UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Curso Académico 2009/2010

Proyecto de Fin de Carrera

Generalización del algoritmo de boosting binario para más de dos clases

Autor: Omar Sanz Ordoñez

Tutores: María Eugenia Castellanos Nueda

1. Resumen

En este trabajo se ha creado una librería dentro del paquete estadístico R, para la generalización del algoritmo de boosting para más de dos clases.

Actualmente en este paquete estadístico existen librerías, por ejemplo gbm (generalized boosted regression models), en la que aparece la implementación de la técnica boosting para la clasificación en problemas de dos clases. Esto supone una limitación a la hora de usar esta técnica ya que en numerosas ocasiones los problemas de clasificación involucran un número mayor de clases.

La librería creada contiene varias funciones auxiliares para el apoyo a la función principal en la que está implementada la técnica boosting para más de dos clases.

Finalmente, y a modo de aplicación de la librería obtenida, se ha realizado un análisis de varios problemas de clasificación, comparando las tasas de error del algoritmo en nuestra librería con otro método de clasificación basado en arboles de clasificación. Este estudio aplicado ha servido también como validación de la librería implementada.

2. Índice

3. Introducción.

En la vida real mucha de la información con la que tenemos que tratar se encuentra en forma de patrones complejos: caras, texto escrito, piezas de música, vinos, coches, etc. En psicología uno de los principales problemas consiste en comprender los procesos biológicos y mentales que transforman los estímulos externos en experiencias perceptivas con significado. Dicho de otra forma, se trata de comprender qué mecanismo usamos para reconocer y clasificar un estímulo complejo. Hasta el momento, los procesos que nos permiten reconocer patrones complejos son todavía un misterio en muchos de sus aspectos. Sin embargo, se asume frecuentemente el siguiente esquema de funcionamiento: antes del reconocimiento, nuestros órganos sensitivos deben percibir primero un patrón; después, para representar una experiencia perceptiva con significado, el mismo patrón o un patrón de la misma clase debe haber sido percibido previamente; finalmente, se debe recordar la pasada percepción del patrón, y se debe establecer de alguna forma una correspondencia o equivalencia entre la percepción pasada y la presente. Desafortunadamente, las investigaciones acerca de cómo tienen lugar estas operaciones en nuestro cerebro y sistema nervioso todavía no han conducido a ningún modelo definitivo.

Con la llegada de los ordenadores, un nuevo horizonte lleno de posibilidades se abrió ante nuestros ojos. Estas máquinas son capaces de evaluar en un tiempo asequible operaciones matemáticas y estadísticas que permiten establecer relaciones entre observaciones pasadas y presentes. Además, es posible almacenar en ellos una gran cantidad de información que se puede recuperar de forma muy rápida cuando se necesite. Llegado este punto, todos los elementos necesarios en el proceso de reconocimiento de patrones estuvieron disponibles, así como la posibilidad de operar con ellos de forma eficiente, por ello surgió la idea de diseñar o programar máquinas que reconociesen patrones y pronto éste se hizo uno de los proyectos más ambiciosos y fascinantes en las ciencias de la información y de la informática.

3.1 Reconocimiento de patrones.

El Reconocimiento de Patrones es un término que se usa para denominar una gran variedad de etapas en una investigación, desde la formulación del problema, recolección de los datos, pasando por la discriminación, la clasificación, para terminar con el análisis de los resultados y su interpretación.

Existen diferentes tipos de clasificaciones posibles dentro de los métodos de reconocimiento de patrones; una posible clasificación es la siguiente:

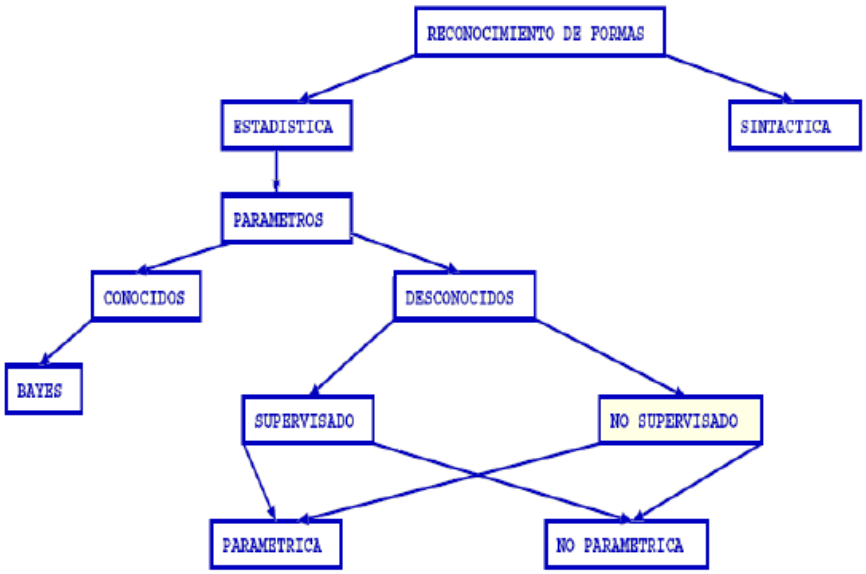


Figura 1. Tipos de reconocimiento de patrones

Como vemos en la figura 1 existen dos tipos de aproximaciones al reconocimiento de patrones que usan diferentes técnicas:

1. Aproximación estadística:

   Se representa cada patrón por un vector de números, y cada clase por uno o varios patrones prototipo usando métodos estadísticos. Se supone que patrones de una misma clase deberían estar cerca en el espacio de patrones, mientras que patrones de clases diferentes deberían estar alejados entre sí.

2. Aproximación sintáctica:

   Los patrones se descomponen en otros más sencillos, y se representan a través de los elementos básicos (terminales) y de ciertas reglas de su formación (gramática). Se estudia si un patrón pertenece al lenguaje asociado a una gramática.

Dentro de estos dos grandes grupos, existen otras subclasificaciones dependiendo de la naturaleza del problema o de las suposiciones que se puedan hacer. Existen además otros enfoques al problema del reconocimiento de patrones, que no son ni estadísticos ni sintácticos, como son:

- El método geométrico. Se centra en encontrar organizaciones o representaciones de datos junto con estructuras y algoritmos de datos asociados, que tengan un significado perceptible.

- Métodos state-space. Estos métodos están interesados en encontrar formas eficaces de búsqueda de las estructuras jerárquicas prevalecientes en muchas tareas de reconocimiento.

En este proyecto nos centramos en los métodos de clasificación enmarcados dentro del reconocimiento de patrones estadístico, con aprendizaje supervisado y técnicas no paramétricas.

Las fases del reconocimiento de patrones son:

1. Adquisición y representación del conocimiento.
2. Aprendizaje.
3. Clasificación.
4. Evaluación.

El preproceso es la primera etapa dentro de la adquisición y representación del conocimiento. Es en esta fase en la que determinamos con exactitud qué vamos a clasificar y dónde. Para ello, hemos de determinar las características qué vamos a observar en los objetos de interés, y cuál es el conjunto de etiquetas que queremos asignarles.

Los elementos de un problema de clasificación son:

1. Los patrones.

   Suponiendo que se trabaja con patrones m-dimensionales, un patrón, **X**, es un vector aleatorio m-dimensional:

   $\mathbf{X}=X_1, X_2,. . ., X_m$, con $X_i \in G_i$ para i = 1, 2, . . . , m.

   El espacio producto $G_i \times \cdots \times G_m$ se conoce como espacio de patrones.

2. Las clases consideradas.

   El primer problema a resolver en el contexto del reconocimiento de patrones es decidir qué características vamos a observar en los distintos objetos.

   Normalmente, se parte de un número grande de características, m, de manera que los patrones son observaciones de un vector aleatorio m-dimensional, y se pasa a un vector aleatorio d dimensional, con $d \leq m$. Este proceso se conoce como de selección y extracción de características. A continuación se comentan brevemente ambas técnicas:

   (a) Selección:

   Buscamos el conjunto de d características que maximiza cierta función que mide la eficacia del clasificador. Existen dos alternativas:

   • Métodos de búsqueda óptima: se evalúa la función para todos los posibles conjuntos de d características. Sólo es factible para problemas pequeños. El más importante es el método de ramificación y poda.

   • Métodos de búsqueda subóptima: se evalúan sólo ciertos subconjuntos.

   Es más eficiente computacionalmente. Existen varios métodos de búsqueda secuencial. En cuanto a la elección de la función a optimizar, ésta puede depender o no del clasificador considerado. En el primer caso, se consideran tasas de error sobre un conjunto de prototipos. En el segundo, se mira la distancia entre las clases.

(b) Extracción:

Produce una transformación lineal del conjunto inicial, obteniéndose y = Ax, siendo A una matriz d × n. Las técnicas más importantes son:

- Análisis de componentes principales.
- Transformaciones de Karhunen-Loeve.
- Análisis factorial.

3. Las clases informacionales.

El objetivo en los problemas enmarcados en el reconocimiento de patrones es asignar a cada patrón una clase informacional, w_C , perteneciente cierto conjunto de clases informacionales,

$\Omega = \{w_1, \ldots, w_j, \ldots, w_C\}$ siendo C el número de clases.

En la construcción de las clases informacionales, pueden aparecer una serie de problemas, cuya resolución depende de las características del problema:

- Podemos dudar si descomponemos una clase en subclases o la dejamos como está.
- Puede producirse un solapamiento entre las clases.

4. La clase de rechazo.

Normalmente se incorpora al conjunto de clases informacionales la llamada clase de rechazo, w_0, para el caso en el que la información disponible no permite hacer una clasificación fiable. Se obtiene entonces el conjunto ampliado de clases informacionales, $\Omega^* = \{w_1, \ldots, w_c, w_0\}$

5. El clasificador.

Un clasificador es una función d: $G_i \times \cdots \times G_d \rightarrow \Omega^*$ que asigna a cada patrón x un elemento d(x) del conjunto ampliado de clases informacionales.

Aunque la construcción del clasificador supone que ya hemos determinado las características de interés y las clases informacionales, el proceso en el reconocimiento de patrones no es necesariamente secuencial: a partir de los resultados de la clasificación podemos modificar éste, añadiendo o suprimiendo clases o cambiando el conjunto de características a observar.

Según tengamos constancia o no de un conjunto previo que permita al sistema aprender, la clasificación puede ser supervisada, parcialmente supervisada o no supervisada.

a) Clasificación supervisada o clasificación con aprendizaje.
Se basa en la disponibilidad de patrones de entrenamiento. Se trata de observaciones de las que se conoce a priori la clase a la que pertenecen y que servirán para generar una función que determine las clases (clases informacionales).

Si se tiene un conocimiento completo acerca de la estructura estadística de cada clase, debemos determinar los parámetros que caracterizan a las distribuciones de probabilidad. Se dice entonces que el aprendizaje supervisado es paramétrico.

Si el modelo estadístico es desconocido o no podemos asumir un modelo paramétrico concreto, se dice que el aprendizaje supervisado es no paramétrico.

Algunos métodos de clasificación supervisada son:

- Funciones discriminantes: En el caso de tratar con dos clases, se obtiene una función g tal que para un nuevo objeto O, si $g(O) \geq 0$ se asigna a la clase 1 y en otro caso a la 2. Si son múltiples clases se busca un conjunto de funciones g_i y el nuevo objeto se ubica en la clase donde la función tome el mayor valor. Estas funciones g, llamadas funciones discriminantes, se obtienen a partir del modelo paramétrico asumido para los patrones observados, en muchos casos normales multivariantes.

- Vecino más cercano: el método consiste en asignar a un nuevo patrón la clase en la que esté el objeto de la muestra original que más se le parece, este parecido se evalúa en función de distancias en el espacio de los patrones.

- Redes neuronales artificiales (RNA o en inglés ANN).
  Se supone que imitan a las redes neuronales reales en el desarrollo de tareas de aprendizaje.

- Árboles de clasificación. Un árbol de clasificación es un conjunto de condiciones organizadas en una estructura jerárquica, de tal manera que la decisión final a tomar se puede determinar siguiendo las condiciones que se cumplen desde el nodo raíz hasta alguna de sus hojas. Se explicará más detalladamente este método en la siguiente sección.

b) Clasificación parcialmente supervisada o aprendizaje parcial.
En estos problemas existe una muestra de objetos sólo en algunas de las clases definidas.

c) Clasificación no supervisada o clasificación sin aprendizaje.
Se utilizan algoritmos de clasificación automática multivariante en los que los individuos más próximos se van agrupando formando clases.

- Restringida: el número de clases en la que se estructurará la muestra está previamente definido.
- Libre: el número de clases en la que se estructurará la muestra depende exclusivamente de los datos.

Algunos métodos de clasificación no supervisada, son:

- Simple Link y Complete Link: parten de grupos unitarios de objetos y van uniendo los grupos más parecidos en cada etapa, hasta cumplir alguna condición.
- ISODATA: se van formando grupos que se ajustan iterativamente usando teoría de probabilidades. En algunas versiones se puede hacer la unión o división de algún grupo.
- C-means: se define un grupo de semillas, se asocia cada objeto al grupo de la semilla más parecida, se toman los centroides de cada grupo como nuevas semillas y se itera hasta que se estabilice.

- Criterios lógico-combinatorios: los criterios que se imponen a los grupos son tales como ser conexos, completos maximales, compactos, etc.

Los campos donde se aplican métodos de reconocimiento de patrones son numerosos. Mencionamos a continuación algunas de estas disciplinas, sin pretender dar una lista exhaustiva.

Comercio/Marketing:

- Identificar patrones de compra de los clientes.
- Buscar asociaciones entre clientes y características demográficas.
- Predecir respuesta a campañas de mailing.
- Análisis de cestas de la compra.

Banca:

- Detectar patrones de uso fraudulento de tarjetas de crédito.
- Identificar clientes leales.
- Predecir clientes con probabilidad de cambiar su afiliación.
- Determinar gasto en tarjeta de crédito por grupos.
- Encontrar correlaciones entre indicadores financieros.
- Identificar reglas de mercado de valores a partir de históricos.

Seguros y Salud Privada:

- Análisis de procedimientos médicos solicitados conjuntamente.
- Predecir qué clientes compran nuevas pólizas.
- Identificar patrones de comportamiento para clientes con riesgo.
- Identificar comportamiento fraudulento.

Transportes:

- Determinar la planificación de la distribución entre tiendas.
- Analizar patrones de carga.

Medicina:

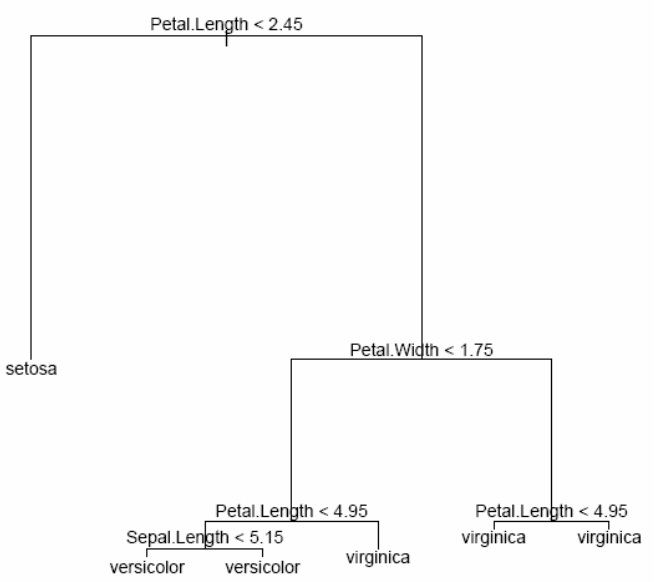
- Identificación de terapias médicas satisfactorias para diferentes enfermedades.
- Asociación de síntomas y clasificación diferencial de patologías.
- Estudio de factores (genéticos, precedentes, hábitos, alimenticios, etc.) de riesgo/ salud en distintas patologías.
- Segmentación de pacientes para una atención más inteligente según su grupo.
- Predicciones temporales de los centros asistenciales para el mejor uso de recursos, consultas, salas y habitaciones.
- Estudios epidemiológicos, análisis de rendimientos de campañas de información, prevención, sustitución de fármacos, etc.

3.2 Árboles de clasificación.

El uso de árboles de decisión tuvo su origen en las ciencias sociales con los trabajos de Sonquist y Morgan (1964) y Morgan y Messenger (1979) realizado en el Survey Research Center del Institute for Social Research de la Universidad de Michigan. El programa AID (Automatic Interaction Detection), de Sonquist, Baker y Morgan (1971), fue uno de los primeros métodos de ajuste de los datos basados en árboles de clasificación.

En estadística, Kass (1980) introdujo un algoritmo recursivo de clasificación no binario, llamado CHAID (Chi-square automatic interaction detection). Más tarde, Breiman y otros (1984) introdujeron un nuevo algoritmo para la construcción de arboles y los aplicaron a problemas de regresión y clasificación. El método es conocido como CART (Classification and regression trees) por sus siglas en inglés. Casi al mismo tiempo el proceso de inducción mediante árboles de decisión comenzó a ser usado por la comunidad de "Machine Learning" (Michalski, (1973), Quinlan (1983)) y la comunidad de "Pattern Recognition" (Henrichon y Fu, 1969). "Machine Learning" es una sub-área de Inteligencia Artificial que se sitúa dentro del campo de ciencias de la computación. "Pattern Recognition" se encuentra dentro del área de Ingeniería Eléctrica. Hoy en día el grupo de investigadores que más están contribuyendo al desarrollo de métodos de clasificación basados en árboles de clasificación se encuadra dentro del área de "Machine Learning".

El término árbol proviene de la representación gráfica, aunque los árboles son mostrados creciendo hacia la parte inferior de la página. La raíz es el nodo superior, en cada nodo se hace una partición hasta llegar a un nodo terminal u hoja. Cada nodo no-terminal contiene una pregunta en la cual se basa la división del nodo. Cada nodo terminal contiene el valor de la clase que le es asignada, usualmente denotada por el nombre de la clase.



En la figura 2 se puede ver un ejemplo de árbol de clasificación sobre los famosos datos Iris de Fisher y Anderson. Como podemos ver hay 6 nodos terminales, cada uno formando una partición que define una determinada clase.

Figura 2. Ejemplo de árbol de clasificación

Por ejemplo, si quisiéramos clasificar un patrón con medidas (6.8, 2.8, 4.8, 1.4) de las variables Sepal.Length, Sepal.Width, Petal.Length y Petal.Width, respectivamente. Como Petal.Length es mayor que 2.45, nos iríamos por el right child (el subárbol derecho).

Después, dado que Petal.Width es menor que 1.75, seguiríamos por la rama de la izquierda, otra vez miramos Petal.Length que es menor que 4.95 y por tanto la clasificación de este patrón es versicolor.

Existen numerosos algoritmos que implementan distintos métodos para la construcción de árboles, entre ellos destacamos los más usados:

- C4.5.

  Introducido por Quinlan (1983) dentro de la comunidad de "Machine Learning". Se basa en la utilización del criterio ratio de ganancia. Usando este método se consigue evitar que las variables con mayor número de posibles valores salgan beneficiadas en la selección. Además el algoritmo C4.5 incorpora una poda del árbol de clasificación una vez que este ha sido calculado. La poda está basada en la aplicación de un test de hipótesis que trata de responder a la pregunta de si merece la pena expandir o no una determinada rama.

• CHAID.

Significa “Chi-square automatic interaction detection”, fue introducido por Kass (1980) y es un derivado del THAID: “A sequential search program for the analysis of nominal scale dependent variables” (Morgan and Messenger, 1973). Está disponible como un módulo del paquete estadístico SPSS. El criterio para particionar está basado en χ^2 y para terminar el proceso se requiere definir de antemano un “threshold” (umbral).

• CART.

Introducido por Breiman et al. (1984), se trata de un método que construye árboles de decisiones binarios. Existe una versión similar llamada IndCART y que está disponible en el paquete IND distribuido por la NASA. El criterio para particionar es la impureza del nodo.

• Árboles Bayesianos.

Está basado en aplicación de métodos Bayesianos a árboles de decisión. Buntine (1992). Disponible en el paquete IND distribuido por la NASA.

• CN2. Introducido por Clark and Niblett (1988). Es una variante de ID, el cual se basa en la reducción de la entropía media para seleccionar el atributo que genera cada partición (cada nodo del árbol), seleccionando aquél con el que la reducción es máxima. Los nodos del árbol están etiquetados con nombres de atributos, las ramas con los posibles valores del atributo, y las hojas con las diferentes clases. La adaptación recogida en el método CN2 permite clasificar ejemplos con atributos que toman valores continuos.

Etapas en la construcción de un árbol:

1. Seleccionar una regla de partición en cada nodo interno. Esto significa determinar qué variable usamos, junto con el umbral, para definir una partición de los datos en un nodo concreto.

2. Determinar qué nodos son nodos terminales. Esto significa que para cada nodo tenemos que determinar si seguimos haciendo particiones, o si se trata de un nodo terminal y asignamos la clase que toque.

Si seguimos haciendo particiones hasta que los nodos terminales tengan todos patrones de la misma clase, es bastante probable que obtengamos un árbol grande que es poco robusto (es decir que a pequeños cambios en los datos observados cambie bastante), en inglés over-fit.

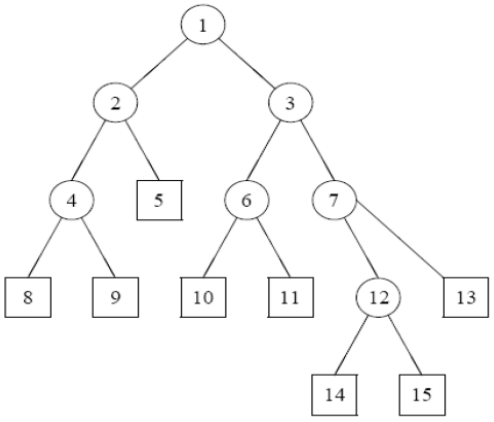
Si por el contrario los nodos terminales resultan relativamente impuros (es decir con mucha mezcla de clases), esto producirá árboles pequeños que son bastante pobres en cuanto a describir la variabilidad de los datos.

Descripción matemática de un árbol de clasificación.

Un árbol de clasificación, T, representa una partición recursiva del espacio de datos, realizada en base a un conjunto de patrones D = $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$, donde cada x_i es un vector compuesto por d variables.

T se define como:

- Un conjunto de enteros positivos;
- Dos funciones, l(t) de left y r(t) de right, definidas de T a T $\cup\{0\}$.
- Cada elemento de T corresponde a un nodo en el árbol. La siguiente figura muestra un árbol con los correspondientes valores de l(t) y r(t) para cada t en T.



t	$l(t)$	$r(t)$
1	2	3
2	4	5
3	6	7
4	8	9
5	0	0
6	10	11
7	12	13
8	0	0
9	0	0
10	0	0
11	0	0
12	14	15
13	0	0
14	0	0
15	0	0

Figura 3. Ejemplo de representación de un árbol

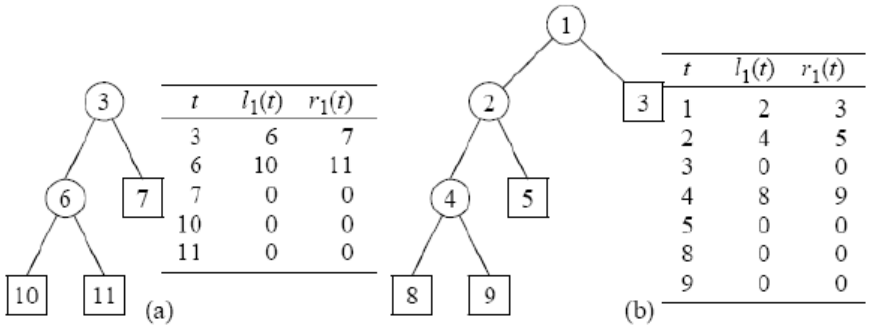
Para cada t ∈ T, ocurre que:

- l(t) = 0 y r(t) = 0 (si se trata de un nodo terminal),
- l(t) > 0 y r(t) > 0 (si es un nodo intermedio).

Aparte del nodo raíz, (denotado usualmente por t = 1), hay un único padre s ∈ T, para cada nodo t≠ 1. Un subárbol es un subconjunto no vacío T_1 de T, junto con dos funciones l_1 y r_1 que verifican: $l_1(t) = \begin{cases} l(t) & \text{si } l(t) \in T1 \\ 0 & \text{otro caso} \end{cases}$

Análogamente para r_1 (t).

A continuación aparecen varios ejemplos de subárboles.



t	$l_1(t)$	$r_1(t)$
3	6	7
6	10	11
7	0	0
10	0	0
11	0	0

t	$l_1(t)$	$r_1(t)$
1	2	3
2	4	5
3	0	0
4	8	9
5	0	0
8	0	0
9	0	0

Figura 4. Ejemplo de subárboles

Un subárbol podado T1 de T es un subárbol de T con el mismo nodo raíz. Esto se denota por T1 ≼ T. Vamos a denotar por $\tilde{T}$ al conjunto de nodos terminales.

Sea {u(t), t ∈ $\tilde{T}$} una partición del espacio de datos $\mathbb{R}^d$ (es decir, u(t) para un t concreto de un nodo terminal es un subespacio de $\mathbb{R}^d$, tal que u(t) ∩ u(s) = ∅ para t≠ s, con t, s ∈ $\tilde{T}$, y $\bigcup_{t\in\tilde{T}} u(t) = \mathbb{R}^d$).

Sean $w_{j(t)} \in \{w_1, \ldots, w_c\}$ las etiquetas que denotan la clase de cada nodo terminal. Un árbol de clasificación consiste en un árbol T conjuntamente con las etiquetas de las clases { $w_{j(t)}$, t ∈ $\tilde{T}$} y una partición {u(t), t ∈ $\tilde{T}$}.

Construcción de un árbol de clasificación

Un árbol de clasificación se construye usando un conjunto de patrones del que conocemos su clasificación, D = $\{x_1, \ldots, x_n\}$, vamos a llamar y_i, i = 1,. . ., n a la etiqueta del patrón *i*, y al global de ambas cosas lo llamamos $\mathcal{L} = \{(x_i, y_i), i = 1,\ldots, n\}$.

Para cada t ∈ $\widetilde{T}$, llamamos *N*(t) al número de muestras de $\mathcal{L}$, que verifican que $x_i \in$ u(t), y llamamos N_j(t) al número de muestras de $\mathcal{L}$, que verifican que $x_i \in$ u(t) y además $y_i = w_j$ (por tanto $\sum_{j=1}^{J} N_j(t) = N(t)$).

Entonces una estimación de p(x ∈ u(t)) basada en $\mathcal{L}$ es $\hat{p}$ (t) = $\frac{N(t)}{n}$

Análogamente un estimador de p(y = w_j | x ∈ u(t)) basado en $\mathcal{L}$ es: $\hat{p}(w_j|t) = \frac{N_j(t)}{N(t)}$

Podemos asignar etiquetas a cada nodo t, de acuerdo a la proporción de muestras de cada clase en u(t), así asignaremos la etiqueta w_j si se verifica que $\hat{p}\,(w_j|\,t) = \max_i\ \hat{p}\ (w_i|\,t)$.

Llamamos t_L = l(t), y t_R = r(t), y estimamos las probabilidades de p(x ∈ u(t_L) | x ∈ u(t)) (respectivamente para t_R) por $p_L = \hat{p}(t_L) = \frac{p(t_L)}{p(t)}$

Reglas de partición

Una regla de partición es una prescripción que nos dice qué variable (o combinación de variables), deben usarse en un nodo concreto para dividir las muestras en subgrupos, y también nos dice el umbral que debemos usar.

Así, una partición consiste en una condición en las coordenadas de un vector x ∈ $\mathbb{R}^d$, denominada s_d. En cada nodo no terminal, t, consideramos los datos x ∈ u(t). Entonces, si x ∈ s_d, el próximo paso en el árbol es a l(t), en otro caso a r(t). Para completar la regla de partición debemos determinar qué condición s_d elegimos.

Vamos a seguir el procedimiento propuesto en Breiman (1984). Se trata de considerar el índice de impureza I(t) (de un nodo ó de un árbol), definido como $I(t) = \emptyset\big(\hat{p}(w_1|t), \ldots, \hat{p}(w_C|t)\big)$, donde $\emptyset$ es una función definida sobre todas las

posibles J-tuplas, $p_1, \dots, p_C$, que verifican ser una distribución de probabilidad, es decir $p_i \geq 0$ y $\sum_{i=1}^{C} p_i = 1$, y que verifica:

1. $\emptyset$ es máxima (máxima impureza) si todas las clases tienen la misma proporción en t, es decir que las $p_i = 1/C$ para todo i.
2. Es mínima (mínima impureza) cuando en t sólo hay patrones de una clase, es decir, que para algún j, $p_i = 1\ y\ p_i = 0$ para el resto de i's.
3. Es una función simétrica $p_1, \dots, p_C$

Una medida de lo buena que es una partición, s_d, en un nodo t, es el cambio producido en el índice de impureza. Se trata de conseguir subconjuntos (en nodos hijos) con menor grado de impureza.
Una posible elección sobre todas las posibles particiones, es elegir una tal que el decrecimiento en la impureza sea máximo cuando nos movemos de un grupo a dos:
$\Delta I(s_d, t) = I(t) - (I(t_L)p_L + I(t_R)p_R)$

Algunas medidas de impureza son:

Índice de entropía $i(p) = -\sum_{j=1}^{J} p_j \log p_j$, con log(0)= 0.

Índice de Gini $i(p) = -\sum_{j \neq i} p_j p_i$.

Finalmente, para elegir s_d se consideran todas las posibles variables en el banco de datos y se evalúan para distintos umbrales de cada una de estas variables, los decrecimientos en la impureza al variar entre los mismos. Así se evalúan, para cada una de las variables $X_1 \dots X_d$, diferentes umbrales discretizando los valores posibles entre el mínimo y el máximo. Por ejemplo, para $X_i\ i = 1 \dots d$, consideramos los valores posibles en un grid: $a_1^i, \dots, a_l^i$ y probamos reglas del tipo $\{X_i \leq a_j^i, 1 \leq j \leq l\}$, eligiendo aquella que mayor decrecimiento produzca.

Por último, debemos determinar cuándo parar, es decir cuando declaramos a un nodo terminal, o por el contrario seguimos dividiendo los nodos.

1. Podemos seguir creciendo indefinidamente, hasta que cada nodo terminal tenga un patrón, (en este caso el error de estimación para el conjunto de entrenamiento será 0), pero tendremos un árbol poco robusto.

2. Otro criterio simple sería fijar un umbral de decrecimiento de impureza β. De nuevo esto puede conducir a malas soluciones (dependiendo si β es bajo o alto).
3. Una mejor estrategia es la poda.
(Michie, D., Siegelharter, D.J. & Taylor, C.C. (1994): "Machine Learning, Neural and Statistical Classification").

4. Objetivos.

El principal objetivo de este proyecto es la generalización del algoritmo de combinación basado en boosting para más de dos clases. El algoritmo binario implementado en la librería de R, gbm, es limitado ya que no nos permite clasificar problemas de reconocimiento con múltiples clases.

Otro objetivo es realizar la implementación de este algoritmo en un software libre, y concretamente en R, dada su flexibilidad y gran difusión.

Por último, también se demostrará en diversas aplicaciones, que el algoritmo implementado es más eficiente para realizar clasificación que los árboles de regresión.

4.1 Métodos basados en la combinación de clasificadores.

En este apartado se explican los métodos que combinan varios algoritmos con el fin de obtener un único método de clasificación. La combinación de clasificadores produce nuevos métodos que clasifican nuevos individuos combinando las decisiones de los algoritmos de los que están compuestos. Éstos se construyen en dos fases:

- En una primera fase, la de entrenamiento, se generan una serie de clasificadores individuales con un algoritmo base.
- En una segunda fase se combinan las distintas hipótesis generadas. La precisión del conjunto puede ser mucho mayor que la precisión de cada uno de los miembros de los que está compuesto.

Esta mejora se podrá obtener únicamente si los clasificadores individuales son suficientemente diversos debido a que si se combinan clasificadores idénticos no se obtendrá ninguna mejora, de hecho se obtendrá la misma respuesta que la basada en cada clasificador individual. Por tanto para construir un clasificador combinado, hay que elegir el algoritmo base y diseñar una metodología que sea capaz de construir clasificadores que cometan errores distintos en los datos de entrenamiento.

Ejemplos de este tipo de métodos son: Bagging y Boosting.

Ejemplos de la aplicación de clasificadores realizados a través de combinación de métodos son:

- Fusión de datos en biomedicina. Como monitores de cuidados coronarios y la segmentación de la imagen a través de ultrasonido para la detección de enfermedades en el esófago (Dawant y Garbay, 1999).
- Identificación de objetivos en el aire (Raju y Sarma, 1991).
- Biométrica. Se combinan los resultados de cinco métodos para la verificación de personas, según la imagen y la voz. (Chatzis, 1999).
- Modelado de procesos químicos. (Sridhar, 1996).

4.1.1. BAGGING (Breiman, 1996).

Este algoritmo produce réplicas del conjunto de entrenamiento (training) y entrena al clasificador con cada réplica creada. Se le aplica a cada clasificador un patrón de test denominado x, el cual es clasificado según el voto de la mayoría, resolviéndose de forma aleatoria los empates que se produzcan.

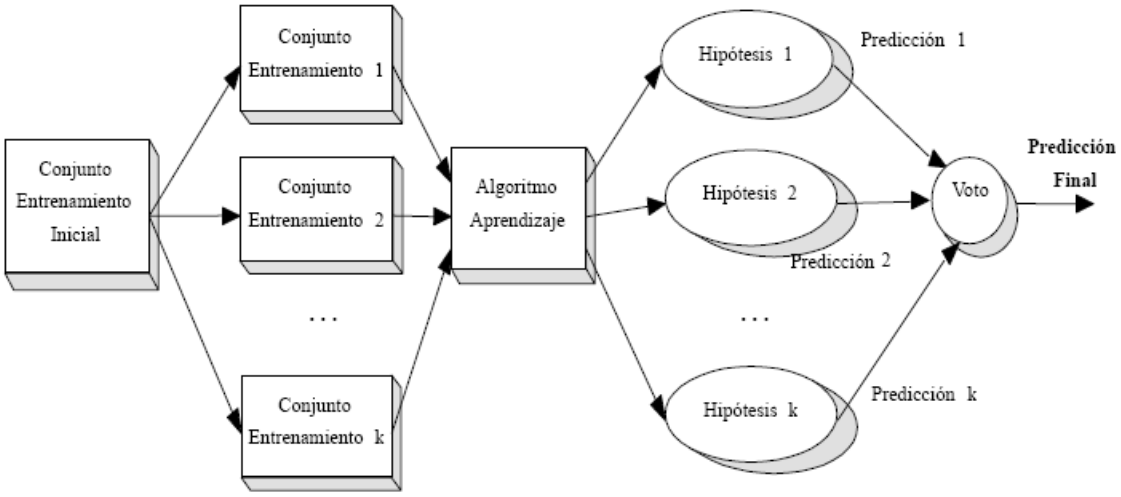


Figura 5. Esquema Bagging

Se generan K muestras bootstrap, cada una de ellas de tamaño $l \leq n$ a partir del conjunto de entrenamiento con reemplazamiento. El resultado de aplicar este procedimiento es la creación de nuevos conjuntos de entrenamiento, denominados Y^b para b=1… K, cada uno con tamaño l, siendo K el conjunto de datos bootstrap que son generados. Se obtiene un clasificador para cada conjunto de datos. El clasificador final es aquel cuya salida es la clase con mayor frecuencia predicha por todos los subclasificadores. Un esquema de este procedimiento aparece en la figura 5.

Supongamos que tenemos un conjunto de entrenamiento (x_i, w_i), donde i = 1,…, n, de x_i patrones y w_i clases.

1. Para cada b = 1,…, K, hacemos lo siguiente.
   (a) Generamos una muestra boostrap de tamaño l por muestreo con reemplazamiento desde el conjunto de entrenamiento, esta muestra se denota por Y^b, en este caso algunos patrones se repetirán, y otros se omitirán.
   (b) Diseñaremos un clasificador, η_b(x).
2. Clasificaremos un patrón de prueba denominado x, mediante la clase predicha para cada η_b(x), de forma que se le asigna a x la clase con mayor frecuencia entre todos los clasificadores obtenidos en el aparatado 1.

Figura 6. Algoritmo Bagging

Como los patrones del conjunto de entrenamiento se escogen de manera aleatoria, la probabilidad, para n grande, de obtener patrones no repetidos en una muestra boostrap es de un 63%. Esto hace que el procedimiento de Bagging sea inestable.

El procedimiento Bagging es particularmente útil en problemas de clasificación mediante redes neuronales y árboles de clasificación ya que éstos son procesos inestables, los clasificadores del vecino más próximo se mantienen estables por eso en éstos últimos no es útil el procedimiento Bagging. Sin embargo para los árboles, una característica negativa es que la interpretación del algoritmo Bagging final ya no tiene una interpretación simple como sucede en el caso de usar un solo árbol.

En los estudios sobre los clasificadores lineales, Skurichina (2001) obtuvo que el algoritmo bagging puede mejorar el rendimiento sobre los clasificadores construidos en tamaños críticos de muestras de entrenamiento, pero cuando el clasificador es estable, por lo general el algoritmo bagging es normalmente inútil. Además, para los tamaños de muestra muy grande, los clasificadores construidos con réplicas bootstrap son similares y la combinación no ofrece ningún beneficio.

El procedimiento, tal como se muestra en la Figura 6, se aplica a los clasificadores cuyas salidas son predicciones de clase. Para los métodos de clasificación que producen estimaciones de las probabilidades a posteriori, $\hat{p}(\omega_j|x)$, existen dos enfoques posibles.

Uno de ellos se basa en tomar una decisión para la clase en función del valor máximo $\hat{p}(\omega_j|x)$, y luego utilizar la votación. Por otra parte, las probabilidades a posteriori pueden ser calculadas entre todas las repeticiones boostrap, obteniéndose $p_K(\omega_j|x)$, y la decisión se basa en el valor máximo de $\hat{p}_K(\omega_j|x)$ obtenido. Breiman (1996) muestra una tasa de error en la clasificación prácticamente idéntico para los dos enfoques en una serie de experimentos con 11 conjuntos de datos. Sin embargo, las estimaciones de las probabilidades a posteriori basadas en todas las réplicas tienden a ser más exactas que los cálculos individuales.

En resumen, este algoritmo es particularmente efectivo con métodos inestables, es decir métodos en los que pequeñas modificaciones del conjunto de datos provocan cambios importantes en la hipótesis (ej.: árboles de decisión). Además, bagging puede mejorar haciendo más inestable el método (por ejemplo cuando se usan árboles, eliminando la poda). También se produce una ligera mejora si las hipótesis tienen asociada alguna medida de certeza como el voto ponderado. Generalmente, la tasa de error decrece con el nº de clasificadores, que puede llegar a ser muy grande (miles). Usar un número elevado de réplicas no produce un sobreajuste del método.

4.1.2. BOOSTING.

Es un algoritmo que combina clasificadores débiles, es decir, métodos de clasificación cuyas estimaciones de los parámetros suelen ser inexactas, a fin de lograr un mejor clasificador. Su gran diferencia con el algoritmo anteriormente explicado es que éste es un procedimiento determinista, que genera conjuntos de entrenamiento y clasificadores de forma secuencial, basándose en los resultados de la iteración anterior. Por el contrario, el algoritmo Bagging genera conjuntos de entrenamiento aleatoriamente y por lo tanto los clasificadores se pueden calcular en paralelo.

Fue propuesto por Freund y Schapire (1996), y se trata de un método que asigna a cada patrón del conjunto de entrenamiento un peso (lo que refleja su importancia), y construye un clasificador usando el conjunto de entrenamiento y el conjunto de pesos. Por lo tanto, es necesario un clasificador que pueda manejar pesos en el conjunto de entrenamiento. Algunos clasificadores son incapaces de apoyarse en patrones de peso. En este caso, un subconjunto de los patrones de entrenamiento puede ser muestreado de acuerdo a la distribución de los pesos y éstos se utilizan para entrenar el clasificador en la próxima etapa de la iteración.

El procedimiento básico de Boosting es AdaBoost (Adaptive Boosting; Freund y Schapire, 1996). En la Figura 7 se muestra el algoritmo básico de AdaBoost para un problema con clasificación binaria. Inicialmente, a todas las muestras de entrenamiento se les asigna un peso inicial igual, $p_i = 1/n$. En cada etapa del algoritmo, un clasificador $\eta_t(x)$ es construido con los pesos obtenidos en la etapa anterior. Entonces, el peso de los patrones clasificados erróneamente aumenta y el peso de los patrones clasificados correctamente disminuye. El efecto de esto es que, los patrones de mayor peso influyen para que el clasificador aprenda más, y así hace que el clasificador se centre más en las clasificaciones erróneas, es decir, en los patrones que están más cerca de las fronteras de decisión . Además existe un error, e_t , que es calculado como la suma de todos los pesos de las muestras mal clasificadas. Este elemento entra dentro del factor $(1 - \mathrm{e}_t)/\mathrm{e}_t$, que sirve para calcular los nuevos pesos aumentado el peso total de las muestras mal clasificadas (si $\mathrm{e}_t < 1/2$). Este proceso se repite en varias iteraciones, de forma que se genera un conjunto de clasificadores. Finalmente, los clasificadores

creados se combinan con una ponderación lineal cuyos coeficientes se calculan como parte del procedimiento de entrenamiento.

Inicializamos los pesos $p_i = 1/n \; i = 1, \ldots, n$

1. Para $t = 1, \ldots, T$,
   (a) Se construye un clasificador $\eta_t(x)$ a partir de los datos de entrenamiento con pesos $p_i, i = 1, \ldots, n$
   (b) Se calcula e_t como la suma de los pesos p_i correspondiente a los patrones mal clasificados;
   (c) Si $e_t > 0.5$ o $e_t = 0$ entonces terminamos el procedimiento, en caso contrario, $p_i = p_i (1 - e_t)/e_t$ para los patrones mal clasificados. Renormalizar los pesos para que su suma sea la unidad.

2. Para un clasificador de dos clases, en el que $\eta_t(x) = 1$ implica $x \in \omega_1$ y $\eta_t(x) = -1$ implica $x \in \omega_2$, se forma una suma ponderada de los clasificadores, η,

$$\hat{\eta} = \sum_{t=1}^{T} \log\left((1 - e_t)/e_t\right) \eta_t(x)$$

   y asignar $x \; a \; \omega_1$ si $\hat{\eta} > 0$

Figura 7. Algoritmo AdaBoost

4.1.3. BOOSTING PARA MÁS DE DOS CLASES.

En el presente proyecto el objetivo es implementar en una librería en R, paquete estadístico, una generalización del algoritmo de Adaboost para más de dos clases. Actualmente existe una librería, gbm (generalized boosted regression modeling), que para el caso de la clasificación implementa un algoritmo de clasificación binario. Dada la multitud de problemas en los que las clases son más de dos, nos planteamos la implementación de un algoritmo boosting para más de dos clases.

Para la generalización del algoritmo AdaBoost, se ha seguido el algoritmo de la figura 8. Este algoritmo se denomina AdaBoost.MH y fue propuesto por Schapire y Singer (1999). Se han realizado varios cambios, como por ejemplo, se amplía el conjunto de entrenamiento (de tamaño n) a un conjunto de entrenamiento de tamaño n x C pares, $((x_i,1),y_{i1})$, $((x_i,2),y_{i2})$, . . ., $((x_i,C),y_{iC})$, i= 1, …, n

Así, cada patrón de entrenamiento se replica C veces y se amplía con cada una de las etiquetas de la clase. Las nuevas etiquetas para un patrón (x, l) tomarán los valores

$$y_{il} = \begin{cases} +1 \; Si \; x_i \in clase \; \omega_l \\ -1 \; Si \; x_i \notin clase \; \omega_l, \; siendo \; i = 1 \dots n, l = 1 \dots C \end{cases}$$

Un clasificador, $\eta_t(x,l)$, es entrenado, usando los datos $\{y_{il}, i = 1 \dots n\}$ para cada $l = 1 \dots C$, el clasificador final, $\hat{\eta}(x,l)$, para la clase l , es una suma ponderada de los clasificadores construidos en cada etapa de la iteración. La decisión final confronta los distintos clasificadores, para cada clase $l = 1 \dots C$, de forma que se asigna x a la clase j si $\hat{\eta}(x,j) \geq \hat{\eta}(x,k) \; k = 1, \dots, C; \; k \neq j$

1. Inicializamos los pesos $p_{ij} = 1/(nC),\ \ i = 1, \dots, n;\ j = 1, \dots, C$
2. Para $t = 1, \dots, T$
   (a) Construimos un clasificador de confianza $\eta_t(x, l)$ desde el conjunto de datos con los pesos $p_{ij,}\ \ i = 1, \dots n;\ j = 1, \dots, C$

   (b) Calculamos $r_t = \sum_{i=1}^{n} \sum_{j=1}^{C} p_{il}\, y_{il}\, \eta_t\,(x_i, l)$

   Y $\alpha_t = \frac{1}{2} \log\left(\frac{1+r_t}{1-r_t}\right)$

   (c) Ajustamos los pesos $p_{ij} = p_{ij} \exp(-\alpha_t\, y_{ij}\, \eta_t(x_i, j))$
   y renormalizamos para que éstos sumen 1.

3. Calculamos $\hat{\eta}(x, l\,) = \sum_{t=1}^{T} \alpha_t\, \eta_t(x, l)$ y asignamos x a ω_j si $\hat{\eta}(x, j) \geq \hat{\eta}\,(x, k)\ \ k = 1, \dots, C;\ \ k \neq j$

Figura 8. Algoritmo AdaBoost Generalizado

4.2. Metodología empleada.

4.2.1. DESARROLLO DE UNA APLICACIÓN INFORMÁTICA.

Como toda aplicación informática, este proyecto ha sido desarrollado en diferentes etapas: planificación, en la cual se planifico los tiempos para este desarrollo; análisis de requisitos, en esta etapa se extrajo todos los requisitos tanto funcionales como no funcionales del proyecto, siendo en la siguiente etapa, diseño, donde se estudio la relaciones entre ellos. En la última etapa, se codificaron estos requisitos en funciones para el paquete estadístico R.

4.2.2. PROGRAMACIÓN ORIENTADA A OBJETOS.

La programación orientada a objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos.

Los objetos son entidades que combinan estado (atributo), comportamiento (método) e identidad:

- El estado está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos).

- El comportamiento está definido por los métodos (procedimientos) con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.

- La identidad es una propiedad de un objeto que lo diferencia del resto (es su identificador).

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener, y reutilizar.

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen

de mecanismos de interacción llamados métodos, que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el estado y el comportamiento.

Existe un acuerdo acerca de qué características contempla la "orientación a objetos", las características siguientes son las más importantes:

- Abstracción:

  Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

- Encapsulamiento:

  Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

- Principio de ocultación:

  Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de

abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

- Polimorfismo:

  Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica.

- Herencia:

  Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y éstas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple.

- Recolección de basura (o Garbage Collector):

  Es la técnica por la cual el ambiente de Objetos se encarga de destruir automáticamente, y por tanto desasignar de la memoria, los Objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo Objeto y la liberará cuando nadie lo esté usando.

Existen numerosos lenguajes orientados a objetos como por ejemplo:

- ABAP
- Ada
- C++
- Clipper
- D
- Delphi
- Eiffel
- Java
- JavaScript
- Oz
- R
- Perl
- PHP (a partir de su versión 5)
- PowerBuilder
- Python
- Ruby
- Visual FoxPro (en su versión 6)
- Visual Basic 6.0
- …

4.2.3. SOFTWARE LIBRE.

El software libre es aquel que puede ser distribuido, modificado, copiado y usado; por lo tanto, debe venir acompañado del código fuente para hacer efectivas las libertades que lo caracterizan.

Dentro de software libre hay, a su vez, matices que es necesario tener en cuenta. Por ejemplo, un software de dominio público significa que no está protegido por el copyright, por lo tanto, podrían generarse versiones no libres del mismo, en cambio el software libre protegido con copyleft impide a los redistribuidores incluir algún tipo de restricción a las libertades propias del software así concebido, es decir, garantiza que las modificaciones seguirían siendo software libre.

También es conveniente no confundir el software libre con el software gratuito, éste no cuesta nada, hecho que no lo convierte en software libre, porque no es una cuestión de precio, sino de libertad. Para comprender este concepto, debemos pensar en la acepción de libre como en "libertad de expresión". Se refiere especialmente a cuatro clases de libertad para los usuarios de software:

- Libertad 0: la libertad para ejecutar el programa sea cual sea nuestro propósito.
- Libertad 1: la libertad para estudiar el funcionamiento del programa y adaptarlo a tus necesidades (el acceso al código fuente es condición indispensable para esto).
- Libertad 2: la libertad para redistribuir copias y ayudar así a tu vecino.
- Libertad 3: la libertad para mejorar el programa y luego publicarlo para el bien de toda la comunidad (el acceso al código fuente es condición indispensable para esto).

Las principales ventajas del software libre son:

- Bajo costo de adquisición y libre uso.
- Innovación tecnológica.

  Los usuarios tienen un destacado papel al influir decisivamente en la dirección hacía donde evolucionan los programas: votando los errores que quieren que

sean corregidos, proponiendo nueva funcionalidad al programa, o contribuyendo ellos mismos en el desarrollo del software.

- Requisitos de hardware menores y durabilidad de las soluciones.

  Aunque resulta imposible generalizar, sí existen casos documentados que demuestran que las soluciones de software libre tienen unos requisitos de hardware menor, y por lo tanto son más baratas de implementar. Por ejemplo, los sistemas Linux que actúan de servidores pueden ser utilizados sin la interfaz gráfica, con la consecuente reducción de requisitos de hardware necesarios.
- Escrutinio público.
- Independencia del proveedor.

  El software libre garantiza una independencia con respecto al proveedor gracias a la disponibilidad del código fuente. Cualquier empresa o profesional, con los conocimientos adecuados, puede seguir ofreciendo desarrollo o servicios para nuestra aplicación.
- Industria local.
- Datos personales, privacidad y seguridad.
- Adaptación del software.
- …

Aunque también posee varias desventajas como:

- La curva de aprendizaje es mayor.
- El software libre no tiene garantía proveniente del autor.
- Se necesita dedicar recursos a la reparación de errores.
- No existen compañías únicas que respalden toda la tecnología.
- Las interfaces gráficas de usuario (GUI) y la multimedia apenas están desarrolladas.
- La mayoría de la configuración de hardware no es intuitiva.
- Únicamente los proyectos importantes y de trayectoria tienen buen soporte, tanto de los desarrolladores como de los usuarios.
- El usuario debe tener nociones de programación.
- En sistemas con acceso a Internet, se deben de monitorear constantemente las correcciones de errores de todos los programas que contengan dichos sistemas, ya que son fuentes potenciales de intrusión.

4.2.4. PAQUETE ESTADÍSTICO R.

Este proyecto se ha implementado en el entorno de programación R.
Como hemos visto anteriormente se engloba en el paradigma orientado a objetos.
A continuación, se realizará una breve introducción del mismo.

R es un entorno de programación y análisis estadístico y gráfico derivado del lenguaje de programación S (Becker, Chambers y Wilks, 1988; Chambers, 1998; Chambers y Hastie, 1992; Venables y Ripley, 2000). Existe una versión de este lenguaje distribuida por Insightful Corporation bajo el nombre comercial de S-Plus, y una versión libre con código abierto conocida como R. Esta última fue desarrollada por Ross Ihaka y Robert Gentleman (ésta es una de las razones del nombre R; Ihaka y Gentleman, 1996) del Departamento de Estadística de la Universidad de Auckland (Nueva Zelanda).
La primera versión de R se difundió rápidamente y la expansión hoy es irrefrenable. Desde su creación, R se alimenta y crece con los trabajos de investigadores provenientes de prácticamente todas las ramas del conocimiento. Las aportaciones desinteresadas de funciones y librerías de propósito tanto general como específico hacen de R un entorno dinámico, formado por una comunidad en movimiento continuo, que se inscribe dentro de la filosofía del software libre.

R, en tanto en cuanto software libre, se inscribe dentro del proyecto GNU, General Public Licence (Licencia Pública General, GNU). Se trata de una licencia creada por Free Software Foundation (Fundación para el software libre), organización fundada por Richard Matthew Stallmana en el año 1985. El principal propósito de la licencia GNU es declarar la libertad del uso, modificación y distribución del software y protegerlo de intentos de privatización que puedan de algún modo restringir su uso.

R es accesible a través de la web CRAN (Comprehensive R Archive Network; http://cran.r-project.org/), sitio oficial de R. Es la página base del proyecto R, desde la cual se puede descargar la última versión del programa, consultar manuales sobre R, obtener ayuda sobre su funcionamiento a través de un sistema de ayuda on-line, y, en definitiva, estar al corriente de los movimientos en este entorno de trabajo.

R es más que un software para el análisis de datos, es un entorno de trabajo que va incrementando continuamente sus capacidades con la incorporación de paquetes y funciones que se integran perfectamente en el sistema R. En la actualidad el entorno R está compuesto por más de 1.400 paquetes integrados.

Dispone de funciones básicas para los más elementales análisis descriptivos (media aritmética, desviación estándar, varianza…) y para los más complejos modelos formales derivados de los últimos avances en el campo de la estadística, psicometría, bioinformática, estadística bayesiana, bioestadística, minería de datos, econometría y finanzas, ecología, marketing, estadística robusta, sensometría, estadística espacial, estadística en las ciencias políticas, visualización y gráficos, análisis de redes neuronales y mucho más.

Las ventajas de R son numerosas:

- R es totalmente libre.
- R se ejecuta en Windows, MacOS, Linux, y muchas variantes de Unix.
- R no es compatible con cualquier actividad comercial, pero tiene un desarrollo de la comunidad muy activa.
- Contiene más de 1.400 paquetes integrados, lo que permite tener una gran funcionalidad.
- Posee un excelente sistema de ayuda.
- Posee un almacenamiento y manipulación efectiva de datos.
- Un conjunto de operadores para los cálculos con matrices, arrays, vectores…
- Posee unas excelentes capacidades gráficas para análisis de datos.
- Es fácil de ampliar con funciones escritas por el usuario.
- …

Uno de los motivos fundamentales por los que se ha elegido este software es porque existen numerosas librerías que contienen funciones eficientes para calcular árboles de clasificación. En particular hemos usado la librería tree en la que los árboles son construidos con el algoritmo CART.

Como se ha mencionado anteriormente, este proyecto se ha realizado con R, sin embargo se podría haber desarrollado con otras alternativas como por ejemplo:

- Matlab
- C
- Fortran
- …

5. Descripción informática.

5.1 Especificación de requisitos.

- Requisitos funcionales

  Requisito_F_1: Generar los índices ordenados de menor a mayor que formarán el conjunto de entrenamiento.

  Requisito_F_2: Generar el conjunto de entrenamiento.

  Requisito_F_3: Generar el conjunto de prueba.

  Requisito_F_4: Calcular los valores predichos por cada clase y cada patrón.

  Requisito_F_5: Generar una salida del programa sencilla y legible.

  Requisito_F_6: Mostrar el error de todos los algoritmos.

  Requisito_F_7: Siempre se usará las funciones con el orden de los parámetros descritos en la declaración de la misma.

- Requisitos no funcionales

  Requisito_NF_1: El software debe ser sencillo, rápido y de fácil manejo.

  Requisito_NF_2: El nombre de las funciones y de sus argumentos deben ser intuitivos.

  Requisito_NF_3: El lenguaje de programación usado en el desarrollo del programa será el paquete estadístico R.

  Requisito_NF_4: El entorno de desarrollo será R con una versión igual o superior a la versión 2.7.2.

5.2 Descripción de casos de uso.

En la figura 9 se muestra la relación existente entre las funciones implementadas en este proyecto para un buen funcionamiento de la librería.

Figura 9. Diagrama de casos de uso

En esta parte del proyecto se explicará cada función implementada, incluyendo las precondiciones y postcondiciones.

Nombre de la función: index.training

Precondiciones:
Los argumentos obligatorios son el conjunto de datos, y el porcentaje de los datos que formaran el conjunto de entrenamiento. La variable discriminante se encontrará en la última posición del banco de datos, si no es así los resultados no serán correctos.
Además el orden de los parámetros es el mencionado, cualquier otro orden producirá resultados erróneos.

Postcondiciones:
Se generan los índices ordenados de menor a mayor que después formarán el conjunto de entrenamiento.

Usuario	Sistema
1. Llama a la función index.training. 6. Asigna la salida de la función a una variable.	2. Modifica el nombre de la variable discrimante y etiqueta las clases con valores numéricos. 3. Genera los índices de manera aleatoria para una cantidad de datos fijada por el porcentaje pasado como parámetro. 4. Ordena los índices. 5. Devuelve los índices.

Nombre de la función: training.test

Precondiciones:

Los argumentos obligatorios son el conjunto de datos, y los índices que formarán el conjunto de entrenamiento, es decir la salida de la anterior función (index. training). Además el orden de los parámetros es el mencionado, cualquier otro orden producirá resultados erróneos.

Postcondiciones:

Se genera el conjunto de entrenamiento con los índices pasados como argumento.

Usuario	Sistema
1. Llama a la función training.test. 3. Asigna la salida de la función a una variable.	2. Devuelve un nuevo banco de datos formado solamente por los individuos correspondientes a los índices que se pasan como parámetro.

Nombre de la función: test.training

Precondiciones:
Los argumentos obligatorios son el conjunto de datos, y los índices que formarán el conjunto de entrenamiento, es decir la salida de la función (index.training).
Además el orden de los parámetros es el mencionado, cualquier otro orden producirá resultados erróneos.

Postcondiciones:
Se genera el conjunto de test con los índices que no formen parte de los índices del conjunto de entrenamiento, dichos índices se pasan como parámetro.

Usuario	Sistema
1. Llama a la función test.training. 3. Asigna la salida de la función a una variable.	2. Devuelve un nuevo banco de datos formado solamente por los individuos que no formen parte de los índices que se pasan como parámetro.

Nombre de la función: AdaBoost.MH

Precondiciones:

Los argumentos obligatorios son el conjunto de datos, el conjunto de test (la salida de la función test.training) y el número de iteraciones que realizará el algoritmo. La variable discriminante se encontrará en la última posición del banco de datos, si no es así los resultados no serán correctos.

Además el orden de los parámetros es el mencionado, cualquier otro orden producirá resultados erróneos.

Postcondiciones:

Se genera una matriz con los valores predichos para cada clase y cada individuo. El tamaño de esta matriz será n x C (siendo n el número de individuos y C el numero de clases).

Usuario	Sistema
1. Llama a la función AdaBoost.MH. 9. Asigna la salida de la función a una variable.	2. Modifica el nombre de la variable discrimante y etiqueta las clases con valores numéricos. 3. Crea los árboles de clasificación para cada clase y cada instante de tiempo. Todos los árboles son almacenados en una lista de árboles denominado etat. 4. Calcula el valor de rt según los pesos, la variable discriminante, Y, y los árboles de clasificación hallados anteriormente. 5. Calcula el valor de alphat en cada instante de tiempo según el valor de rt. 6. Actualiza y normaliza los pesos.

7. Realiza el test, es decir la predicción según los valores de eta.hat.

8. Devuelve eta.hat
(matriz de tamaño n x C (siendo n el número de individuos y C el número de clases) con los valores predichos).

Nombre de la función: AdaBoostMH.Class

Precondiciones:

El argumento obligatorio es la matriz con los valores predichos para cada individuo y clase (la salida de la función anterior AdaBoost.MH).

Postcondiciones:

Se genera una matriz con la clase predicha de cada individuo. El tamaño de esta matriz será n x 1 (siendo n el número de individuos).

Usuario	Sistema
1. Llama a la función AdaBoostMH.Class. 4. Asigna la salida de la función a una variable.	2. Se comparan los valores predichos para cada individuo y clase. La clase que posea mayor valor será la clase predicha. Formará una matriz con las clases predichas por individuo. 3. Devuelve la matriz formada con el valor de la clase predicha.

Nombre de la función: Join.Results

Precondiciones:
Los argumentos obligatorios son el conjunto de test (la salida de la función anterior test.trainig) y la matriz de clases (la salida de la función anterior AdaBoostMH.Class).

Postcondiciones:
Se genera una matriz con la combinación del conjunto de test que se pasa como parámetro, la variable que contiene la clase especificada con valor numérico y el vector de clases predichas por el clasificador. El tamaño de esta matriz será n x (m+2) (siendo n el número de individuos y m el número de variables del conjunto test).

Usuario	Sistema
1. Llama a la función Join.Results. 5. Asigna la salida de la función a una variable.	2. Modifica el nombre de la variable discrimante. 3. Genera una matriz con las clases de manera numérica. 4. Une la matriz de test, la matriz con las clases numéricas y la matriz de las clases predichas pasada como parámetro.

Nombre de la función: Find.Error

Precondiciones:
El argumento obligatorio es la matriz final, es decir, la salida de la función anterior Join.Results.

Postcondiciones:
Se genera el porcentaje de error evaluando el número de clasificaciones correctas e incorrectas.

Usuario	Sistema
1. Llama a la función Find.Error. 4. Asigna la salida de la función a una variable.	2. Calcula el número de clasificaciones incorrectas comparando los valores predichos con el valor real de la clase en cada patrón. 3. Devuelve el resultado multiplicando por 100 para obtener el porcentaje.

6. Aplicación: ejemplo de uso.

En esta apartado se presentan los resultados obtenidos en dos bancos de datos:

Iris: Es un banco de datos público, accesible en R a través de library(class) y data(iris). Son los datos usados por Fisher y Anderson. Este banco de datos contiene un conjunto de medidas en centímetros de las longitudes y anchuras del sépalo y pétalo de 50 flores de 3 especies distintas. Estas especies son setosa, versicolor, y virginica.

Fgl: Es un banco de datos público accesible en R a través de library(MASS) y data(fgl). Son los datos recogidos por B. German. Este banco de datos contiene un conjunto de medidas sobre fragmentos de vidrio recogidos en investigaciones forenses. Contiene 10 variables que son: RI (índice de refracción), Na (porcentaje de sodio), Mg (porcentaje de magnesio), Al (porcentaje de aluminio), Si (porcentaje de silicio), K (porcentaje de potasio), Ca (porcentaje de calcio), Ba (porcentaje de bario), Fe (porcentaje de hierro), type (tipo). Los fragmentos fueron clasificados inicialmente en siete tipos, uno de los cuales estuvo ausente en este conjunto de datos. Las categorías que se presentan son el vidrio de la ventana flotante (Winf: 70), vidrio de la ventana no flotante (WinNF: 76), vidrio de ventanas para vehículos (Veh: 17), contenedores (Con: 13), vajilla (Tabl: 9) y el vidrio de los faros de los vehículos (Responsable: 29).

Debido a que las funciones index.training y AdaBoost.MH dependen de dos parámetros elegidos por el usuario, como son el porcentaje de datos usados para realizar el entrenamiento y el número de iteraciones; se han realizado diferentes pruebas considerando distintos valores de estos parámetros.

En la siguiente tabla aparecen los resultados del error (en porcentaje) obtenido al realizar la clasificación con los datos Iris, usando el algoritmo AdaBoost y un árbol de clasificación sobre el mismo conjunto de entrenamiento y test; donde se han considerado diferentes porcentajes de datos para entrenar y diferentes número de iteraciones al calcular AdaBoost.

Porcentaje entrenamiento	Número de iteraciones	AdaBoost Generalizado	Árbol de clasificación
25	10	2.63 %	5.83
25	50	6.14 %	
25	100	6.14 %	
25	500	6.14 %	
25	1000	6.14 %	
25	2000	7.07 %	
50	10	4.00 %	5.01 %
50	50	13.33 %	
50	100	2.67 %	
50	500	2.67 %	
50	1000	2.67 %	
50	2000	2.67 %	
75	10	2.56 %	6.40 %
75	50	5.12 %	
75	100	2.56 %	
75	500	5.12 %	
75	1000	2.56 %	
75	2000	5.12 %	

Notar que, dado que el árbol de clasificación no depende del número de iteraciones el valor de error aproximado es el mismo al variar este parámetro.

Como se puede observar en la tabla, se obtiene en general una tasa menor de error con el método de combinación de clasificadores que con el árbol de clasificación.

En la siguiente tabla aparecen los resultados del error (en porcentaje) obtenido al realizar la clasificación con los datos Forensic Glass, usando el algoritmo AdaBoost y un árbol de clasificación sobre el mismo conjunto de entrenamiento y test:

Porcentaje entrenamiento	Número de iteraciones	AdaBoost Generalizado	Árbol de clasificación
25	10	8.77 %	34.04 %
25	50	32.09 %	
25	100	35.80 %	
25	500	30.24 %	
25	1000	35.80 %	
25	2000	40.12 %	
50	10	22.93 %	34.40 %
50	50	29.35 %	
50	100	28.44 %	
50	500	29.35 %	
50	1000	26.60 %	
50	2000	31.57 %	
75	10	26.32 %	27.57 %
75	50	19.28 %	
75	100	24.56 %	
75	500	15.78 %	
75	1000	14.03 %	
75	2000	26.31 %	

Como se puede observar en la tabla, se obtiene en general una tasa menor de error con el método de combinación de clasificadores que con el árbol de clasificación.

Este mismo banco de datos ha sido analizado en Schapire y Singer (1999) usando este mismo algoritmo y los autores obtenían errores en el conjunto test (50 %) en torno a 25- 28% para 1000 iteraciones. Este resultado es el que nosotros obtenemos con un conjunto test de 50% de los datos y 1000 iteraciones. Esto valida el buen funcionamiento del algoritmo implementado.

Ejemplo de código para obtener los resultados previos:

```
library(class)
data(iris)
# Probamos usando un 50% de los datos para construir el conjunto de entrenamiento.
index.iris=index.training(iris,50)
training.iris=training.test(iris,index.iris)
test.iris=test.training(iris,index.iris)
# Realizará 50 iteraciones de AdaBoost.MH.
resultado1.iris=AdaBoost.MH(training.iris,test.iris,50)
resultado2.iris=AdaBoostMH.Class(resultado1.iris)
resultadoFinal.iris=Join.Results(test.iris,resultado2.iris)
resultadoFinal.iris
error.iris=Find.Error(resultadoFinal.iris)
error.iris

library(MASS)
data(fgl)
# Probamos usando un 50% de los datos para construir el conjunto de entrenamiento.
index.fgl=index.training(fgl,50)
training.fgl=training.test(fgl,index.fgl)
test.fgl=test.training(fgl,index.fgl)
# Realizará 50 iteraciones de AdaBoost.MH.
resultado1.fgl=AdaBoost.MH(training.fgl,test.fgl,50)
resultado2.fgl=AdaBoostMH.Class(resultado1.fgl)
resultadoFinal.fgl=Join.Results(test.fgl,resultado2.fgl)
resultadoFinal.fgl
error.fgl=Find.Error(resultadoFinal.fgl)
error.fgl
```

7. Conclusiones.

7.1. Logros alcanzados.

Se ha implementado un algoritmo que permite usar el boosting para la clasificación cuando existen más de dos clases, en particular el algoritmo AdaBoost.MH (Andrew Webb. Statistical pattern recognition. 2002).

Se ha probado el algoritmo implementado en dos bases de datos, demostrando que éste funciona adecuadamente.

Tal y como se ha demostrado en los trabajos (Y. Freund y R. Schapire. Experiments with a new boosting algorithm. 1996, Y. Freund y R. Schapire, R.A short introduction to boosting. 1999), los métodos de combinación de clasificadores, como es el boosting, producen mejores resultados que los obtenidos con un solo clasificador débil, como el árbol de clasificación.

7.2. Trabajos futuros.

Entre las posibles líneas futuras de trabajo destacamos; la implementación en la función AdaBoost.MH de la posibilidad de realizar el boosting con varias clases usando validación cruzada (cross-validation); añadir la posibilidad de probar una secuencia de iteraciones de forma que se pueda seleccionar un número de iteraciones óptimo como aquel donde el error de clasificación en el test es el menor posible.

8. Bibliografía.

Breiman, L. (1996): Bagging predictors. *Machine Learning*, 26(2):123–140.

Breiman, L., Friedman, J.H., Olshen, R.A. y Stone, C.J. (1984): *Classification and Regression Trees*. Wadsworth. EE.UU.

Chatzis, V., Bors, A. G. y Pitas I. (1999): Multimodal decision-level fusion for person authentication. *IEEE Trans. on System, Man, and Cybernetics*, part A 29 (6), 674-680.

Dawant, B.M. y Garbay, C. (1999): Special Topic Section on Biomedical Data Fusion. *In. IEEE Trans. Biomed. Eng*. 46, 1169 - 1190

Duda, R.O., Hart, P.E. and Stork, D.G. (2001): *Pattern Classification*. 2nd edn., Wiley, New York.

Freund, Y. y Schapire, R. (1996): Experiments with a new boosting algorithm. *In Proceedings of the Thirteenth International Conference on Machine Learning.*

Freund, Y. y Schapire, R. (1999): A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5): 771- 780.

Hastie, T.J., Tibshirani, R.J. and Friedman, J.H. (2001): *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York.

Henrichon, E. G., Fu, Jr. y K. S. (1969): *Trans. Computers.*

Kass, G. (1980): An exploratory technique for investigating large quantities of categorical data. *Appl. Statist*., 24, 178-189.

Michie, D., Siegelharter, D.J. & Taylor, C.C. (1994): "*Machine Learning, Neural and Statistical Classification*". Ellis Horwood.

Morgan, J. N. y Messenger, R. (1973): *Thaid: A Sequential Analysis Program for the Analysis of a Nominal Scale Dependent Variable*. The University of Michigan Press.

Morgan, J. N. y Sonquist, J.A. (1963): Problems in the Analysis of Survey Data and a Proposal. *Journal of the American Statistical Association*. Vol. 58, septiembre, págs. 415-434.

Naga Raju, M. y Sarma, A.K. (1993): *A study of Industrial Development of Andhra Pradesh.*

Peña, D. (2002): *Análisis de datos multivariante*. McGrawHill.

Schapire, R. y Singer, Y. (1999): *Improving Boosting Algorithms Using Confidence-rated Predictions.*

Skurichina, M. (2000): *Stabilizing Weak Classifiers*.

Skurichina, M. y P. W. Duin, R. (2001): *Bagging and the Random Subspace Method for Redundant Feature Spaces.*

Sonquist, J.A. y Morgan, J.N. (1964): *The Detection of Interaction Effects.* Monografía núm. 35. Survey Research Centre, University of Michigan.

Quinlan, J.R. (1986): Induction of decision trees. *Machine Learning*.

Sonquist, J.A., Baker E.L. y Morgan J.N. (1971): *Searching for Structure*. Survey Research Centre, University of Michigan.

Webb, A. (2002): *Statistical pattern recognition*. Wiley

9. Anexo.

```
index.training <- function(data,percentage){

        col=ncol(data)
        names(data)[col]="Class"
        C=length(unique(data$Class))
        data$Class=as.numeric(data$Class)
        index.train=NULL
        for(c in 1:C){

        index.train=c(index.train,c(sample(which(data$Class==c),((length(which(data$Class==c)))*
        percentage)/100)))
        }
        return(sort(index.train))
}



training.test <- function(data,index.train){
        return(data[index.train,])
}



test.training <- function(data,index.train){
        return(data[-index.train,])
}



AdaBoost.MH <- function(data,test,T){

        library(tree)
        n=nrow(data)
        col=ncol(data)
        names(data)[col]="Class"
        C=length(unique(data$Class))
        w<<-matrix(1/(n*C),nrow=n,ncol=C,byrow=FALSE)

        dim.data=dim(data)
        data.increased=array(NA,dim=c(C,dim.data[1],dim.data[2]+1))
        dimnames(data.increased)=list(1:C,1:dim.data[1],c(colnames(data),"Y"))
        data$Class=as.numeric(data$Class)

        for(c in 1:C){
                aux1=as.numeric(data$Class==c)
                aux1[aux1==0]=-1
                aux=cbind(data,aux1)
                data.increased[c,,]=as.matrix(aux)
        }

        data.increased.final<<-data.increased[,,dimnames(data.increased)[[3]]!="Class"]
        prediction=matrix(NA,nrow=n,ncol=C)
        etat=NULL
        alphat=NULL
```

```
	for (t in 1:T){

		eta=NULL
		l<<-1
		for(l in 1:C){
			eta[[l]]=tree(Y~., data.frame(data.increased.final[l,,]), weights
				=100*w[,l]/sum(w[,l]))
		}
		etat[[t]]<- eta
		aux=0;
		for(l in 1:C){
			prediction[,l]=predict.tree(eta[[l]])
			aux=aux+sum(w[,l]*data.increased[l,,dim.data[2]+1]*prediction[,l])
		}
		rt=aux

		alphat[[t]] <- 1/2*log((1+rt)/(1-rt))

		for(l in 1:C){
			w[,l]=w[,l]*exp(-alphat[[t]]*data.increased[l,,dim.data[2]+1]*prediction[,l])
		}
		w=w/sum(w)
	}

	test=as.data.frame(test)
	colnames(test)=dimnames(data.increased)[[3]][1:col-1]
	eta.hat=matrix(rep(0,C*nrow(test)),nrow=nrow(test),ncol=C,byrow=TRUE);
	prediction=matrix(NA,nrow=nrow(test),ncol=C)
	for(c in 1:C){
		for (t in 1:T){
				prediction=predict.tree(etat[[t]][[c]], newdata =test)
				eta.hat[,c]=eta.hat[,c]+alphat[[t]]*prediction
		}
	}
	return(eta.hat)

}



AdaBoostMH.Class <- function(eta.hat){
	classifier=matrix(NA,nrow=nrow(eta.hat),ncol=1)
	for(num in 1:nrow(eta.hat)){
		classifier[num,]=which(max(eta.hat[num,])==eta.hat[num,])
	}
	return(classifier)
}


Join.Results <- function(test, Classifier){
	col=ncol(test)
	aux=names(test)[col]
	names(test)[col]="Class"
	Class=matrix(NA,nrow=nrow(test),ncol=1)
	Class=as.numeric(test$Class)
	names(test)[col]=aux
	return(test<-cbind(test,Class,Classifier))
}
```

```
Find.Error <- function(test){
        aux=0
        col=ncol(test)
        row=nrow(test)
        for(i in 1:row){
                if(test[i,col-1]!=test[i,col])
                        aux=aux+1
        }
        return((aux/row)*100)
}
```