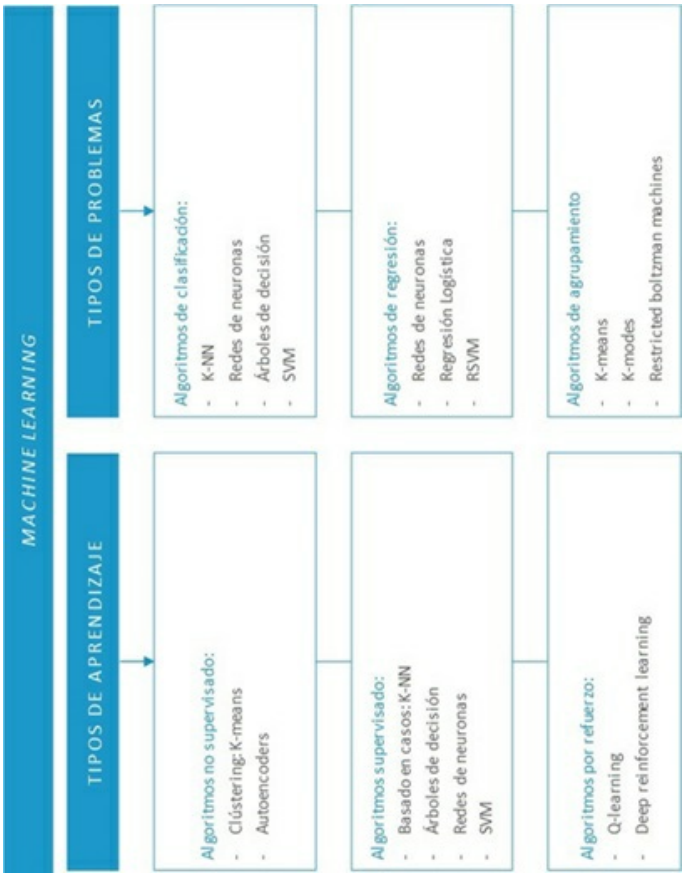
Investigación y Gestión de Proyectos en Inteligencia Artificial

Tema 9. Investigación en aprendizaje automático

Índice

Esquema



9.1. Introducción y objetivos

En este tema se explica a alto nivel el **concepto de aprendizaje automático o *machine learning* (ML).** Estos conceptos son fundamentales hoy en día en la inteligencia artificial y es uno de los campos más activos en esta.

De forma detallada, los **objetivos** que persigue esta unidad son:

- Ser capaz de entender cómo aprende una máquina.
- Conocer los tipos de algoritmos y problemas que existen y que pueden resolverse con *machine learning*.
- Identificar las principales técnicas y algoritmos de *machine learning* disponibles.
- Conocer los proyectos que se pueden crear usando *machine learning*.

9.2. ¿Cómo aprenden las máquinas?

Machine learning es uno de los temas más candentes de la inteligencia artificial actual. Pero como veremos durante este tema, es un inmenso campo de multitud de técnicas que darían cada una de ellas para una asignatura completa para poder tratarlas con profundidad. Lo que pretendemos en este tema es **dar las bases de cómo aprende una máquina y qué diversas implementaciones se han realizado en esta importante área de la inteligencia artificial.**

No está muy claro cuál es el mecanismo interno del aprendizaje del ser humano (o de otros animales). Por lo que **vamos a centrarnos en la funcionalidad,** más que en la estructura. Hay, por tanto, muchas definiciones de aprendizaje, pero vamos a resumirlo en la **capacidad que tienen los seres vivos de mejorar sus aptitudes.** Es decir, la capacidad que tienen los seres vivos de realizar tareas que no sabían hacer y por otro lado de mejorar aquellas que ya sabían hacer. Para esto tiene que haber algún tipo de memoria donde se almacene este nuevo conocimiento, pero no es suficiente con una memoria. Se necesita una abstracción que permita enfrentarse a problemas nuevos que no han sido estudiados previamente.

Una de las principales cualidades que permite el aprendizaje es la **flexibilidad.** Un sistema que aprende se puede adaptar a entornos cambiantes. Frente a soluciones programadas que necesitan para cambiar la modificación del propio programa ya que son soluciones **específicas.** De esta forma el aprendizaje permite a un sistema adaptarse a los cambios en el entorno. Es por ello por lo que está especialmente recomendado en entornos dinámicos.

Pero ¿cómo aprende una máquina? El proceso de aprendizaje consiste en **seleccionar de entre un conjunto de hipótesis posibles para realizar una tarea o resolver un problema, aquellas que realizan la tarea o resuelven el problema de la mejor forma posible.** Así que hay ciertos conceptos que son claves. Por un lado,

el sistema debe sacar conclusiones o crear un modelo que sirva para ser usado de forma general. Normalmente estas conclusiones se extraen de analizar casos particulares. Por lo cual, se necesita extraer cuál es el patrón o las características más relevantes en las que se basa el problema o la tarea. Esto no es más que una muestra de lo que se conoce como **abstracción.** Los humanos deben abstraerse de los pequeños detalles para saber resolver una tarea, ya que para resolver una tarea no se puede atender a todos los pequeños detalles a la vez. Pues básicamente eso es lo que en esencia hace un algoritmo de *machine learning.* Crea una descripción del problema de forma abstracta y resumida de este, que permite adaptar dicho modelo a múltiples problemas, que siendo diferentes entre ellos comparten unas características comunes que son las que lo identifican.

Por lo tanto, se necesita simplificar el problema y en este proceso de simplificación se introducen sesgos.

Los sesgos (o *bias* en inglés) son un **conjunto de datos establecidos a priori que ayuda a reducir la incertidumbre que caracteriza a la selección de la hipótesis correcta en el aprendizaje.** Por ejemplo, cuando usamos ejemplos para aprender, los algoritmos de *machine learning* introducen sesgos o generalizaciones que hacen perderse los detalles de los ejemplos introducidos. Pero esto es fundamental si queremos que el modelo pueda ser aplicado a otros ejemplos no usados en el entrenamiento. Porque, si no se introdujesen estos sesgos en el aprendizaje, entonces los algoritmos de aprendizaje automático podrían resolver muy bien los ejemplos que se han usado para el aprendizaje, pero no serían capaces de resolver otros ejemplos diferentes. Es decir, se habrían especializado, pero no tendrían capacidad de generalización.

Así que **la introducción de sesgos es esencial para que un algoritmo aprenda.** Y esto es algo que hacemos constantemente nosotros como humanos. La mayoría de las ecuaciones que se plantean en física tienen sesgos, ¿cuáles son esos sesgos? Por ejemplo, las ecuaciones de carga eléctrica asumen que los cuerpos están en el

vacío. El movimiento uniformemente acelerado asume que lo haces en el vacío (sin fricción) y que la tierra no es curva (ni el espacio es curvo por la fuerza de la gravedad). La famosa ecuación de la relatividad de Einstein (e=mc2), en realidad solo se aplica en los casos donde haya partículas con masa. Es decir, no se puede aplicar a la luz (que no tiene masa, pero si energía. Por lo que la fórmula general es más compleja, pero esta nos sirve como buena aproximación para la mayoría de los casos). Las ecuaciones de física que estudias en el instituto y en la universidad tienen sesgos. Lo mismo que las reglas de ortografía que tienen excepciones. Como vemos no son reglas totalmente generales, pero se crean para ayudarnos en el proceso de abstracción.

Así que podemos decir que el mecanismo por el cual una máquina o un programa aprende es **gracias a que es capaz de construir un modelo simplificado de un problema con base en ejemplos parciales de ese problema,** intentando extraer las características generales que lo describen. A esto se le denomina sesgo y lo contrario, no introducir sesgos se le denomina **varianza** *(variance)* y es algo que normalmente no queremos. También se les denomina sobreajuste a los datos de entrenamiento. Este problema hace que el sistema pierda la capacidad de abstracción y, por tanto, procesará mal los ejemplos no introducidos como entrenamiento de la red. Paradójicamente, el sesgo introduce un error en el modelo como se puede ver en la figura 1. Pero dicho error nos permite predecir el comportamiento de una nueva variable mucho mejor que el modelo que no proporciona dicho error. A esto también se le denomina en algunos entornos **sobreadaptación** *(overfiting)* del modelo a los datos de entrenamiento.

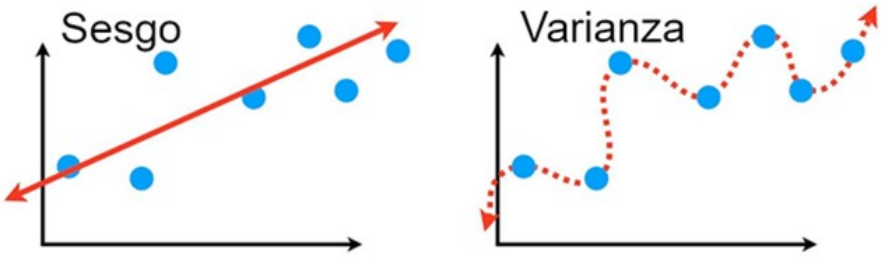


Figura 1. Definición gráfica de sesgo (*bias*) y varianza (*variance*). Fuente: https://skillsfuturetv.com/wp-content/uploads/2020/05/maxresdefault-360.jpg

Ahora que tenemos claro que se necesita generalizar y, por tanto, introducir errores para conseguir aprender. Asimismo debemos tener en cuenta que esos errores no deben ser inducidos por una mala selección de los casos de ejemplo. Seleccionar los casos de entrenamientos ayuda a que el sistema no aprenda sesgos innecesarios o que no son recomendables. Esto ya lo veremos con más detalles, pero por adelantar el concepto aquí vamos a ver un caso de ejemplo de un sesgo aprendido, podemos suponer que si no seleccionamos correctamente un patrón de rostros lo suficientemente diverso a nivel étnico, un algoritmo de *machine learning* puede aprender a reconocer solo caras de occidentales pero no de asiáticos o africanos, ya que utilizará probablemente los rasgos distintivos comunes de las caras occidentales como un patrón común que le maximiza la distinción entre lo que es una cara y lo que no.

Como podemos ver en este caso, el sesgo utilizado para aprender es un **sesgo perjudicial.** Así que hay que tener mucho cuidado cuando se presenten los ejemplos para no introducir sesgos artificiales o que induzcan a un comportamiento del modelo no deseado.

Una vez visto cómo aprenden las máquinas a nivel teórico, queda por describir cómo se realiza a nivel práctico. ¿Cómo llegamos a este aprendizaje? Hay diferentes caminos. Podemos clasificar las diferentes técnicas de aprendizaje en distintos tipos en base a si el aprendizaje es guiado por el diseñador del problema o no.

Ideas clave

9.3. Tipología de un proyecto de aprendizaje automático

Podemos clasificar los algoritmos de aprendizaje en tres grandes tipos en función de la intervención que tenga el diseñador del algoritmo en el aprendizaje.

Aprendizaje supervisado

Este tipo de algoritmo **requiere que los ejemplos que se introduzcan para aprender tengan también la solución al problema.** De esta forma, el aprendizaje supervisado intenta deducir una función que permita obtener a partir de los datos de entrada la solución deseada. Ambos valores se proporcionan al algoritmo de forma que este puede analizar cómo de lejos está su modelo actual de conseguir representar correctamente las salidas deseadas. Es decir, puede usar el error de entrenamiento para tomar decisiones en el aprendizaje. En algunos de estos modelos este error de entrenamiento es clave para mejorar el algoritmo como en el caso de las redes de neuronas.

Como ya hemos comentado en la introducción, **no hay aprendizaje si no hay capacidad de generalización.** Es decir, si no somos capaces de aplicar el modelo a ejemplos que no hemos visto antes y obtener un resultado correcto o aproximado. Por lo tanto, necesitamos que nuestra función prediga el valor de un objeto una vez entrenado, sin conocer cuál es el resultado correcto del mismo. Es decir*, necesitamos que haga una generalización.* Ejemplos de algoritmos de aprendizaje supervisado son las redes de neuronas, *support vector machines*, árboles de decisión, etc.

Para entrenar redes de aprendizaje supervisado, el diseñador debe seleccionar un conjunto de ejemplos de entrenamiento y un conjunto de ejemplos de validación. Esto es así para poder comprobar de forma empírica cuál es la capacidad de generalización de la red.

Así pues, se deberá crear una base de ejemplos con los atributos que describen dicho ejemplo y la clase a la que pertenece. De ellos, seleccionaremos un subconjunto de entrenamiento y otro de test. Normalmente se suele usar un 75 % de entrenamiento y un 25 % de validación o test. Pero no es una regla escrita y puede ser diferente en función del problema a tratar.

Una técnica de pruebas muy utilizada es el *cross validation* o la **validación cruzada.** Sirve para evaluar los resultados de un modelo y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Este método consiste en **calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones.** Se utiliza en entornos donde el objetivo principal es la predicción y se quiere estimar la precisión de un modelo que se llevará a cabo a la práctica. Por ejemplo, se puede dividir los datos en bloques en cuatro y coger tres de ellos para entrenar y uno para validar. Repitiendo el problema cuatro veces cambiando el bloque de validación en cada vez y calculando la media aritmética de las cuatro validaciones, obtendremos una mejor aproximación al error de aprendizaje y minimizaremos la introducción involuntaria de posibles sesgos a la hora de realizar la partición.

El aprendizaje no supervisado

El aprendizaje no supervisado **es aquel que se lleva a cabo sobre ejemplos que no informan del resultado esperado de este.** Por lo tanto, podemos decir que no hay nadie que le está indicando al sistema cómo se resuelve el ejemplo propuesto. Esto debe descubrirlo el propio algoritmo. Por lo tanto, los algoritmos de aprendizaje no supervisado tratan las entradas como variables aleatorias y crean modelos de densidad para los diferentes conjuntos de datos. Ejemplos de algoritmos de aprendizaje no supervisado son los mapas autoorganizados, el análisis multivariante, el clústering, los estimadores de máxima verosimilitud o las redes de neuronas de base radial. Algunos modelos de *deep learning* como los modelos de extracción de

características también son no supervisados. Aunque las redes de neuronas son algoritmos supervisados, existen algunos trucos que detallaremos más adelante para conseguirlo.

Para realizar un entrenamiento de un algoritmo de aprendizaje no supervisado, no debemos construir pares atributos-valor esperado, ya que no sabemos el valor esperado (o no lo necesita saber el algoritmo en cuestión). Esto hace más complejo e impreciso calcular el error de estos modelos. Pero debemos medir si nuestro algoritmo está funcionando bien de todas formas, ¿cómo lo hacemos? Debemos introducir medidas artificiales. Por ejemplo, en un algoritmo de agrupamiento podemos estimar la varianza total del sistema. Pero en estos casos, visualizar las agrupaciones realizadas ayuda mucho a comprender si el agrupamiento se ha realizado correctamente.

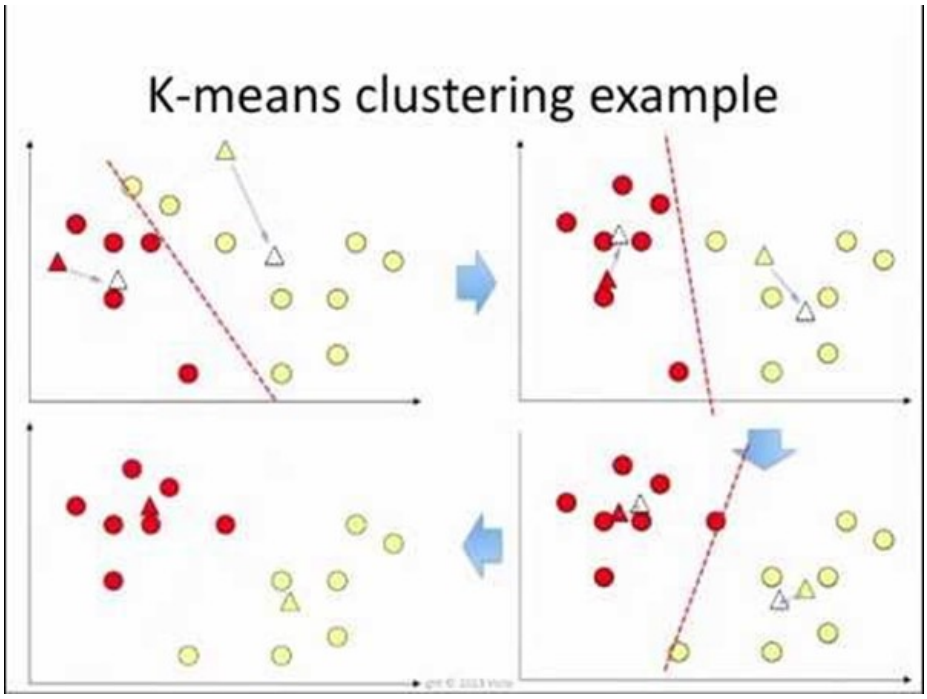


Figura 2. Ejemplo de funcionamiento de *k-means,* un algoritmo no supervisado y cómo visualizar si el agrupamiento realizado es correcto. Fuente: http://www.cs.us.es/~fsancho/?e=77

En el caso de que tengamos las soluciones de las clases de algunos ejemplos, podemos comprobar si ha clasificado correctamente los ejemplos *a posteriori.* Aunque como decimos, esta información solo nos servirá para estimar el error cometido, pero el algoritmo no toma ventaja de esta información para mejorar los resultados.

Aprendizaje por refuerzo

El algoritmo aprende observando el mundo que le rodea. Su información de entrada es el *feedback* o retroalimentación que obtiene del mundo exterior como respuesta a sus acciones. Por lo tanto, el **sistema aprende a base de ensayo-error.** Aunque no de forma aleatoria. Se podría considerar como un aprendizaje no supervisado por nadie, pero hay un pequeño matiz y es que los algoritmos no supervisados ni siquiera tienen el *feedback* de si lo están haciendo bien o mal. Aquí el entorno les está dando una retroalimentación de si lo que hacen está siendo efectivo o no. Por lo tanto, no podemos clasificarlos ni como aprendizaje supervisado ni como no supervisado.

El aprendizaje por refuerzo es el más general entre las tres categorías ya que en vez de que un instructor indique al agente qué hacer, **el agente inteligente debe aprender cómo se comporta el entorno mediante recompensas (refuerzos) o castigos de forma autónoma,** derivados del éxito o del fracaso de sus decisiones. El objetivo principal es aprender la función de valor que ayude al agente inteligente que lo utiliza a maximizar la señal de recompensa recibida y así optimizar sus políticas para comprender el comportamiento del entorno y de esta forma ser capaz de tomar buenas decisiones para conseguir sus objetivos. Por lo tanto, la versatilidad del sistema es máxima y un mismo sistema podría aprender a realizar cosas completamente diferentes en función del entorno donde se mueva. Ejemplos de este tipo de técnicas son *Q-learning, deep reinforcement learning,* simulaciones de Montecarlo, etc.

¿Y cómo se puede evaluar un algoritmo de aprendizaje por refuerzo? Pues hay que ejecutarlo en un escenario diferente al aprendido y tener una medida de calidad de este para saber si el algoritmo funciona y ha conseguido generalizar un modelo. En parte la propia función le dice al algoritmo si está haciendo las cosas bien o no sirve como medida de la calidad de este. Pero hay que probar el sistema en otros entornos para ver si ha sido capaz de aprender una política válida para cualquier entorno similar. Y ahí necesitaremos una medida de rendimiento. Imaginemos que tenemos un sistema de aprendizaje por refuerzo que ponemos a entrenar para que aprenda a jugar a un juego. El sistema debe aprender a jugar al juego a base de prueba y error y en cada decisión que tome de forma acertada, el sistema debe premiar esa política. Pero luego lo tenemos que poner a jugar al juego contra otros jugadores diferentes y tendremos que evaluar si consigue ganar a esos otros jugadores y cómo de rápido o de eficiente ha ganado (cuántas piezas ha perdido, cuántos puntos ha conseguido, en cuánto tiempo ha ganado…). De esta forma validaremos el sistema.

Tipos de problemas

Vistos los diferentes tipos de algoritmos de aprendizaje automático, nos queda por describir los diferentes tipos de problemas que estos pueden abordar. Principalmente **podemos agrupar los problemas en tres tipos:** problemas de clasificación, de regresión y de agrupación.

Es importante detectar cuándo un problema es de clasificación, de regresión o de agrupamiento, porque las técnicas empleadas y los métodos para obtener los datos no son iguales. Tampoco es igual la forma con la que evaluamos los errores que comenten los algoritmos en ambos modelos.

Problema de clasificación

La clasificación es el problema de identificar a cuál, de un conjunto de categorías (subpoblaciones), pertenece una nueva observación, sobre la base de un conjunto de datos de entrenamiento que contienen observaciones (o instancias) cuya

categoría es conocida.

Para que haya un problema de clasificación se deben conocer las clases *a priori* y además estas deben ser un conjunto finito.

Algunos algoritmos que resuelven problemas de clasificación son: árboles de decisión, *support vector machine,* redes de neuronas (discretizando la salida de la red), K-NN, redes bayesianas, *random forest,* etc.

Problema de regresión

Los problemas de regresión **son problemas en los que la salida que se espera es una salida continua.** Un número de casos infinito o al menos indefinido y, por tanto, lo que queremos normalmente es usar una estimación de esta salida lo más precisa posible. Los algoritmos que intentan realizar regresión intentan estimar la función de mapeo (f) de las variables de entrada (x) a las variables de salida numéricas o continuas (y). Ahora, la variable de salida podría ser un valor real, que puede ser un valor entero o de coma flotante.

Algunos algoritmos que resuelven problemas de regresión son: regresión logística, redes de neuronas, *regression support vector machines,* etc.

Problema de agrupamiento

Un problema es de **agrupamiento si no se conoce *a priori* el número de clases ni el tipo de estas y lo que se pretende es precisamente conocer qué clases existen.** Los algoritmos que se aplican al agrupamiento pretenden encontrar cuáles son los grupos que existen en una población y cuáles son los criterios o las características que separan unas clases de otras.

Algunos algoritmos que resuelven problemas de agrupación son: *k-means, k-modes, mean-shift, binary split, deep belief networks, restricted boltzmann machines,* etc.

9.4. Técnicas de aprendizaje automático

A continuación, vamos a citar las principales técnicas y algoritmos de aprendizaje automático de forma resumida, identificado en qué se basan, cuáles son sus ventajas y cuál son sus inconvenientes.

K-NN (*K-nearest neighbor* o k vecinos más cercanos)

Este algoritmo de aprendizaje supervisado es también conocido como **aprendizaje perezoso.** El algoritmo no intenta extraer una característica de los datos haciendo un proceso de entrenamiento exhaustivo con ellos. **Simplemente los almacena para recuperarlos en el futuro.** Cuando se presenta un nuevo caso, este es comparado con la base de casos y se extrae aquel más similar (o los K más similares) para comprobar cuál era la clase (o la acción) que se llevó a cabo para el caso concreto. Asumiendo que, si el caso es similar, la acción tomada anteriormente será una acción que también será válida para el nuevo caso.

Para ello hay que definir una función de similitud entre dos ejemplos. La calidad de esta media de similitud es la que más ayudará a que el sistema se comporte correctamente. Existen multitud de medidas de similitud, por ejemplo, la distancia euclídea, la distancia de Manhattan, la distancia de edición, la distancia de Minkowski, la distancia de Mahalanobis, etc. Cada una de ellas funciona mejor en unos dominios que en otros. También se pueden ponderar los pesos de los atributos para ajustar más la similitud, dando más importancia a ciertos rasgos que a otros.

Ventajas:

- No paramétrico (salvo que usemos distancias ponderadas). No hace suposiciones explícitas sobre la forma funcional de los datos, evitando los peligros de la distribución subyacente de los datos.

- Algoritmo simple tanto de explicar como de interpretar.
- Alta precisión (relativa). Es bastante alta, aunque no superior a otros modelos más sofisticados.
- El proceso de entrenamiento es inmediato.

Las **desventajas** de este algoritmo son:

- Es muy sensible a los atributos irrelevantes. Hacer una buena selección de atributos relevantes es fundamental para este algoritmo.
- Es sensible al ruido, ya que, si un ejemplo es un mal ejemplo de entrenamiento y es el seleccionado como el más similar, daremos una solución errónea. Esto se puede mitigar haciendo que K sea grande, es decir que el número de elementos cercanos sea mayor para que el peso del ruido sea menor.
- La ejecución de este es lenta si hay muchos datos de entrenamiento, ya que tiene que procesar todos los datos. Existen métodos para optimizarlo usando partición espacial pero aun así es más costoso que otros algoritmos.
- Es caro en memoria en tiempo de ejecución. Si la base de casos es muy grande ocupa mucha memoria.

Árboles de decisión

Es un modelo de predicción supervisado utilizado en diversos ámbitos que van desde la inteligencia artificial hasta la economía. Estos árboles **son muy útiles para visualizar las diversas opciones de las que se dispone para resolver un problema o modelar un comportamiento y cuál es la secuencia de pasos**

necesaria para llegar a dicha decisión, ya que se pueden convertir en reglas, que son mucho más legibles para el que investiga el funcionamiento del modelo. Además, se les pueden aplicar métodos de inducción, como por ejemplo la inducción hacia atrás, gracias a los cuales, mediante sencillos razonamientos, se puede descubrir la lógica que el sistema ha tomado para llegar a seleccionar una acción.

Un árbol de decisión está **compuesto de dos tipos de nodos:** un nodo condicional y una clase. Los nodos condicionales son los nodos internos del árbol y la clase, los nodos hoja o terminales.

Cada **nodo condicional** es una pregunta que se hace a las variables del entorno y solo tiene dos caminos, que la condición sea cierta o falsa. En función de si es una u otra, se tomará un camino y se llegará a un nodo hoja u otro. Cuando se llega a un nodo hoja, este determina la clase a la que pertenece el ejemplo suministrado.

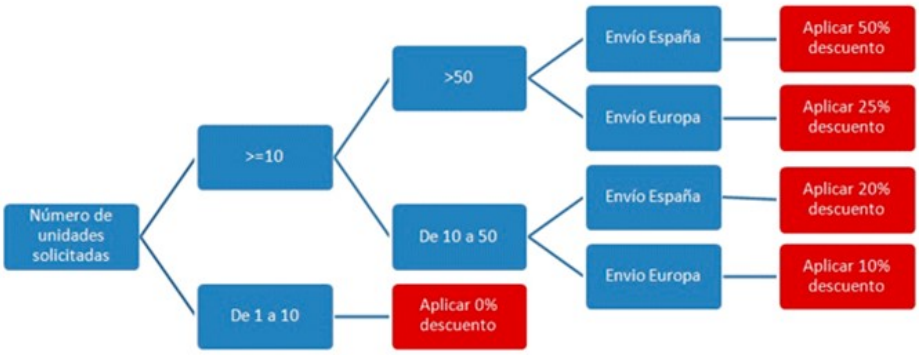


Figura 3. Ejemplo de árbol de decisión. Fuente:
https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n#/media/Archivo:Arbol_decision.jpg

Algunas de las implementaciones más famosas de estos algoritmos son el ID3, J48 o C4.5.

Este algoritmo **utiliza la entropía** (medida de incertidumbre o de desorden) para

ayudar a decidir qué atributo debe ser el siguiente en ser evaluado en el árbol. También nos puede ayudar a descubrir qué atributos son los más relevantes, ya que el algoritmo los colocará más cerca de la raíz. En concreto se pretende maximizar la ganancia de información, donde la entropía se utiliza como una medida del orden de los datos. Es decir, el atributo seleccionado es aquel que deja la información más ordenada o, dicho de otra forma, mejor clasificada. La entropía se calcula con la ecuación.

$$Entropía\,(s) = \sum_{n=1}^{c} -p_i \log 2 p_i$$

Y la ganancia con la ecuación:

$$Ganancia\,(S,A) = Entropia\,(S) - \sum_{v \in Valores(A)}^{A \vee \frac{Sv}{s} Entropia(Sv)}$$

Donde Valores*(A)* es el conjunto de todos los valores posibles para el atributo *A*, y *Sv*, es el subconjunto de *S* para el cual el atributo *A* tiene el valor *v*.

En general, los atributos que separan mejor las clases tienden a reducir más la entropía y, por tal motivo, debe ser seleccionado primero.

Por lo tanto, este algoritmo divide el espacio de soluciones mediante hiperplanos, fijando una de las variables a un valor frontera.

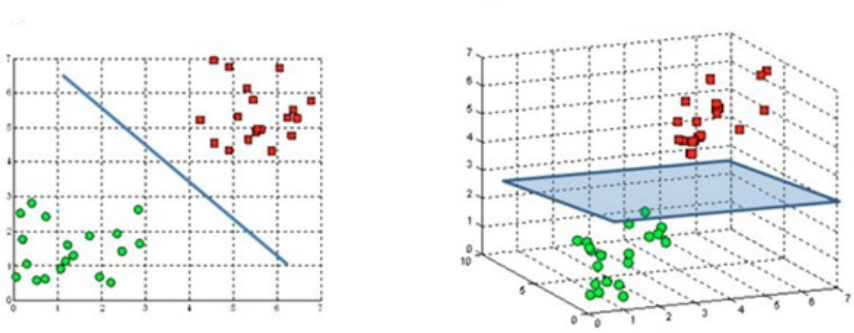


Figura 4. Ejemplo de un hiperplano en 2 y 3 dimensiones que separa dos conjuntos de clases.

Ventajas:

- El entrenamiento es muy rápido.
- Es fácil de interpretar los resultados por un humano, es un algoritmo de caja blanca.
- Para algunos problemas consigue una buena precisión.
- Se pueden convertir fácilmente reglas.
- No requiere una preparación de los datos demasiado exigente.
- Puede trabajar con variables cualitativas y cuantitativas.

Desventajas:

- Es muy dependiente al ruido de la entrada.
- Los árboles de decisión tienden al sobreentrenamiento.

- No se puede garantizar que el árbol generado sea el óptimo.
- Hay conceptos que no son fácilmente aprendibles por los árboles de decisión, ya que las particiones del espacio de soluciones que puede hacer son aquellas que son representables mediante una sucesión de hiperplanos. Si no hay una aproximación lineal al problema, puede que den un modelo poco efectivo.
- Se recomienda balancear el conjunto de datos antes de entrenar.

Existen versiones más modernas de árboles de decisión como es el ***random forest.*** Este ejecuta diferentes árboles de decisión y se realiza un proceso de votación para elegir cuál de los árboles ha generado una mejor predicción. Por ejemplo, nos podemos quedar con la predicción mayoritaria, con la media, etc. Estos métodos obtienen mejores resultados en general que los árboles de decisión convencionales, aunque dificultan la comprensión del modelo generado.

Redes de neuronas

Las redes de neuronas son una analogía del funcionamiento del cerebro humano, que **permiten a las computadoras adoptar ciertas capacidades que se pueden definir como inteligentes,** y que resuelven un importante subconjunto de problemas que otras técnicas computacionales o matemáticas no resuelven o lo hacen de forma más ineficiente, tanto en tiempo como en los resultados obtenidos. Debido a su semejanza con el cerebro, posee muchas capacidades similares a la mente humana. Por ejemplo, son capaces de aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante, etc. Esto hace que ofrezcan numerosas ventajas y que este tipo de tecnología se esté aplicando en múltiples áreas, como aprendizaje adaptativo, autoorganización, tolerancia a fallos, etc.

Las redes de neuronas **se basan en los trabajos realizados por Warren**

Ideas clave

McCulloch y Walter Pittis en 1943, que han servido de base para la aparición de multitud de arquitecturas obtenidas durante los últimos setenta años. Las redes de neuronas se caracterizan por contar con un número de neuronas organizadas de diversas formas, que están conectadas unas a otras a través de enlaces ponderados por unos pesos, que suelen ser recalculables por la propia red, de forma más o menos autónoma, para proporcionar una de las capacidades más importantes de la mayoría de las redes de neuronas, que es su capacidad de aprendizaje. Al igual que el cerebro humano, las neuronas y las redes de neuronas artificiales sustentan su capacidad de cálculo no en el núcleo de las neuronas sino principalmente en las **interconexiones que se realizan entre ellas.**

Normalmente se suelen diferenciar **tres tipos de neuronas en una red,** dependiendo de su función:

- **Neuronas de entrada.** Son las encargadas de recibir los datos del exterior de la red e inician la propagación de dichos datos por toda la estructura. Las conexiones de entrada de estas neuronas (sus dendritas) no realizan ningún cómputo, solo el hecho de captar la información para la red.
- **Neuronas ocultas.** Son aquellas neuronas que permiten enlazar las neuronas de entrada y las neuronas de salida, aunque también pueden estar conectadas a otras neuronas ocultas; y en sus interconexiones es donde precisamente se realiza el proceso de cómputo, tanto al enviar la información a todas las neuronas con las que está interconectada, como al recibir la información.
- **Neuronas de salida.** Son las que muestran al exterior los resultados obtenidos después de que la entrada haya recorrido toda la red. Las conexiones de salida de las neuronas de salida y no realizan ningún cómputo.

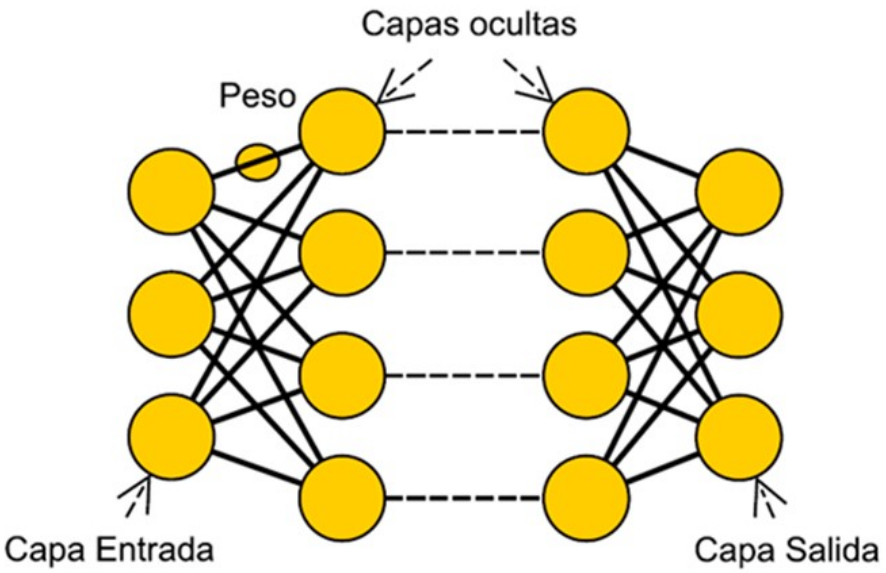


Figura 5. Esquema de una red de neuronas.

Existen **multitud de tipos de redes** de neuronas las principales son las siguientes:

- **Perceptrón multicapa:** es el modelo más simple que consta de una capa de entrada, n capas ocultas y una capa de salida. Es una aproximador universal.
- **Redes de neuronas recurrentes:** algunas de las neuronas de salida son también neuronas de entrada del siguiente ejemplo a procesar. Este tipo de neuronas viene muy bien para predecir sistemas que tengan alguna relación temporal entre los diferentes ejemplos. Por ejemplo, para procesar series temporales o datos secuenciales.
- **Redes de base radial:** calculan la salida de la función en función de la distancia a un punto denominado centro. Al igual que con los perceptrones multicapa, sirven como aproximadores universales y solo tienen una capa oculta. Mientras que las neuronas ocultas poseen carácter local, las neuronas de salida realizan una

combinación lineal de las activaciones de las neuronas ocultas. Este tipo de redes construyen aproximaciones que son combinaciones lineales de múltiples funciones locales no lineales. Entre sus aplicaciones se encuentran análisis de series temporales, procesamiento de imágenes, etc.

- **Redes estocásticas:** son redes recurrentes que utilizan redes bayesianas.
- **Red de Hopfield:** se usan como sistemas de memoria asociativa con unidades binarias. Están diseñadas para converger a un mínimo local, pero la convergencia a uno de los patrones almacenados no está garantizada.

Normalmente, se utiliza para aprender el algoritmo de **retropropagación del error cometido por la red**. No entraremos en detalles, pero la idea es que la derivada del error producido entre la salida generada y la esperada va modificando los pesos de la red en una especie de descenso de gradiente. Esto se hace indispensable para que la red aprenda y conozca el dato correcto del ejemplo que procesa. Como otros algoritmos, si se entrena demasiado tiende a la especialización y a perder capacidad de generalización. Es importante detener el entrenamiento cuando se detecte que este comienza a ofrecer un error de validación creciente. Es decir, que en este tipo de redes es importante entrenar y evaluar con datos no entrenados para saber si la red se sobreadapta a los ejemplos de entrenamiento o no.

El **concepto de *deep learning*** viene precisamente de crear modelos de redes de neuronas con un gran número de capas ocultas. Estos modelos de redes de neuronas ahora son posibles debido a las nuevas técnicas de aprendizaje que se utilizan y al incremento de la potencia de cálculo disponible. Pero en líneas generales siguen las mismas directrices que los modelos simples. Una descripción más detallada de este tipo de redes se sale del objetivo de esta asignatura, pero vamos a enumerar los modelos más conocidos.

Autoencoders

Son **redes neuronales con el objetivo de generar nuevos datos,** primero comprimiendo la entrada en un espacio de variables latentes y luego reconstruyendo la salida con base en la información adquirida. La parte comprimida es una abstracción que sirve para extraer las características importantes de los datos. Este tipo de red **consta de dos partes:**

- **Encoder:** la parte de la red que comprime la entrada en un espacio de variables latentes.
- **Decoder:** la parte que trata de reconstruir la entrada en base a la información recolectada previamente. Se representa mediante la función de decodificación $r = g(h)$.

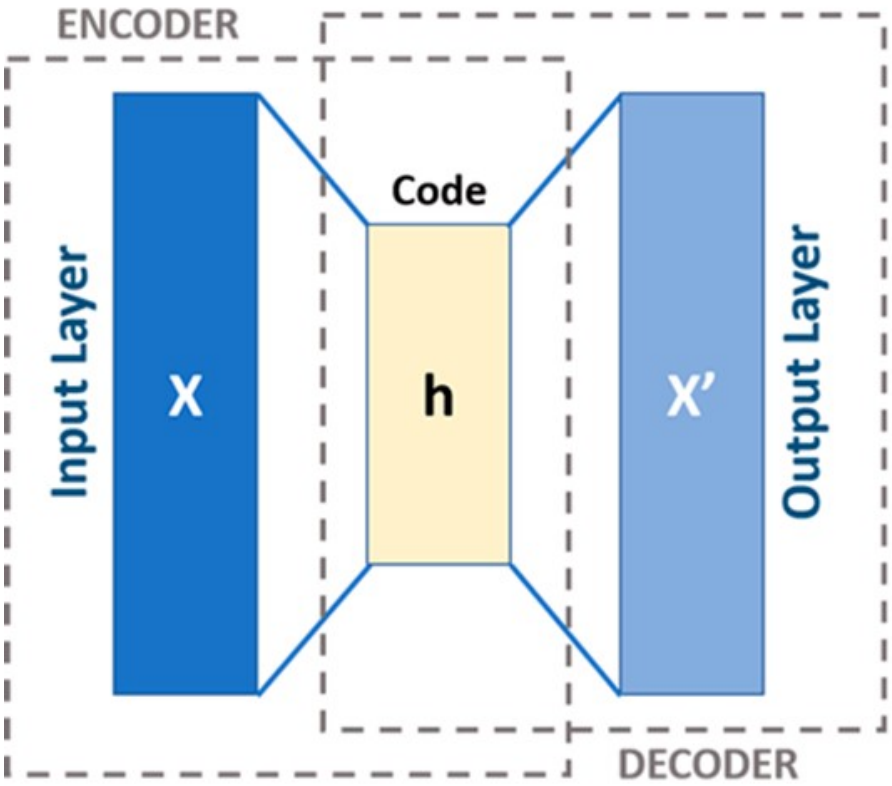


Figura 6. Esquema de un *autoencoder*.

Redes convolucionales

Las redes neuronales convolucionales **son un tipo de redes neuronales donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual.** Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones. Consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Después de cada capa, por lo general, se añade una función para realizar un mapeo causal no lineal. Como cualquier red empleada para clasificación, al principio estas redes tienen una fase de extracción de características, compuesta de neuronas

convolucionales, luego hay una reducción por muestreo y al final tendremos neuronas de perceptrón más sencillas para realizar la clasificación final sobre las características extraídas.

Neural Turing machines

Una máquina de Turing neural es una **red neural extendida con una memoria de trabajo que le proporciona unas altas capacidades de aprendizaje.** La arquitectura de una máquina de Turing neural (NTM) contiene dos componentes básicos: un controlador de red neural y un banco de memoria. Como la mayoría de las redes neuronales, el controlador interactúa con el mundo exterior a través de vectores de entrada y salida. A diferencia de una red estándar, también interactúa con una matriz de memoria utilizando operaciones selectivas de lectura y escritura. Por analogía con la máquina de Turing, las salidas de la red que parametrizan estas operaciones se llaman cabezas, como la cabeza lectora de la máquina de Turing.

Redes recurrentes profundas

Son versiones profundas de las redes recurrentes. Tienen conexiones de retroalimentación. No solo pueden procesar puntos de datos individuales (como imágenes), sino también secuencias enteras de datos (como voz o vídeo). Una de las redes de este tipo más conocidas es la red LSTM (*Long short-term memory*). Esta red es aplicable a tareas como el reconocimiento de escritura, traducción automática, el reconocimiento de voz y la detección de anomalías en el tráfico de la red o sistemas de detección de intrusos.

Ventajas:

- De muy alta precisión.
- Sirven para regresión y para clasificación.

- Las nuevas técnicas las hacen muy escalables.
- Muy tolerantes al ruido.

Inconvenientes:

- Los modelos profundos son muy caros de calcular, se necesita muchísima potencia de cómputo para entrenar.
- Tienden a la sobreadaptación. Hay que vigilar con cuidado el entrenamiento para que no pierdan capacidad de abstracción.
- Son algoritmos de caja negra muy difíciles de analizar por los diseñadores.
- Dependen de ciertos pesos o parámetros iniciales y de cómo se inicien los valores.

Máquinas de vectores de soporte (SVM)

Son un **conjunto de algoritmos de aprendizaje supervisado desarrollados por Vladimir Vapnik** y su equipo en los laboratorios AT&T.

Estos métodos están propiamente relacionados con problemas de clasificación y regresión. Dado un conjunto de ejemplos de entrenamiento podemos etiquetar las clases y entrenar una SVM para construir un modelo que prediga la clase de una nueva muestra.

Los **vectores de soporte son los puntos que definen el margen máximo de separación del hiperplano que separa las clases.** Se llaman vectores porque estos puntos tienen tantos elementos como dimensiones tenga nuestro espacio de entrada. Es decir, estos puntos multidimensionales se representan con vector de *n*

dimensiones.

Hay veces en las que no hay forma de encontrar un hiperplano que permita separar dos clases. En estos casos, decimos que las clases no son linealmente separables. Para resolver este problema **se puede usar un kernel.** El truco del kernel consiste en inventar una dimensión nueva en la que podamos encontrar un hiperplano para separar las clases.

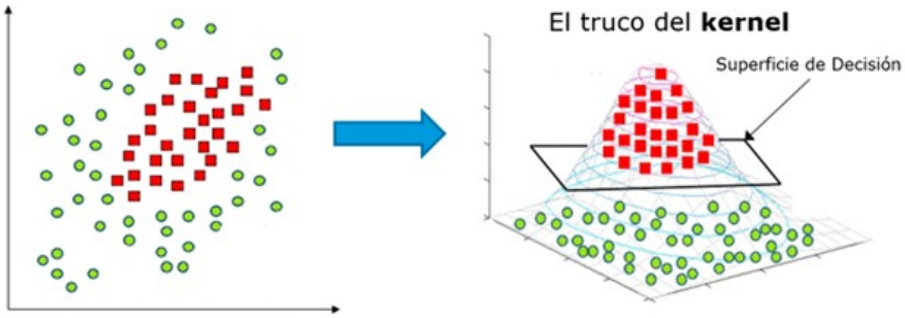


Figura 7. Kernel en SVM. Fuente: https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/

Algunos casos de éxito de las **máquinas de vectores de soporte** son:

- Reconocimiento óptico de caracteres.
- Detección de caras para que las cámaras digitales enfoquen correctamente.
- Filtros de spam para correo electrónico.
- Reconocimiento de imágenes a bordo de satélites (saber qué partes de una imagen tienen nubes, tierra, agua, hielo, etc.).

Ventajas:

- Eficaz en espacios de grandes dimensiones.
- Todavía eficaz en casos donde el número de dimensiones es mayor que el número de muestras.
- Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamada vectores de soporte), por lo que también es eficiente en memoria.
- Versátil: se pueden especificar diferentes funciones del núcleo para la función de decisión. Se proporcionan *kernels* comunes, pero también es posible especificar *kernels* personalizados.

Las **desventajas** son:

- Si el número de características es mucho mayor que el número de muestras evita el exceso de ajuste al elegir las funciones del *kernel* y el término de regularización es crucial.
- Los SVM no proporcionan directamente estimaciones de probabilidad, estas se calculan utilizando una validación cruzada.

Aprendizaje por refuerzo

El aprendizaje por refuerzo es una **serie de técnicas inspiradas en la psicología conductista,** cuya ocupación es determinar qué acciones debe escoger un agente *software* en un entorno dado con el fin de maximizar alguna noción de recompensa o premio acumulado. Esta recompensa o premio se da cuando el agente realiza la acción correcta en un momento dado.

Ideas clave

El agente que aprende **sigue un proceso de decisión de Marcov:**

- El agente percibe un conjunto finito (S) de estados distintos en su entorno, y dispone de un conjunto finito (A) de acciones para interactuar con él.
- El tiempo avanza de forma discreta, y en cada instante de tiempo (t) el agente percibe un estado concreto.
- El entorno responde a la acción del agente por medio de una recompensa. Se formaliza esta recompensa/castigo por medio de un número.
- Tanto la recompensa como el estado siguiente obtenido no tienen por qué ser conocidos a priori por el agente, y dependen únicamente del estado actual y de la acción tomada.

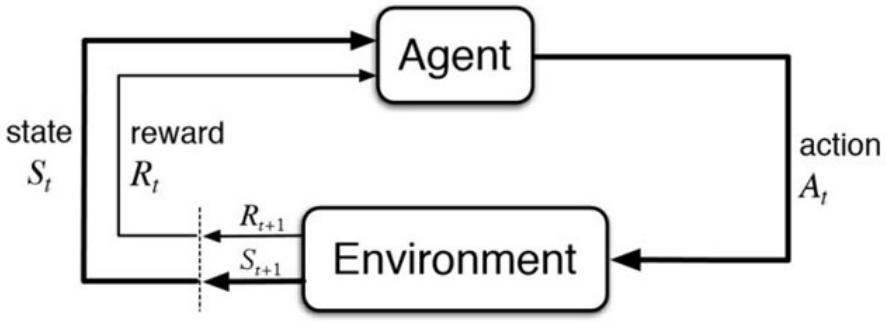


Figura 8. Esquema de aprendizaje por refuerzo. Fuente: https://wiki.pathmind.com/deep-reinforcement-learning

El objetivo del aprendizaje por refuerzo es extraer qué acciones deben ser elegidas en los diferentes estados para maximizar la recompensa. Buscamos, por tanto, **que el agente aprenda una política.** Esta política no es más que un criterio a la hora de seleccionar una acción o la otra.

Existen diferentes implementaciones de aprendizaje por refuerzo, pero nos vamos a centrar en **cómo aprende la política el algoritmo Q-learning.**

El algoritmo, por tanto, tiene una función que calcula la calidad de una combinación estado-acción:

$$Q : S \times A \to \mathbb{R}.$$

Antes de que comience el aprendizaje se inicializa a un valor arbitrario constante. Después, en cada tiempo el agente selecciona una acción (a) observa una recompensa (r) introduce un estado nuevo s(t+1) que depende del estado anterior y de la acción seleccionada, actualizando Q. El núcleo del algoritmo es una actualización del valor de la iteración simple, haciendo la media ponderada del valor antiguo y la información nueva:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Estos valores se van almacenando en lo que se denomina tabla Q que almacena los valores que se van produciendo. La probabilidad de seleccionar un camino bueno es mayor que uno malo. De esta forma el agente cada vez tiende a hacer las acciones que más recompensa le han generado. Aunque se permite la exploración de nuevas políticas dejando siempre una probabilidad de seleccionar otra acción.

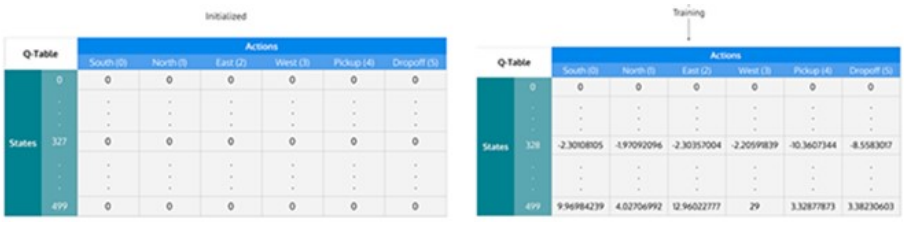


Figura 9. Tabla Q inicializada y actualizada con el entrenamiento.

Existe una versión más avanzada de Q-learning que se ***denomina deep reinforcement learning,*** que es el algoritmo usado por AlphaGo y AlphaStar para

aprender a jugar a *Go* y *Starcraft.* También es el utilizado por OpenAI para aprender a jugar juegos arcade.

El aprendizaje por refuerzo profundo combina redes neuronales artificiales con una arquitectura de aprendizaje por refuerzo que permite a los agentes aprender las mejores acciones posibles en un entorno virtual para alcanzar sus objetivos. Es decir, une la aproximación a la función y la optimización del objetivo, mapeando los pares de estado y acción a las recompensas esperadas. Una red neuronal puede utilizarse para aproximar una función de valor o una función de política. Es decir, las redes neuronales pueden aprender a mapear estados a valores o pares estado-acción a valores Q. En lugar de utilizar una tabla de búsqueda para almacenar, indexar y actualizar todos los posibles estados y sus valores (lo cual es imposible con problemas muy grandes) podemos entrenar una red neuronal en muestras del espacio de estado o acción para aprender a predecir cuan valiosos son estos en relación con nuestro objetivo en el aprendizaje de refuerzo.

Clústering

Los algoritmos de clústering o agrupamiento son **algoritmos de aprendizaje no supervisado que pretenden obtener el conjunto de clases que conforman una determinada colección de valores de entrenamiento.** Suelen realizar una clasificación a partir de un representante, denominado centroide. Este representante es el elemento que mejor divide cada clúster o grupo y los demás se asocian a él por proximidad. Este tipo de algoritmos normalmente son dependientes del número de clases que estimemos que existen. Aunque existen métodos que son capaces de inferir el número de clases que hay.

Para evaluarlos es muy importante la visualización de estos porque no hay un método analítico para saber si la agrupación es del todo correcta si asumimos que los datos no están etiquetados. Podemos, para entrenar el sistema, tener datos etiquetados y con ellos validar el clúster que ha realizado el algoritmo. Pero estos

valores no son tenidos en cuenta por el algoritmo. La medida más utilizada para medir la similitud entre los casos es la matriz de correlación entre los *nxn* casos. Sin embargo, también existen muchos algoritmos que se basan en la maximización de una propiedad estadística llamada verosimilitud. Generalmente, los vectores de un mismo grupo (o clústeres) comparten propiedades comunes. El conocimiento de los grupos puede permitir una descripción sintética de un conjunto de datos multidimensional complejo. De ahí su uso en minería de datos. Esta descripción sintética se consigue sustituyendo la descripción de todos los elementos de un grupo por la de un representante característico de este.

Uno de los algoritmos más conocidos es ***k-means.***

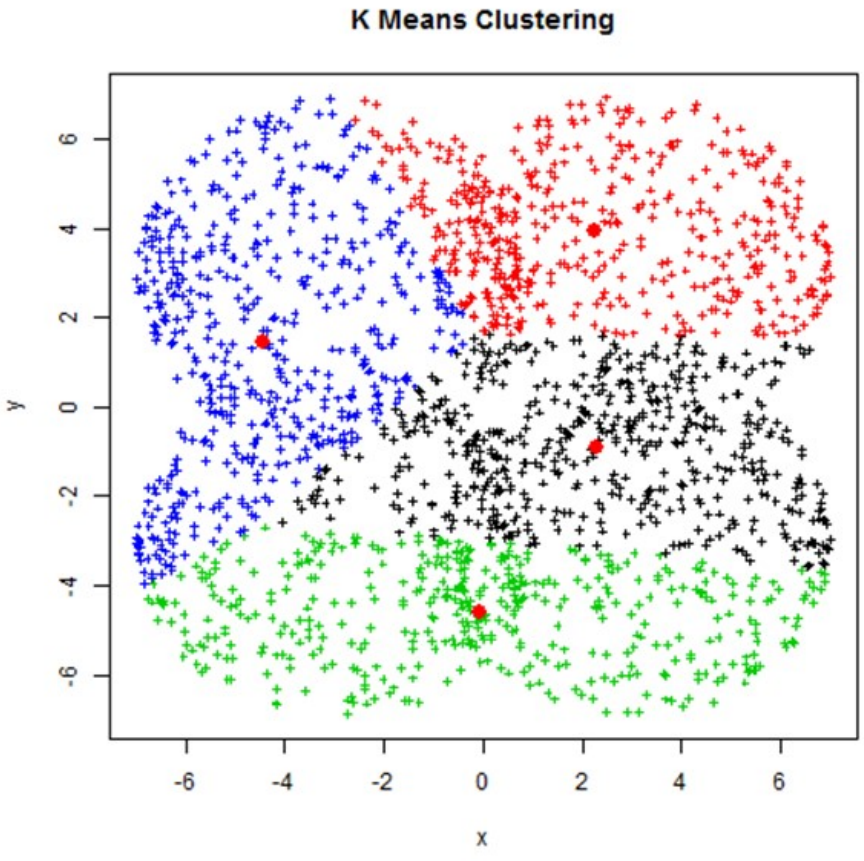


Figura 10. Ejemplo de cómo *k-means* agrupa un conjunto de datos.

Ideas clave

K-means necesita como dato de entrada el número de grupos en los que vamos a segmentar la población. A partir de este número *k* de clúster, el algoritmo coloca primero *k* puntos aleatorios (centroides). Luego asigna a cualquiera de esos puntos todas las muestras con las distancias más pequeñas.

A continuación, el punto se desplaza a la media de las muestras más cercanas. Esto generará una nueva asignación de muestras, ya que algunas muestras están ahora más cerca de otro centroide. Este proceso se repite de forma iterativa y los grupos se van ajustando hasta que la asignación no cambia más moviendo los puntos.

9.5. Proyectos de investigación sobre aprendizaje automático

Los proyectos de investigación en *machine learning* son actualmente los más prolíficos en inteligencia artificial. Pueden ser aplicados a prácticamente todo, pero por hacer un resumen rápido podemos centrarnos en:

- Todo tipo de clasificación automática.
- Reconocimiento de imágenes.
- Programación de agentes autónomos y robótica.
- Regresión.
- Sistemas de recomendación.
- Procesado de imágenes.
- Generación de modelos de predicción de todo tipo.
- Procesado del lenguaje natural.
- Sistemas cognitivos.
- Experiencia de usuario.
- *Lead scoring:* algoritmo que tiene en cuenta una serie de parámetros de los diferentes clientes potenciales que llegan para ordenarlos por prioridad y que los agentes comerciales trabajen de una manera más eficaz.
- Producción inteligente. Con estos proyectos se logra identificar nuevos modelos, tendencias y estacionalidades que permiten a las empresas adelantarse a futuras ventas y cubrir las necesidades del *stock*.

Ideas clave

Los proyectos de *machine learning* suelen requerir de grandes cantidades de computación, por lo que es muy recomendable acudir a soluciones de terceros. Además, como hemos visto, son muy dependientes de la calidad de los datos y de poder acceder a ellos. Se estima que la mayor parte del coste en este tipo de proyectos es la adquisición, procesado y refinamiento de estos datos. Hay que tener en cuenta estas cuestiones a la hora de embarcarse en un proyecto de aprendizaje automático.

Las **principales publicaciones de este medio** son:

- *EEE Transactions on Pattern Analysis and Machine Intelligence.*
- *arXiv: Computer Vision and Pattern Recognition.*
- *Journal of Machine Learning Research.*
- *arXiv: Learning.*
- *Expert System with Application.*
- *International Journal of Computer Vision.*

9.6. Referencias bibliográficas

Marsland, S. (2015). *Machine learning: an algorithmic perspective.* CRC press.

How AI is changing the video game industry: augmentation and synthetic media

AI Business. (26 de febrero de 2020). How AI is changing the video game industry: augmentation and synthetic media. *aibusiness.com.* Recuperado de https://aibusiness.com/author.asp?section_id=789&doc_id=761220

Artículo donde se habla de cómo la IA revolucionará la industria del videojuego.

4 Social Media Data Mining Techniques to Help Grow Your Online Business

Lim, H. (15 de marzo de 2020). 4 Social Media Data Mining Techniques to Help Grow Your Online Business. *hackernoon.com.* Recuperado de https://hackernoon.com/4-social-media-data-mining-techniques-to-help-grow-your-online-business-o6ch32q4

Artículo que describe diferentes técnicas de minería de datos útiles para ayudar al crecimiento del negocio *online*.

Deep Reinforcement Learning

Silver, D. (17 de junio de 2016). *Deep Reinforcement Learning* [Mensaje en un blog]. DeepMind. Recuperado de https://deepmind.com/blog/article/deep-reinforcement-learning

Artículo donde Deep Mind explica cómo funciona el *deep reinforcement learning* y los experimentos que lo demuestran.

Neural Turing Machines

Graves, A., Wayne, G. y Danihelka, I. (1 de junio de 2014). Neural Turing Machines. *deepmind.com.* Recuperado de https://deepmind.com/research/publications/neural-turing-machines

Artículo donde *Deep Mind* donde explica cómo funciona el neural Turing machine.

Test

1. La capacidad de generalización de un algoritmo de *machine learning* viene determinado por:

A. Las máquinas no son capaces de aprender, solo repiten patrones previamente almacenados. Es solo una ilusión de que están aprendiendo.

B. El sesgo que introduce en el aprendizaje.

C. Su capacidad para adaptarse a los datos de entrenamiento.

D. A su similitud con la forma que tiene el cerebro de aprender.

2. Los algoritmos de *machine learning* pueden ser de los siguientes tipos:

A. Supervisado, individual y multivariante.

B. No supervisado, supervisado y clústering.

C. No supervisado, supervisado y por refuerzo.

D. Por refuerzo, clústering y supervisado.

3. El aprendizaje supervisado se caracteriza por:

A. Usar la solución de cada ejemplo como parte del aprendizaje.

B. No conocer la solución de los ejemplos para realizar el aprendizaje.

C. Usar o supervisado, supervisado y por refuerzo.

D. Por refuerzo, clústering y supervisado.

4. El aprendizaje por refuerzo se caracteriza por:

A. No necesitar a un instructor que guie el entrenamiento.

B. Usar el *feedback* del entorno como guía para saber si la acción realizada es la correcta.

C. Selecciona de forma aleatoria la siguiente acción con base en el *feedback* acumulado.

D. Todas las anteriores son correctas.

5. Un problema de clasificación es un problema donde:

A. La variable dependiente es discreta y finita.

B. Es continua e infinita.

C. Es discreta e infinita.

D. Ninguna de las anteriores.

6. ¿Cuál de estas afirmaciones es cierta sobre K-NN?

A. Es un algoritmo de aprendizaje no supervisado que busca almacenar estados antiguos y su solución para recuperarlas posteriormente.

B. Es un algoritmo de aprendizaje supervisado que busca almacenar estados antiguos y su solución para recuperarlas posteriormente.

C. Es un algoritmo de aprendizaje no supervisado que agrupa a los ejemplos por proximidad formando clúster.

D. Ninguna de las anteriores.

7. ¿Cuál de estas afirmaciones es cierta sobre los árboles de decisión?

A. Es un algoritmo no supervisado que permite crear reglas.

B. Segmenta el problema con aproximaciones no lineales.

C. Es un algoritmo de caja negra.

D. Ninguna de las anteriores.

8. ¿Cuál de estas afirmaciones es cierta sobre las redes de neuronas?

A. Es un algoritmo de aprendizaje supervisado muy resistente al ruido.

B. Es un algoritmo de caja negra.

C. Sirve para clasificación y para regresión.

D. Todas las anteriores son correctas.

9. ¿Cuál de estas afirmaciones es cierta sobre el aprendizaje por refuerzo?

A. Utiliza la función de refuerzo que le proporciona el entorno para actualizar la política.

B. Solo se almacena el último *feedback* recibido.

C. El aprendizaje por refuerzo profundo sustituye la función de recompensa por una red de neuronas.

D. Todas las anteriores son correctas.

10. ¿Cuál de estas afirmaciones es cierta sobre el *k-means*?

A . *K-means* agrupa con base en la distancia de un centroide que se le proporciona como ejemplo.

B . *K-means* agrupa con base en la distancia de un centroide que se va calculando en cada iteración como la media de las muestras clasificadas.

C . *K-means* agrupa con base en la distancia de un centroide que se va calculando en cada iteración como el elemento más alejado de los elementos del grupo.

D. Todas las anteriores son falsas.