

Maestría en Análisis y Visualización de Datos Masivos

Técnicas de Inteligencia Artificial

Técnicas de Inteligencia Artificial

Tema 1. Introducción

Índice

[Esquema](#)

[Ideas clave](#)

1.1. ¿Cómo estudiar este tema?

1.2. Aproximación a los conceptos de inteligencia artificial, aprendizaje automático y minería de datos. Interés y aplicaciones

1.3. Definición de aprendizaje, tareas básicas y ejemplos

1.4. Etapas en el descubrimiento de conocimiento

1.5. Referencias bibliográficas

[A fondo](#)

Introducción e historia de la inteligencia artificial

Entrevista a Matilde Santos Peñas sobre la inteligencia artificial

Conciencia artificial y test de Turing

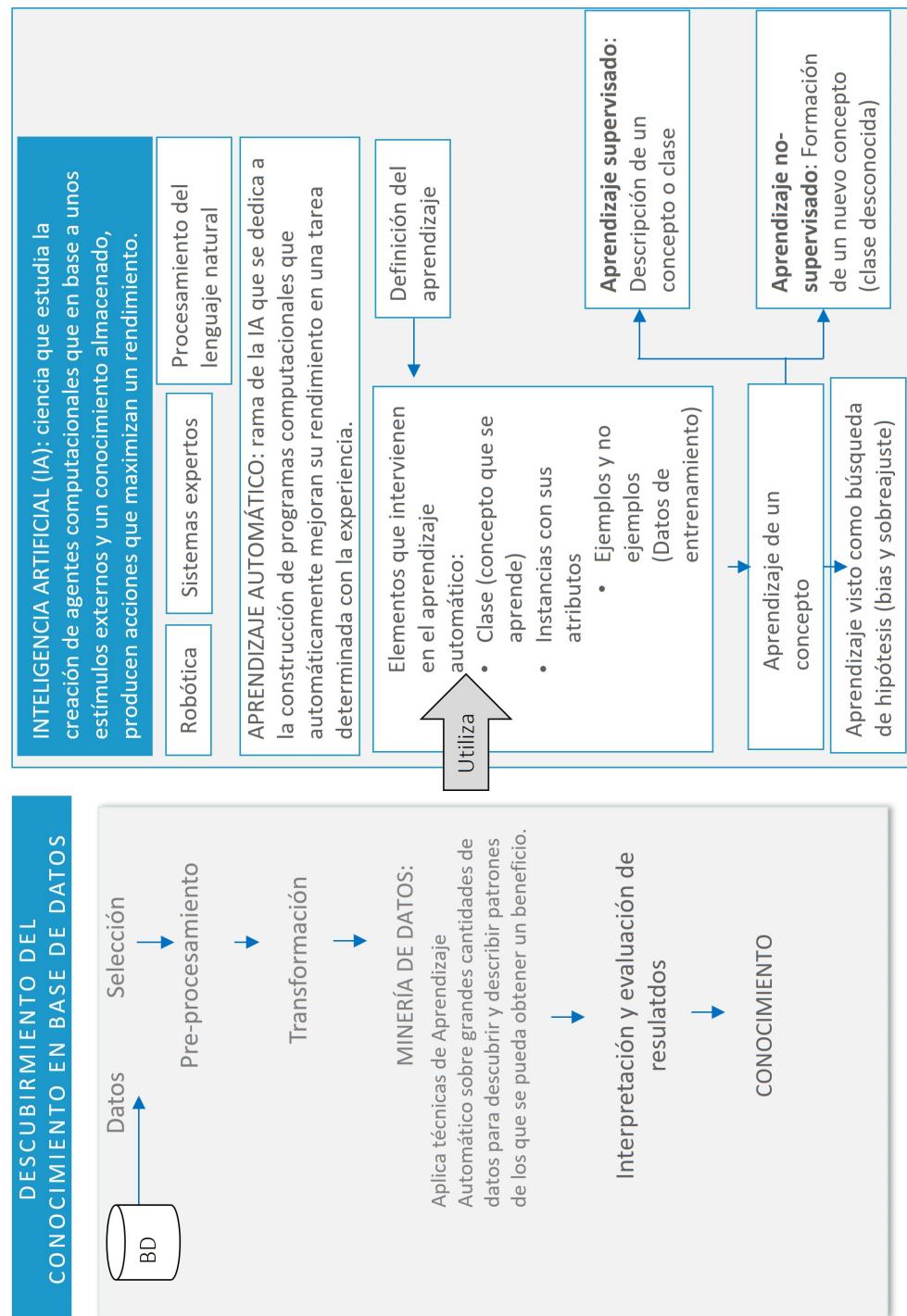
Inteligencia artificial

Presentación introductoria de la minería de datos

AI Topics

[Test](#)

Esquema



1.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral, revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan.

Al comienzo de este tema se describen a alto nivel los conceptos de **inteligencia artificial** y también de la **minería de datos** (Han & Kamber, 2012; Negnevitsky, 2005), la cual aplica técnicas de aprendizaje automático (subrama de la inteligencia artificial) para extraer conocimiento útil de *data warehouses*, *data lakes* (Khine & Wang, 2018) o bases de datos más sencillas. Muestra además la importancia de estos campos y su gran diversidad de aplicaciones.

A continuación, se explican conceptos fundamentales del **aprendizaje automático**: definición de aprendizaje, elementos que intervienen en una tarea de aprendizaje, diversos tipos de aprendizaje acompañados de ejemplos, etc., (Mitchell, 1997; Witten & Frank, 2005). Todos ellos son conceptos importantes en esta asignatura, los cuales aparecerán repetidamente en temas posteriores junto con otros como datos de entrenamiento, instancias, clases, *bias* o sobreajuste.

Al finalizar el tema se describen los pasos de un proceso típico de **descubrimiento de conocimiento en bases de datos**, uno de los cuales consiste precisamente en la aplicación de un proceso de minería de datos para extraer patrones de los datos.

Al finalizar el estudio de este tema serás capaz de:

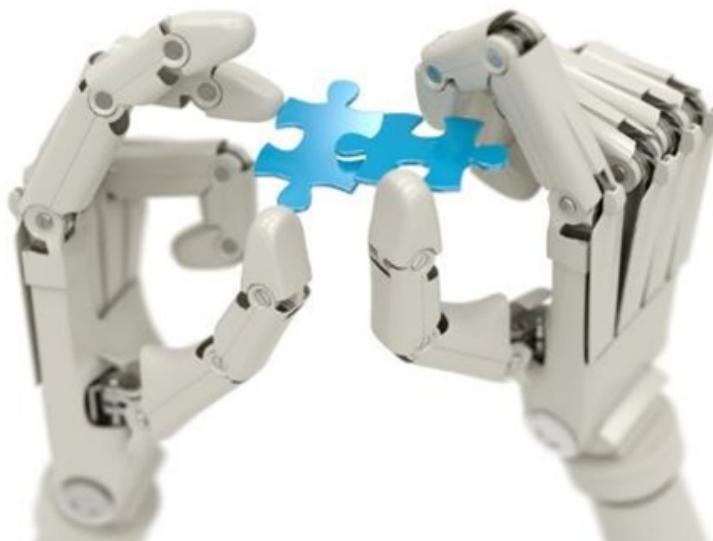
- ▶ Distinguir los conceptos de inteligencia artificial, aprendizaje automático y minería de datos, identificar su interés y posibles aplicaciones.
- ▶ Definir el aprendizaje automático y tareas básicas de descripción o formación de conceptos.
- ▶ Definir los elementos que intervienen en el aprendizaje de conceptos.
- ▶ Identificar las etapas que comprenden el diseño de un sistema de aprendizaje.
- ▶ Identificar las etapas de un procedimiento típico de descubrimiento de conocimiento en bases de datos.

1.2. Aproximación a los conceptos de inteligencia artificial, aprendizaje automático y minería de datos. Interés y aplicaciones

En el Diccionario de la Lengua Española de la Real Academia Española podemos encontrar la siguiente **definición de Inteligencia Artificial (IA)** (ASALE & RAE, s. f.): «Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico». Estas operaciones y procesos de la inteligencia humana son los necesarios para el aprendizaje, la comprensión, la resolución de problemas o la toma de decisiones. Por tanto, la inteligencia artificial es una disciplina amplia que reúne varios campos como el aprendizaje automático, los sistemas expertos, sistemas RPA (*Robotic Process Automation*), la robótica o los sistemas de procesamiento de lenguaje natural.

El objetivo de la IA, desde el punto de vista de la investigación y de la ciencia, es comprender los principios que hacen posible el comportamiento inteligente en sistemas artificiales. Para ello, se deben analizar agentes naturales y artificiales, formular y testear hipótesis sobre lo que implica construir un sistema artificial que realice tareas que requieren inteligencia, así como diseñar y desarrollar el sistema inteligente empírico, esto es, experimentando y comprobando las distintas hipótesis planteadas (Poole & Mackworth, 2010).

Existe mucha controversia y debate sobre si las máquinas pueden ser inteligentes y pensar. Una afirmación interesante es la que se puede encontrar en (Poole & Mackworth, 2010) donde el autor indica que «se puede considerar una máquina inteligente si actúa de manera inteligente, creando inteligencia real de una manera artificial o no natural». Igualmente, de acuerdo a Negnevitsky (Negnevitsky, 2005), una máquina es considerada inteligente si puede conseguir un rendimiento igual al de un humano en una tarea cognitiva.



En gran parte de los libros de inteligencia artificial, cuando se habla de la historia de esta ciencia y sobre el debate del pensamiento artificial en las máquinas, se hace referencia al artículo de **Alan Turing** con título «Computing machinery and intelligence» publicado en el año 1950 (Turing, 1950). Este es uno de los artículos más relevantes y antiguos sobre la inteligencia de las máquinas.

Turing definió el comportamiento inteligente de un ordenador como la «habilidad de conseguir un rendimiento similar al de un humano en tareas cognitivas». Turing propone un juego de imitación en el que un juez humano debe interrogar a otro humano y a un ordenador remotamente y conseguir adivinar quién es el ordenador de sus dos interlocutores. Para que el ordenador no sea descubierto debe, por tanto, responder como un humano lo haría, hacer cálculos con la rapidez de un humano, titubear e incluso enfadarse si las preguntas son provocadoras.

A modo de curiosidad, según la referencia (Ohlsson *et al.*, 2017), la inteligencia artificial ha alcanzado en este sentido el nivel de un niño de 4 años. En la práctica no es habitual la creación de programas con el objetivo de que se comporten intelectualmente como humanos, sino que **se buscan programas que ayuden a las personas a procesar grandes cantidades de datos, tomar decisiones o realizar cálculos rápidos**. Sin embargo, aunque en principio no parezca completamente útil imitar el comportamiento humano, el test de Turing sienta precisamente las bases de un método de validación de sistemas expertos que consiste en la comparación del rendimiento del sistema inteligente con el del rendimiento de varios humanos expertos en una determinada área de conocimiento.

En esta asignatura no se estudia la IA desde un punto de vista teórico o científico, sino desde el punto de vista práctico de la ingeniería. A continuación, se plantea una definición funcional y pragmática de esta ciencia:

Inteligencia artificial

Es una rama de la informática que estudia la creación de agentes computacionales que reciben estímulos externos y, en base a ellos y a un **conocimiento** almacenado en dicho agente, producen resultados o acciones que **maximizan una medida de rendimiento**. El conocimiento almacenado puede ser aprendido por el mismo agente utilizando técnicas de aprendizaje automático o puede ser incorporado por un humano experto en el dominio específico.

En este sentido, y en relación con los **grandes conjuntos de datos (Big Data)**, es de gran interés el campo de la **Minería de Datos**, cuyo objetivo es eminentemente práctico.

Minería de datos

Es un proceso que utiliza técnicas de inteligencia artificial sobre grandes cantidades de datos con el objetivo de descubrir y describir patrones en los datos, a partir de los cuales se pueda obtener un beneficio.

Las fuentes de datos pueden incluir bases de datos, *data warehouses*, *data lakes*, repositorios o información en la web. Las técnicas que en concreto utiliza la minería de datos son esencialmente las denominadas técnicas de **aprendizaje automático**.

El **aprendizaje automático**, una rama de la IA, se refiere a la construcción de programas computacionales que automáticamente mejoran su rendimiento en una tarea determinada con la experiencia.

Así, la minería de datos utiliza técnicas de aprendizaje automático para, por ejemplo, aprender a detectar el uso fraudulento de tarjetas de crédito. A partir de datos de experiencias previas de usos fraudulentos y no fraudulentos de tarjetas de crédito, mediante la aplicación iterativa de técnicas de aprendizaje automático, se puede mejorar en la tarea de detectar usos fraudulentos.

Por otra parte, el aprendizaje automático tiene múltiples aplicaciones en otro tipo de

sistemas como en aquellos relacionados con la robótica, o en sistemas de reconocimiento de habla, por ejemplo (Rogers & Girolami, 2017). En el siguiente apartado se explican conceptos relacionados con el aprendizaje automático que serán de gran interés y uso a lo largo de la asignatura.

En la Figura 1 podemos ver las diferentes ramas de la inteligencia artificial y el espacio que ocupa en dicha taxonomía el aprendizaje automático o *machine learning*, así como sus diferentes subramas.

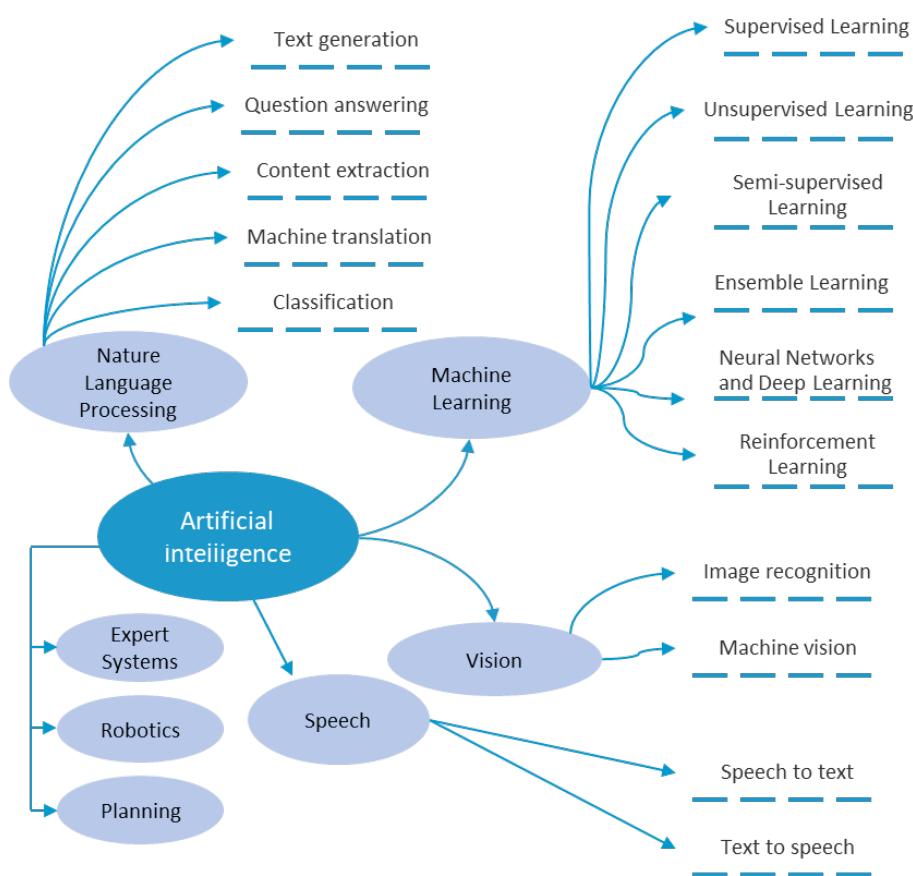


Figura 1. Taxonomía del machine learning dentro de la Inteligencia Artificial. Basado en Panesar, 2019.

Interés y aplicaciones de la inteligencia artificial

La inteligencia artificial tiene aplicaciones innumerables en diversas áreas tales como la robótica, los juegos de ordenador, el *marketing*, la medicina, o la predicción meteorológica (Pannu, 2015).

De hecho, hay muy diversos tipos de problemas que pueden ser resueltos por sistemas inteligentes, como los que se enumeran a continuación:

- ▶ **Diagnóstico:** inferir funcionamientos incorrectos de un objeto a partir de su comportamiento y recomendar soluciones.
- ▶ **Selección:** recomendar la mejor opción de una lista de alternativas posibles.
- ▶ **Predicción:** predecir el comportamiento futuro de un objeto a partir de su comportamiento en el pasado.
- ▶ **Clasificación:** asignar un objeto a una clase definida.
- ▶ **Agrupamientos (*clustering*):** agrupar objetos de acuerdo con sus características.
- ▶ **Optimización:** mejorar la calidad de las soluciones hasta encontrar una óptima.
- ▶ **Control:** gestionar el comportamiento de un objeto en tiempo real para satisfacer ciertos requisitos especificados.

Específicamente, en lo que respecta a las «grandes cantidades de datos», por todos es bien conocido el crecimiento inminente de la cantidad de datos que almacenan y manejan cada vez más y más instituciones y empresas en los últimos tiempos. Aquellas organizaciones que consiguen extraer conocimiento a partir de esos datos y optimizar su aplicación son las que tienen mayor oportunidad de mantenerse

competitivas.

Por ello, dado que la necesidad es la madre de la invención, la aplicación de técnicas de inteligencia artificial a cantidades masivas de datos para su análisis automático (la minería de datos) es un campo que está en pleno auge y en constante evolución. Pueden encontrarse ejemplos de aplicación en múltiples ámbitos, como el médico, el económico-empresarial, el industrial o el marketing, entre muchos otros.

En el campo de la **medicina** se encuentran multitud de aplicaciones de las técnicas de inteligencia artificial. Por ejemplo, analizando grandes cantidades de datos sobre una enfermedad, se puede predecir el diagnóstico y pronóstico de enfermedades. Así, conociendo ciertos síntomas y características de un determinado paciente, se puede diagnosticar con bastante probabilidad una enfermedad y establecer un tratamiento óptimo para dicho paciente (Peng *et al.*, 2020).

El uso de las técnicas de inteligencia artificial en medicina está muy extendido. Dos ejemplos de enfermedades que pueden ser diagnosticadas son, por ejemplo, el Alzheimer o el glaucoma (Farooq *et al.*, 2017). Por otro lado, y como segundo ejemplo, existen sistemas inteligentes basados en heurísticas y reglas para establecer el mejor emparejamiento posible entre un donante y los posibles receptores de un trasplante (Aguado *et al.*, 2019; Namatevs & Aleksejeva, 2017).

En la **educación** también se encuentran aplicaciones de sistemas inteligentes. Los denominados sistemas tutores inteligentes tratan de emular las pautas que los tutores humanos dan para optimizar el estudio del alumno. Por ejemplo, hay sistemas inteligentes que personalizan los contenidos que se presentan al estudiante en función de su progreso en el aprendizaje o de su estilo de aprendizaje (Alonso *et al.*, 2019). También hay sistemas que analizan la solución que un estudiante da a un problema y tratan de guiarle, ofreciendo pistas y otras pautas, para que consiga resolver el problema (García *et al.*, 2015).

Otro campo donde se utilizan mucho las técnicas de inteligencia artificial, en concreto

de aprendizaje automático, es en el **marketing**, en **sistemas de gestión de clientes** (comúnmente denominados por sus siglas en inglés CRM — Customer Relationship Management) y **posicionamiento de productos** (Sterne, 2017; Zeeshan & Saxena, 2020).

La **industria**, y de forma más concreta en torno al paradigma de la **Industria 4.0**, es otro de los grandes beneficiados de la aplicación e implementación de técnicas de inteligencia artificial en sus procesos. La fabricación inteligente, el mejor uso de los datos que genera la empresa a través de sistemas *business intelligence* avanzados, o la implementación de soluciones de robótica (Narasima Venkatesh, 2018; Bayram & İnce, 2018; Cevik Onar & Ustundag, 2018; Sami Sivri & Oztaysi, 2018).

En la **gestión empresarial** también se encuentran aplicaciones inteligentes para, entre otros, la planificación eficiente de recursos o como apoyo a la toma de decisiones, por ejemplo, aquellas relacionadas con el análisis de la solvencia empresarial (Jain *et al.*, 2020; Malhotra *et al.*, 2017).

Un ejemplo que habitualmente se encuentra en los libros de IA o minería de datos es el de uso de un sistema inteligente para la toma de decisiones en lo que respecta a la concesión de préstamos (Witten & Frank, 2005).

De hecho, los bancos son de las primeras entidades que adoptaron soluciones de minería de datos con este fin. Posteriormente, los bancos han ido adoptando estas soluciones como parte de sus sistemas CRM, para el apoyo a la gestión de las relaciones con los clientes, la venta y el **marketing**. Por ejemplo, con estas técnicas, los bancos pueden modelar patrones de comportamientos de aquellos clientes que pierden para poder detectar a priori los potenciales clientes perdidos y tomar las acciones correctivas convenientes.

Por otra parte, los sistemas de comercio electrónico (como Amazon) o los proveedores de contenidos digitales (como Netflix o Spotify) pueden beneficiarse de los sistemas «recomendadores» que predicen el interés de un usuario por ciertos productos a partir de las valoraciones dadas a diversos productos por otros muchos usuarios con perfil similar (Joshi, 2020; Linden *et al.*, 2003).

También es frecuente encontrar sistemas inteligentes para el **diagnóstico y el troubleshooting**, detección de problemas y propuesta de soluciones, como por ejemplo los que tratan de detectar un fallo en una red de ordenadores y guían al administrador de la red en el proceso de diagnóstico y corrección del error, así como en los fallos de los propios ordenadores (Akinnola, 2012; Elmishali *et al.*, 2018; Nushi *et al.*, 2017).

Existen también sistemas inteligentes que proporcionan **rutas óptimas**, ya sean de transporte de mercancías (Abduljabbar *et al.*, 2019; Boru *et al.*, 2019; Mohammed *et al.*, 2017) o en transmisión de datos en redes telemáticas, por ejemplo, en base a ciertos requerimientos como pueda ser la ruta de menor coste o la ruta más rápida (Mali & Gautam, 2018; Matlou & Abu-Mahfouz, 2017).

Referido al **consumo energético**, existen múltiples soluciones en las que la inteligencia artificial sirve como soporte a otros sistemas que son utilizados para reducir el consumo energético. Por ejemplo, dando soporte a la localización en tiempo real para reducir el consumo energético en edificios públicos (García *et al.*, 2017), o bien integrando localización y sensorización para fomentar el ahorro energético en hogares (Óscar García *et al.*, 2017). Igualmente, en la industria de suministro eléctrico se utilizan técnicas inteligentes para predecir la demanda de electricidad por tramos horarios. De esta manera, las compañías suministradoras pueden planificar por ejemplo acciones de mantenimiento (Jozi *et al.*, 2019, 2017; Silva *et al.*, 2020; Vinagre *et al.*, 2016).

En **meteorología** y para la predicción o gestión de catástrofes también se han utilizado sistemas inteligentes (McGovern *et al.*, 2017), así como en **agricultura** para la planificación agrícola, el control de plagas o la gestión de cultivos (Liakos *et al.*, 2018). El artículo (Alonso *et al.*, 2020) expone una solución IoT aplicada en la agricultura y la ganadería en la que se monitorizan cultivos y ganado para optimizar su producción.

Incluso se pueden encontrar ejemplos curiosos en el campo de la **arqueología**, donde la inteligencia artificial se ha utilizado recientemente para descubrir nuevos yacimientos arqueológicos (Davis, 2020; Traviglia & Torsello, 2017).

Todos estos ejemplos son solo una pequeña muestra que pretende ilustrar la gran utilidad de la aplicación de la IA en muy diversos campos.

1.3. Definición de aprendizaje, tareas básicas y ejemplos

Dado que la inteligencia se adquiere mediante la experiencia y el aprendizaje, el concepto de aprendizaje es muy importante en el campo de la inteligencia artificial, siendo precisamente muy populares las técnicas de la rama de aprendizaje automático.

La siguiente frase define el aprendizaje por parte de un ordenador:

Un programa de ordenador aprende de la experiencia E con respecto a una clase de tareas T y una medida de rendimiento P, si su rendimiento en las tareas T, medido en base a la medida P, mejora con la experiencia E. (Mitchell, 1997).

Para ilustrar esta definición y los elementos E, P y T, se exponen a continuación algunos ejemplos de tareas de aprendizaje:

Ejemplo 1: Aprender a detectar robos de tarjetas de crédito.

- ▶ T: detectar robos de tarjetas de crédito.
- ▶ P: porcentaje de robos detectados.
- ▶ E: base de datos de hábitos de compra con la tarjeta de crédito.

Ejemplo 2: Aprender a reconocer la escritura manual.

- ▶ T: reconocer y clasificar palabras escritas en imágenes.
- ▶ P: porcentaje de palabras correctamente clasificadas.
- ▶ E: base de datos de imágenes de palabras manuscritas clasificadas.

Ejemplo 3: Aprender a aparcar un coche utilizando sensores de visión.

- ▶ T: aparcar un coche utilizando sensores de visión.
- ▶ P: porcentaje de aparcamientos correctos.
- ▶ E: secuencias de imágenes y comandos de guiados registrados.

¿Qué contenidos se pueden aprender?

De acuerdo con la teoría de instrucción de Merrill denominada «Teoría de presentación de componentes», «*Component display theory*» (Merril *et al.*, 1994), **las personas pueden aprender** cuatro tipos de contenido:

- ▶ **Hechos:** simples afirmaciones de una verdad, que puede ser una asociación entre una fecha y un hecho, o un nombre y un objeto.
- ▶ **Conceptos:** conjunto de objetos, símbolos o eventos agrupados porque comparten ciertas características y que pueden ser referenciados por un nombre en particular o un símbolo. Los objetos existen en el espacio y tiempo como puede ser una persona, una mesa; los símbolos se refieren a tipos de palabras, números, marcas, como puede ser un predicado o una fracción; los eventos son interacciones específicas de objetos en un periodo de tiempo como puede ser la digestión o la fotosíntesis.
- ▶ **Procedimientos:** conjunto de acciones realizadas en pasos consecutivos para alcanzar un objetivo.
- ▶ **Principios:** relaciones causa-efecto, verdades generales o leyes básicas para afirmar otras verdades.

Específicamente, los ordenadores pueden ser muy eficaces en el **aprendizaje de conceptos** que pueden ser representados mediante distintos tipos de estructuras como árboles, reglas o funciones matemáticas, como se verá más adelante en la asignatura.

También existen sistemas del campo de la robótica, por ejemplo, cuyo objetivo es aprender un procedimiento. Estos sistemas tratan de actuar como humanos y alcanzar objetivos mediante la ejecución de una serie de tareas. Sin embargo, hasta el momento, estos robots no superan en rendimiento a las personas.

¿Qué elementos intervienen en el aprendizaje de un concepto?

Como anteriormente se ha expuesto, el programa de ordenador aprende en base a una experiencia E, que podría ser una base de datos de información sobre transacciones bancarias o una secuencia de imágenes, tal y como se ha visto en los anteriores ejemplos.

Cada una de estas imágenes o cada una de las transacciones que forman parte de la experiencia son denominadas **instancias**.

Una **instancia** es una ilustración específica de un objeto, símbolo, evento, proceso o procedimiento (Merrill, 1994).

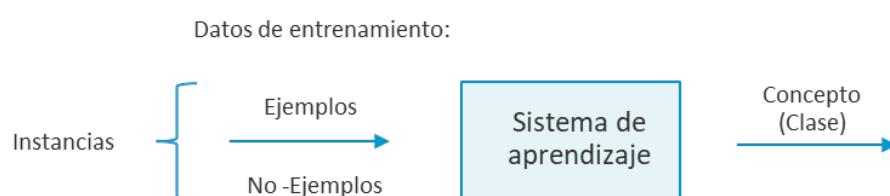


Figura 2. Entradas y salidas de un sistema de aprendizaje.

Como anteriormente se ha definido, el concepto será un conjunto de estas instancias que comparten ciertas características. A estas características se les denomina habitualmente **atributos**. Nótese que la mayoría de las palabras de nuestro idioma se refieren a categorías o clase de objetos y no a los objetos específicos únicos (por ejemplo: perro, gato, mesa, etc.). Al concepto habitualmente se le denomina **clase**. El concepto se puede aprender tanto a partir de instancias que pertenecen a la clase, como a partir de instancias que no pertenecen a la misma.

Por ejemplo, si queremos aprender a detectar usos fraudulentos de tarjetas podemos utilizar tanto datos de transacciones pertenecientes a aquellas habituales de cada cliente como datos de transacciones fraudulentas. Por tanto, la instancia es un término que se refiere tanto a miembros como a no-miembros de una clase.

En el ejemplo previo, las instancias son todas las transacciones registradas en la base de datos. Las instancias que son miembros del concepto en consideración se denominan **ejemplos**, mientras que las que no son miembros se denominan **no-ejemplos** (ver Figura 2). Por ejemplo, si queremos detectar usos fraudulentos a partir de describir las transacciones legales, los ejemplos serán aquellas transacciones legales mientras que los no-ejemplos serán las transacciones no legales.

El conjunto de instancias que forman parte de la experiencia E, que utiliza el sistema para aprender la tarea T, tanto ejemplos como no ejemplos, recibe el nombre de **datos de entrenamiento**.

¿En qué consiste aprender un concepto?

Se aprende un concepto cuando, dado un objeto (símbolo o evento), se puede identificar correctamente el concepto o clase a la que pertenece ese objeto, pudiéndose generalizar la aplicación del nombre de la clase a todos los miembros de esta y discriminando a los miembros que pertenecen a otra clase.

De esta afirmación obtenemos tres nuevos términos, utilizados habitualmente en el

aprendizaje automático:

- ▶ **Clasificación:** identificar la clase de un símbolo específico, objeto o evento; también identificar símbolos, objetos o eventos que no son miembros de una clase
- ▶ **Generalización:** identificar la clase de una instancia desconocida examinando atributos comunes de esa instancia con ejemplos encontrados previamente de esa clase.
- ▶ **Discriminación:** el hecho contrario a generalizar, esto es, dada una instancia desconocida, identificarla como no-miembro de la clase dado que no se encuentran atributos comunes con ejemplos previos de la clase.

En la Figura 3 se muestra un ejemplo muy sencillo. Una vez aprendido el concepto de «animal salvaje», se puede clasificar un animal a priori de clase desconocida dentro de la clase «animal salvaje» dados sus atributos (ej. «el animal vive en la jungla») — **generalización** — y se es capaz de discriminar un animal que no es salvaje por sus atributos (ej. «el animal es criado en una granja») — **discriminación**.

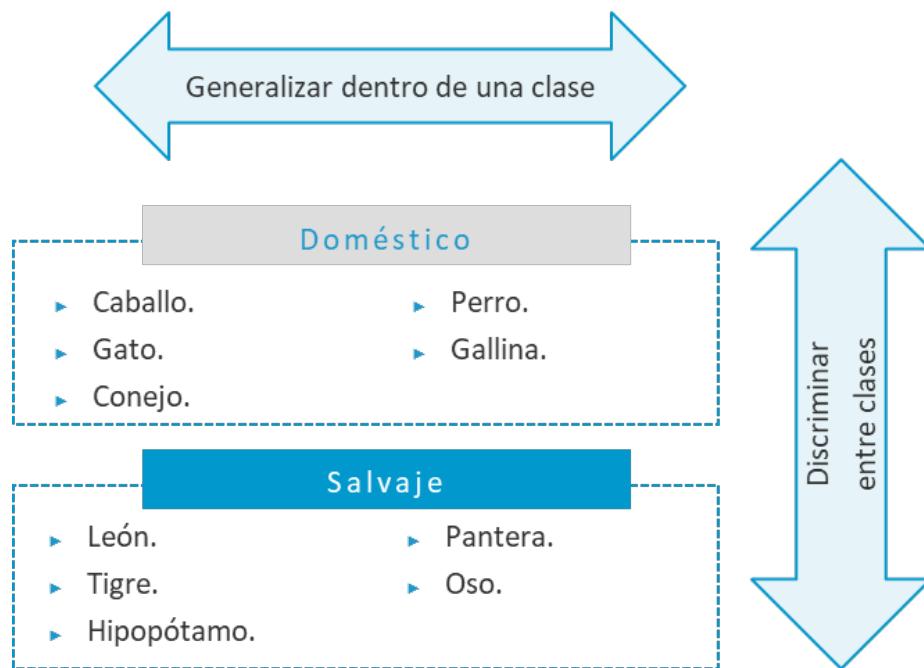


Figura 3. Ejemplo de generalización y discriminación de clases.

Mediante el aprendizaje se pueden resolver diferentes tipos de tareas básicas como la **descripción de conceptos** o la **formación de nuevos conceptos**. Si se ha aprendido el concepto de animal doméstico en base a los atributos de ejemplos bien conocidos de animales domésticos, se habla de una tarea de descripción de conceptos. A este tipo de aprendizaje se le denomina **aprendizaje supervisado**. Sin embargo, cuando en base a unos ejemplos se quiere aprender un nuevo concepto desconocido que los describa, esto es, formar un nuevo concepto, se habla de un **aprendizaje no-supervisado**.

El aprendizaje visto como una búsqueda

El aprendizaje de conceptos se plantea a menudo como una búsqueda en un espacio de posibles hipótesis (esto es, posibles soluciones al problema de aprendizaje) con el fin de encontrar la hipótesis que mejor encaje con los datos de entrenamiento. En este aprendizaje inductivo se puede garantizar que la hipótesis encontrada es la que mejor encaja con los datos de entrenamiento, pero ¿encajará

esa hipótesis también con nuevas instancias? Se asume que sí y se plantea, por tanto, la siguiente hipótesis:

Hipótesis del aprendizaje inductivo de conceptos

Cualquier hipótesis que encaje «suficientemente» bien con un conjunto <<suficientemente» grande de ejemplos de entrenamiento también encajará bien con instancias nuevas.

Esta hipótesis del aprendizaje inductivo permite la generalización. La dificultad está en determinar el espacio de hipótesis posibles en un problema de tal forma que se evite dejar fuera del espacio a la mejor hipótesis. Igualmente puede ser difícil determinar el tamaño de este espacio adecuado, así como el tamaño del conjunto de datos de entrenamiento requerido, de tal manera que se pueda generalizar la solución. Surge aquí el término «**bias**» inductivo.

El bias se refiere a los criterios de selección de las hipótesis y aquellos supuestos y suposiciones que se realizan para generalizar la mejor hipótesis encontrada.

Realísticamente hablando, en muchos problemas se pueden encontrar diferentes posibles descripciones de un concepto que encajan «razonablemente bien» con los datos. Por tanto, se trata de encontrar la que mejor encaja respecto a algún criterio, como la simplicidad. En muchas ocasiones no es viable realizar una búsqueda por un espacio de hipótesis completo, no pudiéndose garantizar, por tanto, que la hipótesis encontrada es la mejor y hay que determinar el probablemente mejor procedimiento de búsqueda de soluciones para el determinado problema.

Por otra parte, en la búsqueda de las mejores hipótesis en base a los datos de entrenamiento, muchas veces se encuentran soluciones demasiado específicas, que cubren de manera muy exacta los datos de entrenamiento. Esto puede generar un problema de **sobreajuste** (*overfitting*), siendo la solución adoptada demasiado específica respecto a los datos de entrenamiento y no se generaliza bien.

Habitualmente, para detectar el posible sobreajuste, se reserva un porcentaje de instancias del conjunto de partida que no intervienen en el entrenamiento, sino que, una vez encontrada la mejor hipótesis se utilizan para corroborar que la generalización de la solución es posible.

Sobre diversos *bias*, el problema del sobreajuste y cómo solucionarlo, se irá profundizando a lo largo de la asignatura, en el estudio de diversas técnicas de aprendizaje automático.

Aprendizaje supervisado

Un niño aprende conceptos viendo ejemplos de esos conceptos, ya sean perros, niños, niñas, mesas o coches. Los niños buscan características (atributos) comunes en, por ejemplo, los diferentes ejemplos denominados «perro» por los adultos.

Analizando estas características comunes, el niño es capaz de clasificar correctamente nuevas instancias que se encuentra de la clase «perro», sea cual sea la raza del perro. El niño ha creado un modelo de clasificación en base a ejemplos de perros clasificados por los adultos, examinando aquellos atributos que definen la clase. Este tipo de aprendizaje se denomina aprendizaje supervisado.

El **aprendizaje supervisado** pretende caracterizar o describir un concepto a partir de instancias del mismo.

El aprendizaje supervisado suele tener dos etapas, tal y como vemos en la Figura 4.



Figura 4. Etapas del aprendizaje supervisado.

Este tipo de aprendizaje es típicamente utilizado para diagnosticar enfermedades. En la Tabla 1 se muestra una base de datos de un centro médico a modo de ejemplo. En la primera fila se muestran los nombres de los atributos. «Número de paciente», «Fiebre», «Dolor de garganta», «Congestión» y «Dolor de cabeza» corresponden a los atributos de entrada y se refieren a los distintos síntomas presentados por los pacientes que han sido diagnosticados de distintas enfermedades. La columna «Diagnóstico» es el atributo de salida o clase, que quiere ser descrito en base a las instancias (datos o atributos de pacientes que ya han acudido a consulta) con el fin de poder predecir el diagnóstico de futuros pacientes en función de los valores de sus atributos de entrada (síntomas).

Número de paciente	Fiebre	Dolor de garganta	Congestión	Dolor de cabeza	Diagnóstico
1	Sí	Sí	No	Sí	Infección de garganta
2	No	No	Sí	No	Alergia
3	No	No	Sí	Sí	Resfriado
4	No	No	Sí	No	Alergia
5	Sí	Sí	Sí	Sí	Infección de garganta
6	No	No	Sí	No	Resfriado
7	Sí	No	Sí	Sí	Resfriado
8	Sí	Sí	No	Sí	Infección de garganta
9	No	No	Sí	Sí	Resfriado

Tabla 1. Ejemplo de base de datos para el diagnóstico de enfermedades. Fuente: elaboración propia.

Tomando los ejemplos de la , en primer lugar se pretende describir las diferentes clases (diagnósticos) aplicando una técnica de aprendizaje automático, como por ejemplo la denominada «árbol de decisión», así como obtener una representación como la mostrada en la , que representa una generalización de los datos de la Tabla 1.

Los datos de los pacientes 1 a 9, utilizados para crear el modelo expuesto en el árbol de decisión, corresponden a los **datos de entrenamiento**. Se suele utilizar unos **datos de prueba** también, cuya clasificación es bien conocida para verificar si la clasificación modelada se puede generalizar. En el siguiente tema se profundizará en estos nuevos conceptos. En este ejemplo concreto se han utilizado los datos de los 9 pacientes como datos de entrenamiento y se ha ejecutado el algoritmo de construcción de árboles de decisión ID3 (con la herramienta [Weka](#)) que se explicará en un tema posterior. Como se puede observar en la Figura 5, esta representación en árbol es muy fácil de interpretar por las personas, dado que es visualmente muy intuitiva, ayudando además el hecho de ser un ejemplo muy sencillo.

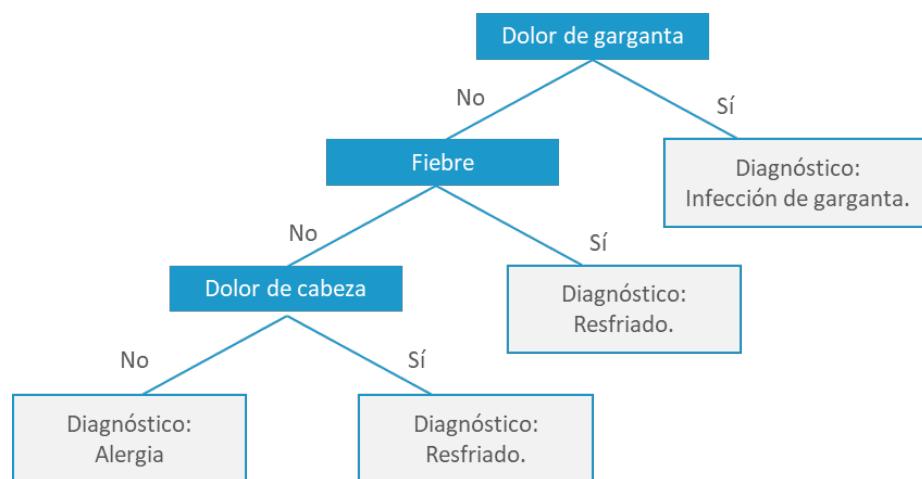


Figura 5. Árbol de decisión correspondiente a los datos de la Tabla 1.

Estos árboles de decisión son fácilmente trasladables a reglas de clasificación, representación igualmente muy intuitiva, como las que siguen:

SI “Dolor de garganta”=Sí

ENTONCES Diagnóstico=“Infección de garganta”

SI “Dolor de garganta”=No AND “Fiebre”=Sí

ENTONCES Diagnóstico=“Resfriado”

SI “Dolor de garganta”=No AND “Fiebre”=No AND “Dolor de cabeza”=Sí

ENTONCES Diagnóstico=“Resfriado”

SI “Dolor de garganta”=No AND “Fiebre”=No AND “Dolor de cabeza”=No

ENTONCES Diagnóstico=“Alergia”

Cuando llegan nuevos pacientes a la consulta cuyos datos se exponen en la Tabla 2 se podrá predecir la clase en base a la generalización expuesta en el árbol de decisión o en las reglas previamente indicadas.

Número de paciente	Fiebre	Dolor de garganta	Congestión	Dolor de cabeza	Diagnóstico
10	Sí	No	Sí	Sí	?
11	No	No	Sí	No	?

Tabla 2. Datos de atributos de entrada (síntomas) de nuevos pacientes.

Así, el paciente número 10 se le diagnostica «Resfriado» mientras que al número 11 se le diagnostica «Alergia». Se trata de un ejemplo muy sencillo con un número de instancias muy bajo, cuyo objetivo ha sido únicamente ilustrar en qué consiste el aprendizaje supervisado. En este ejemplo, el atributo «Número de Paciente» fue eliminado como dato de entrada, ya que no aporta información relevante. Por otra parte, el atributo «congestión» ha sido eliminado por el propio algoritmo de aprendizaje, dado que es un atributo no considerado relevante en la clasificación del diagnóstico.

Aprendizaje no-supervisado

El aprendizaje no-supervisado pretende caracterizar un concepto desconocido a partir de instancias de este. En este caso no existen clases definidas y por tanto se trata de describir un nuevo concepto o clase.

Las técnicas de **agrupamiento** o **clustering** son muy utilizadas en problemas de aprendizaje no-supervisado. Mediante *clustering* las instancias se agrupan de acuerdo con un esquema de similitud. En este tipo de aprendizaje, el no-supervisado, los datos de entrenamiento no especifican qué se está intentando aprender (los agrupamientos), mientras que, en el aprendizaje supervisado, las clases que se están intentando describir sí están especificadas.

El **clustering** se puede utilizar para el **análisis de datos** como un primer paso en la construcción de un modelo de estos. El algoritmo de *clustering* típicamente proporciona visualizaciones como la expuesta en la Figura 6, a partir de las cuales se pueden encontrar atributos similares en las instancias y tratar de extraer conclusiones.

Por ejemplo, en una investigación de mercado las técnicas de agrupamiento pueden ser muy útiles porque permiten agrupar a los consumidores, examinar propiedades comunes de los distintos grupos, que permitan obtener pistas o conclusiones con el fin de posicionar un producto, segmentar mercado, etc.

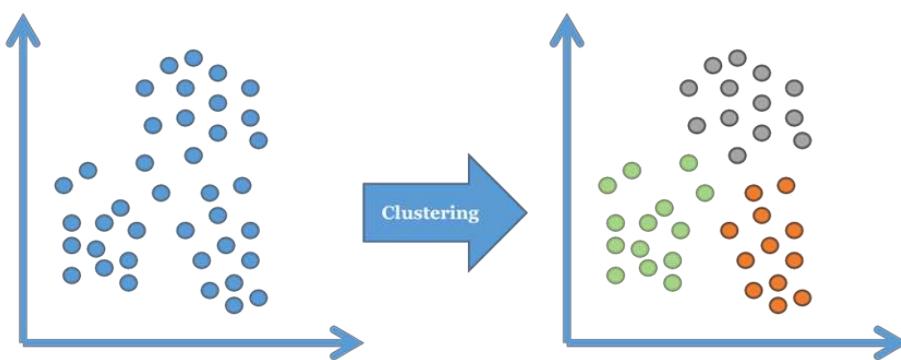


Figura 6. Ilustración ejemplo de aplicación de clustering a un conjunto de datos.

El *clustering* se puede utilizar además en **tareas de generalización**, descubriendo instancias similares, que comparten propiedades, y pudiendo incorporar futuras instancias en los agrupamientos generados.

Por ejemplo, un sistema de aprendizaje *online* puede utilizar una técnica de *clustering* para agrupar alumnos con atributos similares con el fin de, analizado el posible significado de los valores de estos atributos, proporcionarles estrategias de enseñanza adecuadas. Cuando un nuevo alumno se incorpora al sistema y hay datos disponibles, se le incorpora al agrupamiento donde se encuentran los otros alumnos más similares a él, y se le aplican las mismas estrategias de enseñanza.

¿Qué etapas comprende el aprendizaje de un concepto?

La Figura 7 muestra las etapas de aprendizaje de un concepto que se describen a continuación.

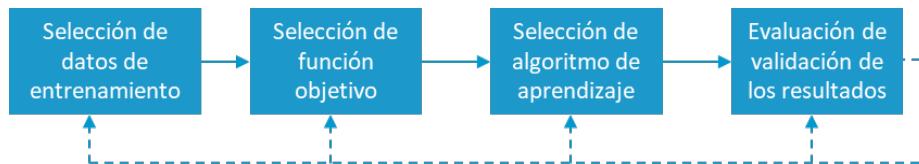


Figura 7. Etapas del aprendizaje de un concepto

Selección del conjunto de datos de entrenamiento

Como previamente se ha descrito, la computadora aprende de una experiencia, reflejada en un conjunto de instancias. La primera etapa para diseñar un sistema de aprendizaje consiste en analizar los datos existentes y seleccionar los datos de entrenamiento. Estos datos de entrenamiento pueden determinar crucialmente el éxito de la tarea de aprendizaje.

La cantidad de instancias disponibles o su relación con el concepto puede ser determinante. Evidentemente, cuanto mayor sea el número de instancias más fácil será la tarea de aprendizaje. Por otra parte, existen datos de entrenamiento que no proveen de una retroalimentación directa al sistema sobre la clase a aprender. Por ejemplo, si una máquina trata de aprender a jugar al ajedrez y cada movimiento realizado es una instancia, se tiene una **retroalimentación directa** si la clase indica si el movimiento es correcto para cada instancia. Sin embargo, si para un movimiento determinado (instancia), solo se tiene información de que ese movimiento corresponde a una serie de movimientos que dieron lugar a una partida que se ganó, la retroalimentación es indirecta.

En este caso la corrección del movimiento se infiere del hecho de que la partida se ganó, pero en realidad todos los movimientos realizados al ganar una partida no tienen por qué ser los correctos u óptimos en el momento en que fueron realizados. En este último caso sería conveniente asignar un factor de confianza que especifique cuánto puede haber influido un movimiento al éxito final de la partida, factor bastante difícil de determinar. Por tanto, el aprendizaje a partir de instancias que proveen

retroalimentación indirecta sobre la clase a aprender es más difícil.

Selección de una función objetivo y su representación

En la siguiente etapa se ha de determinar el tipo de conocimiento que se va a aprender, se trata de determinar una función objetivo y representarla. Una vez representado el problema en una función objetivo, la mejora del rendimiento P en una tarea T se reduce al aprendizaje de una determinada función objetivo. Para el aprendizaje del ajedrez, esta función podría consistir en una función que asigna un determinado número a cada movimiento de los movimientos legales posibles, asignando un número mayor a aquellos movimientos que producen mejores posiciones de tablero.

En el ejemplo mostrado en la explicación previa de aprendizaje supervisado se busca aprender una función que, tomando como entrada valores discretos, produce una salida igualmente discreta con tres valores correspondientes a los tres posibles diagnósticos. La función que aprender es una disyunción de conjunciones lógicas de condiciones de los valores de entrada y puede ser representada mediante un árbol de decisión o un conjunto de reglas del tipo SI-ENTONCES.

En el árbol inducido cada camino desde la raíz a las hojas corresponde a una conjunción de condiciones y el propio árbol engloba la disyunción de las conjunciones. La función expresada como árbol está en la Figura 5 mientras que, en forma de reglas, empleando las conjunciones lógicas AND y OR quedaría de la siguiente manera:

SI “Dolor de garganta”=Sí

ENTONCES Diagnóstico=“Infección de garganta”

SI (“Dolor de garganta”=No AND “Fiebre”=Sí) OR (“Dolor de garganta”=No AND “Fiebre”=No AND “Dolor de cabeza”=Sí)

ENTONCES Diagnóstico=“Resfriado”

SI “Dolor de garganta”=No AND “Fiebre”=No AND “Dolor de cabeza”=No

ENTONCES Diagnóstico=“Alergia”

Selección del algoritmo de aprendizaje para aproximación a la función objetivo

Una vez se conoce el conjunto de entrenamiento y la función objetivo, se debe escoger un algoritmo de aproximación a la función objetivo. En el ejemplo de aprendizaje supervisado, dado que la función se va a representar como un árbol, es lógico escoger un método de construcción de árboles de decisión. Existen diversos métodos para el aprendizaje de árboles y la selección de uno u otro dependerá de diversos factores como si, por ejemplo, las entradas toman valores discretos o continuos, si existen datos ruidosos, etc.

Evaluación y validación de los resultados

Escoger una buena función objetivo o un algoritmo adecuado para una tarea de aprendizaje concreta puede resultar en una tarea complicada. Por tanto, en la etapa de evaluación los resultados podrían no ser los esperados. Por tanto, en la etapa de evaluación hay que considerar si es necesario regresar a un paso previo o validar el aprendizaje realizado. Así el problema de aprendizaje de un concepto puede resultar en un proceso cíclico hasta obtener los resultados deseados.

Ideas clave

1.4. Etapas en el descubrimiento de conocimiento

El aprendizaje implica extraer y ampliar el conocimiento y, desde este punto de vista, se puede ver el objetivo de la minería de datos como un intento por modelar el conocimiento, convirtiendo información no estructurada en conocimiento. De aquí surge el término **Ingeniería de Conocimiento** que versa sobre el proceso de construir un sistema inteligente para obtener conocimiento.

En el marco del máster actual ser capaz de aplicar técnicas de IA para descubrir conocimiento en bases de datos es de gran interés, es lo que se ha definido anteriormente como minería de datos. El término **Descubrimiento de Conocimiento en Bases de Datos**, en inglés ***Knowledge Discovery in Databases (KDD)***, es frecuentemente utilizado como sinónimo de la minería de datos (Fayyad, 2001; Fayyad *et al.*, 1996).

Sin embargo, este término se refiere al **procedimiento completo** necesario para extraer conocimiento potencialmente útil y previamente desconocido a partir de los datos en una base de datos (Roiger, 2003). KDD es un proceso iterativo que incluye etapas previas a la fase de minería de datos propiamente dicha, para la extracción y preparación de los datos, así como etapas posteriores para el análisis de los resultados y toma de decisiones.

Hay diferentes modelos KDD definidos, pero sea cual sea el modelo KDD utilizado, el **primer paso** siempre consistirá en **identificar el objetivo de la aplicación** del procedimiento, esto es, definir el problema identificando la tarea que se ha de realizar, estableciendo hipótesis sobre los resultados deseados y posibles.

El resto de las etapas variará de unos modelos a otros, aunque típicamente se establecerán las siguientes etapas representadas en la Figura 8 (Han & Kamber, 2012; Roiger, 2003):

- ▶ **Integración de los datos:** en la Figura 8 se representa un caso en el que hay múltiples fuentes de datos que pueden ser combinadas y, por tanto, es necesario un primer paso de integración de los datos.
- ▶ **Selección de los datos:** se debe escoger un conjunto de datos considerados relevantes que serán analizados. Esta selección debe ser realizada por personas expertas en el dominio en cuestión o mediante la ayuda de herramientas de descubrimiento de conocimiento. En el ejemplo previo de base de datos sobre síntomas y diagnósticos de enfermedades (Tabla 1) es evidente que el atributo «Número de paciente» no es relevante en el análisis. Sin embargo, la eliminación de atributos de entrada no es siempre tan evidente y se han de utilizar técnicas específicas para ello.
- ▶ **Pre-procesamiento de los datos:** los datos ruidosos y datos inconsistentes son tratados. Por ejemplo, una base de datos puede contener datos duplicados, valores incorrectos o desconocidos. Se pueden utilizar métodos automáticos estadísticos como la aplicación de *outliers* (por ejemplo, eliminar el 5 % de los valores más lejanos a la predicción). Este paso es denominado limpieza de datos por algunos autores y requiere conocer muy bien los datos, habiendo un riesgo de descarte de ejemplos de clases poco frecuentes.
- ▶ **Transformación de los datos:** los datos son normalizados y convertidos si es necesario. Por ejemplo, algunos métodos de minería de datos no pueden procesar datos nominales por lo que se han de convertir los datos nominales a numéricos. Igualmente hay métodos que son mucho más lentos con atributos numéricos que con nominales por lo que se han de discretizar los valores de los atributos de entrada.
- ▶ **Minería de datos.** Se ejecutan técnicas de inteligencia artificial para extraer

patrones en los datos.

- ▶ **Interpretación y evaluación de los resultados.** Se examina la salida del paso anterior para determinar si los datos descubiertos son útiles e interesantes. A partir de aquí hay que tomar la decisión de si repetir los pasos previos usando nuevos atributos o instancias. Se pueden utilizar técnicas de visualización y representación de conocimiento en esta etapa para facilitar la tarea.

Una vez se concluye que el conocimiento descubierto es útil, se aplica ese conocimiento al problema apropiado. Esta etapa puede servir para verificar las conclusiones extraídas en la etapa de interpretación de resultados.

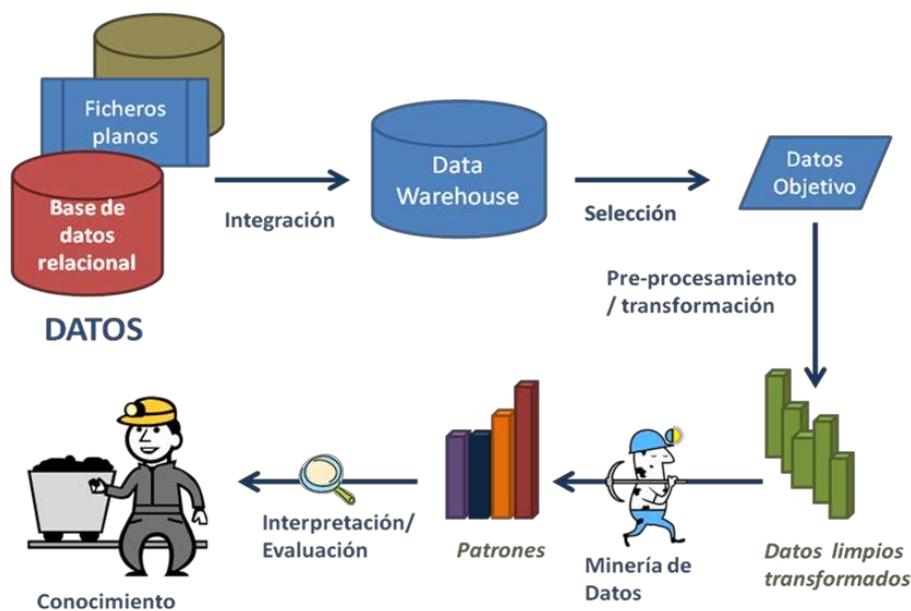


Figura 8. Etapas de un procedimiento típico KDD.

1.5. Referencias bibliográficas

Abduljabbar, R., Dia, H., Liyanage, S. & Bagloee, S. A. (2019). Applications of Artificial Intelligence in Transport: An Overview. *Sustainability*, 11(1), 189. Disponible en

<https://doi.org/10.3390/su11010189>

Aguado, F., Cabalar, P., Fandinno, J., Muñiz, B., Pérez, G. & Suárez, F. (2019). A Rule-Based System for Explainable Donor-Patient Matching in Liver Transplantation. *Electronic Proceedings in Theoretical Computer Science*, 306, 266-272. Disponible en

<https://doi.org/10.4204/EPTCS.306.31>

Akinnola, B. (2012). *Computer Troubleshooting, Using an Expert System: A Research Work*. LAP Lambert Academic Publishing.

Alonso, R. S., Prieto, J., García, Ó. & Corchado, J. M. (2019). Collaborative learning via social computing. *Frontiers of Information Technology & Electronic Engineering*, 20(2), 265-282. Disponible en <https://doi.org/10.1631/FITEE.1700840>

Alonso, R. S., Sittón-Candanedo, I., García, Ó., Prieto, J. & Rodríguez-González, S. (2020). An intelligent Edge-IoT platform for monitoring livestock and crops in a dairy farming scenario. *Ad Hoc Networks*, 98, 102047. Disponible en

<https://doi.org/10.1016/j.adhoc.2019.102047>

ASALE, R. & RAE. (s. f.). Inteligencia | Diccionario de la lengua española. «Diccionario de la lengua española». Edición del Tricentenario. Recuperado de

<https://dle.rae.es/inteligencia>

Bayram, B. & İnce, G. (2018). Advances in Robotics in the Era of Industry 4.0. En A. Ustundag & E. Cevikcan (Eds.), *Industry 4.0: Managing The Digital Transformation* (pp. 187-200). Springer International Publishing. Disponible en

https://doi.org/10.1007/978-3-319-57870-5_11

Boru, A., Dosdoğru, A. T., Göçken, M. & Erol, R. (2019). A Novel Hybrid Artificial Intelligence Based Methodology for the Inventory Routing Problem. *Symmetry*, 11(5), 717. Disponible en <https://doi.org/10.3390/sym11050717>

Cevik Onar, S. & Ustundag, A. (2018). Smart and Connected Product Business Models. En A. Ustundag & E. Cevikcan (Eds.), *Industry 4.0: Managing The Digital Transformation* (pp. 25-41). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-319-57870-5_2

Davis, D. S. (2020). Geographic Disparity in Machine Intelligence Approaches for Archaeological Remote Sensing Research. *Remote Sensing*, 12(6), 921. Disponible en <https://doi.org/10.3390/rs12060921>

Elmishali, A., Stern, R. & Kalech, M. (2018). An Artificial Intelligence paradigm for troubleshooting software bugs. *Engineering Applications of Artificial Intelligence*, 69, 147-156. Disponible en <https://doi.org/10.1016/j.engappai.2017.12.011>

Farooq, A., Anwar, S., Awais, M. & Alnowami, M. (2017). Artificial intelligence based smart diagnosis of alzheimer's disease and mild cognitive impairment. *2017 International Smart Cities Conference (ISC2)*, 1-4. Disponible en

<https://doi.org/10.1109/ISC2.2017.8090871>

Fayyad, U. (2001). Knowledge Discovery in Databases: An Overview. En S. Džeroski & N. Lavrač (Eds.), *Relational Data Mining* (pp. 28-47). Springer. Disponible en https://doi.org/10.1007/978-3-662-04599-2_2

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17(3), 37-37. Disponible en

<https://doi.org/10.1609/aimag.v17i3.1230>

García, Ó., Alonso, R., Prieto, J., & Corchado, J. (2017). Energy Efficiency in Public Buildings through Context-Aware Social Computing. *Sensors*, 17(4), 826. Disponible en <https://doi.org/10.3390/s17040826>

García, Ó., Alonso, R. S., Tapia, D. I. & Corchado, J. M. (2015). CAFCLA: A framework to design, develop, and deploy AMI-based collaborative learning applications. En *Recent advances in ambient intelligence and context-aware computing* (pp. 187–209). IGI Global.

Han, J. & Kamber, M. (2012). *Data mining: Concepts and techniques* (3rd ed). Elsevier.

Jain, A., Shah, D. & Churi, P. (2020). A Review on Business Intelligence Systems Using Artificial Intelligence. En S. Smys, J. M. R. S. Tavares, V. E. Balas, & A. M. Iliyasu (Eds.), *Computational Vision and Bio-Inspired Computing* (pp. 1023-1030). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-030-37218-7_107

Joshi, A. V. (2020). Recommendations Systems. En A. V. Joshi (Ed.), *Machine Learning and Artificial Intelligence* (pp. 199-204). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-030-26622-6_21

Jozi, A., Pinto, T., Marreiros, G., & Vale, Z. (2019). *Electricity consumption forecasting in office buildings: An artificial intelligence approach*. 2019 IEEE Milan PowerTech, 1-6. Disponible en <https://doi.org/10.1109/PTC.2019.8810503>

Jozi, A., Pinto, T., Praça, I., Silva, F., Teixeira, B. & Vale, Z. (2017). Energy consumption forecasting using genetic fuzzy rule-based systems based on MOGUL

learning methodology. *2017 IEEE Manchester PowerTech*, 1-5. Disponible en

<https://doi.org/10.1109/PTC.2017.7981219>

Khine, P. P. & Wang, Z. S. (2018). Data lake: A new ideology in big data era. *ITM Web of Conferences*, 17, 03025. Disponible en

<https://doi.org/10.1051/itmconf/20181703025>

Liakos, K. G., Busato, P., Moshou, D., Pearson, S. & Bochtis, D. (2018). Machine Learning in Agriculture: A Review. *Sensors*, 18(8), 2674. Disponible en <https://doi.org/10.3390/s18082674>

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 76-80. Disponible en <https://doi.org/10.1109/MIC.2003.1167344>

Malhotra, D. k., Nydick, R. L. & Malhotra, K. (2017). Evaluating bank solvency with support vector machines. *International Journal of Business Intelligence and Systems Engineering*, 1(2), 179-195. Disponible en

<https://doi.org/10.1504/IJBISE.2017.088698>

Mali, G. U. & Gautam, D. K. (2018). Shortest Path Evaluation in Wireless Network Using Fuzzy Logic. *Wireless Personal Communications*, 100(4), 1393-1404. Disponible en <https://doi.org/10.1007/s11277-018-5645-1>

Matlou, O. G. & Abu-Mahfouz, A. M. (2017). Utilising artificial intelligence in software defined wireless sensor network. *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 6131-6136. Disponible en

<https://doi.org/10.1109/IECON.2017.8217065>

McGovern, A., Elmore, K. L., Gagne, D. J., Haupt, S. E., Karstens, C. D., Lagerquist, R., Smith, T. & Williams, J. K. (2017). Using Artificial Intelligence to Improve Real-

Time Decision-Making for High-Impact Weather. *Bulletin of the American Meteorological Society*, 98(10), 2073-2090. Disponible en <https://doi.org/10.1175/BAMS-D-16-0123.1>

Merrill, M. D., Tennyson, R. D. & Posey, L. O. (1992). *Teaching concepts: An instructional design guide* (2nd ed). Educational Technology Publications.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mohammed, M. A., Abd Ghani, M. K., Hamed, R. I., Mostafa, S. A., Ahmad, M. S. & Ibrahim, D. A. (2017). Solving vehicle routing problem by using improved genetic algorithm for optimal solution. *Journal of Computational Science*, 21, 255-262. Disponible en <https://doi.org/10.1016/j.jocs.2017.04.003>

Namatevs, I. & Aleksejeva, L. (2017). Decision Algorithm for Heuristic Donor-Recipient Matching. *MENDEL*, 23(1), 33-40. Disponible en

<https://doi.org/10.13164/mendel.2017.1.033>

Narasima, D. (2018). Industry 4.0: Reimagining the Future of Workplace (Five Business Case Applications of Artificial Intelligence, Machine Learning, Robots, Virtual Reality in Five Different Industries). *International Journal of Engineering, Business and Enterprise Applications (IJEBEA)*, 26(1), 5-8. Disponible en

<https://papers.ssrn.com/abstract=3303732>

Nagnevitsky, M. (2005). *Artificial intelligence: A guide to intelligent systems* (2nd ed). Addison-Wesley.

Nushi, B., Kamar, E., Horvitz, E. & Kossmann, D. (2017, febrero 12). *On Human Intellect and Machine Failures: Troubleshooting Integrative Machine Learning Systems*. Thirty-First AAAI Conference on Artificial Intelligence. Thirty-First AAAI Conference on Artificial Intelligence. Disponible en

<https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/15032>

Ohlsson, S., Sloan, R. H., Turán, G. & Urasky, A. (2017). Measuring an artificial intelligence system's performance on a Verbal IQ test for young children. *Journal of Experimental & Theoretical Artificial Intelligence*, 29(4), 679-693. Disponible en <https://doi.org/10.1080/0952813X.2016.1213060>

García, O, Prieto, J., Alonso, R. y Corchado, J. (2017). A Framework to Improve Energy Efficient Behaviour at Home through Activity and Context Monitoring. *Sensors*, 17(8), 1749. Disponible en <https://doi.org/10.3390/s17081749>

Pannu, A. (2015). Artificial Intelligence and its Application in Different Areas. *International Journal of Engineering and Innovative Technology*, 4(10), 79-84. Disponible en

http://www.ijbeit.com/Vol%204/Issue%2010/IJEIT1412201504_15.pdf

Peng, M., Yang, J., Shi, Q., Ying, L., Zhu, H., Zhu, G., Ding, X., He, Z., Qin, J., Wang, J., Yan, H., Bi, X., Shen, B., Wang, D., Luo, L., Zhao, H., Zhang, C., Lin, Z., Hong, L. & Li, J. (2020). Artificial Intelligence Application in COVID-19 Diagnosis and Prediction. *Social Science Research Network*. Disponible en

<https://papers.ssrn.com/abstract=3541119>

Panesar, A. (2019). *What Is Machine Learning? In Machine Learning and AI for Healthcare* (pp. 75-118). Apress, Berkeley, CA.

Poole, D. L. & Mackworth, A. K. (2010). *Artificial intelligence: Foundations of computational agents*. Cambridge University Press.

Rogers, S. & Girolami, M. (2017). *A first course in machine learning* (Second Edition). CRC Press, Taylor & Francis Group, a Chapman & Hall book.

Roiger, R. J. (2003). *Data Mining: A Tutorial-Based Primer*. Addison Wesley.

Disponible en <https://www.crcpress.com/Data-Mining-A-Tutorial-Based-Primer-Second-Edition/Roiger/p/book/9781498763974>

Sami Sivri, M. & Oztaysi, B. (2018). Data Analytics in Manufacturing. En A. Ustundag & E. Cevikcan (Eds.), *Industry 4.0: Managing The Digital Transformation* (pp. 155-172). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-319-57870-5_9

Silva, J., Praça, I., Pinto, T., & Vale, Z. (2020). Energy Consumption Forecasting Using Ensemble Learning Algorithms. En E. Herrera-Viedma, Z. Vale, P. Nielsen, A. Martin Del Rey, & R. Casado Vara (Eds.), *Distributed Computing and Artificial Intelligence*, 16th International Conference, Special Sessions (pp. 5-13). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-030-23946-6_1

Sterne, J. (2017). *Artificial intelligence for marketing: Practical applications*. Wiley.

Traviglia, A. & Torsello, A. (2017). Landscape Pattern Detection in Archaeological Remote Sensing. *Geosciences*, 7(4), 128. Disponible en

<https://doi.org/10.3390/geosciences7040128>

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59(236), 433.

Vinagre, E., De Paz, J. F., Pinto, T., Vale, Z., Corchado, J. M. & Garcia, O. (2016). *Intelligent energy forecasting based on the correlation between solar radiation and consumption patterns*. 2016 IEEE Symposium Series on Computational Intelligence (SSCI), 1-7. Disponible en <https://doi.org/10.1109/SSCI.2016.7849853>

Witten, I. H. & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed). Morgan Kaufman.

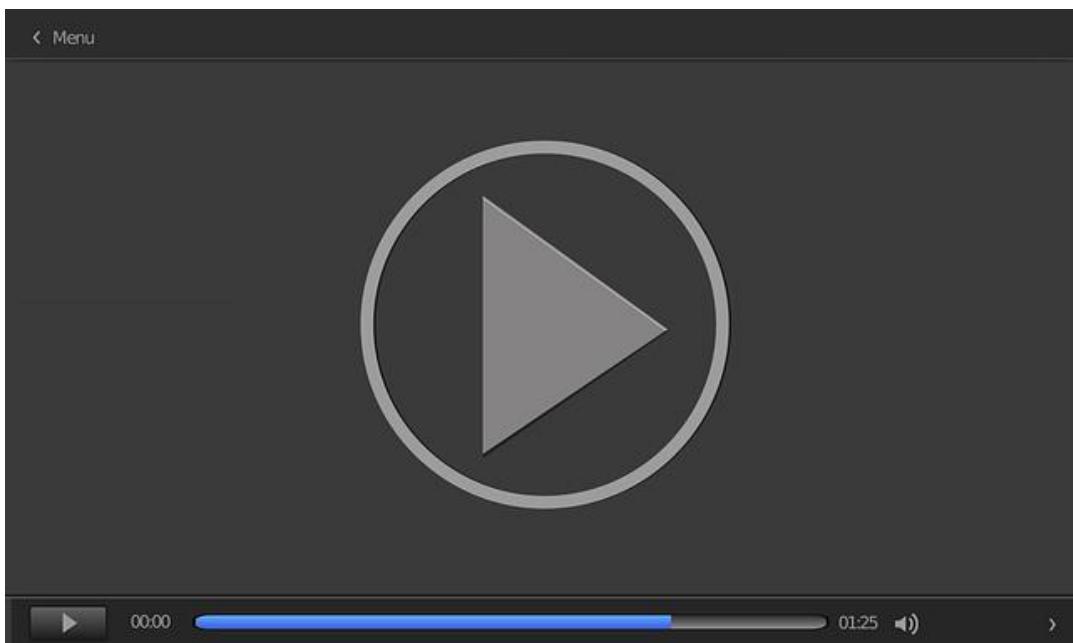
Zeeshan, M. & Saxena, K. (2020). Explorative Study of Artificial Intelligence in Digital Marketing. En A. P. Pandian, R. Palanisamy, & K. Ntalianis (Eds.), *Proceeding of the International Conference on Computer Networks, Big Data and IoT* (ICCBI - 2019)

(pp. 968-978). Springer International Publishing. Disponible en

https://doi.org/10.1007/978-3-030-43192-1_107

Introducción e historia de la inteligencia artificial

En esta lección magistral se relata la historia de la Inteligencia Artificial (IA), desde sus inicios en la ‘era oscura’, hasta los actuales sistemas basados en conocimiento. Se describen de forma resumida algunos de los hechos y personajes más relevantes de la IA, así como la motivación que lleva al surgimiento de alguna de las técnicas que serán estudiadas en posteriores temas de esta asignatura.



01. Introducción e historia de la inteligencia artificial

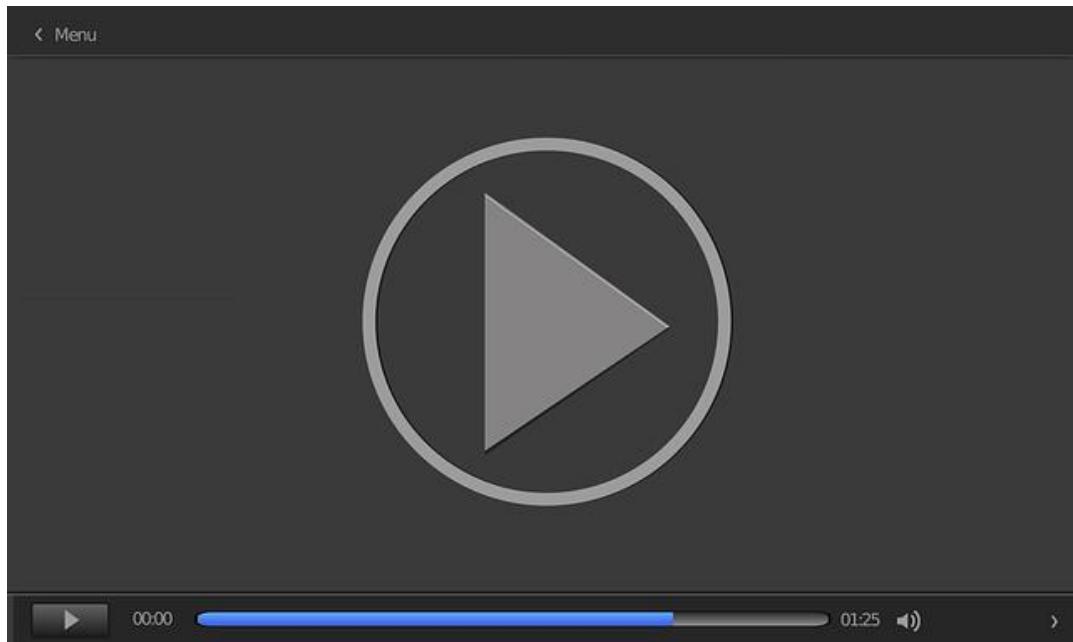
Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=d6992430-08fe-4dfd-bbb4-aff800f9db84>

Entrevista a Matilde Santos Peñas sobre la inteligencia artificial

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
https://www.youtube.com/watch?v=v_1aR6hha2k.

Matilde Santos Peñas, Catedrática de Universidad del área de Ingeniería de Sistemas y Automática en la Universidad Complutense de Madrid y coautora del libro «Inteligencia Artificial e Ingeniería del Conocimiento» de la editorial RA-MA, da una visión general muy didáctica e interesante de diversos aspectos de la inteligencia artificial, mencionando inquietudes personales, cuestiones y técnicas.



Accede al vídeo:

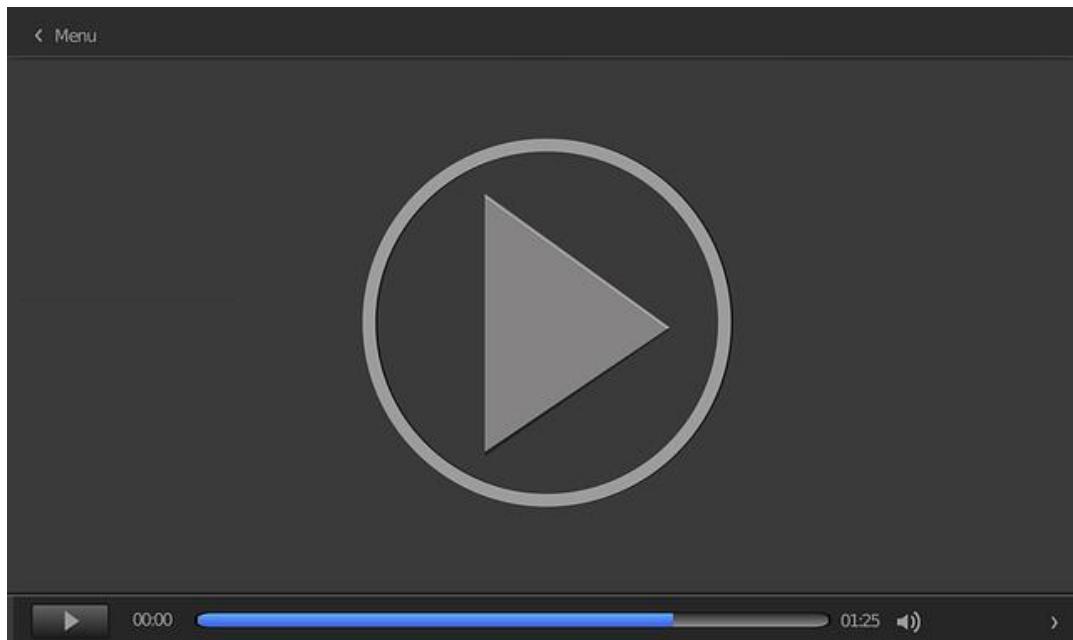
https://www.youtube.com/embed/v_1aR6hha2k

Conciencia artificial y test de Turing

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=Y90a6Xwwh3w>

Extracto del programa Tres14 en el que Raúl Arrabales, ingeniero informático de la Universidad Carlos III de Madrid, habla sobre conciencia artificial y el test de Turing.



Accede al vídeo:

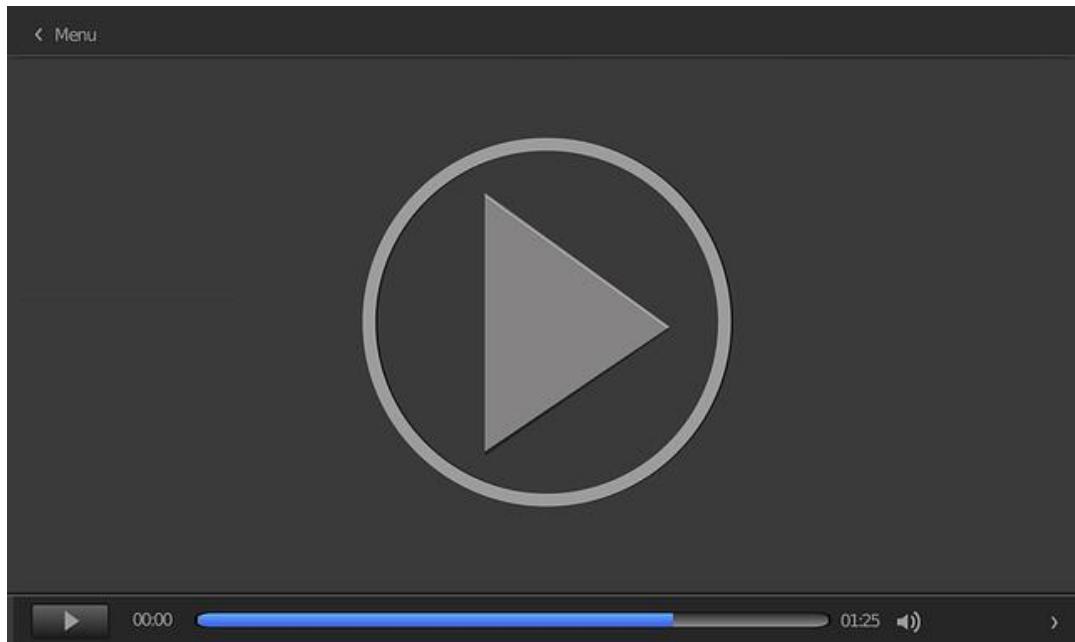
<https://www.youtube.com/embed/Y90a6Xwwh3w>

Inteligencia artificial

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

https://www.youtube.com/watch?v=Ut6gDw_Onwk

Programa muy interesante, de casi media hora de duración, que describe diversas cuestiones y problemas de la inteligencia artificial, centrándose principalmente en la robótica.



Accede al vídeo:

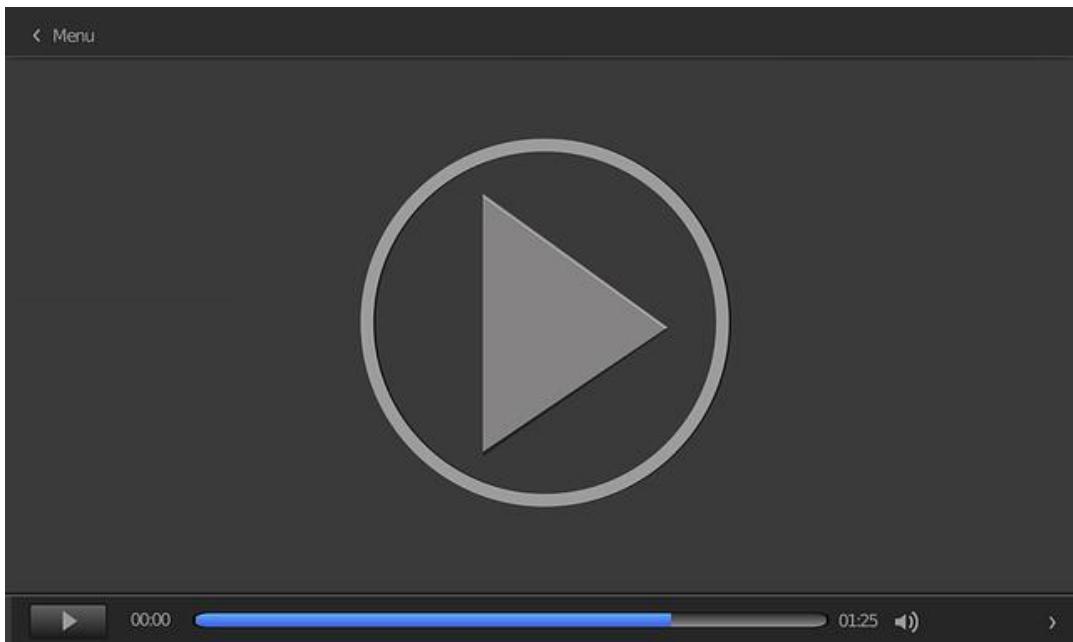
https://www.youtube.com/embed/Ut6gDw_Onwk

Presentación introductoria de la minería de datos

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://youtu.be/QY09nSg-KBk>

Presentación sobre la minería de datos que resume conceptos básicos pero relevantes en esta área de manera clara y sencilla.



Accede al vídeo:

<https://www.youtube.com/embed/QY09nSg-KBk>

AI Topics

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
<http://aitopics.org/>.

AITopics.org es un sitio web que recopila información sobre la investigación y las aplicaciones de la inteligencia artificial. Contiene enlaces muy interesantes a vídeos, materiales para cursos, etc.

- 1.** Indica cuáles de las siguientes afirmaciones son correctas:
 - A. La escuela de la Inteligencia Artificial Fuerte defiende que las máquinas pueden llegar a tener conciencia.
 - B. La escuela de la Inteligencia Artificial Débil no defiende que los procesos cerebrales puedan ser simulados en un computador.
 - C. El juego de imitación de Turing consiste en que una máquina consiga mantener una conversación tal y como lo haría un humano.
 - D. Las técnicas de aprendizaje automático no forman parte del campo de la inteligencia artificial.

- 2.** Indica cuál de las siguientes afirmaciones no es correcta:
 - A. La minería de datos utiliza técnicas de aprendizaje automático para descubrir patrones en grandes cantidades de datos.
 - B. La minería de datos tiene un objetivo fundamentalmente teórico.
 - C. Existe una fase en los procedimientos KDD que consiste en ejecutar técnicas de inteligencia artificial.
 - D. En los procedimientos KDD, previo a la fase de minería de datos, se dan otras fases de selección y transformación de los datos.

- 3.** Indica cuál de las siguientes afirmaciones no es correcta respecto a la experiencia en el aprendizaje de conceptos. La experiencia:
 - A. Consiste en un conjunto de objetos específicos denominados instancias.
 - B. Consiste en una serie de ejemplos y no-ejemplos.
 - C. Consiste en una serie de instancias con atributos de entrada y de salida.
 - D. Consiste en un conjunto de datos denominado datos de prueba.

- 4.** Identificar la clase de una instancia desconocida en base a sus atributos, que se presentan comunes a ejemplos previos encontrados de esa clase es una tarea de:
- A. Discriminación.
 - B. Generalización.
 - C. Clasificación.
 - D. Descripción.
- 5.** Una tarea de aprendizaje consiste en descubrir los síntomas comunes de un grupo de pacientes que presentan una enfermedad bien conocida. Se trata de un problema de aprendizaje de tipo:
- A. Supervisado.
 - B. No-supervisado.
- 6.** Una tarea de aprendizaje consiste en descubrir los síntomas comunes de un grupo de pacientes con diagnóstico desconocido. Se trata de un problema de aprendizaje de tipo:
- A. Supervisado.
 - B. No-supervisado.
- 7.** Indica cuál de las siguientes afirmaciones no es cierta, respecto al descubrimiento de conocimiento en bases de datos:
- A. Es un procedimiento completo necesario para extraer conocimiento a partir de los datos de una base de datos.
 - B. Sea cual sea el procedimiento KDD las fases siempre son las mismas.
 - C. La interpretación de los resultados forma parte del procedimiento KDD.
 - D. Es un proceso que puede ser iterativo.

8. Indica cuál de las siguientes afirmaciones no es correcta:

- A. Los datos de entrenamiento son un conjunto de instancias.
- B. Los datos de entrenamiento pueden contener no-ejemplos.
- C. Los atributos son denominados también ejemplos.
- D. Al concepto que se aprende se le llama también clase.
- E. El concepto es un conjunto de instancias.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto al aprendizaje de conceptos:

- A. Los problemas de aprendizaje se resuelven a veces como una búsqueda en un espacio de hipótesis.
- B. Siempre se aplican las técnicas de búsqueda de la mejor hipótesis sobre el espacio completo de posibles hipótesis.
- C. El tamaño de los datos de entrenamiento no influye en el resultado del aprendizaje.
- D. Se pueden encontrar en la práctica distintas descripciones de un concepto.
- E. El sobreajuste consiste en generalizar demasiado.

10. Indica cuál de las siguientes frases no es correcta respecto a un sistema experto:

- A. Se incorpora en el sistema el conocimiento de un experto humano.
- B. Raramente se ha aplicado un sistema experto a un problema real con éxito.
- C. Suele estar limitado a un dominio de conocimiento.
- D. Puede ser complementado con técnicas de aprendizaje automático como las redes neuronales para obtener reglas a partir de grandes cantidades de datos que el experto humano es incapaz de obtener.

Técnicas de Inteligencia Artificial

Tema 2. Python para la implementación de técnicas de inteligencia artificial

Índice

Esquema

Ideas clave

- 2.1. ¿Cómo estudiar este tema?
- 2.2. Introducción
- 2.3. El lenguaje Python: conceptos básicos e instalación
- 2.4. La sintaxis de Python
- 2.5. Listas, tuplas, conjuntos y diccionarios
- 2.6. Librerías útiles para el análisis de datos
- 2.7. La librería NumPy para el manejo de datos
- 2.8. Importación de datos
- 2.9. Introducción a Machine Learning con librerías en Python
- 2.10. Referencias bibliográficas

A fondo

Gráficos en Python con Matplotlib, Seaborn y Plotly

TensorFlow 2.0 Crash Course

Tutorial de programación en Python

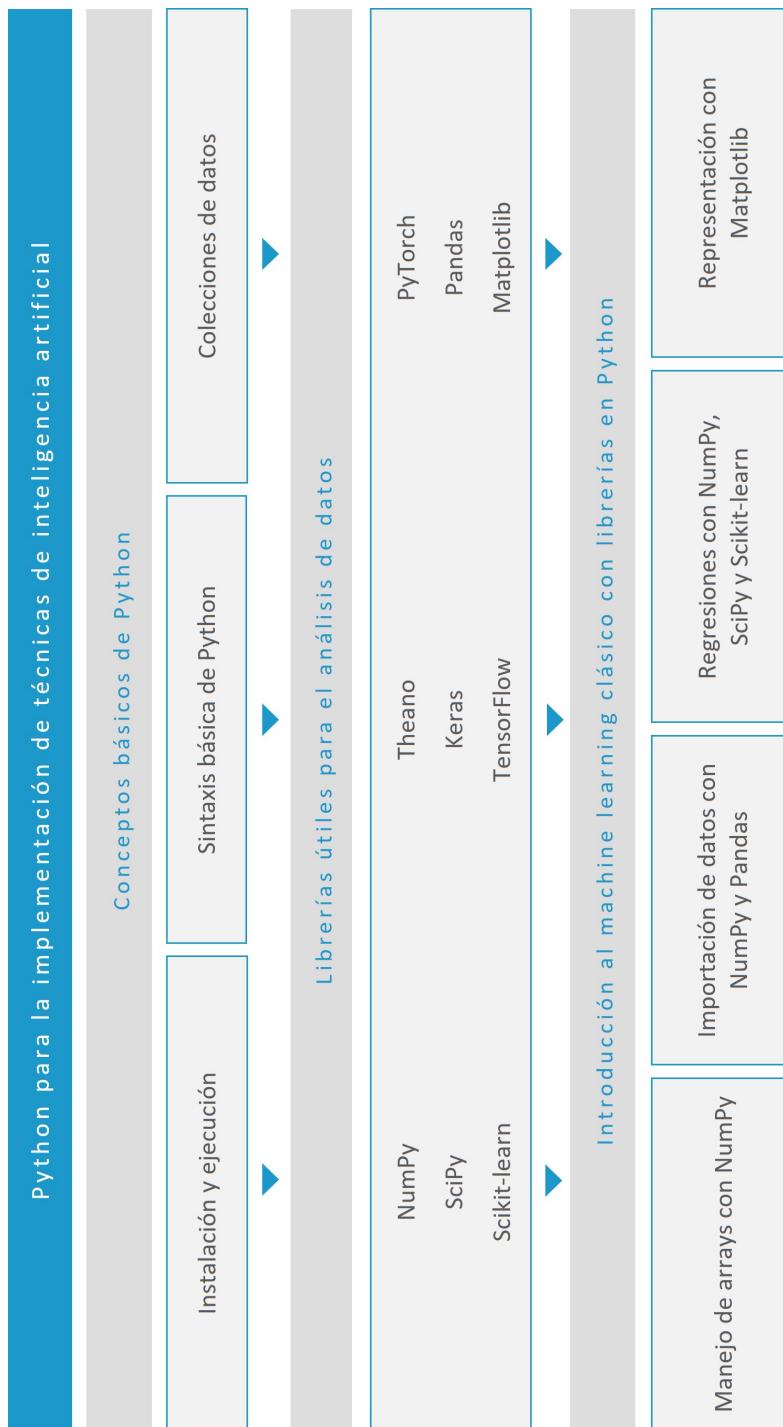
Udacity

Coursera

Unipython

Test

Esquema



2.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral, revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan.

Al inicio de este tema se hace una pequeña introducción en relación con los diferentes **lenguajes empleados en el ámbito de la inteligencia artificial y el aprendizaje automatizado** (*machine learning*) y el aprendizaje profundo (*deep learning*) y por qué se elige Python como lenguaje de trabajo en esta asignatura.

A continuación, se describen los **conceptos básicos** del lenguaje Python, cómo se clasifica como lenguaje de programación, **cómo instalar el intérprete y el gestor de paquetes en una computadora** y cómo comenzar a trabajar en código en Python.

Tras ello, se describe la **sintaxis del lenguaje Python** en cuanto a los componentes del lenguaje imprescindibles para trabajar en aplicaciones de inteligencia artificial y *machine learning*. Esto incluye una descripción de su sintaxis, variables, tipos de datos, números, cadenas y operadores.

A continuación, se describe con mayor detalle las **colecciones de datos existentes en Python** de forma incorporada y que nos permiten trabajar con conjuntos de datos (*datasets*) a la hora de aplicar métodos inteligentes. Esta sección incluye, por tanto, una descripción de las listas, las tuplas, los conjuntos y los diccionarios.

En este sentido, quedan fuera por motivos de planificación del estudio de este tema conceptos intermedios y avanzados del lenguaje Python como el control de flujo, las funciones, los objetos y las excepciones, entre otras cuestiones, y que pueden ser consultados por el alumno recurriendo a los materiales adicionales referidos al final

del tema. Si bien su conocimiento es recomendado, no es imprescindible a la hora de aplicar técnicas de *machine learning*, cuya programación sigue un proceso lineal.

En este tema se tratan las **principales librerías para el tratamiento de datos en Python**, bien sea para el tratamiento básico de los datos (NumPy, Pandas, Theano), bien sea para aplicar técnicas de aprendizaje automatizado clásico (SciPy, Scikit-learn), bien para aplicar redes neuronales y técnicas *deep learning* (Keras, TensorFlow, Pytorch) o bien para la representación de los datos (Matplotlib).

Tras dicha introducción, se describen las **funcionalidades básicas de la librería NumPy**, como librería auxiliar para trabajar con arrays. Además, se describen las distintas formas de **importar datasets empleando Python** y sus librerías (NumPy, Pandas) y se finalizan con unos **conceptos básicos de machine learning clásico** (regresión) empleando NumPy, SciPy, Scikit-learn y Matplotlib.

Al finalizar el estudio de este tema serás capaz de:

- ▶ Manejar la sintaxis básica de Python para emplearlo como lenguaje de análisis de datos y como base para aplicar y probar técnicas de *machine learning* en el resto de la asignatura.
- ▶ Conocer y manejar las colecciones de datos incorporadas en Python para su uso en la aplicación de técnicas de inteligencia artificial.
- ▶ Conocer y recurrir a las librerías más importantes sobre Python para el manejo de datos, el *machine learning* clásico, las redes neuronales y el *deep learning*, así como la representación de datos.
- ▶ Importar *datasets* desde fuentes locales, o fuentes externas *open data*, empleando librerías sobre Python apropiadas para ello.
- ▶ Aplicar algoritmos básicos de regresión empleando Python y las librerías apropiadas.

Ideas clave

Asimismo, a lo largo de todos y cada uno de los siguientes temas, se hará uso de las librerías descritas para aplicar y probar los diferentes métodos de inteligencia artificial, aprendidos en cada uno de dichos temas. Para ello, se introducirá en cada caso al alumno a los conceptos básicos de cada librería según se vaya requiriendo.

2.2. Introducción

Python es un lenguaje de programación creado por Guido van Rossum y lanzado en 1991 (y sí, el nombre viene del célebre grupo humorístico británico Monty Python). Es el lenguaje más empleado en el ámbito de la inteligencia artificial y el *machine learning* (Bansal, 2019), además de ser utilizado también para el desarrollo de aplicaciones web del lado del servidor, el desarrollo de *software* en general, aplicaciones matemáticas o *scripting* en la gestión de sistemas operativos.



Figura 1. Logo de Python.

Tras Python, los lenguajes más empleados en el ámbito de la inteligencia artificial incluyen Java, que ofrece ciertas ventajas como su seguridad, debido a su uso de *bytecode* y *sandboxes*; R, lenguaje basado en gráficos ampliamente empleado en aplicaciones estadísticas, análisis y visualización en *machine learning*; JavaScript (formalmente conocido como ECMAScript) especialmente desde la implementación de Node.js (el equivalente de JavaScript en el lado servidor, a pesar de que JavaScript fue concebido inicialmente para aportar comportamiento dinámico a las aplicaciones web del lado del cliente) y, por supuesto, impulsado por el lanzamiento de TensorFlow.js en 2018; y Scala, uno de los lenguajes que forman parte del *core* de Apache Spark. La muestra las tendencias en las ofertas de trabajo en el portal Indeed en las que se buscan ingenieros en *machine learning* y científicos de datos en función del lenguaje de programación, elaborada por J.F. Puget, ingeniero distinguido en IBM en ese momento (desde 2020 en NVIDIA). C/C++ es también un lenguaje a

tener en cuenta como lenguaje soportado por diferentes API, dada su popularidad como lenguaje de propósito general durante muchos años; así como Matlab, por su implantación como herramienta y lenguaje de tratamiento matemático de datos y su versatilidad manejando arrays.

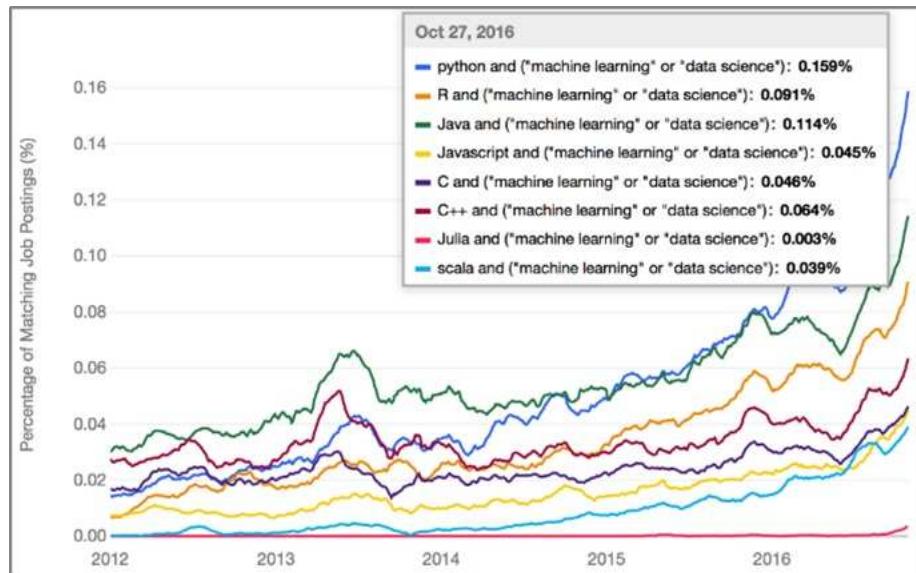


Figura 2. Ofertas de empleo en machine learning y ciencia de datos en el portal Indeed según lenguaje de programación. Fuente: Puget, J.F. (2016).

Según otras fuentes (Bansal, 2019), Python es el líder, con el 57 % de los científicos de datos y desarrolladores de aprendizaje automático que lo usan y el 33 % lo prefiere sobre otros lenguajes para los desarrollos. No solo Python es un lenguaje ampliamente utilizado, sino que es la principal elección para la mayoría de sus usuarios, debido al lanzamiento de TensorFlow al público general por Google en 2015 (si bien posteriormente TensorFlow fue lanzado para otros lenguajes, como veremos más adelante) y una amplia selección de otras librerías.

Python es una opción idónea para los principiantes en este campo. Hay muchas bibliotecas de Python como Teano, Keras y scikit-learn que están disponibles para aplicar algoritmos de inteligencia artificial, incluyendo *machine learning*, *deep learning*, procesamiento del lenguaje natural (NLP – *Natural Language Processing*),

etc. Numpy es una librería que permite resolver muchos cálculos diferentes y Pybrain sirve también para aplicar técnicas *machine learning* en Python.

Otra razón de su popularidad es que su sintaxis es muy simple y pueden ser aprendidas fácilmente, lo que hace que los algoritmos sean fáciles de implementar. Ofrece así, un acceso directo a los usuarios noveles para aplicar técnicas de análisis predictivo. Es uno de los lenguajes preferidos por los desarrolladores que buscan enmarcar mejor las preguntas en las investigaciones y ampliar las capacidades de los sistemas *machine learning* existentes.

2.3. El lenguaje Python: conceptos básicos e instalación

Conceptos básicos

Python trabaja en diferentes plataformas (Windows, Mac, Linux, Raspberry Pi, etc.), tiene una licencia de código abierto (*Python Software Foundation License*, compatible con la licencia GNU) y tiene una sintaxis muy similar al idioma inglés. Su sintaxis permite escribir programas con pocas líneas de código en comparación con otros lenguajes.

Python es un lenguaje interpretado, como JavaScript, PHP o Ruby (es decir, no es compilado como C/C++ o Pascal). Esto nos permite ejecutar líneas de programa (sentencias) una a uno en un intérprete en línea de comando o bien que el código completo se ejecute mediante un intérprete apropiado. Sin embargo, por defecto el intérprete comprueba que el código completo no contiene errores de sintaxis antes de iniciar su ejecución (otros intérpretes como el motor V8 de JavaScript no lo comprueban de antemano por defecto).

Es un lenguaje multiparadigma, puesto que soporta:

- ▶ **Programación orientada a objetos**, como C++, Java, C#, R, entre otros.
- ▶ **Programación imperativa**: es decir, las sentencias modifican el estado del programa, como en C/C++, C#, Java, R; la contraposición sería un lenguaje de programación declarativa, que no describe el flujo de un programa (ejemplos de lenguajes declarativos serían SQL, HTML o XML).

- ▶ En concreto es de programación procedural, es decir, basado en subrutinas.
- ▶ **Programación funcional**, basado en el cálculo lambda y en el que existen subrutinas inspiradas en funciones matemáticas cuya invocación no modifica el estado del programa.

No obstante, la clasificación de los lenguajes de programación en un tipo u otro no siempre es tarea fácil y es habitual que, como en Python, se puedan encajar en diferentes paradigmas al mismo tiempo.

Respecto a su tipado, se puede decir que Python es:

- ▶ **Fuertemente tipado**: es decir, no se permiten violaciones en los tipos de datos de las variables; de modo que el programador ha de cambiar entre uno y otro tipo de datos mediante conversión de tipos explícita (*casting*); ejemplos de lenguajes fuertemente tipados son C/C++, Java, C#, Go o TypeScript (evolución de JavaScript), mientras que ejemplos de lenguajes no tipados serían JavaScript, PHP, Perl, Ruby o Basic.
- ▶ **De tipado dinámico**: habitual en lenguajes interpretados, lo que permite a una variable tomar valores de un tipo u otro, por supuesto, mediante la conversión apropiada; no se debe confundir esto con que sea débilmente tipado, que no lo es; un ejemplo de lenguaje de tipado estático sería C/C++, donde una variable no puede cambiar de tipo en el tiempo; otro ejemplo de lenguaje de tipado dinámico sería JavaScript.

Instalación y primeras pruebas

Podemos comprobar si tenemos ya instalado Python en nuestro ordenador ejecutando la siguiente orden en una consola apropiada (Windows, Linux o Mac):

```
PS C:\Users\xxx> python --version
```

```
Python 3.8.0
```

Lo habitual es que trabajemos ya con una versión 3.8, lanzada en 2019.

En caso contrario, lo descargaremos de la siguiente URL, seleccionando el sistema operativo y arquitectura adecuada: <https://www.python.org/>

La instalación de Python por defecto ya incluye el gestor de paquetes de Python (PIP), desde la versión 3.4.

En caso de que trabajemos con sistemas Linux, podemos buscar el paquete apropiado en nuestro gestor de paquetes, por ejemplo, en el caso de Ubuntu u otras distribuciones basadas en Debian:

```
sudo apt-get update
```

```
sudo apt-get install python
```

```
sudo apt-get install pip
```

Otra opción es utilizar en Windows o Linux gestores de paquetes como Conda o Miniconda que ya incluyen entornos, intérpretes y gestión de paquetes de librerías a emplear. Disponible en <https://docs.conda.io/>

Es interesante también contar con un IDE (entorno de desarrollo integrado) apropiado, algunas opciones gratuitas son Atom o Visual Studio Code, ambas basadas en el framework Electron, y ambas con su propio sistema de extensiones con resaltado de código y *linters* (nos permiten detectar errores de sintaxis mientras

escribimos) para diferentes lenguajes, incluido Python. Disponibles en:

- ▶ <https://atom.io/>
- ▶ <https://code.visualstudio.com/>

Los archivos de nuestro código Python tendrán extensión .py (habitualmente, hay variantes) y podremos ejecutar nuestro código tanto de forma interactiva en un terminal mediante el intérprete en línea de Python:

```
PS C:\Users\xxx > python
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64
bit (AMD64)] on win32
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> print("¡Bienvenidos al tema 2 de la asignatura!")
```

```
¡Bienvenidos al tema 2 de la asignatura!
```

```
>>>
```

O bien creando un archivo con nuestro código (próxima Figura), que luego ejecutaremos con el intérprete (próximo código):

```
print("Hola a todos")
```

```
PS C:\Users\ralorin> python.\test.py
```

```
Hola a todos
```

2.4. La sintaxis de Python

En este apartado veremos las reglas esenciales del lenguaje Python que son necesarias para realizar técnicas de análisis de datos y *machine learning*, así como utilizar las librerías específicas para ello. Evidentemente el lenguaje Python es mucho más amplio que la descripción incluida en este apartado, pero el alumno puede ampliar su conocimiento del lenguaje utilizando para ello los medios aportados al final del tema si así lo requiere, tanto para el uso de Python en otras aplicaciones, como para ampliar sus conocimientos en cuanto a la aplicación de técnicas de inteligencia artificial.

Indentación y comentarios

A diferencia de otros lenguajes, en los cuales no es importante y solo se hace por legibilidad del código, en Python es importante el sangrado (*indentación*) a la hora de definir los distintos bloques de ejecución. Hemos de usar siempre el mismo nivel de sangrado para cada bloque y el deshacerlo hace que se cierre dicho bloque. Además, a diferencia de otros lenguajes de programación como C/C++, Java, C# o JavaScript, no se finaliza cada sentencia con un punto y coma, sino con una nueva línea. Podemos incluir comentarios en el programa de las siguientes formas, que no se ejecutarían, y podemos hacer uso de la palabra clave `pass` en los casos en que necesitemos que no se realice ninguna operación, pero no romper la lógica del programa (quizás más adelante en nuestro desarrollo este `pass` pase a ser nuevo código que actualmente dejamos para más adelante).

```
"""
Ejemplo de programa básico en Python
con el fin de ver indentación y formas de comentarios
"""
if 5 > 2:
    print("5 es mayor que 2")
else:
    # nunca deberíamos estar en este punto del programa
    pass # esta sentencia no hace nada
```

Variables, tipos de datos y casting

En Python no es necesario declarar una variable (a diferencia de lenguajes de tipado estático como C/C++), esta se crea en el momento de utilizarla por primera vez, momento en el cual se define su tipo, que además puede cambiar con el tiempo con una nueva asignación de la variable a un tipo nuevo:

```
x = 5
# ahora x es un número
print(x)          # 5
print(type(x))   # <class 'int'>
print(type(x).__name__) # int

x = "¡hola!"
# ahora x es una cadena
print(x)          # "¡hola!"
print(type(x))   # <class 'str'>
print(type(x).__name__) # str
```

Los nombres de las variables (y del resto de palabras clave u otros elementos como operadores) empleados en Python son case-sensitive (debemos respetar uso de mayúsculas y minúsculas) y pueden contener letras, números y el carácter de subrayado, pero no pueden comenzar por un número. Ejemplos válidos: miVariable, mi_variable, variable1, _variable2

Podemos asignar el mismo o distinto valor a varias variables en una única sentencia, pero esto no las asocia entre sí de ninguna forma.

Podemos usar la función `print()` para imprimir por consola literales y variables, pero no podemos concatenar cadenas y números como se hace en otros lenguajes (como JavaScript):

Por defecto, una variable definida fuera de una función es global, y cuando es definida dentro de una función es local, a no ser que utilicemos la palabra clave `global` para modificar este comportamiento. Una variable global puede ser empleada desde cualquier punto del programa, pero una variable local solo puede ser empleada dentro de la función que la define.

```

x = 5 # variable global

def miFuncion1(a):
    b = 1 # b es local
    return a + b

def miFuncion2(a):
    x = 2 # esta x es local y diferente a x global
    return a + x

def miFuncion3(a):
    global y # y es global
    y = 1
    return a + y

def miFuncion4(a):
    global x # para modificar el valor de la x global
    x = 0
    return a + x

print(miFuncion1(1))    # 4
print(miFuncion2(2))    # 4
print(miFuncion3(3))    # 1
print(y)                # 1
print(miFuncion4(4))    # 4
print(x)                # 0

```

La Tabla 1 muestra los diferentes tipos de datos *built-in* (incorporados *per se*) en Python, aunque el desarrollador puede crear sus propios tipos de datos (clases).

Tipos	Python
Texto	str
Numéricos	int, float, complex
Secuencia	list, tuple, range
Mapas	dict
Conjuntos	set, frozenset
Booleanos	bool
Binarios (no nos interesan, pero existen)	bytes, bytearray, memoryview

Tabla 1. Tipos de datos en Python.

Como se puede ver, existen cuatro tipos de colecciones o arrays en Python, mucho más rico en este sentido que otros lenguajes de programación donde no existen estos tipos de forma incorporada en el lenguaje. Como se puede ver, se pueden mezclar los tipos de los elementos que componen una colección.

Tipo en Python	Descripción	Ejemplos
list	Secuencia ordenada de objetos	[1, 2, 2, "plátano", 1j, True] [0.5, 3.2, 3.2]
tuple	Secuencia ordenada de objetos no modificables (inmutables)	(1, 2, 2, "plátano", 1j, True) (0.5, 3.2, 3.2)
set	Colección de objetos únicos sin orden	{1, 2, 3, "plátano", 1j, True} {"plátano", "fresa"}
dict	Pares clave-valor no ordenados	{"nombre": "Pedro", "edad": 39}

Tabla 2. Tipos de colecciones en Python.

El tipo del dato se asigna automáticamente en el momento de asignar un valor a una variable:

```
x = "Hola"          # str
x = 5               # int
x = 2.5             # float
x = 2 + 1j           # complex
x = ["perro", "gato"] # list
x = ("perro", "gato") # tuple
x = range(5)         # range(0,5)
x = {"nombre": "Pedro", "edad": 39} # dict
x = {"perro", "gato"}    # set
x = True             # bool
```

Para las cadenas podemos usar comillas simples o dobles, según nos convenga.

O se puede especificar explícitamente utilizando el método constructor de cada tipo. Nótese que en este caso el argumento de las colecciones es siempre una tupla dentro del operador paréntesis de la función constructor. En el caso de los diccionarios, se emplea la notación de invocación a la función especificando el valor de cada parámetro específico.

```
x = str("Hola")
x = int(5)
x = float(2.5)
x = complex(2 + 1j)
x = list(("perro", "gato"))
x = tuple(("perro", "gato"))
x = range(5)
x = dict(nombre="Pedro", edad=39)
x = set(("perro", "gato"))
x = bool(3)      # True
```

Cuando trabajemos con números podemos hacerlo con enteros, flotantes o complejos, y la conversión (*casting*) entre unos y otros puede hacerse de forma explícita mediante el uso de métodos constructor:

```
a = 2      # int
b = 2.5    # float
c = 12e3   # float
d = -12.5E5 # float
e = 2j     # complex
f = complex(c)
g = int(2.5)  # 2
h = float(1)   # 1.0
# i = float(2j) # no se puede castear desde complex
```

También se puede castear entre números y cadenas:

```
a = int("3")    # 3
b = str(2)      # "2"
c = str(3.0)    # "3.0"
```

Podemos averiguar si un objeto es de un tipo determinado con la siguiente función:

```
x = 5
print(isinstance(x, int))
#> True
```

Cadenas

Las cadenas funcionan como el resto de los arrays en cuanto a que son iterables y cada elemento es accesible mediante el uso de corchetes. Cada elemento es un carácter Unicode, pero no existe un tipo *char* como en otros lenguajes. Un carácter es una cadena de tamaño 1. Podemos acceder a los elementos de la cadena de forma selectiva utilizando las siguientes posibilidades (similares al resto de colecciones, como veremos más adelante).

```
x = "Bienvenidos a la asignatura IA"  
# arrays comienzan en 0  
print(x[1])                      #> "i"  
# rango [2, 6), siempre el derecho exclusive  
print(x[2:6])                     #> "enve"  
# rango [-6, -4), siempre el derecho exclusive  
# contando desde el final del array  
# siendo el último elemento el -1  
print(x[-6:-4])                  #> "ur"
```

Existen, asimismo, algunos métodos para manipular cadenas que en algunos casos son similares en otras colecciones:

```
x = " Bienvenidos a la asignatura IA "  
  
print(x.strip())  
# "Bienvenidos a la asignatura IA"  
print(x.lower())  
# " bienvenidos a la asignatura ia "  
print(x.upper())  
# " BIENVENIDOS A LA ASIGNATURA IA "  
print(x.replace("B", "X"))  
# " Xienvenidos a la asignatura IA"  
print(x.split(" ")) # elegimos separador como argumento  
# ['', '', 'Bienvenidos', 'a', 'la', 'asignatura', 'IA', '', '']  
print("IA" in x)  
# True  
print("IA" not in x)  
# False
```

Es interesante aprender a formatear la salida de las cadenas para su uso en programas interactivos que creemos y ver los resultados de nuestro análisis de datos:

```
materia = "IA"
alumnos = 150
mensaje = "En la asignatura " + materia
mensaje += " hay " + str(alumnos) + " alumnos"

print(mensaje)
#> En la asignatura IA hay 150 alumnos

print("En la asignatura {} hay {} alumnos".format(materia, alumnos))
#> En la asignatura IA hay 150 alumnos

print("Hay {1} alumnos en la asignatura {0}".format(materia, alumnos))
#> Hay 150 alumnos en la asignatura IA
```

Expresiones booleanas

Los bool pueden tomar valores True o False y nos sirven para evaluar expresiones lógicas en el programa con el fin de controlar el flujo del programa, entre otras posibilidades. Cualquier elemento diferente de cero y colecciones no vacías castearán a True, mientras que números igual a cero o colecciones vacías castearán a False:

```
x = 5
y = 4
z = None      # de tipo especial NoneType
print(x > y)
#> True

# las siguientes expresiones son True
print(bool("Hola"))
print(bool(3))
print(bool(["perro", "gato"]))

# las siguientes expresiones son False
print(bool(z))
print(bool([]))
print(bool(0.0))
print(bool({}))
```

Operadores

En Python existen los siguientes tipos de operadores (que nos interesen):

- ▶ Aritméticos.
- ▶ Asignación.
- ▶ Comparación.
- ▶ Lógicos.
- ▶ Identidad.
- ▶ Membresía.

Operador	Nombre	Ejemplos
+	Adición	$x + y$
-	Sustracción	$x - y$
*	Multiplicación	$x * y$
/	División	x / y
%	Módulo (resto)	$x \% y$
**	Exponenciación	$x ** y$
//	División entera	$x // y$

Tabla 3. Operadores aritméticos en Python.

Operador	Ejemplo	Equivalencia
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x %= y$	$x = x \% y$
**=	$x **= y$	$x = x ** y$
//=	$x //= y$	$x = x // y$

Tabla 4. Operadores asignación en Python.

Operador	Nombre	Ejemplos
<code>==</code>	Igual	<code>x == y</code>
<code>!=</code>	No igual	<code>x != y</code>
<code>></code>	Mayor que	<code>x > y</code>
<code><</code>	Menor que	<code>x < y</code>
<code>>=</code>	Mayor o igual que	<code>x >= y</code>
<code><=</code>	Menor o igual que	<code>x <= y</code>

Tabla 5. Operadores comparación en Python.

Operador	Nombre	Ejemplos
<code>and</code>	Producto lógico	<code>x > 2 and x < 10</code>
<code>or</code>	Suma lógica	<code>x < 5 or x > 15</code>
<code>not</code>	Negación lógica	<code>not (x > 2 and x < 10)</code>

Tabla 6. Operadores lógicos en Python.

Operador	Descripción	Ejemplos
<code>is</code>	Devuelve True si ambas variables son el mismo objeto	<code>x is y</code>
<code>is not</code>	Devuelve True si las variables no son el mismo objeto	<code>x is not y</code>

Tabla 7. Operadores identidad en Python.

Los operadores identidad merecen una explicación y no hay que confundirlos con el operador igualdad, que compara los valores de dos variables. En Python al asignar una variable a otra, en realidad lo que estamos haciendo es asignar por referencia, no por valor. Es decir, no se duplica la memoria si no utilizamos un método que la clone o duplique explícitamente. Esto hace que modificar cualquier de las referencias que apunte a la misma memoria modificará la memoria y se verá alterada si la visualizamos desde cualquier otra variable que apunte a dicha dirección de memoria:

```
x = ["perro", "gato"]
y = ["perro", "gato"]
z = x

print(x == y)    # True
print(y is x)    # False
print(z == x)    # True
print(z is x)    # True

z[0] = "ratón"
print(x)        # ['ratón', 'gato']
```

Operador	Descripción	Ejemplos
in	Devuelve True si una secuencia con el valor especificado <i>x</i> se encuentra en el objeto <i>y</i>	<i>x</i> in <i>y</i>
not in	Devuelve True si ninguna secuencia con el valor especificado <i>x</i> se encuentra en el objeto <i>y</i>	<i>x</i> not in <i>y</i>

Tabla 8. Operadores membresía en Python.

2.5. Listas, tuplas, conjuntos y diccionarios

Tal y como vimos con anterioridad, existen cuatro tipos básicos (no son los únicos) de colecciones o arrays incluidos en el propio Python. Estos nos permitirán manejar los datos sobre los cuales realizar clasificaciones o regresiones, aunque, como veremos, las diferentes librerías *machine learning* pueden tener sus propios tipos de datos para manejar los datos.

Tipo en Python	Descripción	Ejemplos
list	Secuencia ordenada de objetos	[1, 2, 2, "plátano", 1j, True] [0.5, 3.2, 3.2]
tuple	Secuencia ordenada de objetos no modificables (inmutables)	(1, 2, 2, "plátano", 1j, True) (0.5, 3.2, 3.2)
set	Colección de objetos únicos sin orden	{1, 2, 3, "plátano", 1j, True} {"plátano", "fresa"}
dict	Pares clave-valor no ordenados	{"nombre": "Pedro", "edad": 39}

Tabla 9. Tipos de colecciones en Python.

Listas

Las listas son colecciones de objetos ordenadas y modificables. Se escriben utilizando corchetes con los elementos separados por comas. Podemos acceder a los diferentes elementos empleando el operador corchete y acceder a un rango de elementos, al igual que hacíamos con las cadenas. Asimismo, podemos utilizar la función `len()` y el método `append()` para añadir nuevos elementos.

```
x = ["perro", "gato", "ratón"]
print(len(x))
#> 3
print(x)
#> ['perro', 'gato', 'ratón']
x[1] = "gatete"
print(x)
#> ['perro', 'gatete', 'ratón']
print(x[1:])
#> ['gatete', 'ratón']
print(x[-2:-1])
#> ['gatete']
```

Podemos utilizar el operador `in` para iterar sobre los elementos de una lista o para comprobar si un elemento existe:

```
x = ["perro", "gato", "ratón"]

for e in x:
    print(e)

if "perro" in x:
    print("Hay un perro en la lista")
```

Existen diferentes métodos para añadir o quitar elementos de la lista:

```
x = ["perro", "gato", "ratón"]
print(x)
#> ['perro', 'gato', 'ratón']
x.append("loro")
print(x)
#> ['perro', 'gato', 'ratón', 'loro']
x.remove("gato") # elimina el primero que encuentre
print(x)
#> ['perro', 'ratón', 'loro']
x.pop() # elimina el último elemento (o el índice que especifiquemos)
print(x)
#> ['perro', 'ratón']
del x[0]
print(x)
#> ['ratón']
del x # elimina la lista entera
#print(x) # x ya no está definido
```

Es importante recordar que el operador asignación no duplica objetos, de modo que tenemos que clonar su contenido (duplicar los datos en otro espacio de la memoria) si no queremos alterar los datos originales al realizar operaciones. Esto sirve para cualquier colección de datos.

También podemos concatenar dos listas:

```
x = ["perro", "gato", "ratón"]
y = x.copy()      # método copia
z = list(x)       # otra opción: con el constructor

w = x + y        # concatenamos dos listas
print(w)
#> ['perro', 'gato', 'ratón', 'perro', 'gato', 'ratón']
```

Tuplas

Las tuplas son colecciones ordenadas pero inmutables, es decir, no podemos alterar su valor. Las tuplas funcionan de forma muy similar a las listas a la hora de acceder a sus elementos, iterarlas, comprobar si un elemento se encuentra en ellas, etc. Sin embargo, no podemos añadir o eliminar elementos o modificar el valor de uno de sus elementos. Para hacer esto, tendríamos que copiar su valor en una lista, realizar las modificaciones y asignar su contenido completo a la tupla original.

Como comentario particular, si queremos crear una tupla de un único elemento, hemos de añadir una coma al final de dicho elemento, para que no sea confundido con una cadena:

```
x = ("perro")    # str
x = ("perro",)   # tuple
```

Conjuntos

Los conjuntos (set) son colecciones de datos no ordenadas y que no permiten elementos duplicados. Es decir, no podemos predecir en qué orden se mostrarán (print) o iterarán sus elementos (for ... in). De hecho, cada vez que realicemos una ejecución de este tipo el orden puede ser distinto. Tampoco podemos utilizar el operador corchete para acceder a un elemento determinado, puesto que no tienen orden alguno. No pueden contener dos elementos con el mismo valor.

Una vez que creamos un conjunto, no podemos alterar el valor de uno de sus elementos. Lo que sí podemos hacer es añadir nuevos elementos o eliminar los existentes:

```
x = {"perro", "gato"}  
print(x)  
#> {'gato', 'perro'} # u otro orden diferente  
print("loro" in x)  
#> False  
x.add("loro")  
print(x)  
#> {'perro', 'loro', 'gato'} # u otro orden  
x.remove("perro")  
print(x)  
#> {'loro', 'gato'} # u otro orden  
x.discard("perro") # no produce error si no existe  
print(x)  
#> {'loro', 'gato'} # u otro orden  
x.update(("mosca", "pato"))  
print(x)  
#> {'loro', 'pato', 'mosca', 'gato'} # u otro orden
```

Diccionarios

Los diccionarios son el equivalente a mapas o hashes en otros lenguajes de programación. Son un conjunto de elementos clave-valor sin orden y en los cuales se puede modificar el contenido de los elementos. Son también similares a los objetos nativos de JavaScript que pueden ser convertidos a y desde JSON. Sin embargo, son mucho más flexibles y una clave no tiene por qué ser sólo de tipo cadena, sino que puede ser cualquier objeto, incluso un entero, un flotante o un complejo, por ejemplo.

Se puede acceder a cada uno de sus elementos mediante el operador indicando la clave determinada, así como el método `get()`:

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39
}

print(x)
#> {'nombre': 'Pedro', 'apellido': 'García', 'edad': 39}
y = x["nombre"]
print(y)
#> Pedro
z = x.get("nombre")
print(y)
#> Pedro
x["nombre"] = "Perico"
print(x)
#> {'nombre': 'Perico', 'apellido': 'García', 'edad': 39}
```

Tenemos varias formas de iterar por un diccionario:

```
x = {
    "nombre": "Pedro",
    "apellido": "García",
    "edad": 39
}

for i in x:
    print(x[i]) # valor

for i in x.values():
    print(i) # valor

for i, j in x.items():
    print(i, j) # clave valor
```

Si utilizamos el operador de membresía estaremos comprobando si una clave está en el diccionario (no un valor). Podemos usar también la función `len()` como con cualquier otra colección:

```
x = {  
    "nombre": "Pedro",  
    "apellido": "Garcia",  
    "edad": 39  
}  
  
print("apellido" in x)  
#> True  
print("Pedro" in x)  
#> False  
print(len(x))  
#> 3
```

Podemos añadir nuevos elementos clave-valor al diccionario o eliminarlos:

```
x = {  
    "nombre": "Pedro",  
    "apellido": "Garcia",  
    "edad": 39  
}  
  
x["país"] = "España"  
print(x)  
#> {'nombre': 'Pedro', 'apellido': 'Garcia', 'edad': 39, 'país': 'España'}  
x.pop("edad")  
print(x)  
#> {'nombre': 'Pedro', 'apellido': 'Garcia', 'país': 'España'}  
del x["apellido"]  
print(x)  
#> {'nombre': 'Pedro', 'país': 'España'}  
del x  
#print(x) # x ya no existe
```

Además, como cabría esperar, se pueden anidar diccionarios unos dentro de otros.

En realidad, Python nos deja realizar casi cualquier combinación de anidamientos, mezcla o combinaciones de tipos entre sí:

```
x = {  
    "nombre": "Pedro",  
    "apellido": "García",  
    "edad": 39,  
    "mascotas": ["perro", "gato"],  
    "cónyuge": {  
        "nombre": "Rosa",  
        "apellido": "Pérez",  
        "edad": 37,  
    }  
}
```

2.6. Librerías útiles para el análisis de datos

Aunque en este tema nos iniciaremos en el uso de Python para el machine learning usando las librerías NumPy, SciPy, Scikit-learn, Pandas y Matplotlib en las siguientes subsecciones como punto de partida, a lo largo del resto de temas de la asignatura veremos al final de cada tema ejemplos de uso de las librerías más importantes aplicadas a la solución de problemas mediante técnicas de inteligencia artificial, machine learning y deep learning.

Veamos, por tanto, una breve descripción de las librerías más importantes disponibles para Python. Es importante señalar que no son las únicas y que existen, por supuesto, otras muchas y en ocasiones especializadas en un tipo de problemas concreto (por ejemplo, OpenCV es una conocida librería de visión artificial disponible para C y Python). El alumno, como cualquier científico de datos o ingeniero de *machine learning*, ha de estar siempre abierto a la búsqueda de nuevos recursos para solucionar cada problema específico.

Las librerías más interesantes y que analizaremos brevemente son las siguientes, tomando ejemplos de uso de (GeeksforGeeks, 2019):

- ▶ NumPy.
- ▶ SciPy.
- ▶ Scikit-learn.
- ▶ Theano.

- ▶ Keras.
- ▶ TensorFlow.
- ▶ PyTorch.
- ▶ Pandas.
- ▶ Matplotlib.

Antes de nada, vamos a mencionar brevemente cómo instalar y emplear estas librerías llegado el momento.

Instalación de paquetes en Python

Para instalar una librería usaremos generalmente el gestor de paquetes que estemos empleando (PIP o Conda, por ejemplo). En el caso de que estemos usando PIP, podemos buscar la lista de paquetes, disponibles en: <https://pypi.org/>

La instalación de un paquete o librería concreto se realizaría desde línea de comandos como (en el caso de NumPy bajo Windows, por ejemplo):

```
PS C:\Users\xxx> pip install numpy --user
```

Podemos eliminar posteriormente un paquete mediante:

```
PS C:\Users\xxx> pip uninstall numpy
```

Así como ver la lista disponible de paquetes mediante:

```
PS C:\Users\xxx> pip list
```

Para utilizar un paquete en nuestro programa será tan sencillo como:

```
PS C:\Users\xxx> pip uninstall numpy
```

```
import numpy  
  
# otra alternativa si queremos usar alias...  
import numpy as np
```

NumPy

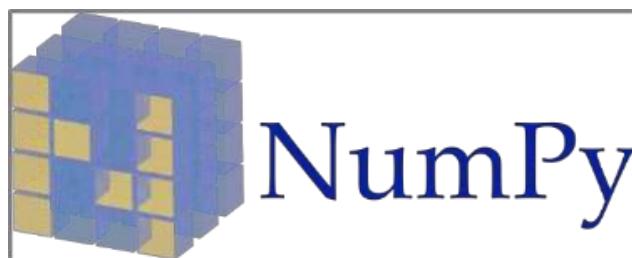


Figura 3. Fuente: <https://numpy.org/>

NumPy es una librería muy popular para el procesamiento de grandes matrices y matrices multidimensionales, con la ayuda de una gran colección de funciones matemáticas de alto nivel. Es muy útil para los cálculos científicos fundamentales en *machine learning*. Es particularmente útil para el álgebra lineal, la transformación de Fourier y las capacidades de números aleatorios. Otras librerías de alto nivel como TensorFlow utilizan NumPy internamente para la manipulación de tensores.

Nota: para que el siguiente ejemplo funcione es necesario instalar NumPy, como se ha indicado antes.

```
# Ejemplo de programa usando NumPy
# para realizar operaciones matemáticas básicas

import numpy as np

# Creación de dos tensores de rango 2 (matrices de 2x2)
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

# Creación de dos tensores de rango 1 (vectores de dimensión 2)
v = np.array([9, 10])
w = np.array([11, 12])

# Producto escalar de vectores
print(np.dot(v, w), "\n")

# Producto de matrix por un vector
print(np.dot(x, v), "\n")

# Producto de matrices
print(np.dot(x, y))
```

SciPy



Figura 4. Fuente: <https://www.scipy.org/>

SciPy es una biblioteca muy popular entre los ingenieros de *machine learning*, ya que contiene diferentes módulos para la optimización, el álgebra lineal, la integración y la estadística. Hay una diferencia entre la biblioteca de SciPy y la pila de SciPy (*SciPy stack*). La librería SciPy es uno de los paquetes centrales que componen la pila de SciPy. SciPy también es muy útil para la manipulación de imágenes.

En el siguiente ejemplo podemos ver cómo manipular una imagen utilizando diferentes librerías con SciPy. Podemos elegir una imagen cualquiera en JPEG en nuestro ordenador y ver el resultado de las operaciones descritas.

Nota: para que el siguiente ejemplo funcione es necesario instalar numpy, como se ha indicado antes, así como las librerías scipy, imageio, Pillow, scikit-image.

```
# Manipulación de imágenes mediante SciPy
import imageio
import skimage
from skimage import transform

# Lectura de una imagen JPEG a un array numpy
img = imageio.imread("C:/Users/xxx/gato.jpg")
print(img.dtype, img.shape)

# Tintamos la imagen
img_tint = img * [1, 0.45, 0.3]

# Guardamos la imagen tintada
imageio.imwrite("C:/Users/xxx/gato_tintado.jpg", img_tint)

# Redimensionamos la imagen tintada
img_tint_resize = skimage.transform.resize(img_tint, (300, 300))

# Guardamos la imagen resultante
imageio.imwrite("C:/Users/xxx/gato_tintado_redimensionado.jpg", img_tint_resize)
```

Scikit-learn



Figura 5. Fuente: <https://scikit-learn.org/>

Scikit-learn es una de las librerías de *machine learning* más populares para los algoritmos de *machine learning* clásico. Está construida sobre dos bibliotecas básicas de Python, NumPy y SciPy. Scikit-learn soporta la mayoría de los algoritmos de aprendizaje supervisado y no supervisado. Scikit-learn también puede ser usado para la minería de datos y el análisis de datos, lo que lo convierte en una gran

herramienta para los que empiezan con el *machine learning*.

En el siguiente ejemplo se importa uno de los *dataset* más conocidos en el aprendizaje de *machine learning* (el dataset «iris», con dimensiones de pétalos y sépalos de flores) y se utiliza un clasificador de árbol de decisión (algoritmo CART, que veremos en el Tema 3). Este es un buen ejemplo, además, de importación tanto de datos como de modelos.

Nota: es necesario instalar el paquete `sklearn` (o `scikit-learn`) para su funcionamiento.

```
# Python script using Scikit-learn
# for Decision Tree Clasifier

# Ejemplo con Scikit-learn
# utilizando un árbol de decisión
from sklearn import datasets
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier

# cargamos el dataset "iris"
dataset = datasets.load_iris()

# ajustamos los datos mediante un modelo CART
model = DecisionTreeClassifier()
model.fit(dataset.data, dataset.target)
print(model)

# realizamos predicciones
expected = dataset.target
predicted = model.predict(dataset.data)

# resumen del ajuste (fit) del modelo
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
```

Obteniendo como resultado:

```
DecisionTreeClassifier(
    ccp_alpha=0.0,
    class_weight=None,
    criterion='gini',
```

```
max_depth=None,  
  
max_features=None,  
  
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,  
min_samples_leaf=1,  
  
min_samples_split=2,  
  
min_weight_fraction_leaf=0.0,  
  
presort='deprecated',  
  
random_state=None,  
  
splitter='best')
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	50
2	1.00	1.00	1.00	50
accuracy			1.00	150
macro avg	1.00	1.00	1.00	150
weighted avg	1.00	1.00	1.00	150

```
[[50 0 0]  
  
[ 0 50 0]  
  
[ 0 0 50]]
```

Theano



Figura 6. Fuente: <http://deeplearning.net/software/theano/>.

El *machine learning* es básicamente matemáticas y estadística. Theano es una popular librería que se utiliza para definir, evaluar y optimizar las expresiones matemáticas que implican conjuntos multidimensionales de manera eficiente. Se consigue optimizando la utilización de la CPU y la GPU. Se utiliza ampliamente para pruebas unitarias y autoverificación para detectar y diagnosticar diferentes tipos de errores. Theano es una biblioteca muy potente que se ha utilizado en proyectos científicos de gran escala e intensivos en computación durante mucho tiempo, pero es lo suficientemente sencilla y accesible para que pequeños desarrolladores la utilicen en sus propios proyectos.

Nota: es necesario instalar el paquete theano para el funcionamiento del siguiente ejemplo.

```
import theano
import theano.tensor as T
x = T.dmatrix('x')
s = 1 / (1 + T.exp(-x))
logistic = theano.function([x], s)
y = logistic([[0, 1], [-1, -2]])
print(y)
```

Obteniendo como resultado:

[[0.5 0.73105858]

[0.26894142 0.11920292]]

Keras



Figura 7. Fuente: <https://keras.io/>

Keras es una de las librerías de *Machine Learning* más populares para Python. Es una API de redes neuronales de alto nivel capaz de funcionar sobre TensorFlow, CNTK, o Theano. Puede funcionar sin problemas en la CPU y la GPU. Keras permite que sea realmente sencillo construir y diseñar redes neuronales para los principiantes en el *machine learning*. Una de las mejores características de Keras es que permite la creación de prototipos de forma fácil y rápida.

Veremos ejemplos de su uso en el Tema 5 de redes neuronales artificiales y en el Tema 6 de *deep learning*.

TensorFlow



Figura 8. Fuente: <https://www.tensorflow.org/>

TensorFlow es una biblioteca de código abierto muy popular para el cálculo numérico de alto rendimiento, desarrollada por el equipo de Google Brain en Google. Como su nombre lo sugiere, TensorFlow es un marco de trabajo que implica la definición y

ejecución de cálculos que implican tensores.

Un tensor es cierta clase de entidad algebraica que generaliza los conceptos de escalar, vector y matriz de una forma que sea independiente de cualquier sistema de coordenadas elegido.

TensorFlow puede entrenar y ejecutar redes neuronales profundas que pueden ser usadas para desarrollar varias aplicaciones de IA. TensorFlow es ampliamente utilizado en el campo de la investigación y aplicación del *machine learning*.

Veremos más ejemplos de su uso en el Tema 5 de redes neuronales artificiales y en el Tema 6 de *deep learning*.

TensorFlow fue lanzado en 2015 como una herramienta de código abierto por Google. Inicialmente estaba disponible en Python. Posteriormente se lanzó TensorFlow.js para su uso mediante JavaScript, bien sea en el mismo navegador o bien del lado del servidor mediante Node.js o incluso en una Raspberry Pi. Más tarde se lanzó TensorFlow 2.0, disponible para C++, Haswell, Java, Go y Rust. También hay bibliotecas de terceros para C#, R y Scala.

Existen versiones de TensorFlow para Windows, Linux y Mac. Además, TensorFlow Lite es la versión de TensorFlow para su uso en terminales móviles o en dispositivos en el borde la red (*Edge Computing*), incluyendo escenarios Internet of Things.

Más aún, Google lanzó en 2016 su unidad de procesamiento tensorial (TPU), un circuito ASIC (Circuito Integrado de Aplicación Específicas) personalizada específicamente para *machine learning* y adaptada para TensorFlow. El TPU es un acelerador de IA programable diseñado para proporcionar alto *throughput* de aritmética de precisión baja (p. ej., 8 bits) y orientado para utilizar o correr modelos más que para entrenarlos. Este tipo de unidades tienen un rendimiento de *machine*

learning por vatio consumido de un orden mayor de magnitud que los sistemas tradicionales. Google Cloud ofrece el uso de TPU en la nube que pueden ser empleados por el público general.

Nota: es necesario instalar el paquete tensorflow o tensorflow-gpu para el funcionamiento del siguiente ejemplo (instalará más de 400MB en el ordenador, incluyendo dependencias, como parte de Keras). Además, bajo Windows es necesario contar con la última versión de Visual C++ Redistributable para su funcionamiento:

<https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads>

```
# Operaciones con tensores (arrays)
# Descarga e instala el paquete TensorFlow 2.0 version. Importa TensorFlow en tu programa:
from __future__ import absolute_import, division, print_function, unicode_literals

# Installa TensorFlow

import tensorflow as tf

# Carga y prepara el conjunto de datos MNIST. Convierte los ejemplos de numeros enteros a numeros de punto flotante:

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
# Construye un modelo tf.keras.Sequential apilando capas. Escoge un optimizador y una función de pérdida para el entrenamiento de tu modelo:

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Entrena y evalúa el modelo:

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test, verbose=2)
```

Veremos una salida similar a la siguiente en el momento en que se descarga el *dataset* MINIST (MNIST es una gran base de datos de dígitos escritos a mano que se usa comúnmente para entrenar varios sistemas de procesamiento de imágenes) de la web de Lecun (Yann Lecun, h=122, es, junto a Geoffrey Hinton y Yoshua Bengio, uno de los conocidos como los tres *Padrinos de la Inteligencia Artificial* o *Padrinos del Machine Learning*).

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
```

```
11493376/11490434 [=====] - 1s 0us/step
```

Y una salida similar a la siguiente durante el entrenamiento y la evaluación del modelo:

```
Train on 60000 samples
```

```
Epoch 1/5
```

```
60000/60000 [=====] - 4s 65us/sample - loss: 0.2982 - accuracy: 0.9128
```

```
Epoch 2/5
```

```
60000/60000 [=====] - 3s 56us/sample - loss:  
0.1424 - accuracy: 0.9571
```

Epoch 3/5

```
60000/60000 [=====] - 3s 57us/sample - loss:  
0.1073 - accuracy: 0.9677
```

Epoch 4/5

```
60000/60000 [=====] - 3s 57us/sample - loss:  
0.0891 - accuracy: 0.9714
```

Epoch 5/5

```
60000/60000 [=====] - 4s 59us/sample - loss:  
0.0745 - accuracy: 0.9768
```

```
10000/10000 - 1s - loss: 0.0760 - accuracy: 0.9781
```

```
[0.0760388328890549, 0.9781]
```

PyTorch



Figura 9. Fuente: <https://pytorch.org/>

PyTorch es otra de las librerías más populares de *machine learning* de código abierto para Python basada en Torch, que es una biblioteca de *machine learning* de código abierto que se implementa en C con un *wrapper* en Lua. Tiene una extensa selección de herramientas y librerías que incluye visión artificial (*Computer Vision*) o Procesamiento de Lenguaje Natural (NLP), entre otros. Permite a los desarrolladores realizar cálculos tensoriales con aceleración en la GPU y también es de ayuda en la

creación de gráficos computacionales.

Nota: es necesario instalar el paquete torch para el funcionamiento del siguiente ejemplo. Sin embargo, conviene hacerlo de la siguiente forma explicada en la web incluso aunque se emplee PIP o Conda. De esta forma se utilizará PIP o Conda, pero descargando el paquete desde la web: <https://pytorch.org/get-started/locally/>

Por ejemplo, para el uso de PIP en Windows y sin un sistema que soporte CUDA (*Compute Unified Device Architecture – Arquitectura Unificada de Dispositivos de Cómputo*) de NVIDIA, seleccionaríamos:



Resultando el siguiente comando a ejecutar en nuestra consola:

```
pip install torch==1.4.0+cpu torchvision==0.5.0+cpu -f
https://download.pytorch.org/whl/torch_stable.html
```

```
# Ejemplo para definir tensores que ajusten a una red neuronal de dos
# capas
# a datos aleatorios y calcule la pérdida
import torch

dtype = torch.float
device = torch.device("cpu")          # Para su uso en la CPU
# device = torch.device("cuda:0")      # Para su uso en la GPU

# N = batch size;
# D_in = dimensión capa de entrada
# H = dimensión capa oculta
# D_out = dimensión capa de salida
N, D_in, H, D_out = 64, 1000, 100, 10

# Creamos datos aleatorios de entrada y de salida
x = torch.randn(N, D_in, device = device, dtype = dtype)
y = torch.randn(N, D_out, device = device, dtype = dtype)

# Inicializamos a pesos aleatorios
w1 = torch.randn(D_in, H, device = device, dtype = dtype)
w2 = torch.randn(H, D_out, device = device, dtype = dtype)

learning_rate = 1e-6
for t in range(500):
    # Paso hacia adelante: calculamos la "y" predicha
    h = x.mm(w1)
    h_relu = h.clamp(min = 0)
    y_pred = h_relu.mm(w2)

    # Calculamos la pérdida
    loss = (y_pred - y).pow(2).sum().item()
    print(t, loss)

    # Retropropagación para calcular gradientes de w1 y w2 respecto a
    # las pérdidas
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Actualizamos los pesos usando gradient descentiente
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

Veremos la siguiente salida con las 500 iteraciones.

0 31596260.0

1 29391412.0

```
2 31699210.0  
...  
498 4.088189234607853e-05  
499 4.0442959289066494e-05
```

Pandas



Figura 10. Fuente: <https://pandas.pydata.org/>

Pandas es una popular librería de Python para el análisis de datos. No está directamente relacionada con el *machine learning*. Como sabemos, los *dataset* han sido preparados antes del entrenamiento. En este caso, Pandas es muy útil, ya que fue desarrollada específicamente para la extracción y preparación de datos. Proporciona estructuras de datos de alto nivel y una amplia variedad de herramientas para el análisis de datos. Asimismo, proporciona muchos métodos incorporados para tantear, combinar y filtrar datos.

Nota: es necesario instalar el paquete pandas para el funcionamiento del siguiente ejemplo.

```
# Ejemplo para ordenar un conjunto de datos en una tabla

import pandas as pd

data = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
        "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],
        "area": [8.516, 17.10, 3.286, 9.597, 1.221],
        "population": [200.4, 143.5, 1252, 1357, 52.98] }

data_table = pd.DataFrame(data)
print(data_table)
```

	country	capital	area	population
0	Brazil	Brasilia	8.516	200.40
1	Russia	Moscow	17.100	143.50
2	India	New Dehli	3.286	1252.00
3	China	Beijing	9.597	1357.00
4	South Africa	Pretoria	1.221	52.98

Matplotlib



Figura 11. Fuente: <https://matplotlib.org/>

Matplotlib es una biblioteca de Python muy popular para la visualización de datos. Al igual que Pandas, no está directamente relacionada con el *machine learning*. Es particularmente útil cuando un programador quiere visualizar los patrones de los datos. Es una librería de ploteo en 2D usada para crear gráficos y diagramas en 2D. Un módulo llamado pyplot facilita a los programadores el trazado, ya que proporciona

características para controlar los estilos de línea, las propiedades de las fuentes, los ejes de formato, etc. Proporciona varios tipos de gráficos y diagramas para la visualización de datos, es decir, histogramas, tablas de error, chats de barras, etc.

Nota: es necesario instalar el paquete matplotlib para el funcionamiento del siguiente ejemplo.

```
# Ejemplo sencillo para trazar una función lineal

import matplotlib.pyplot as plt
import numpy as np

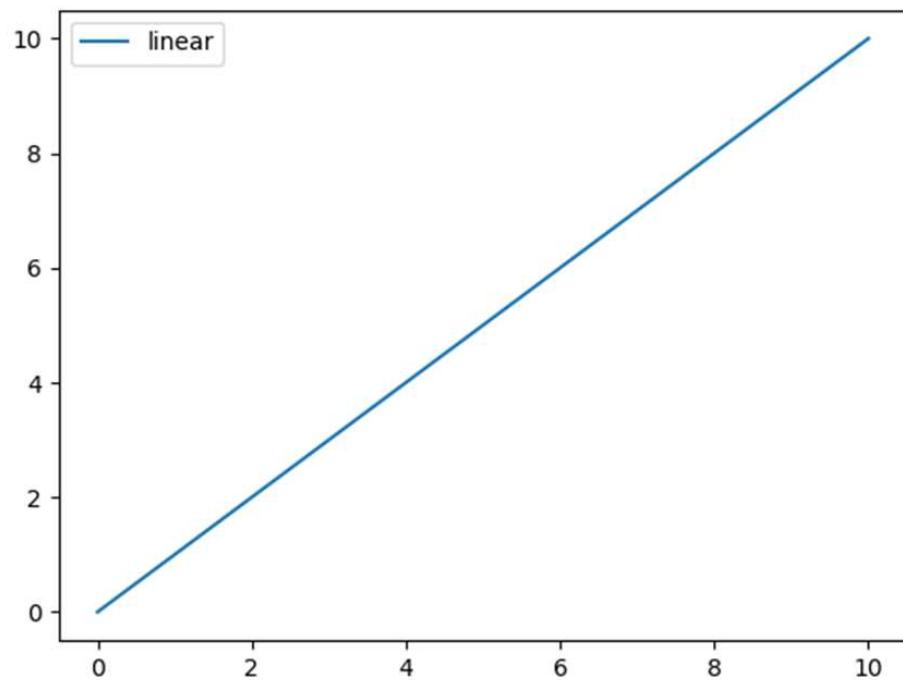
# Preparamos los datos
x = np.linspace(0, 10, 100)

# Dibujamos los datos
plt.plot(x, x, label='lineal')

# Incluimos una leyenda
plt.legend()

# Mostramos el gráfico
plt.show()
```

La salida se mostrará en una ventana que nos permitirá guardar la imagen en un archivo PNG como el siguiente.



2.7. La librería NumPy para el manejo de datos

Conceptos básicos y arrays en NumPy

Tal y como hemos visto, la librería NumPy nos permite trabajar con arrays de forma muy eficiente, así como álgebra lineal, transformadas de Fourier y matrices. Aunque Python dispone de listas que permiten trabajar con arrays, estas son muy lentas. NumPy proporciona arrays 50 veces más rápidos, en parte porque los elementos están todos almacenados en posiciones contiguas de memoria y en parte porque parte de NumPy está escrita en C/C++.

Una vez hemos instalado NumPy siguiendo las explicaciones de las secciones anteriores, podemos usar los arrays de NumPy (objetos de clase `darray`), de la siguiente forma:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
#> [1 2 3 4 5]
print(type(arr))
#> <class 'numpy.ndarray'>
```

Podemos trabajar con escalares, vectores, matrices, arrays tridimensionales o cualquier otra dimensión diferente. El método `ndim` nos permite obtener la dimensión del array, mientras que con el operador corchete podemos acceder a los diferentes elementos según los índices en cada dimensión. Los índices negativos son válidos para acceder desde el elemento final en una dimensión (-1 sería el último elemento).

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim) # 1
print(b.ndim) # 2
print(c.ndim) # 2
print(d.ndim) # 4

print(a)      # 42
print(b[3])   # 4
print(c[0, 1]) # 2
print(d[0, 1, 1]) # 5
```

Array Slicing

Al igual que con los tipos built-in datos (cadenas, listas, etc.), en los arrays NumPy se puede hacer un slicing de los arrays utilizando notación de corchetes y dos puntos, pero, con mucha mayor flexibilidad:

- ▶ array[start:end]
- ▶ array[start:end:step]
- ▶ Si no pasamos un comienzo se considera 0.
- ▶ Si no pasamos un final se considera la longitud del array en esa dimensión.
- ▶ Si no pasamos un step se considera 1.
- ▶ Siempre se incluye el elemento en la posición *start*, pero se excluye el elemento en la posición *end*.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
#> [5 6 7]
print(arr[:4])
#> [1 2 3 4]
print(arr[-4:-1])
#> [4 5 6]
print(arr[1:5:2])
#> [2 4]

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
#> [[2 3 4]
#   [7 8 9]]
```

Tipos de datos en NumPy

NumPy nos permite especificar con mucho mayor detalle los tipos de datos de los elementos en nuestros arrays:

- ▶ i – integer
- ▶ b – boolean
- ▶ u - unsigned integer
- ▶ f – float
- ▶ c - complex float
- ▶ m – timedelta
- ▶ M – datetime
- ▶ O – object

- ▶ S – string
- ▶ U - unicode string
- ▶ V - *chunk* o trozo de memoria para otro tipo de dato (void)

En los tipos de datos i, u, f, S, U podemos especificar el tamaño en bytes, además.

Esto se especifica con la propiedad `dtype`:

```
import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
#> [1 2 3 4]
print(arr.dtype)
#> int32
```

Si un valor no puede ser convertido, se lanzará una excepción de tipo `ValueError`.

Si queremos convertir el tipo de datos de un array existente, usaremos el método `astype()`.

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

newarr = arr.astype(int)
# newarr = arr.astype("i") # alternativa

print(newarr)
#> [1 2 3]
print(newarr.dtype)
#> int32
```

Copy y view

La principal diferencia entre una copia (obtenida con el método `copy()`) y una vista (obtenida con el método `view()`) de un array NumPy es que la copia es un nuevo array, mientras que la vista es una vista, valga la redundancia, del array original. Es

similar al concepto de asignación por referencia o de clonado de los datos en Python. Si modificamos los datos en una copia, esto no afectará al original, y viceversa. Por el contrario, si modificamos los datos en una vista, esto afectará al original, y viceversa. Podemos comprobar si un array es propietario de sus propios datos mediante el atributo base. Este devuelve el array original si el array es una vista. En caso de que el array sea una copia, este atributo devuelve None.

```
import numpy as np

# COPY
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 39

print(arr)
#> [39  2  3  4  5]
print(x)
#> [1 2 3 4 5]
print(arr.base)
#> None
print(x.base)
#> None

# VIEW
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 39

print(arr)
#> [39  2  3  4  5]
print(x)
#> [39  2  3  4  5]
print(arr.base)
#> None
print(x.base)
#> [39  2  3  4  5]
```

La forma (shape) de un array

Con el atributo shape podemos obtener una tupla indicando en cada índice el número de elementos en cada dimensión. Si hemos definido una dimensión mínima con ndmin en la creación del array, esta será una forma de visualizarlo. El método reshape() nos permite redimensionar un array (cambios entre 1-D y 2-D o 3D). **Dicho método devuelve una vista, no una copia.** Con reshape(-1) podemos aplanar un array multidimensional en un array unidimensional.

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
#> (2, 4)

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]], ndmin=3)
print(arr.shape)
#> (1, 2, 4)
newarr = arr.reshape(-1)
print(newarr)
#> [1 2 3 4 5 6 7 8]

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 4)
print(newarr)
#> [[1 2 3 4]
#   [5 6 7 8]]
```

Iterando por los arrays

Podemos usar bucles *for-in* para iterar por los arrays, pero necesitamos uno por cada dimensión:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in arr:
    print(x)
# 1
# 2
# 3

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr: # iteraremos por cada fila
    for y in x: # iteraremos por cada elemento en cada fila
        print(y)
# 1
# 2
# 3
# 4
# 5
# 6
```

Aunque contamos también con métodos auxiliares para iterar, como `nditer()`:

```
import numpy as np

arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])

for x in np.nditer(arr):
    print(x)
# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
```

Operaciones con varios arrays

Podemos concatenar arrays uno a continuación del otro con concatenate(), especificando sobre qué eje (si no se especifica, se considerará el eje 0):

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

arr = np.concatenate((arr1, arr2))
print(arr)
#> [1 2 3 4 5 6]

arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
#> [[1 2 3 4]
#> [5 6]]
```

Podemos dividir arrays utilizando el método `split()`, por ejemplo, subdividiendo un array 2-D en tres arrays 2-D:

```
import numpy as np

arr = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12]])
newarr = np.array_split(arr, 3)

print(newarr)

#> [array([[1, 2],
#           [3, 4]]), array([[5, 6],
#           [7, 8]]), array([[9, 10],
#           [11, 12]])]
```

Búsquedas en arrays

Podemos utilizar el método `where()` para realizar búsquedas de los elementos que cumplan con una condición pasado como argumento a dicho método. Algunos sencillos ejemplos de su uso incluyen la búsqueda de un valor específico o buscar todos los elementos pares del array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 4, 4])

# buscamos los indices donde el elemento sea igual a 4
x = np.where(arr == 4)
print(x)
#> (array([3, 5, 6], dtype=int64),)

# buscamos los indices donde el elemento sea par
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
#> (array([1, 3, 5, 7], dtype=int64),)
```

Ordenando arrays

Con el método `sort()` obtenemos una copia del array, manteniendo el original inalterado. Si ordenamos un array 2-D, el orden se hará fila a fila:

```
import numpy as np

arr = np.array(['perro', 'gato', 'loro'])
print(np.sort(arr))
#> ['gato' 'loro' 'perro']

arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
#> [[2 3 4]
#> [0 1 5]]
```

Filtrado de arrays

Para ello, se usa un array de booleanos que permiten indicar mediante valores `True` o `False` si el elemento correspondiente se mantendrá o no tras el filtrado.

```
import numpy as np

arr = np.array([39, 40, 41, 42])

x = [True, False, True, False]

newarr = arr[x]
print(newarr)
#> [39 41]
```

2.8. Importación de datos

Aunque ya hemos visto algunas en los ejemplos en la sección de librerías, vamos a ver las diferentes formas que tenemos para importar datos en Python a la hora de aplicar métodos *machine learning* (Unipython, 2019; Recuero de los Santos, 2019):

- ▶ Importación de archivos en formato .txt.
- ▶ Importación de archivos planos CSV mediante la librería estándar de Python.
- ▶ Importación de archivos planos CSV mediante NumPy.
- ▶ Importación de archivos planos CSV mediante Pandas.
- ▶ Importación desde una URL.
- ▶ Importación desde otras librerías (*toy datasets*).

Importación de archivos en formato .txt

En todos los casos en los que manejemos ficheros, nos interesa asegurarnos primero de que estamos en el directorio correcto, a no ser que queramos utilizar una ruta absoluta:

```
import os
os.chdir("C:\\\\Users\\\\xxx")
print(os.getcwd())
```

Podemos leer los datos de un fichero de texto mediante el manejo básico de ficheros de texto:

```
import os
nombre = "datos.txt"
archivo = open(nombre, mode='r')
texto = archivo.read()
archivo.close()
# Podemos mostrar en pantalla el archivo que se acaba de cargar
print(texto)
```

Importación de archivos planos CSV mediante la librería estándar de Python

En el caso de que trabajemos con archivos CSV, hemos de tener en cuenta, no solo la ubicación, sino detalles como:

- ▶ Si tiene cabecera (*header*).
- ▶ Si tiene comentarios (incluidos con #).
- ▶ Qué tipo de delimitador utiliza el fichero (coma, punto y coma, espacios en blanco, etc.).

Podemos hacer pruebas descargando el archivo diabetes.csv de:
<https://datahub.io/machine-learning/diabetes/r/diabetes.csv>.

En el caso de la librería estándar de Python, lo cargaríamos con un código similar al siguiente, en el que abrimos el fichero para su lectura y luego empleamos el método csv.reader() para leer línea a línea el fichero y hace una lista de todas las columnas en el objeto reader.

```
import os
os.chdir("C:\\\\Users\\\\xxx")
os.getcwd()

import csv
f = open("diabetes.csv")
reader = csv.reader(f)
for row in reader:
    print (row)
```

Resultando a la salida:

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

['6', '148', '72', '35', '0', '33.6', '0.627', '50', '1']

['1', '85', '66', '29', '0', '26.6', '0.351', '31', '0']

...
```

Importación de archivos planos CSV mediante NumPy

```
import numpy
filename = 'diabetes.csv'
raw_data = open(filename)
data = numpy.loadtxt(raw_data, delimiter=",", skiprows=1)
print(data.shape)
print(data)
```

Resultando a la salida:

```
(768, 9)

[[ 6.    148.     72.     ...,    0.627   50.     1.    ]
 [ 1.    85.     66.     ...,    0.351   31.     0.    ]
 [ 8.    183.     64.     ...,    0.672   32.     1.    ]
 ...
 [ 5.    121.     72.     ...,    0.245   30.     0.    ]
 [ 1.    126.     60.     ...,    0.349   47.     1.    ]
 [ 1.    93.     70.     ...,    0.315   23.     0.    ]]
```

Importación de archivos planos CSV mediante Pandas

Esta forma es muy popular dadas las ventajas que ofrece Pandas en este sentido. Usaremos la función `readcsv()` para la carga, la cual nos ofrece una gran versatilidad a la hora de importar los datos. Con una única línea permite:

- ▶ Detectar automáticamente los headers sin que se lo tengamos que especificar.
- ▶ Saltar líneas con el modificador `skiprows`.
- ▶ Detectar automáticamente el tipo de datos (entero, flotante, cadenas, etc.)
- ▶ Identificar campos erróneos o vacíos.
- ▶ Convertir los datos CSV en un dataframe de Pandas, que son estructuras de datos especialmente diseñadas para aplicar técnicas de ciencia de datos, siendo posible su uso como tablas o series temporales.
- ▶ Emplear la opción `chunksize` para cargar los datos en fragmentos de tamaño configurable en lugar de cargar todos los datos en un único espacio de memoria. De este modo, se mejora la eficiencia.

```
import pandas
filename = 'diabetes.csv'
data = pandas.read_csv(filename, header=0)

print(data.shape)
print (data.head(10))
```

Resultando a la salida:

(768, 9)

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \

Ideas clave

0	6	148	72	35	0	33.6
1	1	85	66	29	0	26.6
2	8	183	64	0	0	23.3
3	1	89	66	23	94	28.1
4	0	137	40	35	168	43.1
5	5	116	74	0	0	25.6
6	3	78	50	32	88	31.0
7	10	115	0	0	0	35.3
8	2	197	70	45	543	30.5
9	8	125	96	0	0	0.0

DiabetesPedigreeFunction Age Outcome

0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1
5	0.201	30	0
6	0.248	26	1
7	0.134	29	0

8		0.158	53		1
9		0.232	54		1

Importación desde una URL

También podemos cargar directamente los datos desde una URL usando Pandas:

```
import pandas
url = "https://www.openml.org/data/get_csv/37/dataset_37_diabetes.arff"
#
data = pandas.read_csv(url)
print(data)

# En Pandas, el modificador tail nos ofrece una vista de los datos mucho más atractiva
# Ver final
data.tail()
```

	preg	plas	pres	...	pedi	age	class
0	6	148	72	...	0.627	50	tested_positive
1	1	85	66	...	0.351	31	tested_negative
2	8	183	64	...	0.672	32	tested_positive
3	1	89	66	...	0.167	21	tested_negative
4	0	137	40	...	2.288	33	tested_positive
..
763	10	101	76	...	0.171	63	tested_negative
764	2	122	70	...	0.340	27	tested_negative
765	5	121	72	...	0.245	30	tested_negative
766	1	126	60	...	0.349	47	tested_positive
767	1	93	70	...	0.315	23	tested_negative

[768 rows x 9 columns]

En este caso, estamos importando los datos del portal OpenML, que, entre otros recursos, incluye más de 20.000 *datasets* abiertos para trabajar con ellos.

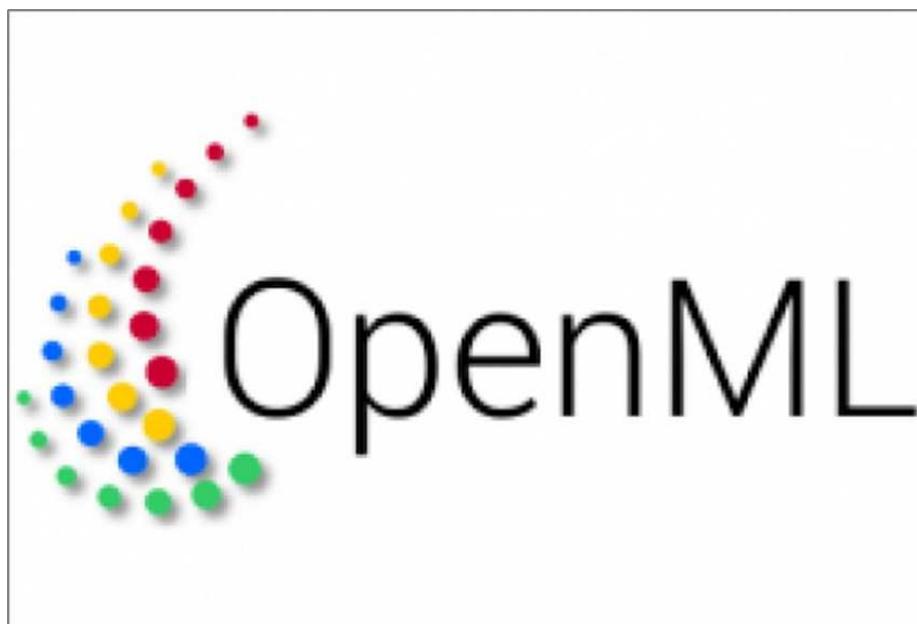


Figura 12. Fuente: <https://www.openml.org/>

Podemos ver los detalles del *dataset* sobre diabetes del ejemplo empleado (o bien buscar otros diferentes utilizando el motor de búsqueda) en:
<https://www.openml.org/d/37>.

Otro portal con más de 30.000 *datasets* para trabajar con ellos es Kaggle, además de incluir otros recursos *machine learning*:



Figura 13. Fuente: <https://www.kaggle.com/datasets>

Importación desde otras librerías

(*toy datasets*)

Como ya hemos visto con anterioridad, existen *datasets* que vienen incluidos «de serie» en las librerías de *machine learning*. Lo hemos visto en los ejemplos de scikit-learn, pero también los hay incluidos en paquetes como statsmodels o seaborn. Este tipo de conjuntos de datos para hacer pruebas se conocen como «*toy datasets*».

2.9. Introducción a Machine Learning con librerías en Python

Medias, medianas, modas, desviación estándar, varianza y percentiles

Como hemos visto con anterioridad, podemos dividir los tipos de datos de nuestro *dataset* en:

- ▶ Numéricos:
 - Discretos (el número de hijos por mujer).
 - Continuos (la longitud del pétalo de una flor).
- ▶ Categóricos (perro, gato, loro).
- ▶ Ordinales (suspenso, aprobado, notable, sobresaliente).

NumPy y SciPy nos ofrecen métodos rápidos para calcular diferentes medidas de un conjunto de datos, como la media, la mediana, la moda, la desviación estándar, la varianza y determinados percentiles, por ejemplo, el percentil 75:

```
import numpy
from scipy import stats

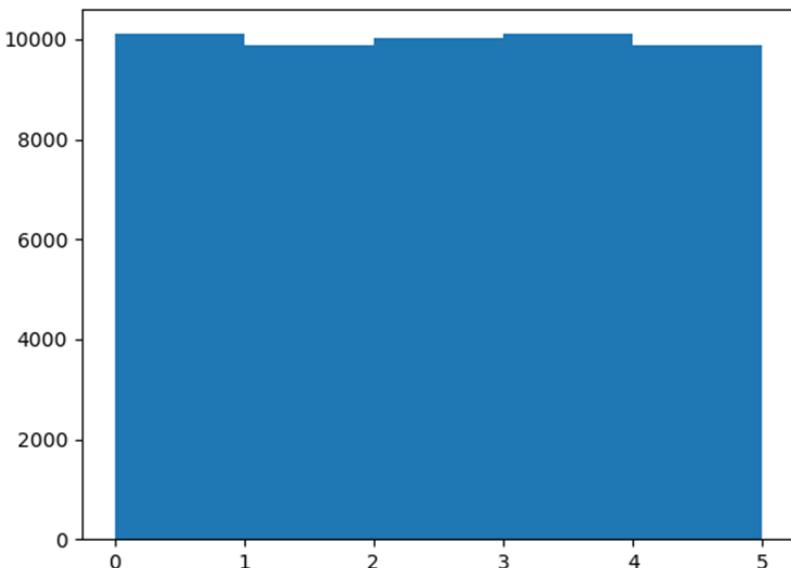
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

print(numpy.mean(speed))
print(numpy.median(speed))
print(stats.mode(speed))
print(numpy.std(speed))
print(numpy.var(speed))
print(numpy.percentile(speed, 75))
```

Distribución de los datos

Podemos generar una gran cantidad de datos aleatorios con NumPy y mostrar el histograma representando la distribución de los datos usando Matplotlib.

```
import numpy  
import matplotlib.pyplot as plt  
  
x = numpy.random.uniform(0.0, 5.0, 50000)  
  
plt.hist(x, 5)  
plt.show()
```

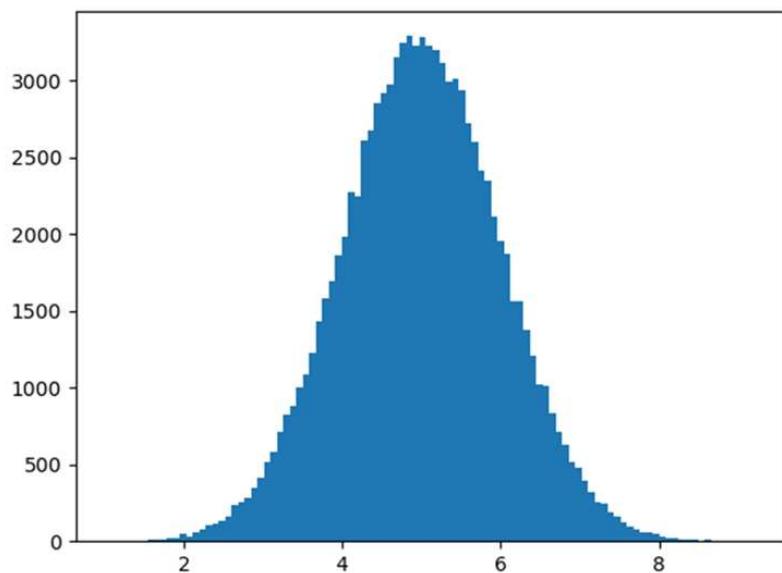


Podemos generar también datos que sigan una distribución Gaussiana o normal:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 100000)

plt.hist(x, 100)
plt.show()
```

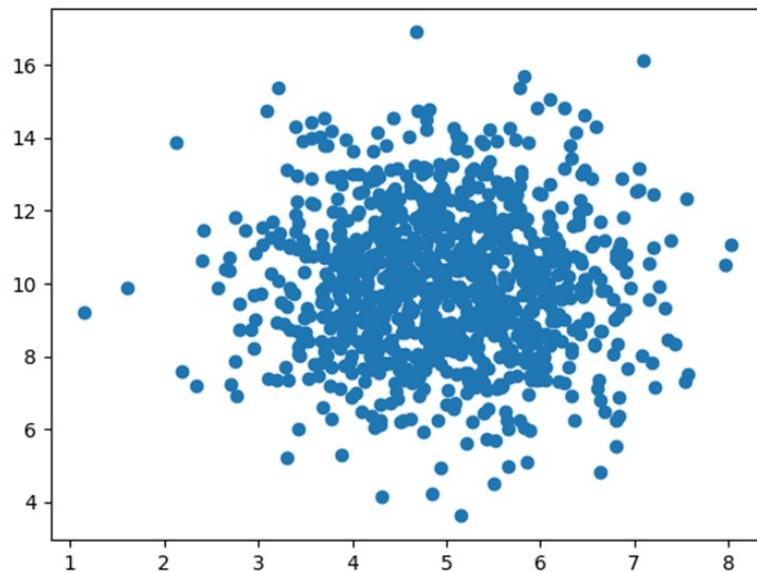


Podemos generar también un *dataset* bidimensional con dos variables independientes que sigan una distribución Gaussiana y mostrar un diagrama de dispersión (*scatter plot*).

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```



Regresiones

Una forma clásica de aprendizaje supervisado cuando se tienen datos continuos es la regresión, que puede ser lineal o polinómica, además de otros muchos más tipos más complejos.

Podemos usar SciPy para llevar a cabo modelos de regresión lineal:

```
import matplotlib.pyplot as plt
from scipy import stats

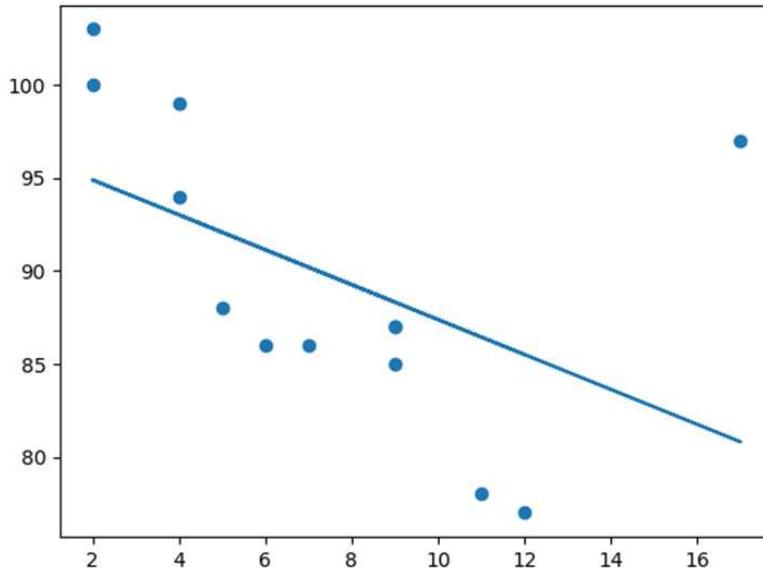
x = [4, 7, 9, 5, 2, 17, 2, 9, 4, 11, 12, 9, 6]
y = [99, 86, 87, 88, 100, 97, 103, 87, 94, 78, 77, 85, 86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
    return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```



Así como modelos de regresión polinómica mediante NumPy:

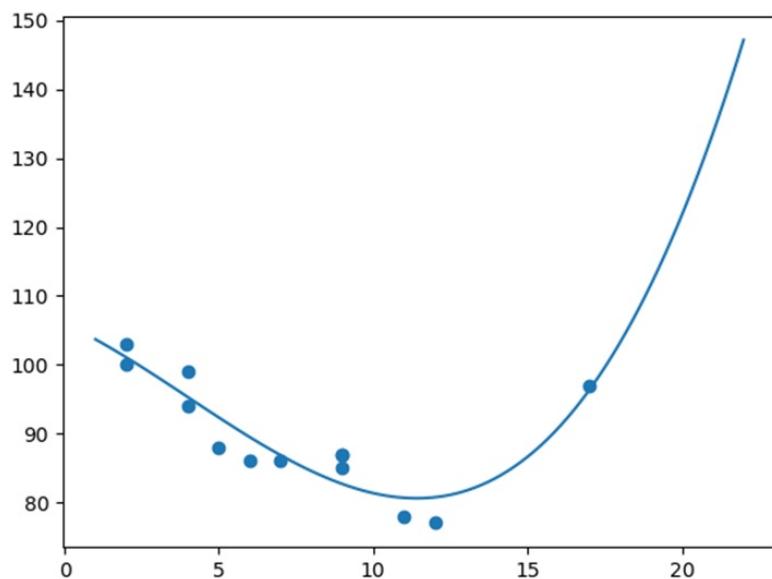
```
import numpy
import matplotlib.pyplot as plt

x = [4,7,9,5,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,100,97,103,87,94,78,77,85,86]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```



También es posible llevar a cabo una regresión lineal múltiple usando Pandas y Scikit-learn. Podemos usar un *dataset* en una URL, calcular el coeficiente de regresión de la regresión, así como realizar una predicción concreta.

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("https://www.openml.org/data/get_csv/52659/no2.arff")

X = df[['cars_per_hour', 'temperature_at_2m']]
y = df['no2_concentration']

regr = linear_model.LinearRegression()
regr.fit(X, y)
print(regr.coef_)
#> [ 0.39296818 -0.03256069]

# tratamos de predecir:
# concentración de NO2 en un punto en el que
# pasan 7 coches a la hora
# la temperatura a 2m de altura es de 5 grados
predictedNO2 = regr.predict([[7, 5]])
print(predictedNO2)
#> [3.57363206]
```

2.10. Referencias bibliográficas

Bansal, H. (2019, octubre 18). Best Languages For Machine Learning in 2020! Medium. Disponible en <https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd242>

Best Python libraries for Machine Learning. (2019, enero 18). GeeksforGeeks. Disponible en <https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/>

Puget, J.F. (2016) The Most Popular Language For Machine Learning and Data Science Is... KDnuggets. Recuperado de <https://www.kdnuggets.com/the-most-popular-language-for-machine-learning-and-data-science-is.html/>

Raschka, S. & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd.

Recuero de los Santos, P. (2019, octubre 1). *Python para todos: 5 formas de cargar datos para tus proyectos de Machine Learning - Think Big Empresas*. Think Big. Disponible en <https://empresas.blogthinkbig.com/python-5-formas-de-cargar-datos-csv-proyectos-machine-learning/>

Rogers, S. & Girolami, M. (2016). *A first course in machine learning*. Chapman and Hall/CRC.

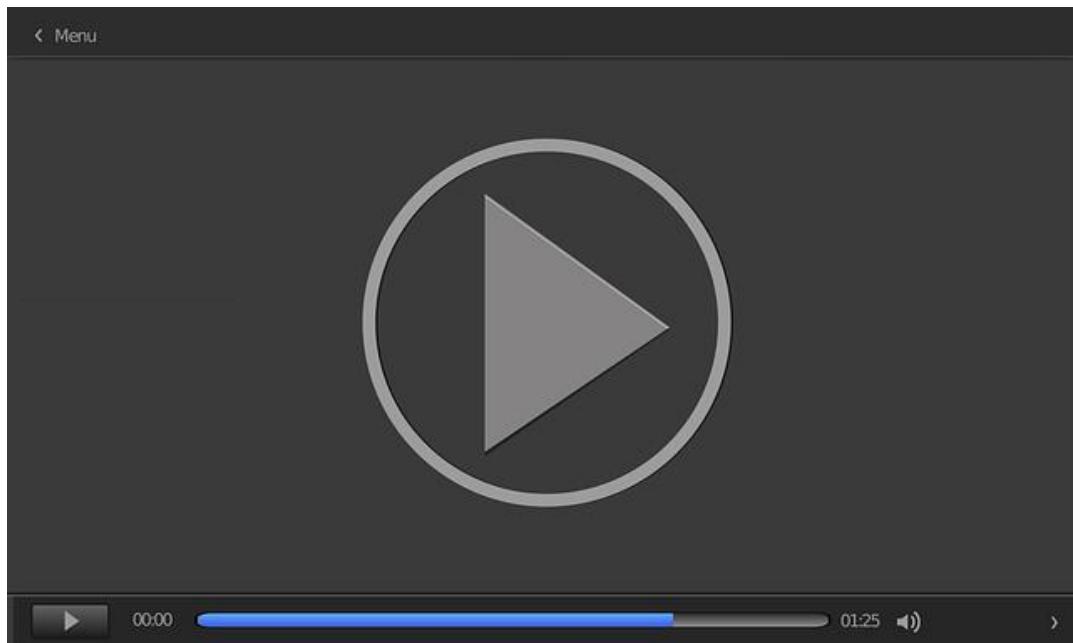
Russell, S. & Norvig, P. (2016). *Artificial Intelligence: A Modern Approach Global Edition* (Third Edition). Pearson.

Unipython (2019). Importar datos con Python. Disponible en <https://unipython.com/importar-datos-con-python/>

Gráficos en Python con Matplotlib, Seaborn y Plotly

<https://www.youtube.com/watch?v=LDojmNnxzto>

Daniel Chávez nos muestra múltiples formas de realizar gráficos desde un entorno Python usando Matplotlib, Seaborn y Plotly.



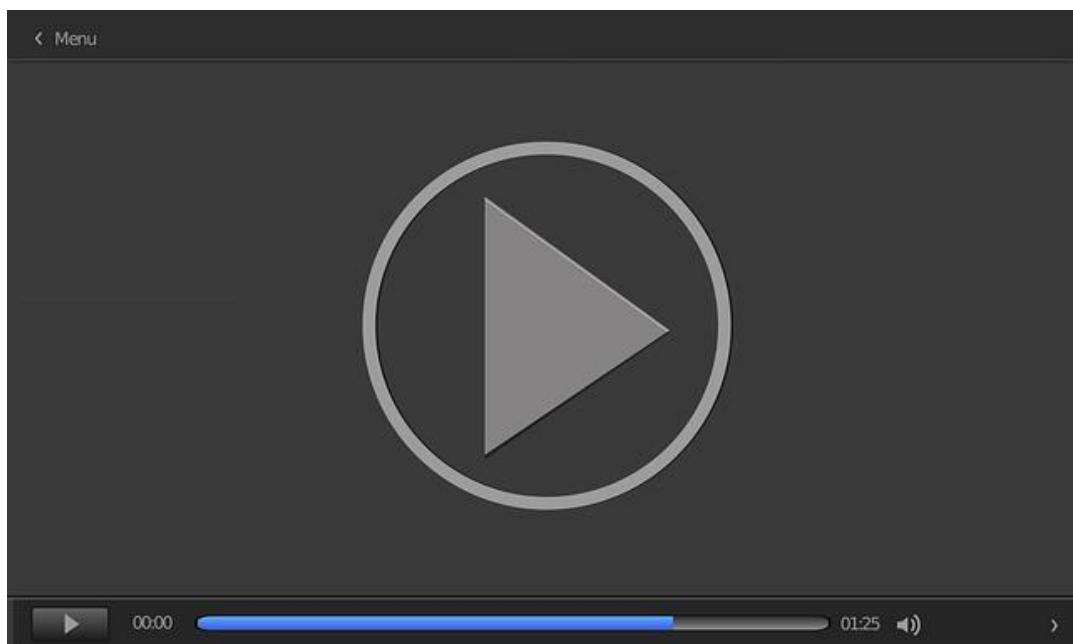
Accede al vídeo:

<https://www.youtube.com/embed/LDojmNnxzto>

TensorFlow 2.0 Crash Course

<https://www.youtube.com/watch?v=6g4O5UOH304>

El equipo de freeCodeCamp.org cuenta con varias sesiones prácticas para aprender de forma intensiva a manejarnos con TensorFlow 2.0. En esta sesión introductoria de más de 2 horas de duración, describen qué son las redes neuronales de forma práctica, y cómo emplear TensorFlow 2.0 para cargar datos, explorarlos, crear un modelo a partir de datos de imágenes de ropa (ejemplo que veremos en profundidad en el Tema 6), emplearlo para realizar predicciones, así como aplicar TensorFlow 2.0 para la clasificación de textos. Finaliza con la instalación de TensorFlow GPU bajo Linux.



Accede al vídeo:

<https://www.youtube.com/embed/6g4O5UOH304>

Tutorial de programación en Python

<https://www.w3schools.com/>

W3Schools es un portal web inicialmente creado para aprender tecnologías web en línea (HTML, CSS, JavaScript, SQL, PHP, C#, Java, etc.) lanzado en 1999 por la empresa noruega Refsnes Data. Actualmente cuenta también con un tutorial para aprender a programar en Python ideal para principiantes, incluyendo tutorial de NumPy y del uso de Python como herramienta para aplicar técnicas machine learning.

Udacity

<https://www.udacity.com/>

Udacity es una organización educativa con ánimo de lucro fundada por Sebastian Thrun (h=149 en Google Scholar, en su momento profesor de Inteligencia Artificial en Stanford y cofundador, también, de la entidad de investigación X Development, anteriormente conocida como Google X), David Stavens y Mike Sokolsky en 2011 que ofrece cursos *online* masivos y abiertos (MOOCs) mediante vídeos de YouTube interactivos. Dispone de diferentes cursos gratuitos accesibles mediante cuenta en Google o Facebook, además de *nano-degrees* de pago. En 2012 se hizo muy popular a partir del curso gratuito «Introduction to Artificial Intelligence», impartido por Sebastian Thrun y Peter Norvig, Director de I+D en Google y coautor con Stuart Russell del célebre «Artificial Intelligence: A Modern Approach».

Incluyendo el mencionado curso, otros cursos gratuitos interesantes para ampliar los conocimientos de esta asignatura son los siguientes:

- ▶ Intro to Python programming (Juno Lee).
- ▶ Intro to Artificial Intelligence (Peter Norvig y Sebastian Thrun).
- ▶ Intro to Machine Learning (Katie Malone y Sebastian Thrun).
- ▶ Intro to TensorFlow for Deep Learning (TensorFlow, Google).
- ▶ Intro to Deep Learning with PyTorch (Facebook Artificial Intelligence).
- ▶ Intro to TensorFlow Lite (TensorFlow Lite).

Coursera

<https://www.coursera.org/>

Coursera es otra plataforma de educación virtual también basada en la filosofía MOOC y también originada y desarrollada por académicos de la Universidad de Stanford, entre ellos Andrew Ng (h=122 en Google Scholar, investigador en inteligencia artificial, machine learning y deep learning). Fue lanzada en 2011 con los cursos gratuitos de *machine learning* e introducción a bases de datos. Existen cursos gratuitos y otros cursos en los que es necesario pagar por obtener el certificado tras realizar pruebas *online*, pero en los que de forma gratuita se puede acceder a parte o la totalidad de los contenidos.

Entre los cursos con contenidos totalmente gratuitos (a excepción del coste del certificado) interesantes para ampliar los conocimientos de esta asignatura son los siguientes:

- ▶ Learn to Program: The Fundamentals (Universidad de Toronto), curso enfocado al aprendizaje de programación mediante Python.
- ▶ *Machine Learning* (Andrew Ng, Universidad de Stanford).

Unipython

<https://unipython.com/>

Otro portal con cursos (algunos con certificado gratuito) de programación en Python (además de otros lenguajes), incluyendo inteligencia artificial o *machine learning*, además de otros campos (web, videojuegos, etc.).

1. Indica la respuesta correcta:

- A. Python es un lenguaje fuertemente tipado y de tipado estático.
- B. Python es un lenguaje de programación declarativo.
- C. Python es un lenguaje de programación imperativo, fuertemente tipado, de tipado dinámico.
- D. Python es un lenguaje de programación multiparadigma y débilmente tipado.

2. Indica el resultado por pantalla de la siguiente sentencia en Python: `print(int(2 + 2j))`

- A. 2.
- B. 2 + 2j.
- C. 0.
- D. Resultará un error.

3. Si en Python la variable `x` es una cadena str cuyo contenido es "Bienvenidos", ¿cuál será el resultado de la siguiente sentencia? `print(x[-4:-2])`

- A. [-4 -3 -2].
- B. Error.
- C. id.
- D. -6.

- 4.** Indica la respuesta incorrecta. Cuando hablamos de Python:
- A. Las listas son colecciones ordenadas y cuyos elementos pueden ser modificados.
 - B. Las tuplas son colecciones no ordenadas y cuyos elementos no pueden ser modificados.
 - C. Los conjuntos son colecciones no ordenadas y cuyos elementos no pueden estar duplicados.
 - D. Los diccionarios son colecciones no ordenadas de elementos clave-valor que pueden ser modificados.
- 5.** Si en un programa en Python existe una variable *x* de tipo tupla y realizamos la asignación *y = x*, señala la opción correcta:
- A. Al modificar un elemento de *y*, el mismo elemento en *x* se verá modificado.
 - B. Al modificar un elemento de *y*, el mismo elemento en *x* no se verá modificado.
 - C. No podemos modificar ningún elemento de *y*, por ser una tupla.
 - D. Ninguna de las anteriores es correcta.
- 6.** Si en un programa en Python existe una variable *x* de tipo lista y realizamos la asignación *y = x*, señala la opción correcta:
- A. Al modificar un elemento de *y*, el mismo elemento en *x* se verá modificado.
 - B. Al modificar un elemento de *y*, el mismo elemento en *x* no se verá modificado.
 - C. No podemos modificar ningún elemento de *y*, por ser una lista.
 - D. Ninguna de las anteriores es correcta.

- 7.** ¿Cuál de las siguientes preguntas es correcta?
- A. TensorFlow fue desarrollada por Google Brain.
 - B. Pandas permite la carga sencilla de datasets en CSV.
 - C. Keras es una librería especialmente enfocada para trabajar con redes neuronales artificiales.
 - D. Todas las anteriores.
- 8.** ¿Con qué función se puede importar un dataset en formato CSV desde un archivo local o una URL?
- A. pandas.importdataset().
 - B. pandas.readcsv().
 - C. pandas.readurl().
 - D. pandas.importcsv().
- 9.** ¿Cuál es el objetivo de TensorFlow Lite?
- A. Para el uso de librerías gratuitas.
 - B. TensorFlow Lite no existe.
 - C. Para su uso con el lenguaje Go.
 - D. Para su uso en dispositivos móviles con reducida capacidad de cálculo.
- 10.** Para mostrar gráficos por pantalla aplicando técnicas machine learning con Python, emplearía:
- A. Keras.
 - B. Scikit-learn.
 - C. Matplotlib.
 - D. SciPy.

Técnicas de Inteligencia Artificial

Tema 3. Árboles de decisión

Índice

.....

Esquema

Ideas clave

- 3.1. ¿Cómo estudiar este tema?
- 3.2. Introducción. Representación del conocimiento mediante árboles de decisión
- 3.3. Descripción de la tarea de inducción
- 3.4. Algoritmo básico de aprendizaje de árboles de decisión: ID3
- 3.5. Espacio de búsqueda y bias inductivo
- 3.6. Métodos de selección de atributos
- 3.7. Sobreajuste y poda de árboles
- 3.8. Medidas de la precisión de la clasificación. Curva ROC
- 3.9. Simplificación de árboles de decisión mediante poda: algoritmo C4.5
- 3.10. Ensemble Learning y Random Forest
- 3.11. Aplicaciones y ejemplos de implementación
- 3.12. Referencias bibliográficas

A fondo

Introducción a la herramienta Weka

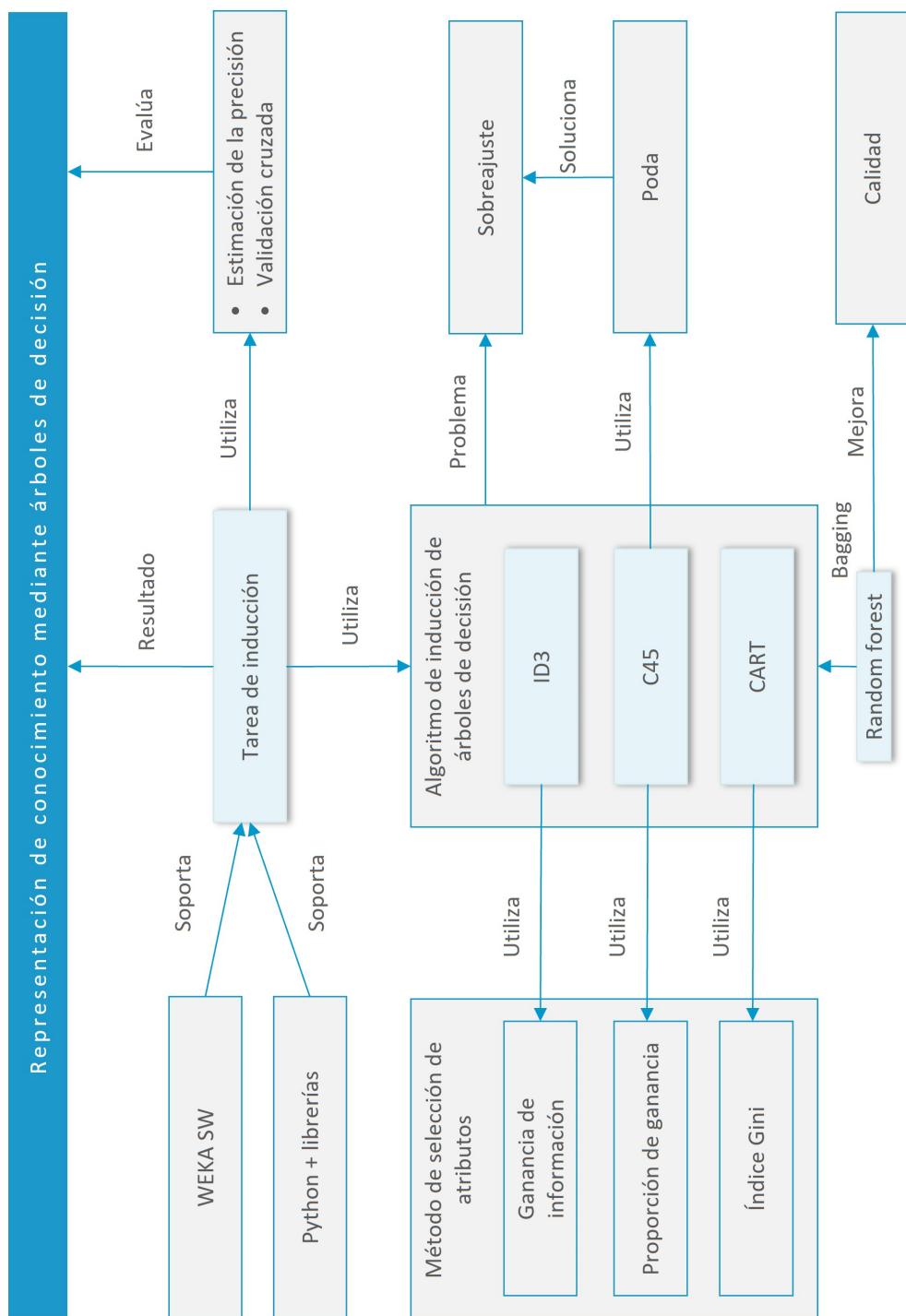
Inducción de árboles de decisión mediante el algoritmo ID3

Software de minería de datos Weka

Weka 3: Software de minería de datos en Java

Wiki de documentación sobre Weka

Test



3.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral. Revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan.

Al finalizar el estudio de este tema el alumno será capaz de:

- ▶ Implementar árboles de decisiones para representación de conocimiento.
- ▶ Aplicar los algoritmos ID3, C4.5 y Random Forest para resolver problemas de aprendizaje.
- ▶ Describir el problema de sobreajuste.
- ▶ Resolver un problema de sobreajuste aplicando poda.
- ▶ Identificar aplicaciones prácticas de la técnica de árboles de decisión.
- ▶ Aplicar algoritmos de árboles de decisión utilizando librerías bajo Python.
- ▶ Utilizar las funciones básicas de Weka.

3.2. Introducción. Representación del conocimiento mediante árboles de decisión

El aprendizaje de árboles de decisión es una de las técnicas más utilizadas para el aprendizaje inductivo, en concreto como técnica de aprendizaje supervisado, el cual es bastante robusto frente a datos ruidosos. Las entradas y salidas de la función objetivo suelen ser valores discretos, aunque en el caso de las entradas también podrían ser valores continuos. La representación de esta función objetivo toma forma de árbol y es interpretada como una serie de condiciones consecutivas que puede ser fácilmente mapeada a reglas.

Mediante un árbol de decisión será posible clasificar instancias cuya clase sea desconocida. Para llevar a cabo esta tarea de clasificación no habrá mas que comparar los atributos de dicha instancia con las ramas del árbol de decisión, de forma que se recorra de raíz a la hoja correspondiente el camino que no vaya marcando el valor de los atributos.

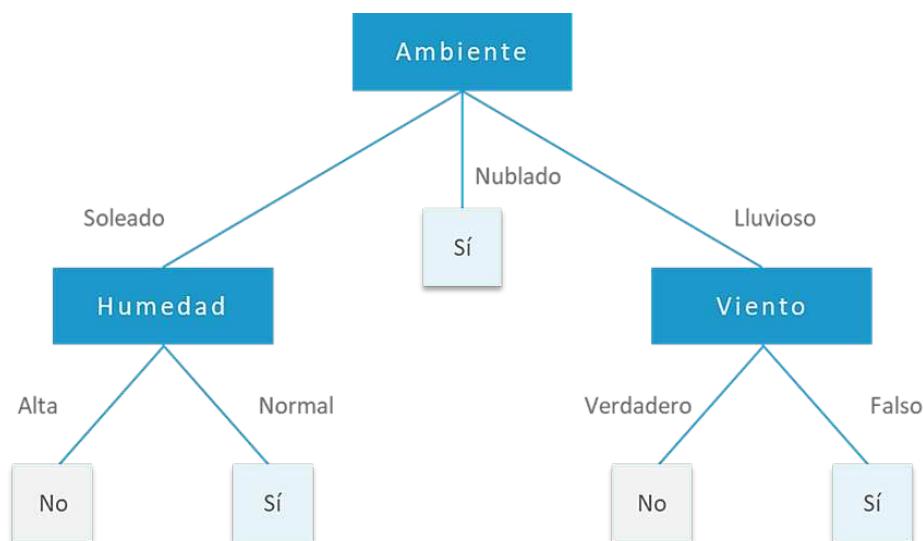


Figura 1. Ejemplo de árbol de decisión para el aprendizaje del concepto «Jugar al aire libre».

La Figura 1 muestra un ejemplo clásico de árbol de decisión. Este problema lo utiliza J.R. Quinlan para ilustrar el algoritmo *ID3* (Quinlan, 1986), que se explica en un apartado posterior, siendo un ejemplo que se puede encontrar adaptado en distintos libros que explican la técnica de árboles de decisión.

Con este árbol se pretende clasificar instancias con atributos relativos a diversos factores del tiempo ambiental con el fin de decidir si se juega al aire libre o no. Por ejemplo, si un sábado amanece lluvioso y con viento (condiciones representadas en las ramas de la derecha) el árbol indica que no es un día adecuado para jugar al aire libre.

¿Cuándo se considera adecuado el aprendizaje mediante árboles de decisión?

En general se siguen los siguientes criterios para responder a esta cuestión (Mitchell, 1997):

- ▶ Las instancias se representan por un conjunto de atributos y sus valores (en el ejemplo anterior, el atributo humedad puede tener el valor alta o normal). Estos valores pueden ser nominales, pero también se pueden resolver problemas con atributos de valores numéricos, mediante la aplicación de los algoritmos adecuados.
- ▶ La función objetivo tiene valores de salida discretos. En el ejemplo se asigna el valor sí o no .
- ▶ Se requieren descripciones disyuntivas. Los árboles de decisión son adecuados para la representación de este tipo de disyunciones. Cada rama desde la raíz a la hoja representa una conjunción lógica (operador AND), mientras que el árbol completo es una disyunción de conjunciones (operador OR). Por ejemplo, la clase sí del ejemplo de la Figura 1 queda representada por tres ramas del árbol que se pueden mapear a la siguiente regla, que es una disyunción de conjunciones que restringen los valores de los atributos de una instancia que pertenece a la clase sí:

SI ambiente es soleado AND humedad es normal

OR ambiente es lluvioso AND viento es falso

OR ambiente es nublado

ENTONCES jugar = sí

- ▶ Los datos de entrenamiento contienen errores o valores de atributos desconocidos.
Los árboles de decisión son robustos frente a errores tanto en la asignación de la clase de una instancia o clasificación de un ejemplo, como frente a si existen valores de los atributos de un ejemplo desconocidos o con errores.

Los árboles de decisión presentan entre otras las siguientes ventajas:

- ▶ Son fáciles de comprender y traducir a reglas.
- ▶ Pueden trabajar con conjuntos de datos tanto numéricos como nominales.
- ▶ Pueden trabajar con datos multidimensionales.
- ▶ No requieren conocimiento en un dominio dado ni establecer parámetros.

Sin embargo, existen algunas restricciones:

- ▶ Los atributos de salida deben ser categorías.
- ▶ No se permiten múltiples atributos de salida. No obstante, se puede emplear un árbol de decisión para cada atributo de salida.
- ▶ Los árboles construidos a partir de datos numéricos pueden resultar muy complejos

Los árboles de decisión han sido aplicados con éxito a problemas del mundo real, resultando **muy adecuados para resolver problemas de clasificación**, esto es, cuando la tarea consiste en clasificar ejemplos dentro de un conjunto discreto de posibles categorías.

Por tanto, se pueden utilizar, por ejemplo, para el diagnóstico de enfermedades, el diagnóstico de fallos de sistemas, la clasificación de clientes con el fin de aplicar estrategias de *marketing* o de detectar si hay riesgo en conceder un préstamo.

3.3. Descripción de la tarea de inducción

La definición de [inducción](#) nos indica que dicha tarea consiste en extraer el conocimiento general implícito a partir de observaciones y experiencias particulares.

En el aprendizaje de árboles de decisión, el espacio de hipótesis es el conjunto de todos los árboles de decisión posibles. La tarea de inducción del árbol de decisión consiste en encontrar el árbol que mejor encaje con los datos de ejemplo, ya clasificados, disponibles. Para cada clase, se ha de encontrar una rama en el árbol que satisfaga la conjunción de valores de atributos representada por la rama.

Cuando se está generando un árbol de decisión un elemento crucial es el **método de selección de atributos**, que determina qué criterio se utiliza para generar las diferentes ramas del árbol, que van determinando la clasificación en las diferentes clases.

El criterio dependerá del tipo de dato que sea el atributo. Por ejemplo, si el tipo de dato del atributo es discreto, se crea una rama para cada valor conocido del atributo. Sin embargo, si el tipo de dato es numérico, hay que establecer algún punto de separación en las ramas como, por ejemplo, si el valor del atributo es mayor o menor a un determinado valor (ver ejemplos en la Figura 2).

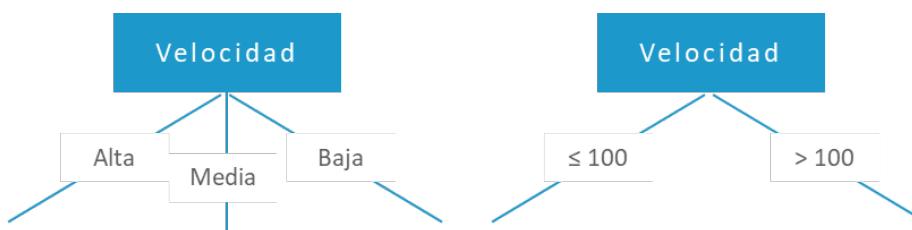


Figura 2. Creación de ramas del árbol en función del tipo de atributo.

Para describir el aprendizaje de árboles de decisión se continúa con el popular problema sobre el tiempo. El conjunto de datos de entrenamiento se muestra en la Tabla 1. El ejemplo tiene 14 instancias de entrada con cuatro atributos de entrada y

un atributo de salida (el concepto a aprender). En el artículo de Quinlan simplemente se indican dos valores P y N, siglas de positivo y negativo, para el atributo de salida (Quinlan, 1986).

En el ejemplo que se presenta se utiliza una versión adaptada presente en varias referencias, en los que el problema consiste en clasificar las mañanas de los sábados, en función de tres factores ambientales, según sean adecuadas o no para jugar al aire libre (por ejemplo, un partido de tenis). Se mantienen los atributos de entrada presentes en el artículo de Quinlan con los siguientes valores (Quinlan, 1986).

- ▶ Ambiente: soleado, nublado, lluvioso.
- ▶ Temperatura: alta, media, baja.
- ▶ Humedad: alta, normal.
- ▶ Viento: verdadero, falso.

El atributo de salida, la clase, puede tomar también dos valores que se utilizan para clasificar las mañanas de los sábados de acuerdo a si se juega o no al aire libre:

- ▶ Clase: sí, no.

Id	Ambiente	Temperatura	Humedad	Viento	Clase
E ₁	Soleado	Alta	Alta	Falso	No
E ₂	Soleado	Alta	Alta	Verdadero	No
E ₃	Nublado	Alta	Alta	Falso	Sí
E ₄	Lluvioso	Media	Alta	Falso	Sí
E ₅	Lluvioso	Baja	Normal	Falso	Sí
E ₆	Lluvioso	Baja	Normal	Verdadero	No
E ₇	Nublado	Baja	Normal	Verdadero	Sí
E ₈	Soleado	Media	Alta	Falso	No
E ₉	Soleado	Baja	Normal	Falso	Sí
E ₁₀	Lluvioso	Media	Normal	Falso	Sí
E ₁₁	Soleado	Media	Normal	Verdadero	Sí
E ₁₂	Nublado	Media	Alta	Verdadero	Sí
E ₁₃	Nublado	Alta	Normal	Falso	Sí
E ₁₄	Lluvioso	Media	Alta	Verdadero	no

Tabla 1. Instancias con sus valores de atributos y clases del problema «Jugar al aire libre».

Un algoritmo básico para construir un árbol de decisión es sencillo.

En la Figura 3 se muestra este algoritmo para el caso en que los atributos de los datos de entrenamiento son nominales. Los parámetros para generar el árbol de decisión son los siguientes:

- ▶ Los datos de entrenamiento E, las instancias ejemplo clasificadas en clases.
- ▶ La lista de atributos de las instancias que van a ser candidatas para determinar la decisión.
- ▶ El método de selección de los atributos. Especifica una heurística para seleccionar el atributo que mejor discrimina los ejemplos para una clase.

```

PROCEDIMIENTO      Inducir_Arbol      (Ejemplos      E,      Lista_Atributos,
Método_Selección_Atributos)
COMIENZO
P1    Crear un nodo N;
P2    SI todos los elementos de E pertenecen a la misma clase, C
      ENTONCES retornar N como nodo hoja etiquetado con la clase C,
P3    SINO SI la lista de atributos (Lista_Atributos) está vacía
      ENTONCES retornar N como nodo hoja etiquetado con la clase más numerosa
      en los ejemplos
P4    SINO aplicar Método_Selección_Atributos(E, Lista_Atributos) para
      seleccionar el atributo A que mejor particiona E
P5          Borrar Atributo A de la lista de Atributos
P6    Lista_Atributos
P7    Etiquetar N con el atributo seleccionado
          PARA CADA valor V de A
          Siendo  $E_v$  el subconjunto de elementos en E con valor V en el atributo A.
P8    SI  $E_v$  está vacío
      ENTONCES unir al nodo N una hoja etiquetada con la clase mayoritaria en
      E.
P9    SINO unir al nodo N el nodo retornado de Inducir_Arbol (Ev,
Lista_Atributos, Método_Selección_Atributos)
          FIN PARA CADA
FIN SI-SINO
FIN

```

Figura 3. Algoritmo básico de construcción de árboles de decisión.

En primer lugar, se crea un nodo raíz que representa a todos los ejemplos en E. Si todas las instancias en E son de la misma clase, el nodo se convierte en una hoja del árbol que es etiquetada con esa clase (P2). Si esto no es así y la lista de atributos no está vacía, se llama al procedimiento Método_Selección_Atributos (paso P4) para determinar el atributo que se ha de comprobar en el nodo N, que será el atributo que implica la mejor división de los ejemplos en clases. Crecerán tantas ramas del nodo N como posibles valores del atributo. Se trata de que los subconjuntos de ejemplos resultantes de la división sean lo más homogéneos posibles (la homogeneidad máxima se da cuando todos los ejemplos de cada subconjunto pertenecen a la misma clase).

En el paso P6 se etiqueta el nodo N con el atributo A seleccionado que servirá como condición para la clasificación. El nodo se ramifica con los diversos valores del atributo y se distribuyen los ejemplos en las diversas ramas de acuerdo al valor del atributo que cumplen.

En la Figura 4 se muestra un ejemplo del resultado de la primera iteración del algoritmo sobre los datos del problema del tiempo mostrados en la Tabla 1. El método de selección del atributo ha seleccionado ambiente como el mejor atributo para dividir los ejemplos teniendo en cuenta específicamente la ganancia de información que se consigue dividiendo los ejemplos con este atributo (en la siguiente sección se explicará este método de selección de atributos que utiliza el algoritmo ID3 basado en la medida de ganancia de información).

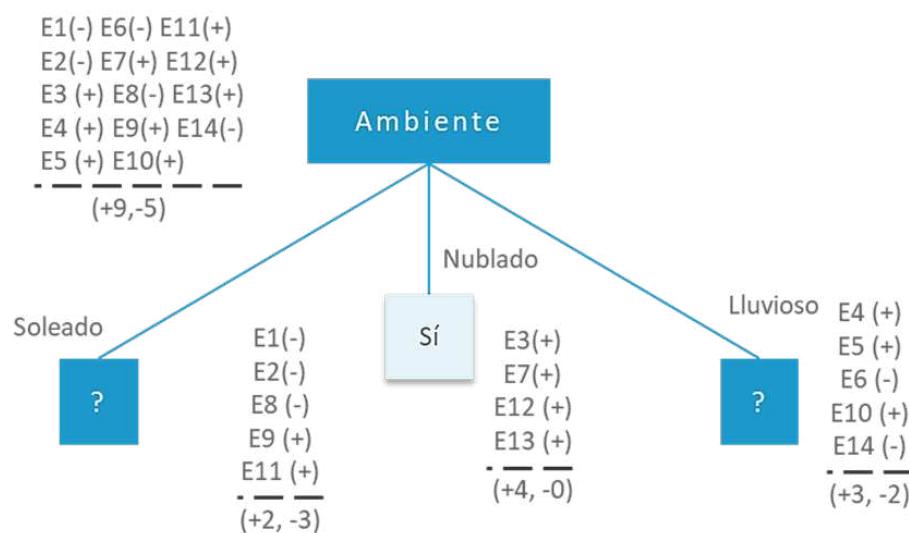


Figura 4. Árbol parcial generado en la primera iteración del algoritmo sobre los datos de ejemplo de la Tabla 1.

Una vez particionados los ejemplos, se vuelve a iniciar el procedimiento en cada nodo hijo, partiendo del subconjunto de ejemplos asignados a ese nodo hijo E_v y de la lista de atributos no utilizados previamente para particionar.

En el ejemplo concreto, para el nodo hijo generado que se une al nodo raíz con la rama nublado, se va a cumplir la condición del paso 2 (P2), es decir, todos los ejemplos en esa partición pertenecen a la clase sí , con lo cual ese nodo se convierte en un nodo hoja etiquetado con la clase sí .

Para los otros dos nodos hijos generados es necesario seleccionar de nuevo un atributo en base a los nuevos subconjuntos generados, en ambos casos compuestos por cinco ejemplos. Este proceso es iterativo hasta que se cumple una de las siguientes condiciones:

- ▶ Todos los ejemplos de E_v pertenecen a la misma clase, con lo cual el nodo se convierte en un nodo hoja (P2).
- ▶ No hay más atributos para particionar ejemplos (P3). En este caso se puede convertir el nodo en una hoja y etiquetarlo con la clase mayoritaria en los ejemplos de E_v . Esta decisión se denomina **votación mayoritaria**.
- ▶ Si no hay ejemplos para una rama, la partición E_v está vacía (P8), de modo que se crea un nodo hoja con la clase mayoritaria en E . Existen variaciones del algoritmo que toman otras decisiones como simplemente etiquetar el nodo hoja como desconocido .

El algoritmo básico aquí descrito es del tipo «**divide-y-vencerás**», construido sin retroceder en ningún caso para volver a reconsiderar una decisión tomada en un paso previo. Esto último relativo a siempre avanzar hacia adelante es denominado **método codicioso** (*greedy* en inglés).

En cada iteración de este algoritmo básico se aplica el método de selección de atributos en base a los atributos y ejemplos, y esto puede suponer un alto requisito de computación.

3.4. Algoritmo básico de aprendizaje de árboles de decisión: ID3

El algoritmo ID3 construye los árboles top-down (de arriba a abajo) utilizando un método de selección de atributos basado en la **teoría de la información**. ID3 considera como heurística que el atributo cuyo conocimiento aporta mayor información en la clasificación es el más útil.

El algoritmo ID3 sigue los pasos del algoritmo básico mostrado en la Figura 4. En primer lugar, el algoritmo, para cada atributo, realiza un cálculo para medir cuán bien ese atributo clasifica los ejemplos disponibles. El atributo mejor clasificador se convierte en la condición del nodo raíz que da lugar a distintas ramas, una por cada valor posible del atributo.

Los ejemplos se distribuyen en los nodos descendientes de acuerdo con su valor del atributo mejor clasificador. Este proceso que se ha aplicado al nodo raíz se repite para los nodos descendientes. Este algoritmo nunca da marcha atrás para reconsiderar decisiones previas.

Como previamente se ha comentado un aspecto importante es el método de selección de atributos, que determina el rendimiento del algoritmo. El algoritmo ID3 utiliza la ganancia de información para seleccionar en cada paso según se va generando el árbol aquel atributo que mejor distribuye los ejemplos de acuerdo a su clasificación objetivo.

¿Cómo se mide la mejor distribución de ejemplos o se selecciona aquel atributo cuyo conocimiento aporta mayor cantidad de información?

Para contestar a esta pregunta ID3 utiliza conceptos que forman parte de la teoría de la información. En concreto, como previamente se ha indicado, utiliza la ganancia de

información que a su vez mide la reducción esperada de entropía. Para comprender esto se va a explicar a continuación el concepto de entropía y, posteriormente, la medida de ganancia de información.

La entropía caracteriza la heterogeneidad de un conjunto de ejemplos. Cuando una clase C puede tomar n valores, la entropía del conjunto de ejemplos E respecto a la clase C se define como:

$$\text{Entropía}(E) = \sum_{i=1}^n -p_i \log_2 p_i (1)$$

Siendo p_i la proporción de ejemplos de E que pertenecen a la clase i .

Si todos los miembros del conjunto E pertenecen a la misma clase, la entropía es nula, es decir, tendrá un valor igual a 0. Sin embargo, esta será igual a 1 si se tiene un mismo número de ejemplos positivos y negativos (cuando se trata de una clasificación booleana), y tendrá un valor comprendido entre 0 y 1 cuando no es igual el número de ejemplos positivos y negativos.

Dado que la entropía mide la heterogeneidad de un conjunto de ejemplos, la ganancia de información utiliza la entropía para medir la efectividad de un atributo para clasificar ejemplos. Específicamente mide la reducción de entropía cuando se distribuyen los ejemplos de acuerdo a un atributo concreto.

Siendo un atributo A con V_a posibles valores y un conjunto de ejemplos E , la fórmula para calcular la ganancia de información viene dada por la expresión (2):

$$\text{Ganancia}(E, A) = \text{Entropía}(E) - \sum_{v \in V_a} \frac{|E_v|}{E} \text{Entropía}(E_v) (2)$$

Ev es el subconjunto de ejemplos para los que el atributo A toma el valor v dentro de los posibles valores de v (especificados en V_a).

Por tanto, en el ejemplo previo del artículo de Quinlan, si se utiliza la ganancia de información para seleccionar los atributos que particionan en cada paso el árbol, para el caso del nodo raíz se tiene:

$$\text{Entropía}(E) = \frac{-9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = +0,94$$

Ganancia (E, ambiente)

$$\begin{aligned} &= \text{Entropía}(E) - \frac{5}{14} \text{Entropía}(E_{soleado}) - \frac{4}{14} \text{Entropía}(E_{nublado}) \\ &- \frac{5}{14} \text{Entropía}(E_{lluvioso}) = +0,9402 - \frac{5}{14} 0,9709 - \frac{4}{14} 0 - \frac{5}{14} 0,9709 = 0,247 \end{aligned}$$

$$\begin{aligned} \text{Ganancia (E, humedad)} &= \text{Entropía}(E) - \frac{7}{14} \text{Entropía}(E_{alta}) - \frac{7}{14} \text{Entropía}(E_{normal}) \\ &= +0,9402 - \frac{7}{14} 0,985 - \frac{7}{14} 0,591 = 0,151 \end{aligned}$$

$$\begin{aligned} \text{Ganancia (E, viento)} &= \text{Entropía}(E) - \frac{8}{14} \text{Entropía}(E_{falso}) - \frac{6}{14} \text{Entropía}(E_{verdadero}) \\ &= +0,9402 - \frac{8}{14} 0,811 - \frac{6}{14} 1 = 0,048 \end{aligned}$$

Ganancia (E, temperatura)

$$\begin{aligned} &= \text{Entropía}(E) - \frac{4}{14} \text{Entropía}(E_{alta}) - \frac{6}{14} \text{Entropía}(E_{media}) - \frac{4}{14} \text{Entropía}(E_{baja}) \\ &= +0,9402 - \frac{4}{14} 1 - \frac{6}{14} 0,918 - \frac{4}{14} 0,811 = 0,029 \end{aligned}$$

De acuerdo con la medida de ganancia de información que aplica el algoritmo ID3 el atributo ambiente es el que proporciona mayor cantidad de información a la hora de predecir la clase. Por lo tanto, con este método de selección de atributos el ambiente es seleccionado como el atributo a comprobar en el nodo raíz y tres ramas se crean, correspondientes a los tres valores que este atributo toma.

Los ejemplos se dividen en tres grupos según su valor de este atributo y se llega por tanto al árbol parcial mostrado en la Figura 4. Dado que todos los ejemplos con ambiente igual a nublado pertenecen a la clase sí (entropía es 0), el nodo unido a través de la rama nublado se convierte en un nodo hoja.

Partiendo de los nodos hijos el procedimiento se repite hasta que o todos los ejemplos pertenecen a la misma clase, o todos los atributos están incluidos en un camino del árbol, o no quedan más ejemplos.

3.5. Espacio de búsqueda y bias inductivo

En el problema de construcción de árboles de decisión, el espacio de hipótesis o posibles soluciones es el conjunto de todos los posibles árboles de decisión. En concreto, el algoritmo ID3 busca por el espacio de hipótesis hasta encontrar el árbol que encaja con los ejemplos de entrenamiento.

ID3 realiza una búsqueda en escalada, guiada por la medida de ganancia de información, desde árboles más sencillos a árboles más complejos, buscando aquel que clasifica correctamente los datos de entrenamiento.

ID3 tiene la ventaja de que **trabaja en un espacio de hipótesis completo**, con lo cual no se da el caso de estar buscando en un espacio de hipótesis en que no se encuentra la solución. Sin embargo, ID3 no trabaja con varias posibles soluciones simultáneamente, aquellas que son consistentes con los ejemplos, sino que solo trabaja con una posible solución. En este caso se corre el riesgo de no construir el **árbol que representa la mejor solución**.

ID3 **no da marcha atrás** en su búsqueda para reconsiderar elecciones previas. Esto puede dar lugar al problema típico de métodos de búsqueda de escalada sin vuelta atrás, **el poder converger hacia una solución óptima local, que no es la mejor solución en global**.

ID3 tiene la ventaja de ser **robusto frente a errores**, dado que en cada paso que se da utiliza todos los ejemplos para hacer los cálculos de la ganancia de información. Sin embargo, esto supone **más carga computacional** que una solución incremental en la que en cada paso solo se tienen en cuenta parte de los ejemplos.

En esta búsqueda por el espacio de hipótesis, ¿en qué se basa ID3 para generalizar el árbol de decisión, esto es, considerar que el árbol clasificará correctamente instancias no utilizadas en la etapa de aprendizaje? En definitiva, entendemos el *bias* como los criterios de selección de una hipótesis en función de una serie de factores y supuestos.

De los posibles árboles que podrían servir para clasificar un conjunto de ejemplos, ID3 escoge como mejor árbol al primero que encuentra en una búsqueda en la que se va generando un árbol desde lo más sencillo a lo más complejo.

ID3, por tanto, funciona siguiendo estas dos premisas:

- ▶ Da preferencia a árboles cortos frente a los largos.
- ▶ Sitúa los atributos que proporcionan mayor ganancia de información más cerca de la raíz.

Con ID3 no se garantiza encontrar el árbol más corto, puesto que no considera en cada paso todos los árboles posibles que se valorarían en una búsqueda exhaustiva en amplitud. Sin embargo, ID3 realiza una búsqueda *greedy* sin retroceso, que pretende encontrar el árbol más corto que a su vez sitúa los atributos que mayor ganancia de información aportan en la zona del árbol más cerca de la raíz.

El *bias* inductivo de ID3 se puede, por tanto, considerar como una preferencia por árboles cortos frente a largos y se prefieren árboles que sitúan cerca de la raíz a los atributos que aportan mayor ganancia de información.

El preferir árboles cortos puede mejorar la **generalización**, puesto que hay hipótesis complejas que encajan muy bien con los datos de entrenamiento pero que no generalizan correctamente datos futuros.

3.6. Métodos de selección de atributos

Existen diversos métodos de selección de atributos empleados en diversos algoritmos de construcción de árboles de decisión como la **tasa de ganancia**, el **índice Gini** o la previamente explicada **ganancia de información** utilizada por el algoritmo ID3. El usar un método u otro dependerá tanto de los datos de entrenamiento disponibles, como del algoritmo específico que se emplee así como de los supuestos y suposiciones que se realizan para generalizar la mejor solución encontrada (*bias*).

Por ejemplo, el **índice Gini**, que **mide la impureza de los datos**, es típicamente utilizado en algoritmos CART (Classification and Regression Trees).

El índice Gini se calcula con la expresión (3):

$$Gini(E) = 1 - \sum_{i=1}^n p_i^2 \quad (3)$$

Siendo n el número de clases totales y siendo p_i el resultado de dividir el número de ejemplos que pertenecen a la clase C_i entre el número de ejemplos totales. El sumatorio se refiere a la suma que comprende dicho cociente para las n clases.

El atributo seleccionado para dividir los ejemplos en subconjuntos será aquel que maximice la reducción de impureza (el que tenga un índice *Gini* menor).

Otro método de selección de atributos se basa en la **proporción de ganancia**, utilizada por el algoritmo C4.5, que se explicará más adelante. Cuando se manejan atributos con muchos valores, la proporción de ganancia puede dar mejores resultados que la medida de ganancia de información, que favorece aquellos atributos con mayor número de valores.

Si, por ejemplo, se da el caso extremo de un atributo con un valor diferente para cada ejemplo, la medida de ganancia de información determinará que este atributo

sea escogido puesto que aporta la mayor información en la clasificación, ya que determina la clase sin ninguna ambigüedad. Sin embargo, realmente no es un clasificador útil para nuevas instancias.

La proporción de ganancia compensa el hecho de que un atributo pueda tener muchos valores dividiendo por la medida denominada información de la división, tal y como se indica en la fórmula (4).

$$\text{Información de la División}(E,A) = -\sum_{i=v_i}^{v_n} Ei \vee \frac{|Ei|}{E} \log_2 \frac{|Ei|}{|E|} \quad (4)$$

Siendo Ei , $Ei+1$, ..., En las diferentes particiones de ejemplos que resultan de dividir el conjunto E de ejemplos teniendo en cuenta los valores vi , $vi+1$, ..., vn que toma el atributo, respectivamente.

Así la proporción de ganancia se calcula como el cociente entre la ganancia de información y la información de la división, tal y como se muestra en la expresión (5).

$$\text{Proporción de Ganancia}(E,A) = \frac{\text{Ganancia de Información}(E,A)}{\text{Información de la División}(E,A)} \quad (5)$$

Dividiendo por la información de la división se consigue penalizar aquellos atributos con muchos valores que se distribuyen muy uniformemente entre los ejemplos. Si un atributo toma dos valores y distribuye los ejemplos en dos grupos de igual tamaño, la *Información de la División* toma el valor 1.

Sin embargo, si un atributo toma un gran número de valores y distribuye los ejemplos uniformemente en estos valores se obtiene un valor de *Información de la División* de $\log 2n$.

Otro método es denominado **Longitud de Descripción Mínima** o Minimum Description Length (MDL). Este método considera el mejor árbol de decisión aquel que requiere un menor número de bits para codificar el propio árbol o bien codificar los casos no clasificados por el árbol, seleccionando la opción más simple de estas.

Otros métodos consideran mejor particionar el árbol basándose en múltiples atributos y no solo en uno.

Realmente los métodos de selección indicados funcionan bien. Utilizan diferentes supuestos y suposiciones sobre la mejor solución encontrada (*bias*) pero no está claro qué método es mejor que otro. En general la complejidad de árbol aumentará conforme la profundidad sea mayor, pero, por otra parte, árboles más cortos pueden generar más errores en las decisiones.

3.7. Sobreajuste y poda de árboles

ID3 crea el árbol de decisión de tal manera que clasifica correctamente los datos de entrenamiento y, aunque parece contradictorio, clasificar demasiado bien no es siempre adecuado ya que los datos de entrenamiento pueden contener ruido o simplemente pueden no ser una muestra suficientemente numerosa de datos que permita generalizar.

El sobreajuste se produce cuando existe una hipótesis H del espacio de hipótesis que se ajusta mejor a los datos de entrenamiento que otra hipótesis H' del espacio de hipótesis, pero, al mismo tiempo, H' se ajusta mejor a todas las instancias (comprendiendo datos de entrenamiento e instancias futuras).

Construir hipótesis con demasiadas condiciones, demasiado complejas, reflejadas en árboles con demasiados nodos, a veces supone sobreajustar la hipótesis a los datos de entrenamiento.

Utilizando el ejemplo previo del problema del tiempo, si el Ejemplo 7 (E7) mostrado en la Tabla 1 tiene asignada la clase No erróneamente, esto daría lugar a la generación de un árbol distinto y más complejo, ya que, en el segundo paso del algoritmo no se generaría el nodo hoja, tal y como se muestra en la Figura 4 (todos los ejemplos con el atributo nublado pertenecían a la misma clase), sino que se seguiría ramificando, dando lugar a un árbol más complejo. Queda como ejercicio del alumno el ejecutar el algoritmo para comprobar qué árbol se generaría.

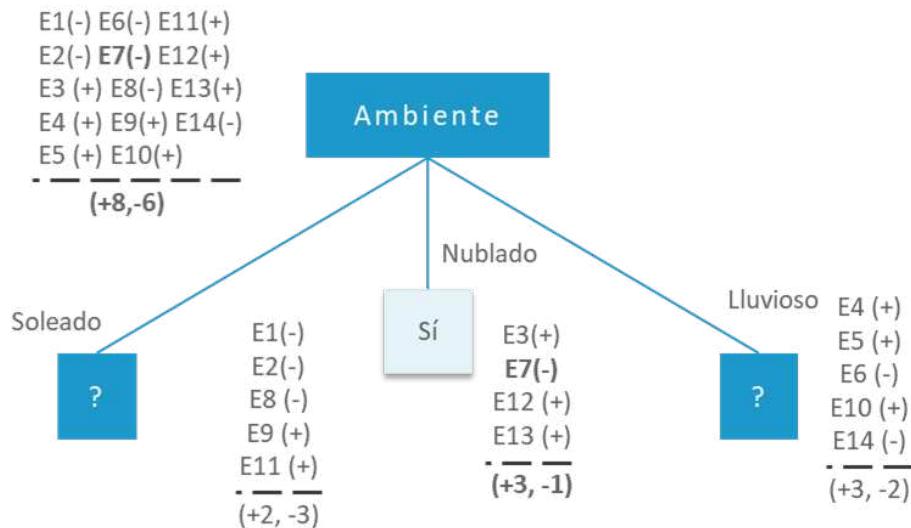
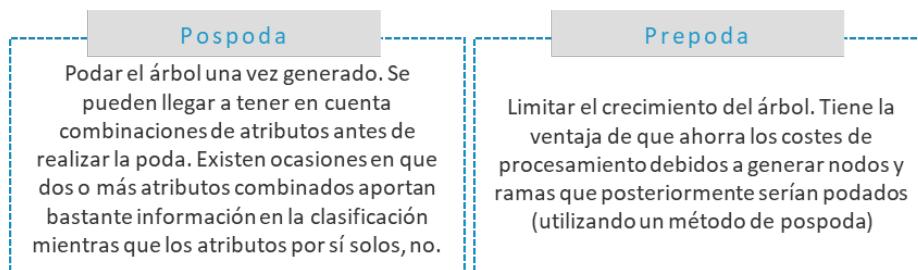


Figura 5. Árbol parcial generado en la primera iteración del algoritmo sobre los datos de ejemplo de la Tabla 1 (con la clase del Ejemplo 7 (E7) cambiada a valor no).

El sobreajuste es un problema práctico en la construcción de árboles de decisión y generalmente se ha de aplicar alguna estrategia para mitigar el problema.

Dos tipos de estrategias para evitar el sobreajuste son:



Si el árbol se mapea a reglas, se puede realizar una poda más eficaz, siguiendo los pasos que se indican a continuación:

- ▶ Generación del árbol de decisión a partir de los datos de entrenamiento.
- ▶ Mapeado del árbol a un conjunto de reglas. Cada camino desde la raíz hasta la hoja corresponde a una regla (una serie de condiciones enlazadas con el operador AND).

- ▶ Poda de cada regla eliminando las condiciones en el antecedente que suponen mejorar la precisión de la clasificación.
- ▶ Ordenación de las reglas en función de la precisión estimada y se clasifican los ejemplos futuros empleando las reglas en este orden.

Una diferencia entre podar el árbol directamente o trabajar en la poda de reglas es que cuando se elimina un nodo de un árbol se están eliminando diferentes reglas que resultan de la condición testeada en ese nodo, mientras que, si se trabaja con reglas, la decisión de poda se realiza de una manera aislada para cada camino que pasa por ese nodo. Además, si se trabaja con reglas en la poda es indiferente si el atributo testeado se encuentra cerca de la raíz o cerca de la hoja.

¿Cómo se puede saber cuál es el tamaño del árbol adecuado? Es difícil determinar cuándo se ha de parar el crecimiento del árbol o hasta cuánto se ha de podar.

Cuando se poda un nodo de un árbol, se eliminan las ramificaciones y se convierte a ese nodo en una hoja a la que se puede asignar la clase mayoritaria de los ejemplos vinculados a ese nodo. La decisión de podar típicamente se tomará en función de medidas estadísticas que permiten conocer cuáles son las ramas menos fiables.

Una práctica frecuente es utilizar la técnica de validación cruzada (cross-validation). La validación cruzada permite estimar el ajuste del modelo a un hipotético conjunto de datos de prueba cuando no se dispone de este conjunto de datos de prueba de manera explícita.

Consiste en dividir el conjunto de ejemplos disponibles en un conjunto de datos de entrenamiento y un conjunto de datos de validación:

- ▶ **Datos de entrenamiento:** se utilizan para generar el árbol.
- ▶ **Datos de validación:** se utilizan para validar la precisión del árbol generado sobre datos futuros.

La etapa de validación nos puede servir para evaluar la efectividad de la poda del árbol a la hora de clasificar instancias futuras. Se puede, por ejemplo, valorar si podar cada nodo del árbol en base a si el árbol podado resultante clasifica al conjunto de validación igual o mejor que el árbol original. Iterativamente se van eliminando nodos según este criterio, eligiendo como nodo al que se poda a aquel que mejora la clasificación del conjunto de datos de validación. De esta manera, los nodos creados debido a datos ruidosos o erróneos aislados son eliminados.

La desventaja de este método es que se dispone de menos datos para el entrenamiento y es preciso disponer de suficientes datos, tanto en el conjunto de datos de entrenamiento como en el conjunto de datos de validación, de tal manera que las muestras sean significantes desde un punto de vista estadístico. Es típico utilizar 2/3 de los datos disponibles para el entrenamiento y 1/3 de los datos disponibles para la validación.

Cuando el conjunto de datos disponible es pequeño se suele emplear la técnica conocida como **validación cruzada de k iteraciones (K -fold cross-validation)**. Mediante esta técnica los datos se dividen en k subconjuntos de igual tamaño. Un subconjunto es utilizado como datos de prueba (o validación), mientras que el resto de los subconjuntos, (los $k-1$ subconjuntos restantes), son utilizados como datos de entrenamiento. La validación cruzada se repite k veces, y en cada una de las repeticiones se utiliza uno de los subconjuntos como datos de prueba.

Para obtener el resultado final, se realiza la media de los resultados obtenidos en cada iteración. El número de iteraciones conveniente dependerá de los datos disponibles y del número de atributos, pero es habitual utilizar una validación cruzada de 10 iteraciones, ya que suele ofrecer buenos resultados.

3.8. Medidas de la precisión de la clasificación. Curva ROC

Como previamente se ha indicado, se puede utilizar un conjunto de ejemplos de los disponibles (datos de validación) para evaluar la hipotética efectividad de la clasificación de instancias futuras por un árbol de decisión inducido y por los diversos árboles que resultan de la poda. Sin embargo, para esta evaluación se está teniendo en cuenta un conjunto de datos de prueba limitados y, por ello, surge la siguiente pregunta: **¿Cómo podemos estimar la precisión real en la clasificación de futuras instancias?** Es evidente que no es lo mismo contar con conjuntos de datos de entrenamiento y validación numerosos que no contar con ellos.

Es lógico que se confíe más en una clasificación inducida a partir de 1 000 000 de datos de entrenamiento que a partir de un conjunto de 100 instancias. Igualmente será más lógico confiar en una validación que utiliza un numeroso conjunto de datos de validación. Si, por ejemplo, se están utilizando $N=100$ ejemplos de prueba y, de esos 100 ejemplos, 90 están bien clasificados de acuerdo con una hipótesis h , se tiene una tasa de éxito te de 90 %. ¿Cómo se puede extrapolar este hecho a futuras instancias? ¿Cuál es la mejor estimación de la precisión de h a la hora de clasificar futuras instancias?

En principio se podría considerar que una buena estimación de la tasa de éxito de clasificación de futuras instancias es 0.9. Sin embargo, también se sabe que, dado que la muestra de futuras instancias no va a ser exactamente igual que la muestra de instancias de prueba N , se debería aplicar un intervalo de confianza en valores alrededor de te con el fin de tener en cuenta esta desigualdad en las muestras.

Para calcular dicho intervalo se asumen ciertas condiciones con el fin de aplicar teorías estadísticas que simplifican el problema. Por ejemplo, se asume que la predicción de la clase de cada una de las instancias es un evento independiente del

resto de predicciones. Este evento tiene dos resultados posibles: éxito o error en la predicción.

Por tanto, las predicciones de las clases de las diferentes instancias constituyen una serie de eventos independientes, con dos posibles resultados, siendo un proceso de Bernoulli. Así, consideramos la tasa de éxito esperada como una variable aleatoria t_e para un conjunto de N muestras, con una media de p y una varianza $p(1-p)/N$.

Si tenemos un gran conjunto de muestras, esto es, si N es grande ($N>100$), la distribución de la tasa de éxito será próxima a una distribución normal. Para una distribución normal, la probabilidad de que una variable aleatoria X , con una media igual a 0, se inscriba dentro de un cierto rango de confianza de anchura $2z$, $P[-z \leq X \leq z] = c$, se puede obtener a partir de tablas de distribución de probabilidad normal $N(0,1)$, que permiten relacionar z con el nivel de confianza deseado.

En la Tabla 2 se muestran algunos valores de z que se pueden extraer de dichas tablas, en este caso se relaciona z con la probabilidad de que la variable X sea superior a z , $P[X \geq z]$. Dado que se asume en dicha tabla que la variable X tiene una media de cero y una varianza de 1, podemos considerar que las cifras de z se miden en desviaciones estándar de la media. Por tanto, un valor de $P[X \geq z] = 0.01$ implica que hay un 1 % de probabilidad de que el valor de X sea superior a 2.33 veces la desviación estándar sobre el valor de la media. Dado que tenemos una distribución simétrica, existe también un 1 % de probabilidad de que el valor de X sea inferior a -2.33 veces la desviación estándar sobre el valor de la media, luego se da la probabilidad de que X se encuentre en el intervalo de confianza $[-2.33, 2.33]$ de $P[-2.33 \leq X \leq 2.33] = 1 - 0.01 - 0.01 = 0.98$.

$P[X \geq z]$	z
1%	2.33
5%	1.65
10%	1.28
25%	0.68
40%	0.25

Tabla 2. Límite del intervalo de confianza z para diferentes valores de confianza.

Para poder utilizar estas tablas cuando la media no es igual a 0 ni la varianza es igual a 1, tenemos que restar a la variable aleatoria te la media p y dividir por la desviación estándar $\sqrt{p(1-p)/N}$:

$$P\left[-z < \frac{t_e - p}{\sqrt{\frac{p(1-p)}{N}}} < z\right] = c \quad (6)$$

A partir de la anterior expresión, se obtienen las siguientes fórmulas para calcular los intervalos de confianza (**Witten & Frank, 2005**). El límite superior del intervalo viene dado por:

$$p = \frac{\left(t_e + \frac{z^2}{2N} + z\sqrt{\frac{t_e}{N} - \frac{t_e^2}{N^2} + \frac{z^2}{4N^2}}\right)}{\left(1 + \frac{z^2}{N}\right)} \quad (7)$$

El límite inferior del intervalo viene dado por:

$$p = \frac{\left(t_e + \frac{z^2}{2N} - z \sqrt{\frac{t_e}{N} - \frac{t_e^2}{N^2} + \frac{z^2}{4N^2}} \right)}{\left(1 + \frac{z^2}{N} \right)} \quad (7)$$

Por ejemplo, si tenemos una tasa de éxito con 1 000 datos de prueba de 0.75 y queremos una confianza $c = 0.80$, utilizamos el valor de la tabla $P[X \geq z] = 10$, siendo $z = 1.28$. Así, el intervalo que se obtendría para p es [0.732, 0.767]. Podemos afirmar con un 80% de confianza que la tasa de éxito real se encontrará en dicho intervalo.

El intervalo será más estrecho según N aumente. Lógicamente es más fiable un resultado de validación cuando se tienen más muestras.

Curva ROC

La medida más popular que se utiliza para determinar el rendimiento de un sistema de clasificación es su precisión (*accuracy*): el porcentaje de instancias clasificadas correctamente. La precisión es una medida fácil de entender y de comparar, pero obvia algunos aspectos importantes en el rendimiento como pueden ser, por ejemplo, el número de verdaderos positivos (TP – *True Positive*), falsos positivos (FP – *False Positive*), verdaderos negativos (TN – *True Negative*) y falsos negativos (FN – *False Negative*). Estos aparecen siempre que clasifiquemos una clase para la que tengamos instancias pertenecientes a esa clase e instancias que no pertenecen a ella. A la hora de clasificar, la mayoría de los clasificadores producen una puntuación sobre la que deciden si la instancia pertenece a una clase o no que varía entre 0 (definitivamente negativo) y 1 (definitivamente positivo). La Figura 6 muestra cómo se clasificaría un conjunto de datos de pacientes sanos y enfermos en función del umbral elegido para la toma de la decisión.

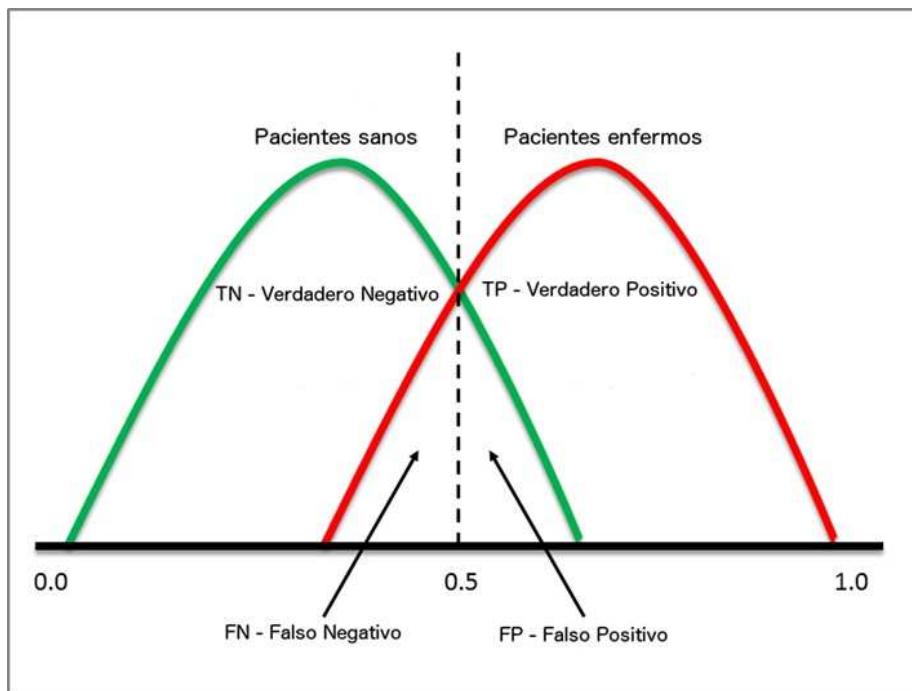


Figura 6. Conjunto de datos de pacientes sanos y enfermos y su clasificación en función del umbral elegido.

Las medidas mencionadas anteriormente dan lugar a otras medidas también interesantes a la hora de evaluar el clasificador:

- ▶ **Sensibilidad (sensitivity):** también conocido como ratio de verdaderos positivos (TPR – *True Positive Rate*). Mide la capacidad del clasificador para detectar la clase en una población con instancias de esa clase, la proporción de casos positivos bien detectados. Se calcula como $TP / (TP + FN)$.
- ▶ **Especificidad (specificity):** mide la capacidad del clasificador para descartar instancias que no pertenecen a la clase en una población con instancias de esa clase, la proporción de casos negativos correctamente detectados. Se calcula como $TN / (TN + FP)$.
- ▶ **Ratio de falsos positivos (FPR – *False Positive Rate*):** Proporción de casos negativos que el clasificador detecta como positivos. De forma más concreta $FP / (FP + TN)$.

En base a estas dos medidas, sensibilidad y especificidad, se forma la curva ROC (*Receiver Operating Characteristic*). Para ello se representa el TPR o sensibilidad en el eje de ordenadas y el FPR en el eje de abcisas ($1 - \text{especificidad}$). La curva ROC traza el TPR frente al FPR a diferentes umbrales de clasificación. Al disminuir el umbral de clasificación clasifica más elementos como positivos, aumentando así tanto los falsos positivos como los verdaderos positivos.

El área de la curva ROC tomará valores entre 0,5 (cuando no el sistema no distinga entre un positivo y un falso positivo) y 1 (el sistema clasifica perfectamente para esta clase). La Figura 7 muestra la curva ROC para dos clasificadores diferentes.

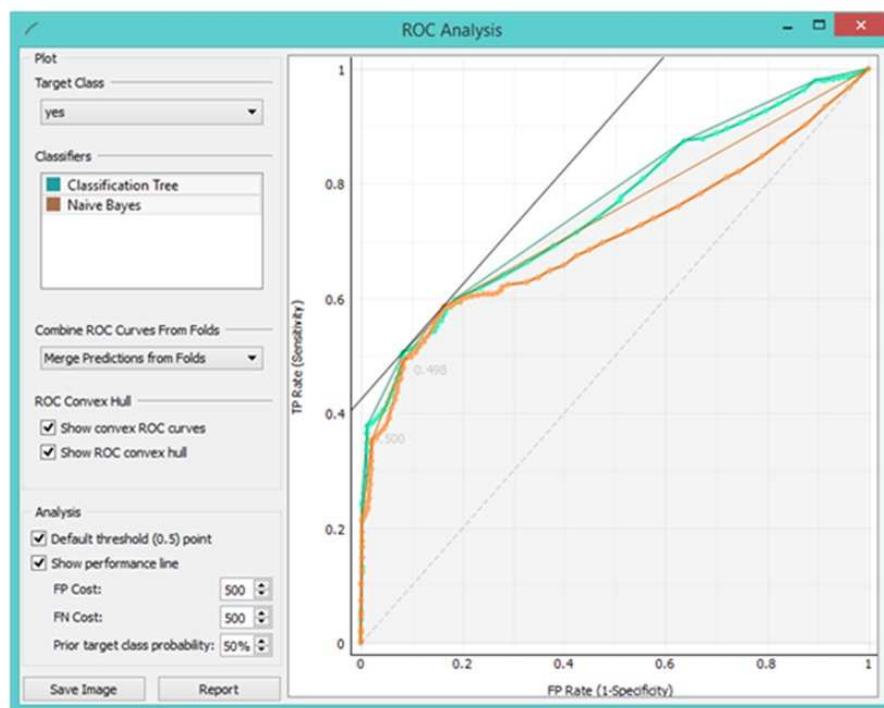


Figura 7. Análisis de la curva ROC de los clasificadores efectuado con la herramienta Orange3. Fuente:

<https://orange3.readthedocs.io/en/3.4.0/widgets/evaluation/rocanalysis.html>

3.9. Simplificación de árboles de decisión mediante poda: algoritmo C4.5

El algoritmo C_{4.5} es un método de inducción de árboles de decisión basado en ID3.

Este algoritmo fue propuesto por J.R. Quinlan que escribió un libro sobre el mismo (Quinlan, 1993).

Mientras que el algoritmo ID3 se aplica a atributos con valores discretos, el algoritmo C_{4.5} se puede aplicar tanto a atributos con valores discretos como a atributos con valores continuos. En este último caso, en cada nodo se ha de establecer un umbral de valor del atributo para realizar la partición de ejemplos. Además, C_{4.5} puede trabajar con datos ausentes, que no son tenidos en cuenta a la hora de calcular las métricas para seleccionar los atributos. El método de selección de atributos que utiliza C_{4.5} es la medida de proporción de ganancia.

C_{4.5} realiza una **poda** tras la generación del árbol (*pospoda*) con el fin de mejorar la generalización del modelo. Una vez que el árbol ha sido generado, C_{4.5} elimina aquellos nodos del árbol para los cuales el resultado de podar es una mejora en la precisión en la clasificación.

¿Cómo el algoritmo C_{4.5} estima la precisión de la clasificación? La estadística en la que se basa el algoritmo C_{4.5} se considera débil, pero, al mismo tiempo, ofrece buenos resultados en la práctica. C_{4.5} no utiliza un conjunto de ejemplos para la validación separado de los datos de entrenamiento, sino que **la validación la realiza a partir de los mismos datos de entrenamiento**.

Tras la poda, C_{4.5} asigna a un nodo la clase mayoritaria en el conjunto de instancias que alcanzan ese nodo, y esto da lugar a un determinado número de errores e debido a que hay instancias que no pertenecen a esa clase. Para dicho nodo la tasa de error tendría un valor $\text{terror} = e/N$, siendo N el número de instancias que lo alcanzan.

Esta estimación, al ser calculada sobre los mismos datos de entrenamiento, resulta en una estimación claramente optimista y la predicción del error está por tanto bastante sesgada.

Para compensar el hecho favorable que supone el emplear los propios datos del entrenamiento para el cálculo de la tasa de error, C_{4.5} realiza lo que se denomina una **poda pesimista**, ajustando esa tasa de error con una penalización al valor obtenido.

Para un determinado nivel de confianza, toma el nivel más desfavorable del intervalo de confianza como medida del error (lo que correspondería al límite superior del intervalo expresado en la Ecuación 7 de la sección anterior) y no se tiene en cuenta la posibilidad de que el valor pueda encontrarse dentro del intervalo de confianza.

Por defecto C_{4.5} toma un valor de confianza $c = 0.25$.

Para estimar la tasa real del error terror_real a partir de la tasa observada terror , obtiene el límite superior del intervalo de confianza a partir de las siguientes expresiones (Witten & Frank, 2005):

$$P \left[\frac{\text{terror} - \text{terror}_{\text{real}}}{\sqrt{\frac{\text{terror}_{\text{real}}(1-\text{terror}_{\text{real}})}{N}}} > z \right] = c \quad (9)$$

Tomando el límite superior del intervalo obtenemos:

$$t_{error_{real}} = \frac{\left(t_{error} + \frac{z^2}{2N} + z \sqrt{\frac{t_{error}}{N} - \frac{t_{error}^2}{N^2} + \frac{z^2}{4N^2}} \right)}{\left(1 + \frac{z^2}{N} \right)} \quad (10)$$

Siendo $z=0.68$ para un valor de $c=0.25$ (ver Tabla 2).

Para conjuntos de datos muy grandes la desviación estándar tenderá a ser pequeña, con lo cual la estimación pesimista se acercará al error observado (Mitchell, 1997).

3.10. Ensemble Learning y Random Forest

En la Figura 1 del Tema 1 vimos que una de las ramas del aprendizaje automático o *machine learning* era el **ensemble learning**, también conocido como **integrated learning**. En castellano las traducciones habituales son **aprendizaje ensemble** o **aprendizaje integrado**.

Actualmente, los métodos más modernos y maduros utilizados para obtener los resultados más precisos en entornos de producción, además de las redes neuronales, que veremos en los Tema 5 y 6, son precisamente los métodos *ensemble learning*, a pesar de que las primeras son más populares.

En los métodos de aprendizaje integrado, la idea es unir un conjunto de algoritmos ineficientes en los que cada uno colabora corrigiendo los errores del resto del conjunto (Krawczyk et al., 2017). De esta manera, se consigue una calidad general más alta que la de los mejores algoritmos individuales que trabajan de forma aislada.

De hecho, se obtienen resultados aún mejores cuando los algoritmos que componen el conjunto son más inestables, en el sentido de predecir resultados completamente diferentes con poco ruido en los datos de entrada. Por lo tanto, es común utilizar algoritmos como la regresión y los árboles de decisión como parte de los métodos de aprendizaje integrado, porque estos algoritmos son muy sensibles a los valores atípicos en los datos de entrada. Por el contrario, algoritmos muy estables como el Naïve Bayes o el k-NN (*k Nearest Neighbors* o los *k vecinos más cercanos*) son utilizados raramente en los métodos *ensemble learning*.

Como vemos, el aprendizaje integrado no está exclusivamente relacionado con los árboles de decisión. Sin embargo, lo abordamos en este tema con el fin de

aprovechar para describir uno de los algoritmos *ensemble learning* más populares: el **random forest**.

Existen tres métodos principales dentro del aprendizaje integrado en función de cómo se combinan los diferentes algoritmos que componen cada método: el **stacking** (apilamiento), el **bagging** (embolsamiento) y el **boosting** (refuerzo). Estos tipos de métodos se utilizan actualmente para cualquier aplicación en la que se puedan aplicar los algoritmos supervisados clásicos, aunque también existen métodos de aprendizaje integrado dentro del aprendizaje no supervisado, así como los sistemas de búsqueda, la visión por ordenador o la detección de objetos.

Stacking

Los métodos *stacking* (o *stack generalization*) son quizás los métodos menos utilizados de todos (comparados con los métodos *bagging* y *boosting*). El concepto en el que se basan es muy simple (ver Figura 8): entrenar diferentes algoritmos (por ejemplo, un k-NN, un árbol de decisión y un SVM – *Support Vector Machine* o Máquina de vectores de soporte, creadas por Vapnik) (Noble, 2006) utilizando los mismos datos y utilizarlos en paralelo para clasificar la misma entrada.

Finalmente, se promedian las salidas de los diferentes algoritmos para obtener un valor de salida final, normalmente utilizando un algoritmo de regresión (Cao y Wang, 2018).

Un ejemplo de este tipo de técnica es el **FWLS** (*Feature-Weighted Linear Stacking* o Apilamiento lineal ponderado por características), utilizado como método de recomendación de contenido en plataformas de comercio en línea (Henriques y Mendes-Moreira, 2016).

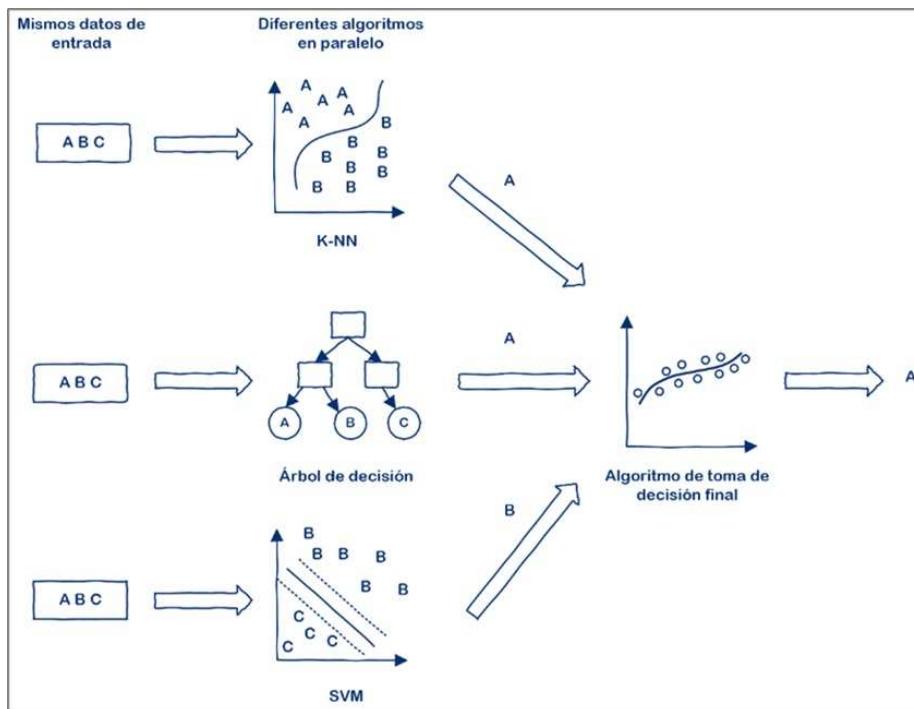


Figura 8. Esquema de funcionamiento en los métodos stacking.

Bagging (y el Random Forest o bagging con árboles de decisión)

Por su parte, en los métodos *bagging* (*Bootstrap AGGREGatING*) se utiliza un conjunto de algoritmos en paralelo en los que todos los algoritmos son iguales (por ejemplo, todos son árboles de decisión), pero cada uno de ellos se entrena con un subconjunto aleatorio de datos extraídos del mismo conjunto de datos de entrenamiento (Wu et al., 2018).

Por último, y al igual que en los métodos *stacking*, se promedian las salidas de los diferentes algoritmos para obtener un valor de salida final, de nuevo normalmente utilizando un algoritmo de regresión. La Figura 9 muestra el esquema de funcionamiento de este tipo de métodos.

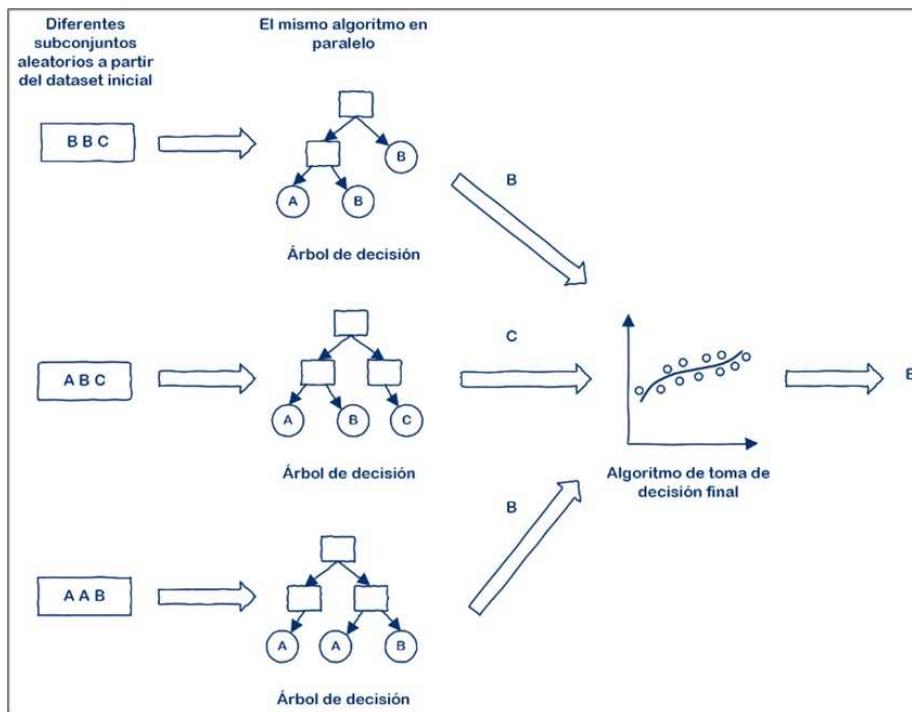


Figura 9. Esquema de funcionamiento en los métodos bagging (usando como ejemplo el random forest).

De hecho, uno de los métodos de apilamiento más conocidos es el *random forest* (o árbol aleatorio), en el que se utilizan precisamente árboles de decisión como algoritmos a aplicar en paralelo. El algoritmo *random forest* se utiliza ampliamente, por ejemplo, en los teléfonos móviles para reconocer dónde están las caras en una imagen cuando se utiliza una aplicación de cámara y mostrar el encuadre de las mismas en la fotografía (Kremic y Subasi, 2016). Aunque puede perderse cierta precisión con respecto a una red neuronal, el bosque aleatorio requiere muchos menos cálculos y es más factible utilizarlo en aplicaciones en tiempo real en dispositivos móviles debido a sus menores requisitos computacionales.

Boosting

En el caso de los métodos ***boosting***, los diferentes algoritmos elegidos se entrena uno por uno de forma secuencial en serie (en lugar de trabajar en paralelo, como sucede en los métodos *stacking* y *bagging*) (Al Daoud, 2019). Es decir, el primer algoritmo se entrena utilizando todos los datos de entrenamiento elegidos como entrada. Después, se observa qué casos han dado peores resultados después del primer algoritmo y se les da prioridad a la hora de elegir el subconjunto de datos de entrenamiento del segundo algoritmo, y así sucesivamente.

Todos los algoritmos podrían ser, por ejemplo, árboles de decisión. La principal ventaja de este tipo de algoritmos es que son muy precisos cuando se aplican como métodos de clasificación. Por el contrario, y como podría deducirse, su principal desventaja es que los algoritmos no se ejecutan en paralelo, sino en serie, y requieren un mayor tiempo de cálculo. No obstante, siguen siendo más rápidos que las redes neuronales.

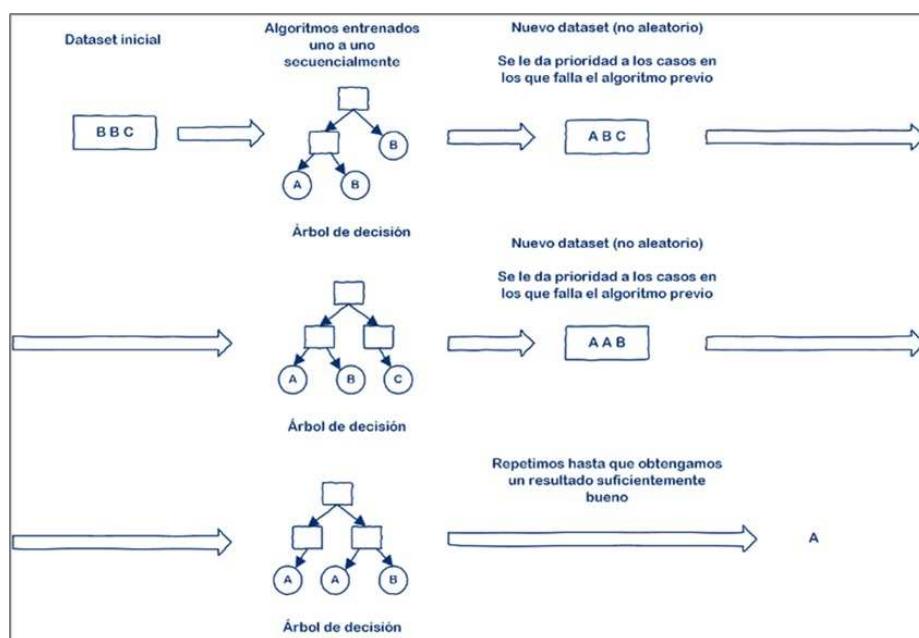


Figura 10. Esquema de funcionamiento en los métodos boosting.

Entre los principales algoritmos *boosting* se encuentran AdaBoost, CatBoost, LightGBM o XGBoost. Una de las principales aplicaciones de este tipo de algoritmos ha sido ordenar los resultados de las búsquedas en Google o en redes sociales (Wu et al. 2010).

3.11. Aplicaciones y ejemplos de implementación

Algunas aplicaciones de los árboles de decisión en la literatura

El aprendizaje del árbol de decisión es una de las técnicas más utilizadas para el aprendizaje inductivo, siendo un método bastante robusto frente a los ruidosos datos comparativos. Los árboles de decisión se utilizan como modelo predictivo para vincular los valores conocidos de los atributos de los elementos, representados como ramas en el árbol, con las conclusiones sobre un valor de atributo de elemento normalmente desconocido, por ejemplo, una etiqueta para la clase de elemento, representada como las hojas del árbol (Shalev-Schwartz y Ben-David, 2014).

Las entradas y salidas de la función objetivo suelen ser valores discretos, aunque también podrían ser continuos en el caso de las entradas. La representación de esta función objetivo toma la forma de un árbol y se interpreta como una serie de condiciones consecutivas que pueden ser fácilmente mapeadas a las reglas de clasificación y pueden ser fácilmente explicadas por un humano. Los árboles cuyas hojas toman valores discretos se denominan **árboles de clasificación**. En cambio, aquellos cuyas hojas pueden tomar valores continuos se denominan **árboles de regresión**. Los árboles de decisión se utilizan generalmente para el análisis de decisiones y también para describir datos en la extracción de datos.

Se utilizan en el diagnóstico de enfermedades en la medicina (Song y Ying, 2015) o en la determinación de la concesión de un préstamo, entre muchas otras posibilidades (Namazkhan et al., 2020). Otros posibles usos de los árboles de decisión incluyen la implementación de clasificadores para filtrado de *sitios webs spam* (Silva et al., 2016).

Un *sitio web spam* es una web creada para deliberadamente engañar y dañar potencialmente a los usuarios con el fin de beneficiarse (el contenido no es real, tratan de suplantar a otras webs, utilizan redireccionamientos engañosos, etc.).

Además, tal y como ya se ha explicado, el algoritmo *random forest* se utiliza ampliamente, por ejemplo, en los teléfonos móviles para reconocer dónde están las caras en una imagen cuando se utiliza una aplicación de cámara y mostrar el encuadre de las mismas en la fotografía (Kremic y Subasi, 2016). Aunque puede perderse cierta precisión con respecto a una red neuronal, el bosque aleatorio requiere muchos menos cálculos y es más factible utilizarlo en aplicaciones en tiempo real en dispositivos móviles debido a sus menores requisitos computacionales.

Ejemplos de implementación

En el anterior tema, vimos un ejemplo de cómo utilizar Python y sus librerías para llevar a cabo una regresión lineal, polinómica o una regresión lineal múltiple. Sin embargo, en esta ocasión estamos trabajando con algoritmos supervisados de clasificación, de modo que no podemos aplicar este tipo de técnicas. Sí que podemos utilizar una **regresión logística**, que, a pesar de su nombre, es un **clasificador**.

La regresión logística se utiliza para dar una probabilidad entre 0 y 1 de que los datos de entrada correspondan a una determinada clase de salida, pero también puede utilizarse para clasificar cualquier número de categorías (por ejemplo, para reconocer qué dígito manuscrito entre 0 y 9 se observa en una imagen). En lugar de utilizar los mínimos cuadrados ordinarios que podrían dar probabilidades inferiores a 0 o superiores a 1, se utilizan **modelos logit**, como la **función sigmoidea**:

$$S(x) = \frac{1}{1 + e^{-x}}$$

¿Y por qué contamos esto? Porque vamos a comparar los resultados obtenidos con un *dataset* determinado cuando utilizamos un árbol de decisión y la regresión logística. **Así aprenderemos también a comparar algoritmos entre sí.**

Nota: más adelante, en el Tema 5, cuando veamos las redes neuronales, aprovecharemos para compararlas con el algoritmo *random forest* en problemas similares, viendo, así, también, un ejemplo de su aplicación con Python.

A continuación, vamos a aplicar tanto la regresión logística como una técnica de árbol de decisión empleando Python y librerías apropiadas. Para ello, vamos a emplear scikit-learn , que implementa una versión optimizada del algoritmo CART, aunque sólo empleando variables numéricas. Si necesitamos emplear variables categóricas, podemos transformarlas previamente en numéricas empleando el método `map()` de pandas . Este método de esta librería nos permite clasificar de forma binaria (para valores -1 o 1) o como categorías (0, 1, 2, ...).

Vamos a emplear uno de los *datasets* más conocidos en *machine learning*, el **iris (el conjunto de datos flor Iris o iris de Fisher)**, introducido por Ronald Fisher en 1936 en un artículo como ejemplo de análisis de discriminante lineal. Este *dataset* es un conjunto de datos multivariante de 150 muestras, 50 de cada una de las tres especies de flor Iris (*Iris setosa*, *Iris virginica* e *Iris versicolor*), con medidas del largo y ancho del sépalo y el pétalo de cada una de ellas. Este *dataset* viene incluido en scikit-learn , de modo que lo podemos importar de dicha librería, como ya hemos visto en el Tema 1. No obstante, lo importaremos de [OpenML](#), que, como vemos, tiene una URL de acceso al CSV mediante:

https://www.openml.org/data/get_csv/61/dataset_61_iris.arff

Largo de sépalo	Ancho de sépalo	Largo de pétalo	Ancho de pétalo	Especies
5.1	3.5	1.4	0.2	I. setosa
4.9	3.0	1.4	0.2	I. setosa
4.7	3.2	1.3	0.2	I. setosa

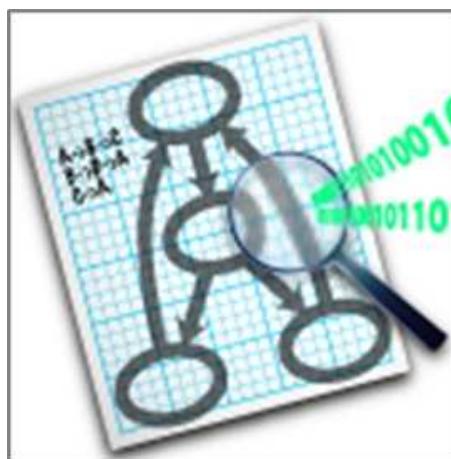
Tabla 3. Conjunto de datos flor Iris o iris de Fisher (muestra de las primeras filas).

En primer lugar, y dando por sentado que seguimos manteniendo instalados los módulos de los ejemplos del Tema 2, instalaremos los módulos `graphviz` (para dibujar diagramas, entre otros, los diagramas de árbol de decisión) y `pydotplus`, si no los tuviéramos ya en nuestro sistema, con la orden habitual:

```
PS C:\Users\xxx> pip install graphviz
```

```
PS C:\Users\xxx> pip install pydotplus
```

Nota: es necesario también instalar los ejecutables de `graphviz` para que el ejecutable `dot` esté en la ruta de búsqueda del sistema y así poder renderizar diagramas:



Fuente: <https://www.graphviz.org/download/>

También será necesario incluir la ruta al ejecutable en nuestro `PATH`, por ejemplo, en Windows 10 esto se realizaría en Panel de Control → Sistema → Configuración avanzada del sistema → Variables de entorno, y ahí editar la variable `PATH` añadiendo, por ejemplo:

```
C:\Program Files (x86)\Graphviz2.38\bin
```


O la ruta de instalación que hayamos empleado. **Hemos de reiniciar Visual Studio Code si estamos realizando este proceso con el IDE abierto.**

Proseguimos. Primero echamos un vistazo a los datos del *dataset* en modo texto, tal y como se explica en los comentarios del código:

```
# Load libraries
from pandas import read_csv

url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.ar
ff"
# La siguiente línea no es necesaria usando esta fuente, pues ya
# incluye la cabecera
#names = ['sepallength', 'sepalwidth', 'petallength', 'petalwidt
#h', 'class']
#dataset = read_csv(url, names=names)
dataset = read_csv(url)

# mostramos la "forma", debería haber 150 entradas con 5 atribut
os cada una
print(dataset.shape)
#> (150 , 5)

# mostramos las 3 primeras entradas para echar un vistazo
print(dataset.head(3))
#>   sepallength  sepalwidth  petallength  petalwidth      cla
ss
#>0       5.1        3.5         1.4        0.2  Iris-
setosa
#>1       4.9        3.0         1.4        0.2  Iris-
setosa
#>2       4.7        3.2         1.3        0.2  Iris-
setosa

# mostramos un resumen estadístico de los datos
print(dataset.describe())
#>      sepallength  sepalwidth  petallength  petalwidth
#>count  150.000000  150.000000  150.000000  150.000000
#>mean    5.843333    3.054000    3.758667    1.198667
#>std     0.828066    0.433594    1.764420    0.763161
#>min     4.300000    2.000000    1.000000    0.100000
#>25%    5.100000    2.800000    1.600000    0.300000
#>50%    5.800000    3.000000    4.350000    1.300000
#>75%    6.400000    3.300000    5.100000    1.800000
#>max    7.900000    4.400000    6.900000    2.500000

# distribución por clases
print(dataset.groupby('class').size())
#>Iris-setosa      50
#>Iris-versicolor  50
#>Iris-virginica   50
#>dtype: int64
```

Echemos un vistazo a los datos ahora en modo gráfico, de nuevo tal y como se explica en los comentarios del código de ejemplo:

```
# Load libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot

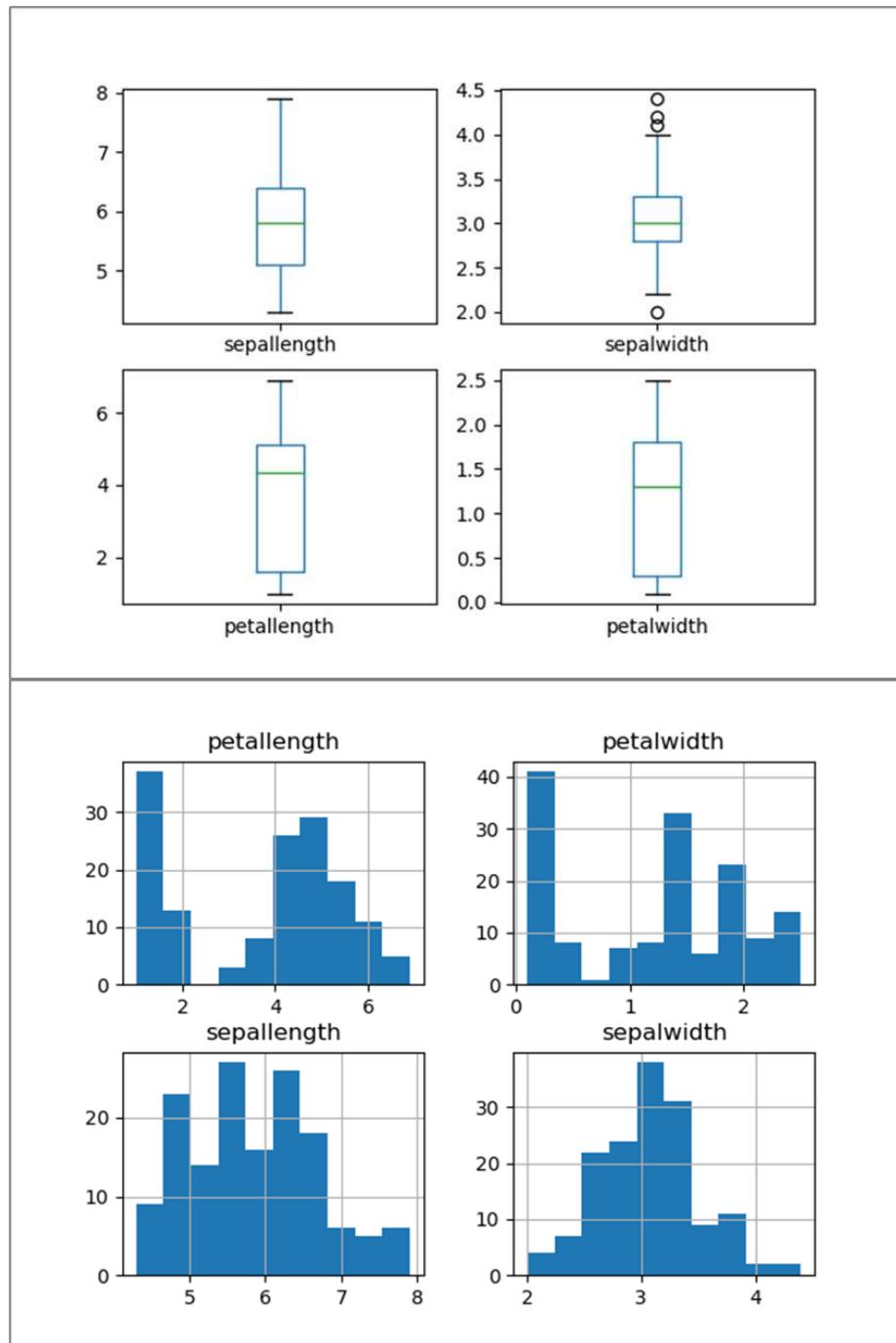
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.ar
ff"
# La siguiente línea no es necesaria usando esta fuente, pues ya
# incluye la cabecera
#names = ['sepallength', 'sepalwidth', 'petallength', 'petalwidt
h', 'class']
#dataset = read_csv(url, names=names)
dataset = read_csv(url)

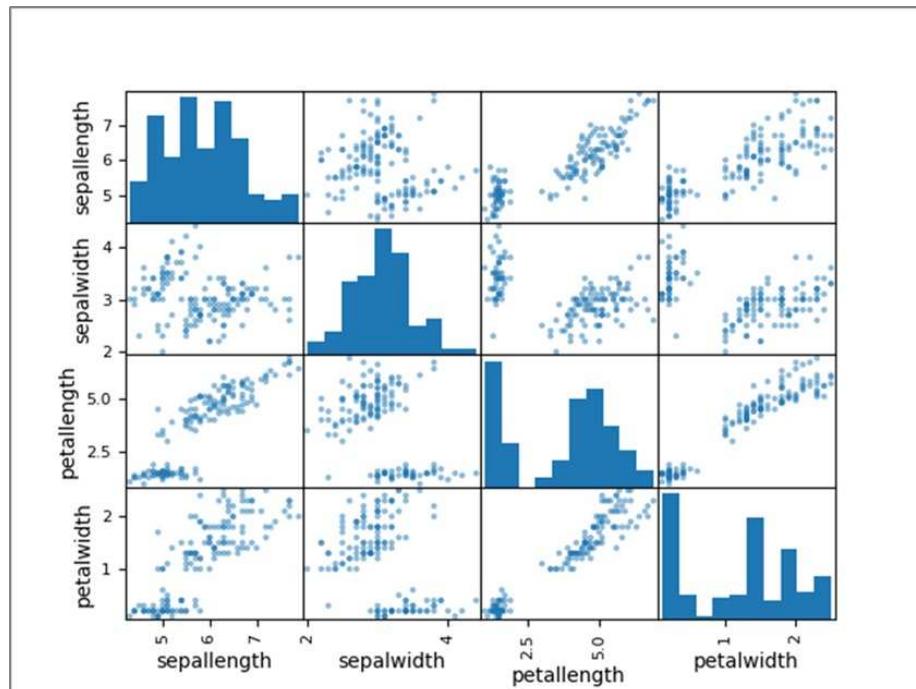
# gráficos univariable:
# diagramas de caja (box and whisker)
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False,
sharey=False)
pyplot.show()

# histogramas
dataset.hist()
pyplot.show()

# gráficos multivariable
# matriz de dispersión
scatter_matrix(dataset)
pyplot.show()
```

Obteniendo los siguientes gráficos consecutivamente:





Ahora vamos a evaluar dos algoritmos *machine learning* y compararlos entre sí: la regresión logística frente al árbol de decisión CART.

Para ello, separaremos nuestro *dataset*: usaremos un 80 % de los datos para entrenar los algoritmos y un 20 % de los datos para hacer los test de predicción. Esta suele ser una proporción habitual.

Además, utilizaremos una **validación cruzada estratificada de 10 veces (*k-fold*)** para estimar la precisión del modelo.

Esto dividirá nuestro conjunto de datos en 10 partes, entrenando en 9 partes y testeando en 1 parte y repetirá para todas las combinaciones de divisiones de entrenamiento-test.

Estratificado, significa que cada pliegue o división del conjunto de datos tendrá como objetivo tener la misma distribución de ejemplo por clase que existe en todo el conjunto de datos de entrenamiento.

```

# Load libraries
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)

# Dividimos el dataset en 80% de datos para entrenar y 20% para testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X, y, test_size=0.20, random_state=1, shuffle=True)

# Cargamos los algoritmos
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('CART', DecisionTreeClassifier()))

# evaluamos cada modelo por turnos
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,
                                 scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

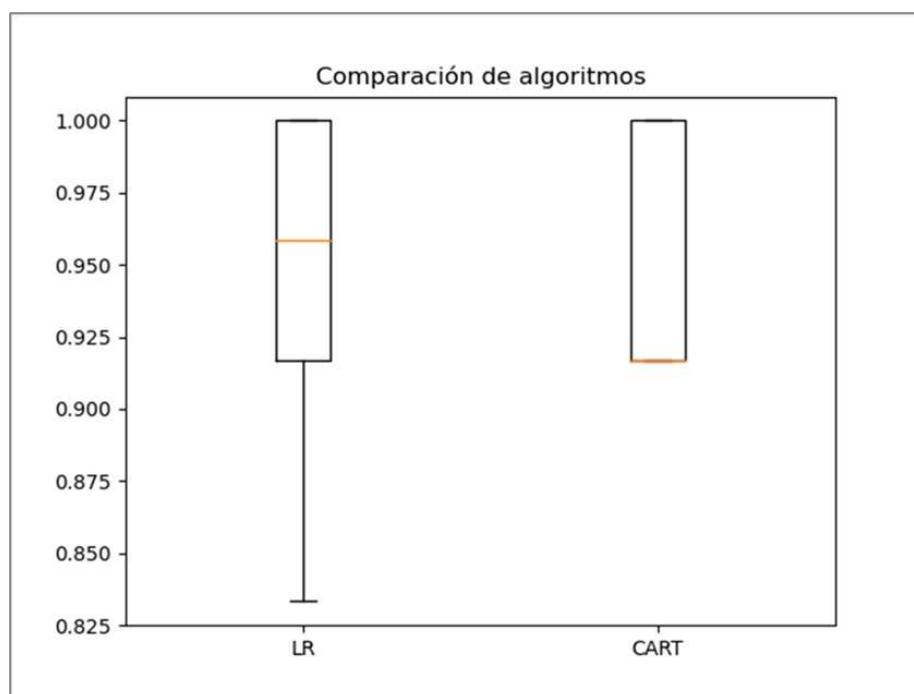
# Comparación de algoritmos
pyplot.boxplot(results, labels=names)
pyplot.title('Comparación de algoritmos')
pyplot.show()

#>LR: 0.941667 (0.065085)
#>CART: 0.950000 (0.040825)

```

Obtendremos la siguiente gráfica comparativa, además. Como vemos en los resultados por consola (mostrados en los comentarios del código) y en la gráfica, el CART tiene una precisión media del 95 % frente al 94 % de la regresión logística, en estas circunstancias. Nótese que hemos evaluado el algoritmo frente en una

validación cruzada estratificada de 10 veces, de ahí que tengamos una media y una desviación para el valor de la precisión. En ambos el extremo superior del bigote está al 100 % (máximo), además del extremo superior de la caja (percentil 75 o cuartil superior). Esto sucede porque tenemos varias pruebas en las cuales la precisión es del 100 %. Sin embargo, en la regresión logística tenemos precisiones mínimas más bajas, por debajo del 85 %, mientras que en el árbol de decisión tanto la mediana (línea horizontal), como el mínimo (extremo inferior del bigote), como el cuartil inferior (percentil 25) permanecen por encima del 90 %.



Vamos a quedarnos, por el interés de este tema, solo con el algoritmo CART. Vamos a realizar predicciones con el algoritmo de árbol, obteniendo resultados por consola como vemos en los comentarios en el código:

```

# Load libraries
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)

# Dividimos el dataset en 80% de datos para entrenar y 20% para testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X, y, test_size=0.20, random_state=1, shuffle=True)

# Realizamos predicciones con el dataset de validación
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# Evaluamos las predicciones, en primer lugar la precisión obtenida
print(accuracy_score(Y_validation, predictions))
#> 0.9666666666666667

# ahora la matriz de confusión (vemos en este ejemplo que sólo hemos cometido un fallo)
print(confusion_matrix(Y_validation, predictions))
#> [[11  0  0]
#> [ 0 12  1]
#> [ 0  0  6]]

# y finalmente un informe de clasificación que ofrece un desglose de cada clase por precisión, recuerdo, puntuación f1 y apoyo, mostrando excelentes resultados (dado que el conjunto de datos de validación era pequeño).
print(classification_report(Y_validation, predictions))
#>              precision    recall   f1-score   support
#>
#>    Iris-setosa       1.00      1.00      1.00       11

```

```
#>Iris-versicolor      1.00      0.92      0.96      13
#> Iris-virginica     0.86      1.00      0.92       6
#>
#>      accuracy           0.97      0.97      0.97      30
#>      macro avg          0.95      0.97      0.96      30
#>      weighted avg        0.97      0.97      0.97      30

# realizamos una predicción de ejemplo:
print(model.predict([[6.0, 3.0, 5.0, 2.0]]))
#[['Iris-virginica']]
```

Y, por último, vamos a visualizar el aspecto del árbol:

```
# Load libraries
from pandas import read_csv
import matplotlib.pyplot as plt
import matplotlib.image as pltimg
import pydotplus
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)

# Dividimos el dataset en 80% de datos para entrenar y 20% para testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X, y, test_size=0.20, random_state=1, shuffle=True)
```

```

# Realizamos predicciones con el dataset de validación
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

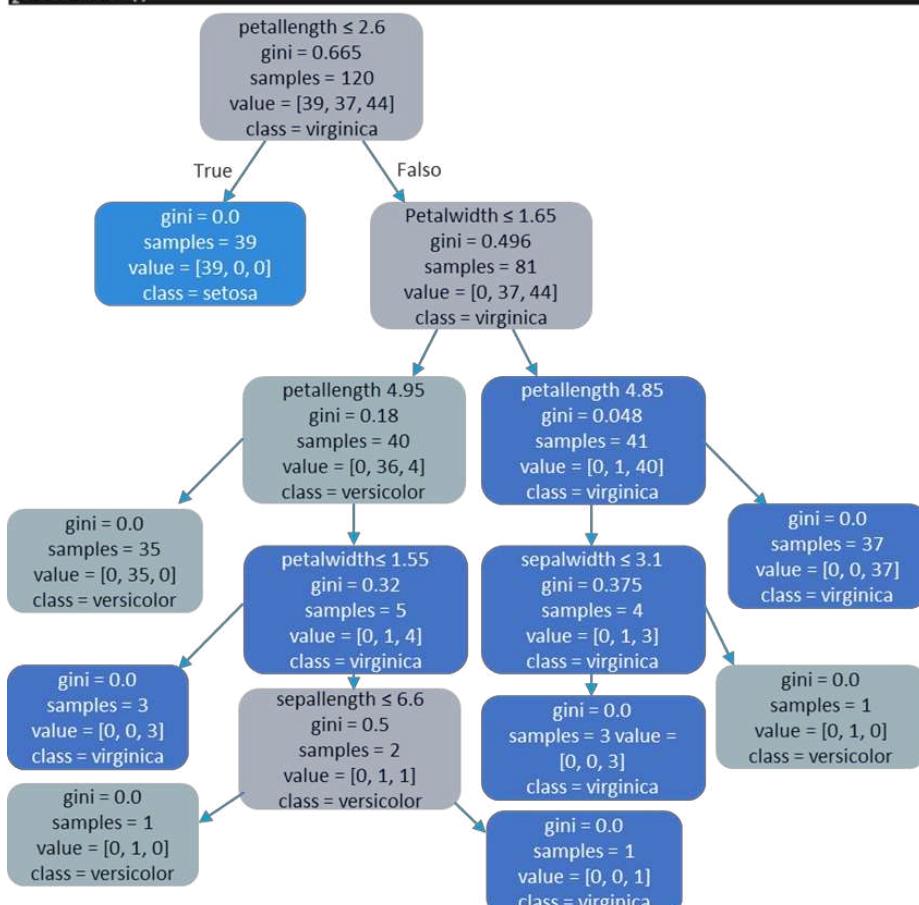
# damos detalles sobre el modelo
print(model)

# mostramos el árbol gráficamente
data = tree.export_graphviz(model, out_file=None, feature_names=
dataset.columns.values[0:4], class_names=["setosa", "versicolor",
, "virginica"], filled=True, rounded=True, special_characters=True)

graph = pydotplus.graph_from_dot_data(data)
graph.write_png('mydecisiontree.png')

img = pltimg.imread('mydecisiontree.png')
imgplot = plt.imshow(img)
plt.show()

```



Obtenemos también los parámetros empleados en el árbol de decisión, que pueden ser modificados para realizar pruebas alternativas al construirlo:

```
DecisionTreeClassifier(ccp_alpha=0.0,  
                      class_weight=None,  
                      criterion='gini',  
                      max_depth=None,  
                      max_features=None,  
                      max_leaf_nodes=None,  
                      min_impurity_decrease=0.0,  
                      min_impurity_split=None,  
                      min_samples_leaf=1,  
                      min_samples_split=2,  
                      min_weight_fraction_leaf=0.0,  
                      presort='deprecated',  
                      random_state=None,  
                      splitter='best')
```

3.12. Referencias bibliográficas

Al Daoud, E. (2019). Comparison between XGBoost, LightGBM and CatBoost Using a Home Credit Dataset. *International Journal of Computer and Information Engineering*, 13(1), 6-10.

Cao, C. & Wang, Z. (2018). IMCStacking: Cost-sensitive stacking learning with feature inverse mapping for imbalanced problems. *Knowledge-Based Systems*, 150, 27-37.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009). The WEKA data mining software: An update. Association for Computing Machinery. Disponible en <https://doi.org/10.1145/1656274.1656278>

Henriques, P. M., & Mendes-Moreira, J. (2016, September). *Combining recommendation systems with a dynamic weighted technique*. In 2016 Eleventh International Conference on Digital Information Management (ICDIM) (pp. 203-208). IEEE.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J. & Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37, 132-156.

Kremic, E. & Subasi, A. (2016). Performance of random forest and SVM in face recognition. *The International Arab Journal of Information Technology*, 13(2), 287-293.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Namazkhan, M., Albers, C. & Steg, L. (2020). A decision tree method for explaining household gas consumption: The role of building characteristics, socio-demographic variables, psychological factors and household behaviour. *Renewable and*

Sustainable Energy Reviews, 119, 109542.

Noble, W. S. (2006). What is a support vector machine? *Nature biotechnology*, 24(12), 1565-1567.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81-106. Disponible en <https://doi.org/10.1007/BF00116251>

Quinlan, J.R. (1993). *C4.5: programs for Machine Learning*. San Francisco: Morgan Kauffmann.

Shalev-Shwartz, S. & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Silva, R. M., Almeida, T. A. & Yamakami, A. (2016, December). *Towards web spam filtering using a classifier based on the minimum description length principle*. In 2016 15th IEEE International.

Song, Y. Y. & Ying, L. U. (2015). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2), 130.

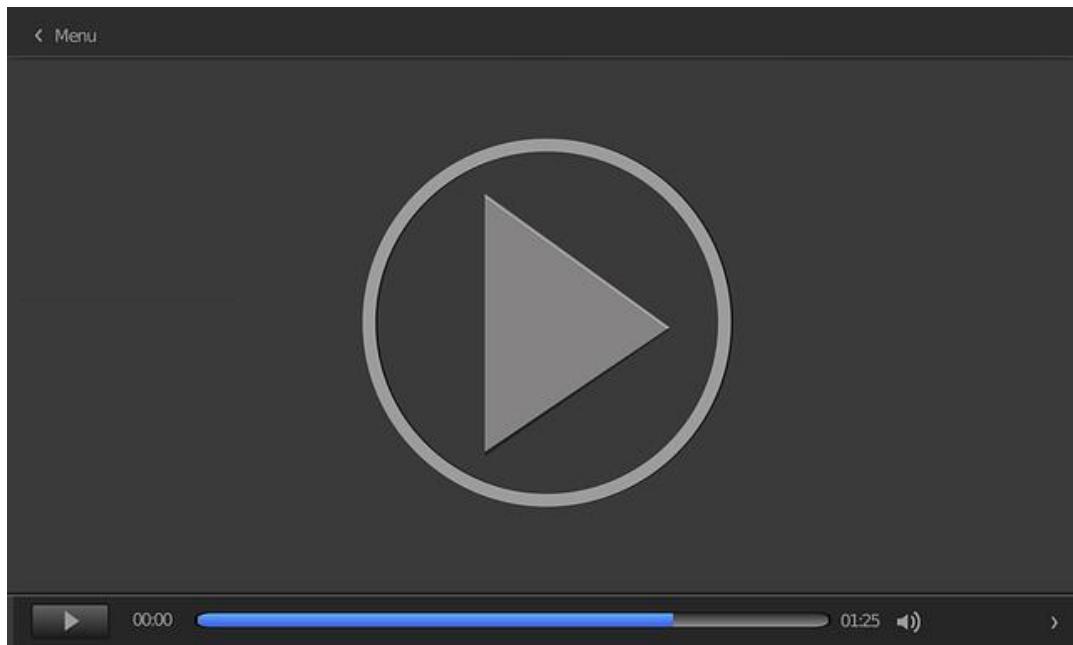
Witten, I.H. & Frank, E. (2005). *Data Mining*. USA: Morgan Kaufmann Publishers.

Wu, Q., Burges, C. J., Svore, K. M. & Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3), 254-270.

Wu, Z., Li, N., Peng, J., Cui, H., Liu, P., Li, H. & Li, X. (2018). Using an ensemble machine learning methodology-Bagging to predict occupants' thermal comfort in buildings. *Energy and Buildings*, 173, 117-127.

Introducción a la herramienta Weka

Esta lección magistral consiste en un tutorial introductorio al uso de la herramienta Weka. Se enseñarán funciones básicas tales como la carga de datos y la ejecución de un algoritmo de clasificación, así como la interpretación de los resultados obtenidos.



03. Introducción a la herramienta Weka

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=aa424a9a-03b6-4e94-9c58-aff800f9dd27>

Inducción de árboles de decisión mediante el algoritmo ID3

Quinlan, J.R. (1986). Induction of Decision Trees. *Machine Learning*, 1, 81-106.

<http://link.springer.com/article/10.1007%2FBF00116251>

Artículo muy ilustrativo escrito por el propio creador del algoritmo ID3, J.R. Quinlan. Resume el problema de aprendizaje de árboles de decisión y propone el algoritmo ID3, exponiendo sus ventajas, limitaciones y dificultades, así como propuestas para solventar estas dificultades.

Software de minería de datos Weka

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 10-18.
http://www.cms.waikato.ac.nz/~ml/publications/2009/weka_update.pdf

Artículo escrito por los creadores de Weka proporcionando una introducción a esta herramienta y contando la historia del proyecto.

UCI Machine Learning Repository

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
<http://archive.ics.uci.edu/ml/>

Esta web contiene casi 300 conjuntos de datos disponibles para ser utilizados en tareas de investigación y aprendizaje de la minería de datos. Tiene una interfaz muy útil para filtrar los conjuntos de datos disponibles en función del tipo de aprendizaje que se quiere realizar (clasificación —aprendizaje supervisado, clustering— aprendizaje no supervisado, etc.), los tipos de atributos que se desea manejar (nominales, numéricos, etc.), número de instancias de ejemplo o el área de conocimiento de los datos.

Weka 3: Software de minería de datos en Java

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<http://www.cs.waikato.ac.nz/ml/weka/>

Sitio web del grupo *Machine Learning Group* de la Universidad de Waikato dedicado a la herramienta Weka. Dispone de una sección para descarga del software, una lista de publicaciones relacionadas con el aprendizaje automático y la minería de datos, así como enlaces a páginas web relacionadas con estos campos y la inteligencia artificial.

Wiki de documentación sobre Weka

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<http://weka.wikispaces.com/>

Wiki de contenido público cuya finalidad es documentar la herramienta Weka. Contiene una sección muy interesante sobre las preguntas más frecuentes (FAQs), incluyendo cuestiones de uso básico y avanzado.

1. Completa esta sentencia con la opción correcta: «La clasificación mediante árboles de decisión es una tarea de aprendizaje de tipo [...]»:

 - A. Supervisado.
 - B. No-supervisado.
 - C. Supervisado y no supervisado.
 - D. Ninguna de las anteriores.

2. Indica cuáles de las siguientes afirmaciones son verdaderas:

 - A. El algoritmo ID3 permite valores numéricos en los atributos.
 - B. El algoritmo C4.5 permite valores numéricos en los atributos de salida.
 - C. Los algoritmos ID3 y C4.5 permiten valores nominales en los atributos de salida.
 - D. El algoritmo C4.5 permite valores numéricos y nominales en los atributos de entrada.

3. Marca de las siguientes opciones aquellas en las que resulta adecuado utilizar un árbol de decisión en la tarea de aprendizaje:

 - A. No se conoce la clase de las instancias disponibles.
 - B. La función objetivo tiene valores de salida discretos.
 - C. Existen varios atributos de salida.
 - D. Los datos de entrenamiento contienen errores.

4. Indica cuál de las siguientes afirmaciones es correcta:

 - A. El método de selección de los atributos especifica una heurística para seleccionar el atributo que mejor discrimina los ejemplos para una clase.
 - B. Un algoritmo básico de construcción de árboles de decisión recibe como entrada los datos de entrenamiento y los atributos de las instancias, y obtiene como salida un método de selección de atributos.
 - C. El método codicioso es un método de selección de atributos.
 - D. C4.5 no utiliza un método de selección de atributos.
5. Cuando se tienen atributos con un gran número de posibles valores ¿qué método de selección de atributos conviene utilizar?

 - A. Ganancia de información.
 - B. Proporción de ganancia.
 - C. Índice Gini.
 - D. Longitud de descripción mínima.
6. Indica cuáles de las siguientes afirmaciones son correctas respecto a ID3:

 - A. ID3 se basa en el método codicioso o greedy.
 - B. ID3 considera como heurística que el atributo cuyo conocimiento aporta mayor información en la predicción de la clase es el más útil.
 - C. ID3 utiliza la medida de proporción de ganancia.
 - D. ID3 trabaja con un espacio de hipótesis incompleto.

7. Indica las respuestas correctas en relación con los métodos de aprendizaje integrado:

- A. En los métodos stacking se utilizan los mismos datos de entrada a diferentes algoritmos en paralelo. Finalmente se utiliza un algoritmo decisor para dar la respuesta final.
- B. En los métodos bagging se utilizan el mismo tipo de algoritmos en paralelo, a los cuales se pasa como entrada diferentes subconjuntos aleatorios a partir del dataset inicial.
- C. Random forest es un tipo de método boosting utilizando árboles de decisión en serie.
- D. En los métodos boosting se parte de un dataset inicial y se entrenan diferentes algoritmos uno a uno secuencialmente, introduciendo como entrada de cada algoritmo un conjunto aleatorio del dataset inicial.

8. Indica cuáles de las siguientes afirmaciones son correctas respecto a C4.5:

- A. C4.5 utiliza la medida de proporción de ganancia.
- B. C4.5 utiliza un método de prepoda.
- C. C4.5 poda el árbol utilizando datos de prueba.
- D. C4.5 realiza una poda una vez se ha generado el árbol.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto al método de validación cruzada:

- A. Utiliza un conjunto de datos de validación y un conjunto de datos de entrenamiento.
- B. La validación cruzada divide los datos disponibles en dos subconjuntos.
- C. La validación cruzada no se puede utilizar para evaluar la efectividad de una poda.
- D. Cuando se tienen pocos datos no conviene utilizar la validación cruzada con k iteraciones.

10. Indica cuáles de las siguientes afirmaciones son correctas respecto a las medidas de precisión en la clasificación:

- A. La tasa de error de una muestra es el cociente entre el número de instancias erróneamente clasificadas y el número total de instancias.
- B. Cuanto mayor sea el número de instancias, el intervalo de confianza será mayor.
- C. C4.5 utiliza los propios datos de entrenamiento para calcular la tasa de error.
- D. Cuanto mayor nivel de confianza se requiere se obtiene un intervalo de confianza menor.

Técnicas de Inteligencia Artificial

Tema 4. Reglas

Índice

[Esquema](#)

[Ideas clave](#)

[4.1. ¿Cómo estudiar este tema?](#)

[4.2. Reglas de clasificación y reglas de asociación](#)

[4.3. Algoritmos de aprendizaje de reglas de clasificación](#)

[4.4. Algoritmos de aprendizaje de reglas de asociación](#)

[4.5. Aplicaciones y ejemplos de implementación](#)

[4.6. Referencias bibliográficas](#)

[A fondo](#)

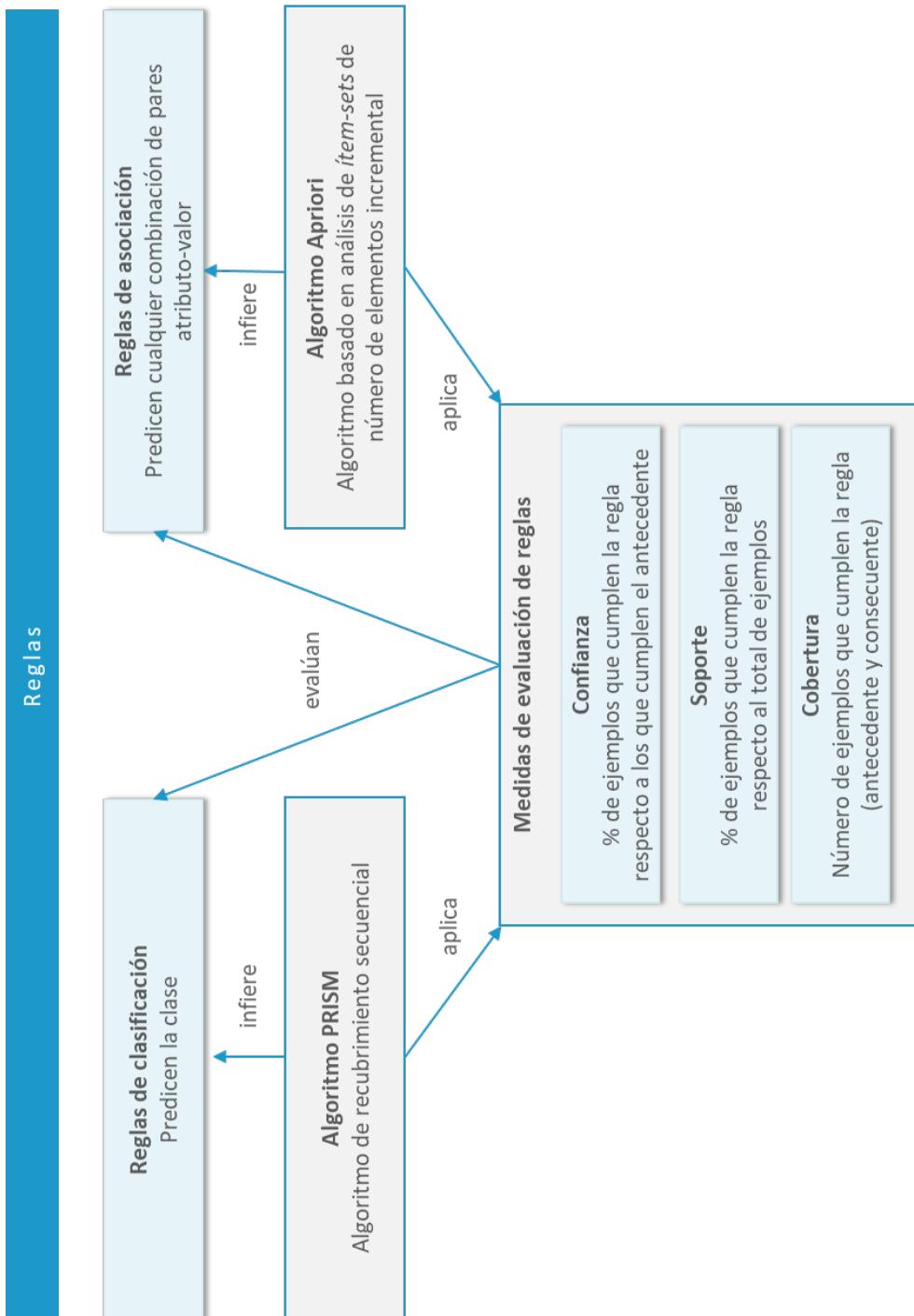
[Aprendizaje de reglas de clasificación y asociación con la herramienta Weka](#)

[La contribución de las reglas de asociación a la minería de datos](#)

[Curva ROC](#)

[Talleres de aprendizaje estadístico](#)

[Test](#)



4.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral, revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan. Es muy recomendable ver la lección magistral antes de realizar las actividades propuestas.

Al finalizar el estudio de este tema serás capaz de:

- ▶ Representar conocimiento mediante reglas de clasificación y de asociación.
- ▶ Aplicar algoritmos básicos de construcción de reglas para resolver problemas de aprendizaje.
- ▶ Identificar aplicaciones prácticas de las técnicas de aprendizaje de reglas de clasificación o asociación.

4.2. Reglas de clasificación y reglas de asociación

La representación del conocimiento, junto con el razonamiento, son unos de los aspectos más importantes de la inteligencia artificial. El objetivo principal de ambos es la búsqueda de una representación de este conocimiento que facilite la inferencia de nuevo conocimiento. Algunas de las características que una buena representación del conocimiento debe cumplir son (Ruiz, 2016):

- ▶ **Comprensible:** fácil de entender por humanos, soportando modularidad y jerarquía (por ejemplo, una Vespa es una moto, que a su vez es un vehículo).
- ▶ **Eficiente:** cumple con el objetivo perseguido utilizando el menor número de recursos posibles.
- ▶ **Adaptable:** facilita la modificación y actualización del conocimiento.
- ▶ **Consistente:** capaz de gestionar y eliminar conocimiento redundante o conflictivo (por ejemplo, «Juan ha comprado una barra de pan» y «la barra de pan ha sido comprada por Juan» implicarían redundancia).
- ▶ **Cobertura:** cubre la información en anchura y en profundidad permitiendo resolver conflictos y eliminar redundancias.
- ▶ **Completitud:** incluye toda la información necesaria.

Los sistemas de reglas son uno de los métodos más extendidos para representar conocimiento. Algunas ventajas de los sistemas de reglas:

- ▶ Modularidad.
- ▶ El conocimiento puede ser ampliado y modificado.
- ▶ Fáciles de entender.
- ▶ Separación entre control y conocimiento.
- ▶ Permiten explicar las decisiones.

De forma general, una representación del conocimiento en forma de reglas estará formada por dos partes:

- ▶ Un **antecedente** que incluye las **condiciones de aplicación** del conocimiento.
- ▶ Un **consecuente** en el que se indica la **conclusión, respuesta o acción** que ha de llevarse a cabo cuando se cumple el antecedente.

La sintaxis básica de una regla es:

SI <antecedente>

ENTONCES <consecuente>

Las reglas pueden presentar múltiples condiciones unidas por los operadores lógicos AND (conjunction) y OR (disyunción). Asimismo, el consecuente puede presentar múltiples conclusiones. De cualquier manera, es recomendable no mezclar en la misma regla conjunciones y disyunciones.

Por ejemplo, la siguiente regla presenta únicamente conjunciones:

SI <antecedente 1>

AND <antecedente 2>

AND <antecedente 3>

ENTONCES <consecuente 1>

<consecuente 2>

Los **antecedentes** de una regla incorporan dos partes: **un objeto y su valor, que van asociados por un operador**. Este operador puede ser matemático para dar un valor numérico al objeto o puede ser lingüístico, con lo que se asigna un valor lingüístico al objeto. Por ejemplo:

SI edad < 25

AND "años con carné de conducir" < 2

AND "número de siniestros previos" > 0

ENTONCES "riesgo de siniestro" es alto

En el ejemplo anterior, los objetos en el antecedente son numéricos mientras que el objeto del consecuente tiene un valor lingüístico. También se le puede asignar al consecuente, valores numéricos e incluso expresiones aritméticas como, por ejemplo:

SI “edad” < 25

AND “años con carné de conducir” < 2

AND “número de siniestros previos” > 0

ENTONCES “riesgo de siniestro” = “edad” * 1.5

La representación del conocimiento mediante reglas de clasificación es una alternativa a los árboles de decisión. De hecho, la representación mediante árboles de decisión se puede mapear a la representación mediante reglas de clasificación y viceversa.

A veces, es interesante trasformar un árbol en un conjunto de reglas como, por ejemplo, en los casos en que se tienen árboles grandes difíciles de interpretar. El antecedente de la regla contiene una serie de restricciones de valores que han de tener los atributos, mientras que el consecuente determina un valor de la clase.

En el árbol la serie de restricciones viene representada por las ramas mientras que el consecuente corresponde a la hoja.

Mientras que las reglas de clasificación predicen la clase, las reglas de asociación predicen valores de atributos, combinaciones de valores de atributos, o la propia clase.

Dado que el consecuente de una regla de asociación puede contener cualquier combinación de valores de atributos, la cantidad de reglas que habría que considerar es muy grande y, por tanto, algunas técnicas que se utilizan para obtener reglas de clasificación no se pueden utilizar para inducir reglas de asociación. Se ha de aplicar, por tanto, métodos que obtengan únicamente aquellas reglas de interés que se apliquen a un número de ejemplos grande y sean precisas.

El interés de las reglas de asociación es descubrir combinaciones de pares atributo-valor que ocurren con frecuencia en un conjunto de datos.

¿En qué casos se puede querer descubrir este tipo de combinaciones? Por ejemplo, en un comercio en línea puede ser muy interesante conocer los productos que los clientes adquieren conjunta y habitualmente con el fin de identificar clientes con patrones de compra similares y así poder predecir posibles compras futuras de estos clientes y realizar ofertas personalizadas.

Como previamente se ha indicado, es posible encontrar un gran número de reglas correspondiente al gran número de posibles combinaciones de valores de atributos. Por tanto, surge la necesidad de utilizar alguna **medida** que indique, por ejemplo, la probabilidad de que el consecuente de la regla se cumpla si se da el antecedente, determinando así la relevancia o interés de la regla.

Especificamente dos medidas populares que se suelen utilizar con el fin de evaluar una regla son:

Confianza

Soporte

La confianza es la probabilidad condicional de que dado un evento A se produzca un evento B.

$$\text{Confianza}(A \rightarrow B) = P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Al hablar de reglas, la confianza se puede expresar como el porcentaje de ejemplos que satisfacen el antecedente y consecuente de la regla entre aquellos que satisfacen el antecedente.

Por ejemplo, en la tienda en línea antes mencionada, mediante la medida de la confianza se pretende conocer la probabilidad de que la compra de un detergente para ropa conduzca a la compra de un suavizante para ropa.

El soporte se refiere al cociente del número de ejemplos que cumplen el antecedente y el consecuente de la regla entre el número total de ejemplos. En notación probabilística se puede expresar como:

$$\text{Soporte}(A \rightarrow B) = P(A \cup B)$$

Esta medida de soporte es interesante para detectar aquellas reglas que, aunque se cumplen en algún caso y aunque puedan tener alta confianza, no son relevantes porque cubren casos poco frecuentes. A continuación, se muestra con un ejemplo en qué consisten las reglas de asociación y cómo se aplican estas dos medidas. Específicamente, se van a utilizar los datos del conocido problema «Jugar al aire libre» mostrados en la Tabla 1.

Id	Ambiente	Temperatura	Humedad	Viento	Clase
E ₁	Soleado	Alta	Alta	Falso	No
E ₂	Soleado	Alta	Alta	Verdadero	No
E ₃	Nublado	Alta	Alta	Falso	Sí
E ₄	Lluvioso	Media	Alta	Falso	Sí
E ₅	Lluvioso	Baja	Normal	Falso	Sí
E ₆	Lluvioso	Baja	Normal	Verdadero	No
E ₇	Nublado	Baja	Normal	Verdadero	Sí
E ₈	Soleado	Media	Alta	Falso	No
E ₉	Soleado	Baja	Normal	Falso	Sí
E ₁₀	Lluvioso	Media	Normal	Falso	Sí
E ₁₁	Soleado	Media	Normal	Verdadero	Sí
E ₁₂	Nublado	Media	Alta	Verdadero	Sí
E ₁₃	Nublado	Alta	Normal	Falso	Sí
E ₁₄	Lluvioso	Media	Alta	Verdadero	no

Tabla 1. Datos del problema «Jugar al aire libre».

Dado que se trata de un problema con un bajo número de instancias a simple vista se puede extraer algunas reglas de asociación:

Regla 1: SI Ambiente es nublado

ENTONCES jugar = sí

Regla 2: SI Temperatura es baja

ENTONCES humedad es normal

Regla 3: SI Temperatura es media

ENTONCES humedad es alta

La regla de asociación 1 se puede considerar una regla de clasificación, ya que el consecuente se refiere al atributo de salida o a la clase. Sin embargo, las otras reglas relacionan atributos de entrada.

Se calculan a continuación los valores de confianza y soporte para estas reglas:

$$\text{Confianza}(\text{Regla 1:ambiente} = \text{nublado} \rightarrow \text{jugar} = \text{si}) = \frac{4}{4} = 1$$

$$\text{Confianza}(\text{Regla 2:temperatura} = \text{baja} \rightarrow \text{humedad} = \text{normal}) = \frac{4}{4} = 1$$

$$\text{Confianza}(\text{Regla 3:temperatura} = \text{media} \rightarrow \text{humedad} = \text{alta}) = \frac{4}{6} = 0.67$$

$$\text{Soporte}(\text{Regla 1:ambiente} = \text{nublado} \rightarrow \text{jugar} = \text{si}) = \frac{4}{14} = 0.29$$

$$\text{Soporte}(\text{Regla 2:temperatura} = \text{baja} \rightarrow \text{humedad} = \text{normal}) = \frac{4}{14} = 0.29$$

$$\text{Soporte}(\text{Regla 3:temperatura} = \text{media} \rightarrow \text{humedad} = \text{alta}) = \frac{4}{14} = 0.29$$

Para los casos de la regla 1 y la regla 2 la confianza es 1, ya que todos los ejemplos que satisfacen el antecedente de las reglas satisfacen también el consecuente. Sin embargo, para el caso de la regla 3, no siempre que se tiene una temperatura = media , se tiene una humedad = alta y, por lo tanto, la confianza de esa regla es menor.

De los seis ejemplos que cumplen el antecedente de la regla 3, hay dos que no cumplen el consecuente. Respecto a la medida de soporte, todas las reglas presentan la misma medida ya que antecedente y consecuente se da en el mismo número de ejemplos.

Este ejemplo contiene pocos datos, pero, cuando se manejan grandes cantidades de datos, el número de posibles asociaciones puede ser muy elevado. **Por lo tanto, se establecen valores mínimos de confianza y soporte para considerar cuáles de las reglas aprendidas son relevantes.**

Es importante tener en cuenta tanto la confianza como el soporte, ya que una regla puede tener una confianza con valor 1 y, sin embargo, representar una relación rara o inhabitual, con lo cual no es relevante.

Además, existen otras métricas que nos proporcionan información adicional a la que proporcionan la confianza o el soporte. Una de estas medidas es el *lift*, que nos indica la relación entre la probabilidad de que el consecuente de la regla se cumpla si se da el antecedente y la probabilidad de que se cumpla el consecuente de la regla.

Más formalmente:

$$lift(A,B) = \frac{P(B \vee A)}{P(B)} = \frac{confianza(A \rightarrow B)}{P(B)}$$

El *lift* mide la correlación entre la ocurrencia de un hecho A y un hecho B:

- ▶ Si $lift = 1$, entonces el hecho A es independiente del hecho B.
- ▶ Si $lift > 1$, entonces existe correlación entre A y B y, por lo tanto, A probablemente implica B. Nos indica que la regla es útil.
- ▶ Si $lift < 1$, entonces existe correlación negativa entre A y B y, por lo tanto, A y B se comportan de forma opuesta (A probablemente implica no B). Nos indica que la regla no es útil.

Generalmente, el aprendizaje de reglas de asociación comprenderá dos fases:

1 Encontrar aquellas reglas cuya frecuencia sea superior a un valor de soporte establecido.

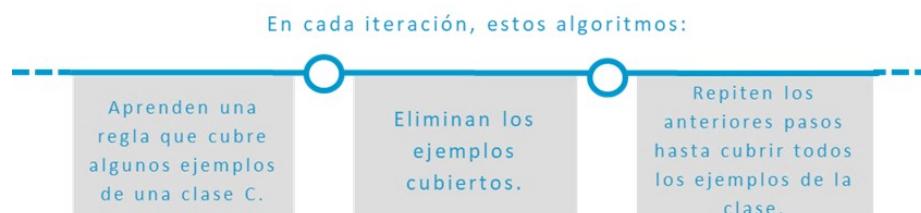
2 De las reglas extraídas en el paso 1, seleccionar aquellas cuya confianza es superior a un valor determinado.

En el apartado 4.4 de este tema se explica de forma detallada el algoritmo *apriori* como procedimiento para generar reglas de asociación.

4.3. Algoritmos de aprendizaje de reglas de clasificación

Como se ha visto con anterioridad, una forma posible de aprender reglas de clasificación es a través de la generación de un árbol de decisión en un primer paso, utilizando uno de los métodos explicados en el Tema 2, y, posteriormente, mapear el árbol generado a un conjunto de reglas equivalente, dando lugar a una regla por cada nodo hoja generado en el árbol.

En este tema se van a explicar los **algoritmos de recubrimiento secuencial** para el aprendizaje directo de conjuntos de reglas de clasificación.



Por tanto, se aprende una regla en cada iteración hasta alcanzar el conjunto final de reglas. La Figura 1 muestra un algoritmo básico de recubrimiento secuencial:

```
PROCEDIMIENTO Recubrimiento_secuencial (Clases, atributos, ejemplos)
COMIENZO
    Reglas ← {}
    Para cada clase C de Clases
        COMIENZO
            E ← Ejemplos
            Mientras (E contenga ejemplos de la clase C)
                COMIENZO
                    Regla ← AprenderUnaRegla (C, E, Atributos)
                    Reglas ← Reglas + {Regla}
                    E ← E - {Ejemplos de E clasificados correctamente por Regla}
                FIN
            FIN
            Devolver Reglas
        FIN
```

Figura 1. Algoritmo básico de recubrimiento secuencial.

El algoritmo de recubrimiento secuencial se basa en el uso iterativo de un procedimiento que seleccione una única regla de buena precisión, pero sin necesidad de que cubra todos los ejemplos positivos.

El algoritmo básico presentado en la Figura 1 podría incluir una condición adicional que evalúe la calidad de la regla aprendida para considerar o descartar esta regla.

Por otro lado, la Figura 2 muestra un **algoritmo básico de aprendizaje de una regla** que realiza una búsqueda codiciosa (*greedy*), sin retroceso, que **busca de lo general a lo específico**. Dado que se trata de un método codicioso, existe el riesgo de no encontrar la mejor regla. Esto no implica que no se pueda conseguir una precisión alta, aun presentando una cobertura incompleta.

La cobertura es otra medida utilizada para evaluar el interés de las

reglas y se define como el número de ejemplos que cumplen la regla (antecedente y consecuente).

```
PROCEDIMIENTO AprenderUnaRegla (Clase, Ejemplos, Atributos)
COMIENZO
    Regla ← regla con antecedente A vacío y con consecuente Clase
    MIENTRAS (regla cubre algún ejemplo negativo AND Atributos ≠ Ø)
        COMIENZO
            Restricciones ← {}
            Para cada atributo A no utilizado en la regla
                COMIENZO
                    Para cada valor v de A
                    COMIENZO
                        Restricciones ← Restricciones + {A=v}
                    FIN
                FIN
                Restriccion ← mejorRestriccion (Restricciones, regla)
                Regla ← añadir restricción al antecedente
                Atributos ← atributos - {atributo de Restriccion}
            FIN
        Devolver Regla
    FIN
```

Figura 2. Algoritmo básico de aprendizaje de una regla.

El **algoritmo PRISM** (Cendrowska, 1987) es uno de los algoritmos más simples de recubrimiento secuencial. Utiliza los algoritmos básicos de recubrimiento secuencial y de aprendizaje de una regla expuestos en la Figuras 1 y en la Figura 2, respectivamente.

De forma más específica, el procedimiento MejorRestricción utilizado en el algoritmo de aprendizaje de una regla que emplea **PRISM se basa en la medida de precisión** denominada **confianza**, tal y como se ha definido previamente.

Enfatizando en su definición, la confianza vendrá dada por el cociente entre el número de ejemplos que satisfacen antecedente y consecuente y el número de ejemplos que satisfacen solo el antecedente.

Para comprender mejor el aprendizaje de una regla utilizando el algoritmo PRISM se utilizará como ejemplo el conocido problema «Jugar al aire libre» (cuyos datos están contenidos en la Tabla 1). En este sentido, la Figura 3 ilustra en forma de árbol para facilitar su comprensión, el aprendizaje de una regla para el caso de la clase jugar=no

Desarrollemos entonces cómo se lleva a cabo la ejecución del algoritmo:

- ▶ En primer lugar, se escoge la regla más general, aquella que no tiene ninguna restricción en el antecedente.
- ▶ A continuación, se utiliza la medida de la confianza para seleccionar la mejor restricción de todas las restricciones posibles. El cálculo de esta medida, para el ejemplo seguido, se muestra en la Figura 3 entre paréntesis junto a las diferentes reglas. Se observa que el mejor valor de confianza es 0.60 y se da para el atributo ambiente y valor soleado. Por tanto, se escoge en este paso el par atributo ambiente y valor soleado.

- ▶ Se añade esta restricción a la regla, eliminando este atributo de la lista de atributos a considerar en la siguiente iteración.
- ▶ El siguiente nivel de árbol muestra la siguiente iteración del algoritmo. Se calcula la confianza para cada una de las posibles restricciones y se encuentra que la confianza es igual a 1 tanto para el atributo temperatura y valor alta como para el atributo humedad y valor alta.
- ▶ Cuando se obtiene un empate entre restricciones, se escoge la restricción de mayor cobertura. Por tanto, en este paso se escoge el par atributo humedad y valor alta porque tiene cobertura 3 frente a la cobertura 2 del par atributo temperatura y valor alta.
- ▶ Se añade la restricción seleccionada a la regla y se elimina este atributo de la lista de atributos a considerar en la siguiente iteración.
- ▶ En la siguiente iteración no se cumple la condición de que existan ejemplos negativos cubiertos por la regla. Entonces, el algoritmo devuelve la regla obtenida mediante las dos iteraciones descritas en los pasos anteriores. En concreto, la regla devuelta es la siguiente:

SI ambiente=soleado AND humedad=alta

ENTONCES jugar=no

ENTONCES jugar=no

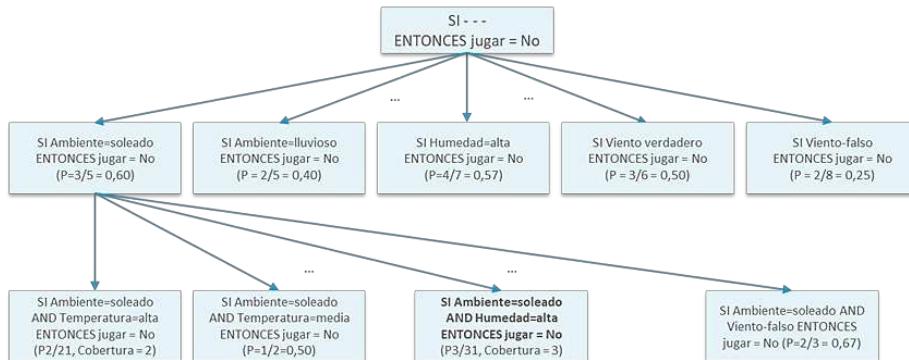


Figura 3. Representación gráfica del aprendizaje de una regla mediante el algoritmo básico para el ejemplo de la Tabla 1 y la clase jugar=no.

Mediante la llamada a este procedimiento de aprendizaje de una regla, PRISM va obteniendo incrementalmente el conjunto de reglas de clasificación para todas las clases existentes.

Otros algoritmos de reglas de clasificación

Existen otros algoritmos de reglas de clasificación además de PRISM, entre ellos:

- ▶ **OneRule y ZeroRule:** son los algoritmos más simples de reglas de clasificación para un conjunto de ejemplos (Holte, 1993). Estos algoritmos generan un árbol de decisión expresado mediante reglas de un solo nivel. Los algoritmos predicen simplemente la clase principal clasifican a través suyo.
- *ZeroRule* asigna un valor único de probabilidad a todas las instancias utilizando la media o la moda de la clase de salida, dependiendo de si trabaja con variables numéricas o nominales. La salida sería la clase más probable.
- *OneRule* utiliza particiones de un solo atributo y asigna valores a las instancias que tienen ese atributo, basándose en la media o la moda, al igual que hace *ZeroRule*, para asignar un valor único de probabilidad a todas las instancias de ese conjunto. De forma más sencilla, para cada valor de atributo el algoritmo calculará la

probabilidad para cada clase.

Tomando como errores las menores probabilidades se calculará el error total de cada atributo. Las reglas que se formen vendrán dadas por el atributo con un menor error total.

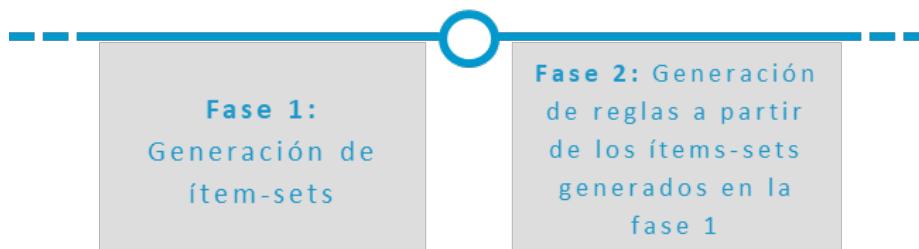
- ▶ **RIPPER** (*Repeated Incremental Pruning Produce Error Reduction*) (Cohen, 1995): algoritmo de reglas de clasificación basado en el algoritmo IREP (*Incremental Reduced Error Pruning*) (Fürnkranz & Widmer, 1994), basado, a su vez, en la técnica REP (*Reduced Error Pruning*) (Bagallo & Haussler, 1990) y en el algoritmo de aprendizaje de reglas *Separate-And-Conquer*.

4.4. Algoritmos de aprendizaje de reglas de asociación

Existen diversos algoritmos para generar reglas de asociación. En esta sección se profundiza en uno de los algoritmos más populares que se denomina **algoritmo apriori** (Agrawal et al, 1993).

El algoritmo apriori pretende generar ítem-sets que cumplan una cobertura mínima de manera eficiente. Un ítem es un par atributo-valor mientras que un ítem-set es un conjunto de pares atributo-valor. Un k-ítem-set es un conjunto de k pares atributo-valor. La cobertura de un ítem-sets se refiere al número de instancias que cumplen los valores en el ítem-set y va a determinar la cobertura de las reglas generadas a partir de dicho ítem-set.

El algoritmo a priori utiliza dos fases:



Mediante un ejemplo se explica a continuación el funcionamiento de este algoritmo. El primer paso es determinar una cobertura mínima, por ejemplo, 3. Seguidamente, se comienza a trabajar con ítem-sets de 1 par atributo-valor, escogiendo aquellos que cumplen como mínimo la cobertura escogida. Para el ejemplo del problema del tiempo de la Tabla 1 se tiene por tanto en la primera iteración del algoritmo los ítem sets de 1 elemento con cobertura superior o igual a 3, tal y como se muestra en la Tabla 2.

En este caso concreto todos los *ítem-sets* de 1 elemento cumplen la condición de la cobertura.

Ítem-sets de 1 elemento	Cobertura
Ambiente soleado	5
Ambiente nublado	4
Ambiente lluvioso	5
Temperatura=alta	4
Temperatura=media	6
Temperatura=baja	4
Humedad-alta	7
Humedad=normal	7
Viento=falso	8
Viento-verdadero	6
Jugar=sí	9
Jugar=no	5

Tabla 2. *Ítem-sets* de 1 elemento del problema «Jugar al aire libre».

En la siguiente iteración el algoritmo combina los *ítem-sets* encontrados en la primera iteración para generar *ítem-sets* de 2 elementos que cumplan la condición de cobertura igual o superior a 3. Estos *ítem-sets* se incluyen en la Tabla 3.

Ítem-sets de 2 elementos	Cobertura
Ambiente lluvioso, temperatura=media	3
Ambiente lluvioso, humedad=normal	3
Ambiente soleado, humedad=alta	3
Ambiente lluvioso, viento=falso	3
Ambiente soleado, viento=falso	3
Ambiente lluvioso, jugar=sí	3
Ambiente=nublado, jugar=sí	4
Ambiente=soleado, jugar=no	3
Temperatura=alta, humedad=alta	3
Temperatura=baja, humedad=normal	4
Temperatura=media, humedad=alta	4
Temperatura=alta, viento=falso	3
Temperatura=media, viento=falso	3
Temperatura=media, viento verdadero	3
Temperatura=baja, jugar=sí	3
Temperatura=media, jugar=sí	4
Humedad=alta, viento=falso	4
Humedad=alta, viento verdadero	3
Humedad=normal, viento=falso	4
Humedad=normal, viento verdadero	3
Humedad=normal, jugar=sí	6
Humedad=alta, jugar=sí	3
Humedad=alta, jugar=no	4
Viento=falso, jugar=sí	6
Viento verdadero, jugar=sí	3
Viento-verdadero, jugar=no	3

Tabla 3. Ítem-sets de 2 elementos del problema «Jugar al aire libre».

En la siguiente iteración se parte de los ítem-sets encontrados de 2 elementos para generar ítem-sets de 3 elementos. Por ejemplo, si se consideran las dos primeras entradas de la Tabla 3 se puede generar el siguiente ítem-set de 3 elementos:

Ambiente=lluvioso, temperatura=media, humedad=normal

Existe un único ejemplo que cumple este ítem-set, luego, al no cumplir la cobertura mínima, no se añade a la tabla de ítem-sets de 3 elementos. Así, se procede combinando el resto de las entradas e incluyendo en la tabla aquellas combinaciones que cumplen la condición de cobertura mínima, obteniendo los *ítem-sets* presentes en la Tabla 4.

Ítem-sets de 3 elementos	Cobertura
Ambiente soleado, humedad alta, jugar=no	3
Ambiente lluvioso, viento=falso, jugar=si	3
Temperatura=baja, humedad=normal, jugar=si	3
Humedad=normal, viento=falso, jugar=si	4

Tabla 4. Ítem-sets de 3 elementos del problema «Jugar al aire libre».

El siguiente paso es obtener *ítem-sets* de 4 elementos que cumplan la condición de cobertura de 3 a partir de aquellos presentes en la Tabla 4. En este paso ya no se obtiene ningún *ítem-set* que cumpla dicha condición.

Una vez obtenidos los *ítem-sets*, se procede a la siguiente fase del algoritmo que consiste en generar las reglas de asociación a partir de los *ítem-sets* encontrados de 3 y 2 elementos. De las reglas generadas, se descartan aquellas reglas que no superan un mínimo valor de confianza.

Por ejemplo, se establece un valor de confianza de 0.9. Para el ítem-set (Ambiente=soleado, humedad=alta, jugar=no) se pueden generar las reglas siguientes:

SI ambiente = soleado

ENTONCES humedad = alta AND jugar = no (P=3/5=0,6)

ST ambiente = soleado AND humedad = alta

ENTONCES jugar = no (P=3/3=1)

SI ambiente = soleado AND jugar = no

ENTONCES humedad = alta (P=3/3=1)

SI humedad = alta

ENTONCES jugar = no AND ambiente = soleado (P=3/7=0.43)

SI humedad = alta AND jugar = no

ENTONCES ambiente = soleado (P=3/4=0.75)

SI jugar = no

ENTONCES humedad = alta AND ambiente = soleado (P=3/5=0.6)

Entre paréntesis, tras cada regla, se indica el valor de la confianza. Únicamente hay dos reglas que superan el valor de confianza mínimo establecido de 0.9 y, por tanto, son las reglas que serán consideradas en el conjunto de reglas final:

SI ambiente = soleado AND humedad = alta

ENTONCES jugar = no (P=3/3=1)

SI ambiente = soleado AND jugar = no

ENTONCES humedad = alta (P=3/3=1)

De la misma manera se procederá a generar las posibles reglas por cada ítem set encontrado en la primera fase, seleccionando únicamente aquellas con las que se obtiene un valor de confianza superior a 0.9. Por ejemplo, otras reglas que se generarían serían:

SI humedad = normal AND viento = falso

ENTONCES jugar = si (P=4/4=1)

SI ambiente = lluvioso AND viento falso

ENTONCES jugar = si (P=3/3=1)

SI temperatura = baja

ENTONCES humedad = normal (P=4/4=1)

De este algoritmo se dice que genera las reglas eficientemente, puesto que en cada fase solo tiene en cuenta un subconjunto de posibles *ítem-sets*, aquellos considerados en la iteración previa por superar una cobertura mínima, y un *ítem-set* de k elementos no va a cumplir la norma de la mínima cobertura a no ser que los *ítem-sets* de $k-1$ elementos candidatos para combinar cumplan también esa condición.

De cualquier manera, para conjuntos de datos grandes este algoritmo puede suponer una alta carga computacional dependiendo de la cobertura especificada.

Por último, se debe remarcar que no siempre una regla de asociación con alta confianza y soporte resulta útil. Por ejemplo, en el caso del ejemplo de la tienda en línea mencionado previamente, si se da el hecho habitual de que los clientes que compran detergente de la lavadora también compran suavizante, esta información puede resultar poco útil a efectos de *marketing*, no siendo necesario promocionar ninguno de los dos productos.

Sin embargo, sí puede resultar útil cuando se encuentra el hecho de que asociando la venta de dos productos se vende más de un producto, o cuando se da el hecho opuesto en el que se encuentra que dos productos, si se asocian, compiten entre sí, con lo cual la regla que les asocia tiene una confianza baja.

Otros algoritmos de reglas de asociación

Existen otros algoritmos de reglas de asociación además de apriori, entre ellos:

- ▶ Part.
- ▶ FP-Growth y TD-FP-Growth.
- ▶ ECLAT (Equivalent CLAss Transformation)

PART

Algoritmo que obtiene reglas de asociación utilizando el algoritmo de árboles de decisión C4.5. PART no necesita realizar una optimización global.

FP-Growth y TD-FP-Growth

Para reducir el coste que la generación de los conjuntos de ítem-sets implica, Han *et al.* (2000) propusieron el algoritmo *FP-Growth*. Al igual que apriori, este el algoritmo permite obtener reglas de asociación a partir de ítem-sets frecuentes, pero sin generar las diferentes reglas candidatas para cada iteración de k elementos. Este algoritmo propone una nueva estructura de patrones frecuentes (*FP – Frequent Patterns*), extendida del árbol de prefijos, que permite almacenar toda la información de las transacciones comprimida.

La eficiencia del algoritmo se logra gracias a tres técnicas: en primer lugar, la compresión de la base de datos; en segundo lugar, limitando la generación de ítem-sets; en tercer lugar, utilizando un método de «*divide y vencerás*» para descomponer la tarea de búsqueda de patrones en varias bases de datos condicionales, reduciendo drásticamente el espacio de búsqueda (Han *et al.*, 2000). Los resultados obtenidos del análisis de cada base de datos se concatenarán en el paso final. La

versión *TD-FP-Growth* cambia el orden de búsqueda de arriba hacia abajo, en oposición al orden de abajo hacia arriba del *FP-Growth*, lo cual ahorra espacio y tiempo (Wang *et al.*, 2002).

ECLAT (*Equivalent CLAss Transformation*):

Algoritmo basado en la búsqueda de *ítem-sets* frecuentes. La diferencia principal con a priori es que éste almacena las transacciones de forma horizontal (elementos que forman una transacción en la misma línea), mientras que ECLAT analiza los datos de forma vertical, conteniendo cada línea un ítem y las transacciones en las que aparece ese ítem (Zaki *et al.*, 1997).

4.5. Aplicaciones y ejemplos de implementación

Algunas aplicaciones de las reglas de asociación en la literatura

El interés de las reglas de asociación es descubrir combinaciones de pares atributo-valor que ocurren frecuentemente en un conjunto de datos. De hecho, también son conocidas como técnicas de patrones de búsqueda (*pattern search*) o *association rule learning* (Fournier-Viger *et al.*, 2017). Entre sus posibles aplicaciones, una de las más interesantes es analizar los carritos de la compra, tanto en los supermercados físicos, para saber cómo distribuir los productos en las estanterías, o en las tiendas *online*.

Es decir, en una tienda *online* puede ser muy interesante conocer los productos que los clientes compran juntos, con el fin de identificar a los clientes con patrones de compra similares, y así poder predecir posibles compras futuras de estos clientes y hacer ofertas personalizadas (por ejemplo, los padres que compran pañales también compran fórmula, así como cerveza cuando no pueden salir a tomar una copa).

Otras aplicaciones similares en las que son útiles incluyen el análisis de patrones de navegación en la web. También se han empleado en aplicaciones médicas para analizar las relaciones entre la recuperación después de operaciones y la posible cronificación de secuelas (Hui *et al.* 2014).

Las reglas de clasificación en la literatura y otros algoritmos clasificadores

Tal y como se ha explicado, los árboles de decisión están muy relacionados con las técnicas de reglas de clasificación, como el algoritmo PRISM (Liu, Gegov y Cocea, 2016). Sin embargo, además de la regresión logística, los árboles de decisión clasificadores y las reglas de clasificación, **existen otros algoritmos clasificadores, incluyendo k-NN, los clasificadores Naïve Bayes y los SVM**, además de, por supuesto, las redes neuronales, con las cuales podemos resolver problemas tanto de regresión como de la clasificación, y que veremos en los siguientes temas.

El **algoritmo k-NN** (*k nearest neighbors* o *k vecinos más cercanos*) se incluye, al igual que los árboles de decisión, en lo que se conoce como «aprendizaje no paramétrico» (Kanj *et al.* 2016). Es decir, a diferencia de algoritmos de «aprendizaje paramétrico», este tipo de método no requiere una función paramétrica predefinida $\mathbf{Y} = f(\mathbf{X})$. Esto hace que este tipo de algoritmo sea adecuado para aquellas situaciones en las que la relación entre \mathbf{X} e \mathbf{Y} es demasiado compleja para ser expresada como un modelo lineal. En el algoritmo k-NN, cada instancia se representa como un vector y para clasificar o hacer una predicción sobre un dato de entrada, se toman los k más cercanos y se calcula la media de sus valores, si estamos trabajando con datos continuos, como el valor estimado de una casa, o su moda, si estamos trabajando con datos categóricos, como la determinación de la raza de un perro. La selección del k se hace por validación cruzada, eligiendo el k que tiene el menor error, en promedio, a lo largo de las diferentes iteraciones. Este algoritmo se utiliza como método de clasificación, como detección de fraudes (Hossain y Uddin, 2018), como método de regresión, como predicción del precio de la vivienda (Nawaz *et al.*, 2019), o para imputar los datos de entrenamiento que faltan, imputando el promedio o el modo de los vecinos en lugar de un valor que falta (Liu *et al.*, 2016).

Los clasificadores **Naïve Bayes** (*clásificadores Bayesianos ingenuos*) son en realidad uno de los casos más sencillos de redes bayesianas o redes de creencias (Gupta *et al.*, 2019; Li, Corchado y Prieto, 2016). Las redes bayesianas son redes acíclicas dirigidas en las que cada nodo representa un estado o condición y cada arco entre dos nodos representa la probabilidad de que, dado el estado o condición del nodo fuente, se produzca el estado del nodo destino, teniendo en cuenta el teorema de Bayes:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

La particularidad de los clasificadores ingenuos de Bayes es considerar una fuerte independencia entre las diferentes características. Este tipo de clasificadores han sido ampliamente utilizados como filtros antispam.

Para ello, tienen en cuenta la frecuencia con la que las diferentes palabras aparecen en los correos electrónicos deseados y en los correos spam. En este sentido, con el tiempo al final de los correos spam se introdujeron una serie de palabras comunes en los correos deseados, atacando así al clasificador Bayes ingenuo. Esto se conoce como *envenenamiento bayesiano*. Así, desde 2010, se utilizan otros tipos de algoritmos para el filtrado antispam (Bhowmick y Hazarika, 2018).

Uno de los métodos utilizados, de hecho, actualmente para el filtrado antispam es el de las **máquinas de soporte vectorial (SVM – Support Vector Machine** (Rana *et al.*, 2018). Las SVM son clasificadores basados en la idea de buscar dos líneas entre los puntos de datos de entrenamiento bidimensionales y con el máximo margen posible entre estas líneas (es decir, un problema de optimización, normalmente utilizando la optimización Lagrangiana). Cuando nuestros datos no son bidimensionales, se buscan hiperplanos en lugar de líneas. Si no es posible dibujar tal línea o hiperplano, tenemos que suavizar la condición de separación añadiendo una función de coste o pérdida, o aumentando el número de dimensiones de los datos (con términos como x^2 , x^3 o incluso $\cos(x)$). Las aplicaciones de las SVM incluyen la clasificación de imágenes (aunque este tipo de aplicación se lleva a cabo actualmente con redes neuronales pre-entrenadas, más adecuadas para imágenes y vídeo), el análisis de sentimientos, la clasificación de texto y contenido en redes sociales (Dang *et al.*, 2016) o la detección de *outliers* (Liu, White y Newell, 2018).

Ejemplos de implementación

En este ejemplo, vamos a aplicar el algoritmo *apriori* utilizando en esta ocasión un pequeño *dataset* en **Kaggle**, proporcionado por Shazad Udwadia (llamado «Grocery Store Data Set») con licencia de dominio público CC0 – Creative Commons 0) y en el

cual se incluyen 20 transacciones de cestas de la compra en una tienda de comestibles de diferentes productos (11 ítems posibles en total). Es un *dataset* muy pequeño, pero nos sirve para efectos ilustrativos.

En el mismo Kaggle podemos ver información sobre el contenido de los datos y algunos detalles estadísticos. Podemos descargar el *dataset* desde el siguiente enlace. Nos solicitará registrarnos si no tenemos ya una cuenta en Kaggle. Para ello podemos utilizar una cuenta de Google, correo electrónico, etc., pero no tiene coste alguno. Disponible en: <https://www.kaggle.com/shazadudwadia/supermarket>

Aunque Kaggle nos permite utilizar *notebooks online* bajo Python o R para trabajar con los *dataset* que comparten otros usuarios sin necesidad de disponer de un sistema Python *offline*, por el momento vamos a seguir trabajando *offline*, para así también seguir aprendiendo a gestionar las librerías de terceros.

En el enlace indicado nos mostrará, en realidad, información sobre el *dataset*, de forma similar a OpenML, y nos permitirá descargarlo a través del botón «*Download*» (o incluso crear un nuevo *notebook* basado en dichos datos). De esta forma, descargaremos un archivo “supermarket.zip”. Descomprimiéndolo obtendremos el archivo “GroceryStoreDataSet.csv” con el que vamos a trabajar. Si abrimos su contenido con cualquier editor de texto, como el propio Visual Studio Code, veremos que no tiene *header* y que cada línea representa una compra de un cliente, conteniendo la lista de artículos adquiridos.

```
"MILK,BREAD,BISCUIT"
"BREAD,MILK,BISCUIT,CORNFLAKES"
"BREAD, TEA, BOURNVITA"
"JAM, MAGGI, BREAD, MILK"
"MAGGI, TEA, BISCUIT"
"BREAD, TEA, BOURNVITA"
"MAGGI, TEA, CORNFLAKES"
"MAGGI, BREAD, TEA, BISCUIT"
"JAM, MAGGI, BREAD, TEA"
"BREAD, MILK"
"COFFEE, COCK, BISCUIT, CORNFLAKES"
"COFFEE, COCK, BISCUIT, CORNFLAKES"
"COFFEE, SUGER, BOURNVITA"
"BREAD, COFFEE, COCK"
"BREAD, SUGER, BISCUIT"
"COFFEE, SUGER, CORNFLAKES"
"BREAD, SUGER, BOURNVITA"
"BREAD, COFFEE, SUGER"
"BREAD, COFFEE, SUGER"
"TEA, MILK, COFFEE, CORNFLAKES"
```

Como CSV, en realidad, solo existe una única columna, de modo que habrá que realizar algo de tratamiento en los datos.

Sin embargo, antes de nada, vamos a instalar una nueva librería que no hemos usado hasta ahora, mlxtend, utilizando nuestro gestor de paquetes, por ejemplo:

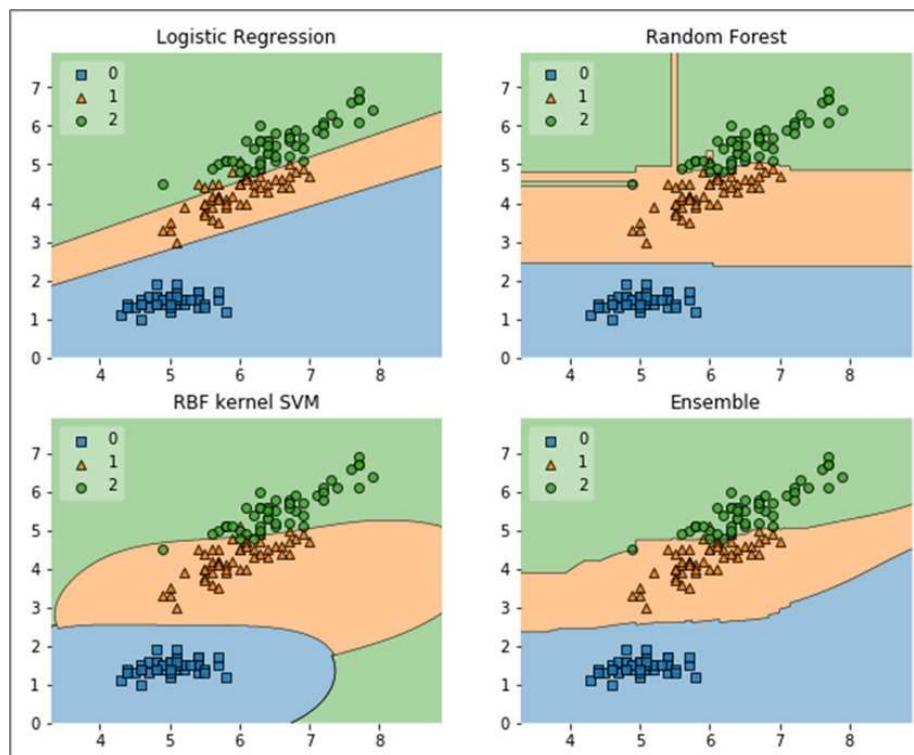
```
PS C:\Users\xxx> pip install mlxtend
```



Fuente: <http://rasbt.github.io/mlxtend/>

MLxtend (*Machine Learning extensions*) incluye extensiones útiles para realizar técnicas de *machine learning*, incluyendo, por ejemplo, el algoritmo *apriori*. Además, incluye interesantes ayudas para utilizar gráficos como regiones de decisión, como

se puede ver a continuación, que podemos utilizar en otro momento para comparar algoritmos de clasificación (aunque no es el caso que nos ocupa, pues ahora no estamos trabajando con clasificadores, que se encontrarían dentro del aprendizaje supervisado, sino con reglas de asociación / búsqueda de patrones, que se encontrarían dentro del aprendizaje no supervisado).



Una vez instalado el módulo, continuemos con el ejemplo que nos ocupa. En primer lugar, utilicemos este código para ver el contenido de los datos una vez cargados:

```
# Librerías necesarias
import pandas as pd
# recordad emplear aquí la ruta absoluta o relativa hasta nuestro dataset descargado
# en función de en qué ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products'],
                 header=None)

print("----df----")
print(df)
print("----df.columns---")
print(df.columns)
print("----df.values---")
print(df.values)
```

Resultando la siguiente salida:

```
---df---
products

0      MILK,BREAD,BISCUIT

1      BREAD,MILK,BISCUIT,CORNFLAKES

2      BREAD,TEA,BOURNVITA

3      JAM,MAGGI,BREAD,MILK

4      MAGGI,TEA,BISCUIT

5      BREAD,TEA,BOURNVITA

6      MAGGI,TEA,CORNFLAKES

7      MAGGI,BREAD,TEA,BISCUIT

8      JAM,MAGGI,BREAD,TEA

9      BREAD,MILK

10     COFFEE,COCK,BISCUIT,CORNFLAKES

11     COFFEE,COCK,BISCUIT,CORNFLAKES

12     COFFEE,SUGER,BOURNVITA

13     BREAD,COFFEE,COCK

14     BREAD,SUGER,BISCUIT

15     COFFEE,SUGER,CORNFLAKES

16     BREAD,SUGER,BOURNVITA
```

17 BREAD, COFFEE, SUGER

18 BREAD, COFFEE, SUGER

19 TEA, MILK, COFFEE, CORNFLAKES

---df.columns---

Index(['products'], dtype='object')

---df.values---

[['MILK,BREAD,BISCUIT']]

['BREAD,MILK,BISCUIT,CORNFLAKES']

['BREAD,TEA,BOURNVITA']

['JAM,MAGGI,BREAD,MILK']

['MAGGI,TEA,BISCUIT']

['BREAD,TEA,BOURNVITA']

['MAGGI,TEA,CORNFLAKES']

['MAGGI,BREAD,TEA,BISCUIT']

['JAM,MAGGI,BREAD,TEA']

['BREAD,MILK']

['COFFEE,COCK,BISCUIT,CORNFLAKES']

['COFFEE,COCK,BISCUIT,CORNFLAKES']

['COFFEE,SUGER,BOURNVITA']

['BREAD,COFFEE,COCK']

```
[ 'BREAD,SUGER,BISCUIT' ]  
  
[ 'COFFEE,SUGER,CORNFLAKES' ]  
  
[ 'BREAD,SUGER,BOURNVITA' ]  
  
[ 'BREAD,COFFEE,SUGER' ]  
  
[ 'BREAD,COFFEE,SUGER' ]  
  
[ 'TEA,MILK,COFFEE,CORNFLAKES' ]]
```

Preprocesamos los datos para que puedan ser usados por mlxtend, primero separando los elementos por las comas y posteriormente utilizando el objeto TransactionEncoder .

```
# Librerías necesarias  
import pandas as pd  
from mlxtend.preprocessing import TransactionEncoder  
  
# recordad emplear aquí la ruta absoluta o relativa hasta nuestro dataset descargado  
# en función de en que ruta estamos ejecutando nuestro script  
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products'], header=None)  
  
# preprocesamos los datos usando las comas como separadores  
data = list(df["products"].apply(lambda x:x.split(',')))  
print("----data----")  
print(data)  
print()  
  
# convertimos los datos a un formato que entienda mlxtend  
te = TransactionEncoder()  
te_data = te.fit(data).transform(data)  
df = pd.DataFrame(te_data,columns=te.columns_ )  
print("----df.head()----")  
print(df.head())  
print()
```

Obteniendo la salida (hemos reducido la segunda salida para que se vea mejor como una tabla):

```
----data----
```

```
[['MILK', 'BREAD', 'BISCUIT'],  
 ['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'],  
 ['BREAD', 'TEA', 'BOURNVITA'],  
 ['JAM', 'MAGGI', 'BREAD', 'MILK'],  
 ['MAGGI', 'TEA', 'BISCUIT'],  
 ['BREAD', 'TEA', 'BOURNVITA'],  
 ['MAGGI', 'TEA', 'CORNFLAKES'],  
 ['MAGGI', 'BREAD', 'TEA', 'BISCUIT'],  
 ['JAM', 'MAGGI', 'BREAD', 'TEA'],  
 ['BREAD', 'MILK'],  
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],  
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],  
 ['COFFEE', 'SUGER', 'BOURNVITA'],  
 ['BREAD', 'COFFEE', 'COCK'],  
 ['BREAD', 'SUGER', 'BISCUIT'],  
 ['COFFEE', 'SUGER', 'CORNFLAKES'],  
 ['BREAD', 'SUGER', 'BOURNVITA'],  
 ['BREAD', 'COFFEE', 'SUGER'],  
 ['BREAD', 'COFFEE', 'SUGER'],  
 ['TEA', 'MILK', 'COFFEE', 'CORNFLAKES']]
```

```
---df.head()---
```

	BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI	MILK
	SUGER		TEA						
0	True		False	True	False	False	False	False	True
	False	False							
1	True		False	True	False	False	True	False	True
	False	False							
2	False		True	True	False	False	False	False	False
	False	True							
3	False		False	True	False	False	False	True	True
	False	False							
4	True		False	False	False	False	False	False	True
	False	True							

Aplicaremos ahora el algoritmo *apriori* y veamos el resultado de las asociaciones ordenadas de forma descendente, en base a su soporte:

```
# Librerías necesarias
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# recordad emplear aquí la ruta absoluta o relativa hasta nuestro dataset descargado
# en función de en que ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products'],
, header=None)

# preprocesamos los datos usando las comas como separadores
data = list(df["products"].apply(lambda x:x.split(',')))

# convertimos los datos a un formato que entienda mlxtend
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data,columns=te.columns_)

# procedemos a aplicar el algoritmo apriori
df1 = apriori(df, min_support=0.01, use_colnames=True)
df1 = df1.sort_values(by="support", ascending=False)
print(df1)
```

	support	itemsets
2	0.65	(BREAD)
4	0.40	(COFFEE)
0	0.35	(BISCUIT)
10	0.35	(TEA)
5	0.30	(CORNFLAKES)
..
55	0.05	(CORNFLAKES, MILK, BISCUIT)
57	0.05	(SUGER, BREAD, BOURNVITA)
17	0.05	(SUGER, BISCUIT)

37 0.05 (CORNFLAKES, MAGGI)

82 0.05 (COFFEE, MILK, TEA, CORNFLAKES)

Por último, creemos una función para predecir el siguiente elemento que escogerá con mayor probabilidad un cliente a su cesta en función del estado actual de elementos en la misma:

```
# Librerías necesarias
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# recordad emplear aquí la ruta absoluta o relativa hasta nuestro dataset descargado
# en función de en qué ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products'], header=None)

# procesamos los datos usando las comas como separadores
data = list(df["products"].apply(lambda x:x.split(',')))

# convertimos los datos a un formato que entienda mlxtend
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data,columns=te.columns_)

# procedemos a aplicar el algoritmo apriori
df1 = apriori(df, min_support=0.01, use_colnames=True)
df1 = df1.sort_values(by="support", ascending=False)

# función auxiliar
def check_in_list(ruleitem, basketitem):
    ret = all(t in list(ruleitem) for t in basketitem)
    return ret
```

```
# función para predecir el próximo elemento más probable que introducirá el cliente en la cesta
def next_item(basketitems):
    if basketitems is None:
        return df1["itemsets"][0]
    max_len_apriori=len(basketitems) + 1
    # lista de solo 1 item:
    df2 = apriori(df,min_support=0.01, max_len=max_len_apriori,
use_colnames=True)
    df2 = df2[df2["itemsets"].apply(len) > max_len_apriori-1]
    df2["check_in_list"] = df2["itemsets"].apply(check_in_list,
args=(basketitems,))
    df2 = df2[df2["check_in_list"] == True]
    df2 = df2.sort_values(by="support",ascending=False)

    if(len(df2) > 0):
        return list(df2["itemsets"])[0]

# Probamos ahora a predecir el próximo valor en la cesta en función de los items actuales en ella:

# partimos de una cesta vacía
item = next_item(None)
print(list(item))
#>['BISCUIT']

# añadimos ese elemento a la cesta y predecimos el segundo:
item = next_item(["BISCUIT"])
print(list(item))
#>['BREAD', 'BISCUIT']

# añadimos ese elemento a la cesta y predecimos el tercero:
item = next_item(["BREAD", "BISCUIT"])
print(list(item))
#>['MILK', 'BREAD', 'BISCUIT']
```

Obviamente podemos elegir un estado de la cesta aleatorio, no hay necesidad de seguir esta secuencia.

4.6. Referencias bibliográficas

Bagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5(1), 71-99. Disponible en <https://doi.org/10.1007/BF00115895>

Bhowmick, A. & Hazarika, S. M. (2018). E-mail spam filtering: a review of techniques and trends. In *Advances in Electronics, Communication and Computing* (pp. 583-590). Singapore: Springer.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4), 349-370. Disponible en: [https://doi.org/10.1016/S0020-7373\(87\)80003-2](https://doi.org/10.1016/S0020-7373(87)80003-2)

Cohen, W. W. (1995). *Fast Effective Rule Induction*. In Proceedings of the Twelfth International Conference on Machine Learning, 115–123.

Dang, N. C., De la Prieta, F., Corchado, J. M. & Moreno, M. N. (2016, June). *Framework for retrieving relevant contents related to fashion from online social network data*. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 335-347). Cham: Springer.

Fournier-Viger, P., Lin, J. C. W., Vo, B., Chi, T. T., Zhang, J. & Le, H. B. (2017). A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(4), e1207.

Fürnkranz, J. & Widmer, G. (1994). Incremental Reduced Error Pruning. En W. W. Cohen & H. Hirsh (Eds.), *Machine Learning Proceedings 1994* (pp. 70-77). Morgan Kaufmann. Disponible en <https://doi.org/10.1016/B978-1-55860-335-6.50017-9>

Gupta, A., Slater, J. J., Boyne, D., Mitsakakis, N., Bélieau, A., Druzdzel, M. J. & Arora, P. (2019). Probabilistic Graphical Modeling for Estimating Risk of Coronary Artery Disease: Applications of a Flexible Machine-Learning Method. *Medical*

Decision Making, 39(8), 1032-1044.

Han, J., Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation. Association for Computing Machinery. Disponible en <https://doi.org/10.1145/335191.335372>

Holte, R. C. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11(1), 63-90. Disponible en <https://doi.org/10.1023/A:1022631118932>

Hossain, M. A. & Uddin, M. N. (2018, October). *A Differentiate Analysis for Credit Card Fraud Detection*. In 2018 International Conference on Innovations in Science, Engineering and Technology (ICISET) (pp. 328-333). IEEE.

Hui, L., Shih, C. C., Keh, H. C., Yu, P. Y., Cheng, Y. C. & Huang, N. C. (2014, May). *The application of association rules in clinical disease: the relationship between recovery after operation of endovascular aneurysm repairing and chronic*. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 712-721). Cham: Springer.

Kanj, S., Abdallah, F., Denoeux, T. & Tout, K. (2016). Editing training data for multi-label classification with the k-nearest neighbor rule. *Pattern Analysis and Applications*, 19(1), 145-161.

Li, T., Corchado, J. M., & Prieto, J. (2016). *Convergence of distributed flooding and its application for distributed Bayesian filtering*. IEEE Transactions on Signal and Information Processing over Networks, 3(3), 580-591.

Liu, C., White, M. & Newell, G. (2018). Detecting outliers in species distribution data. *Journal of Biogeography*, 45(1), 164-176.

Liu, H., Gegov, A. & Cocea, M. (2016). Rule-based systems: a granular computing perspective. *Granular Computing*, 1(4), 259-274.

Liu, Z. G., Pan, Q., Dezert, J. & Martin, A. (2016). Adaptive imputation of missing values for incomplete pattern classification. *Pattern Recognition*, 52, 85-95.

Nawaz, M., Javaid, N., Mangla, F. U., Munir, M., Ihsan, F., Javaid, A. & Asif, M. (2019, July). *An Approximate Forecasting of Electricity Load and Price of a Smart Home Using Nearest Neighbor*. In Conference on Complex, Intelligent, and Software Intensive Systems (pp. 521-533). Cham: Springer.

Rana, S. P., Prieto, J., Dey, M., Dudley, S. & Corchado, J. M. (2018). A Self Regulating and Crowdsourced Indoor Positioning System through Wi-Fi Fingerprinting for Multi Storey Building. *Sensors*, 18(11), 3766.

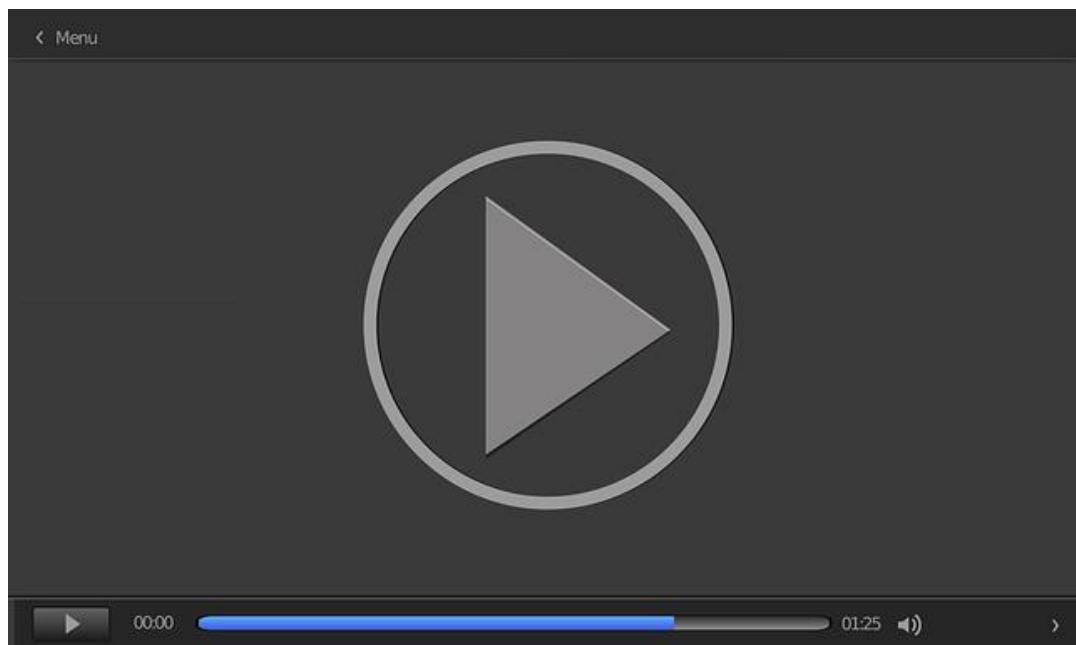
Ruiz, A. (2016, mayo 23). Representación del Conocimiento [Educación]. Disponible en https://es.slideshare.net/Alva_Ruiz/representacion-del-conocimiento-62308123

Wang, K., Tang, L., Han, J., & Liu, J. (2002). Top Down FP-Growth for Association Rule Mining. En M.S. Chen, P. S. Yu, & B. Liu (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 334-340). Springer. Disponible en https://doi.org/10.1007/3-540-47887-6_34

Zaki, M. J., Parthaasarathy, S., Ogihsara, M. & Li, W. (1997). *New Algorithms for Fast Discovery of Association Rules*. In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, 283–286.

Aprendizaje de reglas de clasificación y asociación con la herramienta Weka

En esta lección magistral se mostrará cómo se puede utilizar Weka para obtener reglas de clasificación y de asociación a partir de un conjunto de datos empleando los algoritmos explicados en este tema.



04. Aprendizaje de reglas de clasificación y asociación con la herramienta Weka

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=b26c847e-1c49-4d8c-a7d8-aff800f9df21>

La contribución de las reglas de asociación a la minería de datos

De Moya Amaris, M.E. & Rodríguez Rodríguez, J.E. (2003). La contribución de las reglas de asociación a la minería de datos. *Tecnura*, 7(13), 94-109.
<http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/article/view/6175>.

El artículo comienza con una introducción a la minería de datos y a las reglas de asociación, incluyendo ejemplos ilustrativos. Explica cómo se generan las reglas de asociación mediante el algoritmo *apriori*, incluyendo un pseudocódigo del mismo. Además, profundiza en el tema abordando las reglas de asociación multinivel.

Curva ROC

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:
http://es.wikipedia.org/wiki/Curva_ROC.

Uno de los datos de evaluación que muestra Weka en las salidas de los algoritmos de clasificación es el área ROC. El artículo disponible en Wikipedia sobre la curva ROC explica concisa y claramente en qué consiste esta curva y para qué se utiliza. Además, incluye muchas referencias y enlaces de interés relacionados.

Talleres de aprendizaje estadístico

Accede a los talleres desde el aula virtual o a través de las siguientes direcciones web:

<https://docplayer.es/44657219-Taller-1-grupo-9-tecnicas-de-aprendizaje-estadistico-profesora-claudia-jimenez-r.html>

<https://docplayer.es/66175720-Tecnicas-de-aprendizaje-estadistico-taller-1.html>

Para profundizar en el conocimiento de las técnicas de aprendizaje estadístico se recomienda la lectura de los siguientes talleres.

1. Si en un problema se desea identificar los síntomas correspondientes a tres enfermedades conocidas, las técnicas apropiadas para resolver el problema son (selecciona las opciones adecuadas):

 - A. Árboles de decisión.
 - B. Algoritmo apriori.
 - C. Algoritmo de recubrimiento secuencial.
 - D. Algoritmo PRISM.
2. Si en un problema se desea identificar relaciones entre síntomas de personas que presentan ciertas enfermedades, las técnicas apropiadas son:

 - A. Árboles de decisión.
 - B. Algoritmo apriori.
 - C. Algoritmo de recubrimiento secuencial.
 - D. Algoritmo PRISM.
3. Indica cuáles de las siguientes afirmaciones son verdaderas:

 - A. Las reglas de clasificación predicen la clase.
 - B. Las reglas de asociación predicen combinaciones de atributos o la propia clase.
 - C. Los algoritmos que aprenden reglas de asociación buscan combinaciones de pares atributo-valor que ocurren con cierta frecuencia.
 - D. Las reglas de asociación tienen el mismo objetivo que las reglas de clasificación.

- 4.** Si se quiere conocer el porcentaje de ejemplos que cumplen una regla respecto del total de ejemplos, se ha de aplicar la medida de:
- A. Cobertura.
 - B. Soporte.
 - C. Confianza.
 - D. Cubierta.
- 5.** Si se quiere conocer el porcentaje de ejemplos que cumplen una regla respecto de todos los ejemplos que sólo cumplen el antecedente, se ha de aplicar la medida de:
- A. Cobertura.
 - B. Soporte.
 - C. Confianza.
 - D. Cubierta.
- 6.** ¿Cuáles de los siguientes algoritmos se puede emplear para el aprendizaje de reglas de clasificación?
- A. PRISM.
 - B. C4.5.
 - C. Apriori.
 - D. ID3.

- 7.** Indica cuál de las siguientes afirmaciones es correcta:
- A. No es posible mapear árboles de decisión a reglas de clasificación.
 - B. Los algoritmos de recubrimiento secuencial aprenden una regla en cada iteración.
 - C. En cada iteración, el algoritmo de recubrimiento secuencial exige que la regla cubra todos los ejemplos positivos.
 - D. Apriori es un algoritmo de recubrimiento secuencial.
- 8.** Indica cuáles de las siguientes afirmaciones son correctas:
- A. El procedimiento básico, para aprender una regla, utilizado en los algoritmos de recubrimiento secuencial tiene como parámetro el conjunto de todas las clases.
 - B. El procedimiento, básico para aprender una regla, utilizado en los algoritmos de recubrimiento secuencial añade a la regla un único par atributo-valor en cada iteración.
 - C. El procedimiento de recubrimiento secuencial devuelve una única regla.
 - D. El algoritmo de recubrimiento secuencial elimina los ejemplos cubiertos por la regla generada en cada iteración.
- 9.** Indica cuáles de las siguientes afirmaciones son correctas respecto al algoritmo PRISM:
- A. Es un algoritmo de recubrimiento secuencial.
 - B. Utiliza la medida de precisión o confianza para generar las reglas.
 - C. Parte de la regla más específica alcanzando la más general.
 - D. Es un algoritmo de generación de conjuntos de reglas de los más simples.

10. Indica cuáles de las siguientes afirmaciones son verdaderas respecto al algoritmo apriori:

- A. Genera ítem-sets.
- B. Utiliza la medida de confianza para evaluar las reglas obtenidas.
- C. No genera reglas sino ítem-sets.
- D. Valora los ítem-sets generados mediante una medida de confianza.

Técnicas de Inteligencia Artificial

Tema 5. Redes neuronales artificiales

Índice

Esquema

Ideas clave

- 5.1. ¿Cómo estudiar este tema?
- 5.2. Introducción. Fundamento biológico
- 5.3. La neurona artificial. El perceptrón
- 5.4. Redes neuronales multicapa
- 5.5. Redes neuronales recurrentes. Redes Hopfield
- 5.6. Hacia el deep learning
- 5.7. Aplicaciones y ejemplos de implementación
- 5.8. Referencias bibliográficas

A fondo

Redes neuronales en Weka

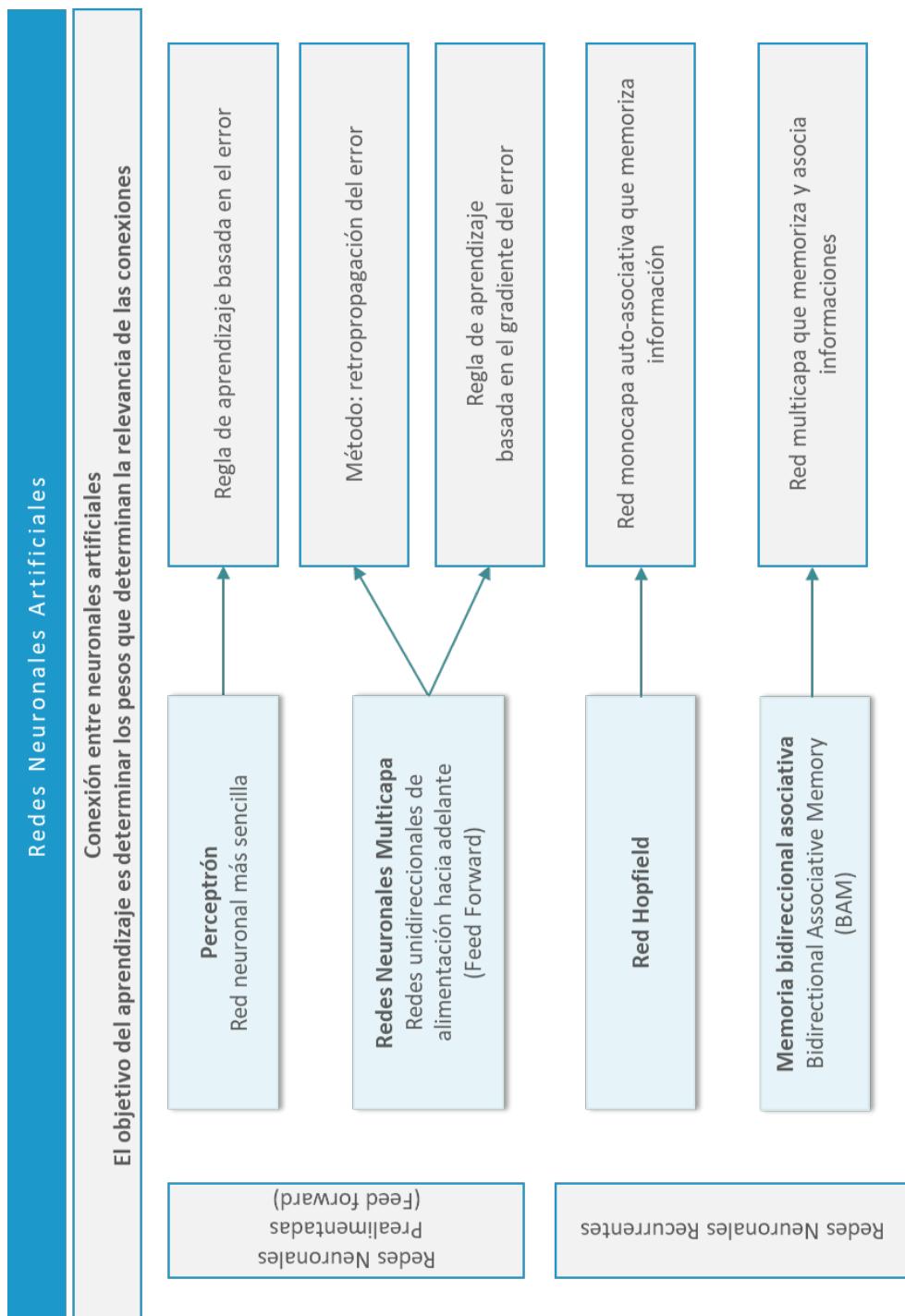
Redes neuronales aplicadas al reconocimiento de imágenes

Aplicaciones de las redes neuronales artificiales

Aplicación de las redes neuronales artificiales en el diagnóstico médico

Redes neuronales

Test



5.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral. Es recomendable ver esta lección antes de realizar la actividad grupal propuesta.

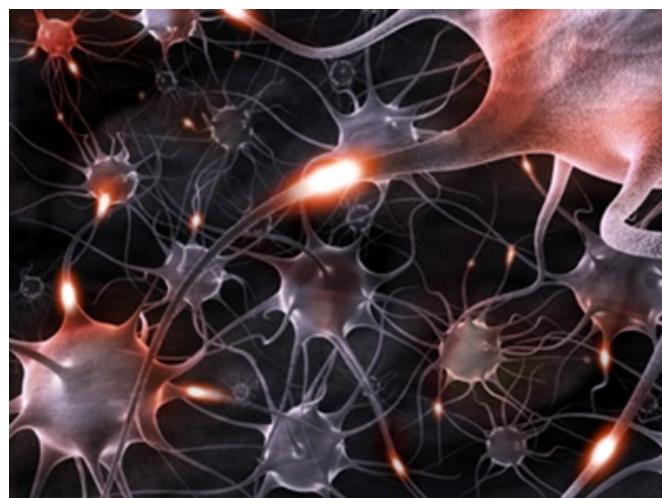
Al finalizar el estudio de este tema serás capaz de:

- ▶ Entender el funcionamiento básico de la neurona artificial y el perceptrón.
- ▶ Describir las redes neuronales multicapa.
- ▶ Diferenciar entre redes de propagación hacia adelante (*feed forward*) y redes bidireccionales (redes recurrentes).
- ▶ Entender la evolución de las redes neuronales hacia el *deep learning*.
- ▶ Identificar aplicaciones prácticas de redes neuronales.

5.2. Introducción. Fundamento biológico

Las redes neuronales artificiales son una de las técnicas de aprendizaje automático por excelencia (Alanis *et al.*, 2019; Graupe, 2019; Rogers & Girolami, 2017). Este tipo de redes permiten que mejore el rendimiento de un programa al desarrollar una tarea determinada gracias a que aprenden de la experiencia. De forma muy resumida, el modelo computacional de las redes neuronales se basa en el funcionamiento del cerebro humano.

El cerebro humano se compone de unidades de procesamiento y transmisión de información denominadas neuronas. Las neuronas reciben estímulos y transmiten los impulsos nerviosos conectándose con otras neuronas o con los músculos. A esta conexión se le denomina **sinapsis**.



El impulso nervioso se transmite desde el axón de una neurona, que sería el transmisor de la señal, hasta las dendritas de otras neuronas, que serían los receptores de los impulsos. Una neurona está compuesta de múltiples receptores (dendritas) y de un solo transmisor (axón) que se podrá ramificar en varias salidas iguales, tal y como podemos observar en la Figura 1. Cuando una dendrita recibe un impulso nervioso, segregá una determinada sustancia química que hace que la

neurona se excite a su vez y, si supera un determinado umbral, se envía el impulso eléctrico hacia el axón, causando la sinapsis y la excitación de otras neuronas.

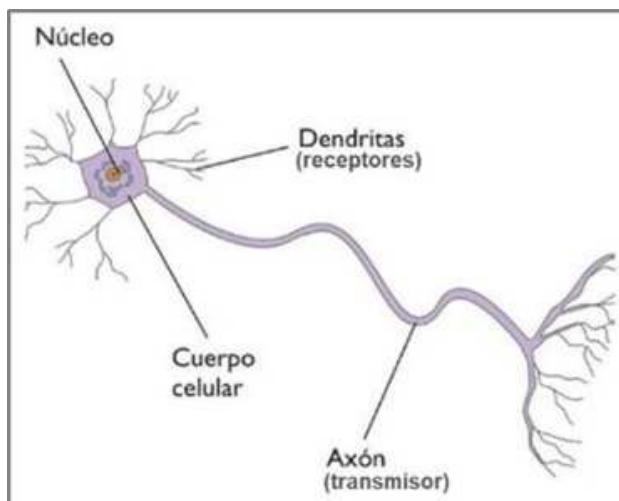


Figura 1. Componentes de una neurona biológica.

El cerebro humano, especialmente cuando se es un niño, es muy maleable. Es en esta fase de la vida cuando precisamente es más fácil aprender, procesar y retener información. Ante la información externa que llega, el cerebro es capaz de procesar esa información y almacenarla. Para ello, las neuronas crean nuevas conexiones con otras neuronas, las cuales se establecen y fortalecen a largo plazo. De este modo, el cerebro aprende las conexiones que dan lugar a respuestas correctas, las cuales se fortalecen, mientras que aquellas que dan lugar a respuestas incorrectas se debilitan y olvidan (Negnevitsky, 2011).

El cerebro humano se puede considerar como un sistema de procesamiento de información no-lineal y muy complejo. La información se almacena y procesa en las redes neuronales «como un todo», globalmente y simultáneamente, utilizando toda la red y no localizaciones específicas (Negnevitsky, 2011).

Tal y como se ha indicado anteriormente, las redes neuronales basan su funcionamiento en la forma de aprender del cerebro. Son capaces de realizar tareas de aprendizaje de manera eficaz, incluso llevando a cabo tareas de reconocimiento que un humano es incapaz de realizar.

Por este motivo son utilizadas en un gran número de aplicaciones. No hay más que realizar una búsqueda en Internet para encontrarse con una numerosa colección de artículos relativos al uso de redes neuronales desde hace décadas para solucionar diversos tipos de problemas en diversos campos como (Abiodun et al., 2018). La siguiente lista muestra algunos ejemplos concretos:

- ▶ Reconocimiento de voz (Graves *et al.*, 2013).
- ▶ Reconocimiento de escritura (Ptucha *et al.*, 2019).
- ▶ Conducción de vehículos autónoma (Pomerleau, 1993).
- ▶ En medicina, por ejemplo, para el análisis de imágenes (Anwar *et al.*, 2018).
- ▶ En telecomunicaciones se utiliza, por ejemplo, en redes móviles e inalámbricas (Zhang *et al.*, 2019).
- ▶ En compañías financieras se han aplicado en el estudio de patrones de uso de tarjetas de crédito con el fin de detectar fraudes (Roy *et al.*, 2018).
- ▶ En servicios web de mapas, las redes neuronales se han utilizado para predecir áreas probables de ser solicitadas en un futuro con el fin de ser precargadas en una caché y mejorar la experiencia del usuario (García *et al.*, 2013).
- ▶ En múltiples áreas de marketing como, por ejemplo, en la predicción de parámetros como el CTR (*Click Through Rate*) (Gao *et al.*, 2018).

Las redes neuronales son apropiadas en problemas donde las instancias tienen un gran número de atributos y cuando la salida puede tener cualquier valor: real, discreto o un vector con una combinación de valores reales y discretos.

Cuando se aplican redes neuronales, una vez aprendida la función objetivo, la evaluación de dicha función objetivo mediante el uso de instancias nuevas es rápida. Sin embargo, el tiempo de entrenamiento de las redes neuronales es largo, por lo que será una condición que ha de tenerse en cuenta a la hora de decidir su uso.

Por otra parte, tampoco ha de importar el no comprender la función objetivo, ya que los pesos que se aprenden son difíciles de interpretar. De hecho, en este sentido existe una corriente de investigación cada vez más en auge enfocada en tratar de explicar estos pesos, lo que se conoce como *Inteligencia Artificial Explicable* (XAI – *eXplainable Artificial Intelligence*) (Adadi y Berrada, 2018).

Una de las grandes ventajas de las redes neuronales es que estas se comportan de manera robusta frente al ruido, por lo que pueden ofrecer buenos resultados, aunque los ejemplos de entrenamiento contengan errores (Mitchell, 1997).

Tal y como hemos adelantado, las neuronas artificiales simulan el comportamiento de las neuronas biológicas. Al igual que las neuronas biológicas, las neuronas artificiales pueden recibir varios estímulos como entradas (emulando a las dendritas), pero únicamente cuentan con una salida (emulando el único axón que tienen las neuronas biológicas). Esta única salida podrá estar ramificada para así conectar con las entradas a otras neuronas a través de unos enlaces ponderados.

La relevancia de una determinada entrada a una neurona vendrá determinada por una serie de pesos que se establecen para los enlaces que conectan las neuronas. Son precisamente estos pesos los que se han de ajustar en la tarea de aprendizaje mediante redes neuronales.

Al igual que las neuronas del cerebro producen sinapsis a partir de un determinado nivel de excitación eléctrica, las neuronas artificiales se excitan con un determinado nivel que se establece ajustando los pesos mencionados para cada entrada a la neurona, así como una *función objetivo* que determina la salida de la neurona a partir de un determinado umbral.

La Figura 2 muestra la estructura típica de una red neuronal básica. En ella puede observarse que está compuesta por una serie de entradas y salidas, así como una capa intermedia de neuronas. Tal y como se ha indicado anteriormente, uno de los objetivos principales de la red neuronal es el ajuste de los pesos de los enlaces que conectan las neuronas. La estructura que muestra la Figura 2 es un ejemplo de diseño básico ya que, tal y como puede intuirse, existen múltiples posibilidades de diseño de una red neuronal.

El primer paso para la implementación de un sistema de aprendizaje basado en redes neuronales es el diseño de una estructura inicial de dicha red. En dicho diseño, tanto el número de capas intermedias como el número de neuronas de cada capa es variable.

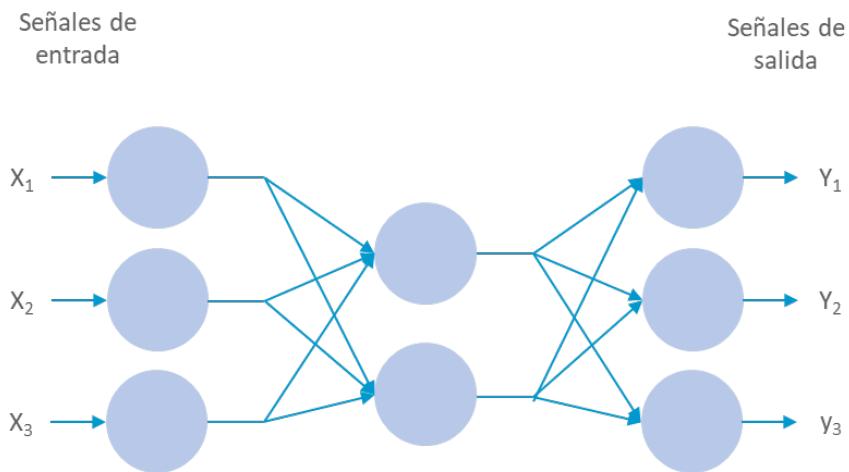


Figura 2. Estructura de una red típica neuronal.

5.3. La neurona artificial. El perceptrón

El componente básico de una red neuronal es la neurona artificial, cuya estructura básica se muestra en la Figura 3. Tal y como se ha indicado en las secciones anteriores, la neurona artificial está compuesta por varias entradas y una salida que puede ramificarse en varias señales iguales.

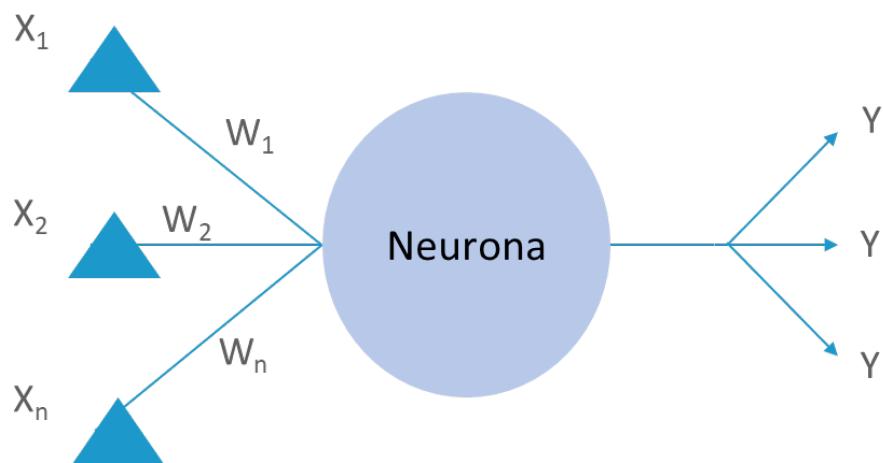


Figura 3. Estructura básica de una neurona artificial

La salida de una neurona se puede calcular utilizando la función **signo**:

$$Y = \begin{cases} +1 & \text{si } X \geq 0 \\ -1 & \text{si } X < 0 \end{cases} \quad (1)$$

$$X = w_0 + \sum_{i=1}^n x_i w_i \quad (2)$$

La salida de la neurona también se puede calcular con otras funciones comúnmente utilizadas, denominándose todas ellas **funciones de activación**, como ya hemos mencionado.

Algunos ejemplos de función de activación son la función escalón, la función lineal, la función sigmoide, la tangente hiperbólica, ReLU (*Rectified Linear Unit*), Leaky ReLU, o *Softmax*, por ejemplo, disponible en <http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>

La red neuronal artificial más sencilla es el **perceptrón**, cuya estructura se muestra en la (Mitchell, 1997). El perceptrón está formado por una sola capa con las entradas y salidas que se necesiten para resolver el problema planteado. La máxima simplificación de este perceptrón es lo que se conoce como **perceptrón simple**, el cual está formado por una sola neurona.

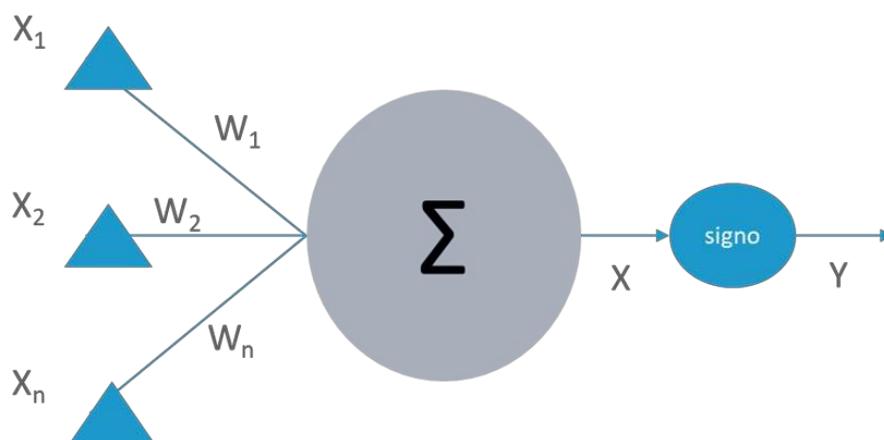


Figura 4. Estructura del perceptrón (Mitchell, 1997).

En esta red neuronal simple, se toman las entradas y se aplican las expresiones (1) y (2) para obtener la salida. Entonces, dada esta red tan simple, ¿en qué se basa el aprendizaje de la red neuronal? A grandes rasgos, el **objetivo del aprendizaje** se reduce a, dado un conjunto de datos de entrenamiento que consiste en una serie de entradas a esa red y las salidas conocidas correspondientes, **escoger los pesos que se ajusten mejor a esas entradas y salidas** definidas a priori.

Este objetivo es el mismo en principio para redes simples o más complejas y, como podemos comprobar, encaja con un modelo de aprendizaje supervisado en el cual los datos de entrenamiento se encuentran etiquetados.

Al formar la red neuronal se desconocerá el valor de los pesos, por lo que estos se asignarán inicialmente de forma aleatoria. De este modo, la salida que se obtiene al aplicar los pesos en cada iteración será diferente a la salida conocida o esperada. Esta diferencia entre la salida obtenida y la esperada es lo que se conoce como el error (también llamado pérdida – *loss* – o costo – *cost*), valor que se utiliza para ajustar los pesos siguiendo la regla de aprendizaje del perceptrón (Rosenblatt, 1960). La formulación matemática de esta idea se la siguiente:

$$w_i(t+1) = w_i + \alpha \times x_i(t) \times e(t) \quad (3)$$

Siendo $e(t)$ la diferencia entre la salida esperada y la salida real en la entrada procesada en la iteración t . En cada iteración se procesaría uno de los datos de entrenamiento disponibles.

α es la tasa de aprendizaje, un valor entre 0 y 1 que pondera la relevancia del error obtenido en última iteración.

El algoritmo de aprendizaje de la red neuronal consta de los siguientes pasos (Negnevitsky, 2011):

- ▶ Se asignan valores aleatorios en el intervalo [-0.5, 0.5] al umbral w_0 y a los pesos w_1, w_2, \dots, w_n .
- ▶ Se activa el perceptrón aplicando las entradas $x_1(t), x_2(t), \dots, x_n(t)$ y teniendo en cuenta la salida deseada $y_d(t)$. Se calcula la salida real en la iteración utilizando la función de activación signo (Ecuación 1). Se puede utilizar cualquier otra función de activación, tal y como se indicó al inicio de esta sección.

- ▶ A partir de la salida real obtenida en el paso 2 se actualizan los pesos aplicando la expresión de la Ecuación 3.
- ▶ Se retorna al paso 2 hasta alcanzar la convergencia, que puede estar determinada por un error máximo admisible, por ejemplo, o un número máximo de iteraciones.

El perceptrón es una red neuronal muy simple y es un tipo de *claseificador o discriminador lineal*. Es decir, durante el entrenamiento trata de elegir la mejor recta (o *hiperplano*, en realidad) que separa los vectores de características que forman los datos de entrenamiento a la entrada, devolviendo un valor -1 o +1 a la salida (si utilizamos la función signo como función de activación).

Esto implica que puede no encontrar una recta discriminante perfecta, sino una aproximada. Por eso hay que poner límites a la convergencia.

Sin embargo, esta simplicidad no obsta para que sobre él esté basado el funcionamiento de diversos tipos de redes neuronales. El funcionamiento de la red neuronal vendrá determinado por los siguientes parámetros:

- ▶ **Arquitectura de la red**, esto es, el número de capas, número de neuronas por capa y las conexiones entre neuronas entre diferentes capas.
- ▶ **Función de activación**: función signo, función escalón, etc.
- ▶ **Algoritmo de aprendizaje** determinado principalmente por la regla de aprendizaje para ajustar pesos (por ejemplo, la expresión de la Ecuación 3).

5.4. Redes neuronales multicapa

La evolución del perceptrón simple en el perceptrón multicapa, mediante la incorporación de capas de neuronas intermedias u ocultas, da lugar a las **redes neuronales multicapa**. Las **redes neuronales multicapa** son redes unidireccionales de alimentación **hacia adelante (feedforward)**. Dado que se trata de redes de alimentación en una única dirección de atrás hacia adelante, la señal de entrada se va propagando a lo largo de las capas hacia la salida.

Estas redes tienen al menos una capa intermedia oculta, como las mostradas en la Figura 2 y en la Figura 5, aunque pueden tener más de una capa oculta. En este sentido, las redes neuronales comerciales (en los casos en los que no hablamos de *deep learning*) incorporan 1 o 2 capas ocultas y estas capas pueden tener desde 10 a 1000 neuronas (Negnevitsky, 2011) aunque, como es esperable, estas cifras aumentan con la mejora de la capacidad de proceso y la optimización de los algoritmos. **Dado que incorporar nuevas capas supone ampliar la carga computacional de manera exponencial**, en la práctica no es recomendable trabajar con un número elevado de capas.

En el caso de modelos *deep learning*, como veremos en el Tema 6, existen ejemplos extremos de arquitecturas de redes neuronales de cientos o incluso un millar de capas ocultas, como algunas *redes convolucionales* empleadas en visión artificial, en concreto las *redes residuales* o *resnet*.

Sin embargo, no siempre una red profunda tendrá un mayor rendimiento que una red poco profunda. Esto es algo que hay que evaluar en cada caso de uso (Wu et al., 2019).

El aprendizaje en las redes multicapas se realiza de la misma manera que el aprendizaje en la red simple del perceptrón expuesta en la sección anterior. A partir de unos datos de entrenamiento disponibles se alimenta la red con las entradas, se calcula la salida y la diferencia entre esta salida y la salida esperada (error) se utiliza para ajustar los pesos con el objetivo de reducir el error.

Hay muchos métodos y algoritmos para el aprendizaje de redes neuronales. En este tema se va a exponer un método comúnmente utilizado para el aprendizaje en redes neuronales denominado *retropropagación del error*, más conocido por su nombre en inglés: ***back-propagation***. Las **redes de retropropagación** son aquellas que utilizan este método, las cuales están formadas por capas en las que **todas las neuronas de una capa se conectan con todas las neuronas de las capas anterior y posterior**.

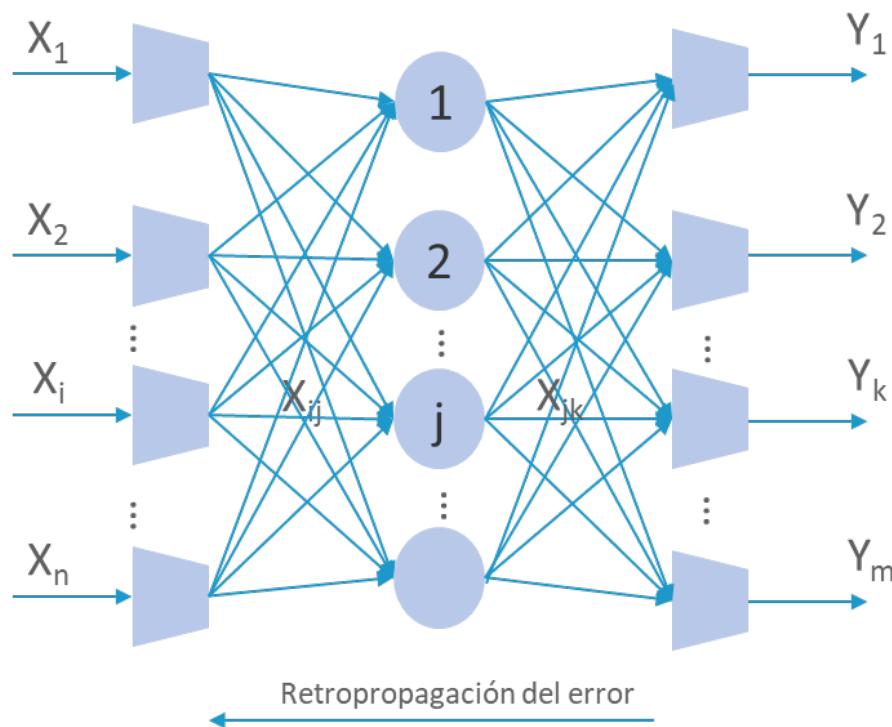


Figura 5. Retropropagación del error en una red neuronal.

La salida de una neurona X se determina con la expresión de la Ecuación 2 y la función de activación es una *función sigmoide*, es decir, *una función $S(x)$ que tiene forma de «S»*. Un ejemplo de función sigmoide es la *función logística*, cuya expresión viene dada por la Ecuación 4.

$$Y = \frac{1}{1 + e^{-x}} \quad (4)$$

Otros posibles ejemplos alternativos de funciones sigmoides son la *arcotangente* o la *tangente hiperbólica*, entre otras.

Una vez definida la arquitectura, habitualmente de 3 o 4 capas, con todas las neuronas entre capas adyacentes conectadas entre sí, y conocida la función de activación, únicamente queda determinar la regla de aprendizaje. En las redes multicapa se utiliza el **gradiente del error** para ajustar los pesos, el cual bien dado por la expresión de la Ecuación 5:

$$\delta_k(t) = \frac{\partial y_k(t)}{\partial X_k(t)} \times e_k(t) = \frac{\partial y_k(t)}{\partial X_k(t)} \times (y_{d,k}(t) - y_k(t)) \quad (5)$$

Siendo:

- ▶ $X_k(t)$: entrada ponderada a la neurona k (calculada según la Ecuación 2) en la iteración t .
- ▶ $y_k(t)$: salida real de la neurona k en la iteración t (calculada según la función de activación de la Ecuación 4).
- ▶ $y_{d,k}(t)$: salida esperada en la neurona k .

En la Ecuación 5, si se sustituye $Y_k(t)$ por la expresión dada en la Ecuación 4, se obtiene la fórmula para el cálculo del gradiente del error, detallada en la Ecuación 6:

$$\begin{aligned}\delta_k(t) &= y_k(t) \times (1 - y_k(t)) \times e_k(t) \\ y_k(t) \times (1 - y_k(t)) \times (y_{d,k}(t) - y_k(t))\end{aligned}\quad (6)$$

Para ajustar los pesos de los enlaces entre la neurona j y la neurona k , es decir, w_{jk} , en la capa de salida se utiliza la expresión indicada en la Ecuación 7:

$$w_{jk}(t+1) = w_{jk}(t) + \alpha \times y_j(t) \times \delta_k(t) \quad (7)$$

Siendo

$y_j(t)$ la salida de la neurona j en la capa oculta.

Para el caso de la capa oculta los pesos se reajustan según la expresión dada por la Ecuación 7, utilizando igualmente el gradiente del error. Sin embargo, el cálculo de este es diferente ya que en la capa oculta no solo hay una salida en la neurona, sino múltiples salidas a las que se aplicarán diferentes pesos. Se emplea, por tanto, la expresión dada por la Ecuación 8:

$$\delta_i(t) = y_i(t) \times (1 - y_i(t)) \times \sum_{k=1}^m \delta_k(t) w_{jk}(t) \quad (8)$$

Siendo m el número de neuronas en la capa de salida,

$\delta_k(p)$ es el gradiente del error calculado para la neurona de salida k -ésima, y w_{jk} , el peso de la conexión entre la neurona j y la neurona k .

$y_j(t)$ es la salida obtenida mediante la Ecuación 4 a partir de las entradas a la neurona j en la iteración t , es decir, $x_1(t), x_2(t), \dots, x_n(t)$:

$$y_j(t) = \frac{1}{1 + e^{-X_j(t)}} \quad (9)$$

$$X_j(t) = w_{0j} + \sum_{i=1}^n x_i(t) \times w_{ij}(t) \quad (10)$$

Por tanto, el algoritmo de aprendizaje para una red multicapa de 3 capas consta de las siguientes fases:

- ▶ Establecimiento inicial de los pesos con valores aleatorios pequeños.
- ▶ Para cada dato de entrada $x_1(t), x_2(t), \dots, x_n(t)$, calcular las salidas de la red al activarla con esos datos de entrada.
- Cálculo del gradiente del error para las neuronas de la capa de salida según la Ecuación 6 y reajuste de sus pesos según la Ecuación 7.
- Cálculo del gradiente del error para las neuronas en la capa oculta, según la Ecuación 8 y reajuste de sus pesos según la Ecuación 7.
- ▶ Repetir los pasos 2 y 3 hasta que se cumpla la convergencia, siendo los errores suficientemente pequeños, o un criterio de parada específico.

De acuerdo con Negnevitsky (2011), los algoritmos puros de retropropagación son raramente utilizados en la práctica por la alta carga computacional, dando lugar a un entrenamiento lento. Sin embargo, se han desarrollado muchas variaciones (Mitchell, 1997) que permiten mejorar el rendimiento, como utilizar una función de activación del tipo tangente hiperbólica o añadiendo un «momento» en el cálculo de los ajustes de los pesos, de tal manera que la actualización de un peso en la iteración t dependa parcialmente de la actualización realizada en la iteración anterior $t-1$.

Realmente, las redes de retropropagación de dos o tres niveles tienen un gran potencial a la hora de representar diferentes funciones (Mitchell, 1997), como funciones booleanas, continuas, funciones de aproximación arbitraria, etc., y este tipo de redes son muy utilizadas en, por ejemplo, problemas de reconocimiento y clasificación de patrones.

5.5. Redes neuronales recurrentes. Redes Hopfield

Las redes neuronales recurrentes tratan de emular las características asociativas de la memoria humana (Mitchell, 1997). La peculiaridad de estas redes es que sus salidas alimentan las entradas, formando un bucle. Se calcula la salida de la red a partir de una entrada dada y se aplica la salida obtenida en la iteración (t) como entrada en la iteración ($t+1$), repitiendo el proceso hasta que se alcanza una estabilidad, siendo la salida constante en las distintas iteraciones.

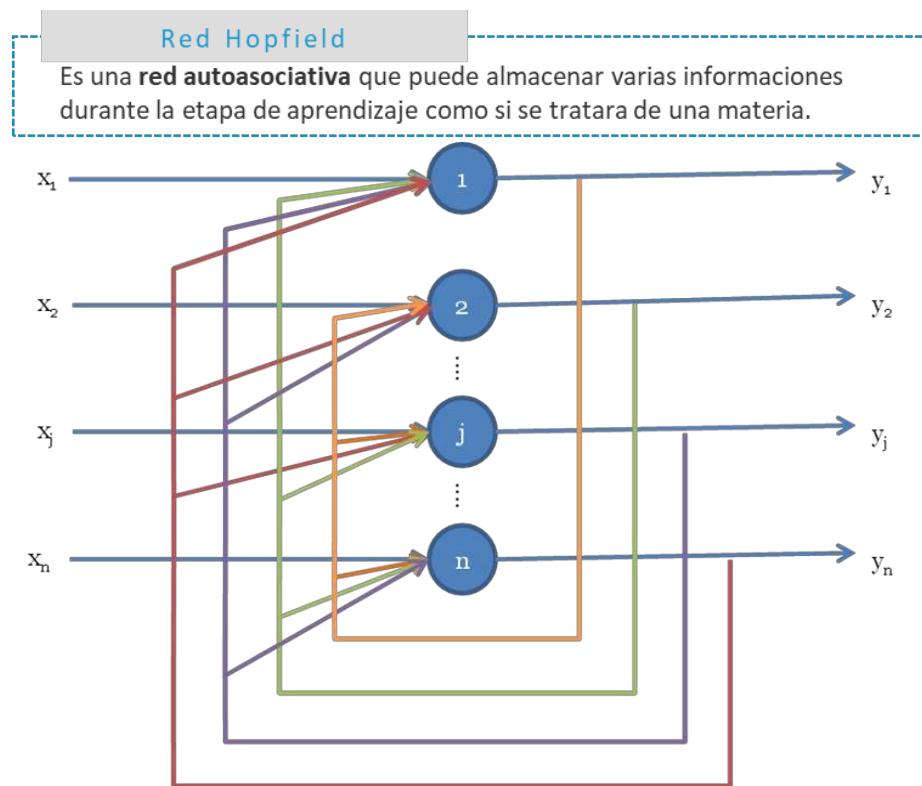


Figura 6. Arquitectura de una red neuronal recurrente monocapa, red Hopfield.

La Figura 6 muestra una red Hopfield monocapa con n neuronas. Las salidas de las n neuronas retroalimentan las entradas a las diferentes neuronas, pero nunca a ellas mismas (no hay autoretroalimentación). En una red Hopfield las salidas pueden ser binarias o números reales. Estas redes pueden utilizar diversas funciones como

función de activación, aunque es habitual utilizar la función signo siguiente (Negnevitsky, 2005).

$$y^{signo} = \begin{cases} +1, & \text{si } X > 0 \\ -1, & \text{si } X < 0 \\ Y, & \text{si } X = 0 \end{cases} \quad (11)$$

El estado de la red viene determinado por el conjunto de salidas $[y_1, y_2, \dots, y_n]$ y el objetivo es que la red sea capaz de almacenar unos determinados estados $Y_1, Y_2, \dots, Y_m, \dots, Y_M$, denominados memorias fundamentales.

Estos estados los almacena la red durante un período de entrenamiento o aprendizaje. Una vez finalizada la etapa de aprendizaje, **si se presenta alguna de las informaciones almacenadas a la entrada, la red se estabiliza ofreciendo a la salida la misma información** que se ha presentado a la entrada y que coincide con la información almacenada.

Si la entrada no coincide con una de las memorias fundamentales, la red evoluciona hacia una salida lo más parecida a una de las informaciones almacenadas.

Con este objetivo, los pasos del algoritmo de entrenamiento de la red Hopfield son los siguientes:

- ▶ 1. Cálculo de la matriz de pesos
- ▶ 2. Comprobación de que la red es capaz de memorizar las memorias fundamentales
- ▶ 3. Comprobación con entradas de prueba de que la red recupera un estado estable

1. Cálculo de la matriz de pesos

El valor del peso w_{ij} de la conexión entre las neuronas i y j se calcula con la siguiente expresión:

$$w_{ij} = \begin{cases} \sum_{m=1}^M y_{m,i} y_{m,j}, & i \neq j \\ 0 & i = j \end{cases} \quad (12)$$

Siendo $y_{m,i}$ e $y_{m,j}$ los elementos i y j del vector de salida deseado Y_m .

Dado que las salidas se pueden representar como vectores de n elementos, se puede calcular también los pesos mediante la siguiente expresión matricial:

$$W = \sum_{m=1}^M Y_m Y_m^T - MI \quad (13)$$

Siendo Y_m el vector n -dimensional que se quiere memorizar, I la matriz identidad y M el número de estados que es necesario que memorizar.

Por ejemplo, si se quieren almacenar las memorias fundamentales: $Y_1=[1, 1, 1]$ e $Y_2=[-1, -1, -1]$, aplicando ambas expresiones (12) o (13), se obtiene la siguiente matriz de pesos:

$$W = \begin{bmatrix} 0 & w_{12} & w_{13} \\ w_{21} & 0 & w_{23} \\ W_{31} & W_{32} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

2. Cálculo de la matriz de pesos

Cuando a la red se le presenta una memoria fundamental Y_m como entrada, la red tiene que ser capaz de memorizarla y proporcionarla a la salida.

Por tanto, para cada memoria fundamental Y_m de n componentes, $m = 1, 2, \dots, M$, si se da una entrada tal que

$$x_{m,i} = y_{m,i}, i = 1, 2, \dots, n$$

Y si la función de activación es la función signo en (11), se ha de cumplir lo siguiente:

$$y_{m,i} = \text{signo} \left(\sum_{j=1}^n w_{ij} - W_0,i \right) \quad (14)$$

Dado que se trata de vectores, se puede realizar la comprobación mediante la expresión matricial siguiente:

$$Y_m = \text{signoo}(WX_m - W_0) \quad (15)$$

Para los datos del ejemplo expuesto anteriormente, si los umbrales w_0 son iguales a 0, se puede comprobar cómo efectivamente, a partir de una entrada $[1 \ 1 \ 1]$ igual a una de las memorias fundamentales Y_1 , la red obtiene la salida correspondiente a esa memoria fundamental:

$$Y_1 = \text{signo} \left(\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Realizando la misma operación, para una entrada igual a Y_2 se comprueba que efectivamente la red obtiene una salida igual a esa memoria fundamental.

3. Cálculo de la matriz de pesos

Se toma un vector $X=[x_0, x_1, \dots, x_n]$ que sea diferente a cualquiera de los de la memoria fundamental y se aplica como entrada a la red obteniendo como salida:

$$y_i = \text{signo} \left(\sum_{j=1}^n w_{ij}x_j(0) - w_{0,i} \right) \quad (14)$$

Siendo $x_j(0)$ el componente j de la entrada en la iteración $t=0$ y $y_i(0)$ la salida de la neurona i en la iteración $t=0$. En forma matricial:

$$Y(0) = \text{signo}(WX(0) - W_0) \quad (15)$$

Se repite el paso anterior hasta que el vector de estado o la salida se mantiene estable, es decir, no cambia. En las diferentes iteraciones, las neuronas están recibiendo como entrada las salidas de las otras neuronas multiplicadas por el peso correspondiente.

Si la actualización de las salidas de las neuronas de la red se realiza de forma simultánea, se hablar de una red de Hopfield **síncrona**. Por tanto, en la iteración $(t+1)$ todas las neuronas van a utilizar como entradas las salidas generadas por las otras neuronas en la iteración (t) . Si las neuronas trabajan asíncronamente, se actualiza solo la salida de una neurona en cada iteración, con lo que se habla de una red **asíncrona**. En este caso la salida a la que converge la red dependerá del orden de activación de las neuronas.

La condición de estabilidad es:

$$y_i(t+1) = \text{signo} \left(\sum_{j=1}^n w_{ij} y_j(t) - w_{0,i} \right) \quad (16)$$

O, en forma matricial:

$$Y(t+1) = \text{signo}(WY(t) - W_0) \quad (17)$$

Para el ejemplo con el que se está trabajando, si se tiene una entrada de prueba $[-1 \ 1 \ 1]$, la salida en la iteración $t=0$ es:

$$Y(0) = \text{signo} \left(\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

La salida no coincide con la entrada, la red no está estable. Esa salida se aplica de nuevo a la entrada y se obtiene en la iteración $t=1$:

$$Y(1) = signo \left(\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$Y(1)$ es precisamente una de las memorias fundamentales y, tal y como se ha comprobado en el segundo paso del algoritmo, la red es capaz de memorizar esa memoria fundamental y, por tanto, ese estado quedará estable en la red.

Un problema de la red Hopfield es que si alcanza un estado estable no siempre ese estado es una memoria fundamental. Por otra parte, las redes Hopfield tienen una **limitación de capacidad**, por lo que el número de informaciones que puede ser aprendido o almacenado es limitado, precisando de gran cantidad de neuronas y de conexiones para almacenar pocas informaciones.

Las redes de Hopfield son redes autoasociativas que pueden recuperar memorias incompletas, o información completa, a partir de información incompleta. Son redes utilizadas en reconocimiento de imágenes o de patrones y en problemas de optimización, por ejemplo.

Sin embargo, las redes de Hopfield no pueden asociar una información con otra. Si en vez de una red recurrente con una capa de neuronas se tienen dos niveles, se pueden generar **memorias bidireccionales asociativas** que sí son capaces de asociar informaciones diferentes.

5.6. Hacia el deep learning

A pesar de resolver múltiples problemas eficazmente, las redes neuronales de retropropagación presentan una serie de limitaciones que deben ser resueltas. Estas limitaciones reducen principalmente la eficacia de este tipo de redes en problemas complejos en los que el número de nodos intermedios es elevado. El elevado número de nodos aumenta el número de conexiones y, por ende, el cálculo de los pesos asociados. De este modo, el entrenamiento de las redes se convierte en un proceso ineficiente y costoso, por lo que han de buscarse soluciones que nos permitan la creación de redes neuronales con decenas y cientos de nodos y capas.

Es entonces cuando aparece el **aprendizaje profundo**, más conocido por traducción al inglés **deep learning** (Graupe, 2019; Pouyanfar *et al.*, 2018). *Deep learning* es un área específica del aprendizaje automático, que se basa en la utilización de redes neuronales con un alto número de nodos y capas. Este nuevo concepto dentro del aprendizaje automático lida con procesos complejos que trabajan con volúmenes elevados de datos, así como con la interconexión necesaria entre los diferentes sistemas que forman parte de la solución completa.

Por otro lado, *deep learning* se encarga del desarrollo de nuevos algoritmos y configuración de redes que mejoren la eficacia de la red neuronal y, a su vez, permitan mejorar su eficiencia. Para ello, entre otras técnicas, esta área incluye la utilización de técnicas de aprendizaje no supervisado en las capas intermedias para que estas aprendan automáticamente en base a la experiencia, incluso conceptos que antes no conocían. Una de las aplicaciones más típicas de *deep learning* es en sistemas que incluyen la extracción de características y la clasificación en la misma solución.

Dada la importancia del *deep learning* en los sistemas de aprendizaje, el siguiente tema abordará en profundidad, valga la redundancia, este método de aprendizaje

automático a través de la presentación de los algoritmos más significativos y múltiples ejemplos de aplicación.

5.7. Aplicaciones y ejemplos de implementación

Algunas aplicaciones de las redes neuronales en la literatura

Las redes neuronales son, sin duda, una de las técnicas de aprendizaje automático (*machine learning*) por excelencia y con mayor potencial en la actualidad. De este modo, las redes neuronales pueden utilizarse como sustitutos de todos los métodos supervisados, no supervisados y semisupervisados (algunos vistos ya previamente y algunos que veremos en temas posteriores) obteniendo una precisión mucho mayor. Obviamente, esta mayor precisión se consigue a cambio de un mayor coste computacional, a la hora de entrenar los modelos y a la hora de testear y ejecutar los modelos en producción, que habrá que tener en cuenta.

De ahí que en algunas ocasiones sea necesario considerar si utilizar redes neuronales (más aún redes neuronales profundas) es lo más adecuado si existen modelos de *machine learning* clásico (árboles de decisión, SVM, etc.) o *ensemble learning* (*random forest*, etc.) que nos den la precisión que necesitamos a cambio de una menor carga computacional. Esto es especialmente importante si estamos aplicando algoritmos de *machine learning* en dispositivos móviles, en dispositivos embebidos como nodos IoT (*Internet of Things*) o que se ejecuten del lado del navegador o cliente web.

Además de para todos los algoritmos supervisados y no supervisados que podamos imaginar, las redes neuronales (entendiéndolas de forma global como redes neuronales y redes neuronales profundas) también se utilizan, entre otras posibles aplicaciones, para la **identificación de objetos en imágenes y vídeos** (Shah, Bennamoun y Boussaid, 2016), el **reconocimiento de voz** (Zhang *et al.* 2018), la **síntesis de voz** (Arik, *et al.* 2017), **análisis de sentimientos y reconocimiento de**

emociones del habla (Fayek, Lech y Cavedon, 2017), **procesamiento de imágenes** (Razzak, Naz y Zaib, 2018), **transferencia de estilos** (por ejemplo, aplicación del estilo de pintura de Van Gogh a cualquier fotografía) (Luan *et al.*, 2017), **procesamiento del lenguaje natural (*Natural Language Processing*)** (Costa-jussà *et al.*, 2017), **traducción automática** (Deng y Liu, 2018), etc.

Ejemplos de implementación

Anteriormente en el Tema 3 (Árboles de decisión) utilizamos diferentes librerías de Python para comparar entre sí el rendimiento de clasificadores como la regresión logística y de un árbol de decisión (utilizando la versión optimizada del algoritmo CART que nos proporcionaba scikit-learn) en la clasificación de especies de flor Iris a partir del *dataset iris de Fisher*.

En esta ocasión vamos a comparar otros algoritmos clasificadores que hemos ido mencionando en los anteriores temas, en concreto el *random forest* (visto en el Tema 3), el algoritmo *k-NN*, los *clásificadores Naïve Bayes*, las *máquinas de soporte vectorial (SVM)*, estos tres últimos mencionados en el Tema 4, y el *perceptrón multicapa (MLP)*, visto en este mismo tema como un ejemplo de red neuronal.

Recordemos la información que teníamos sobre el *dataset* por facilidad de lectura. Este *dataset* es un conjunto de datos multivariante de 150 muestras, 50 de cada una de las tres especies de flor Iris (*Iris setosa*, *Iris virginica* e *Iris versicolor*), con medidas del largo y ancho del sépalo y el pétalo de cada una de ellas. Este *dataset* viene incluido en scikit-learn , de modo que lo podemos importar de dicha librería, como ya hemos visto en el Tema 1. No obstante, lo importaremos de OpenML, disponible en: <https://www.openml.org/d/61>

Que, como vemos, tiene una URL de acceso al CSV mediante:
https://www.openml.org/data/get_csv/61/dataset_61_iris.arff

Largo de sépalo	Ancho de sépalo	Largo de pétalo	Ancho de pétalo	Especies
5.1	3.5	1.4	0.2	I. setosa
4.9	3.0	1.4	0.2	I. setosa
4.7	3.2	1.3	0.2	I. setosa

Tabla 1. Conjunto de datos flor Iris o iris de Fisher (muestra de las primeras filas).

Volvamos a echar un vistazo a los datos del *dataset* en modo texto, tal y como se explica en los comentarios del código, que ya vimos en el Tema 3:

```
# Load libraries
from pandas import read_csv

url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
# La siguiente línea no es necesaria usando esta fuente, pues ya incluye la cabecera
#names = ['sepallength', 'sepalwidth', 'petallength', 'petalwidth', 'class']
#dataset = read_csv(url, names=names)
dataset = read_csv(url)

# mostramos la "forma", debería haber 150 entradas con 5 atributos cada una
print(dataset.shape)
#> (150 , 5)

# mostramos las 3 primeras entradas para echar un vistazo
print(dataset.head(3))
#>   sepallength  sepalwidth  petallength  petalwidth      class
#>0      5.1       3.5        1.4       0.2  Iris-setosa
#>1      4.9       3.0        1.4       0.2  Iris-setosa
#>2      4.7       3.2        1.3       0.2  Iris-setosa
```

```
# mostramos un resumen estadístico de los datos
print(dataset.describe())
#>   sepallength sepalwidth petallength petalwidth
#>count 150.000000 150.000000 150.000000 150.000000
#>mean  5.843333 3.054000  3.758667 1.198667
#>std   0.828066 0.433594  1.764420 0.763161
#>min   4.300000 2.000000  1.000000 0.100000
#>25%   5.100000 2.800000  1.600000 0.300000
#>50%   5.800000 3.000000  4.350000 1.300000
#>75%   6.400000 3.300000  5.100000 1.800000
#>max   7.900000 4.400000  6.900000 2.500000

# distribución por clases
print(dataset.groupby('class').size())
#>Iris-setosa    50
#>Iris-versicolor 50
#>Iris-virginica 50
#>dtype: int64
```

Echemos un vistazo a los datos ahora en modo gráfico. En esta ocasión vamos a instalar un nuevo paquete, llamado *seaborn* (basado, a su vez, en *matplotlib*) que nos permite analizar gráficamente nuestros datos. Disponible en <https://seaborn.pydata.org/>.

Lo instalamos como es habitual utilizando nuestro gestor de paquetes, por ejemplo:

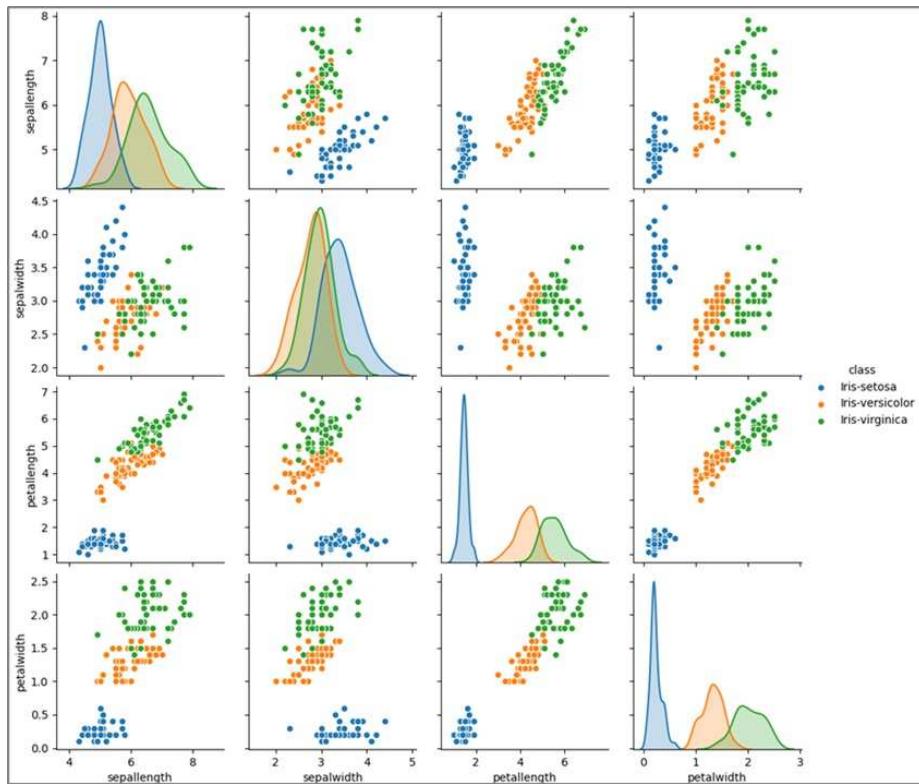
```
PS C:\Users\xxx> pip install seaborn
```

Y, ejecutamos el siguiente código:

```
# Carga de librerías
from pandas import read_csv
import matplotlib.pyplot as plt
import seaborn as sns

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)

sns.pairplot( data=dataset, vars=("sepallength","sepalwidth","petallength","petalwidth"), hue="class" )
plt.show()
```



Ahora vamos a evaluar cinco algoritmos *machine learning* y compararlos entre sí: **un random forest, un clasificador k-NN, un clasificador Naïve Bayes, una SVM y un MLP (perceptrón multicapa)**.

Para ello, de nuevo como hicimos en el Tema 3, separaremos nuestro *dataset*: usaremos un 80 % de los datos para entrenar los algoritmos y un 20 % de los datos para hacer los test de predicción.

Además, de nuevo utilizaremos una **validación cruzada estratificada de 10 veces (k-fold)** para estimar la precisión del modelo.

Como recordaremos, esto dividirá nuestro conjunto de datos en 10 partes, entrenando en 9 partes y testeando en 1 parte y repetirá para todas las combinaciones de divisiones de entrenamiento-test.

Nota: lo habitual es normalizar los datos, especialmente cuando trabajamos con redes neuronales, si bien en este ejemplo no lo estamos realizando. Veremos un ejemplo de normalización en el Tema 6.

```
# Load libraries
import numpy as np
import pandas as pd
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

# inicializamos la semilla para generar números aleatorios
np.random.seed(0)

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)
```

```

# Dividimos el dataset en 80% de datos para entrenar y 20% para testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]

X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20, random_state=1, shuffle=True)

# Cargamos los algoritmos
models = []
models.append(("RF", RandomForestClassifier(n_jobs=2, random_state=0)))
models.append(("KNN", KNeighborsClassifier()))
models.append(("NB", GaussianNB()))
models.append(("SVM", SVC(gamma='auto')))
models.append(("MLP", MLPClassifier(activation="relu", alpha=1e-05, batch_size="auto", beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(3, 3), learning_rate="constant", learning_rate_init=0.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True, solver="lbfgs", tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)))

# evaluamos cada modelo por turnos
results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

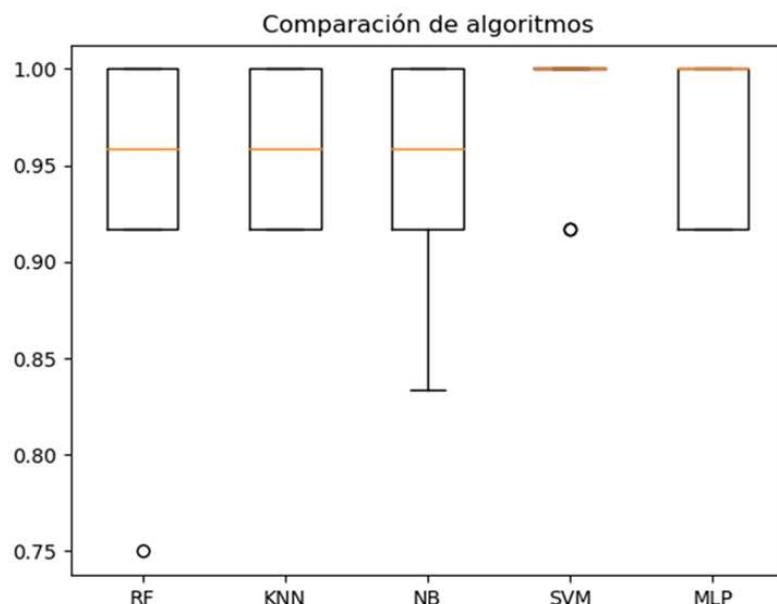
# Comparación de algoritmos
pyplot.boxplot(results, labels=names)
pyplot.title('Comparación de algoritmos')
pyplot.show()

#>RF: 0.941667 (0.075000)
#>KNN: 0.958333 (0.041667)
#>NB: 0.950000 (0.055277)
#>SVM: 0.983333 (0.033333)
#>MLP: 0.966667 (0.040825)

```

Obtendremos la siguiente gráfica comparativa, además. Como vemos en los resultados por consola (mostrados en los comentarios del código) y en la gráfica, aunque el MLP (dos capas ocultas de 3 neuronas cada una) obtiene unos excelentes resultados (96.7 % de precisión), la SVM obtiene unos resultados incluso ligeramente mejores (98.3 % de precisión promedio y una menor desviación estándar de la

precisión en las pruebas). Por supuesto, esto no quiere decir que no podamos encontrar una configuración de capas y neuronas de MLP que supere a la SVM. Podemos realizar diferentes pruebas con más capas y más neuronas, y veremos cómo aumenta el tiempo de entrenamiento y que no siempre se obtienen mejores resultados para un mismo problema. Esto también nos enseña que, para problemas sencillos, los métodos de *machine learning* clásicos pueden ser suficientes a cambio de una menor necesidad de cálculo.



5.8. Referencias bibliográficas

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. & Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Helion*, 4(11). Disponible en <https://doi.org/10.1016/j.heliyon.2018.e00938>

Adadi, A. & Berrada, M. (2018). Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6, 52138-52160.

Alanis, A. Y., Arana-Daniel, N. & Lopez-Franco, C. (2019). Artificial Neural Networks for Engineering Applications. Disponible en <https://www.sciencedirect.com/science/book/9780128182475>

Anwar, S. M., Majid, M., Qayyum, A., Awais, M., Alnowami, M. & Khan, M. K. (2018). Medical Image Analysis using Convolutional Neural Networks: A Review. *Journal of Medical Systems*, 42(11), 226. Disponible en <https://doi.org/10.1007/s10916-018-1088-1>

Costa-jussà, M. R., Allauzen, A., Barrault, L., Cho, K. & Schwenk, H. (2017). Introduction to the special issue on deep learning approaches for machine translation. *Computer Speech & Language*, 46, 367-373.

Deng, L. & Liu, Y. (Eds.). (2018). *Deep learning in natural language processing*. Springer.

Fayek, H. M., Lech, M. & Cavedon, L. (2017). Evaluating deep learning architectures for Speech Emotion Recognition. *Neural Networks*, 92, 60-68.

Gao, H., Kong, D., Lu, M., Bai, X. & Yang, J. (2018). *Attention Convolutional Neural Network for Advertiser-level Click-through Rate Forecasting*. Proceedings of the 2018 World Wide Web Conference, 1855–1864. Disponible en <https://doi.org/10.1145/3178876.3186184>

García, R., Verdú, E., Regueras, L. M., de Castro, J. P. & Verdú, M. J. (2013). A neural network based intelligent system for tile prefetching in web map services.

Expert Systems with Applications, 40(10), 4096-4105. Disponible en <https://doi.org/10.1016/j.eswa.2013.01.037>

Graupe, D. (2019). *Principles Of Artificial Neural Networks: Basic Designs To Deep Learning* (4th Edition). World Scientific Publishing Company. Disponible en <https://books.google.co.uk/books?id=77uSDwAAQBAJ>

Graves, A., Mohamed, A. & Hinton, G. (2013). *Speech recognition with deep recurrent neural networks*. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645-6649. Disponible en <https://doi.org/10.1109/ICASSP.2013.6638947>

Luan, F., Paris, S., Shechtman, E. & Bala, K. (2017). *Deep photo style transfer*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4990-4998).

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Negnevitsky, M. (2011). *Artificial intelligence: A guide to intelligent systems* (3. ed.). Addison Wesley/Pearson.

Pouyanfar, S., Sadiq, S., Yan, Y., Tian, H., Tao, Y., Reyes, M. P., Shyu, M.-L., Chen, S.-C. & Iyengar, S. S. (2018). *A Survey on Deep Learning: Algorithms, Techniques, and Applications*. Association for Computing Machinery. Disponible en <https://doi.org/10.1145/3234150>

Ptucha, R., Petroski Such, F., Pillai, S., Brockler, F., Singh, V. & Hutkowski, P. (2019). Intelligent character recognition using fully convolutional neural networks. *Pattern Recognition*, 88, 604-613. Disponible en <https://doi.org/10.1016/j.patcog.2018.12.017>

Razzak, M. I., Naz, S. & Zaib, A. (2018). Deep learning for medical image processing: Overview, challenges and the future. In *Classification in BioApps* (pp. 323-350). Cham: Springer.

Rogers, S. & Girolami, M. (2017). *A first course in machine learning* (Second Edition). CRC Press, Taylor & Francis Group, a Chapman & Hall book.

Rosenblatt, F. (1960). Perceptron Simulation Experiments. *Proceedings of the IRE*, 48(3), 301-309. Disponible en <https://doi.org/10.1109/JRPROC.1960.287598>

Roy, A., Sun, J., Mahoney, R., Alonzi, L., Adams, S. & Beling, P. (2018). *Deep learning detecting fraud in credit card transactions*. 2018 Systems and Information Engineering Design Symposium (SIEDS), 129-134. Disponible en <https://doi.org/10.1109/SIEDS.2018.8374722>

Shah, S. A. A., Bennamoun, M. & Boussaid, F. (2016). Iterative deep learning for image set based face and object recognition. *Neurocomputing*, 174, 866-874.

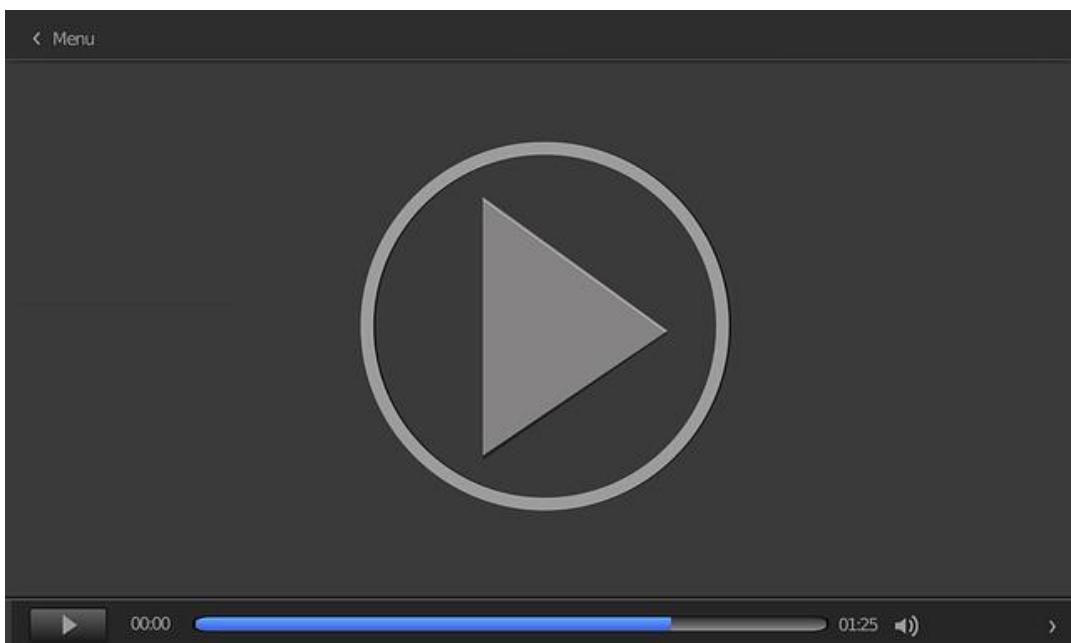
Wu, Z., Shen, C. & Van Den Hengel, A. (2019). Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90, 119-133.

Zhang, Z., Geiger, J., Pohjalainen, J., Mousa, A. E. D., Jin, W. & Schuller, B. (2018). Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology (TIST)* , 9(5), 1-28.

Zhang, C., Patras, P. & Haddadi, H. (2019). Deep Learning in Mobile and Wireless Networking: A Survey. *IEEE Communications Surveys Tutorials*, 21(3), 2224-2287. Disponible en <https://doi.org/10.1109/COMST.2019.290489>

Redes neuronales en Weka

Esta lección magistral consiste en un tutorial sobre el clasificador de Weka MultilayerPerceptron, que permite diseñar redes neuronales multicapa de retropropagación y realizar tareas de clasificación con las mismas.



05. Redes neuronales en Weka

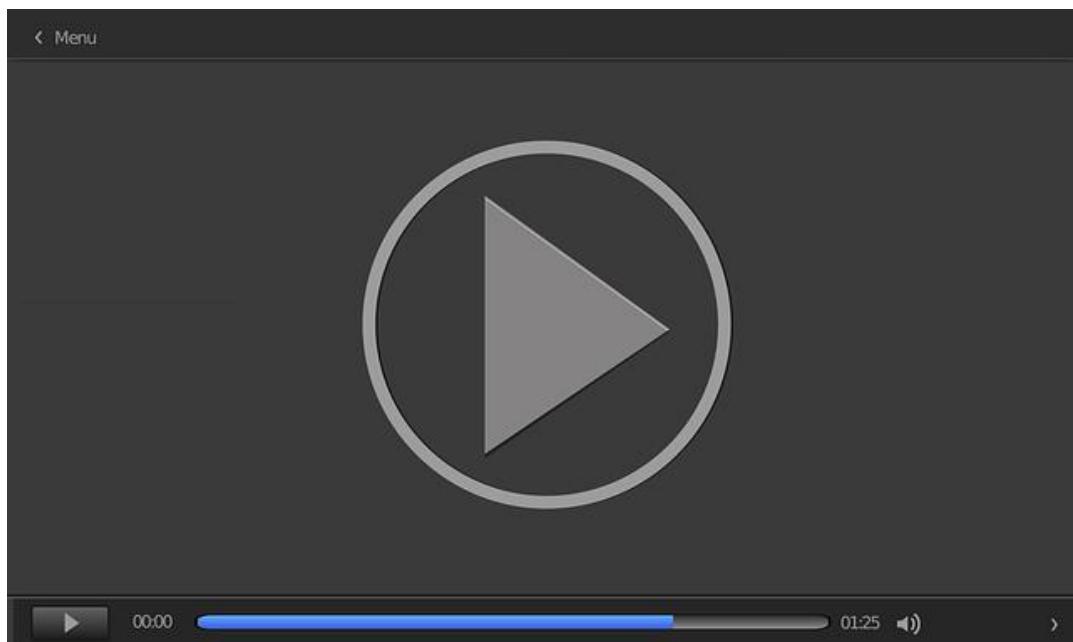
Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=72e16235-2630-4f56-b8e5-aff800fa62fc>

Redes neuronales aplicadas al reconocimiento de imágenes

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:
<https://www.youtube.com/watch?v=vBxjhMzKgOA>

Aitor Moreno de Ibermática explica cómo una red neuronal artificial puede utilizarse para el reconocimiento y la catalogación de imágenes, así como su aplicación en diferentes ámbitos (control de presencia en una empresa, catalogación de productos en un supermercado, etc.).



Accede al vídeo:

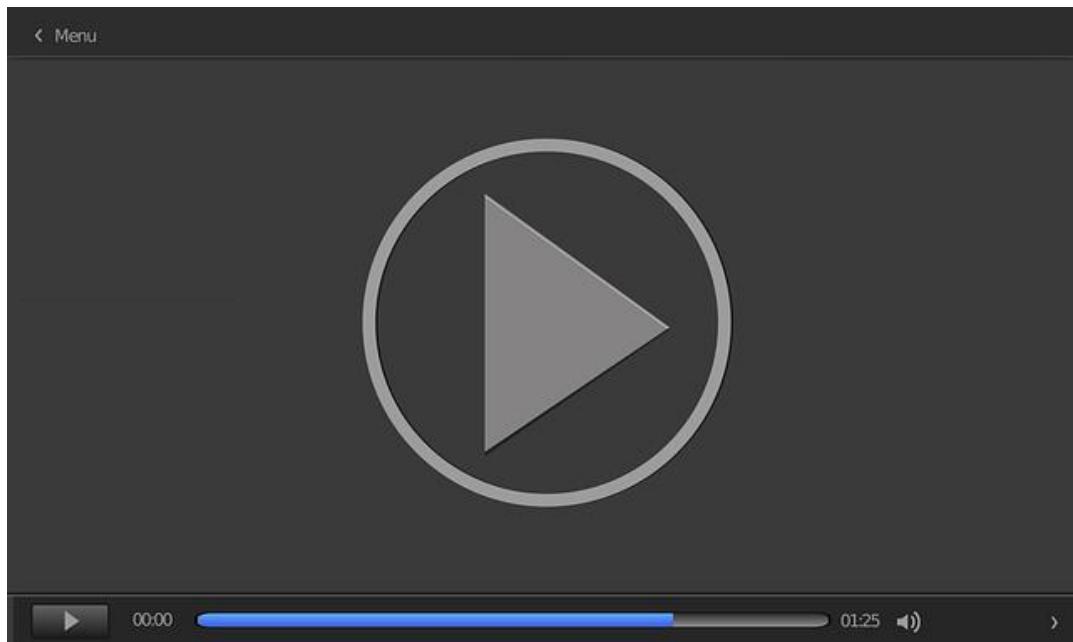
<https://www.youtube.com/embed/vBxjhMzKgOA>

Aplicaciones de las redes neuronales artificiales

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=d4t0WGjuYTs>

Jossué Rodríguez de la Universidad de Panamá realiza un recorrido por las diversas aplicaciones de las redes neuronales en distintos campos.



Accede al vídeo:

<https://www.youtube.com/embed/d4t0WGjuYTs>

Aplicación de las redes neuronales artificiales en el diagnóstico médico

Amato, F., López, A., Peña-Méndez, E.M., Vanhara, P., Hampl, A. & Havel, J. (2013). Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine*, 11(2), 47-113. http://www.zsf.jcu.cz/jab_old/11_2/havel.pdf

El artículo incluye una explicación de las redes neuronales y presenta el uso general de este tipo de técnicas en el diagnóstico médico mediante ejemplos, comentando posibilidades y limitaciones.

Redes neuronales

Accede a la página desde el aula virtual o a través de la siguiente dirección web:
<http://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/neural-networks/index.html>

Sitio web dedicado a las redes neuronales. Es particularmente interesante la sección dedicada a aplicaciones de las redes neuronales, así como la sección de recursos que contiene enlaces a materiales interesantes.

1. Indica cuáles de las siguientes afirmaciones son correctas:

 - A. Las neuronas artificiales simulan el comportamiento de las neuronas biológicas.
 - B. Las neuronas artificiales pueden tener varias salidas que toman valores diferentes.
 - C. Las neuronas artificiales pueden tener varias entradas que toman valores diferentes.
 - D. Las entradas de las neuronas artificiales se ponderan con un peso.
2. Selecciona las opciones correctas. En la fase de entrenamiento de una red neuronal:

 - A. Se ajusta el número de capas de la red neuronal.
 - B. Se ajustan los pesos de los enlaces que conectan las neuronas.
 - C. Se ajusta el número de neuronas por capa de la red neuronal.
 - D. Ninguna de las anteriores.
3. Indica cuál de las afirmaciones siguientes son correctas respecto a la regla de aprendizaje del perceptrón:

 - A. Se utiliza para calcular las salidas de la red.
 - B. Utiliza la diferencia entre la salida real y la salida esperada para ajustar los pesos.
 - C. Se puede establecer una tasa de aprendizaje en la regla que pondrá la relevancia del último peso calculado.
 - D. Se utiliza para ajustar los pesos de la red.

4. Indica cuáles de las siguientes funciones pueden ser utilizadas como funciones de activación en redes neuronales:

- A. Función escalón.
- B. Función sigmoide.
- C. Función tangente hiperbólica.
- D. Todas las anteriores son correctas.

5. Selecciona la opción más correcta. El funcionamiento de una red neuronal queda determinado por:

- A. El número de capas, número de neuronas y las conexiones entre neuronas.
- B. La función de activación, el número de capas, el número de neuronas y los pesos entre las neuronas.
- C. El algoritmo de aprendizaje, la regla de aprendizaje, el número de capas, número de neuronas y las conexiones entre neuronas.
- D. El algoritmo de aprendizaje, la función de activación, el número de capas, número de neuronas y las conexiones entre neuronas.

6. Indica cuáles de las siguientes afirmaciones son correctas:

- A. Las redes neuronales multicapa son redes bidireccionales.
- B. Las redes neuronales multicapa son redes asociativas.
- C. Las redes neuronales multicapa son redes de alimentación hacia adelante.
- D. Las redes neuronales multicapa son redes unidireccionales.

7. Indica cuáles de las siguientes afirmaciones son correctas respecto de las redes multicapa de retropropagación:

- A. Todas las neuronas de una capa se conectan con todas las neuronas de las capas posteriores.
- B. Para el ajuste de los pesos se utiliza el gradiente del error.
- C. Para el ajuste de los pesos se utiliza el error cuadrático medio.
- D. Se utilizan las mismas fórmulas para calcular los pesos relativos a las neuronas de salida y los relativos a las neuronas ocultas.

8. Indica cuáles de las siguientes afirmaciones son correctas respecto de las redes recurrentes:

- A. Emulan la capacidad de almacenamiento de la memoria humana.
- B. Emulan la capacidad de asociación de la memoria humana.
- C. Las salidas de la red alimentan las entradas.
- D. La red Hopfield es una red recurrente.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto de las redes Hopfield:

- A. El estado de la red viene determinado por los pesos de los enlaces.
- B. El objetivo es que la red sea capaz de almacenar unos determinados estados.
- C. El objetivo es que la red sea capaz de almacenar unas memorias fundamentales.
- D. No es preciso comprobar que la red sea capaz de almacenar memorias fundamentales.

10. Indica cuáles de las siguientes afirmaciones son correctas respecto de las redes Hopfield:

- A. Son redes bidireccionales asociativas.
- B. Se utilizan en reconocimiento de imágenes.
- C. Siempre alcanzan un estado estable correspondiente a una memoria fundamental.
- D. Con pocas neuronas y conexiones son capaces de almacenar una gran cantidad de información.

Técnicas de Inteligencia Artificial

Tema 6. Deep learning

Índice

[Esquema](#)

[Ideas clave](#)

6.1. ¿Cómo estudiar este tema?

6.2. El papel del deep learning dentro del machine learning

6.3. Redes neuronales y deep learning

6.4. Redes prealimentadas profundas

6.5. Redes neuronales recurrentes profundas

6.6. Autoencoders

6.7. Redes neuronales convolucionales

6.8. Redes generativas antagónicas

6.9. Aprendizaje por refuerzo

6.10. Aprendizaje por refuerzo profundo

6.11. Ejemplos de implementación

6.12. Referencias bibliográficas

[A fondo](#)

[Introduction to deep learning](#)

[A beginner intro to convolutional neural networks](#)

[Conceptos básicos de deep learning](#)

[Estado del arte del deep learning](#)

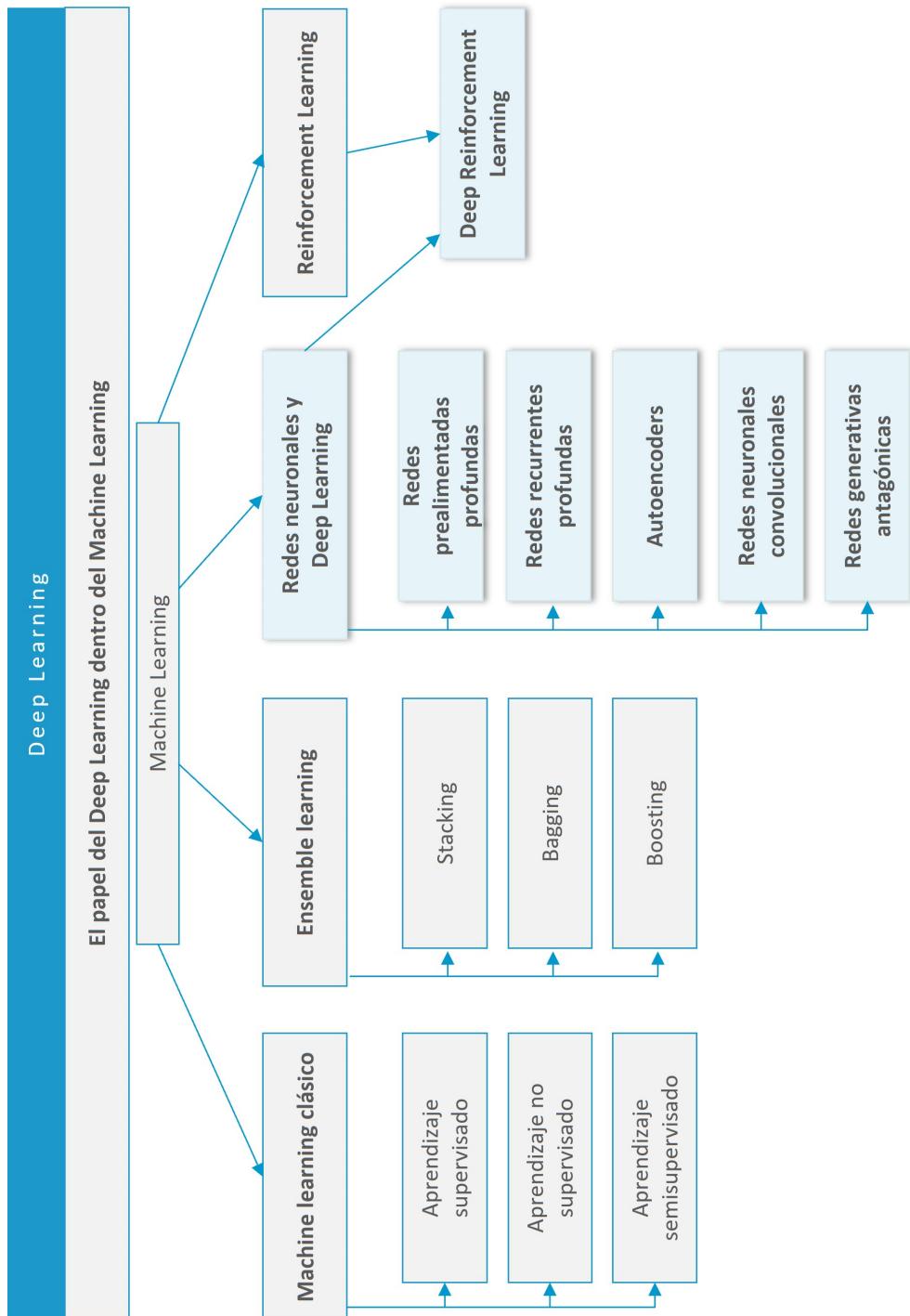
[The neural network zoo](#)

[Guía inicial de TensorFlow 2.0 para principiantes](#)

[Keras](#)

CS231n convolutional neural networks for visual
recognition

[Test](#)



6.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio con el material adicional que se facilita al final de este tema.

Existe una frecuente confusión en los medios *mainstream* e incluso en la publicitación de servicios y software sobre los términos *inteligencia artificial*, *machine learning* (aprendizaje automático o aprendizaje automatizado) y *deep learning* (aprendizaje profundo o aprendizaje automático profundo). En este sentido, antes de hablar sobre el *deep learning* es un buen momento para que nos paremos a poner a cada uno de estos términos en su lugar.

Como ya hemos visto desde el Tema 1 de esta asignatura, la Inteligencia Artificial tiene como objetivo desarrollar y utilizar sistemas informáticos que intentan reproducir los procesos de la inteligencia humana. Estos procesos de inteligencia humana son necesarios para el aprendizaje, la comprensión, la resolución de problemas o la toma de decisiones (Panesar, 2019). La inteligencia artificial es, por tanto, una amplia disciplina que reúne varios campos como el procesamiento del lenguaje natural o *Natural Language Processing* (Lima, de Castro y Corchado, 2015), los sistemas expertos (Faia *et al.*, 2018), los sistemas multiagente (Alonso *et al.*, 2013), los sistemas recomendadores (García *et al.*, 2017), el análisis de voz y la conversión a texto (habla a texto y texto a habla) (Partila *et al.*, 2018), la visión por computador, visión artificial o *computer vision* (Seo *et al.*, 2015), los sistemas de planificación, la computación evolutiva (Pal y Wang, 2017), la robótica y, en el caso de los anteriores temas y el que nos ocupa, el aprendizaje automático o *machine learning*.

Por supuesto, estas ramas no son del todo estancas entre sí, existiendo, a modo de ejemplo, algoritmos de planificación aplicados mediante sistemas multiagente, técnicas de procesamiento del lenguaje natural basadas en *deep learning* (Young et al., 2018), o redes neuronales convolucionales aplicadas a la visión artificial (Rivas et al., 2018), por citar solo algunos ejemplos.

6.2. El papel del deep learning dentro del machine learning

Es importante comprender además el papel que desempeña el aprendizaje automático dentro de las técnicas de inteligencia artificial. En este sentido, existen varias definiciones de inteligencia artificial que según Russell y Norvig (2002) pueden dividirse en cuatro grupos: **pensar racionalmente** (relacionado con «capas de pensamiento»), **pensar humanamente** (relacionado con los sistemas cognitivos), **actuar racionalmente** (relacionado con los agentes racionales) y **actuar humanamente**, lo cual está relacionado con el conocido test de Turing (Turing, 1950), en el que un sistema se denomina inteligente si es capaz de responder a una serie de preguntas escritas y un humano no sería capaz de diferenciar si las respuestas han sido dadas por un humano o por un ordenador.

La definición «pensar humanamente» incluye conceptos y técnicas como el procesamiento del lenguaje natural, la comunicación con el ser humano, la representación del conocimiento, el almacenamiento de lo que un ser humano necesita para escuchar, el razonamiento automático, para sacar conclusiones sobre la información almacenada y, en nuestro estudio, el aprendizaje automático, para adaptarse a nuevas circunstancias para detectar y extraer patrones. Además, se incluiría la visión por ordenador y la robótica para percibir y manipular objetos en el mundo real.

Así, el aprendizaje automático es una rama de la inteligencia artificial que se refiere a la construcción de programas informáticos que mejoran automáticamente su rendimiento en una tarea determinada con la experiencia, como ya vimos en el Tema 1. En el aprendizaje automático, los algoritmos se utilizan para analizar y aprender de los datos. Después de eso, los algoritmos realizan predicciones y toman decisiones sobre eventos en el mundo real. Es decir, estos algoritmos toman una gran cantidad

de datos de entrenamiento (una lista de instancias) para entrenar al modelo sobre la realidad y aprender. Más tarde, este modelo puede aplicarse para hacer predicciones o tomar decisiones sobre las nuevas instancias que se encuentren. Ya hemos visto en anteriores temas cómo realizar esta tarea mediante técnicas de *machine learning clásico* como los árboles de decisión y las reglas o incluso con técnicas consideradas actualmente fuera del *machine learning* clásico, como son las técnicas ensemble (como el *random forest*) o las redes neuronales.

Los primeros algoritmos de aprendizaje de máquinas se remontan en realidad a los años 50, cuando Samuel (1959) demostró en 1956 la capacidad de los programas informáticos para aprender jugando a las damas que funcionaban en un IBM 701, un ordenador del tamaño de una cama de matrimonio. En 2016, gracias a los avances en el aprendizaje profundo, las computadoras son capaces de vencer a otras computadoras el 99,8 % de las veces y ganar a campeones humanos en el juego de Go (Silver *et al.*, 2016). Esto es particularmente impresionante si tenemos en cuenta que se estima que el número de posibles partidas en este juego oriental excede con creces al número de átomos existente en el universo observable. ¿Cómo entrenar a una máquina para esto, si no podemos ni quiera crear *datasets* de tamaño comparable? Veremos más adelante cómo esto es posible.

Los algoritmos de aprendizaje automático pueden subdividirse en seis categorías principales, tal y como puede observarse en la :

- ▶ **Aprendizaje automático clásico** (*classic machine learning*), que engloba:
 - **Aprendizaje supervisado** (*supervised learning*).
 - **Aprendizaje no supervisado** (*unsupervised learning*).
 - **Aprendizaje semisupervisado** (*semisupervised learning*).
- ▶ **Aprendizaje ensemble o aprendizaje integrado** (*ensemble learning*).

- ▶ **Aprendizaje profundo** (*deep learning*).
- ▶ **Aprendizaje por refuerzo** (*reinforcement learning*).

Por supuesto, no es la única forma de clasificar los algoritmos de aprendizaje automático, y esta clasificación varía a lo largo del tiempo en función de la importancia de cada uno de los grupos.

En el **aprendizaje supervisado**, los datos de entrenamiento utilizados por el algoritmo para construir el modelo incluyen casos en que la clase o el resultado previsto es numérico o ha sido etiquetado previamente (generalmente por un humano), como sucede, por ejemplo, en los problemas de regresión o clasificación, respectivamente. Ejemplo de esto son los árboles de decisión, las reglas de clasificación o las SVM (*Support Vector Machines*), ya vistos en temas anteriores.

En el **aprendizaje no supervisado**, los datos de entrenamiento no están etiquetados y se desconoce su resultado esperado. Los algoritmos de aprendizaje no supervisado nacieron más tarde que los supervisados, alrededor de los años 90, y son útiles para descubrir la estructura del conjunto de datos, para agrupar los elementos del conjunto de datos por los elementos más representativos de cada uno o para comprimir los datos existentes. De hecho, este tipo de algoritmos se utilizan a menudo para preprocessar los datos antes de utilizarlos para alimentar un algoritmo supervisado o una red neuronal profunda. Ejemplos de algoritmos de aprendizaje no supervisado son la agrupación (*clustering*, que veremos en el Tema 7), la reducción de la dimensionalidad y el aprendizaje de las reglas de asociación (ya visto en el Tema 3).

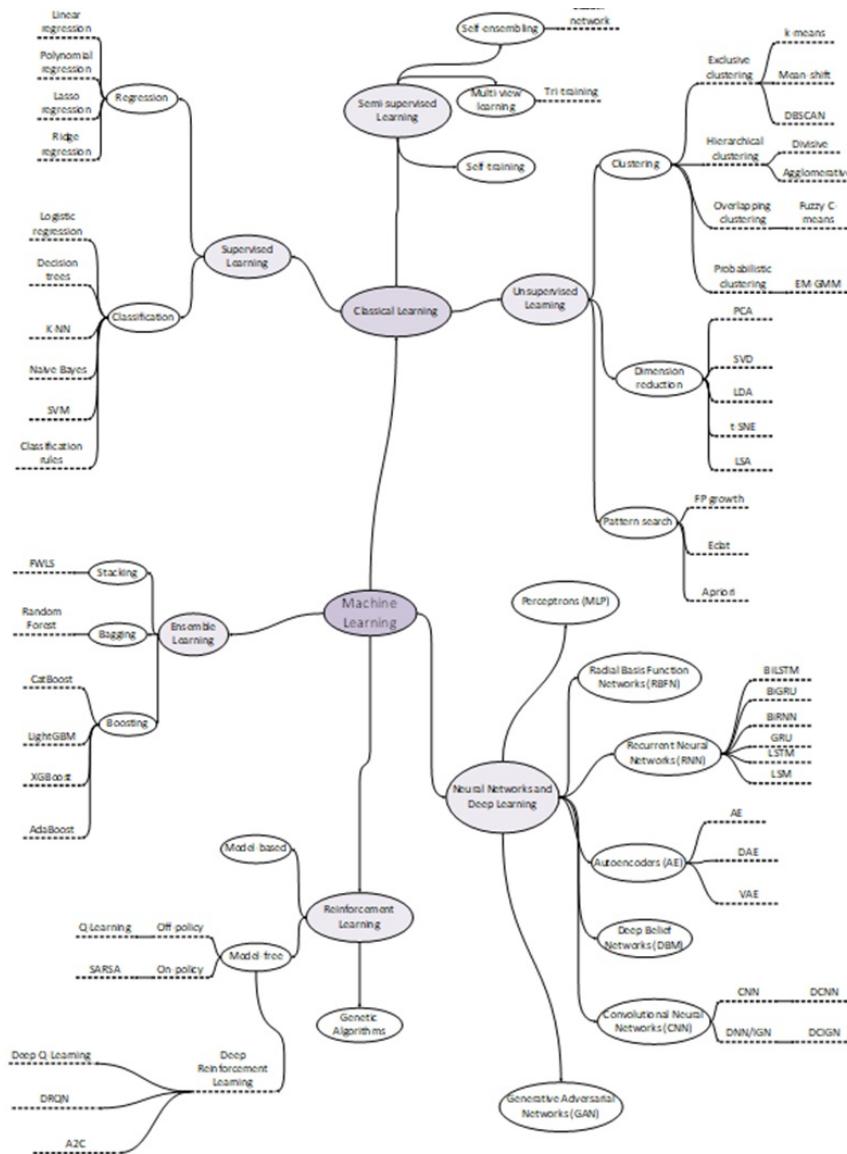


Figura 1. Principales ramas y algoritmos del machine learning.

En el **aprendizaje semisupervisado**, algunos de los datos de capacitación de entrada están etiquetados y otros no. La principal desventaja de los algoritmos de aprendizaje supervisado es que es necesario que el conjunto de datos de formación sea etiquetado por un humano, un científico de datos, un ingeniero de aprendizaje de máquinas o tal vez un grupo externo de trabajadores que utilicen plataformas de *crowdsourcing* (como, por ejemplo, *Amazon Mechanical Turk*) lo que, en cualquier

caso, es un proceso costoso, especialmente para grandes volúmenes de datos. Por otra parte, el problema de los algoritmos no supervisados es que, al menos en la actualidad, el alcance de sus aplicaciones es limitado. El aprendizaje semisupervisado trata de resolver estos inconvenientes combinando datos de formación etiquetados (normalmente una pequeña porción del total) y datos de formación no etiquetados (normalmente la mayoría de los datos de formación). El procedimiento habitual consiste en aplicar un algoritmo no supervisado, como la agrupación, para agrupar los datos y utilizar los datos etiquetados para etiquetar el resto de los datos no etiquetados (Van Engelen y Hoos, 2020).

En realidad, el ***machine learning clásico***, que nació en el decenio de 1950 a partir de métodos estadísticos, se sigue utilizando ampliamente hoy en día y es útil para resolver muchos de los problemas que se plantean. Sin embargo, con el tiempo han surgido nuevos paradigmas como el ***aprendizaje ensemble***, cuando la calidad es un problema real; las redes neuronales y el ***aprendizaje profundo o deep learning***, cuando tenemos datos complejos con características poco claras o cuando queremos alcanzar niveles más altos de precisión que el aprendizaje automático clásico; o el ***aprendizaje por refuerzo o reinforcement learning***, cuando no hay datos de entrada, pero hay un entorno con el que se puede interactuar, y veremos al final de este tema tras ver las diferentes técnicas *deep learning*, para comprender mejor conceptos como el ***aprendizaje por refuerzo profundo o deep reinforcement learning***.

6.3. Redes neuronales y deep learning

Las redes neuronales son, sin duda, una de las técnicas de aprendizaje de máquinas por excelencia y con mayor potencial en la actualidad. Aunque las primeras propuestas de redes neuronales o modelos conexionistas se remontan al decenio de 1950, cuando el perceptrón (la neurona artificial básica más conocida) fue inventado por Rosenblatt (1958), a finales del decenio de 1960 se abandonó el conexionismo en favor del razonamiento simbólico, en parte y según la literatura (Yasnitsky, 2019) debido a la publicación de las obras de Marvin y Seymour (1969).

No fue hasta finales de los años 70 que terminó el *primer invierno de la inteligencia artificial*, con la invención del método de retropropagación y la aparición de sistemas expertos durante los años 80. Al *segundo invierno de la inteligencia artificial* de finales del decenio de 1980 y principios del decenio de 1990 siguió la aparición de agentes inteligentes y el aprendizaje automático basado en métodos estadísticos, incluidas las primeras menciones al aprendizaje profundo de Dechter (1986).

Es necesario considerar que si las redes neuronales artificiales no tuvieron éxito hace décadas y ahora sí es porque, en primer lugar, tenemos una gran capacidad de computación utilizando CPU y GPU, que permiten un alto paralelismo de computación, e incluso TPU, unidades especializadas en computación tensorial (Jouppi *et al.*, 2018). En segundo lugar, ahora contamos con una gran capacidad y técnicas de almacenamiento que antes no estaban disponibles, lo que nos permite entrenar las redes neuronales obteniendo una gran precisión en las clasificaciones y predicciones. Estas dos cuestiones clave han apoyado la gran evolución de las redes neuronales artificiales y las técnicas de aprendizaje profundo en los últimos años.

Se puede definir el **aprendizaje profundo** o **deep learning** como una clase de algoritmos de aprendizaje automático que utiliza múltiples capas para extraer progresivamente características de nivel superior de la entrada bruta. Esto incluye, por tanto, una cascada de capas conectadas entre sí con unidades de procesamiento no lineal en cada una de las capas. De este modo, las primeras capas (inferiores) se corresponden con niveles de abstracción menor, mientras que las últimas capas (superiores) se basan en las anteriores para formar una representación jerárquica de conceptos.

Por ejemplo, en el procesamiento de imágenes empleando redes convolucionales, como veremos más adelante, las primeras capas (también llamadas inferiores) pueden identificar los bordes, mientras que las capas finales (también llamadas superiores) pueden identificar los conceptos relevantes para un humano, como los dígitos o las letras o las caras.

Sin embargo, **en la realidad, apenas existe una clara diferencia entre las redes neuronales artificiales consideradas clásicas y las de aprendizaje profundo o las redes neuronales consideradas profundas** (LeCun, Bengio y Hinton, 2015). Las herramientas y bibliotecas utilizadas para su formación y producción final son las mismas (por ejemplo, TensorFlow, Keras, Torch / PyTorch, Caffee, etc.) y ya suelen considerarse como una rama única dentro del aprendizaje automático.

Mediante las redes neuronales artificiales es posible aplicar técnicas de aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semisupervisado e incluso técnicas de aprendizaje por refuerzo como veremos más adelante.

De esta manera, las redes neuronales pueden utilizarse como sustitutos de los métodos supervisados, no supervisados y semisupervisados, algunos vistos en temas anteriores y algunos que se verán a continuación, obteniendo una precisión mucho mayor (obviamente, a cambio de un mayor coste de computación tanto en la etapa de entrenamiento como en la etapa de *testing* o producción). Además de para resolver cualquier problema que ya podíamos mediante los métodos de *machine learning* clásico, también se utilizan, entre otras posibles aplicaciones, para la **identificación de objetos en imágenes y vídeos** (Shah, Bennamoun y Boussaid, 2016), el **reconocimiento de voz** (Zhang *et al.* 2018), la **síntesis de voz** (Arik, *et al.* 2017), **análisis de sentimientos y reconocimiento de emociones del habla** (Fayek, Lech y Cavedon, 2017), **procesamiento de imágenes** (Razzak, Naz y Zaib, 2018), **transferencia de estilos** (por ejemplo, aplicación del estilo de pintura de Van Gogh a cualquier fotografía) (Luan *et al.*, 2017), **procesamiento del lenguaje natural (*Natural Language Processing*)** (Costa-jussà *et al.*, 2017), **traducción automática** (Deng y Liu, 2018), etc.

Existe una amplia y creciente variedad de redes neuronales artificiales clasificadas según su arquitectura (Liu *et al.*, 2017). Entre las más relevantes se encuentran:

- ▶ **Redes prealimentadas (*Feedforward networks*)** y redes prealimentadas profundas (*Deep Feedforward Networks*).
- ▶ **Redes Neuronales Recurrentes (RNN – *Recurrent Neural Networks*)** y Redes Recurrentes Profundas (*Deep Recurrent Networks*).
- ▶ **Autoencoders (AE)**.
- ▶ **Redes Neuronales Convolucionales (CNN – *Convolutional Neural Networks*)**.
- ▶ **Redes Generativas Antagónicas (GAN – *Generative Adversarial Networks*)**.

Además de estas grandes tendencias, hay muchos otros tipos que deben ser mencionados. Las redes de creencias profundas o *Deep Belief Networks* (DBN) (Bengio, et al., 2007) son arquitecturas apiladas de mayormente RBMs (*Restricted Boltzmann Machines* o máquinas Boltzmann restringidas) o de autocodificadores variacionales (VAE), que veremos más adelante. En realidad, ya hemos hablado de las redes de creencias en el Tema 4, cuando mencionamos los clasificadores *Naïve Bayes*, pues son uno de los casos más simples de red de creencias o redes bayesianas. Esta simplificación viene de considerar que todas las características son fuertemente independientes entre sí. Las máquinas Boltzmann restringidas son arquitecturas que no son realmente redes neuronales, así como las *cadenas de Markov* (MC) o las *cadenas discretas de Markov* (DTMC) o las máquinas Boltzmann (BM), en las que se basan las RBM.

6.4. Redes prealimentadas profundas

Ya vimos en el Tema 5 algunas redes prealimentadas (*Feedforward Networks*), como los perceptrones multicapa (MLP – *Multilayer Perceptron*). En una red prealimentada, todas las neuronas de una capa están conectadas con todas las neuronas de la capa siguiente, no existe conexiones entre neuronas de la misma capa y no existe retroalimentación.

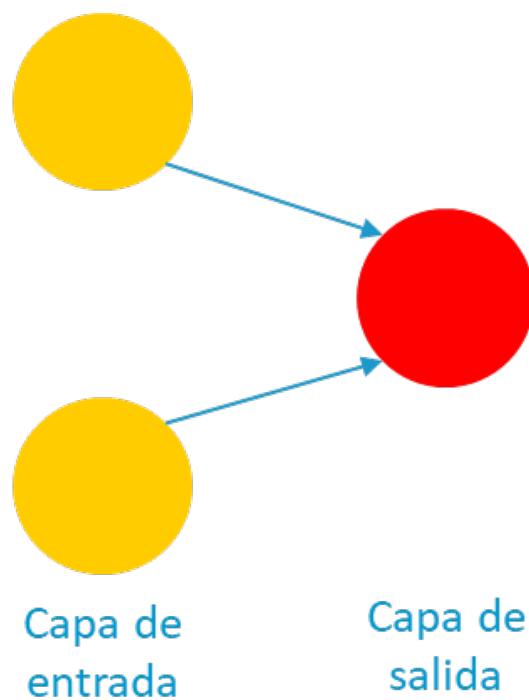


Figura 2. Perceprón.

Como ya se ha mencionado, el perceprón fue creado en los años 50 por Rosenblatt (1958). En un perceprón, mostrado en la Figura 2, hay varias señales de entrada x_i , que son las señales de entrada a toda la red neural artificial o provienen de otros perceptrones. Estas entradas son ponderadas por un conjunto de pesos w_i . La salida Y del perceprón es única, y su valor vendrá determinado por la función de activación elegida, por ejemplo, la función signo. Si la suma ponderada de las entradas supera

un umbral w_0 , entonces la salida del perceptrón tendrá un valor de +1, de lo contrario tendrá un valor de -1. Es decir:

$$y_k = \phi(u) (1)$$

Considerando u la suma ponderada de los pesos:

$$u = w_0 + \sum_{i=1}^n w_i x_i (2)$$

Usando en el ejemplo la función signo como función de activación:

$$y = \begin{cases} +1, & u \geq 0 \\ -1, & u < 0 \end{cases} (3)$$

Aunque existen otras posibles funciones de activación, como la función escalón de Heaviside:

$$y = \begin{cases} +1, & u \geq 0 \\ 0, & u < 0 \end{cases} (4)$$

Una función sigmoide, como la función logística:

$$y = \frac{1}{1 + e^{-u}} = \frac{e^u}{e^u + 1} (5)$$

O, una combinación lineal de las entradas, entre otras muchas posibilidades:

$$y = w_0 + \sum_{i=1}^n w_i x_i (6)$$

Una de las formas de entrenar una neurona básica o perceptrón es la regla de aprendizaje del perceptrón descrita por Rosenblatt (1960). Si la entrada al perceptrón es:

$$X = w_0 + \sum_{i=1}^n w_i x_i (7)$$

Usando en el ejemplo la función signo como función de activación:

$$y = \begin{cases} +1, & X \geq 0 \\ -1, & X < 0 \end{cases} \quad (8)$$

Dados los datos de entrenamiento (entradas a la red y correspondientes salidas conocidas), el entrenamiento consiste en averiguar los pesos del perceptrón que mejor se ajustan a esas entradas y salidas, mediante una serie de iteraciones que ajustan los valores de los pesos. Inicialmente los pesos se eligen al azar. En cada iteración, para cada entrada del entrenamiento, obtendremos una salida que será generalmente diferente de la conocida o esperada:

$$w_i(t+1) = w_i(t) + \alpha x_i(t) e(t) \quad (9)$$

Donde $e(t)$ es la diferencia entre la salida esperada y la salida real en la entrada procesada en la iteración t . Por su parte, α es la tasa de aprendizaje (un valor entre 0 y 1). En cada iteración se procesa uno de los datos de aprendizaje disponibles. El perceptrón se activa aplicando las entradas $xk(t)$ y la salida deseada $yd(t)$.

La salida real en la iteración se calcula utilizando la función de activación escogida, por ejemplo, la función signo de la Ecuación 8. A partir de la salida real obtenida, se actualizan los pesos utilizando la Ecuación 9 y se continúa con la siguiente iteración hasta alcanzar la convergencia.

De hecho, los propios perceptrones se conocen como perceptrones monocapa para diferenciarlos de los perceptrones multicapa, y cuando se utilizan con las funciones de activación de tipo escalonado de Heaviside actúan como clasificadores lineales binarios. Es decir, sirven para trazar una línea recta (si existe) que separa un conjunto de datos de entrenamiento en dos clases (Rodríguez *et al.*, 2008).

El **Perceptrón Multicapa (MLP)**, ya introducido en el Tema 5, es, por tanto, una de las redes neuronales prealimentadas (FFN – *Feedforward Networks*) formadas por varias capas de neuronas (al menos una capa intermedia), cada una de las cuales es

un perceptrón (De Paz, *et al.*, 2013). Son la arquitectura de red neuronal más sencilla (mostrada en la Figura 3), nacida en el decenio de 1980, y pueden utilizarse como clasificadores, así como en el reconocimiento del habla, el reconocimiento de imágenes y la traducción automática, entre muchos otros (Naraei, Abhari y Sadeghian, 2016). Sin embargo, en su momento fueron sustituidos en varias aplicaciones por los SVM debido a su mayor simplicidad.

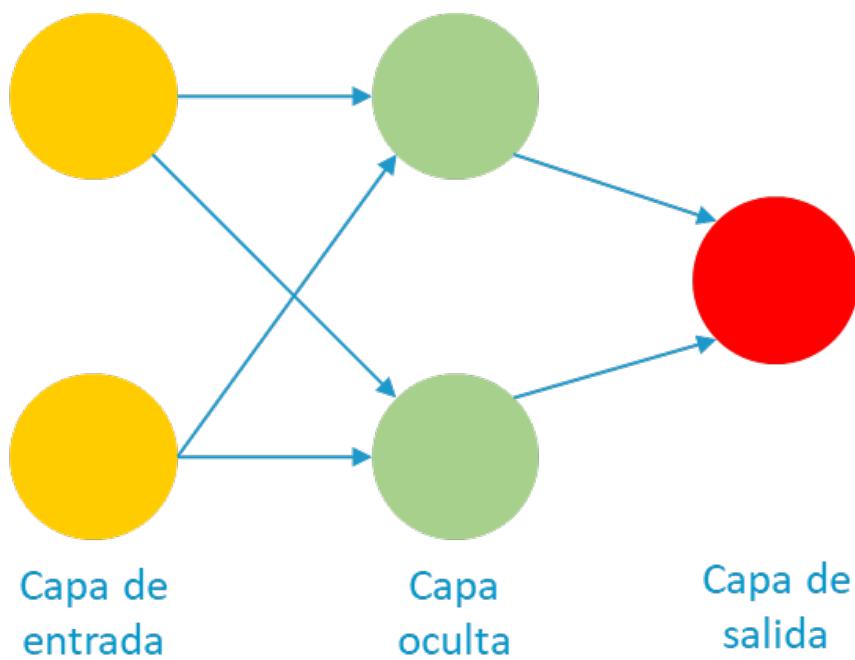


Figura 3. Redes prealimentadas (feedforward networks).

En los MLP, las funciones de activación más utilizadas son las sigmoidales, como la función logística o la tangente hiperbólica, y el entrenamiento se realiza mediante el método de aprendizaje supervisado conocido como **método de retropropagación** (Rumelhart y McClelland, 1986), ya explicado en el Tema 5, y válido en redes en las que todas las neuronas de cada capa se conectan con todas las neuronas de las capas anterior y posterior.

Así, una vez definida la arquitectura (número de capas, neuronas por capa y función de activación), se utiliza el **método del gradiente de error** para ajustar los pesos durante el entrenamiento:

$$\delta_k(t) = \frac{\partial y_k(t)}{\partial X_k(t)} e_k(t) = \frac{\partial y_k(t)}{\partial X_k(t)} (y_{d,k}(t) - y_k(t)) \quad (10)$$

Donde $X_k(t)$ es la entrada ponderada de la neurona k según la Ecuación , $y_k(t)$ la salida real de la neurona k en la iteración t utilizando una función de activación logística, $y_{d,k}$ la salida esperada en la neurona k y, finalmente, $e_k(t)$ es el error en la salida en la iteración t .

Para ajustar los pesos w_{jk} sobre los enlaces entre la neurona j y la neurona k en la capa de salida se utiliza la expresión:

$$w_{jk}(t+1) = w_{jk}(t) + \alpha y_j(t) \delta_k(t) \quad (11)$$

En el caso de una capa oculta, la expresión usada sería:

$$\delta_j(t) = y_j(t) (1 - y_j(t)) \sum_{k=1}^m \delta_k(t) w_{jk}(t) \quad (12)$$

Así, empezamos con pequeños valores aleatorios para los pesos. Para cada dato de entrada $x_k(t)$ se calculan los datos de salida en la red. Se calcula el gradiente de error para las neuronas de la capa de salida y se reinician sus pesos. El gradiente de error para las neuronas en la capa oculta se calcula y sus pesos se restablecen. El ciclo de iteraciones continúa así hasta que se cumple un criterio de parada específico.

En la práctica, especialmente cuando trabajamos con redes profundas con varias capas ocultas, como la mostrada (conceptualmente) en la Figura 4 se utilizan algoritmos de retropropagación modificados, debido a la elevada carga de cálculo del mecanismo original (Ramchoun *et al.*, 2016).

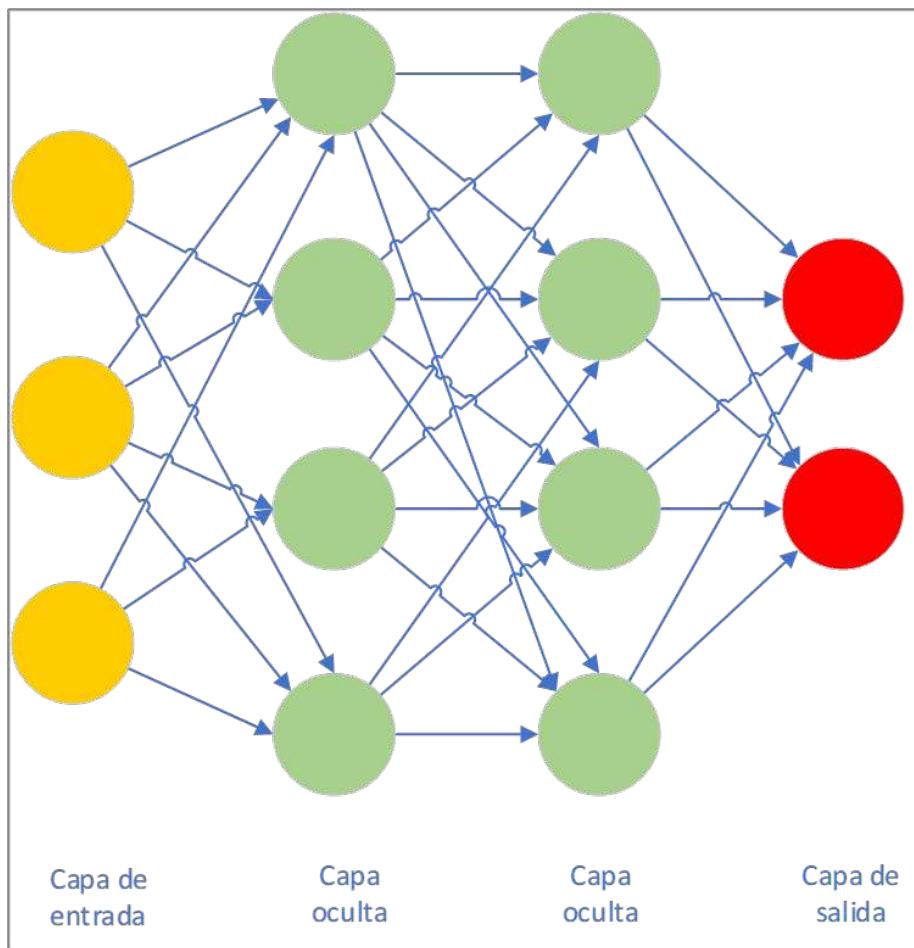


Figura 4. Deep feedforward networks (como perceptrones multicapa o las funciones de base radial).

Las **redes de funciones de base radial (RBF)** son otro tipo de FFNN (*feed forward neural networks*) (Broomhead y Lowe, 1988) que también permite su entrenamiento a través del método de retropropagación, cuya peculiaridad radica únicamente en el hecho de que las funciones de activación son funciones radiales. Es decir, una función real cuyo valor depende solo de la distancia al origen o a un centro alternativo, como la función distancia:

$$\phi(r) = r = \|X_j - X_k\| \quad (13)$$

O, la función gaussiana:

$$\phi(r) = e^{-(\epsilon r)^2} \quad (14)$$

6.5. Redes neuronales recurrentes profundas

Si bien las redes neuronales convolucionales son actualmente las más utilizadas en aplicaciones de imagen y vídeo, las **redes neuronales recurrentes** (Rivas *et al.*, 2019) son las más utilizadas en el campo de los problemas relacionados con el lenguaje, incluyendo la voz y la música. Estos tipos de redes son ideales para procesar datos secuenciales, y se utilizan ampliamente en la traducción automática, en genómica o en proteómica (Cao *et al.*, 2017), el reconocimiento de voz y la síntesis de voz en asistentes inteligentes, así como en la autocompletado de textos (Park y Chiba, 2017).

En problemas como la síntesis de voz a partir de un texto, por ejemplo, la pronunciación de cada sonido depende de los sonidos anteriores. En los sintetizadores antiguos cada sonido se pronunciaba individualmente, y esta era una de las razones por las que sonaban artificiales. Sin embargo, los sintetizadores de hoy en día suenan completamente naturales tanto en la pronunciación como en el acento.

Para ello, las redes neuronales recurrentes tienen neuronas específicas que incluyen una memoria en la que almacenan las entradas correspondientes a ciclos anteriores. De esta manera, son capaces de producir una salida dependiente de la entrada actual y de las entradas anteriores.

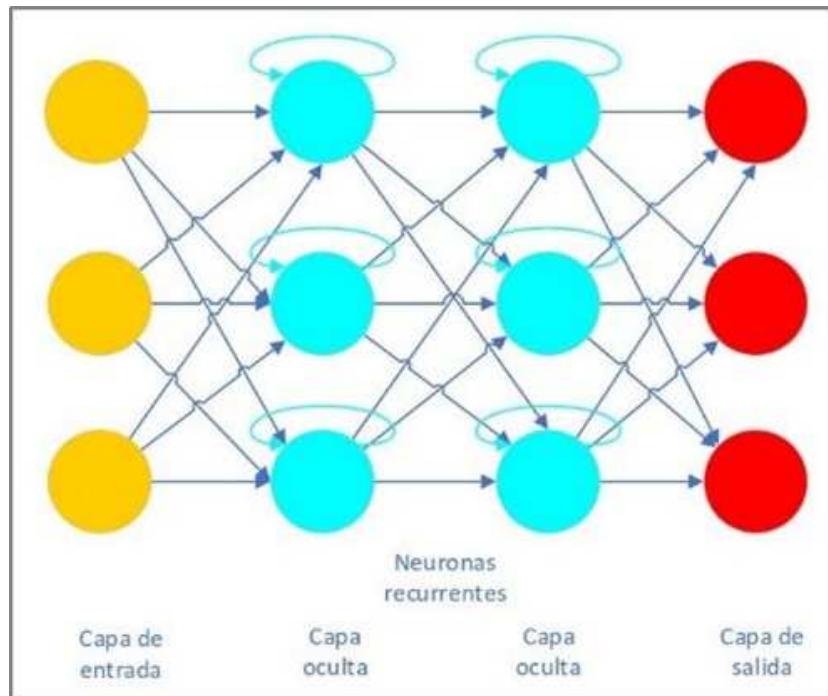


Figura 5. Redes Neuronales Recurrentes (RNN).

Con el tiempo, las **Memorias de Largo y Corto Plazo (LSTM – Long and Short-Term Memories)** (Kim *et al.*, 2016) surgieron para mejorar este concepto. En este tipo de red hay celdas especiales en las que un valor puede ser almacenado, leído o restablecido, por medio de puertas de entrada, salida y olvido.

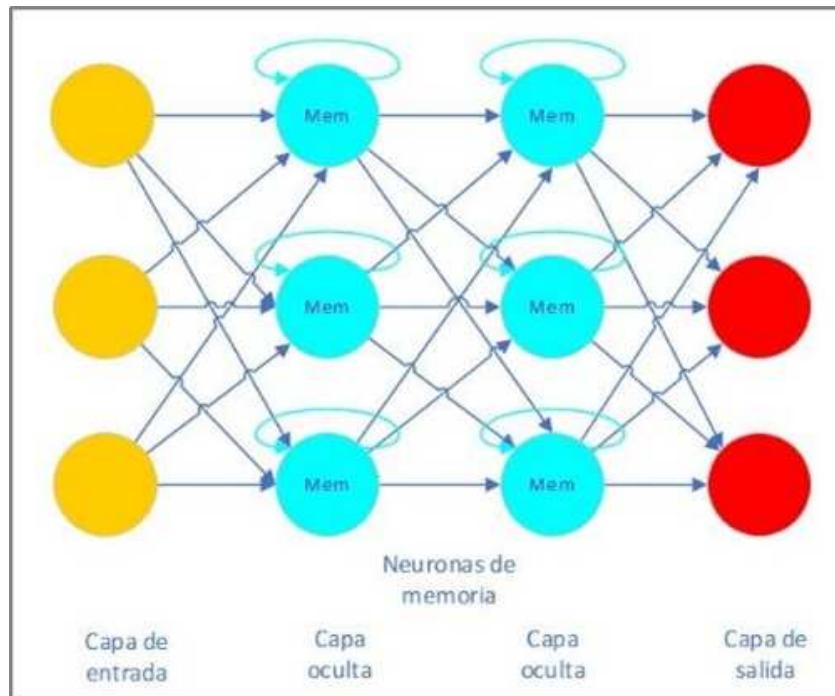


Figura 6. Long and Short-Term Memories (LSTM).

También existe una variación de las LSM conocida como **Gated recurrent units (GRU)** (Chung *et al.*, 2014), donde las células de memoria solo tienen una puerta de actualización y una puerta de reajuste. Estos tipos de redes tienen un mejor rendimiento computacional que los LSTM, pero son ligeramente menos expresivos, por lo que requieren redes más grandes para lograr la misma expresividad que los LSTM, lo que puede reducir el efecto de mejora del rendimiento.

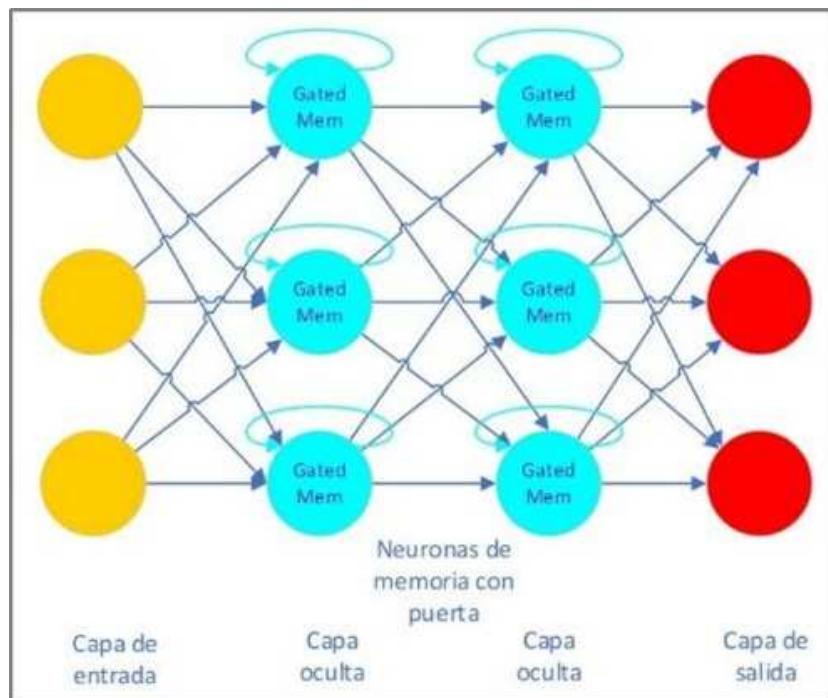


Figura 7. Gated Recurrent Units (GRU).

Las **redes neuronales bidireccionales recurrentes**, las **redes bidireccionales de memoria a largo y corto plazo** y las **unidades bidireccionales de puerta recurrente** (BiRNN, BiLSTM y BiGRU, respectivamente) no están conectadas al pasado, sino también al futuro. Es decir, tienen células de entrada y salida de correspondencia en lugar de células de salida y pueden ser entrenadas no solo para completar los datos al final de una secuencia (el final de una palabra o imagen), sino para completar los datos en medio de dichas secuencias (un hueco en medio de una palabra o imagen) (Mesnil *et al.*, 2014).

6.6. Autoencoders

Los **autocodificadores** o **autoencoders (AE)** son redes neuronales simétricas en forma de reloj de arena en las que las capas ocultas son más pequeñas que las capas de entrada y de salida (que son células de entrada y de salida que coinciden). Los autocodificadores son simétricos alrededor de la(s) capa(s) media(s) (que puede(n) ser una o dos dependiendo de si el número de capas es impar o par), que se denominan el *código*. De la entrada al código, el autocodificador actúa como un codificador (comprimiendo la información), y del código a la salida el autocodificador actúa como un decodificador. Entre las aplicaciones de los autocodificadores se encuentran la compresión de imágenes (Tan y Eswaran, 2011), la reducción de la dimensionalidad, la generación de imágenes o su uso en sistemas de recomendación.

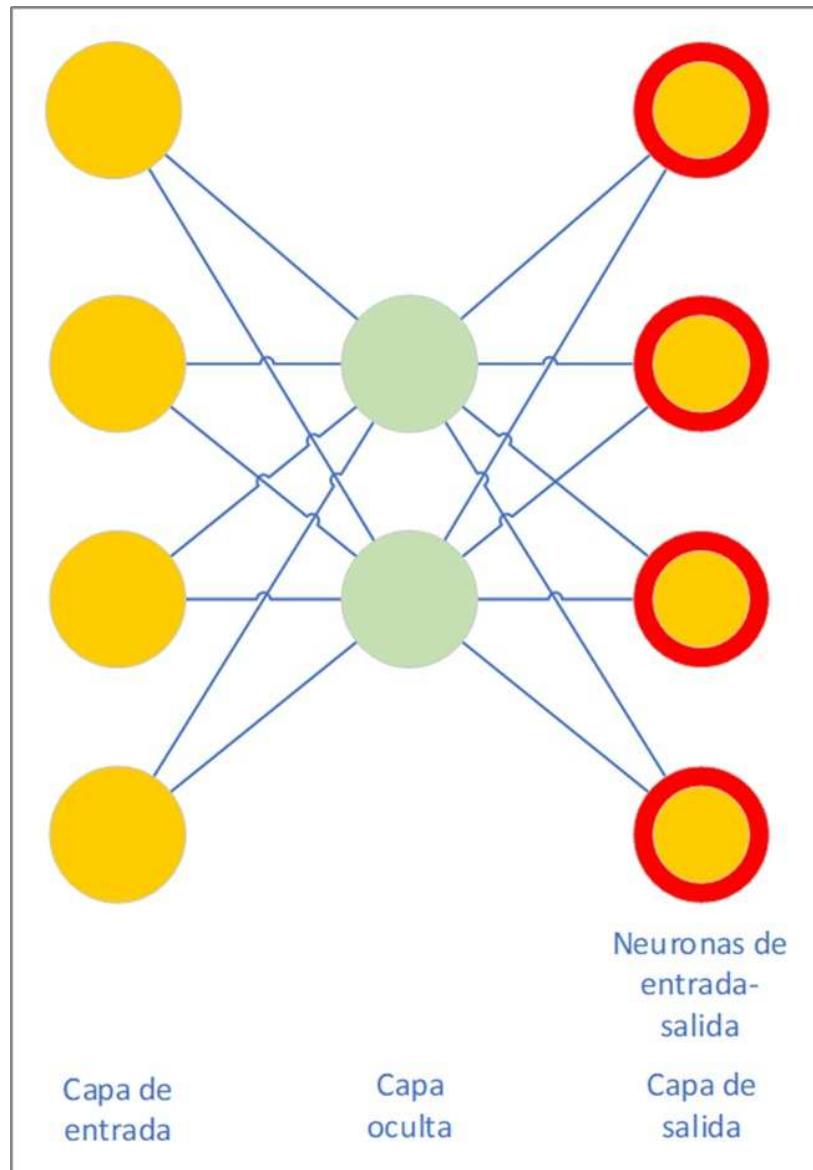


Figura 8. Autoencoders (AE).

Los **autocodificadores de supresión de ruido (DAE – Denoising Autoencoders)** se utilizan para eliminar el ruido de la imagen (utilizando el ruido como entrada en lugar de los datos) (Alex *et al.*, 2017), mientras que para la extracción de características se utilizan los **autocodificadores de dispersión (SAE – Sparse Autoencoders)** (para lo cual se basan en una estructura en la que las capas medias son mayores que las capas de entrada y salida, a diferencia del resto de los autocodificadores) (Zabalza *et al.*, 2016).

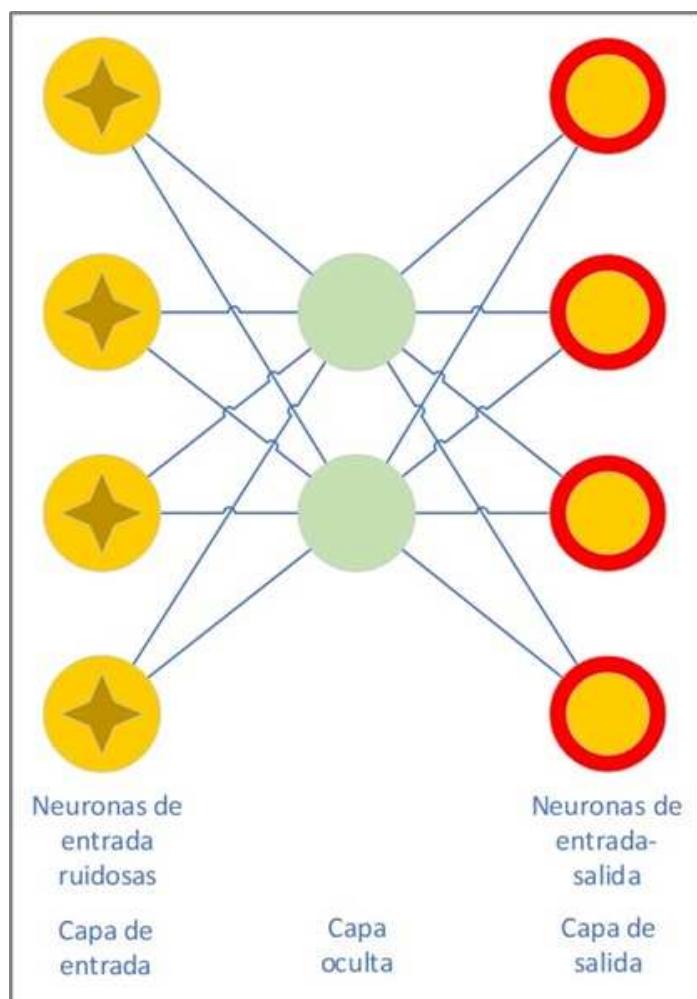


Figura 9. Denoising Autoencoders (DAE).

Los **autocodificadores variacionales (VAE – Variational Autoencoders)** tienen una estructura similar a la de los autocodificadores, pero están relacionados con las

máquinas de Boltzmann (BM) y las máquinas de Boltzmann restringidas (RBM), y se basan en las matemáticas bayesianas para modelar la distribución de probabilidad aproximada de las muestras de entrada. Entre sus aplicaciones tenemos el aprendizaje de representaciones latentes, la generación de imágenes y textos (Semeniuta, Severyn y Barth, 2017), lograr resultados de última generación en el aprendizaje semisupervisado, así como interpolar textos perdidos entre frases.

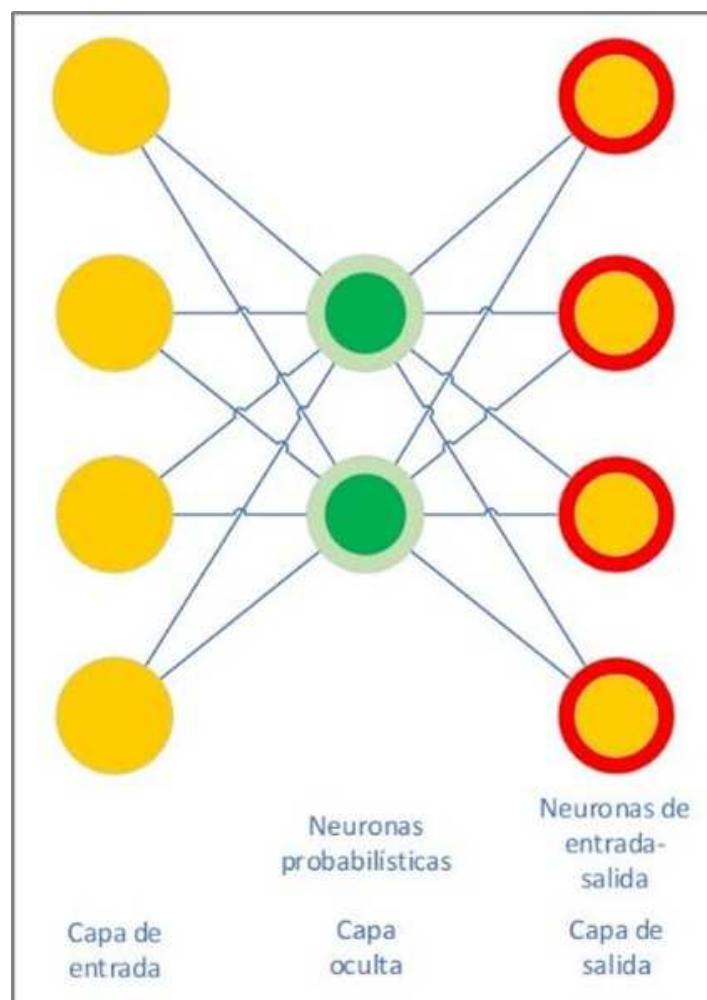


Figura 10. Variational Autoencoders (VAE).

6.7. Redes neuronales convolucionales

Los modelos lineales no funcionan bien para el reconocimiento de imágenes. Imaginemos, por ejemplo, que queremos reconocer animales u objetos en imágenes y tomar una imagen o plantilla promedio de cada clase (por ejemplo, una para perros, otra para gatos, etc.) para usarla en los datos de entrenamiento y luego usar, por ejemplo, un algoritmo clasificador como el k-NN (u otro) en la fase de prueba para medir la distancia a los valores de píxeles de cada imagen no clasificada. La imagen modelo resultante de promediar todos los perros, por ejemplo, sería una imagen borrosa con una cabeza a cada lado. Esto no funcionaría. Lo que necesitamos es aprovechar las características de las redes neuronales profundas para la clasificación de las imágenes utilizando capas de abstracción. A través de este caos oculto las redes pueden aprender características cada vez más abstractas (Karpathy *et al.*, 2016).

En este sentido, son especialmente útiles las **redes neuronales convolucionales (CNN – Convolutional Neural Networks)** y las **redes neuronales convolucionales profundas (DCNN – Deep Convolutional Neural Networks)** (Shin *et al.*, 2016), utilizadas para el reconocimiento de imágenes, reconocimiento de patrones o el análisis del sentimiento en los textos, entre otras aplicaciones.

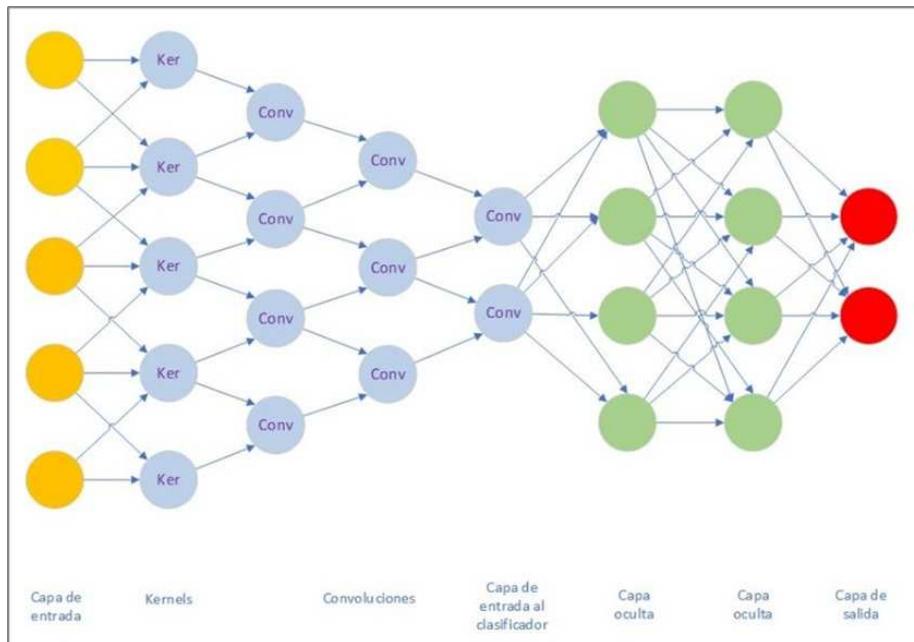


Figura 11. Redes Convolucionales Profundas (DCNN).

Son las más utilizadas en las aplicaciones de búsqueda de objetos en imágenes y vídeos, reconocimiento facial, transferencia de estilos (Gatys, Ecker y Bethge, 2016) o mejora de la calidad de las imágenes. Imaginemos una fotografía representada por los píxeles en escala de grises de la imagen. Primero dividimos toda la imagen en bloques de 8×8 píxeles y asignamos a cada uno un tipo de línea dominante (horizontal, vertical, las dos diagonales, un bloque completamente opaco o completamente vacío, etc.). El resultado es una matriz de líneas que representan los bordes de la imagen. En la siguiente capa tomamos de nuevo un bloque de 8×8 bloques obtenidos en la etapa anterior y extraemos una nueva salida con nuevas características cada vez más abstractas, repitiendo la operación una y otra vez. Esta operación se llama convolución y puede ser representada por una capa de la red neuronal, ya que cada neurona puede actuar como cualquier función.

En las primeras etapas, las neuronas se activan representando la línea dominante en cada célula de 8×8 píxeles. En las etapas intermedias las neuronas representan rasgos como la pata o la cabeza. En las etapas posteriores las neuronas se activan

representando conceptos como el gato o el perro. La salida de la última etapa de convolución se conecta a un MLP que actúa como clasificador basado en las características más abstractas, determinando la probabilidad de pertenecer a una clase final (perro o gato, por ejemplo).

Las **redes deconvolucionales (DN o DNN)**, también conocidas como *Redes Gráficas Inversas*, son CNN invertidas en las que se pueden alimentar valores como «perro» o «gato» y obtener una imagen de uno de los animales como resultado, así como detectar cambios en las imágenes, por ejemplo, (Alcantarilla *et al.*, 2018).

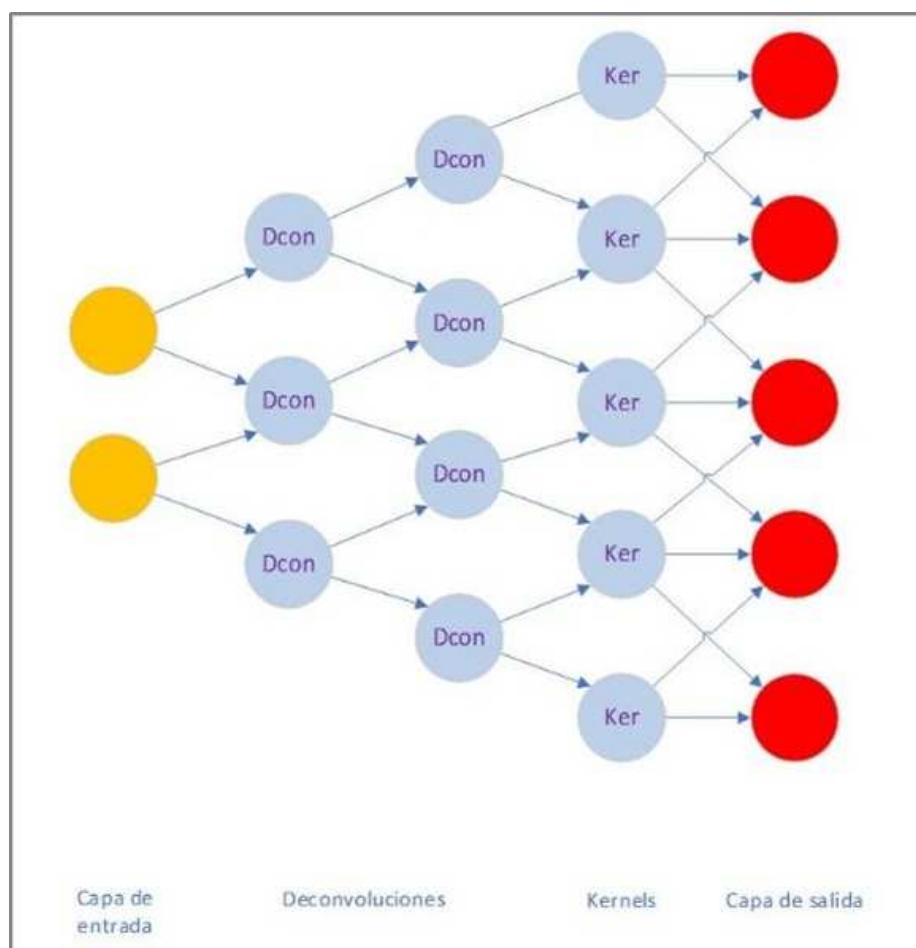


Figura 12. Redes deconvolucionales (DN oDNN).

Las **redes gráficas inversas convolucionales profundas (DCIGN)** son en realidad VAE en las que el codificador es una CNN y el decodificador es una DNN. Estas redes tratan de modelar características como probabilidades, y pueden ser utilizadas para unir dos objetos en una sola imagen, eliminar un objeto de una imagen, rotar un objeto en una imagen 3D, modificar la luz, etc. (Kulkarni *et al.*, 2015).

6.8. Redes generativas antagónicas

Por último, las **Redes Generativas Antagónicas (GAN – Generative Adversarial Networks)** están formadas por dos redes neuronales que trabajan juntas, normalmente una combinación de una FFNN y una CNN. Una de ellas se encarga de generar contenido (es decir, **red generativa**), mientras que la otra se encarga de juzgar o discriminar el contenido (es decir, **red discriminativa**) generado por la primera. También es habitual combinar una Red Deconvolucional (usada como generativa) con una Red Neuronal Convolucional (usada como discriminativa), generando y juzgando imágenes con el fin de sintetizar imágenes artificiales, por ejemplo, para convertir vídeos de caballos en cebras o viceversa, o incorporar caras de personas conocidas como políticos a vídeos de series famosas (lo que se conoce como técnicas *Deepfake*).

La red discriminativa recibe como entrada los datos de formación o el contenido generado por la red generadora. Esto forma un sistema de competición en el que la red discriminativa es cada vez mejor para distinguir los datos reales de los datos generados por la red generadora y, al mismo tiempo, la red generadora es cada vez mejor para generar datos que la red discriminativa es incapaz de distinguir (Goodfellow *et al.*, 2014).

Entre las aplicaciones de las redes generativas antagónicas se incluyen la generación de ejemplos de conjuntos de datos de imágenes, la generación de fotografías de rostros humanos y las poses (Ma *et al.*, 2017), la generación de fotografías realistas o de dibujos animados, traducción de imagen a imagen, traducción de texto a imagen, envejecimiento facial, edición de fotografías, transformación de la ropa en una imagen o predicción de vídeo, entre muchos otros.

6.9. Aprendizaje por refuerzo

El aprendizaje de refuerzo (y el aprendizaje de refuerzo profundo) es una de las tendencias más prometedoras de los últimos años en el campo del aprendizaje de máquinas (Leike *et al.*, 2018). Entre sus aplicaciones se encuentran los vehículos autónomos (Sallab *et al.*, 2017), los robots aspiradores, el comercio automatizado, la gestión de recursos empresariales (incluidos los recursos de red) o los videojuegos.

En este sentido, hemos comentado anteriormente cómo en 2016 los ordenadores consiguieron vencer a otros ordenadores el 99,8 % del tiempo y, por primera vez, los jugadores humanos son campeones en el juego de *Go*. Para entender el logro que esto representa, debemos tener en cuenta que el número de posibles estados en los que se encuentra un tablero de *Go* durante una partida es mayor que el número de átomos que existen en el universo. De hecho, esto se logró a través de técnicas de aprendizaje de refuerzo profundo. Específicamente una búsqueda en el árbol de Monte Carlo, guiada por una «red de valores» y una «red de políticas» (hablaremos de esto más adelante), ambas implementadas por la tecnología de redes neuronales profundas, en lo que se conoce como el algoritmo AlphaGo, desarrollado por investigadores de la compañía DeepMind, adquirida en 2014 por Google (Silver *et al.*, 2016).

AlphaGo se entrenó inicialmente utilizando una base de datos de unos 30 millones de movimientos tomados de juegos de jugadores humanos expertos. Después de alcanzar un cierto nivel, el algoritmo fue entrenado usando técnicas de aprendizaje de refuerzo profundo para mejorar su juego compitiendo contra sí mismo. El primer lanzamiento en 2015, AlphaGo Fan, se ejecutó en 176 GPU distribuidas.

La segunda versión en 2016, AlphaGo Lee, se ejecutó en 48 TPU distribuidas. AlphaGo fue superado por AlphaGo Master en 2017, que funcionaba en una sola máquina con 4 TPU.

Luego vino AlphaGo Zero a finales de 2017, un algoritmo que no fue entrenado por los datos de partidas de competidores humanos. AlphaGo Zero solo podía detectar posiciones en el tablero, en lugar de tener datos de entrenamiento anteriores. AlphaGo Zero también se ejecutó en una sola máquina con 4 TPU y venció a AlphaGo Lee en tres días por 100 juegos a 0 y alcanzó el nivel de AlphaGo Master en 21 días. La red neuronal de AlphaGo Zero fue previamente entrenada usando TensorFlow con 64 procesos de GPU y 19 CPU servidoras de parámetros (Silver *et al.*, 2017).

Después de eso, en 2018 AlphaZero fue capaz de jugar al ajedrez, al shogi y al go, utilizando un enfoque similar al de AlphaGo Zero, es decir, sin ningún dato previo de los jugadores humanos. En ocho horas fue superior al AlphaGo Zero jugando al Go. También utilizó una sola máquina de 4 TPU para la inferencia, y fue entrenado por 5 000 TPU de primera generación que solo competían contra él, sin datos externos (Schrittwieser *et al.*, 2019).

Algoritmos genéticos

En primer lugar, cabe mencionar, **dentro de la informática evolutiva, los algoritmos genéticos**, ya que también se consideran parte del aprendizaje de refuerzo y se han utilizado durante años para estos fines. Aunque no han dado tan buenos resultados como los algoritmos basados en Q-Learning, Deep Q-Learning o A3C, que veremos más adelante, se siguen investigando con nuevas propuestas (Bora, Mariani y dos Santos Coelho, 2019).

El funcionamiento de los algoritmos genéticos se basa en el modelo de selección natural. Se basa en una población inicial de individuos (o cromosomas) que representan posibles soluciones a un problema, generalmente generados al azar o siguiendo ciertas pautas. Se define una función de aptitud que mide la bondad o la aptitud de cada individuo o solución. También define la probabilidad de que los individuos se crucen (por lo general, los individuos más aptos son más probables) o

tengan una mutación (por lo general, esta probabilidad es baja, pero no debería ser cero para no converger rápidamente en los mínimos locales). Así pues, en cada iteración o fase se evalúa la aptitud de los individuos, se seleccionan los mejores individuos (*selección*), se cruzan los individuos entre sí (*cruce*) y se producen las diferentes mutaciones posibles (*mutación*). Para ello se definen operadores específicos de selección, cruce y mutación, con diferentes.

Enfoques basados en modelos (model-based) y sin modelos (model-free)

Una vez que hemos visto de qué son capaces los algoritmos de aprendizaje de refuerzo sin entrenamiento supervisado previo (sin datos de entrenamiento etiquetados), profundizaremos un poco más en los conceptos que hay detrás de este tipo de algoritmos.

*En los algoritmos *reinforcement learning*, el problema no está relacionado con los datos disponibles, sino con un entorno en el que un agente tiene que interactuar.*

Por ejemplo, un robot aspirador en una casa, un coche autónomo en la carretera, un jugador en un videojuego o un agente en un mercado de valores. Aunque existen enfoques basados en modelos, es inviable que un coche autónomo memorice todo el planeta y todas las posibilidades que puede encontrar. Un automóvil autónomo no puede prever todas las infinitas posibilidades de movimientos e incidentes que pueden existir en todas las carreteras del mundo, por lo que el objetivo no debe ser predecir todos los movimientos, sino minimizar el error de conducción. En resumen, se trata de minimizar un error o maximizar una recompensa. En esto se basan los enfoques sin modelos, y son los que vamos a ver a continuación.

Balance o compromiso entre exploración y explotación (exploration-exploitation trade-off)

Es importante entender lo que significa el balance o compromiso exploración/explotación. Imaginemos a un **agente** fingiendo ser un personaje dentro de un videojuego de tipo laberinto. En el laberinto puede haber un importante tesoro que le permite obtener una recompensa de +500 puntos. Además, dentro del laberinto puede encontrar trampas que penalizan con -50 puntos y pequeños tesoros o monedas que proporcionan una recompensa de +5 puntos cada una. El agente, dependiendo de su habilidad para observar el entorno, puede decidir realizar ciertas acciones (avanzar, retroceder, girar, etc., dependiendo de la interfaz permitida). Es importante tener en cuenta que la recompensa no siempre es inmediata. Es decir, encontrar un pequeño tesoro puede venir después de una serie de decisiones encadenadas o una decisión tomada previamente en el tiempo.

Si el agente, después de pasar un cierto tiempo explorando, encuentra una habitación con una cierta cantidad de monedas, puede pasar su tiempo explotando esa situación (recogiendo todas las monedas de la habitación), perdiéndose la mayor recompensa (el tesoro final). Esto se conoce como el **compromiso exploración/explotación** (de Sledge y Príncipe, 2017).

Una posible estrategia simple podría ser que el agente pase el 90 % de su tiempo realizando la acción que le está dando los mejores resultados y el 10 % del tiempo explorando nuevas posibilidades aleatorias incluso lejos de los pequeños tesoros conocidos. Esta estrategia se conoce como **estrategia -greedy (ϵ -greedy)** (Hu, Shi y Liu, 2017), donde ϵ es el porcentaje (fracción) de tiempo que el agente dedica a la exploración en lugar de a la explotación. Esto puede reducirse progresivamente en función de los conocimientos adquiridos por el agente.

Procesos de decisión de Markov (MDP – Markov Decision Processes)

Los movimientos del agente a lo largo del videojuego de tipo laberinto descrito anteriormente pueden formalizarse como un **Proceso de Decisión de Markov (MDP)** (Liu, Cheng y Hsueh, 2017). Un MDP es un proceso que se mueve a través de diferentes estados y en el que existe una probabilidad específica de transitar entre un estado i y un estado j . Por lo tanto, hay un conjunto de estados finitos en los que el agente se puede encontrar en el laberinto. Asimismo, hay un conjunto de acciones (avance, retroceso, giro, etc.) que el agente puede realizar en cada uno de los estados. También hay una recompensa (positiva o negativa) asociada a cada transición. Por otro lado, se considera que existe un **factor de descuento** γ , un valor entre 0 y 1 que cuantifica la importancia entre las recompensas inmediatas y las recompensas futuras. Por ejemplo, $\gamma = 0.95$ implica que una recompensa de +10 que se obtiene después de 5 pasos tendrá un valor actual como recompensa de $0.95^5 \cdot 10 = 0.95 \cdot 0.95 \cdot 0.95 \cdot 0.95 \cdot 0.95 \cdot 10 = 7.74$.

Por último, en el MDP existe lo que se denomina *memorylessness* o falta de memoria, es decir, el agente no necesita conocer los estados anteriores al actual, considera que el futuro solo depende del presente. Por lo tanto, el objetivo del agente es maximizar la suma de las recompensas a largo plazo desde el momento presente hasta el tiempo futuro:

$$\sum_{t=0}^{t=\infty} \gamma^t r(s(t), a(t)) \quad (15)$$

Donde $r(s,a)$ es una función de recompensa que depende del estado s y la acción a .

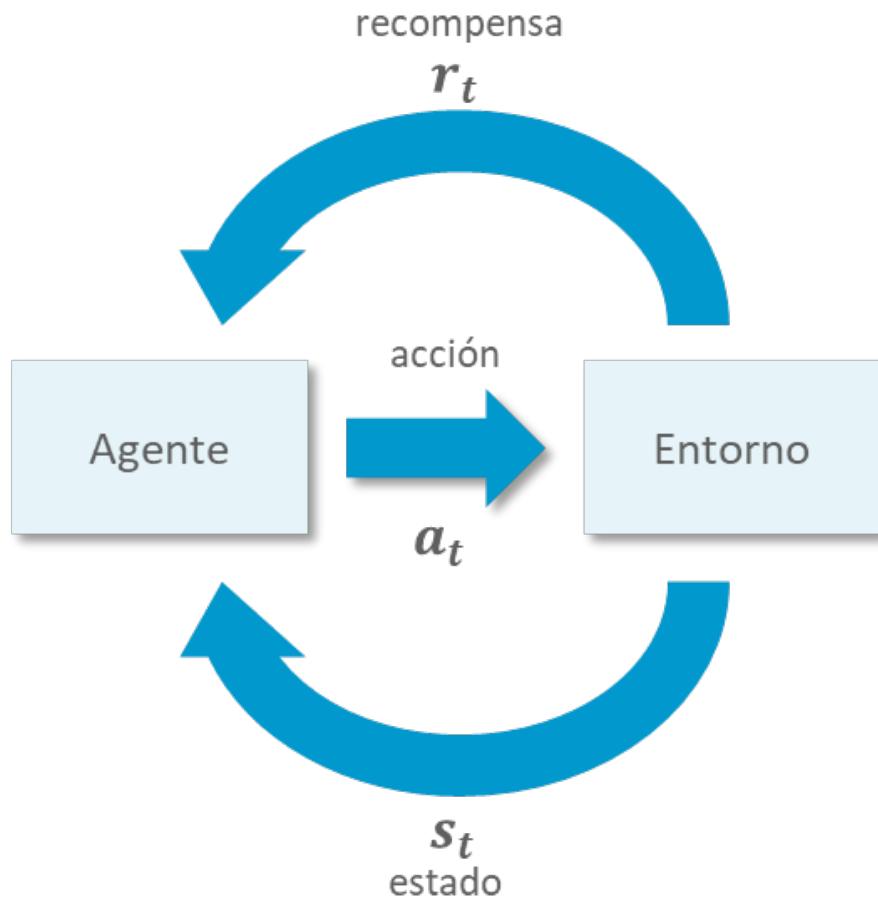


Figura 13. Agente en un escenario Reinforcement Learning libre de modelo.

El algoritmo de aprendizaje Q-learning

El algoritmo de aprendizaje *Q-learning* (Watkins y Dayan, 1992) es una de las técnicas sin modelo (*model-free*) más conocidas en el aprendizaje de refuerzo, y tiene numerosas evoluciones y variantes del mismo (Arulkumaran *et al.*, 2017). Para cualquier proceso de decisión finito de Markov (FMDP – *Finite Markov Decision Process*), el Q-learning (Q viene de *Quality* o calidad) encuentra una política que es óptima en el sentido de que maximiza el valor esperado de la recompensa total en todos y cada uno de los pasos sucesivos, a partir del estado actual.

Así, tenemos una función de valor $Q(s, a)$ que toma como entrada el estado actual s y la acción actual a y devuelve la recompensa esperada por esa acción y todas las acciones sucesivas. Inicialmente, Q devuelve el mismo valor arbitrario. A medida que el agente explora el entorno, Q devuelve una aproximación cada vez mejor del valor de una acción, dado un estado s .

Es decir, la función Q se actualiza progresivamente. Así, el nuevo valor $Q_{new}(s_t, a_t)$ se actualiza en cada iteración a partir del antiguo valor $Q(s_t, a_t)$ con una **tasa de aprendizaje α** , ya que el valor aprendido

$$\left(r_t + \gamma \bullet \max_a Q(S_t + 1, a) \right)$$

se conoce, donde r_t es la recompensa, γ es el factor de descuento y

$$\max_a Q(S_t + 1, a)$$

es la estimación del valor óptimo futuro:

$$Q^{new}(S_t, a_t) \leftarrow (1 - \alpha) Q(S_t, a_t) + \alpha \left(r_t + \gamma \bullet \max_a Q(S_{t+1}, a) \right) \quad (16)$$

De esta manera, el agente tiene un valor estimado para cada par de estados-acción, y cuyo conocimiento aumenta con el tiempo.

Con esta información, el agente puede seleccionar qué acción llevar a cabo en cada momento de acuerdo con su estrategia de selección de acciones, que puede tener en cuenta una estrategia *-greedy*, por ejemplo, para aumentar su conocimiento del medio ambiente.

Aprendizaje de políticas (policy learning) y SARSA

En el algoritmo de Q-learning hay una **función de valor Q** que estima el valor de cada par estado-acción (la recompensa esperada por esa acción y todas las acciones subsiguientes). En el caso del aprendizaje de políticas (Nachum *et al.*, 2017) se trata de aprender una **función de política (s)** que nos proporciona

directamente la mejor acción (es decir, a) a realizar por el agente, dado el estado observado s :

$$a = \pi(s) \quad (17)$$

En este tipo de algoritmos se basan trabajos en los que un agente aprende a jugar al juego de Atari, conocido como Pong, tomando como entrada los píxeles de la pantalla (estado) y obteniendo como salida la probabilidad de mover la paleta hacia arriba o hacia abajo (acción) (Phon-Amnuaisuk, 2017).

Un ejemplo de un algoritmo de aprendizaje por políticas es SARSA (State-Action-Reward-State-Action o Estado-acción-recompensa-estado-acción) que se ha aplicado, entre otras áreas, a la creación de jugadores virtuales (Phon-Amnuaisuk, 2011).

Mientras que el Q-learning aprende los valores Q asociados a la toma de la política óptima considerando el compromiso de exploración/explotación, el SARSA aprende los valores Q asociados a la toma de la política que ella misma sigue:

$$Q^{new}(S_t, a_t) \leftarrow (1 - \alpha) Q(S_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(S_{t+1}, a_{t+1}) \right) \quad (18)$$

De esta manera, el agente tiene un valor estimado para cada par de estados-acción, y cuyo conocimiento aumenta con el tiempo.

Con esta información, el agente puede seleccionar qué acción llevar a cabo en cada momento de acuerdo con su estrategia de selección de acciones, que puede tener en cuenta una estrategia *-greedy*, por ejemplo, para aumentar su conocimiento del medio ambiente.

Aprendizaje de políticas (policy learning) y SARSA

En el algoritmo de Q-learning hay una **función de valor Q** que estima el valor de cada par estado-acción (la recompensa esperada por esa acción y todas las

acciones subsiguientes). En el caso del aprendizaje de políticas (Nachum *et al.*, 2017) se trata de aprender una *función de política* (s) que nos proporciona directamente la mejor acción (es decir, a) a realizar por el agente, dado el estado observado s :

$$a = \pi(s) \quad (17b)$$

En este tipo de algoritmos se basan trabajos en los que un agente aprende a jugar al juego de Atari, conocido como Pong, tomando como entrada los píxeles de la pantalla (estado) y obteniendo como salida la probabilidad de mover la paleta hacia arriba o hacia abajo (acción) (Phon-Amnuaisuk, 2017).

Un ejemplo de un algoritmo de aprendizaje por políticas es SARSA (*State-Action-Reward-State-Action* o Estado-acción-recompensa-estado-acción) que se ha aplicado, entre otras áreas, a la creación de jugadores virtuales (Phon-Amnuaisuk, 2011).

Mientras que el Q-learning aprende los valores Q asociados a la toma de la política óptima considerando el compromiso de exploración/explotación, el SARSA aprende los valores Q asociados a la toma de la política que ella misma sigue:

$$Q^{new}(S_t, a_t) \leftarrow (1 - \alpha) Q(S_t, a_t) + \alpha \left(r_t + \gamma \max_a Q(S_{t+1}, a_{t+1}) \right) \quad (18)$$

6.10. Aprendizaje por refuerzo profundo

El aprendizaje profundo ha acelerado el progreso en el aprendizaje de refuerzo, con el uso de algoritmos de aprendizaje profundo dentro del *reinforcement learning* que definen el campo del aprendizaje de refuerzo profundo (Arulkumaran *et al.*, 2017).

El aprendizaje profundo permite que el aprendizaje de refuerzo se amplíe a problemas de toma de decisiones que antes eran intratables, es decir, entornos con estados dimensionales y espacios de acción elevados, dando lugar al Aprendizaje por Refuerzo Profundo.

Entre los trabajos recientes en el ámbito del aprendizaje profundo de refuerzo, se han registrado dos casos de éxito destacados.

La primera, que marcó el comienzo de la revolución en el aprendizaje por refuerzo profundo, fue el desarrollo en 2015 de un algoritmo que permitió aprender a jugar una serie de videojuegos Atari 2600 a un nivel sobrehumano, directamente desde los píxeles de la imagen. De hecho, este trabajo también fue desarrollado por DeepMind y se utilizan **Deep Q-Networks** para ello. Las DQN son un enfoque que se aproxima a las funciones *Q* usando redes neuronales profundas (Mnih *et al.*, 2015).

El segundo es el ya comentado en la anterior sección, que se refiere a los desarrollos del algoritmo AlphaGo por el equipo de DeepMind, utilizando redes neuronales que se entrenaron mediante aprendizaje supervisado y aprendizaje de refuerzo, en combinación con un algoritmo de búsqueda heurística tradicional (Silver *et al.*, 2016), así como los posteriores AlphaGo Cero y AlphaZero, en los que se eliminó el aprendizaje supervisado inicial (Schrittwieser *et al.*, 2019).

Existen varios algoritmos de Aprendizaje de Refuerzo Profundo (*Deep Reinforcement Learning*) (Arulkumaran *et al.*, 2017), incluyendo las variantes de Aprendizaje

Profundo de Q-Learning (**DQN – Deep Q-Networks**), como el *Double DQN*, el *dueling DQN*, las Redes Profundas Recurrentes de Q-Learning (**DQRN – Deep Q Recurrent Networks**) y el *multi-step DQN*, entre otros, y los basados en el gradiente de política, como REINFORCE, Advantage Actor-Critic (A2C), Gradiente de Política Natural (NPG – *Natural Policy Gradient*), entre muchos otros.

Las redes Q profundas y recurrentes (**DRQN – Deep Recurrent Q-Networks**) se basan en el hecho de que un agente, al igual que un humano cuando trata de resolver un problema, debe ser capaz de tener alguna memoria sobre el tiempo pasado, y no solo basarse en la información que puede ver en ese momento. Con este fin, los DRQN se basan en la idea de utilizar las Redes Neuronales Recurrentes (RNN) para aumentar las capacidades de los DQN. De esta manera, si el agente es un personaje que juega en un laberinto en 3D, basará sus acciones no solo en lo que está viendo en un momento dado, sino también en lo que vio durante los segundos anteriores (por ejemplo, un enemigo o un tesoro que desaparece detrás de un muro a medida que el personaje se mueve), como puede suceder en un juego de FPS (*First-Person Shooter*) como el conocido Doom (Lample y Chaplot, 2017).

Finalmente, y de nuevo del equipo de DeepMind en 2016, surgió el algoritmo Actor-Crílico de Ventaja Asíncrona (**A3C – Asynchronous Advantage Actor-Critic**), que superó a los algoritmos anteriores que competían en la plataforma de juego de Atari (Mnih *et al.*, 2016). El algoritmo A3C combina una red de políticas que decide cómo actuar (es decir, la actuadora) y una red *Q* que decide cuán valiosas son las cosas (es decir, la crítica). Esto se ha utilizado para enseñar a los jugadores de 3D cómo caminar sobre sus propias piernas virtuales en un entorno simulado o incluso para demostrar a los agentes que crean su propio lenguaje para comunicarse con los demás con el fin de lograr un objetivo común (Lowe *et al.*, 2017).

El algoritmo Q-learning y las Deep Q-Networks (DQN)

Como se describió en la sección previa, Q-Learning es un algoritmo de aprendizaje de refuerzo dirigido a aprender una política (Watkins y Dayan, 1992). Esta política o estrategia es el núcleo del agente y es lo que dicta cómo se comporta el agente con el entorno. La política determina qué acción tomará el agente en un estado determinado.

Matemáticamente, la política en el algoritmo de Q-learning está modelada como una **función acción-valor** $Q(s, a)$. Es decir, $Q(s, a)$ determina la política del agente, resultando en el **Q-value** o nivel de calidad que obtiene el agente al tomar la acción a , dado que está en el estado s . Cuanto más se acerque $Q(s, a)$ a la mejor política, mejor habrá sido su aprendizaje. El objetivo del agente es, por lo tanto, aprender la mejor posible $Q(s, a)$, que se conoce como *TD-target* o $Q^*(s, a)$.

En el aprendizaje supervisado, una *función de costos* o *función de pérdidas* mide la calidad de la función $h(x)$ usada como *función de hipótesis* para predecir el valor de una etiqueta dada para una nueva entrada. En el aprendizaje por refuerzo hay un concepto similar de función de pérdida, conocido como *error*. Así, la función de coste o función de pérdida es:

$$Loss = Q(s_t, a_t) \quad (19)$$

Sin embargo, en el aprendizaje por refuerzo, la función objetivo *TD-target* siempre es desconocida. En este sentido, la **ecuación de Bellman** se utiliza para estimar $Q^*(s, a)$ (Watkins y Dayan, 1992).

$$Q(s_t, a_t) = r_{t+1} + \gamma \max Q(s_{t+1}, a) \quad (20)$$

Esta ecuación significa lo siguiente. En el momento t , el agente está en el estado s_t y, por ello, decide tomar la acción a_t . En consecuencia, el agente obtiene una recompensa r_{t+1} y la observación de su entorno indica que está en el nuevo estado s_{t+1} . Según $Q(s, a)$, el agente calcula, en base a su conocimiento hasta ese

momento, el máximo valor Q que puede obtener dado el estado s_{t+1} considerando todas las posibles acciones que puede realizar. Como adelantamos, el *factor de descuento* γ es un hiperparámetro entre 0 y 1 que permite ponderar la importancia del máximo valor Q en el estado futuro. Así:

$$Loss = r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t) \quad (21)$$

Una forma de minimizar la función de pérdida es por el **método de descenso de gradiente** (Van Hasselt, 2010). En este método, la política $Q(s, a)$ se actualiza en base a la recompensa actual y al valor máximo de las recompensas futuras esperadas:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) \quad (22)$$

Como adelantamos, la *tasa de aprendizaje* α es un hiperparámetro entre 0 y 1 que permite controlar la velocidad de convergencia del procedimiento.

El algoritmo Q-Learning puede ser implementado con una tabla conocida como **Q-matrix** (*tabla Q* o *matriz Q*) de dimensiones $S \times A$, donde S es el número de estados posibles y A es el número de acciones que el agente puede realizar en cada estado. El elemento ij de la tabla, o el elemento (s, a) , almacena el valor $Q(s, a)$.

Inicialmente, el agente no tiene ningún valor (o cero) en cada elemento de la tabla, ya que aún no ha explorado el entorno. El agente, entonces, comienza el primer episodio de entrenamiento. Para ello, comienza a moverse a través de los estados s del entorno realizando acciones a y obteniendo recompensas r por ello. Esto permite al agente actualizar la tabla aprendiendo la política $Q(s, a)$.

Una vez terminado el primer episodio, comienza un segundo episodio en el que ya no parte de una Q-matriz vacía, sino que tiene en cuenta la política $Q(s, a)$ que aprendió durante el primer episodio. Repite, en sucesión, una serie de N episodios, cada uno con un número máximo de acciones M (o transiciones entre estados) preestablecidas.

Durante el proceso de aprendizaje, se tienen en cuenta los hiperparámetros explicados anteriormente: la tasa de descuento γ y la tasa de aprendizaje α . Sin embargo, hay otro hiperparámetro a tener en cuenta que ya se introdujo en la sección de exploración, la tasa de exploración ε o el porcentaje de tiempo que el agente dedica a la exploración en lugar de la explotación. Recordamos que ε puede reducirse progresivamente dependiendo de los conocimientos adquiridos por el agente. Normalmente, de hecho, comienza en 1 y termina en 0, reduciéndose en cada episodio por un *declive de la exploración* (*exploration decay*), normalmente $(1 / N)$.

Durante la etapa de aprendizaje, en cada estado de cada episodio, el agente decidirá tomar la política óptima (explotación) con una probabilidad $P_O = (1 - \varepsilon)$, o seguir una política aleatoria (exploración), haciendo una acción aleatoria, con una probabilidad de $P_R = \varepsilon$.

Una vez terminado el entrenamiento del agente, pasa a la etapa de prueba, y siempre tomará la política óptima, a menos que continúe con su aprendizaje también durante la producción.

El problema del método clásico basado en la matriz Q es que requiere un número muy alto de espacios de memoria. Son comunes los problemas en los que el número de estados o incluso el número de acciones es muy alto, de modo que el espacio de memoria $S \times A$ sería difícil de manejar o directamente inmanejable. En estas situaciones, la tabla Q puede ser reemplazada por una red neuronal. **Así, pasamos de un algoritmo Q-Learning a un algoritmo de Q-Learning profundo (Deep Q-Learning).**

De esta manera, podemos utilizar una red neuronal con una capa de entrada de S neuronas (el número de estados posibles) y una capa de salida de A neuronas (el número de acciones posibles en cada estado). El número de capas ocultas y unidades para cada una de las capas ocultas se determinará por el diseño de la red y la prueba de los diferentes modelos. Entrenamos la red neuronal, por ejemplo, utilizando un método óptimo para minimizar la función de pérdida, como el descenso de gradiente. En la fase de prueba, la red neuronal tomará a su entrada el estado s en el que se encuentra en el agente. En la salida obtendremos los A valores $Q(s, a)$ para dicho estado s .

El algoritmo Double Q-learning y las Double Deep Q-Networks (DDQN)

En el algoritmo Q-Learning, la política óptima del agente es elegir siempre la acción que maximiza la recompensa en cada uno de los estados. Sin embargo, durante la etapa de entrenamiento, el agente no conoce absolutamente nada del entorno al comenzar el primer episodio de entrenamiento. El agente estima $Q(s, a)$ inicialmente y actualiza su valor en cada iteración. Estos valores Q son muy ruidosos, por lo que el agente no tiene forma de asegurarse de que la acción con el máximo valor Q esperado sea realmente la óptima.

Desafortunadamente, la acción óptima a veces tiene valores Q más pequeños que la acción no óptima en la mayoría de los casos. De acuerdo con la política óptima en el Q-Learning, el agente tiene una tendencia a elegir la acción no óptima en un estado

dado solo porque tiene el máximo valor Q. Esto es lo que se conoce como **el problema de la sobreestimación del Q-value o sobreestimación del valor de la acción.**

Este problema implica que los ruidos provenientes del valor Q estimado causarán grandes sesgos positivos durante el procedimiento de actualización, complicando el proceso de aprendizaje.

En este sentido, Van Hasselt (2010) propuso el **algoritmo Double Q-Learning**. Para ello, el algoritmo utiliza dos funciones de valor-acción, Q^A y Q^B , como estimadores. Van Hasselt demostró que, aunque Q^A y Q^B sean ruidosas, estos ruidos pueden ser vistos como una distribución uniforme. Por lo tanto, el algoritmo Double Q-Learning resuelve el problema de la sobreestimación del valor de acción.

En Double-Q Learning, la actualización de la política pasa a ser ahora:

$$Q^A(s_t, a_t) \leftarrow Q^A(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q^B(s_{t+1}, a) - Q^A(s_t, a_t) \right) \quad (23)$$

Donde QA es la función para seleccionar la mejor acción a con un valor Q máximo del siguiente estado s_{t+1} :

$$a = \max_{a_{t+1}} Q^A(s_{t+1}, a_{t+1}) \quad (24)$$

Y QB es la función para calcular el valor Q esperado mediante la acción a elegida por $QA \cdot Q_A$. Por lo tanto, actualizamos el valor de la función QA mediante el valor Q esperado proporcionado por la función QB mediante la Ecuación 23.

Sin embargo, el algoritmo Double Q-Learning sigue un método basado en tablas o matrices que, como ya hemos explicado, presenta el problema de requerir una gran cantidad de memoria a medida que aumenta el número de estados y acciones posibles. En este sentido, Van Hasselt, Guez y Silver (2016) propusieron el uso de redes neuronales profundas para resolver este problema. Si en el caso de Deep Q-

En Learning utilizamos una red neuronal para modelar la política de $Q(s, a)$, en el caso de **Double Deep Q-Learning** o las **Double Deep Q-Networks (Double DQN o DDQN)** usamos dos redes neuronales profundas: la *Deep Q-Network* (DQN) (o red Q generada) y la *red objetivo* o *target network*. (o red Q objetivo).

$$Q_{qnet}(s_t, a_t) \leftarrow r_{t+1} + \gamma Q_{tnet}(s_{t+1}, a) \quad (25)$$

Donde Q_{qnet} es la Deep Q-Network para seleccionar la mejor acción a con un valor Q máximo del siguiente estado s_{t+1} :

$$a = \max_{a_{t+1}} Q_{qnet}(s_{t+1}, a_{t+1}) \quad (26)$$

Y Q_{tnet} es la red objetivo para calcular el valor Q esperado por medio de la acción a elegida por la Deep Q-Network Q_{qnet} . Por lo tanto, actualizamos el valor de la Deep Q-Network Q_{qnet} mediante el valor Q esperado proporcionado por la *red objetivo* Q_{tnet} , mediante la Ecuación 25

En las Double Deep Q-Networks también hay dos nuevos pasos. Después de la actualización de la Deep Q-Network, los parámetros de la red objetivo se actualizan en base a los parámetros de la Deep Q-Network para varias iteraciones:

$$Q_{tnet}(s, a) = Q_{qnet}(s, a) \quad (27)$$

Después de eso, los parámetros de la Deep Q-Network se actualizan ahora siguiendo el **optimizador de Adam** (Wu et al., 2019).

En el caso de las Double Deep-Q Networks, tanto la Deep Q-Network como la red objetivo tendrán la misma arquitectura. Es decir, el mismo número de capas, el mismo número de unidades por capa y las mismas conexiones entre capas. De manera similar a lo que ocurre con Deep Q-Learning, podemos utilizar redes neuronales (tanto la Deep-Q Network como la red objetivo) con una capa de entrada con S unidades (el número de estados posibles) y una capa de salida de A unidades (el número de acciones posibles en cada estado). El número de capas ocultas y

unidades para cada una de las capas ocultas será determinado por el diseño de las redes y la prueba de los diferentes modelos. Nuevamente de manera similar a Deep Q-Learning, en Double Deep Q-Learning la red neuronal, en este caso la Deep Q-Network, tomará a su entrada el estado s en el que se encuentra en el agente. En la salida obtendremos los A valores $Q^A(s, a)$ para dicho estado s .

Es habitual utilizar la técnica ***experience replay*** durante la etapa de entrenamiento. Con esta técnica, el agente puede almacenar una cierta cantidad de experiencias $(s_t, a_t, r_t, s_{t+1}, \text{done})$, donde *done* implica si el episodio termina debido a la acción realizada. Una vez que el agente acumula suficientes experiencias, comienza a entrenarse en base a lotes de experiencias.

6.11. Ejemplos de implementación

Vamos a realizar un ejemplo de implementación empleando la librería TensorFlow de Google, ya introducida en el Tema 2.



Figura 14. Fuente: <https://www.tensorflow.org/>

Recordando lo mencionado, TensorFlow es una biblioteca de código abierto muy popular para el cálculo numérico de alto rendimiento desarrollada por el equipo de Google Brain en Google. Como su nombre lo sugiere, TensorFlow es un marco de trabajo que implica la definición y ejecución de cálculos que implican tensores.

Un **tensor** es cierta clase de entidad algebraica que generaliza los conceptos de escalar, vector y matriz de una forma que sea independiente de cualquier sistema de coordenadas elegido.

TensorFlow puede entrenar y ejecutar redes neuronales profundas que pueden ser usadas para desarrollar varias aplicaciones de IA. TensorFlow es ampliamente utilizado en el campo de la investigación y aplicación del *machine learning*.

TensorFlow fue lanzado en 2015 como una herramienta de código abierto por Google. Inicialmente estaba disponible en Python. Posteriormente se lanzó TensorFlow.js para su uso mediante JavaScript, bien sea en el mismo navegador o bien del lado del servidor mediante Node.js o incluso en una Raspberry Pi. Más tarde se lanzó TensorFlow 2.0, disponible para C++, Haswell, Java, Go y Rust. También hay bibliotecas de terceros para C#, R y Scala.

Existen versiones de TensorFlow para Windows, Linux y Mac. Además, TensorFlow Lite es la versión de TensorFlow para su uso en terminales móviles o en dispositivos en el borde la red (*Edge Computing*), incluyendo escenarios Internet of Things.

Más aún, Google lanzó en 2016 su unidad de procesamiento tensorial (TPU), un circuito ASIC (Circuito Integrado de Aplicación Específicas) personalizada específicamente para *machine learning* y adaptada para TensorFlow. El TPU es un acelerador de IA programable diseñado para proporcionar alto *throughput* de aritmética de precisión baja (por ejemplo, 8 bits) y orientado para utilizar o correr modelos más que para entrenarlos. Este tipo de unidades tienen un rendimiento de *machine learning* por vatio consumido de un orden mayor de magnitud que los sistemas tradicionales. Google Cloud ofrece el uso de TPU en la nube que pueden ser empleados por el público general.

Asimismo, Google lanzó los Edge TPU, dispositivos que incluyen una CPU, una GPU y una TPU para aplicar TensorFlow en el Edge. Se pueden considerar una especie de Raspberry Pi con TPU incluida para ejecutar *machine learning* en el extremo de la red.

Nota: es necesario instalar el paquete tensorflow o tensorflow-gpu para el funcionamiento del siguiente ejemplo (instalará más de 400MB en el ordenador, incluyendo dependencias, como parte de Keras). Además, bajo Windows es necesario contar con la última versión de Visual C++ Redistributable para su funcionamiento, disponible en: <https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads>

Vamos a realizar un ejemplo de identificación de ropa en imágenes mediante redes neuronales, siguiendo el tutorial de TensorFlow 2.0 del portal de librería (ver Webgrafía al final del Tema).

Además de TensorFlow, es necesario tener instaladas otras librerías que ya deberíamos haber instalado también de los ejemplos de los temas anteriores : keras, numpy y matplotlib .

Vamos a utilizar el *dataset* Fashion-MNIST (imágenes de ropa), que sustituye al clásico MNIST (dígitos escritos a mano). Este *dataset* contiene 70 000 imágenes de ropa utilizando una baja resolución (28×28 píxeles y 8 bit por píxel, en escala de grises, es decir, conteniendo un valor de entre 0 y 255 por píxel) clasificadas en 10 categorías.

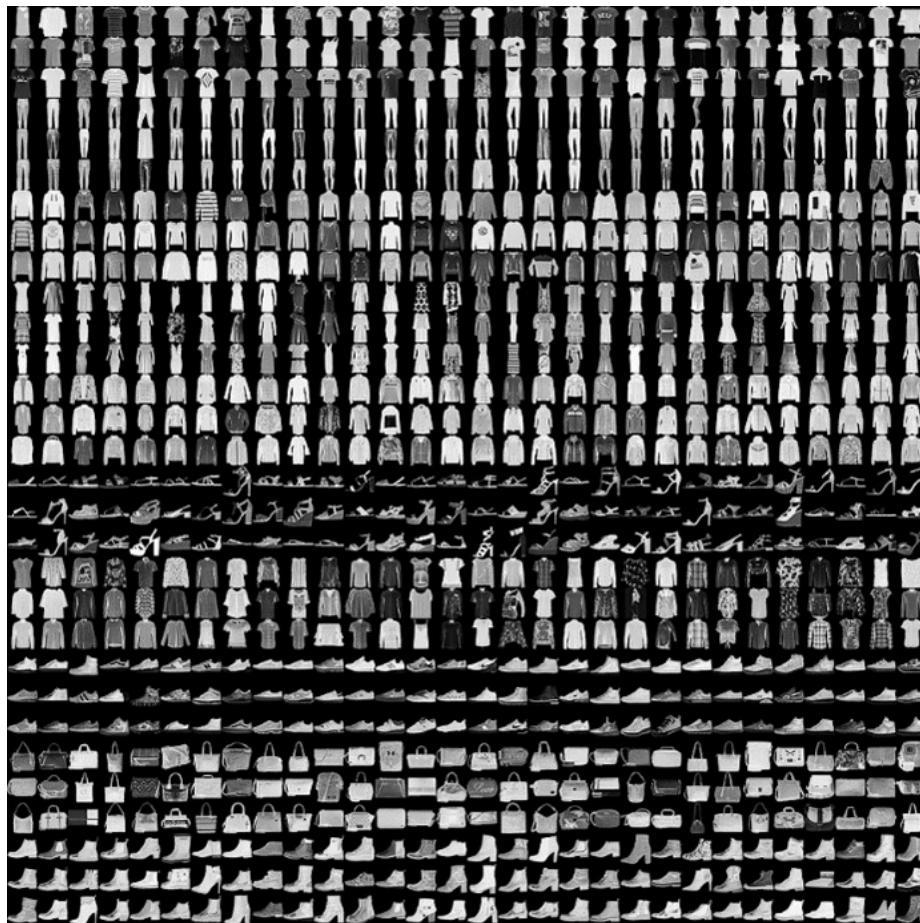


Figura 15. Muestras del dataset Fashion-MINST. Fuente: Zalando, bajo licencia MIT.

De este *dataset* utilizaremos 60 000 imágenes para el entrenamiento de nuestra red neuronal y los 10 000 restantes para el test de la misma.

Exploraremos en primer lugar los datos. Con el siguiente código obtendremos los siguientes arrays:

- ▶ train_images : array con las imágenes del training dataset (28×28 con un entero entre 0 y 255).
- ▶ train_labels : array con las etiquetas del training dataset, un valor de 0 a 9.

- ▶ `test_images` : array con las imágenes del test dataset.
- ▶ `test_labels` : array con las etiquetas del test dataset.

Las etiquetas del 0 al 9 se corresponden con tipos de ropa de la Tabla 1.

Etiqueta	Clase
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Tabla 1. Etiquetas del *dataset* Fashion-MINST.

Dichas etiquetas no están incluidas en el *dataset*, así que las tenemos que incorporar en nuestro código, como se puede ver.

```
from __future__ import absolute_import, division, print_function
, unicode_literals

# TensorFlow y tf.keras
import tensorflow as tf
from tensorflow import keras

# Librerías de ayuda
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# hay 60 000 imágenes en el training dataset
print(train_images.shape)
#> (60000, 28, 28)
print(len(train_labels))
#> 60000

# cada etiqueta es un número entre 0 y 9
print(train_labels)
#> [9, 0, 0, ..., 3, 0, 5]

# hay 10 000 imágenes en el test dataset
print(test_images.shape)
#> (10000, 28, 28)
print(len(test_labels))
#> 10000
```

Habremos visto, además, cómo se ha descargado el *dataset* durante la primera ejecución de código que haga referencia al mismo, con unos mensajes similares a:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [=====] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [=====] - 0s 0us/step
```

Vamos ahora a inspeccionar el primer elemento, antes de ver qué nos encontramos y preprocesar los datos:

```
from __future__ import absolute_import, division, print_function
, unicode_literals

# TensorFlow y tf.keras
import tensorflow as tf
from tensorflow import keras

# Librerías de ayuda
import numpy as np
import matplotlib.pyplot as plt

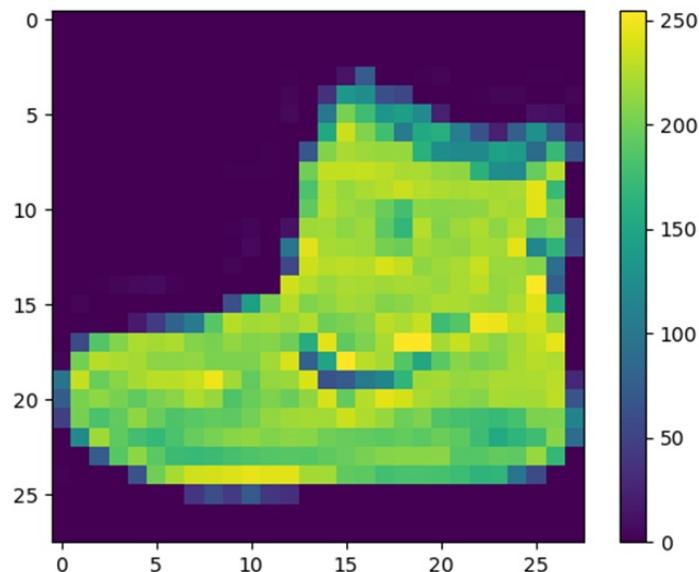
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

Veremos una imagen como la siguiente, en la que vemos los 28×28 píxeles, cada uno con un valor entre 0 y 255:



Normalizaremos los valores, tanto del *training dataset* como del *test dataset*, para que se encuentren entre 0 y 1 antes de ser usados por la red neuronal. Dividimos por 255.0 cada valor de los píxeles. Aprovechamos para ver el contenido de las 25 primeras imágenes:

```
from __future__ import absolute_import, division, print_function
, unicode_literals

# TensorFlow y tf.keras
import tensorflow as tf
from tensorflow import keras

# Librerías de ayuda
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# normalizamos
train_images = train_images / 255.0
test_images = test_images / 255.0
```

```

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()

```



Ahora vamos a construir el modelo. Utilizaremos una red neuronal con $28 \times 28 = 784$ neuronas en la capa de entrada, es decir, aplanamos el array bidimensional en un array unidimensional. Para ello utilizamos una capa de entrada `tf.keras.layers.Flatten`.

Tras dicha capa, empleamos dos capas ocultas de tipo `tf.keras.layers.Dense`, en las cuales todas las neuronas están plenamente conectadas con las capas anterior y

posterior (conexión densa). La primera capa tendrá 128 nodos, y la segunda y última tendrá 10 nodos *softmax* (o función exponencial normalizada, que es una generalización de la función logística), que devuelve un array con 10 probabilidades que sumarán 1 en total. Así, tendremos a la salida las probabilidades de que la imagen de entrada se corresponda con cada una de las 10 clases.

La función *softmax* tiene una forma similar a:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (28)$$

Tras la construcción del modelo hemos de compilarlo. En ese momento, elegiremos la función de pérdidas a minimizar, ya que es la que mide cuán exacto será nuestro modelo, el método optimizador empleado (usaremos el de Adam) y la métrica empleada para monitorizar las etapas de entrenamiento y test (usaremos la exactitud – *accuracy*).

El siguiente código muestra el proceso de construcción, compilación, entrenamiento, evaluación de exactitud, así como predicción de una clase con una muestra concreta y también de forma gráfica para las 15 primeras.

```
from __future__ import absolute_import, division, print_function
, unicode_literals

# TensorFlow y tf.keras
import tensorflow as tf
from tensorflow import keras

# Librerías de ayuda
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# normalizamos
train_images = train_images / 255.0
test_images = test_images / 255.0

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# Creamos el modelo:
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

# Compilamos el modelo:
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])

# Entrenamos el modelo:
model.fit(train_images, train_labels, epochs=10)

# Evaluamos exactitud:
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print("\nTest accuracy:", test_acc)
#> Test accuracy: 0.8848999738693237

# Realizamos predicciones, mostrando la predicción sobre el primer elemento
predictions = model.predict(test_images)
# mostramos las 10 probabilidades
print(predictions[0])
```

```

#> [4.3090558e-09 5.8588463e-09 2.3414977e-09 1.4041760e-
07 5.7719642e-09
#> 1.0011349e-04 4.5159254e-07 2.3861572e-02 1.9545018e-
07 9.7603750e-01]
# nos quedamos con la más elevada
print(np.argmax(predictions[0]))
#> 9

# Mostremos de forma gráfica todo el conjunto de las predicciones de las 10 clases
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array, true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = "blue"
    else:
        color = "red"

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                         100*np.max(predictions_array),
                                         class_names[true_label]),
                                         color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array, true_label[i]
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color("red")
    thisplot[true_label].set_color("blue")

# Mostraremos las primeras 15 imágenes de test con sus predicciones y sus etiquetas reales
# En azul las predicciones correctas y en rojo las incorrectas

```

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions[i], test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

Al entrenar la red veremos un proceso como el siguiente, para las 10 épocas que se han seleccionado:

Epoch 1/10

```
1875/1875 [=====] - 5s 3ms/step - loss: 0.4985 -  
accuracy: 0.8248
```

Epoch 2/10

```
1875/1875 [=====] - 6s 3ms/step - loss: 0.3716 -  
accuracy: 0.8666
```

Epoch 3/10

```
1875/1875 [=====] - 6s 3ms/step - loss: 0.3347 -  
accuracy: 0.8768
```

Epoch 4/10

```
1875/1875 [=====] - 4s 2ms/step - loss: 0.3103 -  
accuracy: 0.8854
```

Epoch 5/10

```
1875/1875 [=====] - 5s 3ms/step - loss: 0.2942 -  
accuracy: 0.8920
```

Epoch 6/10

```
1875/1875 [=====] - 4s 2ms/step - loss: 0.2772 -
```

accuracy: 0.8975

Epoch 7/10

```
1875/1875 [=====] - 4s 2ms/step - loss: 0.2661 -  
accuracy: 0.9003
```

Epoch 8/10

```
1875/1875 [=====] - 4s 2ms/step - loss: 0.2533 -  
accuracy: 0.9057
```

Epoch 9/10

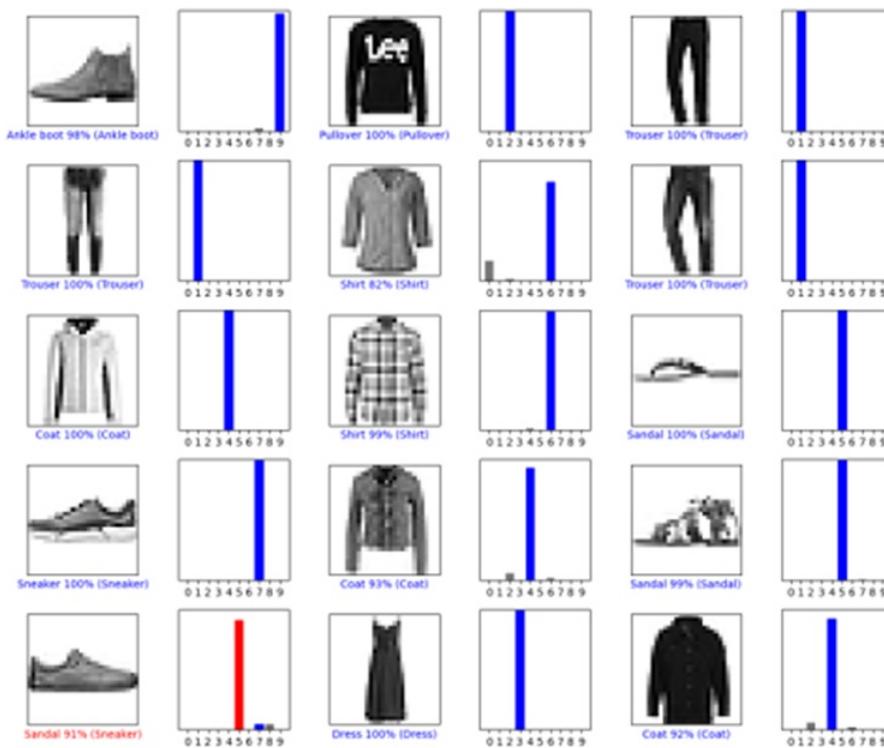
```
1875/1875 [=====] - 4s 2ms/step - loss: 0.2463 -  
accuracy: 0.9074
```

Epoch 10/10

```
1875/1875 [=====] - 4s 2ms/step - loss: 0.2375 -  
accuracy: 0.9103
```

313/313 - 0s - loss: 0.3325 - accuracy: 0.8849

Además de la salida de texto indicada como comentarios en el código, veremos una figura como la siguiente:



Vemos que la precisión sobre los datos de test (88.49 %) es algo menor que con los datos de entrenamiento (llegamos a 91.03 % en la época 10). Esto quiere decir que tenemos algo de *overfitting* o sobreajuste.

6.12. Referencias bibliográficas

Alcantarilla, P.F., Stent, S., Ros, G., Arroyo, R. & Gherardi, R. (2018). Street-view change detection with deconvolutional networks. *Autonomous Robots*, 42, 1301-1322.

Alex, V., Vaidhya, K., Thirunavukkarasu, S., Kesavadas, C. & Krishnamurthi, G. (2017). Semisupervised learning using denoising autoencoders for brain lesion detection and segmentation. *Journal of Medical Imaging*, 4(4).

Alonso, R. S., Tapia, D. I., Bajo, J., García, Ó., De Paz, J. F. & Corchado, J. M. (2013). Implementing a hardware-embedded reactive agents platform based on a service-oriented architecture over heterogeneous wireless sensor networks. *Ad Hoc Networks*, 11(1), 151-166.

Arulkumaran, K., Deisenroth, M.P., Brundage, M. & Bharath, A.A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34, 26-38.

Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems* (pp. 153-160).

Bora, T.C., Mariani, V.C. & dos Santos Coelho, L. (2019). Multi-objective optimization of the environmental-economic dispatch with reinforcementlearning based on non-dominated sorting genetic algorithm. *Applied Thermal Engineering*, 146, 688–700.

Broomhead, D. S. & Lowe, D. (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks*. United Kingdom: Royal Signals and Radar Establishment Malvern.

Cao, R., Freitas, C., Chan, L., Sun, M., Jiang, H. & Chen, Z. (2017). ProLanGO: protein function prediction using neural machine translation based on a recurrent

neural network. *Molecules*, 22(10), 1732.

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Costa-jussà, M. R., Allauzen, A., Barrault, L., Cho, K. & Schwenk, H. (2017). Introduction to the special issue on deep learning approaches for machine translation. *Computer Speech & Language*, 46, 367-373.

De Paz, J. F., Tapia, D. I., Alonso, R. S., Pinzón, C. I., Bajo, J. & Corchado, J. M. (2013). Mitigation of the ground reflection effect in real-time locating systems based on wireless sensor networks by using artificial neural networks. *Knowledge and information systems*, 34(1), 193-217.

Dechter, R. (1986). *Learning while searching in constraint-satisfaction problems* (pp. 178-183). University of California, Computer Science Department, Cognitive Systems Laboratory.

Deng, L. & Liu, Y. (Eds.). (2018). *Deep learning in natural language processing*. Springer.

Faia, R., Pinto, T., Vale, Z. & Corchado, J. M. (2018). Case-based reasoning using expert systems to determine electricity reduction in residential buildings. In *2018 IEEE Power & Energy Society General Meeting (PESGM)* (pp. 1-5). IEEE.

Fayek, H. M., Lech, M. & Cavedon, L. (2017). Evaluating deep learning architectures for Speech Emotion Recognition. *Neural Networks*, 92, 60-68.

García, Ó., Prieto, J., Alonso, R. S. & Corchado, J. M. (2017). A framework to improve energy efficient behaviour at home through activity and context monitoring. *Sensors*, 17(8), 1749.

Gatys, L. A., Ecker, A. S. & Bethge, M. (2016). Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer*

vision and pattern recognition (pp. 2414-2423).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative adversarial nets, in: *Advances in neural information processing systems*, pp. 2672–2680.

Hu, B., Shi, C. & Liu, J. (2017). Playlist recommendation based on reinforcement learning, in: *International Conference on Intelligence Science* (pp. 172–182). Springer.

Jouppi, N., Young, C., Patil, N. & Patterson, D. (2018). Motivation for and evaluation of the first tensor processing unit. *IEEE Micro*, 38(3), 10-19.

Karpathy, A. (2016). Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1, 1.

Kim, J., Kim, J., Lee, S., Park, J. & Hahn, M. (2016, November). *Vowel based voice activity detection with LSTM recurrent neural network*. In *Proceedings of the 8th International Conference on Signal Processing Systems* (pp. 134-137).

Kulkarni, T.D., Whitney, W.F., Kohli, P. & Tenenbaum, J. (2015). Deep convolutional inverse graphics network. In *Advances in neural informationprocessing systems* (pp. 2539–2547).

Lample, G. & Chaplot, D.S. (2017). Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., & Legg, S. (2018). Scalable agent alignment via reward modeling: a research direction. arXiv preprint arXiv:1811.07871.

Lima, A. C. E., de Castro, L. N. & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. *Applied Mathematics and Computation*, 270, 756-767.

Liu, Y.J., Cheng, S.M. & Hsueh, Y.L. (2017). enb selection for machine type communications using reinforcement learning based markov decision process. *IEEE Transactions on Vehicular Technology*, 66, 11330-11338.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y. & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234, 11-26.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O.P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In: *Advances in Neural Information Processing Systems* (pp. 6379-6390).

Luan, F., Paris, S., Shechtman, E. & Bala, K. (2017). Deep photo style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4990-4998).

Ma, L., Jia, X., Sun, Q., Schiele, B., Tuytelaars, T. & Van Gool, L. (2017). Pose guided person image generation. In *Advances in Neural Information Processing Systems* (pp. 406-416).

Marvin, M. & Seymour, A. P. (1969). *Perceptrons*. MIT Press.

Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D. & Zweig, G. (2014). Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3), 530-539.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. & Kavukcuoglu, K. (2016). *Asynchronous methods for deep reinforcement learning*. In *International conference on machine learning* (pp. 1928-1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529.

Nachum, O., Norouzi, M., Xu, K. & Schuurmans, D. (2017). Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems* (pp. 2775-2785).

Naraei, P., Abhari, A. & Sadeghian, A. (2016). *Application of multilayer perceptron neural networks and support vector machines in classification of healthcare data*. In 2016 Future Technologies Conference (FTC) (pp. 848-852). IEEE.

Pal, S. K. & Wang, P. P. (2017). *Genetic algorithms for pattern recognition*. CRC press.

Park, D. H. & Chiba, R. (2017). *A neural language model for query auto-completion*. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 1189-1192).

Panesar, A. (2019). What Is Machine Learning? In *Machine Learning and AI for Healthcare* (pp. 75-118). Berkeley, CA: Apress.

Partila, P., Tovarek, J., Voznak, M., Rozhon, J., Sevcik, L. & Baran, R. (2018). *Multi-classifier speech emotion recognition system*. In 2018 26th Telecommunications Forum (TELFOR) (pp. 1-4). IEEE.

Phon-Amnuaisuk, S. (2011). *Learning chasing behaviours of non-player characters in games using SARSA*. In: European Conference on the Applications of Evolutionary Computation (pp. 133-142). Springer.

Phon-Amnuaisuk, S. (2017). *What does a policy network learn after mastering a pong game?* In International Workshop on Multi-disciplinary Trends in Artificial Intelligence (pp. 213-222). Springer.

Ramchoun, H., Idrissi, M. A. J., Ghanou, Y. & Ettaoui, M. (2016). Multilayer Perceptron: Architecture Optimization and Training. *IJIMAI*, 4(1), 26-30.

Razzak, M. I., Naz, S. & Zaib, A. (2018). Deep learning for medical image processing: Overview, challenges and the future. In *Classification in BioApps* (pp. 323-350). Cham: Springer.

Rivas, A., Chamoso, P., González-Briones, A. & Corchado, J. M. (2018). Detection of cattle using drones and convolutional neural networks. *Sensors*, 18(7), 2048.

Rodríguez, L. P., Crespo, A. G., Lara, M. P. & Mezcua, B. R. (2008). Study of different fusion techniques for multimodal biometric authentication. In *2008 IEEE international conference on wireless and mobile computing, networking and communications* (pp. 666-671). IEEE.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.

Rosenblatt, F. (1960). Perceptron simulation experiments. *Proceedings of the IRE*, 48(3), 301-309.

Rumelhart, D. E. & McClelland, J. L. (1986). The PDP Research Group: Parallel distributed processing: Explorations in the microstructure of cognition. *Foundations*, 1, 3-44.

Russell, S. & Norvig, P. (2002). *Artificial intelligence: a modern approach*. Pearson.

Sallab, A.E., Abdou, M., Perot, E. & Yogamani, S. (2017). Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 23, 70-76.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3), 210-229.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez,

- A., Lockhart, E., Hassabis, D., Graepel, T., *et al.* (2019). *Mastering atari, go, chess and shogi by planning with a learned model*. arXiv preprint arXiv:1911.08265.
- Semeniuta, S., Severyn, A. & Barth, E. (2017). A hybrid convolutional variational autoencoder for text generation. *arXiv preprint arXiv:1702.02390*.
- Seo, J., Han, S., Lee, S. & Kim, H. (2015). Computer vision techniques for construction safety and health monitoring. *Advanced Engineering Informatics*, 29(2), 239-251.
- Shah, S. A. A., Bennamoun, M. & Boussaid, F. (2016). Iterative deep learning for image set based face and object recognition. *Neurocomputing*, 174, 866-874.
- Shin, H. C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I. & Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5), 1285-1298.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G. & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587), 484.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., *et al.*, (2017). Mastering the game of go without human knowledge. *Nature*, 550, 354-359.
- Sledge, I.J. & Príncipe, J.C. (2017). Balancing exploration and exploitation in reinforcement learning using a value of information criterion. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2816-2820). IEEE.
- Tan, C. C. & Eswaran, C. (2011). Using autoencoders for mammogram compression. *Journal of medical systems*, 35(1), 49-58.

Turing, I. B. A. (1950). Computing machinery and intelligence-AM Turing. *Mind*, 59(236), 433.

Van Engelen, J. E. & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, 109(2), 373-440.

Van Hasselt, H. (2010). Double Q-learning. In *Advances in neural information processing systems* (pp. 2613-2621).

Van Hasselt, H., Guez, A. & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Watkins, C.J. & Dayan, P. (1992). Q-learning. *Machine learning* 8, 279-292.

Wu, Z., Shen, C. & Van Den Hengel, A. (2019). Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90, 119-133.

Wu, K., Wang, H., Esfahani, M. A. & Yuan, S. (2019). BND*-DDQN: Learn to Steer Autonomously through Deep Reinforcement Learning. *IEEE Transactions on Cognitive and Developmental Systems*.

Yasnitsky, L. N. (2019). Whether Be New «Winter» of Artificial Intelligence? In *International Conference on Integrated Science* (pp. 13-17). Cham: Springer.

Young, T., Hazarika, D., Poria, S. & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational intelligenCe magazine*, 13(3), 55-75.

Zabalza, J., Ren, J., Zheng, J., Zhao, H., Qing, C., Yang, Z. & Marshall, S. (2016). Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185, 1-10.

Zhang, Z., Geiger, J., Pohjalainen, J., Mousa, A. E. D., Jin, W. & Schuller, B. (2018). Deep learning for environmentally robust speech recognition: An overview of recent

developments. *ACM Transactions on Intelligent Systems and Technology* (TIST), 9(5), 1-28.

Introduction to deep learning

Bettilyon, T. E. (13 de junio de 2018). Introduction to deep learning. *Medium*.

<https://medium.com/tebs-lab/introduction-to-deep-learning-a46e92cb0022>

Un artículo de Medium que nos introduce el deep learning. Puedes encontrar ejemplos prácticos de muchas aplicaciones y tácticas de aprendizaje profundo.

A beginner intro to convolutional neural networks

Gudikandula, P. (23 de marzo de 2019). A beginner intro to convolutional neural networks. *Medium*. <https://medium.com/@purnasaigudikandula/a-beginner-intro-to-convolutional-neural-networks-684c5620c2ce>.

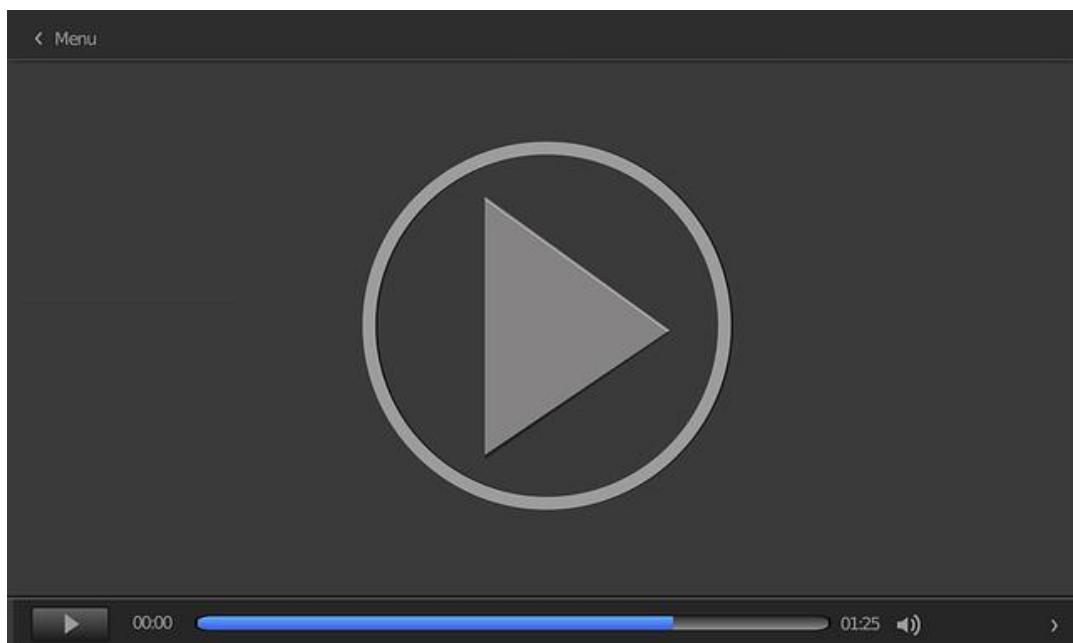
Este artículo nos introduce al mundo de las redes neuronales convolucionales aplicadas al reconocimiento de personas, animales y objetos en imágenes.

Conceptos básicos de deep learning

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://youtu.be/O5xeyoRL95U>

Una conferencia introductoria al *Deep Learning* de Lex Fridman en la que se presentan conceptos básicos y algunas ideas clave, subáreas y el panorama general de por qué las redes neuronales han inspirado a toda una nueva generación de investigadores. Esta conferencia es parte de la serie de conferencias de [Deep Learning](#) del MIT.



Accede al vídeo:

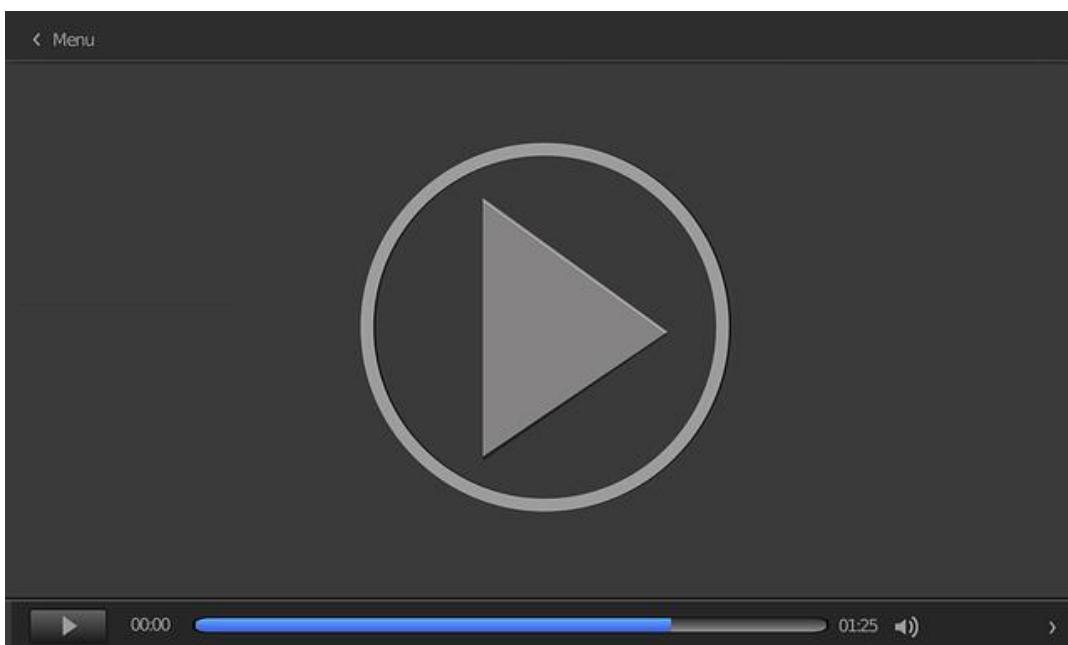
<https://www.youtube.com/embed/O5xeyoRL95U>

Estado del arte del deep learning

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://youtu.be/0VH1Lim8gL8>

Una clase muy completa de Lex Fridman en la que se presenta un estado del arte en I+D sobre *deep learning* a principios de 2020, así como tendencias futuras. Esta conferencia expone un conjunto de aspectos destacados del aprendizaje automático e innovaciones y progresos en *deep learning* y otras técnicas de inteligencia artificial, tanto para la industria como para la sociedad en general. Esta conferencia es parte de la serie de conferencias de [Deep Learning](#) del MIT.



Accede al vídeo:

<https://www.youtube.com/embed/0VH1Lim8gL8>

The neural network zoo

Van Veen, F. (14 de septiembre de 2016). The neural network zoo. *The Asimov Institute*. <https://www.asimovinstitute.org/neural-network-zoo/>

Post de Fjodor Van Veen en el blog del Asimov Institute (instituto de investigación en IA sin ánimo de lucro). Presenta las arquitecturas de las redes neuronales más populares, resumidas en una cheat sheet descargable.

Guía inicial de TensorFlow 2.0 para principiantes

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<https://www.tensorflow.org/tutorials/quickstart/beginner>

Tutorial de TensorFlow 2.0 que incluye ejemplos, como la aplicación de redes neuronales sobre el dataset Fashion-MNIST con imágenes de ropa (y que sustituye al MNIST de Yann Lecun, visto en el Tema 2, con dígitos escritos a mano), como el que se incluye en el apartado 6.12.

Keras

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<https://keras.io/>

Portal de la librería Keras que incluye tutoriales como el que nos introduce al modelo secuencial de capas de redes neuronales, empleado en el ejemplo del tutorial de introducción a TensorFlow. Se explica, así, el proceso de construcción, compilado, entrenamiento y pruebas de las redes. La web incluye, además, toda la documentación de la librería.

CS231n convolutional neural networks for visual recognition

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<https://cs231n.github.io/>

Curso online gratuito de la Universidad de Stanford sobre redes neuronales convolucionales aplicadas al reconocimiento de imágenes. Es uno de los más conocidos en este campo. El módulo 0 nos introduce en la configuración del software y librerías. El módulo 1 es una introducción a las redes neuronales y el módulo 2 describe las redes convolucionales.

- 1.** ¿Cuáles de las siguientes ramas pertenecen al aprendizaje automático clásico?

 - A. Aprendizaje supervisado.
 - B. Aprendizaje ensemble.
 - C. Deep Learning.
 - D. Aprendizaje por refuerzo.

- 2.** Indica todas las respuestas correctas. Los autoencoders:

 - A. Siempre tienen un número par de capas
 - B. Son redes neuronales asimétricas.
 - C. Se pueden emplear para la compresión de imágenes.
 - D. Son un tipo de redes convolucionales profundas.

- 3.** Señala todas las respuestas correctas. Los clasificadores Naïve Bayes:

 - A. Son un tipo de redes de creencias.
 - B. Son un tipo de redes Bayesianas.
 - C. Se basan en que las características son fuertemente dependientes entre sí.
 - D. Tienen un número impar de capas de neuronas recurrentes.

- 4.** Señala todas las respuestas correctas. Las redes generativas antagónicas:

 - A. Están siempre formadas por una red convolucional y una red deconvolucional.
 - B. Pueden estar formadas por una red convolucional y una red prealimentada.
 - C. Están formadas por una red generativa y una red discriminativa.
 - D. No son apropiadas para aplicaciones Deepfake al no ser redes profundas.

5. Señala todas las respuestas incorrectas. Las técnicas de aprendizaje por refuerzo:

- A. Están siempre basadas en algoritmos evolutivos.
- B. Pueden aplicarse en el ámbito de los vehículos autónomos.
- C. Nunca requieren un dataset de entrenamiento, pues no necesitan modelos.
- D. Se pueden aplicar para crear software que comercie de forma automatizada.

6. En un escenario de aprendizaje por refuerzo, un agente actúa con una tasa de exploración igual a 0.9 en un momento dado. Señala las respuestas incorrectas:

- A. En el estado actual, el agente explorará el 90 % del entorno y, tras ello, dará por finalizada la iteración.
- B. Existe una probabilidad del 10 % de que el agente decida explorar el entorno.
- C. Existe una probabilidad del 90 % de que el agente no decida explotar las recompensas existentes en el entorno por medio de sus acciones.
- D. Decidirá explorar el entorno con un 90 % de probabilidades solo si esta es su primera iteración.

7. Señala las respuestas correctas. Las memorias de largo y corto plazo (LSTM – long and short-term memories):

- A. Surgieron para mejorar los perceptrones multicapa, incrementando el número de capas ocultas.
- B. Tienen celdas especiales en las que un valor puede ser almacenado, leído o restablecido, por medio de puertas de entrada, salida y olvido.
- C. Son menos expresivas que su variación las Gated recurrent units (GRU).
- D. Son apropiadas para aplicaciones de reconocimiento de voz.

8. Señala todas las respuestas correctas:

- A. TensorFlow es una librería de redes neuronales utilizada como alternativa a Keras.
- B. TensorFlow solo puede ser empleado bajo lenguaje Python.
- C. TensorFlow Lite permite la aplicación de redes neuronales en dispositivos móviles.
- D. TensorFlow es una librería específica para resolver problemas de mecánica mediante análisis numérico.

9. Señala todas las respuestas correctas:

- A. El algoritmo Q-Learning puede trabajar con datos de entrenamiento etiquetados y no etiquetados.
- B. El algoritmo Q-Learning presenta un problema de sobreestimación del valor de la acción en algunos escenarios.
- C. El algoritmo Double Q-Learning utiliza siempre dos redes neuronales con la misma arquitectura de capas.
- D. El algoritmo Double Q-Learning presenta un problema de sobreestimación del valor de la acción en todos los escenarios.

10. Señala todas las respuestas correctas:

- A. Las redes deconvolucionales pueden aplicarse para obtener una imagen sintética a partir de una etiqueta.
- B. Las redes gráficas inversas convolucionales profundas (DCIGN) son autoencoders variacionales en los que cuales se utilizan dos redes deconvolucionales.
- C. Las redes convolucionales no son aptas para aplicaciones de reconocimiento facial.
- D. En las redes convolucionales la última etapa de convolución se conecta siempre a una etapa deconvolucionadora para obtener las probabilidades de que una entrada pertenezca a una clase.

Técnicas de Inteligencia Artificial

Tema 7. Clustering. Agrupamiento o clasificación no supervisada

Índice

Esquema

Ideas clave

- 7.1. ¿Cómo estudiar este tema?
- 7.2. Conceptos. Tipos de algoritmos de clustering.
Medida de distancia
- 7.3. Agrupamiento exclusivo. El algoritmo k-means
- 7.4. Agrupamiento jerárquico. Algoritmo de agrupamiento jerárquico aglomerativo
- 7.5. Agrupamiento probabilista. El algoritmo EM
- 7.6. Agrupamiento solapado. El algoritmo Fuzzy C-means
- 7.7. Aplicaciones y ejemplos de implementación
- 7.8. Referencias bibliográficas

A fondo

Algoritmos de clustering en Weka

Demostración de k-means mediante un ejemplo de cartas de póker

Agrupamientos solapados, clustering difuso

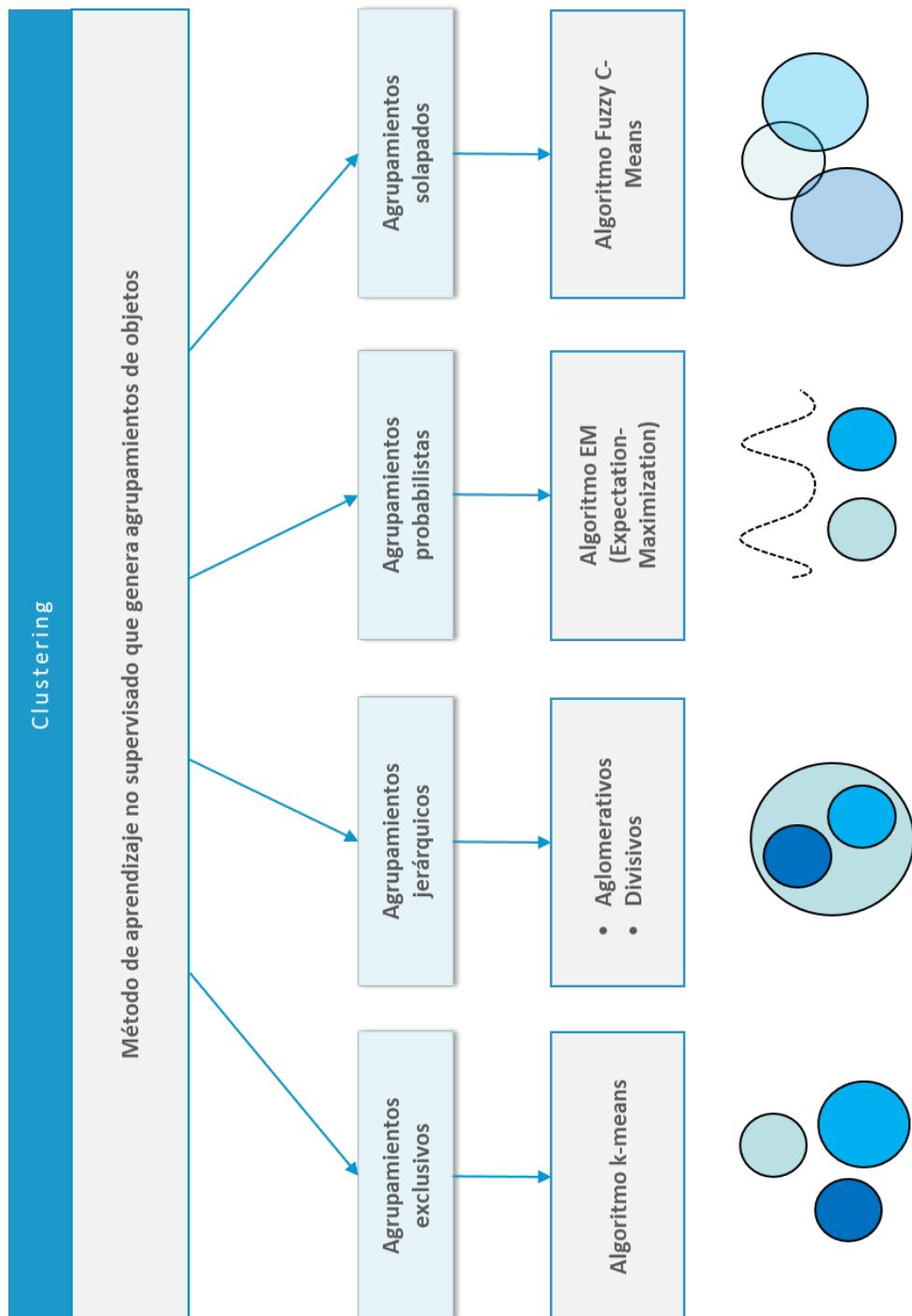
Métodos jerárquicos de análisis clúster

Demostraciones del algoritmo EM

Scikit-learn

Machine Learning Mastery

Test



7.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral, revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan. Es muy recomendable ver la lección magistral antes de realizar las actividades propuestas.

Al finalizar el estudio de este tema el alumno será capaz de:

- ▶ Describir los diferentes tipos de agrupamientos y los algoritmos representativos para obtener estos tipos.
- ▶ Aplicar técnicas de *clustering* para la resolución de problemas de aprendizaje no supervisado.
- ▶ Identificar aplicaciones prácticas de técnicas de *clustering*.

7.2. Conceptos. Tipos de algoritmos de clustering. Medida de distancia

El *clustering* es uno de los métodos de aprendizaje no supervisado más importantes y, como cualquier otro método de aprendizaje no supervisado, busca caracterizar conceptos desconocidos a partir de instancias de los mismos. En este tipo de problemas de aprendizaje no supervisado la clase es desconocida y es, precisamente el descubrimiento de esta clase el objetivo de este aprendizaje, a través de la agrupación de instancias en base a un esquema de similitud (Kaufman & Rousseeuw, 1990; Xu & Wunsch, 2009).

Clustering

Es un método de aprendizaje no supervisado que permite agrupar objetos en clústeres o agrupamientos, cuyos miembros son similares entre sí en cierto modo.

Por lo tanto, el resultado de aplicar una técnica de *clustering* es una serie de agrupamientos o clústeres, los cuales son particiones de un conjunto de instancias.

Clúster

Es una colección de objetos similares entre sí y diferentes a los objetos que pertenecen a otros clústeres.

Diferentes algoritmos de *clustering* pueden dar lugar a diferentes agrupamientos y no es siempre fácil determinar qué es un buen agrupamiento. La calidad de los agrupamientos creados dependerá de la aplicación final, por lo cual es el usuario el que determinará la calidad en base a si los agrupamientos creados son útiles y se ajustan a sus necesidades. Por lo tanto, según el objetivo del problema a tratar, habrá que seleccionar un algoritmo u otro.

Por ejemplo, no es lo mismo tratar de detectar el espacio donde hay una mancha de combustible en el mar, espacio que puede tener una forma irregular, que el tratar de obtener agrupamientos circulares y de similar tamaño con el fin de segmentar un grupo de clientes para ofrecer productos.

A continuación, se enumeran diferentes tipos de algoritmos de *clustering* en base al tipo de agrupamientos que producen:

- ▶ **Agrupamientos exclusivos.** En este tipo de agrupamiento cada uno de los clústeres tiene al menos un objeto y los objetos se agrupan de modo exclusivo, pudiendo pertenecer únicamente a un clúster. Los clústeres pueden generarse por métodos que agrupan los datos creando un número k determinado de clústeres. Generalmente estos métodos de partición utilizan una **medida de distancia** para generar los clústeres. Un ejemplo de este tipo es el algoritmo **k-means** (Qin *et al.*, 2016). También hay otro tipo de algoritmos que típicamente generan clústeres exclusivos, tales como los métodos basados en la **densidad** que van incorporando nuevos objetos a un clúster si la densidad de objetos en «los alrededores» supera un umbral. Los **algoritmos basados en densidad** (en vez de en medidas típicas de distancia) son útiles para generar clústeres de formas irregulares. Algunos de los algoritmos que generan clústeres exclusivos más conocidos son *mean-shift* (Guo *et al.*, 2018) y el DBSCAN (*Density-based spatial clustering of applications with noise*) o *agrupamiento espacial basado en densidad de aplicaciones con ruido* (Shen *et al.*, 2016).
- ▶ **Agrupamientos jerárquicos.** Otro tipo de algoritmos son aquellos que dan lugar a una estructura jerárquica de clústeres. En el primer nivel de la jerarquía se tiene un único clúster que se divide en otros clústeres en una primera iteración. Cada uno de estos clústeres se divide a su vez y se van generando nuevos clústeres en siguientes iteraciones. Esta forma de crear los clústeres partiendo de un gran clúster que contenga todos los datos e irlo dividiendo para formar clústeres más pequeños se denomina aproximación **divisoria**. También existen algoritmos que generan clústeres en el sentido inverso, en este caso denominados **aglomerativos**, los cuales generan en primer lugar los clústeres más pequeños que se agrupan progresivamente para generar la estructura jerárquica (Bouguettaya *et al.*, 2015).

- ▶ **Agrupamientos solapados.** En este tipo de clústeres los objetos se agrupan a través de conjuntos difusos, pudiendo cada objeto pertenecer a uno o más clústeres con diferentes grados de pertenencia. Un algoritmo que produce agrupamientos solapados es el **algoritmo Fuzzy C-means** (Qin *et al.*, 2016).
- ▶ **Agrupamientos probabilistas.** Los clústeres se generan mediante un método probabilístico. Un ejemplo representativo es el **algoritmo EM (Expectation-Maximization)** (Gebru *et al.*, 2016).

El *clustering* se puede utilizar aisladamente con el fin de analizar agrupamientos y extraer conclusiones, pero también se utiliza como etapa previa a la aplicación de otras técnicas. Por ejemplo, se puede aplicar el *clustering* para detectar clases desconocidas, asignar etiquetas a las clases descubiertas correspondientes a los distintos grupos y, a continuación, aplicar una técnica de clasificación para describir los grupos y realizar clasificaciones de instancias futuras desconocidas (G. Sreenivasulu *et al.*, 2017).

Algunos ejemplos de aplicaciones prácticas del *clustering* son:

- ▶ Encontrar objetos representativos de grupos homogéneos. Por ejemplo:
 - En un sistema de aprendizaje en línea se pueden obtener agrupamientos de estudiantes que comparten características y comportamientos comunes en el estudio. Analizando cada grupo se pueden definir estrategias de enseñanza adecuadas a los diversos comportamientos similares, *a priori* desconocidos (Zakrzewska, 2009).
 - En un sistema de gestión de clientes, las técnicas de *clustering* pueden agrupar los diversos clientes en base a características comunes y permitir aplicar estrategias de *marketing* adecuadas a los diversos grupos, posicionar productos, etc. (Abbasimehr

& Shabani, 2019; Chiang, 2018; Holý et al., 2017).

- ▶ Encontrar grupos y describirlos en base a sus propiedades. Por ejemplo, en biología, clasificación de plantas, o en medicina, clasificación de enfermedades raras (Nilashi et al., 2017b, 2017a; Pacheco & López, 2019).
- ▶ Detección de casos anómalos. Por ejemplo, en un sistema de detección de fraudes de tarjetas de crédito o de estafas a seguros, puede servir para detectar comportamientos de clientes que no encajan dentro de ningún agrupamiento y, por tanto, son comportamientos a analizar cuidadosamente, dado que podría tratarse de un fraude (Kumar et al., 2019; Subudhi & Panigrahi, 2017).

Medida de distancia

Las medidas de distancia son utilizadas en un importante número de algoritmos de *clustering*, puesto que muchos algoritmos se basan en medidas de distancia entre objetos para, en función de su «cercanía», incluirlos en el mismo clúster o en clústeres separados.

Según la medida de distancia que se escoja, un algoritmo dará lugar a diferentes clústeres y, por tanto, seleccionar la medida de distancia apropiada es importante, aunque no siempre se puede saber a ciencia cierta cuál es la medida óptima. Es muy típicamente utilizada la medida de distancia euclídea, por ejemplo.

Pero hay otro tipo de medidas muy utilizadas también típicamente por algoritmos de *clustering* no tan extendidas. A continuación, se describen algunas de estas medidas de distancia conocidas como *linkage measures* (medidas de conectividad):

- ▶ **Enlace sencillo(*single-linkage*).** La similitud entre dos clústeres se calcula como la similitud de los **dos puntos más cercanos** pertenecientes a los diferentes clústeres. Así, se considera la distancia más pequeña existente entre uno de los puntos del primer clúster y otro del segundo (ver Figura 1). Siendo p y p' los puntos en los que dos objetos se sitúan, y CA y CB , dos clústeres tal que $p \in CA$ y $p' \in CB$, la similitud se calcula según la siguiente expresión:

$$dist_{single-link} (C_A, C_B) = \min_{p \in C_A, p' \in C_B} \{|p - p'|\}$$

- ▶ **Enlace completo (*complete-linkage*).** Caso opuesto al enlace sencillo ya que se tiene en cuenta la distancia mayor existente entre cualquier punto de uno y otro clúster (Ver Figura 2). Siendo p y p' los puntos en los que dos objetos se sitúan, y CA y CB , dos clústeres tal que $p \in CA$ y $p' \in CB$, la similitud en este caso se calcula según la siguiente expresión:

$$dist_{complete-link} (C_A, C_B) = \max_{p \in C_A, p' \in C_B} \{|p - p'|\}$$

- **Enlace promedio (*average-linkage*).** La distancia entre dos clústeres se calcula como la distancia media entre cualquier punto del primer clúster con cualquier punto del segundo (ver Figura 3), tal y como describe la siguiente expresión:

$$dist_{complete-link} (C_A, C_B) = \frac{1}{n_A n_B} \sum_{p \in C_A, p' \in C_B} |p - p'|$$

Siendo p y p' los puntos en los que dos objetos se sitúan, CA y CB , dos clústeres con un número de elementos nA y nB respectivamente, y tal que $p \in CA$ y $p' \in CB$.

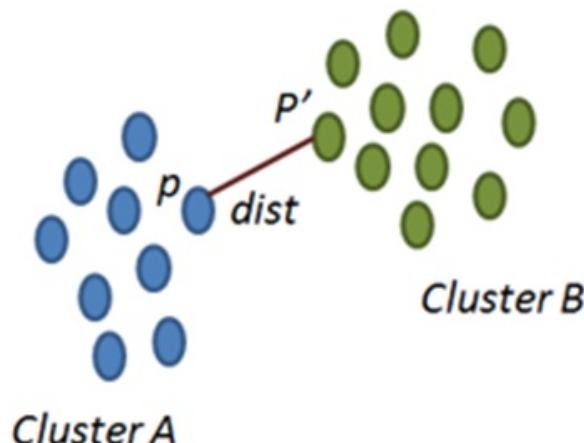


Figura 1. Enlace sencillo entre dos clústeres.

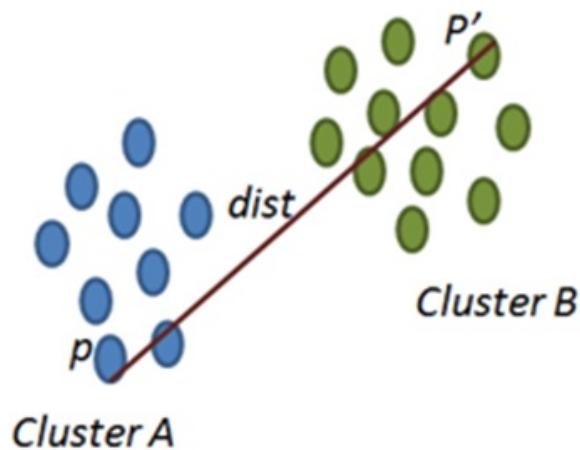


Figura 2. Enlace completo entre dos clústeres.

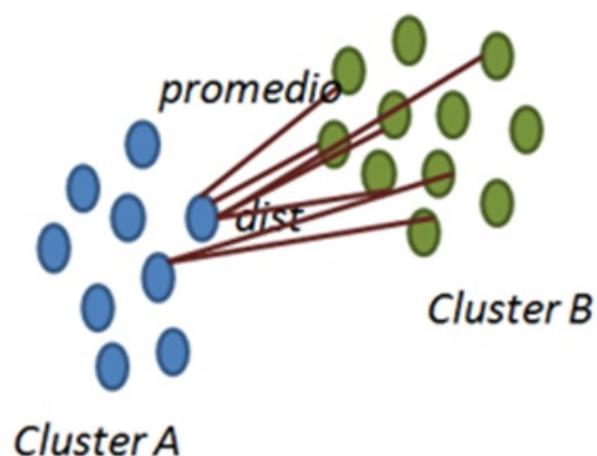


Figura 3. Enlace promedio entre dos clústeres.

7.3. Agrupamiento exclusivo. El algoritmo k-means

Los algoritmos más simples de *clustering* son los que dividen el conjunto de objetos en agrupaciones exclusivas de objetos teniendo como punto de partida el número de clústeres que se desea generar y midiendo la similitud de los objetos en base a una medida de distancia. El algoritmo más popular de este tipo es el **algoritmo k-means**.

Algoritmo K-means

Es un método iterativo basado en distancia que define k centroides de los k clústeres que genera en cada iteración. Un objeto pertenece a un clúster o a otro en base a su distancia a los centroides de los distintos clústeres. El centroide de un clúster lo calcula como el valor medio de los puntos del clúster.

El algoritmo k-means resuelve el problema de agrupamiento siguiendo los siguientes pasos, partiendo de un número determinado k de clústeres definido *a priori*:

- ▶ Selecciona k objetos del **conjunto inicial de manera aleatoria o siguiendo alguna pauta como, por ejemplo, que los objetos estén lo más lejos posible uno de otros**. Cada uno de estos k objetos representan cada uno de los centroides de los k clústeres.
- ▶ El resto de los objetos se asigna al clúster cuyo centroide es más similar. **Esta similitud se calcula mediante una medida de distancia entre el objeto y el centroide**. En este paso ya se obtienen k agrupamientos o clústeres iniciales.
- ▶ Para cada clúster generado en el paso previo, se **recalcula la posición del centroide** o media de los puntos.
- ▶ Los pasos 2-3 se repiten hasta que las posiciones de los centroides no varían.

Aunque el algoritmo k-means puede utilizar diversas medidas de distancia, típicamente utiliza la distancia euclídea que da buenos resultados en muchos problemas.

Para ilustrar gráficamente la creación de estos clústeres, la Figura 4 muestra cinco agrupamientos, resaltados en diversos colores, generados por la herramienta Weka para un conjunto de datos bidimensionales. Se puede observar cómo efectivamente cada clúster presenta una serie de objetos que se agrupan alrededor de la media del mismo. Nótese que en este caso la visualización es muy clara porque los datos de entrada son bidimensionales. Sin embargo, cuando los datos tienen muchas dimensiones, es muy difícil interpretar los clústeres obtenidos a partir de una visualización o análisis y, por tanto, se asigna una clase a cada clúster y se utilizan técnicas de aprendizaje supervisado para describir los mismos.

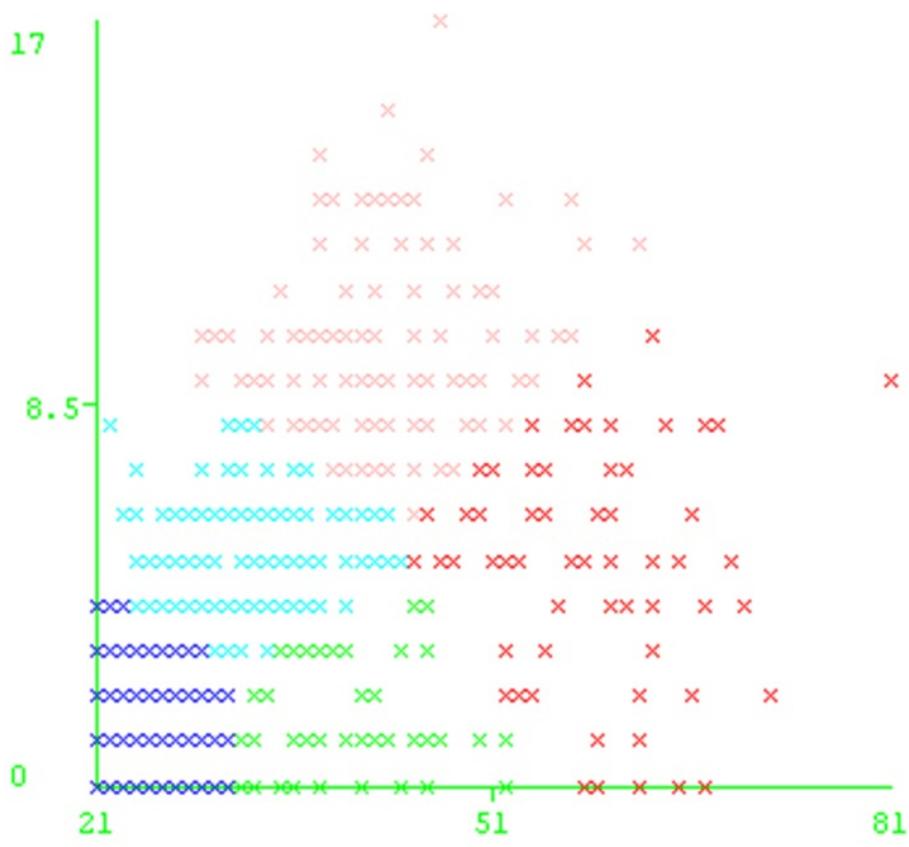


Figura 4. Agrupamientos generados por k-means con la herramienta Weka.

A la hora de utilizar el algoritmo k-means es difícil saber si se está escogiendo el valor k adecuado.

Más aún, el algoritmo es sensible al posicionamiento inicial de los centros de los clústeres. Por tanto, en la práctica se suele ejecutar diversas veces para diversos valores de k y para diversos posicionamientos iniciales de los centroides.

De cualquier manera, se ha de tener cuidado con valores altos de k que podrían dar lugar a problemas de *overfitting* o sobreentrenamiento.

Una limitación del algoritmo es que solo trabaja con datos de valores reales. Solo se puede aplicar cuando el cálculo de la media de los objetos está definido y, por tanto, podría no ser aplicable cuando se manejan valores nominales. Una posible solución es mapear el valor nominal a un valor numérico.

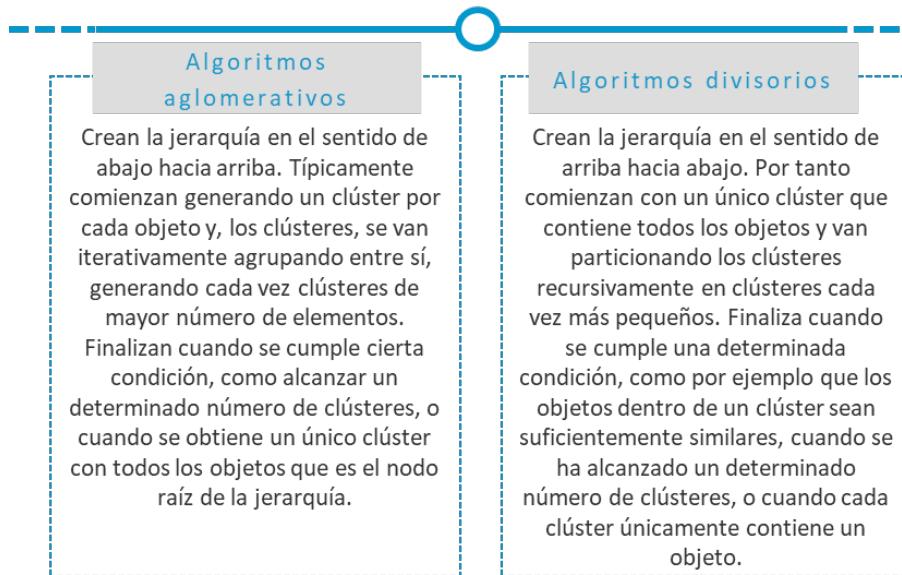
A pesar de las limitaciones y dificultades expuestas, k-means es un método simple y relativamente escalable que en algunos problemas da muy buenos resultados.

7.4. Agrupamiento jerárquico. Algoritmo de agrupamiento jerárquico aglomerativo

Existen problemas donde se desea obtener **jerarquías de clústeres**. Por ejemplo, en un comercio electrónico se puede desear organizar productos en agrupaciones genéricas y luego, dentro de esas agrupaciones, generar nuevas agrupaciones en base a similitudes entre los productos. Esto tiene el beneficio muchas veces de **facilitar el análisis y la visualización de los datos**.

Los métodos de agrupamiento jerárquico trabajan de manera incremental, a diferencia del algoritmo k-means que en cada iteración trabaja con el conjunto completo de datos.

Existen dos tipos de algoritmos jerárquicos:



La Figura 5 muestra el sentido de cómo se dividen o agrupan los clústeres según se trate de un algoritmo divisorio o aglomerativo.

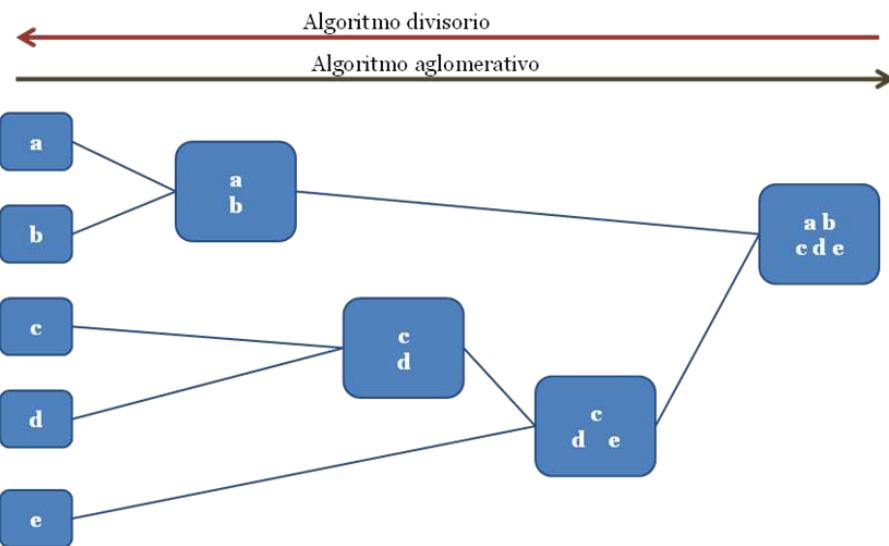


Figura 5. *Clustering* jerárquico aglomerativo y divisorio.

Como se aprecia en la Figura 5 estos algoritmos generan una estructura jerárquica en forma de árbol, siendo el nodo raíz el clúster que contiene el mayor número de elementos y los nodos hojas, los clústeres con menor número de elementos. Cuando se trabaja con algoritmos jerárquicos es frecuente encontrar la representación mostrada en la Figura 6 denominada dendograma.

En esta Figura 6 se representa cómo se agrupan por niveles los objetos en cada iteración del algoritmo de *clustering*. Conforme el clúster tiene un mayor número de objetos, la similitud entre sus objetos disminuye.

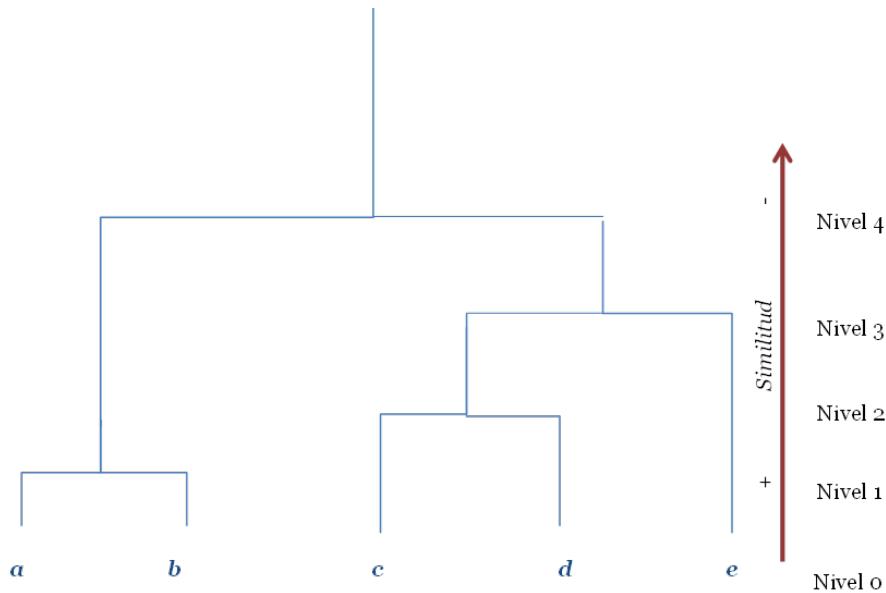
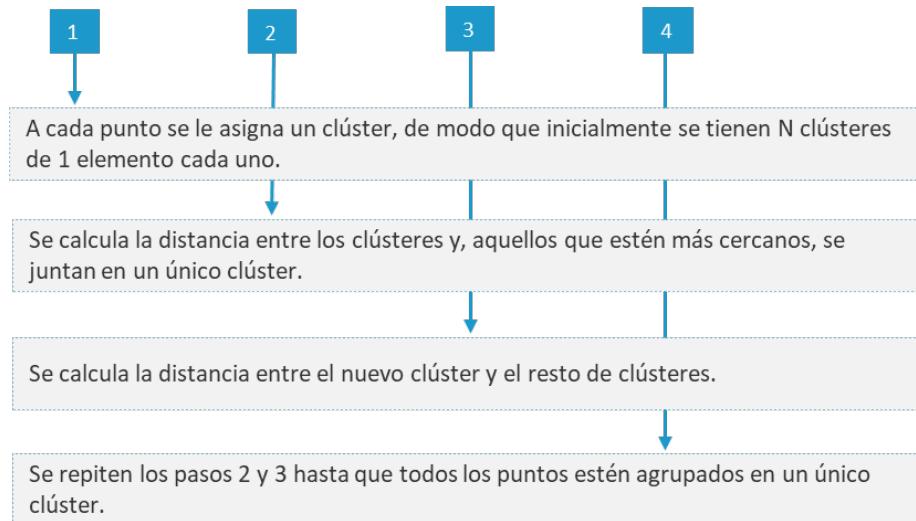


Figura 6. Dendograma representando la jerarquía correspondiente al ejemplo de la Figura 5.

Los algoritmos divisorios son menos frecuentes que los aglomerativos. Una dificultad que presentan es la decisión de cómo dividir un clúster con muchos elementos en clústeres más pequeños, puesto que las posibles combinaciones de diferentes clústeres a partir de un numeroso número de instancias son igualmente muy numerosas.

En los métodos de agrupamiento jerárquico una decisión crucial es **precisamente la decisión de en qué punto dividir o conglomerar** (según se trate de algoritmos divisorios o aglomerativos respectivamente). La medida de la calidad de las particiones de los objetos en clústeres que se utiliza para tomar esta decisión se denomina **utilidad de la categoría**.

Se describe a continuación el funcionamiento de un algoritmo jerárquico aglomerativo básico basado en la medida del enlace sencillo (también pueden emplearse criterios de enlace completo, enlace medio (entre clústeres y dentro de los clústeres), método del centroide, de la mediana, etc.). Dado un conjunto de N puntos, el algoritmo procede de la siguiente forma:



Una vez se ha completado el árbol jerárquico o el dendograma mediante este método, si se quieren obtener un número determinado k de clústeres, simplemente hay que tomar los clústeres del nivel con ese número de clústeres.

El cálculo en cada iteración de las distancias entre el nuevo clúster generado y el resto se puede realizar de diferentes formas, tales como a través de la medida de la distancia de enlace sencillo, enlace completo, promedio, etc.

A continuación, se particulariza el algoritmo jerárquico expuesto previamente para el caso de emplear el enlace-sencillo como método para calcular la similitud entre clústeres. Se utiliza una **matriz de similitud** entre los distintos clústeres de dimensión inicial $N \times N$, conteniendo las distancias entre los distintos puntos (por lo cual también se la denomina matriz de distancias), y los pasos son los siguientes:

- ▶ Se comienza por el nivel 0 en el que se genera un clúster por cada uno de los N puntos del conjunto inicial de instancias.
- ▶ Se busca el par de clústeres C_A y C_B menos distantes teniendo en cuenta la medida de enlace sencillo entre cualquier emparejamiento de clústeres posible.

- ▶ Se unen los clústeres C_A y C_B en uno solo.
- ▶ Se elimina de la matriz de proximidad las filas y columnas que corresponden a los clústeres C_A y C_B y se añade una columna y una fila para el nuevo clúster generado. La proximidad entre el nuevo clúster y los anteriores se recalcula mediante la medida de enlace sencillo.
- ▶ Si todos los objetos están en un único clúster, el algoritmo finaliza. En caso contrario se continua en el paso 2.

Se va a ilustrar a continuación el funcionamiento de este algoritmo con un ejemplo muy sencillo de un problema en el que se pretende generar una estructura de agrupamientos jerárquica a partir de unos datos bidimensionales. Las distancias entre estos datos se muestran en la matriz de similitud incluida en la Tabla 1.

	s	t	u	w
s	0	4	8	3
t	4	0	5	10
u	8	5	0	2
w	3	10	2	0

Tabla 1. Matriz de similitud de partida del ejemplo.

En primer lugar, se crean tantos clústeres como objetos hay (ver Figura 7). A continuación, se busca el par de clústeres menos distantes entre sí, que son aquellos cuya distancia es menor. Según la matriz de la Tabla 1, los clústeres correspondientes a los objetos u y w son los más cercanos, dando lugar al clúster que se denomina C_{u-w} (ver Figura 8).



Figura 7. Iteración 1 del algoritmo de clustering: 1 clúster por cada objeto.

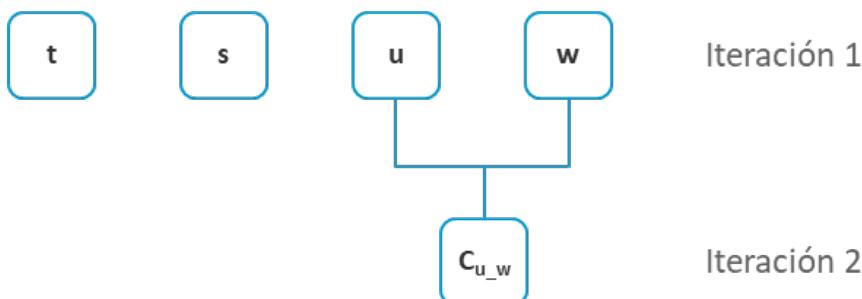


Figura 8. Iteración 2 del algoritmo de clustering.

A continuación, se calcula la distancia de este nuevo clúster C_{uw} con el resto de los clústeres, teniendo en cuenta la medida de enlace sencillo. Por tanto, habrá que tener en cuenta para calcular la distancia al resto de clústeres, la distancia del punto más cercano, sea w o u , al clúster objetivo. Así se obtiene la matriz de similitud mostrada en la Tabla 2.

	s	t	C_{u-w}
s	0	4	3
t	4	0	5
C_{u-w}	3	5	0

Tabla 2. Matriz de similitud del ejemplo (iteración 2).

En la siguiente iteración, la mínima distancia tal y como se observa en la Tabla 2, se encuentra entre el nuevo clúster generado C_{uw} y el punto s . Por tanto, se unen ambos clústeres dando lugar al clúster C_{uws} (ver Figura 9) y se recalcula la matriz de similitud obteniéndose la que se muestra en la Tabla 3.

	t	C_{u-w-s}
t	0	4
C_{u-w-s}	4	0

Tabla 3. Matriz de similitud del ejemplo (iteración 3).

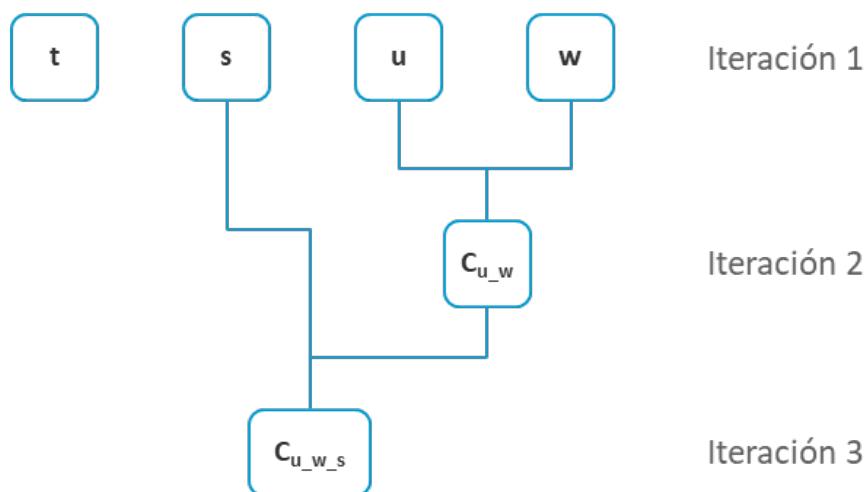


Figura 9. Iteración 3 del algoritmo de *clustering*.

Como último paso solo queda unir los dos únicos clústeres existentes generando un único clúster, y finalizando el algoritmo. El árbol generado se muestra en la Figura 10.

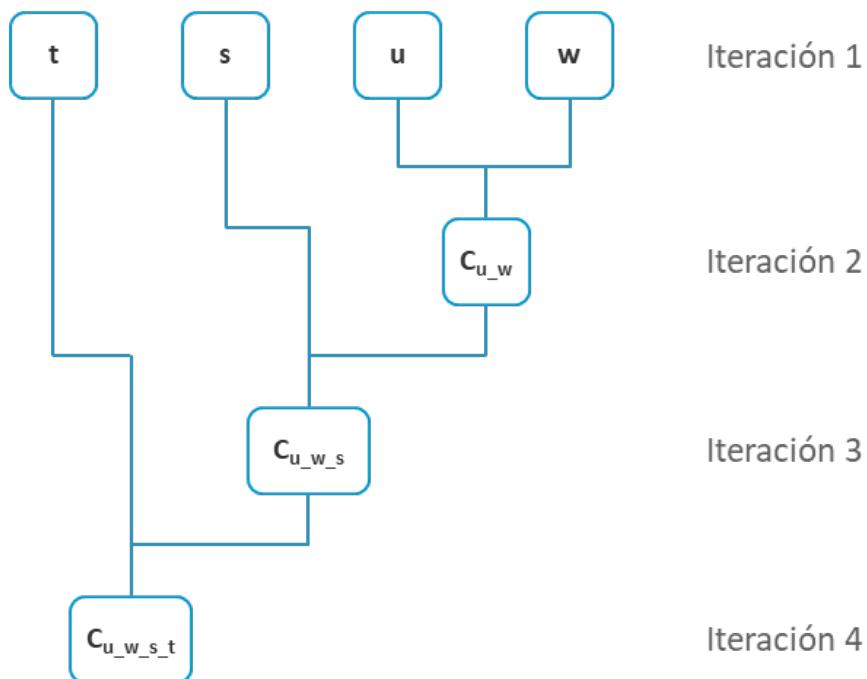


Figura 10. Árbol jerárquico resultante en la iteración 4 del algoritmo de clustering.

El problema de estos métodos de *clustering* jerárquico es que no escalan bien según aumenta el número de puntos. Por otra parte, son métodos que en su forma básica no dan marcha atrás con lo cual se podría no encontrar la solución óptima. También ha de tenerse en cuenta que la elección del criterio marca un mejor o peor funcionamiento de la técnica. Así, el enlace simple proporciona clústeres encadenados mientras que el enlace completo produce clústeres más compactos y es menos sensible a *outliers*. Menos sensible aún a estos es el método del enlace medio o el método de cálculo de centroides (que tiene a formar clústeres compactos, igual tamaño y forma).

7.5. Agrupamiento probabilista. El algoritmo EM

Los algoritmos basados en probabilidad, en vez de situar las instancias en clústeres de una manera exclusiva, indican la **probabilidad de que las instancias pertenezcan a cada uno de los clústeres**.

La base de estos algoritmos es el modelo estadístico denominado **mezclas finitas (*finite mixtures*)**. Una mezcla es un conjunto de k distribuciones de probabilidad que representan k clústeres. Esas distribuciones se refieren a los valores de los atributos de las instancias que pertenecen a un clúster.

De este modo, si se conociera que una instancia es miembro de un clúster, las distribuciones mostrarían la probabilidad de que esa instancia tenga unos determinados valores de atributos.

En muchos problemas no todos los clústeres son iguales de probables, no tienen por qué tener el mismo número de individuos. Por este motivo, es necesario que también se refleje la distribución de la población entre los clústeres.

Clustering probabilista

Dado un conjunto de instancias con sus atributos, el problema que se presenta es **conocer los parámetros que modelan el conjunto de k distribuciones de probabilidad para los k clústeres**, así como la **probabilidad de población**, esto es, que el propio clúster tenga cierto número de instancias.

Un caso sencillo sería tener instancias de un único atributo y tres clústeres que se modelan como una distribución normal con desviaciones media y estándar μ_A y σ_A para el clúster A , μ_B y σ_B para el clúster B , así como μ_C y σ_C , para el clúster C . Las muestras se distribuyen en los clústeres con las probabilidades p_A , p_B y p_C . El problema consiste, por tanto, en modelar esos clústeres, esto es, determinar los valores: μ_A, σ_A , μ_B , σ_B , μ_C , σ_C , p_A , p_B y p_C .

¿Cómo se pueden modelar estos clústeres y determinar dichos parámetros?

Se puede utilizar un proceso similar al del algoritmo k-means.

El algoritmo k-means itera con dos etapas:

- ▶ A cada objeto se le asigna el clúster cuyo centroide está más cercano (los centroides en la primera iteración se suelen determinar aleatoriamente).
- ▶ Una vez se asignan los objetos a cada clúster, se recalcula el centroide de tal manera que la suma de las distancias de los objetos asignados al clúster al centroide sea mínima.

Entonces se puede comenzar con valores «posibles» de esos parámetros que modelan los clústeres, tal y como se «adivina» el centroide en el algoritmo k-means en la primera iteración, y utilizar estos hipotéticos valores para calcular las probabilidades de que cada instancia esté en un clúster. En la segunda etapa se pueden utilizar estas probabilidades para reestimar los parámetros. Este proceso se repetiría iterativamente. Así funciona el algoritmo EM (Expectation – Maximization).

Al igual que el algoritmo k-means, el algoritmo EM itera hasta que no se pueden mejorar los agrupamientos y cada iteración consta de dos etapas:

- ▶ **Esperanza (expectation):** calcula las probabilidades de pertenencia de las instancias a los clústeres de acuerdo con los parámetros de los clústeres probabilísticos.
- ▶ **Maximización (maximization):** determina los nuevos parámetros que maximizan la verosimilitud esperada en el anterior paso (*expected likelihood*).

El algoritmo EM se puede utilizar siempre que se conozca o se estime una forma de la distribución probabilística (por ejemplo, se estima la distribución con forma de gaussiana y se pretende conocer los parámetros que modelan la gaussiana). Es habitual hablar en estos casos de algoritmos EM-GMM (*Expectation-Maximization for Gaussian Mixture Models*).

En muchas aplicaciones el *clustering* probabilístico ha resultado efectivo dado que es más general que otros métodos de *clustering*. Específicamente el algoritmo EM se utiliza frecuentemente porque es simple. Sin embargo, presenta el problema de poder converger a un óptimo local y no global y podría llegar a tener un alto coste computacional si el número de distribuciones probabilísticas que es necesario modelar es elevado.

7.6. Agrupamiento solapado. El algoritmo Fuzzy C-means

El algoritmo k-means agrupa las instancias en grupos exclusivamente. Sin embargo, mediante un algoritmo de agrupamiento solapado, como el algoritmo Fuzzy C-means, una instancia puede pertenecer a más de un grupo o clúster, con un determinado grado de pertenencia.

El algoritmo Fuzzy C-means es un método muy utilizado para obtener clústeres solapados. Permite encontrar una serie de conjuntos difusos representativos de cada clúster y los grados de pertenencia de cada instancia a los mismos.

El funcionamiento de este algoritmo es el siguiente. Dado un conjunto de objetos $x_1, x_2, \dots, x_i, \dots, x_n$, un agrupamiento de los mismos mediante k clústeres difusos $C_1, C_2, \dots, C_j, \dots, C_K$ se puede representar utilizando una matriz divisoria $M = [p_{ij}]$, tal que $1 \leq i \leq n$ y $1 \leq j \leq k$, siendo p_{ij} el grado de pertenencia del objeto x_i al clúster C_j .

El algoritmo Fuzzy C-means se basa en minimizar la siguiente función objetivo:

$$f_m = \sum_{i=1}^n \sum_{j=1}^K p_{ij}^m \|x_i - c_k\|^2$$

Donde:

- ▶ m es un número real mayor o igual a 1 que gobierna la influencia de los grados de pertenencia.
- ▶ p_{ij} es el grado de pertenencia del objeto x_i al clúster C_j incluido en la matriz divisoria antes mencionada.
- ▶ x_i es la i -ésima instancia.

- ▶ q_j es el centro del clúster C_j que se puede calcular como la media, el centroide o cualquier otra fórmula adecuada al problema en concreto.
- ▶ $\| \cdot \|$ se refiere a cualquier medida relativa a la similitud entre la instancia y el centro del clúster.

Los pasos que se siguen son:

- ▶ Inicializar la matriz divisoria de manera aleatoria, pero cumpliendo que:
 - Cada valor asignado p_{ij} está entre 0 y 1.
 - Para cada objeto x_i , se cumple:

$$\sum_{j=1}^K p_{ij} = 1.$$

- ▶ En la iteración t , calcular los centros de los clústeres en base a la siguiente expresión:

$$c_j = \frac{\sum_{i=1}^n p_{ij}^m x_i}{\sum_{i=1}^n p_{ij}^m}$$

- ▶ Actualizar la matriz $M^{(t)}$ obteniendo $M^{(t+1)}$ con la siguiente fórmula:

$$p_{ij} = \frac{1}{\sum_{l=1}^k \frac{\|x_i - c_l\|^{\frac{2}{m-1}}}{\|x_i - c_j\|}}$$

- ▶ Finalizar si se alcanza el criterio de parada que puede establecerse en un número determinado de iteraciones o en que la variación de la matriz sea muy pequeña en la iteración tal que:

$$\|M^{(t+1)} - M^{(t)}\| < \varepsilon$$

Si no se cumple este criterio de parada el algoritmo vuelve al paso 2.

El algoritmo Fuzzy C-means guarda similitud con el algoritmo EM, pero una diferencia importante es precisamente el tipo de clúster objetivo. Tanto k-means como Fuzzy C-means modelan los clústeres típicamente con forma circular, mientras que el algoritmo EM emplea funciones probabilísticas dando lugar a los clústeres probabilísticos modelados por funciones probabilísticas.

7.7. Aplicaciones y ejemplos de implementación

Algunas aplicaciones de los algoritmos de clustering en la literatura

Los algoritmos de *clustering* o agrupación se utilizan para segmentar el mercado en diferentes tipos de consumidores (especialmente útiles para ser utilizados más tarde para recomendar productos y fidelizar a los clientes) (Casabayó, Agell, y Sánchez-Hernández, 2015), compresión de imágenes (Hoeltgen & Breuß, 2018), analizar y etiquetar nuevos datos, detectar comportamientos anómalos como la detección de fraudes (Ahmed, Mahmood e Islam, 2016) detectar objetos a partir de datos de sensores (Li *et al.*, 2018), o para mezclar puntos cercanos en un mapa (utilizado para facilitar la visualización de nodos en los mapas dependiendo del nivel de zoom) (Huang, Liu y Ling, 2019).

Ejemplos de implementación mediante Python

En esta ocasión vamos a probar varios algoritmos de *clustering* utilizando Python y las librerías numpy, scikit-learn y matplotlib , que ya deberíamos tener todas instaladas de los ejemplos de los temas anteriores. No vamos a utilizar pandas , puesto que vamos a realizar pruebas generando **datos sintéticos**, es decir, en lugar de partir de un dataset real, aprenderemos a generar nuestros propios datos para realizar pruebas con los algoritmos. Para ello, nos basaremos en un ejemplo de *Machine Learning Mastery* (ver Webgrafía al final de este tema).

Con el siguiente código creamos un dataset sintético de 1 000 muestras de datos con 2 características, las 2 con información (ninguna es redundante) y «etiquetadas» (aunque esto se lo ocultaremos a los algoritmos de *clustering*) en 3 clases diferentes, con un clúster por clase.

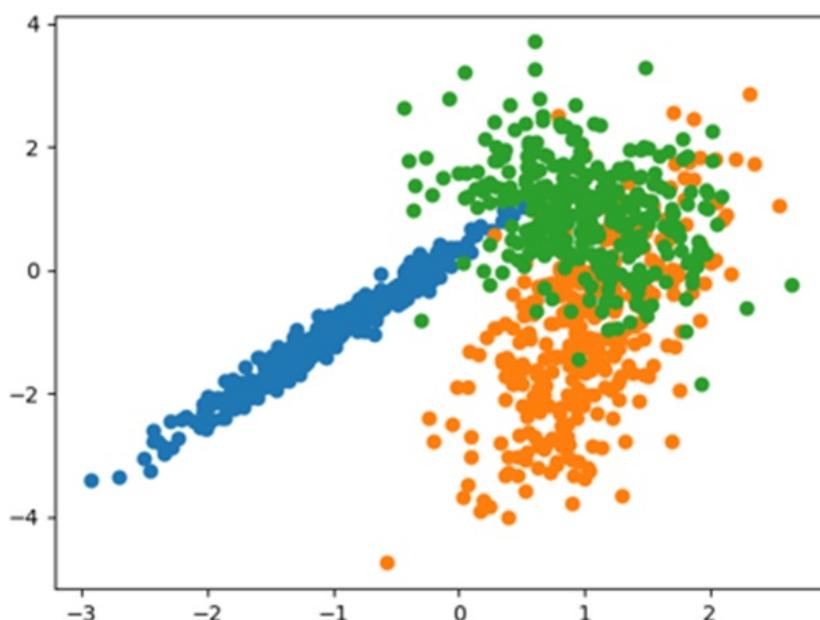
```
# dataset de clasificación sintético
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot

# definimos el dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)

# creamos un diagrama de dispersión para las muestras de cada clase
for class_value in range(3):
    # obtenemos los índices de fila para las muestras dentro de esa clase
    row_ix = where(y == class_value)
    # creamos un diagrama de dispersión con esas muestras
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])

pyplot.show()
```

Veremos en un gráfico similar a este las tres clases diferentes en función de las dos características (en los ejes X e Y), y con un color para cada clase. Vemos cómo cada una está distribuida en único clúster (aunque esto no tiene por qué ser así, si no lo fuera la dificultad del problema sería mayor para un algoritmo no supervisado):



En primer lugar, vamos a probar algunos algoritmos de *clustering* exclusivo, como k-means, mean-shift y DBSCAN.

Probemos el **algoritmo k-means**. En este algoritmo, le hemos de especificar *a priori* el número de clústeres que esperamos que haya. En este caso 3, pero podría ser algo que un supiéramos *a priori*.

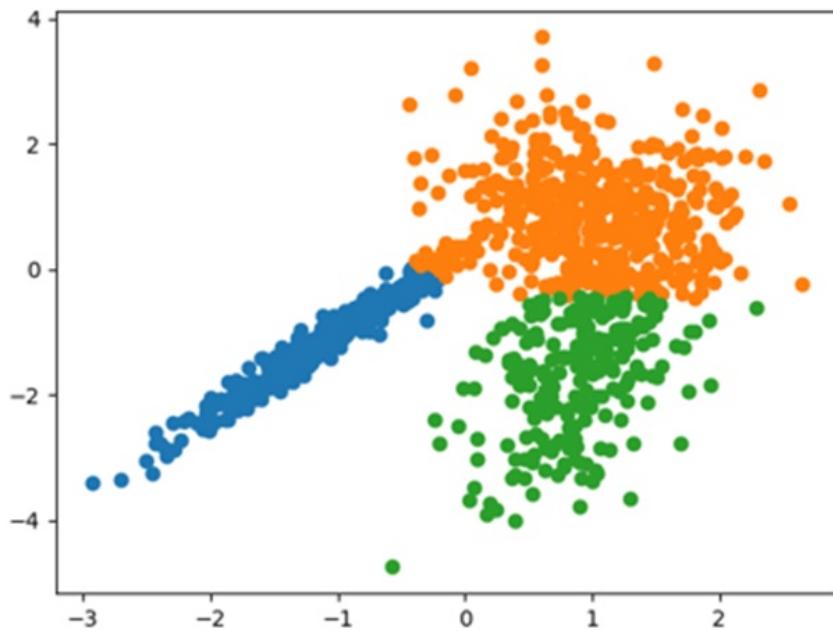
```
# dataset de clasificación sintético
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot

# definimos el dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=
2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_s
tate=4)

# definimos el modelo
model = KMeans(n_clusters=3)
# ajustamos el modelo
model.fit(X)
# asignamos un cluster a cada ejemplo
yhat = model.predict(X)
# obtenemos los clústeres únicos
clusters = unique(yhat)

# creamos un gráfico de dispersión para las muestras de cada clú
ster
for cluster in clusters:
    # obtenemos los índices de fila para las muestras dentro de es
te clúster
    row_ix = where(yhat == cluster)
    # creamos un mapa de dispersión para estas muestras
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# mostramos el gráfico
pyplot.show()
```

Obtendremos la siguiente salida, bastante similar a lo esperado. Podemos realizar pruebas con un número diferente de clústeres esperado (dado que no se conocerían *a priori*, en muchas ocasiones) para ver los resultados que tendríamos.



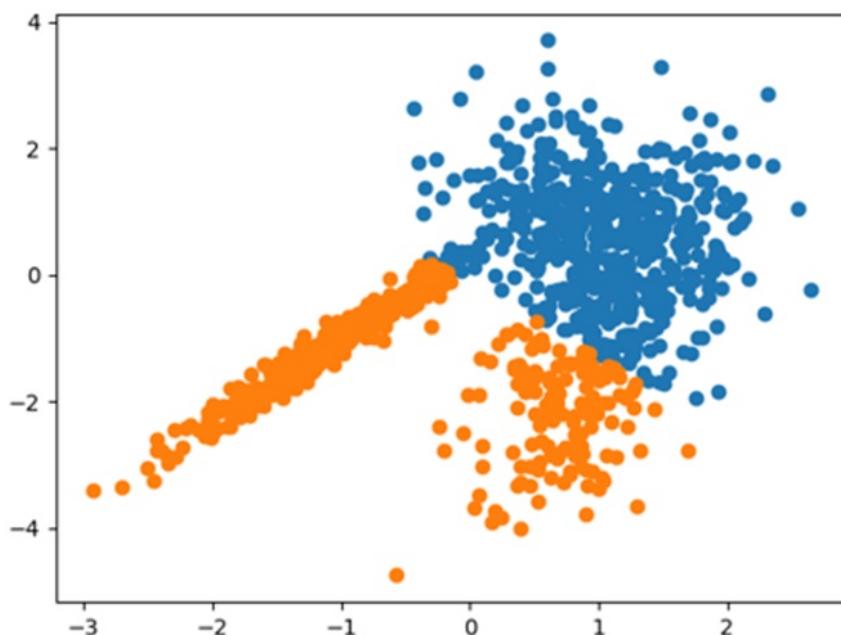
Probemos ahora con ***mean shift***. En este caso, no necesitamos especificar previamente el número de clústeres esperado.

```
# dataset de clasificación sintético
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import MeanShift
from matplotlib import pyplot

# definimos el dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=2,
                           n_informative=2, n_redundant=0, n_clusters_per_class=1,
                           random_state=4)

# definimos el modelo
model = MeanShift()
# ajustamos el modelo y predecimos las clases
yhat = model.fit_predict(X)
# obtenemos los clústeres únicos
clusters = unique(yhat)
```

```
# creamos un gráfico de dispersión para las muestras de cada clúster
for cluster in clusters:
    # obtenemos los índices de fila para las muestras dentro de este clúster
    row_ix = where(yhat == cluster)
    # creamos un mapa de dispersión para estas muestras
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# mostramos el gráfico
pyplot.show()
```



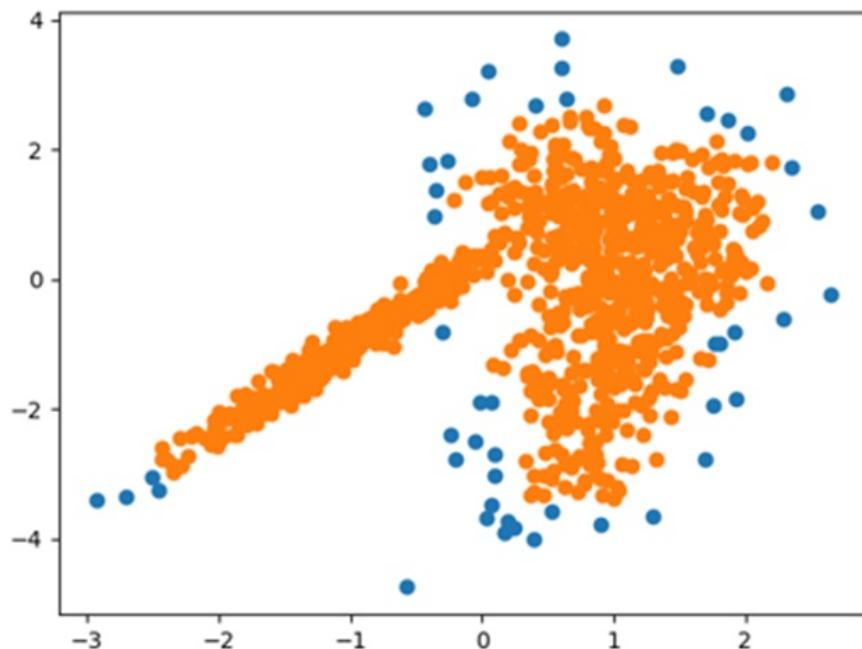
Probemos ahora con el **DBSCAN**, utilizando para los hiperparámetros un valor *épsilon* de 0.3 y un número mínimo de elementos por clúster de 9:

```
# dataset de clasificación sintético
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import DBSCAN
from matplotlib import pyplot

# definimos el dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)

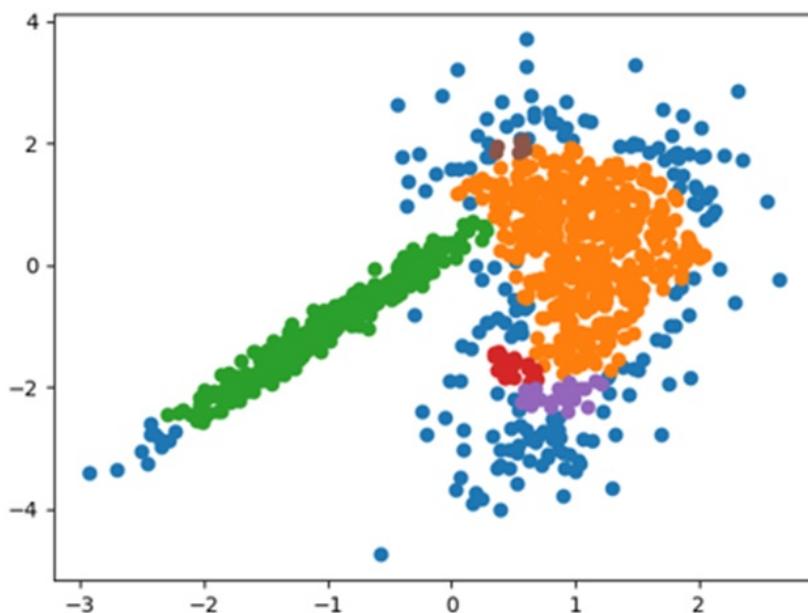
# definimos el modelo
model = DBSCAN(eps=0.3, min_samples=9)
# ajustamos el modelo y predecimos los clústeres
yhat = model.fit_predict(X)
# obtenemos los clústeres únicos
clusters = unique(yhat)

# creamos un gráfico de dispersión para las muestras de cada clúster
for cluster in clusters:
    # obtenemos los índices de fila para las muestras dentro de este clúster
    row_ix = where(yhat == cluster)
    # creamos un mapa de dispersión para estas muestras
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# mostramos el gráfico
pyplot.show()
```



Tal y como podemos ver, con este valor de épsilon el algoritmo no es capaz de diferenciar los tres clústeres, al no haber separación clara entre las fronteras de ellos, y los identifica como uno único clúster, aunque sí nos permite identificar *outliers*.

Probemos con un valor de épsilon más pequeño, 0.2.



Vemos en este caso cómo el algoritmo ha conseguido diferenciar dos clústeres principales, además de otros secundarios.

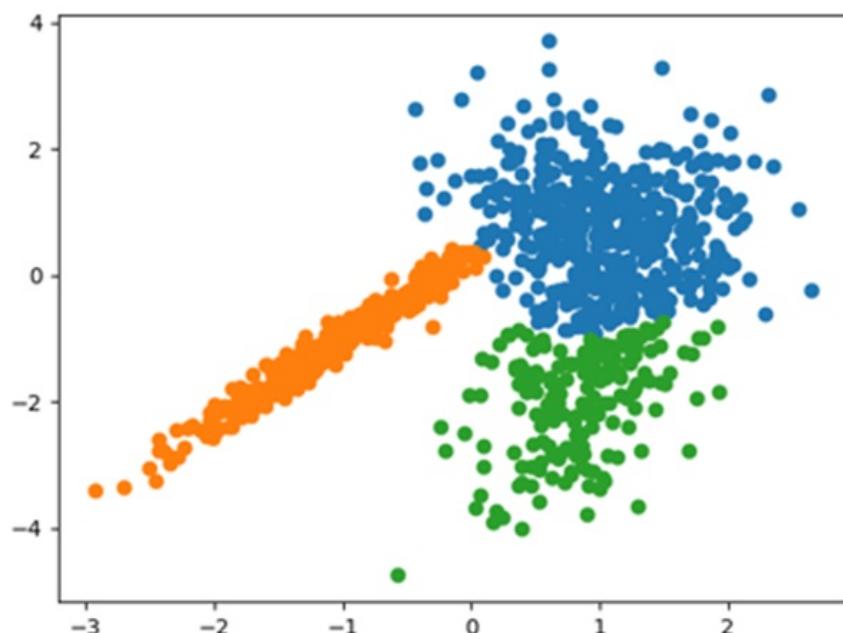
Veamos ahora como ejemplo de *clustering* jerárquico el ***clustering aglomerativo***, eligiendo un número de clústeres esperado de 3:

```
# dataset de clasificación sintético
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import AgglomerativeClustering
from matplotlib import pyplot

# definimos el dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=2,
                           n_informative=2, n_redundant=0, n_clusters_per_class=1,
                           random_state=4)

# definimos el modelo
model = AgglomerativeClustering(n_clusters=3)
# ajustamos el modelo y predecimos las clases
yhat = model.fit_predict(X)
# obtenemos los clústers únicos
clusters = unique(yhat)

# creamos un gráfico de dispersión para las muestras de cada clúster
for cluster in clusters:
    # obtenemos los índices de fila para las muestras dentro de este clúster
    row_ix = where(yhat == cluster)
    # creamos un mapa de dispersión para estas muestras
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# mostramos el gráfico
pyplot.show()
```



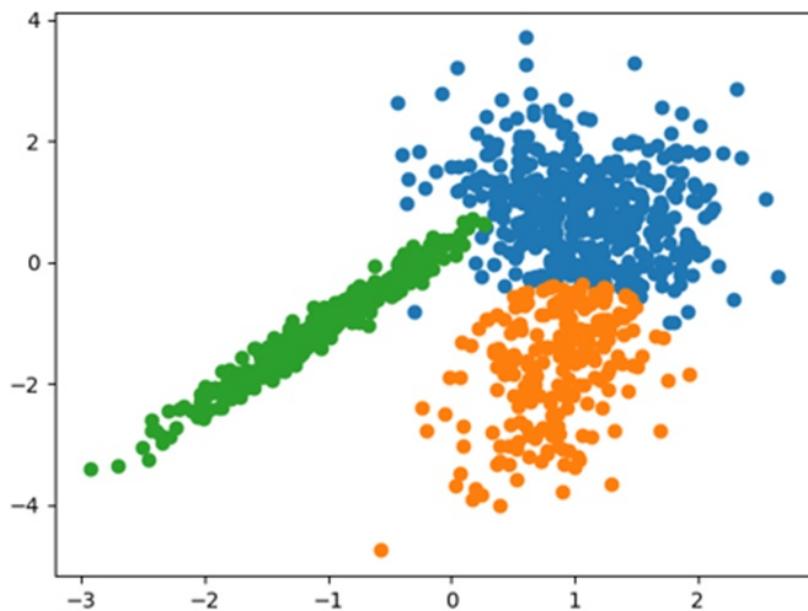
Finalmente, probemos con un **algoritmo probabilista basado en GMM (Gaussian Mixture Model)** eligiendo de antemano que se prevén 3 componentes:

```
# dataset de clasificación sintético
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.mixture import GaussianMixture
from matplotlib import pyplot

# definimos el dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)

# definimos el modelo
model = GaussianMixture(n_components=3)
# ajustamos el modelo
model.fit(X)
# asignamos un clúster a cada ejemplo
yhat = model.predict(X)
# obtenemos los clústers únicos
clusters = unique(yhat)

# creamos un gráfico de dispersión para las muestras de cada clúster
for cluster in clusters:
    # obtenemos los índices de fila para las muestras dentro de este clúster
    row_ix = where(yhat == cluster)
    # creamos un mapa de dispersión para estas muestras
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# mostramos el gráfico
pyplot.show()
```



7.8. Referencias bibliográficas

Abbasimehr, H. & Shabani, M. (2019). A new methodology for customer behavior analysis using time series clustering: A case study on a bank's customers. *Kybernetes*. Disponible en <https://doi.org/10.1108/K-09-2018-0506>

Chiang, W.Y. (2018). Applying data mining for online CRM marketing strategy: An empirical case of coffee shop industry in Taiwan. *British Food Journal*, 120(3), 665-675. Disponible en <https://doi.org/10.1108/BFJ-02-2017-0075>

G. Sreenivasulu, Raju, S. V. & Rao, N. S. (2017). Review of Clustering Techniques. En S. C. Satapathy, V. Bhateja & A. Joshi (Eds.), *Proceedings of the International Conference on Data Engineering and Communication Technology* (pp. 523-535). Springer. Disponible en https://doi.org/10.1007/978-981-10-1675-2_52

Holý, V., Sokol, O. & Černý, M. (2017). Clustering retail products based on customer behaviour. *Applied Soft Computing*, 60, 752-762. Disponible en <https://doi.org/10.1016/j.asoc.2017.02.004>

Kaufman, L. & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis*. Wiley.

Kumar, M. S., Soundarya, V., Kavitha, S., Keerthika, E. S. & Aswini, E. (2019). *Credit Card Fraud Detection Using Random Forest Algorithm*. 2019 3rd International Conference on Computing and Communications Technologies (ICCCT), pp. 149-153. Disponible en <https://doi.org/10.1109/ICCCT2.2019.8824930>

Nilashi, M., Ibrahim, O. bin, Ahmadi, H. & Shahmoradi, L. (2017a). An analytical method for diseases prediction using machine learning techniques. *Computers & Chemical Engineering*, 106, 212-223. Disponible en <https://doi.org/10.1016/j.compchemeng.2017.06.011>

Nilashi, M., Ibrahim, O., Ahmadi, H. & Shahmoradi, L. (2017b). A knowledge-based system for breast cancer classification using fuzzy logic method. *Telematics and Informatics*, 34(4), 133-144. Disponible en <https://doi.org/10.1016/j.tele.2017.01.007>

Pacheco, W. D. N. & López, F. R. J. (2019). *Tomato classification according to organoleptic maturity (coloration) using machine learning algorithms K-NN, MLP, and K-Means Clustering*. 2019 XXII Symposium on Image, Signal Processing and Artificial Vision (STSIVA), pp. 1-5. Disponible en <https://doi.org/10.1109/STSIVA.2019.8730232>

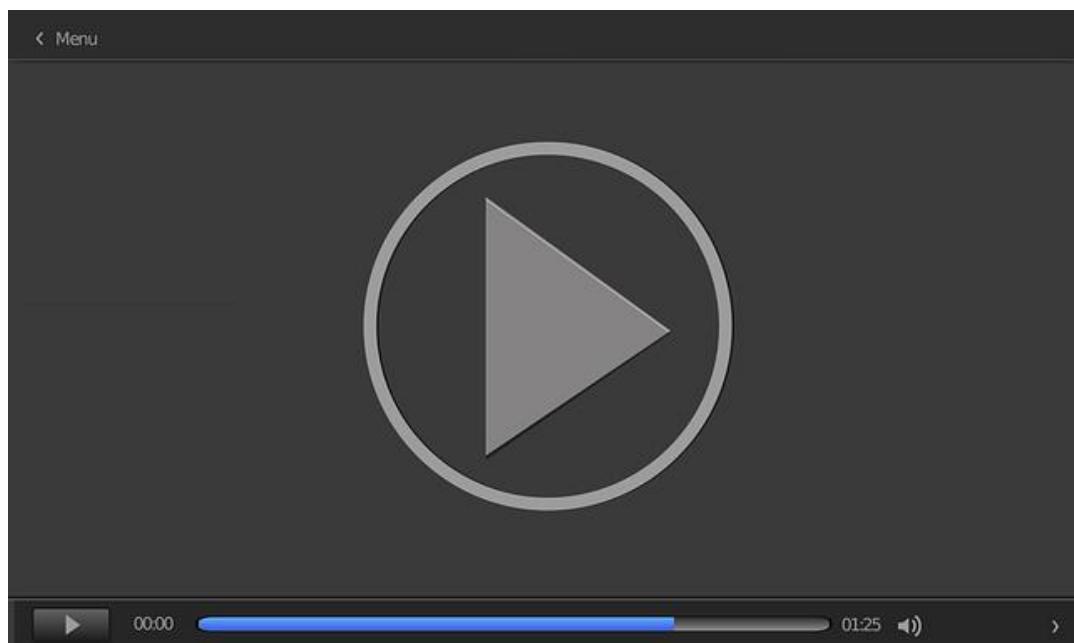
Subudhi, S. & Panigrahi, S. (2017). Use of optimized Fuzzy C-Means clustering and supervised classifiers for automobile insurance fraud detection. *Journal of King Saud University - Computer and Information Sciences*. Disponible en <https://doi.org/10.1016/j.jksuci.2017.09.010>

Xu, R. & Wunsch, D. C. (2009). *Clustering*. Wiley: IEEE Press.

Zakrzewska, D. (2009). Cluster Analysis in Personalized E-Learning Systems. En N. T. Nguyen & E. Szczerbicki (Eds.), *Intelligent Systems for Knowledge Management* (pp. 229-250). Springer. Disponible en https://doi.org/10.1007/978-3-642-04170-9_10

Algoritmos de clustering en Weka

Esta lección magistral consiste en un tutorial sobre los algoritmos de clustering disponibles en Weka. Se mostrarán algunos ejemplos de ejecución, parámetros configurables e interpretación de las salidas obtenidas.



07. Algoritmos de clustering en Weka

Accede al vídeo:

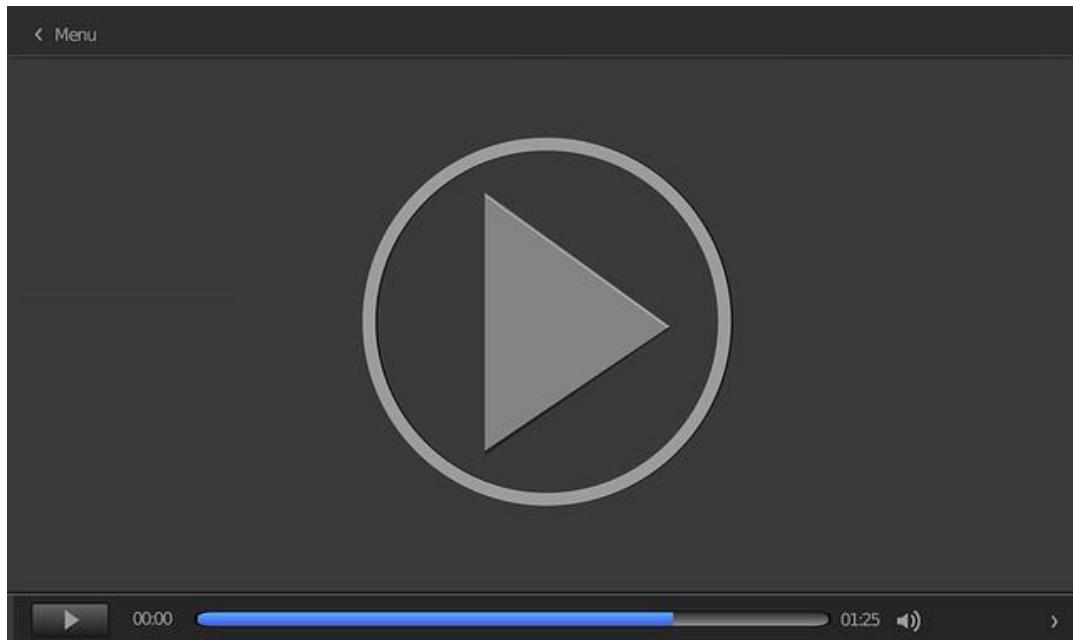
<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=b4fd0dcd-6d38-4b84-a285-aff800f9e0af>

Demostración de k-means mediante un ejemplo de cartas de póker

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=zHbxbb2ye3E>

Este vídeo muestra de una manera muy didáctica el funcionamiento del algoritmo k-means, mediante un ejemplo en el que se clasifican cartas de póker.



Accede al vídeo:

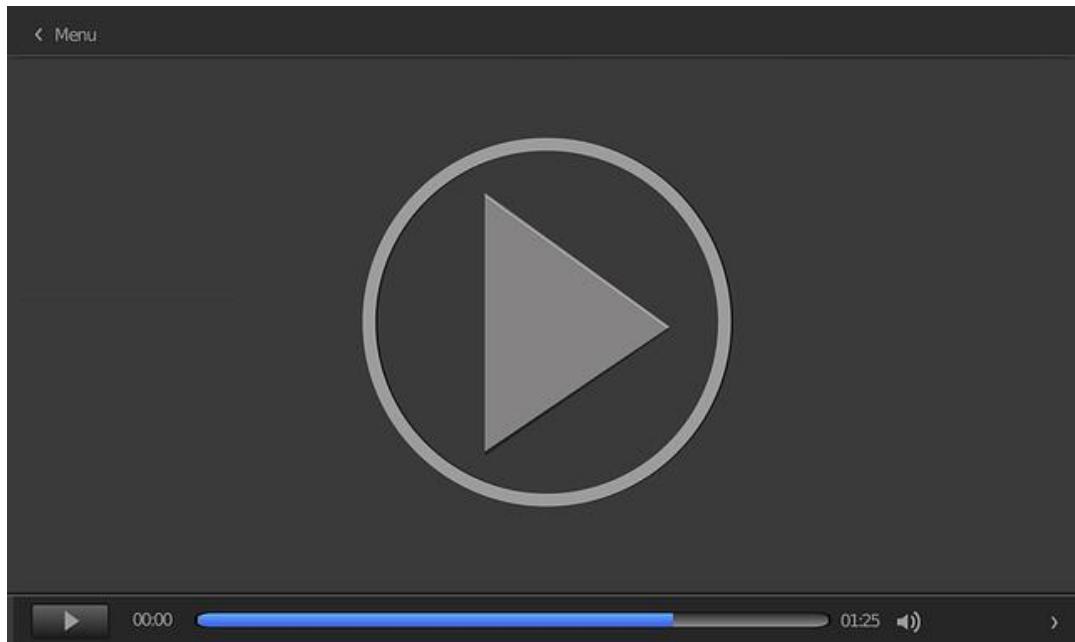
<https://www.youtube.com/embed/zHbxbb2ye3E>

Agrupamientos solapados, clustering difuso

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=9yNtJsFxDQI>

Presentación que explica el algoritmo Fuzzy C-means, así como el algoritmo k-means, incluyendo ejemplos detallados de las distintas iteraciones de los algoritmos. Finaliza con una comparación de ambas técnicas.



Accede al vídeo:

<https://www.youtube.com/embed/9yNtJsFxDQI>

Métodos jerárquicos de análisis clúster

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:

<http://www.ugr.es/~gallardo/pdf/cluster-3.pdf>

Capítulo de la asignatura Ampliación de Análisis de Datos Multivariantes del Departamento de Estadística e Investigación Operativa de la Universidad de Granada, relativo a los algoritmos de clustering jerárquico. Describe diferentes métodos jerárquicos aglomerativos que emplean diversas estrategias para medir la similitud entre objetos con ejemplos ilustrativos que facilitan la comprensión.

Demostraciones del algoritmo EM

Accede a la wiki desde el aula virtual o a través de la siguiente dirección web:
http://wiki.stat.ucla.edu/socr/index.php/SOCR_EduMaterials_Activities_2D_PointSegmentation_EM_Mixture

Wiki del organismo SOCR (Statistics Online Computational Resource) que propone actividades para experimentar con el algoritmo EM con el fin de obtener clústeres a partir de puntos en espacios de 1D, 2D y 3D, haciendo uso de la herramienta SOCR, que se puede ejecutar a través de esta misma página.

Scikit-learn

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<https://scikit-learn.org/stable/modules/clustering.html>

La sección 2.3 sobre algoritmos de clustering en la documentación de scikit-learn es excepcional. Incluye muestras gráficas de las formas de los resultados de los clústeres para diez de los algoritmos de clustering más conocidos, códigos de ejemplo, bibliografía donde se detalla cada algoritmo, así como tablas comparativas que nos recomiendan qué algoritmo conviene emplear en cada caso.

Machine Learning Mastery

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<https://machinelearningmastery.com/>

Portal web creado por Jason Brownlee enfocado al aprendizaje de técnicas de machine learning. Ofrece sus libros en formato eBook (de pago) con diferentes niveles de precios en función de los contenidos incluidos. No obstante, incluye entradas en su blog (sin coste alguno) con ejemplos prácticos paso a paso muy interesantes de aplicación de técnicas de machine learning aplicadas mediante Python.

1. Indica cuáles de las siguientes afirmaciones son correctas:

 - A. El clustering permite agrupar objetos similares entre sí.
 - B. El clustering es un método de aprendizaje supervisado.
 - C. El clustering puede resultar útil como etapa previa a la aplicación de un método de aprendizaje supervisado.
 - D. El clustering da lugar a árboles de decisión.
2. Indica cuál de las siguientes afirmaciones es correcta:

 - A. Diferentes algoritmos de clustering dan lugar a los mismos agrupamientos finales.
 - B. Los algoritmos jerárquicos aglomerativos generan clústeres pequeños que iterativamente van agrupando entre sí.
 - C. Los agrupamientos solapados se obtienen aplicando algoritmos de clustering jerárquicos.
 - D. Los algoritmos de clustering no permiten detectar datos anómalos.
3. Si se pretende generar agrupamientos exclusivos se ha de aplicar:

 - A. Algoritmo Fuzzy C-means.
 - B. Algoritmo k-means.
 - C. Algoritmo EM.
 - D. Ninguno de los anteriores.
4. Si se pretende crear agrupamientos con formas irregulares se ha de aplicar:

 - A. Algoritmo k-means.
 - B. Algoritmos basados en densidad.
 - C. Algoritmo Fuzzy C-means.
 - D. Ninguno de los anteriores.

5. Si se pretende modelar los clústeres mediante una función probabilista se ha de aplicar:

- A. Algoritmo Fuzzy C-means.
- B. Algoritmo k-means.
- C. Algoritmo EM.
- D. Ninguno de los anteriores.

6. Si se mide la similitud entre dos clústeres mediante la medida de enlace completo:

- A. Se tiene en cuenta la similitud entre los dos puntos más cercanos de ambos clústeres.
- B. Se tiene en cuenta la similitud entre los dos puntos más lejanos de ambos clústeres.
- C. Se tiene en cuenta la distancia promedio que existe entre todos los puntos de ambos clústeres.
- D. Se tiene en cuenta la distancia entre los centroides de ambos clústeres.

7. Indica cuáles de las siguientes afirmaciones, respecto del algoritmo k-means, son correctas:

- A. El algoritmo k-means asigna los objetos a los clústeres en función de su cercanía al centroide de cada clúster.
- B. En cada iteración el algoritmo k-means mantiene fijos los centroides.
- C. Es un algoritmo basado en densidad.
- D. En cada iteración el algoritmo recalcula los centroides.

8. Indica cuáles de las siguientes afirmaciones, respecto a los algoritmos de clústering jerárquicos, son correctas:

- A. Se utiliza la medida de utilidad de la categoría para realizar particiones.
- B. En un algoritmo divisorio inicialmente a cada clúster se le asigna un objeto.
- C. Utilizan una matriz de similitud para llevar a cabo la decisión de agrupar clústeres.
- D. Se puede utilizar la medida de enlace promedio para calcular las distancias entre clústeres.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto al algoritmo EM:

- A. Es un algoritmo basado en densidades.
- B. Tiene como base el modelo estadístico denominado mezclas finitas.
- C. El objetivo es conocer los parámetros de una función probabilista general que modela los clústeres.
- D. En la fase de esperanza se calculan las probabilidades de pertenencia de las instancias a los clústeres.

10. Indica cuáles de las siguientes afirmaciones son correctas respecto al algoritmo Fuzzy C-means:

- A. Una instancia puede pertenecer a más de un clúster si se aplica el algoritmo Fuzzy C-means.
- B. Las variables de entrada son conjuntos difusos.
- C. Los clústeres se modelan como conjuntos difusos.
- D. Permite obtener clústeres jerárquicos.

Técnicas de Inteligencia Artificial

Tema 8. Sistemas de recomendación

Índice

Esquema

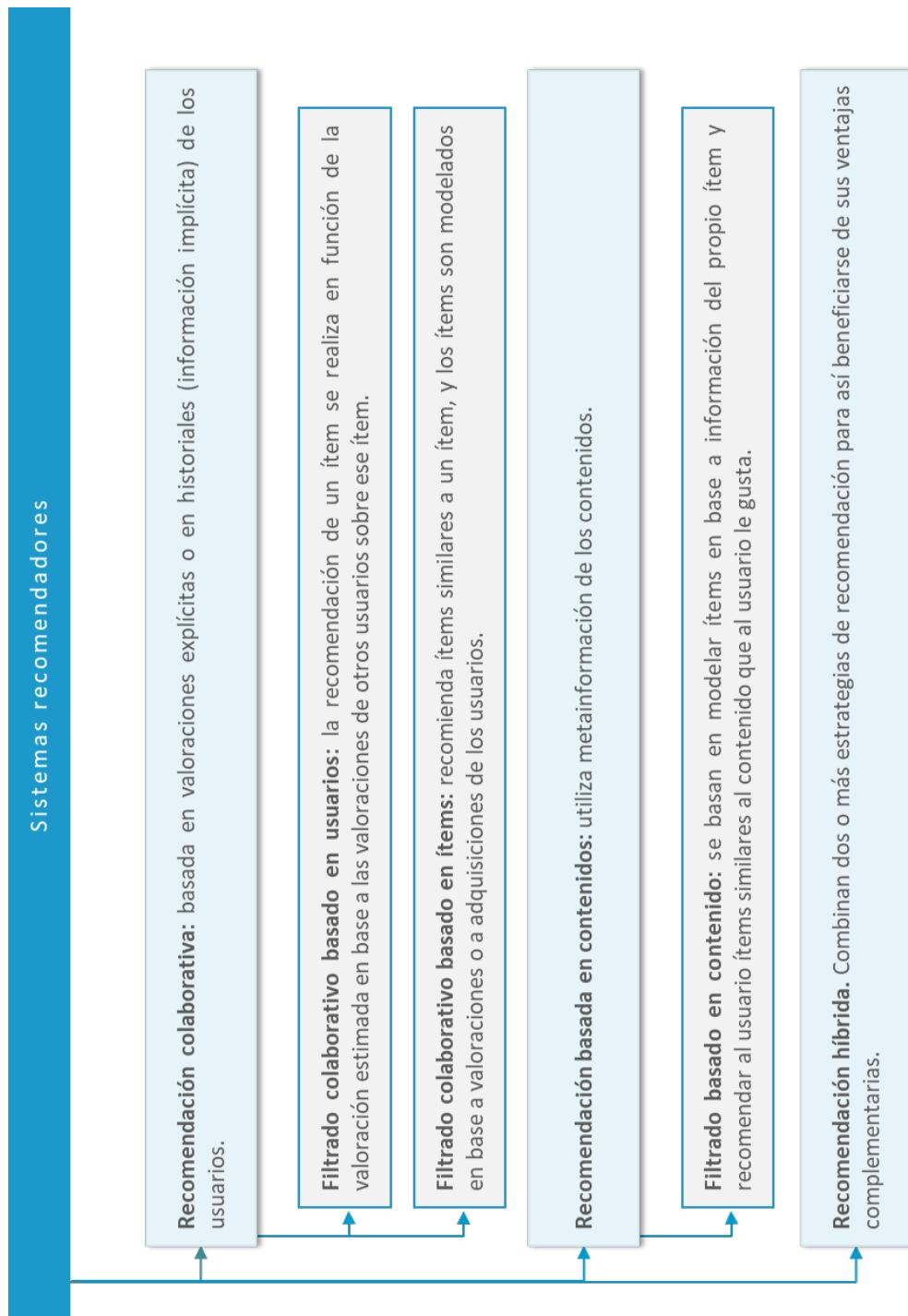
Ideas clave

- 8.1. ¿Cómo estudiar este tema?
- 8.2. Introducción. Tipos de recomendadores y aplicaciones
- 8.3. Recomendación colaborativa. Filtrado colaborativo basado en usuarios. Filtrado colaborativo basado en ítems
- 8.4. Recomendación basada en contenido. Representación del contenido y similitud entre elementos
- 8.5. Sistemas de recomendación híbridos
- 8.6. Ejemplos de implementación
- 8.7. Referencias bibliográficas

A fondo

- Introducción a los sistemas recomendadores
- Taxonomía de agentes recomendadores
- Sistemas recomendadores basados en contenidos
- Recomendación de productos basada en filtrado colaborativo
- Medidas de asociación
- Sistemas de recomendación

Test



8.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral, revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan.

Al finalizar el estudio de este tema serás capaz de:

- ▶ Describir los tres tipos de sistemas de recomendación más significativos: basados en filtrado colaborativo, basados en contenido e híbridos.
- ▶ Aplicar técnicas de filtrado e híbridas para la resolución de problemas de recomendación.
- ▶ Identificar aplicaciones prácticas de recomendadores.

8.2. Introducción. Tipos de recomendadores y aplicaciones

La gran cantidad de contenidos disponibles en Internet en la actualidad supone que los usuarios se encuentren frecuentemente en situaciones en las que han de elegir entre un gran número de opciones y llegan a sentirse abrumados o perdidos. Por lo tanto, cualquier técnica que permita **clasificar, ordenar o filtrar la información disponible** es de gran interés en la actualidad. Estas técnicas se basan, por ejemplo, en modelar contenidos, así como preferencias y comportamientos de usuarios, creando estereotipos, y filtrando la información en función de los estereotipos (Alharthi *et al.*, 2018; Bobadilla *et al.*, 2013; Joshi, 2020; Kunaver y Požrl, 2017).

Montaner *et al.* (2003) describen los principales componentes y procesos de los sistemas recomendadores como los de la Figura 1.

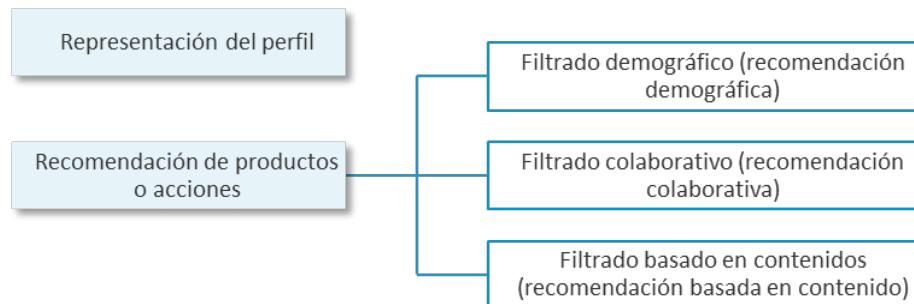


Figura 1. Principales componentes y procesos de los sistemas recomendadores según Montaner *et al.* (2003).

- ▶ **Representación del perfil.** El primer paso al crear un sistema recomendador es diseñar cómo se va a representar el perfil del usuario, esto es, qué información de los usuarios se va a tener en cuenta para las recomendaciones y, por tanto, hay que capturar. Cuando un usuario interactúa con el navegador, si se captura la

información adecuada (por ejemplo, un historial de compras o de navegación) y se interpreta correctamente, se puede conseguir un gran éxito en las recomendaciones, ya que a partir de esta información se pueden deducir preferencias, gustos, hábitos y comportamientos de los usuarios. Habrá que utilizar, por tanto, técnicas para generar inicialmente los perfiles de los usuarios, y posteriormente actualizarlos y mantenerlos.

- ▶ **Recomendación de productos o acciones** a los usuarios mediante una técnica de filtrado que definirá el tipo de recomendación:
 - **Filtrado demográfico (recomendación demográfica)**: utiliza descripciones de personas para aprender la relación entre un ítem de información y el tipo de personas que les gusta ese ítem. Este tipo de técnicas requieren recopilación de datos personales sobre el usuario para clasificarlo demográficamente en un estereotipo.
 - **Filtrado colaborativo (recomendación colaborativa)**: realiza recomendaciones usualmente en base a la retroalimentación o el *feedback* que los conjuntos de usuarios dan explícitamente sobre un ítem. También se pueden basar en información capturada implícitamente como, por ejemplo, el historial de compras de los usuarios.
 - **Filtrado basado en contenidos (recomendación basada en contenido)**: utiliza metainformación de los contenidos de los ítems para aprender la relación entre un usuario y esa metainformación.
 - **Filtrado basado en el conocimiento (KBF – Knowledge-Based Filtering)**: utiliza el conocimiento que se tiene sobre los usuarios (y sus preferencias) y los ítems para determinar qué elementos cumplen con los requisitos de los usuarios y generar recomendaciones en consecuencia. Existen tipos especiales de KBF, como aquellos basados en restricciones para recomendar elementos complejos que rara vez se compran (por ejemplo, una vivienda y un vehículo) y tienen en cuenta restricciones importantes para el usuario, como podría ser el precio (normalmente disponen de pocos datos de interacción entre el sistema y el usuario) (Çano & Morisio, 2017).

- **Filtrado basado en la utilidad:** al igual que los sistemas KBF, este tipo de filtrado no trata de generalizar a largo plazo las recomendaciones sobre sus usuarios, sino basar estas en una evaluación de la correspondencia entre las necesidades del usuario y el conjunto de ítems disponibles. Los recomendadores basados en la utilidad hacen sugerencias basadas en un cálculo de la utilidad de cada ítem para el usuario (Burke, 2002).

Por lo tanto, se ha de escoger una técnica de filtrado y una técnica de emparejamiento, bien entre estereotipo y contenidos (filtrado demográfico), bien entre el perfil de usuario y el contenido (filtrado basado en contenido), bien entre perfiles de usuarios (filtrado colaborativo) o bien mediante filtrado basado en el conocimiento. No obstante, existen también soluciones en las que se combinan varias técnicas (sistemas híbridos), para así intentar extraer el máximo beneficio de cada una de ellas en diferentes pasos del proceso de recomendación (Burke, 2002; Jannach, 2011; Ricci *et al.*, 2011).

Para la **representación del perfil** se pueden utilizar técnicas de inteligencia artificial estudiadas en la asignatura, tales como árboles de decisión, reglas inducidas, redes neuronales o redes bayesianas (Lops *et al.*, 2011; Najafabadi *et al.*, 2019). Las reglas de asociación, por ejemplo, tal y como se comentó en el tema correspondiente, se utilizan precisamente en estos sistemas para la recomendación de productos a partir de otros productos que se han adquirido.

En aplicaciones de comercio electrónico se suele utilizar un historial de las compras y las valoraciones que los usuarios dan sobre los productos comprados para generar reglas de asociación. Este tipo de información es también utilizada cuando se utilizan técnicas de filtrado. Algunos sistemas mantienen matrices de valoraciones usuario-ítem como representación del perfil, como típicamente hacen los sistemas de filtrado colaborativo (Kunaver & Požrl, 2017).



Además, los sistemas recomendadores utilizan a veces técnicas de *clustering* para generar grupos y agregar el perfil del usuario que se está analizando a un determinado clúster de perfiles similares (Singh & Solanki, 2019).

Inicialmente, a la hora de generar los perfiles, algunos sistemas simplemente los generan vacíos y, según el usuario interacciona, los van completando; mientras que otros sistemas, en base a la información disponible del usuario (por ejemplo, la información que el usuario aporta al darse de alta en un sistema), tratan de clasificarlo en un determinado estereotipo (como típicamente realizan los sistemas de filtrado demográfico).

Dado que los intereses de los usuarios cambian y, según transcurre el tiempo e interactúan con los sistemas, se puede conseguir más información de los mismos, por lo que se han de actualizar los perfiles. Algunos de estos sistemas solicitan retroalimentación al usuario de manera explícita y suelen dar muy buenos resultados siempre que los usuarios se animen a realizar valoraciones de los productos. Otros sistemas capturan las interacciones de los usuarios y actualizan su perfil en base a estas.

El tipo de información que implícitamente se puede obtener de estas interacciones es, por ejemplo, el historial de compras, el historial de navegación, el tiempo que un

usuario está en cada página, etc. Hay sistemas que utilizan tanto información explícita como implícita.

A la hora de utilizar esta información para tomar decisiones, dado que las últimas interacciones pueden reflejar en mayor medida los intereses del usuario actuales que las interacciones antiguas, hay sistemas que ponderan estas interacciones.

El problema de los sistemas de filtrado demográfico es que suelen dar recomendaciones muy generales, no siendo suficientemente personalizadas. Además, no suelen adaptarse a los cambios en los intereses de los usuarios. Previamente a entrar en el detalle de cada tipo de técnica se enumeran algunos ejemplos de aplicación de los sistemas recomendadores:

- ▶ **Comercio electrónico.** Amazon es un claro y popular ejemplo de comercio electrónico que recomienda a los usuarios nuevos productos a adquirir. Cuando un usuario visita la página de un producto, Amazon le recomienda productos adquiridos por otros usuarios que han comprado el producto que el usuario está visitando (Linden *et al.*, 2003).
- ▶ **Webs de libros, música o vídeos:** recomiendan a los usuarios música o videos en base a la similitud con contenidos ya visitados o en base a la similitud con otros usuarios que comparten gustos musicales (Alharthi *et al.*, 2018; Augstein *et al.*, 2019; Sánchez-Corcuera *et al.*, 2020; Singhal *et al.*, 2017).
- ▶ **Noticias:** sistemas que recomiendan noticias de interés a un usuario en base a sus intereses, historial de noticias previamente visitadas, etc., (Karimi *et al.*, 2018).

Por otro lado, tal y como se adelantó en párrafos anteriores, los sistemas de recomendación hacen uso de diferentes técnicas de inteligencia artificial, como pueden ser los árboles de decisión (Linda & Bharadwaj, 2018), las reglas (Osadchiy

et al., 2019; Yao *et al.*, 2019), el clústering (Singh & Solanki, 2019), redes neuronales y *deep learning* (Mu, 2018) u otras técnicas de minería de datos (Najafabadi *et al.*, 2019).

En la actualidad, el gran auge de las redes sociales y de los sistemas colaborativos en red hace que los datos de conectividad entre usuarios resulten una base verdaderamente útil para llevar a cabo un eficaz filtrado de contenidos y mejora de la adaptabilidad de estos al usuario (Campana y Delmastro, 2017; Eirinaki *et al.*, 2018).

8.3. Recomendación colaborativa. Filtrado colaborativo basado en usuarios. Filtrado colaborativo basado en ítems

Los sistemas de **recomendación colaborativa** recomiendan los ítems a un usuario basándose en las preferencias o historial de otros usuarios.

Estos sistemas utilizan técnicas de filtrado colaborativo y se pueden distinguir diversos tipos de técnicas como, por ejemplo:

- ▶ **Filtrado colaborativo basados en usuarios:** estiman la valoración que un usuario daría a un determinado contenido para decidir su recomendación. La similitud entre usuarios la miden en base a las valoraciones de los ítems realizadas previamente, y las puntuaciones de ítems no visitados por un usuario se predicen en función de la valoración que otros usuarios, con un alto grado de similitud al usuario en cuestión, han otorgado al citado contenido.
- ▶ **Filtrado colaborativo basado en ítems:** calculan la similitud entre ítems. Estos ítems son descritos en base a datos de los usuarios, por ejemplo, las valoraciones dadas por los diferentes usuarios al ítem, o las adquisiciones realizadas o no realizadas por los distintos usuarios. Por eso, sigue considerándose recomendación del tipo colaborativa. Al usuario se le recomiendan los ítems más similares a un ítem adquirido o a un ítem que se está visitando.

De acuerdo a Montaner *et al.* (2003), se suelen aplicar las siguientes medidas para calcular la similitud entre usuarios, también utilizadas para medir similitud entre ítems:

- ▶ **Similitud coseno** (*Cosine similarity*): mide la similitud entre dos vectores, calculando el coseno del ángulo entre ambos. Si vale 1, ambos vectores son similares totalmente y, si vale 0, son totalmente distintos. El vector estará formado de elementos del perfil del usuario o atributos del ítem.
- ▶ **Medidas de correlación**: por ejemplo, los coeficientes de correlación de Pearson o de Spearman se pueden utilizar para comparar usuarios a partir de sus valoraciones numéricas.

Para llevar a cabo una recomendación de los contenidos a cada usuario es importante conocer en todo momento la valoración que este daría a cada contenido. En algunas ocasiones dicha valoración puede estar disponible en los datos almacenados en el sistema, pero en otros casos se ha de predecir dicha valoración teniendo en cuenta el comportamiento del resto de los usuarios del sistema.

Uno de los algoritmos de predicción más empleado y muy simple en los sistemas de **Filtrado Colaborativo basado en usuario** es **Slope One**. Dicho algoritmo cuenta con una formulación matemática muy sencilla, pero a la vez muy eficiente, tal y como lo definen Lemire *et al.* (2005).

La idea que subyace de la aplicación de este algoritmo, en su versión más simple, es que dadas las valoraciones de dos ítems correspondientes al usuario A y la valoración de uno de estos ítems por parte del usuario B se pueda predecir la valoración correspondiente al ítem que falta de valorar por parte del usuario B. Este ejemplo se puede escalar a un mayor número de usuarios y existen distintas

versiones del algoritmo para perfeccionar los resultados (como pueden ser *Weighted Slope One* y *Bi-polar Slope One*).

Con un ejemplo muy sencillo se ilustra el algoritmo Slope One. Este ejemplo se basa en el expuesto por los creadores del algoritmo (Lemire *et al*, 2005). El algoritmo **Slope One** se basa en la siguiente función de predicción de la valoración de un ítem:

$$f(x) = x + b$$

Siendo b la diferencia media de valoraciones entre ítems.

	Ítem 1	Ítem 2
María	4/10	---
Pablo	2/10	5/10
Luis	3/10	7/10
Alicia	6/10	8/10

Tabla 1. Valoraciones de cuatro usuarios para los dos artículos del ejemplo del algoritmo Slope One.

Se tiene un comercio en línea en el que los ítems reciben una valoración unidimensional, tal y como se muestra en la Tabla 1. El usuario María no ha adquirido el ítem 2 y el sistema recomendador ha de decidir si recomendar ese ítem a María o no. Como sí existe la valoración de María al ítem 1, para predecir su valoración desconocida al ítem 2, el sistema se va a basar en la diferencia media entre la valoración dada al ítem 2 y la valoración dada al ítem 1 por otros usuarios. Esta media es 3 luego el sistema predice que María valorará el ítem 2 con un 7/10.

El algoritmo **Weighted Slope One**, como su propio nombre indica, utiliza pesos para tener en cuenta el hecho de que la información que se obtiene a partir de aquellos ítems que han recibido más valoraciones es más fiable.

El algoritmo *Slope One* se basa en tener valoraciones sobre los ítems. Si no se tienen estas valoraciones se puede tener en cuenta otra información como, por ejemplo, si un ítem ha sido adquirido o no. Por tanto, la información sobre cada ítem en este caso es binaria. Se describe por ejemplo el algoritmo de **filtrado colaborativo ítem-a-ítem** utilizado por Amazon (Linden *et al.*, 2003), que se puede considerar **filtrado colaborativo basado en ítem**.

El algoritmo de **filtrado colaborativo ítem-a-ítem** se centra en encontrar ítems similares y, para ello, construye una tabla de ítems similares que incluye ítems que los usuarios tienden a comprar conjuntamente. La similitud la calcula con la medida de similitud coseno y la tabla se crea en base al algoritmo mostrado en la Figura 2.

Para cada ítem I1

 Para cada cliente C que compró I1

 Por cada ítem I2 comprado por el cliente C

 Registra que el cliente C compró I1 e I2

 Por cada ítem I2

 Calcula la similitud entre I1 e I2

Figura 2. Algoritmo de filtrado colaborativo ítem-a-ítem

Por ejemplo, en la Tabla 2 se muestra una tabla con los datos de compras de 3 usuarios para 3 ítems. Un valor 0 significa que el producto no ha sido adquirido y un valor 1 significa que sí ha sido adquirido.

	Ítem 1	Ítem 2	Ítem 3
María	0	1	0
Pablo	0	1	1
Luis	1	0	1

Tabla 2. Compras realizadas de los tres artículos por parte de los tres usuarios del ejemplo del algoritmo filtrado colaborativo basado en ítem.

La medida de similitud del coseno tiene la siguiente expresión:

$$\text{similitud} = \cos\left(\overrightarrow{A}, \overrightarrow{B}\right) = \frac{\overrightarrow{A} \bullet \overrightarrow{B}}{\|\overrightarrow{A}\| \|\overrightarrow{B}\|} = \frac{\sum_{i=1}^n A_i \bullet B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \bullet \sqrt{\sum_{i=1}^n (B_i)^2}}$$

Por tanto, para el ejemplo de la tabla 2, la similitud entre cada dos ítems en función de las compras realizadas por los diferentes usuarios de esos ítems se calcula como:

$$\text{similitud}(\text{Item 1}, \text{Item 2}) = \frac{(0,0,1) \bullet (0,1,1)}{\|(0,0,1)\| \bullet \|(0,1,1)\|} = 0$$

$$\text{similitud}(\text{Item 1}, \text{Item 3}) = \frac{(0,0,1) \bullet (1,1,0)}{\|(0,0,1)\| \bullet \|(1,1,0)\|} = \frac{1}{\sqrt{2}} = 0.5$$

$$\text{similitud}(\text{Item 2}, \text{Item 3}) = \frac{(1,1,0) \bullet (0,1,1)}{\|(1,1,0)\| \bullet \|(0,1,1)\|} = \frac{1}{2} = 0.5$$

En la Tabla 3 se muestra la matriz de similitud resultante.

	Ítem 1	Item 2	Ítem 3
Item 1	1	0	0.71
Item 2	0	1	0.5
Item 3	0.71	0.5	1

Tabla 3. Matriz Similitud de los tres artículos considerados en el ejemplo del algoritmo de filtrado colaborativo basado en ítem.

Por lo tanto, un usuario que visita la página del ítem 1, recibirá una recomendación del ítem 3, que es el más similar según la matriz de similitud. Un usuario que visite el ítem 2, recibirá una recomendación del ítem 3. Por último, un usuario que visite el ítem 3, recibirá una recomendación del ítem 1. Los ítems 1 y 2 guardan relación con el ítem 3, pero no entre sí. Es decir, o no guardan relación entre sí o son mutuamente excluyentes (si un usuario ha comprado un ítem 1 no comprará el ítem 2, y viceversa).

Los sistemas de filtrado colaborativo basados en usuario presentan el problema de que puede haber ítems sin tener valoraciones, con lo cual no se tiene información a priori para recomendarlos.

Para resolver este problema se podría combinar técnicas de filtrado colaborativo basado en usuario y basado en ítem. Se podría utilizar alguna técnica de recomendación basada en ítem hasta que el ítem es valorado por el necesario número de usuarios. Por otra parte, en aquellos sistemas con gran oferta de contenidos puede suceder que cada contenido presente pocas valoraciones y, por tanto, este tipo de sistemas en general requieren de un gran número de usuarios para dar buenos resultados.

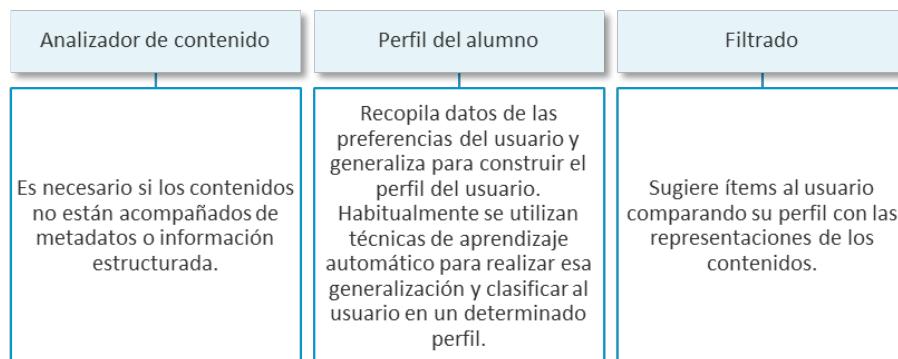
8.4. Recomendación basada en contenido. Representación del contenido y similitud entre elementos

Los sistemas de recomendación basada en contenidos recomiendan a los usuarios ítems que son similares a otros ítems que al usuario le han gustado previamente. Muchos de estos sistemas recomiendan ítems que contienen cierta información textual estructurada acompañando a los contenidos (metainformación o metadatos).

Esto permite que, por ejemplo, si se conoce que a un usuario le gusta la música rock y que escucha muchos grupos de los años 70, se le podrán recomendar ítems del mismo estilo musical, de un grupo que el usuario ya ha escuchado previamente o de grupos de rock de los años 70, etc. Estos sistemas pueden utilizar tanto un perfil de preferencias explícitamente expuestas por el usuario (si existe), como la historia de contenidos visitados o las valoraciones a los contenidos explícitamente aportadas por el usuario.

Los recomendadores basados en contenido proporcionan
recomendaciones comparando representaciones que describen a un
ítem con representaciones de contenido que al usuario le gusta.

Se pueden distinguir tres componentes principales en los recomendadores de contenidos (Lops *et al.*, 2011):



Como se ha comentado, los ítems o contenidos requieren ir acompañados de metadatos o alguna información estructurada que los describa y clasifique para poder ser recomendados. Sucede en muchas ocasiones que esta información es extraída mediante un analizador de contenidos y no tiene un formato bien definido o estructurado (atributos con valores bien definidos) (Lops *et al.*, 2011). Para estructurar esta información, muchos sistemas de recomendación representan el contenido mediante vectores de n dimensiones donde cada dimensión corresponde a un término del vocabulario empleado para describir los contenidos.

Los contenidos se pueden representar con un vector de pesos, donde cada peso indica el grado de asociación entre el contenido y el término. Esto es una representación basada en el espacio de vectores (**VSM – Vector Space Model**).

En estos casos se requiere entonces ponderar términos y medir la similitud entre los vectores. Un modelo muy utilizado en la recomendación de documentos es la **ponderación TF-IDF (Term Frequency-Inverse Document Frequency)**.

El modelo **TF-IDF** establece que aquellos términos que ocurren frecuentemente en un documento pero que ocurren rara vez en otros documentos, serán más relevantes en el tema del documento en concreto.

La función **TF-IDF** que determina los pesos respecto a cada término t_k de un documento d_j siguiendo este método es la siguiente (Lops *et al.*, 2011; Pazzani y Billsus, 2007):

$$TF-IDF(t_k, d_j) = TF(t_k, d_j) \times \log \frac{N}{n_k}$$

Siendo N el total de documentos y n_k , el número de documentos de la colección en el que el término t_k ocurre al menos una vez. Esto implica que los términos raros no sean menos relevantes que los frecuentes.

La función **TF (term frequency)** se calcula como el cociente entre la frecuencia en que el término t_k aparece en el documento d_j y la máxima frecuencia de cualquiera de los términos en ese documento. La multiplicación por el logaritmo pretende que se tenga en cuenta que los términos raros no sean menos relevantes. Por otra parte, para normalizar, y no favorecer a los documentos largos, frente a los cortos, se utiliza la siguiente expresión:

$$w_{k,j} = \frac{TF-IDF(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} TF-IDF(t_s, d_j)^2}}$$

Siendo T el conjunto completo de términos.

Para medir la similitud entre dos documentos se ha de utilizar una medida de similitud, y dado que el documento se está representando mediante un vector, podrá hacerse uso de alguna de las diversas técnicas existentes para el cálculo de distancia entre vectores. Una de las medidas más utilizadas, y que se utiliza como referencia en este tema, es la **similitud del coseno**:

$$\text{similitud}(d_i, d_j) = \frac{\sum_k^T w_{ki} \bullet w_{kj}}{\sqrt{\sum_k^T w_{ki}^2} \bullet \sqrt{\sum_k^T w_{kj}^2}}$$

En sistemas de recomendación basados en VSM, los perfiles de usuario y los contenidos se representan como vectores de pesos calculados de forma análoga a lo que se ha explicado previamente. Una vez se tiene el vector del usuario esta medida de similitud de coseno puede ser utilizada para predecir el interés de un usuario en un ítem.

La recomendación de contenidos se puede tratar como una tarea de clasificación. Con esta aproximación, los atributos de entrada de cada ejemplo corresponden a los metadatos o una representación de un contenido, y la clase corresponde a la valoración de ese contenido por el usuario.

El problema de este tipo de sistemas es que basan la recomendación en la similitud de contenidos, con lo cual se está recomendando al usuario ítems similares y el usuario puede ya no estar interesado en ítems precisamente similares. Además, la similitud entre contenidos se basa en metadatos que no suelen contener información sobre aspectos subjetivos de los ítems.

Por ejemplo, en un sistema que oferta cursos, si un usuario ha realizado un curso sobre Inteligencia Artificial básica, el sistema le puede ofrecer cursos sobre Inteligencia Artificial avanzada. Podría darse el caso de que los contenidos sugeridos tengan en común la temática con el curso ya realizado. Sin embargo, el primero es un curso muy bien explicado mientras que alguno de los cursos ofertados puede ser complicados de seguir.

Este tipo de información subjetiva no se capta en estos sistemas y, por lo tanto, no se puede tener en cuenta. Por ello, existen sistemas recomendadores que utilizan una combinación de técnicas de filtrado colaborativo y técnicas de filtrado basado en contenidos, también conocidos como **sistemas híbridos**.

En el ejemplo anterior, la información de similitud entre contenidos se podría complementar con información subjetiva extraída de valoraciones de usuarios. Además, si se utiliza información de usuarios similares para hacer recomendaciones, se podrá recomendar contenidos no necesariamente similares. Por otra parte, si un usuario es atípico, no es similar a otros usuarios, o no existe un suficiente número de usuarios en el sistema, la información basada en contenidos permitirá ofrecerle algún tipo de recomendación. De igual forma, si un ítem nuevo no ha sido valorado, los sistemas basados en contenido pueden recomendarlo. Además, este tipo de sistemas puede facilitar explicaciones más sencillas al usuario acerca de los motivos de las recomendaciones proporcionadas, las cuales, como se ha indicado, están basadas en las características de los contenidos.

8.5. Sistemas de recomendación híbridos

Tal y como puede intuirse de lo expuesto en las secciones anteriores, los sistemas de recomendación que utilizan filtrados demográficos o colaborativos, así como los basados en contenidos, utilizan estrategias específicas que ayudarán a determinar una recomendación en base, también, a criterios específicos. La intuición o el sentido común indicarían que, para tratar de obtener un sistema más eficaz que tenga en cuenta múltiples criterios, podrían combinarse varias de estas estratégicas, formándose así los **sistemas de recomendación híbridos**.

Los sistemas de recomendación híbridos combinan dos o más estrategias de recomendación para, así, beneficiarse de sus ventajas complementarias (Çano & Morisio, 2017). El objetivo principal de los sistemas de recomendación híbridos es reforzar las ventajas de cada uno de ellos y reducir o eliminar desventajas y limitaciones independientes.

Una primera aproximación es la combinación entre el filtrado colaborativo, para buscar preferencias similares entre los usuarios, y el filtrado basado en contenidos, para así encontrar contenidos similares entre sí (Balabanović & Shoham, 1997). Son múltiples los métodos de hibridación que pueden formarse gracias a la combinación de técnicas. Este tema define los siguientes, basados en la clasificación hecha por Burke (2002):

- ▶ **Ponderados (Weighted):** las puntuaciones obtenidas por varias técnicas se combinan para producir una sola recomendación.
- ▶ **Conmutados (Switching):** el sistema cambia de técnica de recomendación elegida en función de una serie de criterios definidos en su implementación.
- ▶ **Mixtos (Mixed):** se presentan recomendaciones de varios recomendadores diferentes al mismo tiempo.

- ▶ **Combinación de características (*Feature combination*):** se utilizan múltiples salidas de múltiples recomendadores como entradas en un solo algoritmo de recomendación.
- ▶ **En cascada (*Cascade*):** un recomendador refina las recomendaciones que otro ha hecho.
- ▶ **Características aumentadas (*Features Augmentation*):** la salida de una técnica se utiliza como una característica de entrada en otro.
- ▶ **Meta-recomendador (*Meta-level*):** el modelo aprendido por un recomendador se utiliza como entrada en otro.

Tal y como se ha indicado anteriormente, el objetivo principal de la hibridación es aliviar algunos de los problemas y limitaciones asociados a la aplicación de las diferentes técnicas de forma autónoma. Los sistemas híbridos de filtrado de contenido o filtrado colaborativo, independientemente del tipo, siempre presentarán la limitación en su escalado, pues ambas técnicas necesitan una base de datos de calificaciones preexistente que, en la mayoría de las ocasiones, requiere tiempo para completarse. Para solventar este problema se utilizan calificaciones ya existentes o deducidas de otros datos.

Por otro lado, las meta-técnicas evitan el problema de la falta de datos coincidentes entre usuarios o ítems (*sparsity* o escasez) al comprimir las evaluaciones obtenidas de muchos ejemplos en un modelo en el que será más fácil comparar entre los usuarios. Las técnicas basadas en el conocimiento y en la utilidad parecen ser buenas candidatas para la hibridación, ya que no están sujetas a problemas de escalado (Burke, 2002).

A la hora de combinar las técnicas debe tenerse en cuenta el orden de aplicación. Existen combinaciones que serán insensibles al orden (ponderados, mixtos, conmutados y combinación de características). El resultado de combinar primero un

filtrado colaborativo y después un filtrado basado en contenidos o viceversa sería el mismo. Por otro lado, las combinaciones en cascada de características aumentadas o los meta-recomendadores, sí serán sensibles al orden. Por ejemplo, un híbrido de aumento de características que utilice un recomendador basado en contenido para generar características, que sirvan como entrada a un segundo sistema de recomendación de filtrado colaborativo, sería muy diferente de otro que utilizara la colaboración en primer lugar. **Sirva como ejemplo el filtrado de noticias:** el resultado de filtrar, en primer lugar, el contenido y después restringir los usuarios a los que se le entrega será diferente del obtenido de filtrar primero a los usuarios y después decidor el contenido a recomendar. Por este motivo, en este sistema deben considerarse todas las permutaciones posibles (Burke, 2002; Çano & Morisio, 2017).

Además de estas consideraciones teóricas y prácticas, en la actualidad los sistemas híbridos permiten la combinación de información de diferentes fuentes. Así, la información subjetiva generada en redes sociales puede considerarse como un factor para las recomendaciones, o bien podemos utilizar recomendaciones hechas con diferentes fuentes de datos (por ejemplo, en diferentes redes sociales) para obtener una recomendación «más global».

En definitiva, puede comprobarse que el ámbito de aplicación de los sistemas de recomendación se ve ampliamente ensanchado con la combinación de estos, pudiendo mejorar sus resultados si se diseñan de forma adecuada.

8.6. Ejemplos de implementación

En este tema vamos a seguir el ejemplo de Derrick Mwiti (ver Webgrafía al final del tema) para implementar un sistema recomendador en Python usando las librerías pandas, numpy y seaborn, todas ya instaladas en los ejemplos de temas anteriores.

Para ello, vamos a partir del *dataset* de películas MovieLens, disponible en:
<https://grouplens.org/datasets/movielens/>

Como el *dataset* completo tiene más de 25 millones de ratings de 280,000 usuarios sobre 58,000 películas (más de 250 MB de información comprimida), vamos a utilizar una versión reducida en el mismo portal, denominada *Small*. Este *dataset* reducido cuenta con 100,000 *ratings* y 3,600 *tags* aplicados a 9,000 películas por parte de 600 usuarios, ocupando solo 1 MB en su versión comprimida y suficiente para nuestros propósitos de prueba. Lo podemos descargar directamente de:
<http://files.grouplens.org/datasets/movielens/ml-latest-small.zip>

Una vez descargado, lo descomprimimos, resultando en cuatro *datasets* diferentes, de los cuales nos quedaremos con dos, con las siguientes columnas cada uno de ellos:

► **movies.csv:**

- movie_id: identificador de la película (número entero).
- title: título de la misma con su año.
- genres: una combinación de géneros, por ejemplo: Comedy | Romance

► **ratings.csv:**

- user_id: identificador del usuario (número entero).
- movie_id: identificador de la película (número entero).
- rating: puntuación, número real entre 0.0 y 5.0.
- timestamp: marca de tiempo del momento en que se realizó el rating (en número de segundos desde 01/01/1970 UTC).

Podemos echar un vistazo a los CSV con un editor (como Visual Studio Code) o con Excel.

Por ejemplo, el archivo `movies.csv` contiene una cabecera como la siguiente:

```
movied,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
4,Waiting to Exhale (1995),Comedy|Drama|Romance
5,Father of the Bride Part II (1995),Comedy
```

Mientras que el archivo ratings.csv incluye información como:

```
userId,movieId,rating,timestamp
1,1,4.0,964982703
1,3,4.0,964981247
1,6,4.0,964982224
1,47,5.0,964983815
1,50,5.0,964982931
```

Carguemos con pandas el contenido de ratings.csv y mostremos la información:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
print(df.head())
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Nos interesa más el nombre de la película que el identificador, si cargamos los nombres de las películas con el siguiente código y lo mostramos:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('movies.csv')
print(df.head())
```

movieId	genres	title
0	1 Adventure Animation Children Comedy Fantasy	Toy Story (1995)
1	2 Adventure Children Fantasy	Jumanji (1995)
2	3 Comedy Romance	Grumpier Old Men (1995)
3	4 Comedy Drama Romance	Waiting to Exhale (1995)
4	5 Comedy	Father of the Bride Part II (1995)

Dado que ambos *datasets* coinciden en la columna `movieId`, podemos unir ambos datasets mediante `pd.merge()` y veamos el contenido y sus características:

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

print("<HEAD>")
print(df.head())
print("<DESCRIBE>")
print(df.describe())
print("<INFO>")
print(df.info())
```

```

<HEAD>
    userId      movieId  rating   timestamp        title           genres
0       1          1     4.0  964982703  Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
1       5          1     4.0  847434962  Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
2       7          1     4.5  1106635946 Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
3      15          1     2.5  1510577970 Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
4      17          1     4.5  1305696483 Toy Story (1995)  Adventure|Animation|Children|Comedy|Fantasy
<DESCRIBE>
    userId      movieId  rating   timestamp
count  100836.000000  100836.000000  100836.000000  1.008360e+05
mean   326.127564   19435.295718   3.501557   1.205946e+09
std    182.618491   35520.987199   1.042529   2.162610e+08
min    1.000000    1.000000    0.500000   8.281246e+08
25%   177.000000   1199.000000   3.000000   1.019124e+09
50%   225.000000   2991.000000   3.500000   1.186087e+09
75%   477.000000   8122.000000   4.000000   1.435994e+09
max    610.000000   193609.000000  5.000000   1.587799e+09
<INFO>
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100836 entries, 0 to 100835
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   userId      100836 non-null  int64  
 1   movieId     100836 non-null  int64  
 2   rating      100836 non-null  float64
 3   timestamp   100836 non-null  int64  
 4   title       100836 non-null  object 
 5   genres      100836 non-null  object 
dtypes: float64(1), int64(3), object(2)
memory usage: 5.4+ MB
None

```

Podemos ver que el *rating* medio es 3.5, el mínimo 0.5 y el máximo 5.0. Además, contamos con 100,836 entradas o *ratings* de los usuarios a las películas.

Ahora vamos a crear un *dataset* con la calificación promedio de cada película y el número de calificaciones. Vamos a usar estas clasificaciones para calcular la correlación entre las películas más adelante. La correlación es una medida estadística que indica el grado en que dos o más variables fluctúan juntas. Las películas que tienen un alto coeficiente de correlación son las películas más similares entre sí.

En nuestro caso utilizaremos el coeficiente de correlación de Pearson. Este número estará entre -1 y 1. 1 indica una correlación lineal positiva mientras que -1 indica una correlación negativa. 0 indica que no hay correlación lineal. Por lo tanto, las películas con una correlación cero no son para nada similares. Para crear este *dataframe* usamos la funcionalidad de pandas `groupby()`. Agrupamos el conjunto de datos por la columna del título y calculamos su media para obtener la clasificación media de cada

película.

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
print(ratings.head())
```

title	rating
'71 (2014)	4.0
'Hellboy': The Seeds of Creation (2004)	4.0
'Round Midnight (1986)	3.5
'Salem's Lot (2004)	5.0
'Til There Was You (1997)	4.0

Pero, además, queremos ver el número de «audiencias» de cada película. Lo hacemos creando una columna con el número de clasificaciones. Esto es importante para que podamos ver la relación entre la calificación promedio de una película y el número de clasificaciones que obtuvo la película. Es muy posible que una película de 5 estrellas haya sido calificada por una sola persona. Por lo tanto, es estadísticamente incorrecto clasificar esa película como una película de 5 estrellas. Por lo tanto, necesitaremos establecer un umbral para el número mínimo de clasificaciones mientras construimos el sistema de recomendación. Para crear esta nueva columna utilizamos la utilidad de pandas groupby(). Agrupamos por la columna del título y luego usamos la función de conteo para calcular el número de clasificaciones que cada película obtuvo. Luego vemos el nuevo cuadro de datos usando la función head().

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

print(ratings.head())

      rating  number_of_ratings
title
'71 (2014)           4.0            1
'Hellboy': The Seeds of Creation (2004) 4.0            1
'Round Midnight (1986)          3.5            2
'Salem's Lot (2004)           5.0            1
'Til There Was You (1997)       4.0            2
```

Ahora vamos a trazar un histograma usando la funcionalidad de trazado de matplotlib y de seaborn para visualizar la distribución de las clasificaciones de las películas, la distribución del número de clasificaciones por película, así como la relación entre la clasificación de una película y el número de calificaciones recibidas usando diagramas de dispersión (con la función jointplot () de seaborn).

```

import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

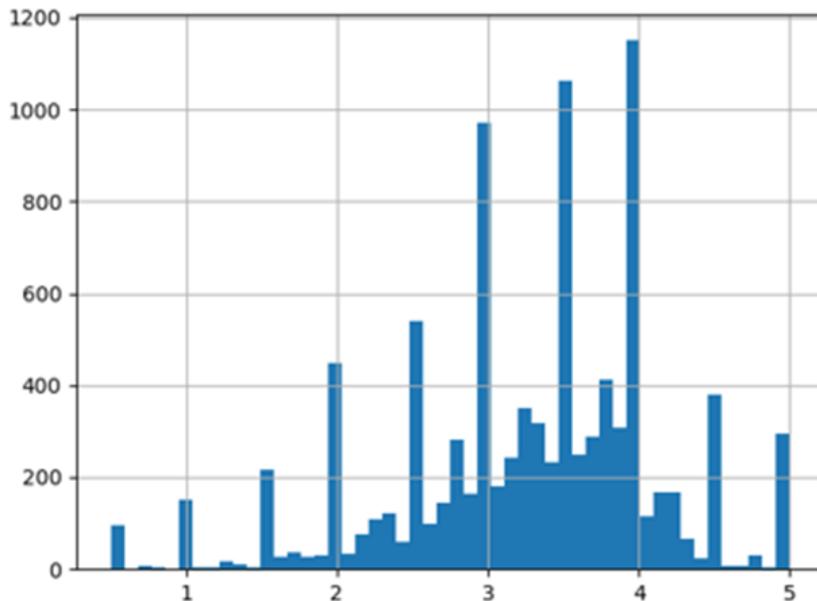
df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

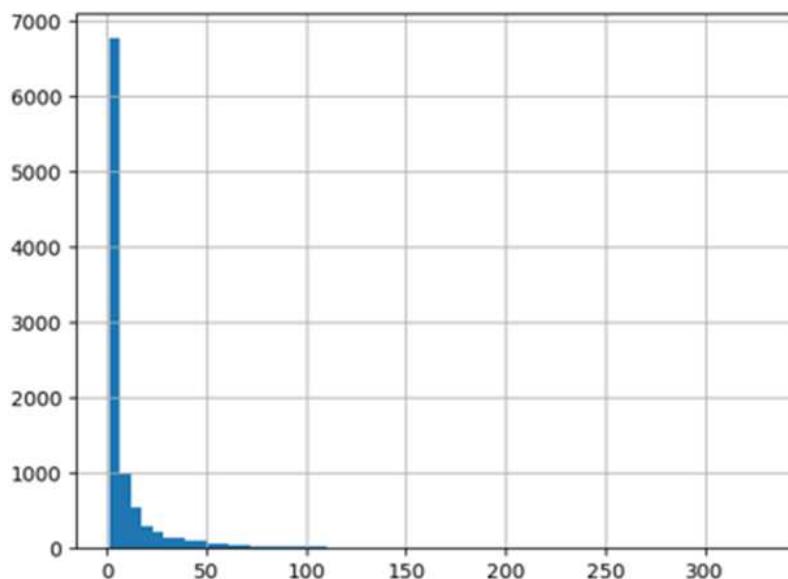
ratings['rating'].hist(bins=50)
plt.show()
ratings['number_of_ratings'].hist(bins=60)
plt.show()
sns.jointplot(x='rating', y='number_of_ratings', data=ratings)
plt.show()

```

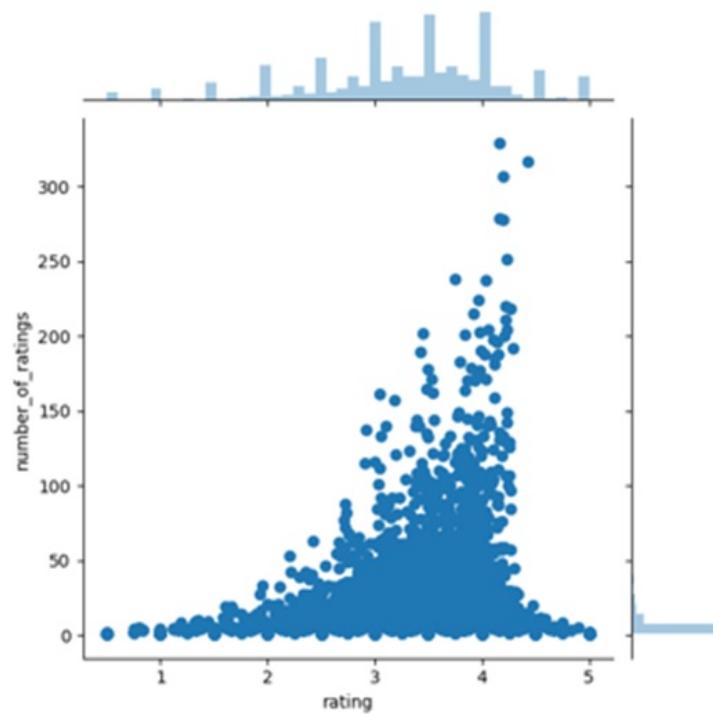
Podemos ver que la mayoría de las películas están clasificadas entre 2.5 y 4:



Del siguiente histograma se desprende claramente que la mayoría de las películas tienen pocos *ratings*. Las películas con más audiencia son las más famosas.



Comprobemos ahora la relación entre la clasificación de una película y el número de clasificaciones.



En el diagrama podemos ver que hay una relación positiva entre la calificación promedio de una película y el número de calificaciones. El gráfico indica que cuantas más calificaciones obtiene una película, más alta es la calificación promedio que obtiene. Esto es importante, sobre todo cuando se elige el umbral para el número de clasificaciones por película.

Ahora vamos a avanzar rápidamente y a crear un sencillo sistema de recomendación basado en ítems. Para ello, necesitamos convertir nuestro conjunto de datos en una matriz con los títulos de las películas como columnas, el identificador de usuario como índice y las clasificaciones como valores.

Haciendo esto, obtendremos un *dataframe* con las columnas como los títulos de la película y las filas como los identificadores de usuario. Cada columna representa todas las clasificaciones de una película por todos los usuarios. La clasificación aparece como NAN cuando un usuario no ha clasificado una determinada película. Usaremos esta matriz para calcular la correlación entre las puntuaciones de una película y el resto de las películas de la matriz. Usamos la utilidad pandas `pivot_table()` para crear la matriz de películas.

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')
print(movie_matrix.head())
```

```

title      '71 (2014) ... À nous la liberté (Freedom for Us)
(1931)

userId      ...
1           NaN      ...
NaN
2           NaN      ...
NaN
3           NaN      ...
NaN
4           NaN      ...
NaN
5           NaN      ...
NaN

```

A continuación, veamos las películas más votadas y escojamos dos de ellas para trabajar con este sencillo sistema de recomendación. Usamos la utilidad de pandas `sort_values()` y establecemos el orden descendente para ordenar las películas más valoradas de mayor a menor. Luego usamos la función `head()` para ver las 10 más votadas.

```

import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', 
values='rating')

print(ratings.sort_values('number_of_ratings', ascending=False).head(10))

```

	rating	number_of_ratings
title		
Forrest Gump (1994)	4.164134	329
Shawshank Redemption, The (1994)	4.429022	317
Pulp Fiction (1994)	4.197068	307
Silence of the Lambs, The (1991)	4.161290	279
Matrix, The (1999)	4.192446	278
Star Wars: Episode IV - A New Hope (1977)	4.231076	251
Jurassic Park (1993)	3.750000	238
Braveheart (1995)	4.031646	237
Terminator 2: Judgment Day (1991)	3.970982	224
Schindler's List (1993)	4.225000	220

Supongamos que un usuario ha visto *Air Force One* (1997) y *Contact* (1997). Queremos recomendar películas a este usuario basadas en su historial de visitas. El objetivo es buscar películas similares a *Contact* (1997) y *Air Force One* (1997) que recomendaremos a este usuario.

Podemos lograrlo calculando la correlación entre las clasificaciones de estas dos películas y las clasificaciones del resto de las películas del conjunto de datos. El primer paso es crear un *dataframe* con las clasificaciones de estas películas de nuestra `movie_matrix`. Así, tendremos los *dataframes* que muestran el `userId` y la calificación que le dieron a las dos películas.

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')

AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']

print(AFO_user_rating.head())
print(contact_user_rating.head())
```

userId

1 NaN

2 NaN

3 NaN

4 NaN

5 NaN

Name: Air Force One (1997), dtype: float64

userId

1 NaN

2 NaN

3 NaN

4 NaN

5 NaN

Name: Contact (1997), dtype: float64

Veremos muchos NAN porque muchos usuarios no las han calificado, pero con `head(50)` podríamos ver ratings de determinados usuarios.

Para calcular la correlación entre dos *dataframes* utilizamos la funcionalidad de los pandas `corrwith().corrwith()` calcula la correlación por pares de las filas o columnas de dos objetos del *dataframe*. Utilicemos esta funcionalidad para obtener la correlación entre la clasificación de cada película y la del *Air Force One*.

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')

AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']

similar_to_air_force_one=movie_matrix.corrwith(AFO_user_rating)
print(similar_to_air_force_one)
```

```
'71 (2014)           NaN
'Hellboy': The Seeds of Creation (2004)      NaN
'Round Midnight (1986)           NaN
'Salem's Lot (2004)           NaN
'Til There Was You (1997)      NaN
...
eXistenZ (1999)          0.129099
xxx (2002)              -0.188006
xxx: State of the Union (2005)      NaN
;Three Amigos! (1986)          -0.187477
À nous la liberté (Freedom for Us) (1931)      NaN
Length: 9719, dtype: float64
```

Veamos las películas similares a Contact (1997):

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')

AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']

similar_to_air_force_one=movie_matrix.corrwith(AFO_user_rating)
similar_to_contact = movie_matrix.corrwith(contact_user_rating)
print(similar_to_contact)
```

```

'71 (2014)           NaN
'Hellboy': The Seeds of Creation (2004)    NaN
'Round Midnight (1986)      NaN
'Salem's Lot (2004)        NaN
'Til There Was You (1997)    NaN
...
eXistenZ (1999)          -0.068793
xXx (2002)                -0.300000
xXx: State of the Union (2005)      NaN
;Three Amigos! (1986)          0.326414
À nous la liberté (Freedom for Us) (1931)    NaN
Length: 9719, dtype: float64

```

Como se ha notado anteriormente, nuestra matriz tenía muchos valores perdidos, ya que no todas las películas fueron clasificadas por todos los usuarios. Por lo tanto, dejamos caer esos valores nulos y transformamos los resultados de la correlación en cuadros de datos para que los resultados se vean más atractivos.

```

import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')

AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']

similar_to_air_force_one=movie_matrix.corrwith(AFO_user_rating)

```

```

similar_to_contact = movie_matrix.corrwith(contact_user_rating)

corr_contact = pd.DataFrame(similar_to_contact, columns=['Correlation'])
corr_contact.dropna(inplace=True)
print("corr_contact")
print(corr_contact.head())

corr_AFO = pd.DataFrame(similar_to_air_force_one, columns=['correlation'])
corr_AFO.dropna(inplace=True)
print("corr_AFO")
print(corr_AFO.head())

```

corr_contact

	Correlation
title	
'burbs, The (1989)	0.486761
(500) Days of Summer (2009)	0.634064
*batteries not included (1987)	0.868599
...And Justice for All (1979)	1.000000
10 Things I Hate About You (1999)	-0.102640

corr_AFO

	correlation
title	
'burbs, The (1989)	0.168459
(500) Days of Summer (2009)	0.086874
*batteries not included (1987)	-0.866025
10 Cloverfield Lane (2016)	0.192450
10 Items or Less (2006)	-1.000000

Estos dos *dataframes* nos muestran las películas más parecidas a las de *Contact* (1997) y *Air Force One* (1997), respectivamente. Sin embargo, algunas de las películas tienen muy pocas clasificaciones y pueden terminar siendo recomendadas simplemente porque una o dos personas les dieron una clasificación de 5 estrellas. Podemos arreglar esto estableciendo un umbral para el número de clasificaciones. En el histograma anterior vimos un fuerte descenso en el número de clasificaciones desde 100. Por lo tanto, fijaremos esto como el umbral. Sin embargo, este es un

número con el que se puede jugar hasta que se consiga una opción adecuada. Para ello, necesitamos unir los dos *dataframes* con la columna `number_of_ratings` en el *dataframe* `ratings`.

```
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')

AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']

similar_to_air_force_one=movie_matrix.corrwith(AFO_user_rating)
similar_to_contact = movie_matrix.corrwith(contact_user_rating)

corr_contact = pd.DataFrame(similar_to_contact, columns=['Correlation'])
corr_contact.dropna(inplace=True)
corr_contact = corr_contact.join(ratings['number_of_ratings'])

print("corr_contact")
print(corr_contact.head())

corr_AFO = pd.DataFrame(similar_to_air_force_one, columns=['correlation'])
corr_AFO.dropna(inplace=True)
corr_AFO = corr_AFO.join(ratings['number_of_ratings'])
print("corr_AFO")
print(corr_AFO.head())
```

```

corr_contact
Correlation  number_of_ratings
title
'burbs, The (1989)          0.486761      17
(500) Days of Summer (2009) 0.634064      42
*batteries not included (1987) 0.868599      7
...And Justice for All (1979) 1.000000      3
10 Things I Hate About You (1999) -0.102640     54

corr_AFO
correlation  number_of_ratings
title
'burbs, The (1989)          0.168459      17
(500) Days of Summer (2009) 0.086874      42
*batteries not included (1987) -0.866025      7
10 Cloverfield Lane (2016)   0.192450      14
10 Items or Less (2006)      -1.000000      3

```

Finalmente, nos quedamos con la similitud de las películas en base a su correlación con las dos con las que estamos probando, pero que tengan más de 100 *ratings* cada una de ellas, y las ordenamos de mayor a menor correlación:

```

import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')

df = pd.read_csv('ratings.csv')
movie_titles = pd.read_csv('movies.csv')
df = pd.merge(df, movie_titles, on='movieId')

ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['number_of_ratings'] = df.groupby('title')['rating'].count()

movie_matrix = df.pivot_table(index='userId', columns='title', values='rating')

AFO_user_rating = movie_matrix['Air Force One (1997)']
contact_user_rating = movie_matrix['Contact (1997)']

```

```

similar_to_air_force_one=movie_matrix.corrwith(AFO_user_rating)
similar_to_contact = movie_matrix.corrwith(contact_user_rating)

corr_contact = pd.DataFrame(similar_to_contact, columns=['Correlation'])
corr_contact.dropna(inplace=True)
corr_contact = corr_contact.join(ratings['number_of_ratings'])
print("corr_contact")
print(corr_contact[corr_contact['number_of_ratings'] > 100].sort_values(by='Correlation', ascending=False).head(10))

corr_AFO = pd.DataFrame(similar_to_air_force_one, columns=['correlation'])
corr_AFO.dropna(inplace=True)
corr_AFO = corr_AFO.join(ratings['number_of_ratings'])
print("corr_AFO")
print(corr_AFO[corr_AFO['number_of_ratings'] > 100].sort_values(by='correlation', ascending=False).head(10))

```

	Correlation	
number_of_ratings		
title		
Sleepless in Seattle (1993)	0.689602	106
American Pie (1999)	0.670109	103
Clear and Present Danger (1994)	0.641203	110
Firm, The (1993)	0.640332	101
Bourne Identity, The (2002)	0.639769	112
Outbreak (1995)	0.586934	101
E.T. the Extra-Terrestrial (1982)	0.569043	122
Apollo 13 (1995)	0.563138	201
Die Hard: With a Vengeance (1995)	0.552904	144
Four Weddings and a Funeral (1994)	0.542013	103

	correlation	number_of_ratings
title		
Clear and Present Danger (1994)	0.698836	110
Net, The (1995)	0.598322	112
Green Mile, The (1999)	0.574799	111
Firm, The (1993)	0.561304	101
Departed, The (2006)	0.543279	107
Apollo 13 (1995)	0.536136	201
Twister (1996)	0.511892	123
American Pie (1999)	0.501064	103
Truman Show, The (1998)	0.500529	125
Cliffhanger (1993)	0.500267	101

8.7. Referencias bibliográficas

Alharthi, H., Inkpen, D. & Szpakowicz, S. (2018). A survey of book recommender systems. *Journal of Intelligent Information Systems*, 51(1), 139-160. Disponible en <https://doi.org/10.1007/s10844-017-0489-9>

Augstein, M., Herder, E. & Wörndl, W. (2019). *Personalized Human-Computer Interaction*. Walter de Gruyter GmbH & Co KG.

Balabanović, M. & Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3), 66–72.

Bobadilla, J., Ortega, F., Hernando, A. & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, 46, 109-132. Disponible en <https://doi.org/10.1016/j.knosys.2013.03.012>

Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4), 331-370. Disponible en <https://doi.org/10.1023/A:1021240730564>

Campana, M. G. & Delmastro, F. (2017). Recommender Systems for Online and Mobile Social Networks: A survey. *Online Social Networks and Media*, 3-4, 75-97. Disponible en <https://doi.org/10.1016/j.osnem.2017.10.005>

Çano, E. & Morisio, M. (2017). Hybrid Recommender Systems: A Systematic Literature Review. *Intelligent Data Analysis*, 21(6), 1487-1524. Disponible en <https://doi.org/10.3233/IDA-163209>

Eirinaki, M., Gao, J., Varlamis, I. & Tserpes, K. (2018). Recommender Systems for Large-Scale Social Networks: A review of challenges and solutions. *Future Generation Computer Systems*, 78, 413-418. Disponible en <https://doi.org/10.1016/j.future.2017.09.015>

Jannach, D. (Ed.). (2011). *Recommender systems: An introduction*. Cambridge University Press.

Joshi, A. V. (2020). Recommendations Systems. En A. V. Joshi (Ed.), *Machine Learning and Artificial Intelligence* (pp. 199-204). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-030-26622-6_21

Karimi, M., Jannach, D. & Jugovac, M. (2018). News recommender systems – Survey and roads ahead. *Information Processing & Management*, 54(6), 1203-1227. Disponible en <https://doi.org/10.1016/j.ipm.2018.04.008>

Kunaver, M. & Požrl, T. (2017). Diversity in recommender systems – A survey. *Knowledge-Based Systems*, 123, 154-162. Disponible en <https://doi.org/10.1016/j.knosys.2017.02.009>

Lemire, D., Boley, H., McGrath, S. & Ball, M. (2005). Collaborative filtering and inference rules for context-aware learning object recommendation. *Interactive Technology and Smart Education*, 2(3), 179-188. Disponible en <https://doi.org/10.1108/17415650580000043>

Linda, S. & Bharadwaj, K. K. (2018). A Decision Tree Based Context-Aware Recommender System. En U. S. Tiwary (Ed.), *Intelligent Human Computer Interaction* (pp. 293-305). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-030-04021-5_27

Linden, G., Smith, B. & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 76-80. Disponible en <https://doi.org/10.1109/MIC.2003.1167344>

Lops, P., de Gemmis, M. & Semeraro, G. (2011). Content-based Recommender Systems: State of the Art and Trends. En F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender Systems Handbook* (pp. 73-105). Springer US.

Disponible en https://doi.org/10.1007/978-0-387-85820-3_3

Montaner, M., López, B. & de la Rosa, J. L. (2003). A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review*, 19(4), 285-330. Disponible en
<https://doi.org/10.1023/A:1022850703159>

Mu, R. (2018). A Survey of Recommender Systems Based on Deep Learning. *IEEE Access*, 6, 69009-69022. Disponible en
<https://doi.org/10.1109/ACCESS.2018.2880197>

Najafabadi, M. K., Mohamed, A. Hj. & Mahrin, M. N. (2019). A survey on data mining techniques in recommender systems. *Soft Computing*, 23(2), 627-654. Disponible en
<https://doi.org/10.1007/s00500-017-2918-7>

Osadchiy, T., Poliakov, I., Olivier, P., Rowland, M. & Foster, E. (2019). Recommender system based on pairwise association rules. *Expert Systems with Applications*, 115, 535-542. <https://doi.org/10.1016/j.eswa.2018.07.077>

Pazzani, M. J. & Billsus, D. (2007). Content-Based Recommendation Systems. En P. Brusilovsky, A. Kobsa & W. Nejdl (Eds.), *The Adaptive Web: Methods and Strategies of Web Personalization* (pp. 325-341). Springer. Disponible en
https://doi.org/10.1007/978-3-540-72079-9_10

Ricci, F., Rokach, L., Shapira, B. & Kantor, P. B. (Eds.). (2011). *Recommender Systems Handbook*. Springer US. Disponible en <https://doi.org/10.1007/978-0-387-85820-3>

Sánchez-Corcuera, R., Casado-Mansilla, D., Borges, C. E. & López-de-Ipiña, D. (2020). Persuasion-based recommender system ensambling matrix factorisation and active learning models. *Personal and Ubiquitous Computing*. Disponible en
<https://doi.org/10.1007/s00779-020-01382-7>

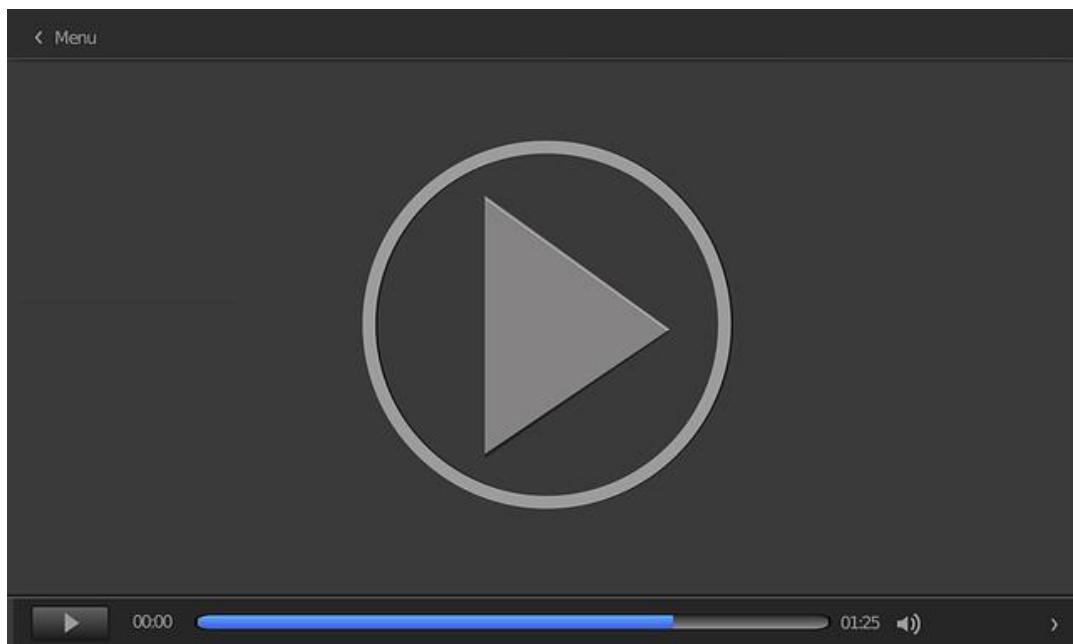
Singh, S. P. & Solanki, S. (2019). Recommender System Survey: Clustering to Nature Inspired Algorithm. En C. R. Krishna, M. Dutta, & R. Kumar (Eds.), *Proceedings of 2nd International Conference on Communication, Computing and Networking* (pp. 757-768). Springer. Disponible en https://doi.org/10.1007/978-981-13-1217-5_76

Singhal, A., Sinha, P. & Pant, R. (2017). Use of Deep Learning in Modern Recommendation System: A Summary of Recent Works. *International Journal of Computer Applications*, 180(7), 17-22. Disponible en <https://doi.org/10.5120/ijca2017916055>

Yao, L., Xu, Z., Zhou, X. & Lev, B. (2019). Synergies Between Association Rules and Collaborative Filtering in Recommender System: An Application to Auto Industry. En F. P. García Márquez & B. Lev (Eds.), *Data Science and Digital Business* (pp. 65-80). Springer International Publishing. Disponible en https://doi.org/10.1007/978-3-319-95651-0_5

Introducción a los sistemas recomendadores

Esta lección magistral consiste en mostrar ejemplos reales de aplicaciones en Internet, con las que el alumno está probablemente muy familiarizado, que realizan recomendaciones de contenidos a los usuarios, comentando las posibles técnicas que pueden facilitar dichas recomendaciones.



08. Introducción a los sistemas recomendadores

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=833c151b-b009-46b4-90b0-aff800fa6142>

Taxonomía de agentes recomendadores

Montaner, M., López, B. & de la Rosa, J.L. (2003). A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review*, 19, 285-330.
<http://eia.udg.edu/~blopez/publicaciones/montaner-aireview03.pdf>

Aunque el artículo presenta el estado de arte de los sistemas recomendadores hace 10 años, es muy interesante su lectura en la actualidad, ya que, en primer lugar, el artículo describe todas aquellas tareas necesarias para que un sistema realice la recomendación en función de su tipo. Para cada tarea, los autores hacen un recorrido de las diferentes técnicas aplicables, siendo muchas de ellas de actualidad. En este recorrido, el artículo tiene en cuenta 37 sistemas reales con diferentes propósitos: recomendadores web, recomendadores de noticias, música y películas, comercio electrónico, etc.

Sistemas recomendadores basados en contenidos

Lops, P., Gemmis, M., & Semeraro, G. (2011). Content-based Recommender Systems: State of the Art and Trends. En F. Ricci *et al.* (eds.). *Recommender Systems Handbook*, p. 73-105. Springer Science+Business Media. <http://www.ics.uci.edu/~welling/teaching/CS77Bwinter12/handbook/ContentBasedRS.pdf>

Artículo completo y claro que describe los sistemas recomendadores basados en contenidos, describiendo su arquitectura y recorriendo las diferentes técnicas utilizadas para representar contenidos, representar perfiles de usuarios, medir similitudes entre contenidos, etc. Además, incluye temas recientes como el *social tagging*, relativo al marcado de documentos por los usuarios y los sistemas que recomiendan estos contenidos en base tags sociales.

Recomendación de productos basada en filtrado colaborativo

Accede a la tesis desde el aula virtual o a través de la siguiente dirección web:

http://ruc.udc.es/dspace/bitstream/2183/10122/2/FormosoLopez_Vreixo_TD_2013.pdf

f

Tesis de doctorado de Vreixo Formoso López (Departamento de Tecnoloxías da Información e as Comunicacións de la Universidade da Coruña) titulado «Técnicas eficientes para la recomendación de productos basadas en filtrado colaborativo», que describe y profundiza en los sistemas de recomendación basados en filtrado colaborativo, abordando diversos problemas que afectan a los algoritmos utilizados en estos sistemas.

Medidas de asociación

Accede al capítulo desde el aula virtual o a través de la siguiente dirección web:

<http://www.ugr.es/~gallardo/pdf/cluster-2.pdf>

Capítulo relativo a medidas de distancias y similitudes, elaborado por el Departamento de Estadística e Investigación Operativa de la Universidad de Granada, que incluye medidas de asociación entre variables y entre individuos. Es un capítulo muy completo comprendiendo medidas frecuentemente utilizadas en algoritmos de *clustering* o en otro tipo de sistemas como en los sistemas recomendadores.

Sistemas de recomendación

Accede al curso desde el aula virtual o a través de la siguiente dirección web:

http://www.ics.uci.edu/~welling/teaching/CS77Bwinter12/CS77B_w12.html

Sitio web de un curso sobre sistemas de recomendación donde están disponibles enlaces relativos a sistemas recomendadores y a *datasets* (conjuntos de datos). Además, se puede acceder a las diapositivas del curso que explican temas relativos a los sistemas de recomendación así como a propuestas de ejercicios con MATLAB.

- 1.** Los sistemas que utilizan valoraciones de los usuarios para recomendar contenidos emplean técnicas de:

 - A. Filtrado demográfico.
 - B. Filtrado colaborativo.
 - C. Filtrado basado en contenidos.
 - D. Filtrado basado en usuarios.

- 2.** Los sistemas que utilizan información descriptiva de los contenidos para recomendar contenidos emplean técnicas de:

 - A. Filtrado demográfico.
 - B. Filtrado colaborativo.
 - C. Filtrado basado en contenidos.
 - D. Filtrado basado en usuarios.

- 3.** Indica cuáles de las siguientes afirmaciones son correctas:

 - A. Se pueden utilizar árboles de decisión o reglas inducidas para representar el perfil del usuario en un sistema recomendador.
 - B. Algunos sistemas recomendadores utilizan técnicas de clústering para generar grupos de usuarios con perfil similar.
 - C. Las técnicas de clústering no son utilizadas por los sistemas recomendadores.
 - D. Los sistemas de recomendación nunca solicitan información al usuario de manera explícita.

- 4.** Indica cuáles de las siguientes afirmaciones son correctas:
- A. Los sistemas de filtrado colaborativo basado en ítems calculan la similitud entre ítems.
 - B. Los sistemas de filtrado colaborativo basado en ítems calculan la similitud entre usuarios.
 - C. Los sistemas de filtrado basado en contenidos calculan la similitud entre ítems.
 - D. Los sistemas de filtrado colaborativo basado en usuarios calculan la similitud entre ítems.
- 5.** Slope One es un algoritmo empleado en los sistemas de:
- A. Filtrado demográfico.
 - B. Filtrado colaborativo basado en usuarios.
 - C. Filtrado basado en contenidos.
 - D. Filtrado colaborativo basado en ítems.
- 6.** La función de predicción de la valoración de un ítem en el algoritmo Slope One se basa en:
- A. La media de valoraciones dadas al ítem por los usuarios.
 - B. La media de la diferencia de valoraciones entre ítems.
 - C. La mediana de valoraciones dadas al ítem por los usuarios.
 - D. El valor máximo de la diferencia de valoraciones entre ítems.

7. Marca las frases que son correctas respecto al algoritmo de filtrado colaborativo ítem a ítem:

- A. Se basa en encontrar ítems similares.
- B. Se basa en encontrar usuarios similares.
- C. Se basa en metadatos de los ítems.
- D. Se basa en datos binarios como las adquisiciones o no adquisiciones de los ítems por parte de los usuarios.

8. ¿Qué sistemas presentan problemas para recomendar ítems recientemente incorporados al sistema?

- A. Filtrado colaborativo basado en usuarios.
- B. Filtrado basado en contenidos.
- C. Filtrado colaborativo basado en ítems.
- D. Ninguno de los anteriores.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto al modelo TF-IDF.

- A. Es un modelo utilizado en los sistemas recomendadores basados en contenidos.
- B. Utiliza la similitud del coseno para medir la similitud entre documentos.
- C. Utiliza la función TF-IDF para calcular la similitud entre documentos.
- D. Favorece a los documentos largos frente a los cortos.

10. Algunos problemas que los recomendadores basados en contenido presentan son:

- A. Al usuario no siempre le interesan los ítems similares.
- B. Cuando un ítem no está valorado no es recomendado.
- C. No tienen en cuenta información subjetiva.
- D. No puede ofrecer información a usuarios atípicos, que no son similares a otros usuarios.

Técnicas de Inteligencia Artificial

Tema 9. Resolución de problemas mediante búsqueda

Índice

Esquema

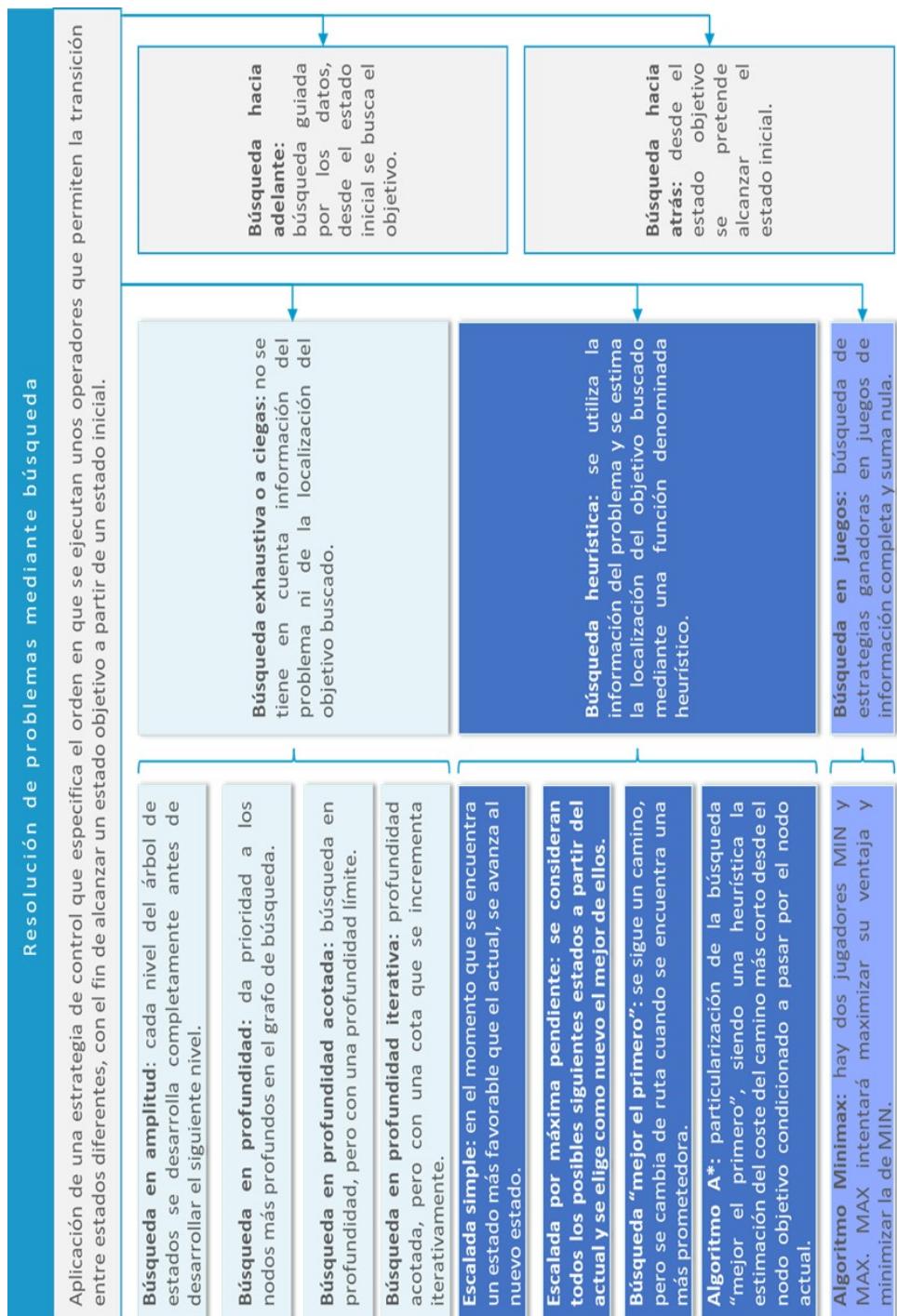
Ideas clave

- 9.1. ¿Cómo estudiar este tema?
- 9.2. Introducción. «El mundo de los bloques»
- 9.3. Dirección de la búsqueda
- 9.4. Búsqueda exhaustiva o a ciegas
- 9.5. Búsqueda heurística
- 9.6. Búsqueda en juegos
- 9.7. Costes
- 9.8. Aplicaciones prácticas y ejemplos de implementación
- 9.9. Referencias bibliográficas

A fondo

- Búsqueda en juegos: algoritmo Minimax
- Métodos de búsquedas, a ciegas y heurísticas
- Making games
- Applets de Java para visualizar el funcionamiento de estrategias de búsqueda a ciegas
- NetworkX

Test



9.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral y revisando los recursos adicionales que se facilitan.

Al finalizar el estudio de este tema el alumno será capaz de:

- ▶ Describir diferentes métodos de búsqueda de soluciones.
- ▶ Seleccionar y aplicar métodos de búsqueda adecuados a diferentes problemas.

9.2. Introducción. «El mundo de los bloques»

Existen dos tipos de agentes: inteligentes y no inteligentes. Empezando por los segundos, un agente (por ejemplo, un robot) se puede programar para actuar de tal manera que alcance unos objetivos fijos, sin que sea capaz de adaptarse a otros objetivos de forma dinámica, lo que implica que no es inteligente. Por otro lado, se puede programar un robot que decida sobre sus acciones en función de su estado, sus capacidades y sus objetivos, siendo en este caso inteligente. Entonces, ¿cómo se puede programar un robot para que tome decisiones? Una solución es que busque, entre todos los estados posibles que representa el mundo, aquel estado que le permita alcanzar un estado objetivo a partir del estado actual (Poole & Mackworth, 2010; Slaney & Thiébaux, 2001).

Existen problemas de IA que pueden ser resueltos mediante búsqueda. En ellos se define un **espacio de estados** y unos **operadores** que permiten la transición entre estados, partiendo de un **estado inicial** y dirigiéndose a un **estado objetivo**. Entonces, el problema a resolver es encontrar el camino que lleve del estado inicial al estado objetivo a través del espacio de estados disponible. En cada estado se pueden aplicar una serie de operaciones para llegar a otro estado distinto, que puede ser el final o no.

En la **resolución de problemas mediante búsqueda** se ha de aplicar una **estrategia de control** que permita encontrar un **camino desde el estado inicial al objetivo**, lo que implica examinar posibles secuencias de acciones y los estados que provocan, seleccionando aquella secuencia que sea la mejor según un determinado criterio.

En la Figura 1 se representa un problema sencillo de búsqueda mediante un grafo. E_i es el estado inicial y E_o es el estado objetivo. Mediante la aplicación de operadores

de estado se cambiará de estados (transición representada por flechas en la Figura 1) y el orden en que se apliquen esos operadores vendrá determinado por el algoritmo de búsqueda.

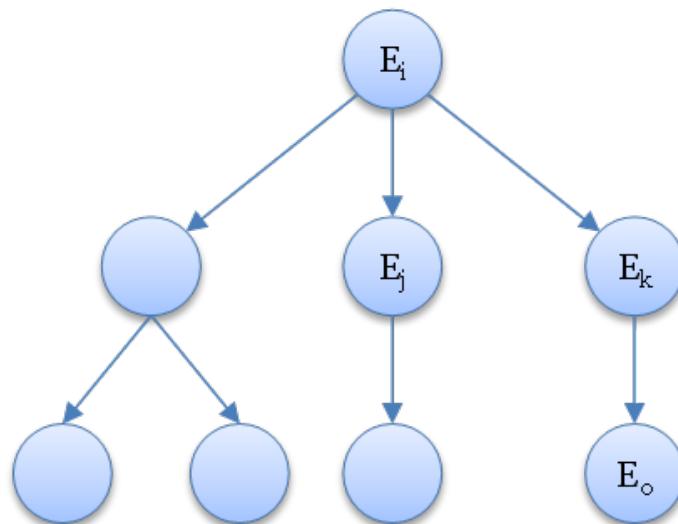


Figura 1. Búsqueda de estados.

Para resolver un problema mediante búsqueda deben definirse en primer lugar los componentes que lo forman:

- ▶ **Descriptores de estado:** estructura simbólica capaz de representar cada estado.
- ▶ **Descriptores de operación:** herramientas computacionales capaces de transformar la representación del estado para poder inspeccionar sistemáticamente el espacio de estados candidatos.
- ▶ **Descriptores de algoritmo de búsqueda:** orden en que se aplican los operadores a los diferentes estados para llegar al estado objetivo. El algoritmo de búsqueda ha de ser un método efectivo de organizar las transformaciones entre estados para así obtener el objeto deseado tan pronto como sea posible.

Este tipo de búsqueda se aplica, entre otros ámbitos, en la búsqueda en juegos. Por ejemplo, en el juego de ajedrez cada contrincante realiza una búsqueda entre todas las jugadas que puede realizar en cada instante para realizar la jugada que le sea lo más favorable posible respecto al estado en el que se encuentra. En este caso las reglas del juego marcan las posibles operaciones de transición entre estados.

La investigación en robótica es uno de los ámbitos en los que se han utilizado técnicas de búsqueda de estados para la resolución de problemas. Programar un robot consiste en integrar varias funciones, entre las que se incluye la percepción del entorno que le rodea a través de sensores, formulación de planes de actuación y la realización de esos planes mediante actuadores. Un controlador recibirá los estímulos, procesará datos, tomará decisiones y enviará los comandos correspondientes a los actuadores que los convertirán en acciones.

A continuación, se muestra un ejemplo referido específicamente a la generación de planes de actuación que se utiliza típicamente para ilustrar la resolución de este tipo de problemas mediante búsqueda. El problema se denomina «Mundo de bloques» y en él se plantea un robot que tiene un repertorio finito de acciones que puede ejecutar para alcanzar un objetivo. El robot dispone de un brazo móvil capaz de cambiar de sitio a unos cuantos bloques situados sobre una mesa. El entorno del robot estará formado por la descripción de los distintos estados por los que atraviesa el mundo de bloques tras las actuaciones del robot sobre él.

El planteamiento sigue el modelo propuesto para el planificador automático utilizado en robótica STRIPS (*Stanford Research Institute Problem Solver*), utilizando un formalismo bastante intuitivo basado en lógica para representar estados y operadores (Fikes & Nilsson, 1971).

Partiendo de la situación inicial mostrada en la Figura 2:

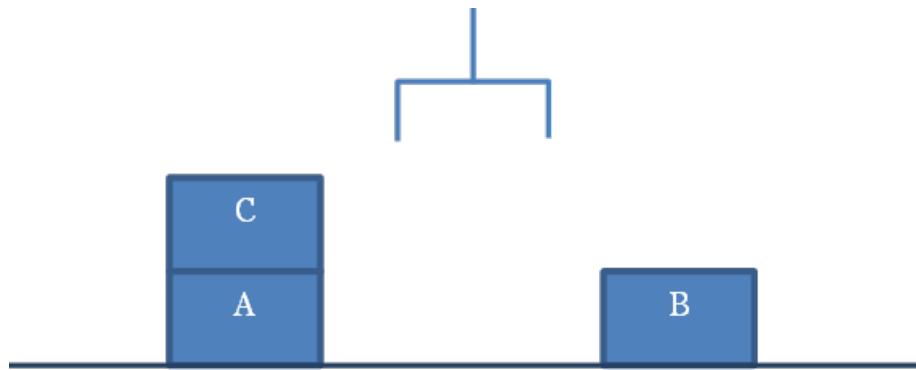


Figura 2. Estado inicial del ejemplo basado en «Mundo de los bloques».

Esta situación de partida se puede describir mediante el conjunto de fórmulas lógicas:

SOBRE(C, A)

SOBRE_LA_MESA(A)

LIBRE(C)

MANO_LIBRE

SOBRE_LA_MESA(B)

LIBRE(B)

Para elaborar el plan de actuaciones, se ha de conocer el objetivo al que se desea llegar. En este caso es que todos los bloques estén uno sobre otro tal y como se muestra en la Figura 3.

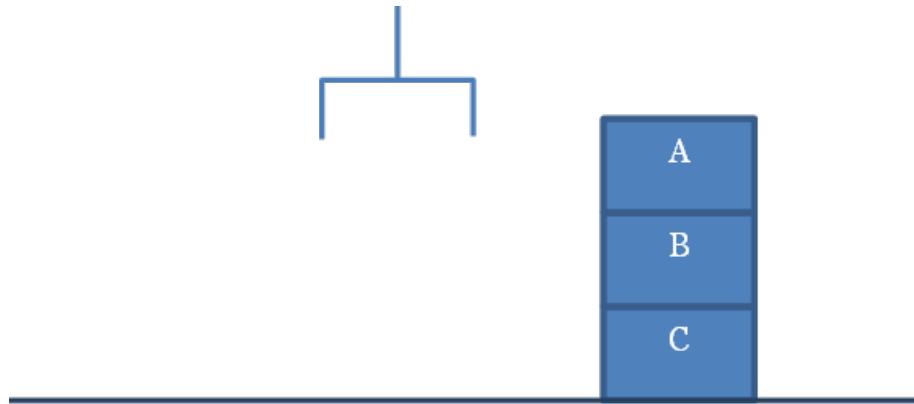


Figura 3. Estado objetivo de «Mundo de los bloques».

Por tanto, el objetivo es obtener una descripción del estado del entorno del robot dado por la conjunción de las fórmulas:

`MANO_LIBRE`

`SOBRE(B,C)`

`SOBRE(A,B)`

`SOBRE_LA_MESA(C)`

Para ello se han de aplicar operadores STRIPS que representan una acción mediante tres componentes:

- ▶ **Precondiciones (P)**: prerequisitos que han de ser ciertos en la descripción del entorno para la realización de la acción.
- ▶ **Borrar (B)**: fórmulas que se han de borrar de la descripción del entorno una vez aplicada la acción ya que dejan de ser ciertas.
- ▶ **Añadir (A)**: fórmulas que se han de añadir a la descripción del entorno una vez aplicada la acción.

El modelo se completa con las reglas que simulan las acciones posibles del robot y que este tiene programadas, tales como:

COGER(X)

P: LIBRE(X), MANO_LIBRE

B: LIBRE(X), MANO_LIBRE

A: COGIENDO(X)

POSAR(X)

P: COGIENDO(X)

B: COGIENDO(X)

A: SOBRE_LA_MESA(X), LIBRE(X), MANO_LIBRE

APIALAR(X,Y)

P: COGIENDO(X), LIBRE(Y)

B: COGIENDO(X), LIBRE(Y)

A: MANO_LIBRE, SOBRE(X,Y), LIBRE(X)

DESAPILAR(X,Y)

P: MANO_LIBRE, LIBRE(X), SOBRE(X,Y)

B: MANO_LIBRE, LIBRE(X), SOBRE(X,Y)

A: COGIENDO(X), LIBRE(Y)

COGER(X) corresponde a la acción de coger el bloque X de la mesa con el brazo, POSAR(X) se refiere a posar el bloque X sobre la mesa, APIALAR(X,Y) es situar el bloque X sobre el bloque Y, y DESAPILAR(X,Y) se refiere a coger el bloque X que está situado sobre el bloque Y.

Estas acciones constan de tres partes. La primera P son las precondiciones que se han de dar para la aplicación de la acción, la segunda B y la tercera A son las fórmulas que se han de borrar y añadir, respectivamente, a la descripción del entorno si esa acción se ejecuta (es, por tanto, parte de la consecuencia de la acción).

La planificación de las acciones del robot se puede entonces obtener mediante la aplicación de algún procedimiento de búsqueda entre las descripciones posibles que resultan de ejecutar distintas acciones que transformen el entorno desde la situación de partida hasta el estado final u objetivo. La solución es el siguiente plan de acciones:

DESAPILAR(C,A)

POSAR(C)

COGER(B)

APIALAR(B,C)

COGER(A)

APIALAR(A,B)

Las siguientes secciones abordan diferentes procedimientos de búsqueda que permiten resolver este tipo de problemas de búsqueda de estados.

9.3. Dirección de la búsqueda

El objetivo del proceso de búsqueda se resume en encontrar un camino a través del conjunto de estados entre el estado o estados iniciales y el estado o los estados finales u objetivos. Esta búsqueda puede hacerse en dos direcciones:

- ▶ **Hacia adelante** (del inicio al fin).
- ▶ **Hacia atrás** (del fin al inicio).

Existen modelos de búsqueda que permiten que estos procedimientos sean simétricos. En este caso cualquiera de estas estrategias de búsqueda tiene validez.

Búsqueda hacia adelante

Se trata de una búsqueda guiada por los datos, aplicando información disponible para obtener nueva información. Por ejemplo, si:

el objetivo es conocer C :

A [0]

A → B [1]

B → C [2]

Conocido A y sabiendo que si se cumple A se cumple B , se busca cambiar del estado [0] al estado [1] para conocer B . Una vez conocido B se llega al estado [2] porque si se cumple B se cumple C y se alcanza así el objetivo de conocer C .

Búsqueda hacia atrás

Se parte del objetivo hasta llegar al estado inicial. Por ejemplo:

A [0]

A → B [1]

B → C [2]

Sabiendo cuál es el objetivo, conocer C , hace falta encontrar B , saber si B es cierto ya que, de acuerdo con [2] , si B es cierto, entonces C es cierto.

C

→ [2]

B

→ [1]

A

Hay por tanto que encontrar A puesto que se sabe que si A es cierto ([0]), B es cierto ([1]) y, por tanto, C es cierto ([2]).

¿Qué método de búsqueda se debe aplicar?

Existen algunos criterios para seleccionar el mecanismo de búsqueda. A continuación, se describen tres de ellos:

- ▶ **Número de estados iniciales y finales:** normalmente es más conveniente ir del conjunto con menor número de elementos al de mayor número de elementos. Por ejemplo, en el ejemplo bien conocido del llamado «problema de la lechera»:

«Se dispone de dos medidores de líquidos; uno de 5 litros y el otro de 7. Inicialmente ambos están vacíos y, en cualquier momento pueden ser llenados o vaciados usando un depósito de leche con suficiente capacidad. La leche también puede ser traspasada de uno a otro medidor hasta que el primero se vacíe o hasta que el segundo se llene. El problema consiste en encontrar la secuencia de acciones que se deben seguir para que en el mayor de los medidores se disponga exactamente de 4 litros».

Se conoce el estado inicial, con ambas jarras vacías, y el conjunto de estados finales: todos los estados posibles que contengan 4 litros en el mayor de los medidores: 6 estados. Parece, por tanto, más sencillo buscar desde el estado inicial alguno de los 6 posibles estados finales. Si se realiza la búsqueda hacia atrás no se sabría desde qué estado final es más fácil encontrar una solución; incluso en algunos casos desde algún estado final no es accesible el estado inicial. En este ejemplo, por tanto, se debería iniciar una búsqueda en paralelo desde todos los nodos finales si quisiéramos adoptar una estrategia de búsqueda hacia atrás.

- ▶ **Factor de ramificación** (*branching factor*): el criterio es caminar en el sentido en que el factor de ramificación sea más pequeño, esto es, el número medio de estados accesibles directamente desde cada nodo en la dirección de búsqueda sea menor. Un posible ejemplo podría ser la resolución del problema de ir desde casa a un lugar desconocido. Dado que un propietario conoce los alrededores de su casa, si busca un camino hacia atrás (de lo desconocido hacia su casa) resultará más fácil que lo contrario, ya que es de suponer que encontrará cruces en los cuales sabrá por dónde volver con más facilidad a su casa que si hiciera el recorrido inverso, pretendiendo en los cruces tratar de llegar al lugar desconocido.

- ▶ **La necesidad de justificar los pasos que se siguen en el razonamiento** . Si existe esta necesidad, es importante proceder en la dirección que más se aproxime al modo de pensar del futuro usuario del sistema. Este es el caso típico de los sistemas expertos de diagnóstico y propuesta de soluciones que siguen un método de razonamiento hacia atrás. Por ejemplo, un ingeniero ante un programa de detección de fallos y diagnóstico siempre querrá justificar los pasos seguidos en el razonamiento, razonando hacia atrás hasta determinar las causas de los fallos. Razonando hacia atrás el sistema podrá responder a preguntas del tipo «¿Por qué debo hacer esta comprobación?» respondiendo «porque eso ayudaría a determinar si el problema es debido a un fallo eléctrico o un fallo de *software*».

El problema que existe con este tipo de criterios tan generales es que no siempre son aplicables. Por ejemplo, en el primer caso, no siempre se sabe cuál es el número de estados finales. Por otra parte, el factor de ramificación es normalmente difícil de aproximar.

**Por último, también debe tenerse en cuenta que existe una tercera
posibilidad y es usar los dos tipos de razonamiento, hacia adelante o
hacia atrás, de forma combinada.**

Otro factor que permite clasificar los algoritmos de búsqueda es la aplicación de la información disponible sobre el problema para su resolución.

Así, los algoritmos de búsqueda se pueden dividir en dos grandes grupos, atendiendo al papel que juega el conocimiento en la estrategia de control de un sistema de búsqueda:

- ▶ **Búsqueda exhaustiva o a ciegas:** no se tiene en cuenta la posible localización del objetivo. Estos algoritmos no utilizan ninguna información del problema e ignoran hacia dónde se dirigen hasta que encuentran el objetivo.
- ▶ **Búsqueda heurística:** se tiene una estimación de lo que falta para llegar al estado objetivo. Los algoritmos utilizan información del problema para localizar el objetivo.

Las siguientes secciones explican estos dos grandes grupos de algoritmos de búsqueda.

9.4. Búsqueda exhaustiva o a ciegas

Búsqueda a ciegas

Consiste en generar todos los estados posibles hasta encontrar la solución. Para generar **todos los estados** se aplica una **regla de forma sistemática**, sin tener en cuenta ninguna información del problema, ni del objetivo que se busca.

Existen varios tipos de algoritmos de búsqueda a ciegas, tal y como se describe en las siguientes subsecciones.

Búsqueda en amplitud

En la búsqueda de estados se va generando un grafo como puede ser una malla o un árbol de estados como los mostrados en la Figura 1 o en la Figura 4. Si la búsqueda es en amplitud, a partir de un nodo se generan todos los hijos y, para cada hijo, todos los nietos, y así sucesivamente.

Por tanto, **cada nivel del árbol de estados se desarrolla completamente antes de desarrollar el siguiente nivel**. Según los nodos de un nivel se generan, si no se ha llegado a la solución, se va al siguiente nivel para generar los nodos siguientes. Así, hasta encontrar el objetivo.

Esta estrategia asigna mayor prioridad a los nodos con menor profundidad, generando los nodos por niveles tal y como se muestra en la Figura 4, en la que los números indican el orden en que se van generando los nodos:

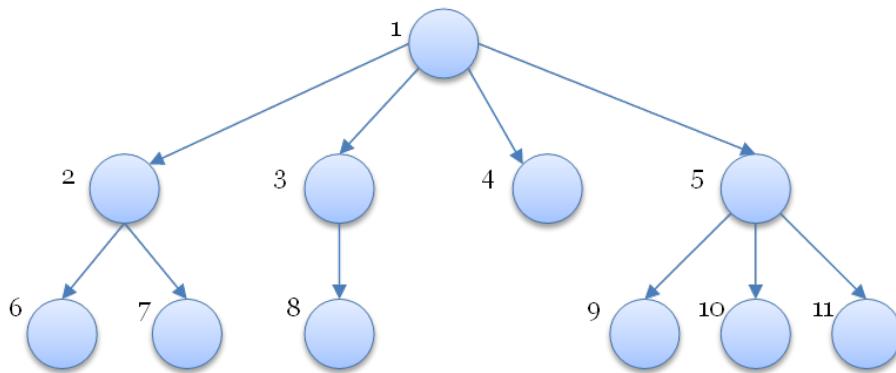


Figura 4. Ejemplo de búsqueda en amplitud.

Así, se hace un recorrido uniforme del grafo que le da un carácter «lento pero seguro». En este algoritmo **siempre se expande primero el nodo menos recientemente creado, siguiendo una estrategia FIFO (First-In, First-Out)**.

Cuando los costes de ir de un nodo a otro son uniformes, se garantiza que se encontrará la solución con el camino más corto, esto es, la secuencia con menor número de estados hasta el estado objetivo. Sin embargo, si el coste no es uniforme, no se podrá garantizar que el camino sea el más corto.

El **problema** es la gran cantidad de memoria necesaria y la lentitud en encontrar la solución si esta se encuentra a gran profundidad. Aunque algorítmicamente se puede considerar una búsqueda perfecta (siempre encuentra solución óptima), no lo es desde el punto de vista de la IA, ya que es inflexible frente a cualquier tipo de conocimiento que se tenga sobre el problema.

Búsqueda en profundidad

Esta estrategia **da prioridad a los nodos más profundos en el grafo de búsqueda** de tal forma que estos se expanden o desarrollan, para generar posibles soluciones, antes que otros nodos menos profundos, tal y como se muestra en la Figura 5.

Tras cada expansión o desarrollo de un nodo, de los hijos que acaban de ser generados se selecciona uno para ser de nuevo desarrollado. Esta exploración hacia

abajo se desarrolla hasta que, por alguna razón, el proceso queda bloqueado. En tal caso, el proceso continúa desde el nodo más profundo que haya sido dejado atrás sin explorar. Dicho de otro modo, desde el último punto en el que se tomó una decisión dejando alguna alternativa sin explorar.

De esta forma, se toma un camino y se sigue hasta que se encuentra la solución o un callejón sin salida, en cuyo caso se vuelve hacia atrás.

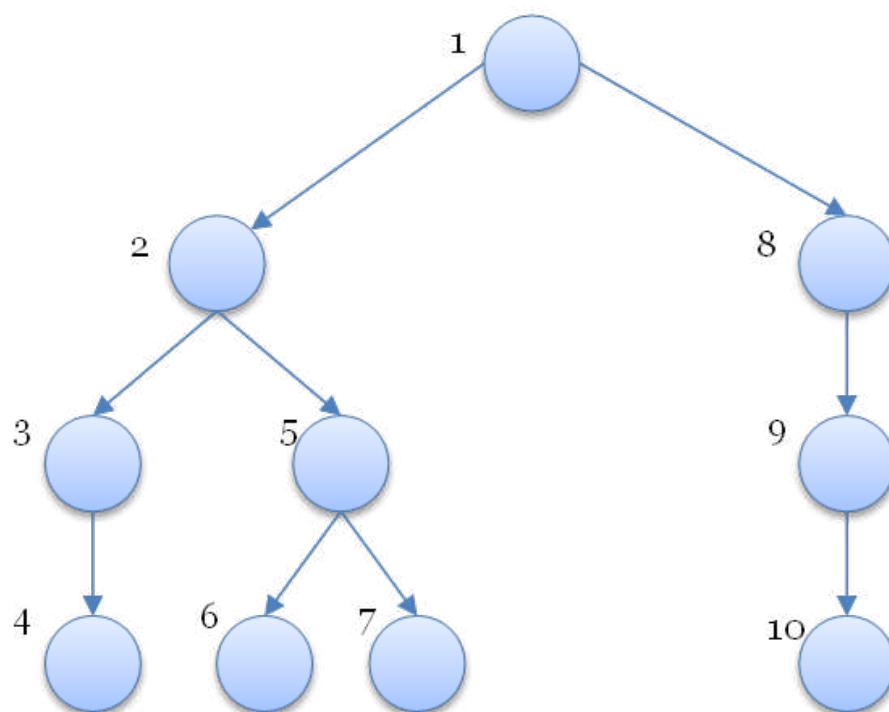


Figura 5. Ejemplo de búsqueda en profundidad.

En este algoritmo siempre se expande el **nodo más recientemente creado, siguiendo la estrategia LIFO (Last-In, First-Out)**.

El **problema** de este algoritmo es que no garantiza que se encuentre la solución por el camino más corto. Además, podría suceder que una solución se encuentre a poca profundidad, pero no se encuentre pronto porque previamente se han explorado caminos muy profundos. Sin embargo, tiene la **ventaja** de que si la tasa de

generación de hijos es muy alta, se mitiga el hecho de poder quedarse sin memoria, tal y como es más probable que pase en la búsqueda en amplitud.

Esta estrategia funciona bien cuando abundan las soluciones en el grafo y todas ellas son igualmente deseables (no conduce más que de casualidad a una solución óptima).

También puede resultar adecuada cuando existen medios que nos pueden indicar pronto el que se ha elegido una dirección equivocada con lo cual podremos bloquear la búsqueda en este camino.

Los dos algoritmos (búsqueda en amplitud y en profundidad) funcionan bien en grafos y en árboles; aunque con árboles en búsqueda en profundidad se puede entrar en un camino infinito. Además, estos dos algoritmos son algoritmos de fuerza bruta, se va de una rama a otra sin tener en cuenta ninguna información sobre el problema.

Búsqueda en profundidad acotada

En vez de coger un camino hasta el final, se aplica búsqueda en profundidad hasta un nivel y, si con esa profundidad no se consigue el objetivo, se realiza búsqueda en profundidad a partir de un nivel superior. Así, se evita centrarse sólo en una parte del árbol.

Se define una profundidad límite más allá de la cual no se buscará. Esto evita que el proceso no pare en grafos infinitos (o en los que haya una rama infinita).

Búsqueda en profundidad iterativa

Se busca con profundidad acotada, partiendo de una cota inicial e incrementándola en uno hasta que se llegue a la solución.

Con esta forma de búsqueda, se intenta combinar las ventajas de búsqueda en anchura y búsqueda en profundidad. Siempre se va a encontrar el camino más corto y no se necesita tener todos los nodos en memoria.

Los algoritmos de búsqueda a ciegas no se pueden aplicar siempre, pero en el caso de que se puedan aplicar dan muy buenos resultados.

A continuación, se expone un ejemplo para ilustrar el funcionamiento de los algoritmos de búsqueda a ciegas. Dado el árbol que se muestra en la Figura 6, se representa en las Figuras 7, 8 y 9 los órdenes en que se expanden los nodos en función del tipo de búsqueda. Nótese que para la búsqueda en profundidad iterativa la cota inicial de profundidad tiene un valor de 0, por lo cual se expande únicamente el nodo raíz. En el siguiente paso, la profundidad tiene un valor de 1, con lo que se extiende el nodo raíz y sus hijos.

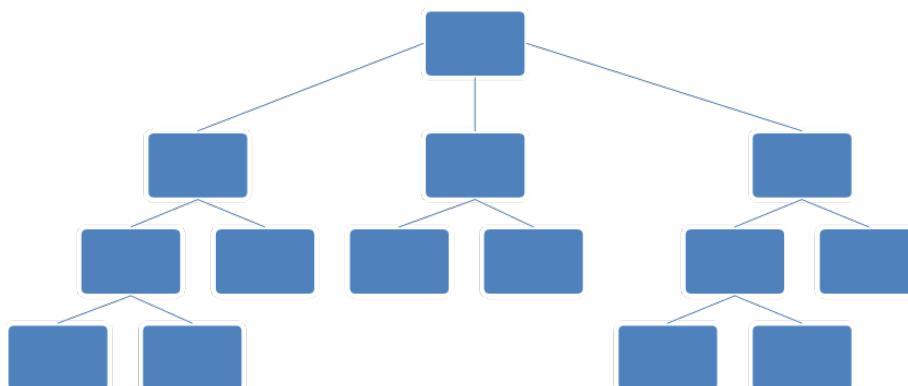


Figura 6. Árbol de ejemplo.

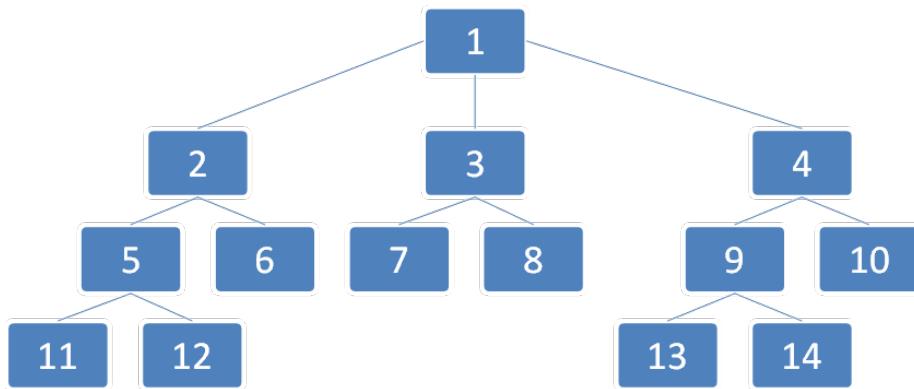


Figura 7. Orden en que se expanden los nodos si se aplica búsqueda en anchura.

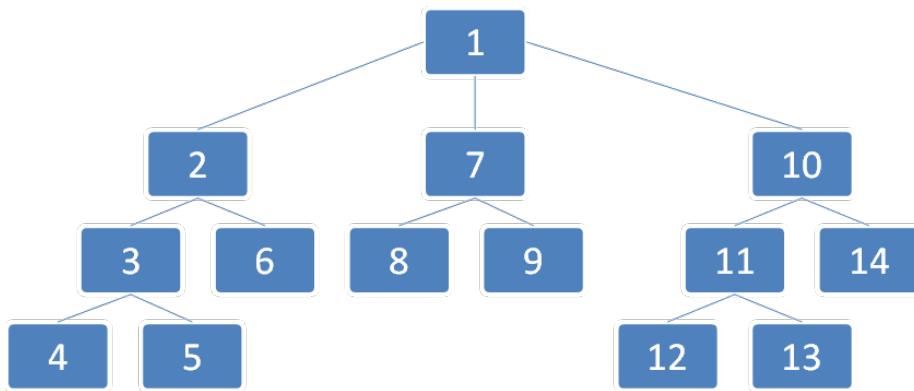


Figura 8. Orden en que se expanden los nodos si se aplica búsqueda en profundidad.

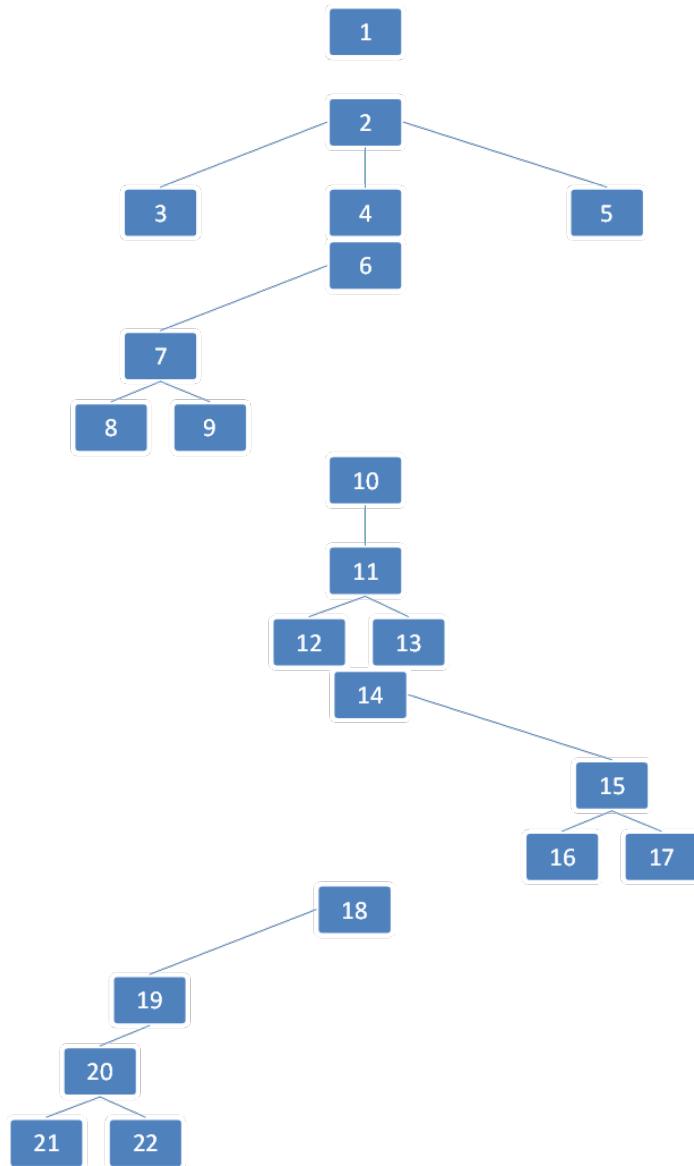


Figura 9. Orden en que se expanden los nodos si se aplica búsqueda en profundidad iterativa.

9.5. Búsqueda heurística

Métodos de búsqueda a ciegas

Son métodos exhaustivos para encontrar caminos a un estado objetivo. En principio proporcionan **una solución al problema** de encontrar el camino, pero **a menudo no es viable** su uso porque la búsqueda expandirá demasiados nodos antes de encontrar un camino.

Para muchas tareas es posible declarar principios o reglas que ayudan a reducir la búsqueda. Cualquier técnica usada para acelerar la búsqueda depende de la información disponible sobre el problema representado por el grafo.

En los métodos de búsqueda heurística se aplica información del problema para encontrar la solución. A esta información disponible se le denomina **información heurística**, mientras que los procedimientos de búsqueda usados son denominados **métodos de búsqueda heurística**.

Estos algoritmos tienen en cuenta que un estado generado puede no ser solución, pero podría estar más cerca de ella. En ese caso la búsqueda se seguiría desde este nodo en adelante. Para aplicar este método de búsqueda se emplea una función denominada **heurístico**, que mide la proximidad de los nodos al objetivo.

Heurístico (regla heurística o método heurístico)

Es una regla de estrategia y simplificación que **limita drásticamente** la **búsqueda de soluciones** en grandes espacios de problemas.

Se enumeran a continuación algunos aspectos en los que la información heurística puede ser aplicada para expandir un árbol de manera eficiente:

- ▶ Decidir qué nodo debe ser expandido o desarrollado en todas sus posibilidades en lugar de tener un criterio puramente igualitario de la potencia de los nodos candidatos.
- ▶ Decidir qué sucesor o sucesores deben ser considerados en primer lugar en lugar de considerar todos los posibles sucesores a la vez.
- ▶ Decidir qué nodos deben ser descartados o podados del espacio de búsqueda.

El objetivo de la búsqueda heurística es, por tanto, reducir el número de nodos examinados y la efectividad depende del **heurístico seleccionado**.

Sin embargo, se da el hecho de que no es fácil seleccionar el heurístico.

A continuación, se muestra un ejemplo basado en el conocido problema del «Puzzle 8» en el que se proponen dos posibles heurísticos.

La solución de este rompecabezas consiste en, a partir de una configuración inicial dada, reordenar ocho elementos numerados móviles situados en un tablero de 3×3 para conseguir una configuración final dada (ver Figura 10).



Figura 10. Problema «Puzzle 8».

Los heurísticos propuestos son:

- ▶ h1: número de fichas mal colocadas, mientras mayor sea este valor, peor.
- ▶ h2: distancia Manhattan: la suma de las distancias de las fichas mal colocadas. La distancia se mide como el número de movimientos necesarios para situar la ficha en la posición final deseada.

La Figura 11 muestra los valores de estos heurísticos para diversas configuraciones según la ficha que se mueva desde la posición inicial mostrada en la Figura 10.

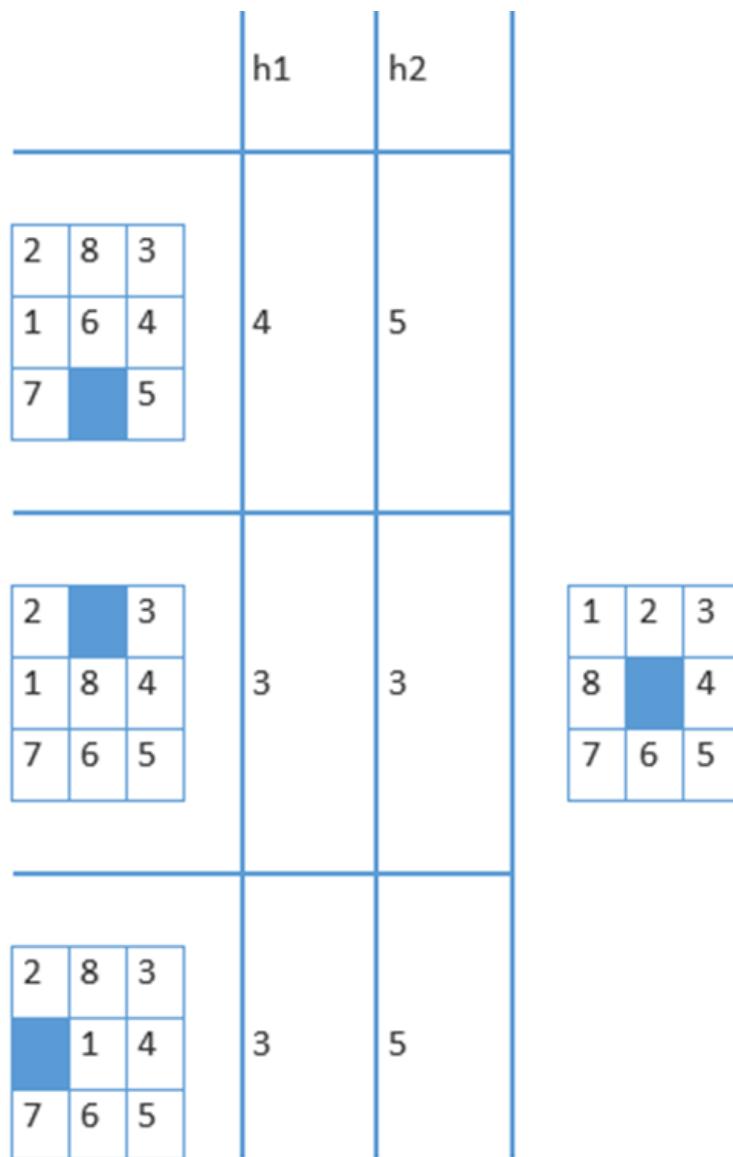


Figura 11. Valores de heurísticos para diversas configuraciones del puzzle 8.

Se ve en este ejemplo sencillo cómo el heurístico que se selecciona determina la rapidez en conseguir el objetivo. El movimiento adecuado es el segundo y, por tanto, el heurístico 2 en este ejemplo parece más adecuado.

Este caso es muy sencillo, pero, en general, no es fácil definir el heurístico adecuado.

Más aún, los heurísticos no garantizan la obtención de soluciones óptimas; de hecho, no garantizan ningún tipo de solución: se estima si un nodo está más cerca que otro para llegar a la solución, aunque como es una estimación podría dar lugar a una equivocación.

En conclusión, todo lo que se puede decir de un heurístico útil es que ofrece soluciones que son suficientemente buenas la mayoría de las veces.

Escalada simple o hill climbing

La interpretación intuitiva más sencilla y de la que toma el nombre el método de búsqueda «escalada simple» consiste en considerar el proceso como la escalada a una montaña: el objetivo es alcanzar la cima y el estado de búsqueda es el punto donde se encuentra el escalador. La regla de la que se dispone es seguir escalando hasta que se llegue a un punto a partir del cual no se pueda ascender más.

Mediante este método en el momento que se encuentra un estado más favorable que el actual, se avanza al nuevo estado encontrado.

Para simplificar la explicación se puede considerar montañas de dos dimensiones con lo que la escalada consiste en la búsqueda de un máximo de una función real de variable real: f . El escalador hipotético, si se encuentra en un punto x_0 , si los saltos que puede dar son de amplitud δ , podría dirigirse a $x_0 + \delta$ o quedarse donde está.

Dependiendo de la altura $f(x_0 + \delta)$ que se obtenga al avanzar y la altura actual del escalador $f(x_0)$ se decidirá avanzar a la nueva posición si la situación tras el avance es mejor que la situación sin avanzar, porque la altura se incrementa. Si la altura no se incrementa al avanzar, se da como solución la posición x_0 (ver Figura 12).

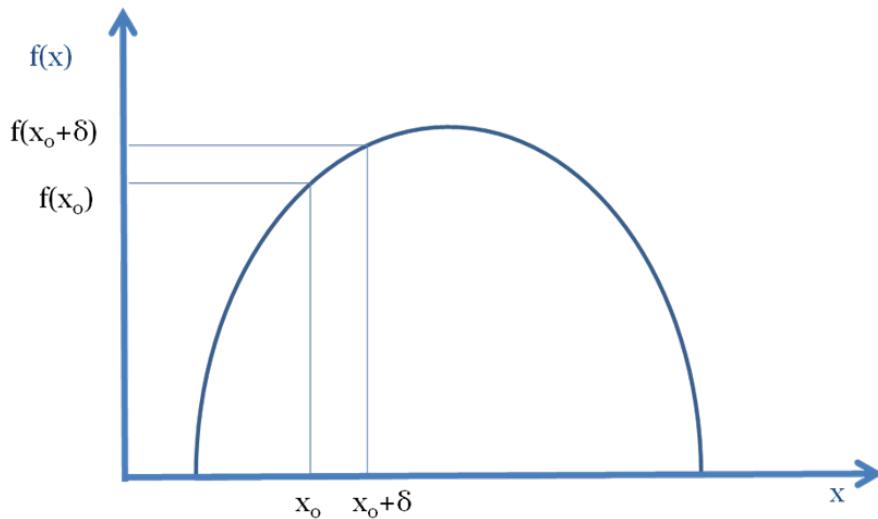


Figura 12. Escalada simple: búsqueda del máximo de una función.

Este método, presenta problemas de máximos locales y/o mesetas como intuitivamente se puede concluir observando las funciones representadas en la Figura 13.

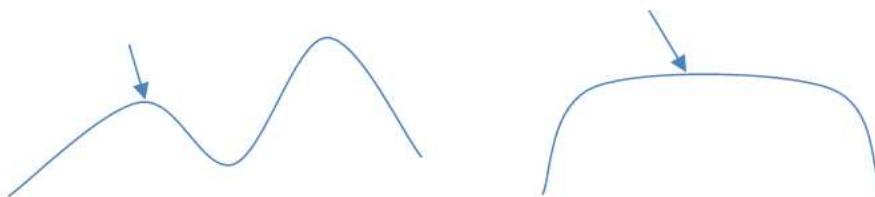


Figura 13. Máximo local (función de la izquierda) y meseta (función de la derecha).

Este método resulta interesante para maximizar o minimizar funciones y es el algoritmo que se utiliza para, por ejemplo, sintonizar una emisora de radio. Se van haciendo movimientos del sintonizador de frecuencias en una dirección siempre que la calidad de la señal recibida mejore. Al llegar a un punto donde no mejora en ninguna de las dos direcciones se detiene la búsqueda en el dial.

Escalada por máxima pendiente

En el método de escalada simple, el algoritmo en el momento que encuentra un estado E' más favorable que el actual E , se avanza al estado E' .

Sin embargo, el método de escalada por máxima pendiente consiste en «explorar los alrededores» antes de seguir avanzando por la montaña. Cuando se está en un punto, se da un paso en todas las direcciones y se va por el mejor camino.

Se consideran todos los posibles movimientos a partir del estado actual
y se elige como nuevo estado el mejor de ellos.

En este caso el escalador hipotético para avanzar desde un punto x_0 , si los saltos que puede dar son de amplitud δ , podría dirigirse o bien a $x_0 - \delta$ o bien a $x_0 + \delta$. Dependiendo de la altura que tengan estas posibilidades, $f(x_0 - \delta)$ y $f(x_0 + \delta)$ respectivamente, y la del propio escalador, $f(x_0)$, se decidirá saltar a la mejor opción. Si la situación actual no es mejorable, la posición x_0 se dará como solución. En el caso del ejemplo mostrado en la Figura 14, la decisión será saltar a $x_0 + \delta$.

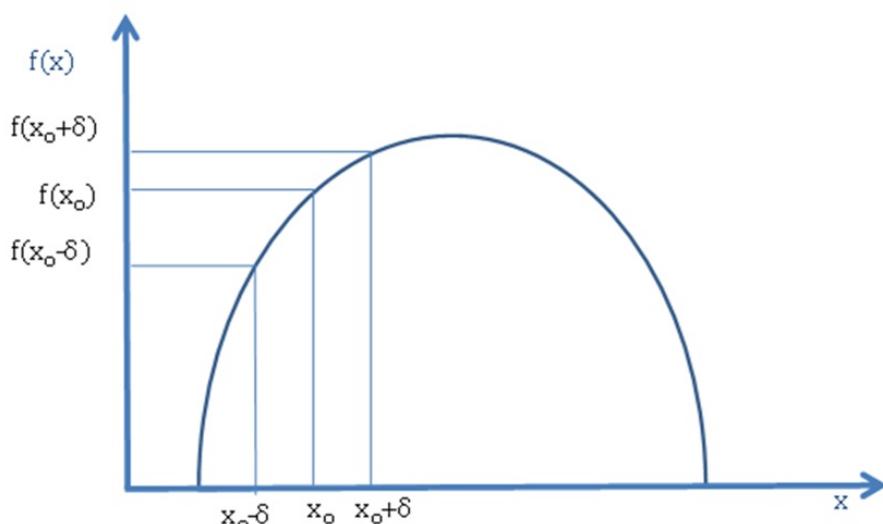


Figura 14. Escalada por máxima pendiente.

Por tanto, mediante este método se exploran todos los posibles siguientes estados y se finaliza cuando se llega a un nodo en el que todos los siguientes estados son peores que el actual. Cuando se salta a la mejor opción, se asume que ese siguiente estado está más cerca del final.

Este algoritmo obtiene una mejor ruta que el método de escalada simple, pero es más lento, porque hay que evaluar todos los siguientes estados posibles y el número de operadores aplicables a un estado puede ser muy grande. Por tanto, puede suponer altos requerimientos de computación y de memoria. Aun así, en general estos algoritmos de escalada suelen ser muy rápidos, aunque, como principal inconveniente, presentan problemas de máximos locales y mesetas.

Búsqueda «mejor el primero» o best-first

Este algoritmo intenta combinar las ventajas de la búsqueda en amplitud y en profundidad.

En cada paso del algoritmo se va a seguir un camino y se va a cambiar de ruta cuando haya una ruta más prometedora.

El funcionamiento es el siguiente: se expande el nodo padre generando nodos hijos. El nodo padre se incluye en la lista de nodos cerrados o ya expandidos, mientras que los nodos hijos se incluyen en la lista de nodos abiertos o no expandidos. Analizando todos los nodos abiertos se comprueba si alguno de los nodos es el objetivo y, si ninguno lo es, **se escoge como siguiente nodo a expandir el que parezca más prometedor**, de acuerdo al valor de una función de evaluación que se esté aplicando. Típicamente esta función se basará en un heurístico que estime la cercanía del nodo al objetivo.

Se muestra, a continuación, un ejemplo para ilustrar los diferentes pasos del

algoritmo y cómo progresan los nodos cerrados y abiertos. Las listas que se muestran en el ejemplo están ordenadas por orden de prioridad, tomando por tanto el primer elemento en la lista como el nodo a expandir. El número asociado a cada nodo indica el valor de la función de evaluación. Un nodo es más prometedor, cuanto más pequeño sea el valor de esta función. El objetivo es P. La Figura 15 muestra el grafo del ejemplo y los diferentes pasos del algoritmo son los siguientes:

ABTOS = [A5]

CDOS = []

Evaluación A5

ABTOS = [B4, C4, D6]

CDOS = [A5]

Evaluación B4

ABTOS = [C4, E5, F5, D6]

CDOS = [B4, A5]

Evaluación C4

ABTOS = [H3, G4, E5, F5, D6]

CDOS = [C4, B4, A5]

Evaluación H3

ABTOS = [O2, P3, G4, E5, F5, D6]

CDOS = [H3, C4, B4, A5]

Evaluación O2

ABTOS = [P3, G4, E5, F5, D6]

CDOS = [02, H3, C4, B4, A5]

Evaluación P3

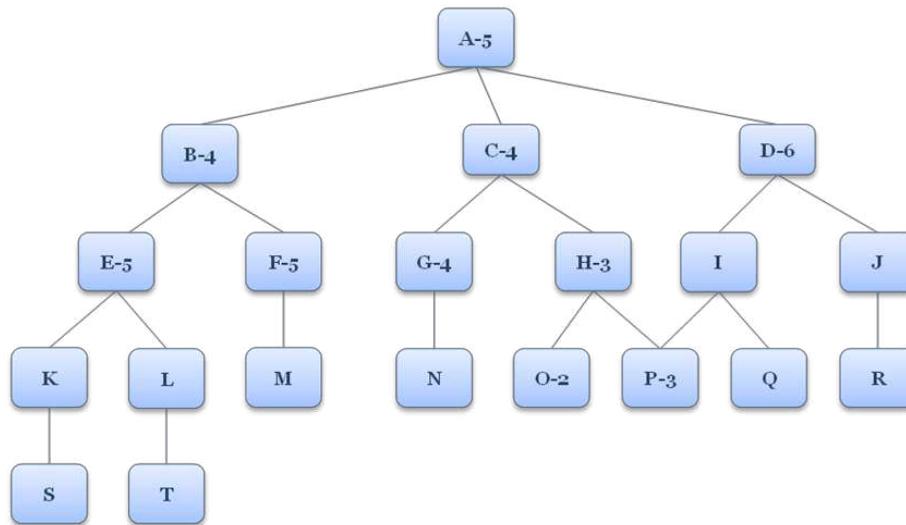


Figura 15. Grafo del ejemplo de búsqueda «primero el mejor».

El problema que presenta este algoritmo es que hay que mantener todos los nodos abiertos (o no expandidos) en la memoria de trabajo. Además, el heurístico puede fallar, como es el caso de la evaluación del nodo O en el ejemplo anterior que no se acerca precisamente al objetivo, aunque el algoritmo es lo suficientemente bueno como para recuperarse del fallo. Por otra parte, este algoritmo tiene la ventaja de valer tanto para árboles como para grafos.

Algoritmo A*

El algoritmo A* se puede considerar una particularización del algoritmo de búsqueda «mejor el primero» en que la función de evaluación $f(n)$ se define como sigue:

Para cada nodo n,

$$f(n) = h(n) + g(n)$$

siendo:

- ▶ $h(n)$: heurístico, coste estimado del camino más corto del nodo actual al nodo objetivo ($h(n)=0$ si n es un nodo objetivo).

$n \rightarrow n_O$

- ▶ $g(n)$: coste real del camino conocido más corto desde el nodo origen al nodo n a evaluar.

$n_i \rightarrow n$

Por lo tanto, **la función de evaluación $f(n)$ es el coste del camino más corto desde el nodo inicial al nodo objetivo condicionado a pasar por el nodo n** . Cabe comentar que la definición de $h(n)$ es una tarea complicada y determinará el rendimiento adecuado de la aplicación de este algoritmo.

El algoritmo seleccionará como nodo a expandir el que tenga un menor valor de $f(n)$. Este algoritmo tiene en cuenta por tanto la medida de la profundidad, utilizando una estimación del coste del camino total.

Si se define $h(n) = 0$, se está realizando una búsqueda en anchura, que siempre da lugar al camino más corto, pero cuyo coste computacional es muy elevado.

Este algoritmo impone una **restricción** para alcanzar el objetivo por el camino de menor coste: la función h nunca ha de sobreestimar el coste para alcanzar el objetivo. Una función h de este tipo es una **heurística admisible**.

A continuación, se utiliza de nuevo el problema del «Puzzle 8» para ilustrar lo que no

se considera una heurística admisible. En la Figura 16, la configuración *F* es la final deseada, la configuración *I* es la inicial y la configuración *A* representa un posible movimiento de una ficha desde la configuración *I*.

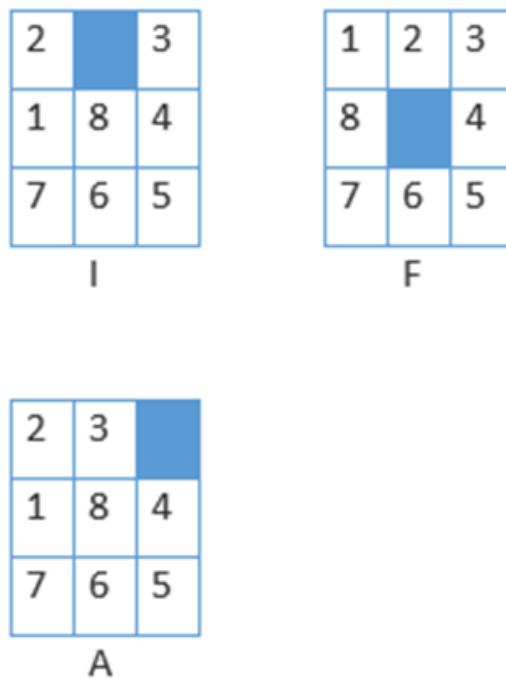


Figura 16. Ejemplo de problema del «Puzzle 8».

Para analizar si la configuración *A* está más cerca del objetivo *F*, se consideran los heurísticos siguientes:

- ▶ $h_1(n)$ = número de fichas mal colocadas.
- ▶ $h_2(n)$ = distancia Manhattan.
- ▶ $h_3(n) = h_2(n) + 3 \cdot S(n)$.

$S(n)$ es la cuenta de secuencia obtenida recorriendo sucesivamente las casillas no centrales y anotando dos puntos en la cuenta por cada ficha no seguida por su ficha sucesora y 0 puntos para las otras; si hay una ficha en el centro se anota 1. Por ejemplo, este cálculo, para la posición inicial sería:

$$\triangleright h_3(\text{posición inicial}) = 3 + 3(2 + 2 + 1) = 18$$

En la Tabla 1 se muestran los valores de estos heurísticos para las configuraciones I y A :

	Inicio I	A
h_1	3	4
h_2	3	4
h_3	18	19

Tabla 1. Valores de los heurísticos propuestos.

Los dos primeros heurísticos son admisibles, mientras que $h_3(n)$ no es admisible, dado que es mayor al número de movimientos que hay que realizar para llegar a la solución F .

9.6. Búsqueda en juegos

Búsqueda en juegos

Consiste en determinar el conjunto de jugadas que permiten ganar al oponente.

Las búsquedas más sencillas son las relativas a problemas de búsqueda de estrategias ganadoras en juegos de dos jugadores con movimientos alternados y en los que no interviene el azar. Concretamente, son juegos de los llamados de **información completa y suma nula**:

- ▶ La **información completa** se refiere al hecho de que cada jugador conoce las jugadas que se hicieron y todas las que se pueden hacer en cada momento y por cada jugador.
- ▶ La **suma nula** se refiere a juegos en los que las situaciones finales son o de victoria para uno de los jugadores (y, por tanto, derrota para el otro) o empate; se excluyen, por tanto, juegos en los que puedan perder o ganar a la vez los dos jugadores.

Juegos típicos de este tipo son el ajedrez, las damas, el tres-en-rayo, etc.

Las partidas posibles que se pueden realizar en estos juegos se pueden describir mediante árboles o grafos en los cuales las hojas corresponden a situaciones de victoria para alguno de los jugadores, de derrota o tablas. Los nodos intermedios representan situaciones donde debe mover uno u otro jugador.

Se pueden diferenciar dos tipos de juegos:

- ▶ Juegos donde es posible desarrollar todo el espacio de estados.
- ▶ Juegos donde no es posible desarrollar todo el espacio de estado, porque es muy grande. En este caso se aplican heurísticos para estimar si una jugada es buena o no.

Como ejemplo de la búsqueda en juegos se explica el algoritmo sencillo **Minimax**.

Se consideran dos jugadores: MIN y MAX y el objetivo es que gane MAX. MAX intentará maximizar su ventaja y minimizar la de MIN.

Cada nivel del árbol va a estar formado por nodos MAX o MIN, según a quien le toque jugar. Se supone que MIN tiene la misma información e inteligencia que MAX. Además, MIN intentará hacer lo peor para MAX. El algoritmo Minimax elegirá el mejor movimiento para MAX suponiendo que el contrincante elegirá lo peor para MAX. Una vez determinado el grafo de todas las posibles jugadas, el proceso a seguir es el siguiente:

- ▶ Asignar a los nodos finales un valor:
 - 1 victoria MAX
 - 0 victoria MIN
- ▶ Propagar las etiquetas o valores de esta función hacia arriba. Si se tiene un nodo MAX se toma el máximo de los hijos y, si se tiene un nodo MIN, se toma el mínimo de los hijos.
- ▶ El camino de victoria se obtiene uniendo el trazado que va por los «1» (ver figura 17).

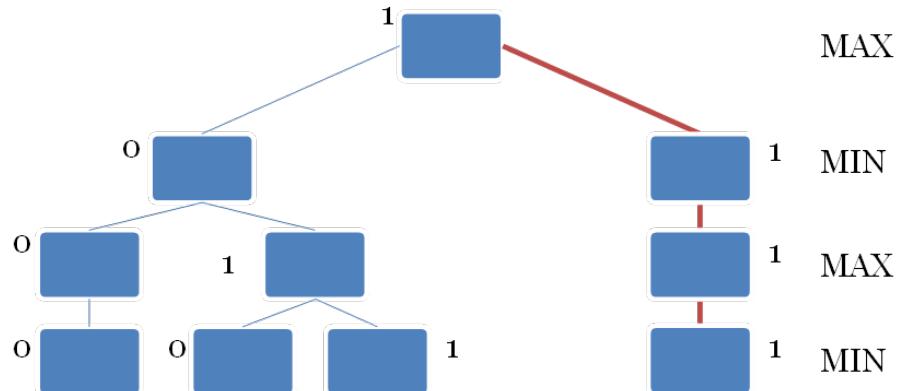


Figura 17. Ejemplo de árbol resultado de ejecutar el algoritmo Minimax.

El procedimiento MINIMAX toma las decisiones óptimas en cada jugada y, si es posible, ganará la partida o, si no, la empatará.

9.7. Costes

Los procedimientos de búsqueda inteligente en **problemas de una cierta magnitud**, como suelen ser los problemas de inteligencia artificial, deben disponer de estrategias de control adecuadas. Esto implica que **se debe guiar la búsqueda** basándose en conocimientos específicos del tema a tratar, tal y como se hace en la búsqueda heurística. El problema radica en saber cómo se maneja y representa el conocimiento para que este sea útil.

En la búsqueda exhaustiva (con información nula) es necesario expandir todos los nodos, lo que conlleva un elevado coste debido a la expansión del árbol. Por otra parte, a medida que se tiene más información (mejor es el heurístico), se expanden solo los nodos necesarios con lo que disminuye el coste debido a la expansión del árbol. Sin embargo, el coste debido a la estrategia de control aumenta a medida que se emplea un heurístico mejor; ya que calcular el heurístico conlleva mucho gasto computacional.

Por lo tanto, el problema del coste se puede resumir en la suma de los costes:

- ▶ Debidos a la estrategia de control.
- ▶ Debidos a la expansión de los nodos del árbol.

Estas consideraciones sobre los costos de la estrategia de control y de la expansión del árbol en un proceso de búsqueda pueden resumirse a través del gráfico de la Figura 18.

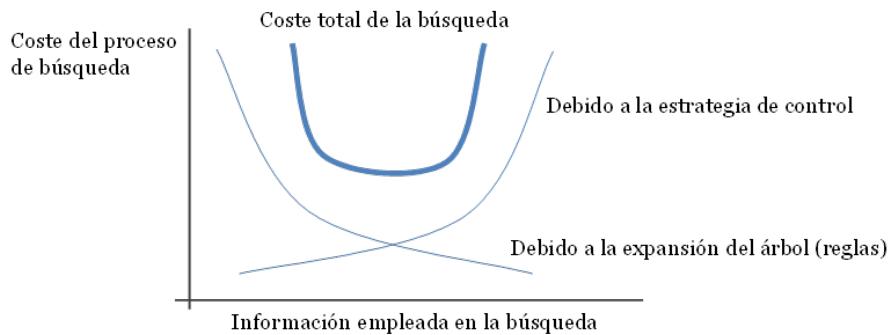


Figura 18. Coste del proceso de búsqueda en función de la información.

Analizando el gráfico anterior, el problema está en encontrar el punto de corte de las gráficas correspondientes al consumo causado por la estrategia de control y al coste debido a la expansión del árbol, ya que marca el mínimo del costo total utilizado en el proceso de resolución del problema.

9.8. Aplicaciones prácticas y ejemplos de implementación

Aplicaciones prácticas en la literatura

Cualquier usuario digital está acostumbrado al uso de aplicaciones como Google Maps o Bing Maps a la hora de encontrar el camino más corto cuando realizan un viaje entre dos puntos, teniendo en consideración las restricciones impuestas por el callejero y el trazado de las vías de comunicación. Los algoritmos más empleados para la planificación de rutas son aquellos basados en el algoritmo de Bellman-Ford, el algoritmo de Floux-Warshall y el algoritmo de Dijkstra, como un ejemplo de algoritmo codicioso (*greedy*) o algoritmo *Shortest Path First* o SPF. Estos algoritmos codiciosos buscan la ruta más corta o, más correctamente, que tienen menor coste para el usuario en términos de combustible, distancia o tarifas. Con el fin de optimizar el coste global, se crearon algoritmos como el A* que incluía un heurístico para combinar el coste de cada paso y el coste global.

Profundizando más en la parte de la algoritmia, actualmente hay muchos algoritmos para encontrar el camino más corto en la literatura. El algoritmo de Dijkstra es el más utilizado para encontrar el camino más corto entre dos vértices (Bozyigit, Alankus, y Nasiboğlu, 2017). El algoritmo encuentra las trayectorias más cortas desde el vértice de la fuente hasta todos los demás vértices (problema de la trayectoria más corta de una sola fuente) en poco tiempo, suponiendo que los pesos de todos los bordes no son negativos. Aunque, el Algoritmo de Dijkstra es eficiente en términos de tiempo de recorrido para determinar el camino más corto de manera óptima, el camino propuesto está lejos de ser la ruta ideal para la mayoría de los usuarios; porque el Algoritmo de Dijkstra no tiene en cuenta el número de transferencias o las distancias a pie. Estas dos deficiencias son desventajas importantes para los usuarios finales.

En este sentido, a última década ha sido testigo de un asombroso progreso en el rendimiento de los algoritmos de trayecto más corto en las redes de transporte. Para el encaminamiento en redes de carreteras, en particular, los algoritmos modernos pueden ser hasta siete órdenes de magnitud más rápidos que las soluciones estándar (Bast *et al.* 2016). Los enfoques exitosos aprovechan las diferentes propiedades de las redes de carreteras que las hacen más fáciles de manejar que los gráficos generales, como la dirección de los objetivos, una estructura jerárquica fuerte y la existencia de pequeños separadores.

Aunque algunas de las primeras técnicas de aceleración se basaban en gran medida en la geometría (después de todo, las redes de carreteras están incrustadas en la superficie de la Tierra), ningún algoritmo actual de última generación utiliza explícitamente las coordenadas de los vértices. Mientras que todavía se ve el desarrollo ocasional (y la publicación) de algoritmos basados en geometría, estos están consistentemente dominados por técnicas establecidas. En particular, el algoritmo reciente de Jerarquías Arteriales (Zhu *et al.*, 2017) se compara con el algoritmo CH (*Contraction Hierarchies*, que tiene consultas ligeramente más lentas), pero no con otras técnicas previamente publicadas (como CHASE, HL y TNR) que lo dominarían fácilmente. Otra lección importante de los desarrollos recientes es que una ingeniería cuidadosa es esencial para liberar toda la potencia computacional de las arquitecturas informáticas modernas. Algoritmos como CRP, CSA, HL, PHAST y RAPTOR, por ejemplo, logran gran parte de su buen rendimiento explotando cuidadosamente la localidad de referencia y el paralelismo (a nivel de instrucciones, núcleos e incluso GPUs).

Este tipo de algoritmos se han utilizado en sistemas que sirven a millones de usuarios cada día, incluyendo proyectos relacionados con el enrutamiento para empresas como Apple, Esri, Google, MapBox, Microsoft, Nokia, PTV, TeleNav, TomTom y Yandex. Aunque las empresas tienden a ser reservadas sobre los algoritmos que utilizan, en algunos casos esto es de conocimiento público. TomTom

utiliza una variante de los indicadores de arco con accesos directos para realizar consultas dependientes del tiempo (Zhu *et al.* 2017). Bing Maps de Microsoft utiliza CRP para el enrutamiento en redes de carreteras (Zhu *et al.* 2017). OSRM, un popular motor de planificación de rutas que utiliza datos de OpenStreetMap, utiliza CH para las consultas (Luxen y Vetter, 2011). El algoritmo Transfer Patterns (Bast *et al.*, 2010) se utiliza para la planificación de viajes de transporte público en Google Maps desde 2010 (Abraham *et al.*, 2012). Actualmente, OpenTripPlanner está utilizando RAPTOR (Abraham *et al.*, 2013).

En este tipo de algoritmos se basa el empleado por Google Maps (que utiliza una evolución mucho más eficiente en términos de rendimiento). Este tipo de búsquedas de rutas es utilizado también por aplicativos para la contratación de servicios VTC como Uber o Cabify. Además, servicios como Google incorporan información parcial sobre los horarios del transporte público terrestre como tren de largo recorrido, cercanías, autobús y metro. Si bien tienen en cuenta los costes de los trayectos en el caso del transporte en vehículo privado (combustible y peajes) no tienen en cuenta los costes de los trayectos en el caso del transporte público.

Ejemplos de implementación en Python

En esta ocasión vamos a emplear la librería NetworkX para realizar unas pruebas de búsquedas en grafos. NetworkX es un paquete de *software* en lenguaje Python para la creación, manipulación y estudio de la estructura, dinámica y función de redes complejas. Con NetworkX es posible cargar y almacenar redes en formatos de datos estándar y no estándar, generar muchos tipos de redes aleatorias y clásicas, analizar la estructura de la red, construir modelos de redes, diseñar nuevos algoritmos de redes, dibujar redes, y mucho más. La librería incluye algoritmos de búsqueda en grafos, como A* o Dijkstra, entre otros. Nos permite, además, dibujar los grafos mediante matplotlib o graphviz , entre otras.



Figura 19. Fuente: <https://networkx.github.io/>

Instalamos el paquete, si no contáramos ya con él, mediante el mecanismo habitual, como, por ejemplo:

```
PS C:\Users\xxx> pip install networkx
```

Creamos una red con distancias entre ciudades ficticias, dibujamos el grafo y buscamos la ruta más corta entre dos de ellas. El algoritmo utilizado (se puede ver el código fuente en la web de la librería) está basado en el algoritmo Dijkstra.

```
import networkx as nx
import matplotlib.pyplot as plt

# creamos un grafo con distancias entre ciudades ficticias no conectadas todas entre sí
G=nx.Graph()

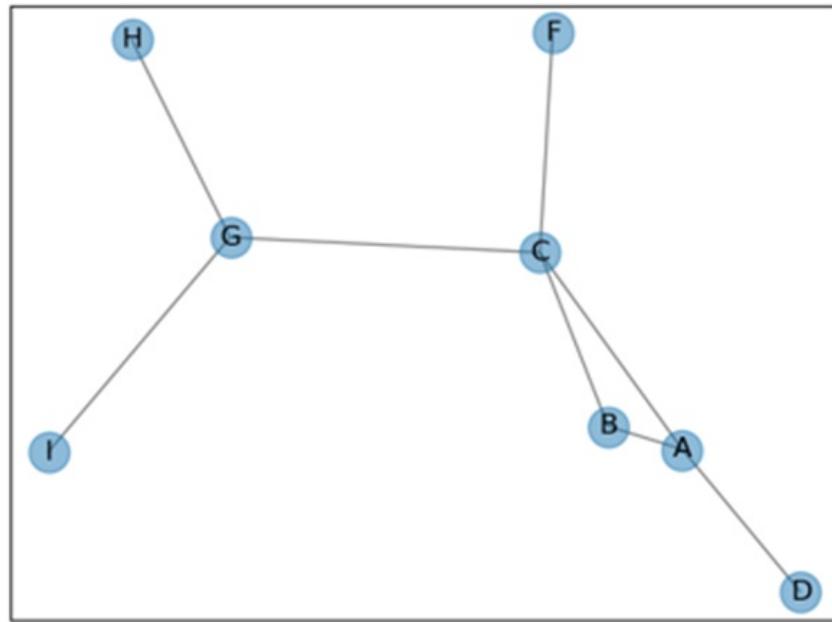
G.add_edge("A", "B", weight=140)
G.add_edge("A", "C", weight=50)
G.add_edge("A", "D", weight=60)
G.add_edge("B", "C", weight=80)
G.add_edge("C", "F", weight=110)
G.add_edge("C", "G", weight=15)
G.add_edge("G", "H", weight=50)
G.add_edge("G", "I", weight=20)

print("shortest path:")
print(nx.shortest_path(G,source="A",target="I"))

pos=nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos, alpha=0.5)
nx.draw_networkx_edges(G,pos, alpha=0.5)
nx.draw_networkx_labels(G,pos)
plt.show()
```

shortest path:

['A', 'C', 'G', 'I']



Podemos aplicar también el algoritmo A* en una malla de 4×4 puntos 2-D, en los cuales definimos la distancia heurística como la distancia euclídea que existe entre dos puntos. De hecho, es una $h(x)$ usualmente empleada, al nunca sobreestimar y tener sentido topográfico. El coste de llegar de una ciudad a otra será la distancia por carretera, pero un buen heurístico es la distancia en línea recta, por ser siempre una distancia menor, pero basada en la realidad y fácilmente calculable.

```

import networkx as nx
import matplotlib.pyplot as plt

G=nx.grid_graph(dim=[4,4])

print("A* path:")
def dist(a,b):
    (x1, y1) = a
    (x2, y2) = b
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2) ** 0.5

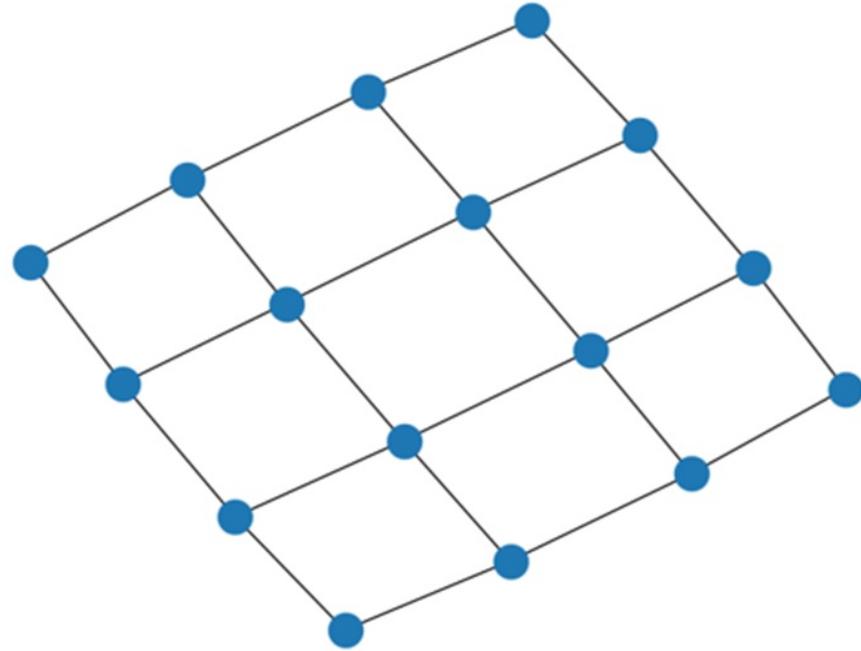
print(nx.astar_path(G, (0,0), (2,2),dist))

nx.draw(G)
plt.show()

```

A* path:

```
[(0, 0), (1, 0), (1, 1), (2, 1), (2, 2)]
```



9.9. Referencias bibliográficas

Abraham, I., Delling, D., Goldberg, A. V. & Werneck, R. F. (2012). Hierarchical hub labelings for shortest paths. En *European Symposium on Algorithms* (pp. 24-35). Berlin: Springer.

Abraham, I., Delling, D., Goldberg, A. V. & Werneck, R. F. (2013). Alternative routes in road networks. *Journal of Experimental Algorithmics (JEA)*, 18, 1-3.

Bast, H., Carlsson, E., Eigenwillig, A., Geisberger, R., Harrelson, C., Raychev, V. & Viger, F. (2010). Fast routing in very large public transportation networks using transfer patterns. En *European Symposium on Algorithms* (pp. 290-301). Berlin: Springer.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P. & Werneck, R. F. (2016). Route planning in transportation networks. En *Algorithm engineering* (pp. 19-80). Cham: Springer.

Bozyigit, A., Alankus, G. & Nasiboğlu, E. (2017). Public transport route planning: Modified dijkstra's algorithm. En *2017 International Conference on Computer Science and Engineering (UBMK)* (pp. 502-505). IEEE.

Fikes, R. E. & Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3), 189-208. Disponible en [https://doi.org/10.1016/0004-3702\(71\)90010-5](https://doi.org/10.1016/0004-3702(71)90010-5)

Luxen, D. & Vetter, C. (2011). *Real-time routing with OpenStreetMap data*. En Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems (pp. 513-516). ACM.

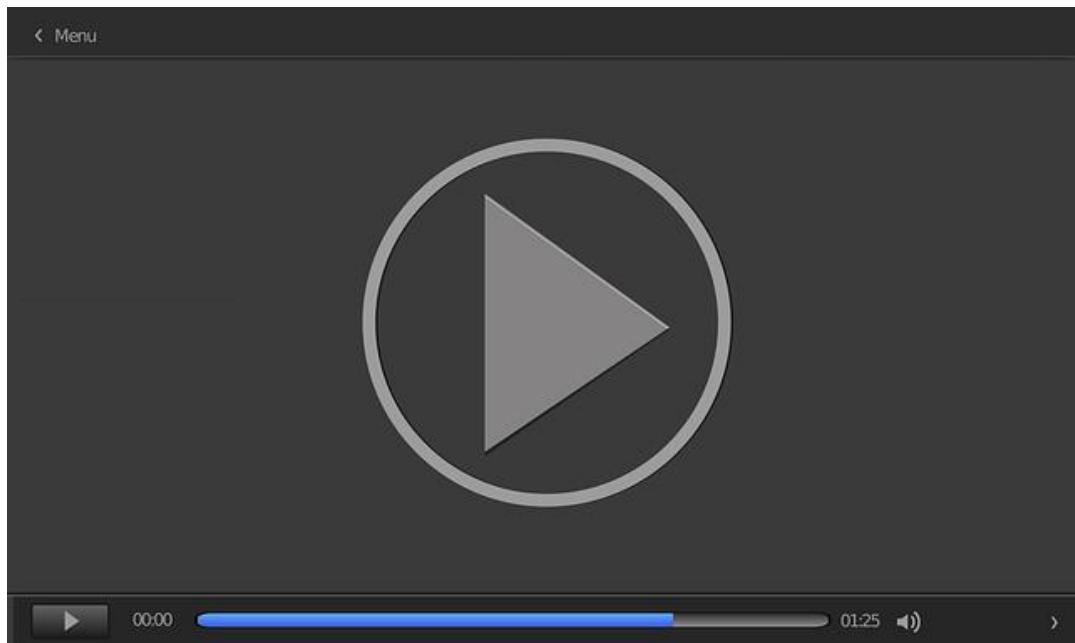
Poole, D. L. & Mackworth, A. K. (2010). *Artificial intelligence: foundations of computational agents*. New York: Cambridge University Press.

Slaney, J. & Thiébaux, S. (2001). Blocks World revisited. *Artificial Intelligence*, 125(1), 119-153. Disponible en [https://doi.org/10.1016/S0004-3702\(00\)00079-5](https://doi.org/10.1016/S0004-3702(00)00079-5)

Zhu, A.D., Ma, H., Xiao, X., Luo, S., Tang, Y. & Zhou, S. (2013). *Shortest path, distance queries on road networks: towards bridging theory and practice*. En Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 2013) (pp. 857-868). ACM Press.

Búsqueda en juegos: algoritmo Minimax

Esta lección magistral consiste en ilustrar el funcionamiento del algoritmo de búsqueda en juegos Minimax aplicándolo al juego de Grundy.



09. Búsqueda en juegos: algoritmo Minimax

Accede al vídeo:

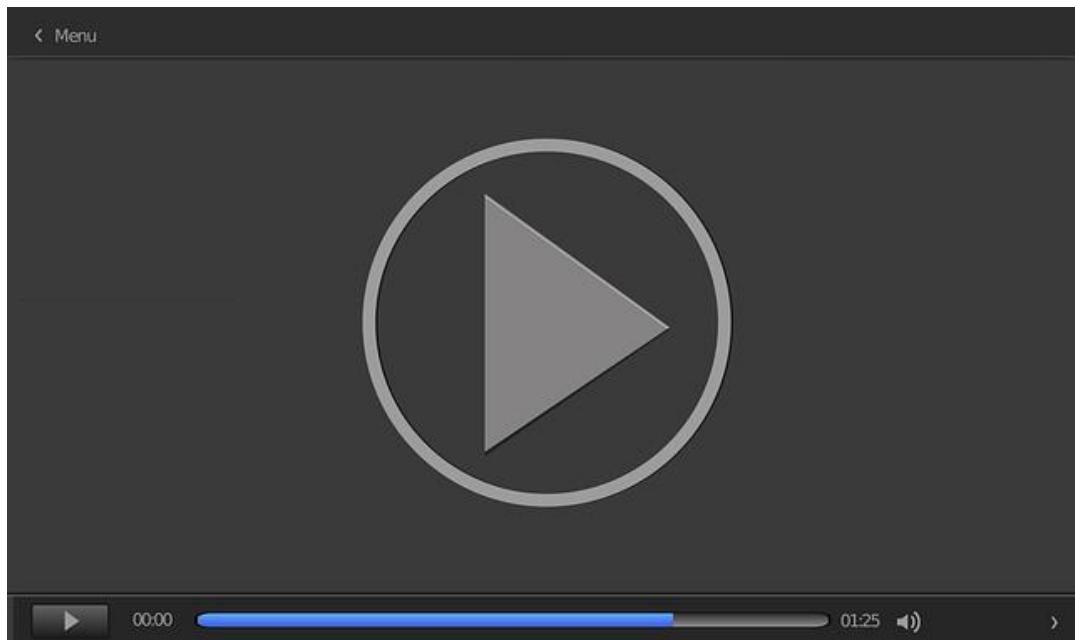
<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=df90cff-6735-4a8f-b687-aff800fac5dc>

Métodos de búsquedas, a ciegas y heurísticas

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=g6l685-LMbU>

Presentación que explica los métodos de búsquedas a ciegas y heurísticas, con un ejemplo muy ilustrativo relativo a un problema de salir de un laberinto.



Accede al vídeo:

<https://www.youtube.com/embed/g6l685-LMbU>

Making games

Accede a la página principal desde el aula virtual o a través de la siguiente dirección web: <http://www-cs-students.stanford.edu/~amitp/>

Accede a la sección de programación de juegos desde el aula virtual o a través de la siguiente dirección web: <http://www-cs-students.stanford.edu/~amitp/gameprog.html>

Página web de un ex estudiante de doctorado de la Universidad de Stanford. Contiene información, recursos y enlaces muy interesantes relacionados con la inteligencia artificial y la creación de juegos. Específicamente se recomienda visitar la sección dedicada a temas de programación de juegos, que incluye recursos relacionados con el tipo de cuestiones que el autor necesitó conocer mientras trabajó en crear juegos. El autor hace hincapié en el algoritmo A*, con enlaces muy interesantes, códigos y demos.

Applets de Java para visualizar el funcionamiento de estrategias de búsqueda a ciegas

Accede a la página principal desde el aula virtual o a través de la siguiente dirección web: <http://cse.unl.edu/~choueiry/S03-476-876/searchapplet/>

Página web interactiva en la que puedes observar el orden en que un determinado algoritmo de búsqueda a ciegas recorre los nodos de un árbol. Específicamente se puede observar el funcionamiento de los algoritmos explicados en este tema. Además, la autora de los *applets* que permiten esta visualización, Naomi Novik, proporciona acceso al código fuente de los *applets* y documentación asociada.

NetworkX

Accede a la página desde el aula virtual o a través de la siguiente dirección web:

<https://networkx.github.io/>

NetworkX es un paquete de software en lenguaje Python para la creación, manipulación y estudio de la estructura, dinámica y función de redes complejas. Con NetworkX es posible cargar y almacenar redes en formatos de datos estándar y no estándar, generar muchos tipos de redes aleatorias y clásicas, analizar la estructura de la red, construir modelos de redes, diseñar nuevos algoritmos de redes, dibujar redes, y mucho más. La librería incluye algoritmos de búsqueda en grafos, como A* o Dijkstra, entre otros. Nos permite, además, dibujar los grafos mediante matplotlib o graphviz, entre otras.

1. Indica cuáles de las siguientes afirmaciones son correctas respecto a los problemas de búsqueda de estados:

 - A. La resolución de un problema mediante búsqueda conlleva definir únicamente dos descriptores: los de operación y los de estado.
 - B. La resolución de un problema mediante búsqueda conlleva definir únicamente dos descriptores: los de operación y los de estado.
 - C. La búsqueda de estados se aplica en juegos y en robótica.
 - D. La resolución de un problema de búsqueda se puede describir como la búsqueda de un camino a través de un conjunto de estados que permite alcanzar un estado objetivo desde un estado inicial.
2. Cuando se conoce un estado inicial y múltiples estados objetivos, ¿qué búsqueda es más conveniente aplicar en principio?

 - A. Búsqueda hacia adelante.
 - B. Búsqueda hacia atrás.
3. Si he de justificar el motivo por el que se da un razonamiento se suele utilizar:

 - A. Búsqueda hacia adelante.
 - B. Búsqueda hacia atrás.

4. Indica cuáles de las siguientes afirmaciones son correctas respecto a los algoritmos de búsqueda a ciegas:

- A. La búsqueda en amplitud sigue una estrategia LIFO para expandir los nodos.
- B. La búsqueda en profundidad garantiza que se encuentre la solución por el camino más corto.
- C. La búsqueda en profundidad no tiene en cuenta ninguna información sobre el problema.
- D. La búsqueda en profundidad iterativa intenta combinar las ventajas de la búsqueda en amplitud y la búsqueda en profundidad.

5. Indica cuáles de las siguientes afirmaciones son correctas respecto a los algoritmos de búsqueda heurística:

- A. Utilizan información heurística que es información sobre el problema para encontrar la solución.
- B. El heurístico es una función que permite garantizar encontrar la solución óptima.
- C. Facilitan la aceleración de la búsqueda.
- D. El heurístico puede utilizarse para, una vez expandido el nodo, decidir qué nodo hijo ha de ser considerado en primer lugar.

6. Indica la afirmación correcta relacionada con la búsqueda heurística:
- La escalada por máxima pendiente presenta problemas de máximos locales pero no de mesetas.
 - La escalada simple considera todos los posibles movimientos a partir del estado actual, escogiendo el mejor de ellos.
 - El algoritmo de búsqueda «primero el mejor» puede presentar problemas de memoria.
 - El heurístico en un algoritmo A* ha de ser admisible, esto es, no debe sobreestimar el coste desde el nodo origen al nodo actual.
7. Si no se dispone de información útil de un problema de búsqueda, ¿qué método de los siguientes se podría en principio aplicar para resolverlo?
- Búsqueda en profundidad.
 - Búsqueda «mejor el primero».
 - Escalada por máxima pendiente.
 - Ninguno de los anteriores.
8. Si se dispone de información útil de un problema de búsqueda, ¿qué método de los siguientes se podría, en principio, aplicar para resolverlo?
- Búsqueda en profundidad iterativa.
 - Búsqueda en profundidad acotada.
 - Búsqueda «mejor el primero».
 - Ninguno de los anteriores.

- 9.** Indica cuáles de las siguientes afirmaciones relativas al algoritmo Minimax de búsqueda en juegos son ciertas:
- A. Minimax es un algoritmo utilizado en la búsqueda en juegos.
 - B. El algoritmo Minimax desarrolla todo el espacio de estados.
 - C. Minimax se aplica a juegos de información completa, pero suma no nula.
 - D. Minimax toma las decisiones óptimas en cada jugada de tal forma que si es posible Min ganará la partida.
- 10.** Si se desea minimizar el coste del proceso de búsqueda se ha de:
- A. Minimizar el coste debido a la expansión del árbol.
 - B. Minimizar el coste debido a la estrategia de control.
 - C. Minimizar la suma de los costes debidos a la expansión del árbol y a la estrategia de control.
 - D. Minimizar los costes debidos a la expansión del árbol y utilizar el mejor heurístico posible.

Técnicas de Inteligencia Artificial

Tema 10. Gestión de la incertidumbre e imprecisión en sistemas expertos

Índice

Esquema

Ideas clave

- 10.1. ¿Cómo estudiar este tema?
- 10.2. Introducción
- 10.3. Razonamiento bayesiano
- 10.4. Factores de certeza
- 10.5. Lógica difusa
- 10.6. Inferencia difusa
- 10.7. Aplicaciones y ejemplos de implementación
- 10.8. Referencias bibliográficas

A fondo

FuzzyCLIPS: introducción práctica

CLIPS: guía del usuario

Guía práctica de FuzzyCLIPS

Guía del usuario de Fuzzy Logic Toolbox (MATLAB)

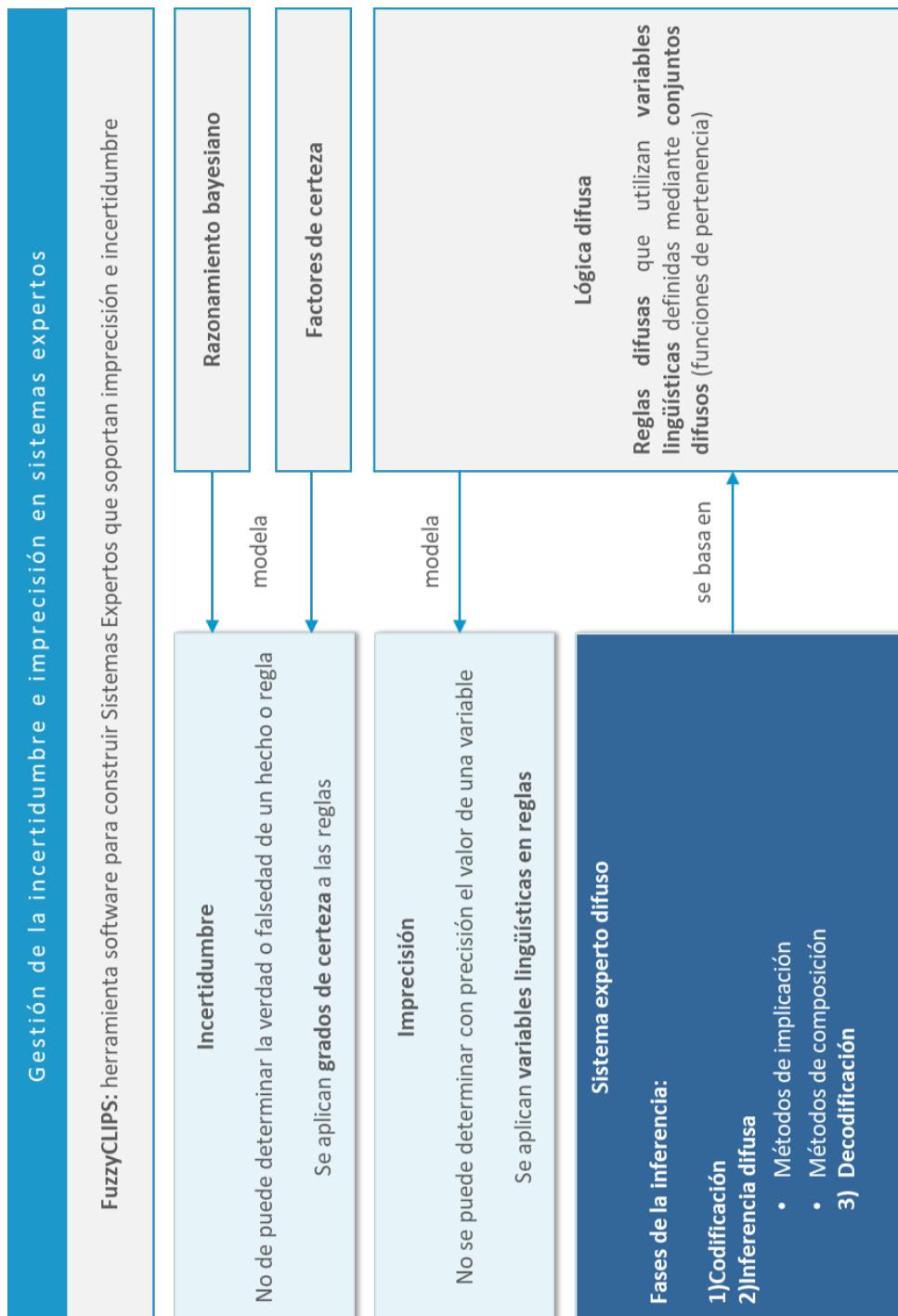
El primer sistema manejando factores de certeza: MYCIN

Introducción a la lógica difusa

Lógica difusa: introducción

Scikit Fuzzy

Test



10.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral y revisando los recursos adicionales que se facilitan.

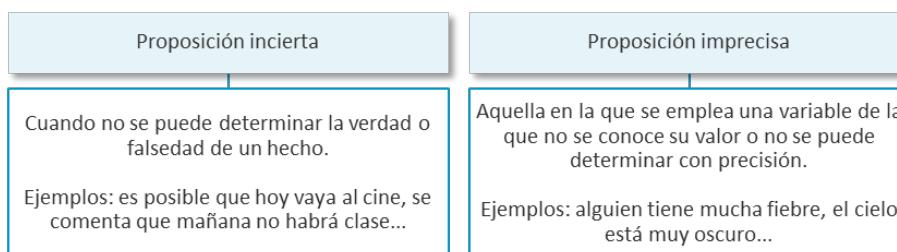
Al finalizar el estudio de este tema el alumno será capaz de:

- ▶ Describir mecanismos que modelan hechos imprecisos e inciertos.
- ▶ Entender los conceptos básicos del razonamiento bayesiano.
- ▶ Entender los conceptos básicos de la lógica difusa.
- ▶ Aplicar estos mecanismos en el desarrollo de un sistema experto sencillo.

10.2. Introducción

En muchas ocasiones se han de resolver problemas a partir de datos cuya certeza no se puede asegurar. Aun así, muchos expertos son capaces de manejar las incertidumbres sobre ciertos hechos y llegar a conclusiones válidas que solucionan un problema. De la misma manera, un sistema experto puede trabajar con hechos inciertos, imprecisos o incompletos.

La creencia de un hecho se puede expresar de diversas maneras. A continuación, se muestran dos tipos de proposiciones que permiten expresar incertidumbre e imprecisión, respectivamente:



Es posible también combinar los dos tipos de proposiciones, dotando de incertidumbre e imprecisión al hecho. Por ejemplo: creo que hoy llueve mucho.

¿Qué mecanismos nos permiten modelar cada tipo de proposición en un sistema experto?

Se pueden emplear diferentes mecanismos que se enumeran a continuación y se explican a lo largo de este tema:



Asignando unos **grados de certeza (CF)** a reglas y hechos y realizando inferencias con estos grados de certeza se puede modelar la incertidumbre. También las teorías

estadísticas permiten modelar incertidumbre. Por otra parte, la **imprecisión** se puede modelar empleando variables difusas en los antecedentes y consecuentes de las reglas, tal y como se explicará posteriormente.

¿A qué puede ser debida la existencia de hechos inciertos o imprecisos?

En un sistema basado en conocimiento la base de conocimiento, la base de hechos y, en definitiva, los datos y reglas con los que se trabaja en la memoria de trabajo pueden no ser perfectos debido a diferentes causas:

► **Información:**

- **Incompleta o desconocida** (por ejemplo, falta de análisis en medicina, falta de variables de campo en sistemas de control).
- **Poco confiable** (por ejemplo, medidores poco confiables, instrumentos imprecisos, análisis poco confiables).
- **Ruido, distorsión** (por ejemplo, ruido o distorsión en sistemas de visión, de reconocimiento de voz, de comunicaciones).

► **Conocimiento:**

- **Impreciso.** El experto de dominio a menudo no puede más que aportar asociaciones vagas de hechos (por ejemplo, si tiene dolor de cabeza posiblemente tiene gripe). Además, el lenguaje por naturaleza es impreciso y vago (por ejemplo, se utiliza con frecuencia términos como «frecuente», «a veces», «a menudo», etc., tal y como se puede comprobar precisamente en el texto de este tema).
- **Contradicitorio.** Se puede dar el caso de que se den proposiciones opuestas proporcionadas por dos expertos de dominio distintos (por ejemplo, si tiene dolor de cabeza es probable que tenga gripe, pero también es posible que no tenga gripe, opiniones encontradas de diferentes expertos). Es habitual que trabajen varios expertos de dominio en la creación de un sistema experto grande y complejo, combinando su conocimiento y permitiendo enriquecer el sistema, pero con el riesgo

de encontrar conclusiones contradictorias, ya que frecuentemente los expertos no opinan de la misma manera ante unos hechos y llegan a diferentes conclusiones.

Dado que el conocimiento es incierto y vago, y los datos son incompletos y ruidosos, en muchos problemas del mundo real, un sistema experto ha de ser capaz de manejar este conocimiento y estos datos.

En este tema se explican primeramente dos métodos populares para manejar la incertidumbre: **razonamiento bayesiano** y **los factores de certeza**. Tras ello, se describirá la **lógica difusa**, que permite trabajar con la imprecisión en los hechos expresados en antecedentes y consecuentes.

10.3. Razonamiento bayesiano

En muchos sistemas de resolución de problemas un objetivo importante consiste en reunir evidencias sobre la evolución del sistema y modificar su comportamiento sobre la base de las mismas. Para modelar este comportamiento se puede utilizar el razonamiento bayesiano.

El concepto fundamental empleado en este razonamiento es el de la probabilidad condicionada de una hipótesis H , dado que se observa la evidencia E .

Esta probabilidad viene dada por la siguiente expresión conocida como **Teorema de Bayes**:

$$p(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \quad (1)$$

La inferencia Bayesiana se aplica mucho en los diagnósticos médicos. Por ejemplo:

SI tiene fiebre	E evidencia
ENTONCES tiene gripe	H hipótesis
$P(H E) = 0,86$	

Para calcular la probabilidad condicionada es necesario tener en cuenta la probabilidad previa de H (la probabilidad que se le asignaría a H si no existe evidencia) y la parte en la que E proporciona evidencia de H . Para lograrlo, se define un conjunto exhaustivo de evidencias mutuamente exclusivas E_1, E_2, \dots, E_n , tal que $\sum_{i=1}^n P(E_i) = 1$, que permite obtener la fórmula siguiente (en el caso particular de dos evidencias mutuamente excluyentes), base del manejo de la incertidumbre en sistemas expertos:

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E|H) \times P(H) + P(E|\bar{H}) \times P(\bar{H})} \quad (2)$$

Para ilustrar cómo se puede utilizar este razonamiento se considera el siguiente ejemplo, donde se quiere analizar la situación económica de una determinada persona:

► Hipótesis:

- Sistema económico solvente: $P(S) = 0.7$
- Insolvencia económica: $P(\bar{S}) = 0.3$

► Factores de interés:

- Vivienda en propiedad: $P(V|S) = 0.5$ y $P(V|\bar{S}) = 0.1$
- Ingresos altos: $P(I|S) = 0.7$ y $P(I|\bar{S}) = 0.2$

Si se quiere saber si es solvente una persona sin vivienda, aplicando la expresión (2) se obtiene:

$$P(S|\bar{V}) = \frac{p(S) \cdot p(\bar{V}|S)}{p(S) \cdot p(\bar{V}|S) + p(\bar{S}) \cdot p(\bar{V}|\bar{S})} = \frac{0.7 \cdot 0.5}{0.7 \cdot 0.5 + 0.3 \cdot 0.9} = 0.56$$

$$P(\bar{S}|\bar{V}) = 1 - 0.56 = 0.44$$

Si se quiere conocer si es solvente una persona con altos ingresos, aplicando la expresión (2) se obtiene:

$$P(S|I) = \frac{p(S) \cdot p(I|S)}{p(S) \cdot p(I|S) + p(\bar{S}) \cdot p(I|\bar{S})} = \frac{0.7 \cdot 0.7}{0.7 \cdot 0.7 + 0.3 \cdot 0.2} = 0.89$$

En este segundo caso se puede llegar a la conclusión más clara de que una persona con altos ingresos es muy probablemente solvente. Sin embargo, la vivienda en propiedad no es un factor que indique claramente la solvencia.

Se puede generalizar la expresión en la ecuación (2) obteniendo la ecuación (3), teniendo en cuenta no solo múltiples evidencias mutuamente exclusivas y exhaustivas, sino también múltiples hipótesis H_1, H_2, \dots, H_m , mutuamente exclusivas y exhaustivas, tal que $\sum_{i=1}^m P(H_i) = 1$, y teniendo en cuenta que se asume independencia condicional entre las distintas evidencias (con el fin de evitar la alta e inmanejable computación que requeriría calcular probabilidades condicionales de todas las posibles combinaciones de evidencias para todas las hipótesis).

$$\begin{aligned} & P(H_i | E_1 E_2 \dots E_n) \\ &= \frac{P(E_1 | H_i) \cdot P(E_2 | H_i) \cdot \dots \cdot P(E_n | H_i) \cdot P(H_i)}{\sum_{k=1}^m P(E_1 | H_k) \cdot P(E_2 | H_k) \cdot \dots \cdot P(E_n | H_k) \cdot P(H_k)} \end{aligned} \quad (3)$$

Existen sistemas que utilizan reglas bayesianas, pero, tal y como se puede concluir observando la ecuación (3), no siempre se puede aplicar este método por los siguientes motivos:

- ▶ El problema de adquisición de conocimiento puede resultar inabordable, ya que es necesario conocer demasiadas probabilidades.
- ▶ Durante los cálculos computacionales, el espacio necesario para almacenar todas las probabilidades, así como el tiempo necesario para calcular las probabilidades, pueden resultar demasiado grandes.

A pesar de todos estos problemas, las estadísticas bayesianas proporcionan una base atractiva para los sistemas que trabajan bajo incertidumbre. Cuando se dispone de datos estadísticos fiables, el método bayesiano puede ser muy adecuado, pero se debe remarcar que, en el caso de grandes bases de conocimiento, podría resultar impráctico por el alto coste computacional.

Por este motivo y especialmente por el hecho la carencia de los datos necesarios para el cálculo de probabilidades, se proponen otros métodos de cálculos más sencillos como el que se expone en el siguiente apartado.

Naïve Bayes

Una de las aplicaciones más conocidas del razonamiento bayesiano son los clasificadores probabilísticos conocidos como **naïve Bayes** (Murphy et al., 2006). Estos clasificadores facilitan predicciones probabilísticas, ya que, mediante el conocimiento previo que tienen, determinan la probabilidad de una hipótesis. En este sentido, utilizando naïve Bayes podrán clasificarse nuevas instancias mediante la combinación de probabilidades de distintas hipótesis.

La **suposición** que emplean los **clasificadores naïve** es que **los atributos de entrada son independientes entre sí con respecto a la clase**. Esta simplificación podría indicar que no se obtendrán buenos resultados pues, en muchas ocasiones, la independencia de los atributos de entrada con respecto a la clase no es cierta. Sin embargo, su aplicación da buenos resultados cuando se dispone de conjuntos de entrenamiento de tamaño medio o grande y, obviamente, cuando los atributos que describen los ejemplos son fuertemente independientes entre sí respecto a la clase. Algunos ejemplos de aplicación exitosa son los problemas de diagnóstico o la clasificación de documentos.

De forma más concreta, con un clasificador naïve Bayes se desea calcular la probabilidad *a posteriori* de una clase, para cada valor de los atributos de entrada elegidos. Si disponemos un conjunto de ejemplos E , cada uno de ellos con un

número n de atributos (a_1, \dots, a_n) y un número k de clases (c_1, \dots, c_k), la probabilidad de que para un determinado valor de los atributos se cumpla una clase j vendría dada por la siguiente expresión:

$$P(a_1, a_2, \dots, a_n \vee c) \quad (4)$$

Aplicando la suposición de independencia de naïve Bayes:

$$P(a_1, a_2, \dots, a_n | c_j) = \prod_i P(a_i \vee c_j) \quad (5)$$

Y, la aproximación del clasificador naïve Bayes vendría determinada por la expresión (6):

$$C_{nb} = \arg \max_{c_i \in C} P(c_j) \prod_i P(a_i \vee c_j) \quad (6)$$

Tomando como ejemplo el problema «Jugar al aire libre» de Quinlan (1986), pueden estimarse las probabilidades de jugar o no para cada uno de los atributos (ver Tabla 1):

- ▶ $P(Sí) = 9/14 = 0.64$
- ▶ $P(No) = 5/14 = 0.36$
- ▶ $P(\text{Ambiente} = \text{Soleado} | \text{Jugar} = \text{Sí}) = 2/9 = 0.22$
- ▶ $P(\text{Ambiente} = \text{Soleado} | \text{Jugar} = \text{No}) = 3/5 = 0.6$
- ▶ $P(\text{Humedad} = \text{Alta} | \text{Jugar} = \text{Sí}) = 3/9 = 0.33$
- ▶ $P(\text{Humedad} = \text{Alta} | \text{Jugar} = \text{No}) = 4/5 = 0.8$

Entonces, si se quisiera clasificar la siguiente instancia:

$E_i = \{\text{Ambiente} = \text{Soleado}, \text{Temperatura} = \text{Media}, \text{Humedad} = \text{Alta}, \text{Viento} = \text{Verdadero}\}$

Se calculan las probabilidades *a posteriori*:

- ▶ $P(\text{Jugar} = \text{Sí}) P(\text{Ambiente} = \text{Soleado} | \text{Jugar} = \text{Sí}) P(\text{Temperatura} = \text{Media} | \text{Jugar} = \text{Sí}) P(\text{Humedad} = \text{Alta} | \text{Jugar} = \text{Sí}) P(\text{Viento} = \text{Verdadero} | \text{Jugar} = \text{Sí}) = 0.0067$
(0.14).
- ▶ $P(\text{Jugar} = \text{No}) P(\text{Ambiente} = \text{Soleado} | \text{Jugar} = \text{No}) P(\text{Temperatura} = \text{Media} | \text{Jugar} = \text{Sí}) P(\text{Humedad} = \text{Alta} | \text{Jugar} = \text{No}) P(\text{Viento} = \text{Verdadero} | \text{Jugar} = \text{No}) = 0.041$
(0.86).

Y, como resultado, el clasificador devuelve que la clasificación con mayor probabilidad *a posteriori* es No jugar.

Id	Ambiente	Temperatura	Humedad	Viento	Clase
E ₁	Soleado	Alta	Alta	Falso	No
E ₂	Soleado	Alta	Alta	Verdadero	No
E ₃	Nublado	Alta	Alta	Falso	Sí
E ₄	Lluvioso	Media	Alta	Falso	Sí
E ₅	Lluvioso	Baja	Normal	Falso	Sí
E ₆	Lluvioso	Baja	Normal	Verdadero	No
E ₇	Nublado	Baja	Normal	Verdadero	Sí
E ₈	Soleado	Media	Alta	Falso	No
E ₉	Soleado	Baja	Normal	Falso	Sí
E ₁₀	Lluvioso	Media	Normal	Falso	Sí
E ₁₁	Soleado	Media	Normal	Verdadero	Sí
E ₁₂	Nublado	Media	Alta	Verdadero	Sí
E ₁₃	Nublado	Alta	Normal	Falso	Sí
E ₁₄	Lluvioso	Media	Alta	Verdadero	no

Tabla 1. Instancias con sus valores de atributos y clases del problema «Jugar al aire libre».

10.4. Factores de certeza

El uso de factores de certeza es una alternativa al razonamiento bayesiano. El primer sistema que utilizó estos factores fue el sistema experto MYCIN (Shortliffe & Buchanan, 1975), escrito en LISP, y que permitía diagnosticar infecciones de sangre y meningitis. Dado que no había datos estadísticos fiables sobre el problema, los creadores de MYCIN utilizaron un factor de certeza para indicar la confianza o certeza del experto humano.

El **factor de certeza** puede tener un valor entre -1 y +1. -1 indica falsedad total mientras que el valor +1 indica certeza total. Un valor positivo indica cierto grado de creencia en la expresión mientras que un valor negativo indica cierto grado de incredulidad respecto a la veracidad de la expresión.

Si se emplean factores de certeza, las reglas en la base de conocimiento irán acompañadas de este factor, que indica el grado de certeza de que se dé el hecho expresado en el consecuente o hipótesis, dado el hecho expresado en el antecedente o evidencia.

Cuando se trabaja con factores de certeza se pueden encontrar reglas como la que sigue:

SI 'vivienda en propiedad'

ENTONCES 'sistema económico es solvente' {cf 0.6}

'sistema económico es insolvente' {cf 0.2}

Entre paréntesis, junto a cada consecuente, se representa el factor de certeza precedido de las siglas *cf* (*certainty factor*). Ese valor, por tanto, representa la creencia en que se dé la hipótesis, dada la evidencia.

En el ejemplo, dada la evidencia ‘vivienda en propiedad’, se tiene una mayor certeza (0.6) de que el sistema económico sea solvente a que sea insolvente (0.2). No es imprescindible que ambos factores de certeza sumen 1, como se refleja en el ejemplo. Puede existir un factor 0.2 que se podría asignar a otro hecho como un término medio entre solvente e insolvente, u otro valor no observado.

¿Cómo se propagan los factores de certeza en el encadenamiento de reglas?

Puede suceder que el propio antecedente de la regla sea incierto y tenga asignado un factor de certeza y este factor se ha de propagar a lo largo de la cadena de reglas.

El factor de certeza para una regla con un único antecedente se puede calcular simplemente multiplicando el factor de certeza del antecedente (la evidencia *E*) por el factor de certeza de la regla cuyo consecuente es *H* mediante la siguiente fórmula:

$$cf(H,E) = cf(e) \bullet cf(7)$$

Si en el anterior ejemplo, el factor de certeza de que la ‘vivienda en propiedad’ es 0.6, entonces se tienen los siguientes factores de certeza para cada una de las hipótesis:

$$cf(H_1,E) = 0.6 \bullet 0.6 = 0.36$$

$$cf(H_2,E) = 0.2 \bullet 0.6 = 0.12$$

Siendo H_1 la hipótesis ‘sistema económico es solvente’ y H_2 , la hipótesis ‘sistema económico es insolvente’.

En el caso de que existan múltiples antecedentes, el factor de certeza se propaga de diferente manera según las evidencias o antecedentes sean una conjunción (AND) o disyunción (OR).

En el caso de que sea una regla conjuntiva del tipo:

SI E1

AND E2

AND E3

...

AND En

ENTONCES

H1 {cf}

El factor de certeza del consecuente, la hipótesis, se calcula de la siguiente manera:

cf

Para el siguiente ejemplo:

SI 'vivienda en propiedad'

AND 'ingresos altos'

ENTONCES 'sistema económico es solvente' {cf 0.6}

Si el factor de certeza del hecho ‘vivienda en propiedad’ es 0.7 y el factor de certeza del hecho ‘ingresos altos’ es 0.4, entonces el factor de certeza del consecuente se calcula de la siguiente manera:

$$cf(H, E_1 \cap E_2) = \min [0.7, 0.4] \bullet 0.6 = 0.24$$

En el caso de que se tenga una regla disyuntiva del tipo:

SI E1

OR E2

OR E3

...

OR En

ENTONCES

H1 {cf}

El factor de certeza del consecuente, la hipótesis, se calcula de la siguiente manera:

$$cf(H_1, E_1 \cap E_2 \cup E_3 \cup \dots \cup E_n) = \max [cf(E_1), cf(E_2), \dots, cf(E_n)] \bullet cf$$

Para el siguiente ejemplo:

SI ‘vivienda en propiedad’

OR ‘ingresos altos’

ENTONCES ‘sistema económico es solvente’ {cf 0.6}

Si el factor de certeza del hecho ‘vivienda en propiedad’ es 0.7 y el factor de certeza del hecho ‘ingresos altos’ es 0.4, entonces el factor de certeza del consecuente se calcula de la siguiente manera:

$$cf(H_1, E_1 \cap E_2) = \max [0.7, 0.4] \bullet 0.6 = 0.42$$

Los factores de certeza y su propagación a lo largo de las reglas encadenadas, aunque no tienen gran base matemática como el razonamiento bayesiano, son muy prácticos en algunos problemas produciendo buenos resultados con menos coste computacional. Además, los métodos estadísticos requieren el conocimiento de gran cantidad de datos.

10.5. Lógica difusa

Lógica difusa

Es una familia de teorías y técnicas basadas en el concepto de conjuntos difusos, también denominados conjuntos borrosos.

Mientras que la teoría de conjuntos tradicional define «ser miembro de un conjunto» como un predicado booleano, la teoría de conjuntos difusos permite representar el «ser miembro de un conjunto» como una distribución de posibilidades.

El padre de la lógica difusa es Zadeh (1965), el cual afirma que «a medida que la complejidad de un problema aumenta, disminuye la posibilidad de analizarlo en términos precisos». Es decir, a medida que el problema es más complejo, es más difícil construir métodos cuantitativos.

La lógica difusa es una extensión de la lógica tradicional, la cual resulta natural y útil, porque las personas normalmente razonamos bien con términos imprecisos. Al perder el enfoque booleano, se cubren más aspectos con una regla y se razona de forma sencilla pero más potente.

La lógica difusa ha sido aplicada en áreas tan diversas como control, medicina, biología, ecología, economía o política. En la ingeniería de control se aplicó por primera vez en 1974 (Mamdani y su grupo del Queen Mary College en el Reino Unido lo aplicaron para el control de una máquina de vapor).

Los sistemas basados en lógica difusa pueden controlar más adecuadamente procesos que estén gobernados por reglas intuitivas que difícilmente pueden expresarse matemáticamente.

La gran potencia de esta metodología programable se debe a la posibilidad de expresar operaciones y controlar las reglas del sistema mediante palabras de uso cotidiano. Por ejemplo, en un sistema de control de un ascensor, podría programarse:

SI está cerca de un piso AND hay orden de parar

ENTONCES disminuir la velocidad

En este caso una entrada al sistema de control sería la posición del ascensor y, como «cerca» es un conjunto difuso (concepto que se explicará a continuación), el valor de verdad de la premisa y, por tanto, el de la velocidad, varían de acuerdo con dicha posición.

La forma de expresar las reglas de operación mediante palabras permite controlar procesos sencillos con una decena de reglas y procesos complejos con 30 o 40, reduciendo considerablemente la cantidad de código de programación y, por tanto, el tiempo de diseño, el tiempo de desarrollo de un prototipo, la carga computacional y de memoria, etc.

Otra ventaja del control difuso es la fácil modificación del funcionamiento del sistema, que puede llevarse a cabo cambiando algunas premisas y operaciones, o añadiendo reglas (el criterio de comportamiento del sistema va implícito en las reglas). Sin embargo, en un sistema convencional, un pequeño cambio puede requerir la derivación completa de nuevas ecuaciones. El control difuso no necesita de una etapa previa de obtención del modelo matemático del proceso.

Conjuntos difusos

En lógica tradicional la pertenencia a un conjunto es binaria (pertenece o no pertenece). La lógica difusa extiende el concepto de pertenencia asignando una probabilidad de pertenencia (ver Figura 1).

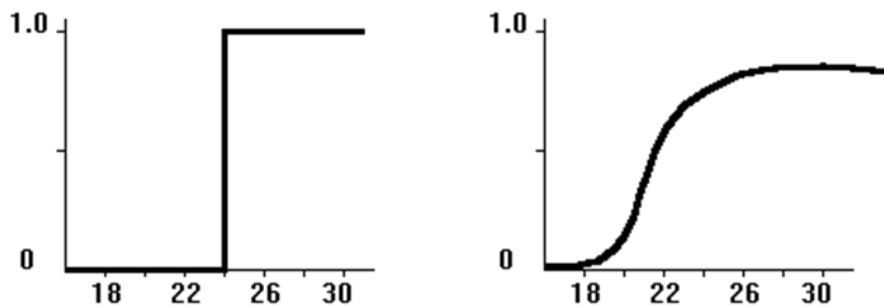


Figura 1. Pertenencia a un conjunto en lógica tradicional (gráfico de la izquierda) y en lógica difusa (gráfico de la derecha).

Se muestran a continuación ejemplos para ilustrar las diferentes lógicas. Primero se considera el conjunto X de los números reales entre 0 y 10, que se llamará **conjunto universal**. Después, se define un subconjunto A de X de todos los números reales que están en el rango entre 5 y 8.

$$A = [5, 8]$$

La función característica del conjunto A asigna un número, 1 o 0 para cada elemento en X , y el valor depende de si el elemento pertenece al subconjunto de A o no. Este resultado se muestra en la Figura 2:

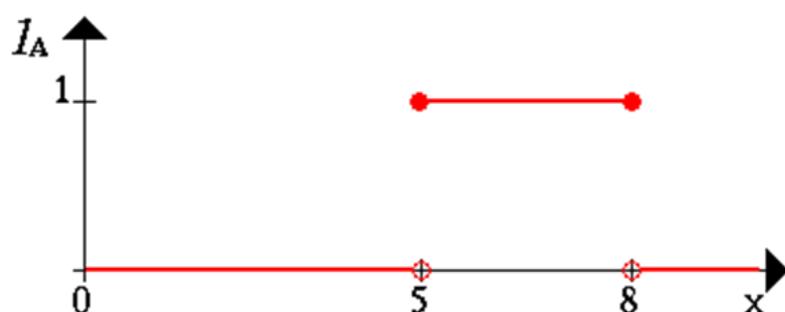


Figura 2. Ejemplo de pertenencia a un conjunto en lógica tradicional.

Los elementos del conjunto A tienen asignados el valor de 1 porque pertenecen a ese conjunto y cada elemento que tiene asignado el número 0 es un elemento que no está en el conjunto A.

Este concepto es suficiente para aplicaciones en muchas áreas. Pero se necesita buscar solución a situaciones donde se requiere flexibilidad. Para ilustrar cómo modelar esta flexibilidad se considera el siguiente ejemplo que consiste en describir a la gente de peso saludable.

Formalmente se puede denotar así:

$$A = \{\text{Conjunto de personas de peso normal}\}$$

Entonces, en general, se toma como límite inferior las personas con Índice de Masa Corporal (IMC) igual a 18.5 mientras que como límite superior se utiliza un IMC igual a 25. Entonces el conjunto A se define como un intervalo abrupto (Ver Figura3):

$$A = [18.5, 25]$$

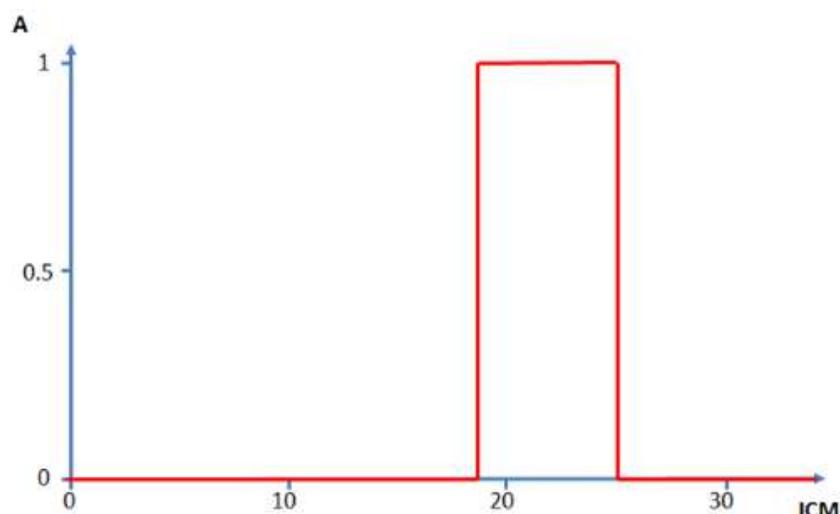


Figura 3. Representación del conjunto abrupto «personas de peso normal».

Surge ante esta representación la pregunta siguiente: ¿Una persona con IMC igual a 25 tiene un peso normal mientras que una persona con IMC igual a 25.1 tiene sobrepeso? Si se utilizan conjuntos abruptos un sistema sí asignaría de esa manera los pesos, mientras que si se definen conjuntos difusos se puede definir un grado de peso y un grado de sobrepeso para las personas cuyo IMC está cerca del límite establecido.

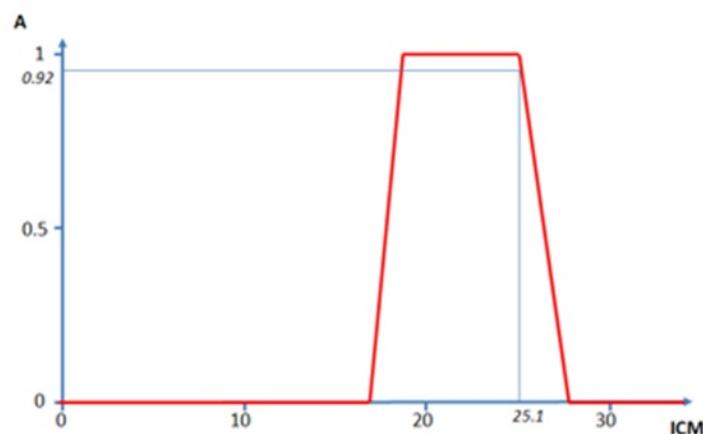


Figura 4. Representación del conjunto difuso «personas de peso normal».

Tal y como se muestra en la Figura 4, hay valores de IMC a los que se les asigna un valor de la función entre 0 y 1, que indica el grado de pertenencia de cada elemento al conjunto A. Si está en 1 significa que el elemento pertenece al conjunto A totalmente y si está en 0 significa que el elemento no pertenece al conjunto A. Para concretar en este ejemplo, si la persona tiene un IMC igual a 25.1 se le considera una pertenencia del 92% al conjunto de personas de peso normal.

La representación del conjunto difuso viene establecida por la función de pertenencia.

Función de pertenencia (): Función que define el conjunto difuso A en el universo U. Para un valor x del universo U, la función de pertenencia $\mu_A(x)$ indica el grado de pertenencia de x al conjunto A, pudiendo ser este grado un valor entre 0 y 1 inclusive. Esto se denota como:

$$\mu_A: U \rightarrow [0,1]$$

A también se le conoce como el **valor de verdad** porque representa el grado en que una proposición es verdadera.

Variables lingüísticas

Una variable lingüística es una variable cuyos valores son términos lingüísticos y se define como un quinteto $(X, T(X), U, G, M)$, donde:

- ▶ X: Nombre de la variable.
- ▶ $T(x)$: Conjunto de valores lingüísticos (atributos, adjetivos) de x.
- ▶ U: Universo de discurso (rango de posibles valores de la variable x).
- ▶ G: Regla sintáctica para generar los valores lingüísticos de x.
- ▶ M: Regla semántica para asociar a cada término lingüístico su significado que es un conjunto difuso en U. Cada conjunto difuso por tanto representa un valor lingüístico de la correspondiente variable lingüística.

Por ejemplo (ver Figura 5):

$X = T = \text{Temperatura}$

$T(x) = \{\text{baja}, \text{muy baja}, \text{moderadamente alta}, \dots\}$

$U = [100^\circ \text{C}, 500^\circ \text{C}]$

G: el término lingüístico «baja» se utiliza para los valores de T por debajo de 10°C

M: el conjunto difuso representando el valor «moderado» se define para una temperatura alrededor de 250°C con una función de pertenencia μ_{moderado} .

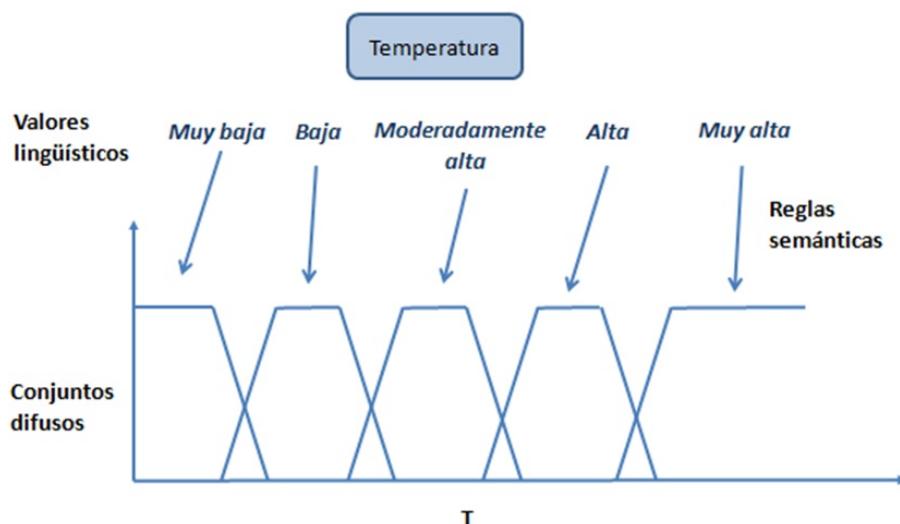


Figura 5. Variable lingüística temperatura.

A los conjuntos difusos se les puede aplicar unos **modificadores lingüísticos** (*hedge*), que son funciones matemáticas que modelan un adverbio lingüístico, modificando la forma de los conjuntos difusos. Se utilizan para cuantificar las etiquetas de una variable lingüística: muy, algo, bastante, poco, etc.

Existen definiciones de modificadores lingüísticos comúnmente utilizadas como por ejemplo las mostradas en la Figura 6.

Modificador «muy»

$$\mu_A^{muy}(x) = [\mu_A(x)]^2$$



Modificador «más o menos»

$$\mu_A^{más o menos}(x) = \sqrt{\mu_A(x)}$$

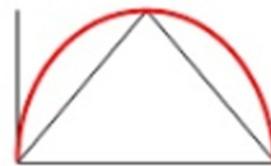


Figura 6. Modificadores lingüísticos «muy» y «más o menos».

Como se puede observar en la Figura 6 el modificador «muy» reduce el grado de pertenencia al conjunto modificado. Así, si la función está representando la altura y a un hombre le corresponde un grado de pertenencia de 0.9 al conjunto de «personas altas», le correspondería un grado de pertenencia de $0.9^2=0.81$ al conjunto de «personas muy altas».

Por el contrario, como se puede observar en la Figura 6, el modificador «más o menos», realiza una operación de expansión, relajando la condición de pertenencia al conjunto y, por tanto, se aumenta el grado de pertenencia de los elementos al conjunto. Si este operador es aplicado al ejemplo de la altura previo, el hombre tiene una pertenencia al conjunto de «personas más o menos altas» de 0.94.

Reglas difusas

Las variables lingüísticas se utilizan en reglas difusas. Una regla difusa consiste en una expresión:

SI x es a

ENTONCES y es b

Donde x e y son variables lingüísticas y a y b son valores lingüísticos que vienen determinados por conjuntos difusos definidos en el universo de discurso de las variables x e y , respectivamente.

SI

temperatura es alta

ENTONCES

riesgo de avería es moderado

El rango de la variable temperatura (el universo de discurso) está establecido entre 100 y 500º C. En la regla se especifica el valor lingüístico «alta» correspondiente a uno de los conjuntos difusos definidos mediante funciones de pertenencia a lo largo de ese universo de discurso.

Otros conjuntos difusos corresponderán a los valores «baja», «moderadamente alta» o «muy alta», por ejemplo. Igualmente, en el consecuente se encuentra una variable lingüística con un valor lingüístico asociado a un determinado conjunto difuso.

A diferencia de las reglas clásicas o no difusas (si no se emplean mecanismos tales como factores de certeza), en que si el antecedente se cumple entonces el consecuente se cumple y se considera totalmente cierto, en las reglas difusas el antecedente se cumple parcialmente y este grado de veracidad se propaga al consecuente y, por tanto, el consecuente también será parcialmente veraz en un grado análogo.

Tanto antecedentes como consecuentes de las reglas difusas pueden tener múltiples condiciones. En el caso de los antecedentes se han de aplicar operadores para generar un valor que afectará por igual a todos los consecuentes. Estos operadores son denominados **métodos de implicación** y se describen en el siguiente apartado.

10.6. Inferencia difusa

En un sistema de control difuso lo primero que hay que hacer es determinar los conjuntos difusos que describen el problema para después, establecer la base de reglas del sistema.

Posteriormente, con estas reglas y a partir de los conjuntos difusos de las variables de entrada, se infieren los conjuntos difusos de las variables de salida; asimismo, luego estos deben combinarse de alguna forma para, en último lugar, obtener conclusiones precisas. Hay variaciones en esta aproximación, pero en esencia, un sistema difuso está compuesto por los elementos que se muestran en la Figura 7 y se explican a continuación.

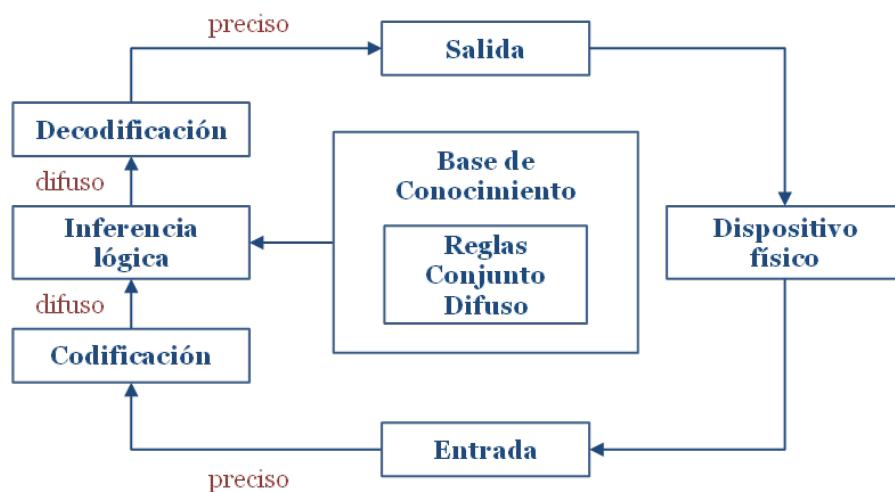


Figura 7. Sistema de control difuso básico.

Codificación (fuzzification)

En esta etapa se toman las variables de entrada al sistema de tipo abrupto o preciso y se les asigna un valor de pertenencia a los distintos conjuntos difusos definidos en el universo de cada variable de entrada.

Implicación y composición

El siguiente paso es interpretar las operaciones SI-ENTONCES de las proposiciones difusas. Por lo tanto, una vez definida la base de reglas se debe establecer la estrategia de implicación y el método de composición del sistema de control difuso.

Los **métodos de implicación** calculan el grado de verdad de las premisas de las reglas y lo aplican a la conclusión, dando como resultado un subconjunto difuso asignado a cada una de las variables de salida. Normalmente, cuando en el antecedente se aplica el operador AND (intersección) se suele utilizar uno de los dos métodos de implicación descritos a continuación ([Fuzzy Logic Toolbox Documentation](#)—MathWorks España)

- ▶ **MIN:** calcula el mínimo de la función de pertenencia de los datos de entrada para cada uno de los antecedentes. Gráficamente, la función de pertenencia de la salida quedará truncada a la altura del grado de verdad de la premisa de la regla.
- ▶ **PRODUCT:** calcula el producto de la función de pertenencia de los datos de entrada para cada uno de los antecedentes. Gráficamente, la función de pertenencia de la salida quedará escalada a la altura del grado de verdad de la premisa de la regla.

Aunque el método MIN es el más comúnmente utilizado, el método PRODUCT, o de escalado, pierde menos información al mantener la forma de la función de pertenencia de salida. Sin embargo, el método MIN se suele preferir porque supone menos coste de cálculo (Negnevitsky, 2011).

En el caso de que se aplique el operador OR correspondiente a la operación lógica de unión se utilizan por ejemplo los siguientes dos métodos de implicación:

- ▶ **MAX:** calcula el máximo de la función de pertenencia de los datos de entrada para cada uno de los antecedentes.
- ▶ **PROBOR** (*probabilistic OR*): calcula la siguiente operación sobre las funciones de pertenencia:

$$probor(\mu_A(x), \mu_B(x)) = \mu_A(x) + \mu_B(x) - \mu_A(x) \bullet \mu_B(x)$$

Una vez evaluadas todas las reglas activas en un momento dado y obtenidos los conjuntos de salida modificados, es necesario combinar todos los subconjuntos difusos asignados a cada variable de salida en un único subconjunto difuso. Para ello, se puede aplicar uno de los siguientes **métodos de composición o agregación** (MathWorks, 2000):

- ▶ **MAX:** obtiene como conjunto de salida el valor máximo de los conjuntos de cada una de las reglas.
- ▶ **SUM:** toma como conjunto de salida la suma de los conjuntos de cada una de las reglas. Por lo tanto, este método de composición presenta el problema de que puede dar lugar a valores mayores que 1. Para evitar esto, se puede usar la suma limitada (*bounded sum*) que toma el mínimo entre el resultado del método SUM y el número 1 (Lowen, 1995).

En la práctica el método más común de implicación/composición es el **método min-max**.

En cualquier caso, mientras que de otros componentes sí que existen aproximaciones y sus propiedades teóricas son bien conocidas, en la literatura hay

poca información sobre cómo se debe elegir el método de composición más adecuado a la hora de desarrollar un sistema difuso.

Se ilustra con un ejemplo cómo se realiza **la inferencia** en un sistema experto difuso en el que se han determinado los conjuntos difusos mostrados en la Figura 8 y se han definido las siguientes reglas:

REGLA 1:

SI x es A

ENTONCES z es J

REGLA 2:

SI x es A

AND y es D

ENTONCES z es K

REGLA 3:

SI x es B

OR y es E

ENTONCES z es L

Se supone que los valores de entrada x e y toman los valores discretos:

$$x = 11.2$$

$$y = 16.3$$

En la Figura 8 se muestra el resultado de asignar a estas dos variables unos valores de pertenencia a los diversos conjuntos difusos determinados para el problema.

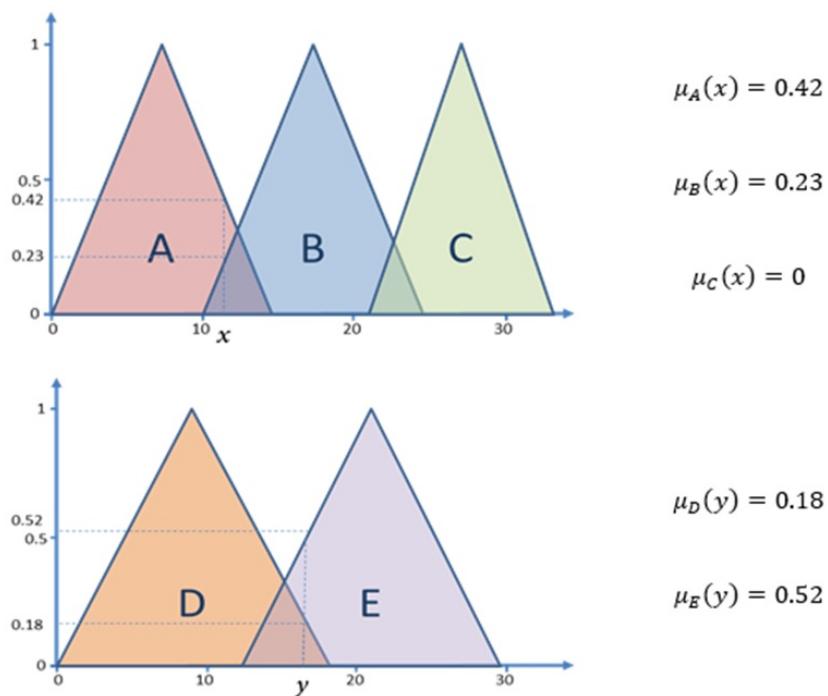


Figura 8. Funciones de pertenencia a los conjuntos difusos definidos en el universo de las variables x e y del ejemplo.

A continuación, se aplican los operadores de implicación que propagan al consecuente el grado de verdad del antecedente. La regla 1 es la más sencilla ya que únicamente tiene un antecedente cuya veracidad directamente se propaga al consecuente como se muestra en la Figura 9.

REGLA 1:

SI x es A (0.42)

ENTONCES z es J (0.42)

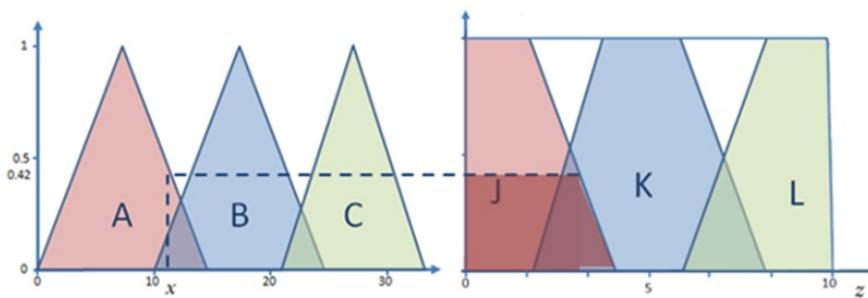


Figura 9. Evaluación de la regla 1.

En el caso de las reglas 2 y 3, dado que tienen dos antecedentes, se ha de aplicar un operador relativo a las operaciones AND y OR para obtener un valor que se propaga al consecuente. En la regla 2 se aplica el método MIN para el operador AND (Figura 10), mientras que en la regla 3 se aplica el método MAX para el operador OR (Figura 11).

REGLA 2:

SI x es A (0.42)

AND y es D (0.18)

ENTONCES z es K (0.18)

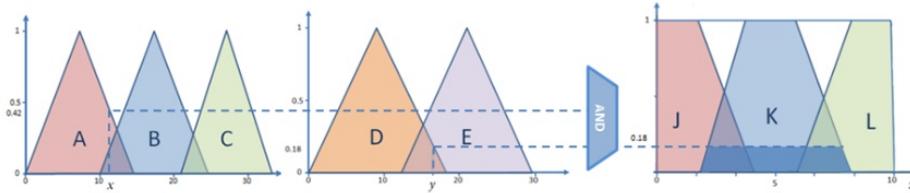


Figura 10. Inferencia de la regla 2. Ejemplo de método de implicación MIN.

REGLA 3:

SI x es B (0.23)

OR y es E (0.52)

ENTONCES z es L (0.52)

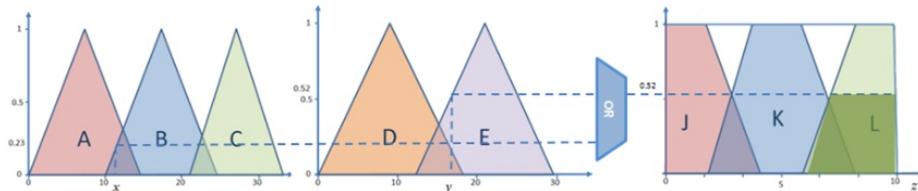


Figura 11. Inferencia de la regla 3. Ejemplo de método de implicación MAX.

La activación de las tres reglas del sistema de ejemplo ha dado lugar a las siguientes conclusiones respecto a z :

z es J (0.42)

z es K (0.18)

$z \text{ es } L(0.52)$

A continuación, se ha de aplicar el operador de agregación con el fin de unificar la salida en un único conjunto difuso. En la Figura 12 se muestra el resultado de esta operación para el caso de aplicar el método de agregación MAX, que es la zona sombreada en gris tras la aplicación del operador MAX.

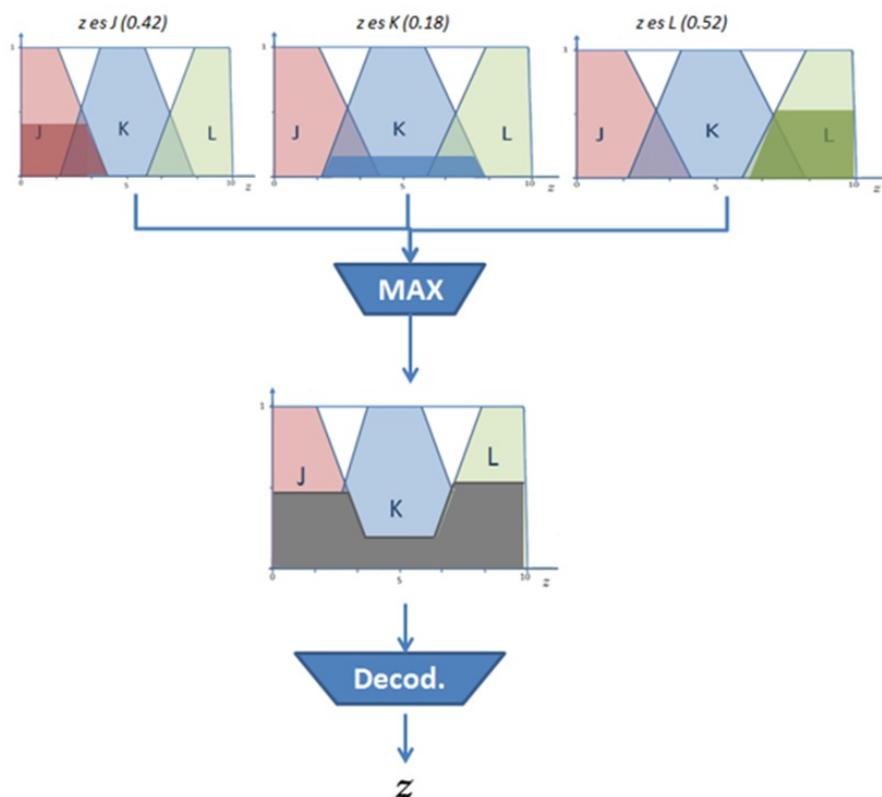


Figura 12. Agregación de las conclusiones de las reglas.

El paso posterior es la interpretación de ese conjunto difuso resultado de la agregación (marcado en gris en la Figura 12), transformándolo en un valor de la variable de salida z .

Decodificación

Una vez aplicados los métodos de composición de las reglas, la salida es un

conjunto difuso; sin embargo, para poder usar esta información como señal de control, esta tiene que ser un número preciso. Por lo tanto, como último paso, en cualquier sistema de control difuso se debe definir el **método de decodificación o defuzzification** a emplear.

En la decodificación se pueden utilizar diferentes métodos (Jantzen, 1999):

- ▶ **Centroid o centroide:** utiliza como salida del sistema, el centro de gravedad de la función de pertenencia de salida, es decir, el punto central del área cubierta por la función de pertenencia. Matemáticamente, se calcula siguiendo la siguiente fórmula:

$$\bar{y} = \frac{\int y \cdot \mu(y) dy}{\int \mu(y) dy}$$

Donde y es la variable de salida y $\mu(y)$ es la función de pertenencia de salida.

- ▶ **MOM (Mean of Maximum):** elige como salida la media de los máximos. Esto mismo se puede ver de una manera más formal en la siguiente expresión matemática:

$$\bar{y} = \sum_{n=1}^N \frac{y_n}{N}$$

Donde y_n es la coordenada de cada máximo y N es el número de máximos.

- ▶ **SOM (Smallest of Maximum):** establece como salida el valor del menor máximo.
- ▶ **LOM (Largest of Maximum):** establece como salida el valor de mayor máximo.

- ▶ En el caso de que solo haya un máximo, MOM, SOM y LOM proporcionarían el mismo resultado.
- ▶ **Bisector:** establece como salida el punto donde el conjunto difuso es dividido en dos subregiones de igual área. En muchos casos, coincide con el resultado obtenido a través del centroide.

$$\bar{y} = \left\{ y \vee \int_{min}^y \mu(y) dy = \int_y^{max} \mu(y) dy \right\}$$

Donde y es la variable de salida, $\mu(y)$ es la función de pertenencia de salida, min es el valor más a la izquierda del universo de la variable de salida y max es el valor más a la derecha del universo de la variable de salida.

Por último, es importante remarcar que, al igual que la construcción de un sistema experto convencional, la construcción de un sistema difuso es un proceso iterativo que implica definir los conjuntos difusos, las reglas difusas, seleccionar los métodos de implicación, composición, decodificación, evaluar los resultados e ir ajustando el sistema iterativamente hasta conseguir los requisitos especificados.

10.7. Aplicaciones y ejemplos de implementación

En este caso vamos a emplear la librería scikit-fuzzy para realizar un ejemplo basado en lógica difusa con Python. Vamos a seguir uno de los tutoriales incluidos en la web de la librería (ver sección A fondo al final del tema).

Instalamos el paquete si no lo tuviéramos ya en nuestro sistema:

```
PS C:\Users\xxx> pip install scikit-fuzzy
```

Vamos a crear un sistema de control basado en lógica difusa que modele cuánta propina daría un cliente en un restaurante. A la hora de dar propina un cliente tiene en cuenta el servicio y la calidad de la comida, digamos que calificados entre 0 y 10.

En base a esto, el cliente decide dejar una propina de entre el 0 % y el 25 % (la problemática está planteada desde el punto de vista de países en los cuales la propina conforma una gran parte del sueldo variable del servicio).

Formulemos el problema a continuación.

Antecedentes (entradas):

- ▶ Servicio
 - Universo:
 - Conjunto
- ▶ Calidad de la comida:
 - Universo: ¿cuán sabrosa estaba la comida, en una escala del 1 al 10?
 - Conjunto difuso: mala, decente, genial.

Consecuentes (salidas):

- ▶ Propina:
 - Universo: ¿de qué porcentaje debería ser la propina en una escala del 0% al 25%?
 - Conjunto difuso: bajo, medio, alto.

Reglas:

- ▶ Si el servicio fue pobre O la calidad de la comida fue pobre, ENTONCES la propina será baja.
- ▶ Si el servicio fue medio, ENTONCES la propina será media.
- ▶ Si el servicio fue bueno O la calidad de la comida fue buena, ENTONCES la propina será alta.

Ejemplo de uso:

- ▶ Si el cliente la dice al sistema:
- ▶ El servicio fue un 9.8.
- ▶ La calidad fue 6.5.
- ▶ Recomendaría dejar:
 - 20.2 % de propina.

Creamos el sistema de control difuso y mostramos mediante gráficas la pertenencia de las diferentes variables (calidad del servicio, calidad de la comida y propina):

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# New Antecedent/Consequent objects hold universe variables and
membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-
membership function population is possible with .automf(3, 5, or
7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a
familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

quality['average'].view()
service.view()
tip.view()

plt.show()
```

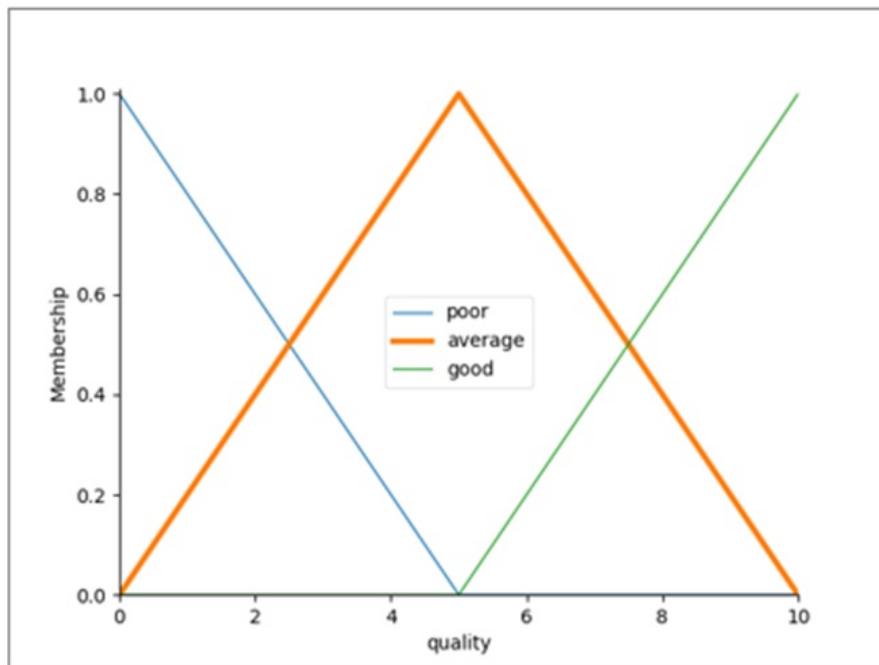


Figura 13. Gráfica de calidad de la comida.

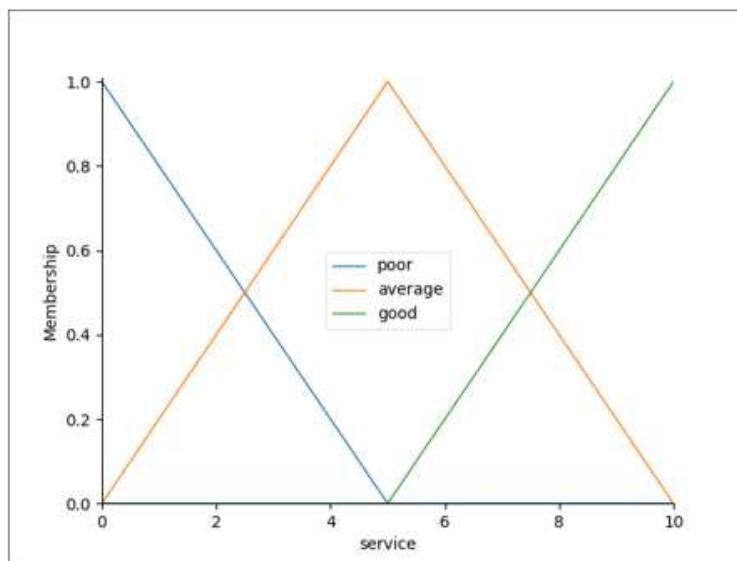


Figura 14. Gráfica de calidad del servicio.

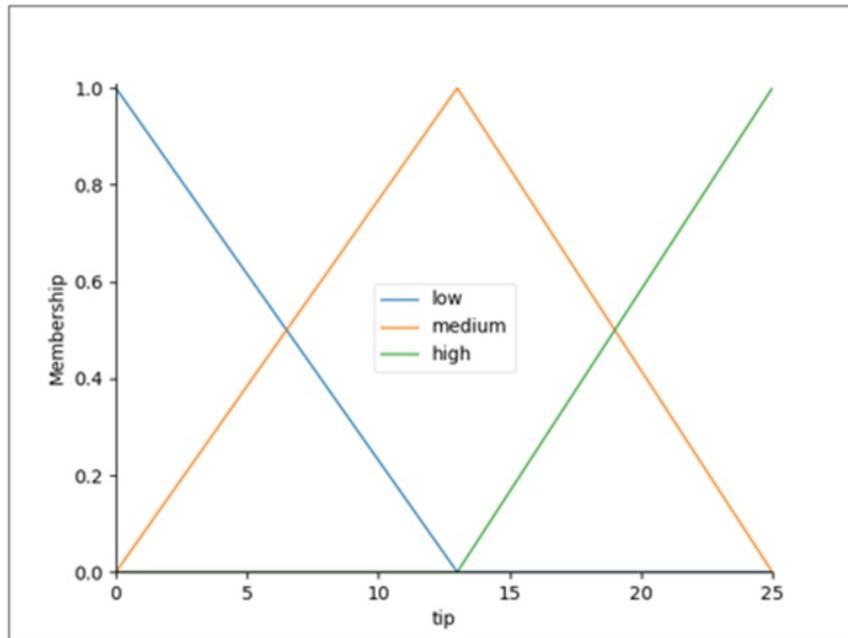


Figura 14. Gráfica de la propina.

Incluyamos ahora la codificación de las reglas establecidas previamente y visualicemos un grafo de ejemplo para la primera regla, recordemos:

- ▶ Si el servicio fue pobre O la calidad de la comida fue pobre, ENTONCES la propina será baja.

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# New Antecedent/Consequent objects hold universe variables and
membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

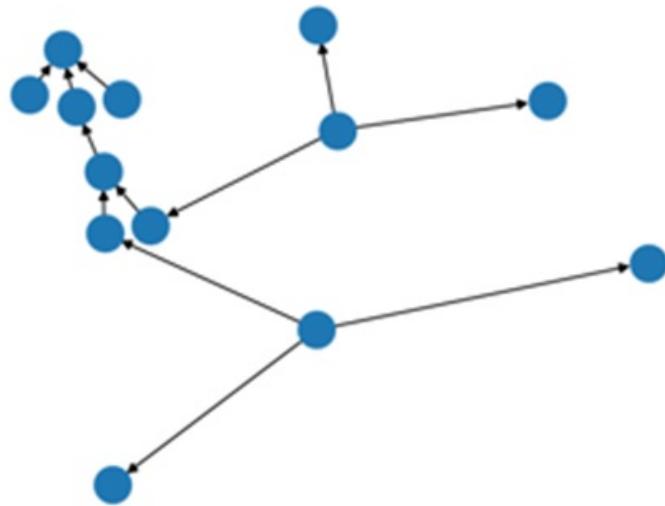
# Auto-
membership function population is possible with .automf(3, 5, or
7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a
familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
)

rule1.view()

plt.show()
```



Finalmente, construimos un sistema de control difuso y hacemos uso de un objeto simulador, para comprobar cuál sería la propina si una cliente valora la calidad de la comida en 6.5 y el servicio en 9.8.

```

import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt

# New Antecedent/Consequent objects hold universe variables and
membership
# functions
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Auto-
membership function population is possible with .automf(3, or
7)
quality.automf(3)
service.automf(3)

# Custom membership functions can be built interactively with a
familiar,
# Pythonic API
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])
)

tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

tipping = ctrl.ControlSystemSimulation(tipping_ctrl)

# Pass inputs to the ControlSystem using Antecedent labels with
Pythonic API
# Note: if you like passing many inputs all at once, use .inputs
(dict_of_data)
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8

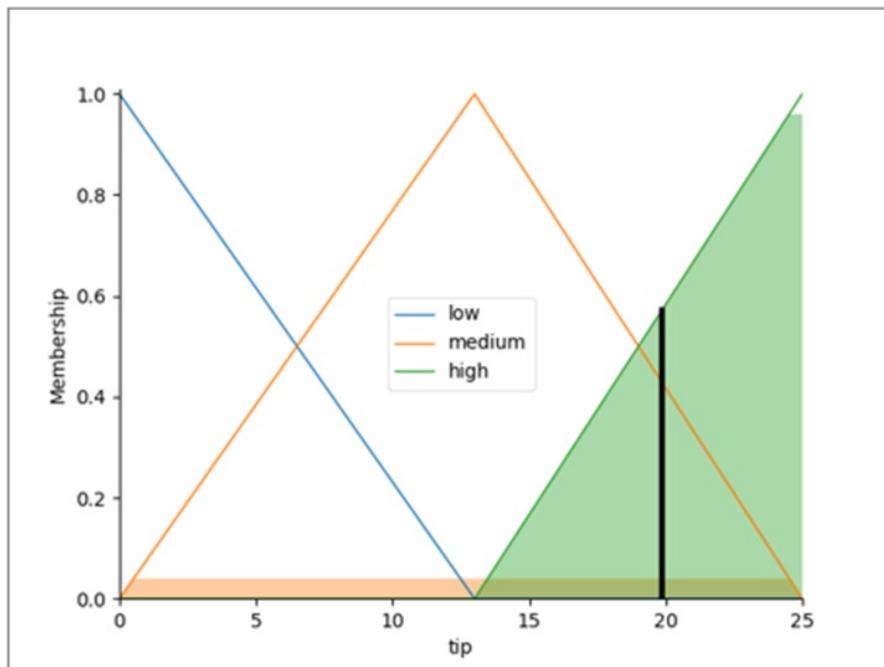
# Crunch the numbers
tipping.compute()

print(tipping.output['tip'])
#> 19.847607361963192
tip.view(sim=tipping)

plt.show()

```

Como vemos, el resultado es una propina del 19.84 %, lo cual vemos también gráficamente durante la ejecución del código.



10.8. Referencias bibliográficas

Jantzen, J. (1999). *Design Of Fuzzy Controllers*. Tech. report no 98-E 864 (design). Technical University of Denmark, Department of Automation.

Lowen, R. (1995). Ordering Fuzzy Real Quantities. En *Fuzzy Logic and Intelligent Systems*, ed. H. Li y M.M. Gupta, 69-83. Massachusetts: Kluwer Academic Publishers.

MathWorks. (2020). *Fuzzy Logic Toolbox*. User's Guide.

Murphy, K. P. & others. (2006). Naive bayes classifiers. *University of British Columbia*, 18, 60.

Negnevitsky, M. (2011). *Artificial Intelligence. A Guide to Intelligent Systems*. UK: Addison-Wesley.

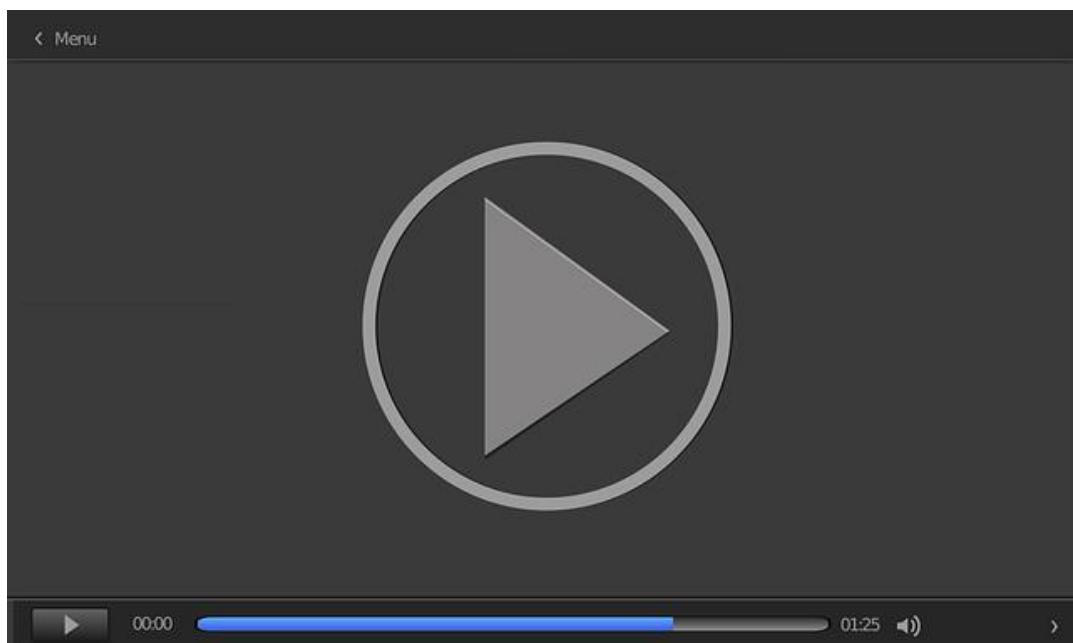
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81-106.
<https://doi.org/10.1007/BF00116251>

Shortliffe, E.H. & Buchanan, B.G. (1975). A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23(3/4), 351-379.

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3), 338-353.

FuzzyCLIPS: introducción práctica

En esta lección magistral se realiza una demostración del uso de la herramienta FuzzyCLIPS, se ejecutan algunos comandos básicos y se muestran algunas opciones interesantes del interfaz de la herramienta.



10. FuzzyCLIPS: introducción práctica

Accede al vídeo:

<https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=9dbc5f20-690c-476e-91b7-aff800fbc5ec>

CLIPS: guía del usuario

Accede a la guía desde el aula virtual o a través de la siguiente dirección web:

<http://clipsrules.sourceforge.net/documentation/v630/ug.pdf>

CLIPS (*C Language Integrated Production System*) es una herramienta para el desarrollo de sistemas expertos diseñada por el centro espacial Johnson (NASA) con el objetivo de ser portable, de bajo costo y de fácil integración con otros sistemas. Para facilitar estos objetivos, CLIPS está programado en C y existen versiones de CLIPS enteramente programadas en ADA y en Java. En el siguiente enlace se puede acceder a una **guía de fácil lectura y muy práctica sobre las características básicas de CLIPS**. Permite de manera rápida aprender a programar un sistema experto sencillo, con ejemplos muy ilustrativos y generales. Está dirigida a usuarios con ninguna o poca experiencia en sistemas expertos.

Guía práctica de FuzzyCLIPS

Accede a la guía desde el aula virtual o a través de la siguiente dirección web:

<http://alumni.cs.ucr.edu/~vladimir/cs171/quickfuzzy.pdf>

Guía práctica que consiste en un resumen de la Guía de Usuario de FuzzyCLIPS incluida en el fichero de descarga de esta herramienta, cuyo autor es Bob Orchard. Permite de manera rápida aprender cómo programar un sistema experto sencillo que maneje incertidumbre o imprecisión, con ejemplos muy ilustrativos y didácticos.

Guía del usuario de Fuzzy Logic Toolbox (MATLAB)

Accede a la guía desde el aula virtual o a través de la siguiente dirección web:

http://www.mathworks.com/help/pdf_doc/fuzzy/fuzzy.pdf

La herramienta *Fuzzy Logic Toolbox* proporciona funciones de MATLAB y otras herramientas que permiten analizar, diseñar y simular sistemas basados en lógica difusa. Aunque la guía está destinada a los usuarios de esta herramienta, es muy didáctica y útil para comprender el funcionamiento de sistemas expertos difusos en general. Específicamente, los métodos de implicación o los de composición son descritos de una manera clara y didáctica con ilustraciones.

El primer sistema manejando factores de certeza: MYCIN

Shortliffe, E.H. & Buchanan, B.G. (1975). A model of inexact reasoning in medicine.

Mathematical Biosciences, 23(3/4), 351-379.

<https://aitopics.org/download/classics:C246A971>

Los autores del artículo proponen los factores de certeza para manejar la incertidumbre en el sistema MYCIN. Mediante este sistema se pretendía diagnosticar infecciones de sangre y meningitis. Dado que no había datos estadísticos fiables sobre el problema, los creadores de MYCIN utilizaron un factor de certeza para indicar la confianza o certeza del experto humano.

Introducción a la lógica difusa

Accede al artículo desde el aula virtual o a través de la siguiente dirección web:

<http://www.cs.us.es/~fsancho/?e=97>

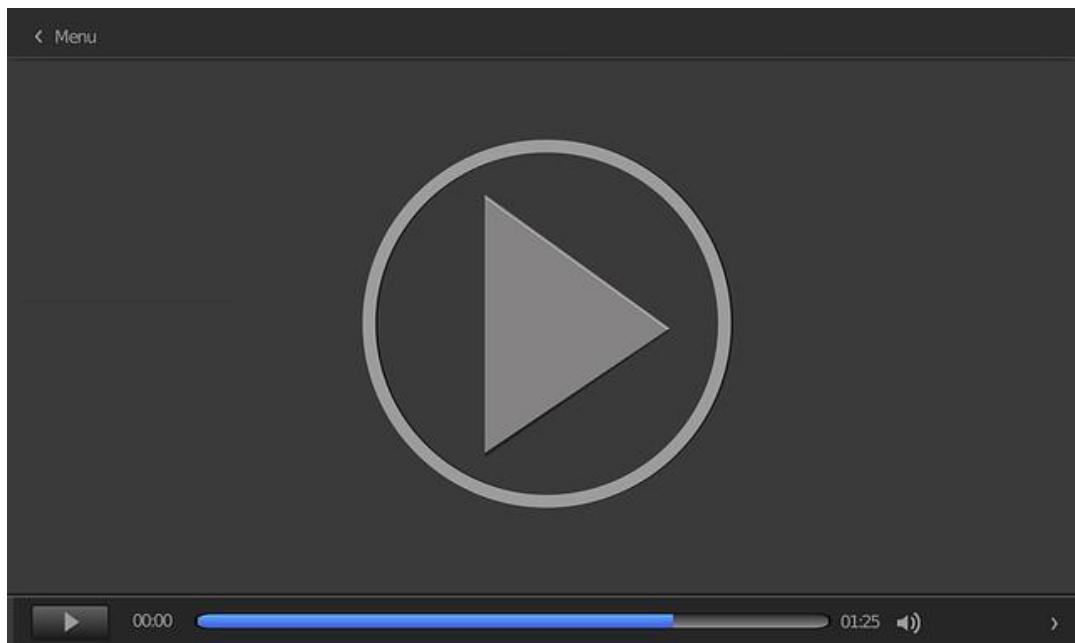
El profesor Fernando Sancho Caparrini expone en un artículo muy completo una gran introducción a la lógica difusa.

Lógica difusa: introducción

Accede al vídeo desde el aula virtual o a través de la siguiente dirección web:

<https://www.youtube.com/watch?v=xLFNUo0mTZE>

El Doctor Erik Zamora introduce en el siguiente vídeo los conceptos de lógica difusa.



Accede al vídeo:

<https://www.youtube.com/embed/xLFNUo0mTZE>

Scikit Fuzzy

Accede a la página web desde el aula virtual o a través de la siguiente dirección:

<https://pythonhosted.org/scikit-fuzzy/>

Documentación y ejemplos de la librería Scikit Fuzzy para Python, empleada en el ejemplo de implementación del tema.

1. Completa la siguiente sentencia con la opción adecuada: «Si no se sabe con certeza que el consecuente de las reglas es resultado de las evidencias en los antecedentes, puedo modelar el sistema utilizando [...]» (varias respuestas posibles):

 - A. Variables difusas.
 - B. Conjuntos difusos.
 - C. Factores de certeza.
 - D. Inferencia bayesiana.

2. Completa la siguiente sentencia con la opción adecuada: «Si no se puede precisar el valor exacto de las variables en los antecedentes de las reglas, se puede diseñar el sistema utilizando [...]»:

 - A. Factores de certeza.
 - B. Teorema de Bayes.
 - C. Lógica difusa.
 - D. Grados de certeza.

3. Indica cuales de las siguientes afirmaciones son ciertas:

 - A. Una limitación de la inferencia bayesiana es que puede implicar la necesidad de conocer un gran número de probabilidades.
 - B. Una ventaja de la inferencia bayesiana es que el coste computacional es bajo.
 - C. Una ventaja de utilizar factores de certeza frente a inferencia bayesiana es que el primero implica menos coste computacional.
 - D. Un factor de certeza puede tener un valor negativo.

- 4.** Indica la opción que finalice de manera correcta la siguiente frase: «En una regla cuyos antecedentes presentan una conjunción, el factor de certeza del consecuente se calcula de la siguiente manera [...]»:
- A. El producto del máximo factor de confianza de los antecedentes por el factor de confianza del consecuente.
 - B. El producto del mínimo factor de confianza de los antecedentes por el factor de confianza del consecuente.
 - C. No se utilizan factores de confianza en los antecedentes.
 - D. Ninguna de las anteriores opciones es correcta.
- 5.** Indica cuáles de las siguientes afirmaciones son correctas:
- A. Una gran ventaja de la lógica difusa es poder expresar las reglas mediante palabras de uso cotidiano.
 - B. Las reglas en la lógica difusa emplean variables numéricas.
 - C. La lógica difusa se basa en el uso de conjuntos difusos.
 - D. La función de pertenencia indica si una variable pertenece o no pertenece a un conjunto.
- 6.** Indica cuáles de las siguientes afirmaciones son correctas:
- A. Una variable lingüística es una variable cuyos valores son términos lingüísticos.
 - B. Los modificadores lingüísticos modelan adjetivos.
 - C. Los modificadores lingüísticos modifican la forma del conjunto difuso.
 - D. Las reglas difusas emplean variables lingüísticas.

- 7.** Indica cuáles de las siguientes afirmaciones son correctas:
- A. Las entradas a un sistema de control difuso son variables precisas o abruptas.
 - B. Las salidas de un sistema de control difuso son variables precisas o abruptas.
 - C. La base de conocimiento de un sistema de control difuso básico contiene las reglas difusas.
 - D. Un sistema de control difuso básico realiza una codificación, una inferencia lógica y una decodificación.
- 8.** Los métodos que calculan el grado de verdad de las premisas o antecedentes de las reglas y lo aplican a la conclusión se denominan:
- A. Métodos de composición.
 - B. Métodos de implicación.
 - C. Modificadores.
 - D. Métodos de decodificación.
- 9.** Completa la siguiente frase. El método centroide es un método de:
- A. Composición.
 - B. Implicación.
 - C. Decodificación.
 - D. Codificación.
- 10.** ¿A qué se refiere el método min-max?
- A. A un método de implicación y composición.
 - B. A un método de composición.
 - C. A un método bayesiano.
 - D. A un método de decodificación.