upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

TEMA 10: ENSEMBLES, ONE-VS-ONE Y ONE-VS-ALL

Mikel Galar Idoate
mikel.galar@unavarra.es

Ciencia de datos con técnicas inteligentes
Experto Universitario en Ciencia de Datos y Big Data

Índice

Índice

Introducción

- **Conjuntos de clasificadores**
  - **Ensembles**
    - Combinaciones de pequeñas **variantes del mismo** clasificador
  - **Sistemas con múltiples clasificadores**
    - **Cualquier combinación** de clasificadores
    - Incluyen combinaciones de diferentes modelos
      - P. e. Red neuronal + Naïve Bayes + Logistic Regression

Introducción

- **Motivación**
  - **Limitación** de modelos individuales
  - Consultas a varios **expertos**
  - Evitar **mínimos locales**
  - Problemas **complejos** para modelos individuales
    - Particionamiento de datos
    - Subproblemas menos complejos
  - **Muchos / pocos datos**
  - **Fusión** de datos de diferentes fuentes

Introducción

- **Diversidad**
  - Factor **clave**
  - **Objetivo**
    - Cometer **fallos diferentes**
    - Su **combinación** lleve a obtener **más aciertos**
    - Clasificadores **complementarios**
  - Diferentes **formas de alcanzarla**
    - Conjuntos de **datos diferentes**
    - **Modelos diferentes**
    - **Particionamiento** de datos
    - **Hibridaciones**

Introducción

- **Clasificadores base**
  - Cada uno de los clasificadores que forma parte del ensemble
- **Clasificadores débiles**
  - Pequeños cambios en los datos producen grandes cambios en el modelo
    - Árboles de decisión
      - Decision Stumps
    - Redes neuronales
    - Regresión logística (sin regularización)
    - Naïve Bayes

Introducción

- **Tipos de ensembles**
  - Diferentes formas de
    - **Generar** clasificadores
    - **Combinar** clasificadores

1. **Selección de clasificadores**
   - *Generación*: Clasificadores especializados en zonas
   - *Combinación*: selección de los más adecuados

2. **Fusión de clasificadores**
   - *Generación*: Clasificadores expertos en todas las zonas
   - *Combinación*: Fusión de todos ellos para obtener un clasificador fuerte
     - Bagging, Boosting (AdaBoost) …

Introducción

- **¿Cuándo mejora un ensemble a un único clasificador?**
  - Los clasificadores base son capaces de **corregirse mutuamente**
    - Si las salidas de todos son iguales (**no hay diversidad**)
      - **No hay mejora**
    - Si cada clasificador comete errores diferentes (**diversidad**)
      - Con una buena combinación **puede haber mejora**
- **Metodología** habitual
  - Diferentes conjuntos de entrenamiento
  - **Re-muestreo + clasificadores débiles**
- También…
  - Diferentes parámetros, características, etc.

Índice

Ensembles basados en variación de datos

- **Algoritmo genérico**
  - **Entrenar T clasificadores base débiles**
    - Cada uno utiliza un conjunto de datos diferente
    - Al ser clasificadores base débiles
      - Cambios en los datos → cambios en los clasificadores
  - **Combinarlos** para obtener la salida final
- **Diferencias** en los modelos:
  - Forma de construir los conjuntos de datos
    - **En paralelo** (conjuntos de datos independientes)
      - Bagging / Random Subspace Method
    - **En serie** (cada conjunto depende del anterior)
      - Boosting

Índice

Bagging

- **Bagging = Boostrap AGGreatING**
  - **Reduce la varianza** → efecto de hacer medias
  - Poco riesgo de sobre-aprendizaje
  - Muy simple
  - Regresión y clasificación
- **Boostrap de m_B ejemplos (bolsa)**
  - Subconjunto de datos obtenido con **re-muestro aleatorio con reemplazamiento**
  - Normalmente, $m_{Train} = m_B$
    - ≈63.2% de las instancias en cada bolsa

Bagging

- **Algoritmo**
  - **Repetir T veces** (número de clasificadores)
    - **Obtener bolsa (boostrap)** del conjunto de entrenamiento
      - **¡¡Mecanismo aleatorio!!**
    - **Entrenar un clasificador débil** con el nuevo conjunto
- **Combinación** de las salidas
  - **Voto simple**
    - Cada clasificador da un voto (0/1) a la clase que predice
  - **Voto ponderado**
    - Si el clasificador devuelve una probabilidad para cada clase
    - Cada clasificador vota con cierta probabilidad a cada calse
  - La clase con mayor número de votos es la predicha
- **Diversidad**
  - Se alcanza por el **mecanismo re-muestro** de ejemplos

Bagging

- **Bagging en clasificación**

Algorithm 1 Bagging

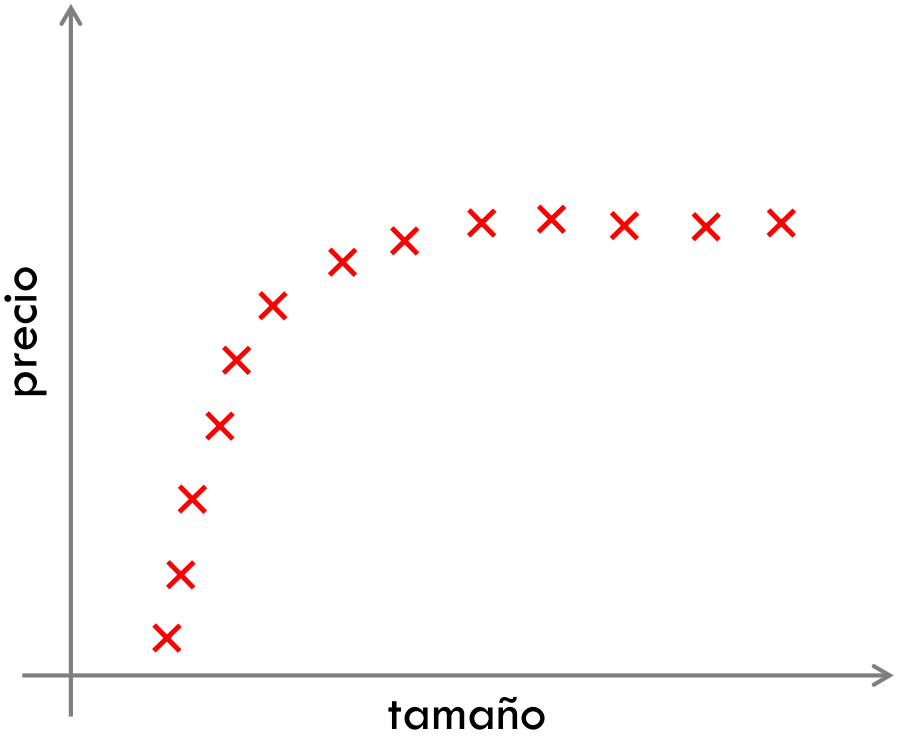
Input: S: Training set; T: Number of iterations;
n: Bootstrap size; I: Weak learner

Output: Bagged classifier: $H(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right)$ where $h_t \in [-1, 1]$ are the induced classifiers

1: **for** $t = 1$ to T **do**
2: $S_t \leftarrow$ RandomSampleReplacement(n,S)
3: $h_t \leftarrow$ I(S_t)
4: **end for**

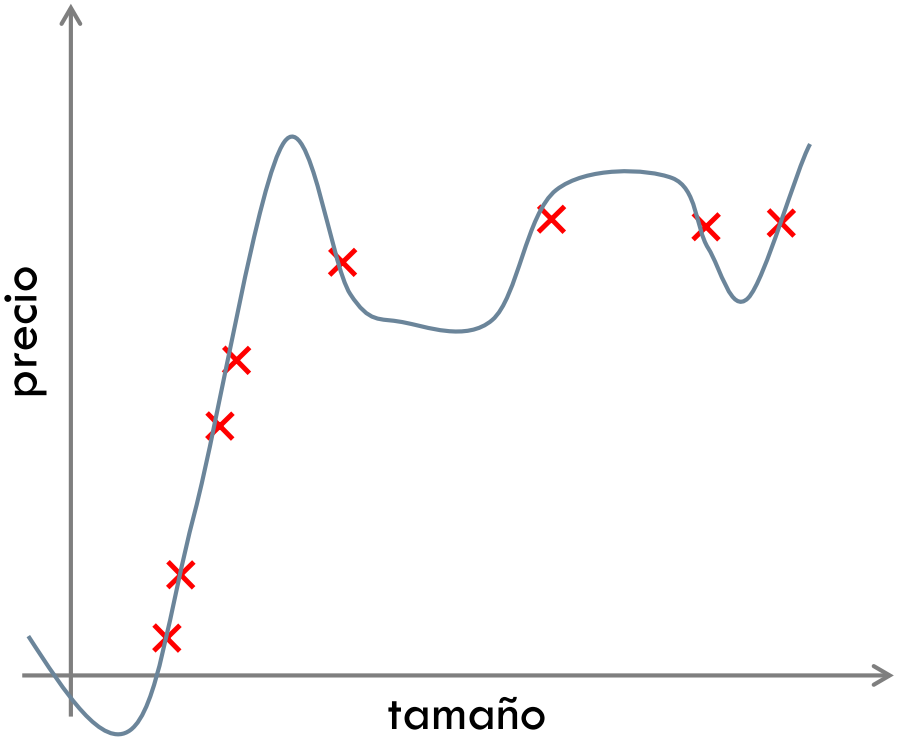
Bagging

- **Ejemplo regresión**
  - Problema inicial: datos train



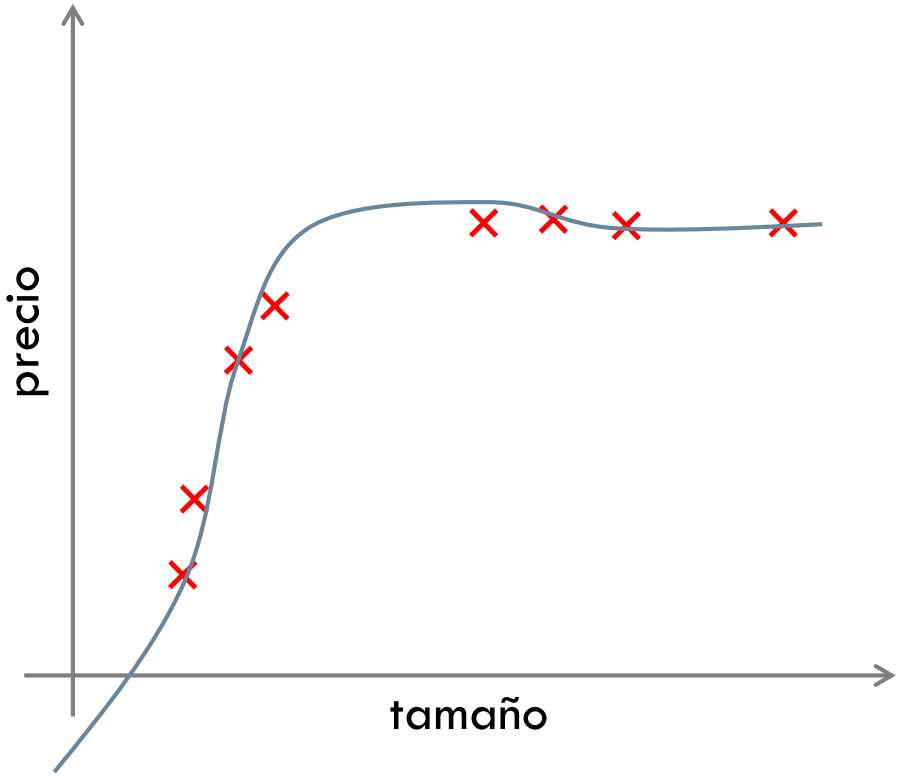
Bagging

- **Ejemplo regresión**
  - Iteración 1: Bootstrap



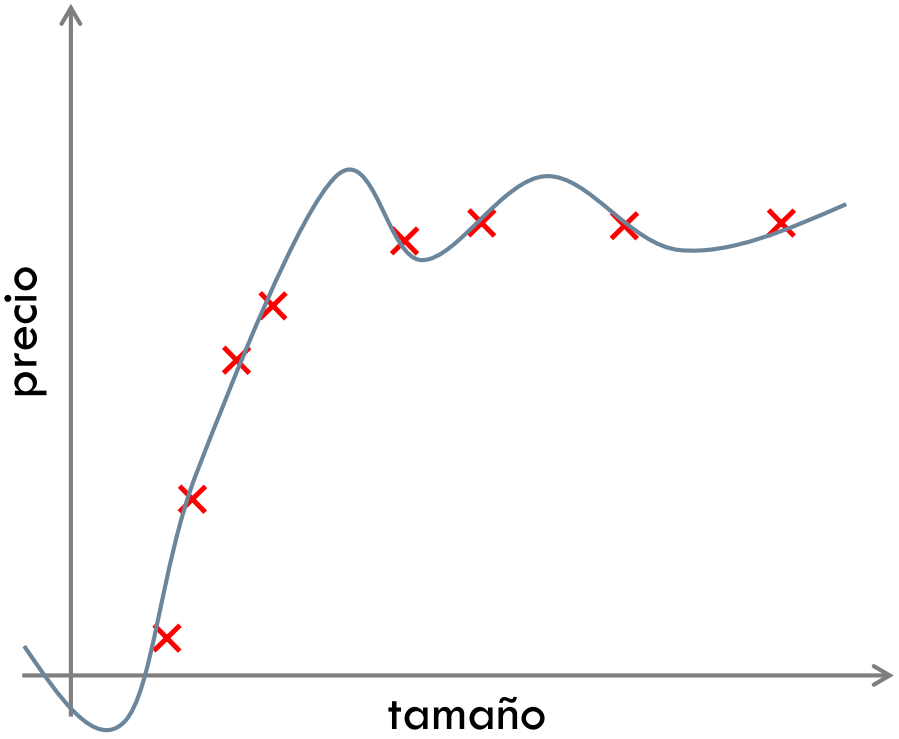
Bagging

- **Ejemplo regresión**
  - Iteración 2: Bootstrap



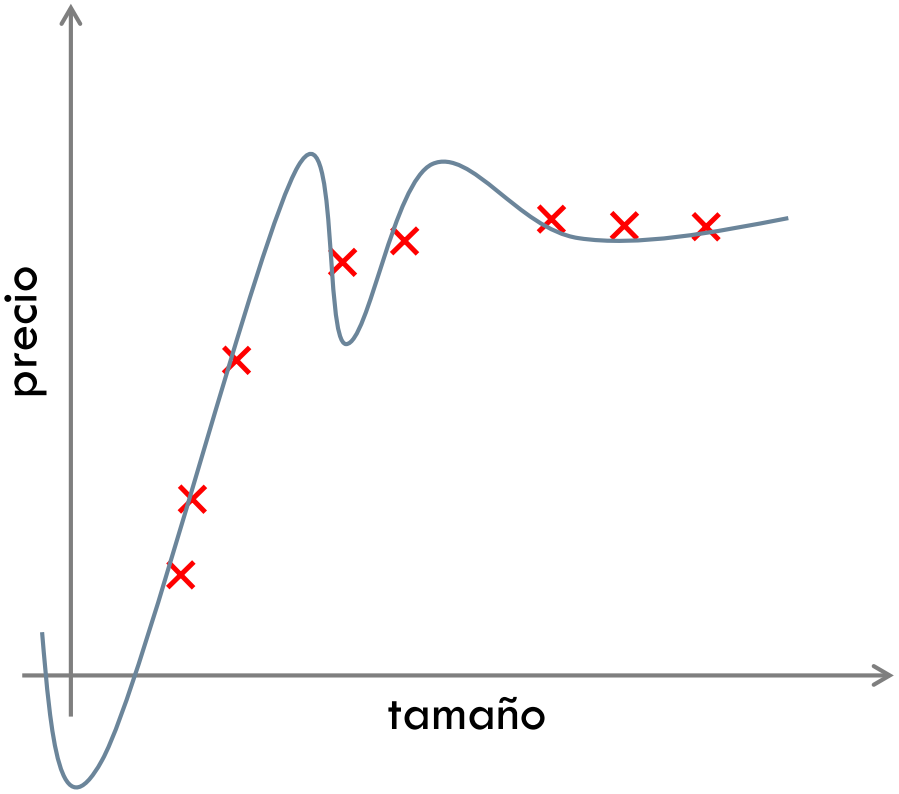
Bagging

- **Ejemplo regresión**
  - Iteración 3: Bootstrap



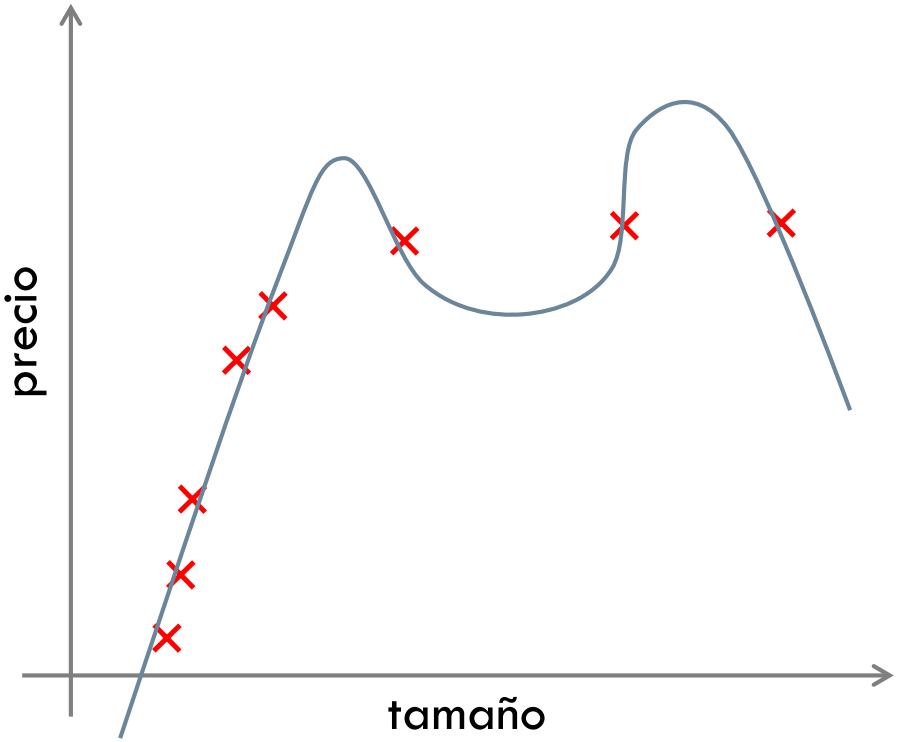
Bagging

- **Ejemplo regresión**
  - Iteración 4: Bootstrap



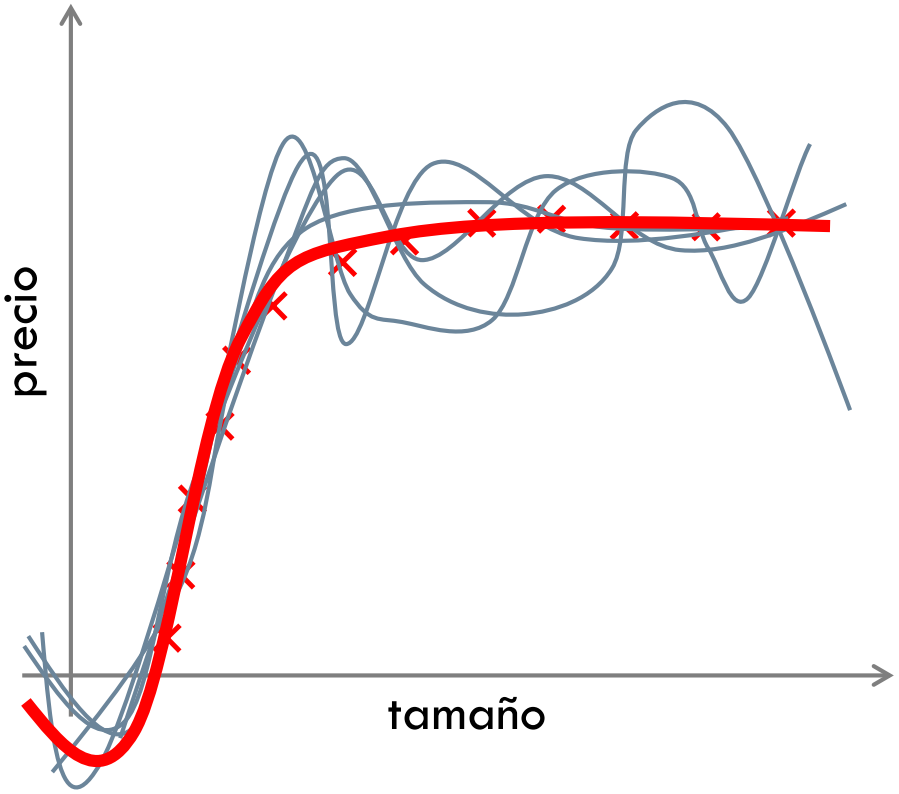
Bagging

- **Ejemplo regresión**
  - Iteración 5: Bootstrap



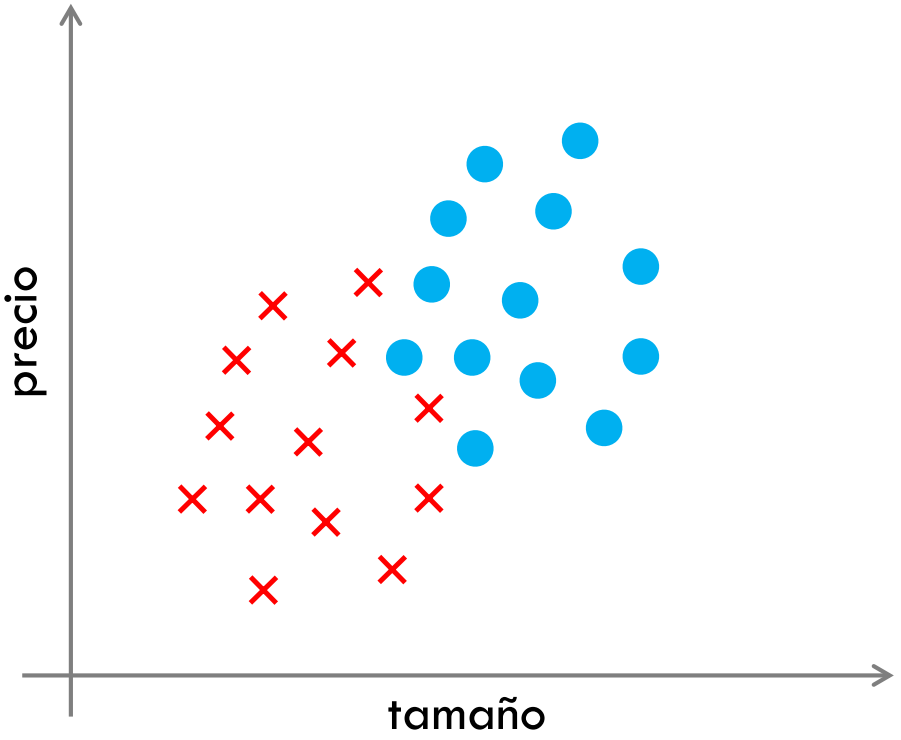
Bagging

- **Ejemplo regresión**
  - Combinación



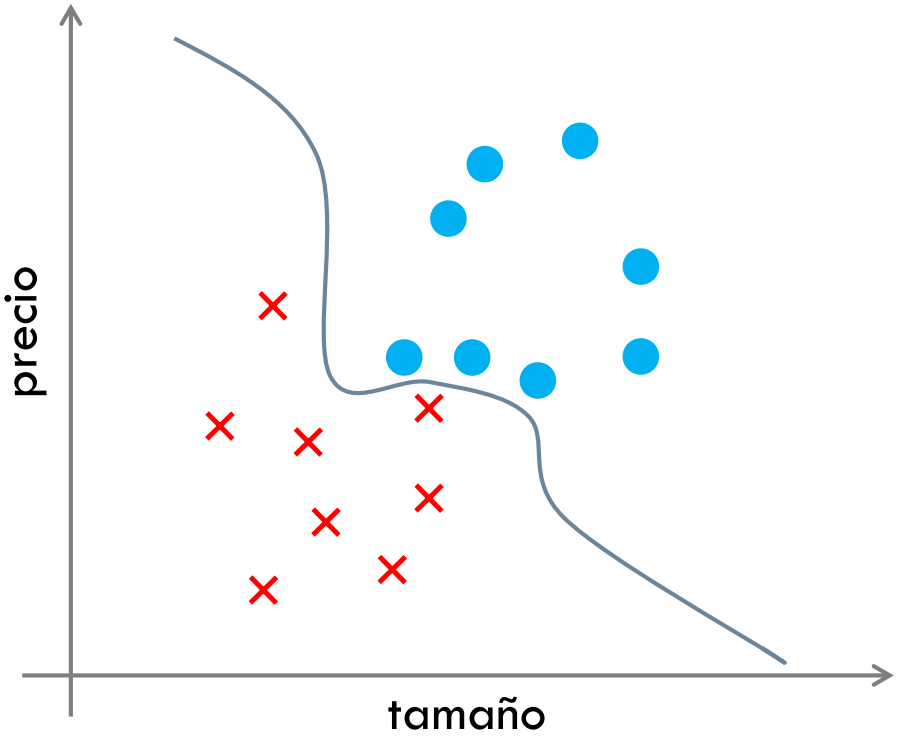
Bagging

- **Ejemplo clasificación**
  - Problema inicial: conjunto de train



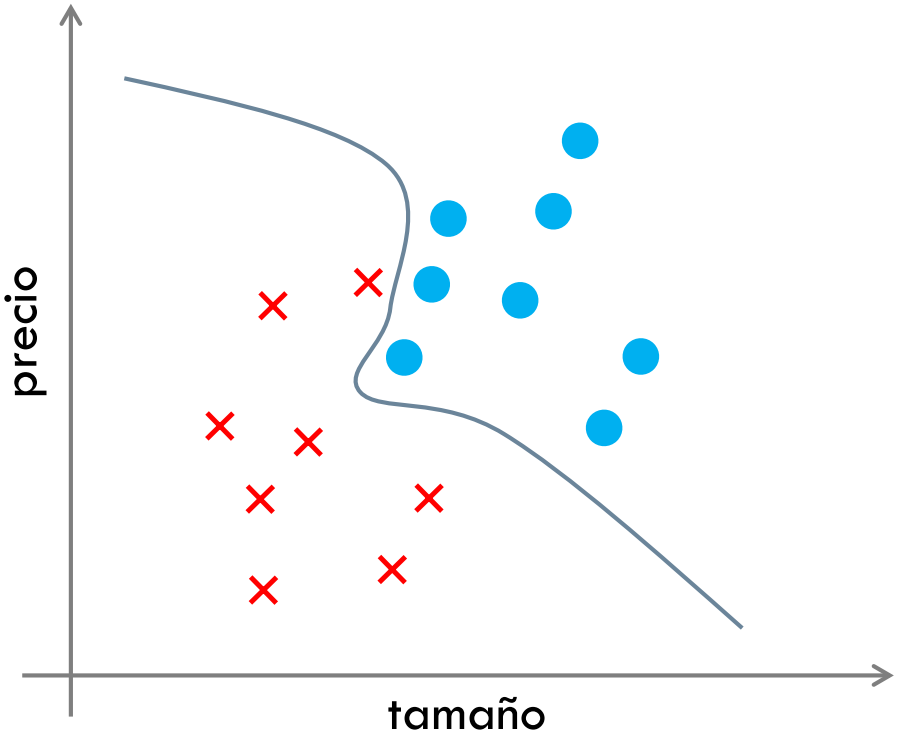
Bagging

- **Ejemplo clasificación**
  - Iteración 1: Bootstrap



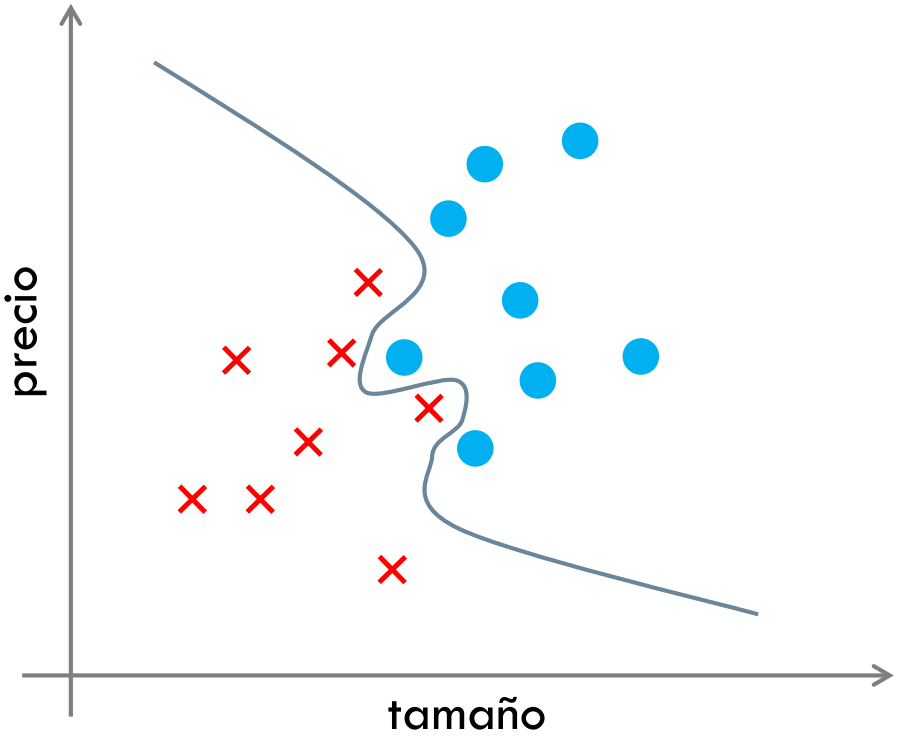
Bagging

- **Ejemplo clasificación**
  - Iteración 2: Bootstrap



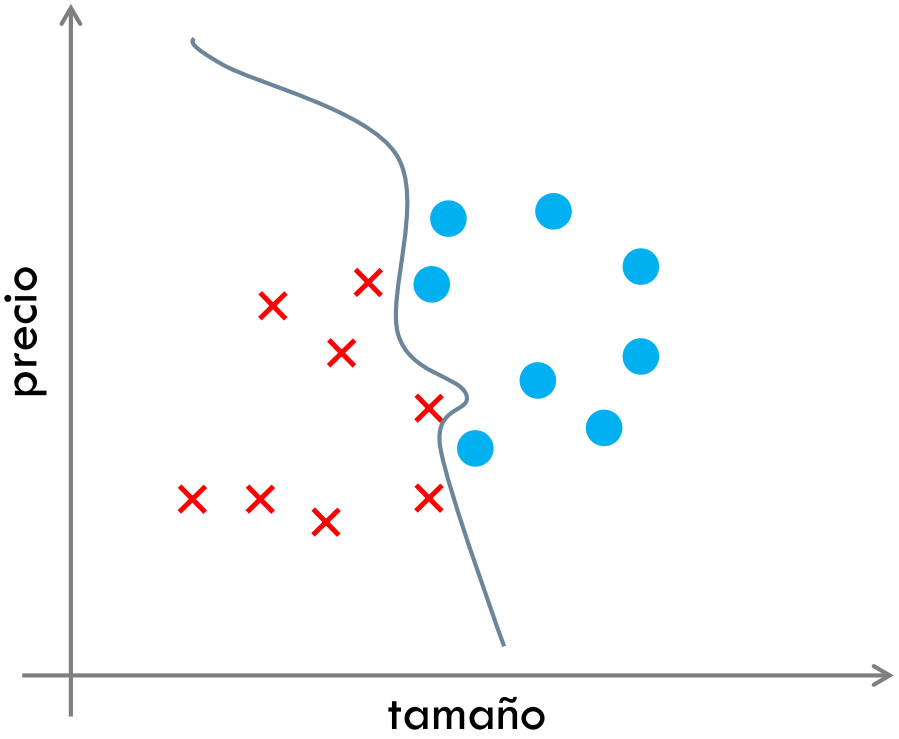
Bagging

- **Ejemplo clasificación**
  - Iteración 3: Bootstrap



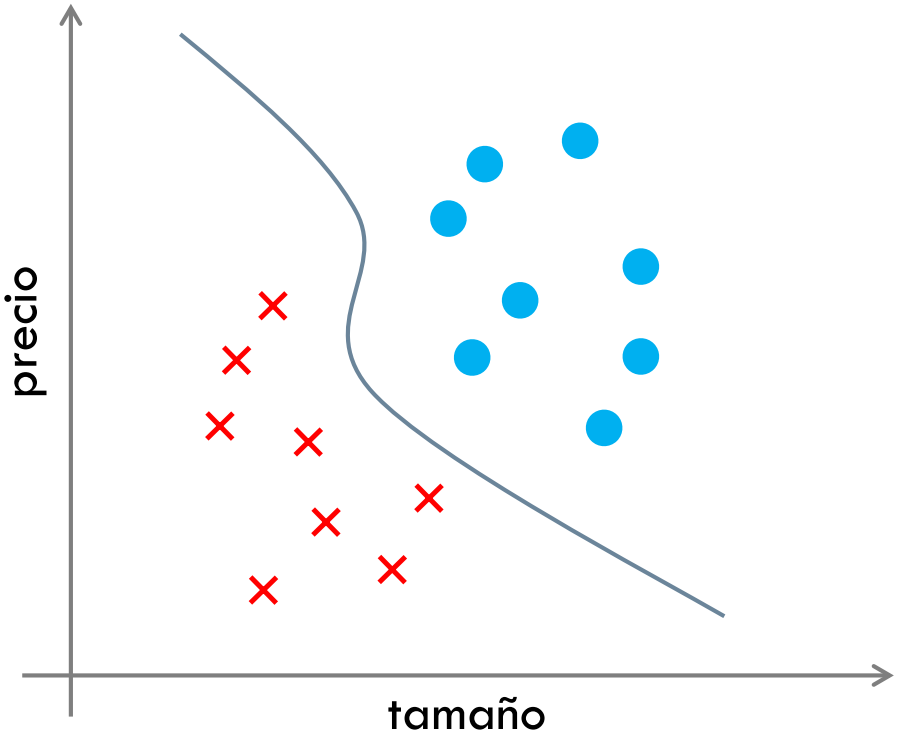
Bagging

- **Ejemplo clasificación**
  - Iteración 4: Bootstrap



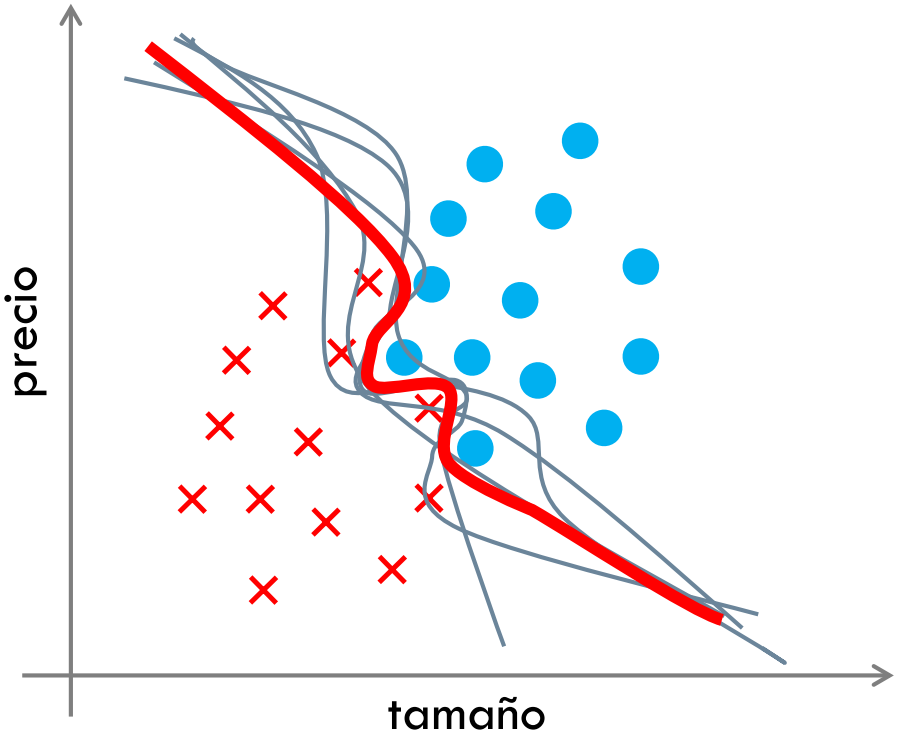
Bagging

- **Ejemplo clasificación**
  - Iteración 5: Bootstrap



Bagging

- **Ejemplo clasificación**
  - Combinación



Bagging en Python

- Paquete

  from sklearn.ensemble import BaggingClassifier

- Método de consutrcción del clasificador

  BaggingClassifier(base_estimator=Clasificador, n_estimators=numeroClasificadores, random_state=semilla)

  - Parámetros
    - Clasificador: clasificador con el que conformar el ensemble
    - numeroClasificadores: número de clasificadores base
    - Semilla: número entero utilizado para evitar la aleatoriedad
- Entrenamiento y clasificación
  - Funciones ***fit*** y ***predict***, respectivamente
- Más información en:
  - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Índice

Random Subspace Method

- **Similar a Bagging**
  - En vez de re-muestrear ejemplos
  - **Re-muestreamos atributos**
  - Adecuando cuando el **ratio atributos/ejemplos es alto**
  - Cada clasificador se entrena con un conjunto diferente de atributos
- **Algoritmo**
  - **Repetir T veces** (número de clasificadores)
    - **Seleccionar aleatoriamente p características** ($p < P$)
    - Crear el conjunto de entrenamiento con **todos los ejemplos utilizando solo las p características** (puede combinarse con bagging)
    - **Entrenar un clasificador débil** con el nuevo conjunto

Random Subspace Method en Python

- Paquete

  from sklearn.ensemble import BaggingClassifier

- Método de consutrcción del clasificador

  BaggingClassifier(base_estimator=Clasificador, n_estimators=numeroClasificadores, max_features=porcentajeVariables, random_state=semilla)

  - Parámetros
    - ***Clasificador***: clasificador con el que conformar el ensemble
    - ***numeroClasificadores***: número de clasificadores base
    - *porcentajeVariables: porcentaje de variables a seleccionar*
    - semilla: número entero utilizado para evitar la aleatoriedad

- Entrenamiento y clasificación
  - Funciones ***fit*** y ***predict***, respectivamente
- Más información en:
  - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Índice

Random forest

- Un random forest es un ensemble en el que los clasificadores base son árboles de decisión
- Aprendizaje de random forest
  - Seleccionar el número de árboles
  - Para aprender cada árbol
    - Seleccionar N ejemplos con reemplazamiento del conjunto de entrenamiento (bootstrap), siendo N el número de ejemplos
      - A los ejemplos no seleccionados se les llama out-of-bag (OOB)
    - Aprender el árbol de decisión con esa muestra de ejemplos
      - En cada nodo se eligen aleatoriamente m atributos ($m \ll M$)
        - M es el número de atributos del problema
        - $m = 1$
        - $m = \sqrt{M}$
      - Se aplica la heurística para determinar el mejor de esos m atributos
    - No se poda el árbol de decisión generado

Random forest

- Clasificación de nuevos ejemplos
  - Clasificar el ejemplo con todos los árboles
  - Cada árbol "vota" a la clase que predice
    - Voto en base a la probabilidad de la predicción
  - Elegir la clase con más votos
    - La clase con mayor probabilidad media

- Estimación del error de random forest
  - Para cada ejemplo de entrenamiento
    - Clasificar el ejemplo con los clasificadores base que no hayan sido construidos con ese ejemplo

Random forest

- El error de random forest depende
  - La diversidad entre cualquier par de árboles
    - Reducir la diversidad aumenta el error del random forest
  - La calidad de los árboles base
    - Aumentar su calidad reduce el error del random forest
- Reducir m aumenta la diversidad y reduce la calidad
  - Aumentar m, reduce la diversidad y aumenta la calidad

Random forest

- Ventajas
  - Buen rendimiento
  - Relativamente robusto frente a ruido y outliers
  - Es rápido (comparado con bagging y boosting)
  - Es simple y fácilmente paralelizable
- Desventajas
  - Pierde la interpretabilidad de los árboles de decisión

Random Forest en Python

- Paquete

  from sklearn.ensemble import RandomForestClassifier

- Método de consutrcción del clasificador

  RandomForestClassifier(n_estimators=numeroClasificadores, criterion=tipoImpureza, max_features=numeroVariables, random_state= semilla)

  - Parámetros
    - ***numeroClasificadores***: número de clasificadores base
    - ***tipoImpureza***: 'gini' o 'entropy'. Es decir, CART o C4.5
    - *numeroVariables: número de variables a examinar en cada nodo*
      - *'auto': examina sqrt(numeroTotalVariables)*
        - *Es igual que utilizar 'sqrt'*
      - *'log2': examina log2(numeroTotalVariables)*
      - *'None': examina todas las variables*
      - *Número entero: número concreto de variables a examinar*
    - semilla: número entero utilizado para evitar la aleatoriedad

- Entrenamiento y clasificación
  - Funciones ***fit*** y ***predict***, respectivamente
- Más información en:
  - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

Índice

Boosting

- **Uno de los 10 mejores algoritmos de Data Mining**
- **Objetivo**
  - **Transformar un clasificar débil** (ligeramente mejor que uno aleatorio) **en un clasificador fuerte**
- **Idea**
  - Se consideran **todos los datos**
  - Se da **más peso a las instancias difíciles**
    - Instancias que han sido mal clasificadas por el clasificador anterior
- **Ventajas**
  - **Reduce el bias (y la varianza)**
  - Aumenta el margen de separación entre clases
  - Fuerte base teórica
- **Problema**: **ruido**, instancias con ruido reciben más peso

Boosting

- **Algoritmo general**
  - Asignar el mismo peso a todos los ejemplos
  - **Repetir T veces** (número de clasificadores)
    - **Aprender un clasificador débil** con los pesos actuales
    - **Modificar los pesos**
      - Aumentar el peso de los ejemplos mal clasificados
      - Decrecer el peso de los ejemplos correctamente clasificados
    - **Asignar un peso al clasificador actual** en base a su precisión
- **Importante**
  - **El clasificador debe poder manejar pesos**
  - Sino hay que utilizar re-muestreo con probabilidades

Boosting

Algorithm 3 AdaBoost

Input: Training set $S = \{\mathbf{x}_i, y_i\}$, $i = 1, \ldots, N$; and $y_i \in \{-1, +1\}$; T: Number of iterations; I: Weak learner

Output: Boosted classifier: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$ where h_t, α_t are the induced classifiers (with $h_t(x) \in \{-1, 1\}$) and their assigned weights, respectively

1: $D_1(i) \leftarrow 1/N$ for $i = 1, \ldots, N$
2: **for** $t = 1$ to T **do**
3: $\quad h_t \leftarrow I(S, D_t)$
4: $\quad \varepsilon_t \leftarrow \sum_{i, y_i \neq h_t(\mathbf{x}_i)} D_t(i)$
5: $\quad$ **if** $\varepsilon_t > 0.5$ **then**
6: $\quad\quad T \leftarrow t - 1$
7: $\quad\quad$ **return**
8: $\quad$ **end if**
9: $\quad \alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
10: $\quad D_{t+1}(i) = D_t(i) \cdot e^{(-\alpha_t h_t(\mathbf{x}_i) y_i)}$ for $i = 1, \ldots, N$
11: $\quad$ Normalize D_{t+1} to be a proper distribution
12: **end for**

La clase viene dada por +1 o -1 en vez del {0, 1} como en LR

Asignamos **pesos de manera uniforme**

Entrenamos el clasificador con pesos

Calculamos el **error** como la suma de los pesos de los ejemplos mal clasificados

Condición de parada si el clasificador es peor que uno aleatorio

Peso para el clasificador

Actualización de pesos

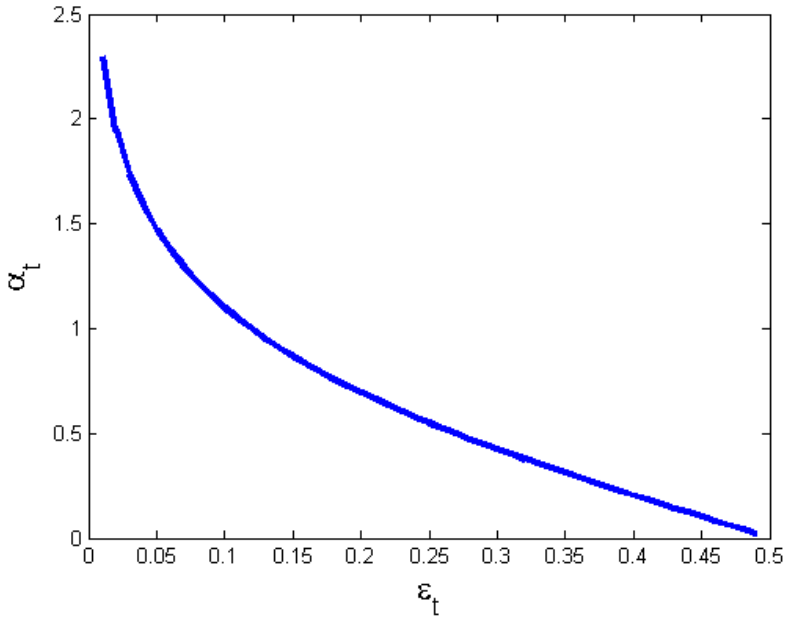
¡¡Minimizamos la función de coste exponencial!!

Boosting

- **Peso para cada clasificador**
  - **El error del clasificador, ε_t, es la suma de los pesos de los ejemplos mal clasificados**
  - **El peso, α_t, está relacionado con el error**

$$\varepsilon_t \leftarrow \sum_{i, y_i \neq h_t(\mathbf{x}_i)} D_t(i)$$

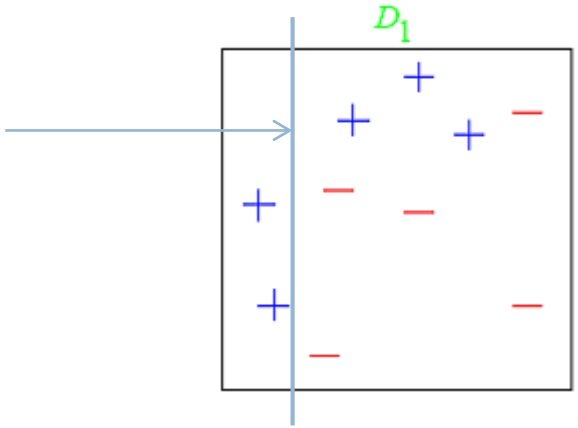
$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$



Boosting

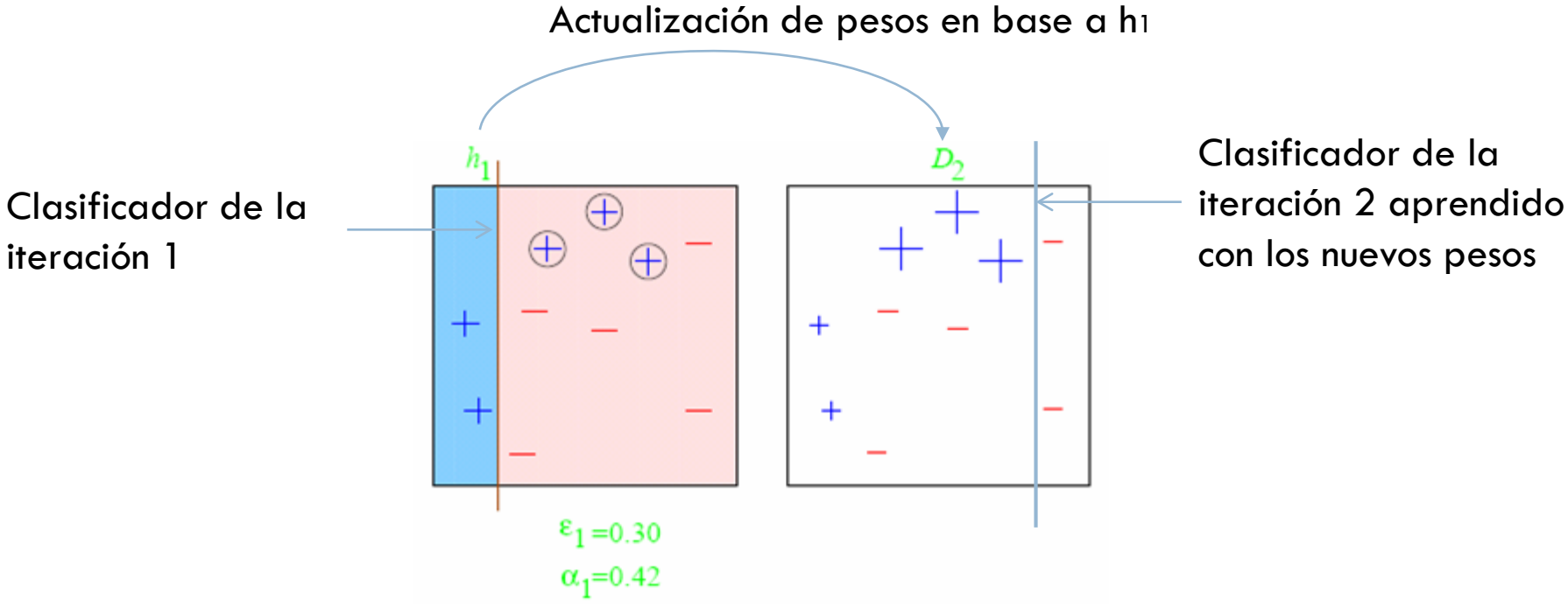
- **Ejemplo de clasificación**
  - El **tamaño** de los ejemplos refleja su **peso**
  - Utilizamos **Decision Stumps** como clasificador base
    - Los veremos más adelante
  - Problema inicial

Clasificador de la iteración 1



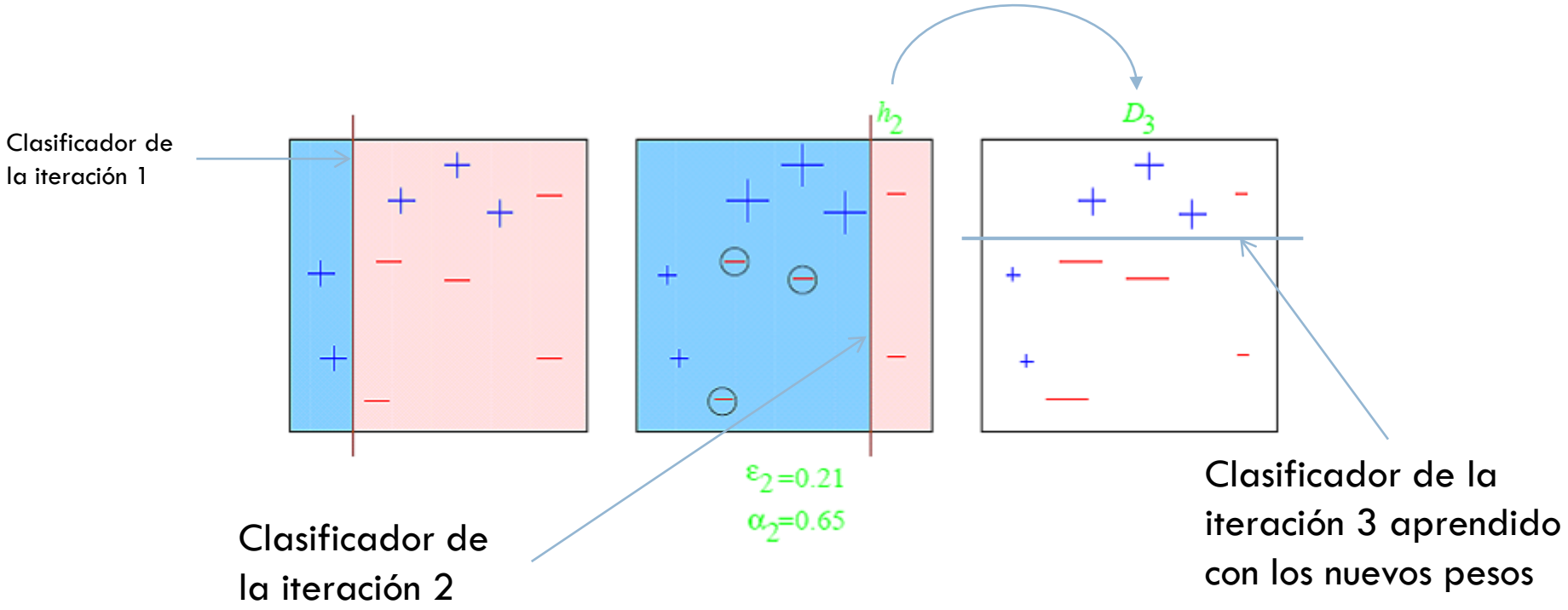
Boosting

- **Ejemplo de clasificación**
  - Iteración 1 y 2



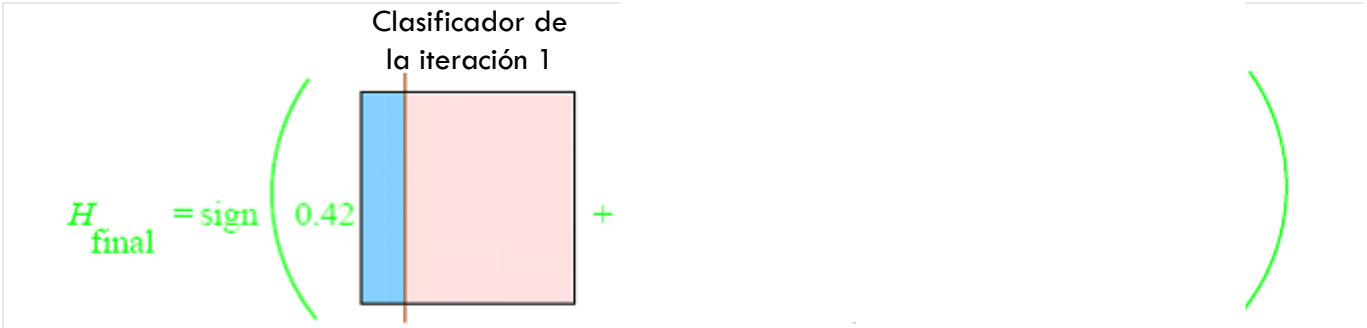
Boosting

- **Ejemplo de clasificación**
  - Iteración 2 y 3



Boosting

- **Ejemplo de clasificación**
  - Fin tras iteración 3 → Combinación



Combinación en base
a los pesos obtenidos

Hemos convertido clasificadores débiles muy simples en un clasificador fuerte capaz de solucionar el problema

Boosting en Python

- Paquete

  from sklearn.ensemble import AdaBoostClassifier

- Método de consutrcción del clasificador

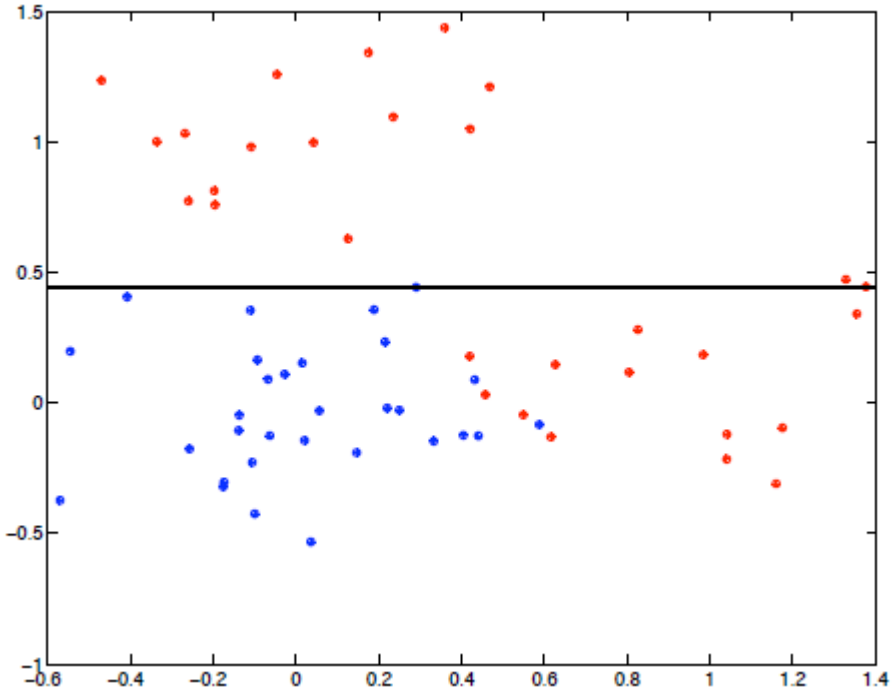
  AdaBoostClassifier(base_estimator= Clasificador, n_estimators= numeroClasificadores, random_state=semilla)

  - Parámetros
    - Clasificador: clasificador con el que conformar el ensemble
    - numeroClasificadores: número de clasificadores base
    - Semilla: número entero utilizado para evitar la aleatoriedad
- Entrenamiento y clasificación
  - Funciones ***fit*** y ***predict***, respectivamente
- Más información en:
  - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier

Índice

Decision Stumps

- **Árbol de decisión de un solo nivel**
  - Utiliza solo **un atributo para clasificar**
  - Solo tiene **dos ramas**



Decision Stump

- **Modelo**

$$h_{\theta,j,c}(x) = \begin{cases} c, & si \quad x_j > \theta \\ -c, & en\ otro\ caso \end{cases}$$

  - **Parámetro j**
    - **Atributo** a utilizar
  - **Parámetro θ**
    - **Umbral** sobre el atributo
  - **Parámetro c**
    - **Clase** con la que se clasifica si se supera el umbral

Decision Stump

- **Aprendizaje**
  - **Ejemplos con costes**
    - Cada ejemplo $(x^{(i)}, y^{(i)})$ tiene un peso $w^{(i)} \geq 0$
      - Coste de su mala clasificación
  - **Objetivo**
    - **Minimizar el error de clasificación ponderado**

$$J(\theta, j, c) = \frac{1}{m} \sum_{i=1}^{m} w_i I(h_{\theta,j,c}(x^{(i)}) \neq y^{(i)})$$

donde $I(\cdot)$ es la función indicatriz, que toma valor 1 si se cumple la condición, 0 en otro caso

Decision Stump

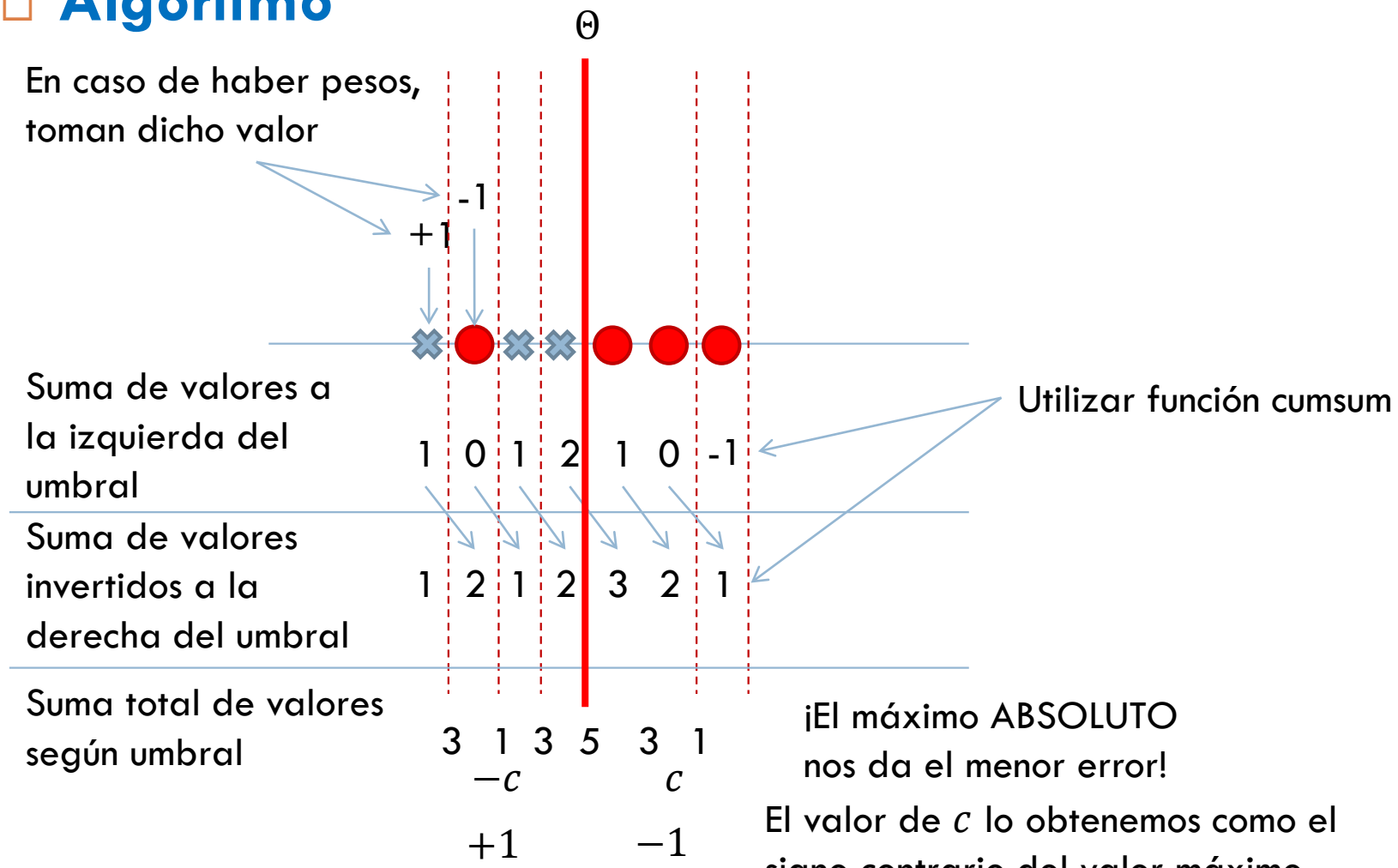
- **Aprendizaje**
  - **Para cada atributo j**
    - **Encontrar el punto de corte θ con menor error**

$$J(\theta, j, c) = \frac{1}{m} \sum_{i=1}^{m} w_i I(h_{\theta,j,c}(x^{(i)}) \neq y^{(i)})$$

  - **Elegir el atributo cuya clasificación obtiene el menor error**

Decision Stump

- **Algoritmo**



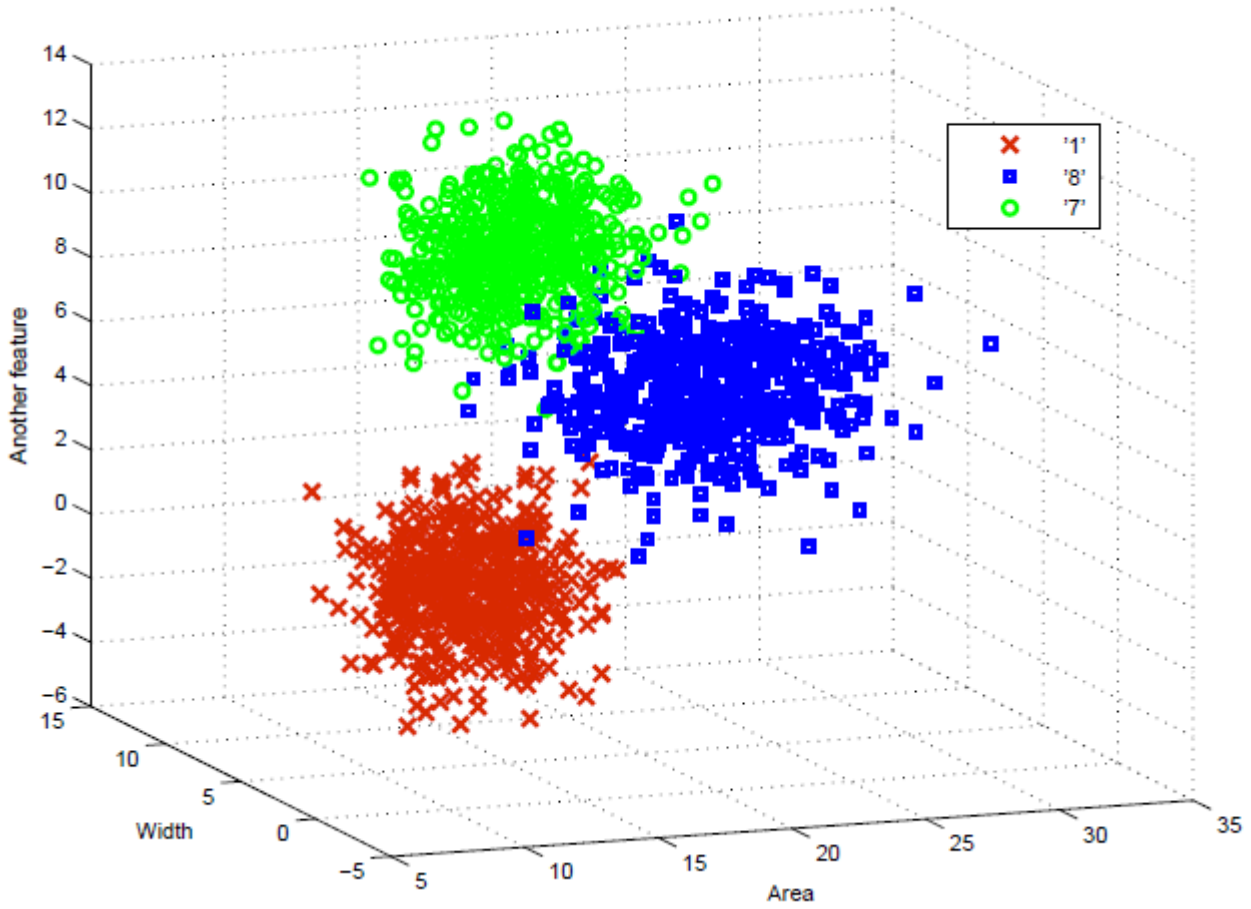
Índice

Ensembles basados en descomposición

- **Objetivo**
  - Afrontar problemas multi-clase con clasificadores binarios
- **Funcionamiento**
  - Descomponer un problema multi-clase
    - En **problemas binarios** (más sencillos de resolver)
  - Aprender **un clasificador para cada subproblema**
  - Para clasificar una **nueva instancia**
    - **Agregar las salidas** de los clasificadores base

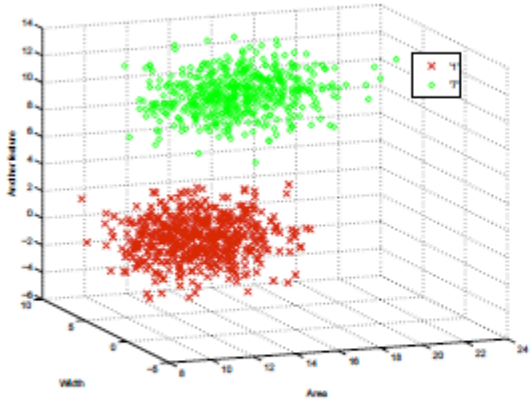
Ensembles basados en descomposición

- **Ejemplo de problema multi-clase**

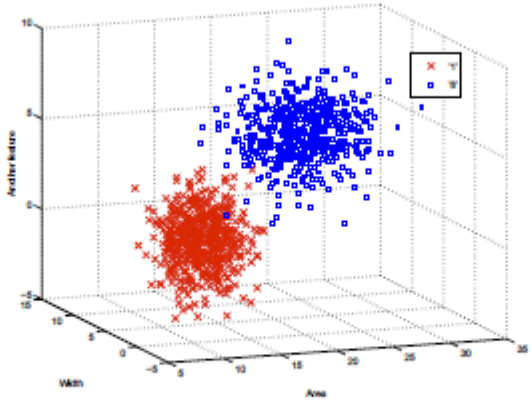


One-vs-One (OVO)

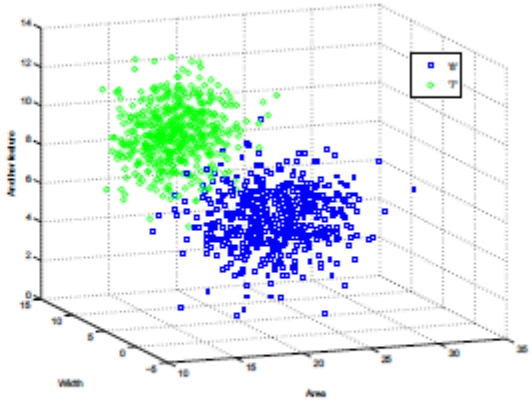
- **Divide el problema multi-clase en tantos problemas como pares de clases**



(a) '1' vs. '7'

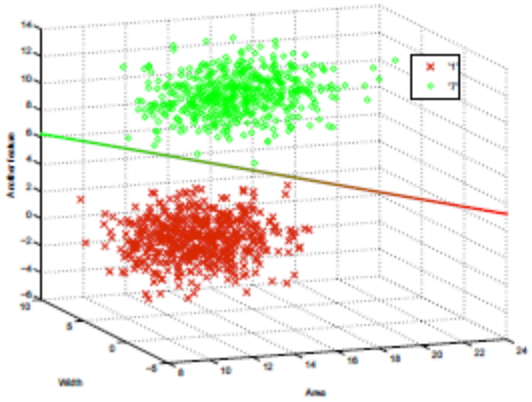


(b) '1' vs. '8'

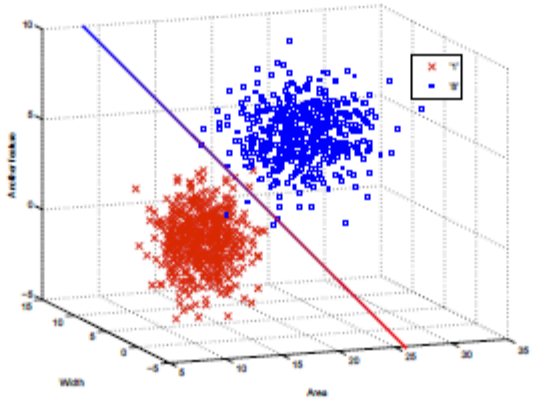


(c) '8' vs. '7'

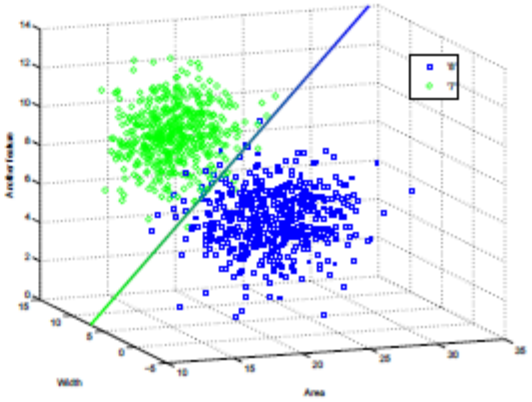
One-vs-One (OVO)



(a) '1' vs. '7'



(b) '1' vs. '8'



(c) '8' vs. '7'

One-vs-One (OVO)

- **Representación de las salidas**
  - **Matriz de votos**
  - $r_{ij} \in [0,1]$ confianza a favor de la clase i frente a la j
  - $r_{ji} = 1 - r_{ij}$

$$R = \begin{pmatrix} - & r_{12} & \cdots & r_{1m} \\ r_{21} & - & \cdots & r_{2m} \\ \vdots & & & \vdots \\ r_{m1} & r_{m2} & \cdots & - \end{pmatrix}$$

One-vs-One (OVO)

- **Agregaciones más comunes**
  - **Voto**
    - Cada clasificador vota por la clase predicha

$$Class = \arg \max_{i=1,\ldots,m} \sum_{1 \le j \ne i \le m} s_{ij},$$

    - donde s_{ij} es 1 si $r_{ij} > r_{ji}$ y 0 en otro caso
  - **Voto ponderado**
    - Cada clasificador vota con cierta confianza a cada clase

$$Class = \arg \max_{i=1,\ldots,m} \sum_{1 \le j \ne i \le m} r_{ij}$$

  - En ambos casos, la clase con mayor confianza es la que se predice

OVO en Python

- Paquete

  from sklearn.multiclass import OneVsOneClassifier

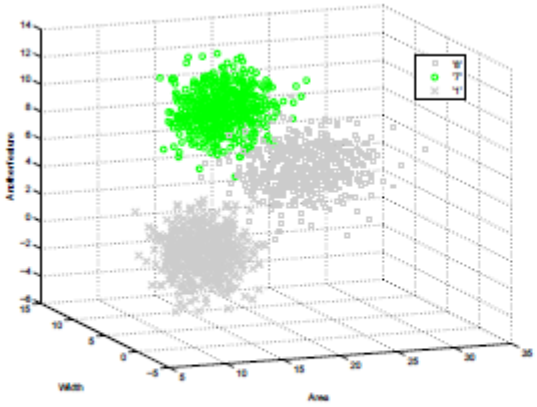
- Método de consutrcción del clasificador

  OneVsOneClassifier(Clasificador)

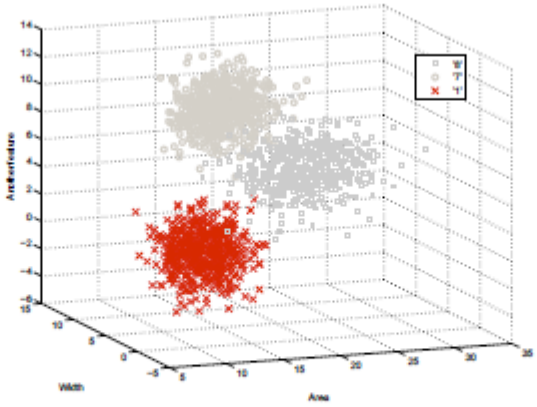
  - Parámetros
    - Clasificador: clasificador con el que conformar el ensemble
- Entrenamiento y clasificación
  - Funciones ***fit*** y ***predict***, respectivamente
- Más información en:
  - http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsOneClassifier.html

One-vs-All (OVA)

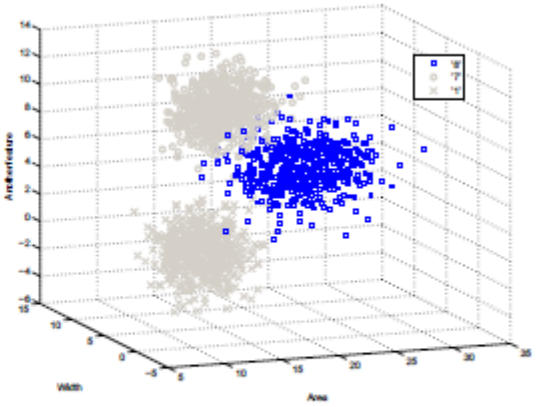
- **Divide el problema multi-clase en tantos problemas como clases**



(a) '7' vs. '1' and '8'

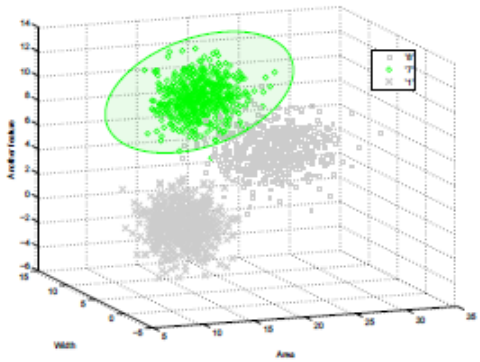


(b) '1' vs. '7' and '8'

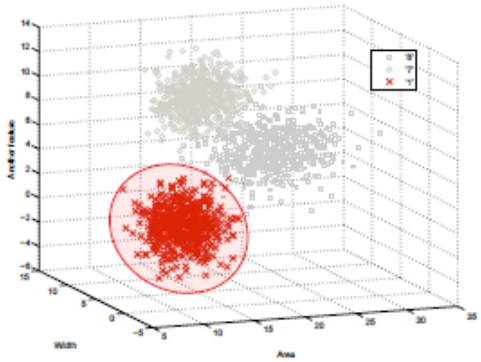


(c) '8' vs. '1' and '7'

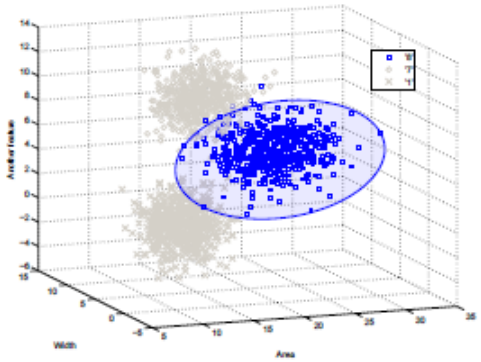
One-vs-All (OVA)



(a) '7' vs. '1' and '8'



(b) '1' vs. '7' and '8'



(c) '8' vs. '1' and '7'

One-vs-All (OVA)

- **Representación de las salidas**
  - **Vector de votos**
  - $r_i \in [0,1]$ confianza a favor de la clase i

$$R = (r_1, r_2, \ldots, r_i, \ldots, r_m)$$

  - **Agregación**
    - **Máximo**
      - Clase de salida = clase con mayor confianza

OVA en Python

- Paquete

  from sklearn.multiclass import OneVsRestClassifier

- Método de consutrcción del clasificador

  OneVsRestClassifier(Clasificador)

  - Parámetros
    - Clasificador: clasificador con el que conformar el ensemble
- Entrenamiento y clasificación
  - Funciones ***fit*** y ***predict***, respectivamente
- Más información en:
  - http://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html#sklearn.multiclass.OneVsRestClassifier

OVO vs. OVA

- **Ventajas OVO**
  - Problemas **más sencillos**
  - Problemas **más pequeños**
  - Computacionalmente **más rápido**
  - Generalmente, **más preciso**
- **Desventajas OVO**
  - **Región no clasificable** (Voto)
  - Clasificadores **no competentes**

OVO vs. OVA

- **Ventajas OVA**
  - Utiliza **todos los ejemplos**
    - **No hay** clasificadores **no competentes**
  - **Agregaciones más simples**
- **Desventajas OVA**
  - Problemas **no balanceados**
  - Problemas **más complejos**
  - Computacionalmente **más costoso**