INSTITUT DE STATISTIQUE

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

DISCUSSION PAPER

0124

HYPER-RECTANGULAR SPACE PARTITIONING CLASSIFICATION TREES : A FEW INSIGHTS

I. DE MACQ and L. SIMAR

http://www.stat.ucl.ac.be

HYPER-RECTANGULAR SPACE PARTITIONING TREES: A PRACTICAL APPROACH

Isabelle De Macq and Léopold Simar[1]

November 2002

ABSTRACT. The process of computation of classification trees can be characterized as involving three basic choices: the type of splits considered in the growing process, the criterion to be optimized at each step of the process, and the way to get right-sized trees. Most implementations are ordinary binary trees, i.e. trees whose successive cuts are made by hyper-planes perpendicular to the axes, while most of the litterature concerns the various possible criteria and pruning methods. L. Devroye, L. Györfy and G. Lugosi (1996) define and consider the remarkable theoretical properties of a binary tree classifier whose prominent feature is the particular type of splits used in its construction: at a given node, partitioning is made by hyper-rectangles rather than hyper-planes. We propose an approximation of the solution for the complex optimization problem involved to allow insights on the practical advantages of those trees. Then we compare the performance of our algorithm with some leading algorithms for ordinary binary trees, namely CART and C4.5 as implemented in the Splus "tree" procedure and in SAS 's Enterprise Miner respectively. For this purpose, data sets which traditionally enhance the weaknesses of classification trees are used, as well as data sets commonly used for comparisons.

KEYWORDS. Binary classification trees, hyper-rectangles, splitting method.

[1]Institut de Statistique, Université Catholique de Louvain, B 1348 Louvain-la-Neuve, Belgium. Research support from "Projet d'Actions de Recherche Concertées" (No. 98/03–217) and from the "Interuniversity Attraction Pole", Phase V (No. P5/24) from the Belgian Government are also acknowledged.

1 Introduction

Classification trees are one of the main techniques used in data mining for "Supervised Learning", a problem usually referred to as discrimination in the statistical litterature: they allow to establish classification rules on the basis of a training sample consisting of correctly classified data. It is a very flexible non parametric process for explaining and/or predicting a categorical response (membership to classes) on the basis of measurements on predictors. The space of the predictor variables is partitioned into regions in which decision is mostly taken by majority vote over the cases of the training sample, with possible adaptations for non equal misclassification costs. Like most tools of data mining, tree classifiers were developed in the perspective to be powerful on very large databases, and for problems where the computational time is a main concern. Most implementations are therefore ordinary binary classification trees, for which the partitioning method is applicable in a computationally very simple manner: the successive cuts in the space of the predictor variables are made by hyper-planes perpendicular to the axes, so that the different attributes can be considered separately at each step of the construction. Besides efficiency, this hierarchical method has proven to have many additional advantages such as interpretability of the rules or invariance with respect to monotone transformations of the data. It is applied to a wide variety of classification problems (medical diagnosis, classification in botany, credit scoring, ...).

The main complain with classification trees is their high variance: a small change in the data often leads to a totally different sequence of splits and hence a completely different structure for the resulting classifier. After a first focus in the literature on pruning methods intended to get more reliable trees, the emphasis was put on the application of more general techniques for stabilizing classifiers like bagging or boosting techniques, and variants of it. This alleviates somewhat the problem but the inherent instability is not removed, and the interpretability is no longer satisfactory. Besides, despite the use of these various pruning and averaging techniques ordinary tree classifiers are not appropriate, or give poor results when there are many regions per class, i.e. on highly fragmented distributions.

In this work, we consider a binary tree classifier introduced by L. Devroye, L. Györfy and G. Lugosi (1996). The prominent feature of this tree classifier is the particular type of splits used in its construction: at a given node, partitioning is made by hyper-rectangles rather than hyper-planes. The splitting criterion (empirical error) is thus optimized over a larger set, including halfspaces as well as bounded hyper-rectangles. The most remarkable property of those “Hyper-rectangular Space Partitioning” trees (HSP trees) is that they provide a consistent sequence of classifiers in a fully automatic way. Compared with ordinary binary trees, they share the property of invariance with respect to monotone transformations of the axes, but their solutions offer more stability.

The high computational order of HSP trees raises the practical problem of finding an efficient algorithm or at least some approximate solution for the method to be applicable on large datasets. The computational order of any naive procedure would be quadratic in the sample size and exponential in the number of predictor variables. This is not usable for practical purposes, but it allows to point out some interesting features on small data sets. In Section 2, HSP trees will be defined more precisely and some glimpse of their strong points will be given. We propose in Section 3 a simple method to handle HSP trees in practice by reducing the set of considered splits and using local adjustments. Though the simplicity and very strong properties of the original method are not preserved by the approximation the practical benefits are worth considering. A few illustrative simulations are presented in Section 4, where an implementation of a HSP tree is compared with other methods of the same category, notably methods with the invariance property. In particular two leading algorithms for ordinary binary trees are used: CART and C4.5 as implemented in the Splus "tree" procedure and in SAS 's Enterprise Miner respectively. For a few standard test designs, namely "XOR" data and "checker-board" data, the behaviour of the various classifiers is compared for increasing levels of the Bayes risk, and the effect of added noisy variables is considered. Finally the method is applied to some broadly used data sets from the public repository UCI (Murphy and Aha, 1996).

2 HSP trees

2.1 The method

Though the method is applicable in a more general context, we consider only discrimination problems between two populations Π_0 and Π_1. Let us denote class membership by the binary variable Y:

$$Y = \begin{cases} 1 & \text{if the case comes from } \Pi_1 \\ 0 & \text{otherwise} \end{cases}$$

and let $X \in I\!R^d$ represent d continuous explanatory variables. The learning sample is denoted by $(X_i, Y_i)_{i=1,\ldots,n}$. Binary classification trees are characterized by successive binary divisions of "nodes" or sets in $\mathbb{R}^d$: if a node u represents the set A in $\mathbb{R}^d$ and its children u', u'' represent A' and A'', then we have $A = A' \cup A''$ and $A' \cap A'' = \emptyset$. The root of the tree, or initial node, represents the whole space $\mathbb{R}^d$ and the leaves, or terminal nodes, form a partition $\mathcal{P}$ of $\mathbb{R}^d$. The decision function $g_{\mathcal{P}}$ on a set $A \in \mathcal{P}$ is by majority vote: for a given $x \in A$,

$$g_{\mathcal{P}}(x) = \begin{cases} 1 & \text{if } \sum_{i:X_i \in A} Y_i > \sum_{i:X_i \in A} (1 - Y_i) \\ 0 & \text{otherwise} \end{cases}$$

Consider the empirical error or percentage of misclassified cases in the learning sample for the set $A \in \mathcal{P}$:

$$\hat{L}_n(A) = \frac{1}{n} \sum_{i=1}^{n} I_{\{X_i \in A\, , g_{\mathcal{P}}(X_i) \neq Y_i\}}.$$

An estimate of the quality of the rule $g_{\mathcal{P}}$ is given by the empirical error of the partition $\mathcal{P}$ that we can define with some abuse of notation as

$$\hat{L}_n(\mathcal{P}) = \sum_{A \in \mathcal{P}} \hat{L}_n(A)\ .$$

The growing process of HSP trees is essentially the same as the one of ordinary binary trees. The criterion used in the minimization process is the empirical error rate, but the main difference resides in the type of the splits: a single split involves the selection of a hyper-rectangle in $\mathbb{R}^d$ rather than a hyper-plane. More precisely let $\mathcal{P}_t$ be the constructed partition at stage t of the process, i.e. after t cuts. Given the partition $\mathcal{P}_t$, a legal hyper-rectangle T is one for which $T \cap A = \emptyset$ or $T \subset A$ for all sets $A \in \mathcal{P}_t$: hyper-rectangles are not allowed to overlap. According to Devroye et al.(1996), we obtain partition $\mathcal{P}_{t+1}$ by adding in some set $R \in \mathcal{P}_t$ a legal hyper-rectangle T minimizing $\hat{L}_n(\mathcal{P}_{t+1})$. Since the decision functions $g_{\mathcal{P}_{t+1}}$ and $g_{\mathcal{P}_t}$ will agree except on the set R, this is equivalent to finding $T \in R$ which minimizes $\hat{L}_n(T) + \hat{L}_n(R - T) - \hat{L}_n(R)$. Starting with the trivial partition $\mathcal{P}_0 = \{\mathbb{R}^d\}$, the splitting rule just described is repeated several times, say k, leading thus to $k+1$ regions. The sequence of partitions is denoted by $\mathcal{P}_0$, $\mathcal{P}_1$,...,$\mathcal{P}_k$. We discuss about the final selection of a classifier from this sequence in the following section.

From the practical point of view the implementation of HSP trees involves some challenging problems amongst which the conception of an algorithm with acceptable computational complexity. Suppose we are given m training cases to be split at a given node. Their ordered values on a coordinate axis define the set of boundaries to be considered on this axis. For any hyper-rectangle of $\mathbb{R}^d$ indeed we can choose a hyper-rectangle determined by the defined boundaries so that it contains the same sample points and therefore results in the same error rate. A hyper-rectangle is determined by two boundaries on each coordinate. Consequently, by taking all possible combinations of boundaries we get at most $(\frac{m(m+1)}{2})^d$ different sets of sample points of any size that can be included in a hyper-rectangle. Such an exhaustive search for an optimal hyper-rectangle at each step of the construction is performed by the naive algorithm whose computational order is $o(n^{2d})$. By avoiding redundancy in the isolated sets of points this reduces to $o(n^{2d'})$ in the (not very likely) case that the data lies in some subspace of dimension $d' < d$.

2.2 Advantages of HSP trees

Simply stated, classification tree analysis aims to obtain a partition allowing the most accurate prediction possible. The Bayes decision function, defined as

$$g^*(x) = \begin{cases} 1 & \text{if } P(Y = 1 \mid X = x) > 0.5 \\ 0 & \text{else} \end{cases}$$

provides a bound on the performance of any classifier in the sense that its probability of error $L^* = L(g^*) = P(g^*(X) \neq Y)$ is the lowest amongst all classifiers g. In a remarkable result HSP trees are shown to be consistent classifiers provided that the number of splits k is $o(\sqrt[3]{n/\log n})$ and assuming X has nonatomic marginals: under these assuptions $L_n(\mathcal{P}_k)$ converges to L^* with probability one (Devroye et al., 1996, theorem 20.9). No sophisticated pruning technique is needed thus to achieve outstanding properties as $\mathcal{P}_k$ directly comes from the sequence of trees obtained during the growing phase. Practically however a choice has to be made between different methods to determine a good value for k. Let us continue the growing process until all final regions are either totally homogeneous in one of both groups or below some fixed threshold of heterogeneity. The consistency property allows to derive a very simple practical rule for selecting a final tree in the sequence: by minimizing the error rate commited on a validation set we are guaranteed to obtain a consistent process. If we can not use some part of the learning sample as a validation sample a value for k can be chosen by cross-validation methods. Determining a right size is then made easier due to the better stability of HSP trees as will be illustrated in the few following experiments. The naive algorithm was implemented in the C language and tested on several data sets.

The "XOR" problem (see Figure 1) traditionally illustrates a major drawback of the cutting method of ordinary binary trees: at each step the cut is made by some hyperplane perpendicular to one coordinate, so that optimization is performed over halfspaces only. Small sets can not be isolated by a single split and the consequences of a particular choice on later steps in the construction are not considered. The extremely simple XOR structure can not be catched properly by a classifier which acts separately on the different explanatory variables even when the Bayes risk is null. Indeed no single split of the form $x_i \leq \beta$ can lead to any substantial improvement of the purity of the partition, whatever the criterion chosen to measure it.

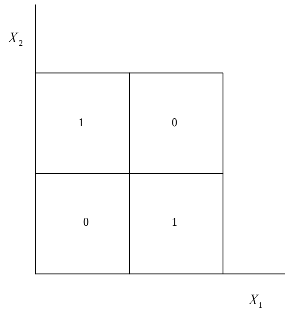

Figure 1: XOR distribution

To handle this particular structure, no complex partitioning method is needed however. Median trees and histogram rules for instance are expected to give excellent results. Median trees (MT) are built as follows: the root is cut at the median point along the first coordinate (step 0); at step k of the process, each of the 2^k existing regions is cut at its median point along the coordinate determined by a forced rotation over the coordinates (see Figure 2, left side for an example in dimension $d = 2$; the first three steps result in the illustrated partition). The accuracy of these trees depend on the order in which the variables are recorded: in a XOR design with added noisy variables $X_3, \ldots, X_d$, they trivially present a high stability. We also considered a version in which the discriminatory variables are read in a random order (RMT). Flexible histogram rules (HR) are d-dimensional grids with a fixed number h of data dependant cut points on each coordinate (Figure 2, right side for an example with $h = 2$). Here accuracy decreases as h increases due to the very particular structure of the data, and because we only considered small to moderate sample sizes (the number of cells of the generated partition grows exponentially with the number of predictors). In the two methods described above, the class labels are not used in the construction of the partition.

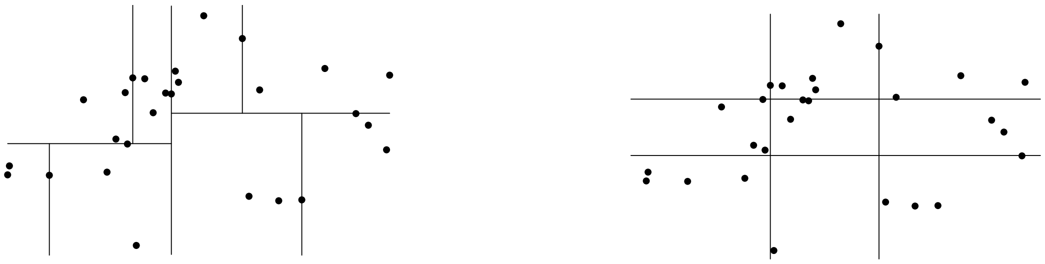

Figure 2: Construction of median trees (left) and histogram rules (right)

On the other hand, more complicated algorithms like CART or C4.5 are less likely to identify the right structure and their solutions are expected to be quasi random at least in the higher levels of the tree. These four methods have been compared with HSP trees on samples of various sizes drawn from the XOR distribution with an increasing number b of noisy variables ($b = 0, \ldots, 3$, 100 simulations in each case). We used the variant of the CART algorithm implemented in the Splus tree procedure (version 6 for Unix) and the C4.5 algorithm as implemented in SAS Enterprise Miner. Nevertheless we do not present the results of the latter method as no non trivial solution was found with this algorithm i.e. the root was not cut. The relative performance of the classifiers was measured by error rates computed on independent test samples. On Figure 3, the mean accuracy of each classifier is plotted against the number of noisy variables for the simulations performed on small sample sizes ($Ntr = 100$). More detailed results are presented in Table 1.

	HSP		HR		MT		RMT		S+	
xor.b0	0.036	(0.002)	0.073	(0.004)	0.080	(0.004)	0.080	(0.004)	0.065	(0.005)
xor.b1	0.041	(0.004)	0.078	(0.005)	0.090	(0.004)	0.108	(0.005)	0.128	(0.011)
xor.b2	0.055	(0.005)	0.085	(0.007)	0.084	(0.004)	0.153	(0.007)	0.189	(0.015)
xor.b3	0.055	(0.005)	0.116	(0.007)	0.090	(0.004)	0.201	(0.008)	0.254	(0.015)

Table 1: Mean error rates and Monte Carlo standard deviations

None of the compared classifiers enjoys the natural stability of median trees for this design, but for general classification purposes median trees as well as histogram rules are mostly not helpful. Ordinary binary trees are totally inefficient especially when some noise is added; they have a very high variability (see Figure 4). On larger sample sizes, they

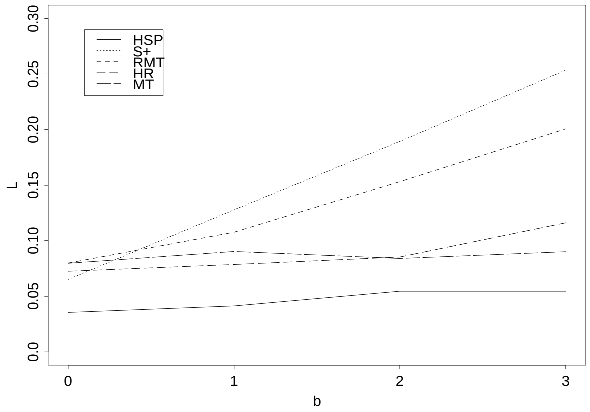

Figure 3: Mean accuracy under increasing noise levels

achieve a better relative accuracy, but they are very complex and present a random structure. With HSP trees, accuracy and readability substantially improve on small and moderate sample sizes. However on very small sample sizes, though accuracy is good and stable with respect to noise for most of the cases, we observe many extreme outliers (see Figure 4). Appropriate cross-validation techniques would have reduced this problem; we did not use it for the experiments performed on this design.

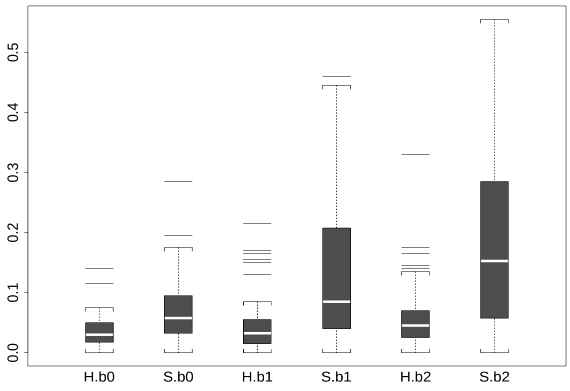

Figure 4: Variability of S+ and HSP trees, XOR design with b noisy variables

Though tree methods are best suited for high-dimensional problems, their application to simple two-dimensional problems allows to visualize on simple graphical representations the decision rules identified by the different classifiers. The next example is another pathological dataset: the 4×4 checker-board data with noise ($L^* = 0.15$). Figure 5 illustrates the behaviour of ordinary binary trees and HSP trees on a random sample of size 800 drawn from this highly fragmented distribution. Ten-fold cross-validation was performed to determine a good tree size: 10 sequences of finer and finer trees were build and evaluated by independant test sets held out for this purpose. The curves represent for each type of classifier the estimated error rate as a function of the tree size. A common feature of this plot is a fairly flat minimum, and trees in this region are candidates for further consideration. This justifies a widespread practical rule which consists in selecting the simplest tree in the sequence such that $\hat{L}_n^*(T)$ is inferior to $\min \hat{L}_n^*(T) + a * \text{std err of } \hat{L}_n^*(T)$, where $a = 0$ ("0-SE rule") and $a = 1$ ("1-SE rule") are common choices. However the minimum error rate clearly appears as a peak value for the HSP trees, indicating a very stable structure amongst the tree replicates. The partitions on the right of Figure 5 are the best corresponding classification rules.

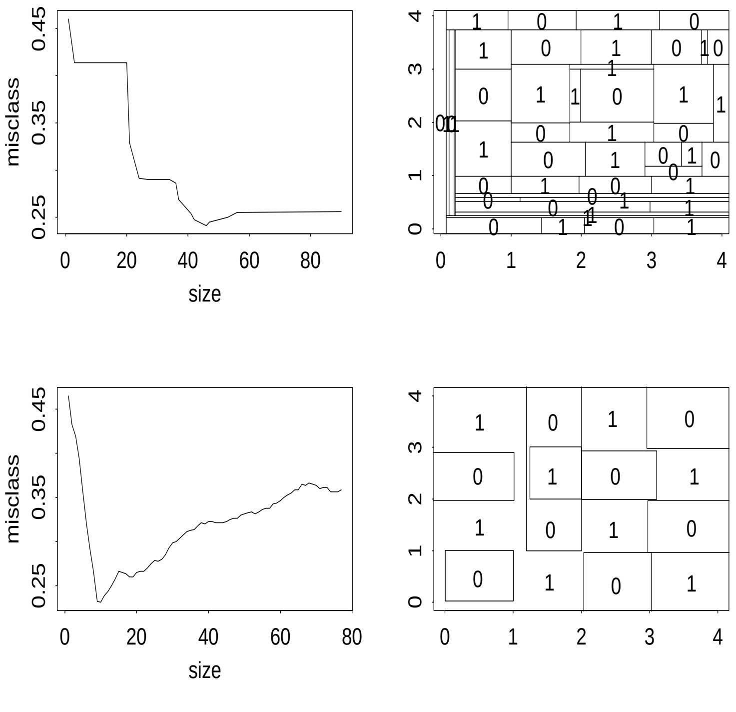

Figure 5: Ordinary and HSP tree classifiers (top and bottom panels respectively) for "4×4 checker-board: average error rate of the cross-validation replicates for increasing tree sizes and best corresponding classifier for a random sample of size 800

On larger sample sizes, ordinary trees globally and roughly come to an accurate classification rule, but the number of steps needed to get a right partition is extremely high and the partition differs from one sample to another: the regions corresponding to a given class will appear as the union of numerous leaves varying from sample to sample. So the tree structure is very unstable and the actual data structure is hidden, making interpretation difficult without graphical visualization, i.e. as soon as the dimension is greater than 2. HSP trees on the other hand possess the ability to recover perfectly the underlying structure. The results of Monte Carlo simulations for a Bayes risk equal to 0 and for increasing sample sizes are presented on Figure 6.

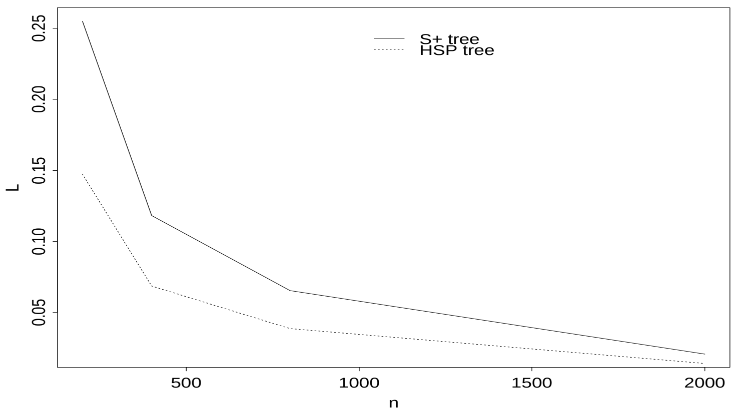

Figure 6: Ordinary and HSP tree classifiers for "4×4 checker-board: mean error rate of Monte Carlo simulations for increasing learning sample sizes

3 A practical approach for HSP trees

Hyper-rectangular space partitioning trees enjoy nice properties and are expected to outperform ordinary binary trees. However they are practically too expensive: any exhaustive search for a hyper-rectangle in a d-dimensional space involves a computational order $O(n^{2d})$. The method described in this section allows to control the feasibility of a HSP tree in terms of computational time needed by means of a user-defined parameter Q.

3.1 Basis step in the construction

Let us denote by q some quantile order with $q = 1/Q$ for some integer $Q > 1$. At any level of the growing process, all the sets -or nodes- of the partition can be written as $H_0 \setminus A$, where H_0 is some hyper-rectangle in $\mathbb{R}^d$, and A is either the empty set or the disjoint union of some hyper-rectangles $H_1, \ldots, H_s$ in H_0. Suppose we are given m training cases in some node $H_0 \setminus \cup_{i=1}^{s} H_i$ to be split. To provide an optimal hyper-rectangle amongst all hyper-rectangles of H_0 that do not overlap $H_1, \ldots, H_s$ the set of boundaries on each coordinate axis is taken as the set of all ordered values of the training cases on that coordinate axis. In order to reduce the computational order, especially high at the top levels of the tree, we will reduce this set to the set of quantiles of order q of the region to be split.

Consider some fixed coordinate x_k and denote as $\{v_1, v_2, \ldots, v_m\}$ the set of the sorted values of the m considered points. We divide it into $1/q = Q$ equal-sized blocks (up to one element) and we determine the sets of lower and upper bounds by taking the first and last elements of each block respectively. More precisely, suppose $Q < m$ and let p and r denote the integer division of m by Q and its integer remaining part: $m \backslash Q = p$ and $r = m \bmod Q$. The m ordered values are divided in r ordered subsets with $p+1$ elements and $Q-r$ with p elements. Then the set of lower bounds on x_k is $\mathcal{L}_k = \{\ell_i \ , \ i = 1, \ldots, Q\}$ where the i^{th} lower bound ℓ_i is defined as

$$\ell_i = v_j \text{ with } j = 1 + \min(i-1, r) * (p+1) + \max(0, i-1-r) * p$$

and the set of upper bounds on x_k is $\mathcal{U}_k = \{u_i \ , \ i = 1, \ldots, Q\}$ where the i^{th} upper bound is defined as

$$u_i = v_j \text{ with } j = \min(i, r) * (p+1) + \max(0, i-r) * p$$

Similar sets are defined for each coordinate. If $Q \geq m$, optimization is over all possible hyper-rectangles.

On the node $H_0 \setminus \cup_{i=1}^{s} H_i$, optimization of the empirical error is performed over the set

$$\mathcal{H}(H_0 \setminus \cup_{i=1}^{s} H_i) = \{(x_1, \ldots, x_{2d}) \in \prod_{i=1}^{d} \mathcal{L}_i \times \mathcal{U}_i \mid x_i \leq x_{i+1} \text{ for } i \text{ odd } \}$$

i.e. the set of hyper-rectangles bounded by quantile points. Analoguously to the case of the exact algorithm, not all elements of $\mathcal{H}$ need to be considered in the optimization process: once all pairs of boundaries have been fixed till some coordinate k, only those points whose first k coordinates lie in the specified boundaries determine potentially useful boundaries in $\mathcal{L}_{k+1} \times \mathcal{U}_{k+1}$.

The computational order reduces to Q^{2d} rather than n^{2d}. As Q is taken much smaller than n, this allows to control computational speed by ajusting Q to keep Q^{2d} below some desired threshold and to handle arbitrary sample sizes. In all the experiments presented in Section 4 the value of Q is optimized over small values ranging from 2 to some maximum $Q(d)$ by using a validation sample.

3.2 Avoiding premature stopping of the growing process

When growing a HSP tree with an exact algorithm, we can always repeat the splitting procedure so many times as to obtain a totally pure partition with respect to the groups to be discriminated. On the contrary, by using a limited number of admissible boundaries in the construction of the hyper-rectangles, we cannot ensure that the process will lead to any given level of purity on the partition. Indeed the set of admissible hyper-rectangles is not guaranteed to contain a solution allowing to reduce the empirical error as we can no longer isolate arbitrary (small) regions. The problem is illustrated on the theoretical example of Figure 7 for $Q = 2$. On the node $H_0 = [0, 2]^2$ the dark hyper-rectangle represents the region where $Y = 1$; optimization will be performed over the hyper-rectangles taking the form $[i, j] \times [k, \ell]$ with $i, j, k, \ell \in \{0, 1, 2\}$, $i < j$ and $k < \ell$. However none of these 8 hyper-rectangles reduces the empirical error as they all contain the same majority group as the search region H_0.

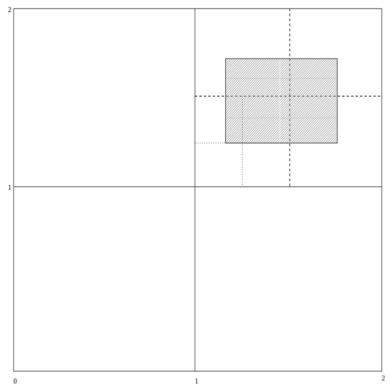

Figure 7: Selection of hyper-rectangle by successive refinements of search region ($Q = 2$)

When some hyper-rectangle H minimizing the empirical error can be found in $\mathcal{H}(H_0 \setminus$

$\cup_{i=1}^{s} H_i$), the optimization process is repeated in the right and left children of the node, i.e. in H and its complementary set $H_0 \setminus (\cup_{i=1}^{s} H_i \cup H)$ respectively. If no hyper-rectangle in $\mathcal{H}$ improves empirical error, the optimization process is repeated in the hyper-rectangle $\tilde{H} = (\ell_{i_1}, u_{i_1}, \ell_{i_2}, u_{i_2}, \ldots, \ell_{i_d}, u_{i_d})$ in $\mathcal{H}$ with highest number of badly classified observations, and the solution $H \subset \tilde{H}$ defines the right and left children of $H_0 \setminus \cup_{i=1}^{s} H_i$. On the example of Figure 7, the optimization process will be repeated first on $\tilde{H}_1 = [1,2]^2$ and finally on $\tilde{H}_2 = [1,1.5]^2$ to yield the hyper-rectangle $H = [1.25,1.5]^2$ and its complementary in $[0,2]^2$ as right and left children of H_0. This growing procedure provides a sequence of finer and finer partitions $\{\mathcal{P}_i\}$ while allowing to control the computational complexity.

The consistency property is not preserved however for this sequence: regions where no single good cut can reduce impurity will remain undivided until late in the process whereas minor local divisions will go on. The example of Figure 8 suffices to highlight the problem. For $Q = 2$, cuts will always be performed in the leftmost square where Y takes value 1 on $[0,1/2] \times [0,1]$ and on all sets of the form $[\sum_{i=1}^{k} 1/2^i, \sum_{i=1}^{k+1} 1/2^i] \times [0,1]$ with k even; the rightmost square will not be split. This problem will be addressed in Section 3.4.

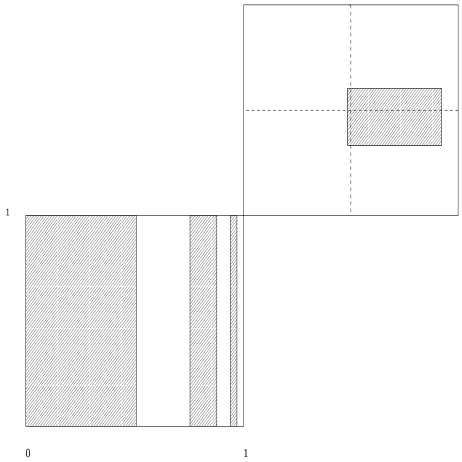

Figure 8: Inconsistency of the sequence of classifiers ($Q = 2$)

3.3 Local adjustments

Suppose that we have split some node H_0 (possibly $H_0 \setminus \cup_{i=1}^{s} H_i$) by adding some hyper-rectangle H, solution of the optimization problem over $\mathcal{H}(H_0)$ and let n_{H_0} be the number of sample points in H_0. Especially at early stages of the process, the solution H serves at the very most as rough approximation of the ideal region to be identified. According to the aim of a modeling algorithm the latter region is characterized by a higher density of the opposite group with respect to the dominant group in H_0 while globally minimizing the error rate on the generated partition. To avoid unnecessary fragmentation of this region over H and its complementary set in H_0 in further steps, the following adjusment procedure is applied. If H is defined by $(\ell_1, u_1, \ldots, \ell_i, u_i, \ldots, \ell_d, u_d)$, we consider each of the $4d$ extensions or reductions of the hyper-rectangle obtained by moving one of the $2d$ boundaries to the inside or to the outside of the hyper-rectangle so as to add or to remove a small fraction of the data: $(\ell_1, u_1, \ldots, \ell_i^+, u_i, \ldots, \ell_d, u_d)$, $(\ell_1, u_1, \ldots, \ell_i^-, u_i, \ldots, \ell_d, u_d)$, $(\ell_1, u_1, \ldots, \ell_i, u_i^+, \ldots, \ell_d, u_d)$ and $(\ell_1, u_1, \ldots, \ell_i, u_i^-, \ldots, \ell_d, u_d)$ for $i = 1, \ldots, d$ where the boundaries $\ell_i^-, \ell_i^+, u_i^-$ and u_i^+ are chosen so as to include or exclude $\alpha\, n_{H_0}$ points at most (Figure 9). On the hyper-rectangle leading to the highest improvement of the empirical error the adjusment procedure is repeated until no improvement can result from any movement. At that stability point, adjustment stops and the current hyper-rectangle is taken as the solution in H_0. The parameter α plays the role of a smoothing parameter. Small values allow to compensate for misdirected steps, but too small values would result in adjustments driven by noise. For the examples presented in Section 4, the value of α has been fixed to 1%.

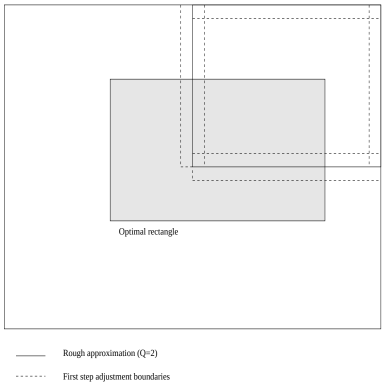

Figure 9: Local adjustment procedure ($Q = 2$)

This produces substantial improvements to the local solution but also to the global tree by reducing the fragmentation of the hyper-rectangles over multiple leaves of the tree.

3.4 Pruning issues

As was already mentioned in Section 3.2 none of the trees appearing in the growing phase of the process is guaranteed to provide a consistent classifier. Analoguously to the case of ordinary binary trees, some pruning method is needed to compensate for this. We use a method related to Reduced Error Pruning (Quinlan, 1993) with minor adaptations intended to take advantage of the particular features of HSP trees. Notice indeed that the left and right children L and R of any node H can either be cut to make H a leave, or they can be replaced in a very natural way by the left and right children of L. In this case the hyper-rectangle defined by R is removed as well as its possible included structure while hyper-rectangles in deeper nodes of the subtree rooted at L do not need to be pruned. This defines quite a natural extension of the notion of subtree due to the type of splits used: disjoint hyper-rectangles can always be removed independently of each other in the structure generated by a HSP tree (see Figure 10). So minor splits having occured before more important ones can very easily be discarded according to any of the numerous existing pruning techniques, using either statistical tests or a separate validation set for instance.

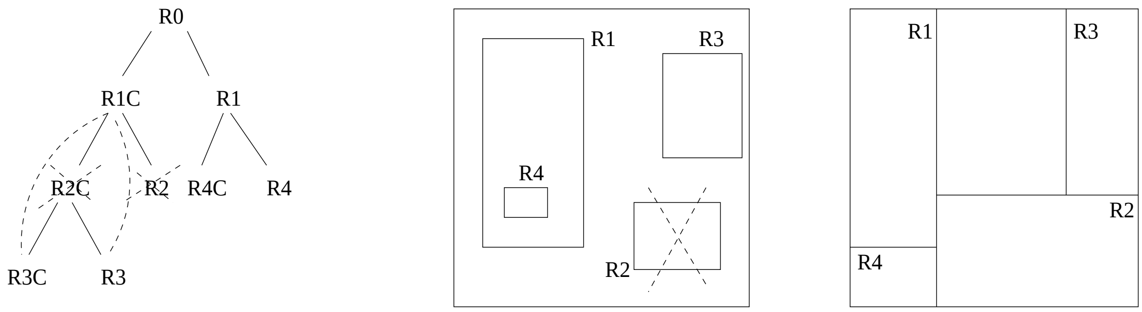

Figure 10: Partitions corresponding to a HSP tree classifier (on the left) and to an ordinary tree classifier (on the right). In this example, for the HSP tree the structure induced by R_3 can easily be maintained whether or not R_2 is removed.

We use the simplest version of the latter method in the following way. The learning sample is randomly split into a training set (approximately 70% of the data) and a validation set. The training set is used to build a maximally extended tree: the growing process described in the previous sections goes on until the finest structure is reached. To select a final tree classifier we extract from this complex tree the structure minimizing the error rate commited on the validation set.

For problems where the number of learning cases is not large, setting aside some part of the data only for pruning purposes may not be an optimal strategy (e.g. Esposito et al.,1997). However we leave the issue of adapting better pruning methods for further research; the results reported in the next section all relate to the performance of HSP trees when used with the reduced error pruning method described above.

4 Some comparative experiments

We present in this section the results of some comparative experiments performed on simulated data sets and on commonly used real data sets from the UCI public repository (Murphy and Aha, 1996).

4.1 Artificial data sets

The simulations presented in this section enhance the potential practical benefits of HSP trees as well as the problems related to the particular approximation used here to reduce their computational complexity. Three data designs have been retained to this end, illustrating some or the other aspect of the method. The error rates provided in all the results that follow come from Monte Carlo experiments for which 100 data sets were simulated; the Monte Carlo standard deviations ($SD/\sqrt{100}$) are mentioned in brackets. The tree solutions obtained by optimization over small values of the parameter Q are noted HSP* in this section.

For the first design we come back to the example of Section 2.2 (XOR data) in a slightly extended setting: additional noisy variables (up to 10) and higher sample sizes (up to 1000) were also considered. With 12 predictors we only optimized over values of Q inferior to 5; but for this particular design even the smallest value $Q = 2$ is expected to provide accurate classifiers. The results are reported in Table 2. The conclusions are identical as for the exact

		S+		HSP ($Q = 2$)	
n=100	xor.b0	0.065	(0.005)	0.040	(0.003)
	xor.b3	0.254	(0.015)	0.050	(0.004)
	xor.b10	0.393	(0.015)	0.060	(0.003)
n=1000	xor.b0	0.008	(0.001)	0.006	(0.001)
	xor.b3	0.063	(0.007)	0.007	(0.001)
	xor.b10	0.204	(0.014)	0.008	(0.001)

Table 2: Mean error rates (and standard deviations) for XOR data sets with b noisy variables

algorithm in this extreme situation: the HSP trees in their approximate version recover perfectly the underlying structure and have an outstanding stability, while this problem is totally impossible for ordinary trees, especially in the presence of noise.

The DAM data sets are drawn from the highly fragmented two-dimensional "4×4 checkerboard" distribution for three levels of the Bayes error L^* (0, 0.10 and 0.20). For this problem the accuracy of the HSP tree classifiers increases with the value of Q, as can be seen on Figure 11. The mean accuracies and Monte Carlo standard deviations for sample sizes equal

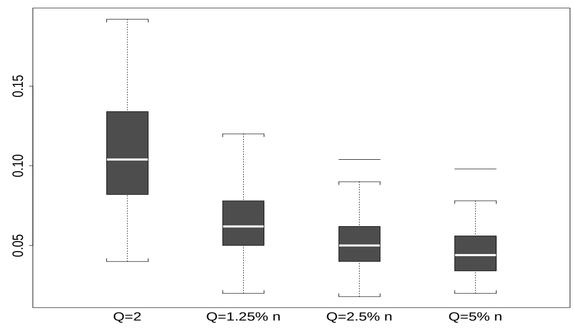

Figure 11: Performance of HSP trees versus parameter Q for the DAM data sets with $L^* = 0$.

to 800 are presented in Table 3, while the boxplots of Figure 12 represent the accuracies as measured by the difference between the probability of error of the classifiers and the Bayes error. The variance of the estimates is still lower for HSP trees, despite the rough

		S+		HSP*	
$n = 800$	$L^* = 0$	0.049	(0.002)	0.046	(0.001)
	$L^* = 0.10$	0.184	(0.003)	0.161	(0.002)
	$L^* = 0.20$	0.314	(0.005)	0.300	(0.004)
$n = 1600$	$L^* = 0$	0.020	(0.001)	0.023	(0.001)
	$L^* = 0.10$	0.136	(0.002)	0.134	(0.002)
	$L^* = 0.20$	0.256	(0.003)	0.249	(0.002)

Table 3: Mean error rates (and standard deviations) for the DAM data sets

approximation. The structure is correctly recovered with some more imprecise boundaries than with the exact algorithm. By doubling the training sample size, S+ trees perform better but their complexity increases dramatically.

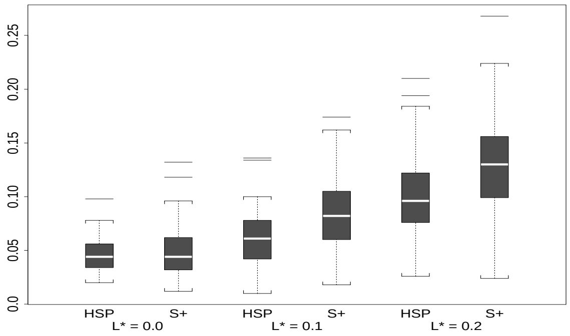

Figure 12: Error rates minus Bayes risk for the DAM data sets.

The KER data sets are drawn from a distribution for which $Y = 0$ everywhere in a cube of $I\!R^3$ except on 3 small disjoint regions ("clusters") where $Y = 1$ with probability p, for p taking values between 0.1 and 0.5. The higher the density on the clusters, the easier the classification task; in this case indeed clusters are not likely to be grouped in one single node, and the accuracy increases with Q. For values of p inferior to 0.3, and more generally for common problems, the effect of the parameter Q varies in a more chaotic way. Some refinements of our current rough implementation would allow to reduce this variability over the parameter Q by a better use of the training sample on the one hand (the same validation sample is used for pruning and for optimizing over Q), and by better choices of the cut points on the coordinates on the other hand (the class labels are not taken into account). Figure 13 represents for several values of p and for $n = 400$ the relative performances obtained with $Q = 2$, i.e. the ratios of error rates to the error rate p of the default rule; the absolute error rates and the Monte Carlo standard deviations are in Table 4.

	S+		HSP*	
$p = 0.1$	0.047	(0.002)	0.055	(0.002)
$p = 0.2$	0.037	(0.002)	0.037	(0.002)
$p = 0.3$	0.036	(0.002)	0.035	(0.001)

Table 4: Mean error rates (and standard deviations) for the KER data sets

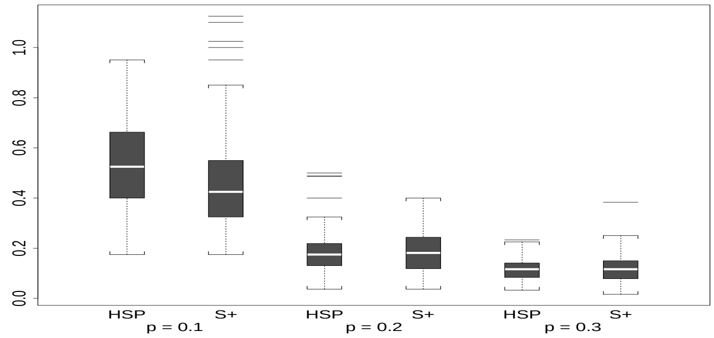

Figure 13: Relative performance on KER data sets for increasing density on clusters.

4.2 Results on problems from the UCI repository

The following five data sets are all discrimination problems coming from the UCI repository and involving two classes: Australian credit data (AC), Bupa liver disorder data (BL), Breast cancer Wisconsin data (BCW), Heart disease data (HD) and Pima Indian diabetes data (PID). The number of predictive variables used and the number of cases are recorded in columns 2 and 3 of Table 5. We reproduce the same holdout resampling experimental design as the one proposed in Shih et al. (2000) so as to ensure valid comparisons with those results: most of the error rates recorded in Tables 5 and 6 come from the detailed appendix of this study or from the results of the Statlog project (Michie et al., 1994). The following 10-fold cross-validation process has been applied to each of the five data sets. The original dataset is divided into 10 disjoint subsets; each of which will form the independent test set for the classifiers constructed with the learning sets composed of the union of the 9 remaining subsets. Sampling is stratified by the class labels to ensure that the subset class proportions are roughly the same as those in the whole data set. The part of the data used to build the classifier is randomly split into training (approximately 70%) and validation sets. The 10 cross-validation estimates of the error rate are averaged to provide an estimate for the classifier build with the whole data set.

Although the algorithm for HSP trees is designed for continuous variables, the extension to categorical variables is practically easily feasible; we include all categorical variables whose number of levels is inferior to 3. The problem of the selection bias (Kim and Loh, 2001) also applies to HSP trees; procedures like those proposed by the latter authors could be extended

Data set	d	n	LDA	QDA	MVmin
BL	5	345	0.326	0.401	0.279
PID	7	532	0.221	0.238	0.221
BCW	9	683	0.039	0.048	0.031
AC	12	690	0.141	0.207	0.131
HD	13	270	0.141	0.248	0.148

Table 5: Cross-validation estimates of error rates for linear and quadratic discriminant analysis, and for multivariate trees.

in our multivariate setting.

Table 5 contains the error rates of linear and quadratic discriminant analysis. Besides those two classical statistical algorithms, we give the minimum error rate over the 22 tree methods compared in Shih et al. (2000), except for the Australian Credit data set which was not included in the latter study. For the four other data sets, the minimum error rate was reached by tree methods allowing linear combinations of the predictors. However these methods do not preserve the property of invariance under monotone transformations of the coordinate axes; on the other hand, they do not allow easy interpretation of the classification rules generated, which is another appreciable feature of ordinary binary trees. The rules generated by HSP trees depend only on the coordinatewise ranks of the data points i.e. they are invariant under monotone transformations of the coordinate axes. On the other hand they are easily interpretable in terms of the original variables similarly to tree methods based on univariate splits. These properties - invariance and interpretability - are shared by the tree methods compared in Table 6: CART as the version implemented in the IND package (Buntine and Caruana, 1992) with the 1-SE pruning rule, C4.5 (Quinlan, 1993), Splus tree (Venables and Ripley, 1994) with the 1-SE pruning rule and CAL5 (Muller and Wysotzki, 1997). The minimum error rate in this category of tree classifiers is recorded in the second column; it is taken over a broader set of classifiers defined in Shih et al. (2000).

The accuracy of HSP trees in their quantile based version compare with that of the more elaborated traditional tree methods, and the tree sizes are always much smaller. The cross-validation estimates of the error rates are given for (very) small values of the parameter Q: $Q = 2$ for data sets with more than 7 predictors, and Q optimized up to 10 and 4 for the BL and PID data sets respectively.

Data set	min	CART	C4.5	S+	CAL5	HSP
BL	0.292	0.319	0.308	0.326	0.420	0.343
PID	0.226	0.239	0.242	0.250	0.226	0.237
BCW	0.034	0.047	0.043	0.050	0.071	0.049
AC	0.131	0.152	0.155	0.145	0.131	0.149
HD	0.196	0.219	0.196	0.233	0.270	0.248

Table 6: Cross-validation estimates of error rates for monotone transformation invariant tree methods.

5 Conclusion

Hyper-rectangular Space Partitioning trees are multivariate tree classifiers designed for continuous predictors. They allow easy interpretation of the solutions, generating rules expressed as simple conditions on the predictive variables, and they possess the property of monotone transformation invariance. Besides their outstanding theoretical properties, the few experiments conducted in this work indicate that it would be worth considering further approaches for fully practical methods. The simple algorithms developed in this paper allowed to highlight some interesting features such as increased stability of the tree solution including in the presence of noise, shorter trees and good relative results especially on moderate or smaller sample sizes. Numerous "natural" developments can be considered relatively to the usual features of tree methods (alternative variable selection criteria, split points selection criteria, pruning techniques,...) and relatively to the particular aspects related to their necessary approximation.

References

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees.* Wadsworth International, Belmont, CA.

Buntine, W. and Caruana, R. (1992). *Introduction to IND Version 2.1 and Recursive Partitioning.* NASA Ames Research Center, Moffet Field, CA.

Clark, L. and Pregibon, D. (1993). Tree-based models. In Chambers, J. and Hastie, T., editors, *Statistical Models in S*, pages 377–419. Chapman & Hall, New York, NY.

Clark, P. and Niblett, R. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–284.

Cohen, W. W. (1995). Fast efficient rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 115–123, Lake Tahoe, CA. Morgan Kaufmann.

Devroye, L., Gyorfi, L., and Lugosi, G. (1996). *A probabilistic theory of pattern recognition.* Springer-Verlag, New-York.

Esposito, F., Malerba, D., and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on pattern analysis and machine intelligence*, 19(5):476–491.

Friedman, J. and Fisher, N. (1999). Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143.

Kim, H. and Loh, W. (2001). Classification trees with unbiased multiway splits. *Journal of Amer. Statist. Assoc.*, 96:598–604.

Michie, D., Spiegelhalter, D. J., and Taylor, C. C. (1994). *Machine Learning, neural and statistical classification.* Ellis Horwood.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill.

Muller, W. and Wysotzki, F. (1997). The decision tree algorithm CAL5 based on a statistical approach to its splitting algorithm. In *Machine Learning and Statistics: The Interface*, pages 45–65. John Wiley & Sons, New York, NY.

Murphy, P. M. and Aha, D. W. (1996). *UCI Repository of machine learning databases.* Department of Information and Computer Science, University of California, Irvine, CA.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Shih, Y.-S., Lim, T.-S., and Loh, W.-Y. (2000). A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40:203–228.

Venables, W. and Ripley, B. (1994). *Modern Applied Statistics with S-Plus.* New-York, NY: Springer-Verlag.