SENATI

Artificial Intelligence with Machine Learning in Java

MATERIAL TÉCNICO DE APOYO

RDA
RECURSO DIDÁCTICO PARA EL APRENDIZAJE

TAREA N°01

Explica el uso de la IA.

Introducción

- **Resumen del curso.**

  El curso "Artificial Intelligence with Machine Learning in Java" ofrece una introducción práctica al campo de la inteligencia artificial (IA) y el aprendizaje automático utilizando el lenguaje de programación Java. Durante el curso, los estudiantes aprenden los fundamentos de la IA y el aprendizaje automático, así como cómo aplicar estos conceptos utilizando Java y diversas bibliotecas populares de aprendizaje automático, como Weka y Deeplearning4j. El curso cubre temas como el procesamiento de datos, la creación de modelos de aprendizaje automático, la evaluación del rendimiento del modelo y la implementación de aplicaciones prácticas de IA y aprendizaje automático. Los estudiantes salen del curso con una comprensión sólida de los conceptos fundamentales de la IA y el aprendizaje automático, así como con las habilidades prácticas necesarias para desarrollar aplicaciones de IA utilizando Java.

- **Introducción a la IA.**

  La inteligencia artificial (IA) es un campo multidisciplinario que abarca diversas áreas de la informática, las matemáticas, la estadística, la neurociencia y la filosofía. Su objetivo es desarrollar sistemas que puedan realizar tareas que normalmente requerirían la inteligencia humana. Estas tareas van desde el reconocimiento de patrones en grandes conjuntos de datos hasta la toma de decisiones complejas en entornos dinámicos y cambiantes.

  Una de las ramas más importantes de la IA es el aprendizaje automático (machine learning), que se basa en algoritmos y modelos estadísticos para permitir a las computadoras aprender de los datos y mejorar su rendimiento con la experiencia. El aprendizaje automático se divide en diversas subáreas, como el aprendizaje supervisado, el no supervisado y el por refuerzo, cada una de las cuales tiene sus propias aplicaciones y técnicas específicas.

  Otra área importante de la IA es el procesamiento del lenguaje natural (natural language processing, NLP), que se centra en la interacción entre las computadoras y el lenguaje humano. Los sistemas de NLP permiten a las máquinas comprender, interpretar y generar lenguaje humano de manera efectiva, lo que facilita la comunicación y la interacción con los usuarios en una amplia variedad de aplicaciones, como los asistentes virtuales, la traducción automática y el análisis de sentimientos en redes sociales.

Además, la inteligencia artificial se aplica en áreas como la robótica, donde se utilizan algoritmos de planificación y control para permitir que los robots realicen tareas físicas de manera autónoma; la visión por computadora, que permite a las máquinas entender y procesar imágenes y videos; y la medicina, donde se utilizan técnicas de IA para el diagnóstico médico, la personalización de tratamientos y la investigación de nuevos fármacos.

En resumen, la inteligencia artificial es un campo amplio y en constante evolución que tiene el potencial de transformar radicalmente la manera en que interactuamos con la tecnología y abordamos los problemas en una amplia gama de sectores y disciplinas. Su capacidad para analizar grandes cantidades de datos, detectar patrones complejos y tomar decisiones basadas en la información la convierte en una herramienta poderosa para resolver problemas y mejorar la eficiencia en la sociedad moderna.

La historia de la Inteligencia Artificial (IA) se remonta a mediados del siglo XX, cuando expertos en tecnología comenzaron a trabajar en la creación de máquinas capaces de emular el comportamiento humano. Desde entonces, la IA ha experimentado avances significativos en áreas como el aprendizaje automático, el procesamiento del lenguaje natural y la robótica.

Los hitos más importantes en esta evolución han sido los siguientes:

- La Conferencia de Dartmouth (1956): En esta conferencia, celebrada en Dartmouth College en 1956, pioneros como John McCarthy, Marvin Minsky, Allen Newell y Herbert Simon acuñaron el término "inteligencia artificial" y establecieron los fundamentos para el estudio de sistemas que imiten la inteligencia humana.

- Desarrollo del lenguaje de programación Lisp: Creado por John McCarthy en la década de 1950, Lisp se destaca por su estructura basada en listas enlazadas, que ofrece flexibilidad y eficiencia en la manipulación de datos. Además, es un lenguaje extensible, lo que permite a los programadores adaptarlo según las necesidades del programa, especialmente en áreas como el procesamiento del lenguaje natural y la representación del conocimiento.

- Programa de ajedrez de IBM: En la década de 1950, los investigadores de IBM desarrollaron un programa de ajedrez llamado "IBM 704", un hito en la aplicación de la lógica y la programación para simular el pensamiento estratégico, aunque rudimentario en comparación con los programas modernos.

- Test de Turing (1950): Propuesto por Alan Turing, este test se convirtió en un referente importante para evaluar la capacidad de una máquina para mostrar un comportamiento inteligente similar al humano.

- Desarrollo de redes neuronales: En los años 40 y 50, Warren McCulloch y Walter Pitts propusieron un modelo matemático de una red neuronal simplificada basada en el funcionamiento de las neuronas biológicas, sentando las bases para el desarrollo de redes neuronales artificiales.

- Perceptrón: Desarrollado por Frank Rosenblatt en la década de 1950, el Perceptrón fue el primer modelo de red neuronal artificial con capacidad de aprendizaje, capaz de reconocer patrones simples y utilizado en diversas aplicaciones de reconocimiento de imágenes.

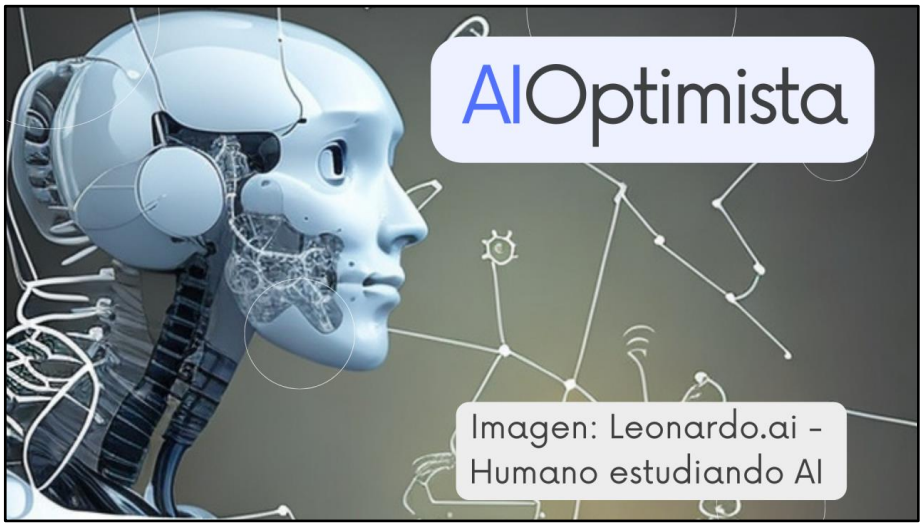

Figura N° 01: Introducción a la Inteligencia Artificial.

➢ **Datos e información.**

En el año 2023, se presentó una notable expansión de la inteligencia artificial (IA) en el ámbito del análisis de datos y otras áreas, lo que brindó oportunidades excepcionales para descubrir ideas valiosas y mejorar la toma de decisiones.

A pesar de la dificultad inherente a predecir el futuro con precisión debido a nuestro sesgo retrospectivo, es posible vislumbrar cómo podemos aprovechar las principales tendencias actuales en IA y extrapolar su impacto a corto y medio plazo en la toma de decisiones empresariales basadas en información sólida.

El progreso en Inteligencia Artificial (IA) y Aprendizaje Automático nos capacita para identificar patrones complejos en conjuntos de datos masivos, realizar predicciones precisas basadas en datos históricos, analizar el comportamiento de los usuarios para ofrecer recomendaciones personalizadas, automatizar tareas repetitivas y optimizar procesos empresariales. Desde predecir ventas hasta detectar

comportamientos fraudulentos, evaluar el riesgo de enfermedades y robotizar operaciones y tareas específicas, todo ello con un enfoque centrado en el individuo y la prestación de servicios personalizados.

- Datos: Los datos son la materia prima de la era digital. Con el crecimiento exponencial de la conectividad y la digitalización en los últimos años, se ha generado una enorme cantidad de datos en todo tipo de formatos y desde diversas fuentes. Estos datos pueden provenir de transacciones comerciales, interacciones en redes sociales, dispositivos conectados a Internet (IoT), registros médicos, registros gubernamentales, entre otros. Los datos son esencialmente puntos de información que, por sí solos, pueden no tener un significado inherente, pero que adquieren relevancia cuando se recopilan, almacenan y procesan adecuadamente.

- Información: La información surge cuando los datos se organizan, analizan y se les da contexto. Mientras que los datos son más bien crudos y desorganizados, la información resulta de interpretar y dar sentido a esos datos. Por ejemplo, si tenemos una base de datos de ventas con fechas, productos y montos, podemos analizar estos datos para obtener información valiosa, como cuáles son los productos más vendidos en ciertas épocas del año o qué patrones de compra tienen ciertos clientes. La información proporciona conocimientos que pueden utilizarse para tomar decisiones informadas y resolver problemas.

- Inteligencia Artificial (IA): La IA es el conjunto de tecnologías y técnicas que permiten a las máquinas simular procesos cognitivos humanos, como el aprendizaje, el razonamiento y la resolución de problemas. Los sistemas de IA utilizan algoritmos y modelos para procesar grandes volúmenes de datos, identificar patrones complejos y aprender de ellos. La IA abarca una amplia gama de aplicaciones, desde sistemas de recomendación en plataformas de streaming hasta vehículos autónomos, asistentes virtuales, diagnósticos médicos y mucho más. En el contexto de los datos y la información, la IA juega un papel crucial al analizar datos para extraer información significativa, automatizar tareas y procesos, y tomar decisiones inteligentes.

La relación entre los datos, la información y la IA es sinérgica y fundamental para el funcionamiento de la economía digital actual. Los datos proporcionan la materia prima necesaria para generar información, mientras que la IA aprovecha esta información para ofrecer inteligencia, automatización y capacidades de toma de decisiones avanzadas. En conjunto, estos elementos impulsan la innovación, mejoran la eficiencia y generan valor en una amplia variedad de sectores y aplicaciones.

- **Categorización de datos.**

  En el contexto de la inteligencia artificial (IA), la categorización de datos es un proceso fundamental que implica organizar y clasificar los datos en grupos o categorías con características similares. Este proceso es crucial para varias aplicaciones de IA, ya que permite identificar patrones, hacer predicciones y tomar decisiones informadas. Aquí hay algunas formas en que se puede categorizar los datos en la IA:

  - **Categorización basada en la naturaleza de los datos:**
    - Datos estructurados: Son datos organizados en un formato predefinido y consistente, como tablas de bases de datos o hojas de cálculo, donde cada campo tiene un tipo de datos específico y las relaciones entre los datos están claramente definidas.
    - Datos no estructurados: Son datos que no tienen un formato predefinido y no se pueden organizar fácilmente en tablas o bases de datos. Esto incluye texto, imágenes, audio, video y otros tipos de datos que no siguen una estructura fija.

  - **Categorización basada en el propósito de la IA:**
    - Datos de entrenamiento: Son datos utilizados para entrenar modelos de IA. Estos datos suelen estar etiquetados con la salida deseada (por ejemplo, etiquetas de clase en clasificación o valores objetivo en regresión) y se utilizan para enseñar al modelo a reconocer patrones y hacer predicciones.
    - Datos de validación: Son datos utilizados para evaluar la precisión y el rendimiento de un modelo de IA entrenado. Estos datos son independientes de los datos de entrenamiento y se utilizan para comprobar si el modelo generaliza bien a datos nuevos y no vistos durante el entrenamiento.
    - Datos de prueba: Son datos utilizados para realizar pruebas finales en un modelo de IA después de que se ha entrenado y validado. Estos datos son completamente nuevos para el modelo y se utilizan para medir su rendimiento en condiciones del mundo real.

  - **Categorización basada en el dominio de aplicación:**

    - Datos médicos: Incluyen información de pacientes, registros de salud, resultados de pruebas, imágenes médicas y otros datos relacionados con la atención médica. Se utilizan en aplicaciones de IA para diagnóstico médico, predicción de enfermedades, análisis de imágenes médicas, entre otros.
    - Datos financieros: Incluyen datos de transacciones, historiales crediticios, precios de acciones, tasas de interés y otros datos relacionados con el sector financiero. Se utilizan en aplicaciones de IA para detección de fraudes, predicción del mercado, gestión de riesgos, entre otros.

- Datos de redes sociales: Incluyen publicaciones, comentarios, interacciones de usuarios, conexiones de redes sociales y otros datos generados por plataformas de redes sociales. Se utilizan en aplicaciones de IA para análisis de sentimientos, recomendaciones personalizadas, segmentación de audiencias, entre otros.

La categorización de datos en la IA es un proceso flexible y adaptable que puede variar según el contexto y los requisitos específicos de cada aplicación. La capacidad para categorizar y utilizar eficazmente los datos es fundamental para el éxito de los proyectos de IA y el desarrollo de soluciones inteligentes en una amplia variedad de campos.

Revolución financiera: Inteligencia Artificial y categorización de datos:
En el mundo financiero contemporáneo, donde los datos fluyen en abundancia como corrientes digitales, la inteligencia artificial (IA) está desempeñando un papel fundamental al redefinir la categorización de datos en el sector financiero. Desde la automatización de procesos hasta la generación de pronósticos predictivos, la IA está remodelando la manera en que las instituciones financieras gestionan y aprovechan la información. En este artículo, exploraremos cómo la inteligencia artificial está moldeando el futuro de la categorización de datos en el ámbito financiero.

- Automatización de tareas rutinarias:

- En el ámbito financiero, la categorización de datos implica comúnmente la clasificación de transacciones, la identificación de patrones de gasto y la asignación de etiquetas a diversas operaciones. La IA ha demostrado ser una herramienta invaluable para automatizar estas tareas repetitivas. Los algoritmos avanzados de aprendizaje automático pueden analizar extensos conjuntos de datos financieros, detectar patrones y asignar categorías de manera precisa y eficiente. Este enfoque automatizado no solo mejora la velocidad de procesamiento, sino que también reduce significativamente la posibilidad de errores humanos, proporcionando una mayor confiabilidad a las instituciones financieras en sus actividades diarias.

- Análisis predictivo para decisiones estratégicas:
  La inteligencia artificial también puede generar perspectivas predictivas que son fundamentales para la toma de decisiones estratégicas en el ámbito financiero. Los modelos de aprendizaje automático pueden analizar datos históricos para prever tendencias futuras, permitiendo a las instituciones financieras anticiparse a cambios en los mercados, el comportamiento del cliente y los riesgos potenciales. Por ejemplo, un banco puede emplear modelos predictivos de IA para evaluar el riesgo crediticio de un cliente, considerando una variedad más amplia de variables y mejorando la precisión de las decisiones crediticias.

- **Personalización de experiencias del cliente:**
  La IA también facilita la personalización de las experiencias del cliente en el sector financiero. Al comprender los patrones de gasto y las preferencias individuales, los sistemas de IA pueden ofrecer recomendaciones y servicios financieros altamente personalizados. Imagina un asistente financiero virtual que, basándose en la categorización precisa de tus transacciones, te brinda consejos personalizados de ahorro, inversiones o planificación financiera. La IA tiene el potencial de transformar la relación entre las instituciones financieras y sus clientes, haciéndola más centrada en el usuario y adaptada a sus necesidades específicas.

- **Detección de fraude y seguridad:**
  La categorización de datos impulsada por la IA es una herramienta poderosa en la detección de actividades fraudulentas. Los algoritmos de aprendizaje automático pueden identificar patrones de comportamiento inusuales o transacciones sospechosas, alertando a las instituciones financieras sobre posibles amenazas de seguridad. Esto no solo protege a los clientes de fraudes financieros, sino que también resguarda la integridad y reputación de las propias instituciones. La IA puede analizar grandes volúmenes de datos en tiempo real, permitiendo una respuesta rápida y efectiva ante posibles amenazas.

- **Inteligencia Artificial en la experiencia del usuario:**
  La inteligencia artificial está desempeñando un papel transformador en la categorización de datos financieros. Desde la automatización de tareas rutinarias hasta la generación de análisis predictivos, la IA está remodelando la forma en que las instituciones financieras operan y sirven a sus clientes. No obstante, es esencial abordar los desafíos éticos y regulatorios para garantizar que la adopción de la inteligencia artificial en finanzas sea segura, ética y beneficiosa para todos los involucrados. La revolución financiera impulsada por la inteligencia artificial está en marcha, y su impacto continuará transformando la forma en que interactuamos con el mundo financiero.

Aprendizaje automático

El aprendizaje automático (ML) es un proceso mediante el cual se utilizan modelos matemáticos de datos para permitir que un sistema aprenda sin la necesidad de instrucciones directas. Este enfoque se considera como una subdisciplina de la inteligencia artificial (IA). El aprendizaje automático emplea algoritmos para descubrir patrones dentro de los datos, y estos patrones se utilizan posteriormente para construir un modelo de datos capaz de realizar predicciones. A medida que se acumula más experiencia y datos, los resultados del aprendizaje automático tienden a ser más precisos, de manera similar a cómo los humanos mejoran con la práctica continua.

La flexibilidad inherente al aprendizaje automático lo convierte en una opción óptima en escenarios donde los datos son cambiantes, las solicitudes o tareas varían constantemente, o donde sería difícil codificar una solución de manera convencional.

El Machine Learning, una disciplina dentro del campo de la Inteligencia Artificial, capacita a las computadoras para identificar patrones en grandes volúmenes de datos y realizar predicciones, también conocidas como análisis predictivo. Este enfoque de aprendizaje permite que las computadoras ejecuten tareas específicas de manera autónoma, sin necesidad de una programación explícita.

Aunque el término "Machine Learning" se utilizó por primera vez en 1959, ha cobrado una mayor relevancia en años recientes debido al aumento en la capacidad de procesamiento de los ordenadores y a la explosión de datos. Estas técnicas de aprendizaje automático son una parte esencial del fenómeno conocido como Big Data.

Los algoritmos de Machine Learning se pueden clasificar en tres categorías principales, siendo las dos primeras las más predominantes:

- Aprendizaje supervisado: Estos algoritmos se basan en un conjunto de datos previamente etiquetados, lo que les permite tomar decisiones o hacer predicciones. Por ejemplo, un filtro de spam que clasifica correos electrónicos como spam o no spam basándose en patrones aprendidos del historial de mensajes.
- Aprendizaje no supervisado: Estos algoritmos no requieren de un conocimiento previo y se enfrentan a conjuntos de datos sin etiquetar, con el objetivo de encontrar patrones significativos para organizarlos. Por ejemplo, en el campo del marketing, se utilizan para analizar grandes cantidades de datos de redes sociales y desarrollar campañas publicitarias altamente segmentadas.
- Aprendizaje por refuerzo: Este tipo de aprendizaje implica que un algoritmo aprenda de su propia experiencia, tomando decisiones y ajustando su comportamiento en función de las recompensas recibidas por decisiones correctas. Actualmente, se aplica en áreas como el reconocimiento facial, diagnósticos médicos y clasificación de secuencias de ADN.

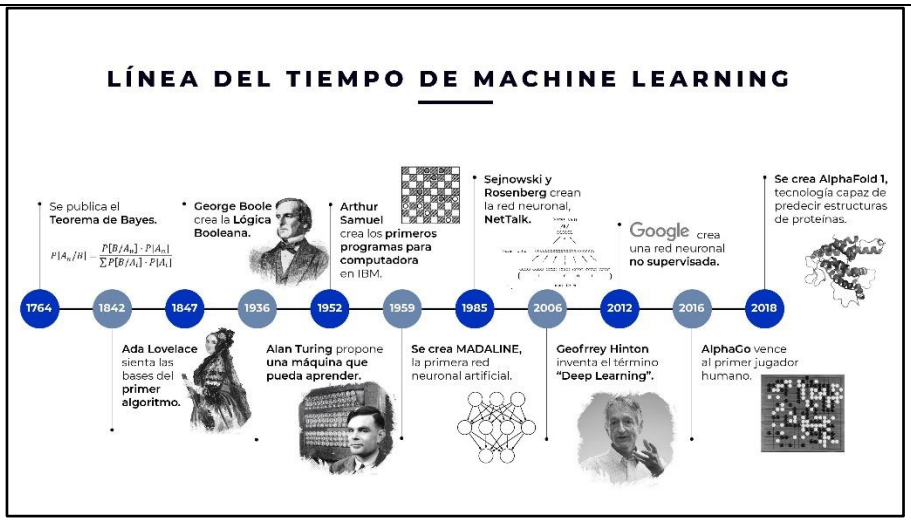

Figura N° 02: Línea de Tiempo de Machine Learning.

Aplicaciones prácticas del Machine Learning.

El Machine Learning se ha convertido en un pilar fundamental de la transformación digital, y su aplicación se extiende a diversos campos, entre los que destacan:

Recomendaciones: Este enfoque permite ofrecer recomendaciones personalizadas de productos en plataformas de comercio electrónico o sugerir música en servicios de streaming. Al analizar el historial de compras o reproducciones de un usuario y compararlo con otros con gustos similares, se pueden hacer recomendaciones más precisas.

Vehículos inteligentes: Según el informe "Automotive 2025: industry without borders" de IBM, se espera que para 2025 los vehículos inteligentes estén presentes en las carreteras. Gracias al Machine Learning, estos vehículos podrán ajustar sus configuraciones internas, como la temperatura o la música, según las preferencias del conductor, e incluso tomar decisiones de conducción autónoma para adaptarse al entorno.

Redes sociales: Plataformas como Twitter utilizan algoritmos de Machine Learning para combatir el spam, mientras que Facebook los emplea para detectar noticias falsas y contenido inapropiado en las transmisiones en vivo, bloqueándolos automáticamente.

Procesamiento de Lenguaje Natural (PLN): Los asistentes virtuales como Alexa o Siri hacen uso del PLN para traducir instantáneamente entre idiomas, reconocer la voz del usuario y analizar sus emociones. Además, el PLN se utiliza para tareas complejas como simplificar la jerga legal en contratos o ayudar a los abogados a organizar grandes cantidades de información relacionada con un caso.

Búsquedas: Los motores de búsqueda emplean el aprendizaje automático para optimizar los resultados en función de su relevancia, midiendo esta mediante los clics del usuario.

Medicina: En el campo de la medicina, se utiliza el Machine Learning para detectar el cáncer de mama de manera más temprana, lo que aumenta las posibilidades de curación. También se aplica eficazmente en la detección precoz de neumonía y enfermedades oculares que pueden causar ceguera.

Ciberseguridad: Los sistemas de seguridad informática, como los antivirus y los motores de detección de malware, emplean el aprendizaje automático para mejorar el escaneo, acelerar la detección y reconocer anomalías con mayor precisión.

Figura N° 03: Aplicaciones prácticas del Machine Learning.

- **¿Por qué ahora?**

  El Machine Learning ha experimentado un aumento significativo en su aplicación y relevancia en los últimos años debido a varios factores clave:

  - Avances en la tecnología: El aumento en la capacidad de procesamiento de las computadoras y el acceso a grandes cantidades de datos han hecho que el Machine Learning sea más práctico y efectivo. Los algoritmos de Machine Learning pueden ahora procesar conjuntos de datos masivos de manera más rápida y eficiente que nunca.

  - Big Data: La explosión de datos generados por dispositivos conectados, redes sociales, transacciones en línea y otras fuentes ha proporcionado a los algoritmos de Machine Learning una abundancia de información para analizar y aprender. Esto ha permitido el desarrollo de modelos más precisos y sofisticados.

- Mejoras en los algoritmos: Se han desarrollado nuevos algoritmos de Machine Learning y se han mejorado los existentes, lo que ha llevado a un mejor rendimiento y precisión en una variedad de aplicaciones. Además, se han hecho avances en técnicas como el aprendizaje profundo, que ha demostrado ser especialmente efectivo en la clasificación de imágenes y el procesamiento del lenguaje natural.

- Aplicaciones prácticas exitosas: La adopción exitosa del Machine Learning en una variedad de industrias, como el comercio electrónico, la salud, la automoción y la ciberseguridad, ha demostrado su valor en la resolución de problemas complejos y en la mejora de procesos empresariales.

- Disponibilidad de herramientas y plataformas: Se han desarrollado una amplia gama de herramientas y plataformas de Machine Learning que facilitan su implementación y uso por parte de empresas y desarrolladores. Esto ha reducido las barreras de entrada y ha democratizado el acceso al Machine Learning.

En resumen, el Machine Learning ha experimentado un crecimiento significativo debido a la convergencia de avances tecnológicos, disponibilidad de datos, mejoras en los algoritmos y la demostración de su eficacia en aplicaciones del mundo real. Esto ha llevado a una adopción más amplia en diversas industrias y ha posicionado al Machine Learning como una herramienta poderosa para la innovación y el progreso.

Figura N° 04: Beneficios de Machine Learning.

- **Flujo de trabajo de aprendizaje automático.**

  AI Platform facilita muchas etapas del flujo de trabajo del Machine Learning (ML). Este documento ofrece una visión introductoria del proceso general de ML y explica cómo cada servicio de AI Platform se aplica en este proceso.

  Para obtener una comprensión inicial de los servicios, consulta la Descripción general técnica de AI Platform.

  Una visión general del Machine Learning.

  El Machine Learning (ML) es un subcampo de la Inteligencia Artificial (IA). Su objetivo es enseñar a las computadoras a aprender a partir de los datos proporcionados. En lugar de escribir código que instruya a la computadora sobre qué acciones realizar, tu código proporciona un algoritmo que se ajusta según ejemplos de comportamiento esperado. El resultado final, que consiste en el algoritmo y los parámetros aprendidos asociados, se denomina modelo entrenado.

  El flujo de trabajo del ML

  El siguiente diagrama ofrece una descripción general de alto nivel de las etapas en el flujo de trabajo del ML. Los cuadros azules indican dónde AI Platform proporciona API y servicios gestionados.

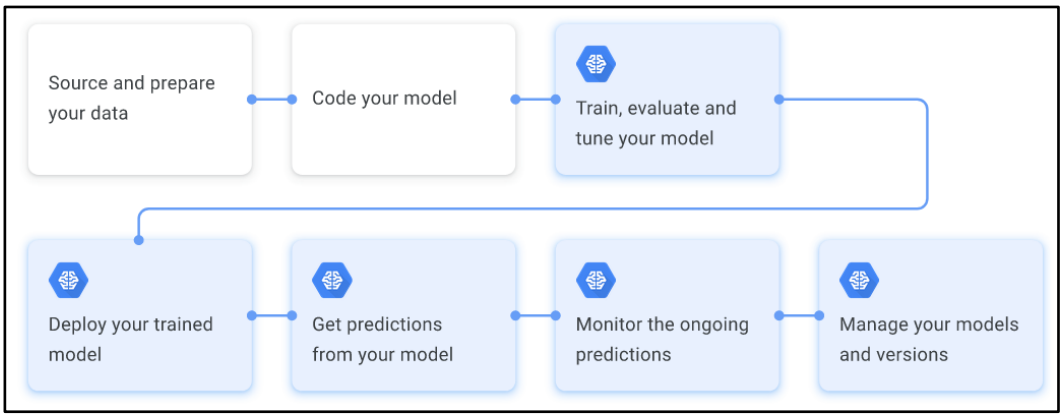

Figura N° 05: Flujo de trabajo del AA.

Para desarrollar y gestionar un modelo destinado a la producción, es necesario completar una serie de fases fundamentales:

- Adquisición y preparación de datos: Obtener y preparar los datos necesarios para el modelo.
- Desarrollo del modelo: Crear el modelo utilizando técnicas adecuadas de machine learning.
- Entrenamiento del modelo de machine learning con los datos proporcionados:
- Entrenar el modelo con los datos disponibles.
- Evaluar la precisión del modelo mediante pruebas.
- Ajustar los hiperparámetros del modelo según sea necesario.

Implementación del modelo entrenado: Desplegar el modelo para su uso en un entorno de producción.

- Solicitud de predicciones al modelo implementado:
- Obtener predicciones del modelo en línea.
- Obtener predicciones del modelo en lotes.

Supervisión continua de las predicciones generadas por el modelo.

- Gestión y mantenimiento de los modelos y sus versiones: Administrar y mantener los modelos y sus diferentes versiones a lo largo del tiempo.

Es importante tener en cuenta que estas etapas son iterativas, lo que significa que puede ser necesario volver atrás en el proceso y reevaluar ciertos pasos en cualquier momento. En las siguientes secciones de esta página, se detallan cada una de estas etapas.

Antes de iniciar, es crucial evaluar el problema que se desea abordar mediante el aprendizaje automático (AA). Se debe dedicar tiempo a analizar detenidamente el problema y plantearse las siguientes preguntas:

- ¿Está claramente definido el problema que se pretende resolver?
  Es esencial tener una comprensión clara del objetivo que se persigue con el modelo y la información específica que se desea obtener de él. Definir con precisión el problema facilita el proceso de diseño y desarrollo del modelo.

- ¿Es el aprendizaje automático la mejor solución para el problema planteado?
  El aprendizaje automático, particularmente el estilo supervisado descrito en esta documentación, es adecuado para ciertos tipos de problemas. Es fundamental evaluar si el aprendizaje automático es la mejor opción considerando la naturaleza del problema y la disponibilidad de datos suficientes para entrenar el modelo. Se debe tener en cuenta que la cantidad y calidad de los datos disponibles son aspectos cruciales para el éxito del modelo.

- ¿Cómo se puede determinar si el modelo ha sido desarrollado de manera satisfactoria?
  Determinar cuándo se considera completada la fase de desarrollo del modelo es uno de los desafíos principales en el proceso de aprendizaje automático. Es importante establecer criterios claros para evaluar el rendimiento del modelo, como el nivel de precisión requerido para cumplir con los objetivos del proyecto. También se deben considerar las implicaciones de los posibles errores y cómo afectarían los resultados finales.

Consigue y prepara tus datos.

Obtener y preparar los datos es un paso fundamental en el proceso de desarrollo de modelos de aprendizaje automático. Aquí hay algunas pautas para llevar a cabo esta etapa:

Adquisición y preparación de datos.
Es esencial contar con un conjunto de datos de entrenamiento amplio que contenga la variable objetivo que se desea predecir, junto con otras características relevantes. Por ejemplo, si se quiere predecir el precio de venta de una casa, se necesitaría un conjunto de datos que incluya información detallada sobre las propiedades en un área específica, incluyendo el precio de venta.

Análisis de datos.
Una vez que se obtienen los datos, es importante analizarlos para comprender su estructura y contenido. Esto puede incluir la consolidación de datos de diversas fuentes, visualización de datos para identificar tendencias y patrones, y uso de herramientas y lenguajes centrados en datos para descubrir insights. También es crucial identificar las características clave que se utilizarán en el modelo y limpiar los datos para corregir posibles errores o valores anómalos.

Preprocesamiento de datos.
En la etapa de preprocesamiento de datos, se transforman los datos en un formato adecuado para el entrenamiento del modelo. Esto puede implicar normalizar datos numéricos, aplicar reglas de formato, reducir redundancias y representar datos de texto numéricamente. Google Cloud ofrece diversas herramientas y servicios para facilitar la exploración y preparación de datos, como las bibliotecas de procesamiento previo de TensorFlow, la implementación de canalizaciones de scikit-learn en AI Platform, y servicios como BigQuery, Dataproc, Dataflow y Dataprep.

Es importante destacar que la adquisición y preparación de datos es un proceso iterativo y a menudo requiere de experimentación y ajustes continuos para garantizar la calidad y la adecuación de los datos al modelo.

TAREA N°02

Utiliza recursividad en la programación.

Árboles y recursividad.

En el ámbito de la programación, el concepto de estructuras de datos se refiere a la organización específica que se aplica a los datos dentro de un sistema informático, con el propósito de maximizar su eficiencia en el uso. Es crucial señalar que la selección de una estructura de datos particular depende del tipo de aplicación o recurso que se vaya a utilizar, ya que cada una se adapta de manera diferente según el contexto y los objetivos específicos que se persigan.

Hasta los años setenta, la programación era relativamente simple y no se prestaba mucha atención a la calidad del código. Sin embargo, durante esa década, un grupo de ingenieros desarrolló un conjunto de técnicas conocidas como ingeniería de software. Estas técnicas marcaron un punto de inflexión en la forma en que se desarrollaban los programas, introduciendo estándares y prácticas para garantizar la calidad del software.

Las estructuras de datos desempeñaron un papel fundamental en esta evolución de la programación. Originalmente, los programas carecían de organización, estructuras de control y datos complejos. Sin embargo, a medida que los lenguajes de programación evolucionaron, los programadores pudieron crear estructuras de datos más elaboradas, lo que permitió el desarrollo de programas más complejos diseñados para satisfacer las necesidades de los usuarios.

A lo largo de este documento se explorarán principalmente dos tipos de estructuras de datos estrechamente relacionadas: la Recursividad y los Árboles Binarios. Aunque no son las estructuras más simples, son de gran importancia en el ámbito de las Estructuras de Datos y están interconectadas de manera significativa, como se demostrará en este artículo. Es ampliamente conocido que estos temas pueden ser altamente complejos según el nivel de profundidad deseado; sin embargo, el objetivo es proporcionar al lector una comprensión sólida que le permita comenzar a programar estas estructuras. Para lograr este propósito, se emplearán definiciones claras, ejemplos en C++, recursos visuales y la implementación de casos relevantes que puedan ser útiles al abordar estos temas. Además, se hará referencia a otras estructuras de datos para enriquecer la comprensión general.

- **Árboles binarios.**
  Un árbol se caracteriza como una estructura no lineal donde cada nodo puede estar conectado con uno o varios nodos, y comúnmente se describe de manera recursiva

como una estructura compuesta por un dato y varios subárboles. Aunque la definición puede parecer simple, las características inherentes a los árboles presentan un grado de complejidad considerable.

Código en Java

```
class Arbol{
private NodoArbol raiz;

//contruir un arbol vacio
public Arbol()
{
raiz = null;
}

//insertar un nuevo nodo en el arbol de busqueda binaria
public synchronized void insertarNodo(int valorInsertar)
{
if(raiz == null)
raiz = new NodoArbol(valorInsertar); //crea nodo raiz

else
raiz.insertar(valorInsertar); // llama al metodo insertar
}
```

Partes de los árboles (Estructura de Árbol)

Raíz: Cada árbol, excepto aquellos que están vacíos, tiene un único nodo raíz. Todos los otros nodos en el árbol se derivan de esta raíz, lo que significa que este nodo no tiene padre y no es hijo de ningún otro nodo.

Nodo: Los nodos son los vértices o elementos individuales del árbol.

- Los nodos con uno o dos subárboles se conocen como nodos internos.
- Los nodos que comparten el mismo padre se denominan hermanos.
- Cada nodo se encuentra en un nivel determinado, que se define por su distancia desde la raíz.
- La profundidad de un árbol se refiere al número máximo de nodos en un camino desde la raíz hasta una hoja, equivalente al nivel más alto más uno.

Nodo terminal: Se refiere a un nodo que se encuentra en el nivel más bajo del árbol y no tiene ningún subárbol.

El árbol binario es una estructura recursiva de conjunto finito de cero o más nodos tales que:

- Existe un nodo denominado raíz del árbol.
- Cada nodo puede tener 0,1 ó 2 subárboles (Izquierdo y derecho).

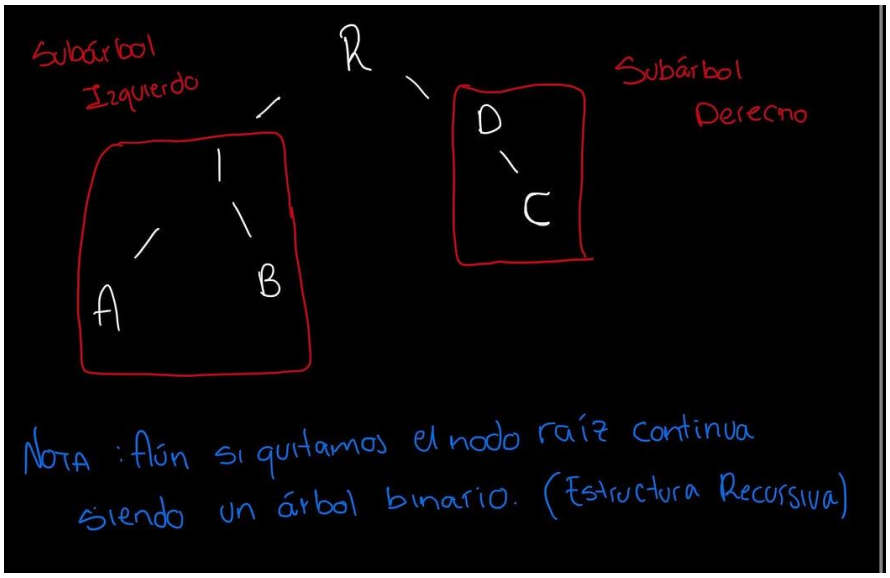

Figura N° 06: Árbol Binario.

¿Qué hace que un árbol binario tenga una estructura recursiva?

La razón principal de considerar un árbol binario como una estructura recursiva es que cada nodo del árbol, junto con todos sus descendientes, conservando el orden original, también forma un árbol o subárbol dentro del árbol principal. Esta característica permite definiciones simples de árboles, las cuales son más adecuadas desde la perspectiva de la teoría de tipos abstractos de datos, y se ajustan a cada uso específico de la noción de árbol.

Tipos de árboles binarios:

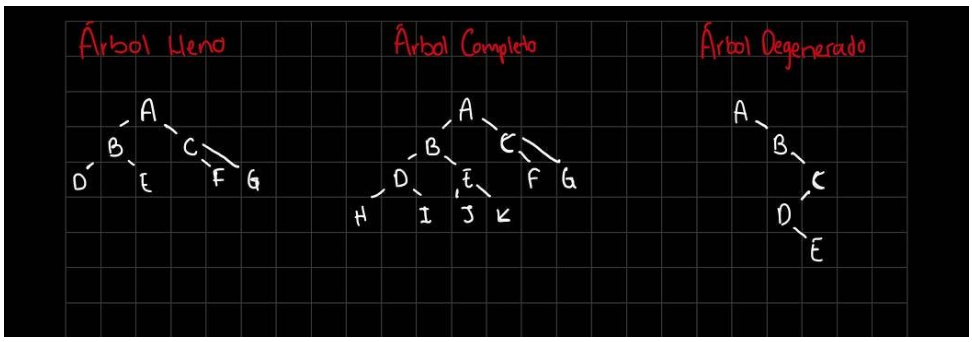

Figura N° 07: Tipos de árboles binarios.

- Árbol Lleno: Se refiere a aquellos árboles que tienen todos sus nodos ocupados hasta el último nivel.

- Árbol Completo: A diferencia de un árbol lleno, aunque también están completamente ocupados, no todos sus nodos llegan hasta el último nivel. Por lo general, el subárbol izquierdo suele tener más nodos que el subárbol derecho.

- Árbol Degenerado: Este tipo de árbol desafía las convenciones de un árbol típico y se asemeja más a una lista enlazada en términos de estructura y organización.

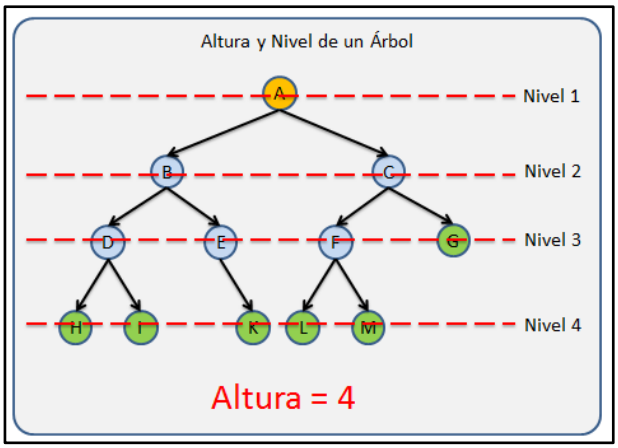

Figura N° 08: Altura y nivel de un árbol.

➢ **Recursividad.**

Un objeto se considera recursivo cuando su definición incluye una referencia a sí mismo, lo que implica que está involucrado en su propia descripción. La recursividad, por su parte, es la capacidad de un subprograma o rutina para invocarse a sí mismo. Al emplear esta técnica, la resolución de un problema se simplifica al abordar una versión más pequeña pero esencialmente similar del mismo.

Es importante tener en cuenta que no todas las funciones pueden invocarse a sí mismas; deben estar específicamente diseñadas con ese propósito para ser recursivas. De lo contrario, podrían resultar en bucles infinitos o en la terminación inadecuada del programa.

Ventajas:

- Algunos problemas pueden ser más fáciles de modelar e implementar utilizando la recursividad.

Desventajas:

- Se requiere la creación de múltiples variables, lo que puede causar problemas de memoria.
- En general, una función recursiva tiende a tomar más tiempo en ejecutarse que una función iterativa.

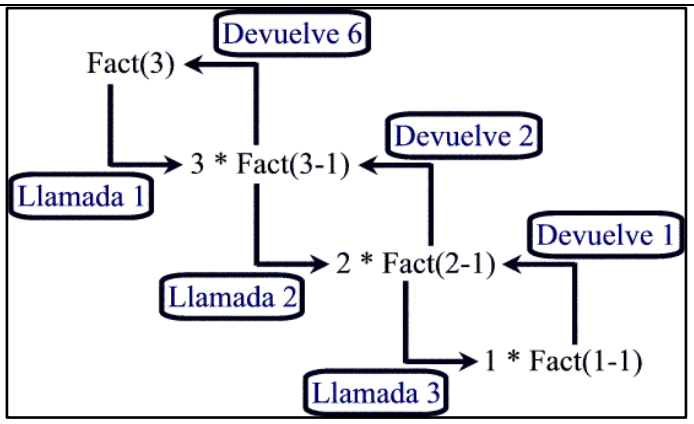

Figura N° 09: Diagrama Recursividad Factorial.

Empecemos por explicar qué es una factorial:

La factorial de un número entero positivo n, denotado como n!, se define como el producto de todos los números enteros positivos desde 1 (es decir, los números naturales) hasta n.

Por lo tanto, la regla o fórmula para una función factorial sería:

$n! = n \times (n-1)!$

lo que significa “la factorial de cualquier número es: el número por la factorial de (1 menos que el número”, por tanto 10! = 10 × 9!

```
12  //Ejemplo para funcion factorial
13  int factorial(int n)
14  {
15      if (n==0) //Caso base
16      {
17          n=1;
18      }else //Caso general
19      {
20          n=*factorial(n-1);
21      }
22      return n;
23  }
```

Figura N° 10: Recursividad, Caso Factorial en Java.

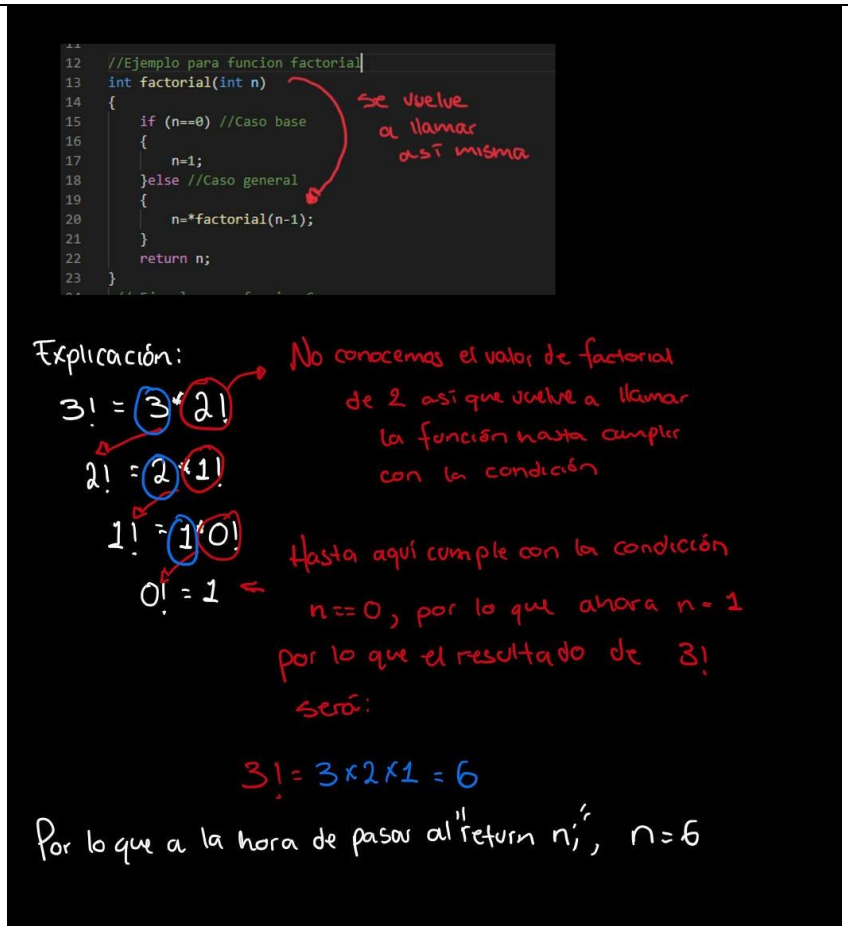

Figura N° 11: Explicación Caso Factorial, Recursividad.

- **Árbol transversal.**

  Un árbol transversal, también conocido como "árbol de recorrido", es una estructura de datos fundamental en informática y ciencias de la computación. Es una forma de organizar datos de manera jerárquica, donde cada nodo puede tener cero o más nodos hijos, y se utiliza comúnmente para representar relaciones jerárquicas entre elementos.

  Características de un Árbol Transversal:

  - Jerarquía: Los nodos están organizados en niveles, donde cada nivel representa un nivel de jerarquía en la estructura de datos.

  - Nodos: Cada elemento en el árbol se denomina "nodo". Cada nodo tiene un valor y puede tener cero o más nodos hijos.

  - Raíz: Es el nodo superior del árbol, desde el cual se derivan todos los demás nodos.

  - Hojas: Son los nodos que no tienen hijos y están en el nivel más bajo del árbol.

- Subárbol: Cada nodo en un árbol puede ser considerado como la raíz de un subárbol, que incluye todos los nodos descendientes de ese nodo.

- Recorrido: Se refiere al proceso de visitar cada nodo del árbol exactamente una vez. Hay varios métodos para recorrer un árbol, como el recorrido en preorden, en orden y en postorden.

Aplicaciones de los Árboles Transversales:

- Estructura de Datos: Los árboles transversales se utilizan para implementar estructuras de datos como los árboles binarios de búsqueda, los árboles AVL y los árboles B.

- Análisis de Algoritmos: Se utilizan para modelar algoritmos de búsqueda y recorrido, así como para representar relaciones jerárquicas en estructuras de datos complejas.

- Bases de Datos: En las bases de datos, se utilizan árboles transversales para representar índices, facilitando la búsqueda y recuperación eficiente de datos.

- Compiladores: Los árboles transversales se utilizan en la fase de análisis sintáctico de un compilador para representar la estructura de un programa fuente de manera jerárquica.

- Graficación y Visualización de Datos: Se utilizan para organizar y visualizar datos en aplicaciones gráficas, como diagramas de organigramas y representaciones de estructuras jerárquicas.

Operaciones comunes en Árboles Transversales:

- Inserción: Agregar un nuevo nodo al árbol.

- Eliminación: Eliminar un nodo existente del árbol.

- Búsqueda: Buscar un nodo con un valor específico en el árbol.

- Recorrido: Visitar cada nodo del árbol en un orden específico, como preorden, en orden y postorden.

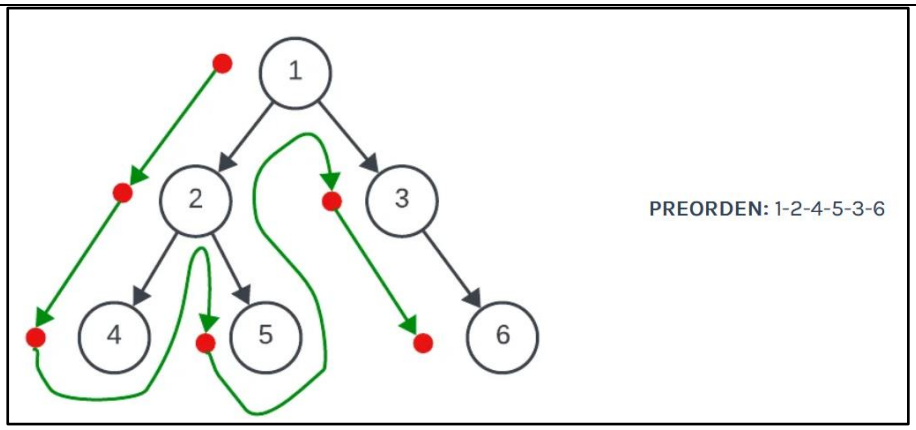

Figura N° 12: Recorrido PreOrden.

Algoritmo en Java:

```
public synchronized void recorridoPreorden()
{
ayudantePreorden(raiz);
}
//metodo recursivo para recorrido en preorden
private void ayudantePreorden(NodoArbol nodo)
{
if (nodo == null)
return;
System.out.print(nodo.datos + " "); // mostrar datos del nodo
ayudantePreorden(nodo.nodoizquierdo); //recorre subarbol izquierdo
ayudantePreorden(nodo.nododerecho); //recorre subarbol derecho
}
```

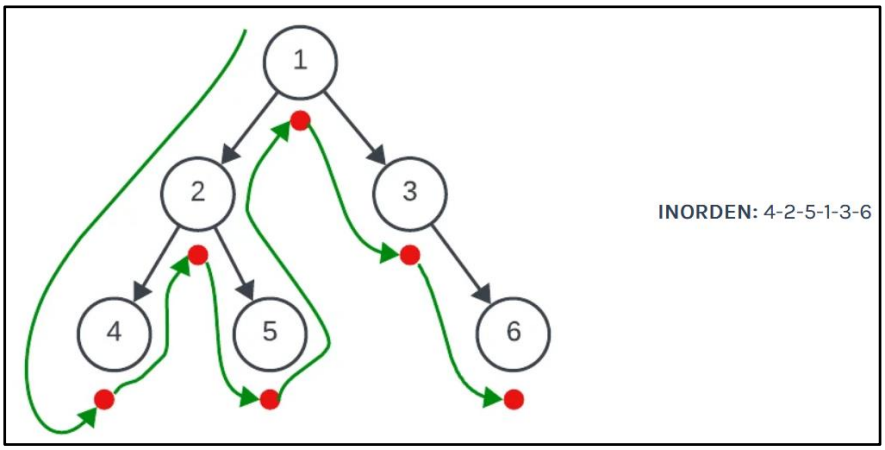

Figura N° 12: Recorrido InOrden.

Algoritmo en Java:

```
public synchronized void recorridoInorden()
{
ayudanteInorden(raiz);
}
// metodo recursivo para recorrido inorden

private void ayudanteInorden(NodoArbol nodo)
{
if (nodo == null)
return;
ayudanteInorden(nodo.nodoizquierdo);
System.out.print(nodo.datos + " ");
ayudanteInorden(nodo.nododerecho);
}
```

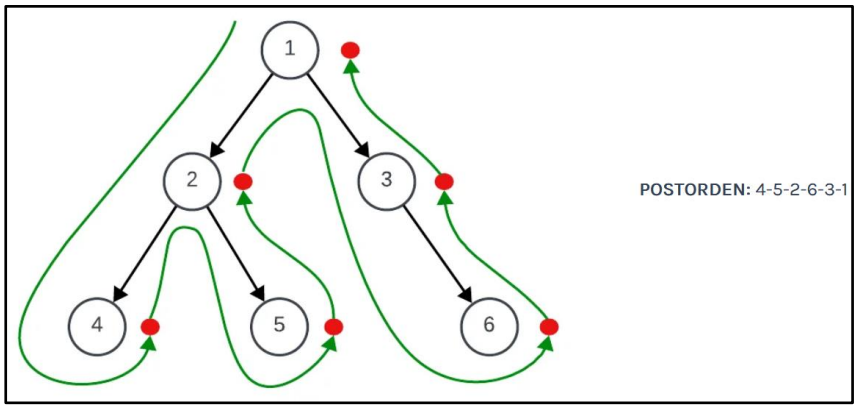

Figura N° 13: Recorrido PreOrden.

Algoritmo en Java:

```
public synchronized void recorridoPosorden()
{
ayudantePosorden(raiz);
}
//metodo recursivo para recorrido posorden
private void ayudantePosorden(NodoArbol nodo)
{
if (nodo == null)
return;
ayudantePosorden(nodo.nodoizquierdo);
ayudantePosorden(nodo.nododerecho);
System.out.print(nodo.datos + " ");
}
}//fin clase arbol
```

- Altura del Árbol: Determinar la altura del árbol, es decir, la longitud del camino más largo desde la raíz hasta una hoja.

- Balanceo: Mantener el árbol equilibrado para garantizar tiempos de búsqueda y recuperación eficientes.

Los árboles transversales son una herramienta poderosa y versátil en informática, utilizada en una variedad de aplicaciones y contextos para organizar y manipular datos de manera eficiente y estructurada.

- **Juego Sí/No.**

El "Juego Sí/No" es un juego en el que una persona piensa en un objeto, animal, lugar o persona, y los otros participantes intentan adivinar qué es haciendo preguntas que solo pueden responderse con "sí" o "no". El objetivo es adivinar la respuesta correcta con el menor número de preguntas posible.

Reglas del Juego:

- Elección de un Objeto: Uno de los participantes selecciona mentalmente un objeto, animal, lugar o persona sin revelar su elección a los demás jugadores.

- Preguntas Sí/No: Los demás jugadores comienzan a hacer preguntas al jugador que seleccionó el objeto. Estas preguntas solo pueden responderse con "sí" o "no".

- Adivinanzas: Los jugadores intentan adivinar qué es el objeto mediante la lógica y la información proporcionada por las respuestas "sí" o "no".

- Respuestas Honestas: El jugador que seleccionó el objeto debe responder honestamente con un "sí" o "no" a cada pregunta realizada por los otros jugadores.

- Límite de Preguntas: Los jugadores pueden acordar un límite de preguntas antes de comenzar el juego para hacerlo más desafiante.

- Ganador: El jugador que adivina correctamente el objeto en menos preguntas gana el juego.

Estrategias:

- Preguntas Estratégicas: Los jugadores deben hacer preguntas que reduzcan rápidamente el número de posibles opciones. Por ejemplo, preguntar si el objeto es un animal, si es de color verde, si es un objeto comúnmente encontrado en una casa, etc.

- Descarte de Opciones: A medida que se recibe información sobre el objeto, los jugadores pueden ir descartando opciones que no coincidan con las respuestas recibidas.

- Mantener un Registro: Es útil que un jugador mantenga un registro de las preguntas realizadas y las respuestas recibidas para ayudar en el proceso de deducción.

El "Juego Sí/No" es una actividad divertida que promueve la lógica, la comunicación y el trabajo en equipo, además de ser una excelente manera de pasar el tiempo con amigos y familiares.

El juego Sí/No se puede jugar utilizando la estructura de un árbol de decisión, que es una representación gráfica de un conjunto de preguntas y opciones que se ramifican en función de las respuestas "sí" o "no". Aquí hay una explicación de cómo se puede jugar el juego Sí/No con un árbol de decisiones:

Paso 1: Creación del Árbol de Decisión

- Raíz del Árbol: La raíz del árbol representa la pregunta inicial que se hace al jugador que seleccionó el objeto. Por ejemplo, "¿Es un animal?"

- Nodos Intermedios: Cada nodo intermedio representa una pregunta adicional basada en la respuesta anterior. Por ejemplo, si la respuesta a la primera pregunta es "sí", el siguiente nodo podría preguntar "¿Vive en el agua?".

- Hojas del Árbol: Las hojas del árbol representan las posibles respuestas al juego. Por ejemplo, si el objeto es un pez, una hoja del árbol podría indicar "pez".

Paso 2: Juego

- Inicio del Juego: Los jugadores comienzan en la raíz del árbol, donde se les hace la primera pregunta.

- Respuestas y Movimiento: Dependiendo de la respuesta del jugador ("sí" o "no"), se mueven a los nodos correspondientes en el árbol y se les hace la siguiente pregunta.

- Adivinación: Los jugadores continúan haciendo preguntas y moviéndose a través del árbol hasta llegar a una hoja, que representa la adivinanza final. Intentan adivinar el objeto basándose en las preguntas y respuestas anteriores.

Paso 3: Ganador

- El jugador que adivina correctamente el objeto en menos preguntas gana el juego.

- Estrategias con Árboles de Decisión:
- Preguntas Claves: Los jugadores deben formular preguntas estratégicas que reduzcan rápidamente el número de opciones.

- Exploración Eficiente: Los jugadores deben moverse de manera eficiente a través del árbol, explorando ramas que les brinden la mayor información posible.

- Registro de Preguntas: Es útil llevar un registro de las preguntas realizadas y las respuestas recibidas para evitar hacer preguntas repetidas y mantener un seguimiento claro de las opciones restantes.

El uso de un árbol de decisiones en el juego Sí/No agrega una dimensión visual y estructurada al juego, lo que puede hacerlo más interesante y desafiante para los jugadores.

Programación de Juego utilizando árboles.

```
import java.util.Scanner;

/**
 * Clase para jugar a adivinar animales
 * @author Amparo L�pez Gaona
 * @version 1a. ed.
 */
class AdivinaAnimal {
    private static NodoArbol raiz;
    private Scanner in;

    /**
     * Constructor por omision. Crea el nodo raiz con un elefante
     */
    public AdivinaAnimal (){
        raiz = new NodoArbol("Elefante");
        in = new Scanner(System.in);
    }

    /**
     * Metodo para jugar a adivinar animales
```

```
   */
public void jugar () {
  NodoArbol nodo = raiz;

  while (nodo != null) {
    if (nodo.izquierda != null) {   //Hay una pregunta
      System.out.println(nodo.valor);
      nodo = (respuesta()) ? nodo.izquierda : nodo.derecha;
    } else {  // Se tiene un animal
      System.out.println("Ya sé, es un(a) " + nodo.valor);
      if (respuesta())
        System.out.println("Gane :) !!");
      else
        animalNuevo (nodo);
      nodo = null;
      return;
    }
  }
}

/*
 * Método privado para leer una respuesta del usuario sólo puede ser sí o no
 * @return boolean -- true si la respuesta es si y false en otro caso.
 */
private boolean respuesta () {

    while (true) {
        String resp = in.nextLine().toLowerCase().trim();

        if (resp.equals("si")) return true;
        if (resp.equals("no")) return false;
        System.out.println("La respuesta debe ser si o no");
    }
}

/**
 * Método para dar de alta un nuevo animal
 * @param nodo -- nodo del cual se va a colgar el animal
 */
private void animalNuevo (NodoArbol nodo) {
  String animalNodo = (String) nodo.valor;
  System.out.println("Cuál es tu animal?");
  String nuevoA = in.nextLine();
```

```
        System.out.println("Qué pregunta con respuesta si/no puedo hacer" +
                " para poder decir que es un(a) " + nuevoA);
        String pregunta = in.nextLine();
        NodoArbol nodo1 = new NodoArbol(animalNodo);
        NodoArbol nodo2 = new NodoArbol(nuevoA);
        System.out.println("Para un(a) " + nuevoA + " la respuesta es si/no?");
        nodo.valor = ""+pregunta+"?";
        if (respuesta()) {
            nodo.izquierda = nodo2;
            nodo.derecha = nodo1;
        } else {
            nodo.izquierda = nodo1;
            nodo.derecha = nodo2;
        }
    }

    public static void main (String [] pps) {
        System.out.println("Juguemos a adivinar animales");
        boolean otroJuego = true;
        AdivinaAnimal juego = new AdivinaAnimal();

        do {
            juego.jugar();
            System.out.println("Jugamos otra vez?");
            otroJuego = juego.respuesta();
        } while (otroJuego);
    }
}
```

TAREA N°03

Construye algoritmo de árbol de decisiones.

Entropía y el algoritmo ID3:
La entropía y el algoritmo ID3 están estrechamente relacionados en el contexto del aprendizaje automático y la construcción de árboles de decisión. Aquí tienes una explicación detallada sobre ambos conceptos:

Entropía:

La entropía es una medida de incertidumbre en un conjunto de datos. En el contexto del aprendizaje automático y los árboles de decisión, la entropía se utiliza para evaluar la homogeneidad de un conjunto de ejemplos.

Fórmula de la Entropía: En un conjunto de datos binario, la fórmula de la entropía se expresa como:

$$H(S) = - p_+ \log_2(p_+) – p_- \log_2(p_-)$$

Donde p_+ es la proporción de ejemplos positivos en el conjunto de datos y p_- es la proporción de ejemplos negativos. La entropía alcanza su máximo valor cuando la proporción de ejemplos positivos y negativos es 0.5 (máxima incertidumbre) y su mínimo valor cuando uno de los dos tipos de ejemplos domina completamente (mínima incertidumbre).

Algoritmo ID3 (Iterative Dichotomiser 3):

El algoritmo ID3 es un algoritmo clásico de aprendizaje automático utilizado para construir árboles de decisión a partir de un conjunto de datos etiquetados. Utiliza la entropía para decidir cómo dividir el conjunto de datos en cada paso de construcción del árbol.

Proceso de Construcción:

- Selección del Atributo: En cada paso, el algoritmo ID3 selecciona el atributo que mejor divide el conjunto de datos en subconjuntos más homogéneos en términos de entropía.

- División del Conjunto de Datos: El algoritmo divide el conjunto de datos en subconjuntos basados en los valores del atributo seleccionado.

- Construcción Recursiva: Este proceso se repite de forma recursiva en cada subconjunto hasta que se alcanza un criterio de parada, como la pureza de los nodos hoja o una profundidad máxima del árbol.

Criterio de Selección del Atributo: El algoritmo ID3 utiliza la ganancia de información para seleccionar el mejor atributo en cada paso. La ganancia de información se calcula como la diferencia entre la entropía del conjunto de datos original y la entropía ponderada de los subconjuntos resultantes después de la división por el atributo.

Consideraciones:

- El algoritmo ID3 es eficaz para conjuntos de datos pequeños y medianos con atributos categóricos.
- Tiene tendencia a sobreajustarse a los datos si no se utiliza una técnica de poda adecuada.

En resumen, el algoritmo ID3 utiliza la entropía como medida de incertidumbre para construir árboles de decisión de manera recursiva, seleccionando en cada paso el atributo que maximice la ganancia de información.

➢ **Algoritmos de árboles de decisión.**
Los algoritmos de árboles de decisión son métodos de aprendizaje automático utilizados para modelar y predecir decisiones basadas en una serie de condiciones. Aquí tienes una descripción de algunos de los algoritmos más comunes:

- ID3 (Iterative Dichotomiser 3):
  Fue uno de los primeros algoritmos desarrollados para la construcción de árboles de decisión.
  Utiliza la ganancia de información y la entropía para seleccionar los atributos y construir el árbol de decisión.
  Es adecuado para datos con atributos categóricos.

```
import java.util.ArrayList;
import java.util.HashMap;
class DecisionTree {
    private class Node {
        String attribute;
        HashMap<String, Node> children;
        Node(String attribute) {
            this.attribute = attribute;
            this.children = new HashMap<>();
        }
    }
```

```
    private Node root;
    public DecisionTree() {
        this.root = null;
    }
    public void buildTree(ArrayList<String[]> data, ArrayList<String> attributes) {
        this.root = buildTreeHelper(data, attributes);
    }
    private Node buildTreeHelper(ArrayList<String[]> data, ArrayList<String>
attributes) {
        Node node = new Node("");

        // If all examples are of the same class, return a leaf node
        if (sameClass(data)) {
            node.attribute = data.get(0)[data.get(0).length - 1];
            return node;
        }
        // If attributes is empty, return a leaf node with the most common class
        if (attributes.isEmpty()) {
            node.attribute = getMostCommonClass(data);
            return node;
        }
        // Select the best attribute
        String bestAttribute = selectBestAttribute(data, attributes);
        node.attribute = bestAttribute;
        // Remove the best attribute from the list
        ArrayList<String> remainingAttributes = new ArrayList<>(attributes);
        remainingAttributes.remove(bestAttribute);
        // Split the data based on the best attribute and recursively build subtrees
        for (String value : getAttributeValues(data, bestAttribute)) {
            ArrayList<String[]> subset = getSubset(data, bestAttribute, value);
            if (subset.isEmpty()) {
                Node leaf = new Node(getMostCommonClass(data));
                node.children.put(value, leaf);
            } else {
                node.children.put(value,                    buildTreeHelper(subset,
remainingAttributes));
            }
        }
        return node;
    }
    private boolean sameClass(ArrayList<String[]> data) {
        String classValue = data.get(0)[data.get(0).length - 1];
        for (String[] example : data) {
```

```
            if (!example[example.length - 1].equals(classValue)) {
                return false;
            }
        }
        return true;
    }

    private String getMostCommonClass(ArrayList<String[]> data) {
        HashMap<String, Integer> classCounts = new HashMap<>();
        for (String[] example : data) {
            String classValue = example[example.length - 1];
            classCounts.put(classValue,  classCounts.getOrDefault(classValue,  0)
+ 1);
        }
        String mostCommonClass = "";
        int maxCount = 0;
        for (String classValue : classCounts.keySet()) {
            int count = classCounts.get(classValue);
            if (count > maxCount) {
                maxCount = count;
                mostCommonClass = classValue;
            }
        }
        return mostCommonClass;
    }

    private String selectBestAttribute(ArrayList<String[]> data, ArrayList<String>
attributes) {
        double maxInfoGain = 0;
        String bestAttribute = "";
        for (String attribute : attributes) {
            double infoGain = calculateInfoGain(data, attribute);
            if (infoGain > maxInfoGain) {
                maxInfoGain = infoGain;
                bestAttribute = attribute;
            }
        }
        return bestAttribute;
    }

    private double calculateInfoGain(ArrayList<String[]> data, String attribute) {
        // Implementation of information gain calculation
        // (e.g., using entropy or Gini impurity)
        return 0.0;
```

```
    }

    private ArrayList<String> getAttributeValues(ArrayList<String[]> data, String attribute) {
        // Retrieve all unique values of the specified attribute from the data
        return new ArrayList<>();
    }

    private ArrayList<String[]> getSubset(ArrayList<String[]> data, String attribute, String value) {
        // Retrieve a subset of the data where the specified attribute has the specified value
        return new ArrayList<>();
    }
}
```

Este código proporciona una estructura básica para construir un árbol de decisión utilizando el algoritmo ID3 en Java. Tendrás que implementar los métodos calculateInfoGain, getAttributeValues y getSubset para completar la funcionalidad del algoritmo. Estos métodos deben calcular la ganancia de información, obtener los valores de los atributos y generar subconjuntos de datos, respectivamente. Además, necesitarás proporcionar una implementación adecuada de la medida de impureza (como la entropía o la impureza de Gini) en el método calculateInfoGain.

- C4.5 (Successor Algorithm of ID3):
  Es una extensión del algoritmo ID3 que puede manejar atributos continuos y categóricos.
  Utiliza la relación de ganancia en lugar de la ganancia de información, lo que lo hace más eficiente.
  Puede manejar datos faltantes y valores de atributos desconocidos.

- CART (Classification and Regression Trees):
  Puede utilizarse tanto para problemas de clasificación como de regresión.
  Utiliza la impureza de Gini como medida para seleccionar los atributos y dividir los nodos.
  Crea árboles binarios de decisión, lo que lo hace más eficiente computacionalmente.

- Random Forest:
  Es una técnica de conjunto que combina múltiples árboles de decisión para mejorar la precisión y reducir el sobreajuste.
  Entrena una serie de árboles de decisión en subconjuntos aleatorios del

conjunto de datos y promedia las predicciones para obtener una salida final.
Es robusto y puede manejar conjuntos de datos grandes con alta dimensionalidad.

- Gradient Boosting Machines (GBM):
  Es una técnica de conjunto que entrena una serie de árboles de decisión de forma secuencial, donde cada nuevo árbol corrige los errores del modelo anterior.
  Utiliza el algoritmo de boosting para mejorar gradualmente la precisión del modelo.
  Es eficaz para problemas de regresión y clasificación.

- XGBoost (Extreme Gradient Boosting):
  Es una implementación optimizada de GBM que utiliza técnicas de regularización y paralelismo para mejorar la velocidad y la precisión.
  Es ampliamente utilizado en competiciones de ciencia de datos debido a su rendimiento superior.
  Proporciona una API fácil de usar y es compatible con múltiples lenguajes de programación.

Estos son solo algunos de los algoritmos de árboles de decisión más comunes. La elección del algoritmo adecuado depende del tipo de datos, el problema que se esté abordando y las características específicas del conjunto de datos.

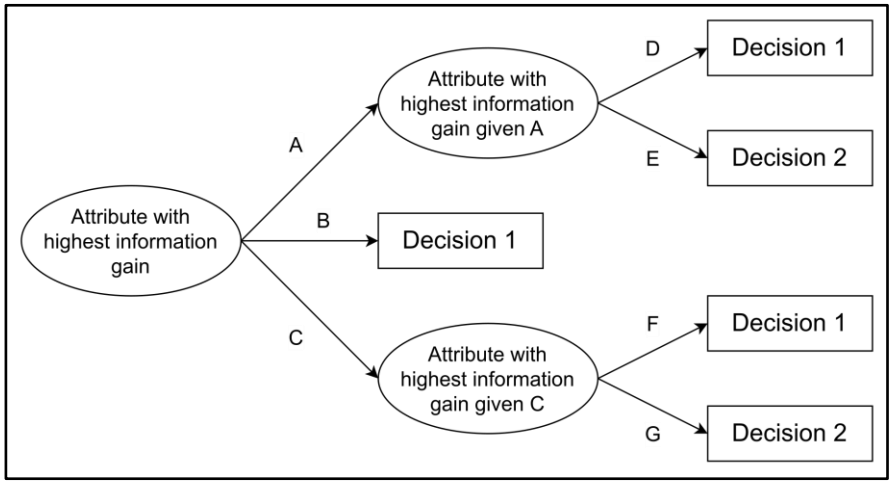

Figura N° 14: Posible árbol de decisión generado por ID3.

- **Entropía de la información.**
  El concepto de entropía de la información es esencial en la teoría de la información. Desde una perspectiva conceptual, la información puede ser entendida como algo que se almacena o se genera como resultado de variables que pueden tomar diferentes valores. Estas variables representan unidades de almacenamiento que pueden contener distintos momentos, cada uno con varios valores específicos,

siguiendo un proceso para aceptar esos valores. De manera informal, obtenemos información de una variable a través de su valor, similar a cómo obtenemos información de un correo electrónico leyendo su contenido. En el caso de la variable, la información proviene del proceso detrás de la variable.

La entropía de una variable es la cantidad de información contenida en esa variable. Esta cantidad de información no solo depende del número de valores diferentes que la variable puede tomar, sino también de la sorpresa que puede generar. Informalmente, la cantidad de información en un correo electrónico es proporcional a la cantidad de "sorpresa" que genera. Por ejemplo, si un correo electrónico simplemente repite información anterior, no aporta nueva información. En cambio, si el correo electrónico revela el resultado de una elección, es más informativo. De manera similar, la entropía de una variable está relacionada con la sorpresa que genera cuando se revela. La entropía de Shannon cuantifica la cantidad de información en una variable, proporcionando el fundamento para una teoría sobre la noción de información.

El almacenamiento y la transmisión de información intuitivamente se espera que estén vinculados a la cantidad de información involucrada. Por ejemplo, la información puede estar relacionada con el resultado de lanzar una moneda. Esta información puede almacenarse en una variable booleana que puede contener valores de 0 ó 1, representando si la moneda salió cara o cruz. En la tecnología de almacenamiento y transmisión digital, esta variable booleana puede representarse en un solo bit, la unidad básica de almacenamiento/transmisión digital. Sin embargo, este bit solo captura el valor bruto de la variable, es decir, los datos sin procesar correspondientes al resultado del lanzamiento de la moneda. No captura de manera concisa la información en el lanzamiento de la moneda, como si la moneda está sesgada o no.

La entropía de Shannon cuantifica, entre otras cosas, la cantidad mínima de almacenamiento y transmisión necesaria para capturar de manera concisa cualquier información (en lugar de datos brutos) y, en casos típicos, es menor que la requerida para almacenar o transmitir los datos sin procesar. También sugiere una forma de representar la información en el menor número de bits posible.

En resumen, a nivel conceptual, la entropía de Shannon es simplemente la cantidad de información en una variable. Más mundanamente, se traduce en la cantidad de almacenamiento, por ejemplo, el número de bits necesarios para almacenar la variable.

Ejemplo de Cantidad de Información

Supongamos que una variable puede tomar 3 valores diferentes: a, b y c, pero la mitad del tiempo toma el valor a, y una cuarta parte del tiempo los valores b y c. Ahora, podemos representar estos valores con solo un bit para el valor a, con 2 bits

para los valores b y c. Esto se conoce como el código de Huffman. Observamos que decodificar la representación es sencillo y no requiere de una representación explícita. Específicamente, cuando el primer bit es 0, el receptor sabe que debe detenerse y leer ese valor; cuando el primer bit es 1, el receptor sabe que también debe leer el siguiente bit para completar la palabra.

Con la representación anterior, necesitamos solo 1 bit la mitad del tiempo (cuando el valor es a), y 2 bits el resto del tiempo (cuando el valor es b o c). Entonces, el promedio de bits necesarios es:

(0.5 * 1) + (0.25 * 2) + (0.25 * 2) = 1.5 bits.

La entropía de la variable anterior, que tiene diferentes probabilidades de tomar valores, es de 1.5.

Entropía de Shannon.

E = --i(p(i)-log2(p(i))))

El almacenamiento requerido varía en casos donde las probabilidades no siguen una distribución conveniente, como en el caso donde los numeradores o denominadores no son potencias de 2. Por ejemplo, consideremos la probabilidad 9/27, que implica 27 posibles valores que una variable puede tomar, de los cuales 9 son relevantes. Dado que log2(27) no es un número entero, surge la pregunta de cómo interpretar esta situación. En el almacenamiento digital, la pérdida de bits se produce en números enteros, no en fracciones.

La fórmula de la entropía, en tales casos, puede parecer una abstracción matemática que no se ajusta necesariamente a la realidad tangible. Sin embargo, para entender sus resultados, podríamos imaginar tecnologías donde las unidades de almacenamiento no se limitan a cantidades enteras y pueden contener valores no enteros. Por ejemplo, en computación cuántica, un bit puede representar simultáneamente los valores 0 y 1 con probabilidades distintas. De manera más mundana, en un sistema de tres estados lógicos, cada unidad de almacenamiento puede asumir uno de tres valores posibles.

Al cambiar la base del logaritmo en la fórmula de entropía, podemos adaptarla a estas situaciones. Por ejemplo, al usar una base 3, la entropía para el ejemplo anterior se convierte en -(9/27) * log3(9/27) = 1/3. Esto significa que la información puede ser representada en una unidad de almacenamiento de tres estados, con una "victoria" de 1/3 de las veces.

Un simple cambio en la base del logaritmo en la fórmula de la entropía nos proporciona el número mínimo de bits necesarios para almacenar información en

una tecnología donde cada unidad de almacenamiento permite estados equivalentes al valor de la base del logaritmo. Aunque en el almacenamiento digital, que utiliza bits binarios, la entropía de Shannon establece un límite inferior en el número de bits necesarios para almacenar o transmitir información. En el ejemplo anterior, requeriríamos 2 bits, siendo 2 el siguiente entero mayor que 1.263, según lo determinado por la fórmula de entropía.

- **Ejemplo resuelto de ID3.**

A continuación, un ejemplo de implementación del algoritmo ID3 (Iterative Dichotomiser 3) en Java. El ID3 es un algoritmo de aprendizaje automático utilizado para la construcción de árboles de decisión. Este ejemplo es bastante básico y asume que tienes un conocimiento básico de árboles de decisión y Java.

```
import java.util.ArrayList;
import java.util.HashMap;

public class ID3 {

    // Clase para representar un nodo del árbol de decisión
    private static class TreeNode {
        String attribute;
        HashMap<String, TreeNode> children;
        boolean isLeaf;
        String label;

        TreeNode(String attribute) {
            this.attribute = attribute;
            this.children = new HashMap<>();
            this.isLeaf = false;
            this.label = null;
        }
    }

    // Método principal para construir el árbol de decisión
    public static TreeNode buildDecisionTree(ArrayList<ArrayList<String>> data,
ArrayList<String> attributes) {
        if (data.isEmpty()) {
            return null;
        }

        // Si todos los ejemplos tienen la misma clase, devolver un nodo hoja con esa
clase
        if (sameClass(data)) {
            TreeNode leaf = new TreeNode(null);
```

```
            leaf.isLeaf = true;
            leaf.label = data.get(0).get(data.get(0).size() - 1);
            return leaf;
        }

        // Si no hay más atributos para dividir, devolver un nodo hoja con la clase mayoritaria
        if (attributes.isEmpty()) {
            TreeNode leaf = new TreeNode(null);
            leaf.isLeaf = true;
            leaf.label = majorityClass(data);
            return leaf;
        }

        // Encontrar el mejor atributo para dividir
        String bestAttribute = findBestAttribute(data, attributes);

        TreeNode root = new TreeNode(bestAttribute);

        ArrayList<String> remainingAttributes = new ArrayList<>(attributes);
        remainingAttributes.remove(bestAttribute);

        // Dividir el conjunto de datos en subconjuntos según los valores del mejor atributo
        HashMap<String, ArrayList<ArrayList<String>>> subsets = splitData(data, attributes.indexOf(bestAttribute));

        for (String value : subsets.keySet()) {
            TreeNode child = buildDecisionTree(subsets.get(value), remainingAttributes);
            root.children.put(value, child);
        }

        return root;
    }

    // Método para comprobar si todos los ejemplos tienen la misma clase
    private static boolean sameClass(ArrayList<ArrayList<String>> data) {
        String firstClass = data.get(0).get(data.get(0).size() - 1);
        for (ArrayList<String> example : data) {
            if (!example.get(example.size() - 1).equals(firstClass)) {
                return false;
            }
        }
```

```
        return true;
    }

    // Método para encontrar la clase mayoritaria en un conjunto de datos
    private static String majorityClass(ArrayList<ArrayList<String>> data) {
        HashMap<String, Integer> classCounts = new HashMap<>();
        for (ArrayList<String> example : data) {
            String label = example.get(example.size() - 1);
            classCounts.put(label, classCounts.getOrDefault(label, 0) + 1);
        }
        String majorityClass = null;
        int maxCount = 0;
        for (String label : classCounts.keySet()) {
            int count = classCounts.get(label);
            if (count > maxCount) {
                maxCount = count;
                majorityClass = label;
            }
        }
        return majorityClass;
    }

    // Método para encontrar el mejor atributo para dividir el conjunto de datos
    private static String findBestAttribute(ArrayList<ArrayList<String>> data,
ArrayList<String> attributes) {
        double bestGain = -1.0;
        String bestAttribute = null;
        for (String attribute : attributes) {
            double gain = informationGain(data, attributes.indexOf(attribute));
            if (gain > bestGain) {
                bestGain = gain;
                bestAttribute = attribute;
            }
        }
        return bestAttribute;
    }

    // Método para calcular la ganancia de información de un atributo
    private static double informationGain(ArrayList<ArrayList<String>> data, int
attributeIndex) {
        // Calcular la entropía del conjunto de datos actual
        double entropyS = calculateEntropy(data);
        // Calcular la entropía después de dividir el conjunto de datos por el atributo
dado
```

```
        double weightedEntropy = 0.0;
        HashMap<String, ArrayList<ArrayList<String>>> subsets = splitData(data, attributeIndex);
        for (String value : subsets.keySet()) {
            double subsetEntropy = calculateEntropy(subsets.get(value));
            double weight = (double) subsets.get(value).size() / data.size();
            weightedEntropy += weight * subsetEntropy;
        }

        // Calcular la ganancia de información
        return entropyS - weightedEntropy;
    }

    // Método para calcular la entropía de un conjunto de datos
    private static double calculateEntropy(ArrayList<ArrayList<String>> data) {
        HashMap<String, Integer> classCounts = new HashMap<>();
        for (ArrayList<String> example : data) {
            String label = example.get(example.size() - 1);
            classCounts.put(label, classCounts.getOrDefault(label, 0) + 1);
        }

        double entropy = 0.0;
        for (String label : classCounts.keySet()) {
            double probability = (double) classCounts.get(label) / data.size();
            entropy -= probability * Math.log(probability) / Math.log(2);
        }
        return entropy;
    }

    // Método para dividir el conjunto de datos en subconjuntos según el valor de un atributo dado
    private static HashMap<String, ArrayList<ArrayList<String>>> splitData(ArrayList<ArrayList<String>> data, int attributeIndex) {
        HashMap<String, ArrayList<ArrayList<String>>> subsets = new HashMap<>();
        for (ArrayList<String> example : data) {
            String value = example.get(attributeIndex);
            if (!subsets.containsKey(value)) {
                subsets.put(value, new ArrayList<>());
            }
            subsets.get(value).add(example);
        }
        return subsets;
    }
```

```
    // Método para clasificar un ejemplo utilizando el árbol de decisión construido
    public static String classify(TreeNode root, ArrayList<String> example) {
        if (root.isLeaf) {
            return root.label;
        }
        String attributeValue = example.get(root.attribute);
        TreeNode child = root.children.get(attributeValue);
        return classify(child, example);
    }

    // Ejemplo de uso
    public static void main(String[] args) {
        // Ejemplo de datos (atributos: Outlook, Temperature, Humidity, Wind)
        ArrayList<ArrayList<String>> data = new ArrayList<>();
        data.add(new ArrayList<>(List.of("Sunny", "Hot", "High", "Weak", "No")));
        data.add(new ArrayList<>(List.of("Sunny", "Hot", "High", "Strong", "No")));
        data.add(new ArrayList<>(List.of("Overcast", "Hot", "High", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Mild", "High", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Cool", "Normal", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain

", "Cool", "Normal", "Strong", "No")));
        data.add(new ArrayList<>(List.of("Overcast", "Cool", "Normal", "Strong",
"Yes")));
        data.add(new ArrayList<>(List.of("Sunny", "Mild", "High", "Weak", "No")));
        data.add(new ArrayList<>(List.of("Sunny", "Cool", "Normal", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Mild", "Normal", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Sunny", "Mild", "Normal", "Strong", "Yes")));
        data.add(new ArrayList<>(List.of("Overcast", "Mild", "High", "Strong", "Yes")));
        data.add(new ArrayList<>(List.of("Overcast", "Hot", "Normal", "Weak",
"Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Mild", "High", "Strong", "No")));

        // Lista de atributos
        ArrayList<String> attributes = new ArrayList<>(List.of("Outlook",
"Temperature", "Humidity", "Wind"));

        // Construir el árbol de decisión
        TreeNode decisionTree = buildDecisionTree(data, attributes);

        // Clasificar un nuevo ejemplo
        ArrayList<String> newExample = new ArrayList<>(List.of("Sunny", "Cool",
"High", "Strong"));
```

```
        String classification = classify(decisionTree, newExample);
        System.out.println("Classification: " + classification);
    }
}
```

Este es solo un ejemplo básico para ayudarte a comprender cómo se implementa el algoritmo ID3 en Java. Puedes ajustarlo y ampliarlo según tus necesidades específicas.

- **Crear un árbol ID3.**

Para crear un árbol ID3, necesitamos definir cómo representar el árbol, cómo calcular la entropía y la ganancia de información, y luego implementar el algoritmo de construcción del árbol. Aquí tienes una implementación básica en Java:

```
import java.util.ArrayList;
import java.util.HashMap;

public class ID3 {

    // Clase para representar un nodo del árbol de decisión
    private static class TreeNode {
        String attribute;
        HashMap<String, TreeNode> children;
        boolean isLeaf;
        String label;

        TreeNode(String attribute) {
            this.attribute = attribute;
            this.children = new HashMap<>();
            this.isLeaf = false;
            this.label = null;
        }
    }

    // Método principal para construir el árbol de decisión
    public static TreeNode buildDecisionTree(ArrayList<ArrayList<String>> data,
ArrayList<String> attributes) {
        if (data.isEmpty()) {
            return null;
        }

        // Si todos los ejemplos tienen la misma clase, devolver un nodo hoja con esa
clase
        if (sameClass(data)) {
```

```
            TreeNode leaf = new TreeNode(null);
            leaf.isLeaf = true;
            leaf.label = data.get(0).get(data.get(0).size() - 1);
            return leaf;
        }

        // Si no hay más atributos para dividir, devolver un nodo hoja con la clase mayoritaria
        if (attributes.isEmpty()) {
            TreeNode leaf = new TreeNode(null);
            leaf.isLeaf = true;
            leaf.label = majorityClass(data);
            return leaf;
        }

        // Encontrar el mejor atributo para dividir
        String bestAttribute = findBestAttribute(data, attributes);

        TreeNode root = new TreeNode(bestAttribute);

        ArrayList<String> remainingAttributes = new ArrayList<>(attributes);
        remainingAttributes.remove(bestAttribute);

        // Dividir el conjunto de datos en subconjuntos según los valores del mejor atributo
        HashMap<String, ArrayList<ArrayList<String>>> subsets = splitData(data, attributes.indexOf(bestAttribute));

        for (String value : subsets.keySet()) {
            TreeNode child = buildDecisionTree(subsets.get(value), remainingAttributes);
            root.children.put(value, child);
        }

        return root;
    }

    // Método para comprobar si todos los ejemplos tienen la misma clase
    private static boolean sameClass(ArrayList<ArrayList<String>> data) {
        String firstClass = data.get(0).get(data.get(0).size() - 1);
        for (ArrayList<String> example : data) {
            if (!example.get(example.size() - 1).equals(firstClass)) {
                return false;
            }
```

```
            }
            return true;
        }

        // Método para encontrar la clase mayoritaria en un conjunto de datos
        private static String majorityClass(ArrayList<ArrayList<String>> data) {
            HashMap<String, Integer> classCounts = new HashMap<>();
            for (ArrayList<String> example : data) {
                String label = example.get(example.size() - 1);
                classCounts.put(label, classCounts.getOrDefault(label, 0) + 1);
            }
            String majorityClass = null;
            int maxCount = 0;
            for (String label : classCounts.keySet()) {
                int count = classCounts.get(label);
                if (count > maxCount) {
                    maxCount = count;
                    majorityClass = label;
                }
            }
            return majorityClass;
        }

        // Método para encontrar el mejor atributo para dividir el conjunto de datos
        private static String findBestAttribute(ArrayList<ArrayList<String>> data,
    ArrayList<String> attributes) {
            double bestGain = -1.0;
            String bestAttribute = null;
            for (String attribute : attributes) {
                double gain = informationGain(data, attributes.indexOf(attribute));
                if (gain > bestGain) {
                    bestGain = gain;
                    bestAttribute = attribute;
                }
            }
            return bestAttribute;
        }

        // Método para calcular la ganancia de información de un atributo
        private static double informationGain(ArrayList<ArrayList<String>> data, int
    attributeIndex) {
            // Calcular la entropía del conjunto de datos actual
            double entropyS = calculateEntropy(data);
```

```
        // Calcular la entropía después de dividir el conjunto de datos por el atributo dado
        double weightedEntropy = 0.0;
        HashMap<String, ArrayList<ArrayList<String>>> subsets = splitData(data, attributeIndex);
        for (String value : subsets.keySet()) {
            double subsetEntropy = calculateEntropy(subsets.get(value));
            double weight = (double) subsets.get(value).size() / data.size();
            weightedEntropy += weight * subsetEntropy;
        }

        // Calcular la ganancia de información
        return entropyS - weightedEntropy;
    }

    // Método para calcular la entropía de un conjunto de datos
    private static double calculateEntropy(ArrayList<ArrayList<String>> data) {
        HashMap<String, Integer> classCounts = new HashMap<>();
        for (ArrayList<String> example : data) {
            String label = example.get(example.size() - 1);
            classCounts.put(label, classCounts.getOrDefault(label, 0) + 1);
        }

        double entropy = 0.0;
        for (String label : classCounts.keySet()) {
            double probability = (double) classCounts.get(label) / data.size();
            entropy -= probability * Math.log(probability) / Math.log(2);
        }
        return entropy;
    }

    // Método para dividir el conjunto de datos en subconjuntos según el valor de un atributo dado
    private static HashMap<String, ArrayList<ArrayList<String>>> splitData(ArrayList<ArrayList<String>> data, int attributeIndex) {
        HashMap<String, ArrayList<ArrayList<String>>> subsets = new HashMap<>();
        for (ArrayList<String> example : data) {
            String value = example.get(attributeIndex);
            if (!subsets.containsKey(value)) {
                subsets.put(value, new ArrayList<>());
            }
            subsets.get(value).add(example);
        }
```

```
        return subsets;
    }

    // Método para clasificar un ejemplo utilizando el árbol de decisión construido
    public static String classify(TreeNode root, ArrayList<String> example) {
        if (root.isLeaf) {
            return root.label;
        }
        String attributeValue = example.get(root.attribute);
        TreeNode child = root.children.get(attributeValue);
        return classify(child, example);
    }

    // Ejemplo de uso
    public static void main(String[] args) {
        // Ejemplo de datos (atributos: Outlook, Temperature, Humidity, Wind)
        ArrayList<ArrayList<String>> data = new ArrayList<>();
        data.add(new ArrayList<>(List.of("Sunny", "Hot", "High", "Weak", "No")));
        data.add(new ArrayList<>(List.of("Sunny", "Hot", "High", "Strong", "No")));
        data.add(new ArrayList<>(List.of("Overcast", "Hot", "High", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Mild", "High", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Cool", "Normal", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Cool", "Normal", "Strong", "No")));
        data.add(new ArrayList<>(List

.of("Overcast", "Cool", "Normal", "Strong", "Yes")));
        data.add(new ArrayList<>(List.of("Sunny", "Mild", "High", "Weak", "No")));
        data.add(new ArrayList<>(List.of("Sunny", "Cool", "Normal", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Mild", "Normal", "Weak", "Yes")));
        data.add(new ArrayList<>(List.of("Sunny", "Mild", "Normal", "Strong", "Yes")));
        data.add(new ArrayList<>(List.of("Overcast", "Mild", "High", "Strong", "Yes")));
        data.add(new ArrayList<>(List.of("Overcast", "Hot", "Normal", "Weak",
"Yes")));
        data.add(new ArrayList<>(List.of("Rain", "Mild", "High", "Strong", "No")));

        // Lista de atributos
        ArrayList<String> attributes = new ArrayList<>(List.of("Outlook",
"Temperature", "Humidity", "Wind"));

        // Construir el árbol de decisión
        TreeNode decisionTree = buildDecisionTree(data, attributes);

        // Clasificar un nuevo ejemplo
        ArrayList<String> newExample = new ArrayList<>(List.of("Sunny", "Cool",
```

```
"High", "Strong"));
        String classification = classify(decisionTree, newExample);
        System.out.println("Classification: " + classification);
    }
}
```

Este código proporciona una implementación básica del algoritmo ID3 en Java. Puedes ajustarlo según tus necesidades específicas, como la estructura de tus datos de entrada y los criterios para calcular la ganancia de información.

REFERENCIAS BIBLIOGRÁFÍCAS

1. Aprendiz de Programacion. (26 de 12 de 2023). *Recorrido árboles binarios – Preorden, inorden y postorden*. https://aprendizdeprogramacion.com/blog/estructuras-de-datos/arboles/recorrido-arboles-binarios/
2. Algotive. (18 de 03 de 2022). *Machine Learning: ¿Qué es el aprendizaje automático y cómo funciona?* https://www.algotive.ai/es-mx/blog/machine-learning-que-es-el-aprendizaje-autom%C3%A1tico-y-c%C3%B3mo-funciona
3. Azure. (s.f.). *¿Qué es el aprendizaje automático?* https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-machine-learning-platform#benefits
4. FinerioConnect. (2024). *Revolución financiera: Inteligencia Artificial y categorización de datos*. https://blog.finerioconnect.com/revolucion-financiera-inteligencia-artificial-y-categorizacion-de-datos/
5. Google Cloud. (s.f.). *Flujo de trabajo de aprendizaje automático*. https://cloud.google.com/ai-platform/docs/ml-solutions-overview?hl=es-419
6. Iberdrola. (s.f.). *Descubre los principales beneficios del 'Machine Learning'*. https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico
7. Kiddle. (s.f.). *Aprendizaje automático para niños*. https://ninos.kiddle.co/Aprendizaje_autom%C3%A1tico
8. Learn. (22 de 12 de 2023). *Algoritmo de árboles de decisión de Microsoft*. https://learn.microsoft.com/es-es/analysis-services/data-mining/microsoft-decision-trees-algorithm?view=asallproducts-allversions
9. Montoya, C. (01 de 06 de 2023). *Introducción a la Inteligencia Artificial. Historia y Conceptos Básicos*. https://www.linkedin.com/pulse/introducci%C3%B3n-la-inteligencia-artificial-historia-y-b%C3%A1sicos-montoya/?originalSubdomain=es
10. Mora Caballero, M. (05 de 12 de 2023). *ID3 y Arbolpedia: Un Enfoque Clásico en Clasificación*. https://www.linkedin.com/pulse/id3-y-arbolpedia-un-enfoque-cl%C3%A1sico-en-clasificaci%C3%B3n-mora-caballero-nuufe/?originalSubdomain=es
11. Palacios, N. (15 de 10 de 2021). *Constendiendo la entropía en la información*. https://kaldt-slange.medium.com/entendiendo-la-entrop%C3%ADa-en-la-informaci%C3%B3n-6bb434cdc377
12. Ponce Pérez, P. (23 de 11 de 2020). *Estructuras de Datos: Recursividad y árboles*. https://a01205866.medium.com/estructuras-de-datos-recursividad-y-%C3%A1rboles-5ad7b75ef1fb
13. Stopa, S. (13 de 07 de 2023). *Uso de Inteligencia Artificial para clasificación con la técnica de Árbol de Decisión*. https://community.listopro.com/uso-de-inteligencia-artificial-para-clasificacion-con-la-tecnica-de-arbol-de-decision/

REFERENCIAS DE IMÁGENES

Figura N° 01: Introducción a la Inteligencia Artificial.
https://media.licdn.com/dms/image/D4E12AQEgNnEkpCLmYg/article-cover_image-shrink_720_1280/0/1685596411404?e=1715817600&v=beta&t=Z0NlgdguYYSJIp_uQAVgaVyhEqj03Fu89yF7U7B_BSE

Figura N° 02: Línea de Tiempo de Machine Learning.
https://www.algotive.ai/hs-fs/hubfs/00%20Blog/02%20Machine%20Learning/linea_tiempo-01.jpg?width=2000&name=linea_tiempo-01.jpg

Figura N° 03: Aplicaciones prácticas del Machine Learning.
https://www.algotive.ai/hs-fs/hubfs/00%20Blog/02%20Machine%20Learning/8_areas_empresariales-1.png?width=1200&height=576&name=8_areas_empresariales-1.png

Figura N° 04: Beneficios de Machine Learning.
https://www.iberdrola.com/wcorp/gc/prod/es_ES/estaticos/Machine-Learning/fondo.png

Figura N° 05: Flujo de trabajo del AA.
https://cloud.google.com/static/ai-platform/images/ml-workflow.svg?hl=es-419

Figura N° 06: Árbol Binario.
https://miro.medium.com/v2/resize:fit:1100/format:webp/0*Xv6Wm80wDNfM0Oah

Figura N° 07: Tipos de árboles binarios.
https://miro.medium.com/v2/resize:fit:1100/format:webp/0*2b0Cg600iKIhyyKW

Figura N° 08: Altura y nivel de un árbol.
https://miro.medium.com/v2/resize:fit:720/format:webp/0*2NFlm_SZvMiSs86-.png

Figura N° 09: Diagrama Recursividad Factorial.
https://miro.medium.com/v2/resize:fit:828/format:webp/0*6EgTEHjncr0u-Oys.gif

Figura N° 10: Recursividad, Caso Factorial en Java.
https://miro.medium.com/v2/resize:fit:720/format:webp/1*bkxUR1u5oMoJCprj9pCThg.png

Figura N° 11: Explicación Caso Factorial, Recursividad.
https://miro.medium.com/v2/resize:fit:1100/format:webp/0*HuG9DzZzULyxVJNJ

Figura N° 12: Recorrido PreOrden.
https://aprendizdeprogramacion.com/wp-content/uploads/2023/12/recorrido-preorden.webp

Figura N° 13: Recorrido PreOrden.
https://aprendizdeprogramacion.com/wp-content/uploads/2023/12/recorrido-postorden.webp

Figura N° 14: Posible árbol de decisión generado por ID3.
https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/ID3_algorithm_decision_tree.svg/330px-ID3_algorithm_decision_tree.svg.png

RDA

RECURSO DIDÁCTICO PARA EL APRENDIZAJE