Smart Blackjack: Aplicación de técnicas de aprendizaje para soporte en juegos de azar

Ivan Gerun

Resumen— El empleo de aprendizaje computacional como soporte a los juegos de azar plantea diferentes retos. Uno de ellos es poder sobreponerse a la aleatoriedad de ciertas circunstancias de las partidas. Para ello, es necesario definir adecuadamente las características que se van a usar en el aprendizaje y seleccionar cuidadosamente el método de clasificación. Este proyecto presenta un primer sistema de soporte en las partidas de Blackjack, donde el jugador se enfrenta al crupier cuyas cartas son desconocidas. Para desarrollar el proyecto, se han seleccionado 5 algoritmos de aprendizaje computacional y se han definido features basadas en el conocimiento de la partida actual y de la baraja de juego. Con el fin de poder testar el sistema desarrollado, se ha diseñado una interfaz gráfica de usuario que permite asimismo guardar las partidas jugadas para crear así una base de datos de pruebas. Se han comparado los resultados obtenidos por los métodos computacionales con los proporcionados por estrategias de juego que representan el comportamiento humano. Los resultados muestran que el uso de sistemas de soporte conlleva un incremento en el número de partidas ganadas, siendo éste incremento mayor para el algoritmo kNN.

Palabras claves— Sistemas de soporte, Juegos de azar, Aprendizaje computacional, Blackjack, kNN, Árboles de decisión, Boosting

Abstract— The use of machine learning to support to gambling poses different challenges. One of them is that support systems should be able to features the randomness of certain circumstances of the games. To cope with this, it is necessary to define adequately the characteristics that are going to be used in learning stage and to carefully select the classification method. This project introduces a support system to support player in Blackjack game. In this game, the player competes against the crupier, whose cards are unknown. To develop the project, five different machine learning algorithms have been selected and features based on the knowledge of the current status of the game and the game deck have been defined. A graphical user interface has been designed to test the system. This interface lso allows saving the games already played to create a test database. We have compared the results obtained by computational methods with those provided by game strategies that represent human behavior. The results show that the use of support systems entails an increase in the number of games won, this increase being greater for the kNN algorithm.

Index Terms— Support systems, Gambling, Machine learning, Blackjack, kNN, Decision trees, Boosting

1 Introducción

HOY en día tres de cada cuatro personas han jugado a juegos de azar a lo largo de su vida [1]. Pero no todos los juegos son iguales, existen dos tipos bien diferenciados. En juegos como la ruleta el jugador sólo puede elegir la cantidad de dinero y la apuesta concreta, el resto depende del azar. El segundo tipo de juego corresponde a aquél donde las posibilidades de ganar o perder no dependen sólo del azar sino también de la habilidad del jugador; la mayoría de estos juegos son de cartas, tales como Póker o Blackjack [2].

Los juegos de azar también se pueden categorizar según el conocimiento de los jugadores respecto del estado del juego. Si todos los jugadores tienen un conocimiento completo sobre todo el estado del juego, se llama un juego con información completa [3]. De lo contrario, es un juego con información incompleta. Por ejemplo, el ajedrez es un juego de información completa porque todos los jugadores tienen acceso a todo el estado del juego. Por otro lado, Poker o Blackjack son ejemplos de juegos de información incompleta, porque cada jugador tiene sus propias cartas que se mantienen ocultas. En nuestro caso, nos centramos en éste segundo tipo de juegos en los que los jugadores toman decisiones basadas en conocimientos incompletos.

Es en este punto donde creemos que la inteligencia artificial puede tener un papel de soporte, empleándose para crear un sistema de ayuda que permita predecir la mejor acción a tomar para sacar el máximo partido.

El objetivo principal de este trabajo es desarrollar un programa para asistencia en juegos de azar. Para poder aplicar este sistema, se decidió elegir el juego BlackJack.

El Blackjack o veintiuno, es un juego que consiste en sumar sin pasarse un valor lo más próximo a 21. Se juega con las 52 cartas de la baraja. Cada carta tiene una puntuación diferente: en orden de mayor a menor los valores de las cartas son siguientes: las cartas del 2 al 10 tienen el mismo valor real que nominal mientras que las cartas J, Q, K valen 10 y el AS vale 1 u 11 dependiendo de lo que convenga al jugador. El jugador juega únicamente contra el crupier, intentando ganar en una de las siguientes maneras.

- E-mail de contacto: ivan.gerun@e-campus.uab.cat
- Mención realizada: Computación
- Trabajo tutorizado por: Jorge Bernal (Ciencias de la Computación)

1. Si con las dos primeras cartas (la inicial y la primera que roba) la suma es 21, se consigue la denominada como jugada máxima "BlackJack". En cualquier otro caso, la suma de 21 ya no es BlackJack.
2. Alcanzar una puntuación final mayor que la del crupier, pero sin superar 21.
3. Si jugador y crupier tienen la misma puntuación empatan.

Para cada turno el jugador tiene dos opciones, robar una carta o plantarse, la idea de este juego es saber el mejor momento para plantar.

El trabajo a desarrollar conlleva tanto el estudio del rendimiento de diferentes algoritmos de aprendizaje como su posterior análisis. Para la selección de algoritmos se realizó una búsqueda basada en las funcionalidades esperadas de la aplicación de soporte. La adición del sistema de soporte a la partida implica la programación del juego con su interfaz gráfica del usuario.

Los objetivos secundarios del proyecto son:

- Desarrollo de una interfaz gráfica que integre todas las funcionalidades necesarias para poder jugar las partidas contra el crupier. El juego tendrá todos los controles para activar el sistema de ayuda de partidas y seleccionar el perfil deseado, donde cada perfil representará diferentes algoritmos de aprendizaje.
- Creación de una base de datos de partidas jugadas para entrenar el sistema según el perfil seleccionado. Se realizarán las pruebas con cada uno de los perfiles utilizando las mismas condiciones para su futuro estudio y comparación de los resultados obtenidos.

El proyecto se ha dividido en cinco fases. Un punto muy importante es la planificación inicial del proyecto donde se han establecido los requisitos del proyecto junto con la organización temporal que se muestra en la Ilustración 1 del anexo. En la fase 2 se desarrolla la versión beta del juego, donde el jugador puede jugar contra el ordenador a través de modo consola, sin interfaz gráfica. Después del testeo y corrección de errores se integra la interfaz gráfica y se crea base de datos con los experimentos iniciales. Una vez finalizada esta fase, en la fase 3 se definieron los diferentes algoritmos de aprendizaje computacional a testear, así como las features necesarias. En la fase 4, se realizan las pruebas y testeo y se corrigen los errores encontrados a partir del análisis de los resultados.

A continuación, se detallarán las secciones de este trabajo. En el apartado **Estado del arte** se muestra un resumen de proyectos similares que se han consultado de cara a reforzar el planteamiento de la solución propuesta. En el apartado **Metodología**, se explicarán las diferentes fases que ha tenido el desarrollo del sistema de soporte, incluyendo el desarrollo de la interfaz. En la sección **Experimentos**, se explica la estructura y el contenido de la base de datos y las métricas utilizadas. En **Resultados**, se muestra y analiza el rendimiento de las diferentes configuraciones del sistema de ayuda propuesto. Por último, en **Conclusiones**, se hará un análisis global del proyecto, las dificultades encontradas y se esboza el trabajo futuro.

2 ESTADO DEL ARTE

La creación de sistemas de soporte ha atraído el interés de la inteligencia artificial desde hace varias décadas. Muchas historias de éxito Chinook (Damas) [4], Logistello (Otelo) [5], Deep Blue [6] y Hydra [7] (Ajedrez) y Maven (Scrabble) [8] han demostrado que programas informáticos pueden superar en habilidad a los jugadores humanos. Sin embargo, aún quedan muchos desafíos en los juegos de información incompleta.

Como primer paso, se ha realizado una búsqueda de juegos de azar que utilizan la Inteligencia Artificial para el aprendizaje y predicción. No se ha encontrado ningún trabajo concreto que trate de la implementación del BlackJack con las funcionalidades deseadas. Por ese motivo, se han buscado diferentes implementaciones de otros juegos de cartas que incluyan un sistema de ayuda al jugador para obtener la ventaja sobre el ordenador u otros jugadores.

Por ejemplo, respecto al **Poker Texas Hold'em** [9] el trabajo consultado se centraba en desarrollar varios agentes simulando el comportamiento típico de un jugador de Poker y otro agente que era capaz de utilizar técnicas de modelado del oponente para seleccionar la mejor estrategia de juego contra cada oponente. Se ha visto que para modelado de oponente utilizan redes neuronales. También, se ha observado la forma en que se diferencian cada uno de los modelos y la estructura del juego utilizada.

Otro ejemplo es el trabajo sobre el juego **Doppelkop** [10], uno de los juegos de cartas alemanes más populares. El trabajo propuesto por los autores se centra en una implementación de funcionalidades inteligentes que combina el algoritmo Classical Upper Confidence (UCT) y árboles de decisión con un tipo especial de redes neuronales recurrentes, Long Short-Term Memory (LSTM). Como resultado el sistema LSTM predijo la siguiente carta en el juego, pero la combinación con UCT logró una mejora de solo 0,04 puntos por juego en promedio contra el mismo jugador UCT sin LSTM, esto indica que la combinación de ambos algoritmos no ha mejorado.

Por último, mostramos el trabajo realizado para soporte al juego **Scopone** [11], que es un juego de cartas muy popular en Italia. Se centraba en los algoritmos de Monte Carlo Tree Search (MCTS) y Information Set Monte Carlo Tree Search (ISMCTS), el primero está pensado para obtener un conocimiento completo del estado del juego, y el segundo para información incompleta. Se explica el proceso de desarrollo de diferentes mejoras para cada uno de los algoritmos y la adaptación al juego Scopone. Se compara con el algoritmo clásico Minimax, que explora por completo el árbol de búsqueda. Se demuestra que los jugadores con la ayuda de IA tienen ventaja sobre los jugadores sin este tipo de ayuda y que los algoritmos de MCTS son muy eficientes en términos de tiempo y memoria.

3 METODOLOGÍA

En este apartado se explicará el desarrollo del proyecto en su completitud. Respecto a la herramienta de desarrollo, se ha optado por utilizar Matlab debido a su facilidad de uso y el conocimiento previo de la misma, lo que nos permite centrarnos en el desarrollo del algoritmo sin preocuparnos de tipos de las variables o tamaño de las mismas.

Para realización de este proyecto se ha seguido la planificación inicial, donde después de cada tarea finalizada se hacía una reunión con el tutor, normalmente cada tarea duraba una semana. Estas reuniones eran muy importantes para el aprendizaje ya que mejoraba de manera continua el trabajo, se comentaba el avance y se realizaban las modificaciones necesarias. Gracias a esto, el trabajo avanzaba según las fases y etapas establecidas.

3.0 Background y conceptos básicos de Blackjack

En todos los casinos o juegos online, el Blackjack es un juego sobre el cual se realizan apuestas. La idea básica es saber en qué momento es mejor apostar para ganar. En nuestro caso dejamos de lado el aspecto económico del juego y nos centramos solamente en las cartas, con el objetivo de incrementar el número de partidas ganadas.

Es importante tener en cuenta el rol del crupier en el juego: él es el que tiene la baraja y reparte las cartas. El crupier nunca se arriesga y si consigue 17 puntos no roba más cartas. El proceso del juego es el siguiente: el crupier reparte dos cartas, a sí mismo y al jugador. El jugador ve sus cartas y una carta del crupier, la otra está boca abajo. Las cartas se giran en el momento en que el jugador sobrepasa 21 puntos o decide plantarse. Cuando las cartas están giradas y el crupier no alcanza 17 puntos, éste roba más cartas.

Para aumentar la posibilidad de ganar en Blackjack se emplean sistemas de conteo de cartas. Estos sistemas consisten en determinar con antelación la siguiente mano o la siguiente carta a robar a partir de la información obtenida de las cartas ya jugadas y visibles. El sistema cuenta las cartas altas y bajas que van apareciendo. Existen diferentes sistemas de conteo, entre ellas hay formas de conteo simples y más complejas. Teniendo en cuenta que el conteo de cartas lo realizará el sistema automático, se ha optado por un sistema complejo llamado Uston SS [12] que nos aportará más información sobre el estado de la baraja. Uston SS es un sistema de conteo de cartas corriente, un conteo positivo (+1, +2, +3) indica al jugador que la baraja está caliente y es favorable para el jugador. La idea principal del conteo es saber en qué momento hay que subir la apuesta, esto significa que podemos controlar más específicamente el estado de la baraja para saber en qué momento es mejor robar carta.

TABLA 1: LOS VALORES DE LAS CARTAS EN EL SISTEMA DE CONTEO DE CARTAS USTON SS

Carta	**2,3,4,6**	**5**	**7**	**8**	**9**	**10,J,Q,K,A**
Valor	+2	+3	+1	0	-1	-2

3.1 Algoritmos de aprendizaje computacional

Para nuestro sistema de soporte utilizamos diferentes algoritmos de entrenamiento y clasificación. El juego nos permite seleccionar cualquier perfil, donde el nombre de cada perfil indica el algoritmo utilizado.

- En el algoritmo **K-Nearest Neighbors** (kNN) los ejemplos de entrenamiento son tratados como vectores en un espacio de características, cada uno con una etiqueta que le relaciona con una clase. La fase de entrenamiento del algoritmo consiste en almacenar los vectores de características y las etiquetas de las clases de las muestras de entrenamiento. En la fase de clasificación la k es una constante definida por el usuario. Cada ejemplo nuevo se clasifica asignando la etiqueta más frecuente entre las k muestras más cercanas a ese punto [13]. La elección de la k óptima depende de los datos, si el número de k es elevado se reduce el efecto del ruido en la clasificación, pero hacen que los límites que separan las clases sean menos distintos [14].

- **Decision tree** (DT) predicen las respuestas a los datos. A partir de un conjunto de datos de entrenamiento y un vector de respuestas se genera un árbol de decisión, que está formado por nodos raíz, nodos hoja y flechas. Cada nodo raíz representa el momento en el que se ha de tomar una decisión, a más nodos raíz se han definido, más grande el árbol será. El nodo de hoja contiene la respuesta, que son nominales del estilo *true* o *false*. Las flechas son las uniones entre un nodo y otro [15]. Para controlar la profundidad de árbol su utiliza un parámetro *MaxNumSplits* que permite limitar el número de nodos raíz totales.

- **AdaBoostM2 (ABM2)** es un algoritmo de *boosting* para clasificación multiclase. Su objetivo es crear un clasificador robusto a partir de un conjunto de clasificadores débiles. Cada clasificador base tiene que minimizar la *pseudo-loss* siempre que sea menor que ½. Esto es fácilmente alcanzable para los clasificadores débiles, donde se garantiza una disminución exponencial de un límite superior en la tasa de errores de entrenamiento. Inicialmente todos los objetos del conjunto de entrenamiento tienen el mismo peso. En cada iteración se incrementa el peso de los objetos mal clasificados por el predictor en esa iteración, en la construcción del próximo predictor, estos objetos serán más importantes [16] [17] [18].

- **Random Subspace** (RS) es un algoritmo de clasificación multiclase donde cada miembro se entrena con todos los ejemplos, pero con un subconjunto de los atributos. La dimensión de estos subconjuntos es el parámetro del método, para la asignación de la clase se establece un sistema de voto mayoritario [19].

- Algoritmo de clasificación multiclase **RUSBoost** (random under-sampling) (RUS) se usa para mejorar el rendimiento del conjunto de datos capacitados, adquiridos a partir del conjunto de datos sesgados. En otras palabras, RUSBoost es uno de los métodos que mejor elimina los desequilibrios de

distribución de datos entre las clases y mejora el rendimiento de la clasificación de los clasificadores débiles. Es un algoritmo de datos híbridos sampling/boosting- Adicionalmente RUSBoost afronta a la extracción aleatoria de datos de modo que el método elimina aleatoriamente los datos del conjunto de entrenamiento hasta que se logra una distribución de clases idónea [20].

3.2 Estructuras de datos usadas

Para el entrenamiento de los diferentes algoritmos presentados, se han definido 8 features:

1. Número de partida: identificador de partida, permite saber si la acción realizada pertenece a la misma partida (no usada en entrenamiento).
2. Puntos del jugador: los puntos que tiene el jugador antes de realizar la acción de robar o los puntos que tiene en el momento de plantarse.
3. Puntos del crupier: los puntos de la carta visible.
4. Media de los puntos que quedan: cociente entre la suma de los puntos de las cartas que aún no se han jugado y el número de cartas que quedan por jugar.
5. Estado de la baraja: según el sistema de conteo de Cartas Uston SS.
6. Indicador si el usuario tiene un As en la mano: 1 – contiene, 0 – no contiene.
7. Acción realizada: 0 – planta, 1 – roba carta.
8. El resultado: 0 – pierde, 1 – gana, 2 – empata, 3 – no ha perdido (no usada en entrenamiento).

TABLA 2: EJEMPLOS DE CONTENIDO DE LA MATRIZ BD_ENTRENAMIENTO

1	19	10	7.4792	-3	0	0	1
2	13	7	7.7619	8	0	1	0
3	19	8	7.8056	13	0	0	1

La base de datos bd_sesion contiene una amplia información de cada partida jugada durante la sesión. Esto quiere decir que, cargando esta matriz, podemos recuperar las partidas guardas y volver a jugar las mismas partidas una y otra vez. El uso de esta función en el apartado de experimentos lo veremos más en detalle. La estructura que tiene esta matriz es la siguiente:

1. Número de partida: identificador de partida, permite identificar y controlar las partidas recuperadas.
2. Struct jugador: contiene toda la información del jugador (su mano, sus puntos) antes de realizar la acción de robar o plantar.
3. Struct crupier: contiene toda la información del crupier (su mano, sus puntos y la baraja) antes de realizar la acción de robar o plantar.
4. Acción realizada: 0 – planta, 1 – roba carta
5. El resultado: 0 – pierde, 1 – gana, 2 – empata, 3 – no ha perdido.

De toda esta información podemos destacar el contenido del struct del jugador, en la Tabla 3, observamos una matriz baraja, es muy importante, ya que contiene la información de las cartas que han salido, podemos hacer un paralelismo con la memoria del jugador. En cambio, el crupier también tiene una matriz baraja, pero en su caso es la baraja física y contiene las cartas a repartir.

TABLA 3: EJEMPLO DEL STRUCT JUGADOR

gan	1
per	0
emp	0
baraja	4x13 double
pt	20
count	4
mano	1x4 card

3.3 Desarrollo interfaz

La interfaz gráfica del sistema de soporte se ha diseñado para recordar al típico tablero del juego BlackJack. El desarrollo de esta interfaz se ha hecho mediante la herramienta GUIDE [21] de Matlab. Para facilitar un uso intuitivo de la misma por parte de un usuario no experto, se ha dividido la parte visual en cuatro secciones importantes, tal como se puede observar en la Figura 1.

A) En la parte superior se ubica el menú desplegable con dos botones funcionales: Archivo y Configuración. Dentro de Archivo se encuentran las siguientes opciones (Fig. 2):

- Cargar: permite cargar diferentes matrices (sesión de juego, bd_entrenamiento y bd_partidas) cargando la matriz sesión de juego nos permite volver a jugar todas las partidas de la sesión cargada. Cargando la matriz bd_entrenamiento, se carga la matriz de entrenamiento para su futuro uso en los clasificadores.
- Guardar sesión de juego: para facilitar las tareas de aprendizaje de nuestro sistema de soporte, se almacenan todas las jugadas con toda la información en cada etapa de la partida: baraja, mano del crupier, mano del jugar y los puntos.

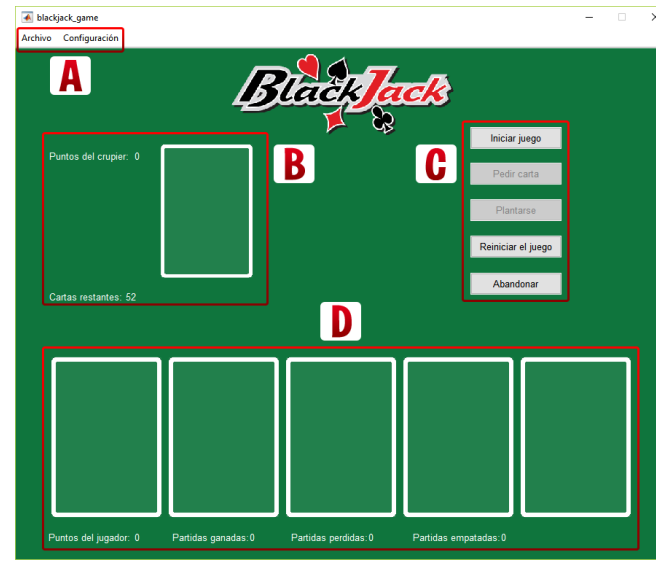

Fig. 1: Interfaz gráfica del juego

- Guardar bd_entrenamiento: en el caso de generar la matriz de entrenamiento nos permite almacenar dicha matriz para su futuro uso.
- Guardar bd_partidas: guarda todas las jugadas, indiferente si se juega de modo automático o manual, representa la matriz de test.
- Salir: cierra el juego.

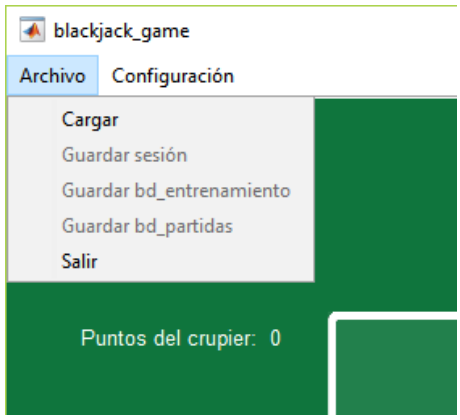

Fig. 2: Menú desplegable de Archivo.

Dentro de Configuración (Figura 3) se encuentran botones que permiten configurar el juego.

- Perfil: permite elegir diferentes perfiles al usuario, se puede cambiar el perfil en cualquier momento.
- Mostrar ayuda: un checkbox que activa la ayuda del sistema automático, mostrando la acción que debe tomar el usuario. Si previamente no se ha seleccionado el perfil, por defecto se activa con el perfil 1.
- Mostrar cartas: el checkbox que se utiliza en caso de autojuego, por defecto está activado, si se desactiva está opción el juego automático irá más rápido, ya que el proceso de mostrar cartas ralentiza el juego.
- Autojuego: el checkbox que se utiliza para poder jugar de forma automática. Funciona solo si previamente se ha cargado la sesión de juego.

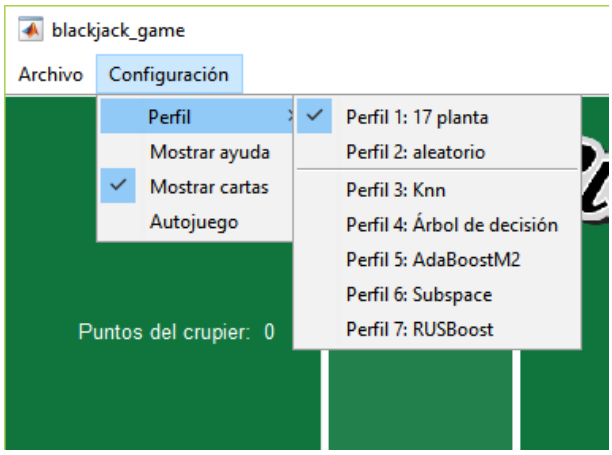

Fig. 3: Menú desplegable de Configuración.

B) Mano del crupier, en esta parte solo se muestra la primera carta del crupier junto con la puntuación de esta carta. Cuando se finaliza la partida, se muestran todas las cartas del crupier. Debajo también hay un indicador de cartas que quedan por salir en la partida.

C) Botones de acción del usuario: Iniciar partida, pedir carta, plantarse… Estos botones se activan/desactivan de forma automática, evitando así acciones no permitidas.

D) Mano del jugador, en esta parte aparecen todas las cartas del usuario, las cartas que se reparten al inicio y las que cogerá a lo largo del juego, además se muestra información adicional del jugador como: los puntos, partidas ganadas y etc.

En la figura 4 se puede observar el tablero con las cartas en la mesa. Donde hay cartas de la mano del jugador y la única carta visible del crupier. Como el modo de ayuda está activado, al jugador le indica con el color naranja sobre el botón que debe presionar el usuario.

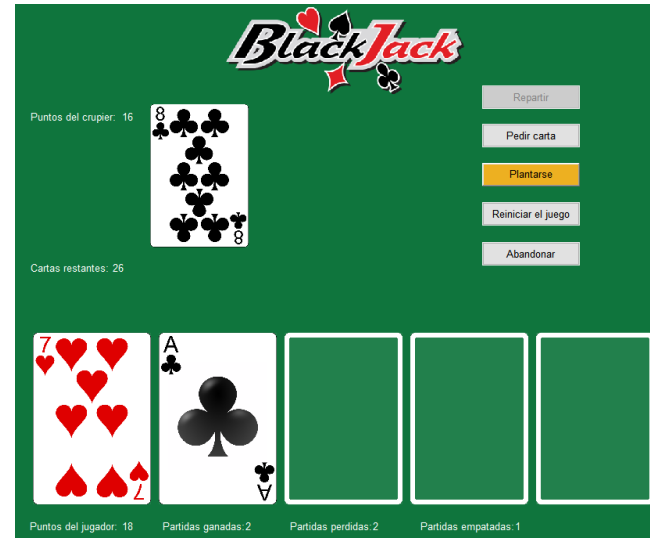

Fig. 4: Tablero del juego, con la ayuda activada.

4 Experimentos

Aunque al usuario se le deja elegir entre cualquier perfil, estamos interesados en saber cuál de ellos puede ser mejor a la hora de ayudar al jugador en la partida. Para poder realizar una comparación justa entre los algoritmos, se ha de preparar los datos de modo que todos estén evaluando la misma situación de juego. En los juegos de azar es complicado comparar las diferentes estrategias debido a que, en cada paso del juego, hay diferentes variables aleatorias (estado de la mano del jugador, la carta que roba). Esto dificulta una comparación exacta entre las diferentes soluciones propuestas. Para poder comparar con más precisión se ha creado un sistema que permite guardar dentro de una matriz bd_sesion estado de cada acción realizada para su futura carga y ejecución. Esto permite recuperar cada estado de la partida y volver a jugar con la ayuda de diferentes algoritmos teniendo el mismo estado de la baraja, de las manos del crupier y jugador. Aunque el estado antes de hacer la acción es idéntico, la acción siguiente sigue siendo aleatoria.

4.1 Base de datos de partidas

La base de datos bd_entrenamiento se utiliza para entrenar diferentes algoritmos de clasificación. Esta base de datos contiene 6000 partidas terminadas, aunque el número de registros almacenados asciende a 6393. Esto es debido a

que una partida a parte de tener el estado de gana o pierde, puede tener el estado "no ha perdido". Este estado se utiliza para el proceso de entrenamiento, ya que, sin él, el modelo entrenado en todas las ocasiones nos sugeriría plantarnos y nunca robaríamos una carta porque el único estado en caso de robar sería perder. Se ha introducido una condición para el estado "no perdido" de cara a ser almacenado en la base de datos: en este caso el jugador tiene que tener un número de puntos igual o superior a 17, robar una carta y no acabar perdiendo.
Para crear la base de datos de entrenamiento, se jugó de forma automática 6000 partidas. De estas partidas, 4000 son jugadas con el perfil 1, y las 2000 que quedan con una condición de plantar solo si el jugador tiene 21 puntos. La base de datos está compuesta por 6000 partidas, estas partidas están repartidas de la siguiente manera: 3588 son partidas ganadas, 1945 partidas perdidas y 467 empatadas.

Para realizar los experimentos se han jugado de forma automática 500 partidas con el perfil 1, este perfil está condicionado por plantar cuando llega a 17 puntos, de esta forma simulamos la función de crupier. Todas estas partidas las almacenamos dentro de la matriz bd_sesion, para poder ejecutar y hacer comparaciones de las mismas partidas, pero utilizando diferentes perfiles con algoritmos de clasificación. El resultado de esta ejecución lo veremos en el apartado de resultados.

4.2 Métricas de rendimiento

Para poder realizar comparaciones de cada perfil, se han establecido las siguientes métricas de comparación: número de partidas ganadas, número de partidas perdidas y número de partidas empatadas. Estos valores se dividen entre el total de partidas jugadas y se comparan con los obtenidos por otros clasificadores. Estos resultados se almacenan en una tabla para cada perfil. Utilizando estas métricas podemos evaluar los resultados y comparar las diferentes estrategias. Como no se puede evitar el factor aleatorio a 100%, ejecutamos cada prueba 5 veces y sacamos la media de cada uno de los resultados, esto nos permite acercarnos más a la realidad.

5 Resultados

5.1 Optimización de parámetros de los clasificadores

Para poder establecer la mejor configuración de cada algoritmo, se han probado diferentes parámetros a la hora de ejecutar las pruebas. Más adelante veremos cada uno de los algoritmos y los resultados obtenidos para diferentes configuraciones. La tabla 4 muestra los resultados obtenidos con el perfil kNN variando el valor de la k. Analizando los datos obtenidos, podemos ver como el porcentaje de las partidas ganadas es más o menos estable, menos el caso con la k igual a 7. Esto se debe a que tanto el modelo entrenado como los datos de test se adaptan mejor a los nuestros datos.

TABLA 4: ALGORITMO DE KNN CON LA K VARIADA

k	ganadas	perdidas	empatadas
3	44,6%	43,0%	12,4%
5	43,9%	43,4%	12,8%
7	**46,9%**	**41,6%**	11,6%
9	44,5%	42,4%	**13,1%**

Observando la tabla 5, vemos los resultados del algoritmo de árbol de decisión variando el número de nodos raíz (splits). Analizando los datos obtenidos, podemos concluir que en general los resultados son bastante buenos, pero utilizando 40 nodos raíz obtenemos el mejor resultado, por tanto, la configuración por defecto quedará con estos parámetros.

TABLA 5: ALGORITMO DE KNN CON LA K VARIADA

splits	ganadas	perdidas	empatadas
5	44,9%	**41,2%**	**13,9%**
15	44,8%	42,7%	12,5%
25	44,6%	42,5%	12,9%
30	45,8%	42,1%	12,2%
35	44,1%	43,3%	12,6%
40	**46,1%**	41,6%	12,2%
50	44,3%	42,8%	13,0%

En el caso del algoritmo AdaBoostM2 (ABM2) el único parámetro que se puede cambiar es *NumLearningCycles*, el cambio de número de ciclos de aprendizaje no afecta a los resultados, por tanto, se ha establecido el parámetro que viene por defecto a 100.
Para el caso del algoritmo Random Subspace (RS) podemos elegir entre dos tipos de aprendizaje: kNN o *Discriminant*, En la tabla 6, observamos que el algoritmo RS usando el tipo débil de apéndice *Discriminant* da mejores resultados. Además, se ha visto que usando kNN la predicción de decisión tarda más que con *Discriminant* y por tanto la ejecución de 500 partidas es mucho más lenta, 114,01s contra 84,25s respectivamente.

TABLA 6: ALGORITMO DE RANDOM SUBSPACE

learner	ganadas	perdidas	empatadas
kNN	44,4%	42,5%	**13,4%**
discriminant	**45%**	**42,4%**	12,6%

5.2 Comparativa entre clasificadores

Ahora que ya tenemos las mejores configuraciones de cada perfil, mostramos los resultados de la comparativa en la tabla 6. "17 planta" es el perfil con el que hemos creado la base de datos de testeo. Respecto al perfil de decisión aleatoria, el resultado obtenido es el esperado, lo cual indica que en los juegos de azar algo de inteligencia (aunque sea humana) va bien para ganar. El perfil con KNN, ha obtenido el mejor resultado en comparación con los demás.

TABLA 7: RESULTADOS DE TODOS LOS PERFILES

perfil	ganadas	perdidas	empatadas
17 planta	37,6%	52,2%	10,2%
aleatorio	32,4%	60,6%	7,0%
kNN	**46,9%**	**41,6%**	11,6%
DT	46,1%	**41,6%**	12,2%
ABM2	45,0%	42,4%	12,6%
RS	44,4%	42,5%	**13,4%**
RUS	44,5%	42,8%	12,8%

Referente a los resultados de otros métodos de clasificación, la primera conclusión que podemos obtener es que el uso de cualquiera de ellos nos proporciona una ventaja con respecto a no usar el sistema de soporte. En este caso podemos ver como kNN ofrece un rendimiento ligeramente superior en cuanto al número de partidas ganadas (superando en casi un 10% al perfil básico) manteniendo asimismo una reducción superior al 10% respecto al número de partidas perdidas.

Como se ha comentado anteriormente, es difícil asegurar que estos resultados no hayan sido influidos ligeramente por el azar o suerte a la hora de que haya salido la carta más apropiada. Los resultados presentados en la Tabla 7 han sido calculados como la media de rendimiento tras 5 ejecuciones, pero en la Tabla 8 mostramos el mejor resultado obtenido por cada método dentro de las 5 ejecuciones del mismo.

En este caso podemos observar como el mejor algoritmo es DT, aunque de nuevo kNN se mantiene cercano en cuanto a rendimiento. Es interesante ver como el resto de algoritmos presenta unos resultados más parecidos a las medias calculadas anteriormente.

TABLA 8: MEJORES RESULTADOS DE TODOS LOS PERFILES

perfil	ganadas	perdidas	empatadas
kNN	47,6%	40,6%	11,8%
DT	**48,8%**	**38,2%**	12,6%
ABM2	46,6%	41,8%	11,6%
RS	45,0%	41,4%	13,6%
RUS	45,2%	41,0%	**13,8%**

6 Conclusiones

6.1 Conclusiones del proyecto

Dentro de este proyecto he tenido la oportunidad de desarrollar un sistema completo de soporte a juegos de azar, incluyendo la programación del mismo y el diseño de la interfaz gráfica de usuario.

Para el desarrollo del proyecto han sido especialmente útiles los contenidos explicados en las asignaturas de Inteligencia Artificial, tales como "Coneixement, Raonament e Incerteça" como "Aprenentatge Computacional".

Más específicamente, el proyecto que he desarrollado se ha centrado en el desarrollo de un sistema de soporte al juego BlackJack. Para poder probar diferentes algoritmos se ha creado una interfaz gráfica amigable e intuitiva, la cual permite jugar al juego, seleccionando cualquier perfil y guardar / cargar sesiones de partidas anteriores. El uso de los algoritmos se ha utilizado para la predicción de la siguiente acción a tomar (robar carta o plantarse).

En la fase de experimentos se ha creado una base de datos de entrenamiento y de test. Se han probado por separado cada uno de los algoritmos con diferentes parámetros. Una vez seleccionado la mejor configuración se ha dejado predefinida en los perfiles.

Los resultados obtenidos nos han permitido averiguar que algoritmo se adapta mejor a nuestros datos, siendo en este caso kNN el algoritmo a elegir si se quisiese garantizar un elevado número de partidas ganadas. El análisis de resultados presenta una mejora de 6,8% hasta 11,2% de partidas ganadas sobre las partidas de test.

Para mí ha sido un proyecto realmente interesante, ya que nunca pensé que en el mundo de un juego tan simple como BlackJack, se puede aplicar tantas técnicas e investigar sobre las estrategias que pueden influir en la partida.

6.2 Dificultades encontradas

Al principio del proyecto surgían algunas dudas en relación de programación en Matlab, especialmente relacionadas con el diseño de la interfaz de usuario.
Otra pequeña dificultad encontrada surgió en la etapa de búsqueda bibliográfica del estado del arte. En primera instancia se encontraba demasiada información sobre proyectos similares, pero hay que decir que parte del desarrollo se ha basado en una implementación antigua de Matlab del juego Blackjack. Nos hemos basado en la clase *card* desarrollada en el mismo [22] aunque dicha clase ha sido completamente reescrita para poder cumplir con las funcionalidades esperadas.

6.3. Trabajo futuro

Una posible línea de continuación del proyecto, sería añadir el concepto de apuesta para poder decir exactamente en qué momento el jugador tiene más posibilidades de ganar, mostrando el porcentaje predicho de éxito.
Otra ampliación posible podría ser el hecho de añadir bots, y poder establecer el estilo de comportamiento, es decir, el que el bot que se arriesga siempre, el que planta con 17 puntos o incluso un bot que estudia el comportamiento del jugador y le intente ganar. La siguiente implementación consistiría, a partir de Server Socket añadir la posibilidad de multijugador, con los jugadores en la red local o internet. Respecto a la parte de inteligencia artficial, nos gustaría poder añadir features con el fin de mejorar los resultados de clasificación o explorar métodos adicionales.

Agradecimientos

Me gustaría agradecer la ayuda recibida a lo largo de todo el proyecto a mi tutor Jorge Bernal, ya que, gracias a él, las dudas y problemas obtenidos durante todo el desarrollo han sido solucionados. Además, el hecho de poder hacer tutorías semanales me ha servido para poder avanzar y no quedarme atascado en ninguna parte.

BIBLIOGRAFÍA

[1] Ministerio de Hacienda y Administraciones Públicas (2015). *Estudio sobre prevalencia, comportamiento y características de los usuarios de juegos de azar en España.*

[2] Blackjack. (2017) Wikipedia, La enciclopedia libre. url: https://goo.gl/iWdcKp, última visita: 10/11/2017

[3] Levin, Jonathan (2002). Games with Incomplete Information

[4] Schaeffer, Jonathan (1997). One Jump Ahead: Challenging Human Supremacy in Checkers. Springer.

[5] Michael Buro (1997), NEC Research Institute – A Strong Learning Othello Program 1997

[6] Newborn, Monty (1996). Kasparov versus Deep Blue: Computer Chess Comes of Age.

[7] ChessBase GmbH (2006). Who will be the next Freestyle Champion url: https://en.chessbase.com/post/who-will-be-the-next-freestyle-champion, última visita: 10/11/2017

[8] Brian Sheppard, Artificial Intelligence 134 (2002). World-championship-caliber Scrabble.

[9] Dinis Félix (2008). Artificial Intelligence Techniques in Games with Incomplete Information: Opponent Modelling in Texas Hold'em

[10] Johannes Obenaus (2017). Implementing a Doppelkopf Card Game Playing AI Using Neural Networks.

[11] Stefano Di Palmar (2013). Monte Carlo Tree Search algorithms applied to the card game Scopone.

[12] K. Uston, A. Snyder, S. Case (1986). The Uston SS Count, Gambling Times Inc.

[13] Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression". The American Statistician. 46 (3): 175–185.

[14] D. Coomans; D.L. Massart (1982). Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-Nearest neighbour classification by using alternative voting rules. Analytica Chimica Acta. 136: 15–27.

[15] Lior Rokach, Oded Maimon (2008). Data mining with decision trees: theory and applications, 13

[16] Freund, Y (2009). "A more robust boosting algorithm."

[17] Freund, Y. and R. E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." J. of Computer and System Sciences, Vol. 55, pp. 119–139.

[18] Friedman, J., T. Hastie, and R. Tibshirani (2000). "Additive logistic regression: A statistical view of boosting." Annals of Statistics, Vol. 28, No. 2, pp. 337–407.

[19] Ho, T. K (1998). "The random subspace method for constructing decision forests." IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 8, pp. 832–844.

[20] Seiffert, C., T. Khoshgoftaar, J. Hulse, and A. Napolitano (2008). "RUSBoost: Improving clasification performance when training data is skewed." 19th International Conference on Pattern Recognition, pp. 1–4.

[21] MathWorks. GUI de MATLAB, url: https://goo.gl/79SsjG, última visita: 10/11/2017

[22] Rasmus Anthin (2005). Playing Cards Toolbox 1.0. url: https://es.mathworks.com/matlabcentral/fileexchange/7963-playing-cards-toolbox-1-0?s_tid=prof_contriblnk, última visita: 7/02/2018

APÉNDICE

A1. REQUISITOS HARDWARE/SOFTWARE

Para poder emprender el proyecto es necesario disponer de un ordenador de sobremesa, u ordenador portátil con unos mínimos de hardware y software para poder tener instalado el software básico necesario del Matlab, por lo tanto, los requisitos mínimos serían:

- CPU: Intel o AMD x86-64 con soporte de instrucciones AVX2
- HDD: 4-6 GB para una instalación típica
- RAM: 1 GB mínimo, 4 GB recomendado
- Tarjeta gráfica: Soporte para OpenGL 3.3

A2. RIESGOS DEL PROYECTO

Descripción			Catalogación	
ID	**Título**	**Descripción**	**Posibilidad**	**Impacto**
R1	Mala planificación temporal o planificación excesivamente optimista	A mediados del proyecto, existe la posibilidad de ir por debajo de los tiempos establecidos o asignar muy poco tiempo para realizar la tarea y no cumplir con los tiempos	Alta	Alto
R2	Incremento no valorado de funcionalidad	Durante el proyecto no realizar incremento de funcionalidad no establecida anteriormente, hasta no cumplir los objetivos establecidos	Baja	Alto
R3	Falta de información	Existe la posibilidad durante el desarrollo de los algoritmos, no encontrar material relacionado en internet	Media	Bajo
R4	Enfermedad	A lo largo del proyecto podría sufrir alguna enfermedad que no me permitiera efectuar la rutina de trabajo como habitualmente se espera	Baja	Medio
R5	Falta de comunicación tutor - alumno	Durante el proyecto se podría dar el caso de que por cualquiera de las dos bandas tutor o alumno, no hubiera la suficiente comunicación necesaria para el fácil desarrollo del proyecto	Baja	Medio
R6	Seguimiento del proyecto	Existe la posibilidad de que no se fuera llevando la documentación al día	Media	Crítico

TABLA 9: RIESGOS DEL PROYECTO

ID	Solución
R1	Durante la planificación inicial intentar establecer unos plazos de tiempo realistas y no ser muy optimista. Hay que intentar cumplir con los tiempos de planificación.
R2	Los objetivos tienes que ser razonables, no dedicar más tiempo en mejoras y ampliaciones de funcionalidad si esto no está previsto, primero desarrollar lo más importante y las mejoras o ampliaciones dejar al final.
R3	Antes de comenzar el proyecto se debe realizar un estudio del estado del arte para hacerse una idea de hasta qué punto se puede contar con el material de internet.
R4	Intentar ir por delante de las metas establecidas para poder superar cualquier imprevisto o enfermedad.
R5	Establecer unas bases de comunicación semanales, y siempre que sea necesario enviar correos para cualquier tema.
R6	Planificar la semana para no dejar todo para el último momento, ir rellenando la memoria con anotaciones y cambios realizados, para no olvidar de ninguna fase importante.

TABLA 10: ACCIONES DE CONTIGENCIA

A3. planificación detallada del proyecto

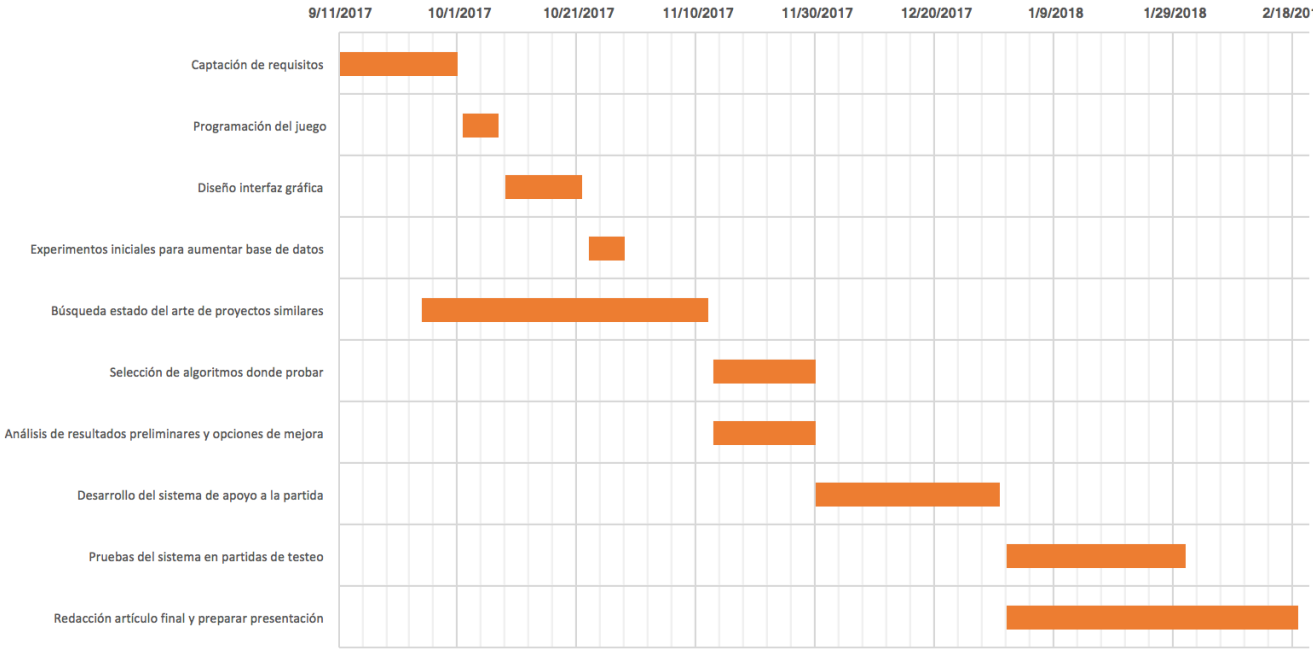

Ilustración 1: Diagrama de Gantt