



Inteligencia Artificial 😊

⌚ Fecha	@15 de agosto de 2024 18:16
📍 Asignatura	Inteligencia Artificial
📎 Materiales	https://drive.google.com/drive/folders/16p796FSGY-KhZycH7csAyjdVah-JfiL6?usp=drive_link
☑ Revisado	<input type="checkbox"/>
Nº ID	5

Definición

Se relaciona con proyecciones de futuro para tomar decisiones, el factor clave es qué tan precisos somos con estas proyecciones. Algunas definiciones posibles de inteligencia artificial según diferentes autores:

- Según HOFSTADTER:
 1. Capacidad de respuesta a las situaciones de manera flexible
 2. Dar sentido a mensajes ambiguos o contradictorios
 3. Reconocer la importancia relativa de los elementos en una situación
 4. Reconocer similitudes entre situaciones por sobre las diferencias
 5. Reconocer diferencias entre situaciones por sobre las similitudes
- Según RICH:
 1. La inteligencia artificial es el producto del estudio de cómo hacer que las computadoras hagan cosas, que hasta el momento las personas hacen mejor
- Según BARR:

1. La inteligencia artificial es la parte de las ciencias computacionales que se encarga de diseñar sistemas de cómputo inteligentes, esto es, sistemas que exhiben características que asociamos con la inteligencia en el comportamiento humano
 - Según BUCHANAN y SHORTLIFFE:
 1. La inteligencia artificial es la rama de la ciencia computacionales que se encarga de la resolución de problemas de una manera simbólica no algorítmica

Una idea importante que agregan es que la inteligencia artificial no resuelva de forma algorítmica sino de forma simbólica. Es decir, sin tener la programación de como lo tiene que resolver, que aprenda y que llegue a la conclusión de como lo tiene que resolver.
 - Según BUCHANAN:
 1. Es la inteligencia artificial es la rama de las Ciencias computacionales que se encarga de las maneras de representar el conocimiento usando símbolos en lugar de números y con métodos heurísticos para procesar la información

Un concepto que agrega es la heurística, o resolución heurística que permita agregar valor con un costo adecuado. La heurística o generalización nos da un valor en la resolución con un costo adecuado
- Entonces para resumir podemos decir que inteligencia artificial es:
1. Capacidad de entendimiento y compresión
 2. Capacidad de resolver problemas
 3. Capacidad de adaptación en un entorno

Inteligencia Artificial múltiple

Ahora veamos un concepto adicional de inteligencia que es la inteligencia múltiple que apunta que una persona no puede abarcar TODAS las disciplinas o ramas de la inteligencia.



Veamos alguno de los tipos de inteligencias:

1. Inteligencia Lingüística:
 - a. Capacidad de dominar el lenguaje tanto el nativo como el de diferentes culturas

- b. Incluye la comunicación oral, escrita y gestual.
- 2. Inteligencia Lógico-matemática:
 - a. Capacidad de razonamiento lógico
 - b. Capacidad de resolución de problemas matemáticos.
 - c. Se mide por velocidad de resolución
- 3. Inteligencia visual - espacial:
 - a. Capacidad de percibir el mundo desde diferentes perspectivas
 - b. Capacidad de elaborar mapas mentales, dibujar y detectar detalles
 - c. Elaboración de un sentido personal de la estética
- 4. Inteligencia Musical
 - a. Capacidad de interpretación y composición de música
- 5. Inteligencia Corporal y Cinestésica
 - a. Comprende las habilidades corporales y motrices
 - b. Expresión de sentimientos mediante el cuerpo
- 6. Inteligencia Intrapersonal
 - a. Capacidad de acceder y reflexionar sobre los sentimientos propios
 - b. Capacidad de introspección y reflexión sobre el actuar de uno
- 7. Inteligencia Interpersonal:
 - a. Capacidad para detectar y entender circunstancias y problemas de los demás
 - b. Capacidad de advertir cosas más allá de los que los sentidos logran captar
- 8. Inteligencia Naturalista:
 - a. Capacidad de detectar, diferenciar y clasificar elementos vinculados al entorno.
 - b. Clasificación clásica o aristotélica vs conceptual

Racionalidad

La racionalidad es la capacidad que permite pensar, evaluar, entender y actuar de acuerdo a ciertos principios de mejora y consistencia, para satisfacer algún objetivo o finalidad.

El ejercicio de la racionalidad está sujeto a mejora continua. Cualquier construcción mental llevada a cabo mediante procedimientos racionales tiene por tanto una estructura lógico-mecánica distinguible (razonamiento).

Razonamiento Lógico

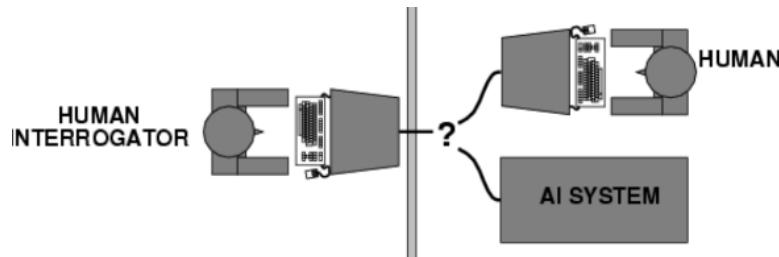
- 1. Proceso mental que implica la aplicación de la lógica
- 2. A partir de una o de varias premisas para arribar a una conclusión que puede determinarse como verdadera, falsa o posible
- 3. Se utiliza el pensamiento deductivo para realizar el análisis de las premisas

4. Permite tomar decisiones y resolver problemas de forma efectiva

Test de Turing

Conocido como juegos de imitación donde un interrogador aislado debe hacer preguntas a dos personas, una de ellas siempre debe decir la verdad y la segunda siempre debe mentir, en función de sus respuestas debe determinar cuál de las dos personas es hombre y cuál es mujer.

En este caso debe determinar si es un ser humano o una máquina quién le responde



Pensar Humanamente

Para emular al humano tenemos que mencionar el enfoque de modelado cognitivo:

El enfoque de modelado cognitivo se refiere al uso de modelos computacionales para entender y simular cómo funciona la mente humana. Este concepto surgió en los años 60, durante lo que se conoce como la "revolución de la ciencia cognitiva". Durante esta época, se combinó el conocimiento y las técnicas de la inteligencia artificial (IA) con métodos experimentales de la psicología para desarrollar teorías más precisas y verificables sobre el funcionamiento mental.

Existen dos enfoques principales en el modelado cognitivo:

1. **Enfoque TOP-DOWN (de arriba hacia abajo):** Este enfoque proviene de la Ciencia Cognitiva. Se basa en predecir y testear el comportamiento humano mediante la creación de modelos que simulen procesos cognitivos, como la percepción, el razonamiento, y la toma de decisiones. Aquí se parte de teorías generales sobre el funcionamiento de la mente y se verifica cómo estas se manifiestan en el comportamiento humano.
2. **Enfoque BOTTOM-UP (de abajo hacia arriba):** Este enfoque está más relacionado con la Neurociencia. Se enfoca en identificar directamente los datos neurológicos, es decir, en cómo las neuronas y redes neuronales contribuyen a los procesos cognitivos. Se estudian los componentes más básicos del cerebro y cómo estos generan funciones mentales complejas.

Ambos enfoques son fundamentales en el campo de la inteligencia artificial y contribuyen al desarrollo de modelos que pueden simular o replicar procesos cognitivos humanos. La combinación de estos enfoques permite una comprensión más holística de la mente, abordando tanto el nivel conductual (TOP-DOWN) como el neurológico (BOTTOM-UP).

Profundizando un poco más tenemos

Pensar Racionalmente - Leyes del Pensamiento

1. **Aristóteles y el pensamiento correcto:** Aristóteles buscó codificar lo que él consideraba el "pensamiento correcto", un proceso de razonamiento que es irrefutable si se sigue correctamente. Este esfuerzo se reflejó en la creación de los silogismos.

2. **Silogismos de Aristóteles:** Los silogismos son estructuras argumentales formales que, si las premisas son correctas, siempre conducen a conclusiones verdaderas. Este es un fundamento de la lógica formal.
3. **Desarrollo de la lógica en la antigua Grecia:** Varias escuelas filosóficas griegas continuaron desarrollando la lógica, creando notaciones y reglas de derivación que facilitaban el razonamiento estructurado.
4. **Progresos hacia 1965:** Para 1965, ya se habían desarrollado programas informáticos capaces de resolver problemas formulados en notación lógica. Estos programas representaban un avance en la IA, al poder manipular símbolos y aplicar reglas lógicas para encontrar soluciones.
5. **Tradición logicista en la IA:** La tradición logicista en IA trata de construir inteligencia artificial basándose en estos programas lógicos, que operan bajo los principios de la lógica formal.
6. **Problemas del enfoque logicista:**
 - **No todo comportamiento inteligente es lógico:** Algunas formas de inteligencia no dependen de la deliberación lógica y pueden no ser capturadas por este enfoque.
 - **Dificultad en expresar conocimiento informal:** Es complicado traducir el conocimiento cotidiano e informal a términos lógicos precisos, que es un requisito de la notación lógica.
 - **Diferencia entre teoría y práctica:** Resolver un problema "en principio" con lógica no siempre se traduce en una solución práctica, ya que la complejidad del mundo real puede hacer que las soluciones lógicas sean ineficientes o inadecuadas.

Este análisis muestra cómo el enfoque logicista de la IA se enfrenta a desafíos cuando se intenta aplicar la lógica formal a problemas del mundo real.

Actuar Racionalmente

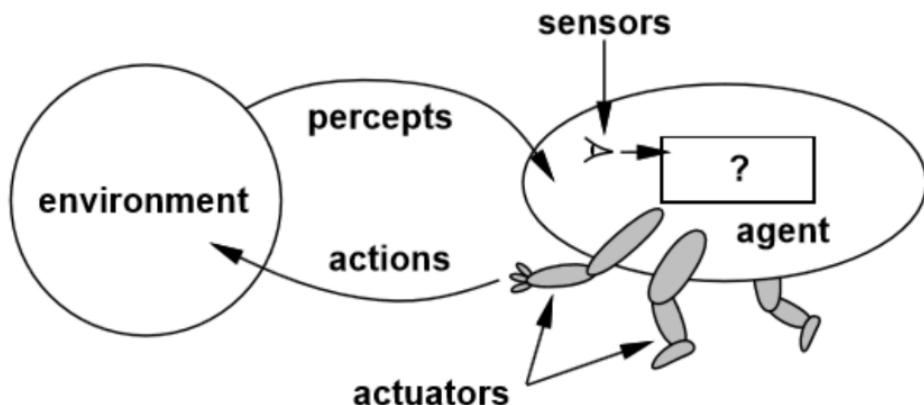
1. Agente: algo que actúa. Se espera que un agente operen autónomamente, perciba sus ambientes, persistan en un gran período de tiempo, se adapten al cambio y logren objetivos.
 - a. Comportamiento racional: hacer lo correcto.
 - b. ¿Qué es lo correcto ?
 - c. Lo que se espera que maximice el logro de la meta, dada la información disponible
 - d. No necesariamente implica pensar, por ejemplo, reflejo parpadeante, pero el pensamiento debe estar al servicio de la acción racional.
2. La diferencia con un automatismo es que agrega un razonamiento lógico para tomar decisiones.
3. Su medida es la medida de performance que implica mencionar que tan bien resuelve una tarea

Agente

1. Un agente es una entidad que percibe, actúa y toma decisiones en un entorno determinado. Pueden ser muy simples o muy complejos, dependiendo de su diseño y función específica
2. Perciben mediante sensores, y analizan en base a la información disponible
3. Actúan por medio de actuadores

4. Un agente es una función de las historias de percepción a las acciones
5. Fórmula: $[f: (P^* \rightarrow A)]$
6. Las historias de percepción describen como uno percibe y procesa la información sensorial en su entorno para tomar decisiones y llevar a cabo tareas específicas.

Si lo graficamos:



1. Una percepción (percept) es la entrada que se produce en un agente a través de sus sensores en un instante dado.
2. Una secuencia de percepciones (percept sequence, marcada como P^*) es la historia completa de todo lo que un agente ha percibido
3. Las acciones de un agente se pueden describir como una función agente que mapea una secuencia de percepciones con una acción.
4. La función de un agente artificial se implementa por un programa agente
5. Ejemplo aspiradora inteligente:
 - a. Si la aspiradora la pongo en un terreno no conocido, que lo estudie
 - b. La unidad de performance es que limpie la mayor cantidad posible gastando la menor batería posible
6. Una función agente es una descripción matemática abstracta, mientras que el programa agente es la implementación concreta, corriendo dentro de algún sistema físico.
7. El programa agente corre en una arquitectura física para producir

Racionalidad en un agente

- Un agente racional es aquel que hace lo correcto.
- Pregunta: ¿Qué es hacer lo correcto ?
- Hacer lo correcto evalúa las consecuencias del comportamiento del agente.
- Medida de performance: evalúa cualquier secuencia dada de estados del ambiente.
- Pregunta: ¿Por qué se habla de estados del ambiente y no de estados del agente ?
- El agente no define la performance, lo que define la performance es la acción que el agente realizó sobre el ambiente. Por lo tanto la medida de performance es un reflejo del estado del

ambiente luego de la acción (no del agente).

Objeciones de Turing

1. Objetivas presentadas por Turing y su relevancia actual

Turing anticipó varias objeciones a su propuesta, que agrupó en su paper. A continuación, algunas de las objeciones más significativas y su relevancia en la actualidad:

1. **La objeción teológica:** Esta objeción sostiene que pensar es una función del alma, y solo los seres humanos, que poseen alma, pueden pensar. Aunque este argumento tiene menos relevancia en el ámbito científico y tecnológico de hoy, sigue siendo un punto de debate en contextos filosóficos y religiosos.
2. **La objeción de la conciencia:** Según esta objeción, una máquina nunca podrá ser consciente, tener emociones, o experimentar la vida subjetivamente. Aunque sigue siendo una objeción válida, la IA ha avanzado mucho en la simulación de comportamientos que parecen conscientes o emocionales, como chatbots que pueden reconocer y responder a emociones humanas. Sin embargo, la conciencia sigue siendo un concepto debatido y no se ha logrado replicar en máquinas.
3. **La objeción de la creatividad:** Esta objeción sostiene que las máquinas solo pueden hacer lo que se les programa, y por tanto, carecen de creatividad. Hoy en día, existen algoritmos de IA que pueden generar obras de arte, música, e incluso escribir textos que parecen creativos, aunque sigue habiendo debate sobre si esto constituye verdadera creatividad o simplemente un resultado de patrones y datos preexistentes.
4. **La objeción matemática:** Basada en el teorema de incompletitud de Gödel, esta objeción sostiene que existen problemas matemáticos que no pueden ser resueltos por máquinas. Aunque este sigue siendo un límite teórico, en la práctica, las máquinas han demostrado ser extremadamente útiles y efectivas en resolver una amplia gama de problemas matemáticos y computacionales.
5. **La objeción de Lady Lovelace:** Esta objeción, basada en los escritos de Ada Lovelace, sostiene que las máquinas no pueden originar nada, solo pueden seguir reglas preprogramadas. Aunque sigue siendo una objeción relevante, los avances en machine learning, particularmente en el aprendizaje profundo, han demostrado que las máquinas pueden generar resultados sorprendentes que no estaban explícitamente programados.

2. Validación de las refutaciones de Turing y nuevas objeciones

Turing presentó refutaciones a estas objeciones en su paper. Algunas de sus refutaciones siguen siendo válidas hoy en día, mientras que otras han sido desafiadas o matizadas por desarrollos posteriores en IA y filosofía de la mente.

- **Refutaciones de Turing:** En muchos casos, las refutaciones de Turing siguen siendo válidas. Por ejemplo, él argumenta que la objeción de la creatividad es más una cuestión de cómo definimos la creatividad que una limitación intrínseca de las máquinas. También aborda la objeción de la conciencia sugiriendo que si una máquina puede comportarse de manera indistinguible de un ser humano, entonces no hay razón para negar que podría ser consciente.
- **Nuevas objeciones:** Desde que Turing escribió su paper en 1950, han surgido nuevas objeciones y preocupaciones:

- **El problema de la alineación:** Preocupaciones sobre si la IA actuará de acuerdo con los valores y objetivos humanos, o si podría actuar de manera perjudicial.
- **Sesgo y ética en la IA:** Las máquinas que aprenden de grandes conjuntos de datos pueden adquirir sesgos presentes en esos datos, lo que plantea problemas éticos importantes.
- **El problema de la caja negra:** A medida que los modelos de IA se vuelven más complejos, se hace más difícil entender cómo toman decisiones, lo que genera preocupación sobre su transparencia y responsabilidad.

Conclusión

Muchas de las objeciones presentadas por Turing aún tienen peso, aunque han sido abordadas en mayor o menor medida a medida que la tecnología ha avanzado. Las refutaciones de Turing en muchos casos siguen siendo persuasivas, pero la aparición de nuevas objeciones muestra que el debate sobre la inteligencia artificial está lejos de resolverse. El papel de Turing sigue siendo un pilar fundamental en la discusión, pero el campo ha evolucionado, y las nuevas preocupaciones reflejan el avance y la complejidad creciente de la IA en el siglo XXI.

▼ Actividad 1

Clasificar los ambientes de los siguientes agentes (sin el criterio de conocido o no)

1. robot que juega al fútbol - Conocido
 - a. Debe conocer las diferentes variables del entorno (arcos, gente, medidas, etc)
2. explorador autónomo en la superficie de Marte - No conocido
 - a. Si bien lo podemos preparar, le pueden pasar situaciones desconocidas.
 - b. Internamente debería guardar un estado interno del terreno. No tiene la noción al 100%}
 - c. Toda la información que recolecta le debe permitir poder tomar decisiones
3. agente para comprar libros de IA usados por internet - Conocido
 - a. Debería tener acceso a la información de internet
 - b. Debería conocer las reglas del negocio
4. agente jugador de partidos de tenis - Conocido
5. agente que practica tenis contra la pared - Conocido
6. agente para practicar salto en alto - Conocido

Agente Racional

1. Es un programa diseñado para tomar decisiones en un entorno dinámico
2. El agente percibe, analiza, toma la decisión más eficiente y actúa
3. Se retroalimenta de las respuestas del entorno, actualiza sus planificaciones
4. La racionalidad en cualquier momento dado depende de cuatro cosas:
 - a. El criterio del éxito, que determina la unidad de performance.
 - b. El conocimiento previo acerca del ambiente por parte del agente
 - c. Las acciones que el agente puede realizar

d. La secuencia de percepciones del agente hasta el momento

Estos cuatro puntos son conocidos como PEAS

- Para cada secuencia de percepciones posibles, un agente racional debería seleccionar una acción que se espera maximice su medida de performance.
- Cada decisión es tomada dada la evidencia proporcionada por la secuencia de percepciones y el conocimiento interno que el agente tenga construido en su interior.

Ejemplo de la aspiradora

¿Es racional el agente aspiradora?

- La medida de performance que define el criterio de éxito
 - Dar un punto por cada rectángulo limpio en cada paso de tiempo, en un tiempo de vida de 1000 pasos.
- El conocimiento previo acerca del ambiente.
 - La geografía del ambiente se conoce a priori, pero la distribución de la suciedad y la ubicación inicial del agente, no.
- Las acciones que el agente puede realizar
 - Las únicas acciones disponibles son: Izquierda, Derecha, Aspira
- La secuencia de percepciones del agente hasta el momento
 - El agente tiene la capacidad de percibir correctamente su ubicación y si esa ubicación está sucia.



Bajo estas condiciones la respuesta es "Sí"

Lo que subyace de esta clasificación de agente racional, es la idea de eximir al desarrollador de codificar todas las posibles soluciones y que sea el agente el que aprenda y que tenga la capacidad de percibir un cambio en el entorno y buscar la solución al problema que debe resolver

Omnisciencia

Omnisciencia: conocimiento de todas las cosas reales y posibles. Cuando un agente es omnisciente, conoce todo el universo de entradas que puede tener junto con la salida adecuada para cada una de estas entradas real de sus acciones. Es decir tiene todo el input y output posible y nada lo puede sorprender. Sus respuestas se tornan como reflejos por lo tanto no tendría razonabilidad.

En cambio la racionalidad busca maximizar la performance esperada. Con la tecnología actual no es posible tener un agente omnisciente, dado que la cantidad de información que se necesita procesar y comprender es muy grande y compleja.

Aprendizaje

1. El Agente racional requiere que recolecte información, además que aprenda sobre lo que percibe.

2. La configuración inicial del agente podría reflejar algún conocimiento previo del ambiente, pero mientras el agente gana experiencia, esto puede ser modificado o aumentado.
3. Ambientes totalmente conocidos a priori
 - a. Existen casos extremos en los que el ambiente es totalmente conocido a priori. En este caso el agente no necesita percibir o aprender.
 - b. Estos agentes son frágiles, porque ante un mínimo cambio en el ambiente, el agente puede comenzar a comportarse no racionalmente.

Ahora como los ambientes no son totalmente conocidos necesitamos que el agente posea el siguiente concepto.

Autonomía

1. Un agente racional debe ser autónomo
2. Debe aprender lo que pueda para compensar un conocimiento parcial o incorrecto del ambiente.
3. No debe depender totalmente del conocimiento del diseñador, sino tratar de capturar ese conocimiento de sus propias percepciones.
4. Por ejemplo, un agente aspiradora que aprende para pronosticar donde y cuando va a aparecer nueva suciedad va a tener mejor performance que uno que no lo hace.

PEAS

Para un agente racional entonces se debe especificar:

1. P (Performance measure): Una medida que se utiliza para evaluar la actuación del agente. Esta medida puede ser una recompensa, una penalización o una combinación de ambos.
2. E (Environment): El entorno en el que el agente opera y realiza sus acciones. El entorno puede ser cualquier cosa, desde un juego de mesa hasta un entorno físico complejo.
3. A (Actuators): Los actuadores son los dispositivos que el agente utiliza para interactuar con el entorno.
Por ejemplo, los actuadores de un robot pueden ser motores o garras, mientras que los actuadores de un programa de ajedrez pueden ser los movimientos de las piezas.
4. S (Sensors): Los sensores son los dispositivos que el agente utiliza para percibir el entorno. Los sensores pueden ser cámaras, micrófonos, sensores de temperatura, etc.

Ejemplo

- PEAS del agente del taxi autónomo

Tipo de Agente	Medida de Performance	Ambiente	Actuadores	Sensores
Taxi autónomo	Seguridad, rapidez, legalidad, confortabilidad, maximización de ganancias	Caminos, otros vehículos, peatones, clientes	Acelerador, freno, bocina, volante	Cámaras, sonar, medidor de velocidad, GPS, acelerómetro, sensores de motor, teclado

Entornos

Hay 3 categorías de los entornos de observación:

1. Totalmente Observable

- a. Los sensores de un agente tienen acceso al estado completo de un ambiente en cada momento del tiempo.
- b. Los sensores deben detectar TODOS los aspectos relevantes para la acción.
- c. Ventaja: no requieren mantener un estado interno para tener trazabilidad del mundo. Es decir en todo momento conoce todo

2. Parcialmente Observable

- a. El agente no tiene acceso completo y directo a toda la información relevante del entorno..
- b. Debe utilizar la información disponible para construir una representación aproximada del estado actual del entorno
- c. Motivos: Ruido, sensores inexactos, o porque parte del ambiente no está al alcance de los sensores
- d. Utiliza técnicas de aprendizaje automático y procesamiento del lenguaje natural para inferir el estado del entorno y tomar decisiones óptimas

3. No Observable

- a. El agente no tiene acceso a la información del entorno
- b. No tiene acceso a los resultados de las acciones
- c. Si el agente no tiene sensores
 - i. Un ejemplo de aplicación práctica de un agente inteligente en un entorno no observable es en el campo de la planificación de rutas de transporte.
 - ii. Planifica la ruta óptima para un vehículo de entrega en función de la información disponible, como los mapas, la ubicación y horarios de entrega
 - iii. No tiene acceso a información sobre el tráfico actual, accidentes o cualquier otro evento que pueda afectar la ruta.
 - iv. Utiliza técnicas de aprendizaje automático y de razonamiento para prever la situación del tráfico y prever posibles problemas en la ruta.
 - v. Puede utilizar técnicas de comunicación para obtener información adicional del conductor del vehículo o de otros sistemas de transporte en

la zona

Learning Rate



Imagina que estás aprendiendo a lanzar una pelota a un objetivo. Si nunca lo has hecho antes, al principio te costará apuntar bien, y necesitarás hacer ajustes después de cada lanzamiento. El learning rate sería como los pasos que das para ajustar tu puntería. Si los pasos son muy grandes, podrías pasarte del objetivo o desviarte mucho. Si los pasos son muy pequeños, te acercarías muy lentamente al objetivo y tardarías más en aprender.

En términos de inteligencia artificial, el learning rate controla la velocidad con la que un modelo ajusta sus predicciones a medida que "aprende" de los errores. Si el learning rate es muy alto, el modelo hace cambios grandes y podría terminar empeorando en lugar de mejorar. Si es muy bajo, el modelo aprende demasiado lento y tarda más en mejorar su precisión.

Es un equilibrio importante para que el modelo aprenda de manera eficiente, ni muy rápido ni muy lento.

Propiedades

Agente Único

Realiza tareas de forma autónoma en un ambiente. Ejemplo: Un agente resolviendo un crucigrama

Multiagente

Es un sistema donde varios agentes inteligentes interactúan entre sí para lograr un objetivo común. Las características son:

1. Cada agente es autónomo, toma sus propias decisiones
2. Pueden tener diferentes capacidades y conocimientos, y pueden estar especializados en diferentes tareas
3. Existe cooperación entre los agentes
 - a. Ejemplo: se pueden utilizar varios robots como agentes para realizar la exploración de un entorno desconocido o la construcción de una estructura compleja

Agente Competitivo

Es un agente que compite con otros agentes por recursos o por alcanzar un objetivo. Sus características son:

1. Toman decisiones en función de su propio interés
2. Si maximizan la performance de un agente, minimiza la de otro. Entonces el ambiente es competitivo
 - a. Ejemplo: En las simulaciones económicas pueden competir por recursos limitados

Agente Colaborativo

Es aquel agente que trabaja en conjunto con otros agentes para alcanzar un objetivo común

1. Toman decisiones en función del bien común del grupo y se comunican y coordinan entre sí
2. Si la maximización de la performance de un agente maximiza la de otro, entonces el ambiente es colaborativo
 - a. Ejemplo: En sistemas de logística, los agentes se coordinan para optimizar la distribución de productos y minimizar los costos de transporte.

Agente Determinístico

El próximo estado del ambiente está completamente determinado por el estado actual y la acción ejecutada por el agente.

1. Siempre tomará la misma acción dado un conjunto particular de percepciones del entorno
2. Su comportamiento está completamente determinado por su programa interno y por la percepción actual del entorno
 - a. Ejemplo: Termostato del aire acondicionado, enciende o apaga la ventilación según la temperatura ambiente

Agente Estocástico

El próximo estado del ambiente no está completamente determinado por el estado actual y la acción ejecutada por el agente.

1. Toma decisiones basadas en una combinación de su percepción del entorno y en una medida de incertidumbre o aleatoriedad.
 - a. Ejemplo: un auto autónomo es estocástico porque ninguno puede predecir exactamente el comportamiento del tráfico



"Estocástico" se refiere a procesos que contienen elementos de azar, pero que están influenciados por ciertas probabilidades o patrones generales. En otras palabras, aunque haya variabilidad y azar, el comportamiento de un proceso estocástico puede ser descrito de manera probabilística o seguir ciertas reglas. La diferencia con algo aleatorio es completamente impredecible, mientras que algo estocástico tiene elementos de azar pero puede seguir un patrón probabilístico que le da cierta estructura o previsibilidad.

Agente Episódico

La experiencia del agente es dividida en episodios atómicos.

1. No mantiene una memoria a largo plazo de sus acciones
2. En cada episodio un agente recibe una percepción y realiza una sola acción. El siguiente episodio no depende de las acciones de los episodios previos.
3. Son útiles en entornos dinámicos que cambian con el tiempo
 - a. Ejemplo: una aspiradora inteligente que debe limpiar una habitación

Agente Secuencial

La experiencia del agente está en secuencia. Sus características son

1. El agente sigue una serie de etapas o pasos que se realizan en un determinado orden para lograr un objetivo determinado
2. Las decisiones actuales podrían afectar las decisiones futuras
3. Los agentes secuenciales toman decisiones basadas en una percepción del estado actual del entorno, así como en una predicción de cómo el entorno cambiará en el futuro
4. Ejemplo: Robot que se encarga del ensamblado de una pieza de producción

Ambientes

1. Estáticos: Los ambientes estáticos son más simples porque el agente no debe permanecer mirando el mundo mientras decide una acción.
2. Dinámicos: Si el ambiente puede cambiar mientras el agente está deliberando, entonces el ambiente es dinámico. De otra forma, es estático.

Agente Discreto

Este agente interactúa con un entorno que tiene estados discretos.

1. Tiene un número limitado de estados posibles en los que el entorno puede encontrarse.
2. También toma acciones discretas, acciones que tienen una limitación finita
 - a. Ejemplo: El ajedrez tiene un conjunto discreto de percepciones y acciones ya que su ambiente tiene un conjunto finito de estados distintos

Agente Continuo

Este agente interactúa con un entorno que tiene estados y acciones continuas

1. Un entorno continuo tiene un número infinito de estados posibles
2. Toma acciones continuas, lo que significa que las acciones no están limitadas a un conjunto finito de acciones
 - a. Ejemplo: un auto inteligente conduciendo de un punto A hacia otro punto B puede tener infinita cantidad de situaciones (tráfico, semáforos, cortes, etc.)

Ambientes conocidos vs desconocidos

Esta distinción se refiere no al ambiente en sí mismo sino al estado de conocimiento del agente acerca de las "leyes físicas" del ambiente.

1. En un ambiente conocido, las salidas (o las probabilidades de las salidas en un ambiente estocástico) vienen dadas para todas las acciones.
2. Si el ambiente es desconocido, el agente va a tener que aprender como trabaja para poder hacer buenas decisiones.
 - a. Ejemplo: En el juego del solitario, se conocen las reglas pero no se sabe cuales son las cartas que faltan. Por lo tanto, el ambiente es parcialmente observable pero conocido (lo conocido aplica a las leyes del ambiente).

Estructura de Agentes

- a. Comportamiento: acción realizada luego de cualquier secuencia de percepciones

- b. Misión de la AI: diseñar programas agentes que implementen la función agente
- c. Arquitectura: se refiere al dispositivo computacional en el cual corre el programa agente



Recordemos que una función agente mapea una secuencia de percepciones con una acción. Un programa agente toma una percepción como entrada y devuelve una acción a través de los actuadores.

La función de un agente es actuar de manera autónoma en un ambiente para alcanzar un objetivo específico. En general, un agente inteligente consta de cuatro componentes principales:

1. Observación: El agente recibe información del ambiente a través de sus sensores, los cuales le permiten percibir el estado actual del ambiente.
2. Acción: El agente selecciona una acción a partir de su conjunto de acciones posibles. La selección de la acción puede estar basada en una estrategia predefinida o en una estrategia de aprendizaje automático.
3. Modelo: El agente tiene una representación interna del ambiente, que le permite anticipar las consecuencias de las acciones que toma. Este modelo puede ser explícito (es decir, que el agente conoce las reglas del ambiente) o implícito (es decir, que el agente aprende a partir de la experiencia).
4. Objetivo: El agente tiene un objetivo o meta que desea alcanzar. Este objetivo puede estar predefinido o puede ser aprendido a partir de la experiencia

Tipo de Agentes

Basados en Tablas

1. Utilizan una tabla para almacenar información sobre el estado del entorno y las acciones que deben tomar en función de ese estado.
2. Funcionan mediante la construcción de una tabla de "política" que contiene información sobre qué acción tomar en cada estado posible.
3. Implementa una función agente deseada.
4. Pero el desafío de la IA es encontrar como escribir programas que produzcan comportamiento racional desde el programa lo más chico posible, en lugar desde una tabla enorme

Desventajas:

1. no existe agente físico en el universo capaz de almacenar la tabla
2. el diseñador no tendría tiempo físico de crear la tabla
3. ningún agente podría aprender todas las entradas desde su experiencia
4. aún si el ambiente fuese simple, el diseñador no sabría como completar la tabla

Agentes Reflejos Simple

- a. Es el más simple

- b. Selecciona acciones en base a la percepción actual, ignorando la historia de percepciones.
- c. Tienen inteligencia limitada.
- d. Reglas Condición-acción: if car-in-front-is-braking then initiate-braking
- e. El agente funciona solo si se puede hacer la decisión correcta en base a la percepción actual, por lo que el ambiente es completamente observable.
- f. La mayor ventaja de este tipo de agente es el costo y la velocidad de respuesta. Ademas pueden colaborar con agentes que si tiene algún tipo de racionalidad

Basados en Modelos

- a. Manejan la observación parcial de una forma simple. Censa el estado actual del modelo, se pregunta como esta mi entorno ya y toma decisiones en función a ese estado. Luego guarda el estado actual es decir el agente mantiene un estado interno que depende de la historia de las percepciones. Se diferencia con el de reflejos simples en que empieza a tomar decisiones en función a la información que posee y no solo por reflejos prefijados.

Basados en objetivos

Este tipo va más allá de la descripción del estado actual, requiere algún tipo de información sobre el objetivo que describe situaciones deseables.

1. El agente combina el modelo (mismo modelo del agente basado en modelos) y el objetivo para tomar una decisión.
2. Búsqueda y Planning son problemas que suelen utilizar agentes basados en objetivos.

Diferencias entre agentes basado en reflejos y agentes basados en objetivos

1. Un agente reflejo mapea directamente percepciones a acciones.
2. Un agente reflejo frena cuando ve las luces de freno del vehículo adelante.
3. Un agente basado en objetivos va a razonar si frenar está en función del objetivo que se propone. Quizás la solución sea bajar la velocidad y no frenar.
4. Un agente basado en objetivos parece ser menos eficiente (su mecanismo de toma de decisiones es más complejo), pero es más flexible porque el conocimiento que soporta sus decisiones es representado explícitamente y puede modificarse.
5. En un agente reflejo se debiera reescribir las reglas condición-acción.
6. El comportamiento del agente basado en objetivos puede cambiarse si se desea ir a un destino distinto, en cambio en el reflejo hay que cambiar las reglas condición-acción.
7. Lograr un objetivo, no siempre genera un comportamiento de alta calidad.
8. Los objetivos pueden ser binarios (feliz o infeliz)
9. La utilidad permite medir que tan feliz se encuentra el agente.
10. Función de utilidad: Es una internalización de la medida de performance

Basados en Utilidad

Un agente racional basado en utilidad elige la acción que maximiza la utilidad esperada dentro de las posibles acciones

1. Es un mecanismo utilizado para la toma de decisiones bajo incertidumbre: ambientes de parcialmente observables y estocásticos.

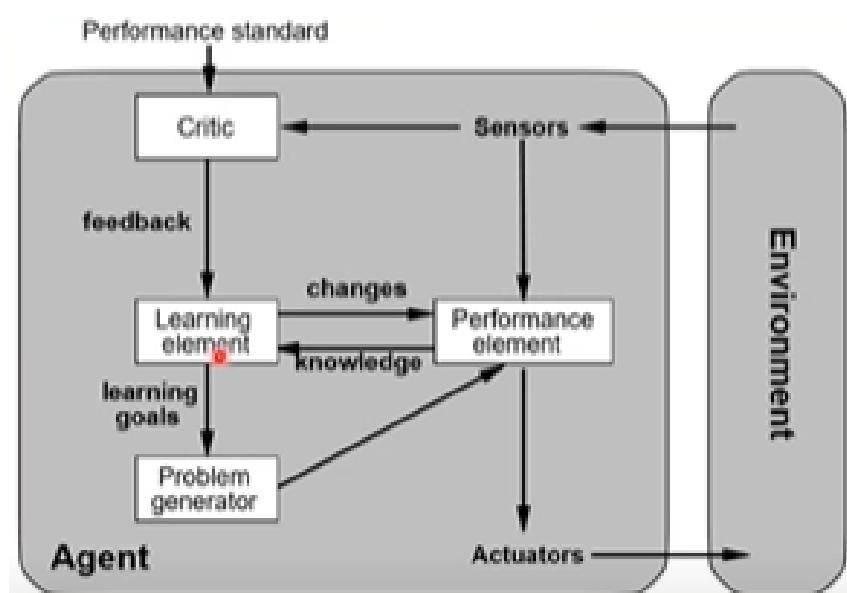
Aprendizaje

- a. Turing en el paper de 1950 considera la idea de programar las máquinas inteligentes a mano. Estima cuanto esfuerzo llevaría y concluye: "algún método más eficiente parecería deseable."
- b. El método que propone es construir máquinas que aprendan y enseñarles.
- c. El aprendizaje permite que los agentes operen en ambientes inicialmente desconocidos y se vuelvan más competentes a medida que aprenden

Los elementos principales del aprendizaje son:

- a. Elemento de aprendizaje: es responsable de hacer mejoras.
- b. Elemento de performance: es responsable de seleccionar acciones externas. Es lo que vimos en los 4 casos anteriores como "agente"
- c. Crítico: genera feedback para el Elemento de aprendizaje
- d. Generador de problemas: sugiere acciones que llevan a nuevas experiencias de aprendizaje.

El funcionamiento de los agentes basados en aprendizaje responden al siguiente esquema



1. El crítico le dice al elemento de aprendizaje que tan bien está haciendo con respecto a un estándar fijo de performance. El estándar de performance debe ser fijo para que el agente no pueda modificarlo por sí mismo. Por lo tanto podemos pensar que el crítico está fuera del agente.
2. El crítico juega 3 roles: evaluador, diagnosticador y terapista.
 - Como evaluador: embebe un estándar por el cual evaluar el comportamiento del elemento de performance.
 - Como diagnosticador: puede localizar las razones de la mala performance.
 - Como terapista: puede hacer recomendaciones de mejora.

3. El elemento critico es responsable de sugerir acciones que lleven a nuevas experiencias de aprendizaje.
4. Si el elemento de performance anda bien, podría hacer las mejores acciones en base a lo que sabe.
5. El generador de problemas permite que el agente explore un poco y haga acciones que quizás estén por debajo de lo óptimo en el corto plazo, pero pueda descubrir acciones mucho mejores en el largo.

Machine Learning

¿Qué es Machine Learning?

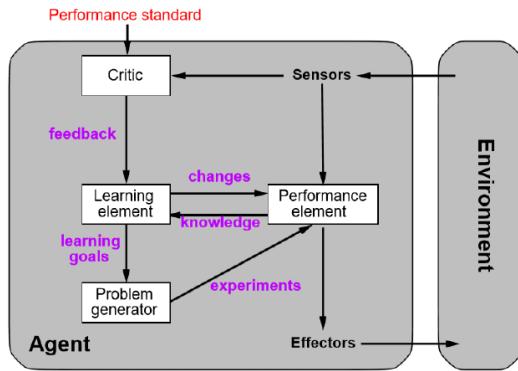
- Se refiere a un enfoque en el que las máquinas pueden aprender a partir de datos sin ser programadas explícitamente.
- En lugar de codificar reglas y algoritmos específicos, se les proporcionan datos y se les permite encontrar patrones y relaciones por sí mismas.
- Permite realizar tareas específicas y tomar decisiones basadas en la información aprendida.
- El machine learning abarca una variedad de técnicas como:
 - Regresión
 - Árboles de decisión
 - Máquinas de soporte vectorial (SVM)
 - Algoritmos de agrupación (clustering)

¿Qué es Deep learning?

- Es una subcategoría del machine learning que se centra en el uso de redes neuronales artificiales profundas para modelar y resolver problemas complejos.
- El término "profundo" se refiere al hecho de que estas redes tienen muchas capas intermedias entre la entrada y la salida, lo que les permite aprender representaciones jerárquicas y abstractas de los datos.
- Es eficaz en procesamiento de imágenes, el procesamiento de lenguaje natural y el reconocimiento de voz.

¿Por qué es necesario que un agente aprenda en lugar de programar esa mejora directamente en el agente desde el comienzo ?

1. Porque quienes diseñan el agente no pueden anticipar todas las posibles situaciones: un robot diseñado para navegar laberintos debe aprender el layout de cada laberinto.
2. Los diseñadores no pueden anticipar todos los cambios en el tiempo: un agente que aprende a predecir acciones debe aprender a adaptarse cuando las condiciones cambian.
3. Muchas veces los programadores humanos no saben programar una solución por sí mismos.



Los sensores captan los cambios del exterior luego se clasifican los elementos críticos

Aprendizaje supervisado

- El agente observa pares input-output y aprende una función que mapea del input al output.
- Implica entrenar un modelo para predecir resultados etiquetados basados en datos de entrenamiento.
- Se utiliza en una amplia gama de aplicaciones, incluyendo el procesamiento del lenguaje natural, la visión por computadora, la detección de fraude y el diagnóstico médico,
- Utiliza 2 técnicas: Clasificación y Regresión
 - La regresión y la clasificación son técnicas fundamentales en el aprendizaje supervisado, que se utilizan para abordar diferentes tipos de problemas.

Regresión

- La regresión es una técnica que se utiliza para predecir valores continuos a partir de un conjunto de variables de entrada.
- El objetivo es encontrar una función que pueda mapear los valores de entrada a un valor de salida continuo.
- Ejemplo: Predecir el precio de una casa en función de su tamaño, el número de habitaciones y otros factores,

Clasificación

- La clasificación se utiliza para predecir valores discretos o categóricos.
- El objetivo es encontrar una función que pueda mapear los valores de entrada a una categoría o etiqueta,
- Ejemplo: Predecir si un correo electrónico es spam o no en función de su contenido

En resumen podes decir que:

- Clasificación: el objetivo es predecir una clase, que es una elección de una lista predefinida de posibilidades.
 - Clasificación binaria: se clasifica en dos clases. Ejemplo clasificar mails en spam / no spam
 - Clasificación multiclase: clasifica en más de dos clases. El ejemplo iris visto anteriormente

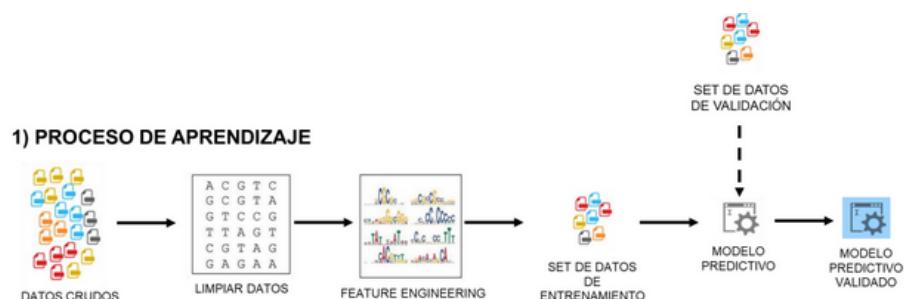
- Regresión: el objetivo es predecir un número continuo (un número real en términos matemáticos).
 - Ejemplo: predecir el ingreso de una persona dado su nivel de educación, edad y donde vive.

La forma de distinguir entre clasificación y regresión es ver si hay continuidad en el objetivo a predecir.

Principio de Funcionamiento

1. Dado un conjunto de entrenamiento de N pares entrada salida:
o $x_1, y_1, x_2, y_2 \dots (x_n, y_n)$
2. donde cada y_i fue generado por una función desconocida $y_i = f(x_i)$, siendo el objetivo descubrir una función $h(x)$ que aproxime el valor verdadero de la función $f(x)$.
3. La función h es la hipótesis.
4. El aprendizaje es la búsqueda a través del espacio de posibles hipótesis por una que tenga buena performance.
5. Para medir la exactitud de la hipótesis se da una conjunto de muestras de un conjunto de test, distintos del set de datos de entrenamiento.

Gráficamente sería



En texto sería:

- a. Tomo los datos
- b. Los limpio
- c. Defino las características relevantes
- d. Entreno el modelo con un set de datos de entrenamiento (gran volumen)
- e. Después valido con un set de datos mas pequeño de los cuales conozco su output para validar si necesito seguir entrenando o no el modelo

2) PROCESO DE APLICACIÓN DEL MODELO GENERADO



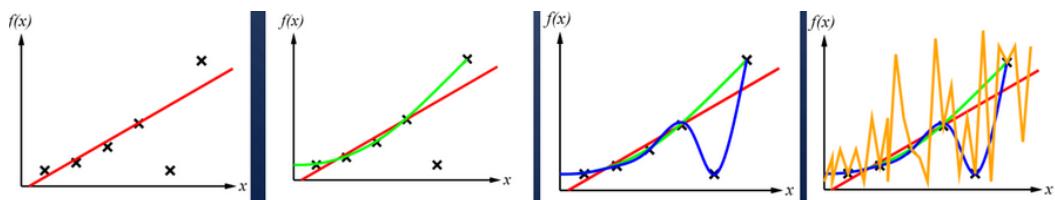
Luego puedo tomar un set de datos totalmente distinto y predecir en productivo

Aprendizaje supervisado inductivo

El aprendizaje supervisado inductivo es un enfoque común en el aprendizaje automático que implica el entrenamiento de un modelo a partir de un conjunto de datos etiquetados, con el objetivo de predecir las etiquetas de nuevos datos.

Se proporciona al modelo un conjunto de datos de entrenamiento que consta de pares de características y etiquetas asociadas. El término "inductivo" se refiere a la idea de que el modelo aprende a partir de ejemplos específicos para construir una regla general que se pueda aplicar a nuevos ejemplos.

Es importante generar distintas iteraciones para hacerse al modelo de datos propuestos. Fíjate como la azul y la naranja se ajustan a los datos pero con características distintas



Aprendizaje no supervisado

1. El agente aprende patrones en los datos que se proporcionan como entrada, aún cuando no exista feedback explícito
2. Se proporciona un conjunto de datos de entrada sin etiquetar al algoritmo y se le permite encontrar patrones o estructuras en los datos por sí mismo
3. No tiene información previa sobre las etiquetas o categorías que deben identificarse en los datos
4. Se utilizan en aplicaciones de segmentación de datos, reducción de dimensionalidad, detección de anomalías y clasificación no supervisada.
5. Ejemplos:
 - a. Clustering
 - b. Detección de outliers
 - c. Reducción de dimensionalidad
 - d. Análisis de componentes principales (PCA)
 - e. Redes neuronales autoorganizativas (SOM).

Aprendizaje por refuerzos

1. El agente aprende de una serie de refuerzos (recompensas o castigos).
2. Se basa en la idea de que una máquina debe aprender a partir de la retroalimentación que recibe del entorno en el que se encuentra.
 - <https://www.youtube.com/watch?v=2tamH76Tjvw>

3. Un posible algoritmo para este tipo de aprendizaje es el de Q-Learning
 - a. Este algoritmo aprende una serie de normas que le diga a un agente qué acción + tomar bajo qué circunstancia
 - b. No requiere un modelo del entorno
 - c. Puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones.
 - d. Para cualquier proceso de decisión de Markov finito (PDMF) encuentra una política óptima en el sentido de que maximiza el valor esperado de la recompensa total sobre todos los pasos sucesivos

Ahora que es un proceso de Markov:

1. Es un modelo matemático utilizado en la teoría de la decisión para describir situaciones en las que la toma de decisiones se realiza en un ambiente estocástico.
2. Una toma de decisión se representa como una secuencia de pasos que no se toman todas a la vez, sino que el proceso se descompone en varios pasos o etapas. En cada paso, el estado del sistema cambia en función de la acción que se haya elegido y del estado anterior. La idea es que el futuro estado depende únicamente del estado presente y no de los estados anteriores (propiedad de "**memoria limitada**" o "sin memoria" de los procesos de Markov). Por ejemplo:
 - a. Si hoy está **soleado**, existe una probabilidad del **80%** de que mañana también esté **soleado** y una probabilidad del **20%** de que mañana esté **lluvioso**.
 - b. Si hoy está **lluvioso**, hay una probabilidad del **60%** de que mañana siga **lluvioso** y una probabilidad del **40%** de que mañana esté **soleado**.

Este comportamiento es un proceso de Markov porque la predicción del clima de mañana depende **únicamente** del clima de hoy, y no de cómo fue el clima antes de hoy.
3. Cada paso implica la elección de una acción a partir de un conjunto de posibles acciones.
4. El agente recibe una recompensa o penalización en función de la acción elegida y del estado actual del ambiente.
5. El objetivo del agente es maximizar la suma total de las recompensas obtenidas en todos los pasos.
6. El proceso de decisión se define por una tupla (S, A, T, R, γ)
 - a. S es el conjunto de estados posibles del ambiente
 - b. A es el conjunto de acciones posibles del agente
 - c. T es la función de transición de estados, que indica la probabilidad de pasar de un estado a otro dado una acción determinada
 - d. R es la función de recompensa, que indica la recompensa recibida por el agente al pasar de un estado a otro
 - e. γ es el factor de descuento, que indica cuánto se valora una recompensa en el futuro en comparación con una recompensa en el presente

Si lo vemos con un ejemplo de que un robot pueda cruzar un laberinto donde cada paso es una celda del laberinto

- S son las celdas del laberinto.

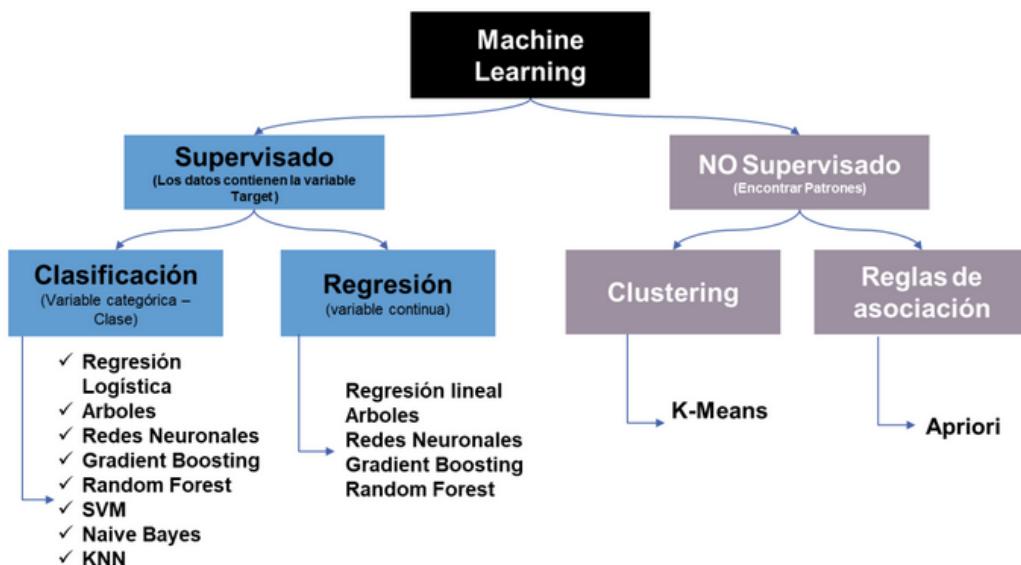
- **A** son las posibles direcciones en las que el robot puede moverse.
- **T** describe la probabilidad de que el robot llegue a su destino previsto, dado que puede haber errores.
- **R** es la función que otorga recompensas o penalizaciones dependiendo de las decisiones que toma el robot.
- **γ** determina cuánto valora el robot las recompensas futuras (llegar a la salida) en comparación con las inmediatas.

El algoritmo se repite hasta que se alcanza el número máximo de iteraciones o hasta que se cumple alguna otra condición de terminación.

Ahora también existe la estrategia epsilon-greedy, que es un enfoque común utilizado para equilibrar la exploración y la explotación y funciona de la siguiente manera:

1. El agente selecciona una acción al azar con probabilidad épsilon (exploración).
2. El agente selecciona la mejor acción conocida hasta el momento con probabilidad 1-épsilon (explotación).
3. El agente explorará nuevas acciones con una frecuencia determinada por el valor de épsilon.
4. Si épsilon es muy alto, el agente explorará con mayor frecuencia, lo que puede ayudar a encontrar nuevas soluciones.
5. Si épsilon es muy bajo, el agente explotará la mejor opción conocida con mayor frecuencia, lo que puede ayudar a maximizar las recompensas a corto plazo.
6. El valor de epsilon se ajusta típicamente a lo largo del tiempo, disminuyendo gradualmente a medida que el agente adquiere más conocimiento.

En resumen hasta ahora tenemos:



Estados de los modelos

Existen 3 tipos de estados por los que puede pasar nuestro modelo:

1. Underfitting

- a. Situación en la que un modelo no puede capturar la complejidad suficiente en los datos de entrenamiento,
- b. El modelo es demasiado simple o tiene muy pocas características, y como resultado, no puede ajustarse adecuadamente a los datos de entrenamiento y proporcionar predicciones precisas.
- c. No puede generalizar bien para hacer predicciones precisas sobre datos nuevos y no vistos
- d. Algunas de las causas comunes de underfitting incluyen:
 - Un modelo demasiado simple o con muy pocas características
 - Un conjunto de entrenamiento pequeño o poco representativo
 - Una selección inadecuada de hiperparámetros del modelo
 - Datos ruidosos o errores en los datos en los sets de entrenamiento

2. Generalizable

- a. Es un modelo que tiene la capacidad de hacer predicciones precisas sobre nuevos datos que no ha visto durante el entrenamiento,
- b. Puede generalizar los patrones y relaciones aprendidos en los datos de entrenamiento a nuevos datos, en lugar de simplemente memorizar los datos de entrenamiento y dar resultados pobres en datos no vistos.
- c. Logra al encontrar un equilibrio adecuado entre la complejidad y la simplicidad del modelo, y al evitar tanto el overfitting como el underfitting.

3. Overfitting

- a. Situación en la que un modelo se ajusta demasiado bien a los datos de entrenamiento, lo que resulta en una pobre capacidad de generalización para nuevos datos.
- b. Ocurre cuando un modelo es demasiado complejo y tiene demasiadas características o parámetros en comparación con el conjunto de datos de entrenamiento, lo que hace que el modelo "memorice" los datos de entrenamiento en lugar de aprender patrones más generales.
- c. Algunas de las causas comunes de overfitting incluyen:
 - Un modelo demasiado complejo o con demasiadas características
 - Un conjunto de entrenamiento pequeño en relación con la complejidad del modelo
 - Entrenamiento excesivo del modelo durante un tiempo demasiado largo

En resumen:

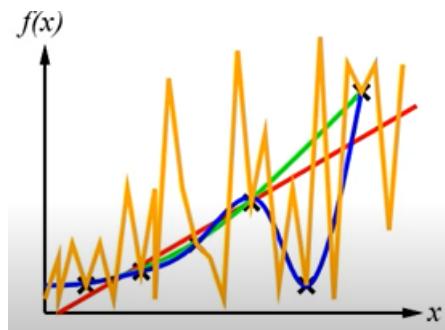
Un modelo más complejo hará que sea más exacto sobre el set de datos de entrenamiento.

Sin embargo, si el modelo se vuelve demasiado complejo, se comienza a focalizar en cada punto de datos individual del set de datos de entrenamiento y el modelo no generaliza bien a nuevos

datos.

Existe un punto intermedio entre complejidad y precisión que permite una buena generalización sin caer en overfitting.

Para seleccionar un modelo de entre un conjunto múltiple de modelos consistentes se utiliza la Navaja de Ockham que dice que "Cuando hay varias explicaciones posibles para un fenómeno, la explicación más simple suele ser la correcta.". Es decir si dos modelos son igualmente buenos para explicar los datos, se debe elegir el modelo más simple. Es importante tener en cuenta que, en algunos casos, un modelo más complejo puede ser necesario para explicar los datos observados. En tales situaciones, se debe equilibrar el principio de parsimonia con la precisión y la capacidad de explicar los datos de manera adecuada. Por ejemplo en el siguiente grafico el modelo naranja y el modelo azul explican los datos pero el modelo azul es más simple, pero dependiendo del caso también podría elegir la verde que si bien no explica TODOS los datos incluye a la mayoría. A nivel costo de procesamiento también vemos el mismo ranking siendo la naranja la de mas costo y la roja la de menos costo



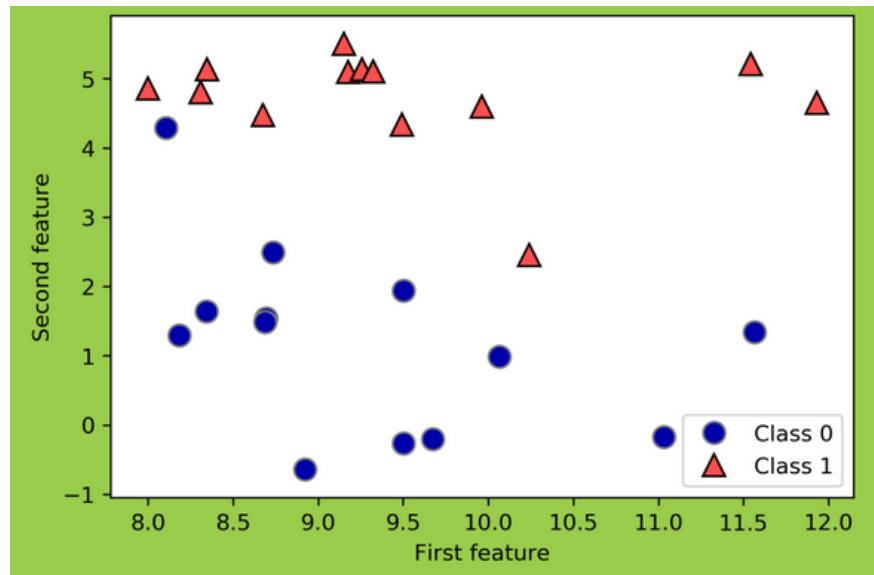
Para seleccionar el modelo más adecuado se utilizan las siguientes técnicas:

1. Validación cruzada: se divide el conjunto de datos en varios subconjuntos de entrenamiento y prueba, y se evalúa el rendimiento del modelo en cada subconjunto. Se elige el modelo con mejor rendimiento en la validación cruzada.
2. Selección de características: se identifican las características más relevantes para el problema y se eliminan las características redundantes o irrelevantes. Se construye un modelo con las características seleccionadas.
3. Regularización: se añade una penalización por complejidad al modelo, lo que favorece modelos más simples. Se ajusta el parámetro de regularización para encontrar el mejor equilibrio entre la complejidad y el rendimiento.

Algoritmo KNN (K-Nearest Neighbors)

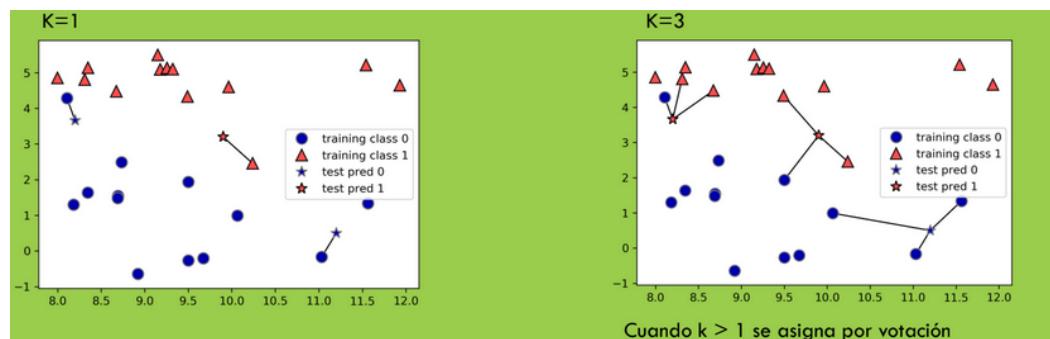
1. Es un algoritmo de aprendizaje automático supervisado utilizado para clasificar objetos en función de su similitud con otros objetos previamente etiquetados.
2. El funcionamiento del algoritmo es el siguiente:
 - a. Se selecciona un valor para el parámetro "K", que representa el número de vecinos más cercanos que se tendrán en cuenta para clasificar un nuevo objeto.
 - b. Se calcula la distancia entre el nuevo objeto y todos los objetos del conjunto de entrenamiento.

- c. Se seleccionan los "K" objetos más cercanos al nuevo objeto, utilizando la distancia como criterio de selección.
- d. Se asigna al nuevo objeto la clase más frecuente entre sus "K" vecinos más cercanos.

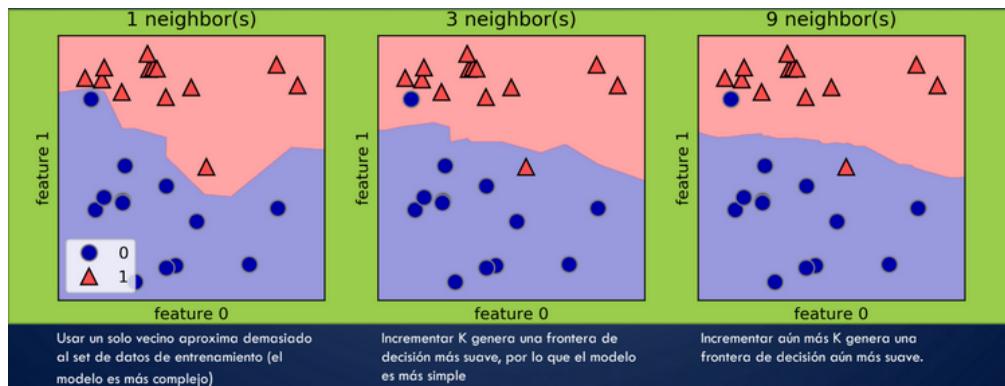


Ahora veamos un ejemplo grafico:

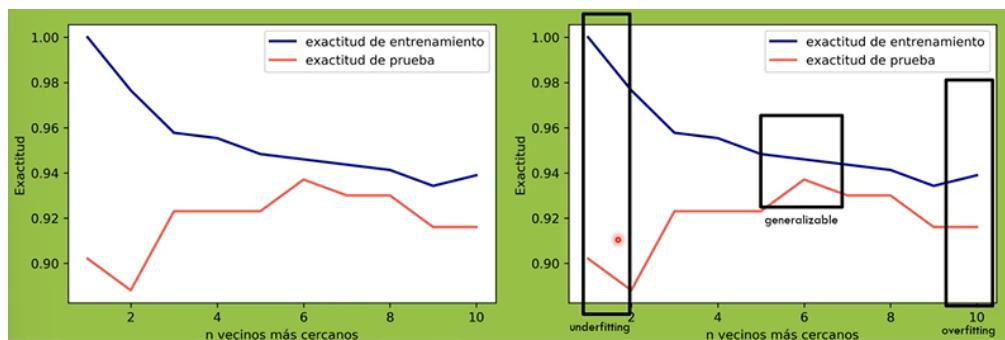
- a. Se agregan 3 nuevos puntos de datos marcados como estrellas que adopta el color al punto mas cercano. Con K=1 caigo en onderfeeting porque solo analizo un solo vecino mas cercano. Si pongo en 3 en donde adopto el color de la mayoría.



- b. Se marcan las fronteras de decisión con los diferentes valores de K en donde con un valor de K bajo la frontera es más irregular y marcada y con valores de K altos la linea comienza a ser mas suave



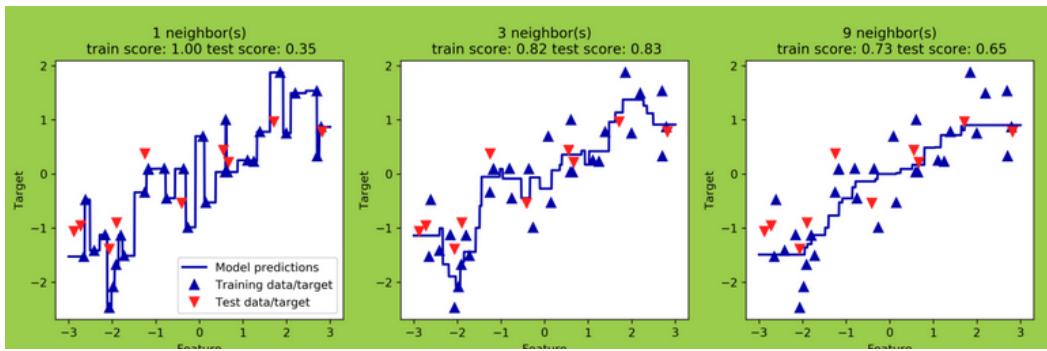
- c. Si tomamos un ejemplo con set de datos con tumores siendo la linea azul los datos de entreganimiento y la roja de prueba. Con K alto le va a costar mucho encontrar los tumores (caigo en overfitting) y si tengo un K bajo la precision es muy baja. Entonces un K medio es el que me acerca más la prueba del set de entrenamiento



De la misma manera que el KNN se usa para clasificación tambien lo podemos usar para regresión en donde:

1. Trata de predecir un valor numérico en lugar de una clase.
2. En lugar de asignar al nuevo objeto la clase más frecuente entre sus vecinos más cercanos, se calcula el promedio de los valores de la variable objetivo de sus vecinos más cercanos.
3. El principio de funcionamiento es:
 - a. Se selecciona un valor para el parámetro "K", que representa el número de vecinos más cercanos que se tendrán en cuenta para predecir el valor de la variable objetivo del nuevo objeto.
 - b. Se calcula la distancia entre el nuevo objeto y todos los objetos del conjunto de entrenamiento.
 - c. Se seleccionan los "K" objetos más cercanos al nuevo objeto, utilizando la distancia como criterio de selección.
 - d. Se calcula el promedio de los valores de la variable objetivo de los "K" vecinos más cercanos y se asigna este valor como la predicción para el nuevo objeto.

Y el resultado es mas o menos lo mismo con un valor de K medio se obtienen valores de prueba mucho más cercanos al de entrenamiento (la taza de prueba con 3 es máxima con 0,83)



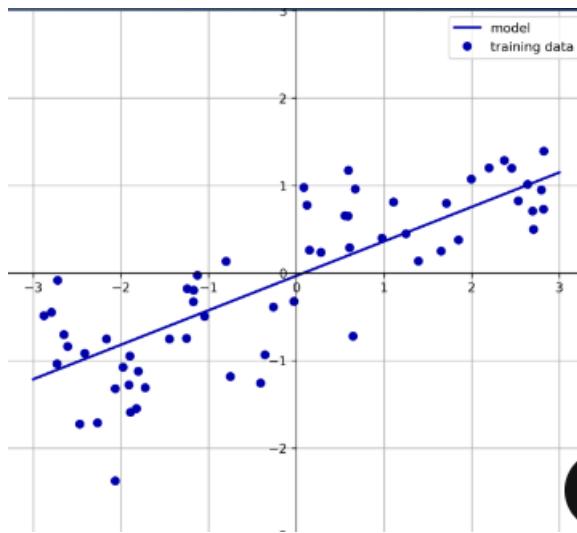
Conceptualmente si tomo K alto estoy queriendo ajustar al modelo a características que no tienen sentido para lo que quiero averiguar.

OLS Regresión

1. Es un método estadístico utilizado para ajustar un modelo lineal a un conjunto de datos, con el objetivo de predecir una variable numérica en función de una o varias variables predictoras.
2. Busca encontrar los valores de los parámetros del modelo lineal que minimizan la suma de los errores al cuadrado entre los valores observados y los valores predichos por el modelo.
 - a. Un modelo de regresión tiene la siguiente forma:

$$y = w_0 \cdot x_0 + w_1 \cdot x_1 + \dots + w_p \cdot x_p + b$$
 - b. En este caso existen $p + 1$ características
 - c. Para un dataset con una sola característica es:

$$y = w_0 \cdot x_0 + b$$
 - d. Que es una recta con pendiente w_0 y b es el desplazamiento de intersección con el eje y
3. Un modelo de regresión encuentra los parámetros w y b que minimizan el error cuadrado promedio entre las predicciones y los valores verdaderos de la regresión en el set de datos de entrenamiento.
4. El error cuadrado promedio es la suma de las diferencias cuadradas entre las predicciones y los valores verdaderos, dividido por la cantidad de muestras.
5. Una vez que se han estimado los parámetros del modelo, se puede utilizar el modelo para hacer predicciones sobre valores nuevos de las variables predictoras.

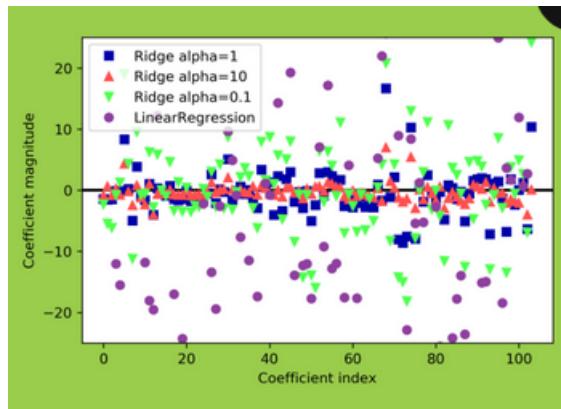


En resumen, OLS es una técnica para encontrar la mejor línea recta (o hiperplano si hay más variables) que se ajuste a los datos, minimizando las diferencias entre lo predicho por el modelo y lo que ocurre en la realidad.

Para hacer la regulación del modelo tenemos dos modelos:

Regresión Contraída (Ridge Regression o Regularización L2)

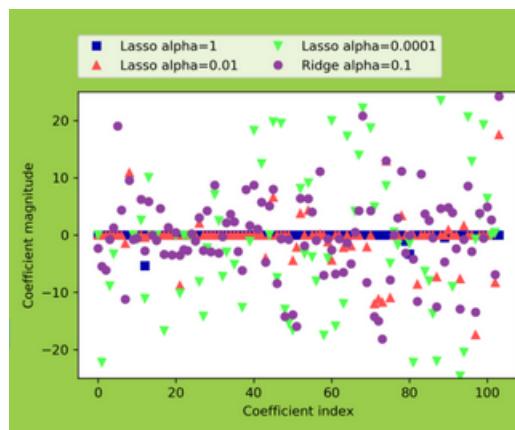
1. Es un modelo lineal para regresión de forma tal que la fórmula que se utiliza es la misma que para OLS, pero los coeficientes w se eligen de forma tal que no solo predigan bien sobre el set de datos de entrenamiento, sino que también se aproximen a 0.
2. En forma intuitiva, cada característica tiene la menor influencia posible sobre la salida (lo cual se traduce a tener una pequeña pendiente).
3. Esto se llama regularización. La regularización es una manera de forzar que se evite el overfitting
4. El training set score de Ridge suele ser más bajo que en la regresión anterior, mientras que el Test set score es más alto.
5. Es esperable porque en la regresión lineal se está haciendo overfitting de los datos.
6. Ridge es más restrictivo.
7. Ridge hace un trade off entre simplicidad del modelo (coeficientes cercanos a 0) y la performance en el set de datos de entrenamiento. Para esto se usa el parámetro Alpha.
8. El valor de Alpha depende del dataset que se está utilizando. En este caso se comenzó con un Alpha = 1.
9. Decrementar Alpha permite que haya menos restricción.
10. Valores muy chicos de Alpha, flexibilizan restricciones
11. El eje X enumera las entradas de $\text{coef_x}=0, x=1$ el coeficiente de la segunda característica así hasta $x=100$.
12. El eje Y son los valores numéricos de los coeficientes.
13. Se puede observar que a menor Alpha, más dispersión en la variación de los coeficientes.



1. Otra forma de entender la influencia de la regularización es dejar fijo un valor de Alpha pero variar la cantidad de datos de entrenamientos disponibles.
2. A la derecha, se tomó el dataset Boston Housing y se evaluó la Regresión Lineal y Ridge con Alpha=1 en subconjuntos de tamaños incrementales.
3. Como uno esperaría, el training score es más alto que el test score para todos los tamaños de data set.
4. Dado que Ridge está regularizado, el training score de Ridge es más bajo que el training score de la regresión lineal. Sin embargo el test score de ridge es mejor, particularmente para set de datos de entrenamiento más chicos. Para menos de 400 puntos la regresión lineal no aprende nada. Con más datos, el modelo mejora y la Regresión Lineal alcanza a Ridge al final.

Regresión Lasso (Lasso Regression o Regularización L1)

1. Al igual que Ridge, Lasso también restringe que los coeficientes se aproximen a 0, pero de una forma diferente llamada Regularización L1.
2. En Lasso, algunos coeficientes son exactamente cero.
3. Esto significa que esas características son ignoradas por el modelo, lo cual puede verse como una forma de selección de características automática.
4. Esto permite ver cuáles son las variables más relevantes del modelo.



Conclusion de la clase:

1. ML sirve para que el agente aprenda

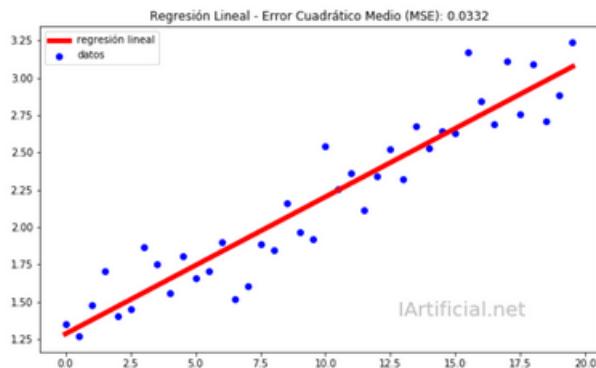
2. Para que el agente aprenda, lo entrenamos.
3. Para entrenarlo le damos datos de entrenamiento y probamos que lo que hizo es correcto o no.
4. Es decir buscamos un modelo generalizables (capacidad de adaptarse a los cambios del ambiente)
5. Ahora como no siempre los input son validos hago los siguientes pasos
 - a. Aplico, en función a todas las características la Y que me da
 - b. Y luego aplico restricciones que me van a servir para calcular el promedio de error
 - c. Con el promedio de error puedo descartar determinadas predicciones porque que tienen una tasa de error mayor a la permitida.
6. En función a estos pasos voy a determinar si el modelo se ajusta a la restricción de costos, predicción, y procesamiento. Entendiendo que muchas veces el modelo más simple resulta ser el correcto.

Modelos Lineales para Clasificación

1. Los algoritmos de modelos lineales son técnicas de aprendizaje automático que se utilizan para predecir una variable de salida en función de una o más variables de entrada.
2. Los modelos lineales también se usan para clasificación.
3. La regresión lineal es una de las técnicas más usadas en Machine Learning.
4. Su fortaleza estriba en su simplicidad e interpretabilidad
5. Se asume que existe una relación lineal entre las variables de entrada y la variable de salida.

Regresión Lineal

1. Es el algoritmo más simple y comúnmente utilizado para modelos lineales.
2. Es una técnica paramétrica de machine learning.
3. Con "paramétrica" queremos decir que incluso antes de mirar a los datos, ya sabemos cuántos parámetros (o coeficientes) vamos a necesitar
4. Establece una relación lineal entre las variables de entrada y la variable de salida mediante la minimización del error cuadrático medio.
5. Usando una sola variable, x, sabemos que una línea necesita 2 parámetros.
6. La fórmula para la regresión lineal con una sola variable x es: $y = wx + b$



Ejemplo:

1. El proceso de aprendizaje consiste en estimar los parámetros w y b . Para: $w=0.0918$ y $b=1.2859$
2. Aplico la fórmula $y = wx + b \Rightarrow y = 0.0918x + 1.2859$ para $x = 5$, el resultado es 1.7449 dado que: $y = 0.0918 \cdot 5 + 1.2859 = 1.7449$
3. En definitiva yo le voy a pasar como entrenamiento los valores de X y de Y para que el modelo estime los parametros w y b que me permitan generar el modelo lineal basado en la formula $y = wx + b$.
4. Para evaluar el modelo voy a usar el Error Cuadrático Medio que es el criterio de evaluación más usado para problemas de regresión.
5. Se usa sobre todo cuando usamos aprendizaje automático supervisado.
6. Para cada dato histórico podremos indicar el resultado correcto.

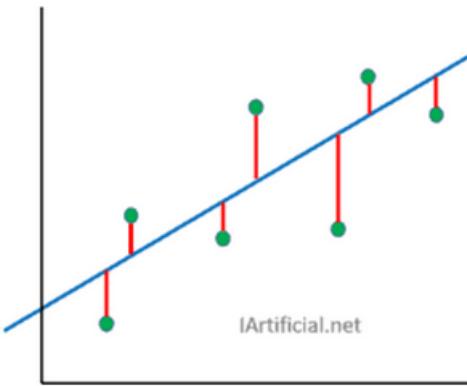
Error cuadrático medio

El ECM se calcula como la media de los cuadrados de las diferencias entre los valores observados y los valores predichos por el modelo.

Si el ECM es bajo, significa que el modelo es capaz de predecir los valores de la variable de salida con mayor precisión. Si el ECM es alto, significa que el modelo no se ajusta bien a los datos y las predicciones pueden ser menos precisas. La fórmula para calcular el ECM es la siguiente: $ECM = (1/n) * \sum((y - y_{pred})^2)$ donde:

- a. "n" es el número de observaciones
- b. "y" es el valor observado,
- c. "y_pred" es el valor predicho por el modelo.

El valor estimado es el valor que nos da el modelo. En este caso, la línea azul.



Un punto importante son las unidades que estamos trabajando para descartar o no los diferentes errores, si tengo un problema en el espacio los errores en milímetros son despreciables y seguramente tenga que evaluar errores en años luz

Ahora cuando estudiamos una situación podemos tener muchas características o atributos para analizar (x_1, x_2, x_3, x_4) y cada una tendrá un peso relativo en el modelo (w_0, w_1, w_2). Siendo la sumatoria de estos parámetros y pesos y el coeficiente de intercección el valor de y ($y = b + w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_Nx_N$)

Tomando esto como base podemos construir la matriz X con la siguiente convención:

1. M filas: cada fila es un dato (por ejemplo, un inmueble, si queremos predecir su valor de venta)
2. N columnas: cada columna es un atributo relevante (por ejemplo, cuántas habitaciones tiene, metros cuadrados, etc)

Si nos metemos en la implementación de la regresión logística tenemos que recorrer los siguientes puntos:

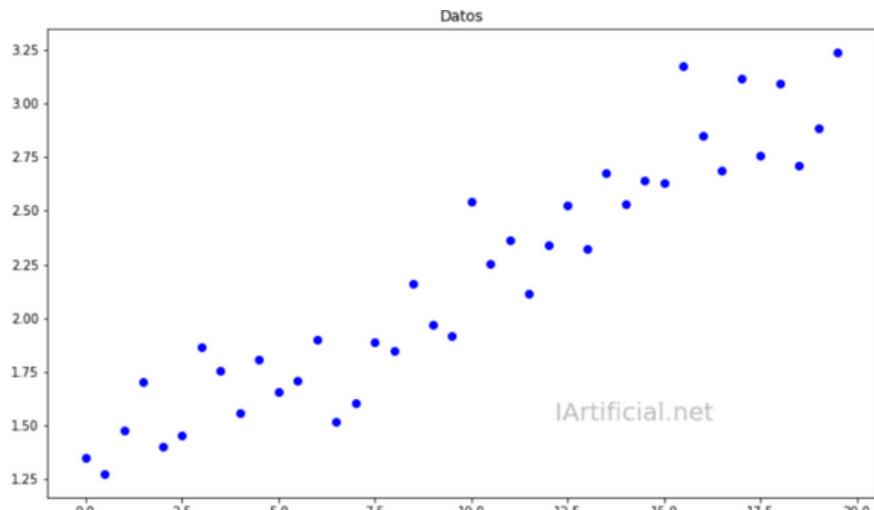
1. Vamos a usar una librería de Python: scikit-learn, para aprendizaje automático.
2. La clase LinearRegression implementa la funcionalidad
3. Generamos datos que siguen una línea, y le añadimos ruido gaussiano.
4. Usamos la librería de python NumPy.
5. La fórmula que utilizada para generar los datos es: $y = 0.1x + 1.25 + N(0, 0.2)$

```

1. import numpy as np #Librería numérica
2. import matplotlib.pyplot as plt # Para crear gráficos con matplotlib
3. %matplotlib inline # Si quieres hacer estos gráficos dentro de un
jupyter notebook
4.
5. from sklearn.linear_model import LinearRegression #Regresión Lineal con
scikit-learn
6.
7. def f(x): # función f(x) = 0.1*x + 1.25 + 0.2*Ruido_Gaussiano
8.     np.random.seed(42) # para poder reproducirlo
9.     y = 0.1*x + 1.25 + 0.2*np.random.randn(x.shape[0])
10.    return y
11.
12. x = np.arange(0, 20, 0.5) # generamos valores x de 0 a 20 en intervalos
de 0.5
13. y = f(x) # calculamos y a partir de la función que hemos generado
14.
15. # hacemos un gráfico de los datos que hemos generado
16. plt.scatter(x,y,label='data', color='blue')
17. plt.title('Datos');

```

Generamos una función lineal random con ruido



Graficamos la data

```

1. # Importamos la clase de Regresión Lineal de scikit-learn
2. from sklearn.linear_model import LinearRegression
3.
4. regresion_lineal = LinearRegression() # creamos una instancia de
LinearRegression
5.
6. # instruimos a la regresión lineal que aprenda de los datos (x,y)
7. regresion_lineal.fit(x.reshape(-1,1), y)
8.
9. # vemos los parámetros que ha estimado la regresión lineal
10. print('w = ' + str(regresion_lineal.coef_) + ', b = ' +
str(regresion_lineal.intercept_))
11.
12. # resultado: w = [0.09183522], b = 1.2858792525736682

```

Entrenamos el modelo e imprimimos los datos que calculo el modelo de w y b

```

1. # vamos a predecir y = regresion_lineal(5)
2. nuevo_x = np.array([5])
3. prediccion = regresion_lineal.predict(nuevo_x.reshape(-1,1))
4.
5. print(prediccion)
6.
7. # resultado: [1.7449]

```

Realizamos una predicción

```

1. # importamos el cálculo del error cuadrático medio (MSE)
2. from sklearn.metrics import mean_squared_error
3.
4. # Predecimos los valores y para los datos usados en el entrenamiento
5. prediccion_entrenamiento = regresion_lineal.predict(x.reshape(-1,1))
6.
7. # Calculamos el Error Cuadrático Medio (MSE = Mean Squared Error)
8. mse = mean_squared_error(y_true = y, y_pred = prediccion_entrenamiento)
9.
10. # La raíz cuadrada del MSE es el RMSE
11. rmse = np.sqrt(mse)
12.
13. print('Error Cuadrático Medio (MSE) = ' + str(mse))
14. print('Raíz del Error Cuadrático Medio (RMSE) = ' + str(rmse))

```

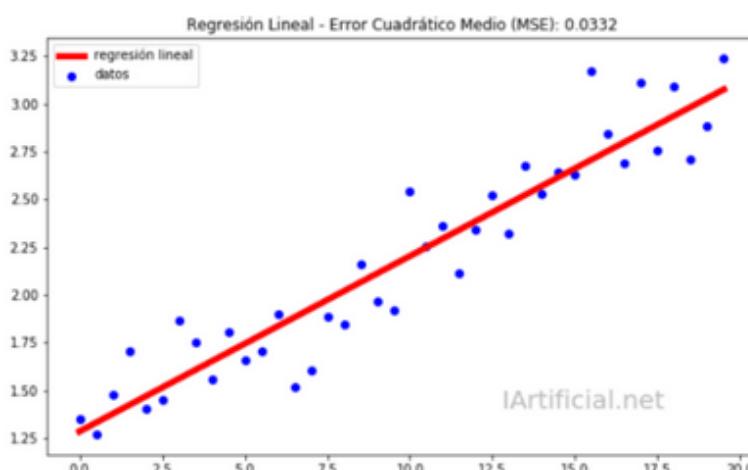
Por ultimo calculamos el error cuadratico medio y la raiz cuadrada de este.

Ahora si queremos medir la calidad del modelo podemos usar el coeficiente de determinación R² que va entre 0 y 1 para esto podemo implementarlo usando los siguiente elementos:

```

1. # calculamos el coeficiente de determinación R2
2. r2 = regresion_lineal.score(x.reshape(-1,1), y)
3.
4. print('Coeficiente de Determinación R2 = ' + str(r2))

```

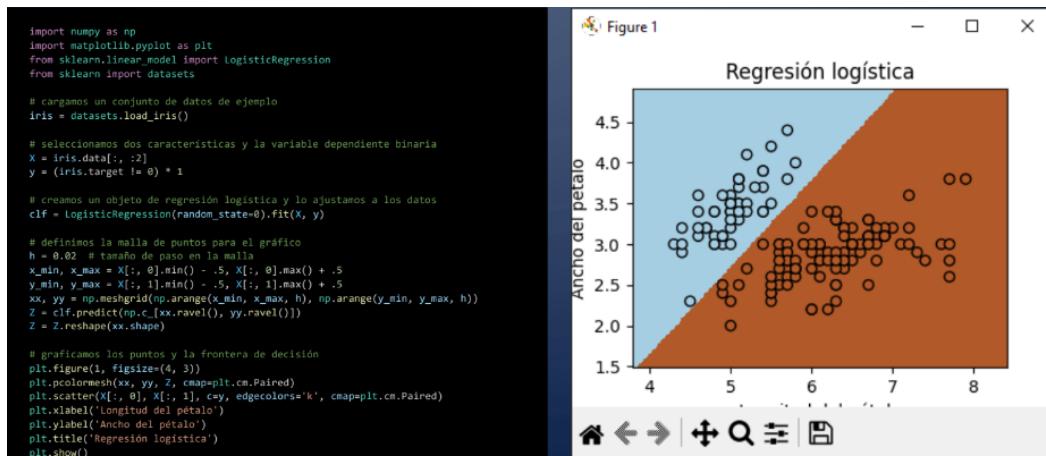


Regresión Logistica

Este algoritmo se utiliza para la clasificación binaria y se basa en la regresión lineal. En lugar de predecir valores continuos, la regresión logística predice la probabilidad de que una muestra pertenezca a una clase particular. Analiza la relación entre una variable dependiente binaria (que toma dos valores posibles, como sí/no, verdadero/falso, éxito/fracaso, etc.) y una o más variables independientes (que pueden ser continuas, categóricas o de cualquier otro tipo). El objetivo es

predecir la probabilidad de que la variable dependiente tome uno de los dos valores posibles, en función de los valores de las variables independientes. El modelo se basa en una función logística, que transforma la variable dependiente en una escala continua entre 0 y 1, lo que representa la probabilidad de que la

variable dependiente tome el valor de "éxito" en lugar del valor de "fracaso". Se utiliza comúnmente en campos como la epidemiología, la biología, la psicología y la investigación de mercado, entre otros, para analizar datos binarios y hacer predicciones sobre la probabilidad de un resultado determinado. Un ejemplo de implementación es el siguiente algoritmo sobre plantas



1. Se un conjunto de datos de ejemplo de la biblioteca Scikit-learn que contiene mediciones de longitud y ancho de pétalos de tres especies diferentes de flores iris.
2. Seleccionamos solo las dos primeras características (longitud y ancho del pétalo) y creamos una variable dependiente binaria que indica si la especie es iris-setosa (valor 0) o una especie diferente (valor 1).
3. Luego creamos un objeto de regresión logística utilizando el método LogisticRegression
4. Ajustamos el modelo a los datos utilizando el método fit.
5. Definimos una malla de puntos para el gráfico utilizando la función meshgrid de NumPy
6. Hacemos predicciones en cada punto de la malla utilizando el método predict del modelo.
7. Graficamos

La regresión logística es un algoritmo de clasificación y no un algoritmo de regresión.

Analisis discriminante lineal (LDA)

1. Es una técnica de análisis estadístico que se utiliza para encontrar una función lineal que maximice la separación entre dos o más grupos de datos.
2. Es un algoritmo de clasificación que se utiliza para separar las muestras en diferentes clases en función de sus características.
3. Este algoritmo busca una combinación lineal de las características que maximice la separación entre las clases.
4. El objetivo del LDA es encontrar una función lineal que pueda usarse para clasificar nuevos datos en uno de los grupos conocidos.

5. Esta función se deriva de las medias y covarianzas de los grupos de datos existentes.
6. El LDA se utiliza comúnmente en el análisis de datos para clasificar a los individuos en diferentes grupos en función de sus características observadas.
7. Se usa en reconocimiento de voz, huellas, clasificación de imágenes
8. Tiene limitaciones como:
 - a. Supone que las distribuciones de los datos en cada grupo son normales y que tienen matrices de covarianza iguales.
 - b. Sin embargo, a menudo es una técnica efectiva y útil para la clasificación de datos en situaciones en las que se conocen las etiquetas de los datos de entrenamiento.

Un ejemplo de implementación puede ser el siguiente:

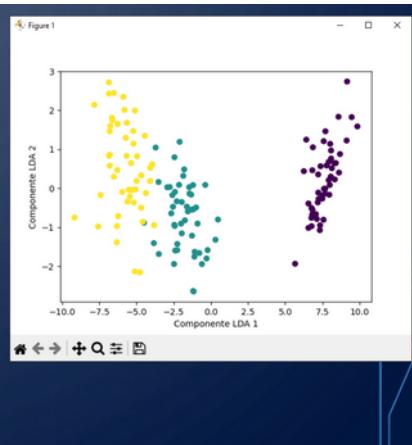
```
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

# Cargar el conjunto de datos Iris
iris = load_iris()

# Definir las características y las etiquetas
X = iris.data
y = iris.target

# Crear un modelo de LDA y ajustarlo a los datos
lda = LinearDiscriminantAnalysis()
X_lda = lda.fit_transform(X, y)

# Crear un gráfico de dispersión de los dos primeros componentes LDA
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y)
plt.xlabel('Componente LDA 1')
plt.ylabel('Componente LDA 2')
plt.show()
```



Utiliza la transformación LDA para reducir la dimensionalidad de los datos a dos componentes principales. Se utiliza un gráfico de dispersión para visualizar los dos primeros componentes LDA. Se utiliza el color para representar las diferentes etiquetas de las flores.

Perceptrón

Es un algoritmo de aprendizaje supervisado que se utiliza para la clasificación binaria. El perceptrón busca un hiperplano que separe las muestras en dos clases. Si las muestras no son linealmente separables, el perceptrón no podrá encontrar una solución. Sus características de funcionamiento son:

1. Utiliza un umbral de decisión que separa los dos grupos.
2. El algoritmo recibe un conjunto de datos etiquetados como entrada y ajusta los pesos de las características de entrada para encontrar un umbral de decisión que clasifique correctamente los datos.
3. El perceptrón toma una entrada, calcula la suma ponderada de las características y la compara con un umbral de decisión para producir una salida.
4. Si la suma ponderada es mayor que el umbral de decisión, la salida es positiva; de lo contrario, es negativa.

Un ejemplo de implementación puede ser el siguiente:

```

import numpy as np

class Perceptron:
    def __init__(self, learning_rate=0.1, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iterations):
            for i in range(n_samples):
                y_predicted = np.dot(X[i], self.weights) + self.bias
                if y_predicted >= 0:
                    y_predicted = 1
                else:
                    y_predicted = -1
                self.weights += self.learning_rate * (y[i] - y_predicted) * X[i]
                self.bias += self.learning_rate * (y[i] - y_predicted)

    def predict(self, X):
        y_predicted = np.dot(X, self.weights) + self.bias
        return np.where(y_predicted >= 0, 1, -1)

```

1. Este código implementa la clase Perceptron en Python.
2. El método fit ajusta los pesos del perceptrón utilizando los datos de entrada y las etiquetas.
3. El método predict utiliza los pesos ajustados para predecir las etiquetas de un nuevo conjunto de datos.

Maquinas de Vectores de soporte (SVM)

Son algoritmos de clasificación y regresión que se utilizan para separar las muestras en diferentes clases. Los SVM buscan el hiperplano que maximiza la distancia entre las muestras de diferentes clases. El hiperplano es una superficie de decisión que divide el espacio de características en dos regiones separadas, una para cada clase.

SVM encuentra el hiperplano óptimo maximizando la distancia entre los puntos de datos de cada clase y el hiperplano, lo que se conoce como margen.

En el caso de datos no linealmente separables, SVM utiliza una técnica llamada kernel trick para mapear los datos a un espacio de características de mayor dimensión, donde los datos son más fácilmente separables.

Una vez encontrado el hiperplano óptimo, SVM puede clasificar nuevos datos basándose en su posición en relación al hiperplano.

Un ejemplo de funcionamiento es:

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC

# Cargar el conjunto de datos de Iris y utilizar solo dos características
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

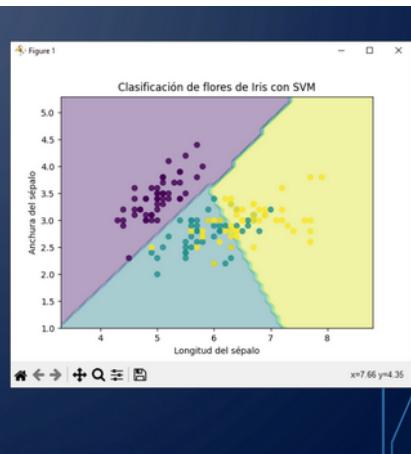
# Crear un modelo SVM y ajustarlo al conjunto de datos
svm = SVC(kernel='linear')
svm.fit(X, y)

# Crear una malla de puntos para el gráfico
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))

# Realizar predicciones en la malla de puntos
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Crear el gráfico y mostrar los puntos de datos y el hiperplano
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X[:, 0], X[:, 1], c=y, alpha=0.8)
plt.xlabel('Longitud del sépalo')
plt.ylabel('Anchura del sépalo')
plt.title('Clasificación de flores de Iris con SVM')
plt.show()

```



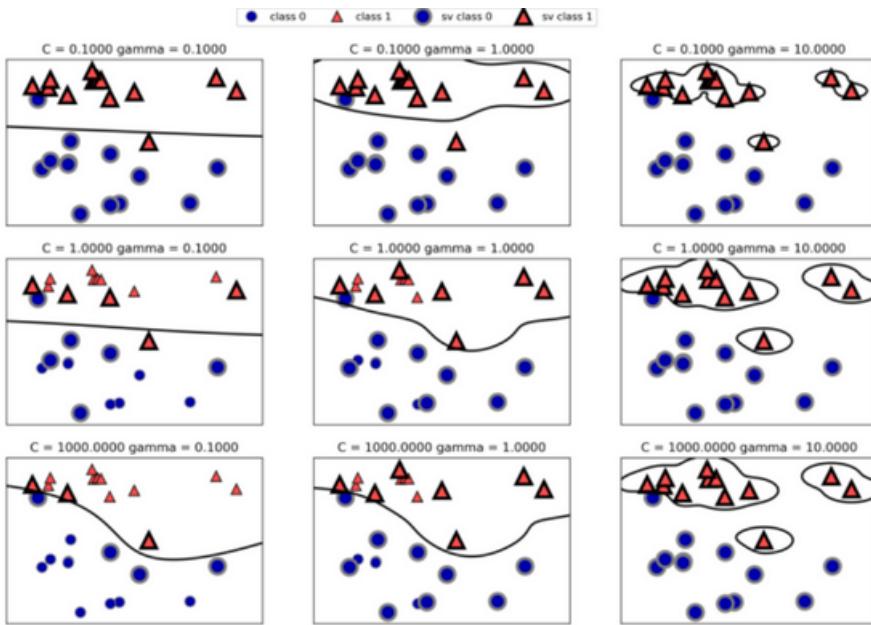
1. Se carga el conjunto de datos de iris y se utilizan solo dos características (longitud y anchura del sépalo).
2. Se crea un modelo SVM y se ajusta al conjunto de datos.
3. Luego, se crea una malla de puntos para el gráfico
4. Se realizan predicciones en cada punto de la malla.
5. Se muestra el gráfico con los puntos de datos, el hiperplano y el margen.

Existen tres propiedades que hacen atractivos SVM:

1. Construyen un separador de márgenes máximos
 - a. Límite de separador con la distancia posible más grande a puntos de la muestra.
 - b. Esto ayuda a generalizar bien.
2. Crean un hiperplano de separadores lineales
 - a. Tienen la habilidad de embeber los datos a un espacio de dimensión mayor utilizando un mecanismo llamado kernel trick
3. Son métodos no paramétricos – retienen los valores de entrenamiento.
 - a. En la práctica a menudo terminan reteniendo solo una fracción pequeña de las muestras.
 - b. Combinan las ventajas de los modelos no paramétricos y paramétricos: tienen la flexibilidad de representar funciones complejas pero son resistentes al overfitting

Yendo más a lo específico el keneltrick es una técnica no lineal que permite hacer esta separación. Utiliza dos formas para mapear, el kernel polinomial y el kernel gaussiano. El gaussiano usa dos parámetros conocidos como gamma y c.

El gamma determina una influencia de cada elemento sobre la línea de separación. Y el c es un parámetro de regularización. En la práctica el gamma da una idea de fuerza gravitacional de los elementos que hace que la línea de separación se acerque a los objetos y el C es un límite de esa fuerza. Gráficamente lo podemos ver en la siguiente imagen



1. De izquierda a derecha se incrementa gamma de 0.1 a 10.
2. Con un valor de gamma bajo muchos puntos se consideran cercanos. Esto significa un modelo de menos complejidad.
3. Un valor alto de gamma considera como cercanos a pocos puntos, por lo que el modelo tiene mayor complejidad.
4. De arriba hacia abajo se incrementa C de 0.1 a 1000.
5. Un C pequeño significa un modelo menos restringido donde cada punto tiene influencia limitada. Se puede ver eso arriba a la izquierda donde el modelo parece casi lineal.
6. Incrementando C, como se puede ver abajo a la izquierda, permite que esos puntos tengan más influencia en el modelo

Arboles de Decisión

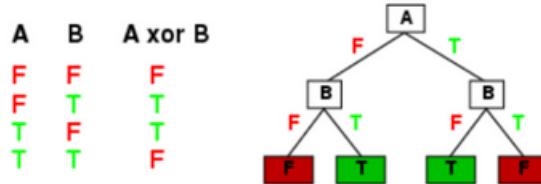
Son un algoritmo de aprendizaje supervisado utilizado para la clasificación y la regresión.

Se basan en una estructura de árbol donde cada nodo representa una característica (o atributo) y cada borde representa una regla de decisión basada en esa característica.

El árbol de decisión comienza con un nodo raíz que representa la característica más importante del conjunto de datos y se divide en dos o más nodos secundarios según una regla de decisión.

Los nodos secundarios se dividen a su vez en más nodos hasta que se llega a los nodos hoja, que representan las clases o valores de regresión finales. El proceso de construir un árbol de decisión implica elegir la mejor característica para dividir en cada nodo.

Un ejemplo posible es tomar un XOR



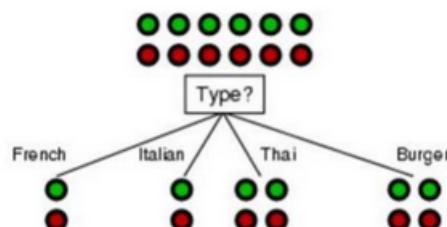
Por ejemplo se pueden construir arboles que representen una situación real de esperar o no esperar en un restaurante teniendo en cuenta las características relevantes

Example	Attributes											Target Wait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est		
X ₁	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T	
X ₂	T	F	F	T	Full	\$	F	F	Thai	30-60	F	
X ₃	F	T	F	F	Some	\$	F	F	Burger	0-10	T	
X ₄	T	F	T	T	Full	\$	F	F	Thai	10-30	T	
X ₅	T	F	T	F	Full	\$\$\$	F	T	French	>60	F	
X ₆	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T	
X ₇	F	T	F	F	None	\$	T	F	Burger	0-10	F	
X ₈	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T	
X ₉	F	T	T	F	Full	\$	T	F	Burger	>60	F	
X ₁₀	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F	
X ₁₁	F	F	F	F	None	\$	F	F	Thai	0-10	F	
X ₁₂	T	T	T	T	Full	\$	F	F	Burger	30-60	T	

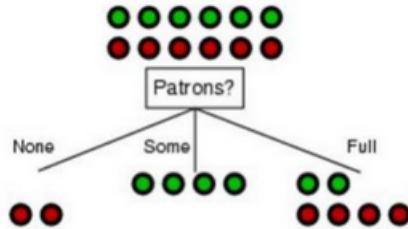
En este caso los $x_1 \times 2 \times 3 \dots \times n$ representan las diferentes personas de la muestra y como se comportan (esperan o no) en función a las diferentes características.

Ahora para determinar que característica es relevante y cual no para determinar el árbol de decisión, podemos testear atributos. Por ejemplo podemos analizar "Type" o "Patrons".

Type es un atributo pobre porque nos deja con 4 resultados que tiene la misma cantidad de positivos que de negativos.



Patrons es un atributo mejor porque si el valor es None o Some, tenemos conjuntos por los cuales podemos responder definitivamente si y no. Si el valor es Full, tenemos una mezcla de valores y necesitamos más información



En función a esto puedo crear diferentes arboles de decisión pero que no me terminan de clasificar de forma exacta la muestra. Para esto lo ideal es llegar al atributo perfecto que divide las muestras en conjuntos de los cuales son TODOS positivos o TODOS negativos. Para esto podemos implementar el algoritmo Choose-Attribute que utiliza la entropía

$$H(V) = I(P(v_1), \dots, P(v_n)) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$

La entropía es una medida de incertidumbre o aleatoriedad en los datos, y se usa para medir la cantidad de información contenida en un mensaje. La entropía se calcula a partir de la probabilidad de los eventos que componen el conjunto de datos. Si todos los eventos son igualmente probables, la entropía es máxima y el conjunto de datos es muy aleatorio e incierto. Si todos los eventos tienen la misma probabilidad, la entropía es mínima y el conjunto de datos es más predecible y menos incierto. Entonces para un set de datos de entrenamiento que contiene p muestras positivas y n muestras negativas el calculo responde a la siguiente formula:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Ahora un atributo elegido A divide el conjunto entrenamiento E en distintos subconjuntos de acuerdo a la cantidad de opciones de A. Para calcular el remanente

$$\text{remainder}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

La ganancia de la información es una reducción de la entropía desde el atributo de prueba:

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{remainder}(A)$$

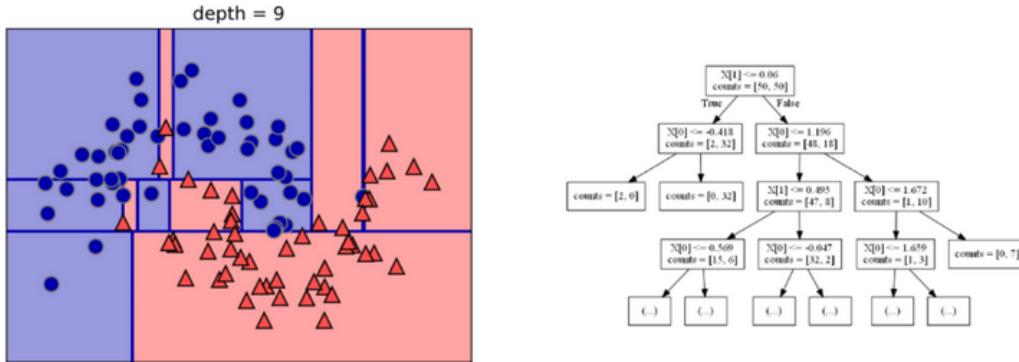
En definitiva se elige el atributo con el IG más grande. En nuestro ejemplo:

$$IG(\text{Patrons}) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = .0541 \text{ bits}$$

$$IG(\text{Type}) = 1 - \left[\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

Patrones tienen el IG más alto de todos los atributos y, por lo tanto, el algoritmo DTL lo elige como la raíz

Ahora al momento de crear un arbol tenemos que tener cuidado en no caer en overfitting



En donde a mayor clasificación mayor riesgo de overfitting. Para evitar esto existen dos tecnicas:

1. Pre-pruning: frenar la creación del árbol de antemano. Esto se puede hacer: limitando la profundidad del árbol limitando la cantidad de hojas a un número máximo requerir un número mínimo de puntos en un nodo para seguir haciendo split
2. Post-pruning: crear el árbol completamente y luego remover o colapsar todos los nodos que proporcionan poca información.

Todo esto se realiza para generar un ensamble, que son metodos que combinan multiples modelos de ML para crear modelos más poderosos. Y aca es donde entra el siguiente concepto

Random Forest

1. Uno de los problemas de los árboles de decisión es que tienen a hacer overfitting.
2. Un random forest es una colección de árboles de decisión donde cada árbol es levemente diferente a los otros.
3. La idea detrás es que cada árbol es bueno para hacer una predicción, pero hacer overfitting de parte de los datos.
4. Si se construyen diferentes árboles se puede reducir el overfitting promediando sus resultados.
5. Se inserta randomización de alguna manera para que cada árbol sea diferente.
6. Existen dos formas de generar la randomización: seleccionar los puntos de datos (bootstrapping) o seleccionar las características
7. Para construir un random forest se debe decidir la cantidad de árboles a construir. Esto se hace con el parámetro `n_estimators` de `RandomForestRegressor` o `RandomForestClassifier`

Bootstraping

Implica seleccionar los puntos de datos aleatoriamente (selección con repetición, es decir el punto de dato seleccionado vuelve al set original)

Se crean nuevos datasets tan grandes como el dataset original pero sin algunos de sus datos (aproximadamente un tercio) mientras otros están repetidos

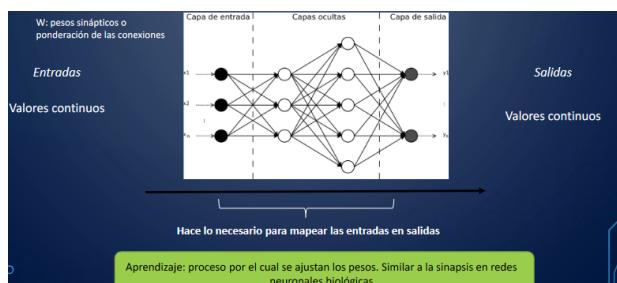
Un aspecto importante en random forest es tener en cuenta la cantidad de características que me genere un numero de arboles razonable. Muchas características implican muchos arboles que pueden generar overfitting.

Redes Neuronales

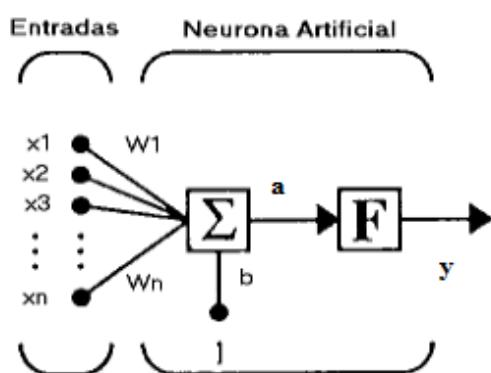
Neurona Artificial

1. Es una abstracción matemática de la neurona biológica
2. Es una unidad de procesamiento de la información
3. Dispositivo simple de cálculo que ante un vector de entradas proporciona una única salida
4. Los elementos de una neurona son:
 - a. Conjunto de entradas
 - b. Pesos sinápticos
 - c. Función de activación: Activa a o no la función de transferencia
 - d. Función de transferencia: Toma el input y lo normaliza
 - e. Bias: entrada constante de magnitud 1 y peso b

Una representación de esto puede ser:



Las neuronas de capa de entrada reciben los input y activan a las siguientes de la capa oculta que una vez que cumplen con su función se conectan a las neuronas de la capa de salida



En el proceso de aprendizaje teniendo la X y la Y de la función debe entrenarse para obtener los parámetros W y B de la función. Recordemos que tenemos $Y = Wx + B$

Perceptrón:

El perceptrón es una red neuronal de una sola capa que se utiliza principalmente para la clasificación binaria, es decir, para separar dos clases de datos.

Un perceptrón es una neurona artificial con una o mas entradas y una salida

Su comportamiento tiene un efecto lineal. Discretiza linealmente las entradas en salidas.

Ahora como toma un decisión:

1. **Perceptrón:** Es un modelo muy básico de cómo funcionan las neuronas en una red neuronal. Básicamente, toma varias entradas (números) y produce una salida, también un número.
2. **Entradas y pesos:** Cada entrada $x_1, x_2, x_3, \dots, x_1, x_2, x_3, \dots$ tiene asociado un "peso" $w_1, w_2, w_3, \dots, w_1, w_2, w_3, \dots$. Este peso representa qué tan importante es cada entrada para el resultado final.
3. **Suma ponderada:** El perceptrón calcula una "suma ponderada", que no es más que multiplicar cada entrada por su peso y luego sumar todos esos productos: $z = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots$
$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots$$
4. **Función de activación:** Después de calcular esa suma, se aplica algo llamado una "función de activación" (en este caso, es una función escalón, que se activa o "dispara" solo si el valor supera cierto umbral).
 - Si el valor de z es mayor que 0, el resultado es 1 (la neurona "se activa").
 - Si z es menor que 0, el resultado es 0 (la neurona "no se activa").

Esto lo puedes ver en la fórmula de Heaviside y la función "signo" $\text{sgn}(z)$ que muestra cómo cambia la salida según el valor de z

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \quad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

En resumen:

- El perceptrón toma varias entradas, las pondera (les da un peso), las suma y luego decide si la salida es 1 o 0 dependiendo del valor de esa suma. Esto es como un interruptor que decide si "encenderse" o no.



epochs son la cantidad de iteraciones

Ahora una neurona aislada dispone de poca potencia calculo. Los nodos se conectan y refuerzan mediante la sinapsis. Entonces:

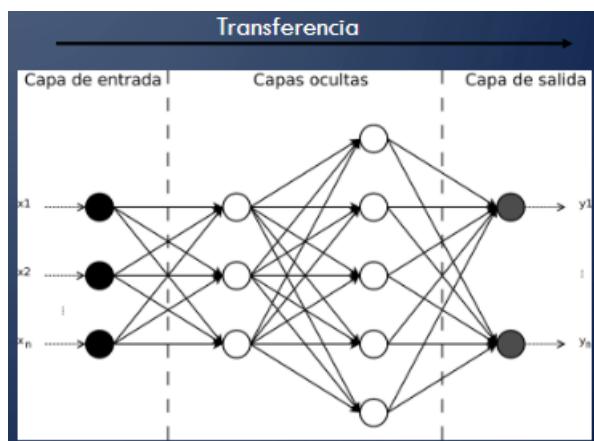
1. las neuronas se agrupan formando una estructura llamada capa
2. Los pesos pasan a ser matrices $W(n \times m)$
3. Las funciones de activación están centradas en cero. Para corregir se agrega una constante b
4. La salida de la red es un vector

A su vez las redes pueden ser de multiples capas en cascada en donde tenemos

- a. La capa de entrada, es la primera capa de la red, y su función es recibir la entrada del modelo y transmitirla a la capa oculta o capas ocultas. Cada neurona de la capa de entrada representa una variable de entrada del modelo, como una imagen o un conjunto de características
- b. La de salida, es la última capa de la red neuronal y su función es producir la salida final del modelo. Cada neurona de la capa de salida representa una clase o una etiqueta de salida del modelo.
- c. Y la oculta, son capas intermedias que procesan la información recibida de la capa de entrada y envían la información procesada a la siguiente capa oculta o a la capa de salida. Cada capa oculta está formada por un número variable de neuronas o nodos, y cada una de estas neuronas está conectada a las neuronas de la capa anterior y de la capa siguiente.

A su vez podemos mencionar dos características de estas conexiones

1. Si no hay realimentación, es decir:
 - a. No hay conexiones entre neuronas de la misma capa
2. Si hay realimentación, es decir:
 - a. Se genera un efecto memoria
 - b. La salida depende de la historia pasada



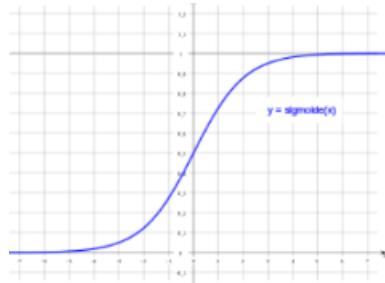
Existen otras capas que se pueden utilizar en ciertos tipos de redes neuronales:

1. La capa de convolución: se utilizan en las redes neuronales convolucionales (CNN), que son especialmente adecuadas para el procesamiento de imágenes. Las capas de convolución aplican un conjunto de filtros a la entrada para extraer características y patrones.
2. La capa de agrupación: se utilizan en las redes neuronales convolucionales y se utilizan para reducir la dimensionalidad de la salida de las capas de convolución.
3. La capa recurrente: se utilizan en las redes neuronales recurrentes (RNN), que se utilizan para el procesamiento de secuencias de datos, como el procesamiento del lenguaje natural. Las capas recurrentes permiten a la red recordar información de entradas anteriores y utilizarla en la toma de decisiones actuales.

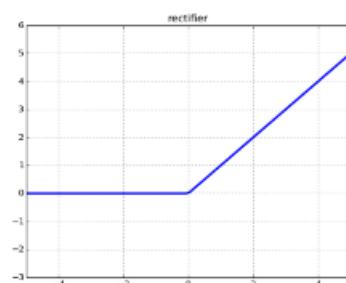
Funciones de Transferencia

Las funciones de transferencia son funciones matemáticas utilizadas en las redes neuronales artificiales para introducir no linealidad en el modelo y permitir la aproximación de funciones complejas. Las funciones de transferencia pueden ser:

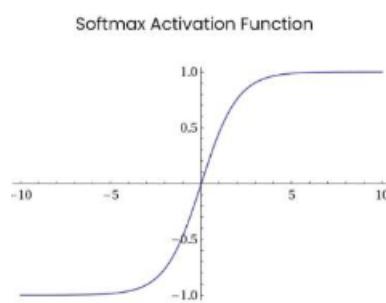
1. Función Sigmoid (Sigmoid): Es una función continua y diferenciable que tiene una forma de "S" y se utiliza comúnmente en las capas ocultas de las redes neuronales. Mapea cualquier valor de entrada a un valor entre 0 y 1, lo que la hace útil para representar probabilidades y estimaciones de probabilidad.



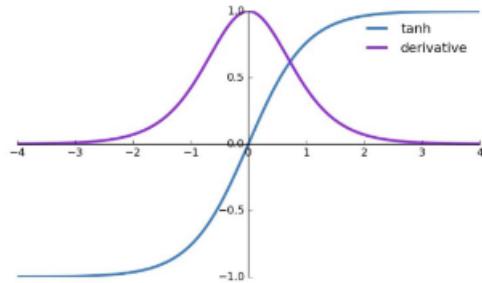
2. Función ReLU (Rectified Linear Unit); La función ReLU es una función no lineal que se utiliza comúnmente en las capas ocultas de las redes neuronales. Todo lo que viene en negativo me devuelve 0 y todo lo que viene en positivo me devuelve una recta lineal. Graficamente seria:



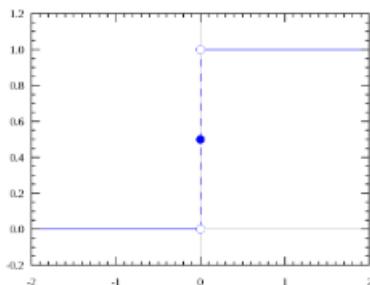
3. Función Softmax: Es utilizada en la capa de salida de las redes neuronales para la clasificación multiclase. Es mas suave que la sigmoide ya que se usa para buscar patrones que al momento de clasificar



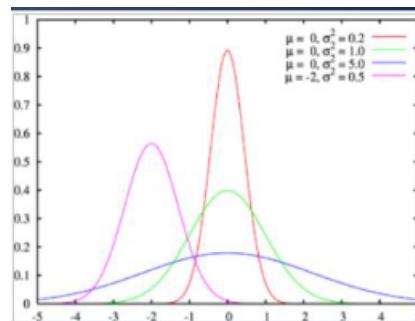
4. Función Tanh (Tangente hiperbólica): Es una función sigmoide centrada en cero que se utiliza comúnmente en las capas ocultas de las redes neuronales. Mapea cualquier valor de entrada a un valor entre -1 y 1, lo que la hace útil para la normalización de los datos de entrada.



5. Escalon: En donde cualquier valor negativo devuelve 0 y positivo devuelve 1



6. Gaussiana:



Entrenamiento

Estas funciones de transferencia son las que me van a devolver como salida y son las que tendremos que volver a entrenar luego de validar el resultado. Ahora cuando hablamos de entrenamiento recordemos que tenemos el supervisado (en donde tenemos la entrada etiquetada), el no supervisado que tenemos solo las entradas y se requiere encontrar las agrupaciones y por ultimo el de refuerzo que se entrena por premiación.

Regla Delta

También conocida como Regla de Widrow-Hoff, es un algoritmo de aprendizaje supervisado utilizado en redes neuronales para ajustar los pesos de las conexiones sinápticas. Esta regla de aprendizaje se utiliza para entrenar una red neuronal a partir de un conjunto de datos de entrenamiento etiquetados. Se utiliza para minimizar la función de costo cuadrático medio, que mide la diferencia entre la salida de la red neuronal y la salida deseada. Esta regla ajusta en función a si la salida de la red es mayor que la salida deseada, se reduce el peso de la conexión sináptica y si la salida de la red es menor que la salida deseada, se aumenta el peso de la conexión sináptica.

MPL (multicapas de perceptrón)

Es un tipo de red que se usa para clasificación y regresión. Es de alimentación hacia adelante en donde cada neurona está conectada a todas las neuronas de la capa anterior y posterior pero no a las neuronas de su propia capa

Cuando una red neuronal artificial contiene una gran cantidad de capas ocultas, se llama red neuronal profunda o (Deep Neural Network o DNN) y cuando las señales fluyen desde las entradas hacia las salidas se denomina Feedforward Neural Network (FNN)

Tipos de entrenamientos en grandes volúmenes

1. Pasada hacia adelante (forward pass)

- **Minibatch:** En lugar de entrenar la red con todos los datos a la vez, se utilizan pequeños conjuntos de datos llamados **minibatches** (por ejemplo, 32 muestras a la vez). Se repite este proceso muchas veces, y cada conjunto completo de entrenamiento se llama **época**.
- **Capas de la red:** Cada minibatch pasa por la red neuronal comenzando desde la **capa de entrada**. Esta capa envía las señales a la **primera capa oculta** de la red. El algoritmo calcula las salidas para cada neurona en la capa oculta.
- **Propagación:** El resultado de la primera capa se pasa a la siguiente capa y luego a la siguiente, hasta llegar a la **capa de salida**.
- **Cálculo del error:** Una vez que se obtiene la salida de la red, se compara con la **salida real deseada** (el resultado que deberíamos haber obtenido). La diferencia entre estos dos valores es el **error**, que se representará como un vector.

2. Pasada hacia atrás (backward pass)

- **Contribución al error:** Aquí se calcula cómo cada conexión en la red contribuyó al error total. Básicamente, se mide cuánto de ese error vino de cada neurona y de cada conexión entre capas.
- **Ajuste de pesos:** Con esta información, se ajustan los **pesos** de las conexiones de la red para reducir el error en futuras pasadas. Esto se hace modificando los pesos hacia la dirección correcta (esto es lo que llamamos **descenso por gradiente**).
- **Repetición:** Este proceso de ajustar los pesos se repite para cada minibatch de datos de entrenamiento, hasta que el error sea lo suficientemente bajo para todos los datos.

Resumen del proceso completo:

1. **Forward pass:** Enviamos los datos de entrada por la red, obtenemos una salida y calculamos el error.
2. **Backward pass:** Calculamos cómo ajustar los pesos de la red para reducir el error en el futuro. Este ajuste se realiza calculando las **derivadas** (esto es parte del proceso de descenso por gradiente).

Cuando hablamos de gradiente descendente hablamos de un algoritmo utilizado para optimizar redes neuronales y mejorar su rendimiento. La idea es ajustar los **parámetros internos** (los pesos) para que la red funcione mejor. El algoritmo sigue estos pasos:

- **Calcular el gradiente:** Esto es como encontrar en qué dirección debemos movernos para reducir el error de la red. Si te imaginas que el error es como una pendiente o colina, el gradiente es la dirección hacia abajo por la que debemos ir para llegar al punto más bajo.

- **Descenso iterativo:** El algoritmo ajusta los parámetros paso a paso, esperando que cada paso reduzca el error.
- **Minibatches:** A veces se toma un conjunto pequeño de datos (llamado minibatch) para hacer los cálculos, en lugar de usar todos los datos de una vez. Esto es lo que lo hace **estocástico** (aleatorio).

2. La metáfora del cuenco

Para entender el proceso, te proponen imaginar un **recipiente como para frutas**. Este cuenco representa la **función de costo o error**.

- Los valores actuales de los **pesos o coeficientes** (parámetros de la red) serían como un punto en la superficie del cuenco. Si el punto está cerca del borde, los pesos tienen un costo alto (mayor error).
- El objetivo es **mover ese punto hacia el fondo del cuenco**, que es donde el error es el menor posible. Este punto en el fondo del cuenco representa el **mejor conjunto de pesos**.
- Cada vez que probamos nuevos valores de los pesos, medimos el costo y ajustamos los pesos para encontrar valores que reduzcan ese costo, acercándonos cada vez más al fondo del cuenco.

Algunas definiciones utiles hasta el momento:

Epoch, batch size, iteration

- **Epoch:** Un **epoch** es un ciclo completo donde todo el conjunto de datos de entrenamiento pasa por la red neuronal una vez. En otras palabras, es una pasada completa por todos los datos.
- **Batch size:** Dado que es difícil procesar todos los datos de una sola vez, se dividen en **batches** o lotes más pequeños. El **batch size** es la cantidad de ejemplos de entrenamiento que se usan en una sola iteración. Por ejemplo, en lugar de procesar 10,000 datos a la vez, podrías procesar 32 o 64 en un batch.
- **Iteración:** Una **iteración** es el número de lotes necesarios para completar un epoch. Si tienes 1,000 datos y el tamaño de tu batch es de 100, necesitarás 10 iteraciones para completar un epoch.

2. Impacto del número de epochs

- **Cambios de parámetros:** Con cada epoch, se ajustan los **pesos** de la red. Al pasar más epochs, los pesos se ajustan más veces, lo que puede llevar a un mejor ajuste. Sin embargo, demasiados epochs pueden provocar **overfitting** (sobreajuste), donde la red aprende demasiado bien los datos de entrenamiento pero no generaliza bien para nuevos datos.
- **Underfitting:** Si se usa solo un epoch, no se entrenará lo suficiente y el modelo no aprenderá adecuadamente (underfitting). Por eso, generalmente se usan múltiples epochs.
- **¿Cantidad óptima de epochs?**: No hay una cantidad fija. Depende de los datos que estés usando. Si usas demasiados, podrías tener **overfitting**; si usas muy pocos, tendrás **underfitting**.

Resumen:

- **Epochs** son ciclos completos por los datos de entrenamiento.

- **Batch size** determina cuántos ejemplos de datos se usan en cada iteración.
- **Iteraciones** son las veces que tienes que pasar batches para completar un epoch.
- **Learnign rate**, Es el hiperparámetro más importante. En general la tasa de aprendizaje óptima es la mitad de la máxima. Una forma de encontrar un buen learning rate es entrenar el modelo para unos cientos de iteraciones, comenzando con un learning rate bajo (ejemplo 0.00001) y gradualmente incrementarlo a un valor grande (ejemplo 10). Esto se hace multiplicando el learning rate por un factor constante en cada iteración.
- El **Batch Gradient Descent** procesa todo el conjunto de datos en una sola vez.
- El **SGD** ajusta después de cada ejemplo individual.
- El **Mini-Batch Gradient Descent** es una combinación que equilibra eficiencia y rendimiento.

Ejemplo explicado:

- **Set de datos:** Tienes un conjunto de datos de **200 muestras**. Estas son las 200 entradas que vas a usar para entrenar tu red neuronal.
- **Tamaño del batch:** El tamaño del batch elegido es **5**. Esto significa que, en lugar de pasar todas las 200 muestras de una vez por la red, las vas a procesar en pequeños grupos (batches) de 5 muestras a la vez.
- **División en batches:** Como tienes 200 muestras y un tamaño de batch de 5, tu conjunto de datos se dividirá en **40 batches** ($200 \div 5 = 40$). Entonces, en cada epoch, tu red procesará 40 batches de 5 muestras cada uno.
- **Actualización de pesos:** Los pesos de la red se actualizan **después de cada batch**. Esto significa que, por cada 5 muestras, la red ajusta sus pesos para mejorar su rendimiento.
- **Epochs:** Si se eligen **1000 epochs**, esto significa que todo el proceso de pasar por las 200 muestras (divididas en 40 batches) se repetirá **1000 veces**. En cada epoch, la red hará 40 actualizaciones de pesos, una por cada batch.

En resumen:

- Tienes **200 datos** que se dividen en **40 batches** de **5 datos** cada uno.
- En cada epoch, la red neuronal procesa los 40 batches, y **ajusta sus pesos 40 veces** por epoch.
- Si tienes **1000 epochs**, eso significa que el proceso completo se repetirá 1000 veces.
- Ahora en cada iteración con un learning rate alto se me puede pasar el modelo y predecir mal. Un learning rate bajo puede ser imperceptible la mejora en cada epoch
- Mayor cantidad de neuronas nos dan mejores predicciones
- En general ReLu funciona bien para capas ocultas

Algunos inconvenientes de las redes neuronales son:

- .Inconvenientes
- Tiempo de entrenamiento no acotado.
- Dependiente de las condiciones iniciales
- Desvanecimiento de los gradientes
- Mínimos locales
- Underfitting

- Memorización o Sobreaprendizaje
- Caracterización de la red. ¿Cantas capas, cuantas neuronas en cada capa,...?

Deep Learning

Deep Learning es una subárea del aprendizaje automático (**machine learning**) que involucra redes neuronales profundas (es decir, redes con muchas capas). El avance del hardware en la última década ha permitido que este campo crezca enormemente, y se han desarrollado técnicas para resolver ciertos problemas en redes neuronales, como:

- **Inicialización de los pesos sinápticos:** El proceso de asignar valores iniciales a los pesos de la red. Si se hace de manera incorrecta, la red puede entrenarse mal o de manera muy lenta.
- **Desvanecimiento de los gradientes:** A medida que se propagan los gradientes a través de muchas capas en la red, pueden volverse extremadamente pequeños, haciendo que los ajustes de pesos sean ineficientes. Esto es un problema importante en redes neuronales profundas.
- **Extracción de características:** Se refiere a cómo las redes pueden aprender a identificar patrones relevantes en los datos. Redes más profundas suelen ser más efectivas en este proceso.

2. Métodos de Deep Learning

Hay varias técnicas y tipos de redes utilizadas en Deep Learning, algunas de las cuales son:

- **Randomly Initialized RNN:** Redes Neuronales Recurrentes, que son especialmente buenas para procesar secuencias de datos, como series temporales o lenguaje natural.
- **Bayesian Triphone GMM-HMM:** Modelos que combinan técnicas probabilísticas (como los modelos de mezcla gaussiana, GMM) y modelos ocultos de Markov (HMM), muy utilizados en procesamiento de voz.
- **Convolutional DNN:** Redes Neuronales Convolucionales (CNN), muy útiles para tareas como el reconocimiento de imágenes, ya que pueden identificar características espaciales en los datos.
- **Bidirectional LSTM:** Una variante de las redes neuronales recurrentes, que puede aprender tanto del pasado como del futuro en una secuencia de datos.
- **Restricted Boltzmann Machines:** Modelos energéticos que se utilizan para representar distribuciones de probabilidad sobre conjuntos de datos.
- **Autoencoders:** Redes diseñadas para aprender a comprimir los datos de entrada en una representación más compacta y luego reconstruirlos. Son útiles para la reducción de dimensionalidad y la extracción de características.

3. Autoencoders

El objetivo principal de un **autoencoder** es aprender una representación eficiente de los datos, extrayendo las **características profundas**. A través de este proceso, los autoencoders aprenden a **comprimir** la información de la capa de entrada y luego a **descomprimirla** para que la salida sea lo más similar posible a la entrada original.

- **Mínimo global:** Los autoencoders buscan ajustar sus pesos de tal manera que el error (la diferencia entre la entrada y la salida) sea lo más bajo posible.

- **Aplicaciones:** Los autoencoders se utilizan en tareas como la reducción de ruido en imágenes, la detección de anomalías o la reducción de la dimensionalidad de datos complejos.

Resumen:

- **Deep Learning** involucra redes neuronales más complejas y profundas, con varias técnicas avanzadas para mejorar su rendimiento.
- **Autoencoders** son un tipo especial de red utilizada para comprimir y descomprimir datos, extrayendo características clave.

Aplicaciones de Deep Learning

- Reconocimiento de imágenes
- Reconocimiento de voz
- Traducción automática
- Generación de texto
- Conducción autónoma
- Diagnóstico médico.

Repaso de Modelos

Regresión lineal

- Utilizado para problemas de regresión, donde se busca predecir un valor numérico a partir de variables independientes.

Regresión logística

- Empleado para problemas de clasificación binaria, donde se intenta predecir una de dos clases posibles.

Árboles de decisión:

- Modelos de aprendizaje supervisado que se utilizan tanto para problemas de clasificación como de regresión.
- Dividen el conjunto de datos en nodos para tomar decisiones basadas en las características.

Bosques aleatorios (Random Forests):

- Una extensión de los árboles de decisión que combina múltiples árboles para mejorar la precisión y reducir el sobreajuste.

Máquinas de vectores de soporte (SVM):

- Utilizadas principalmente en problemas de clasificación, buscan encontrar el hiperplano que mejor separa las clases en el espacio de características.

K-Nearest Neighbors (K-NN):

- Un algoritmo de clasificación que se basa en encontrar los "k" puntos de datos más cercanos para determinar la clase de un nuevo punto.

Redes neuronales artificiales (ANN):

- Modelos inspirados en el funcionamiento del cerebro que se utilizan para una amplia variedad de tareas, incluyendo clasificación, regresión, y procesamiento de lenguaje natural.

Redes neuronales convolucionales (CNN):

- Específicas para tareas de visión por computadora, estas redes son ideales para detectar patrones en imágenes.

Redes neuronales recurrentes (RNN):

- Diseñadas para trabajar con datos secuenciales, como texto o series temporales.

Redes neuronales LSTM y GRU:

- Variantes de RNN que abordan el problema de las secuencias largas y la desaparición del gradiente.

Máquinas de Boltzmann restringidas (RBM):

- Utilizadas en tareas de recomendación y reducción de dimensiones.

Máquinas de aprendizaje extremo (Extreme Learning Machines, ELM):

- Enfoque de aprendizaje automático que se caracteriza por una rápida capacidad de entrenamiento y buenas tasas de generalización.

Modelos de agrupamiento (clustering):

- Algoritmos como K-Means, DBSCAN, y jerárquicos, utilizados para encontrar grupos o clusters en los datos.

Modelos de descomposición de matriz:

- Usados en recomendación y filtrado colaborativo.

Modelos de generación de lenguaje:

- Incluyen GPT (Generative Pre-trained Transformer), LSTM basados en atención, y otros modelos para generar texto y lenguaje natural.

Modelos de aprendizaje por refuerzo (Reinforcement Learning):

- Se utilizan para entrenar agentes que toman decisiones secuenciales en un entorno y maximizan una recompensa.

Algoritmos de agrupamiento espectral:

- Utilizados en análisis de datos de alta dimensionalidad y redes sociales.

Modelos de detección de anomalías:

- Utilizados para identificar patrones inusuales en los datos.

Características de Deep Learning:

- Redes Neuronales Profundas: El término "profundo" se refiere a la profundidad de las redes neuronales utilizadas en deep learning
- Múltiples capas de neuronas interconectadas, lo que les permite aprender representaciones complejas de datos a partir de características más simples
- Aprendizaje Automático de Características: aprende automáticamente las características relevantes de los datos en lugar de depender de características diseñadas

manualmente. Esto hace que el proceso de ingeniería de características sea menos necesario en comparación con otros enfoques de aprendizaje automático.

- Aprendizaje Supervisado y No Supervisado: Se aplica tanto a problemas supervisados (como clasificación y regresión) como a problemas no supervisados (como agrupamiento y reducción de dimensionalidad)
- Grandes Conjuntos de Datos: El deep learning a menudo requiere grandes conjuntos de datos para entrenar modelos eficaces. Esto se debe a que las redes neuronales profundas tienen una gran cantidad de parámetros y necesitan una gran cantidad de datos para generalizar correctamente
- Hardware Especializado: El entrenamiento de redes neuronales profundas puede ser intensivo en términos de recursos computacionales. Las aplicaciones de deep learning se benefician de hardware especializado, como unidades de procesamiento gráfico (GPU) y unidades de procesamiento tensorial (TPU).

Ingeniería de Características

El machine learning ajusta los modelos matemáticos a los datos para obtener información relevante y/o hacer predicciones. Estos modelos toman como base determinadas características, que son representación numérica de un aspecto de los datos sin procesar. Esta ingeniería de las características es un proceso de selección, extracción y transformación de características o atributos relevantes de los datos para mejorar el rendimiento de los modelos de aprendizaje automático. Extrae los datos en crudo y los convierte en formatos adecuados para el machine learning.

Se seleccionan y se transforman las variables o características de entrada para hacer que los modelos de aprendizaje automático sean más precisos y eficientes en la resolución de un problema específico. Hallar y utilizar las características correctas puede aliviar la dificultad del modelado y, por lo tanto, permitir que se generen resultados de mayor calidad.

Ahora los datos provienen de observar el mundo real, cada dato proporciona una pequeña ventana a un aspecto limitado de la realidad. La recopilación de todas estas observaciones nos da una imagen del conjunto. Pero la imagen es desordenada porque está compuesta por mil pequeñas piezas.

Para ordenar los datos utilizamos modelos que tratan de describir las relaciones entre diferentes aspectos de dichos datos

Las fórmulas matemáticas de los modelos relacionan cantidades numéricas entre sí, pero no siempre los datos sin procesar son numéricos. Se debe incluir una pieza que conecte el mundo de las fórmulas con el mundo de los datos y ahí es donde entra ingeniería de las características. Antes de seguir veamos primero qué es una característica:



Una característica es una representación numérica de datos sin procesar. La ingeniería de las características es el proceso de formular las características más apropiadas dados los datos, el modelo y la tarea.

El número de características también es importante. Si no hay suficientes, entonces el modelo no podrá realizar la tarea. Si hay demasiadas, o si la mayoría de ellas son irrelevantes, entonces el modelo será más costoso y difícil de entrenar.

Las características pueden ser:

- a. Numericas
- b. Categoricas de texto

Se pueden obtener mediante técnicas de transformación:

- Normalización
- Escalado
- Discretización
- Codificación one-hot

Variables categoricas

- Son variables que representan una característica o una cualidad que puede ser dividida en categorías o grupos discretos,
- No tienen un valor numérico y no se pueden medir en una escala continua
- Se clasifican en grupos o categorías:
 - Nominales: No tienen un orden natural, lo que significa que no se pueden clasificar en un orden específico. (Color de pelo, Lugar de nacimiento, etc.)
 - Ordinales: Tienen un orden natural, lo que significa que se pueden clasificar en un orden específico. (Nivel de estudios, poder adquisitivo, etc.)

Otras divisiones posibles de las categoricas:

1. La variable cardinal permite realizar cálculos numéricos, conocer las medias o las varianzas.
2. La variable continua puede tomar cualquier valor dentro de un rango determinado. Permiten obtener por ej. la moda o los porcentajes de categorías.
3. Las categóricas son muy útiles para conocer información de tipo cualitativo, es decir, alguna cualidad de los datos. Las continuas nos aportan datos cuantitativos, es decir, cantidades y valores representados por números.

Grados de libertad

1. Número de valores independientes o componentes de un conjunto de datos que pueden variar.
2. En el contexto de diferentes pruebas estadísticas, los grados de libertad tienen significados específicos.
 - a. En la prueba de chi-cuadrado, los grados de libertad se refieren a la cantidad de categorías o grupos que pueden variar en una distribución de frecuencias observadas. En una prueba

de chi-cuadrado de independencia, se calcula como (número de filas - 1)* (número de columnas - 1) en la tabla de contingencia.

- b. En el contexto de la prueba t, los grados de libertad representan la cantidad de datos que son libres de variar después de tener en cuenta ciertas restricciones impuestas por el estudio.
- c. En el contexto de la prueba F, los grados de libertad se refieren a dos conjuntos de datos independientes que se están comparando en términos de varianza.
- d. En modelos de regresión lineal, los grados de libertad pueden estar asociados con el número de variables independientes y el tamaño total de la muestra.

Técnicas estadísticas para variables categóricas

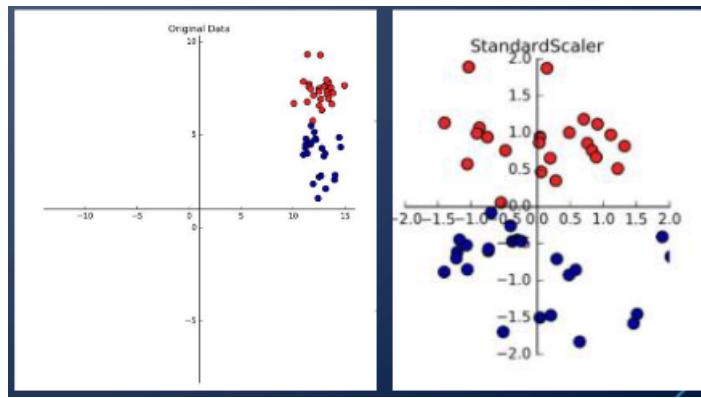
1. Tablas de frecuencia: tabla que muestra la frecuencia o el número de observaciones en cada categoría. Las tablas de frecuencia son útiles para obtener una visión general de la distribución de los datos.
2. Prueba de chi-cuadrado: es una prueba estadística que se utiliza para determinar si existe una relación significativa entre dos variables categóricas. La prueba de chi-cuadrado se basa en comparar las frecuencias observadas con las frecuencias esperadas bajo la hipótesis nula de que no hay relación entre las variables.
3. Regresión logística: es un modelo estadístico que se utiliza para predecir la probabilidad de una respuesta binaria (por ejemplo, sí o no) en función de una o más variables categóricas o continuas

Ahora el siguiente paso de ingeniería de características es evaluar la escala en donde se encuentran los datos. Casi todos los algoritmos funcionan mejor cuando los datos vienen en una escala similar y cercanos a una distribución normal. Para llegar a esta "normalización" podemos evaluar 3 términos:

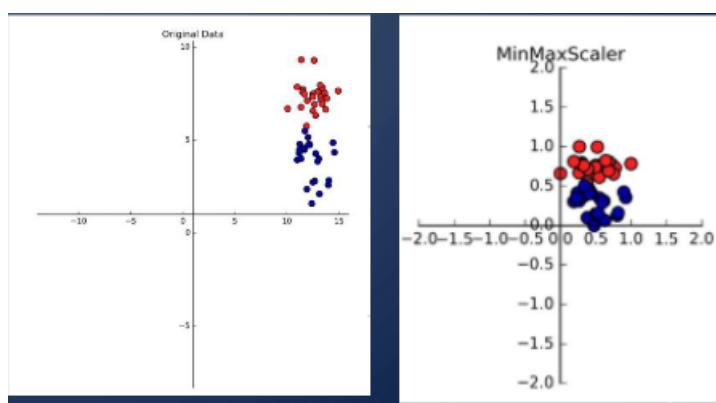
- **Escalada:** significa cambiar el rango de los valores. La forma de la distribución no cambia. Es como construir el mismo modelo, pero más chico.
- **Estandarizar:** significa cambiar los valores de forma tal que el desvío de la distribución estándar de la media sea igual a 1. Devuelve algo que se parece bastante a una distribución normal. A menudo, implica escalada.
- **Normalizar:** puede significar escalar o estandarizar. Se sugiere no usar el término ya que lleva a confusión porque refiere a muchas cosas distintas.

A nivel código tenemos :

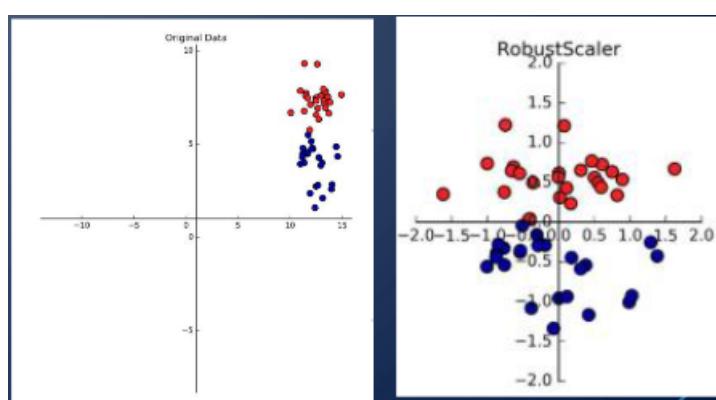
- **StandardScaler:** toma cada característica y le resta la media y divide todos los valores por la desviación estándar. Resulta en una distribución con una desviación estándar igual a 1. Hace que la media de la distribución sea 0 y el 68% de los valores estén entre -1 y 1.
- **StandardScaler** no distorsiona las distancias relativas entre los valores de las características.



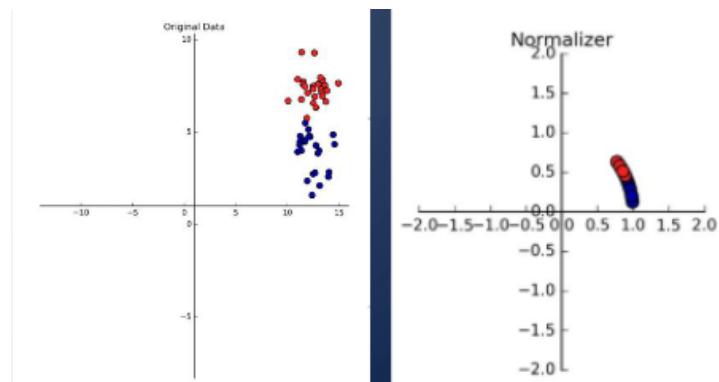
- **MinMaxScaler:** hace que todos los datos estén entre 0 y 1.
- **MinMax Scaler** preserva la forma de la distribución original. No cambia significativamente la información embebida en los datos originales. MinMax Scaler no reduce la importancia de los outliers.
- **MinMax Scaler** es un buen método como para comenzar a probar a menos que se sepa de antemano que se quiere que la característica tenga una distribución normal o que se quiera que los outliers reduzcan su influencia.



- **RobustScaler:** es similar a StandardScaler pero sustrae las medianas y los divide por el rango intercuartil de cada característica (25%-75%). No escala los datos en un intervalo predeterminado como MinMaxScaler, por lo tanto no es necesariamente un Scaler.
- Usarlo si se quiere diluir el efecto de los outliers.



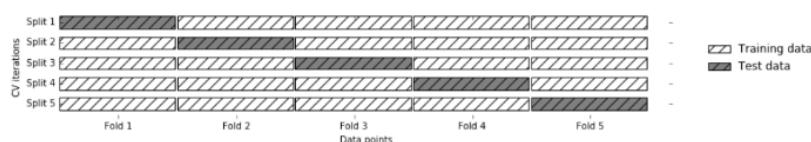
- **Normalizer:** escala cada punto de datos de forma tal que el vector de características tenga una longitud Euclíadiana de longitud 1. Dicho de otra manera, proyecta cada punto en un círculo o esfera de longitud 1. Esta se usa cuando solo importa la dirección (o ángulo) de los datos, no la longitud del vector de características. No suele usarse.



Evaluación del rendimiento del modelo

Para realizar esta evaluación una de las técnicas más usadas es Cross validation:

- Es una técnica estadística utilizada para evaluar el rendimiento de un modelo de aprendizaje automático.
- Método estadístico de evaluación de la performance de la generalización.
- Los datos se partitionan repetitivamente y varios modelos son entrenados.
- Consiste en dividir los datos en varios subconjuntos llamados "folds".
- Entrenar el modelo en algunos de ellos y evaluarlo en los restantes.
- Este proceso se repite varias veces para que todos los folds hayan sido utilizados como conjunto de prueba al menos una vez.
- Yendo mas al detalle tenemos:
 - **División en folds:** Los datos se dividen en varios subconjuntos de tamaño similar llamados **folds**. En el ejemplo de la imagen, los datos se dividen en 5 fold (Fold 1, Fold 2, etc.).
 - **Entrenamiento y evaluación:** En cada iteración (CV Iteration), se selecciona un fold para usarlo como **conjunto de prueba** (Test data) mientras los demás se utilizan como **conjunto de entrenamiento** (Training data). Por ejemplo, en la primera iteración, Fold 1 es el conjunto de prueba, y Fold 2 a Fold 5 son los de entrenamiento.
 - **Repetición del proceso:** Este proceso se repite para que cada fold haya sido utilizado como conjunto de prueba al menos una vez. En la segunda iteración, Fold 2 será el conjunto de prueba, y así sucesivamente hasta que todos los folds hayan sido usados para validar el modelo.
 - **Promedio de resultados:** Al final, los resultados de cada iteración se promedian para dar una estimación más precisa de la capacidad del modelo para generalizar a datos no vistos.



El cross-validation ayuda a obtener una mejor evaluación del modelo, al probarlo varias veces en diferentes subconjuntos de los datos y evitar que dependa demasiado de un solo conjunto de entrenamiento o prueba.

Tambien existen otras formas de validación cruzada:

- **k-fold cross-validation**,
- **leave-one-out cross-validation**,
- **stratified cross-validation**, entre otras.
- La técnica **k-fold cross-validation** es la más utilizada en la práctica.
- En esta técnica, los datos se dividen en k folds, y se entrena el modelo k veces, utilizando cada fold como conjunto de prueba una vez, y los restantes como conjunto de entrenamiento.

Beneficios de Cross Validation:

- Recordar que **train_test_split** realiza un particionamiento aleatorio de los datos.
- Se puede tener suerte y encontrar un data set representativo, pero quizás esto no pase.
- Cuando se usa cross validation, cada muestra va a estar en el set de datos de prueba exactamente una vez. De esta manera, el modelo necesita generalizar bien sobre todas las muestras del dataset para que cross validation devuelva un buen accuracy.
- Tener múltiples splits de los datos proporciona información acerca de qué tan sensible es el modelo a la selección del training set.
- En Cross Validation, se usan los datos más eficientemente.
- Cuando se usa **train_test_Split** se utiliza por lo general el 75% de los datos para training y 25% para test.
- Usando Cross Validation con 5 folds, se usa el 80% de los datos para entrenamiento. Con 10 folds, se usa el 90%.

Desventaja de Cross Validation:

- Costo computacional: como se entrenan k modelos en lugar de un único modelo, va a ser k veces más lento que hacer un split simple de los datos.

Ahora hay que tener en cuenta que necesitamos tambien realizar la validación del modelo entonces. No se puede usar el **test set** para ajustar los parámetros del modelo porque lo estarías "entrenando" con esos datos, lo cual lo haría menos confiable. Para solucionar esto, se introduce el **validation set**, que se usa para ajustar los parámetros, dejando el **test set** exclusivamente para la evaluación final del modelo.

En resumen tenemos tres conjuntos:

- **Training set**: Lo usas para **entrenar** el modelo.
- **Validation set**: Lo usas para **ajustar los hiperparámetros** (configuraciones internas) del modelo.
- **Test set**: Lo usas para **evaluar** qué tan bien funciona el modelo una vez ajustados los parámetros.

Y el proceso queda:

- Primero se divide el conjunto de datos en tres partes: **training**, **validation** y **test**.

- Luego, el modelo se entrena y ajusta usando el training y validation set.
- Finalmente, se reconstruye el modelo combinando el training y validation set, y se evalúa contra el **test set** para ver el rendimiento final.

Lógica Difusa

1. Permite el manejo de la incertidumbre
 - a. Es una rama de la lógica matemática
 - b. Permite el manejo de la incertidumbre
 - c. Permite el manejo de la in precisión en la toma de decisiones
2. Se desprende de la teoría de conjuntos
3. La intersección se la conoce como grado de membresía.
4. Permite que los valores de verdad sean representados como grados de pertenencia a un conjunto borroso, es decir, como valores numéricos en un rango continuo entre 0 y 1

Conjunto difuso

1. Es un conjunto en el que los elementos tienen grados de pertenencia en lugar de una pertenencia binaria (verdadero o falso).
2. La lógica difusa utiliza operadores difusos, como la negación difusa, la conjunción difusa y la disyunción difusa, que operan sobre los grados de pertenencia de los elementos en un conjunto difuso.
3. Se utilizan reglas difusas para modelar el razonamiento basado en la lógica difusa, que son reglas condicionales que relacionan diferentes conjuntos difusos y establecen conclusiones difusas.
4. En la lógica de tradicional no puede existir la incertidumbre
5. En la lógica difusa se manejan grados de incertidumbres
6. Pasos de la lógica difusa
 - a. Fuzzificación
 - b. Reglas difusas
 - c. Defuzzificación

Fuzzificación:

1. Tomar un dato y convertirlo en difuso
2. Hallar el grado de membresía en una serie de conjuntos difusos (rápido, lento, alto, bajo, etc.)

Reglas Difusas:

1. Se construyen en base a las necesidades del requerimiento
2. Se utilizan para conocer el grado de membresía
3. Se relaciona el valor a la membresía

Defuzzificación:

1. Transformar un valor difuso en un valor exacto

Membresía:

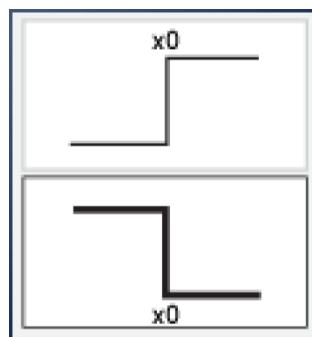
1. Grado de pertenencia o inclusión de un elemento en un conjunto difuso.
2. Se define mediante una función de membresía, que asigna a cada elemento un valor en el rango [0, 1] que representa el grado de pertenencia del elemento a ese conjunto.

Funciones de Membresía:

- Booleana
- Booleana Inversa
- Grado
- Grado Inverso
- Triángulo
- Trapezoide

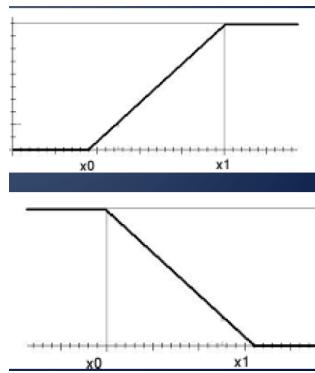
Membresía Booleana / Inversa

1. Se establece con 2 valores "1" o "0"
2. x_0 es el valor que determina si pertenece o no



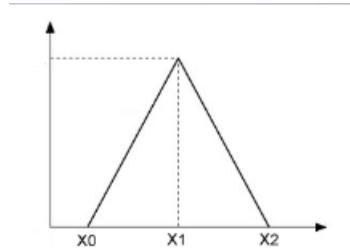
Membresía de Grado / Inversa

1. Cuenta con 2 secciones
2. De x_0 a x_1 es el grado de pertenencia
3. Mayor a x_1 (x_0 en el inverso) es la pertenencia total



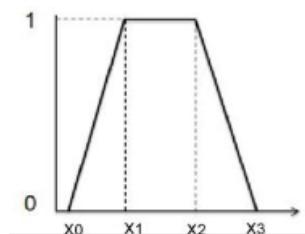
Membresía de Triángulo

1. Cuenta con 2 secciones de pertenencia
2. $x_0 - x_1$
3. $x_1 - x_2$
4. Por fuera no pertenece

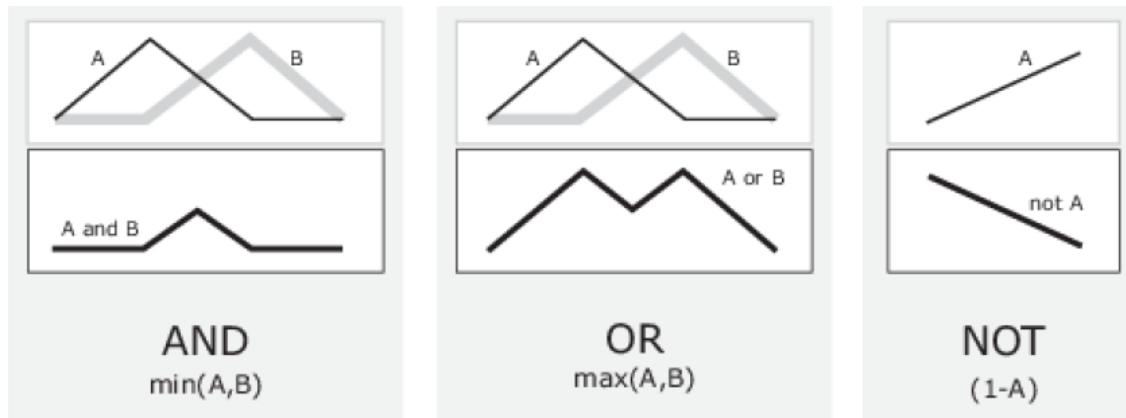


Membresía de Trapezoide

1. Se compone por 3 secciones
2. Graduales $x_0 - x_1, x_2 - x_3$
3. Plena $x_1 - x_2$



Grupos Difusos:



Consiste en tener dos grupos de membresía. En la imagen mostramos 3 criterios de unión de esas membresías. El primero requiere pertenecer a A y B. El segundo con pertenecer a uno u otro ya se es miembro y en el ultimo implica no pertenecer a uno de los grupos.

Problema de Búsqueda

Un **problema de búsqueda** es aquel en el que tienes un conjunto de posibles soluciones y debes explorar sistemáticamente las opciones para encontrar la mejor o una que sea aceptable. Se utilizan comúnmente en inteligencia artificial, ciencias de la computación y optimización.

Estructura de un problema de búsqueda

- Espacio de estados:** Es el conjunto de todos los posibles escenarios o situaciones en las que podrías estar. Cada estado representa un punto o situación en el problema. Por ejemplo, en un laberinto, cada posición es un estado.
- Acciones sucesoras:** Son los movimientos que te permiten pasar de un estado a otro. En el caso del laberinto, estas acciones podrían ser moverte hacia adelante, girar a la izquierda, etc.
- Estado inicial y estado objetivo:** El estado inicial es el punto de partida, y el estado objetivo es lo que quieres lograr. En el ejemplo del laberinto, el estado objetivo sería la salida.
- Solución:** Es una secuencia de acciones que te llevan del estado inicial al estado objetivo, es decir, el conjunto de movimientos necesarios para alcanzar la meta.

Algoritmos de búsqueda

Los **algoritmos de búsqueda** son técnicas usadas para explorar todas las posibles soluciones y encontrar la mejor o una satisfactoria. Estos algoritmos permiten analizar las diferentes rutas dentro del espacio de estados para hallar una secuencia óptima de acciones.

Pasos para resolver un problema de búsqueda

- Formular el objetivo:** Establecer claramente lo que deseas lograr.
- Formular el problema:** Definir los estados y las acciones disponibles para moverte entre ellos.
- Encontrar la solución:** Utilizar un algoritmo para explorar el espacio de estados y encontrar la secuencia de acciones que te lleve desde el estado inicial al estado objetivo.

Ejemplo: Encontrar la mejor ruta en un mapa

1. **Espacio de estados:** Cada lugar posible en el mapa, como tu casa, la esquina de una calle o una plaza.
2. **Acciones sucesoras:** Los movimientos posibles, como caminar, girar, o tomar un transporte. Cada acción puede tener un costo, como el tiempo o la distancia.
3. **Estado inicial:** Es donde empiezas tu búsqueda, por ejemplo, tu casa.
4. **Estado objetivo:** El lugar al que quieras llegar, como una tienda.
5. **Función de acciones con costo:** Algunas rutas pueden ser más rápidas o más largas. El costo puede medirse en distancia o tiempo.
6. **Solución:** La secuencia de acciones que te lleva de tu casa a la tienda de la manera más eficiente (la ruta más corta o rápida).

Espacio de estados

El **espacio de estados** consiste en el estado inicial, las acciones disponibles y el modelo de transición que conecta los estados a través de las acciones. Se puede visualizar como un grafo en el que los nodos son los estados y las aristas son las acciones que te permiten pasar de un estado a otro.

Solución y calidad de la solución

Una **solución** es una secuencia de acciones que te lleva del estado inicial al estado objetivo. La **calidad de la solución** está determinada por el costo del camino, como el tiempo o la distancia recorrida. La **solución óptima** es aquella que minimiza el costo.

El mundo real y la abstracción

El **mundo real** es muy complejo, lleno de detalles irrelevantes que no siempre son útiles para resolver un problema de búsqueda. La **abstracción** es el proceso de simplificar el problema eliminando estos detalles innecesarios, centrándose solo en los aspectos importantes, como los caminos y las acciones clave. Esto facilita el manejo del problema y permite encontrar soluciones más eficientes.

Resumen final

- El **espacio de estados** es como un mapa con diversas opciones de caminos.
- La **solución** es la mejor secuencia de acciones para llegar al destino, y su calidad depende de la función de costo, como distancia o tiempo.
- Para manejar la complejidad del mundo real, aplicamos la **abstracción**, eliminando detalles innecesarios y concentrándonos en los aspectos esenciales para resolver el problema de manera eficiente.

Algoritmo de Árbol de Búsqueda

Este algoritmo consiste en explorar un **espacio de estados** generando nodos sucesores a partir de estados que ya has explorado. Imagina que quieras ir de un punto A a un punto B en un mapa. En este proceso, exploras todas las opciones de rutas posibles como si estuvieras construyendo un árbol donde cada nodo representa un estado o lugar, y cada rama es una posible ruta a seguir.

Conceptos importantes:

- Expansión de nodos:** Cuando exploras un estado (un nodo en el árbol), generas los sucesores, es decir, los nuevos estados a los que podrías moverte.
- Frontera:** Estos son los nodos que aún no has explorado, pero que están "fronterizos" en tu árbol de búsqueda. El objetivo es decidir cuál de estos explorar primero.

Estrategias de búsqueda

- Expansión de planes potenciales:** El algoritmo expande los nodos en la frontera del árbol de búsqueda, donde considera posibles caminos desde el nodo actual.
- Planes parciales:** Se mantiene una lista de posibles planes, que son caminos parciales hacia el objetivo, para decidir cuál es el más prometedor.
- Minimizar la expansión de nodos:** La idea es expandir solo la menor cantidad de nodos necesarios para encontrar una solución eficiente.

Evaluación de estrategias de búsqueda:

Cuando decides cómo expandir los nodos, tienes que evaluar la estrategia basada en estas dimensiones:

- Compleitud:** Si hay una solución, ¿la estrategia siempre la encontrará?
- Complejidad temporal:** ¿Cuántos nodos genera la estrategia antes de encontrar una solución?
- Complejidad espacial:** ¿Cuántos nodos pueden almacenarse en memoria al mismo tiempo?
- Optimalidad:** ¿La estrategia siempre encuentra la solución con el menor costo (por ejemplo, el camino más corto)?

Complejidad en tiempo y espacio:

- b (branching factor):** Es el número máximo de ramas o sucesores que puede tener un nodo en el árbol. Cuantas más opciones tengas desde un nodo, mayor es el valor de **b**.
- d (profundidad de la solución):** Es cuántos niveles del árbol de búsqueda debes recorrer para encontrar la solución más barata o con menor costo.
- m (profundidad máxima):** Es la máxima profundidad que el árbol puede alcanzar. En algunos casos, puede ser infinito si no se limita el espacio de estados.

Ejemplo simple:

En uno de los diagramas aparece un grafo sencillo con 4 nodos (representando estados o lugares), y una de las preguntas es **¿qué tamaño tiene el árbol de búsqueda?**. La respuesta muestra que, dependiendo de las conexiones y la cantidad de repeticiones, el árbol puede crecer enormemente y alcanzar un tamaño infinito si no limitas la búsqueda.

Resumen:

- El **algoritmo de árbol de búsqueda** explora todas las opciones posibles, generando sucesores desde nodos ya explorados.
- La **estrategia de búsqueda** se basa en decidir cuál nodo expandir para encontrar la mejor solución, balanceando entre explorar muchos nodos o hacerlo de forma eficiente.
- Se mide la **complejidad** en términos de tiempo (cuántos nodos se generan) y espacio (cuántos nodos se pueden almacenar), así como su **optimalidad**, es decir, si encuentra siempre el mejor

camino.

En resumen, el algoritmo de árbol de búsqueda es una técnica poderosa pero costosa que requiere estrategias eficientes para manejar el crecimiento del árbol y encontrar la mejor solución sin consumir demasiados recursos.

Estrategias de Busqueda no informadas

Estas estrategias no tienen conocimiento adicional sobre el problema, solo utilizan la información que tienen sobre el espacio de estados y las transiciones entre ellos. Son técnicas genéricas que exploran sistemáticamente los posibles caminos:

1. **Búsqueda Breadth-First (primero en ancho):** Explora todos los nodos al mismo nivel antes de pasar al siguiente nivel.
2. **Búsqueda Uniform-Cost (costo uniforme):** Prioriza la expansión de los nodos que tengan el menor costo acumulado.
3. **Búsqueda Depth-First (primero en profundidad):** Explora primero los caminos hasta el fondo (profundidad máxima) antes de retroceder y explorar otros caminos.
4. **Búsqueda Depth-Limited (profundidad limitada):** Es similar a la búsqueda en profundidad, pero se le da un límite de profundidad a explorar, evitando que se quede atascada en ciclos o ramas muy largas.

Estrategias de búsqueda informadas

Las estrategias informadas tienen conocimiento adicional del problema y lo usan para dirigir la búsqueda hacia una solución más eficiente:

1. **Búsqueda Greedy Best-First (avariciosa mejor-primero):** Elige expandir el nodo que parece más cercano al objetivo, usando una función heurística.
2. **Búsqueda A*:** Combina las estrategias de búsqueda de costo uniforme y heurística, buscando el camino con el costo total más bajo, considerando tanto el costo desde el inicio como una estimación del costo restante hasta el objetivo.

Detalle de estrategia de Busqueda no informadas

Búsqueda en profundidad (Depth-First Search - DFS)

Este algoritmo explora primero los caminos hasta la profundidad máxima antes de retroceder y explorar otras ramas.

- **Cómo funciona:** Parte desde un nodo inicial, se mueve lo más profundo posible por una rama y, si llega a un nodo sin sucesores o alcanza una profundidad máxima, retrocede y explora la siguiente rama disponible.
- **Implementación:** Utiliza una estructura de datos de tipo **LIFO (Last In, First Out)**, es decir, una pila. Los nodos sucesores se agregan al principio de la frontera, lo que garantiza que se profundice primero antes de retroceder.

Evaluación de la búsqueda en profundidad

El rendimiento del algoritmo DFS se puede evaluar en varios aspectos clave:

1. **¿Es completo?**: No. Puede fallar en encontrar una solución si el espacio de estados tiene profundidad infinita o si hay ciclos (repite estados una y otra vez).
2. **¿Tiempo?**: Tiene una complejidad $O(b^m)$, donde b es el número de ramas por nodo y m es la profundidad máxima. Esto puede ser ineficiente si la profundidad del espacio es mucho mayor que la de la solución.
3. **¿Espacio?**: Es lineal, con una complejidad de $O(bm)$, ya que solo se almacenan los nodos en la rama actual.
4. **¿Óptimo?**: No, ya que no garantiza encontrar la solución más corta ni la de menor costo.

Búsqueda en profundidad limitada

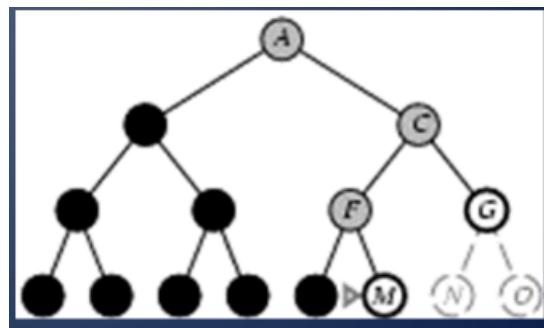
Es una versión mejorada del DFS en la que se establece un límite de profundidad. Esto evita que el algoritmo quede atrapado en ciclos infinitos o explore demasiado profundamente sin encontrar una solución.

- **Beneficios**: Resuelve el problema de los espacios de búsqueda infinitos.
- **Limitaciones**: Puede dejar de ser completo si la solución se encuentra más allá de la profundidad límite.

Conclusión sobre DFS:

La búsqueda en profundidad es útil en ciertos casos por su bajo consumo de memoria, pero no es ideal cuando buscas una solución óptima o en casos de espacios de búsqueda infinitos o con ciclos. El **DFS con profundidad limitada** ayuda a mitigar algunos de estos problemas, pero puede requerir ajustes según el problema específico.

En grafico seria



Voy hasta el final y recien paso al otro

Búsqueda Breadth-First Search (BFS)

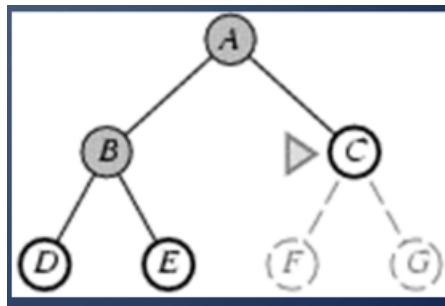
- **Expansión de nodos**: Este algoritmo expande los nodos en el orden en que fueron descubiertos, es decir, primero expande los nodos menos profundos.
- **Implementación**: La frontera se maneja como una cola FIFO (primero en entrar, primero en salir). Los nuevos sucesores de un nodo se agregan al final de la cola.
- **Propiedades**:
 - **Completo**: Sí, si el factor de ramificación b es finito.
 - **Tiempo**: El tiempo de ejecución es proporcional a $O(bd+1)$, donde d es la profundidad del nodo objetivo y b es el número de hijos por nodo.

$O(bd+1)O(b^{d+1})$

- **Espacio:** El algoritmo requiere $O(bd+1)$ de espacio, ya que necesita almacenar todos los nodos generados en memoria.

$O(bd+1)O(b^{d+1})$

- **Óptimo:** Sí, si el costo por paso es constante, BFS encontrará la solución más corta.



A medida que voy agregando voy analizando

Búsqueda de Costo Uniforme (UCS)

- **Expansión por costo:** UCS expande el nodo con el menor costo acumulado, lo que la convierte en óptima cuando los costos de los pasos no son uniformes.
- **Implementación:** Se utiliza una cola de prioridad que selecciona los nodos de acuerdo con el costo del camino $g(n)$.

$g(n)g(n)$

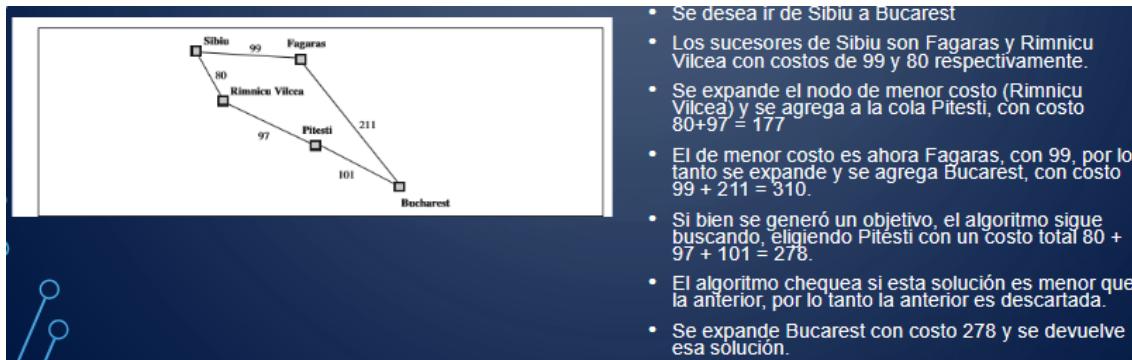
- **Diferencias con BFS:**

1. La prueba de objetivo se aplica cuando el nodo es seleccionado para su expansión, no cuando se genera por primera vez.
2. El algoritmo puede actualizar un nodo si encuentra un mejor camino al mismo.

- **Propiedades:**

- **Completo:** Sí, siempre y cuando el costo de cada paso sea positivo.
- **Tiempo:** El tiempo depende de la cantidad de nodos con un costo acumulado menor o igual al costo de la solución óptima.
- **Espacio:** Requiere almacenar todos los nodos generados con costos menores o iguales a la solución óptima.
- **Óptimo:** Sí, UCS es óptima siempre que los costos de los arcos sean positivos.

Graficamente seria



Detalle de estrategia de Busqueda informadas

Estas búsquedas utilizan información adicional (heurísticas) para tomar decisiones más eficientes que las estrategias no informadas.

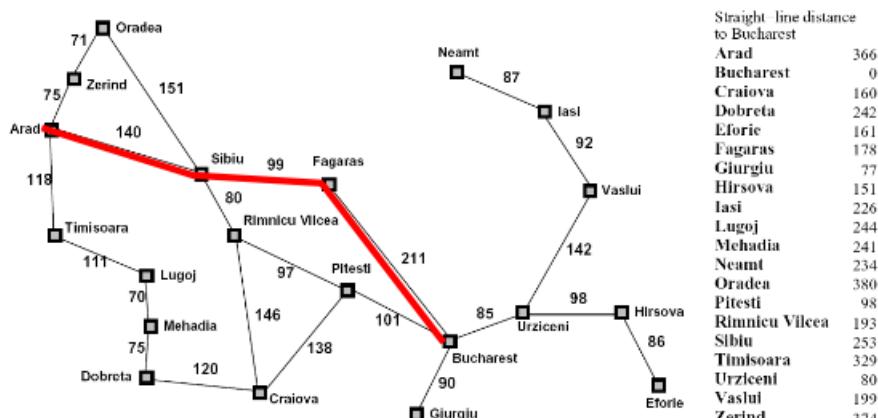
- **Función de Evaluación:** Se basa en $f(n)$, que puede incluir un componente heurístico $h(n)$, que es una estimación del costo desde el nodo actual hasta el objetivo.
- **Heurística:** $h(n)$ estima el camino más corto desde el nodo actual hasta el estado objetivo. Esto se ilustra con el ejemplo de un mapa de Rumanía, donde $h(n)$ es la distancia en línea recta entre las ciudades (como de Arad a Bucarest).

Búsqueda Voraz

Esta estrategia selecciona siempre el nodo que parece estar más cerca del objetivo, usando solo la función heurística

- **Selección del mejor nodo disponible:** En cada etapa, el algoritmo selecciona la mejor opción basándose en la evaluación de la heurística, eligiendo el nodo con el menor valor de $h(n)$.
- **Ejemplo:** Se utiliza el mismo mapa de Rumanía para ilustrar cómo se expande el nodo con la menor distancia en línea recta a Bucarest. Comienza desde Arad y expande los nodos basados en $h(n)$, por ejemplo, Sibiu (con una distancia de 253), Timisoara (329), Zerind (374), etc.
- **Búsqueda voraz:** Solo considera $h(n)$, lo que puede llevar a caminos subóptimos, ya que no toma en cuenta el costo acumulado, solo la distancia estimada al objetivo.

Ejemplo grafico



Busca el camino mas corto basandose en las iteraciones de la funcion heuristica

Búsqueda A*

La **búsqueda A*** es un algoritmo que te ayuda a encontrar el camino más corto o la mejor ruta entre dos puntos. Lo hace siendo muy eficiente, porque combina lo mejor de dos mundos:

1. La **búsqueda en anchura (BFS)**, que busca expandiendo los nodos de manera uniforme.
2. La **búsqueda en profundidad (DFS)**, que trata de ir lo más profundo posible.

Pero lo especial de A* es que usa una pista o estimación (heurística) para ayudarte a tomar mejores decisiones sobre qué camino seguir.

¿Cómo toma decisiones la búsqueda A*?

Para decidir hacia dónde ir, A* utiliza una fórmula que tiene en cuenta dos cosas importantes:

1. **g(n)g(n)g(n)**: El costo que ya has pagado para llegar al punto actual. Es como sumar cuántos recursos o pasos has usado desde el inicio hasta donde estás.
2. **h(n)h(n)h(n)**: Una estimación o "pista" de cuánto te falta para llegar a la meta. Es lo que te dice la heurística: "Mira, desde aquí parece que la meta está a tal distancia en línea recta".

La fórmula que utiliza A* es:

$$f(n) = g(n) + h(n) \quad f(n) = g(n) + h(n)$$

Esta fórmula te dice cuál es el costo total estimado de seguir por un determinado camino. Combina lo que ya has recorrido ($g(n)g(n)g(n)$) y lo que crees que te falta ($h(n)h(n)h(n)$).

¿Qué lo hace diferente?

Lo que hace diferente a A* de otros algoritmos como la **búsqueda de costo uniforme (UCS)** es que A* no solo considera lo que has recorrido ($g(n)g(n)g(n)$), sino también lo que te falta ($h(n)h(n)h(n)$). Por ejemplo:

- **UCS**: Solo le importa cuánto has recorrido.
- **A***: Le importa cuánto has recorrido **y** cuánto te falta según su estimación.

Esto lo hace más eficiente, ya que evita expandir caminos que parecen buenos pero en realidad no lo son.

Ejemplo con el mapa

Supongamos que estamos en **Arad** y queremos llegar a **Bucarest** (como se muestra en el mapa de Rumanía).

1. **Primera elección (Arad)**: Estás en Arad, y A* calcula que el costo desde Arad a Bucarest es de 366, según la distancia en línea recta.
2. **Expansión de nodos**: A* evalúa varias rutas posibles. Por ejemplo, desde Arad puedes ir a **Sibiu**, **Timisoara** o **Zerind**. Utiliza la fórmula para decidir cuál ruta parece mejor.
 - Para **Sibiu**, el costo estimado es = 393 (costo+linea recta)
 - Para **Timisoara**, el costo estimado = 447 (costo+linea recta)
 - Para **Zerind**, el costo estimado = 449 (costo+linea recta)A* escogerá seguir por **Sibiu** porque parece el camino más prometedor
3. **Siguientes pasos**: A* sigue expandiendo los caminos, comparando las rutas posibles

Conclusión

La búsqueda A* es un algoritmo inteligente porque combina lo que ya has recorrido y una pista de cuánto te falta para llegar a la meta. Esto lo hace más eficiente que otros métodos como BFS o UCS, y lo convierte en una de las mejores formas de encontrar el camino más corto o el mejor en un mapa o laberinto.

Juegos en Inteligencia Artificial

- **Ambientes multiagente:** Imagina que estás jugando un videojuego, pero no juegas solo, hay otros jugadores (como en ajedrez o Ta-Te-Ti). Aquí cada jugador tiene su propio objetivo, pero compiten entre sí.
- **Competencia:** En los juegos de IA, los jugadores compiten. Uno intenta ganar y el otro también, por lo que sus intereses están en conflicto. Es lo que pasa en el ajedrez: si tú ganas, tu oponente pierde.
- **Suma cero:** Esto significa que, al final del juego, si tú ganas (+1 punto), el otro pierde (-1 punto). El total siempre será cero (ganancia de uno = pérdida del otro).

2. Problema de Búsqueda en los Juegos

Imagina que el juego es un problema que quieras resolver, y tiene algunos elementos clave:

- **Estado inicial:** Cómo empiezas el juego, como en Ta-Te-Ti, el tablero vacío sería el estado inicial.
- **Jugador:** Quién tiene el turno. En el ajedrez, o Ta-Te-Ti, esto cambia entre los dos jugadores.
- **Acciones:** Los movimientos que puedes hacer desde la posición actual. En Ta-Te-Ti, puedes poner tu X en cualquier casilla vacía.
- **Resultado:** Qué pasa cuando haces un movimiento. Si pones una X en un lugar, el tablero cambia.
- **Prueba de término:** Te dice si el juego ha terminado (alguien ganó o ya no hay movimientos).
- **Puntuación (utility):** El puntaje que obtiene cada jugador al final. En ajedrez, ganar es +1, perder es -1, y empatar es 0.

3. Árbol de Decisiones de Juegos

- Los juegos como el ajedrez o Ta-Te-Ti pueden verse como un árbol de decisiones. Cada vez que haces un movimiento, eliges una rama del árbol, y el oponente elige otra.
- **MAX y MIN:** Uno de los jugadores trata de **maximizar** sus puntos (quiere ganar), mientras que el otro trata de **minimizar** sus puntos (quiere que tú pierdas). Este es el esquema del juego de Minimax.

4. Algoritmo Minimax

Este es el corazón de cómo juegan las máquinas a juegos como el ajedrez.

- **Objetivo:** El algoritmo Minimax busca la mejor jugada para el jugador que quiere ganar (jugador maximizador), asumiendo que su oponente también va a jugar lo mejor posible para hacerlo perder.

- **Búsqueda de todas las jugadas posibles:** Minimax explora todas las jugadas posibles que puedes hacer y las que puede hacer tu oponente. Evalúa todas las consecuencias de esas jugadas y elige la mejor para ti.

¿Cómo funciona?

- Construyes un árbol de decisiones con todas las jugadas posibles.
- En los niveles del árbol, los jugadores se turnan para hacer jugadas.
 - El jugador que maximiza busca el mayor puntaje.
 - El jugador que minimiza busca reducir ese puntaje lo más posible.

Ejemplo:

En Ta-Te-Ti, si es tu turno, quieres poner la X en un lugar que maximice tus posibilidades de ganar. Pero también piensas en lo que tu oponente va a hacer, y quieres minimizar sus posibilidades.

5. Limitaciones del Minimax

- Aunque es una buena estrategia, puede ser **computacionalmente costosa**. ¿Por qué? Porque tienes que explorar todas las jugadas posibles, lo cual puede ser impráctico en juegos complejos como el ajedrez, que tiene demasiadas combinaciones de jugadas.

6. Optimización de Minimax

- Para hacer que Minimax funcione mejor, existen **técnicas de optimización**. Una muy común es la **poda alfa-beta**, que ayuda a reducir el número de jugadas que necesitas evaluar, haciendo el proceso más rápido.

Resumen:

- **Juegos en IA:** Compites con otro jugador y ambos tienen objetivos opuestos (ganar/perder).
- **Minimax:** Busca la mejor jugada para maximizar tu puntaje, mientras considera que el oponente también va a jugar de manera óptima para minimizar tu puntaje.
- **Árbol de decisiones:** Representa todas las jugadas posibles y cómo pueden terminar.
- **Optimización:** Minimax es poderoso, pero puede ser lento si no se optimiza con técnicas como la poda alfa-beta.

Sistemas Expertos

Un **sistema experto** es un programa de computadora que trata de imitar cómo piensa un experto humano para resolver un problema específico. Estos sistemas utilizan **conocimiento** (información) y **reglas** (si pasa esto, entonces haz esto) para tomar decisiones o dar recomendaciones.

Algoritmos en Sistemas Expertos

Los sistemas expertos utilizan diferentes **algoritmos** para llegar a conclusiones o resolver problemas. Aquí algunos ejemplos:

a. Algoritmo de Cadena de Markov

- Este algoritmo se usa para predecir qué va a pasar a continuación, basándose en lo que ha pasado antes. Como si dijeras: "Si ha llovido tres días seguidos, es probable que también llueva mañana".
- Explora **combinaciones posibles** de eventos o estados para encontrar la mejor solución.

b. Algoritmo de Inferencia hacia Delante

- Funciona aplicando reglas lógicas de forma directa, es decir, va de **lo que ya sabes a nuevas conclusiones**.
 - Ejemplo: Si sabes que "todos los pájaros tienen alas" y que "los pingüinos son pájaros", puedes deducir que "los pingüinos tienen alas".
- Se basa en "si esto es cierto, entonces esto también lo es".

c. Algoritmo de Resolución de Unificación

- **Unifica** dos fórmulas lógicas o ecuaciones en una sola.
 - Ejemplo: Si tienes dos fórmulas matemáticas o lógicas, este algoritmo las combina para simplificar el problema.

d. Algoritmo de Retroceso

- Este algoritmo trata de encontrar una solución, pero si en algún punto se da cuenta de que el camino que eligió no sirve, **retrocede** y prueba otro camino. Es como cuando intentas resolver un laberinto y si llegas a un callejón sin salida, vuelves atrás y pruebas otra ruta.

General Problem Solver (GPS)

El **GPS** es un tipo de sistema experto diseñado para resolver problemas usando diferentes algoritmos. Aquí están los algoritmos principales que usa:

a. Algoritmo de Búsqueda en Profundidad

- **¿Qué hace?**: Este algoritmo explora cada camino de una posible solución hasta el final antes de probar otro camino.
- **Ejemplo**: Imagina que estás buscando una salida en un laberinto. Sigues un camino hasta que llegas al final (o una pared), y si no encuentras la salida, retrocedes y pruebas otro camino.

b. Algoritmo de Búsqueda de Amplitud

- **¿Qué hace?**: En lugar de ir hasta el final de cada camino, este algoritmo revisa todas las opciones al mismo nivel antes de ir más profundo.
- **Ejemplo**: En el laberinto, primero miras todas las salidas cercanas antes de explorar caminos más lejanos.

2. Reglas de Producción

- **¿Qué son?**: Las reglas de producción son declaraciones del tipo "si pasa X, entonces haz Y".
- **Ejemplo**: Si el sistema sabe que "si tienes alas, puedes volar", y "los aviones tienen alas", entonces deduce que "los aviones pueden volar". Así es como las reglas de producción ayudan a tomar decisiones lógicas.

3. Perceptrón

- **¿Qué es?**: El **perceptrón** es un modelo de inteligencia artificial básico que fue diseñado para aprender y reconocer patrones, como imágenes o sonidos.
- **¿Cómo funciona?**: Aprende a clasificar información basándose en ejemplos previos. Por ejemplo, si ve muchas imágenes de gatos y perros, puede aprender a distinguir entre ellos

Luego de varios desarrollos a lo largo del tiempo se incorporaron elementos como:

- a. Base de conocimiento, data sobre lo que intenta resolver
- b. factor de certeza, cuantifica la probabilidad de que una respuesta sea verdadera
- c. explicación de razonamiento, como llegó a la conclusión

Componentes de un sistema experto (pregunta de parcial)

- **Base de Conocimiento:**

Es el lugar donde se almacena toda la información que el sistema necesita para resolver problemas. Es como la "biblioteca" del sistema, donde tiene guardado todo el conocimiento relevante.

- **Conocimiento declarativo**: Este tipo de conocimiento es información sobre **objetos, hechos y situaciones** específicas.
- **Conocimiento procesal**: Son las **acciones** que el sistema debe seguir. Se usan para crear las reglas que guiarán el comportamiento del sistema.

- **Motor de Inferencia:**

Este es el "cerebro" del sistema experto. Es el encargado de **tomar decisiones** usando las reglas de la base de conocimiento. Decide **qué reglas aplicar y cuándo hacerlo**. También evalúa si ha encontrado la solución correcta.

- **Interfaz de Usuario:**

Es la parte del sistema con la que interactúan los usuarios. Permite que el usuario haga preguntas y reciba respuestas del sistema de manera sencilla. Con las siguientes características

- **Utilidad**: El sistema experto debe resolver un problema específico, es decir, debe cumplir con una **necesidad concreta**.
- **Usabilidad**: Debe ser fácil de usar para que las personas puedan interactuar con él sin dificultad.
- **Educativo**: Un sistema experto también puede ayudar a los usuarios a aprender, incrementando su conocimiento.
- **Explicativo**: Un buen sistema experto no solo te da una respuesta, sino que también te explica **cómo llegó a esa solución**.
- **Responsivo**: Responde preguntas y dudas del usuario.
- **Aprendizaje**: Algunos sistemas expertos pueden aprender de la experiencia y mejorar con el tiempo.
- **Modificable**: El sistema debe ser capaz de cambiar o actualizarse, como modificar su base de conocimientos o reglas para adaptarse a nuevas situaciones.

Haciendo una radiografia del sistema experto tambien intervienendos elementos que son la memoria:

1. Tipos de Memoria en un Sistema Experto

Un sistema experto tiene dos tipos de memoria principales:

- **Memoria de trabajo:**

- **¿Qué hace?**: Guarda temporalmente información relevante para el proceso de razonamiento. Es como un "bloc de notas" donde se apuntan datos necesarios mientras el sistema toma decisiones.
- **Ejemplo**: Incluye hechos y resultados provisionales que se usan mientras el sistema decide qué hacer a continuación.

- **Memoria de reglas:**

- **¿Qué hace?**: Guarda las **reglas** que el sistema utiliza para tomar decisiones. Estas reglas siguen el formato "**si ocurre esto, haz esto**".
- **Ejemplo**: Las reglas se aplican en función de los datos guardados en la memoria de trabajo.

2. Las Inferencias

El proceso de **inferencia** es cómo el sistema aplica las reglas para tomar decisiones. Existen dos tipos principales:

- **Encadenamiento progresivo (Forward chaining):**

- Este método va **hacia adelante**, aplicando reglas en función de los hechos actuales. Busca llegar a un resultado aplicando reglas de manera secuencial.
- **Etapas:**
 1. Recorrer todas las reglas y ver cuáles se pueden aplicar.
 2. Seleccionar qué regla aplicar.
 3. Aplicar la regla y modificar la memoria de trabajo.

- **Encadenamiento regresivo (Backward chaining):**

- Este método va **hacia atrás**. Parte de un resultado deseado y busca qué reglas deben aplicarse para llegar a ese resultado.
- Es más complejo, pero es más eficiente en ciertos problemas.

3. Pasos para Crear un Sistema Experto

1. **Identificar el problema**: Decide qué problema quieres que el sistema experto resuelva, ya sea algo específico como un diagnóstico médico o la reparación de máquinas.
2. **Adquirir el conocimiento**: Reúne el conocimiento necesario de expertos en el área y de otras fuentes confiables.
3. **Formalizar el conocimiento**: Organiza la información de manera que el sistema pueda utilizarla. Esto implica definir las **reglas de inferencia** y los **hechos** que el sistema usará.
4. **Implementar la base de conocimientos**: Usa lenguajes de programación como Prolog, Lisp o Python para introducir la información en el sistema.

5. **Desarrollar el motor de inferencia:** Este es el "cerebro" del sistema. Se encarga de aplicar las reglas para tomar decisiones y resolver problemas.
6. **Probar el sistema:** Haz pruebas para asegurarte de que el sistema funcione correctamente y resuelva el problema para el cual fue diseñado.
7. **Refinar y mejorar:** Con el uso, pueden surgir nuevas necesidades o mejoras que deban hacerse, como ajustar las reglas o actualizar el conocimiento.