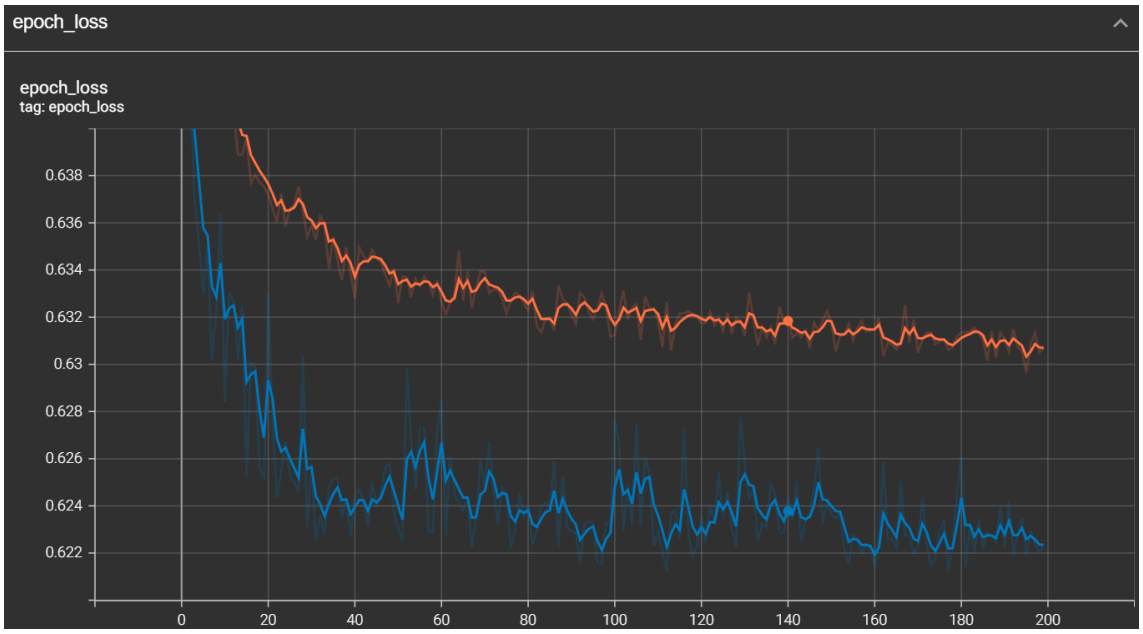


Figura 4.5 Gráfica de la función de pérdidas del modelo no contador de cartas.

En esta gráfica de la función de pérdidas es todavía más visible la ausencia de sobreajuste, lo cuál es algo bueno. Además, tienen un ritmo de aprendizaje bastante bueno, alcanzando una estabilidad alrededor de valores de 0,63 para el entrenamiento y de 0,622 para la validación.

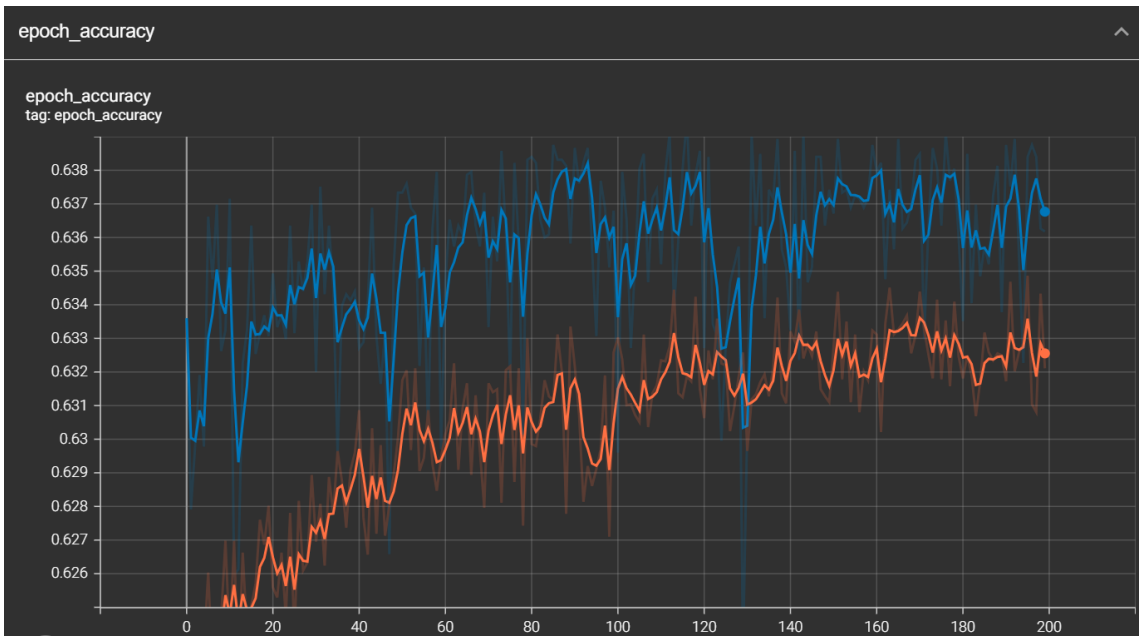
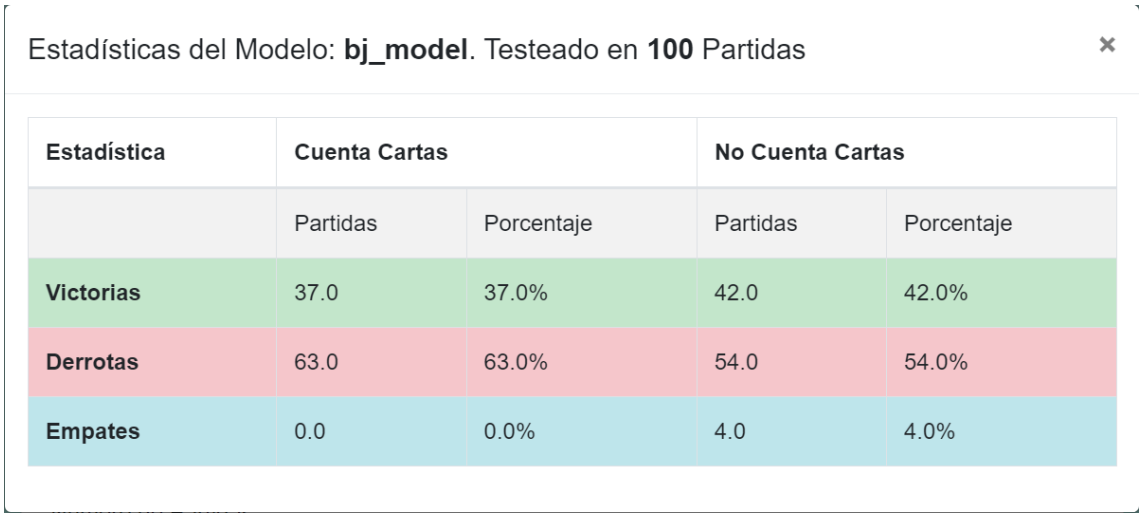


Figura 4.6 Gráfica de la precisión del modelo no contador de cartas.

Para la gráfica de la precisión, se puede volver a notar la ausencia de sobreajuste, lo que indica que el modelo está aprendiendo correctamente. Además, también se cumple lo visto con las anteriores gráficas y es que alcanzan una estabilidad, en esta ocasión, alrededor de los valores de 0,632 para el entrenamiento y 0,637 para la validación.

4.4.3 Resultados de testeos de los modelos

En este apartado, veremos si los modelos predicen correctamente. Para ello, se pondrán a simular 100 partidas en las que jugarán modelo vs crupier. En la Figura 3.7 podremos ver los resultados. Como podemos comprobar, ambos modelos obtienes un porcentaje de victorias superior al 35 % y


Estadísticas del Modelo: **bj_model**. Testeado en **100** Partidas

Estadística	Cuenta Cartas		No Cuenta Cartas	
	Partidas	Porcentaje	Partidas	Porcentaje
Victorias	37.0	37.0%	42.0	42.0%
Derrotas	63.0	63.0%	54.0	54.0%
Empates	0.0	0.0%	4.0	4.0%

Figura 4.7 Resultados de los testeos de los modelos.

en el caso del no contador de cartas incluso superior al 40 %. Podemos sacar en claro entonces que ambos modelos están funcionando de la manera correcta con estas pruebas mostradas.

Finalizada la parte de la explicación de todas las pruebas que se han aplicado o se pueden aplicar para verificar tanto la funcionalidad correcta de todas las funciones, como el correcto despliegue de la página web y su posterior integración con todas las funciones, daremos comienzo al siguiente capítulo del proyecto y es explicar las dificultades que me he encontrado en la realización de éste.

5 Dificultades Encontradas

En este capítulo explicaré las dificultades que me he ido encontrando durante la realización de este proyecto. Además, en caso de haberlas logrado resolver, explicar como lo logré y que herramientas utilicé como ayuda para la resolución.

Dividiré este apartado en las 4 grandes dificultades que tuve:

1. Decidir que tipo de datos serían los que añadiría al set de datos para la creación de un modelo de Inteligencia Artificial lo más certero posible.
2. En el modal donde muestro las estadísticas del testeo del modelo de Inteligencia Artificial, la lógica para poder mostrar una imagen de carga para mantener al usuario informado de lo que sucede y que al terminar el testeo, ocultarlo y mostrar los resultados.
3. Actualización por partes en la pantalla de jugar la partida.
4. Ocultar los botones en la pantalla de jugar la partida, que al ser pulsados se ocultasen y apareciesen otros con la nueva información

Vistas las dificultades y para mantener en la explicación el mismo orden en el que fueron surgiéndome estos problemas, comenzaré con el primero, decidir los datos para el set de datos del modelo.

5.1 Decisión del set de datos

Aquí el problema principal residía en no saber cómo registrar el historial de cartas que iban saliendo de una manera que fuera consistente con otros valores ya definidos e inamovibles: el valor de la mano del jugador, el valor de la carta visible del crupier y si el jugador tendría un AS que pudiera usar como 11. Afortunadamente, el formato de esos valores ya estaba definido: los valores de las manos debían ser enteros y la indicación de tener un AS usable también se podía formatear como un entero. Sin embargo, encontrar la manera adecuada de representar todas las cartas resultaba complicado.

Comencé a investigar posibles formas de transmitir esa información. Inicialmente, pensé en pasar una lista con todas las cartas, pero esto presentaba el problema del formato, ya que para entrenar el modelo usando tablas de la librería numpy, todos los valores del tablas debían ser del mismo tipo. Por lo tanto, necesitaba encontrar una solución diferente. Fue entonces cuando se me ocurrió la idea de convertir esa lista de cartas en una lista de enteros, donde el valor 1 indicaría que esa carta había salido y el valor 0 lo contrario.

Fue entonces que me encontré con un nuevo problema: no sabía cómo implementar mi idea en código de manera que funcionara correctamente. Por lo tanto, decidí buscar ayuda en las páginas

más reconocidas sobre programación, esperando que alguien hubiera tenido el mismo problema. Sin embargo, no tuve éxito en encontrar una solución adecuada.

La solución vino cuando, explorando en GitHub, descubrí que un usuario [8] había creado un conjunto de datos específicamente diseñado para que un modelo de Inteligencia Artificial pudiera contar cartas. Este hallazgo resultó ser crucial. Analicé su código y comprendí que con algunas modificaciones podría adaptarse a mis necesidades.

Tomé su código como base y realicé los ajustes necesarios para que se ajustara a la cantidad de cartas que yo estaba manejando. Gracias a esta solución, el problema quedó resuelto, como se puede apreciar en la explicación detallada de la función, en el Apartado 3.2.1.1.

5.2 Modal de carga en el testeo del modelo

Este problema resultó bastante curioso. Comenzó cuando quise añadir una imagen de carga para que apareciera mientras el modelo era testeado, con el fin de mantener al usuario informado durante el tiempo que se demoraba en recibir y mostrar las estadísticas.

Primero, logré mostrar el modal con la imagen de carga, y justo cuando recibía las estadísticas, lo cambiaba por la tabla con los resultados, utilizando funciones de JavaScript. Sin embargo, me encontré con una dificultad mayor: mantener ese funcionamiento constante cada vez que se ejecutara el testeo del modelo. La plantilla mantenía los valores anteriores y, al volver a llamar la función, aparecía directamente la tabla con los valores previos antes de actualizarse con los nuevos.

Este comportamiento no era el deseado. Necesitaba una manera de controlar el modal en tiempo real para que funcionara correctamente en cada ejecución. Como la función principal de mostrar la imagen de carga ya funcionaba, solo requería algunos ajustes, por tanto para resolver esto, recurrí a mis apuntes de la asignatura de FAST dónde se veía cómo utilizar AJAX para resolver este tipo de problemas.

La manera en la que tenía que preguntar era clara: cómo poder controlar en tiempo real el contenido del modal para decidir al instante qué mostrar y qué ocultar. La clave me la acabó dando ChatGPT, al sugerir introducir en mi función JavaScript el uso de AJAX para la actualización parcial de la página, en este caso, la actualización del contenido del modal. Esta solución consistía en lo siguiente:

1. Iniciar el modal directamente con la imagen de carga.
2. Utilizar AJAX para enviar una solicitud a la función que testearía el modelo y, a través de JsonResponse, enviar las estadísticas en forma de un diccionario JSON. Luego, solo quedaba recuperar estos datos con AJAX.
3. Una vez recuperados los valores con AJAX, cambiar la imagen de carga por la tabla de estadísticas con los valores actualizados.

El código referente a la función explicada estará en el código de la plantilla ***base.html*** (Código A.1) y la explicación del botón encargada de hacer dichas llamadas en el Código A.3.

5.3 Actualizar por partes la pantalla de la partida

Entender y resolver este problema fue bastante complicado debido a la necesidad de actualizar la partida en tiempo real sin utilizar código AJAX puro, es decir, función Javascript donde se implemente el uso de AJAX. Era requerido para poder mostrar las cartas solicitadas por los jugadores de manera inmediata y no al final de la ronda.

Una vez identificado el desafío, inicié la búsqueda de posibles soluciones. Era crucial encontrar una forma de realizar esto sin recurrir al uso de código AJAX puro (función Javascript con código AJAX), lo cual planteaba un obstáculo adicional. Finalmente, encontré una solución potencial

utilizando una librería llamada HTMX. Esta herramienta permitía realizar peticiones AJAX a una URL específica de manera que no se reflejara en la barra de direcciones del navegador, cumpliendo así con mis requerimientos iniciales.

Para aprovechar al máximo HTMX, me dediqué a investigar sus capacidades y aprender cómo integrarlo con las funciones existentes en mi código y las plantillas. Afortunadamente, descubrí que su uso era relativamente sencillo: solo necesitaba especificar las acciones deseadas en la definición del botón HTML.

Sin embargo, esta solución planteó un nuevo desafío. Necesitaba dividir mi pantalla principal en secciones distintas para poder indicar correctamente dónde se deberían realizar los cambios provocados por las llamadas de HTMX.

La solución a este nuevo problema resultó ser directamente compatible con Django. Aprovechando la capacidad del framework para utilizar bucles en las plantillas HTML, pude diferenciar adecuadamente entre los diferentes tipos de jugadores y así incluir las plantillas correspondientes según fuera necesario. Esta integración permitió que, en el código de la función, renderizará cada parte específica de la pantalla según las necesidades del flujo de juego.

En resumen, mediante el uso estratégico de HTMX y las funcionalidades de Django, logré resolver los desafíos técnicos planteados y proporcionar una solución eficaz para actualizar dinámicamente la partida en tiempo real. Haciendo una breve recapitulación de los pasos seguidos para lograr la solución:

1. Uso de HTMX en los botones de inicio/final de turno y de pedir carta/quedarse de los jugadores.
2. Creación de una nueva función que reciba la petición y aplique la lógica de cada jugador.
3. Separación en diferentes plantillas para cada tipo de jugador y sus cartas.

5.4 Ocultar los botones de acción en la pantalla de jugar

El problema que enfrentaba era aparentemente simple pero resultó ser considerablemente complejo: necesitaba poder ocultar y mostrar dinámicamente los botones que controlan el inicio y el final de los turnos de los jugadores en mi aplicación. Aunque la tarea parecía sencilla inicialmente, surgió una serie de desafíos durante la implementación que complicaron su resolución.

Para abordar esta problemática, decidí utilizar JavaScript para crear una función que manipulara los estilos de los botones al ser pulsados. Esta función recibiría como parámetro el botón específico que se activaba y buscaría el `div` contenedor del botón para aplicarle el estilo de "***display: none;***" de esta manera se lograría ocultar de la vista del usuario.

La solución que implementé para los botones del usuario sería bastante similar pero con la diferencia de que esta vez habría que ocultar dos botones a la vez, para ello encapsulé ambos en un mismo `div` que sería el que recibiera el estilo para ocultarse. Esta estrategia no solo simplificó la lógica de ocultamiento y mostrado de los botones, sino que también mejoró la gestión de la interfaz de usuario, asegurando una experiencia fluida y consistente para los jugadores.

El código referente a las funciones está presente en la plantilla base de la aplicación de juego, que puede consultarse en: Código A.5. Además, la división de los botones y los elementos `div` están en las plantillas de los jugadores: Código A.9 y Código A.10.

Dado por finalizado el capítulo de las dificultades que me he ido encontrando a lo largo del desarrollo de este proyecto, pasaré al último capítulo que se incluye en esta memoria y no es otro que las conclusiones a las que he llegado.

6 Conclusiones

En este capítulo explicaré las conclusiones a las que he llegado durante el desarrollo del proyecto, éste ha sido un largo camino de investigación y desarrollo de código que me ha hecho desempolvar los conocimientos adquiridos en las asignaturas de FAST e Ingeniería de Software.

El objetivo principal de este proyecto era la creación de una página web en la que se pudiera jugar al blackjack, con la posterior adición de modelos de Inteligencia Artificial que también pudieran participar en la partida. Este objetivo se considera alcanzado.

A su vez, para poder alcanzar ese objetivo ha sido necesario dedicarle un tiempo amplio al estudio y entendimiento de la Inteligencia Artificial, cómo se programa, se entrena, etc. Además, una vez se programaba, el coste de tiempo para poder entrenarla ha sido algo descomunal debido a la necesidad de realizar pruebas constantes hasta conseguir modelos que cumpliesen mis requisitos (ya mencionados en el Diseño: Apartado 3.1.5). Así que si se hiciera un cómputo temporal de todo el apartado de Inteligencia Artificial sería de alrededor de unos 3 meses.

También, las labores de diseño de la página web han sido bastante laboriosas debido a la serie de dificultades que me fueron surgiendo (cómo se ve en el Capítulo 5). Pero al fin del proyecto, he acabado bastante contento con mi desempeño ya que después de haber sufrido en los aspectos de Frontend en la carrera, haber logrado desarrollar esta página web me ha dejado satisfecho. Inclusive el apartado de la Inteligencia Artificial, a pesar de haber consumido una parte considerable de mi tiempo dedicado al proyecto, ser capaz de haber creado dos modelos de Inteligencia Artificial y que funcionasen, me ha hecho aumentar el abanico de posibles desempeños laborales en el futuro.

Sin embargo, al testear los modelos de Inteligencia Artificial, se confirmó una sospecha previa: el modelo de Inteligencia Artificial que cuenta cartas obtiene resultados iguales o inferiores al modelo que no cuenta cartas, lo cual inicialmente se pensaba que debía ocurrir.

El razonamiento detrás de esta situación es que en el juego del blackjack, el hecho de que un jugador salga vencedor después de varias rondas depende de las ganancias económicas obtenidas y no tanto del número de victorias. Es en esta situación donde la habilidad de contar cartas puede diferenciar a un jugador, pero dado que el enfoque de este proyecto no es generar adicción al juego ni enfocarse en las ganancias económicas, se produce esta "equivocación" o "decepción" en los resultados de los modelos de Inteligencia Artificial.

El razonamiento comentado se puede ver demostrado en la Figura 4.7, dónde los resultados que obtiene el modelo no contador de cartas son mejores que el contador de cartas. Sin embargo, puede que mi razonamiento llegue a estar equivocado y quizás con una elaboración más óptima de los modelos esos resultados den un giro, es por ello que la mejora de los modelos lo considero como líneas futuras de trabajo.

Otra conclusión derivada del desarrollo del proyecto es la poca criticidad del número de partidas simuladas en la creación de los datos y el entrenamiento de los modelos de Inteligencia Artificial, así como para el jugador basado en el algoritmo Q-Learning.

Esto puede provocar un pensamiento de que es totalmente incoherente ya que siempre se ha tenido la creencia de que cuanto mayor número de datos con el que se entrene a un agente, mejores resultados se debería obtener, pero sólo se debe de aplicar esa lógica cuando el banco de datos pueden ser muy diferentes entre ellos, o al menos, la suficiente diferencia cómo para que elegir dos resultados distintos para cada set no produzca discordancia.

En el contexto del proyecto, nuestro banco de datos se basa en una baraja francesa de 52 cartas, de las cuales, para el blackjack, solo hay 13 cartas repetidas 4 veces, y en este caso, repetidas 24 veces (4 palos por 6 barajas). Las combinaciones posibles de cartas son muy restringidas, lo que implica que después de un cierto número de partidas simuladas, algunas partidas se repetirán en el reparto inicial de cartas. Además, la respuesta del jugador puede variar en cada caso, generando un punto de confusión para el modelo de Inteligencia Artificial y el algoritmo Q-Learning.

Esta explicación tiene un trasfondo más matemático en el Capítulo 2.

Este último punto podría ser abordado en un futuro TFG por algún estudiante para mejorar los conjuntos de datos utilizados en el entrenamiento de los modelos o quizás cambiar los parámetros considerados para formar dichos conjuntos. Otra posible línea futura de trabajo sería la inclusión de apuestas en la partida, lo que reflejaría más claramente la diferencia entre un jugador que cuenta cartas y uno que no. Además, se podrían añadir opciones como separar cartas cuando son iguales al inicio de la mano, entre otras.

Aunque parte del código no se expone en esta memoria por razones de copyright, siempre estaré dispuesto a facilitar mi ayuda a futuros estudiantes que deseen avanzar en este proyecto.

Apéndice A
Código del Frontend

En este apéndice veremos todo el código referente al apartado del Frontend, llamados templates o plantillas.

A.1 Plantilla base.html

Código A.1 base.html.

```
{% load static %}

<!doctype html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1,
            shrink-to-fit=no">
        <title>{% block tittle %}Blackjack & IA{% endblock %}</title>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4
            .0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
            +058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="
            anonymous">
        <link rel="stylesheet" href="{% static 'blackjack/style.css' %}">
    </head>
    <body>
        {% block content %}
        {% endblock %}
        <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/
            popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
            ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script
            >
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/
            bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/
            JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></
            script>
        <script>
            $(document).ready(function() {
                $('#statisticsModal').on('hidden.bs.modal', function () {
```

```
                $(this).find('.modal-body').html('<div class="center-
                    container" id="loading"><img src="{% static 'blackjack/
                    images/carga.gif' %}" alt="Cargando..."></div>');
            });
        });
    </script>
    <script>
        $(document).ready(function() {
            $('#test-model-form').on('submit', function(e) {
                e.preventDefault(); // Prevenir el envío del formulario
                    tradicional

                // Mostrar el modal con el GIF de carga
                $('#statisticsModal').modal('show');

                // Enviar la solicitud AJAX
                $.ajax({
                    url: $(this).attr('action'),
                    type: $(this).attr('method'),
                    data: $(this).serialize(),
                    success: function(response) {
                        // Actualizar el título del modal con la información
                            recibida
                        $('#exampleModalLongTitle').html(`Estadísticas del
                            Modelo: <strong>${response.file_model}</strong>.
                            Testeado en <strong>${response.num_games}</strong>
                             Partidas`);

                        // Actualizar el contenido del modal con las nuevas
                            estadísticas
                        $('#statisticsModal .modal-body').html(response.
                            statistics_html);
                    },
                    error: function() {
                        // En caso de error, mostrar un mensaje en el modal
                        $('#statisticsModal .modal-body').html('<p>Hubo un
                            error al obtener las estadísticas. Por favor,
                            intenta de nuevo.</p>');
                    }
                });
            });
        });
    </script>
    <script>
        function toggleField() {
            // Obtener el valor seleccionado en el menú desplegable
            var seleccion = document.getElementById("id_type_player").value;
            var seleccion_modal = document.getElementById("
                id_type_player_modal").value;

            // Obtener el div que contiene el campo dinámico
            var ai_files = document.getElementById("ai_files");
            var q_files = document.getElementById("q_files");
            var ai_files_modal = document.getElementById("ai_files_modal");
            var q_files_modal = document.getElementById("q_files_modal");
```

```
                // Mostrar u ocultar el campo basado en la selección
                if (seleccion === "ai") {
                    ai_files.classList.remove("hidden");
                    q_files.classList.add("hidden");
                }
                if (seleccion === "q") {
                    ai_files.classList.add("hidden");
                    q_files.classList.remove("hidden");
                }
                if (seleccion === "npc") {
                    ai_files.classList.add("hidden");
                    q_files.classList.add("hidden");
                }
                if (seleccion_modal === "ai") {
                    ai_files_modal.classList.remove("hidden");
                    q_files_modal.classList.add("hidden");
                }
                if (seleccion_modal === "q_learning") {
                    ai_files_modal.classList.add("hidden");
                    q_files_modal.classList.remove("hidden");
                }
                if (seleccion_modal === "npc") {
                    ai_files_modal.classList.add("hidden");
                    q_files_modal.classList.add("hidden");
                }
            }
        </script>
        <script>
            // Javascript para llamar al modal de editar jugador
            function edit_player_modal(player_id) {
                $('#edit_modal').modal('show')
                $('#edit_modal').find('#editPlayerId').val(player_id);
            }
        </script>
        <script>
            // Javascript para llamar al modal de eliminar jugador
            function remove_player_modal(player_id) {
                $('#remove_modal').modal('show')
                $('#remove_modal').find('#deletePlayerId').val(player_id);
            }
        </script>
        <script>
            // Javascript para mostrar el modal de inicio de las partidas
            function start_game_modal() {
                $('#continue_game').modal('show');
            }
            function new_game_modal() {
                $('#new_game').modal('show');
            }
        </script>
    </body>
</html>
```

A.2 Plantilla index.html

Código A.2 Plantilla index.html.

```
{% extends 'blackjack/base.html' %}

{% load static %}

{% block content %}
    <div class="col-sm-12">
        <div class="center-container">
            <h1>Inicio</h1>
        </div>
        <div class="center-container">
            <h2>Bienvenido a la página principal, selecciona lo que quieras
                hacer</h2>
        </div>
        <div class="center-container">
            <div class="col-sm-3 margin-left-10">
                <div class="card" style="width: 20rem; height: 25rem;">
                    <div class="card-body">
                        <img src="{% static 'blackjack/images/create_dataset.jpg'
                             %}" class="card-img-top">
                        <h5 class="card-title"><strong>Crear un set de datos</
                            strong></h5>
                        <p class="card-text">Aquí podrás crear un set de datos.
                            Además podrás crear, entrenar y testear modelos de
                            Inteligencia Artificial.</p>
                        <div class="button-container">
                            <a href="{% url 'blackjack:dataset' %}" class="btn
                                btn-primary">Formulario Set de Datos</a>
                        </div>
                    </div>
                </div>
            </div>
            <div class="col-sm-3 margin-left-10">
                <div class="card" style="width: 20rem; height: 25rem;">
                    <div class="card-body">
                        <img src="{% static 'blackjack/images/blackjack_table.jpg
                            ' %}" id="bj_img" class="card-img-top">
                        <h5 class="card-title"><strong>Jugar</strong></h5>
                        <p class="card-text">Aquí podrás jugar directamente,
                            previa al inicio de la partida será necesario que
                            elijas número y tipo de los jugadores.</p>
                        <div class="button-container">
                            <a href="{% url 'blackjack:create_players' %}" class=
                                "btn btn-primary">Creación de la Partida</a>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

A.3 Plantilla dataset.html

Código A.3 Plantilla dataset.html.

```
{% extends 'blackjack/base.html' %}

{% load static %}

{% block content%}
    <div class="home">
        <a href="{% url 'blackjack:index' %}"><img src="{% static 'blackjack/
            images/home.png' %}" id="home"/></a>
    </div>
    <div id="accordion">
        <div class="card">
            <div class="card-header" id="headingOne">
                <h5 class="mb-0">
                    <button class="btn btn-link" data-toggle="collapse" data-
                        target="#collapseOne" aria-expanded="true" aria-controls=
                        "collapseOne">
                    Formulario para crear un set de datos
                    </button>
                </h5>
            </div>
            <div id="collapseOne" class="collapse show" aria-labelledby="
                headingOne" data-parent="#accordion">
                <div class="card-body">
                    <form method="post" action="{% url 'blackjack:dataset' %}">
                        {% csrf_token %}
                        <div class="form-group">
                            <label for="id_games" class="margin-top-10">Número de
                                 Partidas:</label>
                            <input type="number" class="form-control" id="
                                id_games" name="games"
                            value="{{ form.games.value|default_if_none:'' }}" min
                                =0 required>
                            <small id="games_help_block" class="form-text text-
                                muted">
                                Introduce el número de partidas que quieras
                                    simular para crear el dataset. Cómo ayuda => 1
                                     partida === 1 valor para el set.
                            </small>
                        </div>
                        <div class="form-group">
                            <label for="id_file_name" class="margin-top-10">
                                Nombre del Archivo:</label>
                            <input type="text" class="form-control" id="
                                id_file_name" name="file_name"
                            value="{{ form.file_name.value|default_if_none:'' }}"
                                 required>
                            <small id="file_name_help_block" class="form-text
                                text-muted">
                                Introduce el nombre con el que quieras guardar el
                                    archivo. No indicar "_count" en el nombre.
                            </small>
                        </div>
                        <div class="button-container">
                            <button type="submit" class="btn btn-success">Crear
                                Dataset</button>
```

```
                    </div>
                </form>
            </div>
            {% if dataset_success %}
                <div class="alert alert-success" role="alert">
                    El set de datos ha sido creado correctamente!!!
                </div>
            {% endif %}
        </div>
    </div>
    <div class="card">
        <div class="card-header" id="headingTwo">
        <h5 class="mb-0">
            <button class="btn btn-link collapsed" data-toggle="collapse"
                data-target="#collapseTwo" aria-expanded="false" aria-
                controls="collapseTwo">
            Formulario para crear un modelo de Inteligencia Artificial
            </button>
        </h5>
        </div>
        <div id="collapseTwo" class="collapse" aria-labelledby="headingTwo"
            data-parent="#accordion">
            <div class="card-body">
                <form method="post" action="{% url 'blackjack:model' %}">
                    {% csrf_token %}
                    <div>
                        <label for="id_games" class="margin-top-10">Nombre
                            del modelo:</label>
                        <input type="text" id="id_model_name" class="form-
                            control" name="model_name"
                        value="{{ form.model_name.value|default_if_none:'' }}
                            " required>
                        <small id="model_name_help_text" class="form-text
                            text-muted">
                            Introduce el nombre con el que quieras guardar el
                                modelo. No indicar "_count" en el modelo.
                        </small>
                    </div>
                    <div>
                        <label for="id_training_size" class="margin-top-10">
                            Tamaño que se le dedicará al set de entrenamiento:
                            </label>
                        <input type="number" id="id_training_size" class="
                            form-control" name="training_size" min="25" max="
                            95"
                        value="{{ form.training_size.value|default_if_none
                            :'70' }}" required>
                        <small id="training_size_help_text" class="form-text
                            text-muted">
                            Introduce un número entre 25 y 95. Este número ser
                                á el porcentaje destinado al set de datos para
                                 entrenar el modelo.
                        </small>
                    </div>
                    <div>
```

```
                        <label for="id_num_epochs" class="margin-top-10">Nú
                            mero de vueltas que se dará al set de datos para
                            entrenar:</label>
                        <input type="number" id="id_num_epochs" class="form-
                            control" name="num_epochs" min="5"
                        value="{{ form.num_epochs.value|default_if_none:'50'
                            }}" required>
                        <small id="num_epochs_help_text" class="form-text
                            text-muted">
                            Introduce el número de vueltas que quieres que de
                                el modelo al set de datos para entrenarse,
                                mayor a 5.
                        </small>
                    </div>
                    <div>
                        <label for="id_file_name" class="margin-top-10">
                            Nombre del Archivo del Set de Datos:</label>
                        <select id="id_file_name" class="form-select" name="
                            file_name" required>
                            {% for file, _ in data_files %}
                                <option value="{{ file }}">{{ file }}</option>
                            {% endfor %}
                        </select>
                        <small id="file_name_help_text" class="form-text text
                            -muted">
                            Elige el set de datos que quieras para entrenar al
                                 modelo.
                        </small>
                    </div>
                    <div class="button-container">
                        <button type="submit" class="btn btn-success">Crear
                            Modelo</button>
                    </div>
                </form>
            </div>
            {% if model_success %}
                <div class="alert alert-success" role="alert">
                    El modelo de Inteligencia Artificial ha sido creado y
                        entrenado correctamente!!!
                </div>
            {% endif %}
        </div>
    </div>
    <div class="card">
        <div class="card-header" id="headingThree">
        <h5 class="mb-0">
            <button class="btn btn-link collapsed" data-toggle="collapse"
                data-target="#collapseThree" aria-expanded="false" aria-
                controls="collapseThree">
            Testear modelo de Inteligencia Artificial
            </button>
        </h5>
        </div>
        <div id="collapseThree" class="collapse" aria-labelledby="
            headingThree" data-parent="#accordion">
            <div class="card-body">
```

```
                <form id="test-model-form" method="post" action="{% url '
                    blackjack:test_model' %}">
                    {% csrf_token %}
                    <div class="form-group">
                        <label for="id_num_games" class="margin-top-10">Nú
                            mero de Partidas:</label>
                        <input type="number" class="form-control" id="
                            id_num_games" name="num_games"
                        value="{{ form.games.value|default_if_none:'' }}" min
                            =0 required>
                        <small id="num_games_help_block" class="form-text
                            text-muted">
                            Introduce el número de partidas que quieras
                                simular para testear el modelo. Este número se
                                 usará para ambos tipos de modelos.
                        </small>
                    </div>
                    <div class="form-group">
                        <label for="id_file_model" class="margin-top-10">
                            Nombre del Modelo a testear:</label>
                        <select id="id_file_model" class="form-select" name="
                            file_model" required>
                            {% for file, _ in model_files %}
                                <option value="{{ file }}">{{ file }}</option>
                            {% endfor %}
                        </select>
                        <small id="file_model_help_text" class="form-text
                            text-muted">
                            Elige el modelo que quieras testear. Se testeará
                                tanto el modelo contador de cartas como el no
                                contador de cartas.
                        </small>
                    </div>
                    <div class="button-container">
                        <button type="submit" class="btn btn-success">
                            Testear modelo
                        </button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>
<!-- Modal -->
<div class="modal fade bd-example-modal-lg" tabindex="-1" role="dialog"
    aria-labelledby="myLargeModalLabel" aria-hidden="true" id="
    statisticsModal">
    <div class="modal-dialog modal-lg">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="exampleModalLongTitle"></h5>
                <button type="button" class="close" data-dismiss="modal" aria
                    -label="Close">
                <span aria-hidden="true">&times;</span>
                </button>
            </div>
```

```
                <div class="modal-body"></div>
            </div>
        </div>
    </div>
{% endblock %}
```

A.4 Plantilla create_players.html

Código A.4 Plantilla create_players.html.

```
{% extends 'blackjack/base.html' %}

{% load static %}

{% block content%}
    <div class="col-sm-12">
        <div class="center-container">
            {% if players|length < 6 %}
                <div class="{% if players_created == 0 %} col-sm-6 {% else %}
                    col-sm-3 {% endif %}">
                    <div class="card">
                        <div class="card-body">
                            <h5 class="card-title"><strong>Formulario Creación
                                Jugadores</strong></h5>
                            <form method="post" action="{% url 'blackjack:
                                create_players' %}">
                                {% csrf_token %}
                                <div class="form-group">
                                    <label for="id_type_player" class="margin-top
                                        -10">Tipo de Jugador:</label>
                                    <select id="id_type_player" class="form-select
                                        " name="type_player" onchange="toggleField
                                        ()" required>
                                        <option>...</option>
                                        <option value="ai">Modelo de Inteligencia
                                            Artificial</option>
                                        <option value="q">Algoritmo Q-Learning</
                                            option>
                                        <option value="npc">Jugador NPC (Non
                                            Playable Character)</option>
                                    </select>
                                    <small id="type_player_help_text" class="form-
                                        text text-muted">
                                        Elige el tipo de jugador que quieras que
                                            participe en la partida.
                                    </small>
                                </div>
                                <div class="form-group">
                                    <div class="hidden" id="ai_files">
                                        <label for="id_ai_files">Modelo
                                            Inteligencia Artificial</label>
                                        <select id="id_ai_files" class="form-select
                                            " name="ai_files">
```

```
                                    {% for file, _ in ai_files %}
                                        <option value="{{ file }}">{{ file
                                            }}</option>
                                    {% endfor %}
                                </select>
                                <small id="ai_files_help_text" class="form-
                                    text text-muted">
                                    Elige el Modelo de Inteligencia
                                        Artificial que juegue en la partida
                                        .
                                </small>
                            </div>
                            <div class="hidden" id="q_files">
                                <label for="id_q_files">Algoritmo Q-
                                    Learning</label>
                                <select id="id_q_files" class="form-select"
                                     name="q_files">
                                    {% for file, _ in q_files %}
                                        <option value="{{ file }}">{{ file
                                            }}</option>
                                    {% endfor %}
                                </select>
                                <small id="q_files_help_text" class="form-
                                    text text-muted">
                                    Elige la Tabla-Q que juegue en la
                                        partida. Mayor número de partidas,
                                        más probabilidad de victoria.
                                </small>
                            </div>
                        </div>
                        <div class="form-group">
                            <div class="form-check">
                                <input class="form-check-input" type="
                                    checkbox" id="id_count_cards" name="
                                    count_cards" value="True">
                                <label class="form-check-label" for="
                                    id_count_cards">
                                    Cuenta Cartas
                                </label>
                            </div>
                        </div>
                        <div class="button-container">
                            <button type="submit" class="btn btn-success">
                                Añadir Jugador</button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    {% endif %}
    {% if players %}
        <div class="col-sm-6">
            <div class="card">
                <div class="card-body">
                    <h5 class="center-container"><strong>Lista Jugadores
                        Creados</strong></h5>
```

```
                        <table class="table table-striped">
                            <thead>
                                <tr>
                                    <th>Tipo</th>
                                    <th>Nombre Modelo</th>
                                    <th>Cuenta Cartas</th>
                                    <th>Acciones</th>
                                </tr>
                            </thead>
                            <tbody>
                                {% for player in players %}
                                    <tr>
                                        <td>{{ player.to_uppercase }}</td>
                                        <td>{% if player.model_name != "
                                            Desconocido" %}{{ player.model_name
                                             }}{% endif %}</td>
                                        <td>{% if '_count' in player.model_name
                                             or 'hi_lo' == player.model_name %}
                                             SI {% else %} NO {% endif %}</td>
                                        <td>
                                            {% if player.type_player != "user"
                                                %}
                                                <button type="button" class="btn
                                                     btn-primary" data-toggle="
                                                    edit_modal" data-target="#
                                                    edit_modal"
                                                data-id="{{ player.id }}"
                                                    onclick="edit_player_modal
                                                    ({{ player.id }})">Editar</
                                                    button>
                                                <button type="button" class="btn
                                                     btn-danger" data-toggle="
                                                    remove_modal" data-target="#
                                                    remove_modal"
                                                data-id="{{ player.id }}"
                                                    onclick="remove_player_modal
                                                    ({{ player.id }})">Borrar</
                                                    button>
                                            {% endif %}
                                        </td>
                                    </tr>
                                {% endfor %}
                            </tbody>
                        </table>
                        <div class="button-container margin-top-10">
                            <!-- Si se crean jugadores nuevos, sólo se puede
                                empezar una nueva partida -->
                            {% if not players_created or size_before ==
                                players_created %}
                                <button type="button" class="btn btn-success"
                                    data-toggle="continue_game"
                                data-target="#continue_game" onclick="
                                    start_game_modal()">
                                    Continuar Partida
                                </button>
```

```
                                <button type="button" class="btn btn-success
                                    margin-left-10" data-toggle="new_game"
                                data-target="#new_game" onclick="
                                    new_game_modal()">
                                    Empezar Nueva Partida
                                </button>
                            {% else %}
                                <button type="button" class="btn btn-success
                                    margin-left-10" data-toggle="new_game"
                                data-target="#new_game" onclick="
                                    new_game_modal()">
                                    Empezar Nueva Partida
                                </button>
                            {% endif %}
                        </div>
                    </div>
                </div>
            </div>
        {% endif %}
    </div>
</div>
<!-- Modal de Editar Jugador -->
<div class="modal fade" id="edit_modal" tabindex="-1" role="dialog" aria-
    labelledby="edit_modalTitle" aria-hidden="true">
    <div class="modal-dialog modal-dialog-centered" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Editar Jugador</h5>
                <button type="button" class="close" data-dismiss="modal" aria
                    -label="Close"><span aria-hidden="true">&times;</span></
                    button>
            </div>
            <div class="modal-body">
                <form method="post" action="{% url 'blackjack:edit_players'
                    %}">
                    {% csrf_token %}
                    <div class="form-group">
                        <label for="id_type_player_modal" class="margin-top
                            -10">Tipo de Jugador:</label>
                        <select id="id_type_player_modal" class="form-select"
                             name="type_player" onchange="toggleField()"
                            required>
                            <option>...</option>
                            <option value="ai">Modelo de Inteligencia
                                Artificial</option>
                            <option value="q">Algoritmo Q-Learning</option>
                            <option value="npc">Jugador NPC (Non Playable
                                Character)</option>
                        </select>
                            <small id="type_player_help_text" class="form-text
                                 text-muted">
                            Elige el tipo de jugador que quieras que
                                participe en la partida.
                            </small>
                    </div>
                    <div class="form-group">
```

```
                        <div class="hidden" id="ai_files_modal">
                            <label for="id_ai_files">Modelo Inteligencia
                                Artificial</label>
                            <select id="id_ai_files" class="form-select" name=
                                "ai_files">
                                {% for file, _ in ai_files %}
                                    <option value="{{ file }}">{{ file }}</
                                        option>
                                {% endfor %}
                            </select>
                            <small id="ai_files_help_text" class="form-text
                                text-muted">
                                Elige el Modelo de Inteligencia Artificial que
                                     juegue en la
                                partida.
                            </small>
                        </div>
                        <div class="hidden" id="q_files_modal">
                            <label for="id_q_files">Algoritmo Q-Learning</
                                label>
                            <select id="id_q_files" class="form-select" name="
                                q_files">
                                {% for file, _ in q_files %}
                                    <option value="{{ file }}">{{ file }}</
                                        option>
                                {% endfor %}
                            </select>
                            <small id="q_files_help_text" class="form-text
                                text-muted">
                                Elige la Tabla-Q que juegue en la partida.
                                    Mayor número de
                                partidas, más probabilidad de victoria.
                            </small>
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="form-check">
                            <input class="form-check-input" type="checkbox" id
                                ="id_count_cards_modal"
                            name="count_cards" value="True"/>
                            <label class="form-check-label" for="
                                id_count_cards_modal">Cuenta Cartas</label>
                        </div>
                    </div>
                    <div class="button-container">
                        <button type="submit" class="btn btn-success">Editar
                            Jugador</button>
                    </div>
                    <input type="hidden" name="edit_player_id" id="
                        editPlayerId" value=""/>
                </form>
            </div>
        </div>
    </div>
</div>
<!-- Modal de Borrar Jugador -->
```

```
    <div class="modal fade" id="remove_modal" tabindex="-1" role="dialog" aria-
        labelledby="remove_modalTitle" aria-hidden="true">
        <div class="modal-dialog modal-dialog-centered" role="document">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title">Eliminar Jugador</h5>
                    <button type="button" class="close" data-dismiss="modal" aria
                        -label="Close"><span aria-hidden="true">&times;</span></
                        button>
                </div>
                <div class="modal-body">
                    <form action="{% url 'blackjack:edit_players' %}" method="
                        POST">
                        {% csrf_token %}
                        <p>¿Estás seguro de que deseas borrar el jugador?</p>
                        <div class="button-container">
                            <button type="submit" class="btn btn-info">Sí</button
                                >
                            <button type="submit" class="btn btn-info margin-left
                                -10" data-dismiss="modal" aria-label="Close">No</
                                button>
                        </div>
                        <input type="hidden" name="delete_player_id" id="
                            deletePlayerId" value=""/>
                    </form>
                </div>
            </div>
        </div>
    </div>
    <!-- Modal para Continuar Partida -->
    <div class="modal fade" id="continue_game" tabindex="-1" role="dialog" aria
        -labelledby="continue_gameTitle" aria-hidden="true">
        <div class="modal-dialog modal-dialog-centered" role="document">
            <div class="modal-content">
                <div class="modal-body">
                    <form action="{% url 'blackjack:game' %}" method="POST">
                        {% csrf_token %}
                        <p>¿Continuar con la partida anterior?</p>
                        <div class="button-container">
                            <button type="submit" class="btn btn-info">Sí</button
                                >
                            <button type="submit" class="btn btn-info margin-left
                                -10" data-dismiss="modal" aria-label="Close">No</
                                button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
    <!-- Modal para Empezar una Nueva Partida -->
    <div class="modal fade" id="new_game" tabindex="-1" role="dialog" aria-
        labelledby="new_gameTitle" aria-hidden="true">
        <div class="modal-dialog modal-dialog-centered" role="document">
            <div class="modal-content">
                <div class="modal-body">
```

```
                    <form action="{% url 'blackjack:game' %}" method="POST">
                        {% csrf_token %}
                        <p>¿Estás seguro de que deseas empezar una nueva partida?
                            </p>
                        <div class="button-container">
                            <button type="submit" class="btn btn-info">Sí</button
                                >
                            <button type="submit" class="btn btn-info margin-left
                                -10" data-dismiss="modal" aria-label="Close">No</
                                button>
                        </div>
                        <input type="hidden" name="reboot" id="reboot" value="
                            true"/>
                    </form>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

A.5 Plantilla game_base.html

Código A.5 Plantilla game_base.html.

```
{% load static %}

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1,
            shrink-to-fit=no"/>
        <title>{% block tittle %}Blackjack & IA{% endblock %}</title>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4
            .0.0/dist/css/bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
            +058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="
            anonymous"/>
        <link rel="stylesheet" href="{% static 'blackjack/style.css' %}" />
    </head>
    <body id="game">
        {% block content %} {% endblock %}
        <!-- Integración de Bootstrap en el proyecto, incluyendo JQuery, Popper
             y Javascript -->
        <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
            integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
            GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/
            popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
            ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script
            >
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/
            bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/
```

```
            JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></
            script>
        <!-- Integración de HTMX en el proyecto para realizar peticiones AJAX
            evitando el recargo total de la página -->
        <script src="https://unpkg.com/htmx.org@2.0.0" integrity="sha384-
            wS5l5IKJBvK6sPTKa2WZ1js3d947pvWXbPJ1OmWfEuxLgeHcEbjUUA5i9V5ZkpCw"
            crossorigin="anonymous"></script>
        <!-- Funciones Javascript para ocultar el botón que llama a esta funció
            n -->
        <script>
            function hideButton(button) {
                button.style.display="none";
            }
        </script>
        <!-- Función Javascript para ocultar el div donde se ubican los botones
             de acción del usuario -->
        <script>
            function hideParentDiv(button) {
                var parentDiv = button.closest("#action-buttons-user");
                if (parentDiv) {
                    parentDiv.style.display = "none";
                }
            }
        </script>
        <!-- Función Javascript para ocultar el div donde se ubican los botones
             de fin de turno del usuario -->
        <script>
            function hideParentDivUSER(button) {
                var parentDiv = button.closest("#action-buttons-turn");
                if (parentDiv) {
                    parentDiv.style.display = "none";
                }
            }
        </script>
        <!-- Función Javascript para ocultar el div donde se ubican los botones
             de acción del jugador IA -->
        <script>
            function hideParentDivAI(button) {
                var parentDiv = button.closest("#action-buttons-ai");
                if (parentDiv) {
                    parentDiv.style.display = "none";
                }
            }
        </script>
        <!-- Función Javascript para ocultar el div donde se ubican los botones
             de acción del jugador NPC -->
        <script>
            function hideParentDivNPC(button) {
                var parentDiv = button.closest("#action-buttons-npc");
                if (parentDiv) {
                    parentDiv.style.display = "none";
                }
            }
        </script>
        <!-- Función Javascript para ocultar el div donde se ubican los botones
             de acción del jugaor Q-LEARNING -->
```

```
        <script>
            function hideParentDivQ(button) {
                var parentDiv = button.closest("#action-buttons-q");
                if (parentDiv) {
                    parentDiv.style.display = "none";
                }
            }
        </script>
    </body>
</html>
```

A.6 Plantilla game_body.html

Código A.6 Plantilla game_body.html.

```
{% extends 'blackjack/game_base.html' %}

{% load static %}

{% block content %}
    <div class="table" id="table">
        <button type="button" class="btn btn-info" id="statistics-button" data-
            toggle="modal" data-target=".bd-example-modal-lg">
            Ver Estadísticas Generales
        </button>
        {% if created %}<div class="created">NUEVO MAZO CREADO</div>{% endif %}
        {% include 'blackjack/snippets/results.html' %}
        <div class="dealer-area">
            <!-- Creamos un slot de cartas para el crupier -->
            {% include 'blackjack/snippets/dealer_cards.html' %}
        </div>
        <!-- Creamos slots de cartas para los jugadores -->
        {% include 'blackjack/snippets/player_cards.html' %}
    </div>
    <!-- Modal que lanza una tabla con las estadísticas generales de la partida
         -->
    <div class="modal fade bd-example-modal-lg" tabindex="-1" role="dialog"
        aria-labelledby="myLargeModalLabel" aria-hidden="true" id="
        statisticsModal">
        <div class="modal-dialog modal-lg">
            <div class="modal-content">
                <div class="modal-header" id="modal-header-statistics">
                    <h5 class="modal-title" id="exampleModalLongTitle">Estadí
                        sticas Generales - <strong>{{ num_games }}</strong>
                        Partidas Jugadas</h5>
                    <button type="button" class="close" data-dismiss="modal" aria
                        -label="Close">
                    <span aria-hidden="true">&times;</span>
                    </button>
                </div>
                <div class="modal-body">
                    <table class="table table-striped table-bordered">
                        <thead>
```

```
                        <tr>
                            <th>Estadística</th>
                            <th colspan="2">Victorias</th>
                            <th colspan="2">Derrotas</th>
                            <th colspan="2">Empates</th>
                        </tr>
                    </thead>
                    <tbody>
                        <tr>
                            <td>Jugadores</td>
                            <td>Partidas</td>
                            <td>Porcentaje</td>
                            <td>Partidas</td>
                            <td>Porcentaje</td>
                            <td>Partidas</td>
                            <td>Porcentaje</td>
                        </tr>
                        {% for player in players %}
                            <tr>
                            <!-- lo recorremos con un for por todos los
                                jugadores creados -->
                                <th scope="row" class="table-light">
                                    {% if '_count' in player.model_name and
                                        player.type_player == "ai" %}
                                        {{ player.to_uppercase }}_COUNT - {{
                                            forloop.counter }}
                                    {% elif player.model_name == "hi_lo" and
                                        player.type_player == "npc" %}
                                        {{ player.to_uppercase }}_COUNT - {{
                                            forloop.counter }}
                                    {% else %}
                                        {{ player.to_uppercase }} - {{ forloop.
                                            counter }}
                                    {% endif %}
                                </th>
                                <td class="
                                    {% if player.wins > player.losses %}table-
                                        success{% endif %}
                                    {% if player.wins < player.losses %}table-
                                        danger{% endif %}
                                    {% if player.wins == player.losses %}table-
                                        info{% endif %}">{{ player.wins }}</td>
                                <td class="
                                {% if player.wins > player.losses %}table-
                                    success{% endif %}
                                {% if player.wins < player.losses %}table-
                                    danger{% endif %}
                                {% if player.wins == player.losses %}table-
                                    info{% endif %}">{{ player.win_percent }}%
                                    </td>
                                <td class="
                                {% if player.wins > player.losses %}table-
                                    success{% endif %}
                                {% if player.wins < player.losses %}table-
                                    danger{% endif %}
```

```
                                {% if player.wins == player.losses %}table-
                                    info{% endif %}">{{ player.losses }}</td>
                                <td class="
                                {% if player.wins > player.losses %}table-
                                    success{% endif %}
                                {% if player.wins < player.losses %}table-
                                    danger{% endif %}
                                {% if player.wins == player.losses %}table-
                                    info{% endif %}">{{ player.loss_percent
                                    }}%</td>
                                <td class="
                                {% if player.wins > player.losses %}table-
                                    success{% endif %}
                                {% if player.wins < player.losses %}table-
                                    danger{% endif %}
                                {% if player.wins == player.losses %}table-
                                    info{% endif %}">{{ player.ties }}</td>
                                <td class="
                                {% if player.wins > player.losses %}table-
                                    success{% endif %}
                                {% if player.wins < player.losses %}table-
                                    danger{% endif %}
                                {% if player.wins == player.losses %}table-
                                    info{% endif %}">{{ player.ties_percent
                                    }}%</td>
                            </tr>
                        {% endfor %}
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>
{% endblock %}
```

A.7 Plantilla player_cards.html

Código A.7 Plantilla player_cards.html.

```
{% load static %}

<div class="player-area" id="player-area">
    {% for player in players %}
        <div class="card-slot" id="player-slot-{{ forloop.counter }}">
            {% if player.type_player == "ai" %}
                <!-- Cargamos el slot de las cartas pertenecientes a la IA -->
                {% include 'blackjack/snippets/ai_cards.html' %}
            {% elif player.type_player == "q" %}
                <!-- Cargamos el slot de las cartas pertenecientes a la
                    Q_LEARNING -->
                {% include 'blackjack/snippets/q_learning_cards.html' %}
            {% elif player.type_player == "npc" %}
                <!-- Cargamos el slot de las cartas pertenecientes al NPC -->
```

```
                {% include 'blackjack/snippets/npc_cards.html' %}
            {% else %}
                <!-- Cargamos el slot de las cartas pertenecientes al USUARIO --
                    >
                {% include 'blackjack/snippets/user_cards.html' %}
            {% endif %}
        </div>
    {% endfor %}
</div>
```

A.8 Plantilla dealer_cards.html

Código A.8 Plantilla dealer_cards.html.

```
{% load static %}

<div class="card-slot" id="dealer-slot">
    {% for card in dealer.hand %}
        {% if not dealers_turn %}
            {% if forloop.counter == 2 %}
                <img src="{% static 'blackjack/images/cards/DORSO.png' %}" class
                    ="blackjack-cards"
                id="dealer-{{ forloop.counter }}-card"></img>
            {% else %}
                <img src="{% static 'blackjack/images/cards/' %}{{ card }}.png"
                    class="blackjack-cards"
                id="dealer-{{ forloop.counter }}-card"></img>
            {% endif %}
        {% else %}
            <img src="{% static 'blackjack/images/cards/' %}{{ card }}.png"
                class="blackjack-cards"
            id="dealer-{{ forloop.counter }}-card"></img>
        {% endif %}
        <div class="dealer-name"><img src="{% static 'blackjack/images/
            dealers_icon.png' %}" class="dealer-icon"></img></div>
        <div class="dealer-score">{{ dealer.hand_value }}{% if dealers_turn and
             dealer.blackjack %}BJ{% endif %}</div>
    {% endfor %}
</div>
<div class="action-buttons">
    {% if see_results and dealers_turn %}
        <button type="button" hx-post="{% url 'blackjack:select_action' dealer.
            id %}" hx-trigger="click" hx-vals='{"results": "show"}'
        hx-target="#results" hx-swap="outerHTML" class="btn btn-info btn-lg"
            onclick="hideButton(this)">
            VER RESULTADOS
        </button>
    {% endif %}
</div>
```

A.9 Plantilla user_cards.html

Código A.9 Plantilla user_cards.html.

```
{% load static %}

{% for card in player.hand %}
    <img src="{% static 'blackjack/images/cards/' %}{{ card }}.png" class="
        blackjack-cards"
    id="card-{{ forloop.counter }}"></img>
{% endfor %}
<div class="player-name">
    {{ player.to_uppercase }}<img src="{% static 'blackjack/images/user_icon.
        png' %}" class="user-icon"></img>
</div>
<div class="player-score">
    {% if player.blackjack %}
        BJ
    {% else %}
        {{ player.hand_value }}
    {% endif %}
</div>
{% if current_player.type_player == "user" %}
    {% if not current_player.finished %}
        <div class="action-buttons" id="action-buttons-user">
            <!-- Botones de acción para el usuario -->
            <button type="button" hx-post="{% url 'blackjack:select_action'
                current_player.id %}" hx-vals='{"action": "hit"}'
            hx-target="#player-slot-{{ my_slot }}" class="btn btn-info btn-lg">
                PEDIR CARTA
            </button>
            <!-- Hacemos la diferenciación de si el siguiente usuario es dealer
                o no -->
            {% if next_player.type_player == "dealer" %}
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    dealer.id %}" hx-vals='{"action": "stand"}'
                onclick="hideParentDiv(this)" hx-target="#dealer-slot" hx-swap="
                    innerHTML" class="btn btn-info btn-lg margin-left-10">
                    QUEDARSE
                </button>
            {% else %}
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    next_player.id %}" hx-vals='{"action": "stand"}'
                onclick="hideParentDiv(this)" hx-target="#player-slot-{{
                    next_slot }}" hx-swap="innerHTML" class="btn btn-info btn-lg
                     margin-left-10">
                    QUEDARSE
                </button>
            {% endif %}
        </div>
    {% else %}
        <div class="action-buttons" id="action-buttons-turn">
            {% if next_player.type_player == "dealer"%}
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    dealer.id %}" hx-target="#dealer-slot"
                hx-swap="innerHTML" class="btn btn-info btn-lg" onclick="
                    hideParentDivUSER(this)">
                    TURNO DEL JUGADOR: {{ dealer.to_uppercase }} => INICIAR
                </button>
```

```
            {% else %}
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    next_player.id %}" onclick="hideParentDivUSER(this)"
                hx-target="#player-slot-{{ next_slot }}" hx-swap="innerHTML"
                    class="btn btn-info btn-lg">
                    TURNO DEL JUGADOR: {{ next_player.to_uppercase }} - SLOT: {{
                        next_slot }} => INICIAR
                </button>
            {% endif %}
        </div>
    {% endif %}
{% endif %}
```

A.10 Plantilla ai_cards.html

Código A.10 Plantilla ai_cards.html.

```
# ESTA PLANTILLA ES EXACTAMENTE IGUAL PARA EL JUGADOR TIPO NPC Y JUGADOR TIPO Q
    -LEARNING, LO ÚNICO QUE VARÍA ES DONDE PONE AI, DEBERÍA PONER NPC O Q.
{% load static %}

{% for card in player.hand %}
    <img src="{% static 'blackjack/images/cards/' %}{{ card }}.png" class="
        blackjack-cards"
    id="card-{{ forloop.counter }}"></img>
{% endfor %}
<div class="player-name">
    {{ player.to_uppercase }}{% if "_count" in player.model_name %}_COUNT{%
        endif %}
</div>
<div class="player-score">
    {% if player.blackjack %}
        BJ
    {% else %}
        {{ player.hand_value }}
    {% endif %}
</div>
<!-- Botón para iniciar el turno del jugador -->
{% if current_player.type_player == "ai" %}
    <div class="action-buttons" id="action-buttons-ai">
        <!-- Si el jugador no ha terminado su turno, se sigue mostrando su botó
            n de inicio -->
        {% if not current_player.finished %}
            <div id="my-button-ai">
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    current_player.id %}" onclick="hideButton(this)"
                hx-target="#player-slot-{{ my_slot }}" hx-swap="innerHTML" class
                    ="btn btn-info btn-lg">
                    TURNO DEL JUGADOR: {{ current_player.to_uppercase }} - SLOT:
                        {{ my_slot }} => INICIAR
                </button>
            </div>
        {% else %}
```

```
        <!-- En caso contrario, mostramos el botón de iniciar el turno del
            siguiente jugador, haciendo diferencia de si es el crupier o no-->
            {% if next_player.type_player == "dealer"%}
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    dealer.id %}" hx-target="#dealer-slot"
                hx-swap="innerHTML" class="btn btn-info btn-lg" onclick="
                    hideParentDivAI(this)">
                    TURNO DEL JUGADOR: {{ dealer.to_uppercase }} => INICIAR
                </button>
            {% else %}
                <button type="button" hx-post="{% url 'blackjack:select_action'
                    next_player.id %}" onclick="hideParentDivAI(this)"
                hx-target="#player-slot-{{ next_slot }}" hx-swap="innerHTML"
                    class="btn btn-info btn-lg">
                    TURNO DEL JUGADOR: {{ next_player.to_uppercase }} - SLOT: {{
                        next_slot }} => INICIAR
                </button>
            {% endif %}
        {% endif %}
    </div>
{% endif %}
```

A.11 Plantilla results.html

Código A.11 Plantilla results.html.

```
{% load static %}

<div class="results" id="results">
    {% if end_game %}
        RESULTADOS:<br />
        {% for player in players %}
                {% if player in winners %}
                    {{ player.to_uppercase }} - {{ forloop.counter }} - GANA<br /
                        >
                {% elif player in losers %}
                    {{ player.to_uppercase }} - {{ forloop.counter }} - PIERDE<br
                         />
                {% else %}
                    {{ player.to_uppercase }} - {{ forloop.counter }} - EMPATA<br
                         />
                {% endif %}
        {% endfor %}
    {% endif %}
</div>
{% if end_game %}
    <div class="action-buttons-round btn-group dropright">
        <button type="button" hx-get="{% url 'blackjack:select_action' dealer.
            id %}" hx-trigger="click"
        hx-target="#table" hx-swap="outerHTML" class="btn btn-info btn-lg"
            onclick="hideButton(this)">
            INICIAR SIGUIENTE RONDA
        </button>
```

```
        <button type="button" class="btn btn-info dropdown-toggle dropdown-
            toggle-split" data-toggle="dropdown" aria-haspopup="true" aria-
            expanded="false">
            <span class="sr-only">Toggle Dropright</span>
        </button>
        <div class="dropdown-menu">
            <!-- Dropdown menu links -->
            <a class="dropdown-item" hx-post="{% url 'blackjack:game' %}" hx-
                target="#table" hx-vals='{"reboot": "true"}'>REINICIAR PARTIDA</
                a>
            <a class="dropdown-item" href="{% url 'blackjack:create_players' %}"
                >MENÚ CREACIÓN JUGADORES</a>
            <a class="dropdown-item" href="{% url 'blackjack:index' %}">MENÚ
                PRINCIPAL</a>
        </div>
    </div>
{% endif %}
```

Apéndice B
Estilos Usados

Código B.1 style.css.

```
body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
    background: white url("images/blackjack_index.jpg") no-repeat;
}

h1 {
    margin-bottom: 25px;
    font-size: 75px;
}

h2 {
    margin-bottom: 25px;
    font-size: 25px;
}

/* Estilo para ocultar los campos */
.hidden {
    display: none;
}

/* Contenedor para centrar el contenido de la página */
.center-container {
    display: flex;
    justify-content: center;
}

#bj_img {
    height: 11.2rem;
}

/* Contenedor para centrar el botón */
.button-container {
```

```
    grid-column: span 2; /* Hace que el contenedor ocupe todo el ancho del
        formulario */
    display: flex;
    justify-content: center; /* Centra el botón horizontalmente */
}

.btn-link {
    text-decoration: none !important;
}

/* Estilo para el botón */
button[type="submit"] {
    border: none;
    cursor: pointer;
    margin-top: 20px;
}

/* Estilos creados para dar márgenes */
.margin-top-10 {
    margin-top: 10px;
}

.margin-left-10 {
    margin-left: 10px;
}

/* Estilos para la partida */
#game {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    background-image: url("images/mesa.png");
    background-size: 100% 100%;
    background-position: center;
    background-color: rgb(0,41,7);
    position: relative;
}

.card-slot {
    position: absolute;
    width: auto;
    height: auto;
    display: inline-block;
}

#player-slot-1 {
    top: 35%;
    left: 7%;
}

#player-slot-2 {
    top: 50%;
    left: 20%;
}
```

```
#player-slot-3 {
    top: 60%;
    left: 36.5%;
}

#player-slot-4 {
    top: 60%;
    left: 54.5%;
}

#player-slot-5 {
    top: 50%;
    left: 70.2%;
}

#player-slot-6 {
    top: 35%;
    left: 83%;
}

#dealer-slot {
    top: 12%;
    left: 48%;
}

.blackjack-cards {
    width: 5rem;
    height: 7.5rem;
    position: absolute;
}

.player-name {
    position: absolute;
    top: 8rem; /* Ajusta este valor según la altura total de las cartas */
    left: 0;
    text-align: center; /* Centra el texto horizontalmente */
    font-size: 1rem;
    font-weight: bold;
    color: white; /* Color del texto */
}

.player-score {
    position: absolute;
    top: -2rem; /* Ajusta este valor según la altura de las cartas */
    left: 9rem; /* Ajusta este valor según la cantidad total de desplazamiento
        horizontal de las cartas */
    font-size: 2rem;
    font-weight: bold;
    color: red; /* Color del texto */
}

.dealer-score {
    position: absolute;
    top: 2rem; /* Ajusta este valor según la altura de las cartas */
```

```
    left: 6rem; /* Ajusta este valor según la cantidad total de desplazamiento
        horizontal de las cartas */
    font-size: 2.5rem;
    font-weight: bold;
    color: red; /* Color del texto */
}

/* Estilo para cada carta con su desplazamiento y z-index del centro a la
    izquierda*/
#card-1 {
    z-index: 1; /* La más baja en la pila */
    top: 0;
    left: 0;
}

#card-2 {
    z-index: 2; /* Encima de 2H */
    top: -1rem; /* Desplazamiento hacia arriba */
    left: 1rem; /* Desplazamiento hacia la derecha */
}

#card-3 {
    z-index: 3; /* Encima de 2S */
    top: -2rem; /* Desplazamiento mayor hacia arriba */
    left: 2rem; /* Desplazamiento mayor hacia la derecha */
}

#card-4 {
    z-index: 4; /* La más alta en la pila */
    top: -3rem; /* Desplazamiento mayor hacia arriba */
    left: 3rem; /* Desplazamiento mayor hacia la derecha */
}

#card-5 {
    z-index: 5; /* La más alta en la pila */
    top: -4rem; /* Desplazamiento mayor hacia arriba */
    left: 4rem; /* Desplazamiento mayor hacia la derecha */
}

/* Estilo para cada carta con su desplazamiento y z-index del crupier*/
#dealer-1-card {
    z-index: 1; /* La más baja en la pila */
    top: 0;
    left: 0;
}

#dealer-2-card {
    z-index: 2; /* Encima de 2H */
    top: 1rem; /* Desplazamiento hacia arriba */
    left: -1rem; /* Desplazamiento hacia la derecha */
}

#dealer-3-card {
    z-index: 3; /* Encima de 2S */
    top: 2rem; /* Desplazamiento mayor hacia arriba */
    left: -2rem; /* Desplazamiento mayor hacia la derecha */
```

```
}

#dealer-4-card {
    z-index: 4; /* La más alta en la pila */
    top: 3rem; /* Desplazamiento mayor hacia arriba */
    left: -3rem; /* Desplazamiento mayor hacia la derecha */
}

#dealer-5-card {
    z-index: 5; /* La más alta en la pila */
    top: 4rem; /* Desplazamiento mayor hacia arriba */
    left: -4rem; /* Desplazamiento mayor hacia la derecha */
}

#statistics-button {
    position: absolute;
    top: 10%;
    left: 12%;
}

.card-slot .action-buttons {
    position: fixed !important;
    top: 85%;
    left: 37.5%;
}

.user-icon {
    position: absolute;
    width: 2.5rem;
    height: 2.5rem;
    margin-left: 5px;
    margin-bottom: 5px;
}

.dealer-name {
    position: absolute;
    top: 2rem; /* Ajusta este valor según la altura de las cartas */
    left: -7rem; /* Ajusta este valor según la cantidad total de desplazamiento
        horizontal de las cartas */
}

.dealer-icon {
    position: absolute;
    width: 4rem;
    height: 4rem;
}

.results {
    color: rgb(29, 255, 255) !important;
    font-weight: 600 !important;
    font-size: 1rem;
    position: absolute;
    top: 12% !important;
    left: 65%;
}
.action-buttons-round {
```

```
    position: absolute;
    top: 85% !important;
    left: 37.5% !important;
}

/* Estilos para el botón de volver al menú principal en la creación de los sets
     de datos y modelos */
.home {
    position: absolute;
    top: 5%;
    left:90%;
}

#home {
    position: absolute;
    width: 2.5rem;
    height: 2.5rem;
    margin: 5px; /* Espacio opcional alrededor de la imagen */
    border-radius: 5px; /* Opcional, redondea las esquinas de la imagen */
    box-shadow: 0 0 5px rgb(0, 0, 0); /* Sombra para la imagen */
}

a.link {
    color: blue !important; /* Color del texto azul */
    text-decoration: none; /* Quitar subrayado por defecto */
    cursor: pointer; /* Cambiar cursor a modo apuntador */
    transition: color 0.3s ease; /* Transición suave para cambio de color */
}

a.link:hover {
    color: darkblue; /* Oscurece el color al pasar el mouse */
}
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Jose Mariano Alvarez, *El perceptrón como neurona artificial - blog de jose mariano alvarez*, *https://blog.josemarianoalvarez.com/2018/06/10/el-perceptron-como-neurona-artificial/*, 2018, En línea. Consulta: 24-03-2024.

[2] Juan Ignacio Barrios Arce, *La matriz de confusión y sus métricas – inteligencia artificial*, *https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/*, 2019, En línea. Consulta: 22-03-2024.

[3] AWS, *¿qué es el ajuste de hiperparámetros? - explicación de los métodos de ajuste de hiperparámetros - aws*, *https://aws.amazon.com/es/what-is/hyperparameter-tuning/*, 2019, En línea. Consulta: 13-03-2024.

[4] ———, *¿qué es el aprendizaje mediante refuerzo? - explicación del aprendizaje mediante refuerzo - aws*, *https://aws.amazon.com/es/what-is/reinforcement-learning/*, 2024, En línea. Consulta: 13-03-2024.

[5] baeldung, *Epsilon-greedy q-learning | baeldung on computer science*, *https://www.baeldung.com/cs/epsilon-greedy-q-learning*, 2023, En línea. Consulta: 09-04-2024.

[6] Anjali Bhardwaj, *What is a perceptron? – basics of neural networks | by anjali bhardwaj | towards data science*, *https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590*, 2020, En línea. Consulta: 13-03-2024.

[7] Weights & Biases, *Weights & biases: The ai developer platform*, *https://wandb.ai/site*, 2024, En línea. Consulta: 09-07-2024.

[8] Justin Bodnar, *Artificial intelligence in blackjack card counting*, *https://github.com/justinbodnar/blackjack-ai/blob/master/Deck.py*, 2019, En línea. Consulta: 14-03-2024.

[9] Brilliant.org, *Backpropagation | brilliant math & science wiki*, *https://brilliant.org/wiki/backpropagation/*, 2024, En línea. Consulta: 13-03-2024.

[10] Vitaly Bushaev, *How do we 'train' neural networks ? | by vitaly bushaev | towards data science*, *https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73*, 2017, En línea. Consulta: 13-03-2024.

[11] Fernando Sancho Caparrini, *Aprendizaje por refuerzo: algoritmo q learning*, *https://www.cs.us.es/~fsancho/Blog/posts/Aprendizaje_por_Refuerzo_Q_Learning.md*, 2019, En línea. Consulta: 09-04-2024.

[12] Fernando Cardellino, *Tutorial para un clasificador basado en bosques aleatorios: cómo utilizar algoritmos basados en árboles para el aprendizaje automático*, https://www.freecodecamp.org/espanol/news/random-forest-classifier-tutorial-how-to-use-tree-based-algorithms-for-machine-learning/, 2021, En línea. Consulta: 16-03-2024.

[13] Comet, *Comet ml - build better models faster*, https://www.comet.com/site/, 2024, En línea. Consulta: 09-07-2024.

[14] European Commission, *A definition of ai: Main capabilities and scientific disciplines*, https://ec.europa.eu/futurium/en/system/files/ged/ai_hleg_definition_of_ai_18_december_1.pdf (2018), En línea. Consulta: 13-03-2024.

[15] Django, *Django documentation | django documentation | django*, https://docs.djangoproject.com/en/5.0/, 2024, En línea. Consulta: 14-07-2024.

[16] Niklas Donges and Jessica Powers, *What is gradient descent? | built in*, https://builtin.com/data-science/gradient-descent, 2023, En línea. Consulta: 13-03-2024.

[17] DotCSV, *¿qué es el descenso del gradiente? algoritmo de inteligencia artificial | dotcsv - youtube*, https://www.youtube.com/watch?v=A6FiCDoz8_4, 2018, En línea. Consulta: 24-03-2024.

[18] ———, *¿qué es el machine learning? ¿y deep learning? un mapa conceptual | dotcsv - youtube*, https://www.youtube.com/watch?v=KytW151dpqU, 2018, En línea. Consulta: 24-03-2024.

[19] ———, *¿qué es una red neuronal? parte 3 : Backpropagation | dotcsv - youtube*, https://www.youtube.com/watch?v=eNIqz_noix8, 2019, En línea. Consulta: 24-03-2024.

[20] ———, *¡redes neuronales convolucionales! ¿cómo funcionan? - youtube*, https://www.youtube.com/watch?v=V8j1oENVz00, 2021, En línea. Consulta: 24-03-2024.

[21] Temas | Parlamento Europeo, *Inteligencia artificial: oportunidades y desafíos | temas | parlamento europeo*, https://www.europarl.europa.eu/topics/es/article/20200918STO87404/inteligencia-artificial-oportunidades-y-desafios, 2022, En línea. Consulta: 13-03-2024.

[22] FastAI, *fastai | welcome to fastai*, https://docs.fast.ai/, 2024, En línea. Consulta: 09-07-2024.

[23] freeCodeCamp.org, *Introducción a q-learning: aprendizaje por refuerzo*, https://www.freecodecamp.org/espanol/news/introduccion-a-q-learning-aprendizaje-por-refuerzo/, 2023, En línea. Consulta: 09-04-2024.

[24] Les. Golden, *Never split tens! a biographical novel of blackjack game theorist edward o. thorp plus tips and techniques to help you win*, 1st ed. 2017. ed., Springer International Publishing, 2017.

[25] Ligdi Gonzalez, *Regresión logística - teoría - aprende ia*, https://aprendeia.com/algoritmo-regresion-logistica-machine-learning-teoria/, 2019, En línea. Consulta: 15-03-2024.

[26] Muhammad Hamdan, *Vhdl auto-generation tool for optimized hardware acceleration of convolutional neural networks on fpga (vgt)*, https://www.researchgate.net/figure/Activation-Functions-ReLU-Tanh-Sigmoid_fig4_327435257, 2018, En Línea. Consulta: 24-03-2024.

[27] IBM, *What is linear regression? | ibm*, https://www.ibm.com/topics/linear-regression, 2024, En línea. Consulta: 13-03-2024.

[28] _____, *What is logistic regression? | ibm*, https://www.ibm.com/topics/logistic-regression, 2024, En línea. Consulta: 13-03-2024.

[29] _____, *¿qué es el algoritmo de k vecinos más cercanos? | ibm*, https://www.ibm.com/es-es/topics/knn, 2024, En línea. Consulta: 13-03-2024.

[30] _____, *¿qué es el aprendizaje no supervisado? | ibm*, https://www.ibm.com/es-es/topics/unsupervised-learning, 2024, En línea. Consulta: 13-03-2024.

[31] _____, *¿qué es el aprendizaje supervisado? | ibm*, https://www.ibm.com/es-es/topics/supervised-learning, 2024, En línea. Consulta: 13-03-2024.

[32] _____, *¿qué es la simulación montecarlo? | ibm*, https://www.ibm.com/es-es/topics/monte-carlo-simulation, 2024, En línea. Consulta: 09-04-2024.

[33] _____, *¿qué es un bosque aleatorio? | ibm*, https://www.ibm.com/es-es/topics/random-forest, 2024, En línea. Consulta: 13-03-2024.

[34] _____, *¿qué es un árbol de decisión? | ibm*, https://www.ibm.com/es-es/topics/decision-trees, 2024, En línea. Consulta: 13-03-2024.

[35] José Ignacio Illana, *Métodos monte carlo*, https://ugr.es/ jillana/Docencia/FM/mc.pdf (2013).

[36] Javatpoint, *Gini index in machine learning - javatpoint*, https://www.javatpoint.com/gini-index-in-machine-learning, 2024, En línea. Consulta: 13-03-2024.

[37] Shashank Kapadia, *5 data similarity metrics: A comprehensive guide on similarity metrics | towards data science*, https://towardsdatascience.com/5-data-similarity-metrics-f358a560855f, 2022, En línea. Consulta: 17-03-2024.

[38] Keras, *Keras: Deep learning for humans*, https://keras.io/, 2024, En línea. Consulta: 09-07-2024.

[39] Will Koehrsen, *Overfitting vs. underfitting: A complete example | by will koehrsen | towards data science*, https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765, 2018, En línea. Consulta: 13-03-2024.

[40] Ajitesh Kumar, *K-nearest neighbors (knn) python examples*, https://vitalflux.com/k-nearest-neighbors-explained-with-python-examples/, 2022, En línea. Consulta: 16-03-2024.

[41] Bell Labs and Allen Rilley, *Claude shannon demonstrates "theseus" machine learning @ bell labs (high quality) - youtube*, https://www.youtube.com/watch?v=_9_AEVQ_p74, 2019, En línea. Consulta: 22-03-2024.

[42] Disfruta las Matemáticas, *Hallar máximos y mínimos usando derivadas*, https://www.disfrutalasmatematicas.com/calculo/maximos-minimos.html, 2020, En línea. Consulta: 16-03-2024.

[43] Scikit Learn, *scikit-learn: machine learning in python — scikit-learn 1.5.1 documentation*, https://scikit-learn.org/stable/index.html, 2024, En línea. Consulta: 09-07-2024.

[44] Darío León, *Support vector machine*, https://rstudio-pubs-static.s3.amazonaws.com/570352_e34015b16f1a47e883e04c6195d4711f.html, 2024, En línea. Consulta: 13-03-2024.

[45] Math4all, *Las matemáticas del blackjack | math4all*, https://www.math4all.es/las-matematicas-del-blackjack/#estudio, 2019, En línea. Consulta: 13-03-2024.

[46] MathWorks, *Aprendizaje supervisado - matlab & simulink*, https://es.mathworks.com/discovery/supervised-learning.html, 2024, En línea. Consulta: 13-03-2024.

[47] _____, *Introducción a reinforcement learning - matlab & simulink*, https://es.mathworks.com/discovery/reinforcement-learning.html, 2024, En línea. Consulta: 13-03-2024.

[48] _____, *Sobreajuste - matlab & simulink*, https://es.mathworks.com/discovery/overfitting.html, 2024, En línea. Consulta: 16-03-2024.

[49] _____, *Support vector machine (svm) - matlab & simulink*, https://es.mathworks.com/discovery/support-vector-machine.html, 2024, En línea. Consulta: 13-03-2024.

[50] _____, *Unsupervised learning - matlab & simulink*, https://es.mathworks.com/discovery/unsupervised-learning.html, 2024, En línea. Consulta: 13-03-2024.

[51] Marcos Merino, *Conceptos de inteligencia artificial: qué son las gans o redes generativas antagónicas*, https://www.xataka.com/inteligencia-artificial/conceptos-inteligencia-artificial-que-gans-redes-generativas-antagonicas, 2019, En línea. Consulta: 16-03-2024.

[52] Sandra Navarro, *¿qué es el método k folds? | keepcoding bootcamps*, https://keepcoding.io/blog/que-es-el-metodo-k-folds/, 2024, En línea. Consulta: 13-03-2024.

[53] NeptuneAI, *neptune.ai | the mlops stack component for experiment tracking*, https://neptune.ai/, 2024, En línea. Consulta: 09-07-2024.

[54] OpenAI, *Introducing chatgpt - openai*, https://openai.com/index/chatgpt/, 2024, En línea. Consulta: 09-07-2024.

[55] _____, *Openai*, https://openai.com/, 2024, En línea. Consulta: 05-02-2024.

[56] PokerStars, *Historia y evolución del blackjack desde sus orígenes - pokerstars casino blog historia del blackjack y sus variantes | pokerstars casino*, https://www.pokerstars.es/casino/news/historia-y-evolucion-del-blackjack-desde-sus-origenes/2596/, 2023, En línea. Consulta: 22-03-2024.

[57] _____, *La historia del blackjack*, https://www.pokerstars.es/casino/news/la-historia-del-blackjack-como-surgio-y-se-popularizo-este-juego-legendario/2081/, 2023, En línea. Consulta: 22-03-2024.

[58] _____, *Reconocer y leer los tell de un crupier de blackjack - pokerstars casino blog*, https://www.pokerstars.es/casino/news/reconocer-y-leer-los-tell-de-un-crupier-de-blackjack/2606/, 2023, En línea. Consulta: 22-03-2024.

[59] _____, *Guía para aprender las reglas básicas del blackjack - pokerstars casino blog guía básica para principiantes de blackjack online | pokerstars casino*, https://www.pokerstars.es/casino/news/guia-para-aprender-las-reglas-basicas-del-blackjack/2750/, 2024, En línea. Consulta: 22-03-2024.

[60] _____, *¿importan las posiciones en la mesa de blackjack? - pokerstars casino blog*, https://www.pokerstars.es/casino/news/importan-las-posiciones-en-la-mesa-de-blackjack/2727/, 2024, En línea. Consulta: 22-03-2024.

[61] prakharr0y, *What is adam optimizer? - geeksforgeeks*, https://www.geeksforgeeks.org/adam-optimizer/, 2024, En línea. Consulta: 05-06-2024.

[62] Mentes Abiertas Psicología, *El condicionamiento clásico de ivan pavlov - psicólogos a tu alcance en madrid capital - mentes abiertas psicología*, https://www.mentesabiertaspsicologia.com/blog-psicologia/blog-psicologia/el-condicionamiento-clasico-de-ivan-pavlov, 2024, En línea. Consulta: 13-03-2024.

[63] Adrià Puigdomènech, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell, *Agent57: Outperforming the human atari benchmark - google deepmind*, https://deepmind.google/discover/blog/agent57-outperforming-the-human-atari-benchmark/, 2020, En línea. Consulta: 13-03-2024.

[64] PyTorch, *Pytorch*, https://pytorch.org/, 2024, En línea. Consulta: 09-07-2024.

[65] S4PCADEMY, *Back propagation: Algoritmo de aprendizaje automático en redes neuronales - s4pcademy*, https://s4pcademy.com/back-propagation-algoritmo-de-aprendizaje-automatico-en-redes-neuronales/, 2022, En línea. Consulta: 16-03-2024.

[66] Pol Martí Sanahuja, *Entendiendo la curva roc y el auc: Dos medidas del rendimiento de un clasificador binario que van de la mano. – pol martí sanahuja*, https://polmartisanahuja.com/entendiendo-la-curva-roc-y-el-auc-dos-medidas-del-rendimiento-de-un-clasificador-binario-que-van-de-la-mano/, 2021, En línea. Consulta: 13-03-2024.

[67] Javier Santaolalla, *¿en qué consiste el método montecarlo? - youtube*, https://www.youtube.com/watch?v=WJjDr67frtM, 2015, En línea. Consulta: 09-04-2024.

[68] Tarang Shah, *Train, validation and test sets*, https://tarangshah.com/blog/2017-12-03/train-validation-and-test-sets/, 2017, En línea. Consulta: 16-03-2024.

[69] Tavish Srivastava, *11 essential evaluation metrics for evaluating ml models*, https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/, 2024, En línea. Consulta: 13-03-2024.

[70] Great Learning Team, *Types of neural networks and definition of neural network*, https://www.mygreatlearning.com/blog/types-of-neural-networks/, 2024, En línea. Consulta: 13-03-2024.

[71] Kili Technology, *The differences between training, validation & test datasets*, https://kili-technology.com/training-data/training-validation-and-test-sets-how-to-split-machine-learning-data, 2024, En línea. Consulta: 13-03-2024.

[72] TensorBoard, *Tensorboard | tensorflow*, https://www.tensorflow.org/tensorboard?hl=es-419, 2024, En línea. Consulta: 09-07-2024.

[73] TensorFlow, *Tensorflow*, https://www.tensorflow.org/?hl=es-419, 2024, En línea. Consulta: 09-07-2024.

[74] Playground TensorFlow, *A neural network playground*, https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.70795&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false, 2024, En línea. Consulta: 24-03-2024.

[75] A M Turing, *M i n d a quarterly review of psychology and philosophy i.-computing machinery and intelligence*, (1950).

[76] upGrad, *Neural network: Architecture, components & top algorithms | upgrad blog*, *https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/*, 2022, En línea. Consulta: 16-03-2024.

[77] William Vincent, *Introduction | django for beginners*, *https://djangoforbeginners.com/introduction/*, 2024, En línea. Consulta: 14-07-2024.

[78] N. Richard. Werthamer, *Risk and reward the science of casino blackjack*, 1st ed. 2009. ed., Springer New York, 2009.

[79] Wikipedia, *Alan turing*, *https://es.wikipedia.org/wiki/Alan_Turing*, 2024, En línea. Consulta: 13-03-2024.

[80] Juan Tejada Cazorla y Javier Yañez Gestoso, *Estudio de la estrategia óptima para el blackjack*, https://docta.ucm.es/rest/api/core/bitstreams/3691e997-8980-48d3-9a6d-ed2dd8b5eb6d/content (2023).