Conjunto de clasificadores

gonzalo.martinez@uam.es

Guión

- ¿Qué es un conjunto de clasificadores? ¿Cómo construirlo?
- Bagging, Boosting, Random forests, class-switching, xgBoost
- Combinar salida
- *Stacking*
- Otras técnicas
- ¿Por qué funcionan? Historias de éxito

Teorema del jurado de Condorcet

- Alcanzar decisiones mediante la discusión de opiniones forma parte del ser humano

- Se formaliza en el teorema del jurado de Condorcet que dice que dado un jurado y suponiendo:

    1. que comenten errores independientes

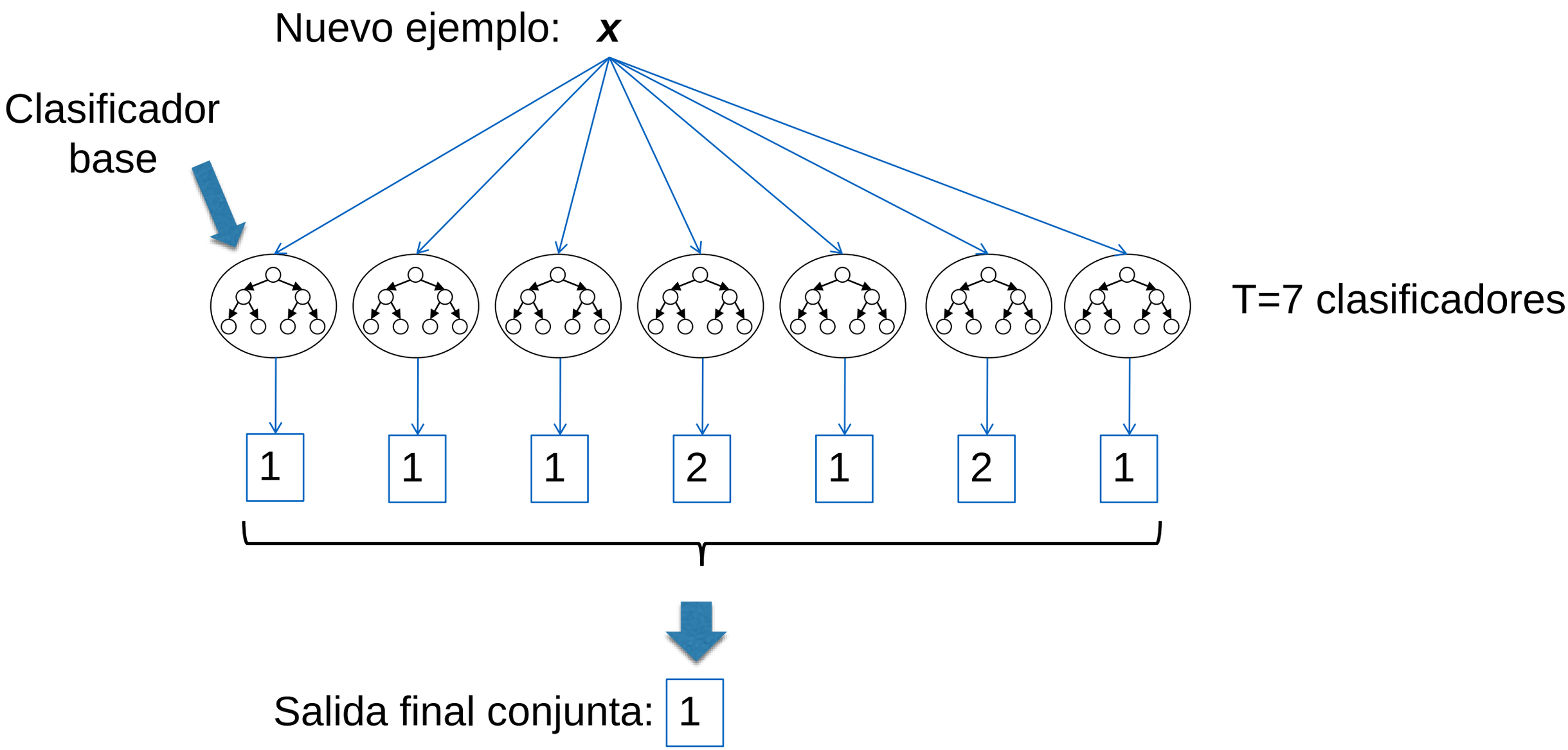
    2. que la probabilidad de cada miembro del jurado de acertar es superior al 50%

La probabilidad del jurado de acertar tiende a 100% al aumentar el numero de miembros del jurado

Nicolas de Condorcet (1743-1794), French mathematician

¿Qué es un conjunto de clasificadores?

- Es una combinación de clasificadores que dan una salida final conjunta.



Idea general

- Generar muchos clasificadores y combinarlos para dar una clasificación final
- Funcionan muy bien. En general, mejor que cualquiera de los modelos que los componen
- Los clasificadores a combinar deben ser distintos
- La clave es la generación de clasificadores diversos a partir de los datos disponibles

¿Cómo construirlos?

- Hay varias técnicas para construir un conjunto de clasificadores diversos:
  - Utilizar distintas versiones modificadas aleatoriamente de los datos de entrenamiento para crear cada clasificador base
  - Inyectar cambios en los algoritmos de aprendizaje que introduzcan aleatorización de los modelos
- Estas estrategias se pueden usar en combinación.
- Generalmente, cuanto mayor sea la aleatorización, mejores serán los resultados

¿Cómo construirlos?

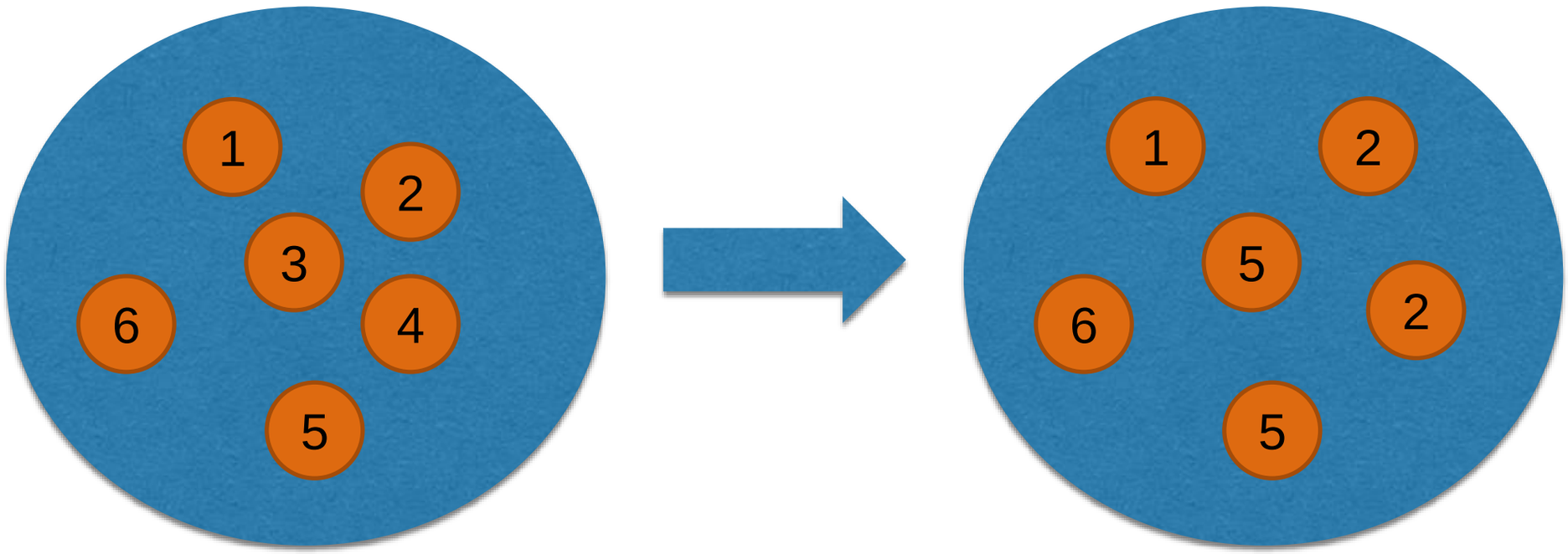
- Modificaciones aleatorias de los datos se pueden generar mediante:
  - Remuestreo de los datos. Por ejemplo por bootstrap (p.e. en bagging), muestreo ponderado (p.e en boosting).
  - Alteración de atributos: Los clasificadores base se entrenan usando distintos subconjuntos de atributos (p.e. en Random subspaces)
  - Modificando las clases: Agrupando clases en dos nuevas superclases aleatoriamente (p.e. ECOC) o modificando las etiquetas aleatoriamente (p.e. Class-switching)

¿Cómo construirlos?

- Aleatorizando los algoritmos de aprendizaje
  - Introducir cierta alatorización en los algoritmos de aprendizaje, de forma que dos ejecuciones consecutivas den clasificadores distintos
  - Ejecutando clasificadores base distintos o con distintos parámetros, etc.

Muestra *bootstrap* (inciso)

- Dado un conjunto de N elementos, una muestra *bootstrap* consiste en extraer N elementos con reemplazamiento.



6 extracciones con repetición

Bagging

Input:

Dataset D = ($\mathbf{X_i}$, y_i) i=1...N

Ensemble size T

```
1.for t=1 to T:
2.    sample = BootstrapSample(D)
3.    h_t = TrainClassifier(sample)
```

Bootstrap

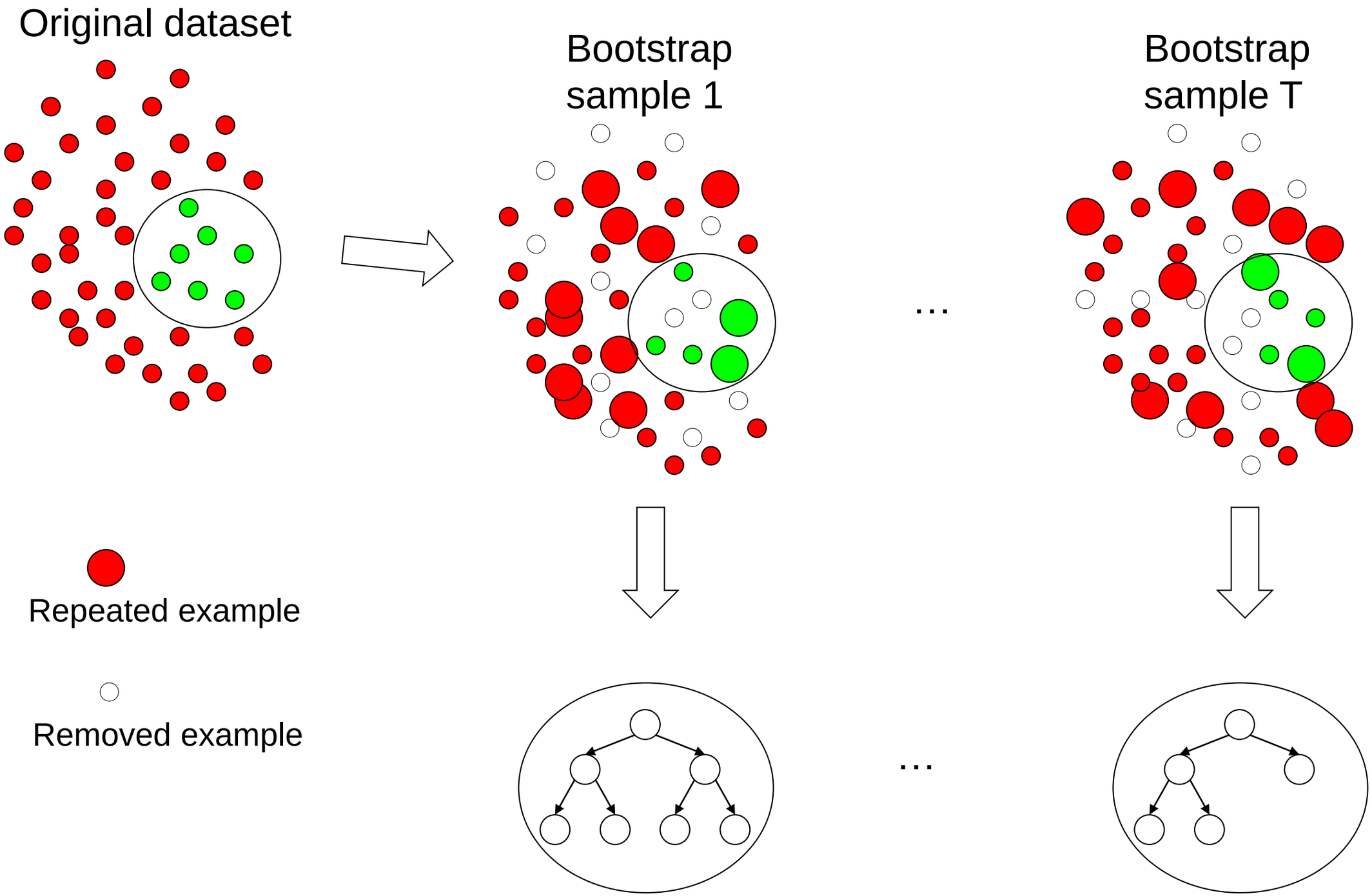
+

Output:

$$H(\mathbf{x}) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} I\left(h_t(\mathbf{x}) = j\right) \right)$$

Agregación (por voto)

Bagging


Original dataset
Bootstrap sample 1
...
Bootstrap sample T
Repeated example
Removed example
...

Consideraciones sobre bagging

- Use un 63,2% de los datos de entrenamiento en media para construir cada clasificador.
- Muy robusto frente a ruido en las etiquetas de clase.
- En general, mejora el rendimiento del clasificador base.
- Se puede paralelizar fácilmente

Random forest

- Breiman define los bosques aleatorios (Random forests) como un conjunto de clasificadores tal que:
  - Tiene árboles de decisión como algoritmo base
  - Introduce aleatorización en el proceso de entrenamiento
- Bajo esta definición Bagging con árboles de decisión es un random forest y de hecho lo es. Sin embargo…

Random forest

- En la práctica, se considera que un random forest es:
  - Cada árbol se genera, como en bagging, usando muestras *bootstrap*
  - Los árboles se construyen de forma que cada división se calcula usando:
    - Un subconjunto aleatorio de los atributos
    - Se selecciona la mejor división de ese subconjunto de atributos
    - Se usan árboles sin podar

Consideraciones sobre random forests

- Su rendimiento es mejor que boosting en media. Y de hecho es uno de los mejores clasificadores.
- Es muy robusto al ruido (¡¡¡no sobre-ajusta!!!)
- Random forest introduce un mecanismo adicional de aleatorización con respecto a *bagging*
- Fácilmente paralelizable
- Los árboles aleatorios nos muy rápidos de construir

Abstract

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, Dinani Amorim; 15(Oct):3133−3181, 2014.

Boosting

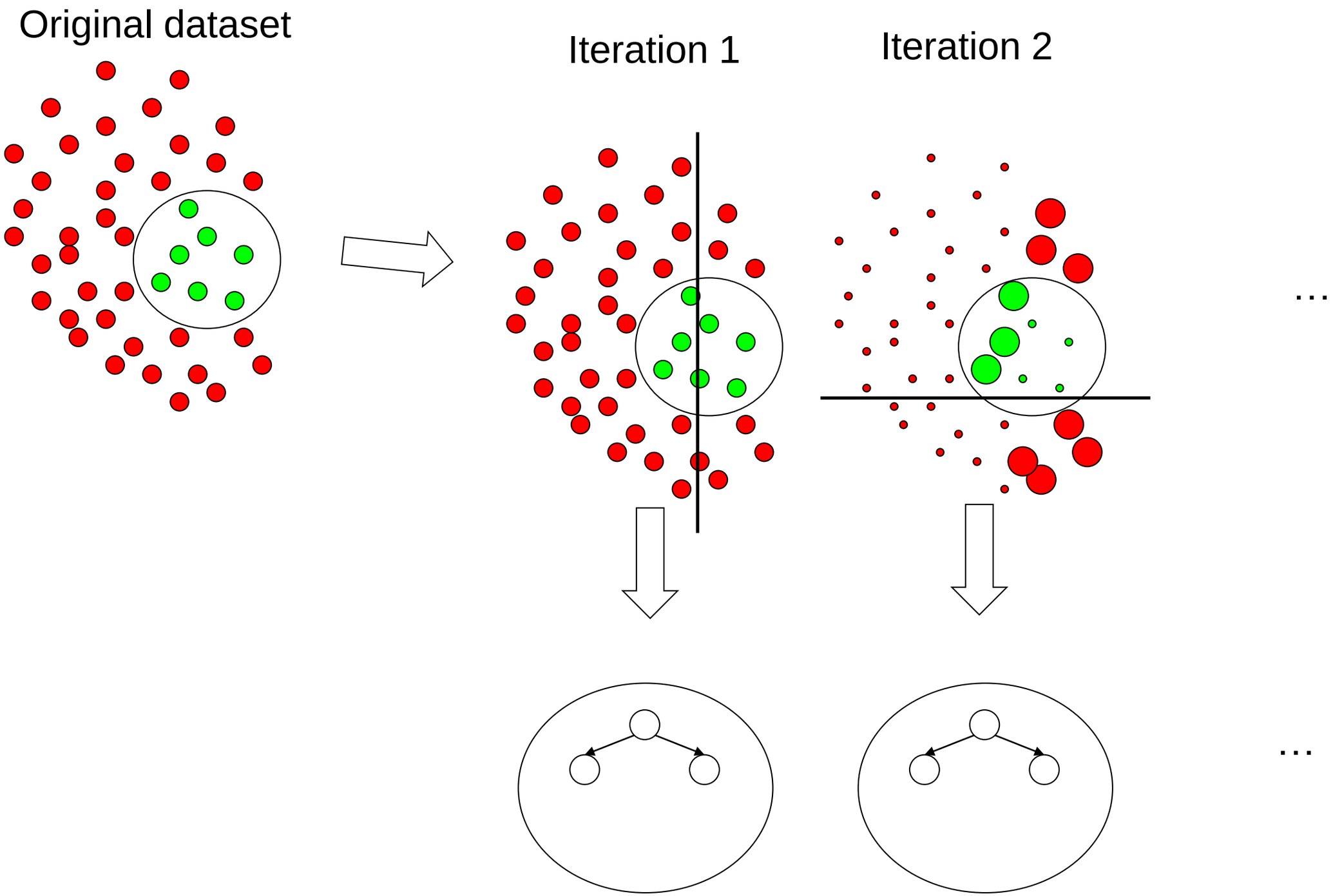
Entrada:

Datos D =($\mathbf{x}_i$, y_i) i=1...N

Tamaño del conjunto T

1. Asignar pesos a los ejemplos a *1/N*
2. **for** t=1 **to** T:
3. h_t = BuildClassifier(D, pesos)
4. e_t = WeightedError(D, pesos)
5. **if** e_t==*0* **or** $e_t \geq 0.5$ **break**
6. Multiplicar los pesos de los ejemplos mal clasificados h_t por $e_t/(1-e_t)$
7. Normalizar pesos

Boosting



Consideraciones sobre boosting

- Obtiene muy buenos resultados en general
- No es robusto frente a ruido en las etiquetas de clase
- Puede incrementar el error del clasificador base
- No se puede implementar fácilmente en paralelo

GradientBoosting

Entrada:

Datos D = {($\mathbf{x_i}$, y_i)} i=1...N
Tamaño del conjunto T
Función de pérdida L

1. $F_0(\mathbf{x}) = \underset{p}{\mathbf{argmin}} \sum_{i=1}^{N} L(y_t, p)$
2. **for** t=1 **to** T:
3. r_i = CalcularResiduos(F_{t-1} ($\mathbf{x_i}$), y_i)
4. h_t = TrainRegressor({($\mathbf{x_i}$, r_i)}$^N_{i=1}$)
5. $\rho_t = \underset{\rho}{\mathbf{argmin}} \sum_{i=1}^{N} L(y_i, F_{t-1}(\mathbf{x}_i) + \rho h_t(\mathbf{x}_i))$
6. $F_t(x) = F_{t-1}(x) + \rho_t h_t(x)$
7. return $F_T(x)$

GradientBoosting para pérdida cuadrática

Entrada:

Datos $D = \{(\mathbf{x_i}, y_i)\}$ $i=1...N$

Tamaño del conjunto T

Función de pérdida $L = (F(x)-y)^2$

1. F_0 = average(**y**) // media del objetivo
2. **for** $t=1$ **to** T:
3. $\quad r_i = y_i - F_{t-1}(\mathbf{x_i})$ para $i=1...N$
4. $\quad h_t$ = TrainRegressor($\{(\mathbf{x_i}, r_i)\}_{i=1}^{N}$)
5. $\quad \rho_t = 1$
5. $\quad F_t(x) = F_{t-1}(x) + \rho_t h_t(x)$
6. return $F_T(x)$

Variantes de Gradient boosting

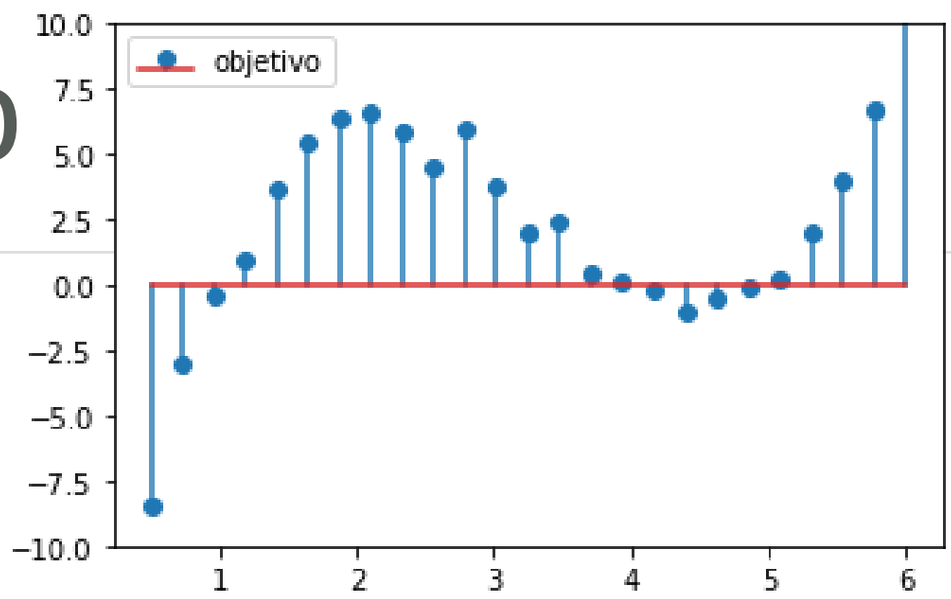
- Tasa de aprendizaje: la regla de adición cambia

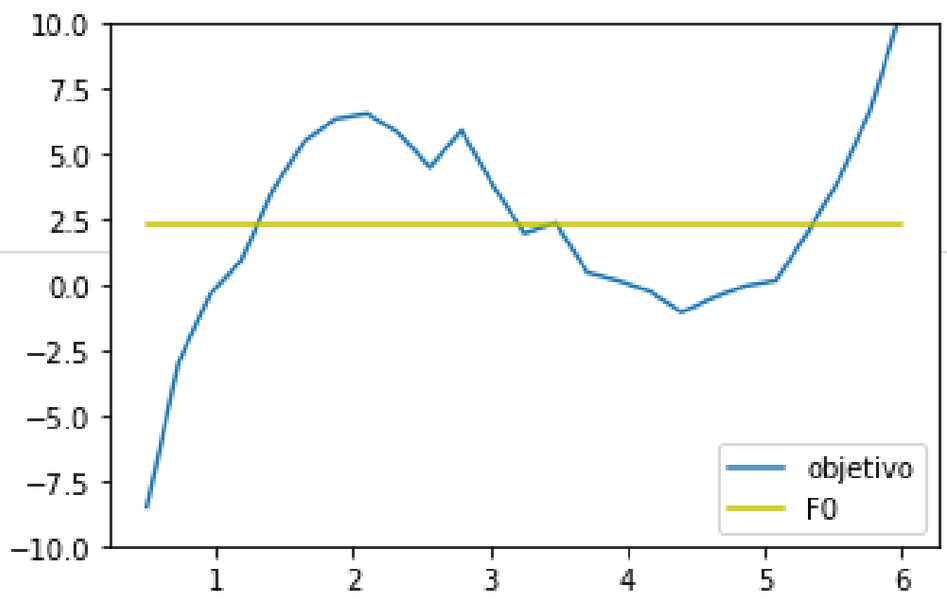
$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1$$

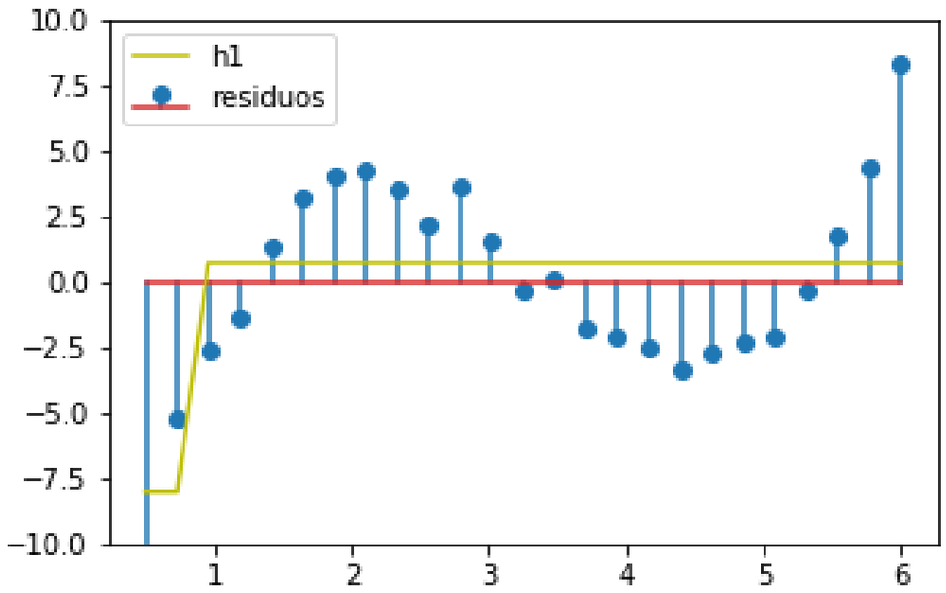
- Controlar la complejidad de los árboles: número de hojas, profundidad, etc.

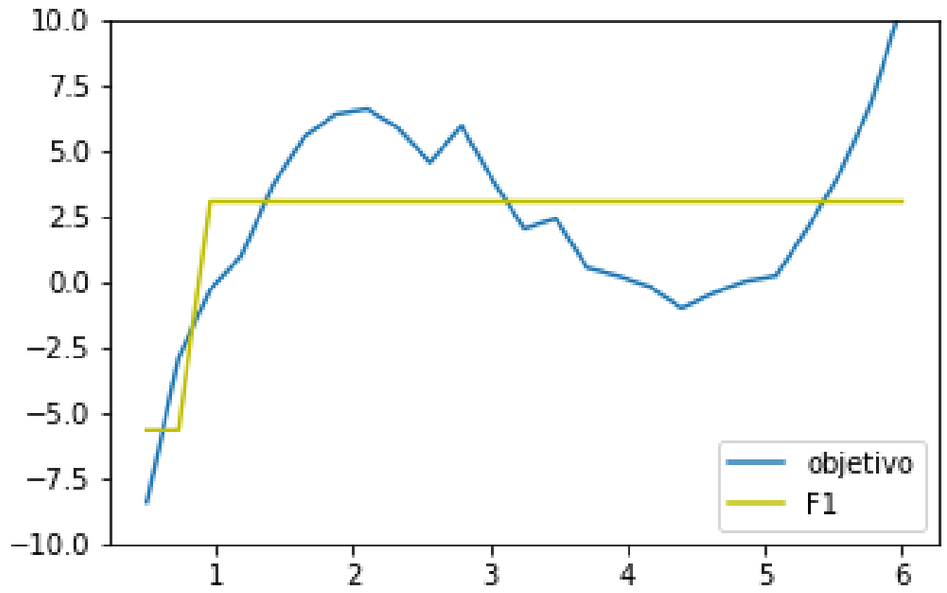
- Stochastic gradient boosting: Usar muestras bootstrap sin replazamiento para entrenar los modelos

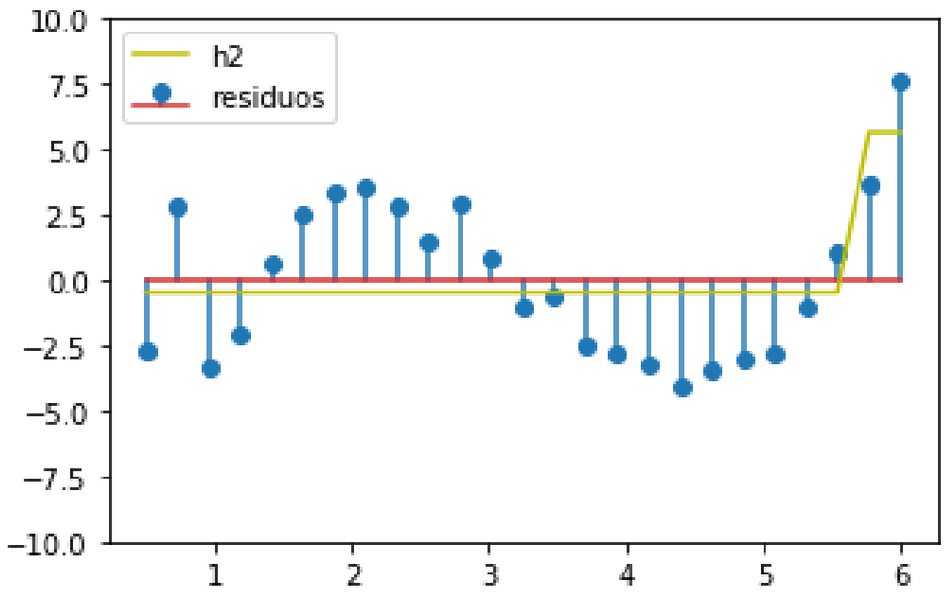
Ejemplo GB

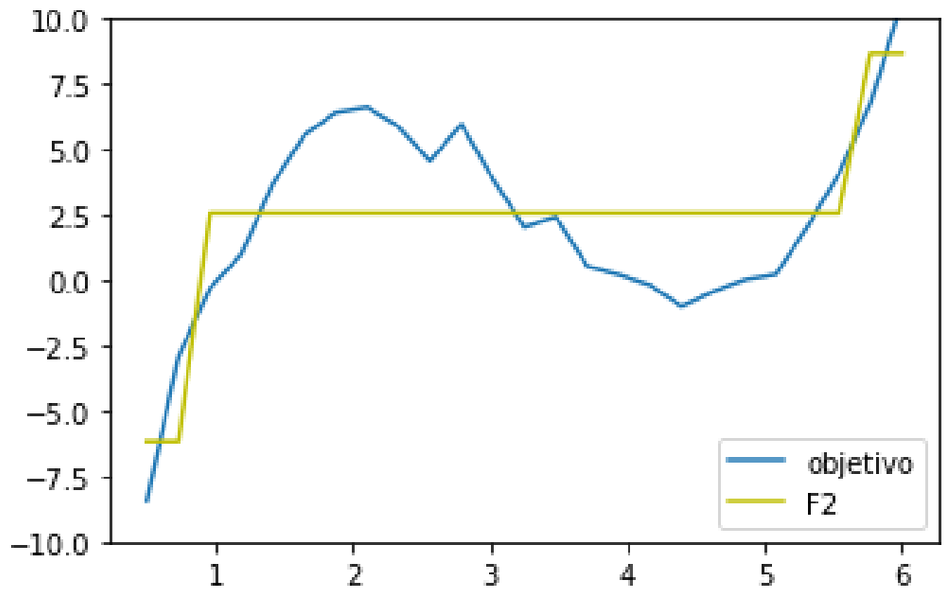

objetivo


objetivo
F0


h1
residuos


objetivo
F1


h2
residuos


objetivo
F2

Consideraciones sobre GradientBoosting

- Obtiene muy buenos resultados en general
- Casi siempre se combina con árboles de decisión sencillos (de poca profundidad)
- Muchos parámetros a ajustar lo que permite que funcione de forma más estable que boosting

XGBoost

- Un algoritmo basado en Gradient Boosting optimizado para ser mucho más rápido:
  - Permite trabajar con datos dispersos
  - Guarda los datos en bloques en un formato que evita tener que reordenar los atributos continuamente

- Incluye un término que penaliza la complejidad en la función de pérdida:

$$\mathbb{E}_{x,y}\left[L(y, F(x))\right] + \sum_{m=1}^{M} \Omega(f_m)$$

XGBoost

- Funcionalidades adicionales:
  - Función de pérdida con un término de penalización de la complejidad.
  - Como Gradient boosting tiene los parámetros:
    - Tasa de aprendizaje
    - Muestreo bootstrap
    - Límites en profundidas, número de hojas, etc.
- Como en random forest tiene muestreo por columnas

Consideraciones sobre XGBost

- Obtiene muy buenos resultados en general
- Tiene decenas de parámetros que hay que ajustar
- XGBoost está logrando los mejores resultados en competiciones de Kaggle.
- Fuente XGBoost: https://github.com/dmlc/xgboost

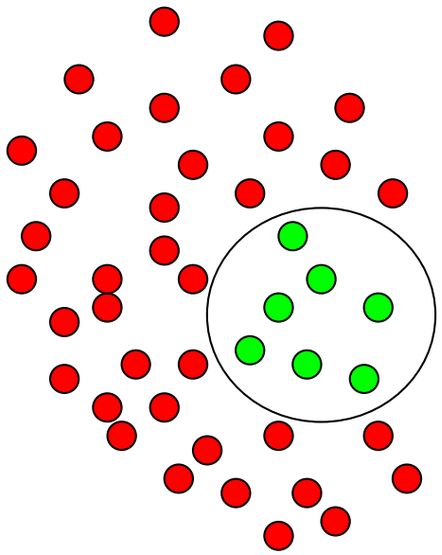
Class switching

- Class switching es un conjunto donde la diversidad se obtiene usando versiones del conjunto de entrenamiento contaminadas con ruido aleatorio en las etiquetas de clase.

- En concreto, los conjuntos de entrenamiento necesarios para construir cada clasificador base, se crean modificando la clase de cada punto por otra con probabilidad p.

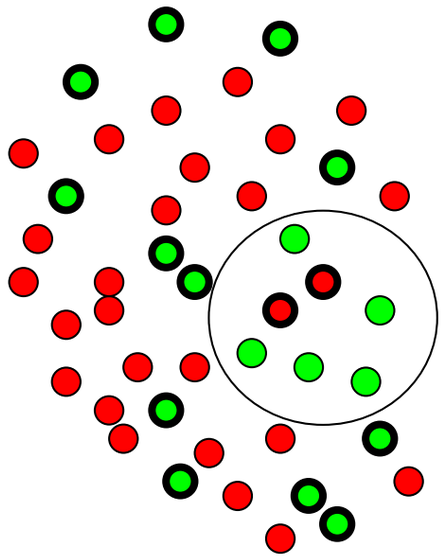
Class switching

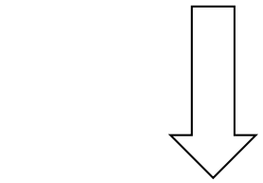
Original dataset

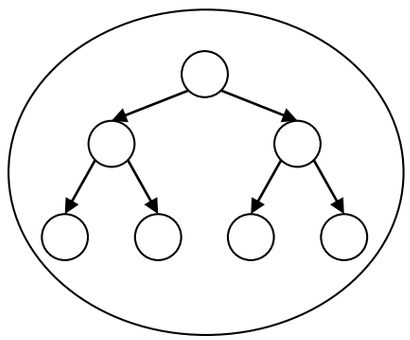


p=30%

Random noise 1

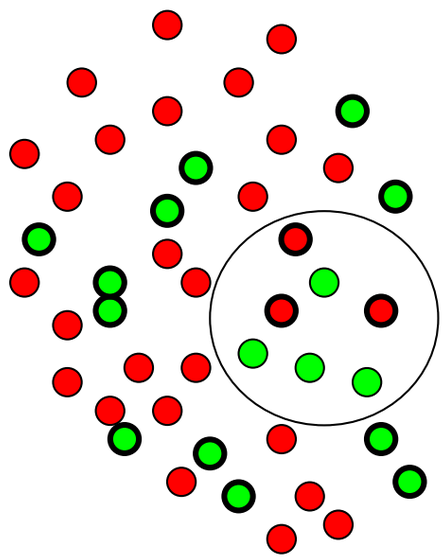


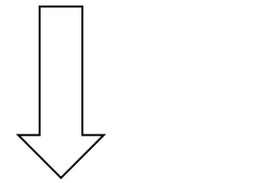


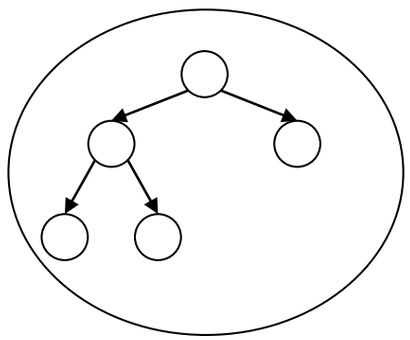


...

Random noise T

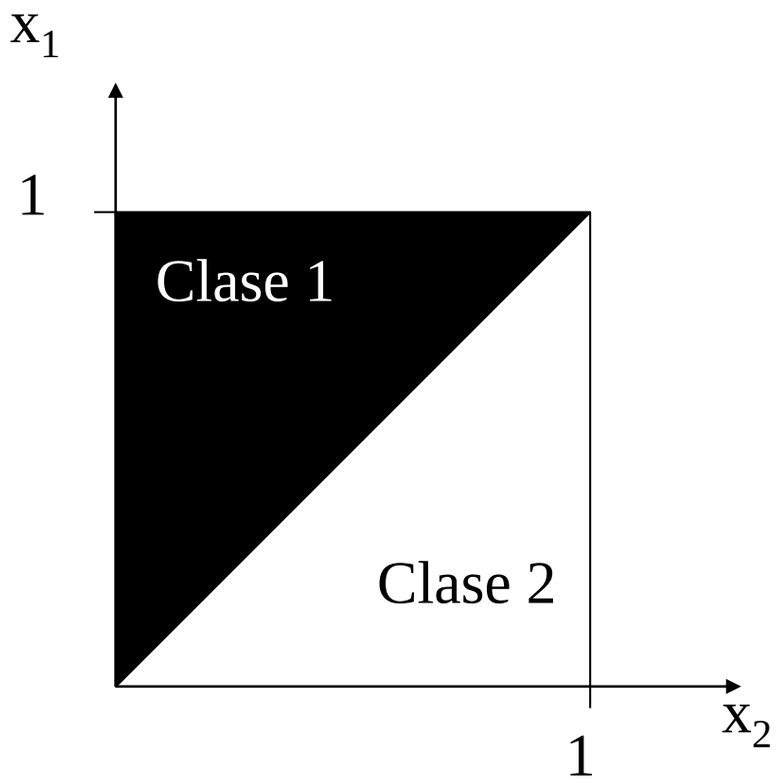






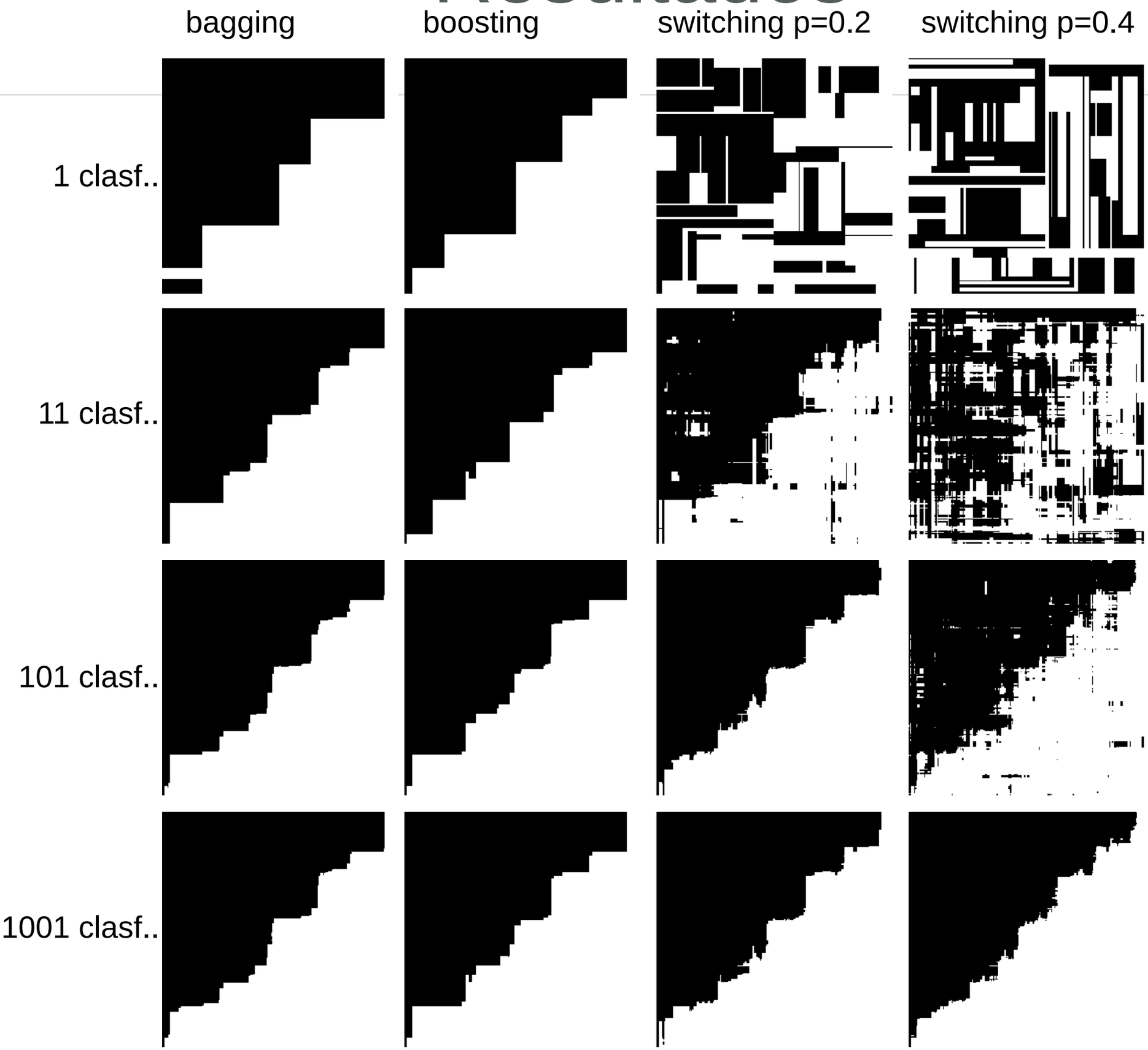
Ejemplo

- 2D ejemplo
- Frontera is x1=x2
- x1~U[0, 1] x2~U[0, 1]



- No es fácil para un árbol de decisión
- Probamos *bagging*, *boosting* y *class-switching* con p=0.2 y p=0.4

Resultados


bagging
boosting
switching p=0.2
switching p=0.4
1 clasf..
11 clasf..
101 clasf..
1001 clasf..

Parametrización

Parámetros utilizados

	Clasificadores base	Tamaño del ensemble T	Otros parámetros /opciones
Bagging	Árboles no podados	O(200) aunque cuantos más mejor	Usar submuestreo
Boosting	Árboles podados o Aprendices débiles	O(100)	
Random forest	Árboles no podados aleatorios	O(200) aunque cuantos más mejor	No. de atributos aleatorios para las divisiones = log(#features) o sqrt(#features)
Class-switching	Árboles no podados	O(1000)	% de ruido en las etiquetas, p~30%

Parametrización

Parámetros utilizados

	Clasificadores base	Tamaño del ensemble T	Otros parámetros /opciones
GradientBoosting	Árboles poco profundos	O(50-100)	•Submuestreo •Varios parámetros de complejidad de los árboles •Tasa de aprendizaje
XGBoost	Árboles poco profundos	O(50-100)	•Submuestreo •Varios parámetros de complejidad de los árboles •Tasa de aprendizaje •Muestreo de columnas •Y muchos más…

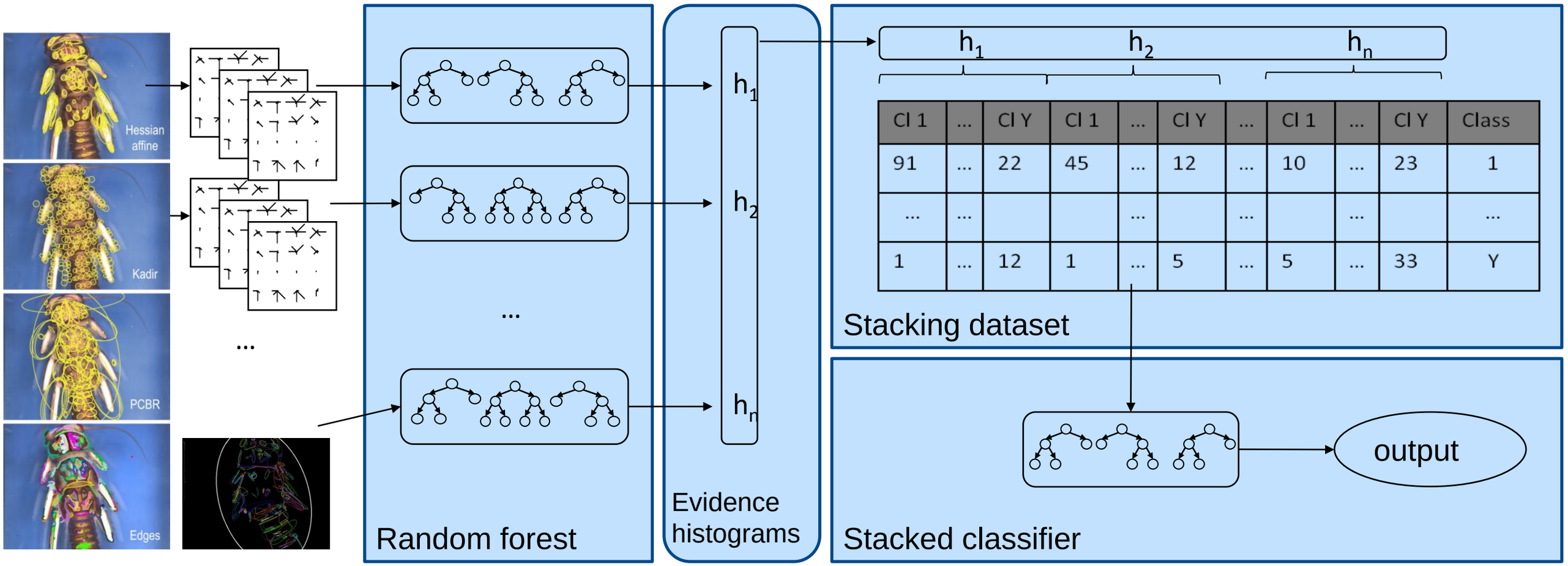
Combinar la salida

- Las técnicas de combinación se pueden dividir en dos grupos:
  - Estrategias de voto: La salida final del conjunto es la clase más votada por los clasificadores base. También puede ser voto ponderado
  - Otras estrategias: Operadores como máximo, mínimo, producto, mediana y media se pueden utilizar para combinar las salidas de los clasificadores base.
- No hay una estrategia ganadora. Depende de muchos factores

Stacking

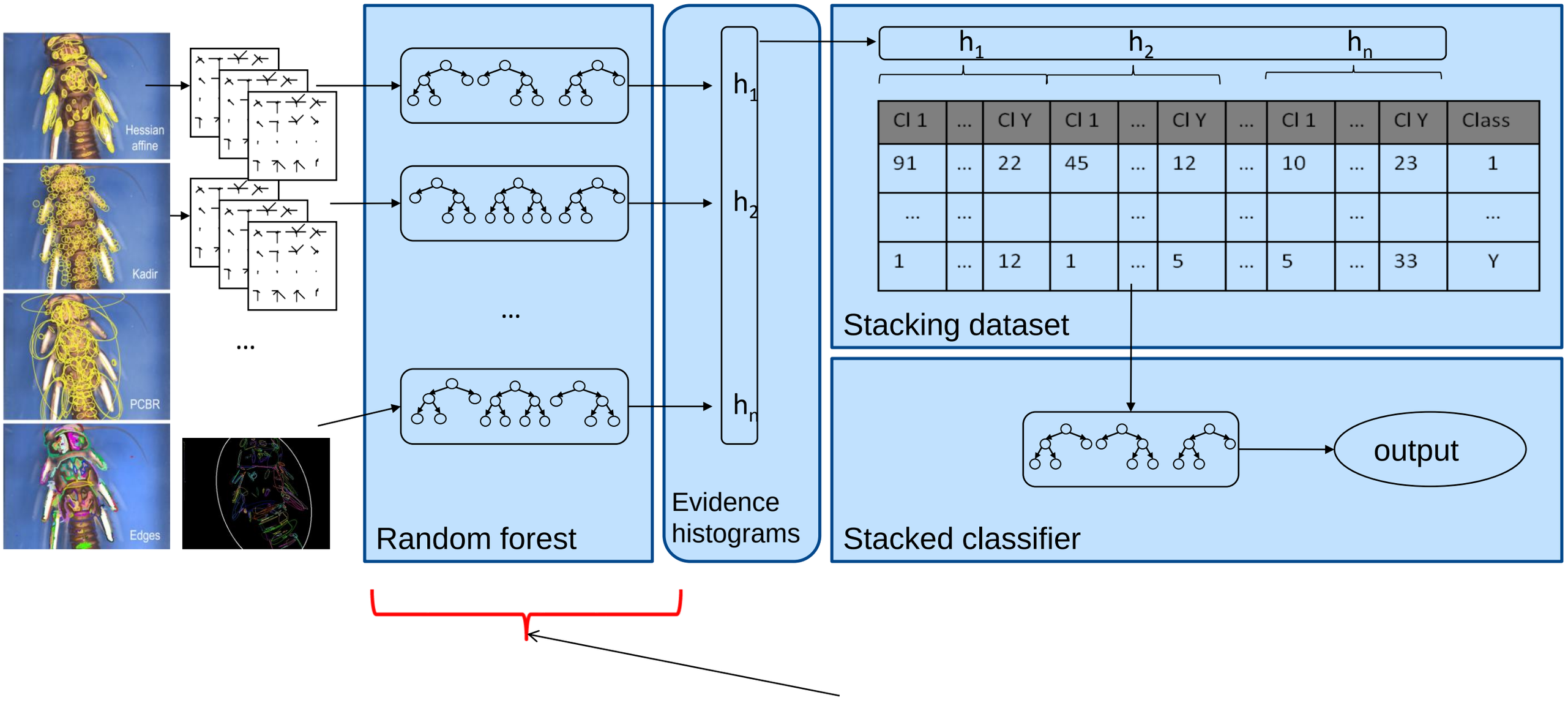
- En *stacking* la fase de combinación también se aprende.
- Primero construimos los clasificadores base usando unos datos de entrenamiento
- Después, las predicciones de estos clasificadores (usando otros datos) se usan como atributos para otro clasificador que aprende a combinarlas (meta-clasificador).

Ejemplo de stacking



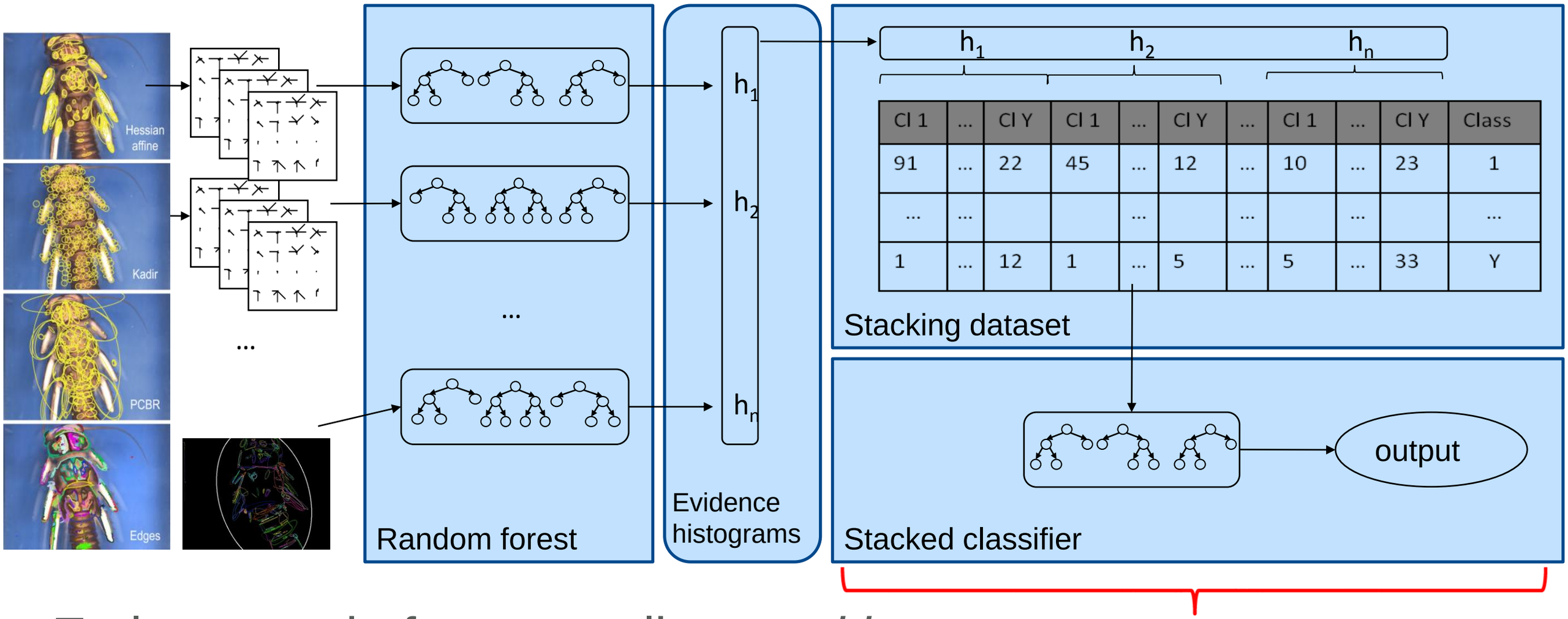
Extraer descriptores

Ejemplo de stacking



1. Se contruyen N Random forest en los descriptores

Ejemplo de stacking



- En la segunda fase se aplica *stacking*:
  - Las salidas de los conjuntos de la primera fase se concatenan
- Se aplica *boosting* a estos nuevos vectores para sacar la clasificación final.

Poda de conjuntos

1.- El orden aleatorio dado por *bagging*

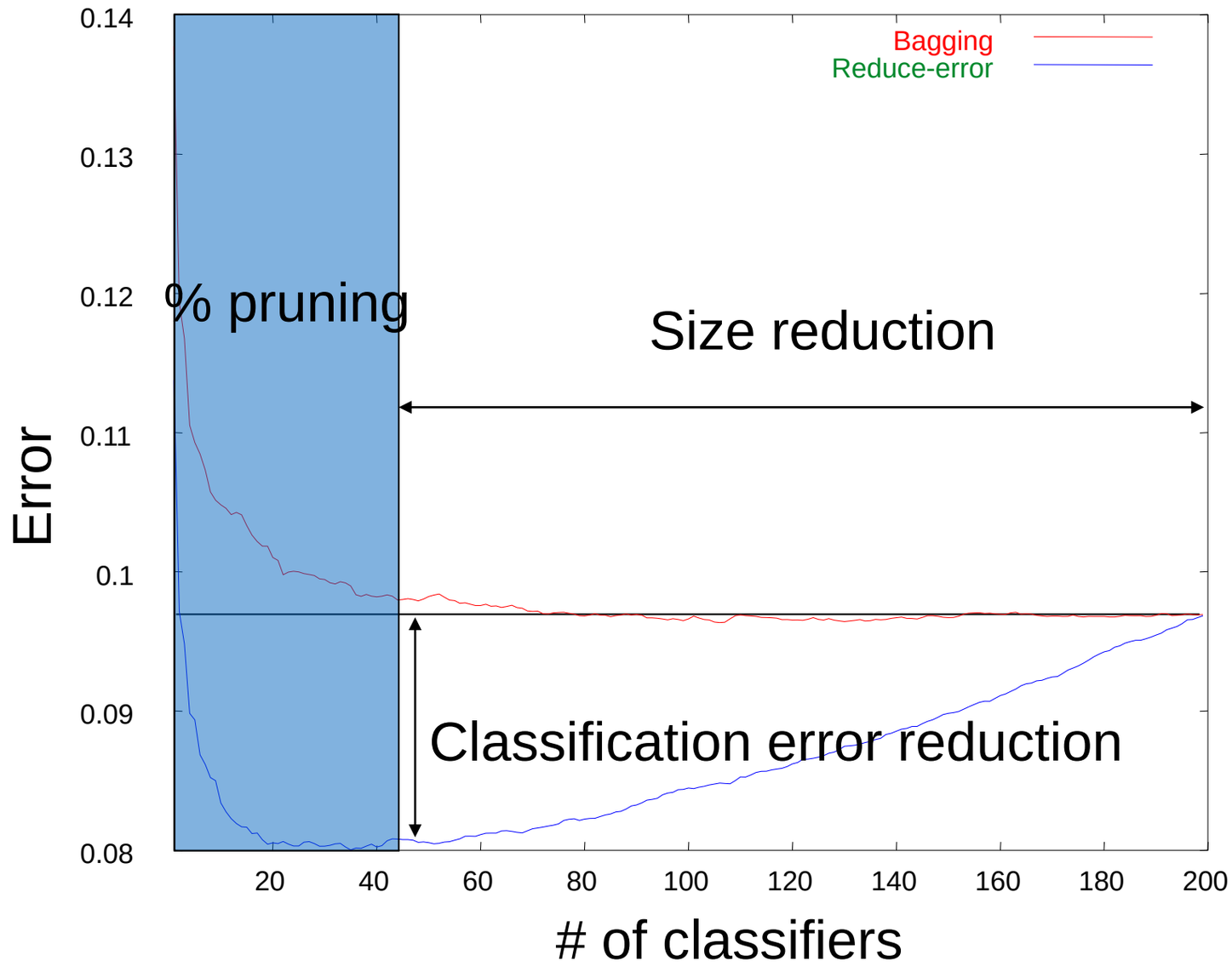
$h_1, h_2, h_3, .., h_T$

2.- Nueva ordenación

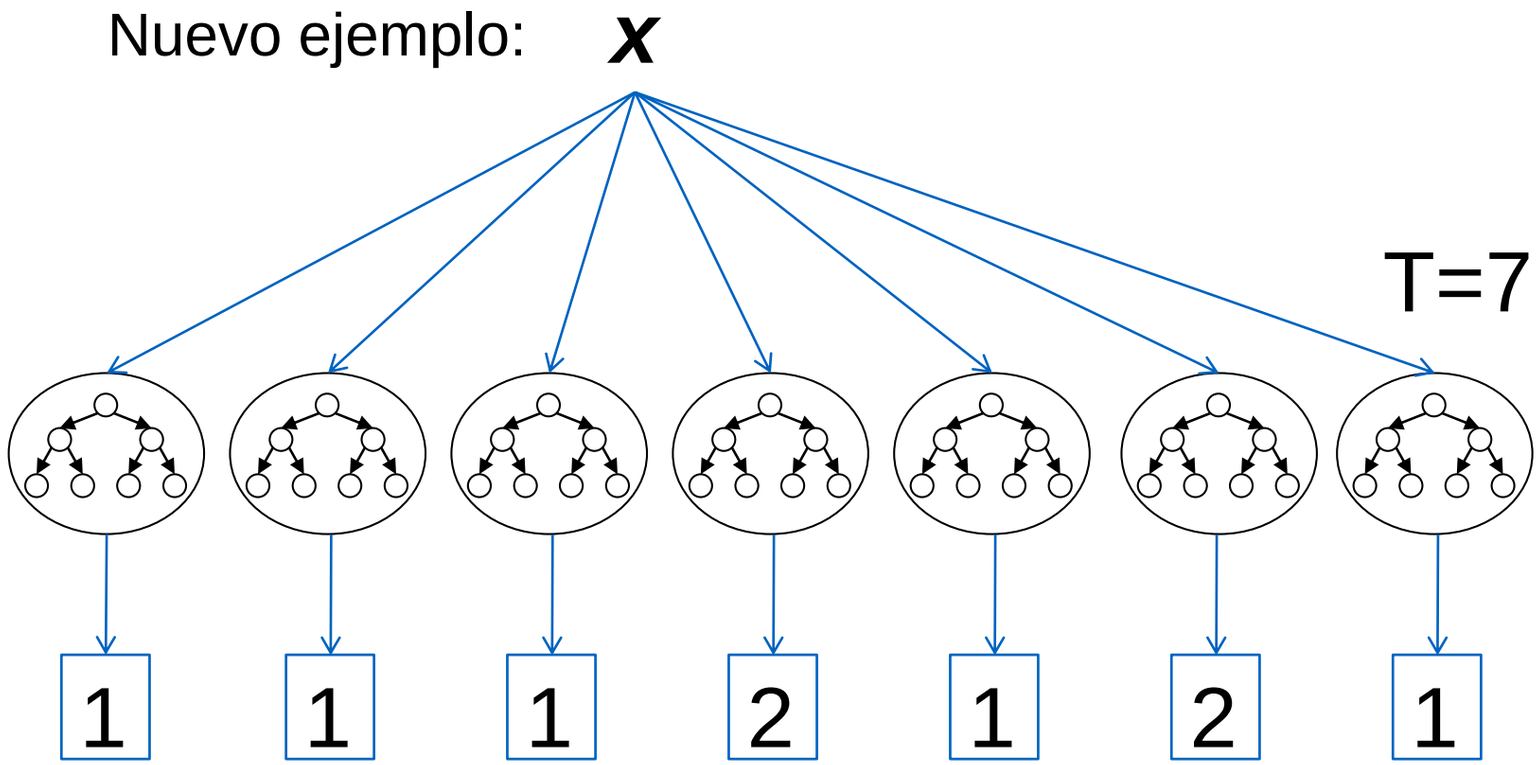
$h_{s1}, h_{s2}, \ldots, h_{sT}$

3.- Poda

$h_{s1}, \ldots h_{sM}$



Poda dinámica de conjuntos

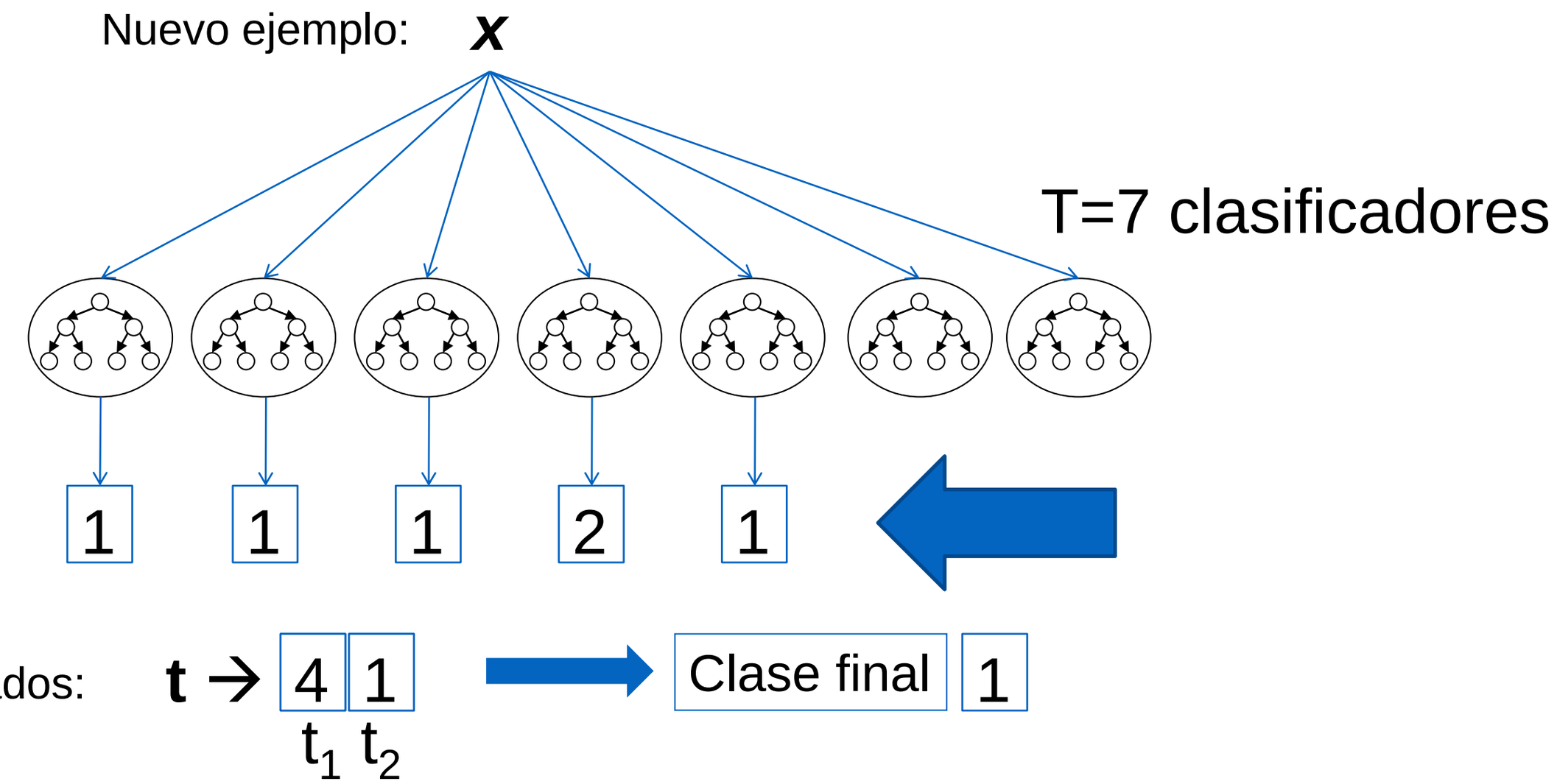


Votos acumulados: $\mathbf{t} \rightarrow$ [5] [2] (t_1 t_2) → Clase final [1]

¿Necesitamos preguntar a todos los clasificadores?

NO

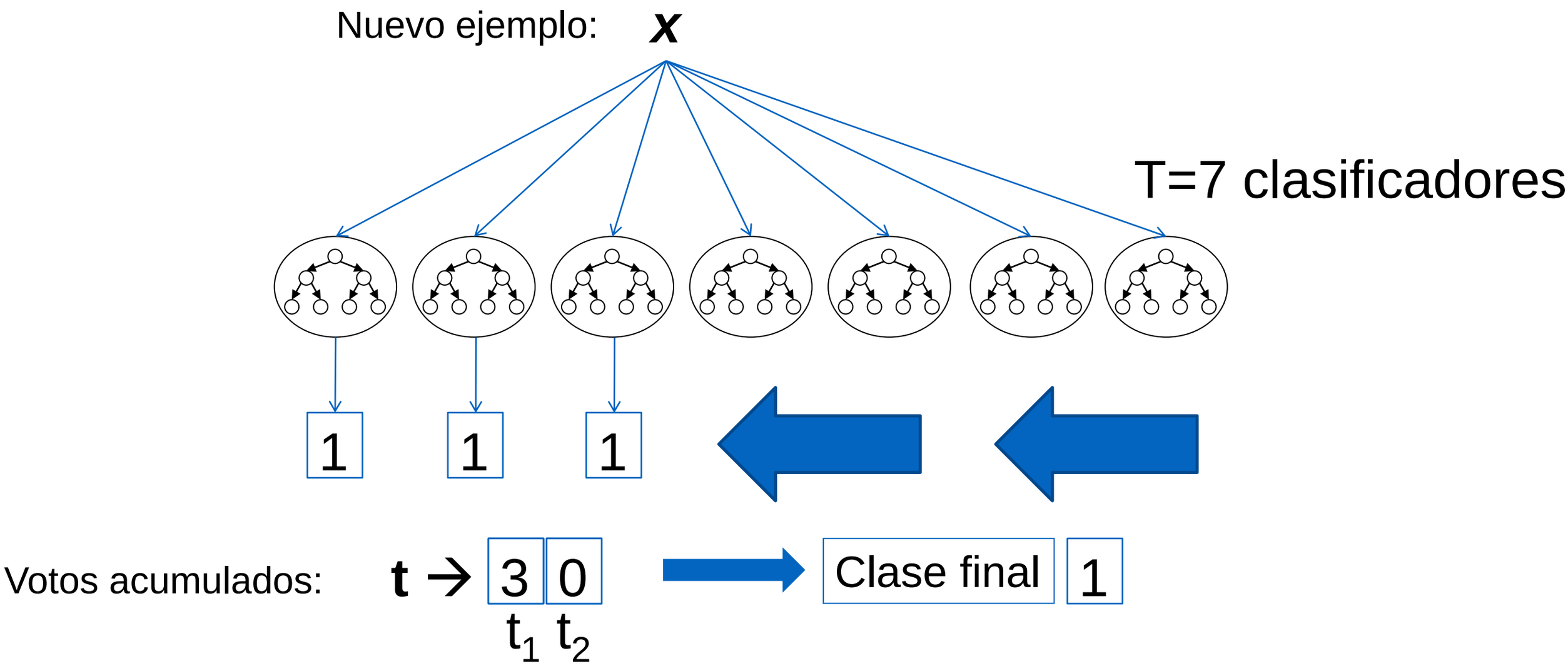
Poda dinámica de conjuntos



¿Necesitamos preguntar a todos los clasificadores?

NO

Poda dinámica de conjuntos



¿Necesitamos preguntar a todos los clasificadores?

NO

Importancia de atributos

- Los conjuntos obtienen % de error mejores que los árboles de decisión. Sin embargo, se pierde su interpretabilidad. Aunque no del todo
  - El número de veces que aparece cada atributo en los árboles y cómo de arriba estén nos da una idea de lo importante que es un atributo.
  - Breiman lo sistematiza con el siguiente algoritmo. Para cada atributo desordenar los datos y ver cómo se deteriora la clasificación. Cuánto más se deteriore más importante es el atributo.

¿Por qué funcionan?

- Algunos motivos:
  - Motivos estadísticos: No hay suficientes datos para que el algoritmo de clasificación obtenga una solución óptima.
  - Motivos computacionales: El algoritmo no puede alcanzar la solución óptima.
  - Motivos expresivos: La solución está fuera del espacio de hipótesis que estamos explorando.

¿Por qué funcionan?

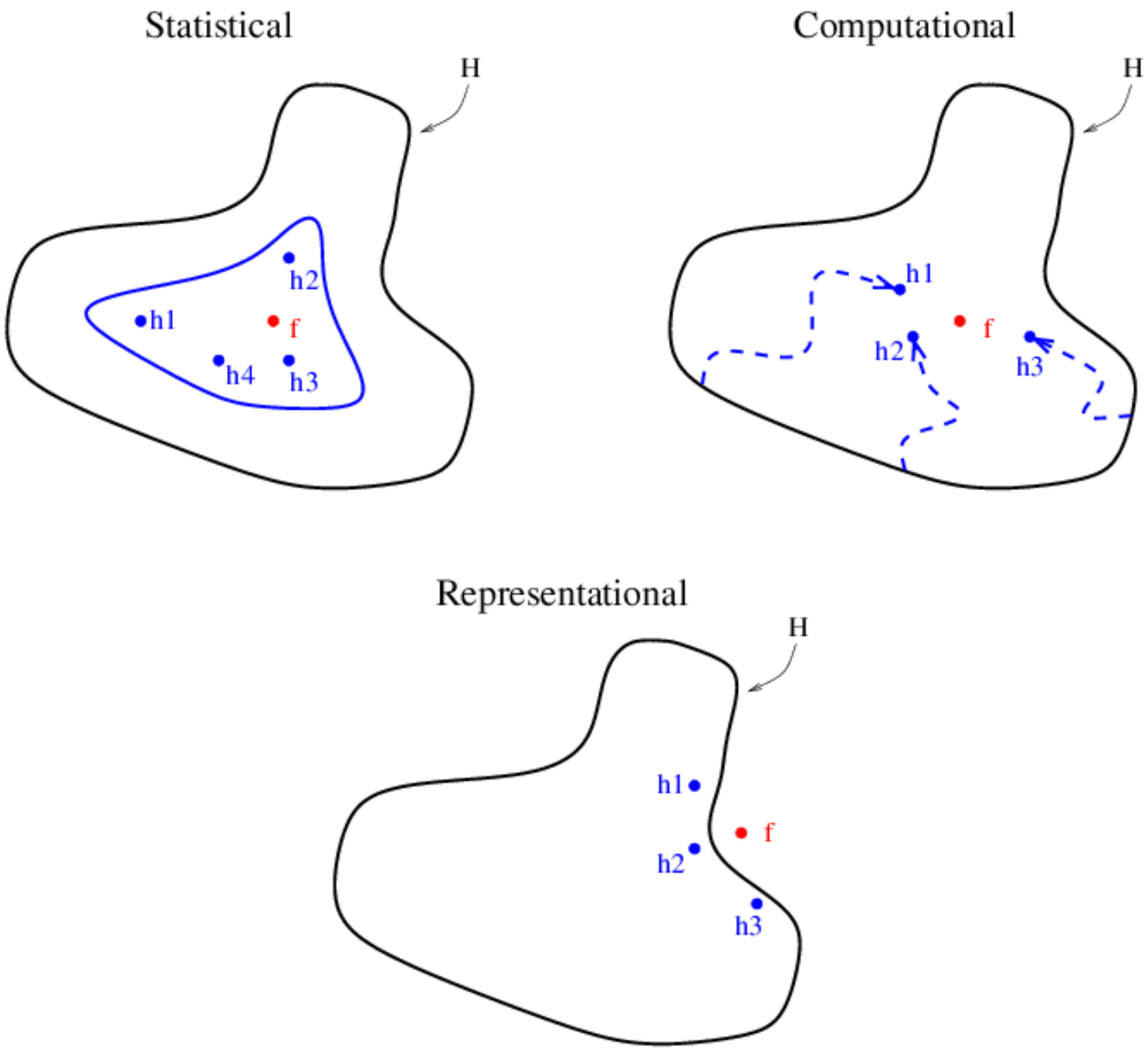
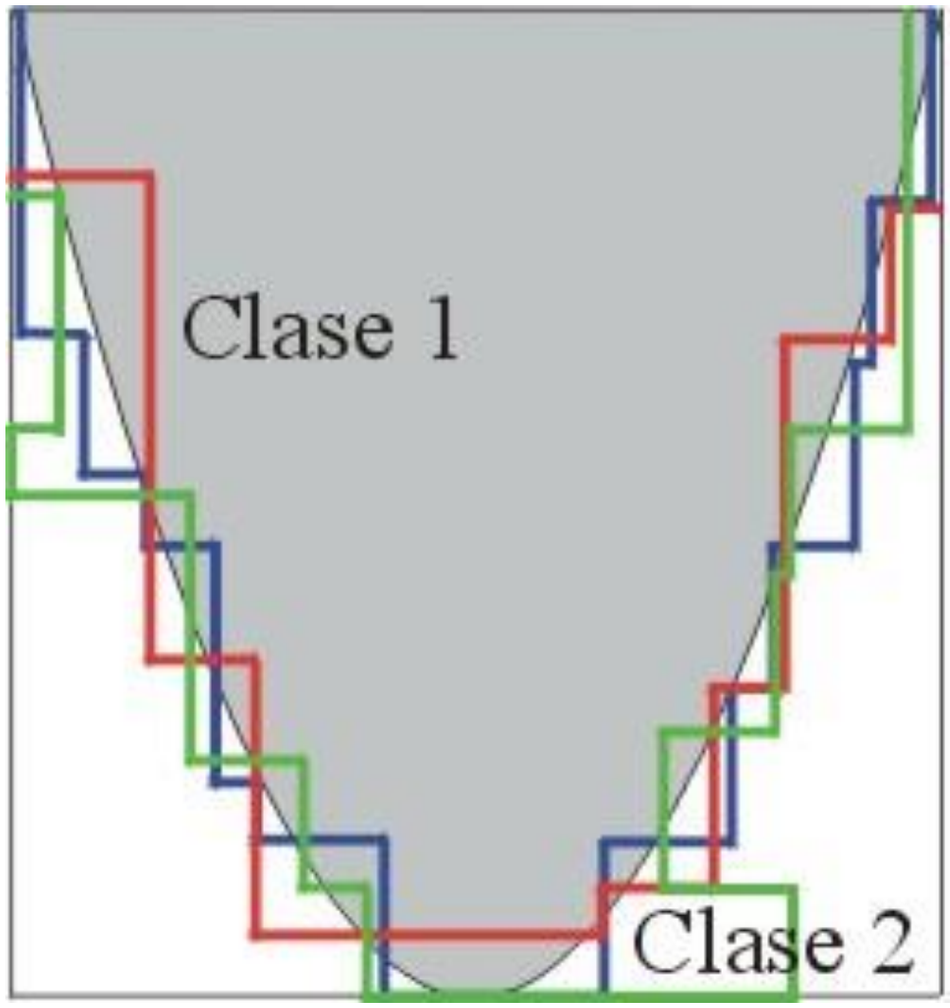


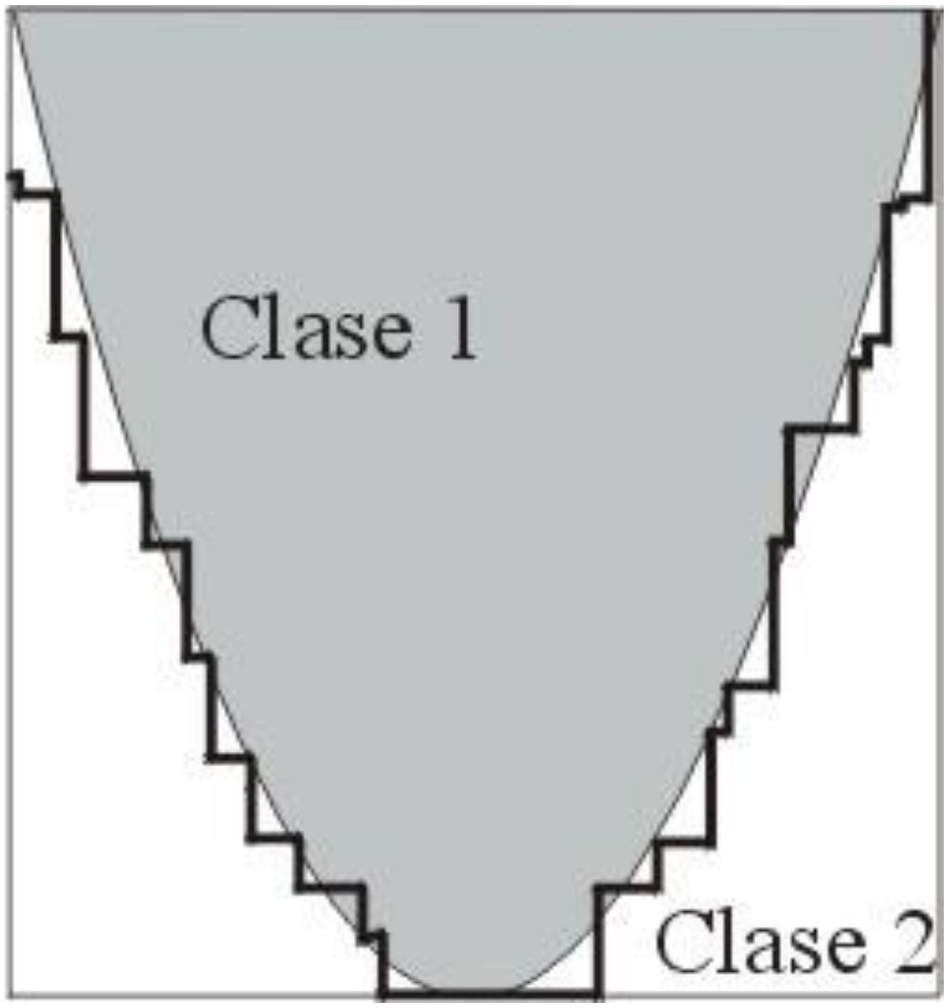
Fig. 2. Three fundamental reasons why an ensemble may work better than a single classifier

Thomas Dietterich

¿Por qué funcionan?

Un conjunto de soluciones subóptimas se puede combinar para compensar sus limitaciones.





Historia de éxito 1: Netflix prize challenge

- Dataset: ratings de 17770 películas y 480189 usuarios

Combina cientos de modelos de tres equipos

Variante de stacking

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 **leaders.**

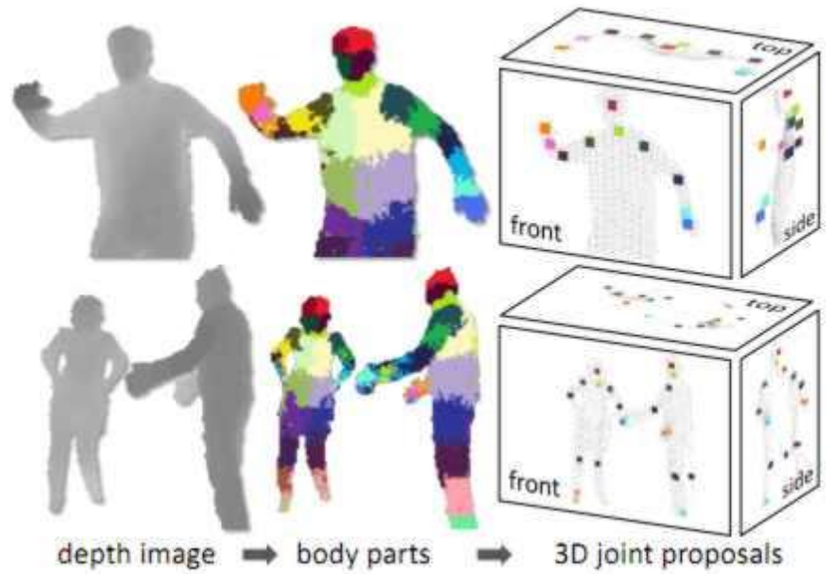
Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

Historia de éxito 2: KDD cup

- KDD cup 2013: Predecir que artículos están escritor por qué autor.
    - El equipo ganador usaba Random Forest y Boosting among other models combined with regularized linear regression.
- KDD cup 2014: Predict funding requests that deserve an A+ in donorschoose.org
    - Multistage ensemble
- KDD cup 2015: Predict dropouts in MOOC
    - Multistage ensemble

Historia de éxito 3: Kinect

- Visión por ordenador
- Clasificar píxeles en partes del cuerpo (mano, cabeza, etc)
- Usa *Random Forests*



Desventajas de utilizar conjuntos

- ¡Ninguna! Bueno puede que alguna…
- Más lento que un clasificador único ya que hay que crear cientos o miles de clasificadores.
  - Se puede mitiga mediante la poda
- Se pierde parte de la interpretabilidad de los árboles

Ventajas

- Una familia de algoritmos con un rendimiento de entre los mejores actualmente, (especialmente ramdom forest). Comparable o mejor que SVMs

- Son algoritmos sin prácticamente parámetros que ajustar.

- Si los algoritmos base son árboles, se pueden crear muy rápido y clasifican muy rápido.