



UNIVERSIDAD DE CHILE

# Minería de Datos

Welcome to the Machine Learning class

---

Valentin Barriere

Universidad de Chile – DCC

CC5205, Otoño 2024

# Aprendizaje supervisado – Modelos

## K-vecinos más cercanos

---

# Outline : K-vecinos más cercanos

## K-vecinos más cercanos

Naive Bayes

Bosque aleatorios

Árboles de decisión

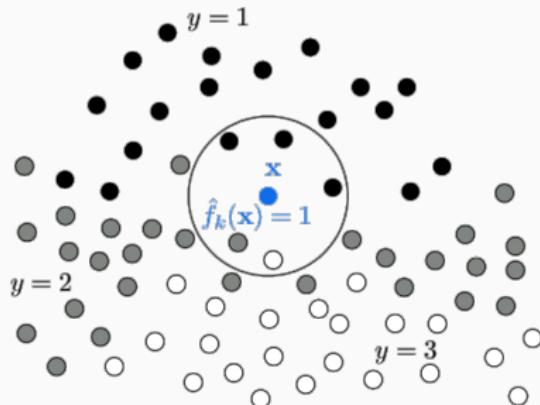
Métodos de conjunto

Bosque aleatorios

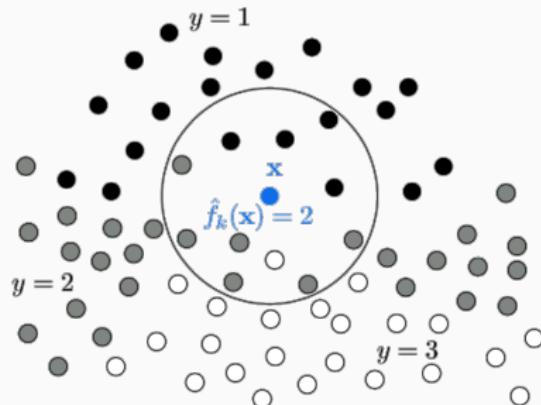
API : Scikit-learn

# K vecinos más cercanos

Miramos los datos más cercanos en el espacio de los descriptores para clasificar nuestro nuevo archivo:

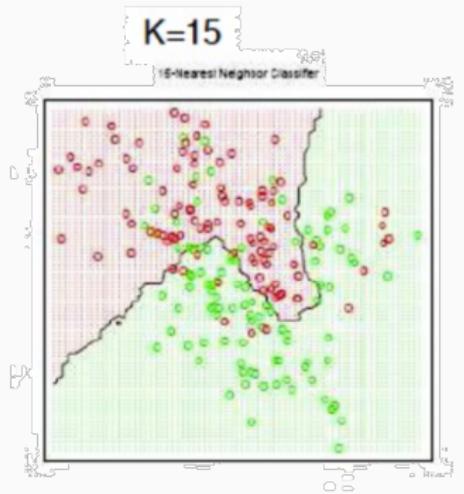
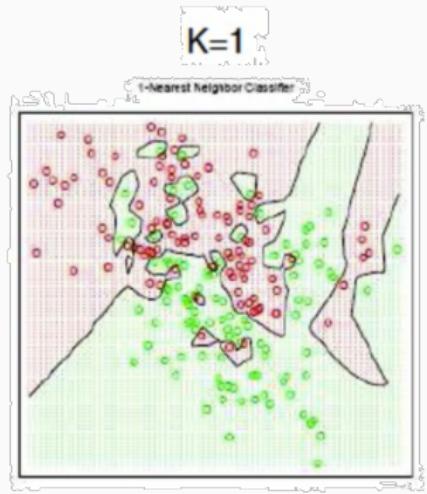


(a)  $k = 5$



(b)  $k = 11$

# K-vecinos más cercanos : Importancia de K



**El número de vecinos es un hiperparámetro del modelo**

- K demasiado pequeño: la función  $f$  es demasiado sensible a los datos: varianza alta
- K demasiado grande: la función  $f$  se vuelve demasiado insensible a los datos: sesgo importante

# K-vecinos más cercanos : Clasificación

K-VMC (K-Nearest neighbors: K-NN)

Caso 2 clases:

$$h_{KNN}(\mathbf{X}) = \arg \max_{y \in \{-1,1\}} \frac{N_y^K(\mathbf{X})}{K}$$

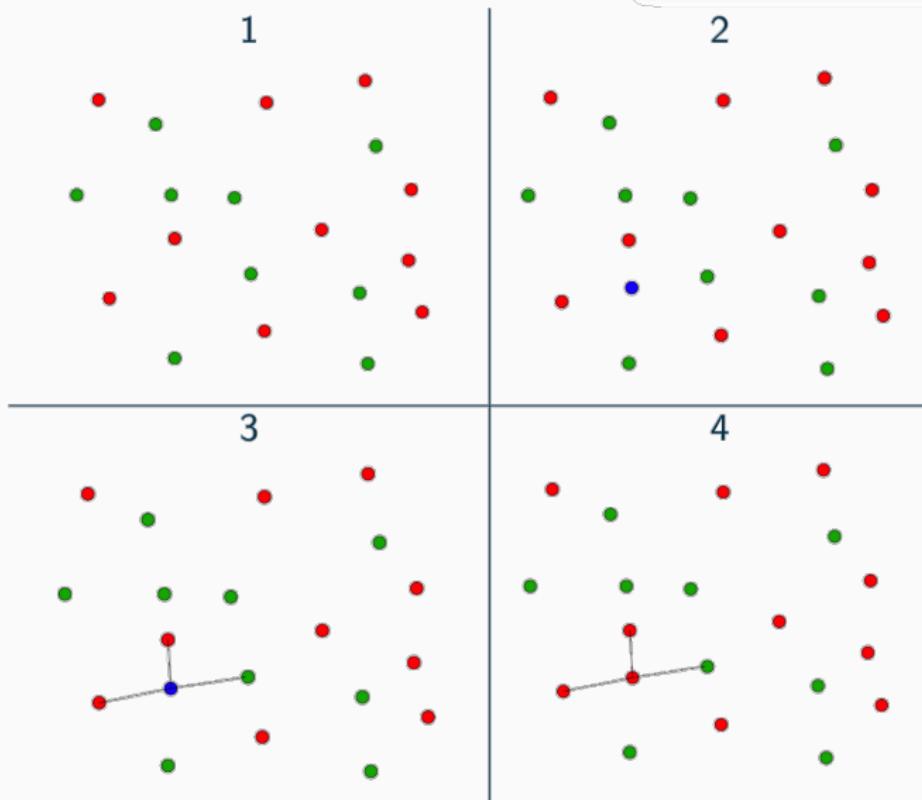
donde:

- Sea  $K$  un número entero estrictamente positivo.
- Sea  $d$  una métrica definida en  $x \times x$
- $S = \{(\mathbf{X}_i, y_i), i = 1, \dots, n\}$
- Para un  $\mathbf{X}$  dado, definir  $\sigma$  la permutación de índices en  $\{1, \dots, n\}$  tal que:

$$d(\mathbf{X}, \mathbf{X}_{\sigma(1)}) \leq d(\mathbf{X}, \mathbf{X}_{\sigma(2)}) \dots \leq d(\mathbf{X}, \mathbf{X}_{\sigma(n)})$$

- $S^K_{\mathbf{X}} = \{\mathbf{X}_{\sigma(1)}, \dots, \mathbf{X}_{\sigma(K)}\}$ :  $K$  primeros vecinos de  $\mathbf{X}$
- $N_y^K(\mathbf{X}) = |\{\mathbf{X}_i \in S^K_{\mathbf{X}}, y_i = y\}|$

# K-vecinos más cercanos : Algoritmo



# K-vecinos más cercanos : Distancias

Métrica para calcular las distancias:

$$d(x, y) = \sqrt[p]{\sum |x - y|^p}$$

## Las métricas más comunes

- Manhattan:  $p = 1$
- Euclidiana:  $p = 2$
- Minkowski: otro  $p$
- Chebyshev:  $p = \infty$
- Mahalanobis:  $d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$

(ver <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.DistanceMetric.html>)

Los **atributos necesitan ser normalizados** porque se utiliza distancia entre las dimensiones, y alguno atributos podrían dominar los otros.

# Transformaciones de los datos: sobre la distribución

## Standardization

El estimador puede comportarse mal si las características individuales no se parecen más o menos a los datos estándar distribuidos normalmente: Gaussiana con **media cero y varianza unitaria**:  $\frac{x - \mu_x}{\sigma_x}$

## Scaling features to a range

Generalmente entre 0 y 1. La motivación para utilizar esta escala incluye la robustez a desviaciones estándar muy pequeñas de las características y la preservación de entradas cero en datos dispersos:  $\frac{x - \min_x}{\max_x - \min_x}$

## Normalización

La normalización es el proceso de **ajustar las muestras individuales para que tengan norma unitaria**.

Mas info [aca](#)

## K-NN: Otros

Los clasificadores K-NN son **lazy learners**:

- No construyen modelos explícitos, es más flexible ya que no necesita comprometerse con un modelo global a priori.
- Al contrario de otros **eager learners** como los árboles de decisión o clasificadores basados en reglas.
- Es independiente del nro. de clases.
- La clasificación es más costosa (memoria y tiempo).

# K-NN: Dimensionalidad

Con alta dimensionalidad, el K-NN está sujeta a la Maldición de la Dimensionalidad:

- **Data Sparsity:** With more features, the data becomes increasingly sparse, making it harder to find similar instances.
- **Noise Accumulation:** Noisy features can dominate the distance calculations, leading to poor predictions.
- **Overfitting:** Models may overfit to the noise in the data rather than the underlying patterns.

Consequences of the Curse of Dimensionality:

- **Decreased Accuracy:** As the number of features increases, the KNN algorithm's accuracy may decrease due to the increased difficulty in finding meaningful neighbors.
- **Increased Computational Cost:** Calculating distances in high-dimensional spaces becomes computationally expensive, leading to slower prediction times.
- **Data Requirements:** The algorithm requires a large amount of

# Naive Bayes

---

# Outline : Naive Bayes

K-vecinos más cercanos

**Naive Bayes**

Bosque aleatorios

Árboles de decisión

Métodos de conjunto

Bosque aleatorios

API : Scikit-learn

## Naive Bayes

The conditional probability of event A given event B is:

$$P(A|B) = \frac{P(A, B)}{P(B)}$$

The Bayes' theorem for events A and B is:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

# Naive Bayes

## Naive Bayes

- Algoritmo clásico de modelización de  $\mathbb{P}\{\mathbf{X}|Y\}$
- Hipótesis fuerte de independencia de las características:  
$$\mathbb{P}\{\mathbf{X}|Y\} = \prod_{i=1}^d \mathbb{P}\{X^{(i)}|Y\}$$
- Ejemplo con  $c$  clases,  $o = (w_1, \dots, w_N)$  observaciones de  $N$  palabras:

$$\hat{c} = \operatorname{argmax}_c P(c|o) = \operatorname{argmax}_c \frac{P(o|c)P(c)}{P(o)}$$

- Hipótesis de constancia:  $P(o)$  es constante:

$$\hat{c} = \operatorname{argmax}_c \frac{P(o|c)P(c)}{P(o)} = \operatorname{argmax}_c P(o|c)P(c)$$

- Hipótesis ingenua: Las variables son independientes una de otra:

$$P(o|c) = P(w_1, \dots, w_N|c) = \prod_{i=1..N} P(w_i|c)$$

$P(w_i|c) = \frac{\text{Count}(w_i \in C)}{\text{Count}(w_i)}$ : se cuentan instancias de los atributos/las clases

El algoritmo utiliza la siguiente fórmula para calcular la probabilidad posterior de una clase  $C$  dado un conjunto de observaciones  $o$ :

$$P(c|o) = \frac{P(o|c) \cdot P(c)}{P(o)}$$

donde:

- $P(c|o)$  es la **probabilidad posterior** de la clase  $c$  dadas las características  $o$
- $P(o|c)$  es la **likelihood** de las características  $o$  dada la clase  $c$ .
- $P(c)$  es la **probabilidad prior** de la clase  $c$ .
- $P(o)$  es la evidencia, que es la probabilidad de las características  $o$

Laplace smoothing es útil para manejar casos donde  $P(w_i|c) = 0$ .

## Naive Bayes: Atributos continuos

Los distintos clasificadores Bayes ingenuos difieren principalmente por las suposiciones que hacen respecto a la distribución de  $P(w_i|c)$ .

Si las características son continuas, es posible utilizar el Bayes ingenuo gaussiano, en el que se supone que el modelo de probabilidad de las características es gaussiano:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Los parámetros  $\sigma_y$  y  $\mu_y$  se estiman utilizando la máxima verosimilitud (i.e., con los valores de los atributos en las clases).

## Naive Bayes: Resumen

- Aunque sea conocido como un clasificador decente, se sabe que es un **mal estimador** de probabilidades.
- Es robusto ante puntos de ruido aislados.
- Robusto a atributos irrelevantes (afectan de igual manera a todas las clases).
- Es particularmente útil en problemas donde las características son categóricas o discretas.
- Ofrece buenos resultados iniciales incluso con pocos datos de entrenamiento
- Puede ser sensible a atributos altamente correlacionados, debido a la suposición de independencia.

## Bosque aleatorios

---

# Outline : Bosque aleatorios

K-vecinos más cercanos

Naive Bayes

**Bosque aleatorios**

Árboles de decisión

Métodos de conjunto

Bosque aleatorios

API : Scikit-learn

# Outline : Árboles de decisión

K-vecinos más cercanos

Naive Bayes

**Bosque aleatorios**

Árboles de decisión

Métodos de conjunto

Bosque aleatorios

API : Scikit-learn

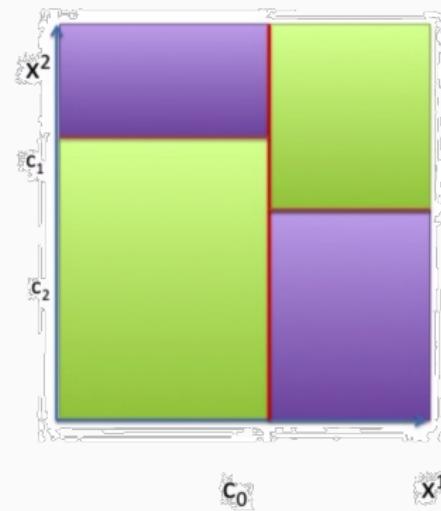
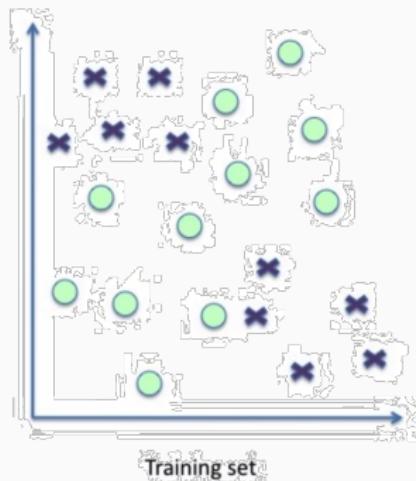
# Árboles de decisión

## 3 Principios

Encontrar umbrales en los diferentes descriptores para poder discriminar las diferentes clases:

**P1** Usando no uno sino varios separadores lineales

**P2** Usando hiperplanos  $x^j = c_k$  para poder mantener una interpretación

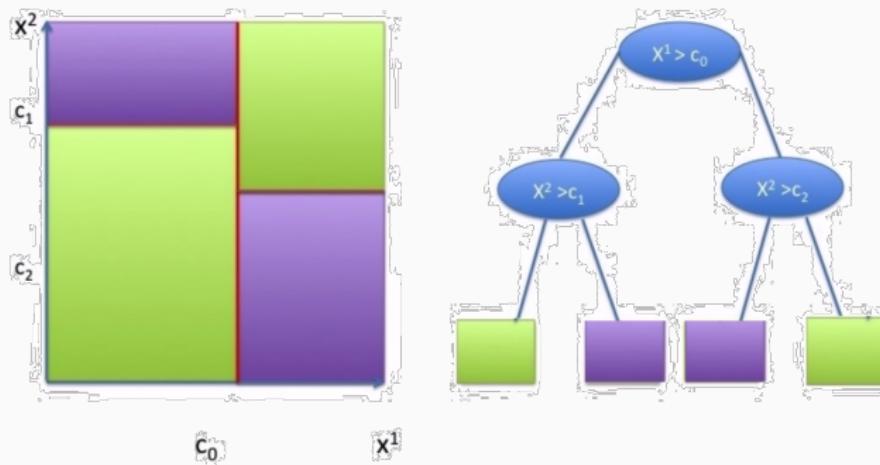


# Árboles de decisión

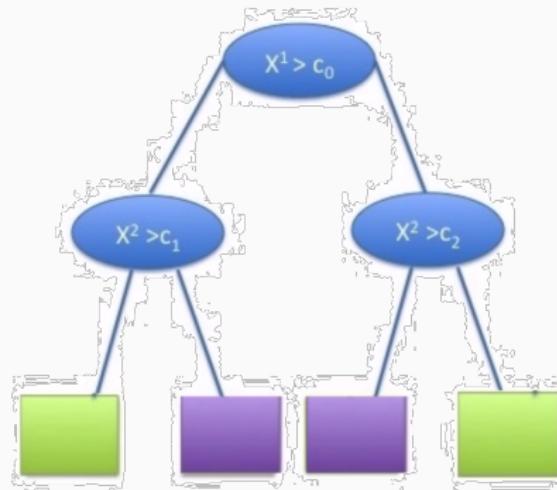
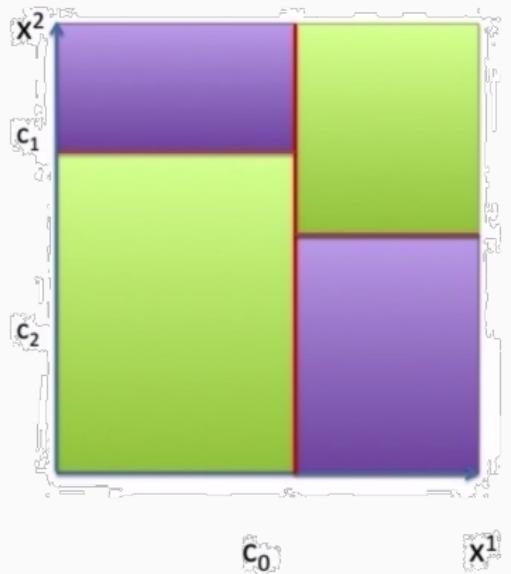
## 3 Principios

Encontrar umbrales en los diferentes descriptores para poder separar el espacio de manera no lineal:

**P3** La función aprendida puede ser representada por una estructura de árbol donde cada nodo está asociado con un hiperplano  $x^j = \theta_j$  y cada hoja a una clase



# Árboles de decisión



Al final de la fase de aprendizaje, se sabe qué características influyen en la decisión final: Si  $x^{k_1} < c_{k_1}$  y  $x^{k_2} > c_{k_2}$  y ..., entonces  $\mathbf{X}$  pertenece a la clase  $k$  (hay  $k$  clases con  $j$  caminos posibles).

## Variables continuas o reales

Dependiendo del tipo de variable  $x^j$ , la separación  $t$  entre 2 ramas se realiza de diferentes maneras:

- Si  $x^j$  es una variable continua:

$$\text{signo}(x^j - c_{k_j})$$

- Si  $x^j$  es una variable categórica con 2 valores  $v_j^1, \dots, v_j^2$ :

$$\mathbb{I}(x^j = v_j^1)$$

## Árboles de decisión: Ejemplo

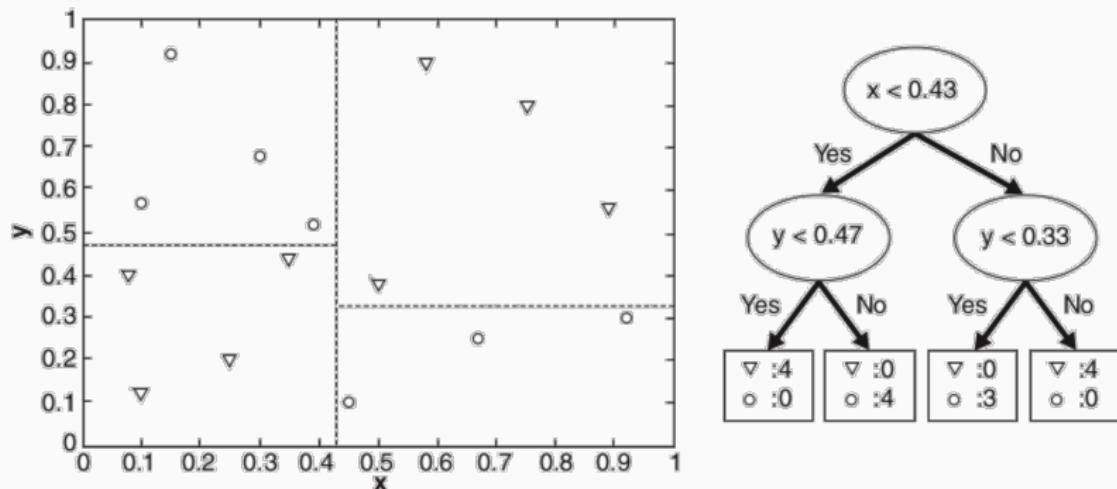


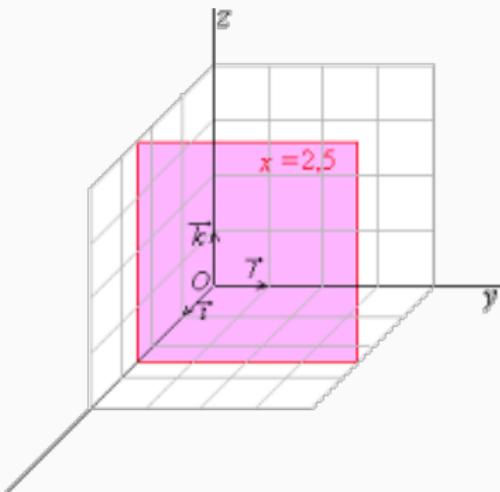
Figure 3.20. Example of a decision tree and its decision boundaries for a two-dimensional data set.

# Partición: Consideremos una nueva clase de funciones I

## Definición

$$\forall \mathbf{X} \in \mathbb{R}^p, h_{arbol}(\mathbf{X}) = \sum_{l=1}^L \alpha_l \mathbb{I}_{\mathcal{C}_l}(\mathbf{X})$$

Donde  $\mathcal{C}_1 \cup \dots \cup \mathcal{C}_L$  es una partición de  $\mathbb{R}^p$  (es decir, cubre todo el espacio) y cada  $\mathcal{C}_l$  está definida por un subconjunto de hiperplanos ortogonales a un vector de la base canónica



## Partición: Consideremos una nueva clase de funciones II

- Sea  $\mathbf{X} \in \mathbb{R}^p$ , tenemos  $I(\mathbf{X}) \in 1, \dots, L$  tal que  $\mathbf{X} \in \mathcal{C}_{I(\mathbf{X})}$  ( $I(\mathbf{X})$  es el índice del subconjunto de la partición en el que se encuentra  $\mathbf{X}$ ).
- La clasificación se realiza por mayoría en  $\mathcal{C}_{I(\mathbf{X})}$  :

$$h_n(\mathbf{X}) = \alpha_{I(\mathbf{X})} = \arg \max_k \sum_{\mathbf{X}^{(i)} \in \mathcal{C}_{I(\mathbf{X})}} \mathbb{I}(y_i = k)$$

### Objetivo

Aprender la partición  $\mathcal{C}_1 \cup \dots \cup \mathcal{C}_L$

## Árboles de decisión : algoritmo

Para un arbol binario:

1. Sea  $\mathcal{S}$  el conjunto de entrenamiento
2. Construir la raíz
3. Encontrar la mejor separación  $t(\mathbf{X})$  de  $\mathcal{S}$  tal que la pérdida  $L(t, \mathcal{S})$  sea mínima
4. Separar el conjunto de entrenamiento en dos subconjuntos  $\mathcal{S}_g$  y  $\mathcal{S}_d$  usando la separación
5. Construir los nodos derechos e izquierdos
6. Medir el criterio de parada a la derecha, si se cumple, el nodo se convierte en hoja, de lo contrario, volver al 3 con  $\mathcal{S}_d$
7. De igual manera a la izquierda con  $\mathcal{S}_g$

## Árboles de decisión : pérdida local y criterio de impureza I

Sea  $\mathcal{S}$  el conjunto de entrenamiento y  $t_{j,\tau}$  una separación :

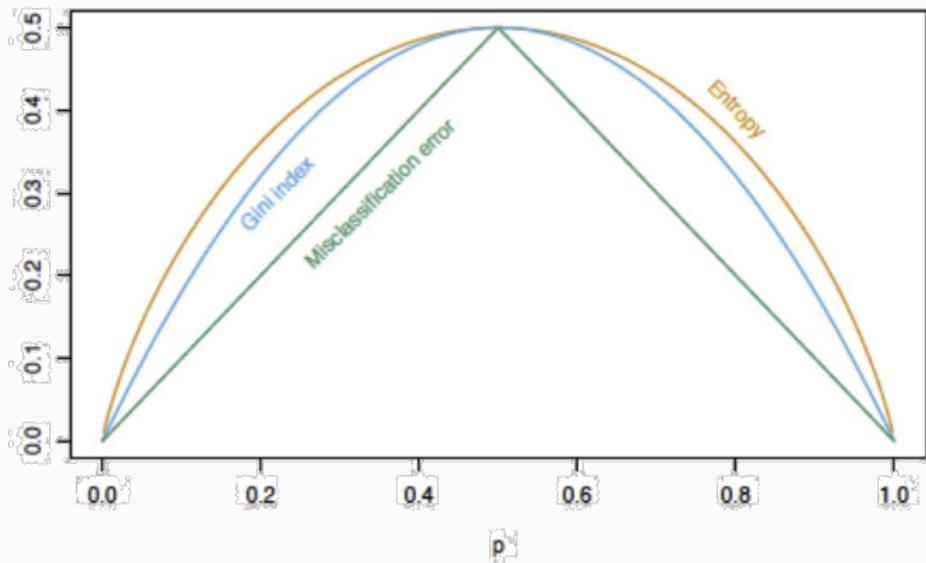
- $\mathcal{D}(\mathcal{S}, j, \tau) = (\mathbf{X}, Y) \in \mathcal{S}, t_{j,\tau} > 0$
- $\mathcal{I}(\mathcal{S}, j, \tau) = (\mathbf{X}, Y) \in \mathcal{S}, t_{j,\tau} \leq 0$

Los  $\tau_i$  pueden ser elegidos con un espaciado constante o mediante un histograma. Entre los parámetros  $(j, \tau) \in 1, \dots, d \times \tau_1, \dots, \tau_C$ , buscamos aquellos que minimizan:

$$L(t_{j,\tau}, \mathcal{S}) = \frac{|\mathcal{D}(\mathcal{S}, j, \tau)|}{n} H(\mathcal{D}(\mathcal{S}, j, \tau)) + \frac{|\mathcal{I}(\mathcal{S}, j, \tau)|}{n} H(\mathcal{I}(\mathcal{S}, j, \tau))$$

$H$  mide la impureza de un subconjunto  $\mathcal{S}$ . Queremos una distribución lo menos homogénea posible en términos de clases (queremos tener una clase de un lado y otra clase del otro).

# Árboles de decisión : pérdida local y criterio de impureza II



**Podemos usar:**

- Entropía cruzada:  $H(S) = -\sum_{c=1}^C p_c(S) \log(p_c(S))$
- Índice de Gini:  $H(S) = \sum_{c=1}^C p_c(S)(1 - p_c(S))$
- Clase mayoritaria:  $H(S) = 1 - p_{clase_{maj}}(S)$

La entropía es máxima por  $p = 0.5$ , lo que sería el más incierto en binario.

# Ejemplo con la Gana de Información

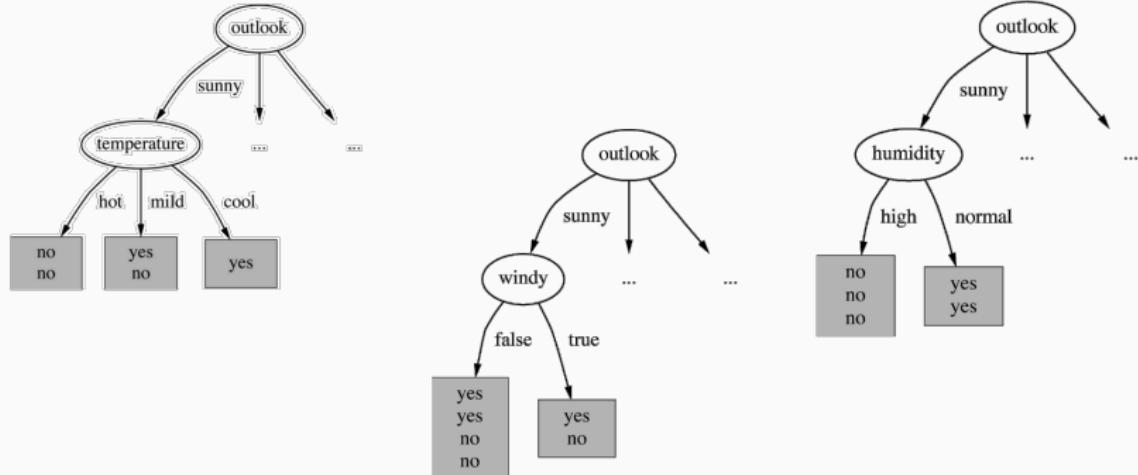
## Information Gain

Usando  $H$ , se puede calcular la gana de información en bits. Lo que es simplemente la diferencia de entropía antes y después de un nodo.

La optimización ayuda a crear el árbol más pequeño posible.  $H$  da un valor en bits que significa el *information gain*:

- **Heurística:** escoge el atributo que produce nodos lo más “puros” posible.
- **Criterio popular de pureza:** *information gain*; crece cuando crece la pureza promedio de los subconjuntos.
- **Estrategia:** escoger el atributo que maximiza el valor de la gana de información.

# Ejemplo con la Gana de Información



$$\text{gain}(\text{Temperature}) = 0.571 \text{ bits}$$

$$\text{gain}(\text{Humidity}) = 0.971 \text{ bits}$$

$$\text{gain}(\text{Windy}) = 0.020 \text{ bits}$$

Humidity permite de **separa bien las clases**, lo que significa **poco entropía**, y un **grande ganancia** de información.

## Árboles de decisión : Criterio de parada

Detener la construcción si se alcanza uno de los siguientes valores:

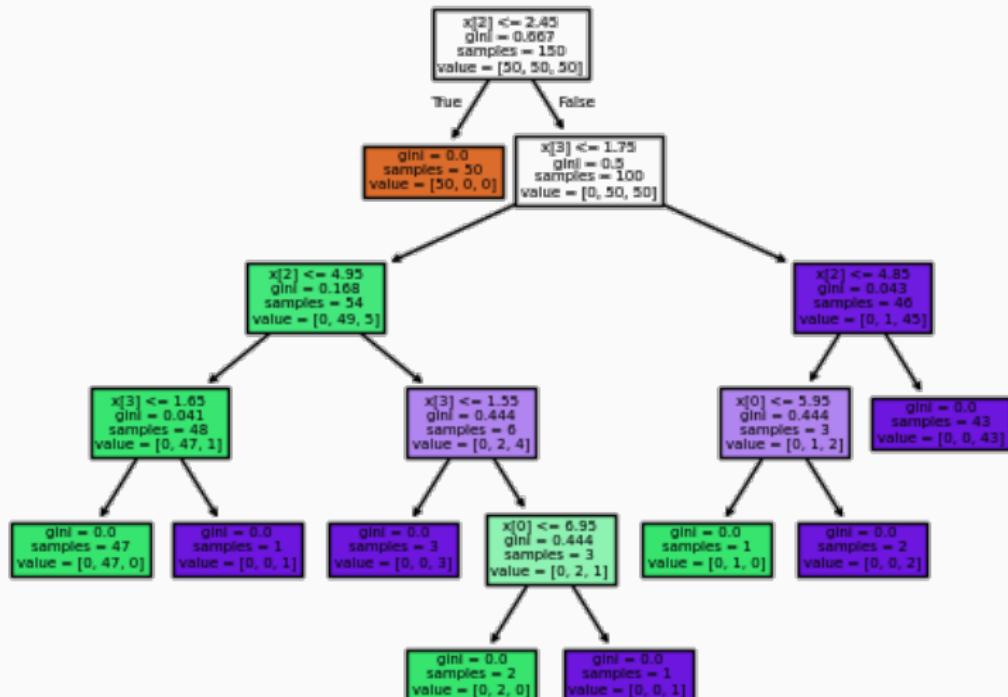
- Profundidad máxima
- Número máximo de hojas
- Número mínimo de datos en una hoja

**Nota:** Si el número mínimo de datos en una hoja es 1, es posible que el árbol crezca hasta una clasificación perfecta del conjunto de entrenamiento, lo que equivaldría a un sobreajuste.

# Árboles de decisión : Scikit

Se puede visualizar los arboles con `sklearn.tree.plot_tree`

Decision tree trained on all the iris features



## Árboles de decisión : selección de modelos

Para seleccionar los hiperparámetros de un árbol de decisión:

- **Validación cruzada** en: Profundidad máxima, número máximo de hojas
- **Poda** : En un conjunto de validación, cortamos las ramas del árbol que no mejoran el rendimiento

# Outline : Métodos de conjunto

K-vecinos más cercanos

Naive Bayes

**Bosque aleatorios**

Árboles de decisión

Métodos de conjunto

Bosque aleatorios

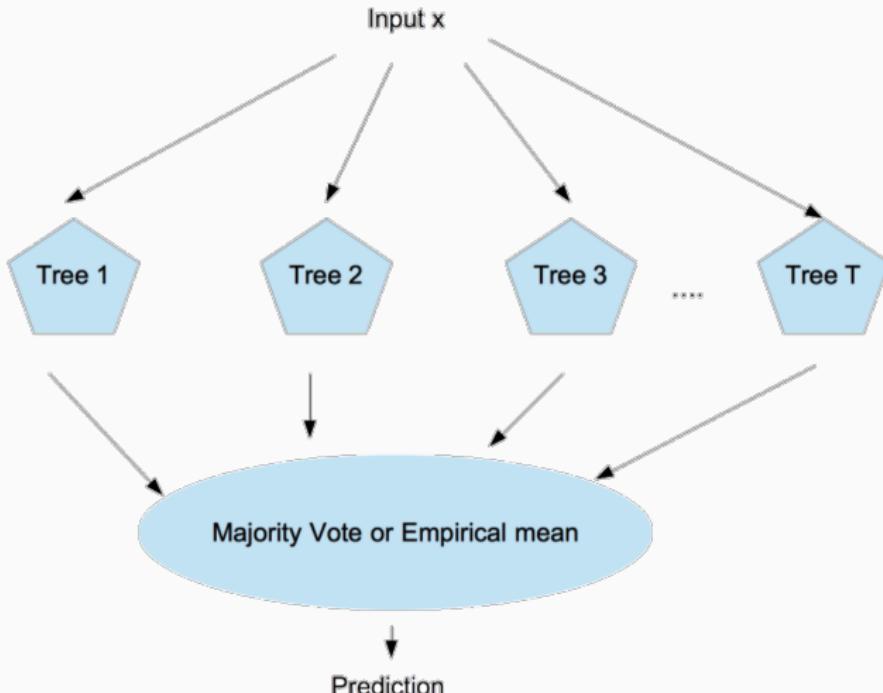
API : Scikit-learn

## Métodos de conjunto

Machine Learning not so "automatic": too many hyperparameters to tune!

## Métodos de conjunto

Machine Learning not so "automatic": too many hyperparameters to tune! **Ensemble learning** improves upon a single predictor by building an **ensemble of weak predictors** (with no hyperparameter)



## Diversity of the base predictors

$$MSE(f_{ens}) = \mathbb{E}[(y - f_{ens}(x))^2] = \frac{1}{T^2} \mathbb{E} \left[ \left( \sum_t \epsilon_t(x) \right)^2 \right]$$

Now, if  $\epsilon_t$  are mutually independent with zero mean, then we have:

$$MSE(f_{ens}) = \frac{1}{T^2} \mathbb{E} \left[ \sum_t \epsilon_t(x)^2 \right]$$

## Diversity of the base predictors

$$MSE(f_{ens}) = \mathbb{E}[(y - f_{ens}(x))^2] = \frac{1}{T^2} \mathbb{E} \left[ \left( \sum_t \epsilon_t(x) \right)^2 \right]$$

Now, if  $\epsilon_t$  are mutually independent with zero mean, then we have:

$$MSE(f_{ens}) = \frac{1}{T^2} \mathbb{E} \left[ \sum_t \epsilon_t(x)^2 \right]$$

⇒ The more diverse are the classifiers, the more we reduce the mean square error

## Diversity of the base predictors

$$MSE(f_{ens}) = \mathbb{E}[(y - f_{ens}(x))^2] = \frac{1}{T^2} \mathbb{E} \left[ \left( \sum_t \epsilon_t(x) \right)^2 \right]$$

Now, if  $\epsilon_t$  are mutually independent with zero mean, then we have:

$$MSE(f_{ens}) = \frac{1}{T^2} \mathbb{E} \left[ \sum_t \epsilon_t(x)^2 \right]$$

### Encourage the diversity of base predictors by:

- using bootstrap samples (Bagging and Random forests)
- using randomized predictors (ex: Random forests)
- using weighted version of the current sample (Boosting)
- with weights dependent on the previous predictor (adaptive sampling)

Un primer algoritmo:

- Construir  $T$  conjuntos de entrenamiento  $\mathcal{S}_1, \dots, \mathcal{S}_T$
- Aprender un modelo  $f_t \in \mathcal{F}$  para cada uno de estos conjuntos de entrenamiento  $\mathcal{S}_t, t = 1 \dots T$
- Calcular la media de los modelos  $f_{ens} = \frac{1}{T} \sum_{t=1}^T f_t(\mathbf{X})$

## ¿Cuándo es útil?

El error de aproximación se debe al sesgo (modelo con respecto a la tarea), a la varianza (modelo con respecto a la partición del conjunto de datos) y al ruido (depende de los descriptores que el modelo puede ver). Hacer una aproximación de varios modelos ayuda a reducir la varianza, lo que es útil para modelos con alta varianza como los árboles de decisión o las redes neuronales (es decir, modelos menos estables).

# Bagging (Bootstrap Aggregating) I

## Definition

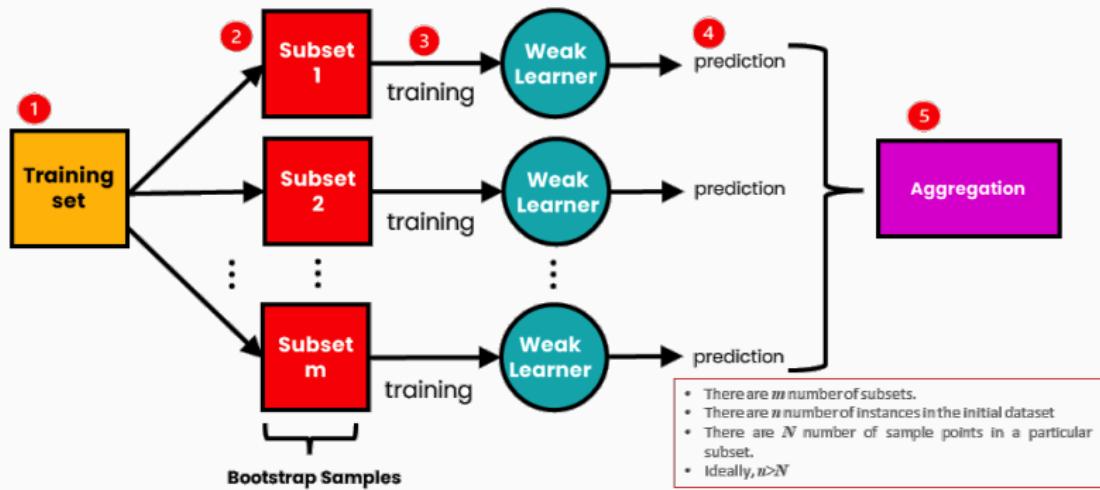
Ensemble learning technique designed to improve the accuracy and stability of machine learning algorithms.

It involves the following steps:

1. **Data Sampling:** Creating multiple subsets of the training dataset using bootstrap sampling (random sampling with replacement):  
*Construir  $T$  conjuntos de entrenamiento  $\mathcal{S}_1, \dots, \mathcal{S}_T$*
2. **Model Training:** training a separate model on each subset of the data: *Aprender un modelo  $f_t \in \mathcal{F}$  para cada uno de estos conjuntos de entrenamiento  $\mathcal{S}_t, t = 1 \dots T$*
3. **Aggregation:** Combining the predictions from all individual models (averaged for regression or majority voting for classification) to produce the final output: *Calcular la media de los modelos*  
$$f_{ens} = \frac{1}{T} \sum_{t=1}^T f_t(\mathbf{X})$$

# Bagging (Bootstrap Aggregating) II

## The Process of Bagging (Bootstrap Aggregation)



It helps to:

- **Reduces Variance:** By averaging multiple predictions, bagging reduces the variance of the model and helps prevent overfitting.
- **Improves Accuracy:** Combining multiple models usually leads to better performance than individual models

# Boosting I

## Definition

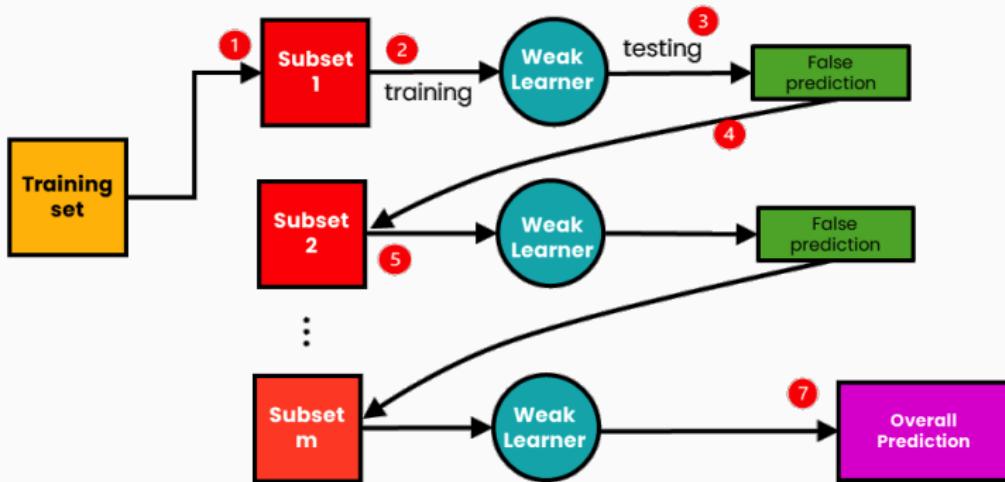
Ensemble learning technique that focuses on creating a strong model by combining several weak models.

It involves the following steps:

1. **Sequential Training:** Training models sequentially, each one trying to correct the errors made by the previous models.
2. **Weight Adjustment:** Each instance in the training set is weighted. Initially, all instances have equal weights. After each model is trained, the weights of misclassified instances are increased so that the next model focuses more on difficult cases.
3. **Model Combination:** Combining the predictions from all models to produce the final output, typically by weighted voting or weighted averaging.

# Boosting II

## The Process of Boosting



It helps to:

- **Reduces Bias:** By focusing on hard-to-classify instances, boosting reduces bias and improves the overall model accuracy.
- **Produces Strong Predictors:** Combining weak learners leads to a strong predictive model.

# Outline : Bosque aleatorios

K-vecinos más cercanos

Naive Bayes

**Bosque aleatorios**

Árboles de decisión

Métodos de conjunto

Bosque aleatorios

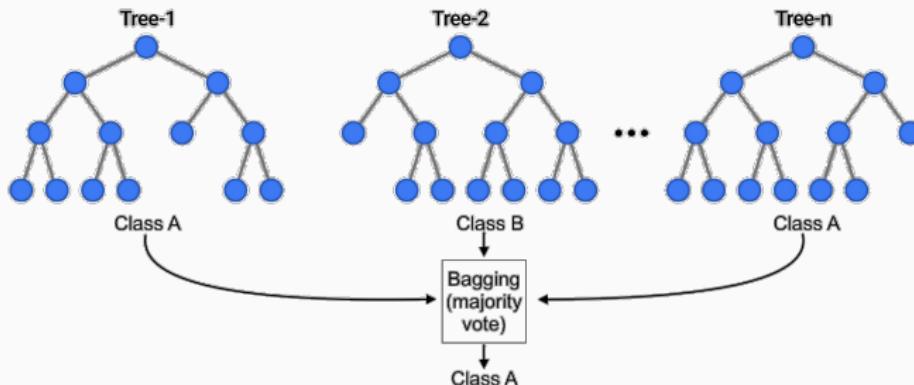
Regresión por Bosques Aleatorios

API : Scikit-learn

# Bosques Aleatorios

## Algoritmo

- $F$  descriptores candidatos, conjunto de entrenamiento  $\mathcal{S}_{train}$
- Para  $t = 1 \dots T$ 
  - $\mathcal{S}_{train}^{(t)} \leftarrow n_t$  muestras tomadas al azar con reemplazo<sup>1</sup> desde  $\mathcal{S}_{train}$
  - $h_{arbol}^{(t)} \leftarrow$  árbol de decisión aleatorio aprendido desde  $\mathcal{S}_{train}^{(t)}$
- $h^T = \frac{1}{T} \sum_t h_{arbol}^{(t)}$



<sup>1</sup>Bootstrap

# Bosques Aleatorios : aprendizaje de un árbol

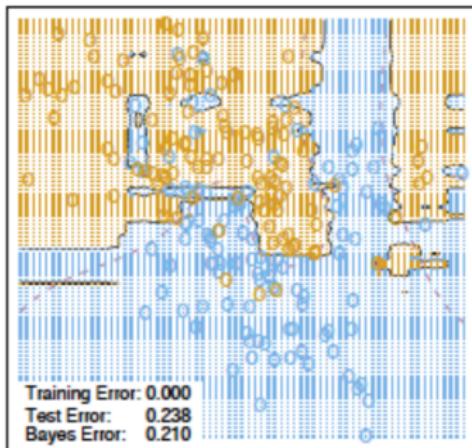
---

Aprendizaje de un único árbol aleatorio:

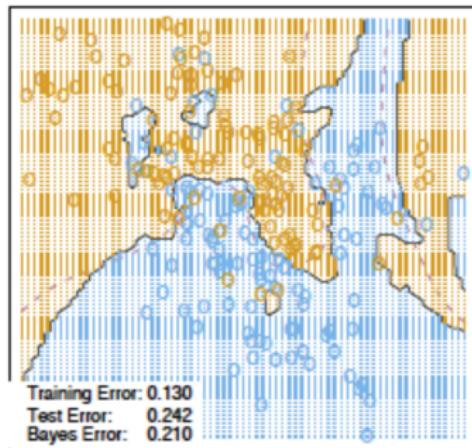
- Para seleccionar una separación en un nodo:
  - $R_k(F) \leftarrow n_t$  seleccionar aleatoriamente (sin reemplazo)  $k$  descriptores en los  $F$ , con  $k \ll F$
  - Elegir la mejor separación en  $R_k(f)$  (considerar los diferentes puntos de separación)
- No podar este árbol

# Bosques Aleatorios : visualización

Random Forest Classifier



3-Nearest Neighbors



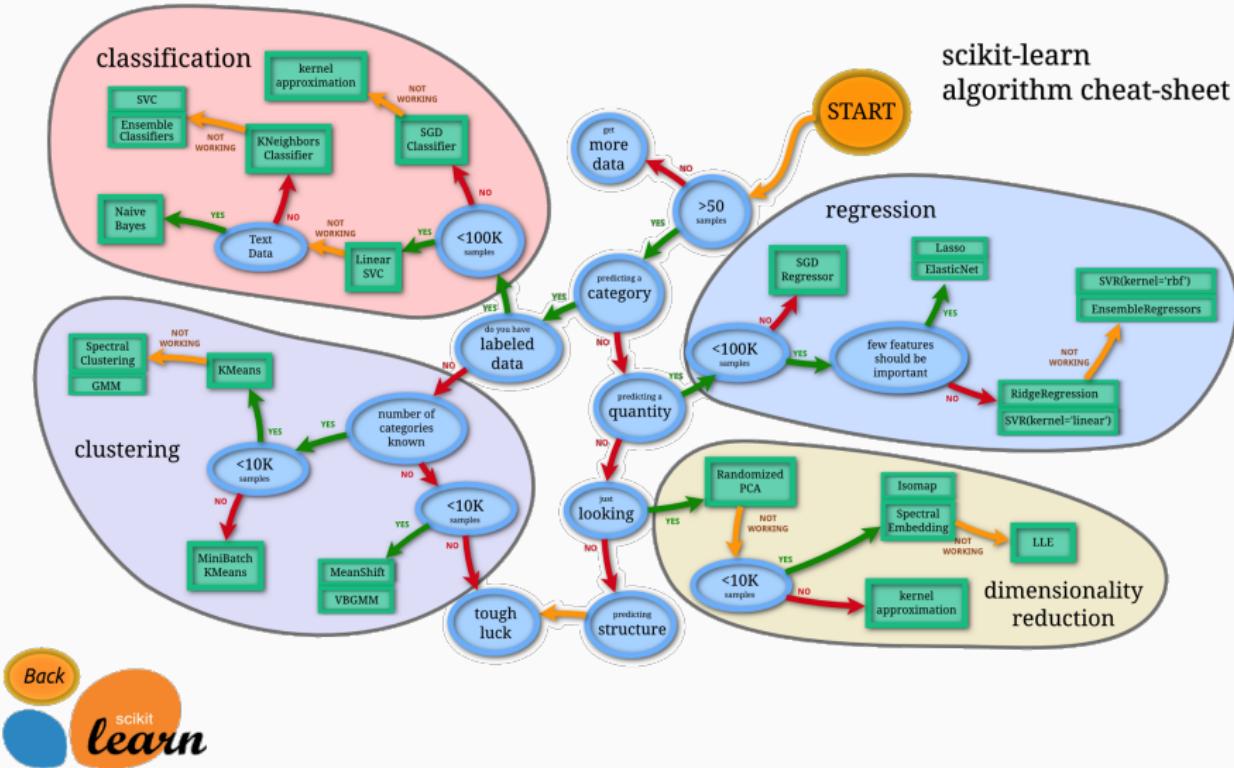
# Resumen

Algo	Type	Tolerance number features	Parametrization	Memory size	Minimal required quantity	Com m	Overfitting Tendency	Difficulty	Time for Learning	Time for predicting
Linear Regression	R	Weak	Weak	Small	Small	++	Low	Weak	Weak	Weak
Logistic Regression	C	Weak	Simple	Small	Small	++	Low	Weak	Weak	Weak
Decision Tree	R & C	Strong	Simple / intuitive	Large	Small	+++	Very high	Weak	Weak	Weak
Random Forest	R & C	Strong	Simple / intuitive	Very Large	Large	++	Average	Average	Costly	Costly
Boosting	R & C	Strong	Simple / intuitive	Very Large	Large	+	Average	Average	Costly	Weak
Naive Bayes	C	Weak	No params.	Small	Small	++	Low	Weak	Weak	Weak
SVM	C	Very strong	Not intuitive	Small	Large	--	Average	High	Costly	Weak
Neural Network (NN)	C	Very strong **	Not intuitive	Inter	Large	---	Average	Very high	Costly	Weak
Deep Neural Network	C	Very strong **	Not intuitive	Very Large	Very Large	---	High	Very high	Very costly	Weak
K-Means	CL*	Strong	Simple / Intuitive		Small	+		High	Weak	
One class SVM	A	Very strong	Not intuitive	Weak	Large	--	Average	High	Costly	Weak

Un resumen de los principales modelos de ML

[Otra tabla](#)

# Resumen: Cheatsheet Sklearn



# Árboles de regresión

Solo hay que cambiar la función  $H$  para que se adapta a un problema de regresión.

Maximización de la reducción de la varianza:

$$L(t_{j,\tau}, \mathcal{S}) = VAR_{emp}(\mathcal{S}) - \frac{|\mathcal{D}(\mathcal{S}, j, \tau)|}{n} VAR_{emp}(\mathcal{D}(\mathcal{S}, j, \tau)) - \frac{|\mathcal{I}(\mathcal{S}, j, \tau)|}{n} VAR_{emp}(\mathcal{I}(\mathcal{S}, j, \tau))$$

Donde:

$$VAR_{emp}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{(x^i, y^i) \in \mathcal{S}} (y^i) - \bar{y}$$

## **API : Scikit-learn**

---

# Outline : API : Scikit-learn

---

K-vecinos más cercanos

Naive Bayes

Bosque aleatorios

Árboles de decisión

Métodos de conjunto

Bosque aleatorios

**API : Scikit-learn**

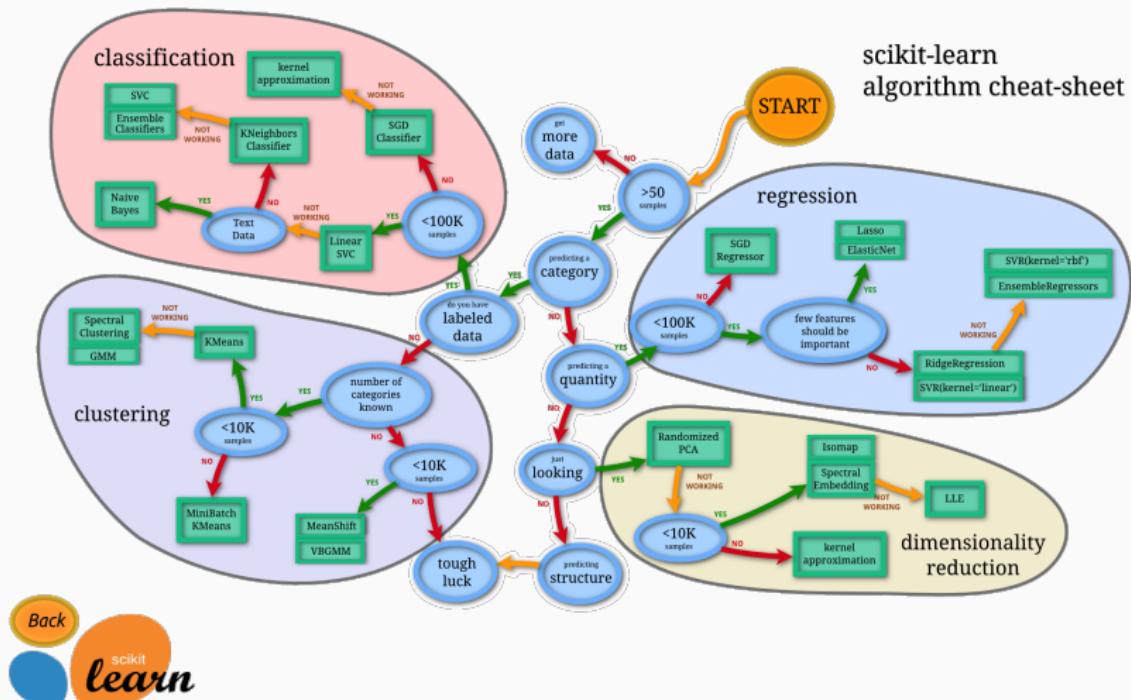
## Scikit-learn: biblioteca de Python

Biblioteca de Aprendizaje Automático muy fácil de usar:

<https://scikit-learn.org/>



# Scikit-learn: mapa de posibilidades



# Scikit-learn: funciones normalizadas

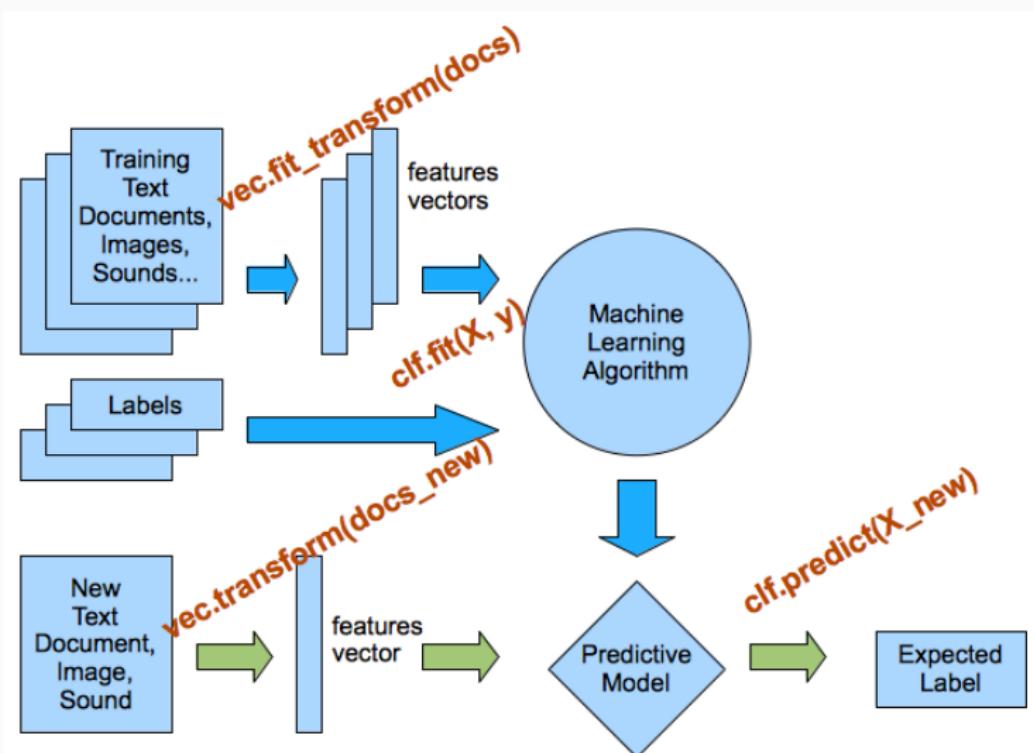
Funciones normalizadas para un aprendizaje rápido y claridad en el código:

- Los diferentes modelos (o conjuntos de datos) se implementan como clases (tipos de variables)
- Se pueden llamar métodos en estas variables para las diferentes etapas del algoritmo: Extracción de características, Aprendizaje de parámetros, Predicción,...

```
>>> from sklearn.datasets import load_iris
>>> data = load_iris()
>>> X, y = data.data, data.target
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)
>>> from sklearn.linear_model import SGDClassifier
>>> clf = SGDClassifier(max_iter=1000, tol=1e-3)
>>> clf.fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.92...
```

# Scikit-learn: funciones normalizadas

Funciones normalizadas para un aprendizaje rápido y claridad en el código:



# Scikit-learn: Conjuntos de datos

## Múltiples conjuntos de datos disponibles :

- Conjuntos de datos "juguetes": precios de casas de Boston, plantas de Iris, Diabetes, reconocimiento de vinos, ...
- Conjuntos de datos reales: Caras Olivetti (imagen), Coberturas forestales (geográficas), Conjunto de datos RCV1 (texto), ...

```
>>> from sklearn.datasets import load_wine
>>> data = load_wine()
>>> data.keys()
['target_names', 'data', 'target', ...
'DESCR', 'feature_names']
```

# Scikit-learn: Extracción de características

## Funciones ya codificadas para vectorizar fácilmente características

- Desde diccionarios: `sklearn.feature_extraction`
- Desde imágenes: `sklearn.feature_extraction.image`
- Desde texto: `sklearn.feature_extraction.text`

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'Este es el primer documento.',
...     'Este es el segundo segundo documento.',
...     'Y el tercero.',
...     '¿Este es el primer documento?',
... ]
>>> X = vectorizer.fit_transform(corpus).toarray()
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
[0, 1, 0, 1, 0, 2, 1, 0, 1],
[1, 0, 0, 0, 1, 0, 1, 1, 0],
[0, 1, 1, 1, 0, 0, 1, 0, 1]]...)
```

# Scikit-learn: Selección de características

## Técnicas para seleccionar características

- Eliminar características de baja varianza: VarianceThreshold
- Usar modelos para seleccionar características: SelectFromModel
- Eliminación recursiva de características: RFE

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectFromModel
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> X.shape
(150, 4)
>>> lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
>>> model = SelectFromModel(lsvc, prefit=True)
>>> X_new = model.transform(X)
>>> X_new.shape
(150, 3)
```

# Scikit-learn: Preprocesamiento de datos

Normalización, estandarización necesaria para un mejor aprendizaje

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
scaler.transform(X_test)
```

Valores faltantes

```
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp.fit([[1, 2], [np.nan, 3], [7, 6]])
SimpleImputer(copy=True, fill_value=None, missing_values=nan, strategy=mean)
>>> X = [[np.nan, 2], [6, np.nan], [7, 6]]
>>> print(imp.transform(X))
[[4.          2.          ]
 [6.          3.666...]
 [7.          6.          ]]
```

# Scikit-learn: Validación cruzada

Funciones para la validación de modelos y separación en conjuntos de entrenamiento/prueba:

- De manera simple `train_test_split`, para una K-CrossVal KFold, en Leave-one-out `LeaveOneOut`, conservando las proporciones de las etiquetas `StratifiedKFold`
- Haciendo directamente la CrossVal: `cross_val_score`

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)

>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
```

# Scikit-learn: Modelos de aprendizaje supervisado

Numerosos modelos supervisados normalizados para una fácil utilización:

- Regresión Logística/Lineal, Análisis Discriminativo Lineal/Quadrático, SVM, K-NN, Descenso de Gradiente Estocástico, Bayesiano Ingenuo, Árboles de decisión, Bosques Aleatorios, ...
- Cada uno de estos modelos se implementan como clases python con métodos que son comunes a todas las clases: fit, score, predict\_proba, predict, ...

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0, solver='lbfgs',
...                           multi_class='multinomial').fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
>>> clf.score(X, y)
0.97...
```

# Scikit-learn: Modelos de aprendizaje no supervisado

Numerosos modelos no supervisados normalizados para clustering, descomposición de señales en componentes, reducción de dimensiones:

- Clustering: K-medias, PCA, Factorización de Matrices no negativas, Asignación Latente de Dirichlet, t-sne,...
- Estos modelos se implementan como clases python con métodos que son comunes a todas las clases: fit, fit\_transform, predict,  
...

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...                 [4, 2], [4, 4], [4, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([0, 0, 0, 1, 1, 1], dtype=int32)
>>> kmeans.predict([[0, 0], [4, 4]])
array([0, 1], dtype=int32)
>>> kmeans.cluster_centers_
array([[1., 2.],
       [4., 2.]])
```

# Scikit-learn: Métricas

Múltiples métricas disponibles para la evaluación de modelos:

- de clasificación: Recuerdo, Precisión, F1, AUC, ROC, Matriz de confusión, ...
- de regresión:  $R^2$ , Error Cuadrático Medio, ..
- de clustering: Completitud, V-medida, ...

```
>>> import numpy as np
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
```

# Scikit-learn: Encontrar los hiperparámetros con una búsqueda

Funciones para encontrar los hiperparámetros óptimos realizando validaciones cruzadas para diferentes hiperparámetros.

- En una cuadrícula dada: GridSearchCV
- En relación con una distribución aleatoria de parámetros: RandomizedSearchCV
- Estos modelos se implementan como clases python con métodos que son comunes a todas las clases: fit, fit\_transform, predict,...

```
>>> from sklearn import svm, datasets
>>> from sklearn.model_selection import GridSearchCV
>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
>>> svc = svm.SVC(gamma="scale")
>>> clf = GridSearchCV(svc, parameters, cv=5)
>>> clf.fit(iris.data, iris.target)
```

**Questions?**

## References i