Contenido

1. ARBOLES DE DECISIÓN Y CONJUNTOS DE ÁRBOLES.

Los **árboles de decisión** son una de las técnicas más populares del *machine learning* para la resolución de problemas reales. Se utilizan tanto en **tareas de clasificación** como en **tareas de regresión**, lo que les confiere, al igual que a algoritmos como SVM, un gran potencial para abordar una amplia variedad de casos de uso prácticos.

Se trata de una técnica **conceptualmente sencilla**, fácil de interpretar, y que además suele proporcionar **resultados bastante competitivos**, lo que explica su uso tan extendido tanto en entornos académicos como profesionales.

Concretamente, los árboles de decisión se encuadran dentro de la categoría de **algoritmos de aprendizaje supervisado**. Esto implica que el algoritmo recibe un **conjunto de datos etiquetados**, a partir del cual aprende a construir un modelo. Dicho modelo define un **límite de decisión** (o función hipótesis ajustada), aunque en este caso el proceso de construcción es muy diferente al de otros algoritmos vistos anteriormente.

Los árboles de decisión son **clasificadores no lineales**, lo que significa que pueden trabajar con conjuntos de datos que **no son linealmente separables**. A diferencia de algoritmos como la regresión logística o SVM, no construyen límites de decisión mediante líneas o planos rectos, sino a través de **particiones sucesivas del espacio de características.**

Una de sus características más importantes, y la que les otorga un gran potencial, es su capacidad para **predecir tanto valores continuos** dentro de un rango amplio (tareas de regresión) como **valores discretos**, pertenecientes a un conjunto limitado de clases (tareas de clasificación).

Finalmente, estos algoritmos toman decisiones estableciendo un conjunto de **reglas de decisión del tipo "if–then–else"**, que se aplican de forma jerárquica hasta llegar a una **predicción final**, ya sea una clase concreta o un valor numérico.

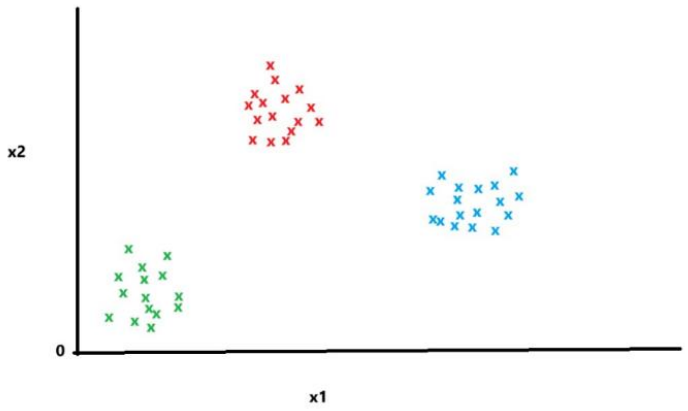
Para explicar en detalle cómo funciona un **árbol de decisión**, comenzaremos analizando su funcionamiento en **tareas de clasificación**, ya que este caso resulta más intuitivo y facilita la comprensión de los conceptos básicos.

Supongamos que disponemos de un **conjunto de datos** formado por dos **características de entrada**, que denominaremos x_1 y x_2, y una **variable de salida** y. En este ejemplo, la variable de salida puede tomar **tres valores distintos**, es decir, estamos ante un problema de **clasificación multiclase**.

$$(x_1^{(1)},x_2^{(1)},y^{(1)}),\ldots,(x_1^{(m)},x_2^{(m)},y^{(m)})$$

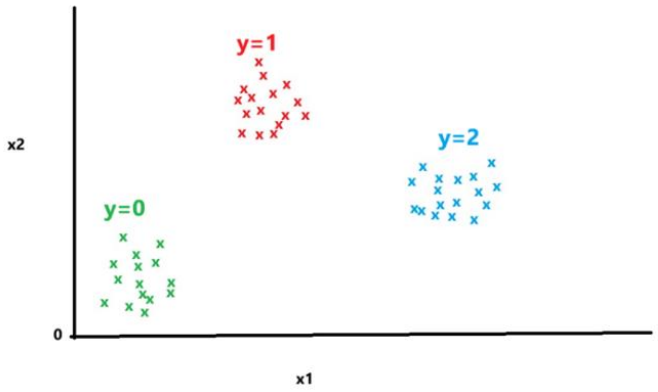
$$y \in \{0,1,2\}$$

Si representamos este conjunto de datos en forma de **tabla bidimensional**, cada una de las **filas** corresponderá a un **ejemplo del conjunto de entrenamiento**, mientras que cada una de las **columnas** representará una de las variables del problema: la característica de entrada x_1, la característica de entrada x_2 y, finalmente, la variable de salida y.



Veamos ahora un **ejemplo concreto**. Imaginemos que disponemos de **40 ejemplos** en nuestro conjunto de datos de entrenamiento. Si analizamos la distribución según el valor de la etiqueta de salida y, observamos lo siguiente:

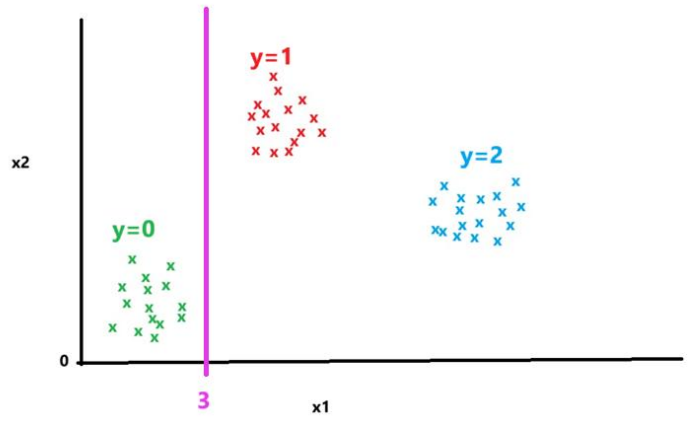
- 12 ejemplos pertenecen a la clase $y = 0$.
- 12 ejemplos pertenecen a la clase $y = 1$.
- 16 ejemplos pertenecen a la clase $y = 2$.



El objetivo del árbol de decisión será **dividir estos datos** en distintos subconjuntos, buscando **valores de corte** en las características de entrada que permitan separar lo mejor posible las distintas clases.

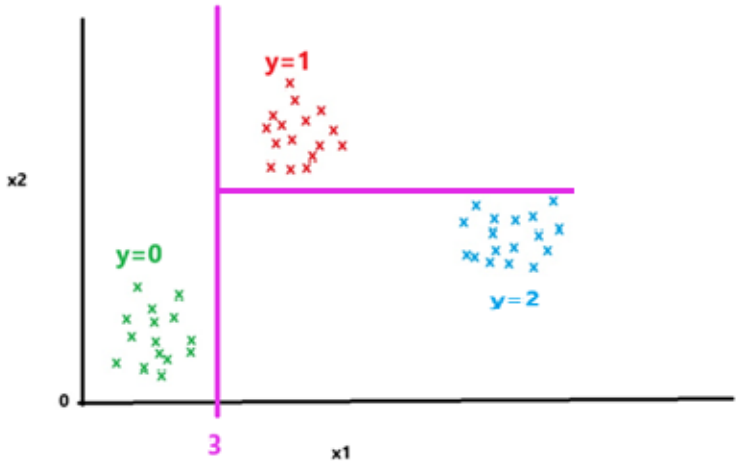
En primer lugar, construimos un **primer límite de decisión** que separe la clase $y = 0$ del resto. Para ello, establecemos la siguiente regla:

- Si $x_1 \leq 3$, el ejemplo se asigna al grupo de la clase $y = 0$.

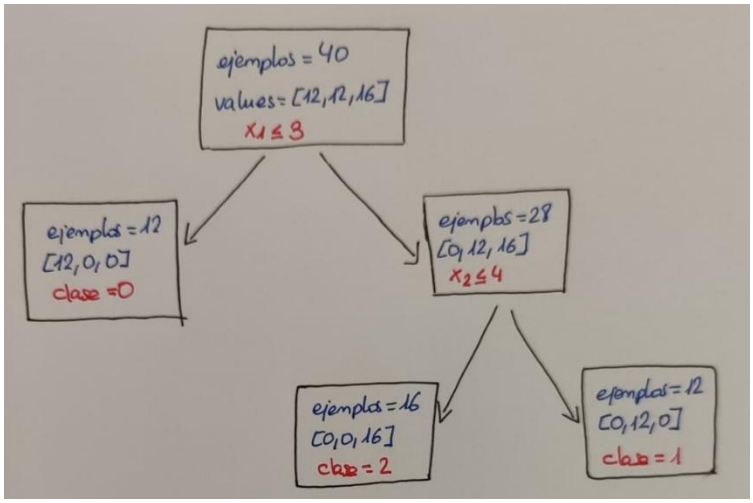


A continuación, para los ejemplos que no cumplen esta condición, establecemos un **segundo límite de decisión** con el objetivo de separar otra de las clases:

- Si $x_2 \leq 4$, el ejemplo se asigna a la clase $y = 2$.



Hasta ahora hemos descrito el proceso de forma **gráfica**, atendiendo a cómo se van realizando las separaciones en el espacio de características. Si representamos este mismo proceso **en forma de árbol**, obtendríamos una estructura de decisiones jerárquica como la siguiente:



- **Nodo raíz**
  - 40 ejemplos
  - Distribución de clases: $[12, 12, 16]$
  - Primera regla: $x_1 \leq 3$
- **Rama izquierda**
  - 12 ejemplos
  - Todos pertenecen a la clase $y = 0$
  - Es una **hoja** (nodo puro)
- **Rama derecha**
  - 28 ejemplos
  - Distribución: $[0, 12, 16]$
  - Segunda regla: $x_2 \leq 4$

- **Ramas finales**
  - Si $x_2 \leq 4 \rightarrow$ 16 ejemplos → **clase 2**
  - Si $x_2 > 4 \rightarrow$ 12 ejemplos → **clase 1**

Limitaciones de los árboles de decisión

A pesar de su sencillez e interpretabilidad, los **árboles de decisión presentan una serie de limitaciones** cuando se aplican a problemas reales. Más adelante veremos cómo muchas de estas limitaciones pueden **mitigarse mediante técnicas de *Ensemble Learning***, y en particular mediante el uso de **conjuntos de árboles**, como los **Random Forest**.

A continuación, se describen las **principales limitaciones** que deben tenerse en cuenta al utilizar un árbol de decisión en un caso de uso real:

I. Tendencia al sobreajuste (*overfitting*).

La mayoría de las implementaciones de árboles de decisión (como la utilizada en **Scikit-learn**) pueden llegar a generar **árboles excesivamente complejos**, lo que provoca **sobreajuste** sobre el conjunto de datos de entrenamiento.

El **sobreajuste** (*overfitting*) en *Machine Learning* ocurre cuando un modelo aprende en exceso los detalles específicos y el ruido de los datos de entrenamiento, perdiendo así la capacidad de **generalizar** a datos nuevos.

Un modelo sobreajustado presenta las siguientes características:

- Memoriza los datos de entrenamiento en lugar de aprender patrones generales.
- Obtiene un rendimiento muy alto en el conjunto de entrenamiento, pero bajo en datos de prueba.
- Se adapta demasiado a irregularidades o ruido presentes en los datos.

Este problema es especialmente frecuente cuando el conjunto de datos tiene **muchas características** o contiene **valores atípicos**, ya que el árbol puede generar un número muy elevado de ramas que dificultan la generalización.

II. Optimización basada en óptimos locales.

La construcción de un árbol de decisión se basa en la elección del **mejor corte en cada nodo**, comparando únicamente el nodo padre con sus posibles nodos hijos. Este enfoque implica que el algoritmo **optimiza de forma local**, sin tener en cuenta el efecto global sobre la estructura completa del árbol.

Como consecuencia, **no existe garantía de que el árbol obtenido sea óptimo a nivel global**, ya que decisiones aparentemente buenas a corto plazo pueden conducir a una estructura subóptima a largo plazo.

III. Inestabilidad del modelo.

Otra limitación importante de los árboles de decisión es su **inestabilidad**. **Una pequeña variación en el conjunto de datos de entrenamiento** puede dar lugar a un **árbol completamente diferente**.

Esto significa que, aunque dos conjuntos de datos sean muy similares (por ejemplo, difieran solo en unos pocos ejemplos que no alteran significativamente la tendencia general), el árbol generado puede cambiar notablemente su estructura. Esta sensibilidad hace que los árboles individuales sean poco robustos frente a pequeñas perturbaciones en los datos.

IV. Mitigación de las limitaciones.

Gran parte de estas limitaciones pueden **reducirse utilizando conjuntos de árboles que toman decisiones conjuntas**, como ocurre en los **Random Forest**. Estas técnicas combinan múltiples árboles entrenados sobre subconjuntos aleatorios de los datos, logrando modelos más **estables**, **robustos** y con **mejor capacidad de generalización**.

Además, en los árboles de decisión individuales, el sobreajuste puede atenuarse mediante el uso de **hiperparámetros o criterios de parada**, que permiten controlar la complejidad del modelo. Estos hiperparámetros regulan, entre otros aspectos, cómo crece el árbol y hasta qué punto se realizan las divisiones, evitando que el modelo se ajuste en exceso a los datos de entrenamiento.

2. ENSEMBLE LEARNING.

Vamos a introducir ahora una nueva construcción en el apartado siguiente denominada **Random Forest**, que se basa en un enfoque de aprendizaje conocido como **Ensemble Learning**. Este tipo de técnicas surge, precisamente, para **solucionar gran parte de las limitaciones** que presentan los árboles de decisión individuales.

El *Ensemble Learning* se fundamenta en un **principio clave**:

> *Si se combinan las predicciones de varios modelos, es muy probable obtener una predicción mejor que la proporcionada por un único modelo de forma individual.*

En otras palabras, en lugar de confiar en un solo algoritmo de clasificación, se entrenan **varios modelos**, potencialmente basados en algoritmos distintos, y posteriormente se **agregan sus predicciones**. En problemas de **clasificación**, esta agregación suele realizarse mediante un mecanismo de **votación**, quedándose con la clase que más veces ha sido predicha por los modelos. A este enfoque se le conoce como **Voting Classifier**.

El conjunto de modelos que colaboran para realizar una predicción conjunta recibe el nombre de **ensemble**, y la técnica empleada se denomina **Ensemble Learning**.

Este tipo de métodos se utiliza habitualmente en las **fases finales de un proyecto de *Machine Learning***.

- Primero se entrenan y evalúan distintos modelos de forma individual, ajustando sus hiperparámetros y comprobando su rendimiento.
- Una vez verificado que estos modelos funcionan correctamente, se pueden combinar para mejorar la calidad de las predicciones.

Por ejemplo, supongamos un problema de **clasificación de correos electrónicos como spam o no spam**. Podemos entrenar distintos modelos sobre el mismo conjunto de datos, como:

- un modelo de **regresión logística**,
- un modelo basado en **SVM**,
- y un **árbol de decisión**.

Cada uno de estos modelos genera su propia predicción. Mediante técnicas de *Ensemble Learning*, podemos **poner en común estas predicciones** y obtener una decisión final más robusta que la que produciría cualquiera de los modelos por separado.

La eficacia de este enfoque se apoya en una **propiedad estadística**: cuando se combinan las predicciones de varios modelos razonablemente buenos e independientes entre sí, la **probabilidad de error del conjunto disminuye**, aumentando así la calidad global de la predicción.

Existen varias técnicas principales dentro del *Ensemble Learning*, entre las que destacan:

- **Bagging.**
- **Pasting.**
- **Boosting.**
- **Stacking.**

En este tema nos centraremos en el método de **bagging**, que es la base sobre la que se construye el algoritmo **Random Forest**, uno de los ensembles más utilizados en *Machine Learning*.

Bagging paso a paso.

Bagging viene de *Bootstrap Aggregating* y consiste en **entrenar muchos modelos similares** (normalmente el mismo algoritmo) sobre **versiones distintas del conjunto de entrenamiento** y luego **combinar** sus predicciones. El funcionamiento es el siguiente:

1) Partimos del dataset original.
Tenemos un conjunto de entrenamiento D con Nejemplos.

2) Generamos varios "datasets bootstrap".
Creamos B subconjuntos $D_1, D_2, \dots, D_B$ mediante **muestreo con reemplazo** (*bootstrap*).

- Cada $D_b$tiene tamaño N(normalmente).
- Como hay reemplazo, algunos ejemplos se repiten y otros quedan fuera.
- Los ejemplos que no aparecen en $D_b$se llaman **out-of-bag (OOB)** para ese modelo.

3) Entrenamos un modelo por cada subconjunto.
Entrenamos el mismo tipo de modelo (por ejemplo, un **árbol de decisión**) en cada D_b. Resultado: tenemos B modelos $M_1, M_2, \dots, M_B$.

4) Combinamos predicciones (agregación).

- **Clasificación:** cada modelo vota una clase → se usa **votación mayoritaria**.
- **Regresión:** cada modelo predice un valor → se usa la **media** (o mediana) de predicciones.

Esto es la parte de *aggregating*: en vez de una decisión única, usamos una decisión "consensuada".

5) (Opcional, pero muy útil) Estimamos rendimiento con OOB.
Como cada modelo deja fuera algunos ejemplos, esos ejemplos OOB pueden usarse para evaluar sin un conjunto de validación aparte:

- Para cada ejemplo, se predice usando solo los modelos donde fue OOB.
- Se calcula la métrica final con esas predicciones. Esto se conoce como **OOB score**.

¿Por qué bagging ayuda con árboles de decisión?
Un árbol de decisión individual tiende a ser:

- **muy flexible** (puede ajustarse muchísimo),
- y **muy sensible a los datos** (inestable).

Bagging funciona especialmente bien cuando el "modelo base" tiene **alta varianza**, y los árboles la tienen.

3. RANDOM FOREST.

Un **Random Forest** se corresponde con un **ensemble de árboles de decisión**, es decir, un conjunto de árboles que colaboran para realizar una predicción conjunta en lugar de depender de un único modelo.

En lugar de entrenar un solo árbol, se entrenan **múltiples instancias de árboles de decisión** sobre el mismo conjunto de datos, pero introduciendo **aleatoriedad** tanto en los ejemplos como en las características.

El número de árboles que forman el *forest* es un **hiperparámetro** del modelo (por ejemplo, n_estimators en *Scikit-learn*). Este número puede ajustarse en función del problema y de la **capacidad de cómputo disponible**, ya que, en general, un mayor número de árboles suele mejorar el rendimiento hasta cierto punto, a costa de un mayor tiempo de entrenamiento.

Entrenamiento mediante bagging.

Random Forest se entrena utilizando la técnica de *Ensemble Learning* conocida como **bagging** (*Bootstrap Aggregating*). El proceso de construcción es el siguiente:

1. Partimos del **conjunto de datos de entrenamiento original**.

2. Generamos varios **subconjuntos bootstrap** mediante muestreo aleatorio **con reemplazo**, lo que implica que:
   - algunos ejemplos pueden repetirse dentro de un mismo subconjunto,
   - otros pueden no aparecer.

3. Con cada uno de estos subconjuntos se entrena una **instancia diferente de un árbol de decisión**. Este procedimiento permite que cada árbol vea una versión ligeramente distinta de los datos, lo que contribuye a reducir la **varianza** del modelo final.

   Aleatoriedad en la selección de características

   Además del muestreo aleatorio de ejemplos, Random Forest introduce una segunda fuente clave de aleatoriedad. En lugar de buscar el mejor par *(característica, valor de corte)* entre **todas las características disponibles**, cada árbol (y más concretamente, cada nodo del árbol) selecciona un **subconjunto aleatorio de características** y busca el mejor corte únicamente dentro de ese subconjunto. Esta es precisamente la razón por la que el algoritmo recibe el nombre de **Random Forest**. Como consecuencia:
   - cada árbol utiliza **datos distintos** (bagging),
   - y también **características distintas** en cada división. Esto provoca que las predicciones realizadas por los distintos árboles sean **menos correlacionadas entre sí**, lo que mejora la calidad del ensemble.

Gracias a esta combinación de técnicas, Random Forest consigue mitigar varias de las limitaciones de los árboles de decisión individuales:

- **Reduce el sobreajuste**, al promediar o votar las predicciones de múltiples árboles.
- **Disminuye la inestabilidad**, ya que pequeñas variaciones en los datos no afectan de forma drástica al modelo final.
- **Aumenta la capacidad de generalización**, al evitar que todos los árboles aprendan exactamente los mismos patrones.

En resumen, Random Forest combina **bagging + selección aleatoria de características + agregación de predicciones**, dando lugar a un modelo más **robusto, estable y preciso** que un árbol de decisión individual.

Predicción final en Random Forest

Una vez que todos los árboles del *Random Forest* han sido entrenados, el modelo realiza la **predicción final combinando las predicciones individuales** de cada árbol. El mecanismo concreto depende del tipo de problema:

- **Predicción en problemas de clasificación (votación):**

En **clasificación**, cada árbol del bosque realiza una **predicción independiente** sobre la clase a la que pertenece un ejemplo.

- Cada árbol "emite un voto" por una clase.
- La predicción final del Random Forest es la **clase que obtiene mayor número de votos** entre todos los árboles.

Este proceso se conoce como **votación mayoritaria** (*majority voting*). Gracias a este enfoque, aunque algunos árboles se equivoquen, el resultado final suele ser correcto siempre que la mayoría de los árboles capturen correctamente el patrón general de los datos.

- **Predicción en problemas de regresión (media):**

En **regresión**, cada árbol predice un **valor numérico continuo**.

- La predicción final del Random Forest se obtiene calculando la **media aritmética** de todas las predicciones individuales.
- En algunos casos también puede utilizarse la mediana, aunque la media es el enfoque habitual.

Este promedio suaviza las predicciones extremas y reduce el efecto de árboles que puedan haber aprendido ruido.

Hiperparámetros clave de Random Forest

El comportamiento y rendimiento de un Random Forest se controla mediante una serie de **hiperparámetros**, que permiten ajustar la complejidad del modelo y su capacidad de generalización.

n_estimators

- Indica el **número de árboles** que forman el bosque.
- Valores más altos suelen mejorar el rendimiento y reducir la varianza.
- Aumentar este valor incrementa el **coste computacional**, pero no suele provocar sobreajuste.
- Es uno de los hiperparámetros más importantes.

max_depth

- Define la **profundidad máxima** de cada árbol.
- Si no se limita, los árboles pueden crecer mucho y sobreajustar.
- Valores más pequeños generan árboles más simples y generalizan mejor.

max_features

- Especifica el **número de características** que se consideran en cada nodo para buscar el mejor corte.
- Introduce aleatoriedad y reduce la correlación entre árboles.
- Valores habituales:
  - Clasificación: $\sqrt{n\_features}$
  - Regresión: $n\_features$

  Este hiperparámetro es clave para que Random Forest sea realmente "random".

min_samples_split

- Número mínimo de ejemplos necesarios para dividir un nodo.
- Evita divisiones basadas en muy pocos datos.
- Ayuda a reducir el sobreajuste.

min_samples_leaf

- Número mínimo de ejemplos que debe tener una hoja.
- Fuerza hojas más "grandes" y estables.
- Muy útil cuando hay ruido en los datos.

bootstrap

- Indica si se utiliza muestreo con reemplazo (bagging).
- En Random Forest suele estar activado (True).
- Permite el uso de muestras **out-of-bag (OOB)** para evaluación.