GUÍA DE ESTUDIO

Tan, Pang-Ning, Steinbach, Michael & Kumar, Vipin. (2014). *Introduction to Data Mining*. Pearson.

Capítulo 4. Clasificación: conceptos básicos, árboles de decisión y evaluación de modelos

Nombre: _____

1) Explique qué es el **Proceso de Clasificación.**

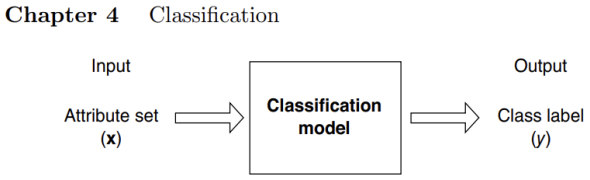

Figura 1: Proceso de clasificación

Es el proceso de aprender una función **f** que mapea cada conjunto de atributos **x** (discretos o continuos) hacia una de las etiquetas de la clase predefinidas **y** (donde es un atributo discreto). Donde el propósito es construir un modelo capaz de asignar a nuevos registros su clase correcta con base a los atributos de entrada. Cuando se hace de manera práctica, este modelo se entrena con un conjunto de ejemplos etiquetados posteriormente se asignan clases desconocidas.

En las estas técnicas son mas adecuadas para describir conjuntos de datos con categorías binarias o nominales, y son menos efectivas para las categorías ordinales, ya que no consideran el orden implícito entre las categorías. Otras formas de relación como las subclase-superclase entre categorías también se ignoran.

2) Describa los siguientes modelos:

   a) Modelo descriptivo
   Es un modelo de clasificación, que sirve como herramienta explicativa para **distinguir entre objetos de diferentes clases**. Como ejemplo se tiene identificar mamíferos de reptiles o aves.

b) Modelo predictivo
Es un modelo de clasificación, usado para **asignar la etiqueta de clase a registros desconocidos**, a partir de sus atributos. Como ejemplo se puede clasificar una nueva especie como mamífero o no mamífero.

3) Describa el enfoque general para resolver problemas de clasificación (Sección 4.2). Consiste en:

- Dividir los datos en un conjunto de entrenamiento (estas clases son conocidas) y un conjunto de prueba (que son desconocidas).
- Utilizar un algoritmo de aprendizaje para inducir un modelo de clasificación aparir del conjunto de entrenamiento.
- Se evalua el modelo aplicando al conjunto de prueba y midiendo su presicion con las diferentes metricas como la *exactitud* o *tasa de error*

Como se observa es un enfoque sistemático. Entre los modelos de clasificación se encuentran: clasificadores de arboles de decisión, basados en reglas, redes neuronales, maquinas de soporte vectorial, ingenuos en Bayes. Estas técnicas emplean un algoritmo para identificar que modelo se ajusta mejor a la relación del conjunto de atributos y la etiqueta de datos de entrada, para poder predecir correctamente las etiquetas de clase que no se conocen.

Un **objetivo clave** es construir modelos de clasificación con buena capacidad de generalización, es decir que se asignan con precisión las nuevas etiquetas de clase previamente desconocidos.

4) Describa la forma en que se resuelve un problema de clasificación (Sección 4.2.1).

Primero se parte de un conjunto de registros con etiquetas conocidas de las que se construye el modelo. Luego se aplica el modelo sobre nuevos registros de las que se desconoce su etiqueta, donde se asigna una nueva etiqueta a este. La efectividad de esta se mide a través de una matriz de confusión que indica los aciertos y los errores del modelo.

		Predicted Class	
		Class =1	Class =2
Actual Class	Class =1	f_11	f_10
	Class =2	f_01	f_00

Pero para resumir esta información con un solo numero se ocupan las siguientes metricas:

- $$\text{precisión} = \frac{\text{Numero de predicciones correctas}}{\text{Numero total de predicciones}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$
- $$\text{tasa de error} = \frac{\text{Numero de predicciones erroneas}}{\text{Numero total de predicciones}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

5) Identifique los criterios considerados en la construcción de la figura 4.4 (Sección 4.3.1). Se presenta un árbol de decisión que ilustra cómo resolver un problema de clasificación binaria, en este caso, determinar si un animal pertenece a la clase **mamífero** o **no mamífero**. Este ejemplo se basa en una versión simplificada del problema de clasificación de vertebrados. El propósito es mostrar cómo un modelo de clasificación puede tomar decisiones secuenciales a partir de atributos observables hasta llegar a una conclusión final.

El árbol representa la idea central de la clasificación: formular **preguntas jerárquicas sobre los atributos de los ejemplos** para dividir progresivamente el conjunto de datos hasta alcanzar una asignación de clase lo más pura posible. Cada nodo corresponde a una pregunta, y las ramas representan las respuestas posibles que conducen a nuevas divisiones o a una etiqueta de clase final.

Criterios usados en la construcción.
Los autores construyen el árbol de la Figura 4.4 basándose en criterios **lógicos y biológicos** que permiten distinguir entre mamíferos y no mamíferos de forma sencilla, jerárquica y comprensible. Los criterios empleados fueron los siguientes:

a) **Temperatura corporal (*Body Temperature*).**

- Este es el primer criterio y ocupa el **nodo raíz** del árbol de decisión.
- La elección de este atributo responde a que es una característica biológica fundamental que permite una separación clara entre los *vertebrados de sangre fría (cold-blooded)* y los *de sangre caliente (warm-blooded).*
- En el contexto biológico, los mamíferos son siempre de sangre caliente, por lo que esta división inicial produce una **partición pura** para los organismos de sangre fría: todos ellos son no mamíferos.
- De esta manera, una de las ramas del árbol (la correspondiente a "cold-blooded") concluye inmediatamente con una hoja rotulada como **Non-mammals**.

- La otra rama (los animales de sangre caliente) requiere un criterio adicional para distinguir entre aves y mamíferos, ya que ambas categorías comparten esta característica.

b) **Modo de reproducción (*Gives Birth*).**

- Este segundo atributo se aplica a la subpoblación de *sangre caliente*, puesto que dentro de este grupo hay tanto mamíferos como aves.
- La pregunta que plantea el nodo interno es: "¿Las hembras de la especie dan a luz a crías vivas?"
- Si la respuesta es *sí*, la especie pertenece a la clase **Mamíferos**, ya que dar a luz es una característica distintiva de este grupo.
- Si la respuesta es *no*, la especie se clasifica como **No mamífero**, ya que probablemente sea un ave u otro vertebrado de sangre caliente que se reproduce por huevos.
- Este segundo criterio permite **refinar la pureza de la clasificación** en el subconjunto de especies de sangre caliente, produciendo dos hojas finales que representan clases mutuamente excluyentes.

c) **Criterio jerárquico y de pureza.**

- El orden de los atributos no es arbitrario: primero se selecciona aquel que produce la **separación más clara y pura** entre las clases (en este caso, la temperatura corporal).
- Después se elige un atributo secundario que refine la distinción en el subconjunto restante (el modo de reproducción).
- Este procedimiento sigue el principio general de los árboles de decisión: en cada nodo se selecciona el atributo que **reduce más la impureza** o **maximiza la ganancia de información**.
- Así, el árbol se construye siguiendo una lógica de partición jerárquica en la que las preguntas iniciales dividen el espacio de atributos de manera global y las posteriores lo hacen de manera más específica.

Razonamiento detrás de la estructura.
El árbol resultante representa un proceso de decisión progresivo y jerárquico:

- En el primer nivel, la división basada en la *temperatura corporal* separa de inmediato a todos los no mamíferos de sangre fría, eliminando la ambigüedad en esa porción del conjunto de datos.

- En el segundo nivel, el criterio de *modo de reproducción* diferencia a los mamíferos (que dan a luz) de los no mamíferos de sangre caliente (aves u otros animales ovíparos).
- El resultado es un árbol con un **nodo raíz** (*Body Temperature*), un **nodo interno** (*Gives Birth*) y tres **hojas** que representan las clases:
  - *Non-mammals* (vertebrados de sangre fría),
  - *Mammals* (vertebrados de sangre caliente que dan a luz),
  - *Non-mammals* (vertebrados de sangre caliente que no dan a luz).

En síntesis, los criterios utilizados en la Figura 4.4 combinan **conocimiento biológico y principios de aprendizaje supervisado**, siguiendo una jerarquía de decisiones que maximiza la pureza de las clases en cada división del árbol.

6) Describa los dos pasos del Algoritmo de Hunt para la construcción de árboles de decisión (Sección 4.3.2).
El **Algoritmo de Hunt** es un procedimiento recursivo propuesto para la construcción de árboles de decisión. Representa el enfoque general de muchos métodos modernos de inducción de árboles, como ID3, C4.5 y CART. Su objetivo es dividir el conjunto de datos de manera progresiva hasta obtener nodos homogéneos (puros) respecto a la clase. La idea central es que en cada paso se selecciona el atributo que mejor separa las clases, y el proceso continúa recursivamente en los subconjuntos resultantes.

El algoritmo opera sobre un conjunto de registros D_t asociado a un nodo t. Este conjunto contiene instancias de entrenamiento, cada una con una etiqueta de clase conocida. El algoritmo sigue dos pasos principales:

a) **Paso 1: Determinar la condición de parada o asignación directa de clase.**
- Si todas las instancias de D_t pertenecen a la misma clase y_t, el nodo t se convierte en un **nodo hoja** y se etiqueta con esa clase.
- Si D_t está vacío (es decir, no hay registros que satisfagan la condición de división anterior), se crea un nodo hoja etiquetado con la **clase mayoritaria** del nodo padre. Esto evita que el árbol quede incompleto.
- Este paso garantiza que el algoritmo detenga la expansión en regiones del espacio de atributos donde los datos ya son puros o insuficientes para continuar dividiendo.

b) **Paso 2: Dividir el conjunto de datos con base en un atributo de particionamiento.**

- Si el conjunto D_t contiene instancias de múltiples clases, el algoritmo selecciona un atributo A y una condición de particionamiento que permitan dividir D_t en subconjuntos más homogéneos.
- La selección del atributo se hace según un **criterio de pureza** (por ejemplo, entropía, índice Gini o ganancia de información). El atributo elegido es aquel que produce la mejor separación entre clases, reduciendo la impureza total del nodo.
- Cada partición generada se convierte en un nuevo nodo hijo, y el algoritmo se aplica **recursivamente** a cada uno de esos nodos.
- Este proceso continúa hasta que todos los nodos son puros o se alcanza una condición de detención (como un número mínimo de instancias por nodo o una ganancia de información mínima).

Descripción conceptual del proceso:
El Algoritmo de Hunt sigue un esquema de *divide y vencerás*. Comienza con el conjunto de entrenamiento completo en la raíz, identifica la mejor división posible y repite el proceso en cada subconjunto. En cada nivel, las decisiones se vuelven más específicas, lo que conduce a un modelo jerárquico que clasifica ejemplos nuevos evaluando sus atributos a lo largo del árbol.

En resumen, el algoritmo se caracteriza por:

- Una **condición base** que define cuándo crear un nodo hoja (cuando los datos son puros o vacíos).
- Un **paso recursivo** que selecciona el mejor atributo de división y repite el proceso en los nodos hijos.

7) Explique los dos casos que requieren condiciones adicionales en la construcción de un árbol de decisión considerando la combinación de valores y etiquetas asociadas en los elementos (p. 154).
Durante la construcción de un árbol de decisión mediante el **Algoritmo de Hunt**, existen situaciones especiales en las que las reglas básicas del proceso no son suficientes para continuar con la partición de los datos. En tales casos, es necesario aplicar **condiciones adicionales** para asegurar que el árbol pueda completarse y asignar una etiqueta de clase apropiada a cada nodo. Los autores identifican dos situaciones

principales que requieren estas condiciones:

a) **Caso 1: Una partición queda vacía (no existen registros que satisfagan la condición de división).**

- Este escenario ocurre cuando, después de seleccionar un atributo y aplicar un criterio de particionamiento, uno o más subconjuntos resultantes no contienen instancias de entrenamiento.
- Por ejemplo, si el atributo elegido tiene un valor posible que no está presente en los datos actuales, el subconjunto correspondiente será vacío.
- En esta situación, el algoritmo no puede continuar el proceso recursivo porque no hay ejemplos para construir el nodo.
- La solución propuesta es asignar a ese nodo hoja la **clase mayoritaria del nodo padre**. De este modo, el árbol conserva coherencia y se evita que ramas incompletas queden sin etiqueta.
- Esta estrategia garantiza una clasificación razonable en casos en que ciertos valores no están representados en el conjunto de entrenamiento.

b) **Caso 2: Todos los registros del nodo tienen los mismos valores de atributo, pero distintas etiquetas de clase.**

- En este caso, el conjunto de instancias D_t no puede dividirse más, ya que todos los registros presentan idénticos valores en sus atributos predictivos.
- Sin embargo, sus etiquetas de clase difieren, lo que significa que el árbol no puede encontrar un atributo que produzca una nueva separación significativa.
- Este escenario suele deberse a ruido en los datos, inconsistencias de etiquetado o a que los atributos disponibles no son suficientes para explicar completamente la variabilidad de las clases.
- Dado que no es posible realizar una división útil, el algoritmo detiene el proceso en ese nodo y lo convierte en una hoja.
- La etiqueta asignada a esa hoja corresponde a la **clase mayoritaria entre las instancias presentes en el nodo**.

Síntesis del comportamiento:
Ambos casos reflejan situaciones donde el algoritmo no puede continuar de manera natural la división recursiva. En el primer caso, el problema surge por la ausencia de ejemplos en una partición; en el segundo, por la imposibilidad de encontrar diferencias en los valores de atributo. En ambos, la solución consiste en **asignar una clase**

dominante, asegurando que el árbol permanezca completo y consistente aun ante datos incompletos o ambiguos.

8) Responda las siguientes preguntas:

   a) ¿Cómo se deben dividir los registros del conjunto de entrenamiento?

   Los registros del conjunto de entrenamiento deben dividirse con base en el **atributo que proporcione la mejor separación entre las clases**. En cada nodo del árbol, se evalúan los posibles atributos y condiciones de particionamiento, eligiendo aquel que:

   - **Maximiza la ganancia de información** o la reducción de impureza del nodo (según el criterio empleado: entropía, índice Gini, o error de clasificación).
   - Genera **subconjuntos lo más puros posible**, es decir, grupos en los que la mayoría de los registros pertenecen a una sola clase.
   - Divide los datos de forma **recursiva y jerárquica**, creando ramas que representan decisiones sucesivas basadas en distintos atributos.

   En términos prácticos, el proceso busca minimizar la mezcla de clases dentro de cada subconjunto, de modo que cada división acerque al árbol a una clasificación más precisa.

   b) ¿Cómo debe terminar el procedimiento de división?
   El procedimiento de división debe finalizar cuando el nodo alcanza una **condición de detención**. Estas condiciones se establecen para evitar divisiones innecesarias o imposibles y garantizar que el árbol sea manejable y generalice adecuadamente. Entre las principales condiciones de parada se incluyen:

   - Cuando **todas las instancias del nodo pertenecen a la misma clase**, es decir, el nodo es puro.
   - Cuando **no existen atributos restantes** que puedan utilizarse para dividir los datos.
   - Cuando **todos los registros tienen los mismos valores en sus atributos**, aunque las etiquetas de clase difieran; en ese caso, se asigna la clase mayoritaria.
   - Cuando se cumple algún **criterio de detención predefinido**, como un número mínimo de instancias por nodo o una ganancia de información menor a un umbral determinado.

Una vez que se cumple cualquiera de estas condiciones, el nodo se convierte en una **hoja**, y se le asigna la clase dominante o más representativa entre los registros que contiene.

9) Explique los métodos para expresar la condición de particionamiento aplicable a cada uno de los siguientes tipos de atributo:
Durante la construcción de un árbol de decisión, el algoritmo debe establecer **condiciones de particionamiento** que determinen cómo se divide el conjunto de datos en cada nodo. Estas condiciones dependen del tipo de atributo que se está evaluando, ya que cada uno requiere una forma distinta de formular las comparaciones o divisiones.

a) Binario
Un atributo binario es aquel que sólo puede tomar dos valores posibles, como por ejemplo *Sí/No*, *Verdadero/Falso*, o *Caliente/Frío*. En este caso, la condición de particionamiento es directa y sencilla:

- Se utiliza una prueba de igualdad del tipo $A = v$, donde A es el atributo y v uno de los dos valores posibles.
- La división produce dos ramas: una para los registros que cumplen la condición ($A = v$) y otra para los que no la cumplen ($A \neq v$).
- En la mayoría de los algoritmos, como ID3 o C4.5, esta división es **binaria por naturaleza**, ya que los dos valores posibles generan dos subconjuntos disjuntos.

Dado que los valores binarios representan categorías excluyentes, no se requiere agrupar ni ordenar; la separación es inmediata y exhaustiva.

b) Nominales
Los atributos nominales pueden tomar **más de dos valores distintos** que no tienen un orden natural entre sí. Ejemplos típicos son: *Color* = {*Rojo, Verde, Azul*} o *Tipo de vehículo* = {*Auto, Camión, Motocicleta*}. En estos casos, existen dos formas de expresar la condición de particionamiento:

- **División múltiple (multiway split):** cada valor distinto del atributo genera una rama diferente. Por ejemplo, el atributo *Color* produciría tres ramas: una para los registros con *Rojo*, otra con *Verde* y otra con *Azul*.
- **División binaria:** se agrupan los valores posibles en dos subconjuntos disjuntos, utilizando una condición del tipo $A \in S$ y $A \notin S$, donde S es un

subconjunto de los valores posibles del atributo. Por ejemplo, se podría dividir entre $Color = \{Rojo,\ Verde\}$ y $Color = \{Azul\}$.

La división binaria suele ser preferida cuando el número de valores distintos es grande, ya que evita generar árboles con demasiadas ramas y mejora la capacidad de generalización.

c) Ordinales

Los atributos ordinales poseen valores que pueden ordenarse de manera natural, como $Tamaño = \{Pequeño,\ Mediano,\ Grande\}$ o $Nivel = \{Bajo,\ Medio,\ Alto\}$. En este caso, el criterio de particionamiento debe respetar el orden inherente de los valores:

- Se pueden convertir los valores en una secuencia ordenada $v_1 < v_2 < v_3 < \ldots < v_k$.
- La condición de particionamiento se expresa como una comparación del tipo $A \leq v_i$ o $A > v_i$, donde v_i representa un punto de corte entre dos niveles consecutivos del atributo.
- De este modo, el conjunto de valores se divide en dos subconjuntos: aquellos que son menores o iguales al umbral y los que lo superan.

Este enfoque garantiza que las divisiones mantengan el significado ordinal del atributo y evita particiones arbitrarias que rompan la relación de orden existente.

d) Continuos

Los atributos continuos pueden tomar un número muy grande o incluso infinito de valores posibles dentro de un rango, como por ejemplo *Edad*, *Temperatura*, *Ingreso* o *Peso*. Para estos atributos, la condición de particionamiento se formula comparando el valor del atributo con un umbral o punto de corte:

- Se utiliza una prueba del tipo $A < v$ o $A \geq v$, donde v es un valor numérico que divide el dominio del atributo en dos intervalos.
- Para determinar el mejor valor de v, se ordenan los registros según los valores del atributo y se prueban posibles puntos de división entre instancias de clases distintas.
- El valor óptimo de v es aquel que produce la **mayor reducción de impureza** (es decir, maximiza la ganancia de información o minimiza el índice Gini).

- De esta forma, cada nodo genera dos ramas: una que agrupa los registros con valores menores al umbral y otra con los que lo superan.

En algunos algoritmos, los atributos continuos pueden dividirse más de una vez en distintos puntos de corte a lo largo del árbol, lo que permite representar relaciones no lineales entre los atributos y las clases.
Es decir:

- Los **binarios** se separan mediante igualdad.
- Los **nominales** pueden generar divisiones múltiples o binarias por agrupación de categorías.
- Los **ordinales** utilizan comparaciones basadas en su orden inherente.
- Los **continuos** se dividen con umbrales numéricos que maximizan la pureza de los subconjuntos.

10) Explique el criterio que se utiliza en las medidas para seleccionar la mejor forma de dividir los registros (Sección 4.3.4).

En la construcción de árboles de decisión, el objetivo principal de cada división es **separar los registros del conjunto de entrenamiento de la manera más pura posible**, es decir, que los subconjuntos resultantes contengan instancias predominantemente de una sola clase. Para determinar cuál atributo y qué condición de particionamiento logran la mejor separación, se utilizan medidas que cuantifican el grado de **impureza** o **mezcla de clases** en un nodo.

El criterio general consiste en seleccionar, en cada nodo, el atributo que produzca la **mayor reducción de impureza** después de la división. En términos formales, si $I(t)$ representa la impureza del nodo t, y la partición genera subconjuntos $t_1, t_2, \ldots, t_k$, entonces la calidad de la división se evalúa mediante la expresión:

$$\Delta = I(t) - \sum_{j=1}^{k} \frac{N(t_j)}{N(t)} I(t_j)$$

donde:

- $I(t)$ es la impureza del nodo padre antes de dividir,
- $I(t_j)$ es la impureza del nodo hijo j,
- $N(t_j)$ es el número de instancias en el nodo hijo,
- $N(t)$ es el número total de instancias en el nodo padre.

La cantidad Δ se conoce como **ganancia de impureza** o **reducción de impureza**, y representa cuánto mejora la pureza del conjunto después de aplicar una división. El atributo con el valor de Δ más alto se elige para dividir el nodo.

Medidas comunes de impureza:

a) **Entropía (usada en ID3 y C4.5):**
La entropía mide el desorden o incertidumbre de las clases en un nodo. Se define como:

$$I(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$$

donde $p(i|t)$ es la proporción de instancias de la clase i en el nodo t.
La entropía es mínima (0) cuando el nodo es completamente puro y máxima cuando las clases están distribuidas uniformemente.

b) **Índice Gini (usado en CART):**
El índice Gini mide la probabilidad de clasificar incorrectamente una instancia si se selecciona aleatoriamente según la distribución de clases en el nodo. Se define como:

$$I(t) = 1 - \sum_{i=1}^{c} [p(i|t)]^2$$

Su valor mínimo (0) se alcanza cuando el nodo contiene solo una clase y máximo cuando las clases están igualmente mezcladas.

c) **Error de clasificación:**
Evalúa la proporción de instancias que no pertenecen a la clase mayoritaria del nodo:

$$I(t) = 1 - \max_i p(i|t)$$

Aunque es simple, es menos sensible a los cambios en la distribución de clases, por lo que se usa principalmente con fines ilustrativos.

Interpretación del criterio de selección:

- Una buena división es aquella que produce **subnodos con baja impureza**, es decir, con registros mayoritariamente de una sola clase.
- Cuanto mayor sea la ganancia Δ, más efectiva será la división, pues el árbol reduce su incertidumbre global.

- Si un atributo genera muchas divisiones con pocos registros en cada rama, puede introducir sobreajuste; por ello, algunos algoritmos (como C4.5) ajustan la ganancia mediante el **Gain Ratio**, penalizando divisiones con demasiados valores.

En resumen, el criterio utilizado en todas las medidas es el mismo: **elegir el atributo que maximice la reducción de impureza o la ganancia de información**, de modo que cada decisión en el árbol acerque el modelo a una clasificación más precisa y homogénea.

11) Justifique por qué los autores mencionan que el mejor atributo de división es el *Car Type* en el árbol de la figura 4.12 (Sección 4.3.4).
Los autores comparan el efecto de dividir el conjunto de datos utilizando distintos atributos —específicamente, *Gender* y *Car Type*— para ilustrar cómo las medidas de impureza permiten identificar la mejor división posible en un árbol de decisión.

Antes de la división, la distribución de clases en el nodo raíz es uniforme, es decir, (0,5, 0,5), lo que indica máxima impureza porque hay la misma cantidad de registros de cada clase. Cuando se divide por el atributo *Gender*, los subconjuntos resultantes presentan distribuciones de (0,6, 0,4) y (0,4, 0,6). Aunque la proporción entre clases cambia ligeramente, ambos subconjuntos siguen conteniendo registros de ambas clases, por lo que la impureza permanece relativamente alta.

En cambio, cuando se realiza la división con el atributo *Car Type*, los subconjuntos resultantes son mucho más homogéneos: algunos contienen sólo ejemplos de una clase (por ejemplo, $C_0 : 8, C_1 : 0$), mientras que otros tienen una proporción fuertemente inclinada hacia una sola clase. En términos de las medidas de impureza, esto implica que:

- La **entropía** y el **índice Gini** de los nodos hijos son más bajos que los obtenidos al dividir por *Gender*.
- La reducción de impureza Δ (diferencia entre la impureza del nodo padre y el promedio ponderado de los nodos hijos) es significativamente mayor.

Por lo tanto, el atributo *Car Type* genera **particiones más puras**, ya que separa de forma más clara las clases, minimizando la mezcla entre ellas. Desde el punto de vista de la teoría de la información, esto equivale a obtener una **mayor ganancia de información**, y desde el enfoque de Gini, a producir el **índice de impureza más bajo**.

Los autores justifican así que *Car Type* es el mejor atributo de división porque:

a) Produce subconjuntos con distribuciones de clase mucho más sesgadas hacia una sola categoría, reflejando una estructura de decisión más clara.

b) Reduce de manera notable la incertidumbre en la clasificación al nivel del nodo.

c) Cumple con el principio de selección del árbol de decisión: elegir en cada paso el atributo que maximice la reducción de impureza.

En resumen, *Car Type* es el mejor atributo de división porque conduce a la mayor pureza posible en los nodos resultantes, demostrando empíricamente que las medidas de impureza —ya sea entropía, Gini o error de clasificación— son consistentes al identificarlo como la opción óptima para separar las clases.

12) Explique en qué consiste el criterio de impureza de nodos tomado en cuenta para la elección de atributos de particionamiento (Sección 4.3.4).

El **criterio de impureza de nodos** es una medida cuantitativa que indica qué tan mezcladas están las clases dentro de un nodo de un árbol de decisión. Se utiliza para determinar qué atributo y qué condición de particionamiento permiten **dividir los registros del conjunto de entrenamiento de la manera más efectiva**, de modo que los subconjuntos resultantes sean lo más homogéneos posible.

En la construcción de un árbol de decisión, el objetivo es minimizar la impureza en los nodos hijos después de una división. Para ello, se comparan las distribuciones de clases antes y después del particionamiento. Cuanto más puro sea un nodo (es decir, cuanto más concentradas estén las instancias en una sola clase), menor será su valor de impureza. En cambio, un nodo con clases mezcladas tendrá una impureza más alta.

Formalmente, si $p(i|t)$ representa la proporción de registros de la clase i en el nodo t, las medidas de impureza más comunes son las siguientes:

$$\text{Entropía}(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$$

$$\text{Índice Gini}(t) = 1 - \sum_{i=1}^{c} [p(i|t)]^2$$

$$\text{Error de clasificación}(t) = 1 - \max_i[p(i|t)]$$

donde c es el número total de clases y, por convención, $0 \log_2 0 = 0$ en los cálculos de entropía:contentReference[oaicite:0]index=0.

Interpretación de las medidas de impureza:

- Cuando un nodo contiene instancias de una sola clase (por ejemplo, $(p_0, p_1) = (1, 0)$), su impureza es **cero**, ya que el nodo es completamente puro.
- Cuando las clases están equilibradas (por ejemplo, $(p_0, p_1) = (0{,}5, 0{,}5)$), la impureza alcanza su **valor máximo**, reflejando la máxima incertidumbre.

Las tres medidas tienen un comportamiento similar, como se muestra en la Figura 4.13 del texto: todas alcanzan su máximo cuando $p = 0{,}5$ (clases balanceadas) y su mínimo cuando $p = 0$ o $p = 1$ (clase única). La diferencia principal entre ellas radica en su sensibilidad a cambios en la distribución de clases. La entropía responde más rápidamente a pequeñas variaciones, el índice Gini es más simple de calcular, y el error de clasificación sirve como una medida aproximada del desempeño del modelo:contentReference[oaicite:1]index=1.

Criterio de selección: Para decidir el mejor atributo de división, el algoritmo calcula la **reducción de impureza** o **ganancia** obtenida tras aplicar una condición de particionamiento. La ganancia se define como:

$$\Delta = I(t) - \sum_{j=1}^{k} \frac{N(t_j)}{N(t)} I(t_j)$$

donde $I(t)$ es la impureza del nodo padre, $I(t_j)$ la de cada nodo hijo, y $\frac{N(t_j)}{N(t)}$ el peso proporcional del tamaño de cada subconjunto. El atributo que produce el valor de Δ más alto se considera el mejor para dividir el conjunto en esa etapa del árbol.

En síntesis, el criterio de impureza de nodos permite evaluar la calidad de una partición midiendo cuán bien separa las clases. El mejor atributo de particionamiento es aquel que genera la **mayor reducción de impureza**, logrando subconjuntos más homogéneos y mejor definidos, lo que a su vez incrementa la capacidad predictiva del árbol de decisión.

13) Explique el significado de Δ (Sección 4.3.4).
En el contexto de la inducción de árboles de decisión, el símbolo Δ representa la **ganancia de impureza** o **ganancia de información**, una medida cuantitativa que evalúa qué tan efectiva es una división para separar las clases dentro de un nodo. Su propósito es comparar la pureza del nodo antes y después del particionamiento, determinando qué atributo produce la mejor separación de los datos.

Cuando un nodo t se divide en varios subnodos hijos $t_1, t_2, \ldots, t_k$ según una condición de prueba, la ganancia Δ se define como:

$$\Delta = I(t) - \sum_{j=1}^{k} \frac{N(t_j)}{N(t)} I(t_j)$$

donde:

- $I(t)$ es la **impureza del nodo padre** antes de la división.
- $I(t_j)$ es la impureza del **nodo hijo** j después de la división.
- $N(t_j)$ es el número de registros en el nodo hijo t_j.
- $N(t)$ es el número total de registros en el nodo padre.

La expresión mide la diferencia entre la impureza original del nodo y la impureza ponderada de los subnodos resultantes. Si la división logra que los subnodos sean más homogéneos (es decir, con valores de impureza bajos), el valor de Δ será alto. En cambio, si los subnodos conservan una mezcla similar de clases, Δ será pequeño o incluso cercano a cero.

Interpretación del valor de Δ:

- Un valor grande de Δ indica que la división produce una **mejor separación entre clases**, ya que reduce significativamente la incertidumbre o desorden de los datos.
- Un valor pequeño de Δ sugiere que la división no mejora mucho la pureza del conjunto y, por tanto, el atributo correspondiente no es un buen candidato para dividir el nodo.
- Si $\Delta = 0$, significa que la división no aporta ninguna mejora respecto al nodo original, pues los subnodos tienen la misma distribución de clases que el padre.

En los algoritmos de árboles de decisión, como ID3, C4.5 o CART, Δ es el criterio central para seleccionar el mejor atributo en cada paso del proceso de particionamiento. Cuanto mayor sea la ganancia de impureza, mayor será la capacidad del atributo para generar un árbol más eficiente, preciso y representativo de la estructura subyacente de los datos.

14) Considerando la figura 4.14, explique la conveniencia de la elección del atributo B para realizar el particionamiento (Sección 4.3.4).
Se comparan dos posibles atributos de división, denominados A y B, en un conjunto de datos con dos clases balanceadas. Antes de realizar cualquier división, el índice de Gini del nodo padre es 0,5, indicando una mezcla perfecta entre ambas clases. El objetivo es identificar cuál de los dos atributos produce subconjuntos más homogéneos después del particionamiento.

Si el conjunto se divide utilizando el atributo A, los autores calculan que los índices de Gini de los nodos hijos son 0,4898 y 0,480, respectivamente. El promedio ponderado de ambos valores, considerando el tamaño relativo de cada subconjunto, se obtiene mediante: $I_A = \frac{7}{12}(0{,}4898) + \frac{5}{12}(0{,}480) = 0{,}486$. En cambio, al realizar la división con el atributo B, los subconjuntos resultan más homogéneos, con un promedio ponderado del índice de Gini igual a 0,375. Este valor es considerablemente menor que el obtenido con el atributo A, lo que significa que los registros quedaron más "puros" después de la división.

Dado que el criterio de selección de atributos en los árboles de decisión se basa en la **reducción de impureza** (o **ganancia de Gini**), el atributo que produzca el valor ponderado de impureza más bajo —es decir, la mayor diferencia entre la impureza del nodo padre y la impureza promedio de los nodos hijos— será el preferido. Matemáticamente, esto corresponde a maximizar:

$$\Delta = I(\text{padre}) - \sum_j \frac{N_j}{N} I_j,$$

donde I_j representa la impureza del subnodo j. En este caso, como el promedio ponderado del índice de Gini de los nodos hijos generados por B (0,375) es menor que el de A (0,486), se tiene que:

$$\Delta_B > \Delta_A.$$

Esto significa que la división basada en B reduce más la impureza total del sistema, separando mejor las clases y produciendo subconjuntos más puros.

Interpretación conceptual:

- El atributo B produce una estructura de árbol más informativa, porque distingue con mayor claridad entre las dos clases.
- La disminución más pronunciada del índice de Gini implica que los nodos hijos tienen distribuciones de clase más sesgadas hacia una categoría, lo que incrementa la pureza.
- En términos de la teoría de la información, esto equivale a una mayor **ganancia de información** o reducción de entropía.

Por estas razones, los autores concluyen que la elección del atributo B es más conveniente para realizar el particionamiento, ya que produce una división más efectiva de los datos, minimizando la impureza de los nodos descendientes y maximizando la capacidad del árbol para clasificar correctamente los registros futuros.

15) Analice el efecto que tienen las divisiones binarias con respecto a la división múltiple de la figura 4.15 (Sección 4.3.4).
Se comparan dos enfoques distintos para dividir un atributo nominal con varios valores posibles: la **división múltiple (multiway split)** y la **división binaria (binary split)**. Ambos métodos buscan reducir la impureza de los nodos, pero lo hacen de formas diferentes y con efectos particulares sobre la estructura y complejidad del árbol de decisión.

1. División múltiple (multiway split):
En una división múltiple, cada valor posible del atributo genera una rama distinta en el árbol. Por ejemplo, si el atributo *Car Type* puede tomar los valores *Sports*, *Luxury* y *Family*, el nodo se divide en tres ramas, una por cada categoría. Este tipo de división:

- Produce un solo nivel de división para cubrir todas las categorías del atributo.
- Tiende a generar un árbol menos profundo, ya que todos los valores se evalúan simultáneamente.

- Sin embargo, puede conducir a un árbol más **ancho** (con muchos nodos hijos), y en algunos casos, a una posible fragmentación de los datos, ya que cada rama recibe menos ejemplos para su posterior particionamiento.
- Además, al dividir en muchos subconjuntos, algunos nodos pueden quedar con muy pocos registros, lo que reduce la fiabilidad estadística de las decisiones posteriores.

En este sentido, la división múltiple es útil cuando las categorías son pocas y bien definidas, o cuando cada valor está claramente asociado con una sola clase. No obstante, puede ser menos efectiva cuando el número de categorías es grande o las clases no están perfectamente separadas.

2. División binaria (binary split):
En una división binaria, en lugar de crear una rama por cada valor del atributo, se agrupan los valores en dos subconjuntos disjuntos. Por ejemplo, el atributo *Car Type* podría dividirse en dos grupos: {Sports, Luxury} y {Family}. De esta forma:

- El árbol crece mediante una secuencia de decisiones binarias que pueden refinar progresivamente la partición del espacio de atributos.
- Cada división binaria busca la combinación de valores que maximice la reducción de impureza en esa etapa, generando una estructura más flexible y controlada.
- Permite capturar relaciones más complejas entre categorías al realizar divisiones sucesivas en diferentes niveles del árbol.

El enfoque binario es más general y puede adaptarse mejor a conjuntos de datos donde las categorías tienen interdependencias o relaciones jerárquicas. Sin embargo, esto puede generar un árbol más profundo y, por tanto, potencialmente más costoso en términos computacionales.

3. Comparación y análisis del efecto:

- **Profundidad y complejidad del árbol:** Las divisiones múltiples tienden a producir árboles más anchos pero menos profundos, mientras que las binarias generan árboles más profundos pero con un crecimiento más controlado.
- **Precisión y generalización:** Las divisiones binarias suelen permitir una mejor generalización, ya que las agrupaciones se eligen según la reducción de impureza

más efectiva. En contraste, las divisiones múltiples pueden sobreajustarse si los valores del atributo están demasiado fragmentados.

- **Consistencia del criterio de partición:** La división binaria se ajusta mejor a los algoritmos que aplican un criterio uniforme de selección de atributos (como el índice de Gini o la entropía), pues cada decisión optimiza localmente la pureza del nodo.
- **Interpretabilidad:** Los árboles construidos con divisiones múltiples son más fáciles de interpretar en las primeras capas, ya que muestran directamente todas las categorías posibles, aunque se vuelven difíciles de analizar si el número de valores crece.

Las divisiones binarias permiten un control más fino sobre la reducción de impureza y producen árboles con mejor capacidad de ajuste progresivo, mientras que las divisiones múltiples ofrecen simplicidad y menor profundidad a costa de menor flexibilidad. La elección entre una u otra depende del equilibrio buscado entre interpretabilidad, eficiencia y precisión en la clasificación. En la Figura 4.15, los autores destacan que la división basada en el atributo *Car Type* muestra cómo la agrupación binaria puede generar una menor impureza promedio que la separación múltiple directa, evidenciando su conveniencia en la práctica de inducción de árboles de decisión.

16) Explique cómo se realiza la división de atributos continuos (Sección 4.3.4).
La división de atributos continuos en un árbol de decisión se basa en encontrar un **punto de corte óptimo** o **umbral** que permita separar los registros en dos grupos de manera que las clases queden lo más puras posible. A diferencia de los atributos categóricos, cuyos valores se agrupan o comparan por igualdad, los atributos continuos —como la edad, el ingreso o la temperatura— se dividen utilizando relaciones de orden.

El método general consiste en definir una condición de particionamiento del tipo:

$$A \leq v \quad \text{o} \quad A > v$$

donde A es el atributo continuo y v es un valor numérico que se prueba como posible punto de división. Esta condición genera dos subconjuntos (una división binaria): uno con los registros que cumplen $A \leq v$ y otro con los que cumplen $A > v$. El objetivo es encontrar el valor v que produzca la **menor impureza promedio ponderada** entre ambos subconjuntos, es decir, que maximice la ganancia de información o la reducción

del índice Gini.

Procedimiento para encontrar el punto óptimo de división:

a) Se ordenan los registros de entrenamiento según los valores del atributo continuo A.

b) Se consideran todos los valores posibles del atributo como candidatos a punto de corte. En la práctica, sólo se evalúan aquellos valores que se encuentran **entre instancias de clases diferentes**.

c) Para cada posible valor v, se calculan:

- El número de instancias que cumplen $A \leq v$ y $A > v$.
- El grado de impureza (por ejemplo, índice Gini o entropía) de los dos subconjuntos.
- La impureza promedio ponderada después de la división:

$$I_{\text{div}} = \frac{N_{\leq v}}{N} I_{\leq v} + \frac{N_{>v}}{N} I_{>v}.$$

d) Se selecciona el valor v^* que minimiza I_{div}, o equivalentemente, el que maximiza la ganancia de impureza:

$$\Delta = I(\text{padre}) - I_{\text{div}}.$$

Aspectos prácticos:

- Este proceso de búsqueda es computacionalmente costoso, pues requiere evaluar $O(N)$ posibles divisiones para cada atributo continuo.
- Para reducir el costo, algunos algoritmos aplican **estrategias de discretización**, agrupando los valores en intervalos y asignando a cada intervalo un valor ordinal antes de construir el árbol.
- Otros métodos pueden considerar divisiones múltiples (rango de valores), aunque la mayoría de las implementaciones utilizan divisiones binarias, ya que simplifican la interpretación y mantienen la coherencia del criterio de impureza.

En síntesis, la división de atributos continuos consiste en explorar posibles puntos de corte y seleccionar aquel que maximiza la pureza de los nodos hijos. Este procedimiento convierte un dominio numérico continuo en una decisión binaria basada en un umbral óptimo, lo que permite que el árbol de decisión capture relaciones no lineales entre los valores numéricos y las clases de salida.

17) Explique la forma en que se podrían tratar atributos de clave principal aplicando criterios de división (*Gain Ratio*, Sección 4.3.4).
En los árboles de decisión, las medidas de impureza tradicionales como la **entropía** o el **índice de Gini** pueden favorecer indebidamente a los atributos que poseen un gran número de valores distintos. Este problema se vuelve evidente cuando el atributo evaluado es una **clave principal** o un identificador único —por ejemplo, un número de cliente (*Customer ID*) o matrícula—, ya que cada registro tendría un valor diferente, generando divisiones aparentemente "puras" pero sin ningún poder predictivo real.

En la Figura 4.12 y la discusión correspondiente, los autores explican que si se usa una clave principal como criterio de particionamiento, cada registro formaría su propio grupo (partición pura). En términos matemáticos, la impureza después de la división sería mínima (cercana a cero), y por tanto la ganancia de información (Δ) parecería máxima. Sin embargo, este resultado es engañoso: el modelo estaría **sobreajustando** completamente los datos, ya que aprendería detalles específicos de cada ejemplo en lugar de patrones generales.

Para corregir este sesgo, el algoritmo C4.5 introduce el **Gain Ratio**, una versión ajustada de la ganancia de información definida como:

$$\text{Gain Ratio} = \frac{\Delta_{\text{info}}}{\text{SplitInfo}},$$

donde:

$$\text{SplitInfo} = -\sum_{i=1}^{k} P(v_i) \log_2 P(v_i),$$

y $P(v_i)$ es la proporción de instancias que toman el valor v_i del atributo y k es el número de divisiones resultantes:contentReference[oaicite:0]index=0.

El denominador, SplitInfo, mide la cantidad de información generada simplemente por la distribución de los valores del atributo, sin considerar la clase. Si un atributo produce muchas divisiones (como ocurre con una clave principal), su SplitInfo será muy grande, reduciendo considerablemente el valor final del *Gain Ratio*. Así, el criterio penaliza a los atributos con demasiados valores distintos, evitando que sean seleccionados por su ganancia aparente.

Interpretación:

- La **ganancia de información** mide cuánta reducción de impureza genera una división, pero no considera cuántas ramas produce.
- La **información de división** (SplitInfo) actúa como un factor de normalización que compensa el sesgo hacia atributos con muchos valores.
- En consecuencia, el **Gain Ratio** favorece atributos que no sólo dividen bien los datos, sino que también generan un número razonable de particiones equilibradas.

Ejemplo ilustrativo: Si el atributo *Customer ID* tiene 1000 valores distintos (uno por registro), el cálculo de SplitInfo = $\log_2(1000)$ produce un valor alto, lo que hace que el *Gain Ratio* resultante sea muy bajo, descartando este atributo como candidato. En cambio, un atributo como *Car Type*, con sólo tres categorías (*Sports*, *Luxury*, *Family*), genera un SplitInfo mucho menor, por lo que su *Gain Ratio* será más alto y su selección más apropiada.

En resumen, el criterio de **Gain Ratio** permite tratar adecuadamente los atributos de clave principal evitando su elección errónea al penalizar las divisiones con excesiva granularidad. De esta manera, el árbol de decisión mantiene su capacidad de generalización y evita el sobreajuste asociado a atributos con valores únicos o demasiado específicos.

18) Explique por lo menos seis de las once características de la inducción del árbol de decisión (Sección 4.3.7).
Los árboles de decisión presentan una serie de características fundamentales que explican su amplio uso en la minería de datos y el aprendizaje automático. Estas propiedades abarcan tanto sus ventajas prácticas como sus limitaciones teóricas. A continuación, se describen en detalle seis de las once características más importantes analizadas por los autores:

1. Método no paramétrico.
Los árboles de decisión no requieren ninguna suposición previa sobre la forma de la función subyacente que relaciona los atributos con la clase. Esto significa que no dependen de hipótesis sobre la distribución de los datos (como normalidad o linealidad), a diferencia de otros modelos estadísticos. Esta independencia los hace extremadamente flexibles, pues pueden adaptarse a datos cuyos patrones son altamente no lineales o que contienen interacciones complejas entre variables. En otras palabras, el modelo

se ajusta directamente a la estructura observable de los datos sin imponer una forma funcional fija, aprendiendo las relaciones a partir de divisiones jerárquicas basadas en la pureza de las clases.

2. Uso de heurísticas y complejidad del problema.
Encontrar el árbol de decisión globalmente óptimo —es decir, el que produzca la menor tasa de error posible— es un problema de tipo NP-completo. Esto implica que no existe un algoritmo eficiente conocido que garantice la solución exacta en un tiempo razonable, especialmente cuando el número de atributos o de ejemplos de entrenamiento es grande. Por esta razón, los algoritmos más utilizados (como ID3, C4.5 y CART) aplican estrategias heurísticas de búsqueda local. Estos métodos siguen un procedimiento *voraz* (greedy): en cada paso se elige el atributo que produce la mayor reducción de impureza, sin considerar los efectos globales sobre el árbol completo. Aunque no garantiza la mejor estructura posible, esta aproximación logra soluciones de buena calidad en tiempos de ejecución aceptables.

3. Eficiencia computacional y escalabilidad.
Los árboles de decisión se construyen mediante un proceso de particionamiento recursivo descendente que examina los datos solo unas cuantas veces. Por este motivo, son muy eficientes desde el punto de vista computacional. El costo de construir un árbol crece aproximadamente en forma lineal con el número de registros y de atributos, lo que permite aplicarlos incluso en conjuntos de datos de gran tamaño. Una vez generado el árbol, la clasificación de una nueva instancia es también muy rápida, pues basta con seguir una secuencia de comparaciones desde la raíz hasta una hoja. Esta eficiencia hace que los árboles de decisión sean adecuados para sistemas de apoyo a la decisión en tiempo real y para aplicaciones con millones de observaciones.

4. Interpretabilidad y transparencia del modelo.
Una de las razones más poderosas para utilizar árboles de decisión es su facilidad de interpretación. La estructura del árbol puede representarse como un conjunto de reglas de la forma "Si condición, entonces clase", lo que permite comprender de manera directa cómo el modelo toma sus decisiones. Cada nodo interno corresponde a una prueba condicional sobre un atributo, y cada hoja representa una categoría de salida. A diferencia de otros modelos más opacos, los árboles permiten identificar fácilmente las variables más relevantes para la clasificación y las relaciones jerárquicas entre ellas. Este nivel de transparencia es especialmente útil en contextos donde la explicación del resultado es tan importante como la predicción misma, como en medicina, finanzas o análisis de políticas públicas.

5. Robustez ante ruido y valores atípicos.

Los árboles de decisión muestran una tolerancia considerable al ruido en los datos. Si algunos registros contienen errores, estos tienden a afectar solo a las ramas específicas donde ocurren, sin distorsionar significativamente la estructura general del árbol. Además, muchos algoritmos incorporan mecanismos para reducir el sobreajuste causado por el ruido, como la **poda** posterior (*post-pruning*), que elimina ramas poco significativas, o la **poda previa** (*pre-pruning*), que detiene el crecimiento del árbol cuando la ganancia de impureza es muy baja. De este modo, el modelo se mantiene generalizable y evita memorizar irregularidades del conjunto de entrenamiento.

6. Manejo de atributos redundantes e irrelevantes.
Los árboles de decisión seleccionan los atributos de manera adaptativa según su poder de discriminación, por lo que los atributos irrelevantes tienden a ser ignorados. Si existen atributos redundantes —por ejemplo, variables altamente correlacionadas—, el árbol usualmente elige uno de ellos y descarta el resto, sin degradar significativamente el rendimiento del modelo. No obstante, si el conjunto de datos contiene una gran cantidad de atributos sin información útil, el árbol puede crecer innecesariamente y volverse más complejo. Por esta razón, es común aplicar una etapa de **selección de características** o **filtrado de variables** antes del entrenamiento, lo cual mejora la eficiencia y la capacidad de generalización del modelo.

7. Fragmentación de datos y tamaño de muestra.
A medida que el árbol se expande, los datos disponibles en cada nodo se dividen en subconjuntos cada vez más pequeños. En niveles profundos, puede haber nodos con muy pocos registros, lo que reduce la fiabilidad estadística de las decisiones en esas ramas. Este fenómeno se conoce como **fragmentación de datos**. Cuando el número de ejemplos en un nodo es demasiado pequeño, los cálculos de impureza dejan de ser representativos y el modelo puede volverse inestable. Una forma de mitigar este problema es detener el crecimiento del árbol cuando el número de instancias por nodo cae por debajo de un umbral mínimo o cuando la ganancia de información se vuelve insignificante. Esto también ayuda a controlar el sobreajuste y a mantener un equilibrio entre precisión y generalización.

En conjunto, estas características reflejan el equilibrio que los árboles de decisión logran entre flexibilidad, interpretabilidad y eficiencia. Aunque su simplicidad los hace propensos a sobreajustar si no se controlan adecuadamente, su capacidad para adaptarse a distintos tipos de datos, su resistencia al ruido y su transparencia explicativa los convierten en una de las herramientas más versátiles y valiosas dentro del análisis de datos y la minería de información.

19) Explique lo siguiente (Sección 4.4):

a) En qué consisten los errores de entrenamiento y los errores de generalización.

b) Las características que tiene un buen modelo.

c) ¿Qué significa el sobreajuste?

d) ¿Qué significa el subajuste?

En el proceso de inducción de árboles de decisión, el rendimiento del modelo se evalúa a partir de dos tipos principales de error: el error de entrenamiento y el error de generalización. Comprender su diferencia es esencial para analizar la calidad del modelo y su capacidad de predecir correctamente casos nuevos.

a) Errores de entrenamiento y errores de generalización.

El **error de entrenamiento** mide la proporción de ejemplos del conjunto de entrenamiento que el modelo clasifica incorrectamente. Este error refleja qué tan bien el árbol logra reproducir los patrones presentes en los datos que utilizó para su construcción. En general, conforme el árbol crece y añade más nodos, el error de entrenamiento tiende a disminuir, pues cada división adicional ajusta mejor el modelo a las particularidades del conjunto de entrenamiento.

El **error de generalización**, en cambio, mide el desempeño del modelo sobre un conjunto de datos independiente que no se utilizó para construirlo (conjunto de prueba). Este error indica la capacidad del modelo para **generalizar** lo aprendido a situaciones nuevas. Aunque un árbol más grande suele tener un error de entrenamiento más bajo, esto no garantiza una mejor generalización. De hecho, un modelo excesivamente complejo puede memorizar detalles y ruido del conjunto de entrenamiento, produciendo un error de generalización más alto.

b) Características de un buen modelo.

Un buen modelo de clasificación logra un **equilibrio adecuado entre ajuste y simplicidad**. Debe capturar los patrones reales de los datos sin sobreajustarse a las irregularidades o al ruido. En términos de los errores mencionados:

- Presenta un **bajo error de generalización**, incluso si su error de entrenamiento no es mínimo.
- Tiene una **estructura compacta y comprensible**, evitando divisiones innecesarias o redundantes.

- Posee una **capacidad de predicción estable**, de modo que pequeñas variaciones en los datos no produzcan cambios drásticos en la estructura del árbol.
- Equilibra la **complejidad del modelo** con su capacidad para representar relaciones significativas entre atributos y clases.

En otras palabras, un modelo es bueno no porque "memorice" los datos, sino porque logra representar correctamente las tendencias generales que subyacen a ellos.

c) Significado del sobreajuste.
El **sobreajuste** (*overfitting*) ocurre cuando un árbol de decisión se ajusta demasiado a los datos de entrenamiento, capturando no solo las relaciones verdaderas, sino también el ruido o las variaciones aleatorias propias del conjunto. Este fenómeno suele presentarse cuando el árbol crece sin restricciones, añadiendo divisiones que explican casos muy específicos pero poco representativos.

Un árbol sobreajustado exhibe:

- Un **error de entrenamiento muy bajo** (incluso nulo), ya que clasifica correctamente casi todos los ejemplos conocidos.
- Un **error de generalización alto**, porque no logra clasificar correctamente nuevos ejemplos fuera del conjunto original.

El resultado es un modelo complejo y frágil, incapaz de generalizar. Para evitarlo, se aplican técnicas de **poda** que eliminan nodos o ramas innecesarias, reduciendo la complejidad del árbol y mejorando su rendimiento sobre datos no vistos.

d) Significado del subajuste.
El **subajuste** (*underfitting*) es el caso opuesto: ocurre cuando el modelo es demasiado simple para representar las relaciones presentes en los datos. Un árbol subajustado puede detener su crecimiento antes de capturar patrones relevantes, generando divisiones insuficientes o excesivamente generales.

En este escenario:

- Tanto el **error de entrenamiento** como el **error de generalización** son altos.

- El modelo falla en identificar estructuras significativas en los datos, lo que reduce su poder predictivo.

El subajuste suele deberse a restricciones demasiado severas en el crecimiento del árbol (por ejemplo, límites muy bajos de profundidad o de número mínimo de instancias por nodo). En la práctica, se busca un punto intermedio entre el sobreajuste y el subajuste, donde el modelo tenga suficiente complejidad para representar los datos sin perder su capacidad de generalización.

Los errores de entrenamiento y generalización son medidas complementarias que permiten evaluar el equilibrio del modelo. Un árbol bien construido mantiene bajo su error de generalización, evitando tanto la rigidez de un modelo subajustado como la fragilidad de uno sobreajustado.

20) Explique el comportamiento de las tasas de error de entrenamiento y prueba expresados en la figura 4.23 (Sección 4.4):

a) Cuando el árbol es pequeño (pocos nodos).

b) Conforme aumenta la cantidad de nodos en el árbol.

La Figura 4.23 ilustra el comportamiento típico de las tasas de error de entrenamiento y de prueba conforme aumenta la complejidad del árbol de decisión, medida por el número de nodos. Este análisis permite comprender cómo la estructura del árbol influye en la capacidad del modelo para ajustarse a los datos y para generalizar a nuevas observaciones.

Cuando el árbol es pequeño, con pocas divisiones o nodos, el modelo es simple y general. En esta etapa inicial, el árbol sólo captura las tendencias más amplias de los datos, por lo que su capacidad de clasificación es limitada. Como consecuencia:

- El **error de entrenamiento** es relativamente alto, ya que muchas instancias no se clasifican correctamente. El árbol no logra representar adecuadamente la complejidad de los patrones presentes en el conjunto de entrenamiento.
- El **error de prueba** también es alto, pues el modelo no dispone de suficiente estructura para generalizar con precisión a nuevos datos.

Este escenario corresponde al fenómeno de **subajuste**, en el cual el modelo es demasiado simple y su desempeño es pobre tanto en los datos de entrenamiento como en

los de prueba.

A medida que aumenta la cantidad de nodos, el árbol se vuelve más profundo y complejo. En esta etapa intermedia, las divisiones adicionales permiten capturar mejor las relaciones entre los atributos y las clases, lo que conduce a una reducción significativa del error de entrenamiento. Simultáneamente, el error de prueba también disminuye, ya que el modelo comienza a representar de manera más precisa los patrones subyacentes en los datos. En este punto, el árbol alcanza una zona de equilibrio en la que:

- La **tasa de error de entrenamiento** se mantiene baja.
- La **tasa de error de prueba** alcanza su valor mínimo.

Este es el rango ideal de complejidad del árbol, donde el modelo logra un equilibrio óptimo entre ajuste y generalización.

Sin embargo, conforme se agregan más nodos y el árbol continúa creciendo, empieza a memorizar detalles específicos del conjunto de entrenamiento, incluidos valores atípicos o ruido. En consecuencia:

- El **error de entrenamiento** sigue disminuyendo y puede llegar a ser prácticamente nulo, ya que el árbol se ajusta casi perfectamente a los datos utilizados para su construcción.
- El **error de prueba**, en cambio, comienza a aumentar, reflejando que el modelo ha perdido su capacidad de generalizar.

Este fenómeno es característico del **sobreajuste**: el árbol ha capturado patrones espurios o particularidades que no se repiten en nuevos conjuntos de datos. En la gráfica, esta situación se representa por la divergencia de las dos curvas: la de entrenamiento sigue descendiendo, mientras la de prueba se eleva gradualmente.

En resumen, la figura muestra tres fases claras en la evolución de los errores:

a) En los árboles pequeños, ambos errores son altos debido al subajuste.

b) En los árboles de tamaño intermedio, el error de prueba alcanza su mínimo, indicando un equilibrio óptimo.

c) En los árboles muy grandes, el error de entrenamiento sigue bajando, pero el de prueba aumenta, manifestando sobreajuste.

El objetivo del proceso de construcción o poda del árbol es precisamente detener el crecimiento en la región intermedia, donde la tasa de error de prueba es mínima, garantizando así un modelo con buena capacidad predictiva y generalización equilibrada.

21) Describa de forma breve los siguientes casos de sobreajuste del modelo:

a) Sobreajuste por presencia de ruido.

b) Sobreajuste debido a la falta de muestras representativas.

c) Sobreajuste debido a la falta de muestras representativas.

El sobreajuste puede originarse por diversas causas relacionadas con la calidad y cantidad de los datos de entrenamiento. En todos los casos, el resultado es un modelo que se ajusta excesivamente a las particularidades del conjunto utilizado para construirlo, perdiendo su capacidad para generalizar correctamente. A continuación se describen los principales casos:

Sobreajuste por presencia de ruido.
El ruido se refiere a errores, inconsistencias o variaciones aleatorias en los datos que no representan patrones reales del fenómeno que se intenta modelar. Cuando un árbol de decisión crece sin restricciones, puede llegar a crear ramas específicas para clasificar ejemplos ruidosos o atípicos. Estas ramas capturan irregularidades que no se repiten en los datos futuros, haciendo que el modelo sea más complejo pero menos generalizable. En este caso:

- El árbol memoriza valores anómalos, produciendo divisiones muy específicas que no aportan valor predictivo.
- Se obtiene un **error de entrenamiento bajo**, pero un **error de prueba alto**, porque las reglas derivadas del ruido no son válidas fuera del conjunto original.
- Este tipo de sobreajuste suele mitigarse mediante técnicas de **poda**, eliminando nodos poco significativos, o bien aplicando un **preprocesamiento de los datos** para limpiar errores y valores atípicos antes del entrenamiento.

En resumen, el ruido induce al árbol a aprender detalles irrelevantes, generando un modelo excesivamente detallado y poco robusto.

Sobreajuste debido a la falta de muestras representativas.
El segundo tipo de sobreajuste ocurre cuando el conjunto de entrenamiento es **pequeño o no representativo** del dominio completo del problema. En tales casos, el árbol aprende patrones que reflejan más las particularidades del conjunto reducido que las verdaderas características de la población. Esto provoca que el modelo se comporte correctamente sólo para los ejemplos vistos, pero no para nuevos datos. Entre sus efectos se destacan:

- La selección de divisiones basadas en correlaciones espurias derivadas del tamaño limitado de la muestra.
- La generación de ramas que reflejan distribuciones de clase artificialmente sesgadas.
- Una alta variabilidad en los resultados al entrenar con diferentes subconjuntos del mismo dominio.

El problema radica en que, al tener pocos ejemplos o ejemplos no representativos, el árbol no puede distinguir entre patrones verdaderos y coincidencias accidentales. Este tipo de sobreajuste puede reducirse ampliando el conjunto de entrenamiento, utilizando técnicas de validación cruzada o aplicando métodos de **regularización** que limiten la profundidad del árbol.

Sobreajuste por falta de muestras representativas (reiteración conceptual).
En algunos contextos, la falta de representatividad no solo implica un número insuficiente de ejemplos, sino también una **distribución desigual entre las clases**. Si ciertas clases o regiones del espacio de atributos están poco representadas, el árbol puede construir ramas sesgadas que reflejan solo los patrones de las clases dominantes. Esto conduce a decisiones parciales y a errores sistemáticos al clasificar instancias de las clases minoritarias.

En este sentido:

- El modelo tiende a **sobreajustarse a las clases mayoritarias**, ignorando las minoritarias.
- Las reglas generadas no son generalizables, porque el árbol no ha observado ejemplos suficientes de todos los posibles escenarios.
- Se puede mitigar el problema aplicando **técnicas de balanceo de clases** (como *oversampling* o *undersampling*) o empleando conjuntos de datos más amplios y diversos.

En conjunto, estos casos muestran que el sobreajuste no solo depende de la complejidad del modelo, sino también de la calidad y representatividad del conjunto de entrenamiento. Un árbol equilibrado debe evitar aprender tanto el ruido como las irregularidades propias de muestras limitadas o sesgadas, buscando siempre capturar los patrones generales que realmente caracterizan al fenómeno estudiado.

22) Describa los siguientes procesos de evaluación del desempeño de un clasificador:

a) Método de retención (*holdout*) (Sección 4.5.1)

b) Submuestreo aleatorio (Sección 4.5.2)

c) Validación cruzada (Sección 4.5.3)

d) Bootstrap (Sección 4.5.4)

Los métodos de evaluación del desempeño de un clasificador permiten estimar qué tan bien un modelo aprendido a partir de un conjunto de datos se comportará frente a ejemplos no vistos. En la literatura se describen cuatro enfoques principales: el método de retención, el submuestreo aleatorio, la validación cruzada y el método bootstrap. Cada uno ofrece una estrategia distinta para dividir los datos y medir la precisión del modelo.

Método de retención (*Holdout*).
El método de retención consiste en dividir el conjunto original de datos etiquetados en dos subconjuntos disjuntos: uno para entrenamiento y otro para prueba. El modelo se construye usando el conjunto de entrenamiento y luego se evalúa su precisión sobre el conjunto de prueba. Comúnmente, la división se realiza en proporciones como 50-50 o 2/3 para entrenamiento y 1/3 para prueba. Este procedimiento ofrece una estimación directa del error de generalización.

Sin embargo, presenta limitaciones importantes:

- El modelo se entrena con menos ejemplos, ya que una parte de los datos se reserva para prueba, lo que puede reducir su calidad.
- La estimación de la precisión depende de la manera en que se particionan los datos, pudiendo variar significativamente con diferentes divisiones.
- Si el conjunto de prueba es pequeño, la estimación de la precisión tiene alta varianza y un intervalo de confianza amplio.
- La distribución de las clases puede quedar desequilibrada entre los subconjuntos, afectando la validez de los resultados.

En resumen, aunque simple, este método puede producir resultados inestables si el conjunto de datos es pequeño o poco representativo:contentReference[oaicite:0]index=0.

Submuestreo aleatorio (*Random Subsampling*).
El submuestreo aleatorio, también conocido como método de repetición del *holdout*, mejora la estimación repitiendo varias veces el proceso de división aleatoria en conjuntos de entrenamiento y prueba. En cada iteración, se construye un modelo y se mide su precisión; luego, la precisión final se obtiene como el promedio de los resultados de todas las iteraciones. Matemáticamente:

$$acc_{sub} = \frac{1}{k} \sum_{i=1}^{k} acc_i,$$

donde acc_i representa la precisión del modelo en la i-ésima repetición.

Este enfoque reduce el efecto del azar al utilizar diferentes divisiones del conjunto de datos. Sin embargo, tiene desventajas:

- No garantiza que todos los registros sean usados la misma cantidad de veces para entrenamiento o prueba.
- Algunos ejemplos pueden seleccionarse repetidamente mientras otros nunca se incluyen en la prueba.

Aunque ofrece una estimación más estable que el método de retención simple, sigue siendo sensible al número de repeticiones y a la aleatoriedad de las particiones:contentReference[oaicite:1]index=1.

Validación cruzada (*Cross-Validation*).
La validación cruzada es una mejora sistemática del submuestreo aleatorio. En este método, el conjunto de datos se divide en k particiones de tamaño similar. Durante cada iteración, una partición se usa como conjunto de prueba y las $k-1$ restantes como conjunto de entrenamiento. Este proceso se repite k veces, de modo que cada registro se usa exactamente una vez para prueba y $k-1$ veces para entrenamiento. La precisión total se obtiene promediando los errores de todas las ejecuciones:

$$acc_{cv} = \frac{1}{k} \sum_{i=1}^{k} acc_i.$$

La versión más común es la validación cruzada de 10 pliegues (*10-fold cross-validation*). Un caso especial es la **validación cruzada "dejar uno fuera"** (*leave-one-out*), donde $k = N$ (el número total de instancias). En este caso, cada registro se utiliza una sola vez como prueba, maximizando el uso de los datos para entrenamiento.

Las ventajas de este método incluyen una mejor estimación del error de generalización y menor varianza respecto a los métodos anteriores. Su principal inconveniente es el costo computacional, especialmente en el caso del método *leave-one-out*, que requiere construir N modelos independientes:contentReference[oaicite:2]index=2.

Bootstrap.
El método *bootstrap* utiliza una estrategia diferente basada en el muestreo **con reemplazo**. A partir del conjunto original de N registros, se generan muestras de tamaño N seleccionando registros aleatoriamente y permitiendo que un mismo registro sea elegido varias veces. En promedio, cada muestra contiene alrededor del 63.2 % de los registros únicos del conjunto original (porque $1 - e^{-1} \approx 0{,}632$), mientras que los restantes forman el conjunto de prueba.

Para cada muestra se entrena un modelo y se evalúa su precisión; este proceso se repite b veces, y el desempeño global se calcula combinando las precisiones obtenidas. Una de las variantes más conocidas es el **bootstrap .632**, donde la precisión final se ajusta ponderando la precisión de los modelos entrenados y la de un modelo construido con todo el conjunto original.

El *bootstrap* es muy útil cuando el conjunto de datos es pequeño, ya que aprovecha al máximo la información disponible y proporciona una estimación más robusta del error de generalización. No obstante, es más costoso computacionalmente y puede introducir un sesgo si los registros se repiten en exceso dentro de las muestras:contentReference[oaicite:3]index=3.

En conjunto, estos métodos ofrecen diferentes compromisos entre precisión, costo computacional y estabilidad. El método de retención es el más simple, la validación cruzada equilibra eficiencia y confiabilidad, y el *bootstrap* ofrece la estimación más robusta cuando se dispone de pocos datos.