

Procedimientos de Machine Learning

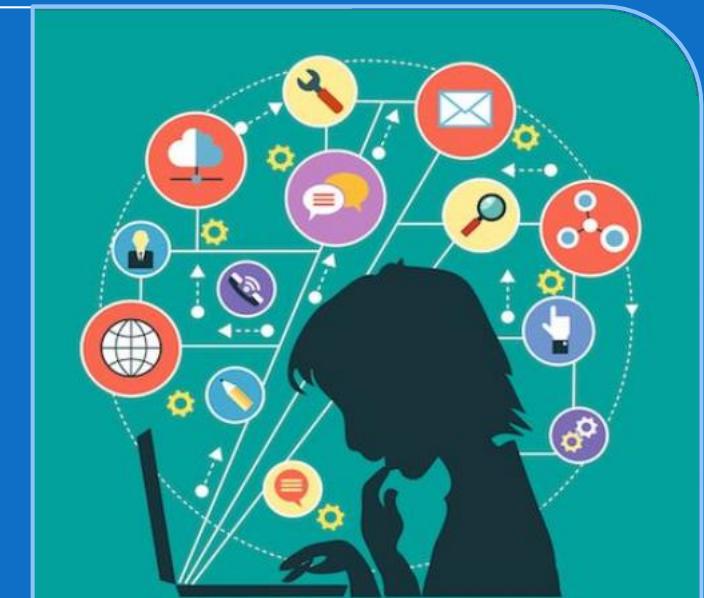
Preprocesamiento de *machine learning*



Modelos de Inteligencia Artificial

Profesor:
Jose Luis Domenech Ballester

Especialización en: Inteligencia Artificial y Big Data

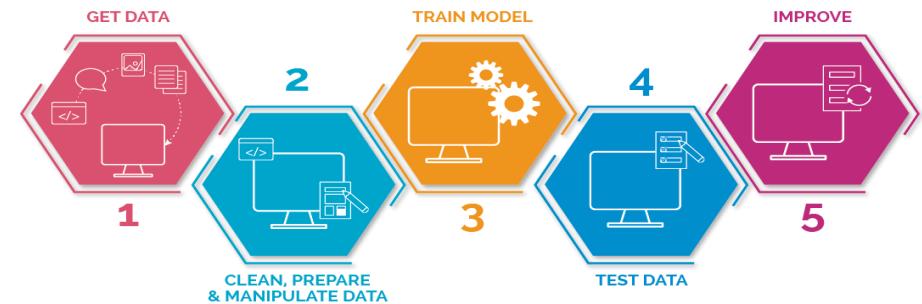


1. Introducción

¿Qué es el preprocessamiento de datos y por qué es tan importante?

El **preprocesamiento de datos** es el conjunto de técnicas y pasos que se aplican a los datos **antes de entrenar un modelo de Inteligencia Artificial**, con el objetivo de:

- Limpiar los datos
- Transformarlos
- Adaptarlos al formato que los algoritmos pueden utilizar



- ***Los algoritmos no aprenden de los datos reales, aprenden de cómo se los damos.***
- ***En Machine Learning, el 70–80% del trabajo está en los datos, no en el modelo.***

1. Introducción

¿Por qué es tan importante el Preprocesamiento?

Datos reales suelen ser:

- Incompletos (valores faltantes)
- Inconsistentes
- Desbalanceados
- Con ruido
- Con escalas muy diferentes
- Con texto o categorías

Motivos principales:

- 1.Los algoritmos no manejan bien errores
- 2.Muchos algoritmos no aceptan texto
- 3.Las escalas afectan al aprendizaje
- 4.El ruido genera sobreajuste
- 5.Los datos desbalanceados engañan las métricas



Datos ideales para los algoritmos:

- Numéricos
- Limpios
- Escalados
- Representativos
- Sin fugas de información

- *Los datos reales casi nunca están listos para entrenar un modelo.*
- *Cambiar de algoritmo suele mejorar poco.*
- *Mejorar los datos suele mejorar mucho.*

2. Flujo del Preprocesamiento de Datos

Preprocesamiento Básico

1. Análisis Exploratorio de Datos (EDA)
2. Limpieza de datos
3. Tratamiento de valores faltantes
4. Tratamiento de outliers
5. Codificación de variables categóricas
6. Escalado



Preprocesamiento Avanzado

7. Transformación y Creación de Nuevas Variables
8. Selección de características
9. Reducción de dimensionalidad
10. Balanceo de clases

Evaluación ligada al preprocesamiento

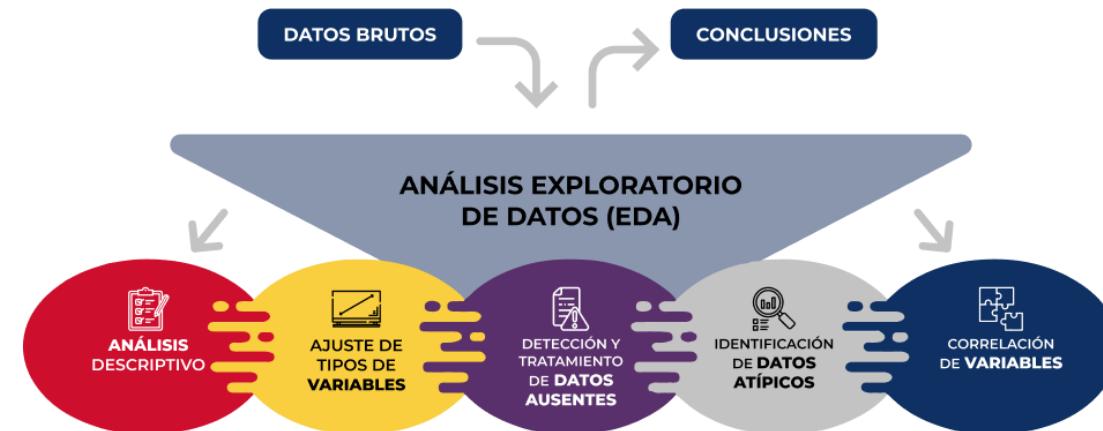
11. Validación cruzada
12. Data leakage
13. Pipelines

3. Análisis Exploratorio de Datos (EDA)

3.1 ¿Qué es la exploración de datos (EDA)?

La **Exploración Inicial de los Datos (EDA)** es el proceso de analizar y entender un dataset antes de modificarlo, con el objetivo de:

- Conocer su estructura
- Detectar problemas
- Tomar decisiones informadas de preprocessamiento



No se puede limpiar ni transformar lo que no se entiende.

3. Análisis Exploratorio de Datos (EDA)

3.2 ¿Por qué es obligatoria antes de preprocesar?

Sin EDA:

- Se eliminan datos útiles
- Se imputan valores incorrectos
- Se escalan variables que no deberían



Con EDA:

- Las decisiones son justificables
- El preprocesamiento tiene sentido
- El modelo mejora

Preguntas que debe responder el EDA:

- ¿Cuántas filas y columnas hay?
- ¿Qué tipo de variables tengo?
- ¿Hay valores faltantes?
- ¿Existen valores extremos?
- ¿Las variables tienen sentido?
- ¿Está balanceada la variable objetivo?

Después del EDA deberíamos saber:

- ✓ Qué columnas limpiar
- ✓ Qué columnas transformar
- ✓ Qué columnas eliminar
- ✓ Qué problemas existen
- ✓ Qué técnicas necesitaremos después



EDA no es mirar datos, es entenderlos.

El EDA no arregla nada, pero te dice qué arreglar.

4. Limpieza de Datos (Data Cleaning)

¿Qué es la limpieza de datos?

La limpieza de datos es el proceso de **corregir o eliminar datos incorrectos, inconsistentes o irrelevantes** del dataset.



No todo dato es información.



¿Por qué es necesaria?

Los modelos de IA:

- No detectan errores humanos
- No distinguen valores “raros” de valores “válidos”
- Aprenden patrones incorrectos si los datos están sucios



Un modelo aprende errores si los errores están en los datos.

Problemas típicos en datos reales:

- Filas duplicadas
- Valores inconsistentes
- Errores de captura
- Variables irrelevantes
- Formatos incorrectos



*El objetivo no es “dejar bonito el dataset”, sino **hacerlo fiable**.*

5. Tratamiento de valores faltantes (Missing Values)

5.1 ¿Qué es?

Los **valores faltantes** (missing values) son datos ausentes en el dataset (NaN, None, NaT, cadenas vacías, “NA”, “?”...).

Aparecen por errores de captura, encuestas incompletas, sensores, integraciones entre sistemas, etc.

Tipos de missing:

- **MCAR** (Missing Completely At Random): falta al azar (p.ej., error puntual).
- **MAR** (Missing At Random): falta depende de otras variables (p.ej., ingresos faltan más en jóvenes).
- **MNAR** (Missing Not At Random): falta por el propio valor (p.ej., no declaran ingresos altos).

5. Tratamiento de valores faltantes (Missing Values)

MCAR (Missing Completely At Random)

Falta completamente al azar, sin depender de ninguna variable ni del propio valor.

◊ Ejemplo:

- Se pierde un dato por un error puntual del sistema
- Un sensor falla aleatoriamente
- Un formulario se corrompe al guardarse



Consecuencia:

- No introduce sesgo (solo pierdes datos)

5. Tratamiento de valores faltantes (Missing Values)

⚠ MAR (Missing At Random)

La ausencia del dato depende de otras variables observadas, pero no del propio valor faltante.

◊ Ejemplo:

- Los ingresos faltan más en personas jóvenes
- Falta un campo más a menudo en usuarios con cierto perfil
- Un test no se hace si el paciente es menor de X años

🧠 Consecuencia:

- Hay sesgo, pero puede corregirse usando otras variables

5. Tratamiento de valores faltantes (Missing Values)



MNAR (Missing Not At Random)

La ausencia del dato depende del propio valor que falta.

◊ Ejemplo:

- Personas con ingresos muy altos no los declaran
- Pacientes con síntomas graves no responden a cierta pregunta
- Notas muy malas no se registran “a propósito”



Consecuencia:

- Es el caso más peligroso: el sesgo está en el propio dato oculto
- Difícil de corregir solo con técnicas automáticas

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.1 Eliminar filas (listwise deletion)

☞ Útil si:

- Hay **muy pocos missing** (ej. < 1–5%)
- El dataset es grande
- Falta al azar (MCAR) y no te sesga

⚠ Riesgo:

- Pierdes datos y puedes introducir sesgo si el missing no es aleatorio.

5.2.2 Eliminar columnas (Drop columns)

Motivo

Si una columna está casi vacía, imputarla es “inventar” demasiadas cosas.

☞ Cuándo es mejor

- Missing muy alto (\approx 60–80% o más)
- La variable no es esencial o hay alternativas
- El coste de imputar supera el beneficio

⚠ Riesgos

- Perder una variable importante de negocio



Antes de borrar, pregunta: “¿esta variable es imprescindible para el objetivo?”

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.3 Imputación Simple

Imputar datos significa **rellenar valores faltantes** (missing values) con valores estimados, siguiendo una estrategia concreta, para poder trabajar con el dataset sin perder filas o columnas.

5.2.3.1 Media

Qué hace:

Sustituye los valores faltantes por la media.

☞ **Cuándo usarla:**

- Distribución aproximadamente normal
- Pocos outliers

⚠ **Riesgos:**

- Sensible a valores extremos
- Reduce variabilidad

5.2.3.2 Mediana

Qué hace:

Sustituye por el valor central.

☞ **Cuándo usarla:**

- Distribuciones sesgadas
- Presencia de outliers

⚠ **Riesgos:**

- Puede ocultar patrones reales

5.2.3.3 Moda

Qué hace:

Sustituye por la categoría más frecuente.

☞ **Cuándo usarla:**

- Variables categóricas
- Pocos niveles

⚠ **Riesgos:**

- Refuerza la categoría dominante

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.4 Imputación por constante (“desconocido”, “no informado”)

Motivo

A veces el missing es información: el usuario no lo declara, el sensor falla, etc.

☞ Cuándo es mejor:

- MNAR sospechoso
- Variables categóricas (“ocupación”, “sexo”, “nivel estudios”)

⚠ Quieres mantener trazabilidad de “faltaba”

Riesgos

- “Desconocido” puede convertirse en categoría dominante
- En numéricas, un 0 puede significar algo real (cuidado)

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.5 Imputación por grupos (Group-wise)

Motivo

Es una técnica para **rellenar valores faltantes (missing)** usando **estadísticos calculados dentro de cada grupo**, en lugar de usar un valor global.

No es lo mismo imputar ingresos en Madrid que en un pueblo

☞ Cuándo es mejor:

- MAR(falta al azar) claro (depende de otra variable)
- Tienes una variable que agrupa bien (país, provincia, sector, categoría)
- El grupo tiene suficiente tamaño

⚠ Riesgos

- Grupos muy pequeños → imputación inestable
- Si el missing está concentrado en un grupo, puede falsear ese grupo

Ejemplo sencillo (el de Madrid vs pueblo)

Supón este dataset:

Persona	Ciudad	Ingresos
A	Madrid	28000
B	Madrid	32000
C	Pueblo	15000
D	Pueblo	✗

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.6 Indicador de missing (Missing flag)

Motivo

Consiste en **crear una nueva variable binaria** que indique **si un dato estaba ausente o no, además de imputar el valor**.

No sustituye a la imputación: **la complementa**.

☞ Cuándo es mejor:

- MNAR o sospecha de patrón
- Si sospechas que los *missing* tienen significado
- En variables como ingresos, historial, datos opcionales
- Cuando imputas pero no quieres perder la señal “faltaba”

⚠ Riesgos:

- Si el missing es aleatorio, el flag no ayuda (pero suele no molestar)



Lo correcto en ML es:

☞ Probar con y sin indicador y comparar métricas

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.7 KNN Imputer (vecinos)

Motivo

Imputa usando ejemplos “parecidos” según otras variables que estén relacionadas entre si.

KNN imputa usando distancias, y las distancias **solo tienen sentido entre variables numéricas relacionadas** con la que queremos imputar.

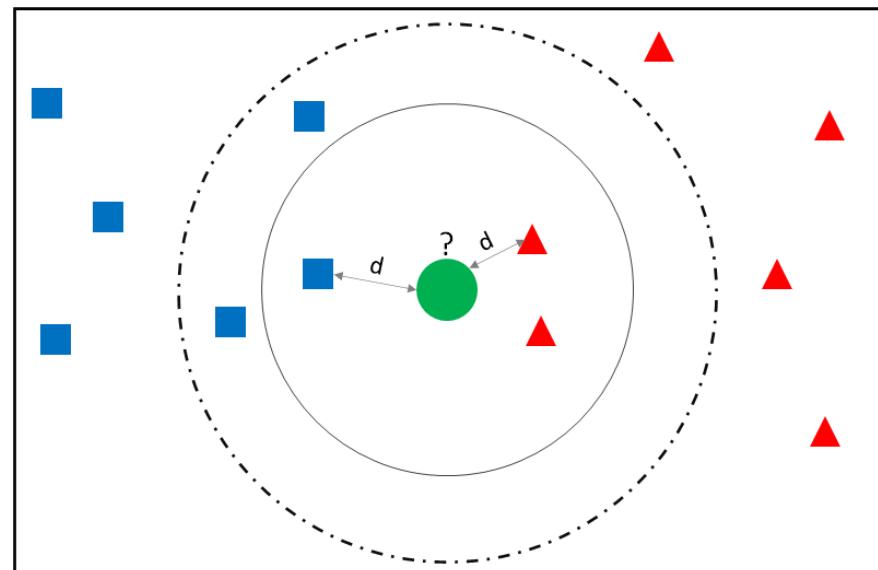
Es decir, para calcular el valor con **KNN** solo utilizar las columnas del dataset correlacionadas.

Cuándo es mejor:

- Variables correlacionadas
- Datasets pequeños
- MAR moderado
- Quieres algo mejor que media/mediana sin modelar demasiado

Riesgos:

- Sensible a escalado (hay que escalar numéricas)
- Costoso en grandes datasets
- Puede “inventar” valores no realistas si hay ruido



5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.7 IterativeImputer (model-based)

Cada variable con missing se predice con un modelo usando las demás.

La idea es, **si falta un valor, lo predigo usando el resto de columnas como si fuera un problema de ML**.

Por defecto usa el estimador **BayesianRidge** (regresión lineal bayesiana) pero se pueden usar otros.

Lo que hace IterativeImputer:

1. **Elige una columna con missing**, por ejemplo ingresos
2. Usa las otras columnas (edad, experiencia, ...) como features
3. Entrena un modelo SOLO con las filas donde ingresos NO falta
4. **Predice los ingresos faltantes**
5. Pasa a la siguiente columna con missing
6. Repite el proceso varias veces (iterative)

☞ Cuándo es mejor:

- MAR claro y relaciones entre variables
- Quieres imputación “inteligente”
- Dataset mediano (coste mayor)

⚠ Riesgos:

- Puede introducir patrones falsos si MNAR
- Es fácil pasarse de listo sin validar
- Requiere más cuidado (y reproducibilidad)

5. Tratamiento de valores faltantes (Missing Values)

5.2 Técnicas Principales

5.2.8 Series temporales (ffill/bfill/interpolate)

Motivo

En el tiempo, el valor suele depender del anterior o del siguiente.

Forward fill (ffill)

Rellena con el último valor conocido.

10 → 11 → **X** → 12
10 → 11 → 11 → 12

- ✓ Muy usado en sensores y logs
- ✓ Asume continuidad
- ✗ Malo si hay saltos bruscos

Backward fill (bfill)

Rellena con el siguiente valor conocido.

10 → 11 → **X** → 12
10 → 11 → 12 → 12

- ☞ Usa información futura
- ⚠ Peligroso en ML predictivo

Interpolación (interpolate)

Calcula un valor intermedio.

10 → 11 → **X** → 12
10 → 11 → 11.5 → 12

- ✓ Ideal si el fenómeno cambia gradualmente
- ✓ Muy común en datos físicos
- ✗ Inventa valores (ojo)

6. Tratamiento de valores atípicos (outliers)

¿Qué es?

Un outlier es un valor que **se aleja significativamente del patrón general** de los datos.

¿Por qué son importantes en Machine Learning?

Los outliers pueden:

- Distorsionar medias y varianzas
- Afectar gravemente a modelos sensibles a escala
- Provocar sobreajuste
- Empeorar la convergencia del entrenamiento

👉 Qué hacer con ellos:

- Eliminar (si son errores)
- Limitar (clipping)
- Mantener (si son casos reales importantes)

📌 Modelos especialmente sensibles:

- Regresión lineal
- KNN
- SVM
- Redes neuronales

📌 Modelos más robustos:

- Árboles de decisión
- Random Forest

🔍 Cómo detectarlos:

- Boxplots
- Rango intercuartílico (IQR)
- Z-score (desviaciones estándar)

6. Tratamiento de valores atípicos (outliers)

¿Qué es?

Un outlier es un valor que **se aleja significativamente del patrón general** de los datos.

¿Por qué son importantes en Machine Learning?

Los outliers pueden:

- Distorsionar medias y varianzas
- Afectar gravemente a modelos sensibles a escala
- Provocar sobreajuste
- Empeorar la convergencia del entrenamiento

👉 Qué hacer con ellos:

- Eliminar (si son errores)
- Limitar (clipping)
- Mantener (si son casos reales importantes)

📌 Modelos especialmente sensibles:

- Regresión lineal
- KNN
- SVM
- Redes neuronales

📌 Modelos más robustos:

- Árboles de decisión
- Random Forest

🔍 Cómo detectarlos:

- Boxplots
- Rango intercuartílico (IQR)
- Z-score (desviaciones estándar)

6. Tratamiento de valores atípicos (outliers)

6.1 ¿Cómo Detectar los Outliers?

- **Visuales:** boxplot, scatter, etc.
- **Estadísticos:** IQR, Z-score, MAD, percentiles
- **Distancia / densidad:** KNN, LOF, DBSCAN
- **Modelos:** Isolation Forest, One-Class SVM, Autoencoders

6. Tratamiento de valores atípicos (outliers)

6.1.1 Rango intercuartílico (IQR)

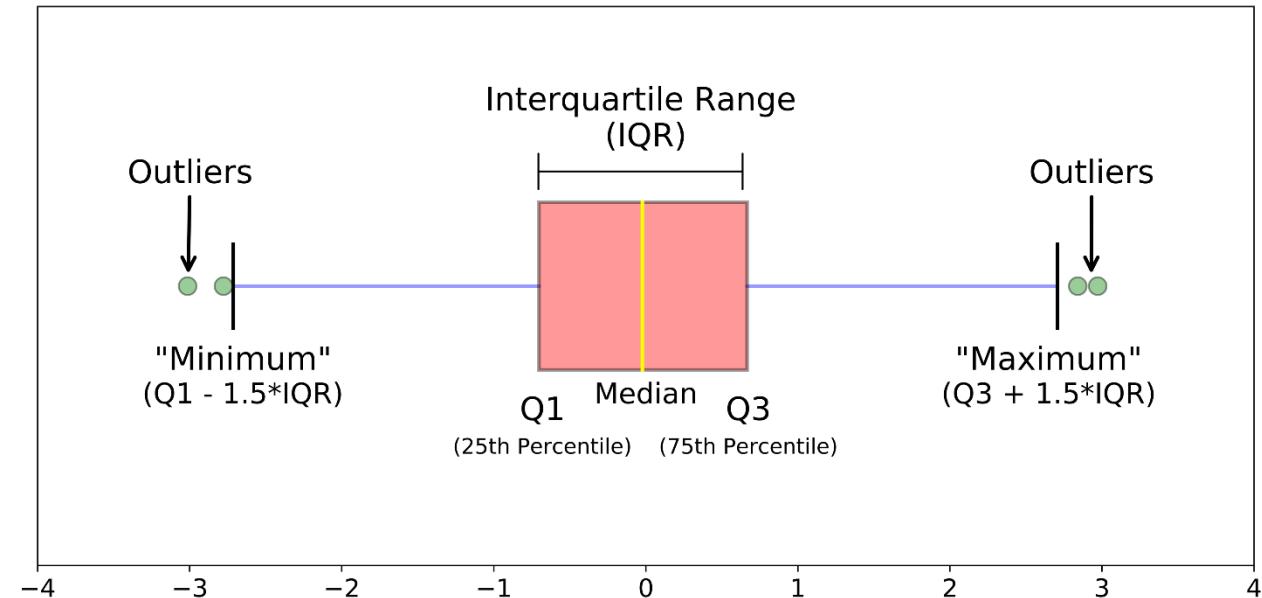
¿Qué es el IQR?

Se define como: $IQR = Q3 - Q1$

Donde:

- **Q1 (primer cuartil)** = percentil 25
- **Q3 (tercer cuartil)** = percentil 75

El IQR contiene el **50% central de los datos**



🧠 ¿Para qué sirve?

- Medir **cuánta dispersión** hay en la parte central de los datos
- **Detectar valores atípicos (outliers)** de forma robusta

Se define un rango “normal”:

$$\text{Lower/Minimum} = Q1 - 1.5 * \text{IQR}$$

$$\text{Upper/Maximum} = Q3 + 1.5 * \text{IQR}$$

Valores fuera de ese rango
☞ **posibles outliers**

6. Tratamiento de valores atípicos (outliers)

6.1.1 Rango intercuartílico (IQR)

💡 ¿Por qué es robusto?

Porque:

- Usa **percentiles**, no la media
- No se ve afectado por valores extremos
- Funciona bien con distribuciones **asimétricas** o con outliers fuertes



Cuándo usar el IQR

- Cuando tus datos **no siguen una distribución normal**
- Cuando sospechas que hay **outliers fuertes**
- En análisis exploratorio de datos (EDA)
- Analizas **una columna cada vez**
Es el método típico detrás del **boxplot**
- Cuando quieres una **regla simple y explicable**



Cuándo no es ideal:

- Cuando los datos son claramente **normales y limpios**
un **Z-score** puede ser más natural y preciso.
- Cuando los outliers son **casos reales e importantes**

6. Tratamiento de valores atípicos (outliers)

6.1.2 Z-score (desviaciones estándar)

¿Qué es el IQR?

Se define como:
$$z = \frac{X - \mu}{\sigma}$$

Donde:

- x = valor del dato
- μ = media
- σ = desviación estándar

El resultado te dice:

“Este valor está a z desviaciones estándar de la media”.

 ¿Para qué sirve?

Principalmente para:

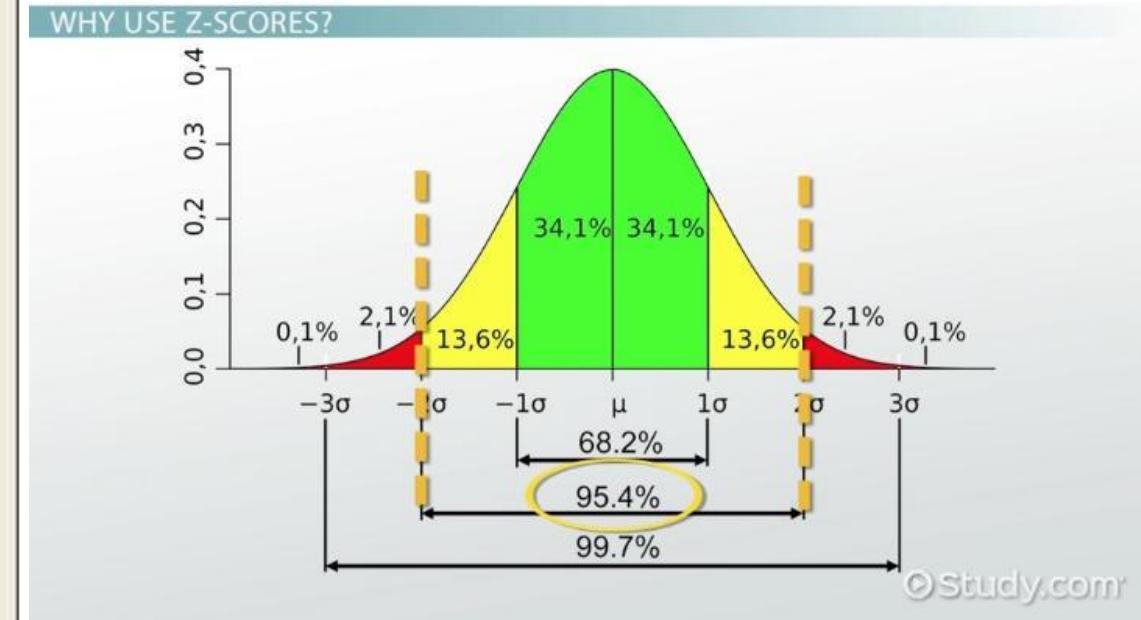
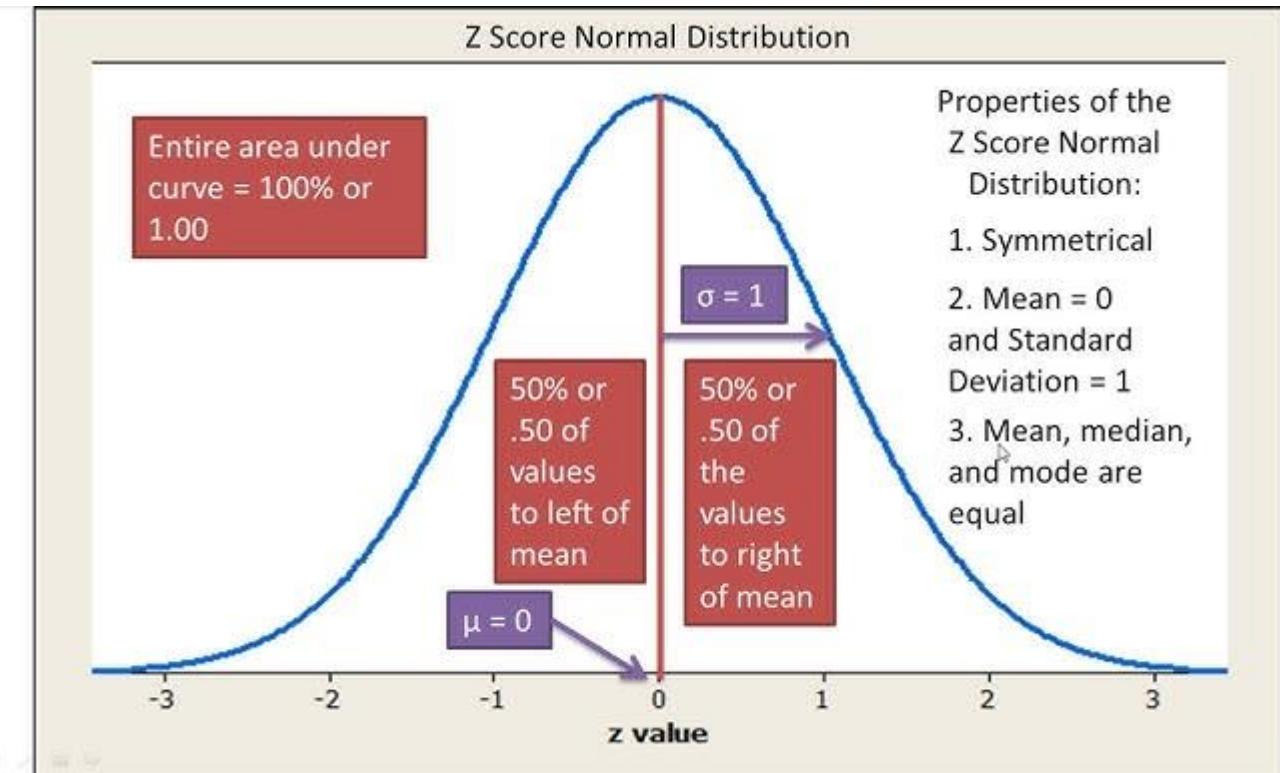
- **Detectar outliers**
- **Estandarizar variables** (ponerlas en escala media 0, desviación 1)
- Comparar valores en distintas escalas

Regla típica para outliers:

 Si $|z| > 3 \rightarrow$ posible outlier

6. Tratamiento de valores atípicos (outliers)

6.1.2 Z-score (desviaciones estándar)



6. Tratamiento de valores atípicos (outliers)

6.1.2 Z-score (desviaciones estándar)

Cuándo usar Z-score:

1. Cuando los datos son aproximadamente normales
2. Cuando el dataset está bastante limpio.
 - Sin muchos valores raros
3. En entornos controlados
 - Resumen de calidad, fabricación, laboratorio, etc.
 - Donde un valor a $\pm 3\sigma$ suele ser realmente anómalo.
4. Cuando quieres una regla simple y estándar
 - “Más de 3 desviaciones estándar” es una regla clásica
 - Fácil de explicar y justificar

Cuándo NO usar Z-score:

1. Cuando los datos son asimétricos o sesgados
2. Cuando ya hay outliers fuertes en los datos
 - Esos outliers influyen en la media y la desviación
3. Cuando el problema es no lineal o multivariante
 - Z-score es univariante
 - No detecta combinaciones raras de variables
4. Cuando los outliers son casos reales importantes

6. Tratamiento de valores atípicos (outliers)

IQR vs Z-score

Aspecto	Z-Score	IQR
Se basa en	Media y desviación	Cuartiles
Sensible a outliers	<input checked="" type="checkbox"/> Mucho	<input checked="" type="checkbox"/> Poco
Robusto	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> Sí
Supone normalidad	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> No
Funciona bien con datos sesgados	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> Sí
Uso típico	Datos “limpios” y normales	Datos reales y asimétricos



Z-Score funciona bien si los datos son “bonitos” y normales.

IQR funciona mejor cuando los datos son “reales” y tienen colas y valores extremos.

6. Tratamiento de valores atípicos (outliers)

6.2 Tratamiento de los Outliers

1. Eliminación de outliers (trimming)

Borrar las filas con valores extremos.

2. Recorte / Clipping

Limitar valores a un rango. Reducir impacto

3. Transformaciones (log, sqrt, Box-Cox, Yeo-Johnson)

Cambiar la escala para **comprimir valores grandes** (ingresos, precios, etc.)

4. Imputación / Reemplazo por estadísticos robustos

Cambiar outliers por:Mediana,Percentiles,Moda. Sospechas outlier es un error

6. Tratamiento de valores atípicos (outliers)

6.2.1 Eliminación de outliers (trimming)

Qué es:

Borrar las filas con valores extremos.

 Cuándo usarlo:

- Sabes que son **errores de medición/entrada**
- Dataset grande (perder algunas filas no duele)
- Los outliers **no representan casos reales**

 Ventajas:

- Simple y efectivo
- Mejora mucho modelos sensibles
(lineal, KNN, K-means...)
- Reduce ruido

 Desventajas:

- Pierdes información
- Puedes eliminar **casos raros pero importantes**
- Introduce sesgo si no eran errores reales

6. Tratamiento de valores atípicos (outliers)

6.2.2 Recorte/ Limitar (Clipping)



Qué es:

Limitar valores a un rango:

- Si $x < p1 \rightarrow p1$ Ej: clip(lower, upper)
- Si $x > p99 \rightarrow p99$

Cuándo usarlo:

- Los valores extremos son **posibles pero exagerados**
- No quieres perder filas
- Quieres **reducir su impacto**, no borrarlos

Ventajas:

- Conservas todas las muestras
- Reduce mucho el efecto de extremos
- Fácil de explicar e implementar

Desventajas:

- Distorsiona los valores originales
- Mete valores “artificiales”
- Puede ocultar problemas reales del dato

7. Codificación de variables categóricas

¿Por qué hay que codificar las variables categóricas?

Problema

La mayoría de algoritmos de Machine Learning:

- Solo trabajan con números
- No entienden texto ni etiquetas

Técnicas principales de codificación:

- Label Encoding - Ordinal Encoding (sklearn)
- One-Hot Encoding (OHE)
- Encoding por frecuencia
- Target Encoding (conceptual / avanzado)

7. Codificación de variables categóricas

7.1 Label Encoding

¿Qué hace?

Convierte categorías en números enteros.

¿Para qué sirve?

- Para **identificar categorías** con números
- Es común en:
- Variables objetivo (y) en clasificación
- Modelos basados en árboles (a veces)
- Casos donde el número es solo un **código**

⚠ Riesgos

- Introduce orden artificial si se usa mal
- El modelo puede interpretar un orden o distancia que no existe:



Cuándo usar Label Encoding

- Para la **variable objetivo (y)** en clasificación
- Con **modelos basados en árboles** (suelen ser más robustos)
- Cuando el número es solo un **ID/código**



Cuándo NO usarlo

- En features categóricas **sin orden** con modelos lineales.
- Cuando ese “orden artificial” puede confundir al modelo



7. Codificación de variables categóricas

7.2 One-Hot Encoding (OHE)

Qué hace

Crea una columna binaria por categoría con valores 0 o 1.

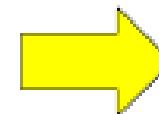
🧠 Cuándo usarla:

- Variables categóricas **nominales** (sin orden)
- Cuando el número de categorías es **moderado**
- Con modelos sensibles a la escala y a la distancia
Regresión, KNN, SVM, redes neuronales

⚠️ Riesgos

- Muchas categorías → muchas columnas
- Riesgo de multicolinealidad (se soluciona con `drop_first`)

Color
Red
Red
Yellow
Green
Yellow



	Red	Yellow	Green
Red	1	0	0
Yellow	1	0	0
Green	0	1	0
Yellow	0	0	1

7. Codificación de variables categóricas

Label Encoding vs OHE

Aspecto	Label Encoding	One-Hot Encoding (OHE)
Qué hace	Asigna un número entero a cada categoría	Crea una columna binaria (0/1) por cada categoría
Ejemplo	rojo=0, verde=1, azul=2	color_rojo, color_verde, color_azul
Introduce orden	⚠ Sí, artificial (aunque no quieras)	✗ No, todas las categorías son independientes
Introduce distancias	⚠ Sí (1 está “más cerca” de 2 que de 0)	✗ No hay distancias entre categorías
Nº de columnas	<input checked="" type="checkbox"/> 1 sola columna	✗ k columnas (o k-1 con drop="first")
Riesgo principal	✗ El modelo puede aprender orden/diferencias falsas	✗ Aumenta mucho la dimensionalidad
Modelos adecuados	♣ Árboles, Random Forest, GBM; variable objetivo (y)	📐 Modelos lineales, KNN, SVM, redes neuronales
Tipo de variable ideal	Nominal (solo como ID) o target	Nominal (sin orden)
Interpretabilidad	Media (el número no tiene significado real)	Alta (cada columna = una categoría)

7. Codificación de variables categóricas

Label Encoding vs OHE

Cuándo usar Label Encoding

- ✓ Para la **variable objetivo (y)** en clasificación
- ✓ Con **modelos basados en árboles** (Decision Tree, Random Forest, XGBoost...)
- ✗ No recomendado para features categóricas nominales en modelos lineales o de distancia

Cuándo usar One-Hot Encoding (OHE)

- ✓ Para **variables categóricas nominales** (sin orden)
- ✓ Con modelos:
 - Regresión lineal / logística
 - KNN
 - SVM
 - Redes neuronales
- ✓ Cuando el número de categorías es **bajo o medio**
- ⚠ Cuidado si hay **muchísimas categorías**
(explota en columnas)



Regla rápida:

- ¿La variable **no tiene orden?**
OHE
- ¿Usas **árboles** o es el **target**?
Label Encoding
- ¿Hay muchas categorías?
Piensa en **alternativas a OHE**

7. Codificación de variables categóricas

7.4 Encoding por frecuencia (Frequency Encoding)

¿En qué consiste?

El **encoding por frecuencia** sustituye cada categoría por un número que representa **lo frecuente que es esa categoría en el dataset**.

Hay dos variantes muy usadas:

1. Frequency encoding (conteo)

Reemplaza por el **número de veces** que aparece la categoría.

2. Relative frequency encoding (proporción)

Reemplaza por el **porcentaje / probabilidad** de aparición.

7. Codificación de variables categóricas

7.4 Encoding por frecuencia (Frequency Encoding)

Conteo:

color	color_freq
rojo	50
azul	30
verde	20

Proporción (o Relativa):

color	color_freq
rojo	0.5
azul	0.3
verde	0.2

7. Codificación de variables categóricas

7.3 Encoding por frecuencia (Frequency Encoding)

¿Cuándo usarla?

👉 Cuando hay muchas categorías

One-Hot con alta cardinalidad explota en muchas columnas.

Ejemplos:

ciudad, barrio, producto, código

Frequency encoding mantiene **1 sola columna**.

⚠️ Riesgos

1) Introduce un “orden artificial”

El modelo podría interpretar:

- 0.50 “es mayor” que 0.20
- y que esa diferencia tiene significado lineal

2) Puede meter “información global” (data leakage)

Si calculas frecuencias con todo el dataset antes del split, estás usando datos del test.

En producción:

- se calcula **solo en entrenamiento**
- y se aplica al resto.

7. Codificación de variables categóricas

7.3 Encoding por frecuencia (Frequency Encoding)

¿Cómo mitigar el Orden Artificial?

StandardScaler

StandardScaler transforma una variable para que tenga:

- **media = 0**
- **desviación estándar = 1**

La fórmula es:

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

donde:

μ = media

σ = desviación estándar

Valores: [0.50, 0.3, 0.2]

Media = 0.333

Std \approx 0.118

Tras escalar:

0.50 \rightarrow +1.41

0.30 \rightarrow -0.87

0.20 \rightarrow -0.64

☞ ¿Qué ha pasado?

- Sigue existiendo orden
- Pero **ya no es “el doble”**
- El modelo ve “más frecuente / menos frecuente”, no magnitud absoluta

☞ ¿Cómo ayuda con el orden artificial?

StandardScaler:

- ✓ Reduce el impacto de la magnitud
- ✓ Hace comparables variables distintas
- ✓ Evita que una codificación domine a otras

☞ Cuándo usar StandardScaler

- ✓ Modelos lineales (regresión, logística)
- ✓ KNN / SVM
- ✓ Redes neuronales
- ✓ Frequency Encoding cuando **no quieres que pese demasiado**

7. Codificación de variables categóricas

7.3 Encoding por frecuencia (Frequency Encoding)

¿Cómo mitigar el Orden Artificial?

Transformación logarítmica

Los valores grandes se “encogen” y los pequeños se “estiran”, sin mantener la proporcionalidad lineal.

La fórmula es: $x_{log} = \log(1 + x)$

Valores de frecuencia:

rojo → 0.50

azul → 0.25

verde → 0.24

morado → 0.01

Tras log1p:

$\log1p(0.50) \approx 0.405$

$\log1p(0.25) \approx 0.223$

$\log1p(0.24) \approx 0.216$

$\log1p(0.01) \approx 0.010$

❖ ¿Qué ha cambiado?

- Las diferencias grandes se reducen
- Las categorías raras siguen siendo distinguibles
- El modelo deja de ver saltos “brutos”

☞ ¿Cómo ayuda con el orden artificial?

La transformación log:

- ✓ Reduce la influencia de categorías muy frecuentes
- ✓ Hace la relación **no lineal**
- ✓ Evita que el modelo interprete diferencias como proporcionales

☞ Cuándo usar transformación log

- ✓ Frequency Encoding con muchas categorías
- ✓ Cuando “raridad” importa más que “cantidad”
- ✓ Para **mitigar el orden artificial** introducido por Frequency Encoding

7. Codificación de variables categóricas

7.4 Target Encoding(Mean Encoding)

¿En qué consiste?

Target Encoding sustituye cada categoría por un valor numérico que representa la relación estadística de esa categoría con la variable objetivo (target).

Normalmente:

- en **clasificación** → media del target (probabilidad)
- en **regresión** → media del valor objetivo

Ejemplo conceptual sencillo. Supongamos:

canal	target
web	1
web	0
app	1
app	1
tienda	0
tienda	0

Media del target por categoría:

- web → $(1+0)/2 = 0.50$
- app → $(1+1)/2 = 1.00$
- tienda → $(0+0)/2 = 0.00$

Tras Target Encoding:

canal	canal_target_enc
web	0.50
app	1.00
tienda	0.00

7. Codificación de variables categóricas

7.4 Target Encoding(Mean Encoding)

¿Qué hace realmente?

- ✓ Reduce variables categóricas a **una sola columna**
- ✓ Captura información **supervisada** (usa el target)
- ✓ Muy potente con **alta cardinalidad**

⚠ También es **muy peligrosa** si se hace mal.

¿Cuándo usar Target Encoding?

Casos adecuados

- Variables categóricas con **muchas categorías**
- Cuando One-Hot no es viable
- Modelos potentes (GBM, XGBoost, CatBoost, RF)
- Dataset suficientemente grande
- Validación cruzada disponible

Ejemplos:

- ciudad
- producto
- usuario
- código postal

¿Cuándo NO usarla?

- ✗ Datasets pequeños
- ✗ Sin validación cruzada
- ✗ Para introducir rápido (baseline)
- ✗ Sin control de leakage

7. Codificación de variables categóricas

Frequency vs Target Encoding

	Target Encoding	Frequency Encoding
Usa el target	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> No
Qué informa	Relación categoría–objetivo	Popularidad / rareza
Riesgo de leakage	 Alto si se hace mal	 Medio
Introduce orden	<input checked="" type="checkbox"/> Sí (por target)	<input checked="" type="checkbox"/> Sí (por frecuencia)
Cuándo es útil	Cuando la categoría tiene señal predictiva	Cuando hay alta cardinalidad
Típico uso	Modelos lineales, boosting, tabular	Árboles, high-cardinality

7. Codificación de variables categóricas

Frequency vs Target Encoding



¿Cuál suele dar mejores resultados?



Target Encoding

👉 Suele dar mejores resultados predictivos que Frequency Encoding cuando:

- La variable categórica tiene **relación real con el target**
- Hay **muchas categorías** (alta cardinalidad)
- Se aplica **correctamente** (sin data leakage, con smoothing / CV)

¿Por qué?

👉 Porque mete directamente en el modelo **información del objetivo**.



Frequency Encoding

👉 Suele ser **más estable y seguro**, pero **menos potente**:

- No usa el target
- Solo dice si una categoría es **frecuente o rara**
- Es útil cuando:
 - No quieres riesgo de leakage
 - La frecuencia en sí misma es informativa
 - Como baseline rápido en alta cardinalidad

8. Escalado de Datos

8.1 Escalado

El **escalado de datos** consiste en **transformar las variables numéricas** para que estén en una **escala comparable**, sin cambiar su significado.

Hace que los números sean comparables entre sí y que tengan el mismo peso en el modelo.



Idea clave

El escalado no cambia la información, solo cambia la unidad de medida.

Ejemplo:

- Edad en años: 18–90
- Ingresos en euros: 1.000–100.000

Sin escalar:

- Ingresos dominan cualquier modelo

Con escalado:

- Ambas variables pesan de forma justa

8. Escalado de Datos

8.1 Escalado

Algoritmos sensibles al escalado

Muy sensibles

- KNN
- SVM
- Regresión lineal / logística
- Redes neuronales
- PCA
- LDA

Poco sensibles

- Árboles de decisión
- Random Forest
- XGBoost
- Naive Bayes

Qué NO hace el escalado

- ✗ No elimina outliers
- ✗ No cambia la forma de la distribución
- ✗ No hace los datos normales

Para qué sirve el escalado

- ✓ Evita que una variable domine por su rango
- ✓ Mejora el entrenamiento de muchos modelos
- ✓ Hace comparables las variables



Si el algoritmo usa distancias o gradientes, hay que escalar.

8. Escalado de Datos

8.1.1 Técnicas Escalado

- StandardScaler
- RobustScaler
- MinMaxScaler
- MaxAbsScaler

8. Escalado de Datos

8.1.1 Técnicas Escalado

8.1.1.1 StandardScaler

StandardScaler transforma las variables para que tengan media 0 y desviación estándar 1.

No cambia la forma de la distribución, solo la escala.

- ✓ Hace comparables las variables
- ✓ Muy usado en modelos lineales y basados en distancia

¿Qué hace exactamente?

Para cada valor x , aplica esta transformación:

donde:

- μ = media de la variable
- σ = desviación estándar

$$x_{escalado} = \frac{x - \mu}{\sigma}$$

8. Escalado de Datos

8.1.1 Técnicas Escalado

8.1.1.1 StandardScaler

Supongamos estas personas:

Persona	Edad (años)	Ingresos (€)
A	20	1.200
B	40	30.000
C	60	60.000
D	80	100.000

Con StandardScaler:

Persona	Edad (escalada)	Ingresos (escalados)
A	-1.34	-1.20
B	-0.45	-0.10
C	0.45	0.60
D	1.34	1.70

Observa:

- Edad → valores pequeños
- Ingresos → valores enormes

Un modelo “ve” ingresos como mucho más importantes **solo por magnitud**.

Ahora:

- Ambas variables están en la **misma escala**
- Media = 0
- Desviación = 1

8. Escalado de Datos

8.1.1 Técnicas Escalado

8.1.1.1 StandardScaler

Interpretación:

- **Valores negativos** → por debajo de la media
- **Valores positivos** → por encima de la media
- El tamaño indica cuánto se aleja del promedio

¿Para qué sirve StandardScaler?

Sirve para:

- Evitar que variables con valores grandes dominen el modelo
- Hacer comparables variables con distintas unidades
- Mejorar la convergencia de algoritmos



Cuándo usar StandardScaler

- Con modelos **sensibles a la escala**
- Cuando las variables tienen **unidades o rangos muy distintos**
- Cuando los datos son **más o menos normales**
Las distribuciones no están súper sesgadas

No hay outliers extremos sin tratar



Cuándo no es necesario (o no ideal)

- Con modelos basados en árboles
- Si hay **muchos outliers extremos**
Mejor RobustScaler



StandardScaler pone todas las variables en el mismo “idioma numérico”.

8. Escalado de Datos

8.1.1 Técnicas Escalado

8.1.1.2 RobustScaler

RobustScaler es una técnica de escalado que utiliza estadísticas robustas:

- **Mediana** (en lugar de la media)
- **Rango intercuartílico** (IQR) (en lugar de la desviación estándar)

RobustScaler escala los datos sin dejar que los outliers manden. Es la alternativa a **StandarScaler** cuando hay **Outliers**.

¿Qué hace exactamente?

Aplica esta transformación:

donde:

$$x_{scaled} = \frac{x - \text{mediana}}{IQR}$$

- **IQR**=Q3 - Q1
- **Q1** = percentil 25
- **Q3** = percentil 75

Interpretación:

- **Valores negativos** → por debajo de la mediana
- **Valores positivos** → por encima de la mediana
- El tamaño del valor indica cuántos rangos intercuartílicos (**IQR**) se aleja del valor típico

8. Escalado de Datos

8.1.1 Técnicas Escalado

8.1.1.3 MinMaxScaler

MinMaxScaler es una técnica de preprocessamiento que escala los valores de una variable para que estén dentro de un **rango fijo[0,1]**, manteniendo las proporciones entre los datos.

¿Qué hace MinMaxScaler?

- Transforma los valores usando el mínimo y el máximo de la variable
- El valor mínimo pasa a 0
- El valor máximo pasa a 1
- El resto de valores se distribuyen proporcionalmente entre 0 y 1
- No cambia la forma de la distribución, solo el rango

Para cada valor x:

Donde:

$$x_{\text{escalado}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- **x** → valor original
- **x_{min}** → valor mínimo de la variable
- **x_{max}** → valor máximo de la variable

Qué hace la fórmula:

- Si **x=x_{min}** → resultado = 0
- Si **x=x_{max}** → resultado = 1
- Si x está entre ambos →

8. Escalado de Datos

8.1.1 Técnicas Escalado

8.1.1.4 MaxAbsScaler

MaxAbsScaler es una técnica de escalado que divide cada valor de una variable por su valor máximo absoluto, haciendo que los datos queden en el rango [-1, 1].

¿Qué hace MaxAbsScaler?

- Calcula el valor máximo absoluto de cada variable
- Divide cada valor por ese máximo
- El valor con mayor magnitud pasa a 1 o -1
- El resto de valores quedan proporcionalmente escalados
- No cambia la forma de la distribución, solo el rango

Para cada valor x:

$$x_{\text{escalado}} = \frac{x}{\max(|x|)}$$

Interpretación:

- Si $x=\max(|x|) \rightarrow \text{resultado} = 1$
- Si $x=-\max(|x|) \rightarrow \text{resultado} = -1$
- Si $x=0 \rightarrow \text{resultado} = 0$
- Si x está entre esos valores $\rightarrow \text{resultado entre } -1 \text{ y } 1$

8. Escalado de Datos

8.1.1 Técnicas Escalado

Cuando usar **MinMaxScaler** o **MaxAbsScaler**

La clave es si al escalar se necesita mantener el 0 intacto y trabajar con valores positivos y negativos, en ese caso **MaxAbsScaler**, si debe estar en el rango [0,1], entonces **MinMaxScaler**

¿Para qué sirve el rango [-1, 1]?

El signo del valor (positivo / negativo) es importante y el 0 tiene un significado especial.

Es decir:

- 0 representa ausencia, neutralidad o equilibrio
- Valores positivos y negativos tienen significado distinto
- El modelo debe distinguir dirección, no solo magnitud

8. Escalado de Datos

8.1.1 Técnicas Escalado

Cuando usar **MinMaxScaler** o **MaxAbsScaler**

Algoritmos / casos donde el signo importa:

A) Redes neuronales (según activación):

1. Activación **tanh**

$$\tanh(x) \rightarrow [-1, 1]$$

Aquí:

- 0 es el centro
- Valores positivos y negativos son simétricos

 **MaxAbsScaler**

2. Activación **sigmoid**

$$\text{sigmoid}(x) \rightarrow [0, 1]$$

Aquí:

- No hay valores negativos
- El 0 no es centro, es límite

 **MinMaxScaler**

3. Activación **ReLU**

$$\text{ReLU}(x) = \max(0, x)$$

Aquí:

- Valores negativos se anulan
- El modelo funciona mejor con entradas positivas

 **MinMaxScaler**

8. Escalado de Datos

Ejemplo Práctico

Ejemplo MUY claro (el mismo problema, dos decisiones):

Tengo que predecir edad, ingresos, visitas, ...

Opción A: Regresión logística

- Usa gradientes
- Compara magnitudes

👉 **StandardScaler o RobustScaler**

Opción B: Redes neuronales

- Espera entradas en [0,1]

👉 **MinMaxScaler**



Los datos son los mismos, cambia la técnica porque cambia lo que necesita el modelo

9. Transformación y creación de características (Feature Engineering)

9.1 Qué es y por qué es importante

Feature Engineering es el proceso de **crear, transformar o combinar variables** para que el modelo pueda capturar mejor patrones reales.

¿Qué aporta?

- Mejora el rendimiento del modelo
- Facilita el aprendizaje de patrones relevantes
- Reduce el ruido y el sobreajuste
- Permite usar modelos más simples
- Aporta interpretabilidad
- Incorpora conocimiento del dominio

¿Por qué es necesaria?

- Los datos originales suelen ser incompletos o poco informativos
- Muchas relaciones importantes no están explícitas
- Los modelos clásicos dependen directamente de la calidad de las variables



Feature Engineering convierte datos en información útil para el modelo.

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.1 Creación de Nuevas Variables

9.2.1.1 Interacciones entre variables

Consiste en combinar dos o más variables para capturar relaciones implícitas.

Ejemplos:

- Precio total = precio × cantidad
- Productividad = producción / horas
- Riesgo = frecuencia × impacto

Ventajas:

- Captura relaciones no lineales
- Mejora modelos simples

Desventajas:

- Puede aumentar mucho el número de variables
- Riesgo de sobreajuste

Cuándo usar:

- Modelos lineales
- Variables relacionadas conceptualmente

Cuándo no usar:

- Datos muy ruidosos
- Pocos datos disponibles

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.1 Creación de Nuevas Variables

9.2.1.2 Ratios y proporciones

Expresan relaciones relativas entre magnitudes.

Ejemplos:

- Beneficio / coste
- Goles / partidos
- Consumo / habitantes

Ventajas:

- Más informativas que valores absolutos
- Alta interpretabilidad

Desventajas:

- Problemas con valores cero
- Sensibles a errores

Cuándo usar:

- Economía, empresa, análisis social

Cuándo no usar:

- Cuando el denominador es inestable

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.1 Creación de Nuevas Variables

9.2.1.3 Agregaciones

Creación de variables resumen a partir de múltiples registros.

Ejemplos:

- Media de compras por cliente
- Total de visitas
- Máximo histórico



Ventajas:

- Aporta contexto global
- Muy potente en Big Data



Desventajas:

- Puede perder información temporal
- Riesgo de *data leakage*



Cuándo usar:

- Datos transaccionales
- Datos relacionales



Cuándo no usar:

- Si rompe la secuencia temporal

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.1 Creación de Nuevas Variables

9.2.1.4 Variables basadas en conocimiento del dominio

Uso de fórmulas o reglas propias del problema.

Ejemplos:

- IMC = peso / altura²
- Consumo energético por m²
- Índice de riesgo financiero



Ventajas:

- Muy alta calidad informativa
- Reduce complejidad del modelo



Desventajas:

- Requiere conocimiento experto
- Difícil de generalizar



Cuándo usar:

- Dominios bien definidos



Cuándo no usar:

- Problemas poco conocidos

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.1 Creación de Nuevas Variables

9.2.1.5 Ventanas temporales (lags y rolling features)

Generación automática de contexto histórico.

Ejemplos:

- Valor del día anterior
- Media móvil 7 días
- Tendencia



Ventajas:

- Muy potentes en predicción temporal



Cuándo usar:

- Series temporales



Desventajas:

- Incrementan dimensionalidad
- Riesgo de fuga de información



Cuándo no usar:

- Datos independientes

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.2 Transformación de Variables

9.2.2.1 Transformaciones matemáticas

Modifican la distribución de una variable.

Ejemplos:

- Logarítmica
- Raíz cuadrada
- Inversa ($1/x$)

Ventajas:

- Reduce asimetría
- Mejora estabilidad del modelo

Desventajas:

- Menor interpretabilidad
- No siempre aplicables

Cuándo usar:

- Distribuciones sesgadas

Cuándo no usar:

- Variables ya bien distribuidas

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.2 Transformación de Variables

9.2.2.2 Discretización semántica

Convierte variables continuas en categorías con significado.

Ejemplos:

- Edad → joven / adulto / senior
- Nota → suspenso / aprobado / notable

Ventajas:

- Mayor interpretabilidad
- Reduce ruido

Desventajas:

- Pérdida de precisión
- Elección subjetiva de rangos

Cuándo usar:

- Cuando importa la interpretación
- Modelos basados en Reglas o Árboles

Cuándo no usar:

- Modelos que aprovechan continuidad
- En modelos basados en distancias o gradientes

9. Transformación y creación de características (Feature Engineering)

9.2 Principales Técnicas

9.2.2 Transformación de Variables

9.2.2.3 Transformaciones temporales

Extraen información de una fecha

Ejemplos:

- Día de la semana (lunes)
- Mes (Agosto)
- Es fin de semana (si/no)



Ventajas:

- Captura patrones temporales
- Esencial en series temporales



Desventajas:

- Puede introducir redundancia



Cuándo usar:

- Datos con componente temporal
- Modelos basados en Reglas o Árboles



Cuándo no usar:

- Datos atemporales
- En modelos basados en distancias o gradientes

10. Selección de características (Feature Selection)

10.1 ¿Qué es la selección de características?

Feature Selection es el proceso de elegir un subconjunto de variables relevantes y eliminar:

- variables irrelevantes
- variables redundantes
- variables ruidosas

A veces, menos variables producen modelos más simples, más rápidos y más robustos.

¿Por qué es necesaria?

La selección de características ayuda a:

- reducir sobreajuste
- mejorar la generalización
- reducir coste computacional
- mejorar la interpretabilidad
- evitar multicolinealidad



Feature Engineering crea variables; Feature Selection decide cuáles merecen quedarse.

10. Selección de características (Feature Selection)

10.2 Métodos de Selección de Variables (Feature Selection)

Los métodos de selección de variables se agrupan en tres grandes categorías, según cómo y cuándo se decide qué variables se quedan:

1. Métodos Filter

Seleccionan variables **antes** de entrenar el modelo.

Son **rápidos**, independientes del algoritmo.

2. Métodos Wrapper

Usan un modelo para evaluar subconjuntos de variables.

Más precisos, pero **costosos**.

3. Métodos Embedded

La selección ocurre **durante el entrenamiento**.

10. Selección de características (Feature Selection)

10.2 Métodos de Selección de Variables (Feature Selection)

10.2.1 Métodos Filter

Los **métodos filter** son técnicas de **selección de características** que evalúan cada variable usando **criterios estadísticos sobre los datos, sin entrenar ningún modelo de ML**.

Es decir:

- Miran la relación entre:
 - Feature \leftrightarrow target
 - O feature \leftrightarrow feature
- Y deciden:
 - Qué variables **conservar**
 - Cuáles **eliminar**

Antes de entrenar el modelo

 ¿Cuándo usar métodos Filter?

1. Cuando necesitas algo muy rápido y barato
2. Cuando quieres un método independiente del modelo

10. Selección de características (Feature Selection)

10.2 Métodos de Selección de Variables (Feature Selection)

10.2.1 Métodos Filter

10.2.1.1. Variance Threshold (Filtro por varianza)

Variance Threshold es un método Filter que elimina variables cuya varianza es muy baja, es decir, variables que casi no cambian entre las observaciones.

Evalúa cada variable de forma independiente

 Si una variable apenas cambia, no puede explicar diferencias en el target.

¿Qué mide exactamente?

Mide la varianza estadística de cada variable:

- Varianza alta → la variable toma valores distintos → puede aportar información
- Varianza baja → la variable es casi constante → poco informativa

10. Selección de características (Feature Selection)

10.2 Métodos de Selección de Variables (Feature Selection)

10.2.1 Métodos Filter

10.2.1.1. Variance Threshold (Filtro por varianza)

¿Cuándo aplicar Variance Threshold?



Cuándo usarlo

- Primer paso del preprocessamiento
- Datasets grandes
- Variables generadas automáticamente
- Limpieza básica de ruido
- Antes de cualquier modelo



Cuándo NO usarlo (o usarlo con cuidado)

- Variables binarias raras pero importantes (ej. fraude = 1 solo en el 1%)
- Variables categóricas codificadas (One-Hot)
- Cuando la ausencia de cambio es informativa

Variable	Valores	Varianza
edad	18–70	Alta
constante	siempre 1	0
casi_constante	98% ceros	Muy baja

10. Selección de características (Feature Selection)

10.2 Métodos de Selección de Variables (Feature Selection)

10.2.1 Métodos Filter

10.2.1.2. Correlación con el Target

Los **métodos de correlación** miden la relación estadística entre cada variable y la variable objetivo, sin entrenar ningún modelo. Evalúa cada variable de forma independiente



La correlación mide cuánto cambia el target cuando cambia una variable.

Los principales algoritmos son:

- Correlación **Pearson** (lineal)
- Correlación **Spearman** (monótona)
- Correlación **Point-biserial** (clasificación)

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.2.1 Correlación Pearson (lineal)

La correlación de **Pearson** mide la **relación lineal** entre una variable y otra (el target), es decir si al aumentar una variable la otra aumenta/disminuye proporcionalmente.

- Valores entre -1 y 1
- 0 → no relación lineal
- >0.6 → relación fuerte
- ±1 → relación lineal perfecta

X: 1 2 3 4
Y: 2 4 6 8

- Si X se duplica → Y se duplica
- Si X aumenta 1 → Y aumenta siempre lo mismo

El **signo** indica la dirección:

- + → cuando una sube, la otra suele subir
- → cuando una sube, la otra suele bajar

¿Cuándo usar Pearson?

- ✓ Variables numéricas continuas
- ✓ Se sospecha relación lineal
- ✓ Exploración inicial
- ✓ Modelos lineales

Limitaciones

- ✗ No detecta relaciones no lineales
- ✗ Sensible a outliers
- ✗ Asume relación lineal
- ✗ Si no hay relación lineal, puede engañar

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.2.2 Correlación Spearman

La correlación de **Pearson** mide la **relación** entre una variable y otra (el target), es decir si al aumentar una variable aumenta la otra aumenta/disminuye **pero no de un modo proporcional o lineal**.

Ejemplo:

X: 1 2 3 4
Y: 1 4 9 16

☞ Spearman dice: relación fuerte

☞ Pearson dice: relación débil (no lineal)

Entonces, ¿para qué sirve Spearman?

Usa Spearman cuando:

- Pearson te da bajo pero **visualmente ves relación**
- La variable es:
 - sesgada (ingresos, ventas)
 - no lineal
- Hay outliers

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.2.2 Correlación Spearman

Pearson se pregunta:

¿Esto se parece a una línea recta?

Spearman se pregunta:

¿Cuando uno sube, el otro suele subir?



**Pearson detecta líneas rectas.
Spearman detecta tendencias.**

El **signo** indica la dirección:

- + → cuando una sube, la otra suele subir
- → cuando una sube, la otra suele bajar

Casos y decisiones

Situación real	Pearson	Spearman	Qué hacer
Relación lineal	Alto	Alto	Mantener
Relación no lineal	Bajo	Alto	Mantener
Sin relación	Bajo	Bajo	Eliminar
Relación en U	Bajo	Bajo	Revisar

- Valores entre -1 y 1
- 0 → no relación lineal
- 0.1-0.3 → relación débil
- 0.3-0.6 → relación moderada
- >0.6 → relación fuerte

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.2.2 Correlación Spearman



Cuándo ES recomendable

- Cuando **Pearson da bajo**, pero **sospechas que sí hay relación**
- Cuando la relación **no es lineal**, pero sí **ordenada**
- Cuando las variables están **sesgadas** (ingresos, ventas, tiempos)
- Cuando hay **outliers**
- Cuando solo importa el **orden** (más/menos), no la magnitud exacta
- En **exploración inicial** como complemento a Pearson



Limitaciones importantes

- No detecta relaciones **no monótonas** (en U, ∩, etc.)
- No mide proporcionalidad (no sirve para modelos lineales)
- Pierde información de magnitud (solo usa orden)
- No sustituye a Pearson en análisis lineales
- Puede dar valores altos aunque la relación no sea útil para un modelo lineal

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.3. Dependencia Categórica

10.2.1.3.1 Chi-cuadrado (χ^2)

El test de **Chi-cuadrado** mide si existe **dependencia estadística entre dos variables categóricas**, es decir, si la distribución de una variable cambia en función de las categorías de la otra.

Ejemplo:

Target binario (válido)

canal	target
web	0
app	1
tienda	0

Target multiclase (TAMBIÉN válido)

canal	target
web	bajo
app	medio
tienda	alto

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.3.1 Chi-cuadrado (χ^2)

¿Qué significa el valor de χ^2 ?

χ^2 mide cuánto se aleja la distribución observada de lo que esperaríamos si no hubiera relación.

- **χ^2 pequeño** → lo observado es muy parecido a “azar”
- **χ^2 grande** → lo observado es muy distinto a “azar”

¿Qué es “ χ^2 alto” o “ χ^2 bajo”?

χ^2	Interpretación aproximada
≈ 0	No relación
1 – 5	Relación débil
5 – 20	Relación moderada
> 20	Relación fuerte

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.3.1 Chi-cuadrado (χ^2)

Interpretación de χ^2 (SOLO NO basta)

⚠ El valor de χ^2 por sí solo NO se interpreta aisladamente

porque depende de:

- tamaño del dataset
- número de categorías
- número de clases del target

☞ Por eso **SIEMPRE** se interpreta junto al **p-value**.

Si no hay un χ^2 alto + p muy bajo → Revisar

Interpretación conjunta: $\chi^2 + p\text{-value}$

canal_app	$\chi^2 = 35.4$	p = 0.00001
canal_web	$\chi^2 = 12.8$	p = 0.00035
canal_tienda	$\chi^2 = 1.1$	p = 0.29

canal_app

- χ^2 alto + p muy bajo
- ✓ Dependencia clara → mantener

canal_web

- χ^2 medio + p bajo
- ✓ Dependencia significativa → mantener

canal_tienda

- χ^2 bajo + p alto
- ✗ Puede ser casualidad → eliminar

10. Selección de características (Feature Selection)

10.2.1 Métodos Filter

10.2.1.3.1 Chi-cuadrado (χ^2)

Casos en los que SÍ usarlo

- La **variable explicativa es categórica** (nominal)
- El **target es categórico**
- Estás resolviendo un problema de **clasificación**
- Quieres evaluar **dependencia entre categorías**
- Estás usando un método **Filter** (sin entrenar modelo)

Casos en los que NO usarlo

- Variables numéricas continuas
- Problemas de **regresión**
- Variables **ordinales** donde el orden importa
- Categorías con muy pocos ejemplos

10. Selección de características (Feature Selection)

10.2.2 Métodos Wrapper

Los métodos wrapper son técnicas de selección de características que prueban distintos subconjuntos de variables entrenando un modelo y evaluando su rendimiento, y se quedan con el conjunto que mejor funciona.

📌 ¿Qué eliges en un Wrapper?

Tú defines:

- **Modelo:** (Regresión logística, SVM, Random Forest, etc.)
- **Métrica:** (accuracy, F1, AUC, RMSE, MAE, etc.)
- **Estrategia de búsqueda:** Forward, Backward, Stepwise, RFE...
- **Esquema de validación:** hold-out o cross-validation

El wrapper **no sabe** qué es “mejor” en abstracto.

Para él, **mejor = lo que maximiza (o minimiza) la métrica que tú le das.**



El wrapper **no busca “las variables más importantes”**, busca:

“Las variables que hacen que **este modelo obtenga mejor esta métrica**”

10. Selección de características (Feature Selection)

10.2.2 Métodos Wrapper

10.2.2.1. Forward Selection (Selección hacia delante)

¿En qué consiste?

- Empiezas con 0 variables
- Añades la variable que más mejora el modelo
- Luego pruebas añadir cada una de las restantes
- Te quedas con la que más mejora
- Repites hasta que:
 - No mejora
 - O alcanzas un número máximo de variables



Cuándo usarlo

- Cuando tienes **muchas variables** y quieres empezar simple
- Cuando sospechas que **solo unas pocas variables son**
- Cuando quieres un modelo **simple e interpretable**
- Cuando no puedes entrenar con **todas las variables a la vez**

10. Selección de características (Feature Selection)

10.2.2 Métodos Wrapper

10.2.2.2. Backward Elimination (Eliminación hacia atrás)

¿En qué consiste?

- Empieza con todas las variables
- Elimina en cada paso la variable que menos aporta
- Se detiene cuando eliminar empeora el modelo
- Compara métricas.
Pej si es regresión puedo elegir R2 ajustado y si es de clasificación, Accuracy.



Cuándo usarlo

- Número moderado de variables
- Dataset bien preparado
- Cuando quieres depurar un conjunto ya grande

10. Selección de características (Feature Selection)

10.2.2 Métodos Wrapper

10.2.2.3. Recursive Feature Elimination (RFE)

¿Cómo funciona RFE? (paso a paso)

El funcionamiento de RFE es **siempre el mismo**, independientemente del modelo:

▣ Proceso iterativo

1. Entrena un modelo con todas las variables
2. Calcula la importancia de cada variable según el modelo
 1. coeficientes (regresión, SVM lineal)
 2. importancias (árboles)
3. Elimina la variable menos importante
4. Vuelve a entrenar el modelo con las variables restantes
5. Repite el proceso hasta alcanzar:
 1. un número deseado de variables
 2. o una condición de parada

Coeficiente:

- Es un parámetro interno del modelo que mide cuánto influye una variable en la predicción.
- Lo utiliza para ordenar las variables
- Cada modelo lo calcula diferente



RFE no es un optimizador de variables, sólo elige la N mejores con las que quedarse.

10. Selección de características (Feature Selection)

10.2.2 Métodos Wrapper

10.2.2.4. RFECV (Recursive Feature Elimination with Cross-Validation)

RFECV es un método Wrapper de selección de variables que **combina la eliminación recursiva de características (RFE) con validación cruzada para determinar automáticamente el número óptimo de variables** que maximiza el rendimiento del modelo

¿Cómo funciona RFECV? (paso a paso)

1. Comienza con **todas las variables**
2. Entrena el modelo
3. Calcula la **importancia de cada variable**
4. Elimina la **menos importante**
5. Evalúa el rendimiento del modelo con **validación cruzada**
6. Repite el proceso
7. **Selecciona el número de variables que obtiene la mejor métrica media**



A diferencia de RFE, **el criterio de parada NO es fijo**, es el **máximo rendimiento en validación cruzada**.

10. Selección de características (Feature Selection)

10.2.2 Métodos Wrapper

10.2.2.4. RFECV (Recursive Feature Elimination with Cross-Validation)

¿Qué características tiene RFECV?

✓ Características principales

- Usa un **modelo real**
- Elimina variables de forma iterativa
- Evalúa rendimiento en cada paso
- Determina automáticamente el número óptimo de variables
- Tiene en cuenta interacciones entre variables

✗ Limitaciones

- Coste computacional **muy alto**
- No escalable a muchos atributos
- Dependiente del modelo elegido
- Sensible a la métrica y al esquema de validación

¿Qué criterio utiliza RFECV?

RFECV utiliza como criterio:

- Una **métrica de rendimiento**, por ejemplo:
 - accuracy
 - F1-score
 - AUC
 - RMSE (regresión)
- Calculada mediante **validación cruzada**

10. Selección de características (Feature Selection)

10.2.3 Métodos Embedded

Los **métodos embedded** (o “embebidos”) son técnicas de **selección de características** que **realizan la selección de variables durante el propio entrenamiento del modelo**, como parte del algoritmo.

Es decir:

- No es un paso separado (como filtros o wrappers)
- El **modelo ya incluye un mecanismo interno** para:
 - Penalizar
 - Ignorar
 - O eliminar variables poco útiles

10. Selección de características (Feature Selection)

10.2.3 Métodos Embedded

💡 ¿Cuándo usar Métodos Embedded?

1. Cuando quieres algo más eficiente que Wrappers

- Embedded: Entrenan **una vez** (o pocas)
- Wrappers: Entrenan **muchas veces**

Si tienes:

- Muchas variables
- Modelos costosos
→ Embedded es mejor opción

2. Cuando usas modelos que ya incluyen selección interna

Ej:

- Lasso / Elastic Net
- Árboles, Random Forest, XGBoost, etc.

Tiene sentido:

Aprovechar ese mecanismo en vez de añadir un wrapper externo.

3. Cuando quieres un buen compromiso entre:

- Calidad del subconjunto
- Coste computacional



Regla práctica

- Muchas variables, quieres algo rápido y razonable
→ **Embedded**
- Pocas variables, quieres el mejor subconjunto posible
→ **Wrapper**
- Exploración rápida o primer filtrado
→ **Filtro**

10. Selección de características (Feature Selection)

10.2.3 Embedded

10.2.3.1. Regularización L1 (Lasso)

La regularización L1 (Lasso) es una técnica que **añade una penalización al modelo para forzar que algunos coeficientes se reduzcan exactamente a cero, eliminando automáticamente las variables asociadas.**



Lasso hace que el modelo “apague” variables que no considera importantes.

¿Cómo funciona L1 ?

- Coeficientes pequeños → se empujan a **0**
- Algunos coeficientes → quedan **exactamente en 0**
- Variables con coeficiente 0 → **eliminadas**

Ejemplo:

Modelo sin regularización:	Con Lasso:
edad → 0.6	edad → 0.5
ingresos → 1.8	ingresos → 1.6
gasto → 0.4	gasto → 0
visitas → 0.1	visitas → 0

10. Selección de características (Feature Selection)

10.2.3 Embedded

10.2.3.1. Regularización L1 (Lasso)

Lasso es una forma de entrenar un modelo lineal (Regresión lineal o logística) añadiendo una penalización L1 a los coeficientes.

El modelo sigue siendo: $y = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$



¿Qué hace Lasso en la práctica?

1. Ajusta el modelo (como una regresión normal)

- Aprende los coeficientes w_1, w_2, \dots, w_n

2. A la vez, empuja muchos coeficientes a 0.

- La penalización L1 “castiga” tener muchos coeficientes distintos de 0

Resultado:

- Algunas variables quedan con: $w_i = 0$
- Esas variables No influyen en el modelo
→ quedan **eliminadas**

10. Selección de características (Feature Selection)

10.2.3 Embedded

10.2.3.1. Regularización L1 (Lasso)

 ¿Cuándo usar Lasso (L1)?

 Usar L1 cuando:

- Usas **modelos lineales**
- Tienes **muchas variables**
- Hay **variables irrelevantes o redundantes**
- Quieres **interpretabilidad**
- Buscas selección automática de variables

Ejemplos típicos:

- selección de features en regresión
- clasificación con muchas variables

 **No usar L1 cuando:**

- Las relaciones no son muy lineales
- Las variables están muy correlacionadas (L1 puede quedarse solo con una)
- No has escalado las variables
- Usas modelos que no tienen coeficientes

 **Requisitos importantes**

- **Escalado obligatorio** (StandardScaler)
- Interpretar coeficientes con cuidado
- Sensible a colinealidad

10. Selección de características (Feature Selection)

10.2.3 Embedded

10.2.3.2. Árboles de Decisión

Los métodos Embedded con árboles de decisión seleccionan variables midiendo cuánta mejora aportan al modelo al reducir la impureza del target en cada división.



Una variable es importante si ayuda a separar mejor las clases (o valores) del target.

¿Cómo “decide” un árbol qué variables son relevantes?

Durante el entrenamiento, el árbol:

1. Evalúa posibles divisiones con **todas las variables**
2. Elige la variable que **más reduce la impureza**
3. Repite el proceso en cada nodo

Variables que:

- aparecen muchas veces
- aparecen en nodos altos
- reducen mucho la impureza

➔ **son más importantes.**

10. Selección de características (Feature Selection)

10.2.3 Embedded

10.2.3.2. Árboles de Decisión

Modelos que implementan este Embedded:

- DecisionTreeClassifier
- RandomForestClassifier
- GradientBoostingClassifier
- XGBoost, LightGBM

¿Cuándo usar Embedded con árboles?



Usar cuando:

- Hay relaciones no lineales
- Hay interacciones entre variables
- No quieres escalar
- Quieres selección automática
- Dataset mediano/grande



Evitar cuando:

- Necesitas interpretabilidad lineal
- Variables con muchos valores (sesgo)
- Dataset muy pequeño

Elige la variable que más reduce el error

El árbol usa **ingresos** para hacer el split.

Variable	Reducción de impureza
edad	0.05
ingresos	0.40 <input checked="" type="checkbox"/>
visitas	0.02

10. Selección de características (Feature Selection)

10.3 Importancia de Variables

La **importancia de variables** es una medida que indica cuánto contribuye cada variable al predictivo del modelo.

Cuando es utilizada con fines de selección, consiste en:

👉 **Identificar qué variables aportan poco al rendimiento del modelo y evaluar si pueden eliminarse sin deteriorar la métrica objetivo**

Es decir:

- ¿Qué variables influyen más?
- ¿Cuáles aportan poco?
- ¿Cuáles podrían eliminarse sin empeorar el modelo?



La **importancia de variables** puede utilizarse como **criterio de selección** cuando se emplea como herramienta de diagnóstico **posterior al entrenamiento**, es decir una vez el modelo ya está entrenado, y siempre que su uso esté **respaldado por pruebas experimentales** que validen que la eliminación de variables no perjudica la métrica prioritaria del modelo.

10. Selección de características (Feature Selection)

10.3 Importancia de Variables

Principales Técnicas de Importancia de Variables

1. Técnicas Basadas en el Modelo (Model-Based Importance)

- Basadas en coeficientes de modelos lineales
- Basadas en reducción de impureza (árboles)

2. Técnicas Basadas en Rendimiento (Model-Agnostic)

- Permutation Importance

3. Técnicas de explicación (Explainability Methods)

- SHAP

10. Selección de características (Feature Selection)

10.3.1 Técnicas Basadas en el Modelo (Model-Based Importance)

10.3.1.1 Técnicas basadas en coeficientes de modelos lineales

Son métodos de **importancia de variables** que utilizan directamente los **coeficientes del modelo lineales** para medir el peso de cada variable.

Se aplican cuando el modelo tiene la forma:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

🧠 ¿Cómo se interpreta?

Supongamos que en un modelo tenemos:

Variable	Coeficiente
OverTime	+1.20
MonthlyIncome	-0.75
YearsAtCompany	-0.40



Modelos donde se usan:

- Logistic Regression
- Linear Regression
- SVM lineal

Interpretación:

- **OverTime** aumenta la probabilidad de abandono.
- **Mayor salario** reduce la probabilidad.
- **Más años** en la empresa reducen el riesgo.

Cuanto mayor es la magnitud absoluta del coeficiente mayor influencia.

10. Selección de características (Feature Selection)

10.3.1 Técnicas Basadas en el Modelo (Model-Based Importance)

10.3.1.1 Técnicas basadas en coeficientes de modelos lineales



¿En qué casos utilizar?

- Cuando el modelo es lineal
- Cuando buscas alta interpretabilidad
- Cuando quieres selección sencilla
Variables con coeficientes cercanos a cero
pueden eliminarse (previa validación).



Resumen final

- Basadas en coeficientes.
- Solo para modelos lineales.
- Requieren escalado.
- Muy interpretables.
- Sensibles a colinealidad.



Cuándo NO utilizarlas

- Modelos no lineales
- Variables correlacionadas pueden:
 - Repartirse efecto.
 - Cambiar coeficientes de forma inestable.
- Si no se han escalado variables



Limitaciones

- Solo capturan relaciones lineales
- No capturan interacciones complejas
- Sensibles a colinealidad
- Requieren escalado

10. Selección de características (Feature Selection)

10.3.1 Técnicas Basadas en el Modelo (Model-Based Importance)

10.3.1.2 Técnicas basadas en Reducción de impureza(Árboles)

Son métodos que usan **árboles de decisión** para estimar la importancia de las variables midiendo:

Cuánto ayuda cada variable a **hacer los nodos más “puros”** (o reducir la impureza), es decir, a que los datos dentro de cada nodo sean más homogéneos.

Una variable será más importante si:

- Se usa muchas veces en los splits
- Y esos splits **mejoran mucho** la calidad de las particiones

 Una variable será más importante si:

- Se usa muchas veces en los splits
- Y esos splits **mejoran mucho** la calidad de las particiones

10. Selección de características (Feature Selection)

10.3.1 Técnicas Basadas en el Modelo (Model-Based Importance)

10.3.1.2 Técnicas basadas en Reducción de impureza(Árboles)

¿Qué es la impureza?

La **impureza** mide:

↳ Qué tan **mezclados** están los datos dentro de un nodo.

- **Nodo puro:**
 - Casi todos los ejemplos son de la misma clase (o valores similares en regresión)
- **Nodo impuro:**
 - Hay mucha mezcla de clases (o mucha variación de valores)



El objetivo de un árbol es:

Hacer splits que **reduzcan la impureza** lo máximo posible.

10. Selección de características (Feature Selection)

10.3.1 Técnicas Basadas en el Modelo (Model-Based Importance)

10.3.1.2 Técnicas basadas en Reducción de impureza(Árboles)

Ejemplo de salida:

	feature	importance
0	f1	0.42
1	f2	0.31
2	f3	0.18
3	f4	0.06
4	f5	0.03



¿Cómo interpretar estos valores?

Los valores:

- Son **importancias relativas**
- Están **normalizadas** (suman 1)

Interpretación:

- f1 = 0.42 → es la **más usada** y la que más reduce la impureza en el bosque
- f2 = 0.31 → también muy importante, pero menos que f1
- f5 = 0.03 → apenas contribuye al modelo



OJO:

- **No significa** que f1 explique el 42% del problema
- Significa:

→ Que el 42% de la **reducción total de impureza** del modelo se atribuye a splits con f1

10. Selección de características (Feature Selection)

10.3.2 Técnicas Basadas en Rendimiento (Model-Agnostic)

¿Qué son?

Son métodos de **importancia de variables** que miden **cuánto empeora el rendimiento del modelo** cuando se altera o elimina una variable.

Se llaman **model-agnostic** porque:

- Son independientes del modelo. Funcionan con **cualquier modelo**
- No dependen de coeficientes ni de la estructura interna del modelo

 La idea es muy intuitiva:

Si estropeo una variable y el modelo rinde mucho peor → esa variable es importante.

10. Selección de características (Feature Selection)

10.3.2 Técnicas Basadas en Rendimiento (Model-Agnostic)

10.3.2.1 Permutation Importance

Consiste en **desordenar** (mezclar los valores) de una variable y posteriormente volver a entrenar el modelo y **ver cuánto empeora** el rendimiento del modelo .



Idea intuitiva:

- Si al **romper** una variable (mezclar sus valores) el modelo rinde **mucho peor**
→ esa variable es **importante**
- Si el rendimiento **casi no cambia**
→ esa variable es **poco o nada importante**

No mira el modelo por dentro:
solo mira **qué pasa con la métrica**.

Height at age 20 (cm)	Height at age 10 (cm)	...	Socks owned at age 10
182	155	...	20
175	147	...	10
...
156	142	...	8
153	130	...	24

10. Selección de características (Feature Selection)

10.3.2 Técnicas Basadas en Rendimiento (Model-Agnostic)

10.3.2.1 Permutation Importance

Supón que tu modelo tiene:

- Accuracy original = **0.90**

Tras permutar cada variable:

Regla mental

- Gran caída de métrica → variable clave
- Pequeña caída → poco importante
- Sin caída → irrelevante

Variable	Accuracy tras permutar	Caída	Interpretación
X1	0.70	0.20	Muy importante
X2	0.88	0.02	Poco importante
X3	0.90	0.00	Irrelevante
X4	0.80	0.10	Importancia media

10. Selección de características (Feature Selection)

10.3.2 Técnicas Basadas en Rendimiento (Model-Agnostic)

10.3.2.1 Permutation Importance

Cuándo usarla?

- Con **cualquier** modelo
- Cuando quieras:
 - Medir **impacto real en el rendimiento**
 - Interpretar modelos *black-box*
 - Validar importancias de otros métodos (árboles, coeficientes)

Ventajas

- Independiente del modelo
- Interpretable (impacto real en la métrica)
- Comparable entre variables
- Muy intuitiva

Desventajas / Limitaciones

- Computacionalmente cara
- Problemas con variables correlacionadas
- Depende de la métrica elegida y del conjunto de validación
- No implica causalidad

Cuándo usarla con cuidado

- Si hay **muchas variables correlacionadas**
- Si el dataset es **muy grande** (es costosa)
- Si el conjunto de validación es **pequeño o ruidoso**

10. Selección de características (Feature Selection)

10.3.3 Técnicas de explicación (Explainability Methods)

¿Qué son?

Son métodos que buscan responder a preguntas como:

- ¿Por qué el modelo ha hecho esta predicción?
- ¿Qué variables han influido más y en qué sentido?

 Es decir, no solo miden **rendimiento**, sino que explican:

- Cómo toma decisiones el modelo
- Qué variables influyen
- Cuánto y en qué dirección
- A nivel:
 - Global (modelo en general)
 - Local (una predicción concreta)

10. Selección de características (Feature Selection)

10.3.3 Técnicas de explicación (Explainability Methods)

10.3.3.1 SHAP (SHapley Additive exPlanations)

Explica una predicción repartiendo el resultado del modelo entre las variables, indicando cuánto aporta cada una (en positivo o en negativo).

🧠 Se basa en:

- **Teoría de juegos** (valores de Shapley)
- Cada variable es un “jugador” que contribuye al resultado final

📌 Puede dar:

- Explicaciones **globales** (qué variables influyen más en el modelo)
- Explicaciones **locales** (por qué un caso concreto tuvo esa predicción)

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

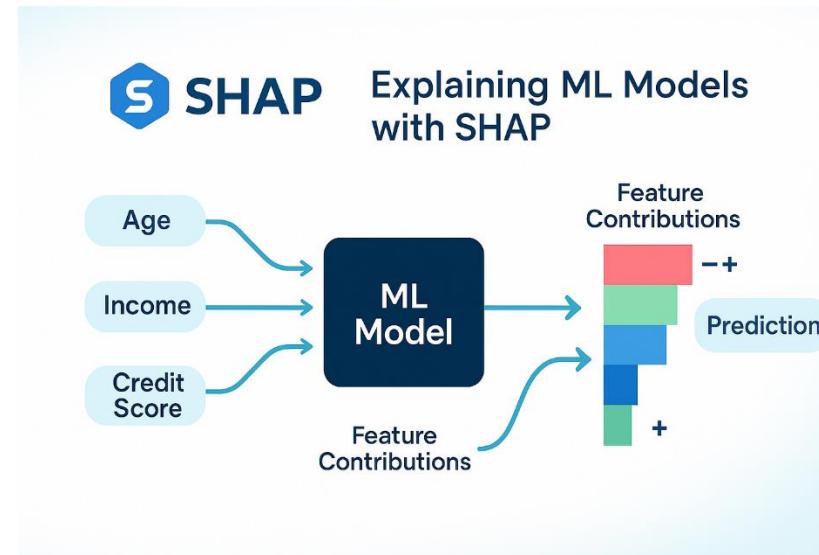
🎮 ¿Qué es la teoría de juegos?

La **teoría de juegos** es una rama de las matemáticas que estudia:

Cómo se reparte un resultado cuando **varios “jugadores” contribuyen** a él.

No tiene que ver solo con juegos de mesa, se usa en:

- Economía, Política, Negociación, Redes
- Y en ML para **repartir responsabilidad / contribución**



10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)



¿Qué es la teoría de juegos?

Ejemplo simple (repartir una recompensa):

Imagina:

- 3 personas hacen un trabajo en equipo
- Ganan **100€**
- La pregunta es:

Supón:

- A trabaja mucho
- B trabaja algo
- C casi no aporta



¿Cómo repartimos esos 100€ de forma justa según lo que aportó cada uno?

La teoría de juegos propone una solución “justa”:

- Mirar **qué aporta cada uno en todas las combinaciones posibles**
- Y asignar a cada uno su **contribución media** al resultado final

Ese valor se llama:

👉 **Valor de Shapley**

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

🎮 ¿Qué es la teoría de juegos?

Ejemplo simple (repartir una recompensa):

Equipo	Resultado	¿Cómo repartimos los 100€ de forma justa según lo que aporta cada uno?
Solo A	60 €	La idea de Shapley es:
Solo B	20 €	<ul style="list-style-type: none">• Mirar cuánto añade cada persona cuando se une a un grupo
Solo C	10 €	<ul style="list-style-type: none">• Hacer la media de sus aportes en todas las combinaciones
A + B	90 €	Resultado final :
A + C	60 €	<ul style="list-style-type: none">• A aporta ≈ 60 €
B + C	20 €	<ul style="list-style-type: none">• B aporta ≈ 30 €
A + B + C	100 €	<ul style="list-style-type: none">• C aporta ≈ 10 € <p>↳ Suman: $60 + 30 + 10 = 100$ €</p> <p>Eso es:</p> <p>Repartir el resultado total según la contribución real de cada uno.</p>

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)



Ejemplo de explicación local con conclusiones:

Supón un modelo que predice la **probabilidad de que un cliente compre**.

Predicción base del modelo → Probabilidad media (base) = 0.40

👤 Cliente concreto

Datos del cliente:

- ingresos = **45.000 €**
- visitas_web = **12**
- edad = **22**
- ruido = irrelevante

SHAP para este cliente:

Variable	Valor	Aporte SHAP
ingresos	45.000	+0.30
visitas_web	12	+0.10
edad	22	-0.05
ruido	—	+0.00



Cálculo

$$\Leftrightarrow 0.40 \text{ (base)} + 0.30 \text{ (ingresos)} + 0.10 \text{ (visitas)} - 0.05 \text{ (edad)} = 0.75$$

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)



Ejemplo de explicación local con conclusiones:



Interpretación (qué ha pasado)

☞ **Ingresos = 45.000 €:**

- Es **alto respecto a la media**
- El modelo ha aprendido que:

“Ingresos por encima de ~40.000 € aumentan mucho la probabilidad de compra”

- Por eso aporta **+0.30** → es el factor más decisivo

☞ **Visitas_web = 12:**

- Es un valor **alto**
- El modelo interpreta:

“Muchos intentos / interés → más probabilidad de comprar”

- Aporta **+0.10** → refuerza la decisión

☞ **Edad = 22:**

- Para este modelo:
- “Clientes muy jóvenes compran un poco menos”
- Por eso resta **-0.05** → frena un poco la predicción

☞ **Ruido:**

- No tiene efecto → el modelo prácticamente no lo usa

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

 Ejemplo de explicación local con conclusiones:

 Conclusiones prácticas

Para **este cliente concreto**, podemos decir:

- ✓ La predicción alta (0.75) se debe **principalmente a los ingresos altos**
- ✓ Las visitas también ayudan, pero menos
- ⚠ La edad joven juega un poco en contra
- ✗ Hay variables que el modelo ignora

👉 Y además podemos sacar reglas tipo:

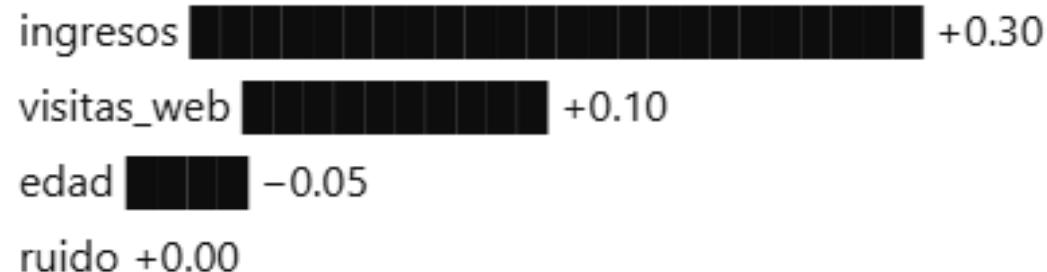
- “Ingresos por encima de ~40.000 € empujan fuertemente la predicción hacia compra”
- “Muchas visitas refuerzan la decisión”
- “Ser muy joven reduce ligeramente la probabilidad según este modelo”

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

Ejemplo de explicación global con conclusiones

Tras calcular SHAP en todo el dataset, obtienes este resumen de impacto medio de cada variable:



¿Cómo interpretarlo?

- **ingresos** → es la variable **más influyente**
- **visitas_web** → es importante, pero menos
- **edad** → tiene un efecto pequeño
- **ruido** → prácticamente **no aporta nada**



10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

 Ejemplo de explicación global con conclusiones

 ¿Cómo se usa esto para selección de variables?

1. Detectar variables inútiles

Si una variable tiene impacto SHAP ≈ 0 :

- Como **ruido** en nuestro ejemplo:
 - No cambia las predicciones
 - El modelo no la usa
 -  **Candidata clara a eliminar**

Resultado:

- Simplificas el modelo
- Menos ruido
- Más interpretabilidad
- A veces incluso mejor generalización

2. Detectar variables secundarias

Variables con impacto pequeño, como:

- **edad** (-0.05 en nuestro ejemplo)

Puedes preguntarte:

- ¿Aporta lo suficiente para justificar su complejidad?
- ¿Mejora algo las métricas si la quito?
- ¿Está correlacionada con otra?

Opciones:

- Probar a **quitarla y reevaluar**
- O dejarla si es barata de mantener y estable

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

 Ejemplo de explicación global con conclusiones

 ¿Cómo se usa esto para selección de variables?

3. Identificar variables clave

Variables con impacto alto:

- **ingresos** (+0.30)
- **visitas_web** (+0.10)

- Para selección de características:
-  No deberías quitarlas
-  Son las que hay que:
 - Cuidar
 - Medir bien
 - Limpiar bien
 - Monitorizar en producción

 **Conclusión:**

Estas variables son **imprescindibles** para el modelo.



Las **explicaciones globales de SHAP** permiten identificar qué variables son realmente usadas por el modelo, lo que las convierte en una herramienta muy útil para la selección de características.

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

Conclusiones Finales:

- ✗ No es buena idea usar **SOLO SHAP** para selección de variables.
- ✓ Lo correcto es **combinarlo con otros métodos** (Métodos Filter, Embedded o Wrapper).

⚠ Limitaciones de usar solo SHAP

✗ Puede:

- Repartir importancia entre variables correlacionadas
- Ocultar variables útiles que otro modelo sí usaría

✗ Explica:

- El modelo actual
- No necesariamente la “mejor” selección posible de variables

✗ No optimiza directamente:

- El conjunto de variables
- Solo **interpreta** un modelo dado

10. Selección de características (Feature Selection)

10.3.3.1 SHAP (SHapley Additive exPlanations)

➤ Flujo de trabajo recomendado:

1. Preprocesamiento:

- Quitar constantes, casi constantes, duplicadas

2. Selección de Variables:

- Correlaciones, varianza.

3. Entrenar modelo

4. Usar SHAP:

- Ver qué usa realmente el modelo
- Quitar variables inútiles

5. Volver a aplicar Selección de Variables (Opcional):

- RFE / Forward / Lasso para afinar

6. Reentrenar y validar métricas

11. Reducción de la Dimensionalidad

11.1 ¿Qué es la Reducción de la Dimensionalidad?

La **reducción de dimensionalidad** es el conjunto de técnicas cuyo **objetivo es reducir el número de variables** de un dataset **en un conjunto más pequeño**, conservando la mayor cantidad posible de información relevante.



La reducción de dimensionalidad comprime la información de muchas variables en pocas, facilitando el aprendizaje del modelo.

¿Por qué se usa?

- Reducir complejidad del modelo
- Evitar la maldición de la dimensionalidad
- Eliminar redundancia (variables correlacionadas)
- Acelerar el entrenamiento
- Facilitar visualización (2D / 3D)



Esto es clave:

- **Selección de variables** → quita columnas
- **Reducción de dimensionalidad** → crea nuevas columnas

11. Reducción de la Dimensionalidad

11.2 Principales Técnicas de Reducción de la Dimensionalidad?

1. Técnicas Feature Engineering

Reducción de dimensionalidad manualmente (Feature Engineering a partir de variables correlacionadas)

2. Técnicas Lineales

Transforman los datos originales mediante combinaciones lineales de las variables, creando nuevas dimensiones que capturan la mayor información posible.

- PCA
- LDA

3. Técnicas No Lineales

Buscan representar los datos en menos dimensiones preservando relaciones complejas y no lineales entre las observaciones.

- Kernel PCA
- t-SNE
- UMAP

4. Métodos basados en Modelos

Utilizan modelos de Machine Learning para aprender automáticamente una representación comprimida de los datos.

- Autoencoders

11. Reducción de la Dimensionalidad

11.2.1 Técnicas Feature Engineering

Consiste en aplicar Técnicas de Feature Engineering basadas en combinación de variables correlacionadas.

La idea es combinar en una sola variable, varias variables correlacionadas y así reducir el número de variables.

Ejemplo : Señales financieras → Índice de riesgo

Variables correlacionadas:

- volatilidad_7d
- volatilidad_30d
- volatilidad_90d

Nueva variable:

$$\text{indice_volatilidad} = (\text{volatilidad_7d} + \text{volatilidad_30d} + \text{volatilidad_90d}) / 3$$

11. Reducción de la Dimensionalidad

11.2.2 Técnicas Lineales

11.2.2.1 PCA — Principal Component Analysis

PCA es una técnica de reducción de dimensionalidad que **transforma las variables originales en un nuevo conjunto de variables llamadas componentes principales**, ordenadas según la cantidad de información (varianza) que explican.

¿Qué es la varianza?

La **varianza** mide **cuánto se dispersan los datos respecto a su valor medio**.

Dicho de forma simple:

☞ **La varianza indica cuánto cambian los datos.**

- **Poca varianza** → datos muy parecidos
- **Mucha varianza** → datos muy distintos

¿Qué significa “explicar más varianza” en PCA?

☞ Ese componente recoge una mayor parte de los cambios importantes de los datos.

Es decir:

- Captura más información
- Resume mejor los datos originales

11. Reducción de la Dimensionalidad

11.2.2 Técnicas Lineales

11.2.2.1 PCA — Principal Component Analysis

🔍 ¿Qué hace exactamente PCA?

- Analiza cómo varían las variables entre sí
- Encuentra nuevas direcciones (componentes)
- Cada componente es una **combinación lineal** de las variables originales
- Ordena los componentes de mayor a menor importancia
- Permite quedarse solo con los primeros (menos dimensiones)

🧠 ¿Qué es un componente principal?

- Una nueva variable
- Formada como suma ponderada de las originales
- **No está correlacionada** con las demás componentes

$$\text{PC1} = 0.6 \times \text{edad} + 0.7 \times \text{ingresos} + 0.3 \times \text{visitas}$$

Requisitos para usar PCA

- ✓ Variables numéricas
- ✓ Escalado previo de los datos (muy importante)
- ✓ Dataset con colinealidad o muchas variables
- ✓ No necesitar interpretabilidad directa
- ✓ No requerir target

- 📌 PCA **no tiene sentido** si:
- tienes muy pocas variables
 - necesitas explicar cada variable original

11. Reducción de la Dimensionalidad

11.2.2 Técnicas Lineales

11.2.2.1 PCA — Principal Component Analysis

Ventajas de PCA

✓ No necesita target

Funciona en problemas **no supervisados**.

✓ Reduce dimensionalidad de forma general

Resume la información de muchas variables en pocas.

✓ Elimina colinealidad

Los componentes principales no están correlacionados entre sí.

✓ Acelera el entrenamiento de modelos

Menos variables → modelos más rápidos.

✓ Útil para visualización

Permite representar datos en 2D o 3D.

✓ Base matemática sólida y muy usada

Técnica clásica y ampliamente aceptada.

Desventajas de PCA

✗ No usa el target

Puede conservar varianza que no es relevante para la predicción.

✗ Pérdida de interpretabilidad

Los componentes no tienen significado directo.

✗ Es lineal

No captura relaciones no lineales.

✗ Requiere escalado

Si no se escalan las variables, el resultado puede ser engañoso.

✗ Puede eliminar información útil para clasificar

Porque solo optimiza varianza, no separación de clases.

11. Reducción de la Dimensionalidad

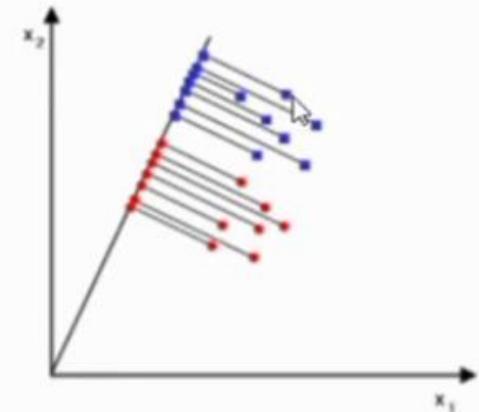
11.2.2 Técnicas Lineales

11.2.2.2 LDA — Linear Discriminant Analysis

LDA es una técnica que puede usarse como **reducción de dimensionalidad** supervisada que **busca proyectar los datos en un espacio de menor dimensión maximizando la separación entre las clases** del target. **Es también un algoritmo de Clasificación Lineal**

¿Qué es lo que busca LDA?

- Reduce las dimensiones a $N-1$
Busca una línea (o plano en general) que maximice la distancia entre las medias de las clases y minimice la dispersion dentro de cada clase
- Quiere que los centros de las clases queden lo más lejos posible entre sí después de la proyección.
- Quiere que los puntos de la misma clase queden lo más juntos posible.



 **LDA busca una proyección donde las clases estén muy separadas entre sí y muy compactas internamente.**

11. Reducción de la Dimensionalidad

11.2.2 Técnicas Lineales

11.2.2.2 LDA — Linear Discriminant Analysis

Ventajas de LDA

- **✓ Usa el target**
Tiene en cuenta las clases y no solo los datos.
- **✓ Reduce dimensionalidad de forma útil para clasificación**
No conserva información “general”, conserva la que **separa clases**.
- **✓ Muy eficaz con pocas dimensiones**
Especialmente bueno cuando hay muchas variables numéricas.
- **✓ Rápido y computacionalmente barato**
Mucho más rápido que técnicas no lineales.
- **✓ Mejora modelos de clasificación lineales**
A menudo mejora el rendimiento frente a usar las variables originales.

Desventajas de LDA

- **✗ Es lineal**
No captura relaciones no lineales entre variables.
- **✗ Máximo N-1 dimensiones**
Con pocas clases, reduce muy poco (ej. 2 clases → 1 dimensión).
- **✗ Supone clases relativamente bien distribuidas**
Funciona peor si las clases están muy mezcladas o son muy complejas.
- **✗ Sensible a outliers**
Valores extremos pueden afectar a la separación.
- **✗ Menos interpretable que variables originales**
Las nuevas variables son combinaciones de las originales.

11. Reducción de la Dimensionalidad

11.2.2 Técnicas Lineales

11.2.2.2 LDA — Linear Discriminant Analysis



Usa LDA cuando:

- El problema es de **clasificación**
- Tienes **variables numéricas**
- Quieres reducir dimensionalidad **para clasificar**
- Las clases están razonablemente separadas
- Quieres una técnica rápida y sencilla
- Tienes más variables que observaciones



No uses LDA cuando:

- No tienes target (problema no supervisado)
- El problema es de regresión
- Las relaciones son muy no lineales
- Hay muchas clases con pocos datos
- Necesitas interpretabilidad variable a variable

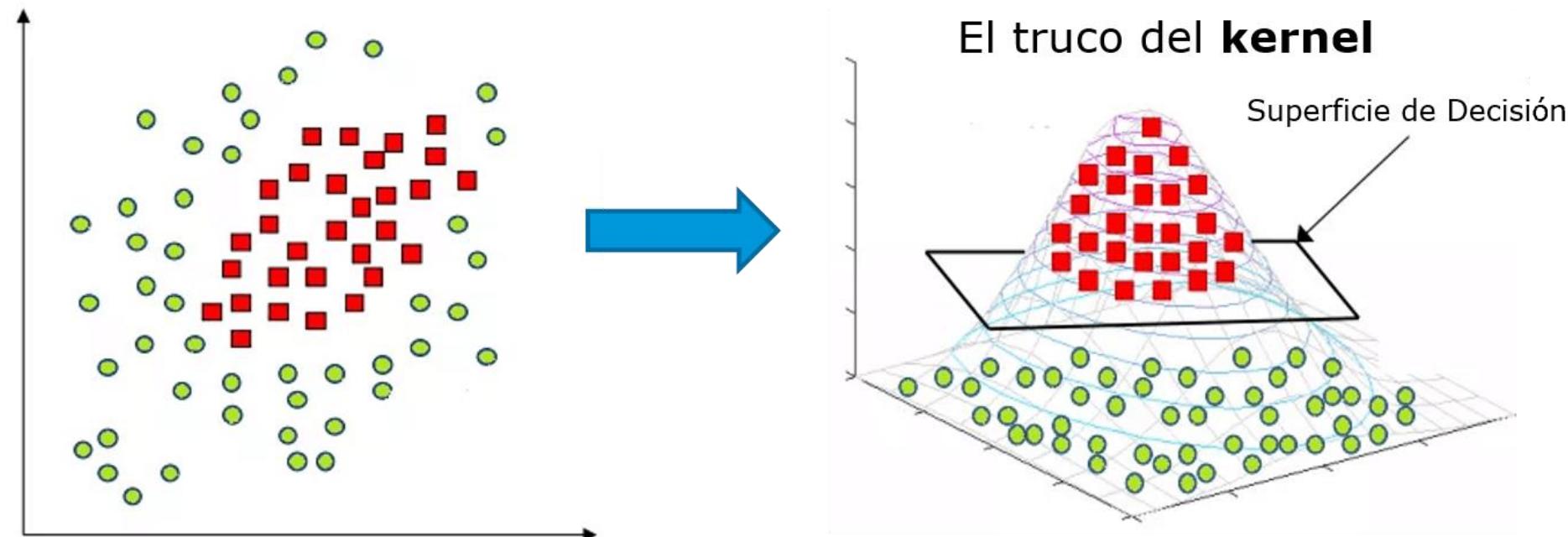
11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.1 Kernel PCA (Principal Component Analysis o análisis de componentes principales)

Kernel PCA es una extensión de **PCA** que permite capturar relaciones no lineales entre las variables.

De forma semejante a como ocurre en las máquinas de vector soporte, utiliza una función de kernel para transformar los datos de forma no lineal a un espacio de mayor dimensión en el que resulte más sencillo identificar relaciones entre los datos.



11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.1 Kernel PCA

🧠 ¿Por qué PCA normal no es suficiente?

PCA normal:

- Busca direcciones rectas
- Maximiza varianza en líneas rectas

Pero muchos datos reales:

- Están en **curvas**
- En **espirales**

🔍 Entonces... ¿qué hace Kernel PCA?

Kernel PCA:

1. Proyecta los datos a un **espacio de mayor dimensión**
 2. En ese espacio, la estructura se vuelve más simple
 3. Aplica PCA allí
 4. Vuelve a un espacio reducido
- f es una función **no lineal**
 - definida implícitamente por el kernel
 - **no se expresa como suma de variables**

📌 Diferencia CLAVE con PCA:

PCA (lineal):

$$PC1 = 0.6 \cdot X_1 + 0.3 \cdot X_2 - 0.1 \cdot X_3 + \dots$$

- ✓ Puedes ver pesos
- ✓ Puedes interpretar

Kernel PCA (no lineal):

$$KPC1 = f(X_1, X_2, X_3, \dots)$$

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.1 Kernel PCA

Ventajas de Kernel PCA

- ✓ Captura **estructuras no lineales**
- ✓ No necesita target (no supervisado)
- ✓ Útil para visualización compleja
- ✓ Extiende PCA sin cambiar el concepto base

Desventajas de Kernel PCA

- ✗ Más costoso computacionalmente
- ✗ Difícil de interpretar
- ✗ No escala bien a datasets grandes
- ✗ Elección del kernel y parámetros es crítica

¿Cuándo usar Kernel PCA?

Úsallo cuando:

- PCA no separa bien los datos
- La estructura es **no lineal**
- Quieres reducir dimensionalidad sin target
- Dataset no demasiado grande

No lo uses cuando:

- PCA ya funciona bien
- Necesitas interpretabilidad
- Dataset muy grande
- Buscas simplicidad

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.2 t-SNE

t-SNE es una técnica de reducción de dimensionalidad diseñada para visualizar datos complejos, colocando cerca en 2D o 3D los puntos que eran parecidos en el espacio original.

Dicho aún más simple:

- ☞ **t-SNE** reordena los puntos para que los que se parecen estén juntos y los que no, queden separados.

¿t-SNE crea nuevas variables o no?

☞ **t-SNE crea nuevas COORDENADAS, no nuevas variables utilizables.**

Observación 1 → (x₁, x₂, x₃, ..., x₂₀)
Observación 2 → (x₁, x₂, x₃, ..., x₂₀)



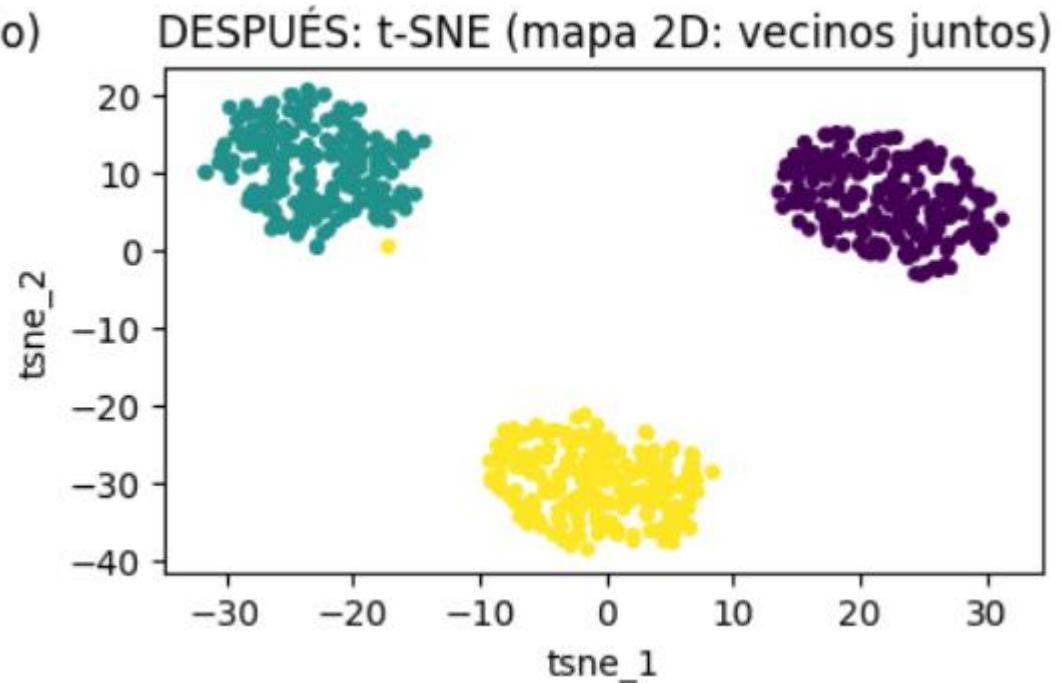
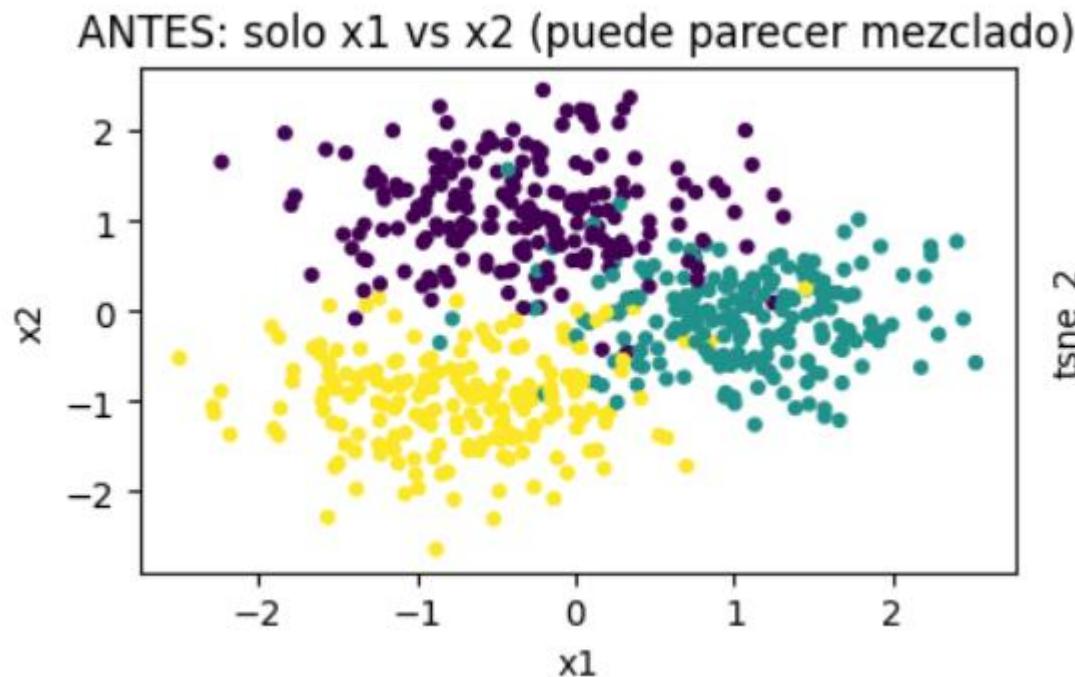
Después de t-SNE (2D):

Observación 1 → (tsne_{_1}, tsne_{_2})
Observación 2 → (tsne_{_1}, tsne_{_2})

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.2 t-SNE



11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.2 t-SNE

La ventaja Real de t-SNE:



t-SNE permite VER estructuras que no puedes detectar numéricamente.

Concretamente:

- ✓ Ver clústeres ocultos
- ✓ Ver si las clases se separan
- ✓ Ver si hay subgrupos
- ✓ Ver solapamientos
- ✓ Ver ruido

Ejemplo realista

Imagina que entrenas un modelo y no sabes por qué falla.

t-SNE puede mostrarte:

- que las clases están mezcladas
- que hay 3 subgrupos dentro de una clase
- que hay outliers



No arregla el modelo, pero te dice qué está pasando.

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.2 t-SNE

Ventajas de t-SNE

- ✓ **Hace visible la estructura de datos complejos**

Revela clústeres que no se ven en 2D con variables originales o PCA.

- ✓ **Excelente para exploración y análisis visual**

Ideal para “entender qué pasa” dentro de los datos.

- ✓ **Preserva vecindarios locales**

Los puntos que eran parecidos en alta dimensión aparecen juntos.

- ✓ **Muy útil con datos de alta dimensión**

Texto, embeddings, imágenes, salidas de redes neuronales.

- ✓ **No necesita el target**

Técnica no supervisada.

Desventajas de t-SNE

- ✗ **No es interpretable**

Las coordenadas (tsne_1, tsne_2) no tienen significado.

- ✗ **No sirve para entrenar modelos**

No debe usarse como paso de preprocesamiento para ML.

- ✗ **No conserva distancias globales**

La distancia entre grupos no es fiable.

- ✗ **No es estable**

Dos ejecuciones pueden dar mapas distintos (si no fijas random_state).

- ✗ **Costoso computacionalmente**

Lento en datasets grandes.

- ✗ **Difícil de aplicar a nuevos datos**

No hay una transformación clara “nuevo punto → mapa”.

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.2 t-SNE

✓ ¿Cuándo aplicar t-SNE?

✓ Úsallo cuando:

- Quieres **visualizar datos complejos**
- Quieres detectar **clústeres ocultos**
- Quieres comprobar si las clases **se separan**
- Estás en **fase exploratoria**
- Analizas embeddings (NLP, imágenes, deep learning)
- Quieres una **visualización clara en 2D o 3D**
- ✅ t-SNE es una **herramienta de diagnóstico visual.**

✗ ¿Cuándo NO aplicar t-SNE?

✗ No lo uses cuando:

- Quieres **entrenar un modelo**
- Necesitas **interpretar variables**
- Quieres comparar métricas numéricas
- Necesitas estabilidad entre ejecuciones
- Dataset muy grande (miles o millones de puntos)
- Necesitas transformar nuevos datos en producción

📌 En estos casos, usa:

- PCA
- LDA
- Autoencoders
- UMAP (según el caso)

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.3 UMAP

UMAP (Uniform Manifold Approximation and Projection) es una técnica **no lineal** de reducción de dimensionalidad que proyecta datos de alta dimensión a 2D/3D preservando la **estructura de vecindad** de los datos y parte de su estructura global.

✅ ¿Cuándo aplicar UMAP?

- Tienes datos de **muchas dimensiones**
- En Algoritmos de Clustering (DBSCAN/HDBSCAN)
- Quieres:
 - Ver clusters
 - Detectar subgrupos
 - Explorar estructura
- El dataset es: Mediano o grande
- Quieres algo más rápido y estable que t-SNE
- Combinado

⚠️ No es ideal cuando:

- Quieres:
 - Interpretabilidad de variables
 - Componentes con significado
 - Una transformación lineal estable para modelos clásicos
- Necesitas:
 - Reproducibilidad simple y explicación matemática sencilla

11. Reducción de la Dimensionalidad

11.2.3 Técnicas No Lineales

11.2.3.3 UMAP

✓ Ventajas de UMAP frente a t-SNE

- **Más rápido** y escala mejor a grandes datasets
- Preserva **mejor estructura global**
- Resultados **más estables** entre ejecuciones
- Más flexible con hiperparámetros
(`n_neighbors`, `min_dist`)
- Mejor para:
 - Exploración interactiva
 - Comparar distintas proyecciones

📌 Regla uso Técnicas No Lineales

¿Visualizar y explorar datos complejos?
→ **UMAP**

¿Máxima separación visual de clusters pequeños?
→ **t-SNE**

¿Crear nuevas features no lineales para un modelo?
→ **Kernel PCA**

11. Reducción de la Dimensionalidad

11.2.4 Métodos basados en Modelos

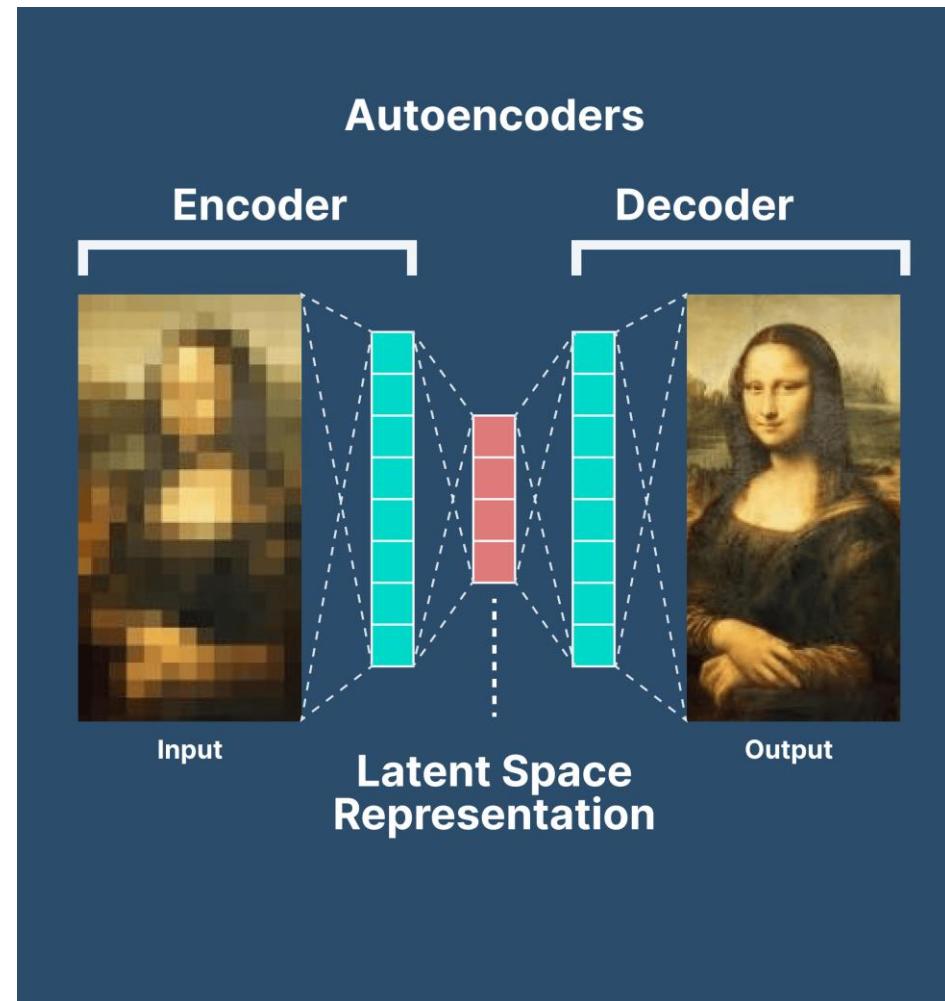
11.2.4.1 Autoencoders

Un **autoencoder** es un tipo de red neuronal artificial que se utiliza para aprender representaciones eficientes de datos.

El objetivo principal de un **autoencoder** es **reducir la dimensionalidad de los datos de entrada**, es decir, **comprimirlos en un espacio de características más pequeño**, y luego **reconstruir los datos de salida originales** a partir de esta representación comprimida.



Un autoencoder aprende qué información es importante porque solo puede pasar una versión comprimida de los datos.



11. Reducción de la Dimensionalidad

11.2.4 Métodos basados en Modelos

11.2.4.1 Autoencoders



Ventajas de los autoencoders

- Capturan **relaciones no lineales**
- Muy flexibles
- Escalan bien con muchos datos
- Útiles en datos complejos (imágenes, texto, sensores)



Limitaciones

- Requieren **muchos datos**
- Costosos computacionalmente
- Difíciles de interpretar
- Necesitan ajuste de arquitectura



Cuándo usar autoencoders

- ✓ Muchas variables
- ✓ Relaciones no lineales
- ✓ Dataset grande
- ✓ PCA no es suficiente



Cuándo NO usarlos

- ✗ Dataset pequeño
- ✗ Necesitas interpretación
- ✗ Problemas simples

12. Balanceo de Clases

12.1 ¿Qué es el Balanceo de Clases?

El **balanceo de clases** es un conjunto de técnicas utilizadas en aprendizaje automático y minería de datos para tratar **datasets desbalanceados**, es decir, **aquellos en los que unas clases tienen muchas más muestras que otras**.

Ejemplo típico:

- 95 % correos *no spam*
- 5 % correos *spam*

Si no se corrige este desbalance, el modelo puede “aprender” a predecir siempre la clase mayoritaria y aun así obtener una alta precisión, pero **mal rendimiento real**.

¿Por qué es importante el balanceo de clases?

- Evita que el modelo esté **sesgado hacia la clase mayoritaria**
- Mejora métricas relevantes como:
 - *Recall*
 - *F1-score*
 - *AUC-ROC*
- Es crucial en problemas como:
 - Detección de fraude
 - Diagnóstico médico
 - Fallos industriales

12. Balanceo de Clases

12.2 Principales Técnicas de Balanceo de Clases

1. A nivel de datos

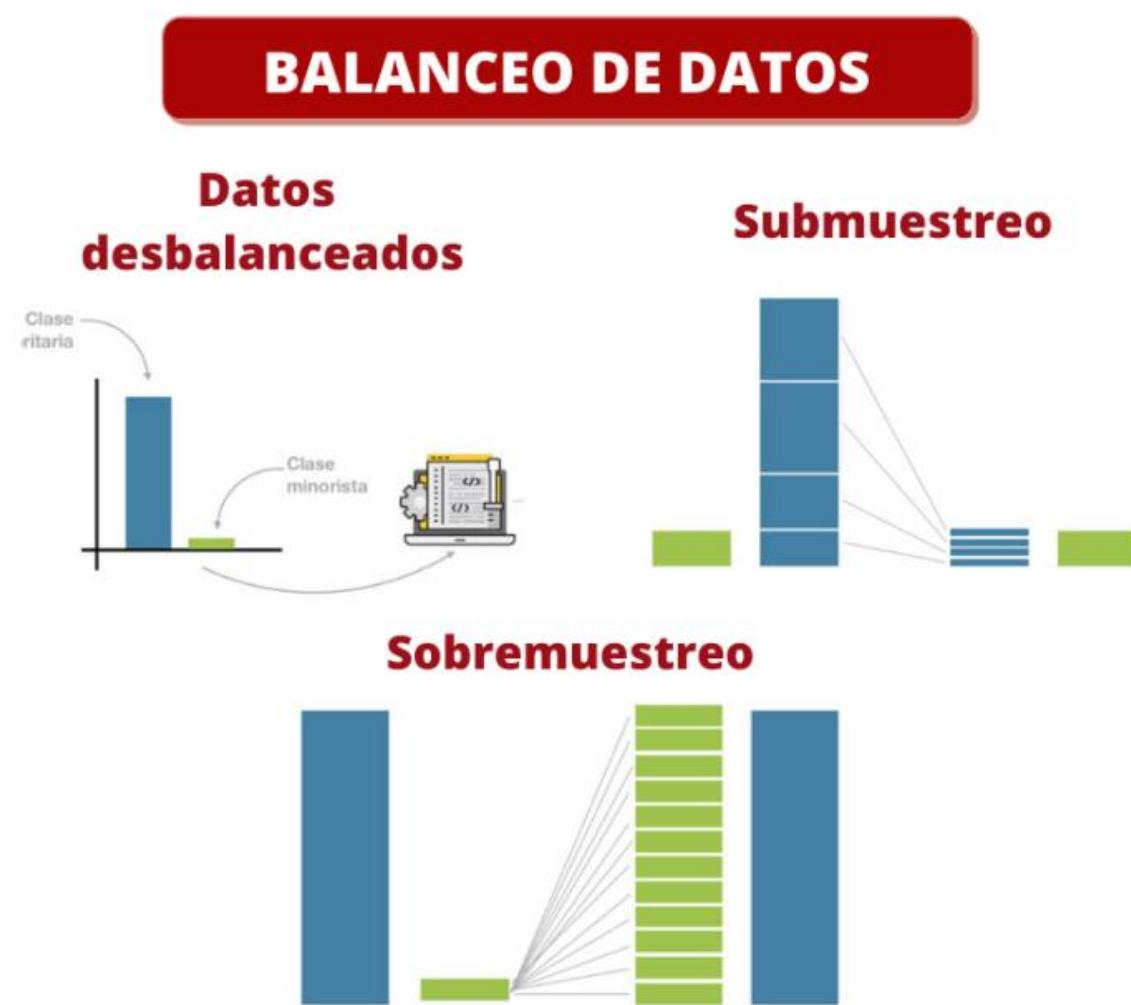
- Submuestreo (Undersampling)
- Sobremuestreo (Oversampling)

2. A nivel de Algoritmos

- `class_weight="balanced"`
- `class_weight={0:1, 1:5}`

3. A nivel de Modelo

- Balanced Random Forest
- EasyEnsemble
- RUSBoost



12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.1 Submuestreo (Undersampling)

El **undersampling** elimina **ejemplos de la clase mayoritaria**, ya sea de forma aleatoria o siguiendo ciertos criterios, hasta conseguir un conjunto más equilibrado.

Tipos comunes de Undersampling

- **Random Undersampling**: eliminación aleatoria de muestras
- **Tomek Links**: elimina ejemplos ambiguos cerca de la frontera de decisión
Es el único que no se puede fijar cantidad o porcentaje a eliminar
- **NearMiss**: conserva los ejemplos más cercanos a la clase minoritaria
- **Cluster Centroids**: reemplaza grupos por centroides

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.1 Submuestreo (Undersampling)

Ventajas del Submuestreo

- Reduce el tamaño del dataset
- Disminuye el tiempo de entrenamiento
- Evita el sesgo hacia la clase mayoritaria
- Sencillo de aplicar

Desventajas del Submuestreo

- Pérdida de información relevante
- Riesgo de eliminar patrones importantes
- Puede empeorar la generalización
- Resultados inestables si es aleatorio

¿Cuándo usar Undersampling?

- Datasets muy grandes
- Desbalance extremo (ej. 1:100 o más)
- Cuando la clase mayoritaria tiene mucha redundancia
- Modelos costosos computacionalmente
- Primeras pruebas o prototipos rápidos

¿Cuándo NO usar Undersampling?

- Datasets pequeños
- Cuando cada dato es crítico (medicina, fraude)
- Si la clase mayoritaria es compleja
- Cuando hay ruido y solapamiento entre clases
- Si ya tienes pocas muestras totales

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.1 Submuestreo (Undersampling)

Tomek Links

Un **Tomek Link** es un par de puntos de **clases distintas** que son **vecinos más cercanos entre sí**.

Eso suele indicar:

- Solapamiento entre clases
- Frontera ambigua
- Ruido en los datos

¿Qué hace el algoritmo?

- Busca todos los pares de puntos (A, B) tales que:
 - A es el vecino más cercano de B
 - B es el vecino más cercano de A
 - Y pertenecen a **clases distintas**
- Cada uno de esos pares es un **Tomek Link**.
- En práctica para balanceo:
 - Se **elimina el punto de la clase mayoritaria** de cada par
 - (A veces se eliminan ambos, pero lo habitual es quitar solo el mayoritario)

¿Qué efecto tiene?

- ✓ Limpia la frontera entre clases
- ✓ Reduce **solapamiento y ruido**
- ✓ No cambia mucho el tamaño del dataset
- ✗ No balancea mucho por sí solo

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.1 Submuestreo (Undersampling)

NearMiss

Reduce la clase mayoritaria quedándose **solo con los ejemplos más cercanos a la clase minoritaria** (los casos difíciles, cerca de la frontera).

Es decir, hace lo opuesto a **Tomek**:

- Elimina ejemplos mayoritarios “fáciles” (lejos de la minoritaria)
- Conserva ejemplos mayoritarios “difíciles” (cerca de la minoritaria)

🧠 ¿Qué hace el algoritmo?

Para cada punto de la **clase mayoritaria**:

1. Busca sus **k vecinos más cercanos de la clase minoritaria**

2. Calcula una **distancia** (media u otra según la variante)

3. Usa esa distancia como medida de:

“Qué tan cerca está este punto de la otra clase”



¿Qué efecto tiene?

- Reduce mucho la clase mayoritaria
- Centra el entrenamiento en la **frontera entre clases**

Luego:

- Ordena los puntos mayoritarios por esa distancia
- Se queda con los **más cercanos**
- Elimina el resto

Así:

El dataset resultante queda centrado en la **zona de la frontera entre clases**

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.1 Submuestreo (Undersampling)

Tomek Links vs NearMiss

🧠 ¿Cuándo usar cada uno?



Usa **NearMiss** cuando:

- La clase mayoritaria es **muy grande**
- Necesitas:
 - Reducir **mucho** el tamaño del dataset
 - Acelerar entrenamiento
- Tu objetivo es:
 - Mejorar rendimiento en la **clase minoritaria**
- Tu modelo:
 - Aprende con ejemplos cercanos a la frontera (SVM, k-NN, regresión logística)



Usa **Tomek Links** cuando:

- Hay **solapamiento** entre clases
- Quieres:
 - **Limpiar frontera**
 - Quitar puntos ambiguos / ruidosos
- No quieres:
 - Perder muchos datos
- Especialmente útil:
 - Como post-procesado tras **SMOTE**

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.2 Sobremuestreo (Oversampling)

El **sobremuestreo** es un conjunto de técnicas que **aumentan el número de muestras de la clase minoritaria** para reducir el desbalanceo entre clases.

Tipos comunes de Undersampling

1. Random OverSampling
2. SMOTE (Synthetic Minority Over-sampling Technique)
3. Borderline-SMOTE
4. ADASYN
5. SMOTE + Limpieza (métodos combinados)

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.2 Sobremuestreo (Oversampling)

Ventajas

- ✓ No se pierde información, ya que no se eliminan muestras.
- ✓ Mejora el aprendizaje de la clase minoritaria (recall, F1).
- ✓ Útil en datasets pequeños o cuando la clase minoritaria es crítica.
- ✓ Permite ajustar la proporción entre clases.

Desventajas

- ✗ Riesgo de sobreajuste, especialmente con Random Oversampling.
- ✗ Puede generar datos sintéticos poco realistas (SMOTE).
- ✗ Aumenta el coste computacional.
- ✗ Puede amplificar ruido o errores de etiquetado.

Cuándo usar Oversampling

- ✓ Dataset pequeño o mediano.
- ✓ Clase minoritaria importante (fraude, diagnóstico, fallos raros).
- ✓ Variables mayoritariamente continuas.
- ✓ Modelos sensibles al desbalanceo (logística, SVM, KNN, redes neuronales).

Cuándo NO usar Oversampling

- ✗ Dataset muy grande.
- ✗ Muchas variables categóricas.
- ✗ Datos muy ruidosos.
- ✗ Clases muy solapadas.

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.2 Sobremuestreo (Oversampling)

Tipos comunes de Undersampling

1. Random OverSampling

📌 ¿Qué hace?

Duplica aleatoriamente muestras existentes de la clase minoritaria

🔍 Características

- Muy simple
- Rápido

⚠️ No crea información nueva

Riesgo

- Sobreajuste (mismas muestras repetidas)

2. SMOTE

(Synthetic Minority Over-sampling Technique)

📌 ¿Qué hace?

- Crea **nuevas muestras sintéticas**
- Interpolando entre vecinos de la clase minoritaria

🔍 Características

- Más realista que duplicar
- Muy usado en la práctica

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.2 Sobremuestreo (Oversampling)

Tipos comunes de Undersampling

3.Borderline-SMOTE



¿Qué hace?

- Genera muestras **cerca de la frontera de decisión**
- Se centra en los casos difíciles



Ventaja

- Mejora la clasificación en zonas críticas



¿Qué cambia?

Primero clasifica los ejemplos minoritarios en:

- **Safe** (rodeados de minoritarios)
- **Danger** (cerca de mayoritarios)
- **Noise** (rodeados de mayoritarios)



- Genera ejemplos **solo (o sobre todo)** a partir de los puntos en **zona Danger** (cerca de la frontera)
- Refuerza la clase minoritaria justo donde el modelo se equivoca más: en la frontera.

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.2 Sobremuestreo (Oversampling)

Tipos comunes de Undersampling

4. ADASYN

¿Qué hace?

- Variante de SMOTE
- Genera **más ejemplos** en las zonas **más difíciles**
- (donde la minoritaria está rodeada de mayoritarios)
- y **menos** en las zonas fáciles.

Idea clave

“Más datos donde más se necesitan”

Regla rápida:

- Balanceo simple y estable → **SMOTE**
- Reforzar frontera → **Borderline-SMOTE**
- Enfocar en zonas difíciles → **ADASYN**

¿Qué cambia?

- 1.Para cada ejemplo minoritario:
 - Mira cuántos de sus vecinos son de la **clase mayoritaria**
- 2.Eso le da una medida de:

“Qué tan difícil es clasificar este punto”
- 3.Luego:
 - Genera **más puntos sintéticos** alrededor de los ejemplos más difíciles
 - **Y menos o ninguno** alrededor de los fáciles

Es decir:

La cantidad de datos sintéticos depende de la dificultad local.

12. Balanceo de Clases

12.2.1 A Nivel de Datos

12.2.1.2 Sobremuestreo (Oversampling)

Tipos comunes de Undersampling

5.SMOTE + Limpieza (métodos combinados)

SMOTE + Tomek Links

🧠 ¿Qué hace:

- Quita **pares ambiguos** (Tomek Links)
- Normalmente elimina **solo la clase mayoritaria**

✅ Ventajas:

- Limpieza **suave**
- Frontera más clara
- No borra muchos datos

✗ Desventajas:

- No limpia ruido fuerte

👉 Cuándo usar:

- Solapamiento moderado
- Opción “segura” y equilibrada

SMOTE + ENN

🧠 Qué hace:

- Elimina puntos **mal clasificados por sus vecinos**
- Puede borrar puntos de **ambas clases**

✅ Ventajas:

- Limpieza **agresiva**
- Reduce ruido y solapamiento fuerte

✗ Desventajas:

- Puede borrar **demasiados datos**
- **Puede borrar datos reales y correctos, incluso de la clase minoritaria.**

👉 Cuándo usar:

- Datos **muy ruidosos** o muy mezclados
- Se quiere limpiar mucho.

12. Balanceo de Clases

Undersampling vs Oversampling

Reglas prácticas:

- ◊ **Pocos datos** → Oversampling
- ◊ **Muchos datos** → Undersampling
- ◊ **Clase minoritaria crítica** → Oversampling
- ◊ **Clase mayoritaria redundante** → Undersampling
- ◊ **Dudas** → probar ambos con validación cruzada



- **Oversampling añade información (real o sintética); undersampling sacrifica información para ganar eficiencia.**
- **Oversampling prioriza aprender mejor la clase rara; undersampling prioriza simplificar el problema.**

12. Balanceo de Clases

12.2.2 A Nivel de Algoritmos

Son técnicas que **No modifican el dataset**, sino que **modifican cómo el algoritmo aprende**, dando más **importancia** a los errores en las clases minoritarias.

12.2.2.1 class_weight Opción `class_weight="balanced"`



¿Qué hace?

Calcula automáticamente los pesos según la **frecuencia de cada clase**:

$$\text{peso_clase} = \frac{N}{n_{\text{clases}} \cdot N_{\text{clase}}}$$

Donde:

- N = nº total de muestras
- N_claseN = nº de muestras de esa clase

Ejemplo:

1000 muestras: Pesos aproximados:

- Clase 0: 900 • Clase 0: $1000/(2 \cdot 900) \approx 0.56$
- Clase 1: 100 • Clase 1: $1000/(2 \cdot 100) = 5$

☞ Los errores en la clase minoritaria “pesan” mucho más.



Efecto típico en métricas:

Si aumentas el peso de la **clase minoritaria**:



Suele mejorar

- **Recall** de la clase minoritaria (menos falsos negativos)
- A veces **F1** de la minoritaria



Suele empeorar

- **Precision** de la clase minoritaria (más falsos positivos)
- **Accuracy** global (porque sacrificas aciertos en la mayoritaria)

12. Balanceo de Clases

12.2.2 A Nivel de Algoritmos

12.2.2.1 class_weight Opción `class_weight={0:1, 1:5}`

🧠 ¿Qué hace?

Tú defines **explícitamente** el peso de cada clase.

Significa:

- Un error en clase 1 vale **5 veces más** que un error en clase 0

📌 ¿Por qué usarlo?

- Para **ajustar el trade-off** entre:
 - Precision vs Recall
 - Falsos positivos vs falsos negativos
- Ej: En detección de fraude:
 - Prefieres fallar pocos fraudes
→ das más peso a la clase “fraude”

✓ Ventajas

- Control total
- Puedes optimizar una métrica concreta (recall, F1, etc.)

✗ Inconvenientes

- Necesita:
 - Pruebas
 - Validación
 - Búsqueda de hiperparámetros

12. Balanceo de Clases

12.2.3 A Nivel de Modelos

Son **Modelos** diseñados **específicamente** para trabajar con clases desbalanceadas, donde el propio modelo **integra el balanceo en su funcionamiento interno**.

Tipos principales:

- 1) Balanced Random Forest
- 2) EasyEnsemble
- 3) RUSBoost

Su idea común es:

Entrenar **varios modelos** sobre **subconjuntos balanceados** del dataset (mediante undersampling de la clase mayoritaria) y **combinar sus predicciones** para mejorar el rendimiento en la clase minoritaria sin perder toda la información de la clase mayoritaria.

La diferencia está en:

Cómo crean los subsets y cómo entran/combinan los modelos.

12. Balanceo de Clases

12.2.3 A Nivel de Modelos

12.2.3.1 Balanced Random Forest

Cómo crea los subsets

- Para **cada árbol** del bosque:
 - Toma **toda la clase minoritaria**
 - Hace **undersampling aleatorio** de la mayoritaria hasta igualar tamaños
- Resultado:
 - Cada árbol ve un **dataset balanceado distinto**

Cómo entrena y combina

- Entrena un **árbol por subset**
- Los árboles se entran:
 - De forma **independiente**
- Combina:
 - Por **votación mayoritaria** (como Random Forest)

Clave

Es un Random Forest donde **el balanceo ocurre por arbol**

12. Balanceo de Clases

12.2.3 A Nivel de Modelos

12.2.3.2 EasyEnsemble

Cómo crea los subsets

- Genera **K subsets balanceados**:
 - Cada uno con **toda la minoritaria**
 - Y un **undersampling** distinto de la mayoritaria
- Estos subsets se crean:
 - **Antes** de entrenar los modelos

Cómo entrena y combina

- En cada subset:
 - Entrena un **clasificador completo** (a menudo un AdaBoost)
- Obtienes:
 - K clasificadores
- Combina:
 - Por **votación** o promedio de probabilidades

Clave

Se entrenan múltiples modelos sobre distintos subconjuntos balanceados del dataset y se combinan sus salidas para tomar la decisión final.

12. Balanceo de Clases

12.2.3 A Nivel de Modelos

12.2.3.3 RUSBoost

Cómo crea los subsets

- No crea todos los subsets al principio.
- En **cada iteración de boosting**:
 - Aplica **Random Undersampling** a la clase mayoritaria
 - Construye un subset balanceado **dinámicamente**

Cómo entrena y combina

- Entrena un **clasificador débil** en cada iteración
- Cada nuevo clasificador:
 - Da más peso a los ejemplos difíciles (boosting)
- Combina:
 - Como boosting: **suma ponderada** de clasificadores

Clave

Es boosting donde **cada iteración usa un subset balanceado distinto**.

12. Balanceo de Clases

12.2.3 A Nivel de Modelos

Métodos de balanceo a nivel de modelo — resumen

Método	Cuándo usar	Mejora	Empeora
Balanced Random Forest	Opción robusta y estable para desbalance	Recall (minoritaria), F1	Accuracy, Precision
EasyEnsemble	Mayoritaria muy grande,quieres estabilidad	Recall, F1 (minoritaria)	Accuracy, Precision
RUSBoost	Desbalance muy fuerte,frontera difícil	Recall (mucho), F1	Precision, Accuracy, sensible al ruido



Todos priorizan detectar mejor la clase minoritaria (suben recall/F1) a costa de bajar la accuracy y, a menudo, la precision.

¿Qué has aprendido?

- Qué es el Preprocesamiento de datos.
- Por qué es tan importante.
- Flujo de Preprocesamiento de datos.



“La inteligencia artificial es la nueva electricidad.”

Andrew Ng, director del laboratorio de Inteligencia Artificial en Stanford