Introducción al reconocimiento de patrones

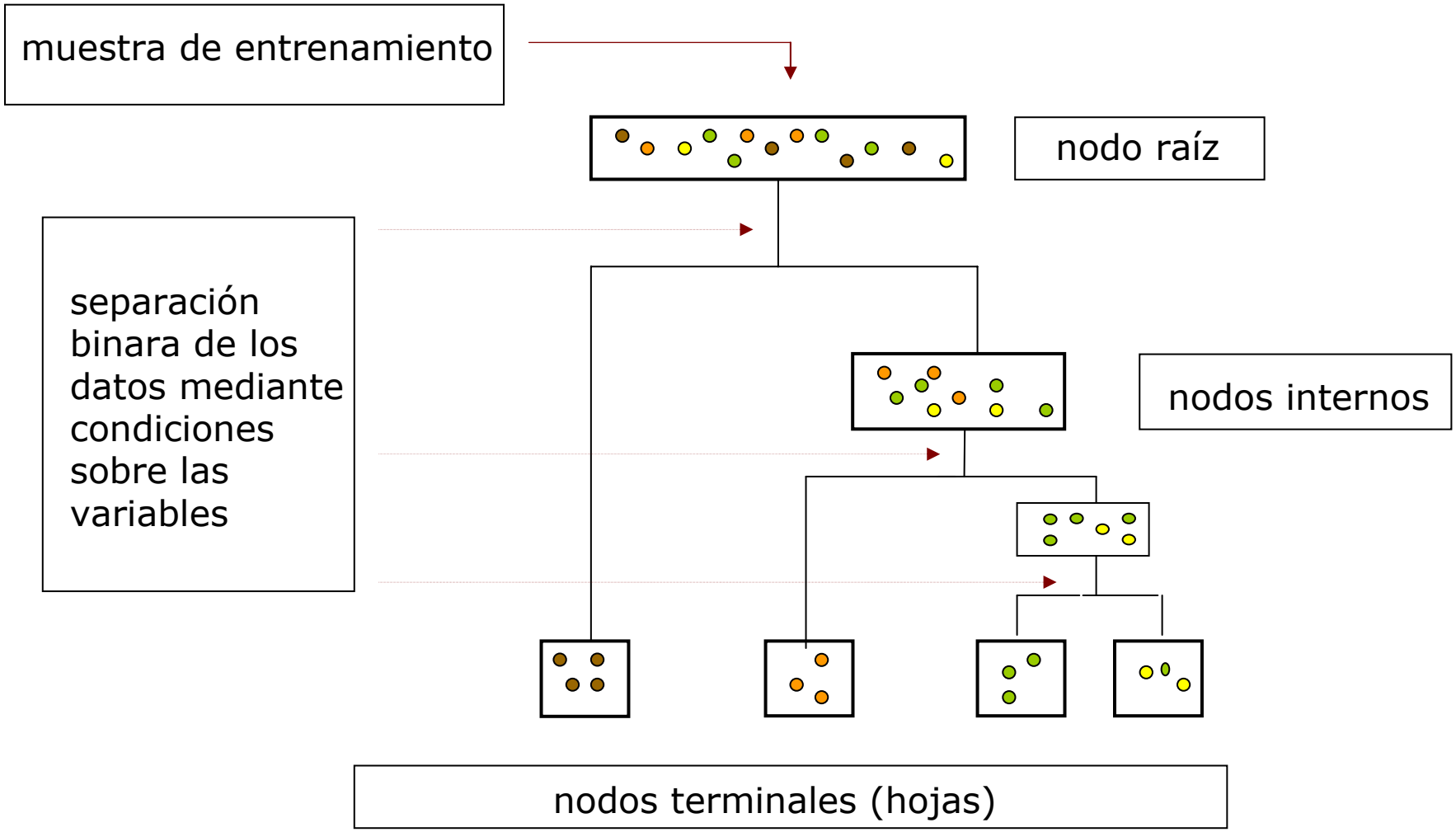
Bagging & Boosting

aplicados a árboles de clasificación

Andrea Mesa

diciembre 2008

Árboles de clasificación



Etapas en la construcción de un árbol

- elección de una medida de impureza
- elección del tamaño del árbol
- asignación de una clase a las hojas

Medidas de impureza

- error de clasificación $$i(t) = 1 - \max_{j=1,\cdots,c} p_j$$
- entropía $$i(t) = -\sum_{j=1}^{c} p_j log P_j$$
- índice de Gini $$i(t) = 1 - \sum_{j=1}^{c} p^2(w_j)$$

Tamaño del árbol

- criterios de parada

  umbrales sobre reducción de impureza o sobre el número de observaciones por nodo, test χ^2 , etc.

- poda

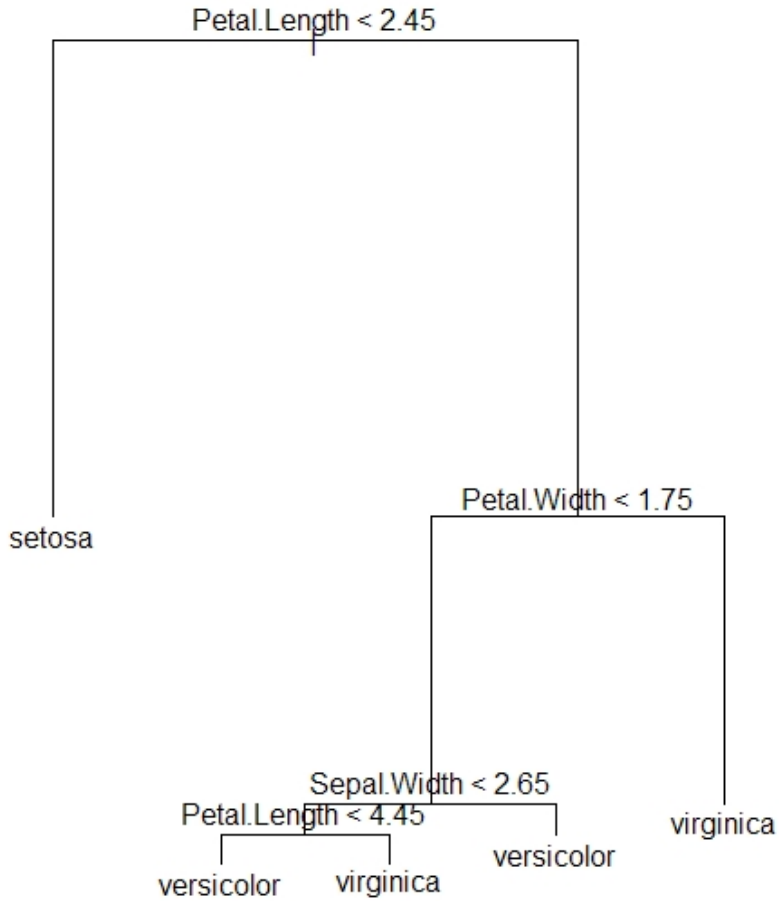
Asignación de una clase a las hojas

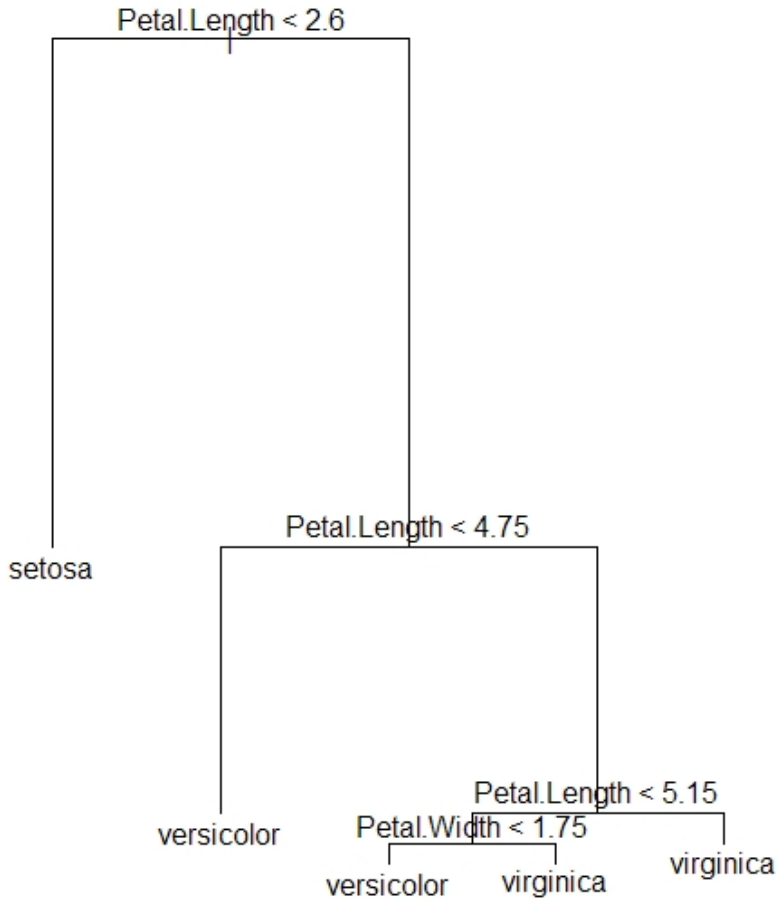
- se asigna a cada hoja la clase más representada: voto mayoritario simple
- en caso de empate la asignación es aleatoria

Ventajas y desventajas

ventajas	desventajas
• interpretabilidad • no se impone restricciones sobre las variables	• inestabilidad

Los árboles son algoritmos inestables





Bagging

Bootstrap **AGG**regat**ING**

Breiman (1994)

El clasificador bagging combina los outputs de varios clasificadores construidos utilizando remuestras (muestras *bootstrap*) del conjunto de entrenamiento.

La clase que recibe más votos entre los clasificadores es la clase asignada por el clasificador bagging.

Muestras bootstrap

- se asigna a cada una de las n observaciones el mismo peso (1/n).
- se muestrea de forma aleatoria y con reemplazo obteniendo K nuevas muestras, cada una de las cuales tiene distribución asintótica próxima a la empírica de la muestra original.
- cada observación tiene una probabilidad de salir sorteada en la remuestra igual a $1-(1-1/n)^n$.
- para valores grandes de n la probabilidad se aproxima a $1-1/e \sim 0.63$

Ejemplo de muestras bootstrap

```
s=randsample(1:10,10,true)

s =   10     8     2     5    10    10     5     9     1     4

s =    9     1     2     3     2     7     3     2     1     8

s =    5    10     5     5     9     6     3     7     9     1

s =    7     4     9     6     8     5     4     2     2     7

s =    4     6     2     7     4     9     9     6     5     9
```

Algoritmo bagging

1. conjunto de entrenamiento $\mathcal{L}$ de tamaño n
2. for $k = 1$ to K:
   a) muestra bootstrap $\mathcal{L}^k$ de tamaño n de $\mathcal{L}$
   b) se entrena un árbol T^k utilizando $\mathcal{L}^k$
3. Output

$$\varphi_{Bagging}(x) = arg \max_{m=1,\cdots,c} \#\{k; T(x, \mathcal{L}^k) = m\}$$

Bases de datos

	# entrenamiento	# test	# variables	# clases
Iris	100	50	4	3
Breast	466	233	9	2
Vowel	660	330	10	11

Datos disponibles en http://www.ics.uci.edu/mlearn/MLRepository.html

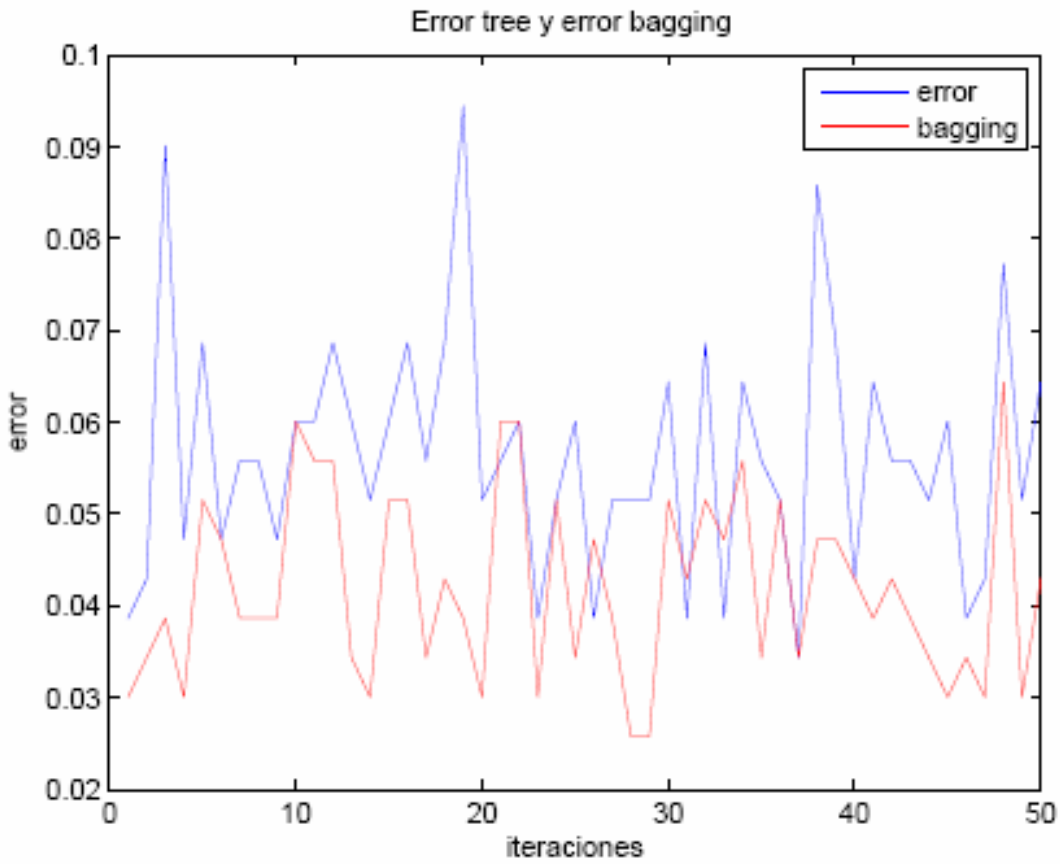
Árboles de clasificación utilizados

- medida de impureza: índice de Gini
- parada: test χ^2
- asignación de clases: voto mayoritario

Bagging aplicado a Breast

15 remuestras y 50 particiones en entrenamiento y test

```
Error test promedio para un arbol: 0.056567
Error Bagging promedio:            0.041974
```



Boosting

Boosting es un proceso iterativo que combina los output de varios clasificadores débiles para obtener un clasificador más eficiente.

Algoritmos de boosting implementados:

- Adaboost.M1 (Freund & Schapire, 1997)
- SAMME (Zhu et al., 2006)

Adaboost.M1

- Algoritmo de boosting más conocido para bases de datos con dos clases.
- Bajo desempeño en problemas de varias clases.
- Al inicio se considera una distribución uniforme para los pesos de las observaciones, en las sucesivas iteraciones esta distribución cambia, asignando mayor peso a las observaciones mal clasificadas en el paso anterior (remuestras ponderadas)
- Los clasificadores que muestran mejor performance en el entrenamiento tienen mayor peso en la votación final.

Ejemplo de remuestras ponderadas

vector de pesos w=[.3 .3 .05 .05 .05 .05 .05 .05 .05 .05]

s=randsample(1:10, 10, true, w)

s= 1 2 7 1 1 1 1 3 1 1

s= 1 5 2 9 2 2 7 2 1 4

s= 7 1 4 2 7 2 5 2 2 1

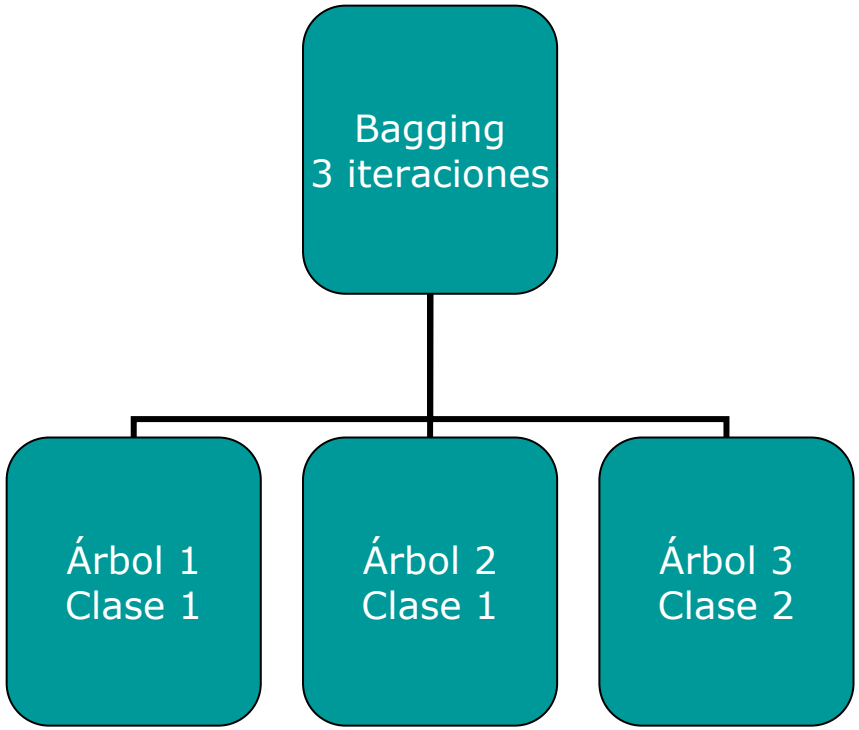
s= 1 4 2 2 1 4 2 8 8 2

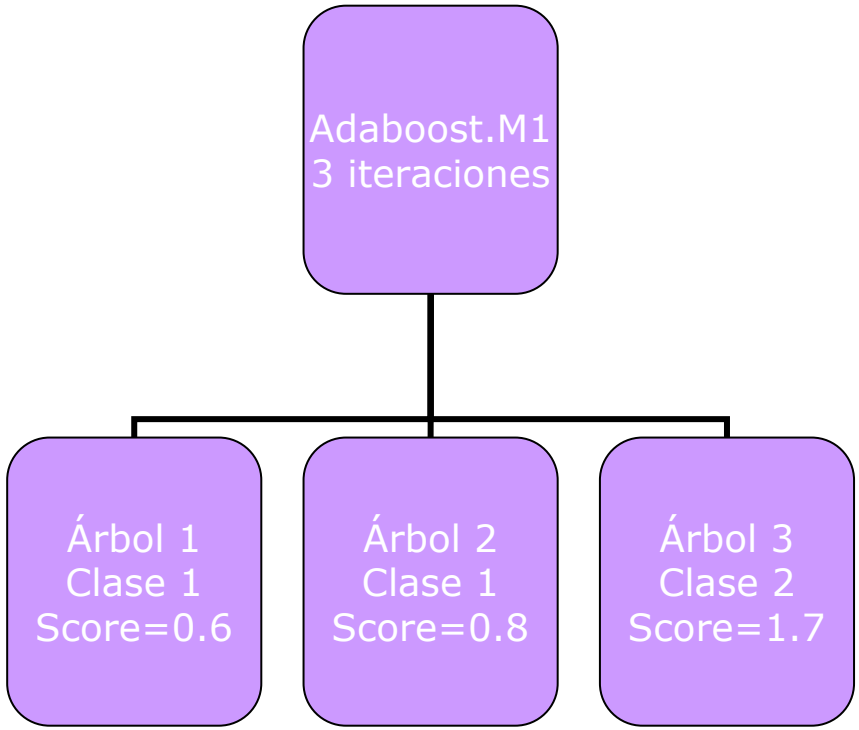
s= 1 1 8 1 1 4 1 2 1 10

Adaboost.M1

1. Inicialización de los pesos de las observaciones $w_n = \frac{1}{N} \quad n = 1, \ldots, N$
2. for $k = 1$ to K:
   a) construcción de un clasificador $T^k(x)$ con el conjunto de entrenamiento ponderado por w_n^k
   b)
   $$\epsilon_k = \sum_{n=1}^{N} w_n^k I_{\{j_n \neq T^k(\mathbf{x}_n)\}}$$
   c) $\beta_k = \frac{1-\epsilon_k}{\epsilon_k}$
   d) actualización de los pesos w^k
   $$w_n^{(k+1)} = \frac{w_n^k \beta_k^{I_{\{j_n \neq T^k(x_n)\}}}}{\sum_{n=1}^{N} w_n^k} \qquad n = 1, \ldots, N$$
3. Output
   $$\varphi_{adaboost}(x) = Arg\, \max_j \sum_{k=1}^{K} \ln(\beta_k) I_{\{T^k(x)=j\}}$$

Bagging vs Adaboost.M1





Votos clase 1: 2

Votos clase 2: 1

Predicción Bagging: clase 1

Votos clase 1: 2

Votos ponderados clase 1: 1.4

Votos clase 2: 1

Votos ponderados clase 2 : 1.7

Predicción Adaboost.M1: clase 2

Adaboost.M1 aplicado a Breast

```
50 iteraciones
Error adaboost:  0.034335

50 iteraciones (con poda)
Error adaboost:  0.034335

50 iteraciones (con poda)
Error adaboost:  0.034335

100 iteraciones (con poda)
Error adaboost:  0.042918

100 iteraciones
Error adaboost:  0.034335
```

SAMME

Stagewise **A**dditive **M**odeling using a **M**ulticlass **E**xponential loss function

- Difiere de Adaboost.M1 sólo en el cálculo de β_k
- Mejor performance que Adaboost.M1 para bases de datos con varias clases

SAMME

1. Inicialización de los pesos de las observaciones $w_n = \frac{1}{N} \quad n = 1, \ldots, N$
2. for $k = 1$ to K:

   a) construcción de un clasificador $T^k(x)$ con el conjunto de entrenamiento ponderado por w_n^k

   b)

$$\epsilon_k = \sum_{n=1}^{N} w_n^k I_{\{j_n \neq T^k(\mathbf{x}_n)\}}$$

   c) $\beta_k = \frac{1-\varepsilon_k}{\varepsilon_k}(c-1)$

   d) actualización de los pesos w^k

$$w_n^{(k+1)} = \frac{w_n^k \beta_k^{I_{\{j_n \neq T^k(x_n)\}}}}{\sum_{n=1}^{N} w_n^k} \qquad n = 1, \ldots, N$$

3. Output

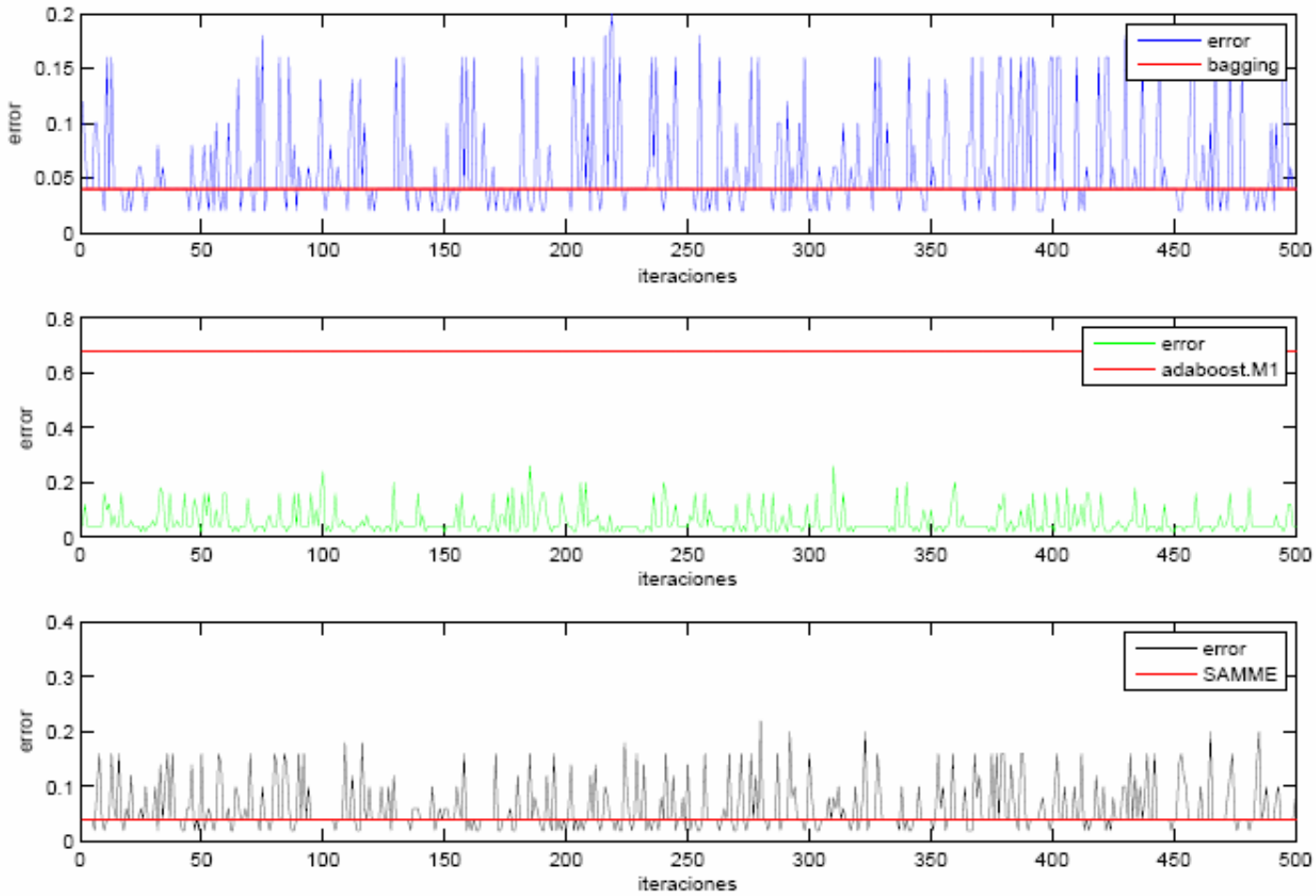
$$\varphi^a(x) = Arg\,\max_j \sum_{k=1}^{K} \ln(\beta_k) I_{\{T^k(x)=j\}}$$

SAMME aplicado a Breast

```
50 iteraciones
Error samme:   0.042918

50 iteraciones (con poda)
Error samme:   0.038627

50 iteraciones (con poda)
Error samme:   0.038627

100 iteraciones (con poda)
Error samme:   0.021459

100 iteraciones
Error samme:   0.038627
```

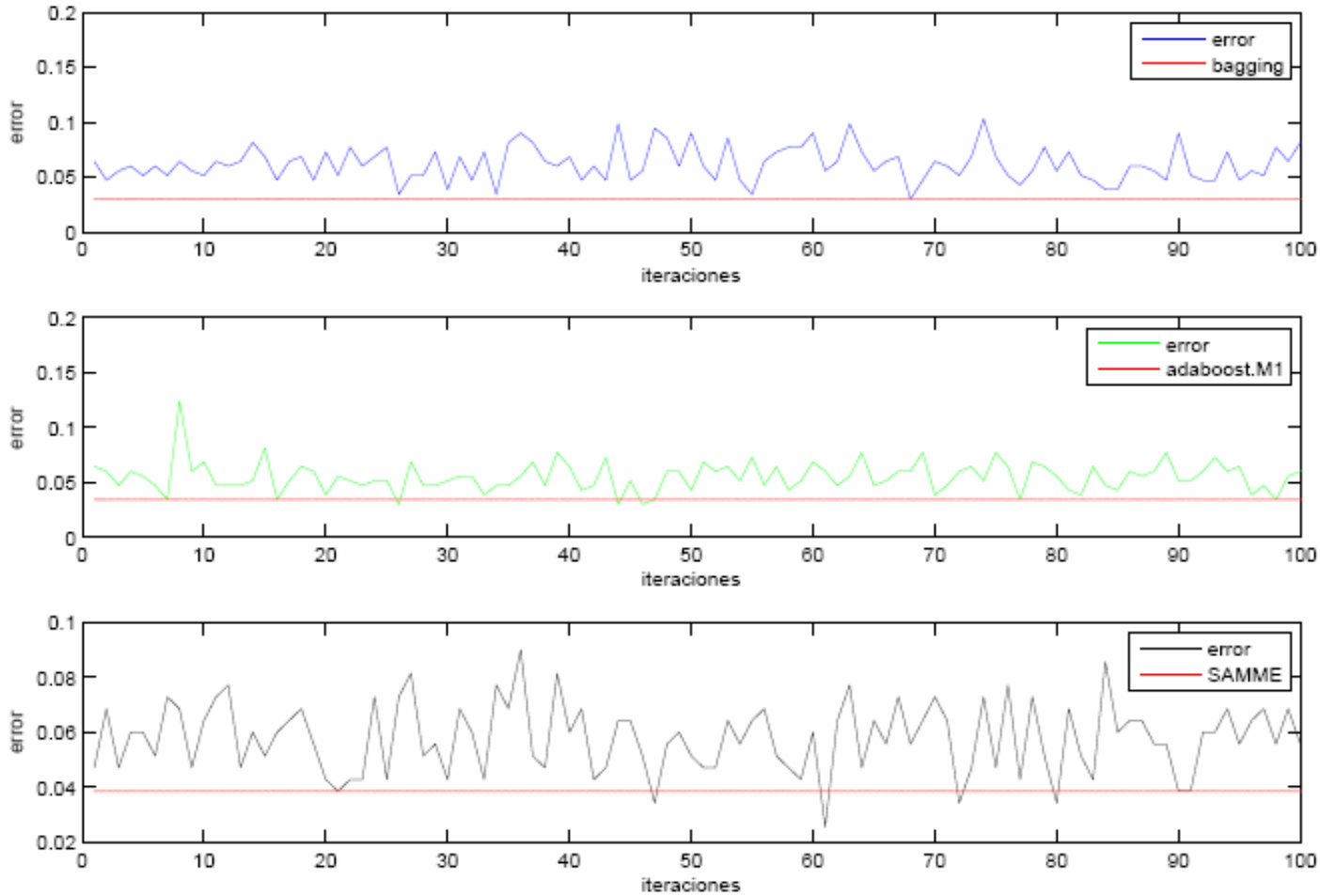
Comparación de algoritmos: Iris

```
Error bagging: 0.04
Error adaboost:0.68
Error samme:   0.04
```



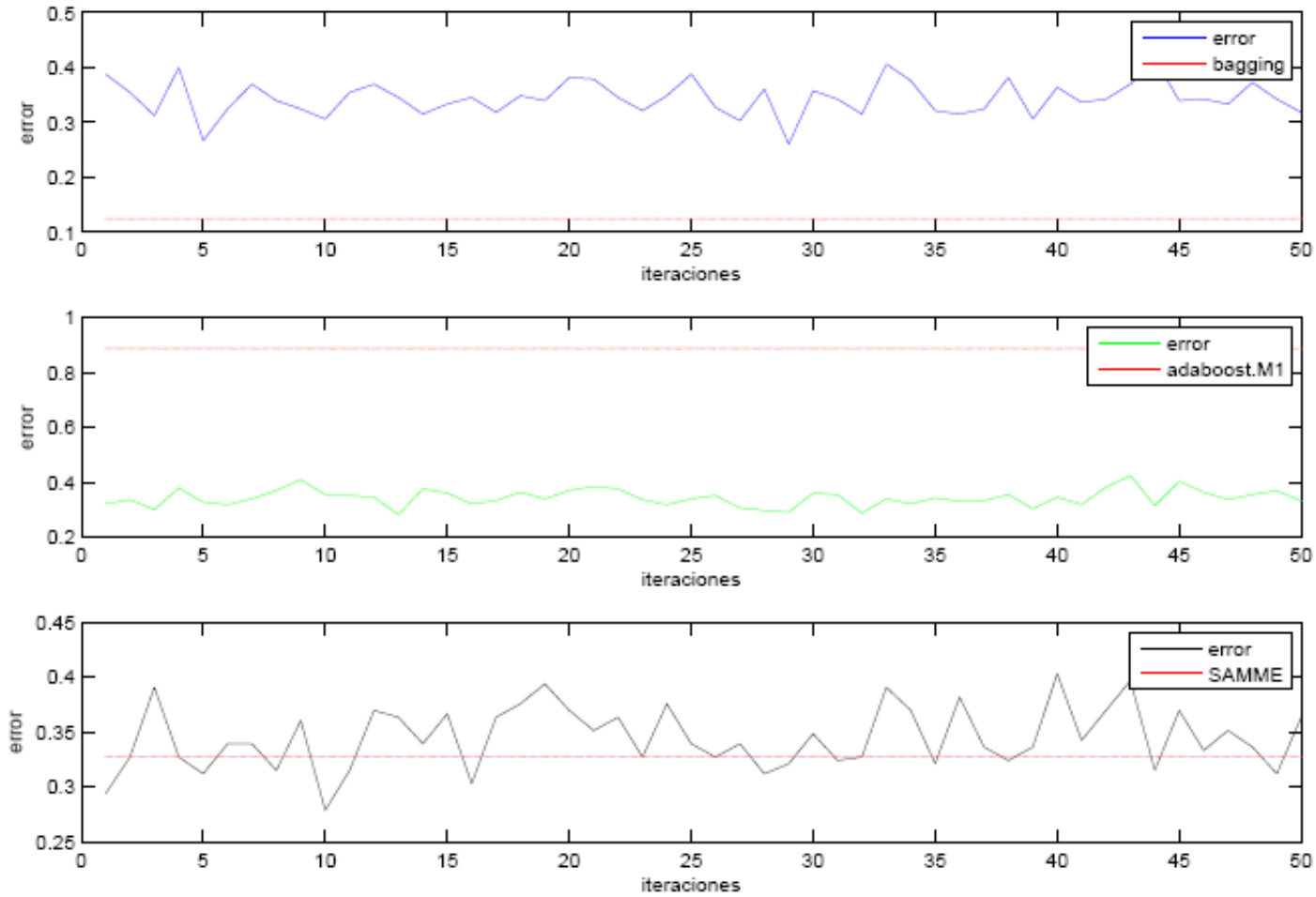
Comparación de algoritmos: Breast

```
Error bagging: 0.030043
Error adaboost:0.034335
Error samme:   0.038627
```



Comparación de algoritmos: Vowel

```
Error bagging: 0.12424
Error adaboost:0.88788
Error samme:   0.32727
```



Resultados

	Bagging	Adaboost.M1	SAMME	Tree error promedio en 50 iteraciones
iris	0.04	0.68	0.04	0.06
breast	0.03	0.03	0.03	0.05
vowel	0.12	0.88	0.32	0.27

Conclusiones

- Existen muchos algoritmos de Boosting, que varían en la forma de calcular los errores y ponderaciones en cada iteración. Además de los considerados en este trabajo: Adaboost.M2, Adaboost.OC, Adaboost.ECC, Adaboost.MH, etc.

- Debe repetirse la división aleatoria de los datos en conjuntos de entrenamiento y testeo varias veces y promediar el error de clasificación de cada algoritmo considerando un mayor número de iteraciones que las realizadas en este trabajo (costo computacional)

- Más allá de las diferencias en los resultados de cada algoritmo, cabe destacar la necesidad de utilizar bagging o algún algoritmo de boosting cuando se decide trabajar con árboles de clasificación o con otro clasificador inestable como ser las redes neuronales.

  Otra opción: random forest.

Referencias

- Breiman, L. *Bagging predictors*. Machine Learning, 24, 123-140 (1996)
- Breiman, L. *Bias, Variance and Arcing Classifiers*. Technical Report. Statistics Department, University of California (1996)
- Zhu et al. *Multiclass Adaboost* (2006)

Bonus

Adaboost.OC

1. inicialización de los pesos de las observaciones $w_{n,j} = \begin{cases} \frac{1}{N(J-1)} & \text{if } j \neq j_n \\ 0 & \text{if } j = j_n \end{cases}$ $n = 1, \ldots, N;\ j = 1, \ldots, J.$

2. for $k = 1$ to K:

   a) cálculo de $\mu_k : Y \to \{0, 1\}$.

   b)
   $$U_k = \sum_{n=1}^{N} \sum_{j \in C} w_{n,j} I_{\{\mu_k(j_n) \neq \mu_k(j)\}}.$$

   c)
   $$W_k(n) = \frac{1}{U_k} \sum_{j \in C} w_{n,j} I_{\{\mu_k(j_n) \neq \mu_k(j)\}}.$$

   d) se construye el clasificador $T^k(\mathrm{x})$ usando los datos re-etiquetados $\mathcal{L}^k = \{(\mathrm{x}_n, \mu_k(j_n)) : 1 \leq n \leq N\}$ con pesos $W^k(n)$.

   e) $\tilde{T}_k(\mathrm{x}) = \{j \in \mathcal{C} : T_k(\mathrm{x}) = \mu_k(j)\}$.

   f) cálculo del error
   $$\varepsilon_k = \frac{1}{2} \sum_{m=1}^{N} \sum_{j \in C} w_{n,j} \left(I_{\{j_n \notin T_k(\mathbf{x_n})\}} + I_{\{j \in T_k(\mathbf{x_n})\}} \right)$$

   g) $\alpha_k = \frac{1}{2} \ln \left(\frac{1-\varepsilon_k}{\varepsilon_k} \right)$.

   h) actualización de pesos $w_{n,j}^k$
   $$w_{n,j}^{(k+1)} = \frac{w_{n,j}^k \exp\left(\alpha_k \left(I_{\{j_n \notin T_k(\mathbf{x_n})\}} + I_{\{j \in T_k(\mathbf{x_n})\}} \right) \right)}{Z_k} \qquad n = 1, \ldots, N$$

   Z_k factor de normalización.

3. Output
   $$\varphi^a(x) = Arg \max_j \sum_{k=1}^{K} \alpha_k I_{\{T_k(x) = \mu_k(j)\}}$$

Idea de Adaboost.OC

Paso 1: inicialización de pesos

Paso k:

- se aplica mu (coloring function)
- se calcula epsilon
- se calcula alfa
- se actualizan los pesos wnj

Output: clase mayoritaria ponderada

Adaboost.OC aplicado a Iris

Paso 1

dada la observación (xn,jn), si por ejemplo jn=1, entonces se le asigna a la observación un vector de pesos wn=(0 1/2*N 1/2*N)

Paso K

K=1	< mu >	1 0 0	< eps >	0.155	< alp >	0.848
K=2	< mu >	0 1 0	< eps >	0.123	< alp >	0.981
K=3	< mu >	0 1 1	< eps >	0.195	< alp >	0.71
K=4	< mu >	1 0 1	< eps >	0.163	< alp >	0.817
K=5	< mu >	1 0 0	< eps >	0.22	< alp >	0.633

salida del programa escrito en R

se actualizan los pesos w

Output

error Adaboost.OC = 0.0789899