

K SCHOOL

Supervised Learning

Javier Cañadillas - javier@canadillas.org



Esta presentación:

bit.ly/ks-sl-day1

Notas adicionales:

bit.ly/ks-sl-notes

Sobre mi (*shameless self promotion*)

- **Hoy: Google Cloud Customer Engineer**
- Antes: Arquitecto y Especialista diversas áreas en Oracle, HP y Sun
- Profesor invitado Master Data Science profesional UNED 2018
- Profesor asistente IE - School of Human Sciences & Technology
- Profesor UC3M - IoT Software Engineering & PhD student
- Computer Science Master Degree (UC3M)
- Licenciado en Ciencias Físicas - Física Industrial (UNED)
- Ingeniería aeronáutica - Navegación y Transporte Aéreo (UPM)

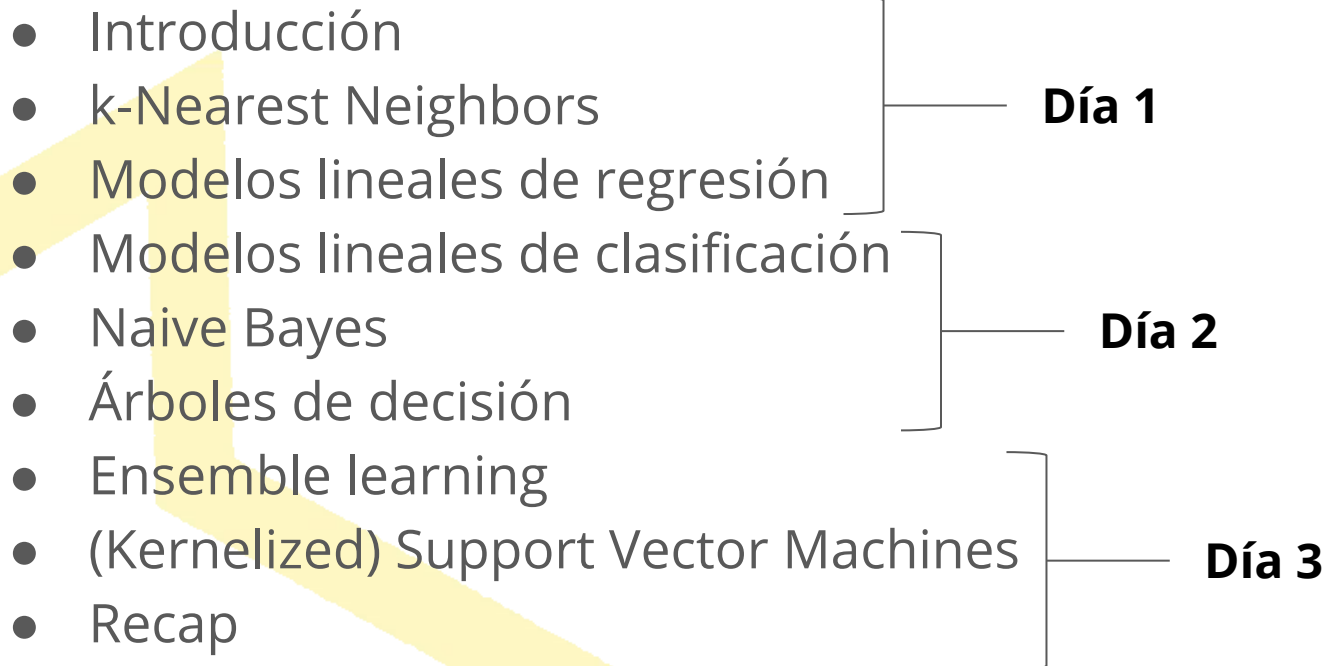
Temario

- Introducción - Aprendizaje supervisado, regresión y clasificación
- Modelos de aprendizaje supervisado
- Evaluación de modelos - Sobre la marcha
- Laboratorios y ejercicios
- Refrescos, recapitulaciones y conclusiones

NOT(Temario)

- Matemáticas avanzadas y demostraciones
- Ingeniería de características
- Técnicas de descenso de gradiente
- Optimización y evaluación avanzada de modelos

Agenda y reparto aproximado

- Introducción
 - k-Nearest Neighbors
 - Modelos lineales de regresión
 - Modelos lineales de clasificación
 - Naive Bayes
 - Árboles de decisión
 - Ensemble learning
 - (Kernelized) Support Vector Machines
 - Recap
- Día 1**
- Día 2**
- Día 3**
- 

Repositorio de código

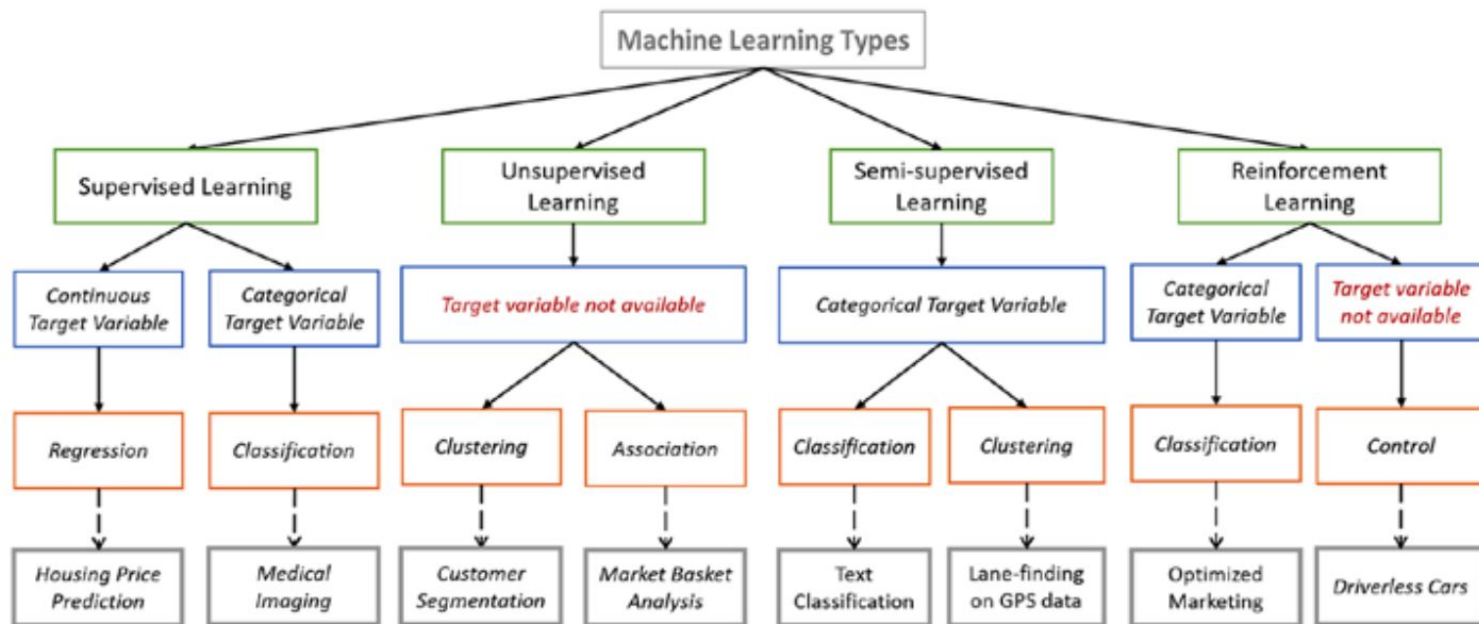


gitlab.com/Sh3llD00m/[kschool-supervised-learning.git](https://gitlab.com/Sh3llD00m/kschool-supervised-learning.git)



0. Introducción

Aprendizaje supervisado en ML



Aprendizaje Supervisado vs No Supervisado

- Tenemos conocimiento previo de valores de salida (labels o etiquetas)
- Objetivo: encontrar función que, dado un dataset de entrada y unas salidas, aproxime de la mejor forma la relación existente entre ellos.
- Tipos de algoritmos supervisados:
 - Clasificación (binaria o multiclase)
 - Regresión (predicción número continuo)

Motores de reglas ↔ Aprendizaje Supervisado

```
import numpy as np
from sklearn.datasets import make_classification

rs = np.random.RandomState(42)
X, y = make_classification(n_samples = 10, random_state = rs)
```

Motores de reglas ↔ Aprendizaje Supervisado

```
def tomar_decision_super_importante(X):  
    """  
    Decidir si pasa algo gordo  
    """  
    row_sums = X.sum(axis=1)  
    return (row_sums > 0).astype(int)  
  
tomar_decision_super_importante(X)
```

Ejemplo de aprendizaje supervisado

```
from sklearn.linear_model import LogisticRegression

def aprender_leccion_vital(X,y):
    """
    Aprender una lección y aplicarla en el futuro
    """
    model = LogisticRegression().fit(X,y)
    return (lambda x: model.predict(x))

# Aprender una lección y aplicarla
decision_informada = aprender_leccion_vital(X,y)(X)
print(decision_informada)
```

¿Qué es aprendizaje supervisado?

Es un método de aprendizaje que **aprende una función** a partir de un **conjunto de muestras ya etiquetada** que **aproxima valores futuros de y**

$$\hat{y} = f(X, y; \theta)$$

Valor futuro

Función de
predicción

Matriz de
muestras

Vector de
etiquetas

Parámetros

Función de coste (*loss function*)

- Cuantifica un coste que el algoritmo minimiza
- Es una medida de lo bien que un modelo se ajusta a un problema

$$L(y, \hat{y}) \in \mathbb{R}$$

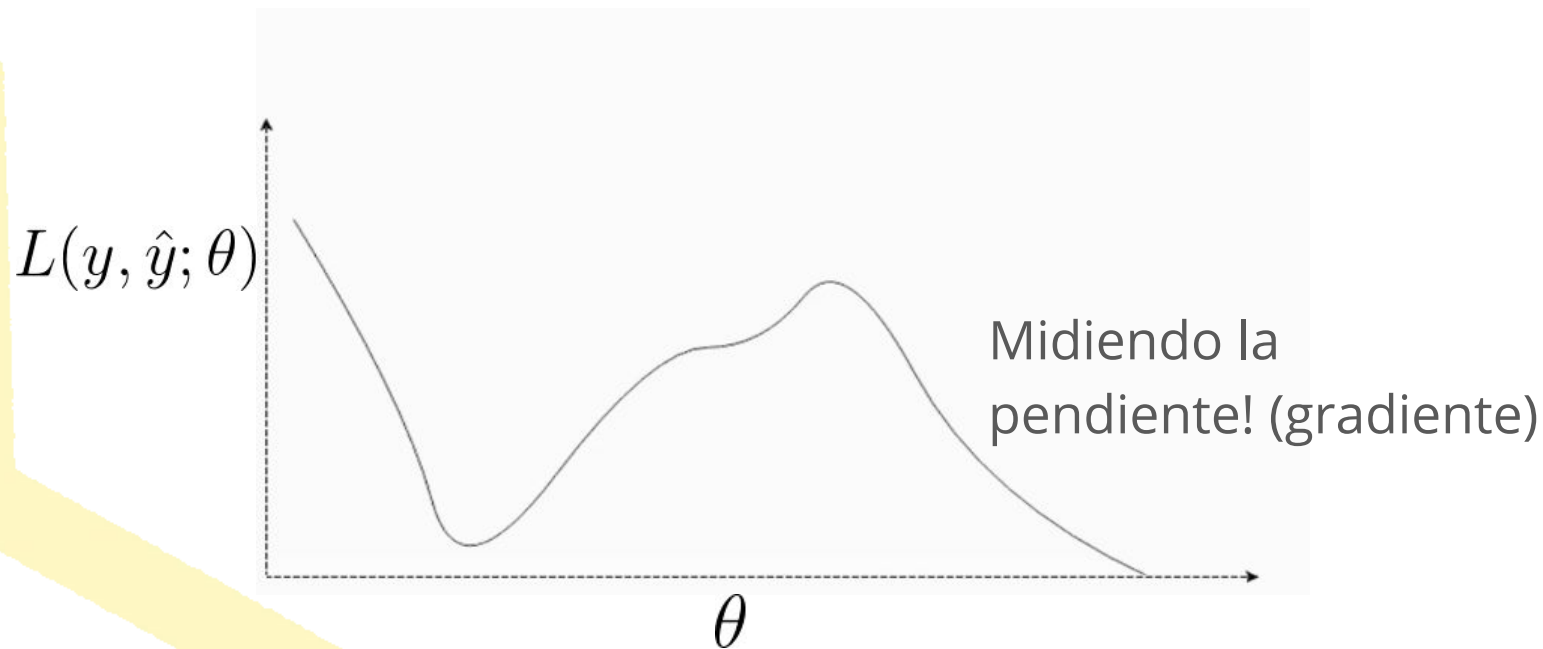
$$L = - \sum_x p(x) \log(1 - p(x))$$

Entropía cruzada (clasificación)

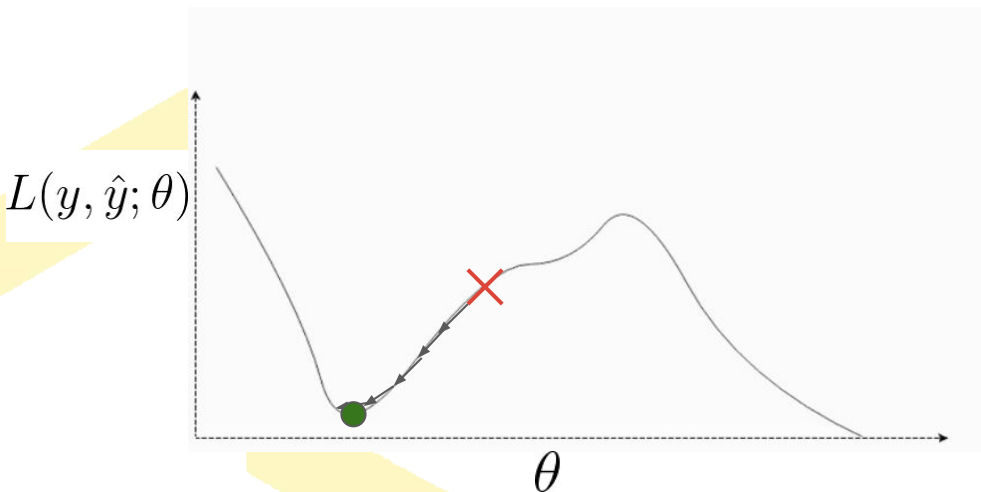
$$L = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Sumatorio error cuadrático (regresión)

Queremos minimizar L - ¿cómo?



Algoritmo general de minimización de gradiente



1. Inicializamos los parámetros/pesos (θ) de manera aleatoria
2. Calculamos el gradiente \mathbf{G} de la función de coste \mathbf{L} con respecto a los parámetros
3. Actualizamos los pesos con una cantidad proporcional al gradiente: $\mathbf{w} = \mathbf{w} - (\text{learning_rate}) * \mathbf{G}$
4. Repetimos hasta que L ya no se reduce más o alcanzamos otro criterio de parada.

¿Quién querrá comprar un barco?

Edad	# coches en propiedad	Tiene casa	# niños	Estado civil	Tiene perro	Tiene barco
66	1	sí	2	viudo	no	sí
52	2	sí	3	casado	no	sí
22	0	no	0	casado	sí	sí
25	1	no	1	soltero	no	no
44	0	no	2	divorciado	sí	no
39	1	sí	2	casado	sí	no
26	1	no	2	soltero	no	no
40	3	sí	1	casado	sí	no
53	2	sí	2	divorciado	no	sí
64	2	sí	3	divorciado	no	no
58	2	sí	2	casado	sí	sí
33	1	no	1	soltero	no	no

Modelo 1

Edad	# coches en propiedad	Tiene casa	# niños	Estado civil	Tiene perro	Tiene barco
66	1	sí	2	viudo	no	sí
52	2	sí	3	casado	no	sí
22	0	no	0	casado	sí	sí
25	1	no	1	soltero	no	no
44	0	no	2	divorciado	sí	no
39	1	sí	2	casado	sí	no
26	1	no	2	soltero	no	no
40	3	sí	1	casado	sí	no
53	2	sí	2	divorciado	no	sí
64	2	sí	3	divorciado	no	no
58	2	sí	2	casado	sí	sí
33	1	no	1	soltero	no	no

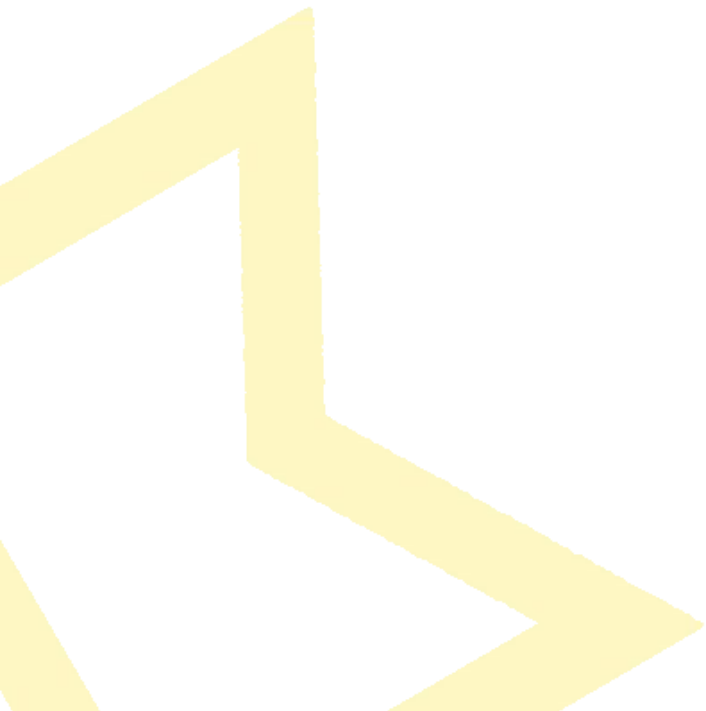
(> 45 años) y (< 3 niños o (no divorciado)) → Sí

Modelo 2

Edad	# coches en propiedad	Tiene casa	# niños	Estado civil	Tiene perro	Tiene barco
66	1	sí	2	viudo	no	sí
52	2	sí	3	casado	no	sí
22	0	no	0	casado	sí	sí
25	1	no	1	soltero	no	no
44	0	no	2	divorciado	sí	no
39	1	sí	2	casado	sí	no
26	1	no	2	soltero	no	no
40	3	sí	1	casado	sí	no
53	2	sí	2	divorciado	no	sí
64	2	sí	3	divorciado	no	no
58	2	sí	2	casado	sí	sí
33	1	no	1	soltero	no	no

> 50 años → Sí

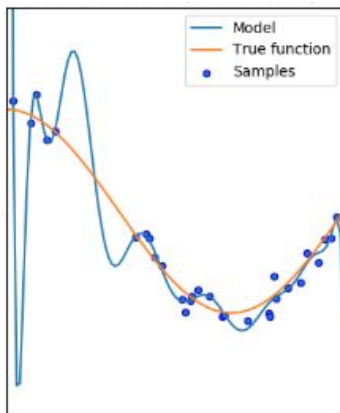
¿Cuál es mejor?



Underfitting, Overfitting y generalización

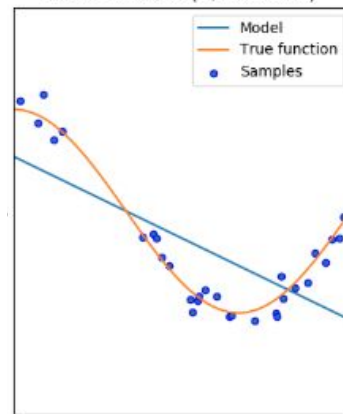
Sobreajuste (overfitting)

- Alto grado de complejidad
- Funciona bien en training set
- No generaliza bien

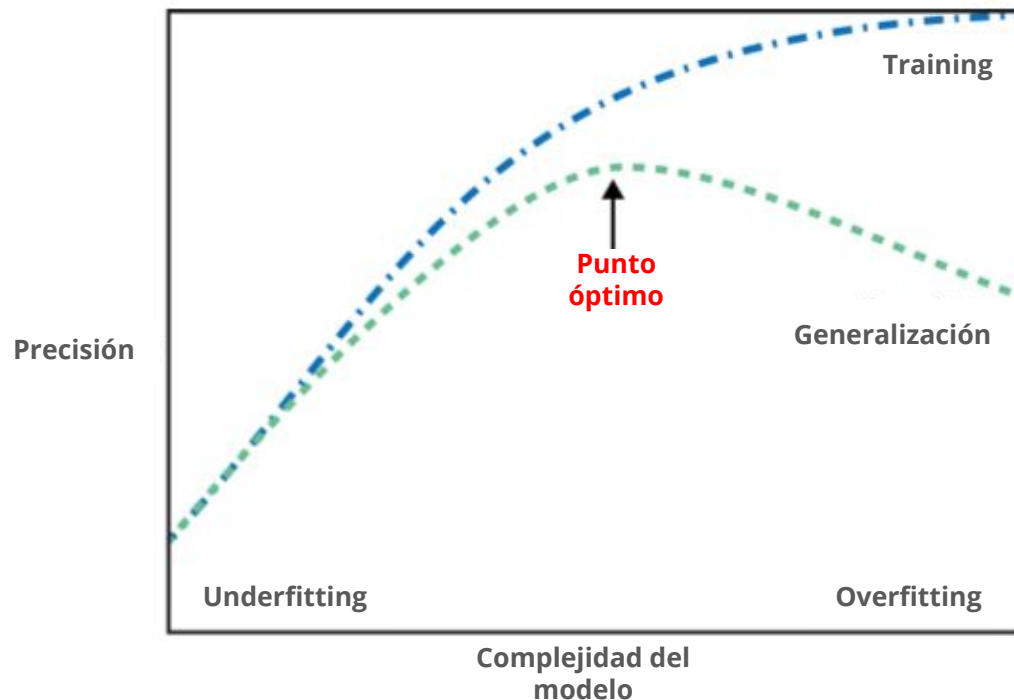


Subajuste (underfitting)

- Demasiado sencillo
- Predice mal, incluso en training set
- Generaliza demasiado



Se trata de buscar un compromiso



Complejidad del modelo y tamaño del dataset

- \uparrow Dataset \rightarrow \uparrow Complejidad
 - ¿Y si tenemos 100k entradas de posibles compradores de barcos?
- **Recoger más datos** funciona muy bien para los modelos de Aprendizaje Supervisado



Lab 0 - Datasets de ejemplo

1. Conclusiones

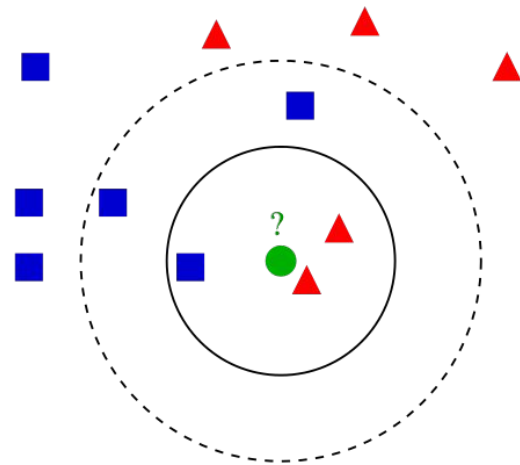
- El uso de **datasets sencillos** es útil
- Hay que probar con **datasets reales conocidos**
- Importante desarrollar una **intuición**
- Realizar siempre **trabajo exploratorio** de los datos



1. k-Nearest Neighbors

k-Nearest Neighbors

- Algoritmo sencillo
 - No paramétrico
 - Basado en instancias
- Clasificación, o regresión
 - Training mínimo: construir modelo → almacenar dataset
- Predicción: encontrar **datapoints más próximos** en training set
 - Por defecto: min(métrica *Minkowski* o distancia euclídea)

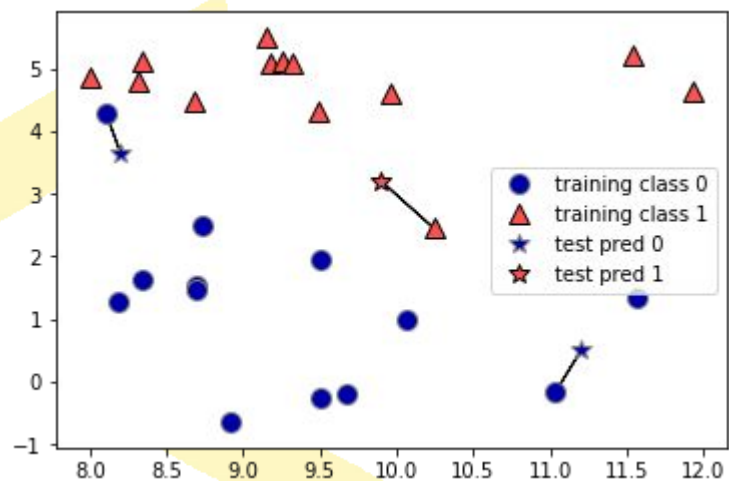


Manhattan
$$d(p, q) = \sum_{i=1}^n |q_i - p_i|$$

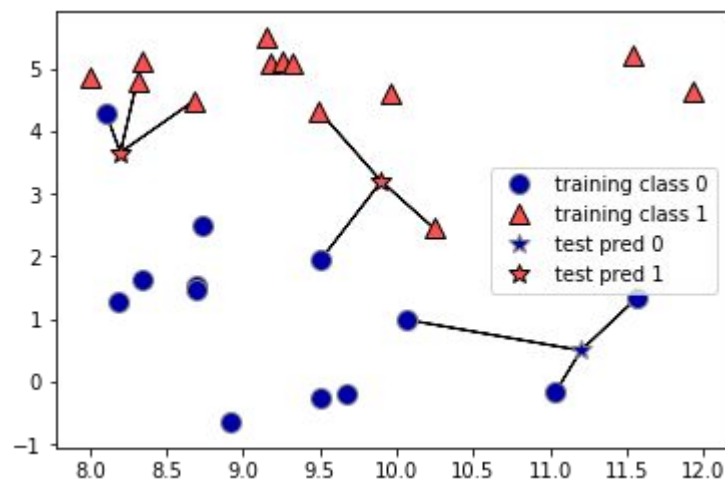
Euclídea
$$d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

Minkowski
$$d(p, q) = \sum_{i=1}^n ((|q_i - p_i|)^q)^{\frac{1}{q}}$$

k-Nearest Neighbors



Neighbors = 1



Neighbors = 3



Lab 1 - k-Nearest Neighbors



1. Conclusiones

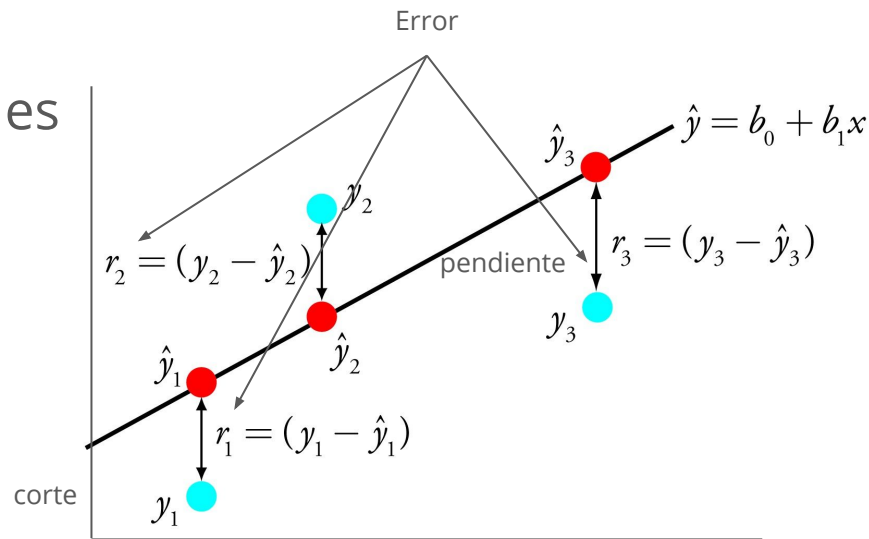
- Dos parámetros importantes
 - Número de vecinos → 3 a 5, pero probar
 - Métrica de distancia → Fuera de alcance
- Fácil de entender, rendimiento razonable
 - Training rápido
 - Testing lento para grandes datasets
- Se usa menos que la **regresión lineal**

2. Modelos lineales de regresión

Regresión Lineal

- Simple, pero potente
- Aproxima la relación entre variables de entrada y la variable objetivo mediante un hiperplano
 - Recta, en una dimensión
- Su forma general es:

$$\hat{y} = w_0 \cdot x_0 + w_1 \cdot x_1 + \dots + w_p \cdot x_p + b$$



Un mínimo de matemáticas

$$\hat{y} = w_0 \cdot x_0 + w_1 \cdot x_1 + \dots + w_n \cdot x_n \quad x_0 = 1, w_0 = b \quad \leftarrow \text{Misma ecuación que antes}$$

$$h_{\theta}(x) = \theta^T x \begin{pmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}$$

$$J(\theta) = \frac{1}{2m} (X\theta - y)^T (X\theta - y) \quad \leftarrow \text{Queremos minimizar esta función de coste, que no es más que la expresión matricial de los mínimos cuadrados}$$

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0 \quad \leftarrow \text{El es gradiente} \quad J(\theta_{0\dots n}) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\theta = (X^T X)^{-1} X^T y \quad \leftarrow \text{Esta es la ecuación normal, cálculo a fuerza bruta de los pesos, que supone que la matriz es invertible (mucho suponer)}$$

Coeficiente de determinación

- R^2 da una medida de cómo de bien se ajusta un modelo a los datos
 - También correlación (R) $\rightarrow R^2$ más interpretable

Es el porcentaje de variación en los datos explicado por la relación entre dos variables

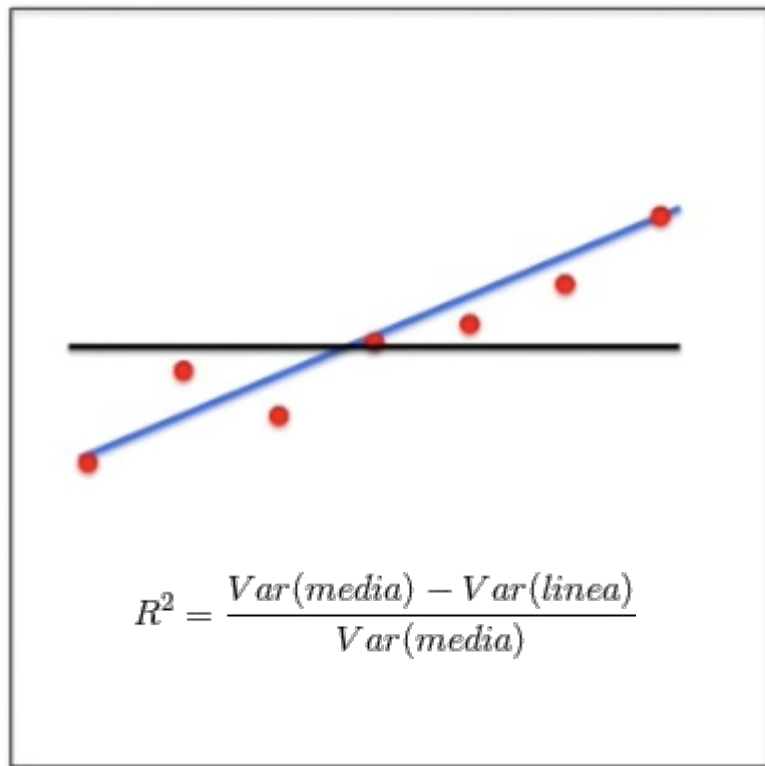
O, dicho de otra forma:

- ¿Cuánto mejor se ajusta la regresión lineal a los datos que simplemente la media?

R^2 - Significado

- $\text{Var}(\text{media}) = 32$
- $\text{Var}(\text{línea}) = 8$
- $R^2 = 0.81$, hay un **81% menos de variación** alrededor de la línea que alrededor de la media
- **La mayoría de la variación en los datos se explica por la relación tamaño/peso**

Peso
ratón

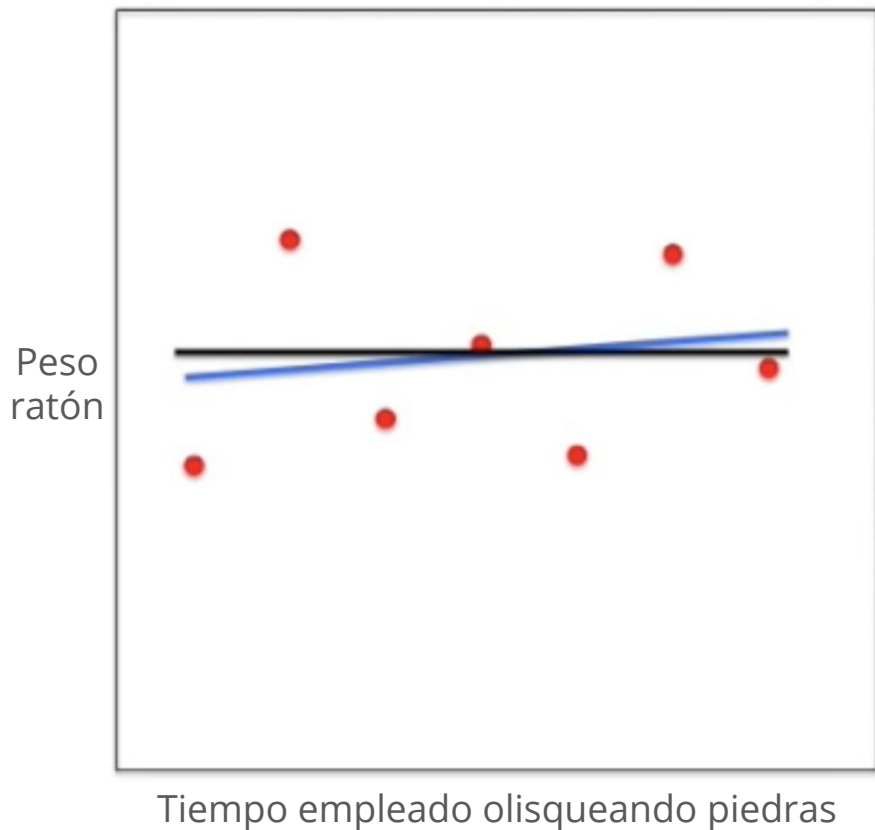


$$R^2 = \frac{\text{Var}(\text{media}) - \text{Var}(\text{línea})}{\text{Var}(\text{media})}$$

Tamaño ratón

R^2 - Significado

- $\text{Var}(\text{media}) = 32$
- $\text{Var}(\text{línea}) = 30$
- $R^2 = 0.06$, hay un **6% menos de variación** alrededor de la línea que alrededor de la media
- **Apenas hay variación de los datos explicada por la relación olisqueo/peso**





Lab 2 - Modelos lineales de Regresión



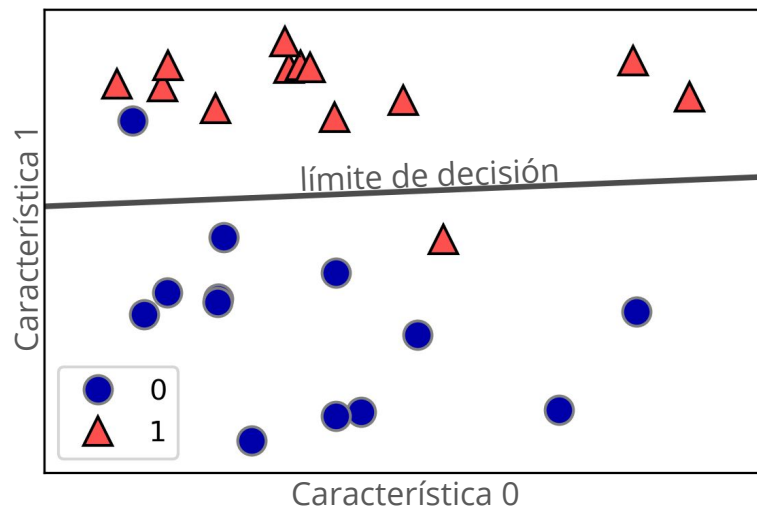
3. Modelos lineales de clasificación

Modelos lineales para clasificación

- Clasificación binaria, o multiclase
- Resolvemos para la ecuación:

$$\hat{y} = w_0 \cdot x_0 + w_1 \cdot x_1 + \dots + w_p \cdot x_p + b$$

- Binaria: $y > 0$, clase 1, $y < 0$, clase -1
- $Y \rightarrow$ límite de decisión (hiperplano)
- Tipos de algoritmos
 - Cómo miden la idoneidad de w y b (L)
 - Qué tipo de regularización utilizan



Modelos lineales de clasificación más comunes

- **Regresión logística**

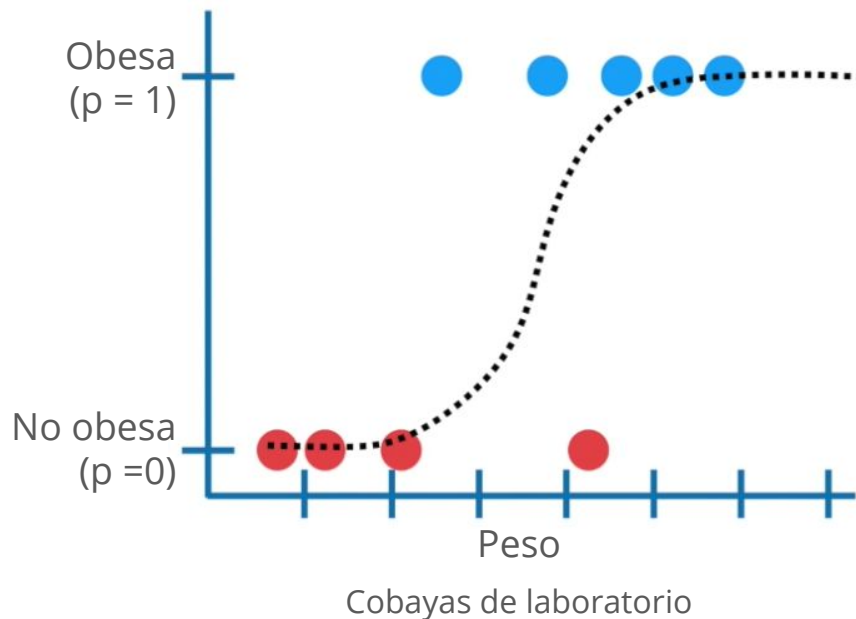
- `linear_model.LogisticRegression`

- Linear Support Vector Machines (Linear SVMs) o **Support Vector Classifiers (SVCs)**

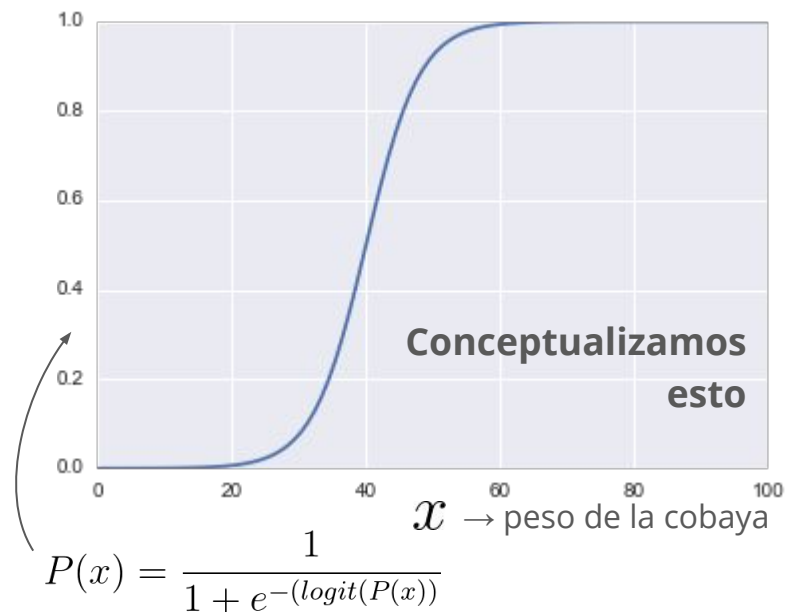
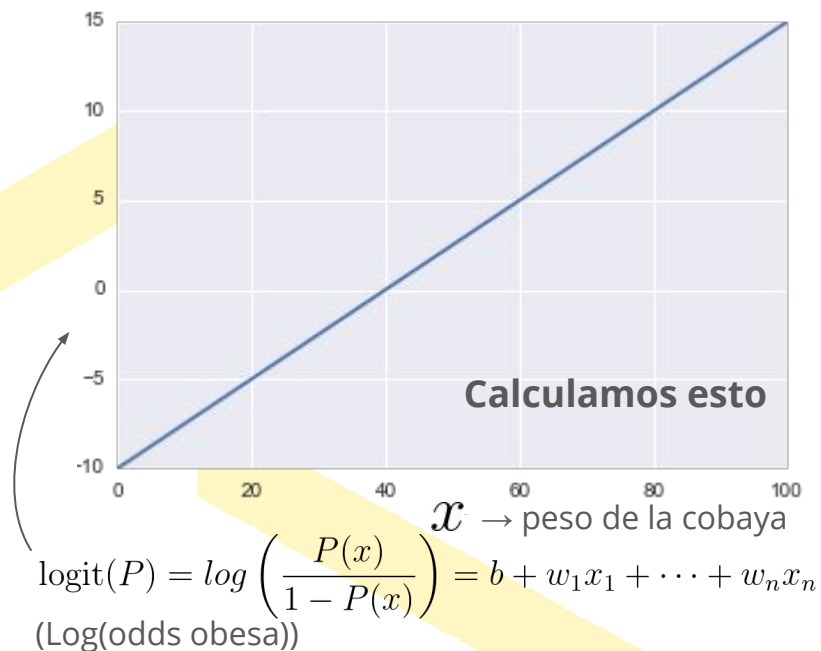
- `svm.LinearSVC`

Regresión Logística

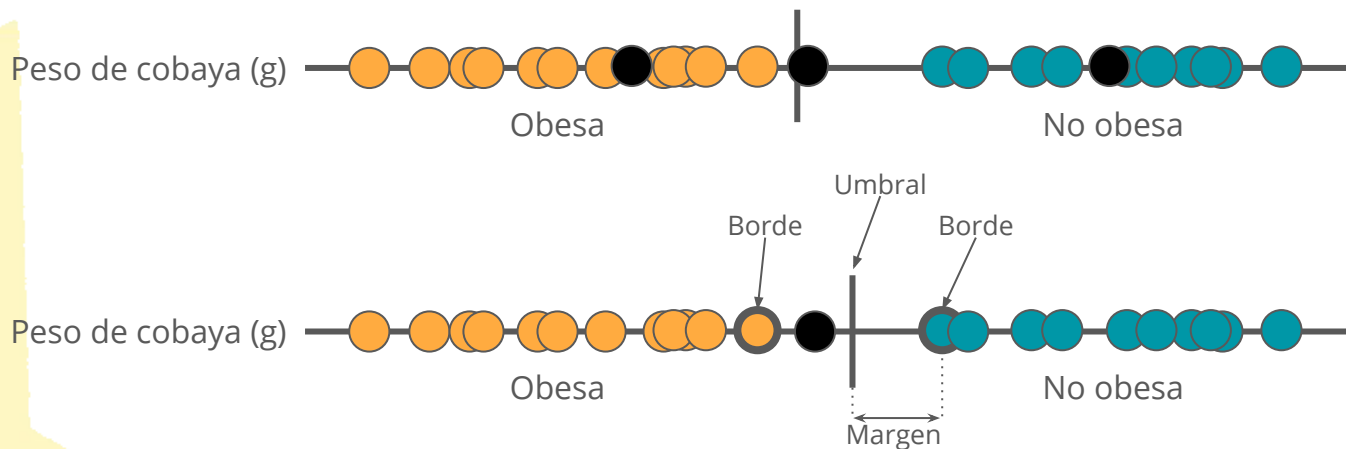
- Es una regresión, produce valores continuos
- Se utiliza para clasificación, con los valores 0 o 1
- En vez de ajustar línea, ajusta una función logística a los datos
- Si $p > 50\% \rightarrow 1$, $p < 50\% \rightarrow 0$
- No puede usar OLS, ni R^2 es válido \rightarrow se usa la **máxima verosimilitud**



Un mínimo de matemáticas

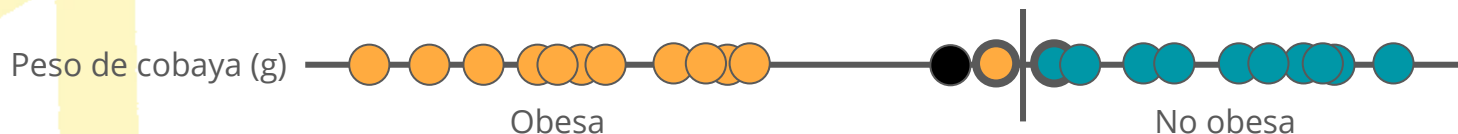


Maximum margin classifier



- La distancia entre el umbral y el borde se llama **margen**
- Cuando la distancia es máxima, estamos usando un **clasificador de margen máximo**

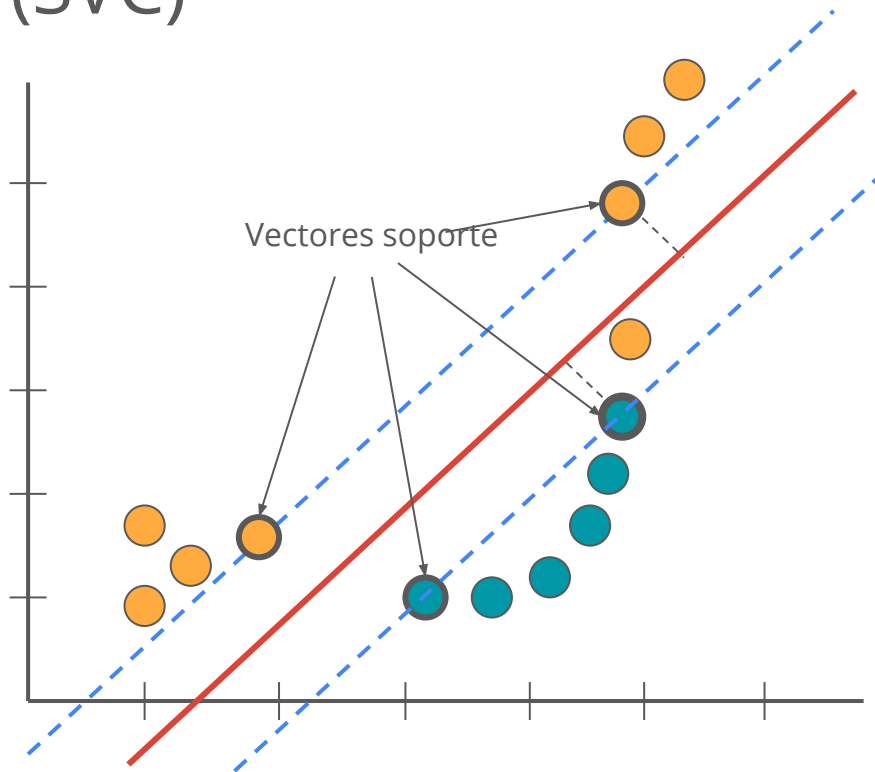
Support Vector Classifier



- Un clasificador de margen máximo es demasiado sensible a valores atípicos
- Tenemos que permitir **clasificaciones erróneas** → *soft margin*

Support Vector Classifier (SVC)

- Usamos validación cruzada para obtener el mejor soft margin → **Support Vector Classifier (SVC)**
- La frontera es un **hiperplano** (línea → hiperplano dimensión 1)





Lab 3 - Modelos Lineales de Clasificación

Conclusiones - Modelos Lineales

- Parámetro principal: alpha (regresión) o C (LinearSVC, LogisticRegression)
 - Optimizar parámetro - se buscan en escala logarítmica
- Elegir regularización L1 o L2

Ventajas - Modelos Lineales

- Entrenamiento y predicción rápidos
- Escalan bien con tamaño de dataset, tolerancia a datos dispersos
 - Datasets muy grandes → solver = 'sag' en LogisticRegression y Ridge
- Fácil entender cómo se predice
- Funcionan bien con ratio características/muestras alto

Inconvenientes - Modelos Lineales

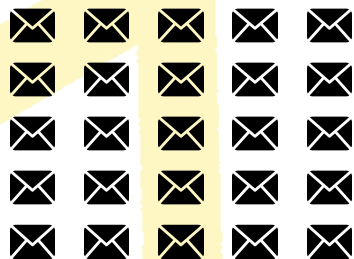
- Fácil entender cómo se predice **pero**
 - No siempre está claro por qué los coeficientes son como son (Datasets con características con alta correlación)
- En espacios con bajas dimensiones, no logran la mejor generalización



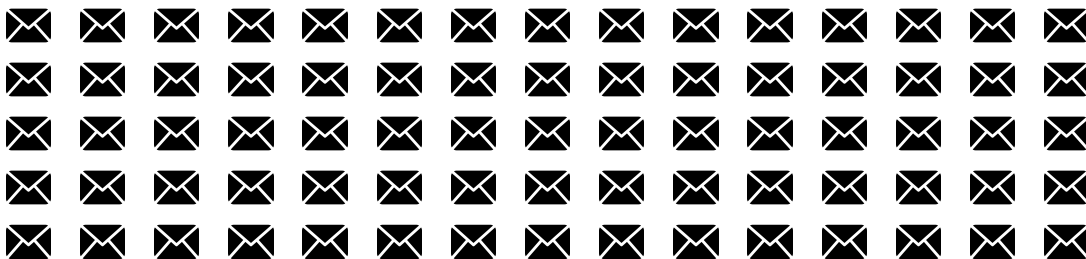
4. Clasificador Naive Bayes

Spam Detector

Spam



No spam



Spam Detector

	Spam (S)		No spam (H)	
Total	25		75	
Buy (B)	20	4/5	5	1/5
Cheap (C)	15	3/5	10	2/15
Buy & Cheap (B ∩ C)	12	12/25	2/3	2/225

Naive, pero mejor que asumir cero o incrementar los datos

$$P(S|B \cap C) = \frac{12}{12 + 2/3} = \frac{36}{38} = 94.737\% \quad \leftarrow \text{Prob. de que un correo que contiene Buy y Cheap sea spam}$$

Spam Detector

$$P(S|\mathbf{B}) = \frac{P(\mathbf{B}|S)}{P(\mathbf{B}|S)P(S) + P(\mathbf{B}|H)P(H)} \longleftarrow \text{Teorema de Bayes de evaluación de hipótesis basada en evidencia}$$

$$P(\mathbf{B} \cap \mathbf{C}) = P(\mathbf{B})P(\mathbf{C}) \longleftarrow \text{Asunción naive de ortogonalidad de características}$$

$$P(S|\mathbf{B} \cap \mathbf{C}) = \frac{P(\mathbf{B}|S)P(\mathbf{C}|S)P(S)}{P(\mathbf{B}|S)P(\mathbf{C}|S)P(S) + P(\mathbf{B}|H)P(\mathbf{C}|H)P(H)}$$

In a nutshell

- Similares a los modelos lineales
- Aprenden parámetros analizando las características de manera individual, sin correlaciones o interdependencias
- Tres implementaciones:
 - GaussianNB: datos continuos, altas dimensiones
 - BernoulliNB: datos binarios
 - MultinomialNB: datos de conteo (enteros, frecuencias de aparición de palabras en una frase....)

Lab 4 - Naive Bayes Classifier

Conclusiones

- MultinomialNB y BernoulliNB tienen ambos un único parámetro a ajustar, **alpha** (controla complejidad del modelo)
 - Se añaden *alpha* más muestra virtuales con valores positivos para todas las características
 - Alpha $\uparrow \rightarrow \uparrow$ suavizado $\rightarrow \downarrow$ complejidad.
 - Optimiz. de alpha no crítico para rendimiento, pero mejora precisión

Ventajas

- Parecidas a las de los modelos lineales
 - Entrenamiento y predicción rápidos
 - Proceso de entrenamiento comprensible
- Robustos a los parámetros
- Buenos modelos de base, usados en datasets muy grandes

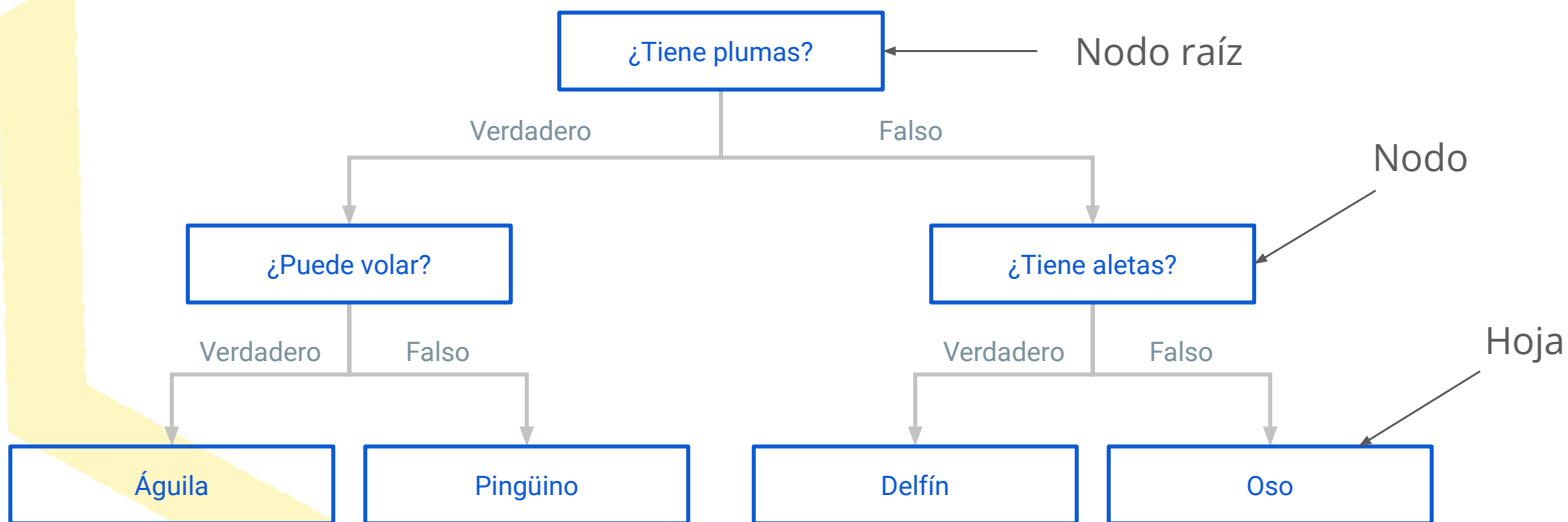
Inconvenientes

- Aunque el modelo se parece a una predicción lineal, los coeficientes no tienen el mismo significado

A large, stylized yellow geometric shape, resembling a thick, open letter 'K' or a series of connected line segments, is positioned on the left side of the slide.

5. Decision Trees

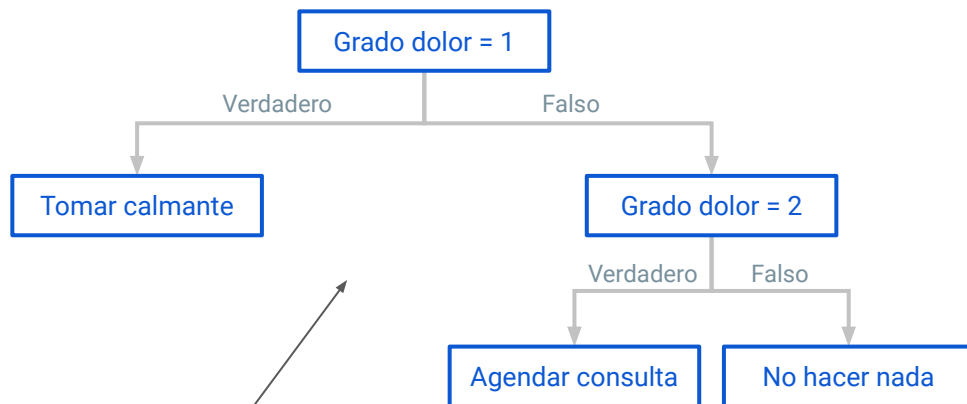
Ejemplo de árbol de decisión binario



Otros tipos de clasificaciones



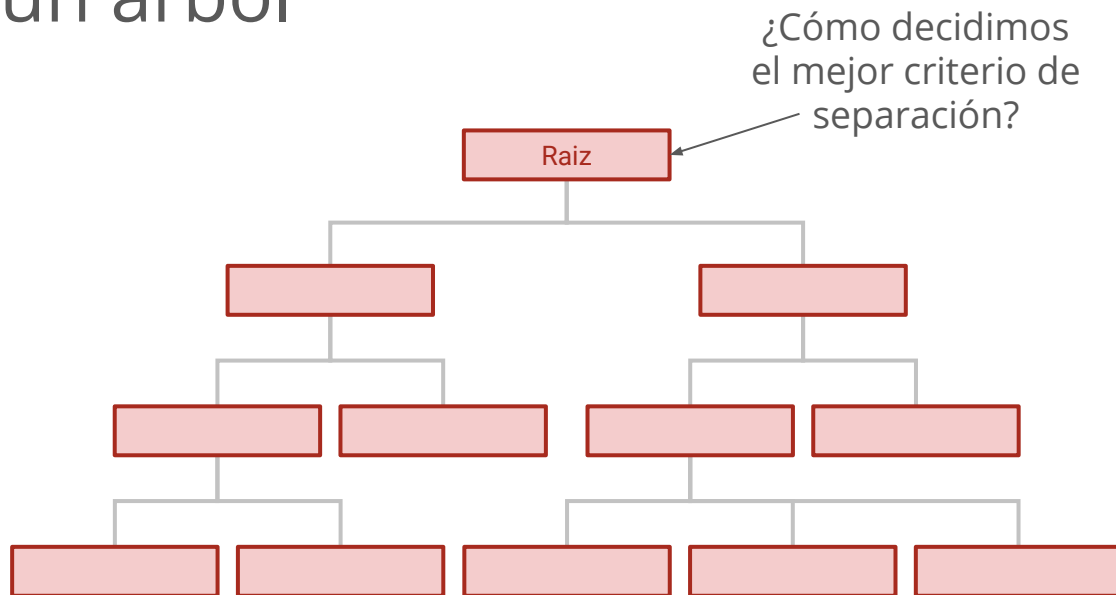
Datos numéricos
con clasificación
categórica



Datos rango
numérico con
clasificación
categórica

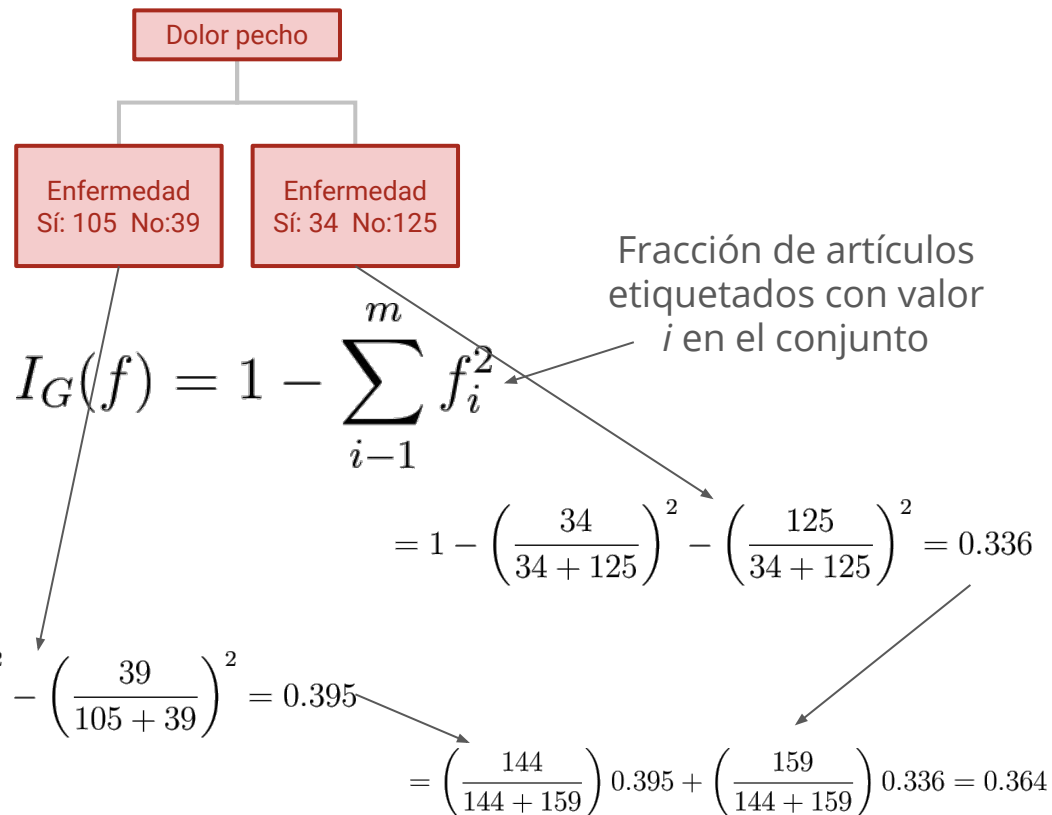
Cómo se construye un árbol

Dolor pecho	Buena circulación	Bloqueo arterial	Enfermedad coronaria
No	No	No	No
Sí	Sí	Sí	Sí
Sí	Sí	No	No
Sí	No	???	Sí
...



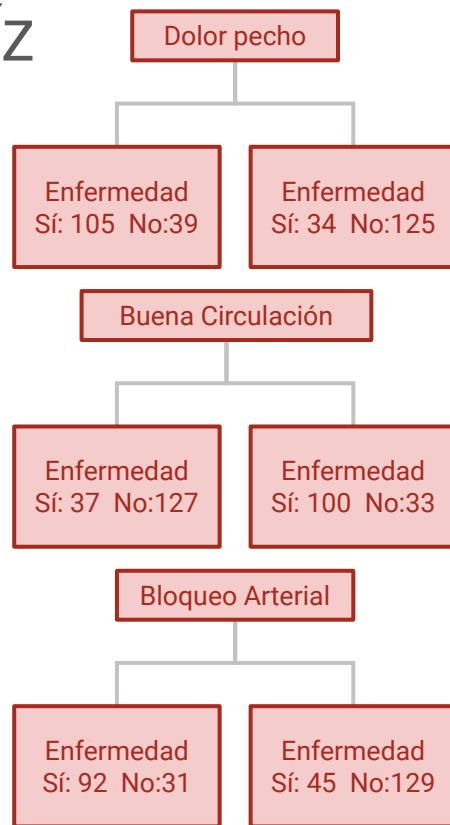
Impureza de Gini

Dolor pecho	Buena circulación	Bloqueo arterial	Enfermedad coronaria
No	No	No	No
Sí	Sí	Sí	Sí
Sí	Sí	No	No
Sí	No	???	Sí
...



Selección del nodo raíz

Dolor pecho	Buena circulación	Bloqueo arterial	Enfermedad coronaria
No	No	No	No
Sí	Sí	Sí	Sí
Sí	Sí	No	No
Sí	No	???	Sí
...



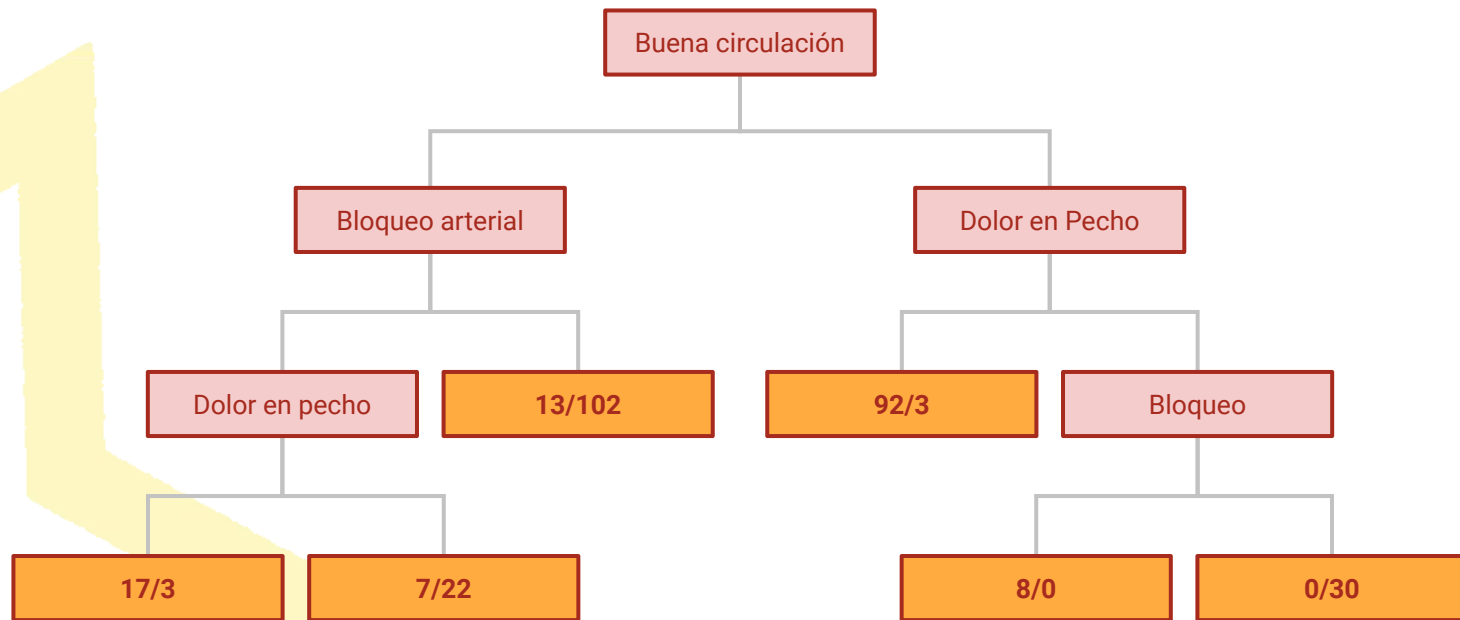
0.364

Ésta es la configuración más "pura"

0.360

0.381

Se aplica iterativamente y..



Lab 5 - Decision Trees

Conclusiones

- Control de complejidad → *pre-pruning*
 - `max_depth, max_leaf_nodes, min_samples_leaf`

Ventajas

- El modelo es fácilmente visualizable y comprensible para cualquiera
- Es invariante frente al escalado de los datos
 - Características procesadas de manera independiente
 - No hacen falta estrategias de pre-procesado como normalización o estandarización de características

Inconvenientes

- Tendencia al sobreajuste
- Mal rendimiento de generalización → **Ensemble learning**

A large, stylized yellow geometric shape, resembling a thick arrow or a series of connected line segments, pointing towards the right. It is positioned on the left side of the slide, partially overlapping the title text.

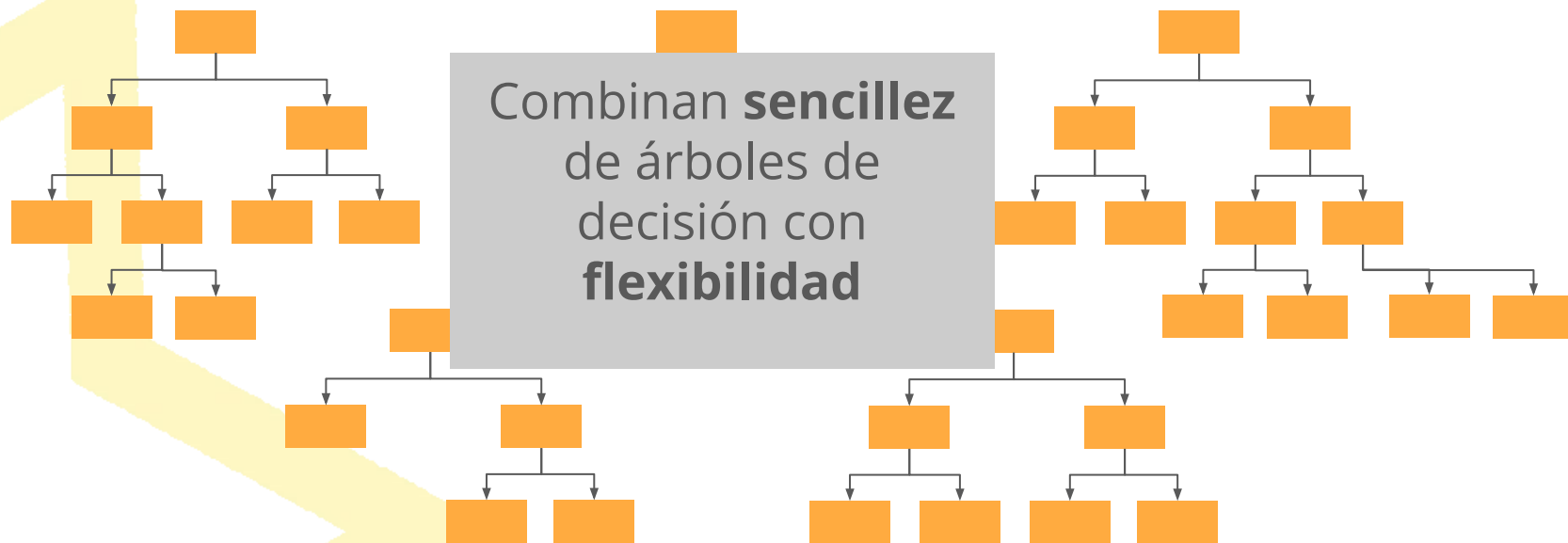
6. Ensembles of Decision Trees

Ensembles de árboles de decisión

- Combinar métodos ML para crear mejores modelos
- 2 tipos muy frecuentes con base en árboles de decisión
 - Random Forests
 - Gradient boosting machines

Random Forests

- Problema fundamental → **imprecisión** (inflexibilidad)



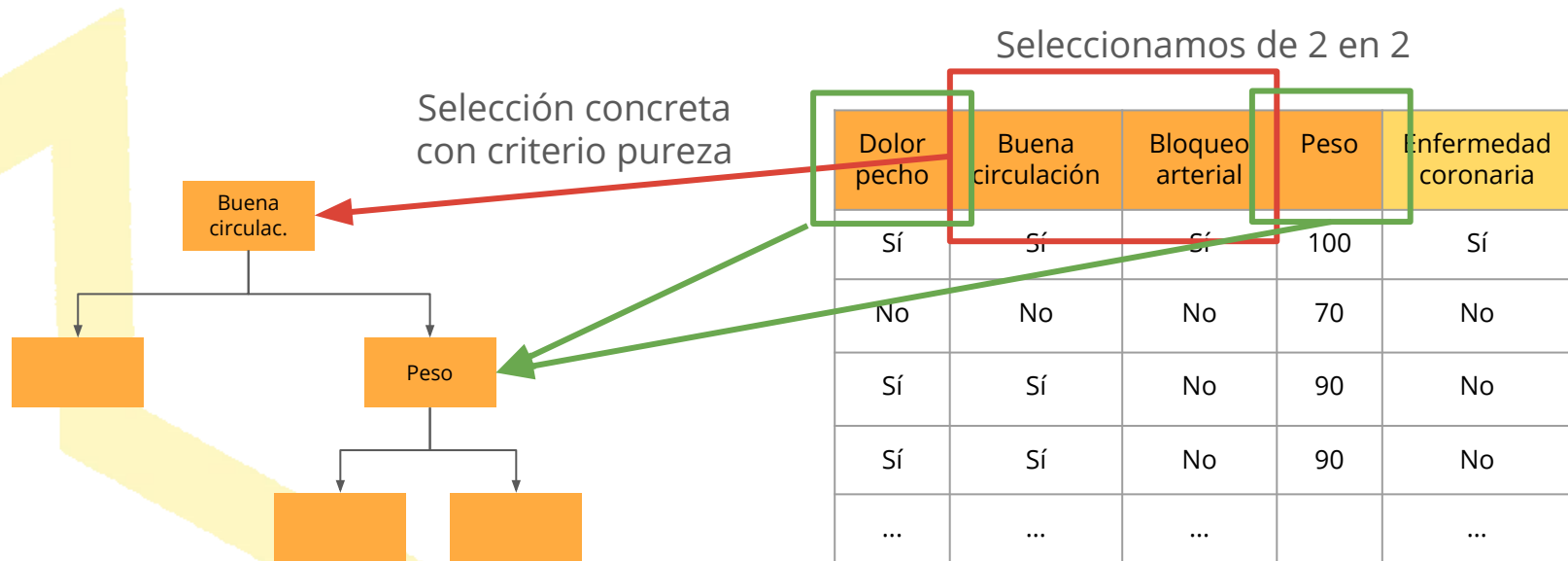
1. Crear un dataset *bootstrapped* (remuestreo)

Dolor pecho	Buena circulación	Bloqueo arterial	Peso	Enfermedad coronaria
No	No	No	70	No
Sí	Sí	Sí	100	Sí
Sí	Sí	No	90	No
Sí	No	???	75	Sí
...

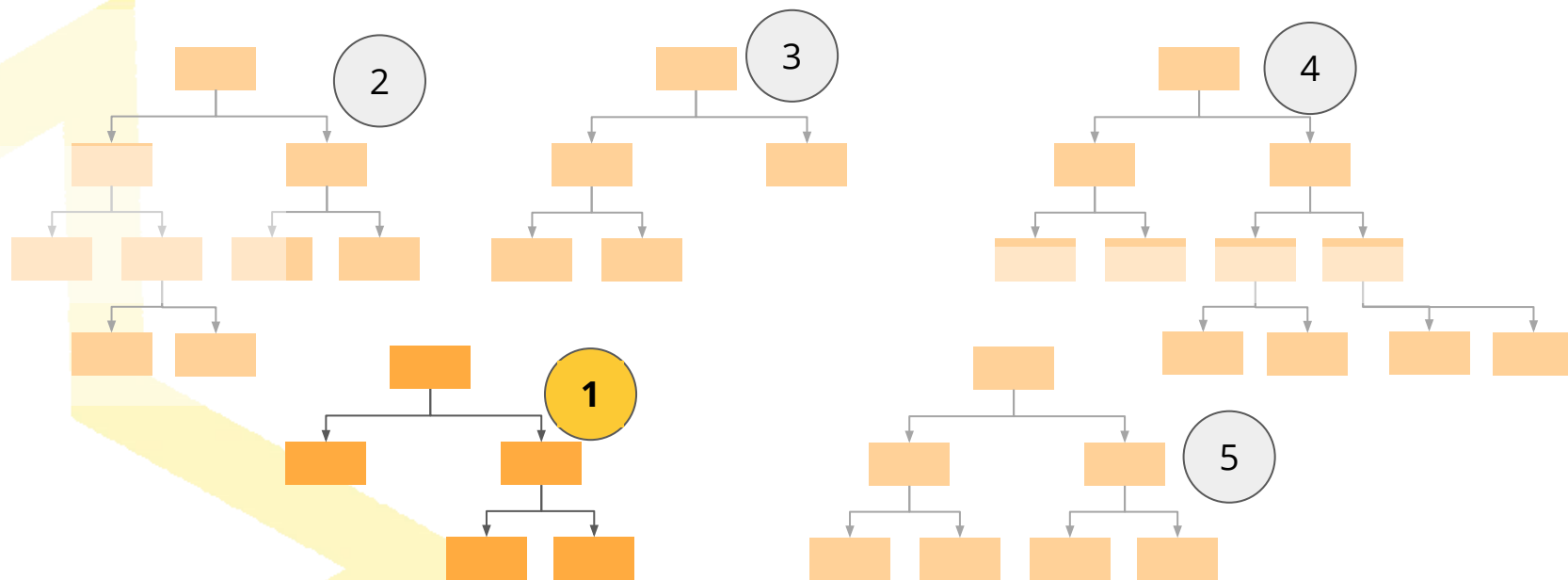
Dolor pecho	Buena circulación	Bloqueo arterial	Peso	Enfermedad coronaria
Sí	Sí	Sí	100	Sí
No	No	No	70	No
Sí	Sí	No	90	No
Sí	Sí	No	90	No
...

Selección aleatoria de
filas (con repetición)
para cada árbol

2. Construir nodos con features aleatorias



3. Iterar, cientos de veces



4. Ejecución de una predicción (**Bagging**)

- Introducimos muestra en cada árbol, y anotamos la predicción
- Contamos el resultado y elegimos la predicción máxima

Enfermedad coronaria	
Sí	No
14	2

Grado de exactitud de un Random Forest

- Elegimos el out-of-bag dataset
 - Muestras no incluidas en la creación inicial del árbol
- Ejecutamos la predicción del random forest para este dataset, contabilizando si ha sido correcta o no.
- Tenemos una **medida de la exactitud** del random forest por el **porcentaje de acierto** en el out-of-bag dataset → **Out-of-bag Error**

Optimización de la creación de un Random Forest

- Seguimos este proceso iterativo
 - Construir random forest
 - Calcular error out-of-bag
 - Cambiar número de variables
 - Repeat until out-of-bag error es mínimo
- Típicamente, se empieza con el cuadrado del número de variables
 - Se prueba varias veces por encima y debajo de ese número

En Scikit-Learn

- RandomForestRegressor y RandomForestClassifier
- Número de árboles a generar → parámetro `n_estimators`
- Límite de features a seleccionar → parámetro `max_features`
 - `max_features = n_features?`
 - `n_features` ↑?
 - `n_features` ↓?

Lab 6a - Random forests

Conclusiones

- Unos de los métodos más usados para regresión y clasificación
- Potentes, funcionan sin mucha optimización paramétrica
- No requieren de escalado de los datos
- Ajuste de:
 - `n_estimators` → mejor cuanto más grande
 - `max_features` → empezar con valor por defecto
 - Clasificación: $(n_features)^2$
 - Regresión: $\log_2(n_features)$
 - `max_depth` → pre pruning
 - `max_leaf_nodes` → para reducir consumo recursos

Ventajas

- Comparten las de los árboles de decisión, sin sus inconvenientes.

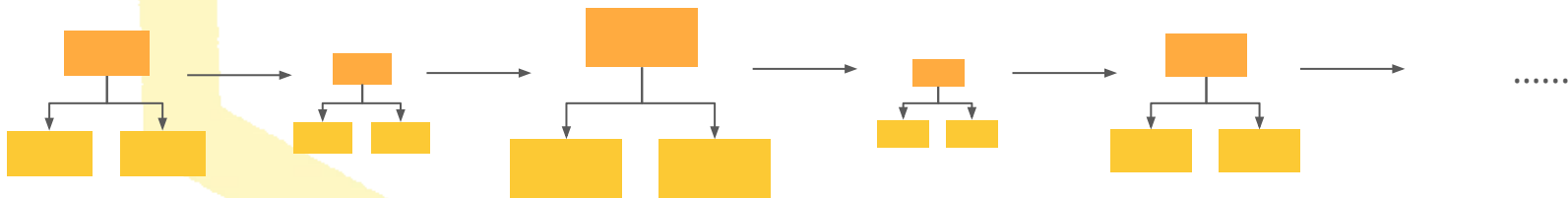


Inconvenientes

- No suministran una visión tan compacta del proceso de decisión como los árboles de decisión
- La construcción de un gran bosque consume recursos (parámetro `n_jobs` para paralelizar), son más lentos de entrenar o predecir que modelos lineales.
- No funcionan bien en datos de muy alta dimensión o dispersión (e.g. texto)

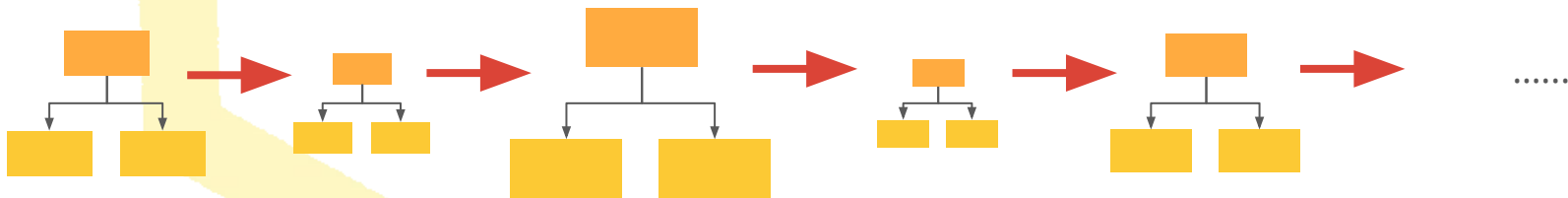
Boosting Machines

- Construimos árboles de manera secuencial con *weak learners*
- **Cada árbol intenta corregir los errores del previo**
- No hay aleatoriedad, hay *pre pruning* fuerte (*stumps*)




Gradient Boosting Machines

- Corrección de errores del árbol previo
 - **AdaBoost:** asignamos mayor peso a los datapoints que queremos corregir en el siguiente árbol
 - **Gradient Boosting:** uso de gradientes en una función de pérdida o coste ($y = ax + b + e$)



Gradient Boosting Machines


- Aparecen con frecuencia como ganadores en competiciones de ML
- Más sensibles a parametrización que los random forests, más precisión
- Parámetros importantes
 - Pre pruning
 - Número de árboles (`n_estimators`)
 - `learning_rate` → dureza de ajuste de errores



Lab 6b - Gradient Boosting Machines

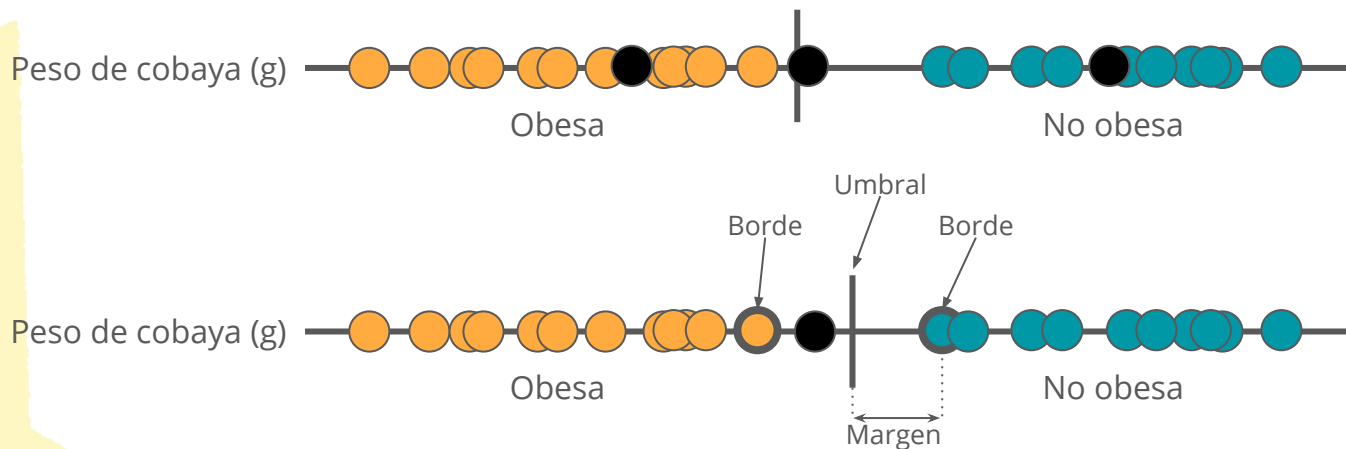
Conclusiones

- Unos de los métodos más usados y potentes
- Requieren optimización fina y tiempo de entrenamiento
- Funcionan bien sin escalado y con tipos distintos de características
- No funcionan muy bien con datasets dispersos en altas dimensiones
- Parámetros, interconectados:
 - `n_estimators`
 - `learning_rate`
 - `max_depth`

A large, stylized yellow geometric shape, resembling a thick arrow or a series of connected line segments, pointing towards the right. It is positioned on the left side of the slide, partially overlapping the title text.

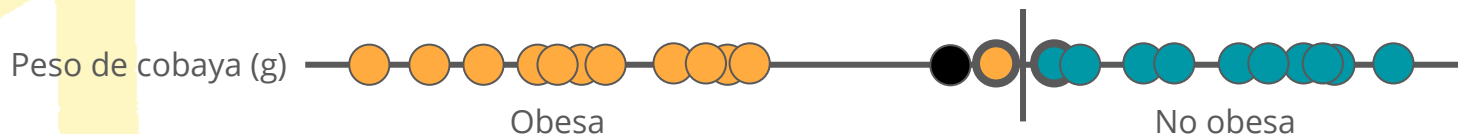
7. (Kernelized) Support Vector Machines

Maximum margin classifier



- La distancia entre el umbral y el borde se llama **margen**
- Cuando la distancia es máxima, estamos usando un **clasificador de margen máximo**

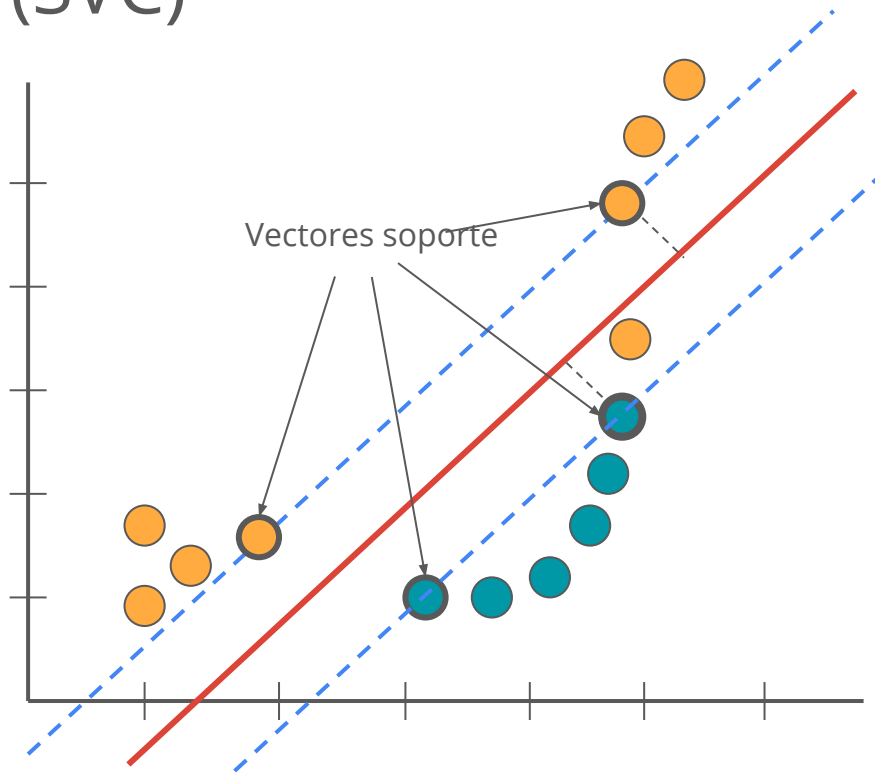
Support Vector Classifier



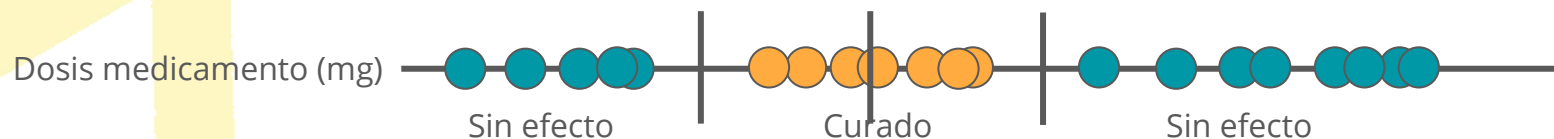
- Un clasificador de margen máximo es demasiado sensible a valores atípicos
- Tenemos que permitir **clasificaciones erróneas** → *soft margin*

Support Vector Classifier (SVC)

- Usamos validación cruzada para obtener el mejor soft margin → **Support Vector Classifier (SVC)**
- La frontera es un **hiperplano** (línea → hiperplano dimensión 1)



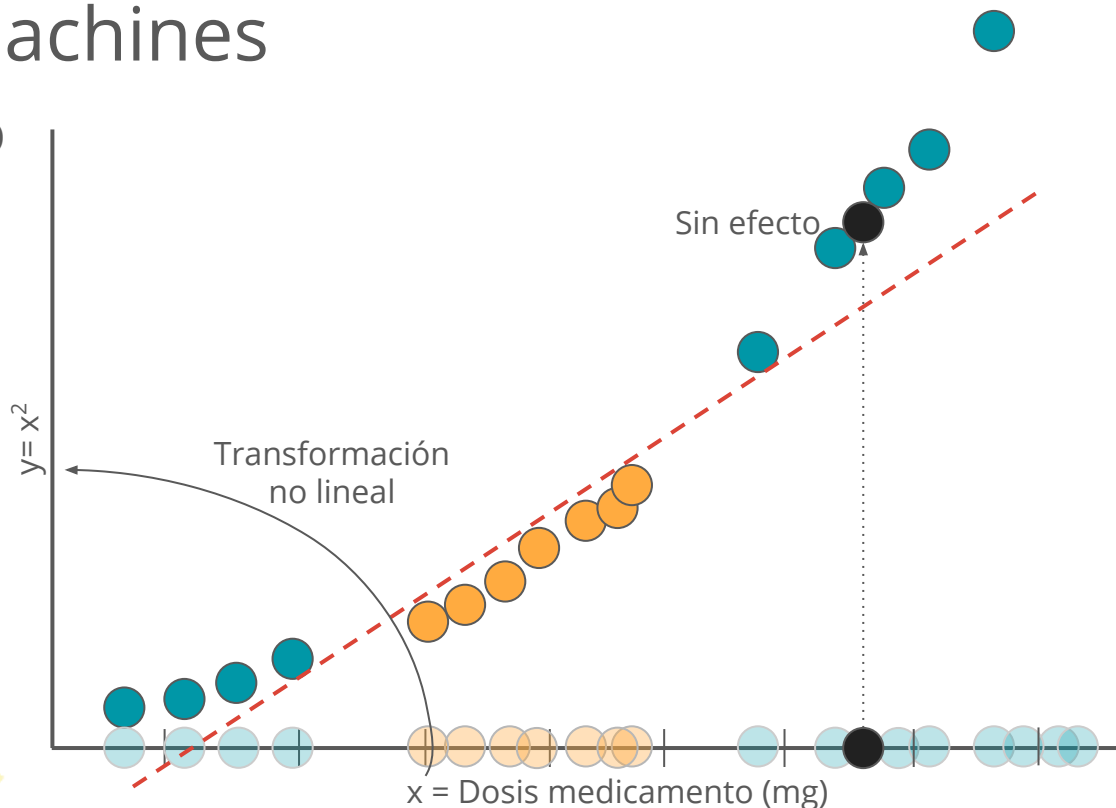
Support Vector Classifier?



- SCV no funciona bien en este tipo de datos

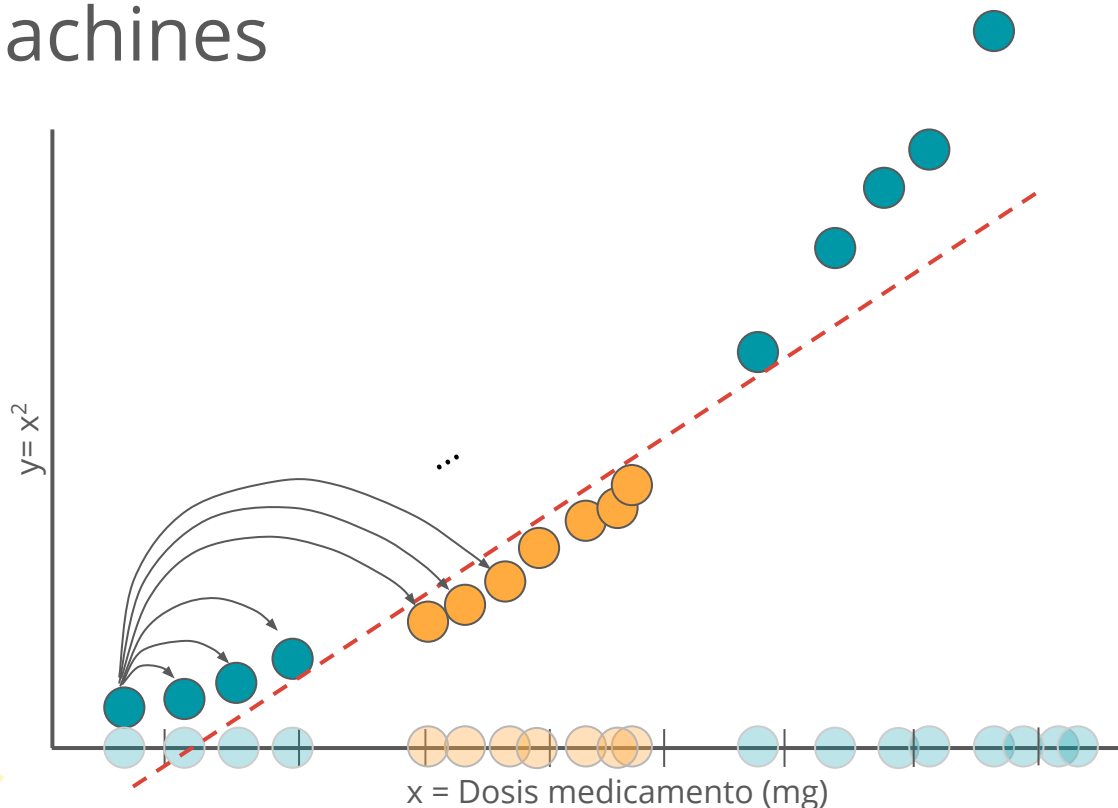
Support Vector Machines

- Los datos son ahora 2D
- Podemos usar un SVC para delimitar la frontera
- SVMs encuentra SVCs en dimensiones más altas **de manera sistemática**



Support Vector Machines

- $y=x^2$ es un Kernel polinómico
- Para encontrar la frontera, calculamos relaciones entre cada par de observaciones (**producto**)
- Finalmente, aplicamos **validación cruzada**



Funciones Kernel (y su truco)

- Movemos dataset a un espacio de más dimensiones
- Para encontrar el SVC, lo que nos interesa es el **producto entre puntos** en ese nuevo espacio, no la transformación en sí a ese espacio de más dimensiones
- *"Si un algoritmo se describe en términos de productos en el espacio de entradas, entonces puede trasladarse al espacio de características sustituyendo esos productos por $K(x_i, x_j)$ (a esto se le llama Kernel Trick)"*

Subiendo dimensiones - *The Kernel Trick*



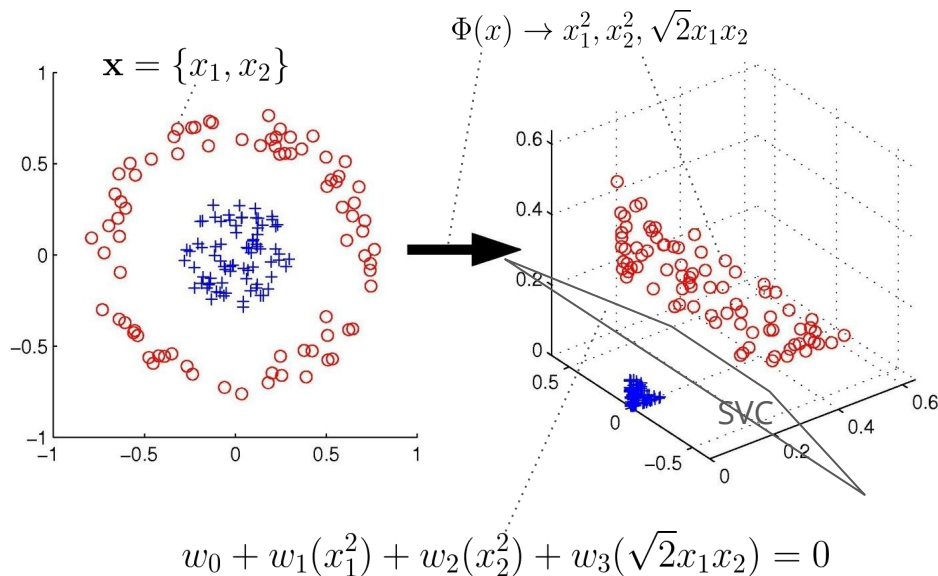
Bender:

1. Transformación todos los puntos $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ con $\Phi(x)$
2. Cálculo de SVC mediante multiplicación en \mathbb{R}^3

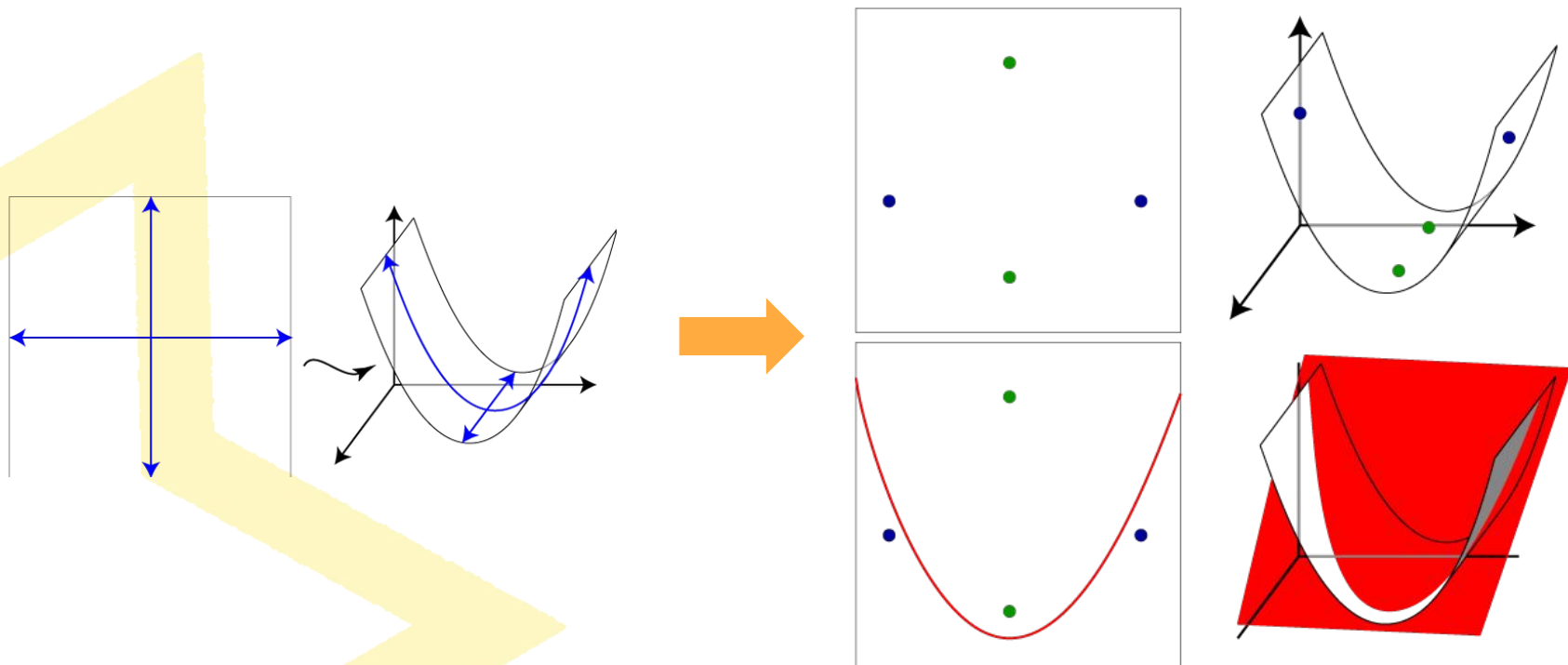


Flexo:

1. Encontrar **una función** (¡kernel!) donde calcular productos sea sencillo
2. Calcular en \mathbb{R}^2 , **sin** $\Phi(x)$



Subiendo dimensiones - Kernel Trick



The Kernel Trick



Bender:



$$\begin{aligned} & \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \\ &= \langle \{x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}\}, \{x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2}\} \rangle \\ &= x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} \end{aligned}$$

- 3 x 2 ops para transformar + 3 ops producto = **9 operaciones**
- Para 5D, **16 operaciones**, más dimensiones, aún más operaciones



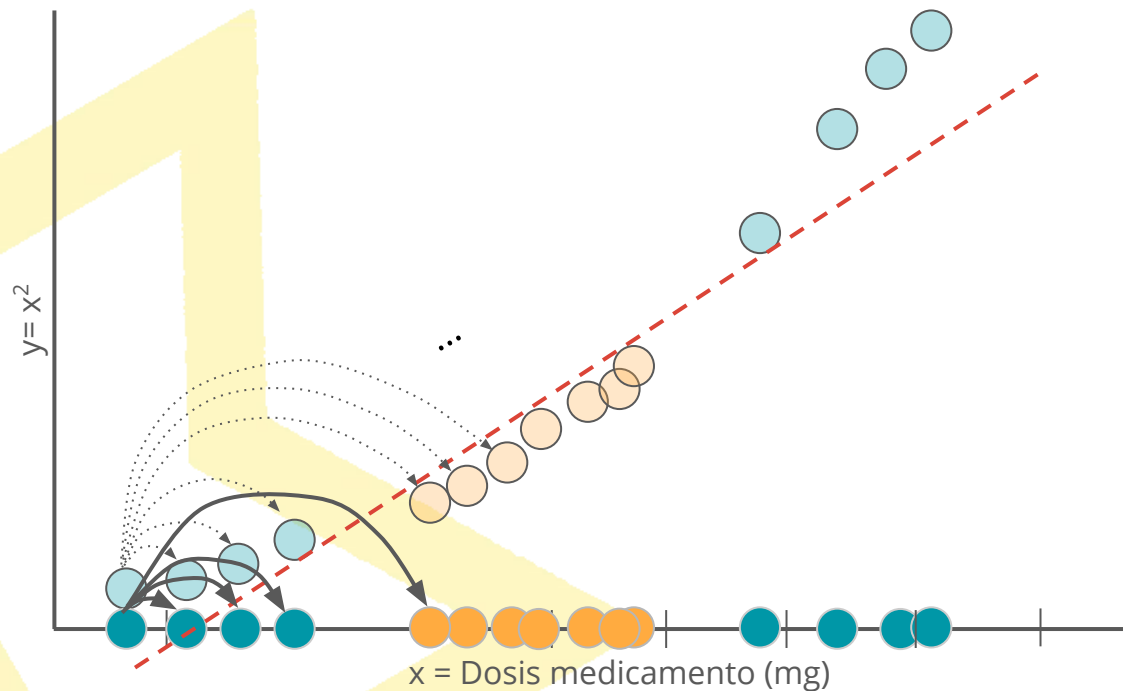
Flexo:

Kernel $K(x_i, x_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^2$
(Kernel polinómico)

$$\begin{aligned} & \langle \mathbf{x}_i, \mathbf{x}_j \rangle^2 \\ &= \langle \{x_{i1}, x_{i2}\}, \{x_{j1}, x_{j2}\} \rangle^2 \\ &= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\ &= x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} \end{aligned}$$

- 2 ops producto + 1 op *transformación* = **3 operaciones**
- 5D, 9D, más Ds.... **¡3 operaciones!**
Kernel $K(x_i, x_j) = (1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^2$

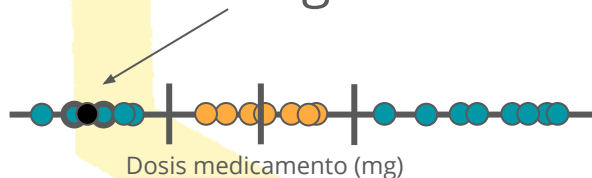
The kernel trick - conceptualmente



*Una función Kernel nos **permite operar en el espacio original de características** sin calcular las coordenadas de los datos en un espacio de mayores dimensiones.*

Radial Basis Function (RBF) Kernel

- Encuentra SVCs en **dimensiones infinitas** (no visualizable)
- En el caso de una observación 1D, RBF se comporta como un k Nearest Neighbors



$$K(x_i, x_j) = e^{\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}$$

$-\frac{1}{2\sigma^2} = \gamma$
Parámetro del kernel

$$= C \left\{ 1 - \underbrace{\frac{\langle x_i, x_j \rangle}{1!}}_{\text{1er Orden}} + \underbrace{\frac{\langle x_i, x_j \rangle^2}{2!}}_{\text{2do Orden}} - \underbrace{\frac{\langle x_i, x_j \rangle^3}{3!}}_{\text{3er Orden}} + \dots \right\}$$

donde, $C = e^{(-\frac{1}{2}\|x_i\|^2)} e^{(-\frac{1}{2}\|x_j\|^2)}$

Sigue siendo una operación en el espacio de entrada (2D en este caso)

¡Infinitas dimensiones!

Lab 7 - Kernelized SVMs



Parámetros importantes

- Regularización C :
- Elección del kernel, ej. RBF
- Parámetros específicos del kernel elegido, ej. *gamma* para RBF

Conclusiones

- Modelos potentes, funcionan bien en multitud de datasets
- Permiten fronteras de decisión complejas, incluso con pocas características
- Funcionan bien con baja y alta dimensión de características
 - No escalan muy bien con el número de muestras
 - ~1000 muestras → OK, 100k ó más → elevado consumo de recursos
- Requieren de mucho preprocesado de datos y optimización de parámetros
 - Preferencia por modelos basados en árbol

Conclusiones

- Difíciles de inspeccionar
- Merece la pena considerarlos, especialmente si:
 - Características en mismas unidades (ej.: intensidades de pixel)
 - Características en similar escala



8. Recap

Generalización

- Difíciles de inspeccionar
- Merece la pena considerarlos, especialmente si:
 - Características en mismas unidades (ej.: intensidades de pixel)
 - Características en similar escala

Modelos

- **kNN**: datasets pequeños, bueno como baseline, fácil de explicar
- **Modelos lineales**: primer modelo habitual, datasets muy grandes, \uparrow dim características
- **Naive Bayes**: sólo clasificación. Más rápido que lineales, bueno para \uparrow datos y \uparrow dim características. Menos precisos que lineales.
- **Decision Trees**: Muy rápidos, no requieren escalado datos. Visualizables y explicables

Modelos

- **Random Forests:** mejor rendim. que un sólo árbol, robustos y potentes. No requieren escalado datos. Malos con \uparrow dim características y \uparrow dispersión datos.
- **Gradient Boosted Trees:** Algo más precisos que random forests. Entrenamiento más lento pero predicción más rápida, menos consumo RAM. Más optimización paramétrica que
- **Support Vector Machines:** potentes para tamaños medios de dataset. Sensibles a parámetros, requieren escalado datos.

Tengo un dataset nuevo, ¿y ahora qué?

1. **Empieza con un modelo sencillo** (Lineal, Naive Bayes, kNN)
2. **Prueba a ver hasta dónde llegas con él**
3. Aprovechalo para **conocer más tus datos**
4. **Considera un algoritmo más complejo** (Random Forests, Gradient Boosted Decision Trees, SVMs o... redes neuronales)
5. Siempre, **investiga qué han hecho otros**



¡Gracias!

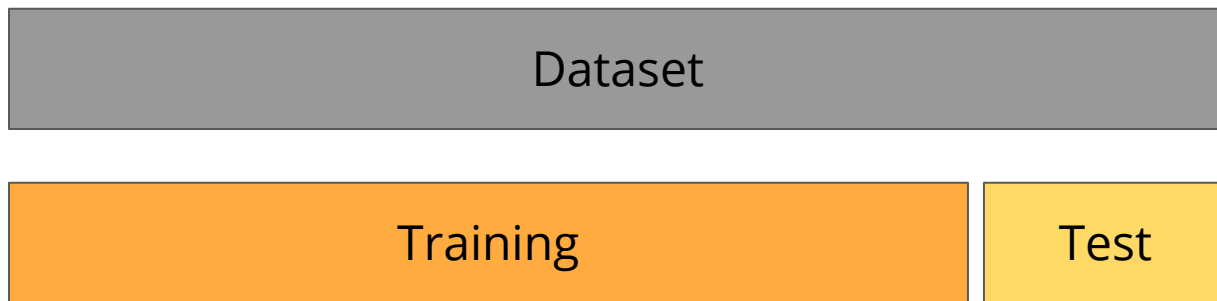


Material Adicional



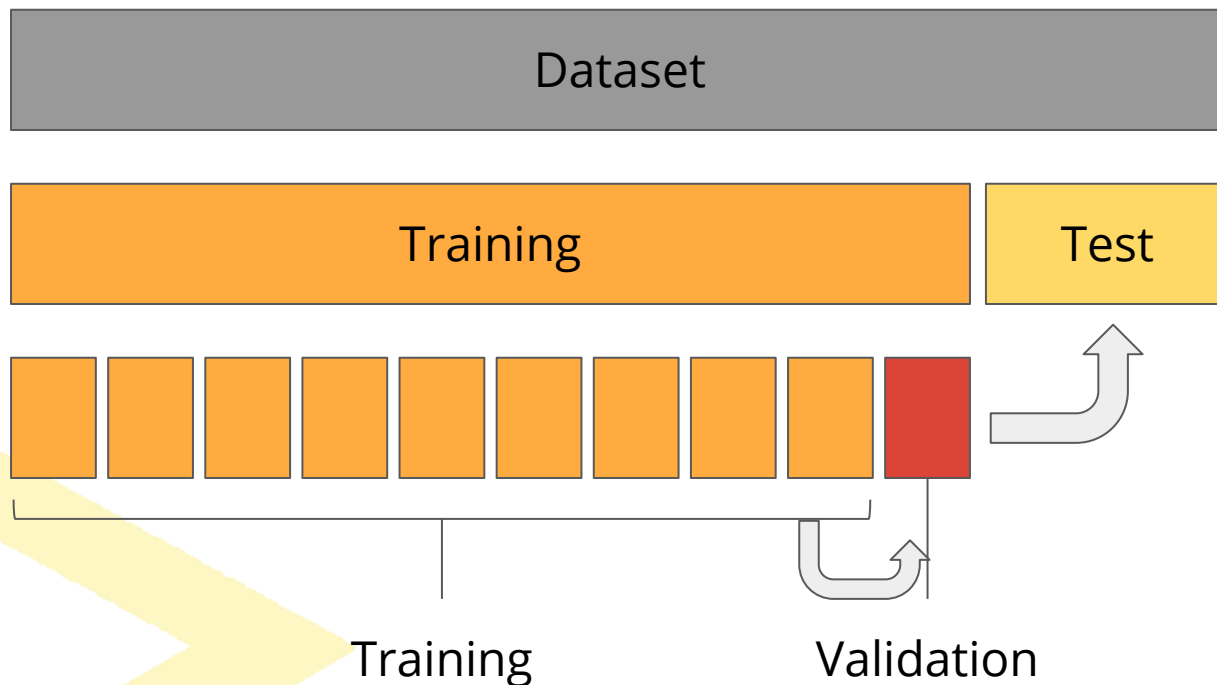
Separación Training/Test y Validación Cruzada

Separación Training - Test





Cuando buscamos optimizamos parámetros del modelo, ¿no corremos el riesgo de optimizar con respecto a test?

Separación Training - Validation - Test



Validación cruzada de k (10) iteraciones

 Nuevo set de training
 Set de validación

