P2 | Segundo Parcial v2

Miguel Ángel Fernández Gutiérrez

A menos que se especifique lo contrario, las ilustraciones que aparecen han sido tomadas de

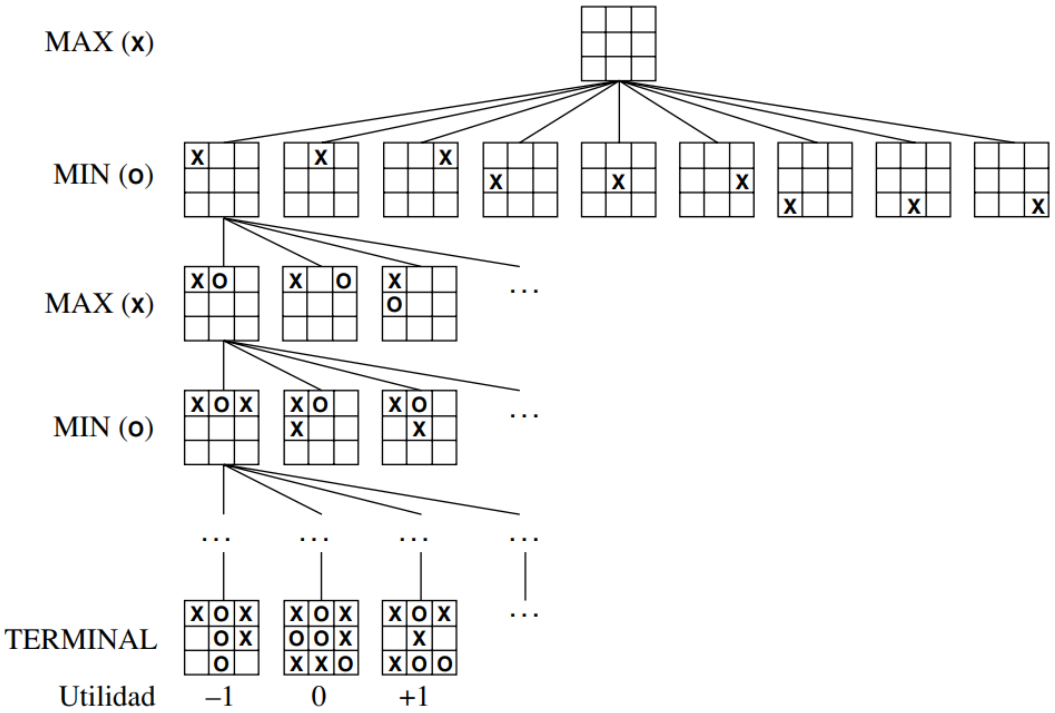
S. Russell, P. Norvig (2004). *"Inteligencia Artificial: un Enfoque Moderno"* (2nd ed.). Pearson-Prentice Hall.

Pregunta 1. Componentes de un juego.

Un **juego** puede definirse formalmente como una clase de problemas de búsqueda con los siguientes componentes:

- El **estado inicial**, que incluye la posición del tablero e identifica al jugador que mueve.
- Una **función sucesor**, que devuelve una lista de pares (*movimiento*, *estado*), indicando un movimiento legal y el estado que resulta.
- Un **test terminal**, que determina cuándo se termina el juego. A los estados donde el juego se ha terminado se les llaman estados terminales.
- Una **función utilidad** (también llamada función objetivo o función de rentabilidad), que da un valor numérico a los estados terminales.
- El **número de jugadores** y el **orden** de actuación de éstos.
- La **información** disponible de los jugadores, que puede ser **perfecta** en el caso de que los jugadores dispongan de toda la información sobre el juego (como es el caso del ajedrez o el go), o **imperfecta**, cuando no dispone de toda la información (como es el caso del póker).
- La existencia o no de **movimientos de azar**, que hagan que el juego sea o no sea determinista.
- La existencia o no de **pagos colaterales** (*equilibrio de Nash*). El equilibrio de Nash ocurre cuando los jugadores están satisfechos con el estado del juego en un momento.
- El **reparto de beneficios** que sucede en **juegos de suma nula**, aquellos en los que el beneficio de un jugador supone la pérdida de otro; o en **juegos de suma no nula**, en los que la ganancia de un jugador no implica la pérdida de otro.

El estado inicial y los movimientos legales definen el **árbol del juego**. A continuación, un árbol de búsqueda para el juego *tres en raya* (también llamado *tic-tac-toe*), con dos jugadores: MIN y MAX. El nodo de arriba es el nodo inicial, y MAX mueve primero. Vemos que el valor asignado a cada nodo terminal puede ser +1, 0 ó -1 (en función de si gana MAX, hay empate o gana MIN, respectivamente).



Pregunta 2. ¿Qué es el factor de ramificación y cómo afecta a la complejidad de un juego? Describe en líneas generales el algoritmo minimax y el de la poda alfa-beta.

Factor de ramificación

El **factor de ramificación** es el máximo número de sucesores de cualquier nodo. Un factor de ramificación más bajo reducirá la complejidad del problema, pues supondrá menos exploración para alcanzar la solución.

Algoritmo minimax

En un problema de búsqueda usual, la solución óptima es una secuencia de movimientos que conduce a un estado objetivo. En el caso de los juegos, sin embargo, debemos de tener en cuenta las acciones de nuestro contrincante. En un juego de dos jugadores, MIN y MAX, en el que queramos que el jugador MAX gane, MAX debe encontrar una *estrategia contingente*, que tenga en cuenta las acciones del jugador MIN, para ganar.

El **algoritmo minimax** es un algoritmo que calcula la **decisión minimax** del estado actual. Una decisión minimax es aquella que conduce al jugador MAX al sucesor con el valor minimax más alto, es decir, el que nos conducirá a que MAX gane (la utilidad para MAX de estar en el estado correspondiente, asumiendo que ambos jugadores juegan óptimamente desde allí al final del juego.).

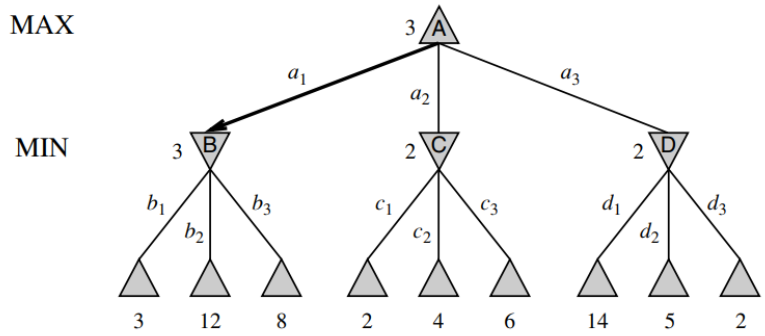
En términos de componentes de un juego, el algoritmo minimax es apropiado para juegos bipersonales de suma nula con información perfecta.

Es un algoritmo recursivo, que avanza en profundidad hasta llegar a un nodo hoja o a un cierto límite de exploración, y va devolviendo los valores de ese nodo hacia arriba. En cada nivel, tomamos el máximo o el mínimo de los sucesores, de tal modo que obtengamos la mejor decisión anteriormente descrita.

Hemos de tener en cuenta que no siempre es posible representar el árbol completo del juego, incluso en juegos tan sencillos como el *tic-tac-toe* (del que hablamos ya en la pregunta 1). No hablemos por tanto ya de juegos mucho más complejos, como las damas, el ajedrez o el go, donde es imposible hacer una exploración total hasta la terminación. Usaremos, en su lugar, la estrategia de encontrar una buena jugada de forma inmediata. En otras palabras:

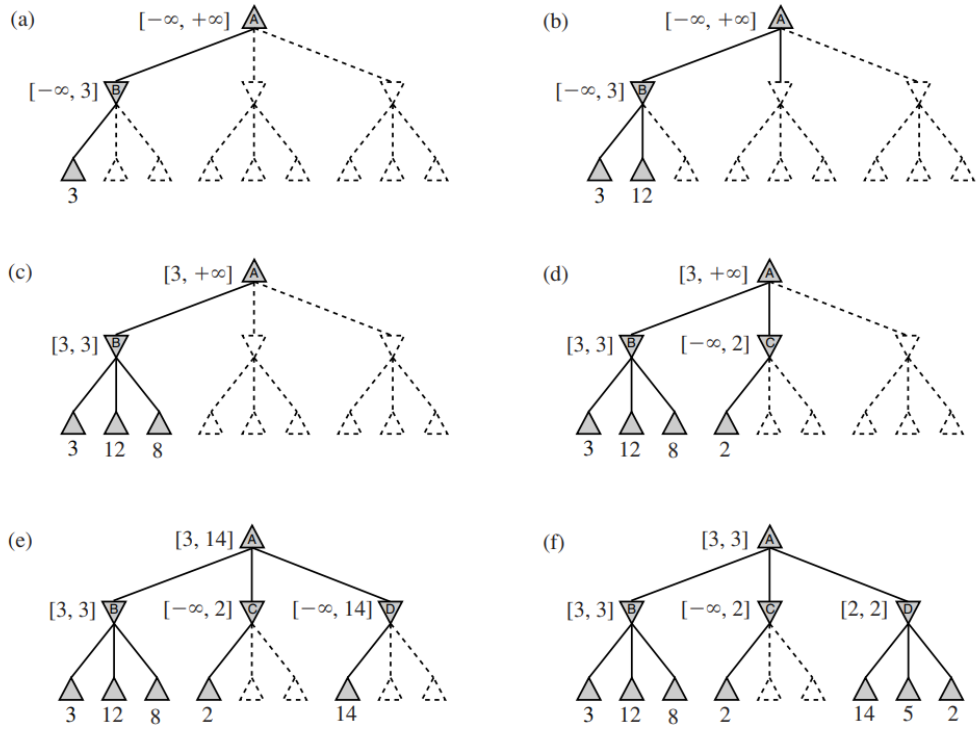
- En caso de que sea posible, haremos el **árbol completo**, describiendo todos los estados de juego posibles, y etiquetando los nodos hoja con los valores de utilidad para MAX. El objetivo es encontrar un conjunto de movimientos *accesible* que dé como ganador a MAX. Se propagan los valores de las jugadas terminales de las hojas hasta la raíz. Una *estrategia ganadora* para MAX es un subárbol en el que todos los nodos terminales son ganadores.
- En otro caso, encontramos una buena jugada *inmediata*, para lo cual la heurística tendrá una importancia vital. Generaremos los sucesores en este caso poco a poco, usando la **regla minimax**.

A continuación, un árbol de juegos de dos capas. Los nodos △ son *nodos MAX*, en los que les toca mover a MAX, y los nodos ▽ son *nodos MIN*. Los nodos terminales muestran los valores de utilidad para MAX, los otros nodos son etiquetados por sus valores minimax. El mejor movimiento de MAX en la raíz es a_1, porque es el que le conduce al sucesor con el valor minimax más alto, y la mejor respuesta de MIN es b_1, porque es el que le conduce al sucesor con el valor minimax más bajo.



Algoritmo de poda alfa-beta

El problema del algoritmo minimax es que tiene que examinar un número de estados que crece exponencialmente en el número de movimientos. No podemos eliminar este exponente, pero podemos dividirlo a la mitad (en el caso más favorable). Esto es así porque es posible calcular la decisión minimax correcta sin mirar todos los nodos del árbol de juegos: recurriremos a la **poda**.



El **algoritmo de poda alfa-beta** se comporta de forma similar al algoritmo minimax, pero introducimos las variables auxiliares α y β, que modificaremos conforme recorremos el árbol. Dependiendo de estos valores, decidiremos qué nodo explorar:

- α: valor de la mejor opción hasta el momento para MAX a lo largo del camino (valor más alto).
- β: valor de la mejor opción hasta el momento para MIN a lo largo del camino (valor más bajo).

Sólo nos interesará proseguir la búsqueda desde ese nodo si podemos mejorar α. La búsqueda alfa-beta actualiza el valor de α y β según se va recorriendo el árbol y termina la recursión cuando encuentra un nodo peor que el actual valor α o β correspondiente.

Al principio del proceso de búsqueda α toma el valor $-\infty$, mientras que β toma el valor $+\infty$.

Desde un nodo MAX, iremos actualizando el valor de α a medida que vayamos explorando cada arco que cuelgue de ese nodo. Dicha actualización consiste en comparar el valor obtenido por cada arco con el valor actual de α. Si el valor obtenido es mayor, ése será el nuevo valor de α. Por tanto, α almacena el mejor valor que hemos encontrado desde el nodo en que estamos.

Desde un nodo MIN, iremos actualizando el valor de β a medida que vayamos explorando cada arco que cuelgue de ese nodo. Dicha actualización consiste en comparar el valor obtenido por cada arco con el valor actual de β. Si el valor obtenido es menor, ése será el nuevo valor de β.

Por tanto, β almacena el peor valor (para MAX) encontrado por MIN hasta el momento. Sólo interesará que la búsqueda prosiga si MIN pudiera reducir β.

El intervalo (α, β) contiene los valores que se puede seguir consiguiendo en el proceso de búsqueda. Cuando α crece tanto que sobrepasa a β, o cuando β disminuye tanto que se hace menor que α, se puede hacer una poda y no seguir explorando el subárbol que cuelga desde el nodo en que nos encontramos, prosiguiendo la búsqueda en profundidad desde su nodo padre.

Pregunta 3. ¿Qué problemas plantea el cálculo de predicados en la resolución de problemas de IA?

El cálculo de predicados en la resolución de problemas de IA presenta problemas tanto **semánticos** como **computacionales**:

- Problemas semánticos:
  - Es difícil expresarlo todo en fórmulas lógicas.
  - Es complicado hacer razonamientos acerca de predicados, ya que estos se aplican a objetos o a funciones, pero nunca sobre otros predicados.
  - Es complicado decir que dos cosas sean iguales, porque para serlo deberían ser iguales en todos sus atributos, y hay propiedades que no podemos conocer.
  - No se deberían resolver problemas en los que dispongamos de información incompleta y/o imprecisa, pues es difícil de representar. Es importante diferenciar aquí entre la **probabilidad**, que es la dificultad de predecir algo de forma exacta, aun teniendo toda la información; y la **vaguedad**, o falta de información precisa.
  - Las excepciones: a veces una condición no es cierta aunque haya una regla para ello. Por ejemplo, es útil decir que "los pájaros vuelan", pues la mayoría lo hace, pero no es una afirmación universalmente cierta.
- Problemas computacionales:
  - Consistencia (solidez, *soundness*): lo que se demuestra como verdadero lo es realmente.
  - Completitud (*completeness*): puede ser que una cierta afirmación sea verdadera, pero no pueda demostrarse, pues será necesario partir de unos axiomas adecuados.
  - Es complicado representar el tiempo: en cada instante puede cambiar el estado del conocimiento. El problema estriba en que el tiempo es continuo, y es difícil discretizarlo e implementarlo debidamente.
  - Es computacionalmente pesado. Si para cada demostración vamos añadiendo un axioma, conforme se van obteniendo cláusulas el dominio crece exponencialmente.

Como conclusión, podemos decir que el cálculo de predicados es apropiado cuando disponemos de toda la información necesaria, y ésta es lo suficientemente precisa.

Pregunta 4. Modelos de conocimiento heredable ¿Qué tipo de conocimiento organizan las redes semánticas? Describir en líneas generales el concepto de *frame*.

Modelos de conocimiento heredable

Los **modelos de conocimiento heredable** son aquellos que se acercan más a la forma en la que los humanos almacenamos el conocimiento (en contraposición al cálculo de predicados).

¿Qué tipo de conocimiento organizan las redes semánticas?

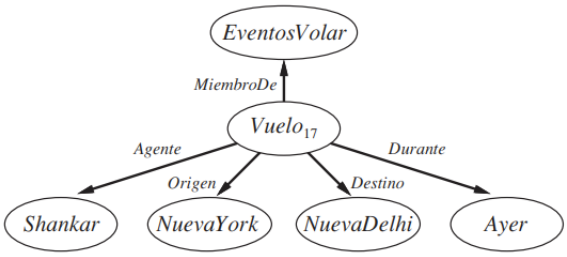
Las **redes semánticas** son redes asociativas que intentan representar el lenguaje natural. Pueden ser representadas como grafos dirigidos, en los que los nodos son instancias y los arcos relaciones. Como mecanismo de razonamiento usan la *herencia*: un concepto puede heredar las propiedades de otro concepto que se encuentre en un nivel de jerarquía más alto.

Concepto de *frame*

Cuando tenemos que representar un sistema complejo, las redes semánticas se vuelven inmanejables. Para esto fueron creados los *frames*.

Un ***frame*** o **marco** es una estructura de datos para representar una situación estereotipada. Cada marco se caracteriza por un conjunto de ***slots*** o **campos** que se asocian en general a atributos, y que en conjunto sirven para identificar a los marcos.

Están especialmente concebidos para tareas de reconocimiento: la información recibida hace que se activen unos *frames* y esto, a su vez, provoca la activación de otros *frames* conectados con los primeros, dando así lugar a una **red de activación** cuyo objeto es predecir y explicar la información que se va a encontrar en esa situación. Este reconocimiento suele llamarse **herencia** o, más generalmente, **reconocimiento descendiente**.



Ejemplo de red semántica

```
instancia arteria-branquial-izquierda es
    instancia-de arteria;
        diámetro = 0.4;
        situación = brazo;
        sangre = rica-oxigeno;
    end
instancia brazo es
    instancia-de miembro;
        posición = superior;
    end

clase arteria es
    subclase-de vasos-sanguíneos;
        situación = {brazo, cabeza,
                pierna, tronco};
    end

clase miembro es
    subclase-de nil;
        posición = {superior, inferior};
    end
```

Ejemplo de *frame*

Pregunta 5. Estructura y componentes de un sistema experto.

Primero, observaremos que un **sistema basado en el conocimiento** es aquel programa (en general, sistema hardware-software) que emplea masivamente conocimiento para resolver un problema dentro de un dominio determinado. Necesitan tres componentes básicos:

- Una **base de conocimiento** que contenga el conocimiento necesario sobre el dominio (universo) del problema a resolver.
- Un **motor de inferencia** que permite razonar sobre el conocimiento de la base de conocimiento y los datos proporcionados por el usuario.
- Una **interfaz de usuario** para la entrada/salida de datos de cada problema. Los datos de un problema son conocidos, a veces, como **base de hechos**.

Un **sistema experto** es un sistema basado en el conocimiento que es capaz de comportarse como un experto (humano) en un cierto dominio de actividad. Además de resolver el problema, pueden ser consultados y justifican su razonamiento. Estos sistemas tienen, adicionalmente (a los componentes anteriormente mencionados):

- Un **módulo de explicaciones/justificación**, para explicar cómo se ha llegado a la solución.
- La **memoria de trabajo**, que contiene información relevante que el motor de inferencia ha usado para razonar las respuestas para el usuario.

Pregunta 6. Paradigmas de Aprendizaje Automático.

El **aprendizaje automático** es el campo de estudio de las ciencias de la computación que da a las computadoras la capacidad de aprender sin ser explícitamente programadas. Otra definición posible es que se compone de programas que mejoran su comportamiento con la experiencia.

Tenemos los siguientes paradigmas de aprendizaje:

- **Aprendizaje memorístico.** Se establece una correspondencia uno-a-uno entre las entradas y la representación almacenada. Por tanto, el almacenamiento y la recuperación se basan en la asociación.
- **Aprendizaje deductivo.** Hace uso del "razonamiento artificial": inferencia y deducción empleando modelos lógicos.
- **Aprendizaje analítico.** Es un aprendizaje basado en explicaciones. Se construye una explicación para cada ejemplo en relación con un concepto dado y se generaliza la explicación de modo que pueda emplearse en el futuro.
- **Aprendizaje analógico.** Entender y resolver una situación por su parecido con otras anteriormente resueltas.
- **Aprendizaje inductivo.** Es el paradigma más ampliamente estudiadodentro del aprendizaje. Hablando en términos muy generales, se trata de aprender un concepto o una clasificación a partir de ejemplos y contraejemplos.

Consideraremos, además, las siguientes técnicas de aprendizaje:

- **Aprendizaje supervisado.** Para cada entrada, se dispone de un profesor/supervisor que proporciona una salida deseada, ya sea una clase o un valor a aproximar (clasificación vs regresión).
- **Aprendizaje no supervisado.** No se dispone de una salida deseada cada entrada, sino que se busca agrupar/clasificar los datos en función de ciertas características (medida de distancia).
- **Aprendizaje por refuerzo.** Se aprende a partir de la información obtenida al realizar procesos de ensayo-error en los que se obtienen "señales" de beneficio/coste.

Pregunta 7. Describir el problema del ruido y el del sobreajuste en aprendizaje automático.

El problema del ruido

El problema del **ruido** se presenta al tener dos o más ejemplos con la misma descripción (en términos de atributos) pero con diferentes clasificaciones. Esto puede ocurrir bien porque las mediciones sean incorrectas, o bien porque de todos los atributos posibles se han escogido atributos que no permiten diferenciar a estos ejemplos.

El ruido afecta siempre a la tasa de aciertos y errores, por lo que es deseable eliminarlo. Para ello, basta medir mejor o tomar una mayor muestra de datos, o tener en cuenta más u otros atributos.

El problema del sobreajuste

El problema del **sobreajuste** se presenta cuando se encuentran "regularidades" poco significativas en los datos. Se dice que una hipótesis h se sobreajusta al conjunto de entrenamientos si existe alguna otra hipótesis h' tal que el error de h es menor que el error de h' sobre el conjunto de entrenamiento, pero es mayor sobre la distribución completa de ejemplos del problema (entrenamiento + test).

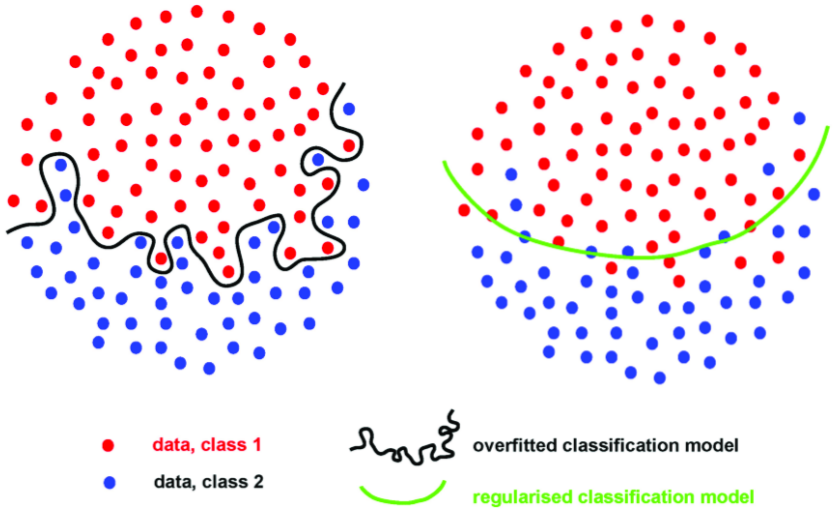


Ilustración tomada de ResearchGate

Pregunta 8. ¿Qué son y cómo se construyen los árboles de decisión?

Los **árboles de decisión** son árboles que toman como entrada un objeto o una situación, descrita a través de sus atributos, y devuelve una "decisión" (el valor previsto de salida para esa entrada). Podemos tener tanto atributos discretos, que proporcionarán una salida discreta (**clasificación**), como atributos continuos, que proporcionarán una salida continua (**regresión**).

Son los árboles usados en el aprendizaje automático por estrategia *divide y vencerás* (*splitting*): partir los datos sucesivamente en función del valor de un atributo seleccionado cada vez.

Podemos inferirlos (crearlos) de diversas formas:

- **Trivial.** Se crea una ruta en el árbol por cada instancia de entrenamiento. Genera árboles muy grandes, y no funcionan bien para instancias nuevas.
- **Óptima.** Crea el árbol más pequeño posible compatible con todas las instancias. Para ello, serían necesarias crear todas las posibles soluciones, algo que no es computacionalmente viable.
- **Pseudo-óptima (heurística).** En cada nivel, seleccionamos un atributo en función de la división que produce.

En cuanto a la elección de atributos, un buen atributo debería dividir el conjunto de ejemplos en subconjuntos que sean o "todos positivos" o "todos negativos".

A continuación, un árbol de decisión para decidir si esperar por una mesa.