Curso Introducción al Reconocimiento de Patrones, IIE
Trabajo final

Andrea Mesa

Diciembre 2008

Resumen

Bagging y Boosting son técnicas de combinación de clasificadores utilizables tanto en el caso de problemas de dos clases como de varias clases. El objetivo de este trabajo es presentar Bagging y algunos algoritmos de Boosting aplicados a árboles de decisión y comparar su desempeño evaluando los resultados obtenidos al aplicarlos a distintos conjuntos de datos.

1. Árboles de clasificación

Se considera un conjunto de datos etiquetados $\{(x_n, j_n) : 1 \leq n \leq N\}$, donde $x_n \in \mathbb{R}^d$ y $j_n \in \mathcal{C} = \{1, 2, \cdots, c\}$ y se busca elaborar un modelo que permita clasificar una nueva observación x_n en alguna de las c clases. Los árboles de decisión son modelos que pueden ser utilizados tanto cuando la variable de salida es categórica, como en este caso, o continua. En el primer caso se denominan árboles de clasificación y en el segundo de regresión.

Se comienza dividiendo el conjunto de datos de manera aleatoria en muestras de entrenamiento $\mathcal{L}$ y testeo $\mathcal{T}$ conteniendo 2/3 y 1/3 de las observaciones respectivamente. Para la construcción de los árboles se eligió trabajar con el índice de Gini como medida de impureza, construyendo un árbol de decisión con el conjunto de entrenamiento y midiendo el error de clasificación con los datos de testeo.

Los árboles de decisión son algoritmos inestables, esto es, pequeñas modificaciones en el conjunto de entrenamiento producen grandes cambios en el clasificador lo que se visualiza claramente al observar los resultados obtenidos cuando se repite el procedimiento de partición del conjunto de datos y construcción del árbol.

Se considera la base de datos `Breast` que consta de datos de cáncer de mama y contiene 699 observaciones correspondientes a dos clases (benigno y maligno) y 9 variables que

miden características celulares de los tejidos a clasificar.
A modo de ejemplo se muestra sólo la partición del nodo 1 en el split izquierdo de cada árbol obtenido para tres particiones diferentes del conjunto de datos `Breast` en entrenamiento y testeo:

```
|---|(node 1) cz <= 2
|-----|(node 2) nn <= 3
|-------|(node 3) ct <= 7.5
|---------|(node 4) scz <= 5.5
|-----------|(node 5) class: benigno
|---------|(node 4) scz > 5.5
|-------------|(node 6) class: benigno
|-------|(node 3) ct > 7.5
|---------------|(node 7) class: maligno
|-----|(node 2) nn > 3
|-----------------|(node 8) bn <= 1
|-------------------|(node 9) class: benigno
|-----------------|(node 8) bn > 1
|---------------------|(node 10) class: maligno

|---|(node 1) cz <= 2
|-----|(node 2) nn <= 3
|-------|(node 3) scz <= 5.5
|---------|(node 4) bn <= 4
|-----------|(node 5) class: benigno
|---------|(node 4) bn > 4
|-------------|(node 6) class: benigno
|-------|(node 3) scz > 5.5
|---------------|(node 7) class: maligno
|-----|(node 2) nn > 3
|-----------------|(node 8) class: maligno

|---|(node 1) cz <= 2
|-----|(node 2) nc <= 4
|-------|(node 3) ct <= 6
|---------|(node 4) nn <= 3
|-----------|(node 5) class: benigno
|---------|(node 4) nn > 3
|-------------|(node 6) class: benigno
|-------|(node 3) ct > 6
|---------------|(node 7) class: maligno
|-----|(node 2) nc > 4
|-----------------|(node 8) ct <= 3.5
|-------------------|(node 9) class: benigno
|-----------------|(node 8) ct > 3.5
|---------------------|(node 10) class: maligno
```

Para tres particiones distintas del conjunto de datos original se tienen errores de clasificación y número de patrones mal clasificados diferentes:

```
Error (test): 0.064103   Cantidad de patrones mal clasificados: 15
Error (test): 0.051282   Cantidad de patrones mal clasificados: 12
Error (test): 0.042735   Cantidad de patrones mal clasificados: 10
```

2. Bagging

El método Bagging (Breiman, 1996) consiste en construir un clasificador combinando distintas versiones de clasificadores. Estas versiones se construyen utilizando nuevos conjuntos de entrenamiento creados por la generación de muestras *boostrap* del conjunto de entrenamiento $\mathcal{L}$. De ahí el nombre Bagging, que es acrónimo de **B**oostrap **Agg**regat**ing**.

Si el modelo es de regresión, es decir predice una salida numérica, el estimador bagging se crea promediando los outputs de los distintos estimadores. Si el modelo es de clasificación, la salida del clasificador combinado será aquella clase que resulte ser la "más votada" por los diferentes clasificadores.

El método Bagging se utiliza cuando el método de predicción es inestable, es decir, cuando pequeñas modificaciones en el conjunto de entrenamiento producen grandes cambios en el predictor, como por ejemplo en *árboles de decisión*.

2.1. Algoritmo

Una vez dividido el conjunto de datos en conjuntos de entrenamiento $\mathcal{L}$ y testeo $\mathcal{T}$, se aplica Bagging (figura 1) que puede resumirse en los siguientes pasos:

1. usando $\mathcal{L}$ se construye un árbol de decisión y se mide el error de clasificación, `e_test` con $\mathcal{T}$
2. se toma una muestra bootstrap $\mathcal{L}_{\lfloor}$ de $\mathcal{L}$ y se contruye un árbol usando $\mathcal{L}_{\lfloor}$.
   Se repite el procedimiento B veces
3. a cada dato de $\mathcal{T}$ se le asigna la clase votada por la mayoría de los árboles construidos en el paso anterior. La proporción de veces que la clase estimada difiere de la verdadera es el error bagging: `error_bagg`

Luego puede repetirse la división de los datos en $\mathcal{L}$ y $\mathcal{T}$ varias veces y calcular los errores promedio.

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

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

Figura 1: Bagging

2.2. Aplicación a un conjunto de datos

Se considera el conjunto de datos `Breast` y se aplica el procedimiento descrito más arriba, tomando 15 remuestras del conjunto de entrenamiento y iterando 50 veces (50 particiones del conjunto de datos original) para calcular los errores promedio. Para comparar con el error cometido al considerar sólo un árbol de clasificación se calculó también para este caso el error promedio en las 50 iteraciones (figura 2).

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

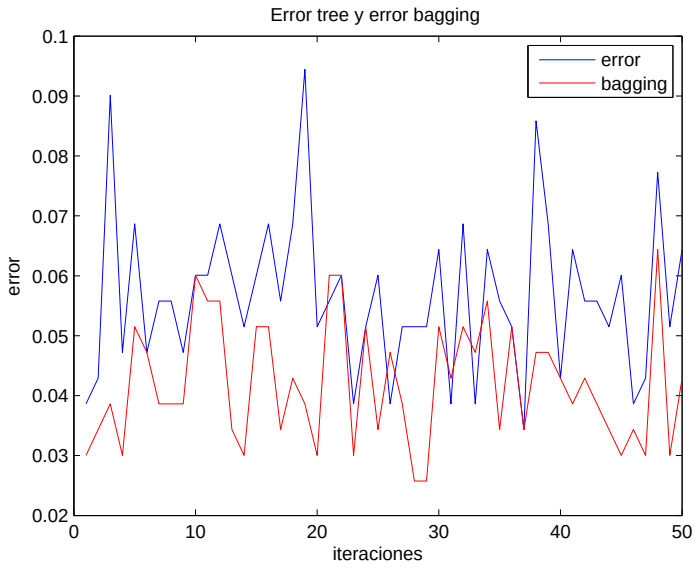

Figura 2: Errores promedio con 50 particiones de datos Breast en conjuntos de entrenamiento y testeo

Si se considera un mayor número de remuestras, para 5 particiones diferentes del conjunto de datos se obtiene:

```
50 iteraciones
Error bagging:  0.034335

50 iteraciones
Error bagging:  0.030043

50 iteraciones
Error bagging:  0.030043

100 iteraciones
Error bagging:  0.030043
```

3. Adaboost.M1

Boosting es un procedimiento iterativo que combina la salida de varios clasificadores para producir un clasificador combinado. En un principio fue pensado para combinar clasificadores débiles (cuyo despempeño es apenas mejor que la clasificación al azar) pero es utilizado con otros clasificadores como los árboles de decisión.

Adaboost.M1 (Freund y Schapire, 1997), es el algoritmo de boosting más conocido para la clasificación de patrones correspondientes a dos clases.

3.1. Algoritmo

La idea del algortimo Adaboost.M1 (figura 3) consiste en:

1. dividir el conjunto de datos en conjuntos de entrenamiento $\mathcal{L}$ y testeo $\mathcal{T}$
2. asignar a todas las observaciones de $\mathcal{L}$ el mismo peso $w_n = \frac{1}{N}$
3. for $k = 1$ to K
   - se construye un árbol de clasificación T_k con el conjunto de entrenamiento $\mathcal{L}_k$ escogido de $\mathcal{L}$ con la distribución de pesos w_n^k
   - se calcula el error de clasificación ϵ_k sobre el conjunto de entrenamiento $\mathcal{L}_k$
   - se calcula $\beta_k = \frac{1-\varepsilon_k}{\varepsilon_k}$
   - se actualizan los pesos w_n^k incrementando el peso de los patrones mal clasificados por el árbol T_k multiplicándolos por el factor $\beta_k > 1$

- se asigna a cada árbol T_k un peso igual a $\ln(\beta_k)$ que representa su precisión sobre el conjunto de entrenamiento $\mathcal{L}_k$
- se calcula el error de clasificación del árbol T_k con el conjunto de testeo $\mathcal{T}$

4. se construye el clasificador final con el cual se asigna a cada nueva observación la clase más votada por la mayorí ponderada

Adaboost.M1 requiere que el error ϵ_k para cada árbol sea menor que $1/2$, pues en caso contrario $\beta_k < 1$ y los pesos de las observaciones mal clasificadas decrecen en lugar de aumentar en el paso k.

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

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

Figura 3: Adaboost.M1

3.2. Aplicación a un conjunto de datos

Se utiliza nuevamente la base de datos `Breast` que corresponde a un problema de clasificación en dos clases. Se divide el conjunto de datos `Breast` en muestra de entrenamiento y de testeo y se aplica Adaboost.M1. Este proceso de repite para 5 particiones distintas del conjunto de datos y para un número de iteraciones distinto. En algunos casos se utilizó la función `tree_prune` para podar los árboles en cada iteración.

```
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
```

4. SAMME

Cuando el número de clases es mayor que dos, Adaboost.M1 falla en la clasificación, por esta razón, para resolver el problema de clasificación en varias clases, se han propuesto varios algoritmos de boosting, como por ejemplo **S**tagewise **A**dditive **M**odeling using a **M**ulticlass **E**xponential loss function, SAMME, algoritmo desarrollado por Zhu, J., Rosset, S. Zou, H. y Hastie, T. (2006).

4.1. Algoritmo

SAMME difiere de Adaboost.M1 sólo en el cálculo de β_k. En SAMME $\beta_k = \frac{1-\epsilon_k}{\epsilon_k}(c-1)$ siendo c el número de clases (figura 4).

En Adaboost.M1 para que $\beta_k > 1$ es necesario imponer la condición $\epsilon < \frac{1}{2}$; en el caso de SAMME esta condición se convierte en $1 - \epsilon > \frac{1}{c}$.

Cuando el problema de clasificación es de dos clases, $c = 2$, la condición $\epsilon < \frac{1}{2}$ impone que el clasificador tenga una performance mejor que la de una clasificación aleatoria. Pero para el caso de $c > 2$ la probabilidad de clasificar bien debe ser mayor que $1/c$ que sería la precisión alcanzada por la clasificación al azar.

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

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

Figura 4: SAMME

4.2. Aplicación a un conjunto de datos

Se divide el conjunto de datos `Breast` en muestra de entrenamiento y de testeo y se aplica el algoritmo SAMME. Este proceso de repite para 5 particiones distintas del conjunto de datos y para un número de iteraciones distinto. En algunos casos se utilizó la función `tree_prune` para podar los árboles en cada iteración.

```
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
```

5. Otros algoritmos de Boosting

Existen muchas otras variantes del algoritmo Adaboost.M1 para problemas de clasificación con varias clases: AdaBoost.M2, Adaboost.OC, Adaboost.ECC, entre otros.

A modo de ejemplo se resume a continuación uno de estos algoritmos, Adaboost.OC, escogido pues utiliza una técnica para la construcción y evaluación de los árboles en cada iteración k diferente a la de los algoritmos ya presentados.

5.1. Adaboost.OC

Adaboost.OC (Schapire et al., 1997) combina Boosting con Output Coding, técnica introducida por Dietterich, T. y Bakiri, G. (1995) (figura 5).

La idea consiste en inicializar el vector de pesos de las observaciones del conjunto de entrenamiento $\mathcal{L}$ de la siguiente manera:

$$w_{n,j} = \frac{I_{\{j \neq j_n\}}}{N(c-1)} = \left\{ \begin{array}{ll} \frac{1}{N(c-1)}, & j \neq j_n \\ 0, & j = j_n \end{array} \right\}$$

donde j_n es la etiqueta de la observación x_n. En cada iteración k del algoritmo se re-etiquetan las observaciones x_n mediante una función μ_k (*coloring function*) y con estos patrones re-etiquetados ponderados por los pesos $w_{n,j}$ se entrena el clasificador T_k. Se tiene un error cuando el output del clasificador T_k es distinto de $\mu_k(j)$. Luego se actualizan los pesos y se construye el clasificador final de manera similar a los demás algoritmos de boosting.

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 $T^k(\mathbf{x})$ usando los datos re-etiquetados $\mathcal{L}^k = \left\{(\mathbf{x}_n, \mu_k(j_n)) : 1 \leq n \leq N\right\}$ con pesos $W^k(n)$.
   e) $\tilde{T}_k(\mathbf{x}) = \{j \in \mathcal{C} : T_k(\mathbf{x}) = \mu_k(j)\}$.
   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
   $$\varphi^a(x) = Arg\, \underset{j}{\text{máx}} \sum_{k=1}^{K} \alpha_k I_{\{T_k(x) = \mu_k(j)\}}$$

Figura 5: Adaboost.OC

6. Comparación de algoritmos

Como forma de evaluar el desempeño de los tres algoritmos elegidos para su implementación, i.e. Bagging, Adaboost.M1 y SAMME fueron aplicados a tres conjuntos de datos con diferentes características (cuadro 1). En todos los casos se comienza con la división aleatoria de los datos en 2/3 y 1/3 para las muestras de entrenamiento y testeo.

6.1. Bases de datos

Los conjuntos de datos con los que fueron ensayados los distintos algoritmos provienen del UCI Machine Learning Repository y se encuentran disponibles en
http://www.ics.uci.edu/ mlearn/MLRepository.html

Breast
Base de datos que consta de datos de cáncer de mama y contiene 699 observaciones correspondientes a dos clases (benigno (458 observaciones) y maligno (241 observaciones)) y 9 variables que miden características celulares (Clump Thickness (ct), Uniformity of Cell Size (cz), Uniformity of Cell Shape (cs), Marginal Adhesion (ma), Single Epithelial Cell Size (scz), Bare Nuclei (bn), Bland Chromatin (bc), Normal Nucleoli (nn), Mitoses (mi)).

Iris
Base con 150 datos de flores de Iris correspondientes a 3 clases (Setosa, Virginia y Versicolor) a los cuales se les midió 4 variables (largo y ancho del sépalo, largo y ancho del pétalo).

Vowel[1]
Base de datos que contiene 990 observaciones correspondientes a 11 clases de pronunciación de vocales en el idioma inglés y 10 variables utilizadas en la digitalización de los sonidos. Los símbolos que representan las vocales y las palabras en las cuales fueron medidos los 11 sonidos son:
i (heed), E (head), a (hard), O (hod), U (hood), 3: (heard), I (hid), A (had), Y (hud), C: (hoard), u: (who'd).

El problema de clasificación de los datos `Vowel` es difícil y generalmente los métodos de clasificación alcanzan un 40 % de error con los datos de testeo (Hastie, Tibshirani & Friedman, 2001).

datos	# muestra entrenamiento	# muestra test	#variables	#clases
Breast	466	233	9	2
Iris	100	50	4	3
Vowel	660	330	10	11
Vowel2	528	462	10	11

Cuadro 1: Bases de datos

[1]Vowel2 conjunto de datos donde la división en muestras de entrenamiento y testeo es la dada por la base de datos original

6.2. Resultados

Iris

Luego de 500 iteraciones los valores del error de clasificación obtenidos en Bagging, Adaboost.M1 y SAMME son:

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

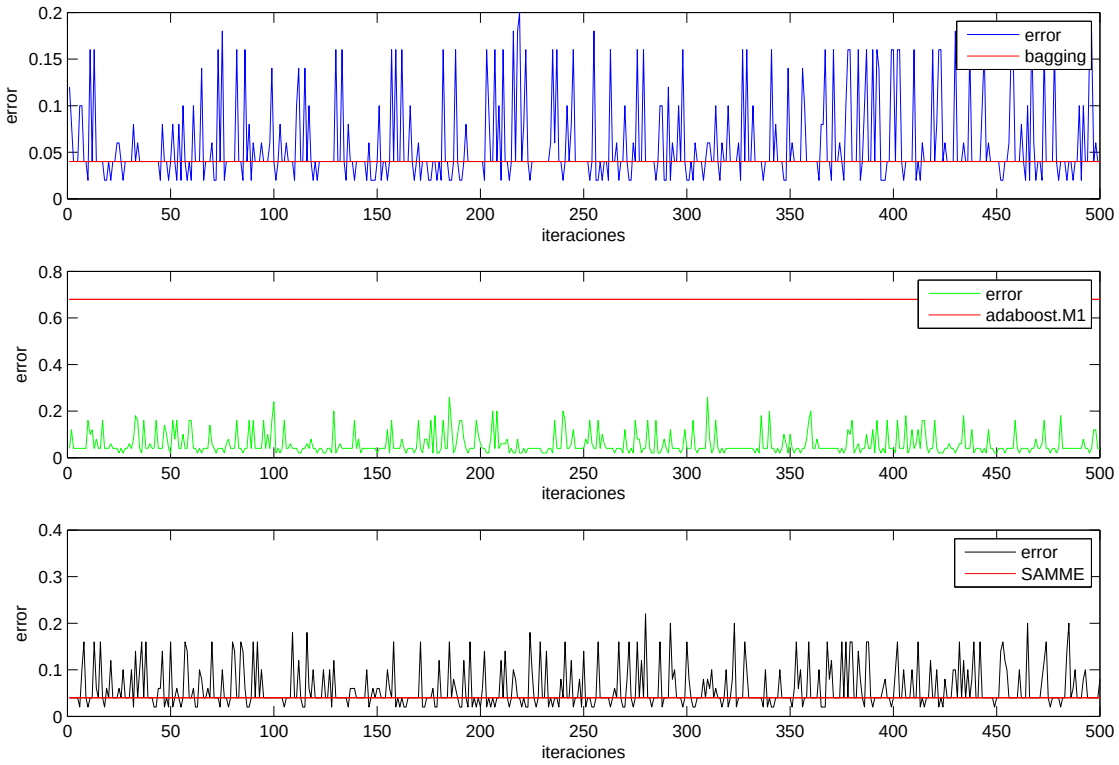

Figura 6: Errores bagging, adaboost.M1 y SAMME en 500 iteraciones para datos Iris

Breast

Luego de 100 iteraciones los valores del error de clasificación obtenidos en Bagging, Adaboost.M1 y SAMME son:

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

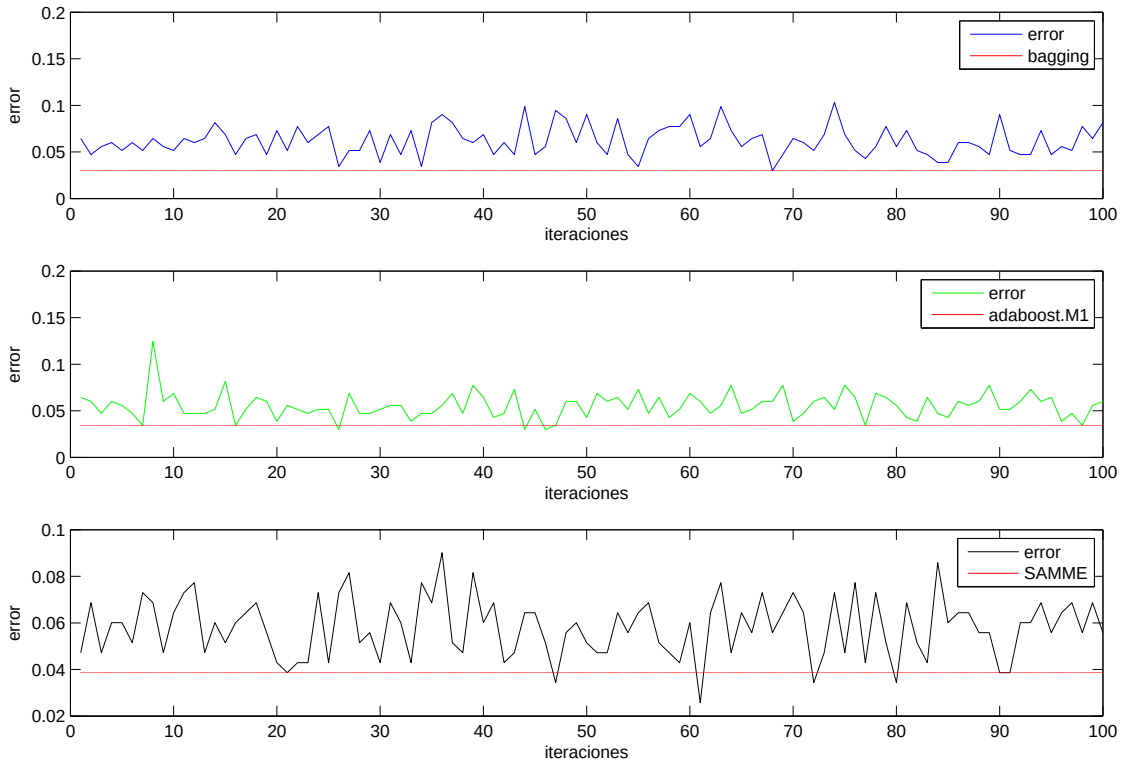

Figura 7: Errores bagging, adaboost.M1 y SAMME en 100 iteraciones para datos Breast

Vowel

Luego de 50 iteraciones los valores del error de clasificación obtenidos en Bagging, Adaboost.M1 y SAMME son:

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

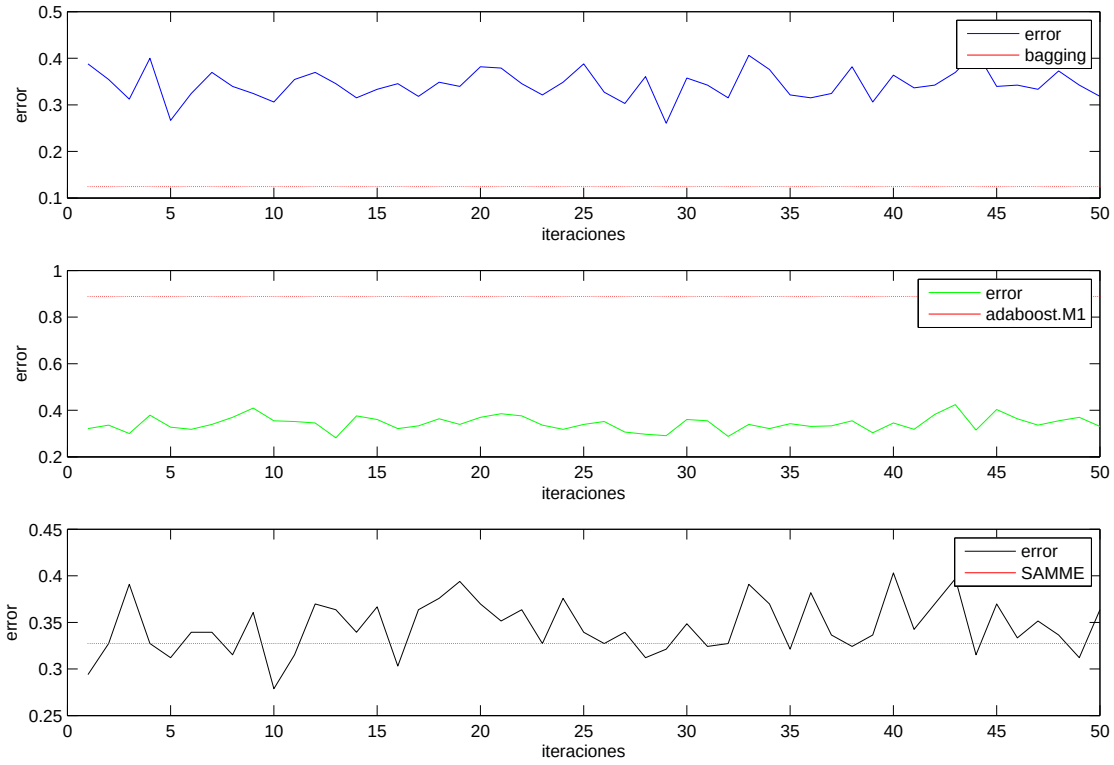

Figura 8: Errores bagging, adaboost.M1 y SAMME en 50 iteraciones para datos Vowel

El decrecimiento en el desempeño de los tres algoritmos aplicados a `Vowel2` se debe a que la muestra de entrenamiento es de menor tamaño que en `vowel` y además se realizó un menor número de iteraciones (30). La muestra de entrenamiento consta de 528 observaciones y la de testeo 462 escogidas de la siguiente forma: las primeras 66 filas corresponden a datos de la persona 1 ensayando los sonidos en 6 instancias diferentes, las segundas 66 filas corresponden a la persona 2, etc.) siendo 8 las personas elegidas para el entrenamiento y 7 para la muestra de testeo. Los resultados obtenidos son:

```
Error bagging: 0.54113
```

```
Error adaboost:0.90909
Error samme:   0.58009
```

6.3. Conclusiones

En el caso del problema de clasificación de dos clases, es decir al trabajar con los datos `Breast`, los tres algoritmos presentan porcentajes de error de clasificación del orden del 3 % que es además inferior al que se obtiene si se considera sólo un árbol de clasificación (5 %).

Cuando el problema es de varias clases los desempeños varían significativamente.

Para el problema de clasificación en tres clases (datos `Iris`) Bagging y SAMME alcanzan un error de clasificación de 0.04, mientras, que como era de esperarse, la performance de Adaboost.M1 es inferior dado que es un algoritmo desarrollado para un problema de clasificación binario.

Al considerar los datos `Vowel` para los cuales el número de clases es 11, los tres algoritmos se comportan de manera deficiente, en particular Adaboost.M1. lo cual es razonable. Los porcentajes de error de clasificación alcanzados por Bagging y SAMME son acordes con los que pueden encontrarse en la bibliografía. Como se menciona al describir las bases de datos, `Vowel` es un conjunto de datos difícil de clasificar.

Cabe destacar que en la bibliografía consultada el número de iteraciones por algoritmo es superior a las realizadas en este estudio, donde además no se dividió varias veces el conjunto de datos original en entrenamiento y testeo para calcular errores promedio puesto que ello supone un alto costo computacional.

Referencias

[1] Breiman, L. Bagging Predictors. *Machine Learning*, 24, 123-140 (1996)

[2] Breiman, L. Bias, Variance and Arcing Classifiers. *Technical Report*. Statistics Department, University of California. (1996)

[3] Dietterich, T, Bakiri,G. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research* 2, 263-286, (1995)

[4] Guruswami, V. Sahai, A. Multiclass Learning, Boosting, and Error-Correcting Codes. *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 145-155, ACM Press.(1999)

[5] Zhu, J. Rosset, S. , Zou, H. Hastie, T. Multiclass AdaBoost. (1996)