



Inteligencia Artificial

Introducción

Nos llamamos *Homo sapiens*, el hombre sabio, porque nuestra inteligencia es muy importante para nosotros. Durante miles de años, hemos tratado de comprender cómo pensamos y actuamos, es decir, cómo nuestro cerebro, un mero puñado de materia, puede percibir, comprender, predecir y manipular un mundo mucho más grande y complicado que él mismo. El campo de la inteligencia artificial, o AI, se ocupa no solo de comprender, sino también de construir entidades inteligentes: máquinas que pueden calcular cómo actuar de manera efectiva y segura en una amplia variedad de situaciones novedosas.

¿Qué es la IA?

En los últimos años, inteligencia artificial (IA) ha sido objeto de una inflación mediática intensa. Machine learning, deep learning e inteligencia artificial vienen en incontables artículos, ya sea en artículos académicos, científicos o en noticias mediáticas. (*Deep learning with Python - Francois Chollet*).

El futuro que ya está aquí nos ofrece chat-bots, autos que se manejan solos, asistentes virtuales, etc. Esto seguramente promete en su furor de aplicación, una mejor calidad de vida, reducir costos y riesgos en los negocios, optimizar el uso de recursos **pero** con el riesgo de que cierto trabajos que antes eran desempeñado por

profesionales humanos son y serán reemplazados por máquinas inteligentes que harán tareas mucho más complejas y a una gran velocidad gracias a las tecnologías descentralizadas y el big data.

La inteligencia artificial nació en 1950 de la mano de los pioneros en el campo de la ciencia de la computación, cuando estos empezaron a preguntarse si las computadoras podían “pensar”. La definición más concisa de IA podría ser: **el esfuerzo para automatizar tareas intelectuales que normalmente desempeña un humano.**

Alan Turing (1950) propuso el **test de turing**, como pensamiento experimental para responder a la pregunta de si las máquinas pueden pensar?. Una computadora pasa la prueba si un interrogador humano, después de plantear algunas preguntas escritas, no puede saber si las respuestas escritas provienen de una persona o de una computadora.

Sin embargo este test no demostraría que es “inteligente” Otros investigadores han propuesto el `total Turing Test`, que requiere interacción con objetos y personas en el mundo real. Para pasar el *total Turing test*, un robot tiene que tener:

- **computer vision** y **speech recognition** para percibir el mundo.
- **robotics** para manipular objetos y moverse por sí mismo.

Así es, la IA es un campo general que engloba Machine Learning y Deep Learning, pero también incluye muchos más enfoques que no tienen nada que ver con un *aprendizaje*.

Por ejemplo en el enfoque de **Symbolic AI**, que fue el paradigma dominante desde 1950 hasta 1980, muchos expertos pensaban que una inteligencia artificial al nivel humano se podría lograr a través de *reglas específicas* largas e intrincadas para manipular conocimiento. Así era con los primeros programas de ajedrez, que hardcodeaban reglas por programadores (no Machine Learning).

El anterior enfoque era bueno para problemas bien definidos, donde las reglas hardcodeadas funcionaban bien, **pero** se volvió intratable establecer reglas para complejos y difusos problemas. Estos eran: automatizar el trabajo de rutina, comprender el habla, las imágenes, hacer diagnósticos en medicina y respaldar la investigación científica básica.

En resumen: En los primeros días de la inteligencia artificial, el campo abordaba y resolvía rápidamente problemas que son intelectualmente difíciles para los seres humanos pero relativamente sencillos para computadoras: problemas que pueden describirse mediante una lista de reglas matemáticas formales. El verdadero desafío

para la inteligencia artificial resultó ser resolver las tareas que son fáciles de realizar para las personas pero difíciles de describir formalmente, problemas que resolvemos intuitivamente, que se sienten automáticos, como reconocer palabras habladas o rostros en imágenes. (*Deep Learning Book - Yoshua Bengio*)

Enfoques de IA

Thinking humanly: The cognitive modeling approach

Thinking like a person. Para afirmar esto debemos conocer como los humanos piensan. Podemos aprender sobre el pensamiento humano de tres formas:

- **introspección:** tratando de atrapar nuestros propios pensamiento mientras pasan.
- **experimentos psicológicos:** observando una persona en acción.
- **imágenes cerebrales:** observando el cerebro en acción.

Con esa información sobre la mente y el cerebro, es posible expresar la teoría como un programa de computadora. Si la entrada/salida de una programa tiene comportamientos similares a los humanos, queda en evidencia que dichos mecanismos pueden ocurrir en humanos.

Esto era estudiado por el campo de la **cognitive science**, que buscaba unificar experimentos psicológicos sobre la mente humana con modelos computacionales de la IA.

Thinking rationally: The laws of thought approach

“Socrates is a man; all men are mortal; therefore, Socrates is mortal”. Aristóteles (filosofo griego) fue uno de los primeros en codificar el *pensamiento correcto* (right thinking en inglés). El **silogismo** provee patrones para estructurar argumentos que siempre alcanza conclusiones correctas cuando se parte de premisas correctas.

Lógica.

En 1965, los programas podían, resolver *cualquier* problema mediante notación lógica. La tradición logística espera estos programas para crear sistemas inteligentes. El problema es que se requiere conocimiento certero sobre el mundo, lo cual resulta *utópico*. La teoría de la **probabilidad** viene a solventar las partes faltantes, permitiendo un razonamiento rigurosos con información no certera. Esto permite es crear un *modelo comprensivo de pensamiento racional* que permite

desde obtener información cruda perceptual, hasta comprender como funciona el mundo y predecir el futuro. Pero no genera un comportamiento inteligente.

Acting rationally: The rational agent approach

Un *agente* es algo que actúa. Todos los programas de computadora hacen algo, pero los agentes computacionales hacen más: operan de forma autónoma, perciben su ambiente, persisten largos periodos de tiempo, se adaptan al cambio, crean y persiguen objetivos.

Un **agente racional** es aquel que actúa para alcanzar el mejor resultado o el mejor esperado resultado.

Así como en el anterior enfoque se hacía énfasis en una correcta inferencia, un *agente* también tiene que hacer inferencias correctas para poder actuar *racionalmente* eligiendo la mejor acción para ello. Aunque hay formas de actuar razonablemente sin involucrar una inferencia. Por ejemplo: retroceder ante una estufa caliente es un acto reflejo más exitoso que una acción lenta decidida por una deliberación mental cuidadosa.

El enfoque de agente racional para la IA tiene dos ventajas sobre los otros enfoques. Primero, es más general que el enfoque de las “leyes del pensamiento” porque la inferencia correcta es solo uno de varios mecanismos posibles para lograr la razonabilidad. En segundo lugar, es más susceptible al desarrollo científico. El estándar de razonabilidad está matemáticamente bien definido y es completamente general. A menudo podemos trabajar a partir de esta especificación para derivar diseños de agentes que probablemente lo logren, algo que es en gran medida imposible si el objetivo es imitar el comportamiento humano o los procesos de pensamiento.

Más tarde, los métodos basados en la teoría de la probabilidad y el aprendizaje automático permitieron la creación de agentes que podían tomar decisiones bajo incertidumbre para lograr el mejor resultado esperado.

Acting humanly: The Turing test approach

Test de turing  (explicado anteriormente).

¿Qué es: Machine Learning?

Por lo general Machine Learning se entiende como una sub-área de la Inteligencia Artificial, en donde la computadora *aprende* a resolver una determinada tarea. Pero, ni la inteligencia artificial es *inteligente* ni las maquinas *aprenden* en el sentido de la palabra [4].

Es mucho más apropiado dar un fundamento desde la aplicación de la ciencia de datos donde Machine Learning resulta:

Construir modelos de datos

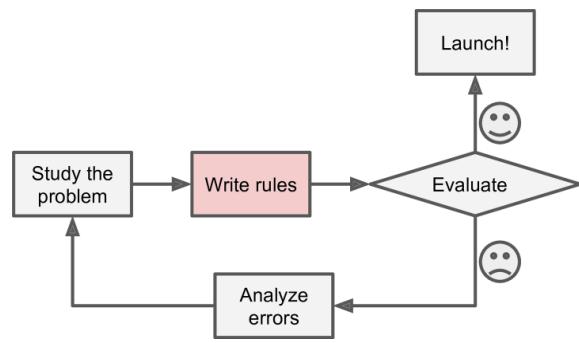
Es decir, construir modelos matemáticos que sirvan para comprender los datos, y mediante la observancia de los datos, aprender a encontrar una estructura que permita realizar predicciones en datos no observados [1].

El modelo matemático es importante porque es el camino por el cual se puede lograr un entendimiento de datos que a simple vista son desordenados, sin un patrón en concreto, con ruido y datos faltantes. El modelo es quién realizará la relación entre los datos de entrada y de salida para que se pueda responder a preguntas cómo son: ¿Cómo puedo tener una vida saludable?, ¿Vale la pena invertir en determinado negocio o divisas?, ¿Cuánto cuesta poner otra planta de producción?, entre otras preguntas que hacen necesario este tipo de desarrollos [2].

Importancia

La importancia radica en la posibilidad de que una determinada tarea (por ejemplo: clasificar mails en spam/no-spam) sea de manera "automática", es decir sin necesidad de escribir el conjunto de técnicas, algoritmos y lógica en código, sino que a través de un proceso matemático encontrar una función que al recibir una entrada como variable independiente (por ejemplo el texto de un mail) devuelva una variable dependiente (spam/no spam).

En el caso clásico de un programa para clasificar mails como *spam* y *no-spam* se debería considerar palabras como: *te ganaste 90000 USD, gratis, fortuna, elegido*; todas aquellas que sean indicadores de un posible mail spam. Después, habría que escribir un algoritmo que detecte dichas palabras en cada mail, etiquetándolas



(clasificación a priori) respectivamente en spam y no-spam para cada caso. Y así probarlo, ver los errores, arreglar bugs y volver a probar. Esta iteración repetirla hasta que se logre una cierta satisfacción con el algoritmo [3].

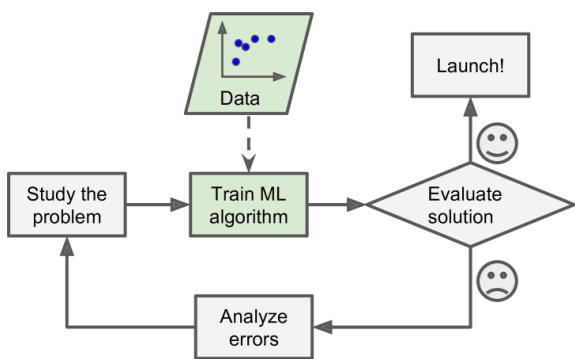
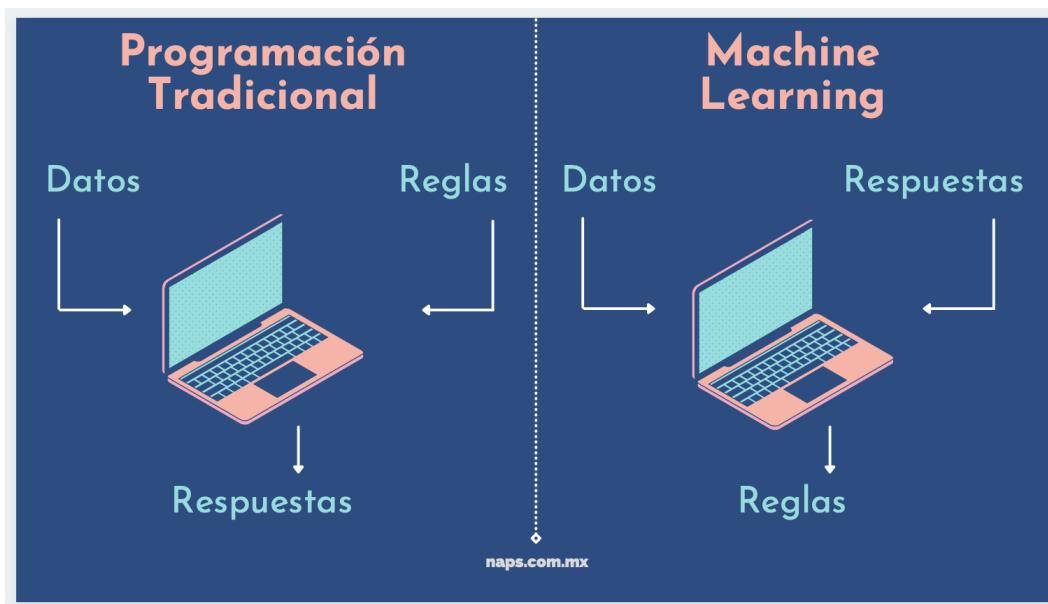


Figura 2: extraída de [3] que representa el enfoque con Machine Learning de programación.

Figura 1: extraída de [3] que representa el enfoque clásico de programación.

En cambio, cuando usamos Machine Learning permitimos que sea la computadora la encargada de extraer patrones importante del texto de los mails, de manera tal que desarrolle una lógica automática (a través de funciones matemáticas) que al pasarle a la entrada un mail pueda predecir a que categoría pertenece. Con esto se reduce la cantidad de líneas del programa, es más sencillo de mantener y por lo general resulta con mayor exactitud en la predicción [3].

Paradigmas



Las máquinas aprenden **identificando patrones** existentes entre los datos.

Algoritmos de Machine Learning

Las categorías de algoritmos hace referencia al *tipo de aprendizaje*, esto es al enfoque que se tiene a la hora de implementar un algoritmo basándose en los datos y cómo serán manejados durante el entrenamiento.

Los tipos de aprendizaje son los siguientes:

- Aprendizaje Supervisado
- Aprendizaje No Supervisado
- Aprendizaje Semi-Supervisado
- Aprendizaje Reforzado
- Transferencia de Aprendizaje

Aprendizaje Supervisado

Tiene que ver con modelar matemáticamente la relación que existe entre las características de los datos y las etiquetas asociadas a cada dato; de tal manera, que al usar datos nuevos poder inferir la etiqueta en función de sus características [1].

Es como estudiar para un examen las tablas de multiplicar mirando los resultados, y el día del examen preguntan cual es el resultado de un determinado producto. Los números de cada producto serían las características de los datos (el producto en sí) y los resultados las etiquetas asociadas. La lógica asociada a cada producto sería el modelo de Machine Learning donde se puede relacionar las características a y b con el resultado y : $y = a * b$.

El aprendizaje supervisado es uno de los tipos más utilizados para entrenar modelos bajo la *supervisión* de las propias etiquetas originales versus las predichas. Las aplicaciones de este método pueden ser dos:

- **Clasificación:** donde las etiquetas son datos cualitativos, es decir las clases son una categoría, por ejemplo: spam/no-spam, tipos de sentimientos (alegría, amor, miedo, enojo), etc. A su vez dependiendo de la cantidad de categorías o clases para clasificar se divide en:

- **Binaria**: por lo general se utiliza para saber si existe o no una categoría, o bien entre dos categorías muy distintas como pueden ser: perros versus gatos.
- **Multi-clase**: como su nombre lo indica existe una diversidad de clases que el modelo inferirá según los datos de entrada, y aquella clase que tenga mayor probabilidad será la que se considere como la elegida por el modelo para etiquetar esos datos. Esto puede ser por ejemplo clasificar varios tipos de flores en función de sus imágenes.
- **Multi-etiqueta**: este caso es no tan frecuente pero se utiliza en aquellos casos que es necesario etiquetar los datos con varias clases a la vez. Por ejemplo un modelo que clasifique tweets puede tener en cuenta si el tweet es de odio y además si es racista, homofóbico, tóxico; etc. De esta manera se clasificará varias clases en simultáneo en base al entrenamiento del modelo.



Figura 3: extraída de [1] representa un algoritmo de clasificación mediante una simple línea recta con vectores de soporte que optimizan la distancia entre clases.

- **Regresión**: hay caso en lo que se busca predecir una cantidad numérica en base a las características de los datos de entrada. Esto sirve para saber el valor de una casa en función de los m², la cantidad de baños, vecindario; etc. O bien saber cuánto puede costar un seguro médico en función de los datos del paciente: fumador, obesidad, años, sexo, etc. Los tipos de regresiones:

- **Regresión Lineal Simple**: es ampliamente utilizada como principio para entender el comportamiento de la relación entre características y etiquetas. Sirve en casos donde los datos tienen un comportamiento aproximadamente lineal.
- **Regresión Lineal Multivariante**: muy utilizada en caso donde existen muchas variables independiente que explican una variable dependiente.
- **Regresión Polinómica**: en caso de que los datos no tengan una tendencia del tipo lineal se utiliza la regresión polinómica que ofrece más precisión. Esto se deriva de un previo análisis de los datos con *ingeniería de características* donde a través de técnicas de visualización y estadísticas se puede saber cual es el comportamiento de los datos y sus características.
- **Regresión Logística**: la regresión logística es generalmente usada como herramienta para clasificación binaria, aunque se puede utilizar para predecir el valor de probabilidad de una clase, por lo que resulta una regresión numérica.

Algoritmos de Aprendizaje Supervisado

- Clasificación
 - Regresión logística
 - Support Vector Machines
 - Random Forest (árboles de decisión con métodos en ensamble)
 - Naive Bayes (teorema de bayes)
 - K-Nearest Neighbours
 - Redes neuronales
- Regresión
 - Regresión Lineal (OLS, Lasso)
 - Regresión Multivariable (OLS, Lasso, Ridge)
 - Regresión Polinómica (Polynomial Features)
 - Support Vector Machines (SVM)
 - Random Forest
 - Redes Neuronales

Aprendizaje No Supervisado

Este tipo de entrenamiento se realiza sin la presencia de etiquetas conocidas asociadas a las características de los datos. Se utiliza generalmente en datos no estructurados para conocer qué tipo de clases puede existir agrupando datos según ciertas similitudes.

Es como dejar que los datos *hablen por sí mismos* [1]. Por ejemplo, digamos que queremos saber sobre los usuarios que visitan un blog. El algoritmo debería ser capaz de agrupar a los usuarios por similitudes que los conecten. Por ejemplo: un algoritmo de *clustering*, que es lo habitual en este tipo de entrenamientos, sería capaz de decirnos que de las personas que visitan el blog un 30% son amantes de la literatura de ciencia ficción y les gusta leer el blog el fin de semana, y el 70% restante son personas que les gusta la tecnología y lecturas académicas que leen el blog durante la semana [3].

Toda esta información sería posible gracias al algoritmo de Machine Learning que es capaz de obtener patrones de los datos y clasificar dichos patrones sin un *supervisor*.

El aprendizaje no supervisado tiene las siguientes aplicaciones:

- **Clustering**: estos algoritmos buscan diferenciar grupos de datos dentro de un conjunto de datos sin estructura. De tal forma de que cada grupo de datos tenga una etiqueta.
- **Reducción de dimensiones**: busca reducir la cantidad de características de los datos hasta aquellas que tienen mayor correlación y pueden afectar a los datos. Por esto se llama reducción de dimensiones, si los datos tienen dos características *feature1* y *feature2* la relación entre ambas será en un plano 2D y las etiquetas corresponderán a un tercer plano, y si aumentan la cantidad de características aumenta la cantidad de dimensiones y resulta más difícil al algoritmo encontrar una relación entre los datos más sencilla para agruparlos.
- **Detección de anomalías**: por ejemplo detectar inusual transacción de tarjeta de crédito, a fin de prevenir fraude, encontrar defectos en manufacturación, o simplemente remover *outliers* de un dataset antes de alimentar otro algoritmo de entrenamiento [3].



Figura 4: imagen extraída de [3] que muestra la detección de anomalías .

Algoritmos de Aprendizaje No Supervisado

- Clustering
 - K - Means
 - Hierarchical Cluster Analysis (HCA)
 - Gaussian Mixture Models
 - Spectral Clustering
 - Expectation Maximization
- Reducción de dimensiones
 - Mainfold Learning
 - Isomap
 - Locally - Linear Embedding (LLE)
 - Principal Component Analysis (PCA)

Aprendizaje Semi-Supervisado

Algunos algoritmos tienen que trabajar con dataset parcialmente etiquetado, es decir, que algunos datos tendrán etiquetas y otros no. Es en estos casos donde es necesario un entrenamiento semi-supervisado [3].

Por lo general la cantidad de datos sin etiquetas es mucho mayor que los datos etiquetados, y siendo la tarea igual que en un entrenamiento supervisado, el algoritmo deberá trabajar de cierta forma para poder etiquetar datos nuevos [4].

Un ejemplo de esto es Google Fotos, que al subir fotos etiquetadas con cierta persona *A* y después con cierta persona *B*, el algoritmo deberá ser capaz de primero agrupar las fotos por personas y luego trabajando con las etiquetas aprenderá a reconocer nuevas imágenes sin etiquetar de las personas *A* y *B* [3].

Algoritmos de Aprendizaje Semi-Supervisado

La mayoría de los algoritmos de entrenamiento semi-supervisado son combinaciones de sin supervisión y supervisados. Por ejemplo: *deep belief networks* (DBNs) usa como base *restricted Boltzmann machines* (RBMs) que es un componente no supervisado, y luego todo el sistema es ajustado con entrenamiento supervisado.

Aprendizaje Reforzado

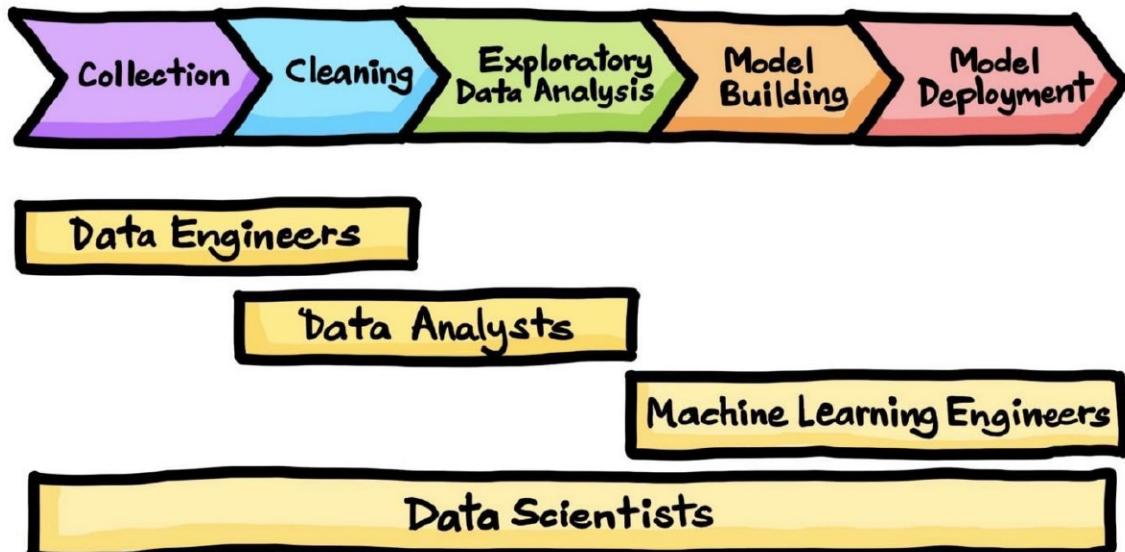
Se denomina así al tipo de entrenamiento que tiene en consideración el ambiente donde se entrena a través de un vector de características. Este vector de características proveniente del ambiente se denomina *state* [4].

En cada *state* que se entrena el algoritmo se ejecutan *actions*, y diferentes acciones traen consigo *rewards*. Todo esto a fin de que el algoritmo aprenda la mejor estrategia o *policy* que le otorgue una *reward*. Esta ultima es una función que relaciona el vector *state* de características como variable independiente y como salida una acción óptima para ejecutar en ese *state*. La acción será óptima si maximiza la recompensa esperada en promedio [4].

Algoritmos de Aprendizaje Reforzado

Muchos robots que aprenden a caminar implementan aprendizaje reforzado. También el algoritmo AlphaGo de DeepMind que se utilizó en 2016 para vencer al campeón de Go utiliza *reinforcement learning* [3].

Flujo de trabajo - Ingeniero en Machine Learning



Flujo de trabajo general cuando se trabaja con datos.

Casos de uso de Machine Learning

Marketing y Ventas

La recomendación de productos basados en nuestros consumos anteriores es una de las más comunes aplicaciones de ML. En función de nuestro historial de compras se puede determinar un perfil de cliente y pueden llegar ciertas promociones. Esta es una de las grandes tendencias dentro del mundo de Marketing y Ventas. Podemos saber en base datos históricos, por ejemplo, cuáles de nuestros clientes están en probabilidad de irse y también cuales ventas tienen posibilidad de cerrarse.

Servicios y Financieros

Por la naturaleza de los datos financieros, el ML es muy utilizado en este sector para dos fines principales: identificar **insights** importantes en los datos y prevenir el fraude.

Salud

El machine learning es una tendencia en rápido crecimiento en la industria de atención a la salud, gracias a la aparición de dispositivos y sensores que pueden usar datos para evaluar la salud de un paciente en tiempo real.

Petróleo y Gas

Cómo encontrar nuevas fuentes de energía. Análisis de minerales del suelo. Predicción de fallos de sensores de refinerías. Optimización de la distribución de petróleo para hacerla más eficiente y económica, etc.

Industria

Mediante reconocimiento de imágenes es posible contar las piezas y obtener un inventario de los productos, que nos puede servir para gestionar el stock y así saber cuánto tenemos que comprar, en cuestión de segundos.

Atención a clientes por medio de voz y texto

Ya no será necesario contar con un call-center para ciertos servicios, debido a que mediante un asistente de voz que identifique ciertas palabras o frases podemos brindar la respuesta adecuada. Por ejemplo, si un usuario quisiera dar de alta o de baja un servicio el sistema podría responder a la petición de voz, haciéndolo más rápido y eficiente.

Seguridad de la información y antifraudes

Se puede detectar usuarios fraudulentos, en base a conocer el comportamiento histórico de la persona. Esto permite reducir altamente el porcentaje de robos por medio de tarjetas de crédito falsas y/o robadas.

Transporte

Analizar datos para identificar patrones y tendencias, es clave para la industria del transporte, que se sustenta en hacer las rutas más eficientes y anticipar problemas potenciales para incrementar la rentabilidad. Los aspectos de análisis y modelado de datos del machine learning, son herramientas importantes para las compañías de mensajería, transporte público y otras organizaciones de transporte.

Fases de un proyecto de ML

1. Definir el objetivo

Para esto es necesario entender el problema (es importante que haya buena comunicación entre los integrantes del grupo de trabajo), y así poder saber qué objetivos alcanzar.

Para esta etapa hay una serie de preguntas que nos ayudarán:

- ¿Qué deseamos hacer?

- ¿Cómo podremos hacerlo?
- ¿Es posible lo que deseo con los **datos** que tengo disponibles?

2. Recolección de datos

Los datos pueden provenir de distintas fuentes:

- **Fuente primaria:** datos recolectados mediante investigación propia o propios de la empresa, institución, etc.
- **Fuente secundaria:** datos comprados o provenientes de una organización aliada, externa.
- **Fuente terciaria:** datos ya procesados, donde no conocemos al generador original.

3. Preparación de los datos (pre-procesamiento)

Una vez que disponemos de los datos, es necesario realizar la **limpieza** de los mismos. Esta etapa, como hemos visto con anterioridad, se centra en la **manipulación** de los datos.

También en un proceso llamado **feature engineering**, se pueden crear variables explicativas adicionales, también conocidas como predictor o *features*, a través de una combinación de dominio y variables estructuradas existentes.

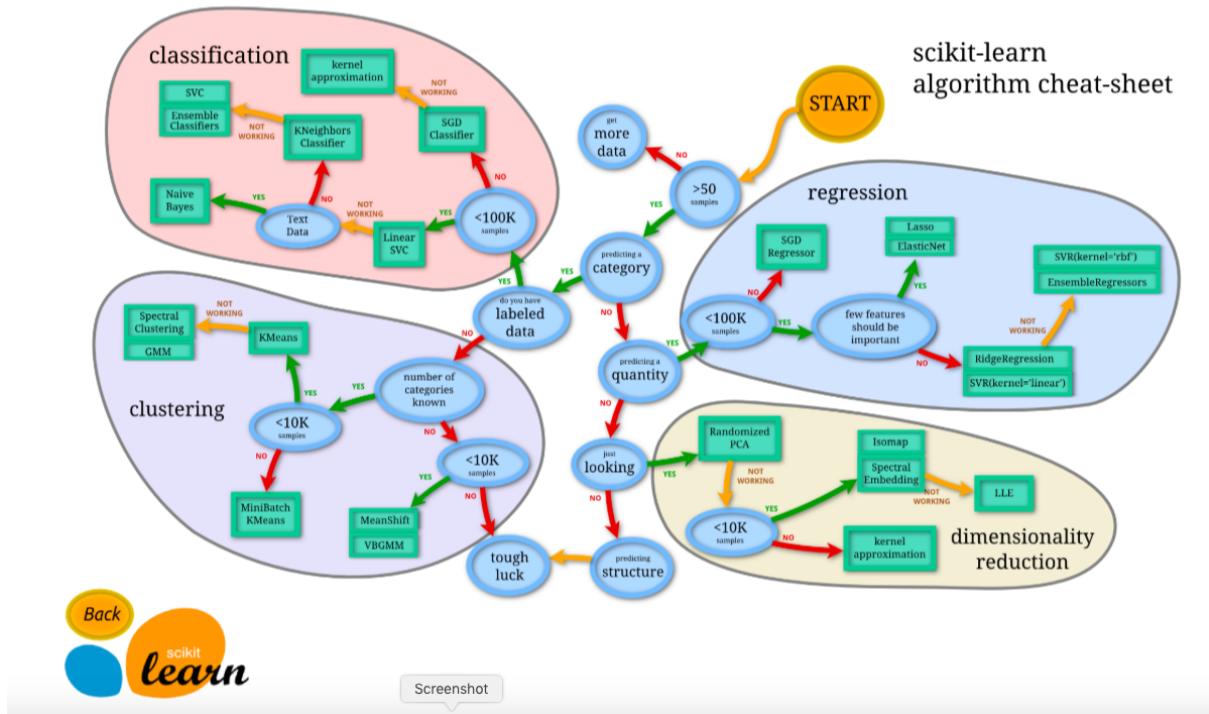
Las tareas clásicas son:

- Categorizar los valores de las variables.
- Eliminar valores atípicos (outliers).
- Inferir datos faltantes.
- Normalizar (también llamado escalar) los valores numéricos o estandarizar.

4. Elección del algoritmo

Estos es, en función del problema que deseamos resolver debemos elegir los algoritmos que queremos utilizar.

Fijémonos en esta ruta de algoritmos de **Scikit-Learn** que nos va a guiar en función de los datos y lo queramos hacer.



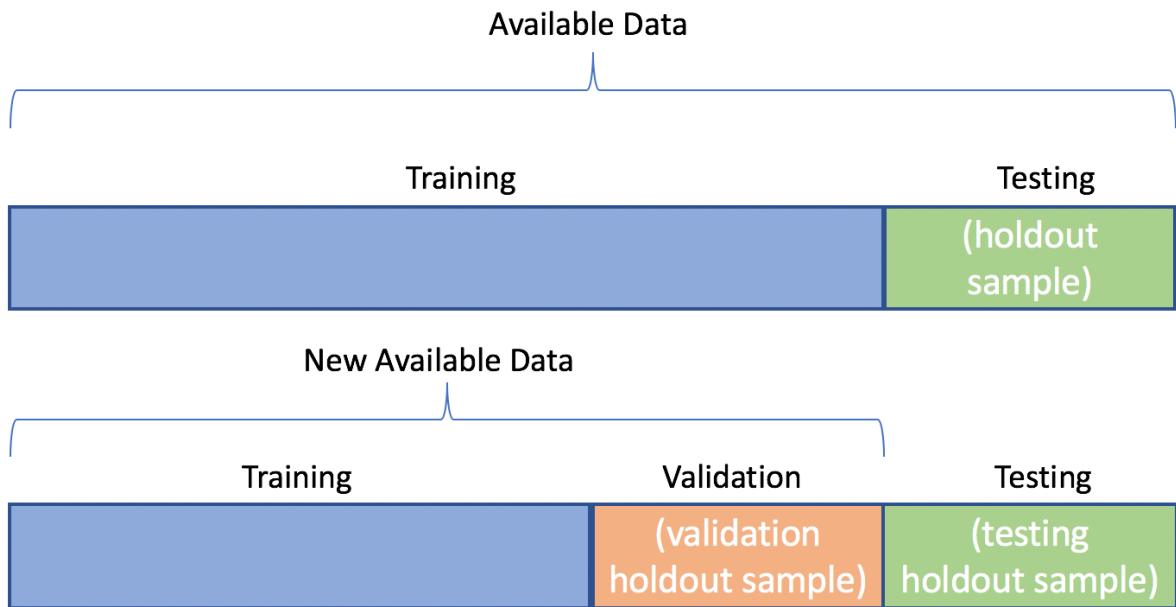
Roadmap de algoritmos en función de la problemática.

5. Entrenar el modelo

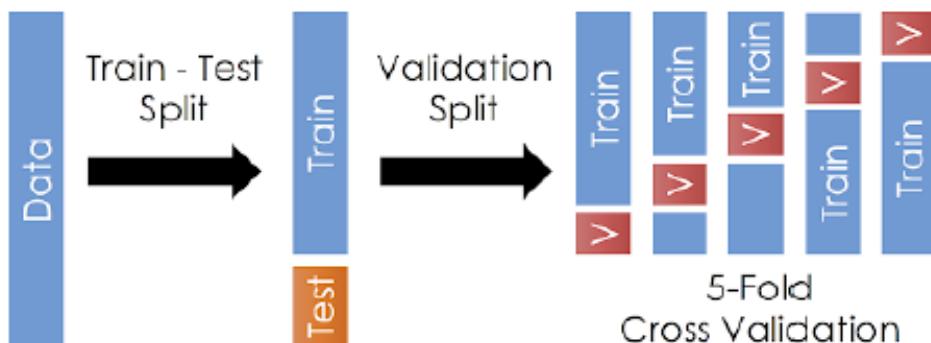
El proceso de entrenamiento de un modelo de ML, consiste en proporcionar al modelo datos de los cuales pueda aprender.

Para realizar el entrenamiento, dividiremos los datos con los que contamos en dos fracciones: datos de entrenamiento y datos de testeо. Se suele utilizar un 70/30 % o 80/20%. Sin embargo, esta proporción es una convención, no hay una proporción correcta.

Sin embargo, las buenas prácticas hacen al programador. Es necesario separar en un tercer grupo que se llama **Validation** (70% train, 15% validation, 15% test) que servirá de prueba para ir validando el entrenamiento a medida que este va sucediendo (por ejemplo: el entrenamiento dura 100 iteraciones, cada 10 se hará una validación) esto permite mejorar la performance del modelo poniendo a prueba los **conocimiento aprendidos** hasta el momento.



Una estrategia avanzada para la validación de los modelos es utilizar validación cruzada (*cross validation*). En esta técnica, lo que se realiza es dividir los datos de entrenamiento en diferentes partes, las cuales se utilizan para obtener una mejor estimación del error del entrenamiento.



Validation por k-folds. En este ejemplo se usaron 5-folds durante el entrenamiento.

6. Evaluar el modelo

En esta etapa lo que se hace es utilizar el modelo entrenado con los datos del set de **test**. Esto es para poner a prueba la capacidad de **generalización** del modelo frente a datos nunca observados (ya que no se usaron durante el entrenamiento).

Para evaluar la performance del modelo existen una serie de métricas que dependen del tipo de algoritmo.

- Regresión:

- Mean Absolute Error, Root Mean Squared Error, Mean Squared Error, R2, ROC-AUC
- Clasificación:
 - Accuracy (exactitud), precision, recall (sensible), f1-score($\frac{\text{precision}+\text{recall}}{\text{precision}\times\text{recall}}$)

7. Optimización del modelo

Una vez evaluado el modelo podemos distinguir mediante sus métricas dos situaciones extremas:

1. **Sobreajuste (overfitting)**: se produce cuando nuestro modelo aprende a la perfección los datos de entrenamiento, por lo que no es capaz de generalizar, y cuando le lleguen nuevos datos obtiene pésimos resultados. Formas de prevenir el overfitting:
 - a. Aumentar el número de datos totales.
 - b. Ajustar los hiperparámetros (son los parámetros que gobiernan el modelo) de nuestros modelos.
 - c. Utilizar modelos más simples (reducir la cantidad de neuronas/capas en una red neuronal).
 - d. Controlar el número de iteraciones (Early Stopping).
2. **Subajuste (underfitting)**: se produce cuando nuestro modelo no es capaz de identificar patrones. Por lo general las métricas gráficamente son constantes o varían muy poco. Y tendrá siempre pésimos resultados. Las formas de prevenirlo son:
 - a. Tratamiento adecuado de los datos (limpieza y normalización).
 - b. Utilizar modelos más complejos (mayor cantidad de neuronas/capas en una red neuronal).
 - c. Ajuste de hiperparámetros de nuestros modelos.
 - d. Mayora cantidad de iteraciones del algoritmos (Early stopping).

8. Deployar el modelo

Esta etapa es la final, pero no termina acá el trabajo de una Machine Learning Engineer. Consiste en la implementación en producción de nuestro modelo. Por lo

general se puede poner en un servidor propio o en algun servicio cloud: AWS, Azure y GCP.

9. Monitorear el modelo

Este paso significa llevar métricas de como funciona el modelo en producción y establecer formas de incorporar mejorar o actualizaciones.

Agentes inteligentes

Esta sección se basa en el enfoque de la IA racional: **sistemas que actúan racionalmente**. Esto se explica en contraposición con *los sistemas que actúan como humanos*: el comportamiento humano es una serie de eventos evolutivos que nos ha conformado como somos hoy en día. Lo más lógico es tratar de abordar la IA como agentes racionales, aunque partimos de la premisa que la racionalidad perfecta es una utopía.

Se estudian los agentes inteligentes porque es *algo que actúa* en base a un programa. Y mientras más programas diversos incluya, más *atributos* va a tener.

Entonces partimos de que un agente es un programa al cual se le agrega:

- Controles autónomos: para realizar diferentes acciones (movilizarse, tomar objetos, etc).
- Sensores: de forma de *percibir* el entorno (cámaras, ultrasonido, acelerómetros, etc).
- Persistir en el tiempo: esto es que el programa debe auto-mantenerse en el tiempo.
- Poder alcanzar diferentes objetivos que se le proponga.

A este agente, en su complejidad, se le agrega el enfoque de *pensar racionalmente*, ya que el pensamiento racional le permite *inferir* resultados o escenarios posibles para alcanzar o maximizar sus objetivos.

Agentes y su entorno

Definición: un agente es cualquier cosa capaz de percibir su medio ambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores.

Pensemos en nosotros mismos como agentes inteligentes 😎, tenemos sensores, controles autónomos, persistimos en el tiempo y tenemos un pensamiento racional que nos permite hacer *inferencias* en base a lo que percibimos de nuestro entorno, sumado a un previo conocimiento incorporado.

El término *percepción* significa que el agente puede recibir entradas en cualquier momento. Y la secuencia de percepciones es el historial completo de lo que el agente ha recibido.

En general: **un agente tomará una decisión en un momento dado dependiendo de la secuencia completa de percepciones hasta ese instante.**

La función que describe el comportamiento de un agente se puede escribir en forma de tabla, teniendo en cuenta todas las secuencias de percepción y determinando que acción lleva a cabo el agente en respuesta. Esta función del agente se implementará en el **programa del agente**. Es importante destacar que la función del agente es la descripción matemática abstracta y el programa es la implementación completa de la función, es la arquitectura del agente.

Para ilustrar esta idea se utilizará un ejemplo muy simple, el mundo de la aspiradora presentado en la Figura 2.2. Este mundo es tan simple que se puede describir todo lo que en él sucede; es un mundo hecho a medida, para el que se pueden inventar otras variaciones.

Este mundo en particular tiene solamente dos localizaciones: cuadrícula A y B. La aspiradora puede percibir en qué cuadrante se encuentra y si hay suciedad en él. Puede elegir si se mueve hacia la izquierda, derecha, aspirar la suciedad o no hacer nada.

Una función muy simple para el agente vendría dada por: si la cuadrícula en la que se encuentra está sucia, entonces aspirar, de otra forma cambiar de cuadrícula. Una muestra parcial de la función del agente representada en forma de tabla aparece en la

Figura 2.3.

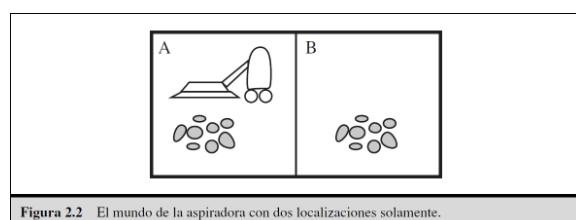


Figura 2.2 El mundo de la aspiradora con dos localizaciones solamente.

Secuencia de percepciones	Acción
[A, Limpio]	Derecha
[A, Sucio]	Aspirar
[B, Limpio]	Izquierda
[B, Sucio]	Aspirar
[A, Limpio], [A, Limpio]	Derecha
[A, Limpio], [A, Sucio]	Aspirar
—	—
—	—
[A, Limpio], [A, Limpio], [A, Limpio]	Derecha
[A, Limpio], [A, Limpio], [A, Sucio]	Aspirar
—	—
—	—

Figura 2.3 Tabla parcial de una función de agente sencilla para el mundo de la aspiradora que se muestra en la Figura 2.2.

De acá se desprende preguntarse: ¿Qué hace que la aspiradora sea un agente bueno o malo, inteligente o no-inteligente?

Buen comportamiento: el concepto de racionalidad

Un agente es *racional* cuando hace lo *correcto*, pero, ¿Qué significa hacer lo correcto?.

Podemos aproximar a decir que lo correcto es aquello que permite al agente obtener un resultado mejor. Por tanto, se necesita determinar una forma de medir el éxito. Ello, junto a la descripción del entorno y de los sensores y actuadores del agente, proporcionará una especificación completa de la tarea que desempeña el agente.

Medidas de rendimiento



"Lo que no se define no se puede medir. Lo que no se mide, no se puede mejorar. Lo que no se mejora, se degrada siempre". *William Thomson Kelvin (Lord Kelvin), físico y matemático británico (1824 – 1907)*

Las medidas de rendimiento incluyen los criterios que determinan el éxito en el comportamiento del agente. Además es importante utilizar medidas de rendimiento objetivas, que normalmente determinará el diseñador encargado de la construcción del agente.

Para el ejemplo de la aspiradora 🤖 se puede proponer utilizar como medida de rendimiento la cantidad de suciedad limpiada en un período de ocho horas. Un agente racional puede maximizar su medida de rendimiento limpiando la suciedad, tirando la basura al suelo, limpiándola de nuevo, y así sucesivamente. Una medida de rendimiento más adecuada recompensaría al agente por tener el suelo limpio. Por ejemplo, podría ganar un punto por cada cuadrícula limpia en cada período de

tiempo (quizás habría que incluir algún tipo de penalización por la electricidad gastada y el ruido generado).

Como regla general, es mejor diseñar medidas de utilidad de acuerdo con lo que se quiere para el entorno, más que de acuerdo con cómo se cree que el agente debe comportarse.

Racionalidad

La racionalidad depende de cuatro factores:

- La medida de rendimiento que define el criterio de éxito.
- El conocimiento del medio en el que habita.
- Las acciones que el agente puede llevar a cabo.
- La secuencia de percepciones del agente hasta ese momento.

Estos cuatro puntos nos lleva a la **definición** de agente racional:



En cada posible secuencia de percepciones, un agente racional deberá emprender aquella acción que supuestamente maximice su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en el conocimiento que el agente mantiene almacenado.

Omnisciencia, aprendizaje y autonomía

Es necesario tener cuidad de no confundir **omnisciencia** con racionalidad. La primera hace referencia al conocimiento absoluto, tanto el resultado de su acción como del entorno, por lo que para un agente (y hasta un humano) este concepto resulta imposible. En cambio la racionalidad es diferente de la *perfección*, la racionalidad maximiza el rendimiento esperado y la perfección el resultado real. Por lo que podemos decir que la definición de **racionalidad** propuesta no requiere omnisciencia, solo dependerá de la secuencia de percepción.

Esto significa que el agente puede mejorarse hasta cierto punto. Dependerán sus acciones futuras en base a la información recopilada, lo cual es esencial en la racionalidad. Pero un agente racional no solo debe recopilar información (*percepción*) sino **aprender**. Debe aprender lo máximo posible de lo que está percibiendo. La configuración inicial del agente puede reflejar un conocimiento preliminar del entorno, pero a medida que el agente adquiere experiencia éste

puede modificarse y aumentar. Hay casos excepcionales en los que se conoce totalmente el entorno a priori. En estos casos, el agente no necesita percibir y aprender; simplemente actúa de forma correcta.

<https://www.youtube.com/watch?v=UwsrzCVZAb8>

El otro rasgo que debe considerarse de un agente racional es la **autonomía**. Estos es que el agente debe saber aprender a determinar como compensar el conocimiento incompleto o parcial inicial. De otra forma, se dice que el agente carece de autonomía cuando se apoya en el conocimiento inicial proporcionado por su diseñador.

En la práctica, pocas veces se necesita autonomía completa desde el comienzo: cuando el agente haya tenido poca o ninguna experiencia, tendrá que actuar de forma aleatoria a menos que el diseñador le haya proporcionado ayuda. Así de la misma forma que la evolución proporciona a los animales sólo los reactivos necesarios para que puedan sobrevivir lo suficiente para aprender por ellos mismos, sería razonable proporcionar a los agentes que disponen de inteligencia artificial un conocimiento inicial, así como de la capacidad de aprendizaje.

La naturaleza del entorno

Para construir un agente racional hay que centrarse en el **entorno de trabajo** que es el *problema* al cual el agente es la *solución*. Para esto hay que especificar un entorno de trabajo con sus diferentes posibilidades.

Especificación del entorno de trabajo

Para discutir la racionalidad del agente aspiradora simple hay que especificar:

- **Rendimiento**
- **Entorno**
- **Actuadores**
- **Sensores del agente**

Todo esto forma el **entorno de trabajo** cuya denominación se utiliza **REAS**. En el diseño de un agente lo primero es especificar el entorno de trabajo lo más completo posible.

Caso de ejemplo de especificación del entorno de trabajo de un agente taxista:

Tipo de agente	Medidas de rendimiento	Entorno	Actuadores	Sensores
Taxista	Seguro, rápido, legal, viaje confortable, maximización del beneficio	Carreteras, otro tráfico, peatones, clientes	Dirección, acelerador, freno, señal, bocina, visualizador	Cámaras, sónar, velocímetro, GPS, tacómetro, visualizador de la aceleración, sensores del motor, teclado

Figura 2.4 Descripción REAS del entorno de trabajo de un taxista automático.

Propiedades de los entornos de trabajo

Para diseñar el agente, los primero como dijimos es establecer el entorno de trabajo. Es importante identificar el *tipo de entorno de trabajo* entre un gran numero pero finito de categorías. Estas categorías determinan hasta cierto punto el diseño más adecuado del agente y la técnica de implementación (pueden existir más pero se establecen las siguientes).

- Totalmente observable: Si los sensores del agente le proporcionan acceso al estado completo del medio en cada momento, entonces se dice que el entorno de trabajo es totalmente observable. Un entorno de trabajo es, efectivamente, totalmente observable si los sensores detectan todos los aspectos que son relevantes en la toma de decisiones.
- Parcialmente observable: Un entorno puede ser parcialmente observable debido al ruido y a la existencia de sensores poco exactos o porque los sensores no reciben información de parte del sistema.
- Determinista: Si el siguiente estado está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno es determinista.
- Estocástico: Si el siguiente estado **no** está totalmente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno es estocástico. El agente taxi es claramente estocástico en este sentido, ya que no se puede predecir el comportamiento del tráfico exactamente.
- Estratégico: Si el medio es determinista, excepto para las acciones de otros agentes, decimos que el medio es estratégico.

- Episódico: En un entorno de trabajo episódico, la experiencia del agente se divide en episodios atómicos. Cada episodio consiste en la percepción del agente y la realización de una única acción posterior. No depende de las acciones que se realizaron en episodios previos. Muchas tareas de clasificación son episódicas. Por ejemplo, un agente que tenga que seleccionar partes defectuosas en una cadena de montaje basa sus decisiones en la parte que está evaluando en cada momento.
- Secuencial: la decisión presente puede afectar a decisiones futuras. El ajedrez y el taxista son secuenciales: en ambos casos, las acciones que se realizan a corto plazo pueden tener consecuencias a largo plazo.
- Estático: Si el entorno **no** puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es estático para el agente. Los medios estáticos son fáciles de tratar ya que el agente no necesita estar pendiente del mundo mientras está tomando una decisión sobre una acción, ni necesita preocuparse sobre el paso del tiempo.
- Dinámico: Si el entorno puede cambiar cuando el agente está deliberando, entonces se dice que el entorno es dinámico para el agente. Los medios dinámicos están preguntando continuamente al agente qué quiere hacer; si no se ha decidido aún, entonces se entiende que ha tomado la decisión de no hacer nada.
- Semidinámico: Si el entorno no cambia con el paso del tiempo, pero el rendimiento del agente cambia, entonces se dice que el medio es semidinámico. (*Ajedrez por tiempo*).
- Discreto: Un medio con estados discretos como el del juego del ajedrez tiene un número finito de estados distintos. El ajedrez tiene un conjunto discreto de percepciones y acciones.
- Continuo: El taxista conduciendo define un estado continuo y un problema de tiempo continuo: la velocidad y la ubicación del taxi y de los otros vehículos pasan por un rango de valores continuos.
- Agente individual: Un agente resolviendo un crucigrama es *individual*.
- Multi-agente: Un agente que juega al ajedrez está en un entorno de *dos agentes*.

Nota: acá hay una diferencial sutil, y es que no se puede saber con certeza si hay otras entidades que son agentes. Por

ejemplo: en el caso del taxista (agente A) tiene que tratar con otros vehículos que pueden ser agentes o mero objetos como un poste de luz. La forma de saberlo es si otro auto está maximizando una medida de rendimiento que dependa del agente A, entonces sería un agente B. Esto es lo que se llama **multiagente cooperativo**. En el caso del ajedrez, donde un rendimiento depende del disminuir el rendimiento del agente adversario el entorno es **multiagente competitivo**.

Estructura de los agentes

Hasta este momento se ha hablado de los agentes describiendo su conducta, la acción que se realiza después de una secuencia de percepciones dada. Ahora, se trata de centrarse en el núcleo del problema y hablar sobre cómo trabajan internamente. El trabajo de la IA es diseñar el **programa del agente** que implemente la función del agente que proyecta las percepciones en las acciones. Se asume que este programa se ejecutará en algún tipo de computador con sensores físicos y actuadores, lo cual se conoce como **arquitectura**:

$$\text{Agente} = \text{arquitectura} + \text{programa}$$

La arquitectura debe ser adecuada al programa (si dice caminar debe tener piernas) aunque puede ser un PC que muestre por pantalla un resultado o un automóvil.

Programa de los agentes

Los programas de los agentes que se describen en este libro tienen la misma estructura: reciben las percepciones actuales como entradas de los sensores y devuelven una acción a los actuadores.

Se resumen cuatro tipos básicos de programas para agentes que encarnan los principios que subyacen en casi todos los sistemas inteligentes:

- Agentes reactivos simples.
- Agentes reactivos basados en modelos.
- Agentes basados en objetivos.
- Agentes basados en utilidad.

1 - Agente reactivos simples:

El tipo de agente más sencillo es el agente reactivo simple. Estos agentes seleccionan las acciones sobre la base de las percepciones actuales, ignorando el resto de las percepciones históricas. Por ejemplo el agente aspiradora simple es un reactivo simple. También posee una conexión denominada **regla de condición-acción** (condicionales) en base a la entrada de los sensores, si se cumple alguna condición se dispara una acción dentro del medioambiente que se encuentre.

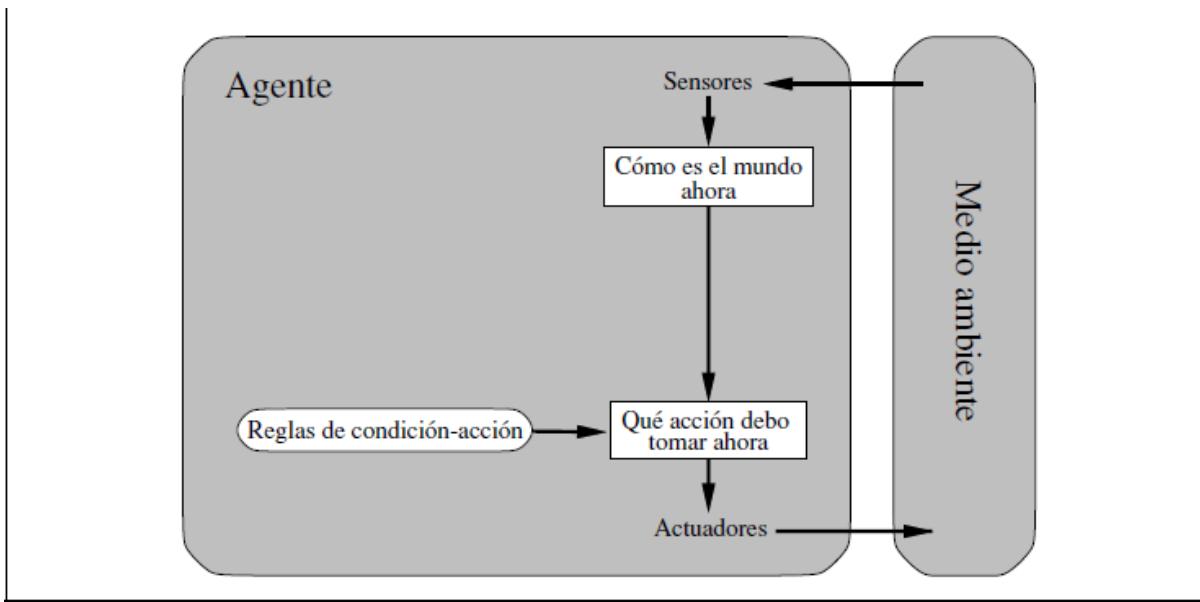


Diagrama esquemático de un agente reactivo simple.

2 - Agente reactivos basados en modelos:

La forma más efectiva que tienen los agentes de manejar la visibilidad parcial es almacenar información de las partes del mundo que no pueden ver. O lo que es lo mismo, el agente debe mantener algún tipo de **estado interno** que dependa de la historia percibida y que de ese modo refleje por lo menos alguno de los aspectos no observables del estado actual.

La actualización de la información de estado interno según pasa el tiempo requiere codificar dos tipos de conocimiento en el programa del agente. Primero, se necesita alguna información acerca de cómo evoluciona el mundo independientemente del agente, por ejemplo, que un coche que está adelantando estará más cerca, detrás, que en un momento inmediatamente anterior. Segundo, se necesita más información sobre cómo afectan al mundo las acciones del agente.

Este conocimiento acerca de *¿cómo funciona el mundo?* se denomina modelo del mundo. De ahí el nombre de **basado en modelos**.

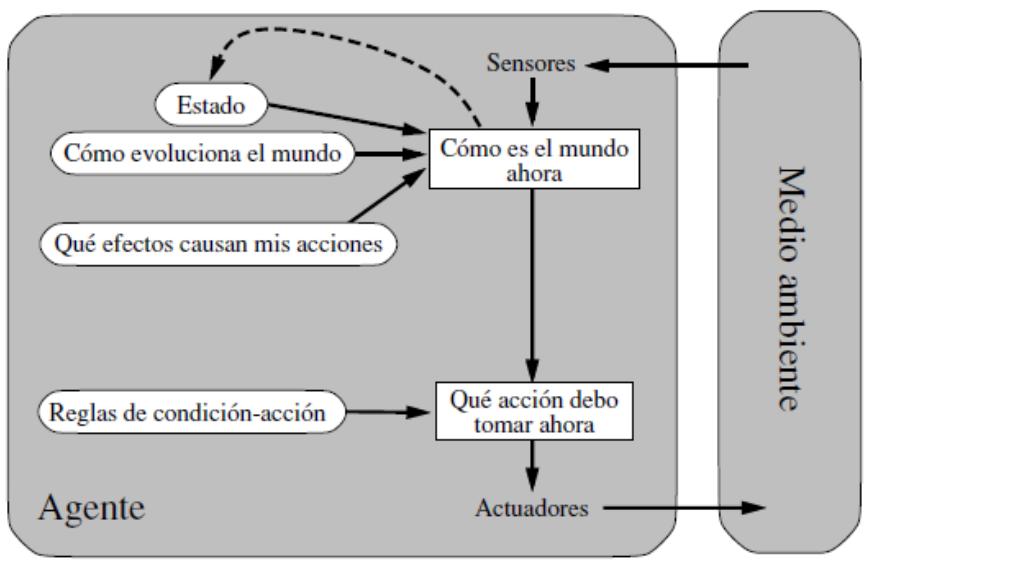


Diagrama de un agente reactivo basado en modelos.

3 - Agentes basados en objetivos

El conocimiento sobre el estado actual del mundo no es siempre suficiente para decidir qué hacer. Por ejemplo, en un cruce de carreteras, el taxista puede girar a la izquierda, girar a la derecha o seguir hacia adelante. La decisión correcta depende de dónde quiere ir el taxi. En otras palabras, además de la descripción del estado actual, el agente necesita algún tipo de información sobre su meta. **Búsqueda y planificación** son los subcampos de la IA centrados en encontrar secuencias de acciones que permitan a los agentes alcanzar sus metas.

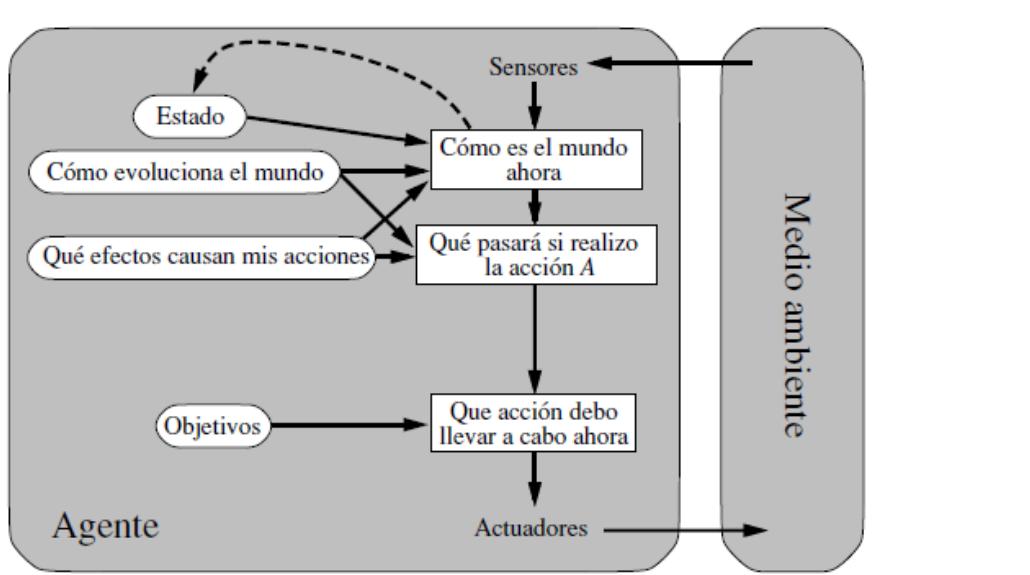


Diagrama de un agente basado en modelos y en objetivos.

4 - Agentes basados en utilidad

Las metas por sí solas no son realmente suficientes para generar comportamiento de gran calidad en la mayoría de los entornos.

Una **función de utilidad** proyecta un estado (o una secuencia de estados) en un número real, que representa un nivel de felicidad. La definición completa de una función de utilidad permite tomar decisiones racionales en dos tipos de casos en los que las metas son inadecuadas. Por ejemplo, velocidad y seguridad, la función de utilidad determina el equilibrio adecuado entre ambas.

Cuando haya varios objetivos por los que se pueda guiar el agente, y ninguno de ellos se pueda alcanzar con certeza, la utilidad proporciona un mecanismo para ponderar la probabilidad de éxito en función de la importancia de los objetivos.

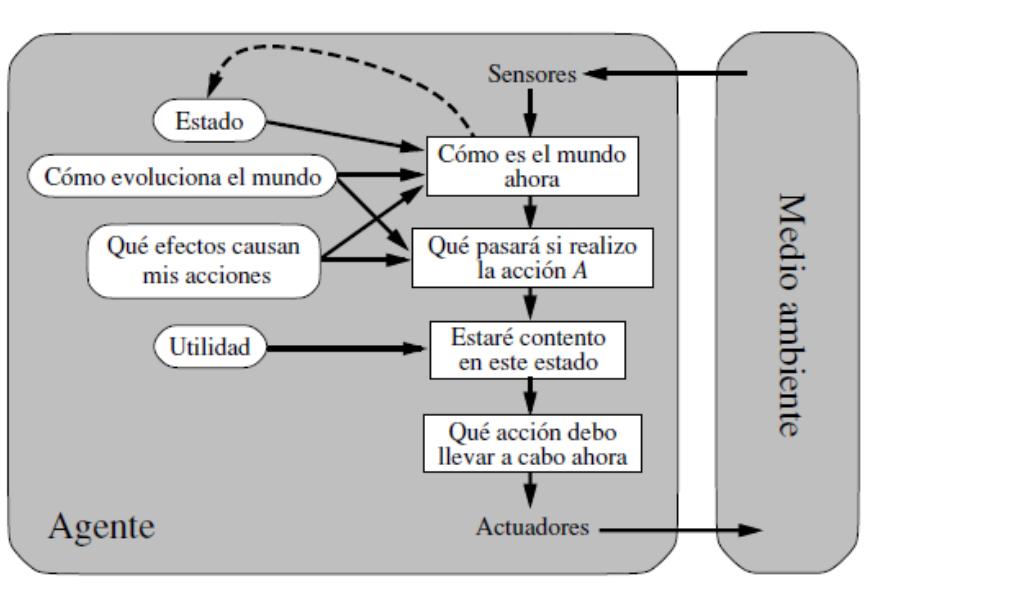


Diagrama de agente basado en utilidad y basado en modelos.

5 - Agentes que aprenden

Se han descrito programas para agentes que poseen varios métodos para seleccionar acciones. Hasta ahora no se ha explicado cómo poner en marcha estos programas de agentes. Turing (1950), consideró la idea de programar sus máquinas inteligentes a mano. Estimó cuánto tiempo podía llevar y concluyó que sería deseable utilizar algún método más rápido. El método que propone es construir máquinas que aprendan y después enseñarles.

El **aprendizaje** tiene otras ventajas, como se ha explicado anteriormente: permite que el agente opere en medios inicialmente desconocidos y que sea más competente que si sólo utilizase un conocimiento inicial.

El diseño del **elemento de aprendizaje** depende mucho del diseño del elemento de actuación. ¿Qué tipo de elemento de actuación necesita el agente para llevar a cabo su objetivo, cuando haya aprendido cómo hacerlo?

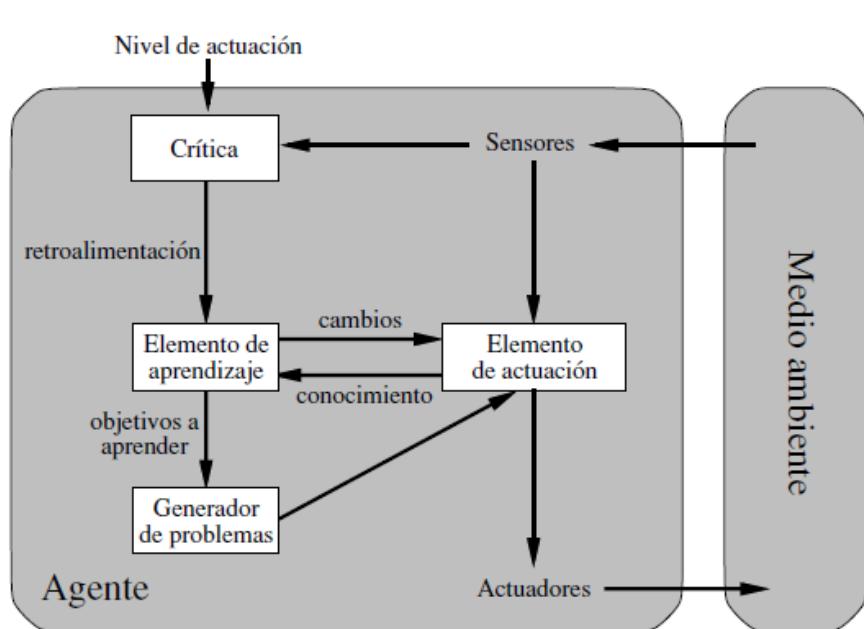


Diagrama de modelo general para agentes que aprenden.

La **crítica** indica al elemento de aprendizaje qué tal lo está haciendo el agente con respecto a un nivel de actuación fijo (indicación del éxito del agente).

El último elemento es el **generador de problemas**. Es responsable de sugerir acciones que lo guiarán hacia experiencias nuevas e informativas. Lo interesante es que si el elemento de actuación sigue su camino, puede continuar llevando a cabo las acciones que sean mejores, dado su conocimiento. Pero si el agente está dispuesto a explorar un poco, y llevar a cabo algunas acciones que no sean totalmente óptimas a corto plazo, puede descubrir acciones mejores a largo plazo. El trabajo del generador de problemas es sugerir estas acciones exploratorias.

Para concretar el diseño total, se puede volver a utilizar el ejemplo del taxi automatizado. El elemento de actuación consiste en la colección de conocimientos y procedimientos que tiene el taxi para seleccionar sus acciones de conducción.

Heurística

Conjuntos de técnicas o herramientas para resolver un problema.

Las revelaciones sobre la toma de decisiones de nuestro cerebro, le valió un premio Novel al psicólogo Daniel Kahneman por su trabajo junto a Amos Nathan Tversky. En él desvelaban los dos métodos que utiliza el cerebro para llegar a razonamientos. Uno más rápido y propenso a errores y otro más metódico y efectivo.

Un sistema es para aquellos casos donde la velocidad lo es todo (alejar la mano de algo caliente) y el segundo sistema es para aquellas situaciones donde reflexionar nos ayuda a resolver un problema.

El primer sistema ha desarrollado una serie de Heurísticos (*reglas generales*). Ya que con estos heurísticos las decisiones se pueden tomar rápidamente. Pero estas reglas tienen errores que nos llevan a sesgos *sistemáticos*. Un ejemplo conocido es el sesgo de confirmación: se trata de que prestamos atención en el entorno a todo aquello que confirma lo que pensamos/creemos y esto hace que nos cueste trabajo dejar de creerlo.

Principios del pensamiento Heurístico:

- Representatividad
- Disponibilidad
- Ajuste y anclaje

En resumen: Las heurísticas son técnicas de resolución de problemas complejos dividiéndolos en problemas más sencillos.

Metaheurística

Las metaheurísticas son técnicas de búsqueda y optimización que se utilizan para resolver problemas complejos en diversas áreas, como la ingeniería, la informática, la economía y la ciencia en general.

A diferencia de los algoritmos clásicos, que encuentran soluciones exactas para problemas específicos, las metaheurísticas están diseñadas para encontrar soluciones aproximadas a problemas difíciles o imposibles de resolver mediante métodos analíticos.

Algunos ejemplos de metaheurísticas son el enfriamiento simulado (simulated annealing), algoritmo genético, enjambre de partículas y la búsqueda tabú. Cada una de estas técnicas tiene sus propias características y se utiliza para resolver diferentes tipos de problemas.

- El recocido simulado, por ejemplo, es útil para resolver problemas de optimización combinatoria, como el problema del viajante de comercio.
- El algoritmo genético se utiliza para resolver problemas de optimización y selección natural, como la evolución de especies.
- El enjambre de partículas es útil para resolver problemas de optimización numérica, como la sincronización de señales.
- Y la búsqueda tabú se utiliza para resolver problemas de optimización en entornos dinámicos y complejos.

En resumen, las metaheurísticas son una herramienta poderosa para resolver problemas difíciles y complejos en diversas áreas. Su capacidad para encontrar soluciones aproximadas en un tiempo razonable las convierte en una opción atractiva para muchas aplicaciones prácticas.

Sin embargo, a veces los algoritmos metaheurísticos puede que no converjan a un mínimo global, en la optimización de alguna función *fitness*, y también son sensibles al ajuste de los parámetros por parte del usuario.

Tipos de metaheurísticas

Single solution based

Las metaheurísticas basadas en una sola solución se enfocan en encontrar la mejor solución posible a un problema dado, partiendo de una solución inicial y haciendo iteraciones para mejorarla. Estas metaheurísticas funcionan explorando el espacio de soluciones, evaluando la calidad de cada solución y ajustando la solución actual hasta que se encuentre una solución óptima o se alcance un límite de tiempo o iteraciones.

Algunos ejemplos de metaheurísticas basadas en una sola solución son: simulated annealing, iterated local search, variable neighborhood search, y guided local search.

- Enfriamiento simulado (simulated annealing): Inspirada en el proceso de enfriamiento de un metal, en el que se va reduciendo su temperatura lentamente para que las moléculas tengan tiempo de acomodarse en la estructura cristalina. En el enfriamiento simulado, se parte de una solución inicial y se van generando soluciones aleatorias en su vecindad. Si la nueva solución es mejor que la actual, se acepta; si no, se acepta con una probabilidad que disminuye a medida que avanza la búsqueda. El enfriamiento simulado

puede ser utilizado para problemas de optimización combinatoria, como el problema del viajante de comercio.

- Búsqueda tabú (tabu search): Esta técnica se enfoca en evitar que la búsqueda se quede atrapada en ciclos o en soluciones subóptimas. Se lleva un registro de las soluciones previamente visitadas y se les asigna una "etiqueta" para evitar volver a visitarlas en el futuro. La búsqueda tabú puede ser utilizada para problemas de optimización combinatoria, como el problema del coloreo de grafos.
- Búsqueda local iterada (iterative local search): Esta técnica se enfoca en mejorar de manera iterativa una solución inicial. Se parte de una solución aleatoria y se aplican operadores de perturbación para generar soluciones en su vecindad. Se selecciona la mejor solución de la vecindad y se repite el proceso hasta que se alcance un criterio de parada. La búsqueda local iterada puede ser utilizada para problemas de optimización combinatoria, como el problema de la mochila.

Population solution based

Las metaheurísticas basadas en poblaciones se enfocan en encontrar la mejor solución posible a un problema dado partiendo de una población inicial de soluciones y haciendo iteraciones para mejorar la población hasta que se alcance un límite de tiempo o iteraciones. Estas metaheurísticas funcionan explorando el espacio de soluciones, evaluando la calidad de cada solución y generando nuevas soluciones mediante operadores de diversidad y cruzamiento para mantener la variabilidad genética de la población.

En estas metaheurísticas basadas en poblaciones podemos encontrar:

- Algoritmos evolutivos: Imitan los comportamientos de la evolución biológica con características de recombinación, mutuación y selección. Dentro de estos el más conocido es el Algoritmo Genético, basado en la teoría de evolución de Darwin. Otro conocido es el Differential Evolution, Genetic Programming y Biogeography-Based Optimizer.
- Algoritmos basados en la física: Cómo su nombre nos dice son algoritmos inspirados en las leyes de la física. Algunos ejemplos son Big-Bang-Big-Crunch (BBCB), Central Force Optimization (CFO) y Gravitational Search Algorithm.
- Algoritmos humanistas: Estos imitan comportamientos humanos. Uno de ellos es el Socio Evolution and Learning Optimization.

- Inteligencia de Enjambre: Estos son algoritmos conocidos basados en comportamientos sociales entre organismos que viven en manadas, enjambres o bandadas, etc. También llamados algoritmos bio-inspirados o inspirados en la naturaleza. Por ejemplo: el comportamiento de bandadas de pájaros inspiro el algoritmo Particle Swarm Optimization (PSO), o está el Ant Colony Optimization, Artificial Bee Colony, Grey Wolf Optimization, Harrys Hawks Optimization, etc.

En resumen, las metaheurísticas basadas en poblaciones son una herramienta poderosa para resolver problemas difíciles y complejos en diversas áreas. Su capacidad para encontrar soluciones aproximadas en un tiempo razonable las convierte en una opción atractiva para muchas aplicaciones prácticas.

Algoritmos de búsqueda

Búsqueda heurística: se orientan a reducir la cantidad de búsquedas para encontrar una solución. Cuando un problema es representado como un árbol de búsqueda, el enfoque heurístico intenta reducir el tamaño del árbol cortando nodos pocos prometedores. La heurística no garantiza que no siempre se toma la decisión correcta, por lo que el método no es óptimo sino suficientemente bueno. Para esto se implementan *métodos de búsqueda heurística*.

Estos métodos disponen de alguna información sobre la proximidad de cada estado a un estado *objetivo* lo que permite explorar los caminos más prometedores. Las características de estos métodos son:

- No garantiza que se encuentre una solución aunque existan soluciones.
- Si encuentran una solución no se asegura que esta tenga las mejores propiedades.
- A veces se encontrará una solución en un tiempo razonable.

Entre las técnicas heurísticas se destacan:

- Best-first search.
- A*
- Hill climbing
- ACO

Teorema No Free Lunch

Todos los algoritmos de optimización propuestos hasta la fecha muestran un rendimiento equivalente en promedio si los aplicamos a todas las tareas de optimización posibles.

Para resolver esta cuestión, dos investigadores norteamericanos, David Wolpert y William Macready, publicaron un artículo en 1997 donde establecieron un teorema denominado “No free lunch”, que traducido sería algo así como “no hay comida gratis” (piénsenlo como en la trampa de un ratón). Dicho teorema establece que, por cada par de algoritmos de búsqueda, hay tantos problemas en el que el primer algoritmo es mejor que el segundo como problemas en el que el segundo algoritmo es mejor que el primero.

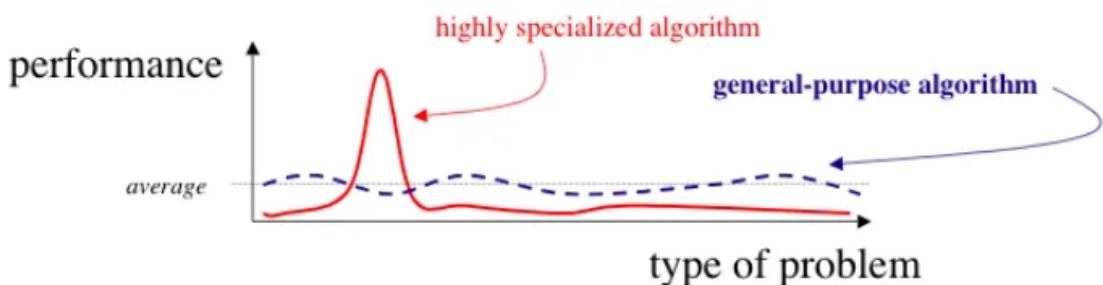


Figura extraída de <https://medium.com/@LeonFedden/the-no-free-lunch-theorem-62ae2c3ed10c>

Por un lado lo que nos dice este teorema resulta en que la idea de un algoritmo inteligente funcional para la generalidad de problemas es casi una utopía dejarlo en las manos de una sola metaheurística, tal vez requiera de otras tecnologías como *deep learning*.

Por otro lado genera opciones para trabajar en algoritmos específicos para la problemática, brindando la posibilidad de que estas investigaciones continúen.

Referencias

1. VanderPlas, J. (2016). *Python data science handbook: Essential tools for working with data.* " O'Reilly Media, Inc."
2. Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: principles and techniques for data scientists.* " O'Reilly Media, Inc.".
3. Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media.

4. Burkov, A. (2019). *The hundred-page machine learning book* (Vol. 1, pp. 3-5). Canada: Andriy Burkov.
5. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
6. Brewka, G. (2005). Artificial intelligence—a modern approach by Stuart Russell and Peter Norvig, Prentice Hall. Series in Artificial Intelligence, Englewood Cliffs, NJ.

Made with ❤️ Ignacio Bosch