Revista Electrónica de Comunicaciones y Trabajos de ASEPUMA. **Rect@**

Volumen 11. Paginas 139 a 154

USO DEL METAHEURÍSTICO GRASP EN LA CONSTRUCCIÓN DE ÁRBOLES DE CLASIFICACIÓN

Joaquín Pacheco[1], Esteban Alfaro[2], Silvia Casado[1], Matías Gámez[2] y Noelia García[2]

(1) Departamento de Economía Aplicada. Universidad de Burgos;

(2) Facultad de Ciencias Económicas y Empresariales. Universidad de Castilla La Mancha

Recibido 26/04/2010
Revisado 07/06/2010
Aceptación 10/09/2010

RESUMEN: En este trabajo se propone un nuevo método para la construcción de árboles binarios de clasificación. El objetivo es la construcción de árboles sencillos, es decir, con la menor complejidad posible, lo cual hace que sean de fácil interpretación y propicia el equilibrio entre optimización y generalización en los conjuntos test. El método propuesto se basa en la estrategia metaheurística GRASP usada en la literatura en problemas de optimización. El método básicamente modifica la forma de elección del atributo que determina la partición en cada nodo. Para ello incorpora aleatoriedad de forma controlada. A través de una serie de experimentos computacionales se compara nuestro método GRASP con la forma tradicional de seleccionar atributos. Se puede concluir que nuestro método GRASP, con pequeños niveles de aleatoriedad, consigue árboles significativamente menos complejos que los obtenidos con la forma tradicional.

Palabras claves: Árboles de Clasificación; Metaheurísticos; GRASP.

ABSTRACT: This paper proposes a new method for constructing binary classification trees. The aim is to build simple trees, i.e. trees which are as uncomplicate as possible, thereby facilitating interpretation and favouring the balance between optimization and generalization in the test data sets. The proposed method is based on the metaheuristic strategy known as GRASP in conjunction with optimization tasks. Basically, this method modifies the criterion for selecting the attributes that determine the split in each node. In order to do so, a certain amount of randomisation is incorporated in a controlled way. We compare our method with the traditional method by means of a set of computational experiments. We conclude that the GRASP method (for small levels of randomness) significantly reduces tree complexity without decreasing classification accuracy.

Keywords: Decision Trees; Metaheuristics; GRASP

1. Introducción

En el problema de clasificación se tiene un conjunto de *n* individuos o casos donde cada uno de ellos viene caracterizado por un cierto número *m* de variables y de los que se conoce su clase o clasificación correcta. A través de este conjunto de individuos, denominado conjunto de entrenamiento, se pretende diseñar y generalizar el conjunto de reglas que permitan clasificarlos con la mayor precisión posible.

Existen varias metodologías para abordar el problema de clasificación: redes neuronales (Sanchez, 1997), árboles de clasificación (Breiman *et al*, 1984.), análisis discriminante clásico (Huberty, 1994), regresión logística (Komarek, 2004), etc. Los árboles de clasificación o modelos basados en árboles se caracterizan por la sencillez de su representación y de su forma de actuar, además de la fácil interpretación dado que la información que se extrae del árbol se puede poner en forma de reglas de decisión.

Un árbol de clasificación se representa gráficamente mediante nodos y ramas. Cada nodo simboliza una cuestión o decisión sobre una de las características de los ejemplos. El nodo inicial se suele llamar nodo raíz. De cada nodo pueden salir dos o más ramas, dependiendo de que la respuesta a la cuestión planteada sea binaria o no. Finalmente, se alcanzan los nodos terminales u hojas y se toma una decisión sobre la clase a asignar. Cuando se presente un nuevo ejemplo o patrón al árbol, éste lo filtrará a lo largo de los tests que contienen los nodos. Cada test tiene salidas mutuamente excluyentes y exhaustivas, lo que significa que los ejemplos que se asignen a una de las salidas, no se pueden asignar a otra y además todos los ejemplos se asignarán a una de las salidas. Es decir, ningún ejemplo se asignará a dos salidas de un mismo test, pero tampoco habrá ejemplos que no se asignen a alguna de las salidas. Los árboles de decisión pueden trabajar tanto con variables continuas como con variables categóricas, de dos o más categorías.

Para la construcción del árbol en cada nodo se establece unas condiciones sobre un atributo, dividiendo o particionando así el conjunto de casos en subconjuntos que cumplen cada condición. Los subconjuntos se vuelven a dividir añadiendo nuevos niveles al árbol hasta detenerse mediante algún criterio. El criterio de parada detiene el proceso de división de nodos, cuando un nodo cumple esta condición se dice que es un nodo terminal u hoja. Los casos pertenecientes a un nodo hoja tendrán cierta homogeneidad, por lo que al nodo se le asigna una etiqueta con la clase mayoritaria y si todos los ejemplos del nodo hoja son de la misma clase se dice que es un nodo puro.

De entre todas las posibles particiones que se pueden hacer, se selecciona la que permita obtener un mejor resultado, es decir, que obtenga un mayor incremento en la homogeneidad o pureza de los nodos hijos con respecto al nodo padre. Por ello se evalúan todos los atributos disponibles para seleccionar aquél que permita dividir el conjunto original en subconjuntos que sean lo más homogéneos posibles en cuanto a la variable objetivo o clase. Esto nos lleva a la necesidad de establecer una medida de

impureza que debería ser cero si el nodo es puro y máxima cuando todas las clases tengan la misma proporción de casos (Breiman *et al.*, 1984).

Se ha observado como, cuando se desarrolla el árbol hasta alcanzar hojas puras o muy pequeñas se corre el riesgo de sobreajuste o sobreaprendizaje. Esto se debe a que en las últimas divisiones la selección de la partición más adecuada se hace en base a un conjunto de ejemplos muy pequeño. De ahí que construir árboles complicados o con un número de nodos demasiado grande puede llevar a la obtención de un porcentaje de aciertos muy elevado en el conjunto de entrenamiento y sin embargo desfavorecer en gran medida la generalización, es decir que el porcentaje de aciertos se reduce considerablemente cuando estamos ante casos distintos de los del conjunto de entrenamiento. Cuando se diseñan métodos de clasificación no sólo es importante obtener un porcentaje de aciertos elevado en el conjunto de entrenamiento sino que también hay que tener en cuenta la capacidad de generalización, es decir que se puedan extraer unas reglas generales que sirvan para clasificar correctamente conjuntos de datos distintos de los que integran el conjunto de entrenamiento. Se ha observado que un excesivo aumento en la complejidad del árbol conlleva a menudo un descenso en la capacidad de generalización (ver en Weiss y Kulikowski, 1991, pág. 126).

De ahí, que sea interesante construir árboles sencillos que sean eficaces en la clasificación, esto es, que con el menor número de nodos posibles consigan obtener un buen porcentaje de aciertos en la clasificación. El objeto de este trabajo es plantear y tratar de resolver el siguiente problema: dado un nivel de aciertos prefijado en el conjunto de entrenamiento se trata de construir el árbol que alcance ese nivel con el mínimo número de nodos terminales posible. Es por tanto un problema de optimización. Concretamente en este trabajo se va a considerar un nivel de aciertos del 100%, aunque el método que se va a proponer puede ser adaptado fácilmente a otros niveles previos. Se ha de indicar que en este trabajo se van a considerar árboles binarios, que quizás sean los más populares (ver en Duda, Hart y Stork, 2000 pág 397) y pueden ser usados por variables numéricas y/o categóricas, bien de tipo ordinal o nominal.

A diferencia de los métodos tradicionales que eligen la mejor partición posible (i.e. atributo) según la medida o criterio de reducción de la impureza que se use, en este trabajo cuando se construye el árbol, de entre todas las posibles particiones que se pueden hacer no se selecciona la mejor, sino que se selecciona una de las mejores aleatoriamente hasta obtener un árbol. Este proceso se repite en varias iteraciones lo que permite obtener diferentes árboles y elegir el mejor, es decir, el de menor número de hojas terminales. Así es como actúa el algoritmo usado en este trabajo, que está basado en la conocida estrategia GRASP (Greedy Randomize Adaptive Search Procedure) para problemas de optimización. GRASP es una estrategia metaheurística que inicialmente se dio a conocer en el trabajo de Feo y Resende (1989) a finales de los ochenta y se trata de procedimientos de búsqueda basados en funciones ávidas, aleatorias y adaptativas.

El uso de heurísticos y metaheurísticos en estadística y minería de datos esta tomando gran relevancia en la actualidad y tiene un futuro prometedor (Winker y Gilli; 2004). Existen numerosos trabajos realizados en estos últimos años. Por ejemplo podemos destacar los siguientes: en clasificación (Belacel, Raval y Punnen, 2007; Pendharkar, 2006), en selección de variables para clasificación (Garcia et al. 2006; Pacheco et al., 2006; Yang y Olafsson, 2006; Pacheco, Casado y Núñez, 2009), en clusters (Belacel, Hansen y Mladenović, 2003; Pacheco y Valencia, 2003; Woodruff y Reiners, 2004), en análisis en componentes principales (Cadima, Cerdeira y Minhoto, 2004), en modelos de predicción y regresión (Gatu y Kontoghiorghes, 2003, 2005; Hofmann, Gatu y Kontoghiorghes, 2007; Baragona, Battaglia y Cucina, 2004; Kapetanios, 2007), etc .

El trabajo se organiza como sigue: en la siguiente sección se describen los algoritmos usados tradicionalmente para construir árboles y en la sección 3 se describe el algoritmo GRASP usado para construir el árbol en este trabajo. En la sección 4 se exponen las experiencias computacionales realizadas y finalmente en la sección 5 se exponen las principales conclusiones del trabajo.

2. Descripción de los algoritmos tradicionales

Sea A = $\{a_1, a_2, \ldots, a_n\}$ el conjunto de entrenamiento, V = {1, 2, …, m} el conjunto de variables o atributos, y x_{ij} el valor del caso *i* en la variable *j*. Asimismo, sea K el número de clases. De ahora en adelante se considerará que los atributos son de tipo numérico, aunque entendemos que los métodos descritos se pueden adaptar muy fácilmente a atributos binarios y ordinales.

Los algoritmos para construir árboles de clasificación binarios parten del nodo inicial que tiene asignado todo el conjunto de entrenamiento A y se divide este conjunto en 2 subconjuntos. Estos 2 subconjuntos se asignan a los 2 nodos de las ramas salientes. En cada nodo se repite esta operación. Este proceso recursivo finaliza cuando se alcanza un criterio de parada preestablecido (todos los elementos del subconjunto del nodo pertenecen a la misma clase o al menos hay cierto grado de homogeneidad). En ese caso este nodo se dice que es terminal u hoja y se le asigna la clase mayoritaria en la partición.

La partición en cada nodo se realiza según un atributo *j* y un valor h_j. De forma que el correspondiente subconjunto $T \subset A$ se divide en dos T_1 y T_2 de la siguiente manera $T_1 = T \cap \{a_i / x_{ij} \leq h_j\}$ y $T_2 = T \cap \{a_i / x_{ij} > h_j\}$. Por tanto antes de realizar la partición hay que encontrar el atributo y valor (también denominado "punto de corte" de dicho atributo) que más reduzca la impureza en los nodos salientes.

Según lo comentado anteriormente, y sin entrar en detalles de programación y sin excluir alternativas o variantes similares a esta, cada nodo tiene asociado una variable binaria que se denotará *Terminal*, que tomará el valor 0 si es un nodo terminal o 1 en caso contrario.

- *Si Terminal = 0, tiene definida el campo clase*

- *Si Terminal = 1, tiene definidas los campos:*

(a) atributo (entero de 1 a m)
(b) punto_corte (h_j), y
(c) nodo_hijo_izda, nodo_hijo_derecha (los nodos salientes).

Es por tanto una estructura recursiva que finaliza si *Terminal* = 0. Dado un nodo *nodo_x* y su correspondiente subconjunto de casos $T \subset A$, los algoritmos tradicionales para construir árboles se basan en el siguiente método recursivo:

Procedimiento Fijar_caracteristicas_nodo(T; *nodo_x*)
Si se dan condiciones de parada entonces hacer:
Terminal = 0 y *clase* = clase mayoritaria en T
En caso contrario,
a) *Terminal* =1
b) Para j = 1,2,...,*m*: Determinar el valor h_j con el que se alcanza el mayor incremento de pureza Δ_j en las particiones determinadas por ese atributo
c) Determinar j* = argmax$\{\Delta_j / j = 1,2,\ldots,m\}$
d) Hacer *atributo* = j* y *punto_corte* = h_j
e) Hacer $T_1 = T \cap \{a_i / x_{ij^*} \leq h_{j^*}\}$ y $T_2 = T \cap \{a_i / x_{ij^*} > h_{j^*}\}$
e) Ejecutar Fijar_caracteristicas_nodo(T_1, *nodo_hijo_izda*) y
Fijar_caracteristicas_nodo(T_2, *nodo_hijo_derecha*)

---

de forma que el árbol se construye ejecutando
Fijar_caracteristicas_nodo(A, *nodo_inicial*).

Como se ha definido en el pseudocódigo, Δ_j es el mayor incremento de pureza o, por decirlo de otra manera, reducción de impureza, que se puede conseguir con el atributo j. Se denota por h_j el valor con el que se alcanza esta máxima reducción. Los principales algoritmos de la literatura se diferencian unos de otros básicamente en las funciones para medir la impureza y los criterios usados para determinar los mejores atributos y valores. También han surgido variantes según la forma de seleccionar los valores candidatos a punto de corte de cada atributo.

En función de cual sea la medida de impureza se distinguen diversos algoritmos. Así Breinman *et al*. (1984) propuso la función de Gini como medida de impureza en árboles de clasificación y regresión (CART, Clasification And Regression Trees) y alternativamente Quinlan propuso los algoritmos ID3 en 1986 (Quinlan, 1986) y C4.5 en 1993 (Quinlan, 1993) que usan como medida de impureza la entropía. Concretamente hacen uso de dos criterios basados en la entropía para valorar particiones, *Information*

Gain en el caso del algoritmo ID3 y *Gain Ratio* es usado en el algoritmo C4.5. A continuación se definen estas funciones y criterios.

Sea $T \subset A$ un conjunto de n_T casos y n_k , el número de elementos o casos de la clase k en T, para k=1,…,K, se define :

$$\text{Gini(T)} = \text{GI(T)} = 1 - \sum_{i=1}^{K} \left(\frac{n_k}{n_T} \right)^2 . \tag{1}$$

$$\text{Entropía (T)} = \text{E(T)} = - \sum_{k=1}^{K} \frac{n_k}{n_T} \cdot \log_2 \left(\frac{n_k}{n_T} \right) . \tag{2}$$

Los criterios más comúnmente usados para valorar una partición se definen como sigue. Sea un conjunto $T \subset A$ y una partición de T en dos subconjuntos T_1 y T_2, sean $n_{T1} = |T_1|$, $n_{T2} = |T_2|$ y $n_T = |T|$, los valores *Information Gain* (*G*) y *Gain Ratio* (*R*) de dicha partición se definen como:

$$G = E(T) - \left[\frac{n_{T1}}{n_T} \cdot E(T_1) + \frac{n_{T2}}{n_T} E(T_2) \right] \tag{3}$$

$$R = \frac{G}{I} \tag{4}$$

donde

$$I = -\frac{n_{T1}}{n_T} \cdot \log_2 \left(\frac{n_{T1}}{n_T} \right) - \frac{n_{T2}}{n_T} \cdot \log_2 \left(\frac{n_{T2}}{n_T} \right) \tag{5}$$

En este trabajo se toma como medida de reducción de la impureza el valor de *Información Gain*, y por tanto este es el criterio para elegir los atributos y sus puntos de corte. En cuanto a la forma de determinar el mejor punto de corte h_j para cada atributo, de forma general se opera de la forma siguiente: Las instancias se ordenan según el valor del atributo en orden ascendente y se calcula el punto medio entre cada par de valores. Para cada uno de estos puntos medios se calcula la impureza de las particiones correspondientes y se selecciona el punto medio mejor según el criterio considerado. De forma que en el nodo inicial se examinan *n*-1 puntos medios. Así actúa el algoritmo C4.5, que aunque ha tenido mucho éxito, no funciona bien en la búsqueda del punto de corte con atributos numéricos (Fayyad y Irani, 1993). Un método de selección alternativo fue propuesto en Fayyad y Irani (1992) que se basa en la premisa de que sólo es necesario considerar como candidatos los puntos en los cuales hay un cambio de clase. Con este método, conocido como método de Fallad, se ha mejorado considerablemente la eficiencia del algoritmo C4.5 en la selección de puntos. Otras alternativas se pueden encontrar en Quinlan (1996), Wu y Urpani, (1999) y Yen y Chu (2007).

3. Método GRASP para construir árboles

GRASP son las iniciales en inglés de Procedimientos de Búsqueda basados en funciones Ávidas, Aleatorias y Adaptativas (*Greedy Randomize Adaptive Search Procedures*); aunque inicialmente se dieron a conocer en el trabajo de Feo y Resende (1989) a finales de los ochenta, han tenido un desarrollo más reciente que los otros metaheurísticos. Una amplia descripción de esta metodología se puede encontrar en Feo y Resende (1995) y Pitsoulis y Resende (2002).

La idea que subyace en esta estrategia es la siguiente: no siempre la mejor elección aparente en un determinado paso lleva a la mejor solución final. Para ilustrar esta idea consideremos el siguiente ejemplo sencillo del problema de la mochila: Dados 3 objetos con pesos, valores y ratios (valor/peso) que se muestran en la figura 1 y una mochila de capacidad 12, se trata de seleccionar un subconjunto de ellos que cabiendo en la mochila maximicen el valor.

Objeto	Valor	Peso	Rat = val/pes
1	14	10	1,4
2	10	8	1.25
3	5	4	1.25

Fig. 1: Ejemplo de funcionamiento de GRASP

Es conocido que para este problema el ratio valor/peso suele ser una buena función guía para elegir que objeto se introduce en la mochila en cada paso. Ahora bien este criterio no siempre lleva a las mejores soluciones finales: Así en este ejemplo, el objeto a introducir según el ratio sería el objeto 1 (ratio =1,4); al introducirlo en la mochila no se podría introducir posteriormente otro objeto ya que se superaría la capacidad de la mochila y por tanto el valor de esta sería 14. Sin embargo, si en vez de elegir el objeto de mayor ratio es introducido el objeto de segundo mayor ratio, por ejemplo el 2 (ratio = 1,25) posteriormente se puede introducir el objeto 3, con lo que el valor de la mochila sería 15.

Por tanto lo que propone GRASP en cada paso es lo siguiente: se construye una lista con los mejores elementos para introducir en la solución según la función guía, lista de candidatos, y se elige aleatoriamente un elemento de la lista. Por tanto se hace uso de aleatoriedad pero de forma controlada ya que el elemento se escoge aletoriamente entre los mejores según la función guía. Este proceso de construcción se repite varias veces obteniéndose un conjunto de buenas soluciones diferentes. Entre ellas se elige la mejor, que en muchas ocasiones mejora la obtenida de forma determinística, i.e., eligiendo el mejor elemento en cada paso según la función guía.

En nuestro caso, el método recursivo para construir árboles se modifica de la siguiente manera

---

Procedimiento Fijar_caracteristicas_nodo_greedy_random (α,T, *nodo_x*)
Si se dan condiciones de parada entonces hacer:
Terminal = 0 y *clase* = clase mayoritaria en T
En caso contrario,
a) *Terminal* =1
b) Para j = 1,2,…,*m*: Determinar el valor h_j con el que se alcanza el mayor incremento de pureza Δ_j en las particiones determinadas por ese atributo
c) Determinar $\Delta max = \max\{\Delta_j / j = 1,2,\ldots,m\}$ y $\Delta min = \min\{\Delta_j / j = 1,2,...,m\}$
d) Construir $L = \{ j = 1,2,...,m / \Delta_j \geq \alpha \cdot \Delta max + (1-\alpha)\cdot \Delta min \}$
e) Escoger $j^* \in L$ aleatoriamente
f) Hacer *atributo* = j* y *punto_corte* = h_{j^*}
g) Hacer $T_1 = T \cap \{a_i / x_{ij^*} \leq h_{j^*}\}$ y $T_2 = T \cap \{a_i / x_{ij^*} > h_{j^*}\}$
h) Ejecutar:
Fijar_caracteristicas_nodo_greedy_random (α,T_1, *nodo_hijo_izda*)
Fijar_caracteristicas_nodo_greedy_random(α,T_2, n*odo_hijo_derecha*)

---

donde $\alpha \in [0,1]$. Obsérvese que si $\alpha = 0$ la elección es totalmente aleatoria, y si $\alpha = 1$ totalmente determinística. La elección del α adecuado suele ser una cuestión importante.

El funcionamiento de nuestro algoritmo GRASP se muestra a continuación:

Método GRASP_para_árboles de Clasificación(α, *nodo_best, n_hojas_best*)
Hacer n_hojas_best=∞
Repetir
Fijar_caracteristicas_nodo_greedy_random(α, A, nodo_initial)
Si n_hojas(nodo_initial) < n_hojas_best entonces hacer:
n_hojas_best:= n_hojas(nodo_initial)
nodo_best:=nodo_initial
hasta alcanzar un criterio de parada

---

donde *n_hojas(nodo)* es el número de hojas que contiene *nodo*.

En cada iteración se construye un árbol de clasificación. La solución final es la mejor solución que se obtiene tras todas las iteraciones. El criterio de parada puede consistir en un número máximo de iteraciones, un tiempo de computación determinado o un número de iteraciones en que no haya habido mejora.

4. Experiencias computacionales

Para chequear y comparar la eficacia del funcionamiento de nuestro algoritmo GRASP en la construcción del árbol de clasificación se han hecho una serie de pruebas con diferentes bases de datos. Para ello se han usado 17 conjuntos de datos que agrupamos en dos subconjuntos, las 8 primeras bases de datos son de mayor tamaño o más complejas y las 9 restantes son pequeñas o más sencillas. Estas bases de datos se pueden encontrar en el conocido repositorio de datos de la Universidad de California, UCI (ver Murphi y Aha, 1994). Se puede acceder a través de la siguiente página web: www.ics.uci.edu/~mlearn/MLRepository.html. Concretamente se trata de las siguientes bases de datos:

- *Spambase Database: 57 variables, 2 clases y 4601 casos.*
- *Covertype Data: Se trata de una base de datos forestal, con 54 variables explicativas, 8 clases y más de 580000 casos. Para las pruebas se ha hecho una selección aleatoria de 6000 casos de entre los casos de las dos primeras clases.*
- *Financial Database: 93 variables o ratios, 2 clases y 3000 casos o empresas.*
- *Waveform Database: 40 variables con valores continuos, 3 clases y 5000 instancias. Se han tomado solo las instancias correspondientes a las dos primeras clases, concretamente 3347 instancias.*
- *Optical Database: 64 variables, 5620 casos y 10 clases.*
- *Conect-4 Opening Database: 42 variables nominales, 3 clases y 67557 casos. Las 42 variables nominales se han transformado en 126 variables binarias. Para las pruebas se ha hecho una selección aleatoria de 6000 casos.*
- *German-Data-Numeric Database: 24 variables numéricas, 2 clases y 1000 casos*
- *Contraceptive Database: 9 variables, 1473 casos y 3 clases.*
- *Primary Tumor Domain Database: Las 22 clases con las que inicialmente contaba se agruparon en dos: la clase con mayor número de casos y el resto. Se eliminaron las dos variables en la que faltaban datos para el mayor número de casos y después se eliminaron también los casos en los que faltaban datos. Se transformaron las variables categóricas en variables binarias. De esta forma se obtuvieron 17 variables binarias, 2 clases y 336 casos.*
- *Vehicle Database: 18 variables y 2 clases. Para el conjunto de entrenamiento se han considerado las instancias correspondientes a las clases "bus" y "saab", esto es 435 instancias.*
- *Ionosphere Database: Con 34 variables, 2 clases y 351 casos.*
- *Heart Database: Después de transformar las variables categóricas en binarias, se han obtenido 22 variables. Hay 270 casos y 2 clases.*
- *Wisconsin Database Breast Cancer: 30 variables, 2 clases y 569 casos.*
- *Mushrooms Database: Con 22 variables nominales, 2 clases y más de 8100 casos. Las 22 variables nominales se han transformado en 121 variables binarias: 1 por cada variable binaria (i.e. con 2 posibles respuestas), y 1 por respuesta posible para el resto de variables. Se han eliminado los casos en los que faltan datos quedando finalmente 5644 casos.*
- *House-Votes-84 Database: En primer lugar se eliminó la variable en la que faltaban datos para el mayor número de casos y después se eliminaron también*

los casos para los que faltaban datos. De esta forma quedaron 15 variables binarias, 2 clases y 281 instancias.

- *SPECT Database: 22 variables, 2 clases y 267 casos.*
- *Dermatology Database: Se elimino una variable por faltarle datos con lo quedaron 33 variables, 6 clases y 366 casos.*

Todos los experimentos se han realizado en un ordenador PC Pentium IV 2.4 GHz. usando el compilador BORLAND DELPHI (ver. 5.0).
Para cada una de las 17 bases de datos consideradas se han ejecutado el algoritmo tradicional, (i.e. en cada paso el mejor atributo), y el método GRASP propuesto en este trabajo tomando como diferentes valores de α: 0.2, 0.4, 0.6, 0.8, 0.9, 0.95 y 0.99. El criterio de parada del GRASP es un máximo de 25 iteraciones. Para cada base de datos se ha realizado un 1x10 cross-validation para medir y contrastar los métodos usados. Esto consiste en dividir cada base de datos en 10 conjuntos del mismo tamaño. A continuación en cada ejecución se elimina sistemáticamente cada uno de estos conjuntos, considerando como conjunto de entrenamiento al conjunto formado por la unión de los restantes nueve conjuntos y como conjunto test el conjunto que se había eliminado previamente. De esta forma se realizan 10 ejecuciones por cada base de datos y, por tanto, 170 ejecuciones de cada método. Para cada uno de estos métodos en cada ejecución se registran los siguientes valores:

- *Precisión: proporción de éxitos en el conjunto test*
- *Precisión_Relativa: ratio Precisión/(Precisión del algoritmo tradicional)*
- *Complejidad : nº de hojas terminales*
- *Complejidad_Relativa: ratio Complejidad/(Complejidad del algoritmo tradicional)*

Obviamente para el método tradicional no es necesario calcular Precisión_Relativa y Complejidad_Relativa. Además para el método GRASP también se calculan los siguientes estadísticos:

- *N_mejor : Nº de pruebas en las que el árbol obtenido tiene menos hojas que el obtenido por el método tradicional*
- *N_empates : Nº de pruebas en las que el árbol obtenido tiene el mismo número de hojas que el obtenido por el método tradicional.*

Como se ha comentado antes la construcción de cada árbol debe asegurar el 100% de casos bien clasificados en A. En el método tradicional el criterio para elegir el mejor atributo y punto de corte es *Information Gain*. Esta es también la función guía usada por nuestro método GRASP. En todos los casos se usa el método Fayyad para formar el conjunto de valores candidatos a punto de corte de cada atributo. El criterio de parada de nuestro GRASP es un máximo de 25 iteraciones.

A continuación se muestran los resultados, en 3 tablas, correspondientes a las bases de datos más sencillas, las más complejas y a todas en conjunto. Para *Precisión*, *Precisión_Relativa*, *Complejidad* y *Complejidad_Relativa* se muestran las medias y las desviaciones típicas. Para *Precisión* y *Complejidad* se muestran además el estadístico t correspondiente al test (de la t de student) de diferencias de medias entre el método

GRASP y el tradicional. Para *Precisión_Relativa* y *Complejidad_Relativa* se muestra el estadístico t correspondiente al test de comparar la media de ese valor frente a 1. Se indican en negrita los casos con un valor t significativamente alto en los valores de *Precisión*, *Precisión_Relativa*, y significativamente bajo en los valores de *Complejidad* y *Complejidad_Relativa* (considerando un nivel de significación del 10%).

Tabla 1.- Bases sencillas (9)

		GRASP							**Método**
	α	0,2	0,4	0,6	0,8	0,9	0,95	0,99	**tradicional**
Precisión	mean	0,882	0,887	0,892	0,882	0,887	0,887	0,885	0,885
	std	0,106	0,102	0,095	0,103	0,101	0,097	0,093	0,099
	t	*-0,210*	*0,141*	*0,504*	*-0,167*	*0,142*	*0,163*	*0,021*	
Precisión_Relaiva	mean	0,999	1,005	1,011	0,998	1,003	1,004	1,002	
	std	0,086	0,074	0,065	0,049	0,049	0,044	0,038	
	t	*-0,128*	*0,615*	***1,612***	*-0,453*	*0,649*	*0,869*	*0,497*	
Complejidad	mean	38,844	30,944	28,233	27,789	28,700	29,178	29,844	30,478
	std	20,467	19,406	19,185	19,573	20,279	20,286	20,355	20,583
	t	*2,734*	*0,156*	*-0,757*	*-0,898*	*-0,584*	*-0,427*	*-0,208*	
Complejidad_Relativa	mean	1,380	1,040	0,922	0,897	0,933	0,952	0,978	
	std	0,303	0,155	0,085	0,076	0,067	0,058	0,039	
	t	*11,905*	*2,479*	***-8,742***	***-12,876***	***-9,561***	***-7,801***	***-5,386***	
N_mejor		7	36	72	81	63	56	36	
N_empates		3	10	10	5	26	33	54	

Tabla 2.- Bases complejas (8)

		GRASP							**Método**
	α	0,2	0,4	0,6	0,8	0,9	0,95	0,99	**Tradicional**
Precisión	mean	0,760	0,773	0,778	0,784	0,784	0,784	0,785	0,784
	std	0,133	0,144	0,142	0,144	0,141	0,141	0,143	0,142
	t	*-1,096*	*-0,507*	*-0,283*	*0,003*	*-0,011*	*-0,020*	*0,032*	
Precisión_Relativa	mean	0,973	0,986	0,993	1,000	1,001	1,001	1,001	
	std	0,062	0,048	0,043	0,033	0,031	0,032	0,020	
	t	*-3,850*	*-2,613*	*-1,496*	*0,034*	*0,243*	*0,200*	*0,437*	
Complejidad	mean	738,338	538,225	444,675	408,300	404,063	405,075	411,113	418,738
	std	501,019	366,307	316,619	304,871	301,224	302,582	310,228	316,423
	t	*4,824*	*2,208*	*0,518*	*-0,212*	*-0,300*	*-0,279*	*-0,154*	
Complejidad_Relativa	mean	1,919	1,364	1,090	0,978	0,967	0,969	0,982	

		GRASP							Método
	α	0,2	0,4	0,6	0,8	0,9	0,95	0,99	**Tradicional**
	std	0,621	0,294	0,119	0,031	0,027	0,022	0,022	
	t	*13,229*	*11,089*	*6,780*	***-6,355***	***-10,964***	***-12,263***	***-7,640***	
N_mejor		0	0	15	61	73	77	67	
N_empates		0	0	1	4	3	1	10	

-

Tabla 3.- Todas las bases (17)

		GRASP							Método
	α	0,2	0,4	0,6	0,8	0,9	0,95	0,99	**Tradicional**
	mean	0,825	0,833	0,838	0,836	0,838	0,838	0,838	0,837
Precisión	std	0,134	0,136	0,132	0,133	0,132	0,130	0,129	0,131
	t	*-0,900*	*-0,295*	*0,062*	*-0,091*	*0,071*	*0,074*	*0,035*	
	mean	0,987	0,996	1,002	0,999	1,002	1,002	1,002	
Precisión_Relativa	std	0,076	0,064	0,056	0,042	0,041	0,039	0,031	
	t	*-2,242*	*-0,840*	*0,575*	*-0,364*	*0,685*	*0,832*	*0,641*	
	mean	368,018	269,665	224,206	206,853	205,341	206,071	209,265	213,188
Complejidad_Relativa	std	490,084	356,948	300,860	282,728	279,181	280,046	285,721	291,212
	t	*3,541*	*1,598*	*0,343*	*-0,204*	*-0,254*	*-0,230*	*-0,125*	
	mean	1,634	1,193	1,001	0,935	0,949	0,960	0,979	
Complejidad_Relativa	std	0,549	0,281	0,132	0,072	0,055	0,046	0,032	
	t	*15,051*	*8,931*	*0,105*	***-11,826***	***-12,166***	***-11,359***	***-8,298***	
N_mejor		7	36	87	142	136	133	103	
N_empates		3	10	11	9	29	34	64	

De los resultados de la tabla 1 se pueden extraer las siguientes observaciones:

- *Para valores altos de α ($\alpha \geq 0.6$) nuestro GRASP obtiene árboles con un número de nodos terminales significativamente inferior a los obtenidos por el método tradicional. Esto se concluye de los valores de Complejidad_Relativa y su correspondiente valor t. De la misma forma valores bajos de α, i.e. procedimientos quizás excesivamente aleatorios, dan lugar a árboles más complejos que los obtenidos tradicionalmente.*
- *En general para valores intermedios y altos de α ($\alpha \geq 0.4$) nuestro GRASP mejora el porcentaje de aciertos en los conjuntos test obtenido por el método tradicional. Sin embargo esta mejora solo es significativa para $\alpha = 0.6$, si se observa en los valores de Precisión_Relativa y su correspondiente t.*

De los resultados de la tabla 2 se puede extraer que:

- *Nuestro GRASP obtiene mejores resultados con valores altos de α ($\alpha \geq 0.8$) tanto en complejidad como en aciertos.*
- *Para estos valores de α el porcentaje de aciertos es muy ligeramente superior al método tradicional, aunque esta mejoría no es significativa. Sin embargo, sí que se obtienen árboles de complejidad significativamente menor que los métodos tradicionales.*

Finalmente de la tabla 3 que resume las dos anteriores se observa que:

- *Por lo general a medida que se aumenta el valor de α disminuye la complejidad y aumenta el porcentaje de aciertos. Los mejores resultados se obtienen para α = 0.9 y 0.95 y en menor medida para α = 0.99.*
- *Con estos valores de α se mejoran los resultados, tanto en complejidad como en Precisión, del método tradicional. No obstante, esta mejora solo es significativa para la complejidad.*
- *De la evolución de N_mejor y N_empates según los valores de α a partir de 0.8, es interesante observar que a medida que éste crece α nuestro GRASP frente al método tradicional, empata más veces, gana menos, pero también pierde menos. En otras palabras gana cierta robustez.*

5. Conclusiones

En este trabajo se propone un nuevo método para la construcción de árboles de clasificación con la menor complejidad posible; concretamente árboles binarios. Este método se basa en la estrategia GRASP y básicamente consiste en modificar la forma de elegir el atributo que determina la partición en cada nodo. Mientras tradicionalmente se elige el mejor atributo, con nuestra metodología se construye una lista con los mejores atributos y se elige uno al azar. Es decir se trata de usar de forma equilibrada la aleatoriedad y los criterios o funciones tradicionales para elegir el atributo. Este método puede ser usado con diferentes funciones para medir la pureza o criterios para elegir el atributo (*Indice Gain*, *Gain Ratio*, etc), con diferentes criterios para finalizar, con diferentes formas de determinar el conjunto de puntos de corte, con diferentes métodos de poda posterior. Los resultados obtenidos muestran que con tamaños adecuados de la lista se consiguen árboles significativamente menos complejos que con métodos tradicionales, facilitando así su interpretación y permitiendo el equilibrio entre optimización y generalización en los conjuntos test. Además también el grado de éxito en los conjuntos test es mejor, aunque no de forma significativa. En general los mejores resultados se consiguen con listas pequeñas (valores altos de α, i.e., menor aleatoriedad), aunque para bases de datos sencillas puede ser recomendable una mayor aleatoriedad.

6. Agradecimientos

Los autores quieren agradecer la ayuda financiera recibida del Ministerio de Ciencia y Tecnología y fondos FEDER (Plan Nacional de I+D+I - Proyecto ECO2008-06159/ECON), de la Conserjería de Educación del Junta de Castilla y León (Proyecto BU008A10-2) y de "CajaBurgos" y la Universidad de Burgos (Proyectos de convocatoria propia 2009).

7. Referencias

1. M.S. Sanchez (1997). Redes neuronales en clasificación, Tesis Doctoral, Universidad de Valladolid.
2. L. Breiman, J.H. Friedman, R. Olshen and C.J. Stone, *Classification and regression trees* (Belmont Wadsworth International Group, 1984).
3. C.J. Huberty (1994). Applied Discriminant Analysis. Wiley Interscience
4. P. Komarek (2004). Logistic regression for data mining and high-dimensional classification. Phd Thesis, Carnegie Mellon University, Pittsburgh, Pennylvania, USA.
5. S.M. Weiss and C. Kulikoswki, Computer systems that learn: Classification and prediction methods from statistics, neural nets, machine learning and expert systems (Morgan Kauffmann Publisher, 1991).
6. R.O. Duda, P.E. Hart and D.G. Stork, *Pattern classification* (2nd Ed. John Wiley, 2000).
7. T.A. Feo and M.G.C. Resende, A Probabilistic heuristic for a computationally difficult Set Covering Problem. *Oper Res Lett*, **8**(1989), 67-71.
8. P. Winker and M. Gilli (2004). Applications of optimization heuristics to estimation and modelling problems, *Computational Statistics & Data Analysis*, 47, 2, 211-223.
9. N. Belacel, H.B. Raval and A.P. Punnen (2007). Learning multicriteria fuzzy classification method PPOAFTN from data, *Computers & Operations Research*, 34, 7, 1885-1898.
10. P.C. Pendharkar (2006). A data mining-constraint satisfaction optimization problem for cost effective classification, *Computers & Operations Research, 33, 11, 3124-3235.*
11. F.C.García, M. García, B. Melián, J.A. Moreno y M. Moreno (2006). Solving feature selection problema by a parallel scatter search. *European Journal of Operational Research*, 169, 2, 477-489.
12. J. Pacheco, S. Casado, L. Núñez and O. Gómez (2006). Analysis of New Variable Selection Methods for Discriminant Analysis, *Computational Statistics and Data Analysis*, 51, 3, 1463-1478.
13. J.Y. Yang and S. Olafsson (2006). Optimization-based feature selection with adaptive instance sampling. *Computers & Operations Research, 33, 11, 3088-3106.*
14. J .Pacheco, S. Casado and L. Núñez (2009). A variable selection method based in Tabu Search for Logistic Regression Models, *European Journal of Operational Research*, 199, 1, 506-511.
15. N. Belacel, P. Hansen and N. Mladenovic (2003). Fuzzy J-Means: a new heuristic for fuzzy clustering, *Pattern Recognition*, 35, 2193-2200.
16. J. Pacheco and O. Valencia (2003). Design of hybrids for the minimum sum-of-squares clustering problem, *Computational Statistics and Data Analysis,* 43, 2, 235-248.
17. D.L. Woodruff and T. Reiners (2004). Experiments with, and on, algorithms for maximum likelihood clustering, *Computational Statistics & Data Analysis, 47, 2, 237-253.*
18. J. Cadima, J.O. Cerdeira and M. Minhoto (2004). Computational aspects of algorithms for variable selection in the context of principal components, *Computational Statistics & Data Análisis*, 47, 2, 225-236.
19. C. Gatu and E.J. Kontoghiorghes (2003). Parallel algorithms for computing all possible subset regression models using the QR Descomposition, *Parallel Computing,* 29, 505-521.

20. C. Gatu and E.J. Kontoghiorghes (2005). Efficient strategies for deriving the subset {VAR}models. *Computational Management Science,* 2, 4, 253-278.
21. M. Hofmann, C. Gatu and E.J. Kontoghiorghes (2007). Efficient algorithms for computing the best subset regression models for large-scale problems, *Computational Statistics and Data Analysis*, 52, 1, 16-29.
22. R. Baragona, F. Battaglia and D. Cucina (2004). Fitting piecewise linear threshold autoregressive models by means of genetic algorithms, *Computational Statistics and Data Analysis,* 47, 2, 277-295.
23. G. Kapetanios (2007). Variable selection in regression models using nonstandard optimisation of information criteria, *Computational Statistics and Data Analysis*, 52, 1, 4-15.
24. J.R. Quinlan, Induction of decision trees. *Machine Learning* **1**(1986), 81–106.
25. J.R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufman, 1993*)*.
26. U.M. Fayyad and K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: *Proceedings of the 13th International Joint Conference on Artificial Intelligence* (1993*)*, 1022–1027.
27. U.M. Fayyad and K.B. Irani, On the handling of continuous-valued attributes in decision tree generation. *Machine Learning* **8** (1992), 87–102.
28. J.R. Quinlan, Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research* **4** (1996), 77–90.
29. X. Wu and D. Urpani, Induction by attribute elimination, *IEEE Transactions on Knowledge and Data Engineering,* **11- 5** (1999), 805–812.
30. E. Yen and I.W.M. Chu, Relaxing instance boundaries for the search of splitting points of numerical attributes in classification trees, *Information Sciences*, **177** (2007); 1276-1289.
31. T.A. Feo and M.G.C. Resende, Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, **2** (1995), 1-27.
32. L.S. Pitsoulis and M.G.C. Resende, Greedy Randomized Adaptive Search Procedures in *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende Eds. (Oxford University Press, 2002), 168-182.
33. P.M. Murphy and D.W. Aha (1994). UCI repository of Machine Learning (University of California, Department of Information and Computer Science, 1994) http://www.ics.uci.edu/~mlearn/MLRepository.html.