Experiments on a Representation-Independent "Top-Down and Prune" Induction Scheme

Richard Nock[1], Marc Sebban[1], and Pascal Jappy[2]

[1] Univ. Antilles-Guyane, Dept of Maths and CS,
Campus de Fouillole, 97159 Pointe-À-Pitre, France
{rnock,msebban}@univ-ag.fr
[2] Leonard's Logic, 20 rue thérèse, 75001 Paris, France
pjappy@leologic.com

Abstract. Recently, some methods for the induction of Decision Trees have received much theoretical attention. While some of these works focused on efficient top-down induction algorithms, others investigated the pruning of large trees to obtain small and accurate formulae. This paper discusses the practical possibility of combining and generalizing both approaches, to use them on various classes of concept representations, not strictly restricted to decision trees or formulae built from decision trees. The algorithm, WIRE*i*, is able to produce decision trees, decision lists, simple rules, disjunctive normal form formulae, a variant of multilinear polynomials, and more. This shifting ability allows to reduce the risk of deviating from valuable concepts during the induction. As an example, in a previously used simulated noisy dataset, the algorithm managed to find systematically the target concept itself, when using an adequate concept representation. Further experiments on twenty-two readily available datasets show the ability of WIRE*i* to build small and accurate concept representations, which lets the user choose his formalism to best suit his interpretation needs, in particular for mining purposes.

1 Introduction

Many of the classical problems in designing machine learning (ML) algorithms can be understood by means of accuracy, time/space complexity, size and intelligibility issues. Generally, satisfying most of them is essentially a matter of compromises. In such cases, the problem is to transform rapidly enough the dataset to a useful compact representation that, while capturing most of the generalizable knowledge of the original data, will stay sufficiently small to be intelligible and interpretable. While the rapid increase in computer's performances has somewhat de-emphasized the time requirements to obtain the algorithm's outputs [Qui96], the other requirements cannot be easily solved. As an example, it was recently observed that the end user of ML algorithms is likely to appreciate various output types against other ones, that is, not a single *concept representation* class fits to all users, and the ability to shift in practice the output type is of great importance. This is also important from a theoretical viewpoint.

J.M. Żytkow and J. Rauch (Eds.): PKDD'99, LNAI 1704, pp. 223–231, 1999.

Some problems admit a small coding on some concept representation, but lead to overly large representations on other classes.

There has been recently much work to establish sound theoretical bases for the induction of decision trees, to explain and improve the behavior of algorithms such as C4.5 [KM96, KM98, SS98]. Such algorithms proceed by a "top-down and prune" scheme: a large formula is induced, which is pruned in a latter step, to obtain a small and accurate final output. While [KM96, SS98] have focused on improving the top-down induction, [KM98] have established the theoretical bases of a new pruning scheme, with theoretically proven near-optimal behavior. These schemes, thought initially focused on decision trees, have remarkable general properties, which can be applied outside the class of decision trees. A previous study [NJ98] shows that the class of decision lists, which shares close properties with decision trees, can benefit of principles closely related to the top-down induction. In that paper, we are concerned by the generalization of the whole "top-down and prune" scheme to a very large scope of concept representations. More precisely, we propose a general principle derived from the weak learning framework of [SS98] and the pruning framework of [KM98], to which we relate to as WIRE*i* (for Weak Induction REpresentation-*i*ndependent). WIRE*i* is able to induce on any problem formulae such as Decision Lists (DL), Decision Committees (DC, a variant of multilinear polynomials), Decision Trees (DT), Disjunctive Normal Form formulae (DNF), simple monomials, and more.

WIRE*i* is much different from approaches such as C4.5RULES, which proposes to induce rules from DT. Indeed, in C4.5RULES, a DT is always primarily induced, which in a subsequent step is transformed into a set of rules. WIRE*i*, on the other hand, processes directly formulae inside the chosen class. Experiments carried out on twenty-two publicly available domains reveal that on each dataset, concept representations built from various classes can be much different from each other while still being small and accurate. WIRE*i* was also able to exhibit on runs over noisy domains the target formula itself, thus achieving an optimal compromise between accuracy and size. The time complexity of WIRE*i* compares favorably to that of classical approaches such as C4.5.

After a general presentation of WIRE*i*, and its applications to a large scope of concept representation classes, we relate experiments conducted using WIRE*i* on twenty-two domains, almost all of which can be found on the UCI repository of machine learning database [BKM98].

2 WIRE*i*

Throughout the paper, the following notations are used: LS denotes the set examples used for training, each of which is described with n attributes, and belongs to one class among c. The following subsections present the basis of the growing and pruning algorithms. For the sake of clarity, an applicative example (generally on DT) is provided for all, and the specific applications to other classes are presented on a subsequent devoted part.

2.1 A General Top-down Induction Algorithm

The principle is to repeatedly optimize, in a top-down manner, a particular Z criterion over the partition induced by the current formula f on LS. This partition into subsets $LS_1, LS_2, ..., LS_k$ satisfies the following two axioms:

1. $\forall 1 \leq j \leq k$, any two examples of LS_j are classified exactly in the same *fashion*,
2. $\forall 1 \leq i < j \leq k$, any two examples of respectively LS_i and LS_j are *not* classified in the same fashion.

It is important to note the term "fashion" instead of "class". Two examples classified in the same fashion follow exactly the same path in the formula, *e.g.* the same leaf in a DT. To each example is associated a weight, which mimics its appearance probability inside LS (if uniform, all weights equal $1/|LS|$, where $|.|$ is the cardinality function). We adopt the convention that examples are described using couples of the type (o, c_o), where o is an observation, and c_o its corresponding class; its weight is written $w((o, c_o))$. Fix as $[\![\pi]\!]$ the function returning the truth value of a predicate π. Define for any class $1 \leq l \leq c$ and any subset LS_j of the partition the following quantities:

$$W_+^{j,l} = \sum_{(o,c_o) \in LS_j} w((o,c_o))[\![c_o = l]\!] \ ; \ W_-^{j,l} = \sum_{(o,c_o) \in LS_j} w((o,c_o))[\![c_o \neq l]\!]$$

In other words, $W_+^{j,l}$ represents the fraction of examples of class l present in subset LS_j, and $W_-^{j,l}$ represents the fraction of examples of classes $\neq l$ present in subset LS_j. The Z criterion of [SS98] is the following:

$$Z = 2 \sum_j \sum_l \sqrt{W_+^{j,l} W_-^{j,l}}$$

The core of procedure `TDbuild` simply consists in repeatedly optimizing the decreasing of the current Z, until either no decreasing is possible, or some upper-bound $Imax$ of the formula's size is reached. In order to keep a fast procedure, in any rule-based formula (*e.g.* DL, DNF), the current search is focused on a currently grown rule, until a new one is grown when no addition in the current rule decreases Z.

2.2 A General Pruning Algorithm

The objective is to test exactly once the removal of each of the subparts of the formula f obtained from `TDbuild`. The test is bottom-up for all formula based on literal or rule-ordering, such as decision trees or decision lists. For other formulae without ordering, such as DNF, it "simulates" a bottom-up scanning of the formulae. The ordering of the former formulae supposes that some parts Q of the formula may only be reached after having reached another part P. In the case of decision trees, P is an internal node, and all possible Q are internal

nodes belonging to the subtree rooted at P. The test evaluates the possible removal of all Q *before* testing P, and whenever P is removed, all depending Q are also removed, leading to the entire pruning of the subtree rooted at P, itself replaced by the best leaf other the examples reaching P. Tests for other formulae will be detailed in their devoted subsections. Algorithm 1 presents

Algorithm 1: `BUprune`(LS, f, δ, s_{eq})

Input: sample LS, formula f, real $0 < \delta < 1$, integer s_{eq}
Output: a formula f
foreach $P \in f$ **scanned bottom-up do**
  $C :=$`Ways(`P`)`; $H :=$`Sons(`P`)`; $s_{loc} :=$ |`Reach(`P, f, LS`)`|$\times s_{eq}/(|LS|(c-1)^2)$;
  $\alpha := \sqrt{(\log C + \log H + 2\log s_{eq}/\delta)/(s_{loc})}$;
  $\epsilon_P :=$ `lError(`f,`Reach(`P, f, LS`))`; $\epsilon_\emptyset :=$ `lError(`$f \backslash P$,`Reach(`P, f, LS`))`;
  if $\epsilon_P + \alpha \geq \epsilon_\emptyset$ **then**
    `Remove(`P,f`)`;
return f

`BUprune`. We emphasize the fact that `BUprune` is an *application* of the theoretical results of [KM98]. The parameters used are the following ones. `Ways`(.) returns the number of distinct formulae which could replace in f the series of tests to reach P. In a decision tree, this represents the number of distinct monomials whose length equal the depth of P. `Sons(.)` returns the number of distinct subformulae in f that could be placed after P, without changing the size of f. In a decision tree, this represents the number of distinct subtrees that can be rooted at P without changing the whole number of internal nodes of f. `Reach(.,.,.)` returns the subset of examples from LS reaching P in f. In a decision tree, this represents the subset of LS reaching the internal node P. `lError(.,.)` returns the *local error* over `Reach(.,.,.)`, in the formula f (for ϵ_P), or f to which P and all subformulae of P are removed (for $\epsilon_\emptyset$). In the case of a decision tree, the latter quantity corresponds to the local error of the best leaf rooted at P. The term "local error" is very important: in particular, the distribution used to calculate `lError(.,.)` is such that all examples from $LS \backslash$`Reach(`P, f, LS`)` have zero weight. s_{eq} is a correction factor, which is not in [KM98]. We now explain its use. The test to remove P is optimistic, in that we face the possibility to overprune the formula, all the more if LS is not sufficiently large. For example, consider the case $c = 2$, $|LS| = 2000$, |`Reach(`P, f, LS`)`| $= 100$, $\delta < .20$ and $s_{eq} = |LS|$. Then we obtain $\alpha > .40$, even when considering $C = H = 1$. Experimentally, this shortcoming may lead to an empty formula, by pruning all parts of the initial formula. In order to overcome this difficulty, we have chosen to "mimic" the re-sampling of LS into another set of size $s_{eq} > |LS|$, in which examples would have exactly the same distribution as in LS. In our experiments, the values of C and H, since having a hard fast calculation, were approximated with upperbounds as large as possible, still in order not to face this possibility of overpruning. The bounds are not as tight as one could expect, yet they gave

experimentally good results. We now go on detailing the algorithms `TDbuild` and `BUprune` for various kinds of formalisms.

3 Applications of WIRE*i* to Specific Classes

Fix $u(k) = 2^k \times n!/((n-k)!k!)$ (fast approximations of $u(.)$ can be obtained by Stirling's formula). This represents the number of Boolean monomials of length k over n variables. The application of WIRE*i* to DT mainly follows from our preceding comments, and the results of [SS98, KM98, Qui96]. Due to the lack of space, we only detail results on other formalisms. The most simple is for monomials. When a single monomial f is needed, associated to a fixed class to which we refer as the positive class, only algorithm `TDbuild` is used. There are only two subsets LS_1 and LS_2 in the partition of LS, containing respectively examples satisfying the monomial, and those which do not satisfy the monomial. We additionally put the following constraint: each test added keeps the positive class as the majority class for the examples satisfying f. This gives the algorithm WIRE*i*(RULE).

Decision Lists: WIRE*i*(DL). `TDbuild`: for a DL with m monomials, the partition of LS contains $m+1$ subsets. The m first subsets are those corresponding to a monomial, and the $(m+1)^{th}$ corresponds to the default class. `Optimize(.)` proceeds as follows. Each possible test is added to the last rule of the decision list. When no further addition of a test decreases the Z value, a new rule, created in the last position, is investigated.
`BUprune`: for a DL with m monomials, each P is a monomial, and the monomials are tested from the last monomial of the DL to the first one. `Reach(.,.,.)` returns the subset of examples reaching P. When pruning a monomial P, all monomials following P (that were not pruned) are removed with P. The best default class other the training sample replaces P. Fix as l the position of P inside the DL. We then choose $C = u(l-1)$. Fix as t the average number of literals of each monomial following P. Then, we fix $H = (m-l)u(t)$.

DNF: WIRE*i*(DNF) , is used when $c = 2$. `TDbuild`: for a DNF with m monomials, the partition can contain up to $min\{|LS|, 2^m\}$ subsets (this quantity is never greater than $|LS|$, which guarantees efficient processing time). Each subset contains the examples satisfying exactly the same subset of monomials. While there is no ordering on monomials, algorithm `TDbuild` is still bottom-up. Each test is added to a current monomial. When no further addition of a test into this monomial decreases the Z value, a new monomial is created, initialized to $\emptyset$, and treated as the current monomial. The same constraint as for monomials is used when minimizing Z: each test added to a monomial keeps the positive class as the majority class for all examples satisfying this monomial.
`BUprune`: while there is no ordering on monomials, the bottom-up fashion is still preserved for the formula f. Each P represents a monomial of the DNF, and when removing P, no other monomial is removed. `Reach(.,.,.)` returns the

subset of examples satisfying P. Fix as l the total number of monomials $\neq P$, inside f, that could be satisfied while satisfying P, and t their average length. In other words, each of these monomials must not have a contradictory test with P. Fix as $|P|$ the number of literals of P. We choose $C = u(|P|)$ and $H = l \times u(t)$. In addition, s_{loc} is the cardinality of the examples satisfying P.

Decision Committees: WIRE*i*(DC). We use DC with constrained vectors. Such a DC [NG95, NJ99] contains two parts:

- A set of unordered couples (or rules) $\{(m_i, \boldsymbol{v}_i)\}$ where each m_i is a monomial, and each $\boldsymbol{v}_i$ is a vector in $\{-1, 0, 1\}^c$ (the values correspond to the natural interpretation "is in disfavor of", "is neutral w.r.t.", "is in favor of" one class).
- A Default Vector $\boldsymbol{D}$ in $[0, 1]^c$.

For any observation o we calculate $\boldsymbol{V}_o$, the sum of all vectors whose monomials are satisfied by o. The index of the maximal component of $\boldsymbol{V}_o$ gives the class assigned to o. If it is not unique, we take the index of the maximal component of $\boldsymbol{D}$ corresponding to the maximal component of $\boldsymbol{V}_o$. Algorithm `BUprune` has the same structure as for DNF. However, in order not to artificially increase the power of the vectors by multiplying the appearance of some monomials, we do not authorize the addition of multiple copies of a single monomial, a case which can only occur when the current Z is not decreased.
`TDbuild`: it is the same as for DNF, except that we remove the constraint on choosing monomials discriminating the positive class. Before executing algorithm `BUprune`, we calculate the components of each $\boldsymbol{v}_i$. To do so, we use the algorithm of [NJ99] which proceeds by minimizing Ranking Loss as defined by [SS98].

4 Experimental Results

WIRE*i* was evaluated on a representative collection of twenty-two problems, most of which can be found on the UCI repository [BKM98]. The only exceptions were the "LEDeven" and "XD6" domain. "LEDeven" consists in the noisy ten-classes problem "LED10" with classes reunited into *odd* and *even* classes. "XD6" consists in a two-classes problem with ten description variables for each example. The target concept, from which all examples are uniformly sampled, is a DNF with three variables in each of its monomials, described over the first nine variables. The tenth variable is irrelevant in the strongest sense. A 10% classification noise is added, which also represents Bayes optimum. References for all the datasets, omitted due to space constraints, can be found in [BN92, Hol93, Qui96], or on the UCI repository [BKM98]. All algorithms are ran using $\delta = 15\%$, $|s_{eq}| = 10000$ and $Imax = 40$, in order to make clear comparisons. Ten complete 10-fold stratified cross-validations were carried out with each database. In a ten-fold cross-validation, the training instances are partitioned into 10 equal-sized subsets with similar class distributions. Each subset in turn is used for testing

while the remaining nine are used for training. Due to the lack of space, only experiments with WIREi(DC), WIREi(DNF) and WIREi(DL) are shown in table 1. With respect to each algorithm, in its first column are shown error rates averaged over the 10-fold cross-validations, with the average number of monomials (second column), and the average total number of literal on the third column (if a literal appears k times, it is counted k times). Column "Others" relates various results among the best we know, for which the experiments were carried out under a similar setting as ours. Over the 22 datasets, WIREi outperforms many of the traditional approaches. Comparing the errors gives already an advantage to WIREi(particularly WIREi(DL)), but improvements become more sensible as the errors are compared in the light of the corresponding formula's sizes. Size reductions, while still preserving in many cases a better error, can range towards magnitude order of twenty or more. In particular, WIREi(DL) is a clear winner against CN2 when considering both errors and sizes. But there is more to say, when comparing approaches head to head.

More than performing a comparison between accuracies, we performed a comparison between the classifiers themselves on specific problems. On "XD6", we observed that the classifiers built for both WIREi(DNF) and WIREi(DL) are always exactly the target formula, beating in both accuracy and size classical DT induction approaches [BN92]. However, coding the target formula with DT can be done modulo the creation of comparatively large trees, which is more risky when building formula in a top-down fashion: chances are indeed larger that the formula built deviates from the optimal one. This clearly accounts for the representation shift WIREi proposes. On "Vote0", we still obtained exactly the same classifiers for WIREi(DNF) and WIREi(DL), with one literal. This problem is known to have one attribute which makes a very reliable test [BN92], attribute which is precisely always selected by WIREi(DNF) and WIREi(DL). In order to cope with this problem, [BN92] propose to remove this attribute, which gives the "Vote1" problem. While DT induction algorithms give much larger formulae, WIREi(DL) always manages to find a two-tests rule which still gives very good results, and might contain useful informations for Data Mining purposes. However, the problem seems indeed more difficult since WIREi(DC) finds more complex formulae with average accuracy slightly below 10%, a seldom result if we refer to the collection of reported studies in [Hol93], none of which break the 10% barrier. This stability property was also remarked on the "Horse-Co" problem, where both WIREi(DL) and WIREi(DNF) even encompassed DT approaches using a very simple concept.

On the "LED10" domain, WIREi(DC) obtained on average a result a little above the 24% Bayes error rate, but WIREi(DL) performed very poorly (while DT give intermediate results). Interestingly, when transforming the problem to "LEDeven", WIREi(DL) achieved near-optimal prediction, with a completely stable classifier, but WIREi(DC)'s prediction degraded with respect to Bayes. A simple explanation for this behavior is that "LED10" is a problem which can be encoded very efficiently using simple linear frontiers around classes [NG95], and it was proven that linear separators, while being DC with one-rule monomials

(remark that WIRE*i*(DC)'s monomials contain on average 1.43 literals), are very difficult to encode using simple DLs [NG95]. On the other hand, "LEDeven" can be related to much simpler concepts, which can be very efficiently coded using DL. We have remarked that the DL obtained by WIRE*i*(DL) were very accurate in that among their two rules, the first contained a test which discriminates all but one (the "4") even digits against all odd digits, and the second coupled with the default class, led to an efficient test to discriminate under noise the "4" digit against all odd digits. When comparing "Glass" and "Glass2", which is a modified version of "Glass" [CB91], the interest of the hypothesis concept shift between the two problems is clear, as WIRE*i*(DC)performed well on "Glass2", while WIRE*i*(DL) gave the best results on "Glass".
Following all these observations, we can say that WIRE*i* is an experimental evidence of the power of simple induction schemes such as "Top-down and prune", which received recently much attention to establish sound theoretical foundations. Though many works were primarily based on decision trees [KM96, KM98], theoretical results seem to be practically scalable to various different classes of formalism representation, three of which were explored in depth in our experiments. Additionally, experimental results reveal that applications of the generic algorithm WIRE*i* to specific classes can exhibit even better behavior than algorithms specifically dedicated to the same classes.

References

[BKM98] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases. 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[BN92] W. Buntine and T. Niblett. A further comparison of splitting rules for Decision-Tree induction. *Machine Learning*, pages 75–85, 1992.

[CB91] P. Clark and R. Boswell. Rule induction with CN2: some recent improvements. In *Proc. of the 6 th European Working Session in Learning*, pages 151–161, 1991.

[Dom98] P. Domingos. A Process-oriented Heuristic for Model selection. In *Proc. of the 15 th International Conference on Machine Learning*, pages 127–135, 1998.

[FW98] E. Franck and I. Witten. Using a Permutation Test for Attribute selection in Decision Trees. In *Proc. of the 15 th International Conference on Machine Learning*, pages 152–160, 1998.

[Hol93] R.C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, pages 63–91, 1993.

[KM96] M.J. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 459–468, 1996.

[KM98] M. J. Kearns and Y. Mansour. A Fast, Bottom-up Decision Tree Pruning algorithm with Near-Optimal generalization. In *Proc. of the 15 th International Conference on Machine Learning*, 1998.

[NG95] R. Nock and O. Gascuel. On learning decision committees. In *Proc. of the 12 th International Conference on Machine Learning*, pages 413–420, 1995.

[NJ98] R. Nock and P. Jappy. On the power of decision lists. In *Proc. of the 15 th International Conference on Machine Learning*, pages 413–420, 1998.

Table 1. Comparisons of various approaches of WIREi (whose least errors are underlined for each domain; a "/" mark for DNF denotes a domain with $c > 2$ classes).

	WIREi(DC)			WIREi(DNF)			WIREi(DL)			
Domain	err (%)	m_{DC}	l_{DC}	err (%)	m_{DNF}	l_{DNF}	err (%)	m_{DL}	l_{DL}	Others
Balance	22.24	6.2	13.7	/	/	/	23.01	4.2	13.5	32.1 *b*
Breast-W	4.08	5.4	22.8	13.80	1.6	2.9	6.90	1.7	6.7	4.0 *b*
Echo	28.57	2.0	3.9	36.42	1.3	2.3	30.00	0.7	1.6	$32.3_{35.4}$ *c*
Glass	53.91	1.3	1.8	/	/	/	38.69	6.0	19.2	$41.50_{32.8}$ *c*
Glass2	21.10	6.6	18.2	23.52	5.7	15.4	22.35	11.2	27.1	20.3 *b*
Heart-St	22.96	3.9	11.7	34.44	2.0	6.6	23.53	9.4	22.6	21.5 *b*
Heart-C	24.87	4.4	14.3	23.87	2.0	5.7	21.93	2.8	7.7	$22.5_{52.0}$ *c*
Heart-H	20.67	5.2	13.8	24.00	1.3	5.1	23.00	1.2	4.5	$21.8_{60.3}$ *c*
Hepatitis	21.76	3.9	10.1	24.70	3.2	6.5	18.82	1.7	4.2	$19.2_{34.0}$ *c*
Horse-Co	22.31	9.5	27.0	$^{\triangle}$13.68	$^{\triangle}$1.0	$^{\triangle}$2.0	$^{\triangle}$13.68	$^{\triangle}$1.0	$^{\triangle}$2.0	14.92 *b*
Iris	5.33	1.9	4.6	/	/	/	2.67	2.0	4.8	$4.9_{3.5}$ *a*
Labor	15.00	4.1	9.5	43.33	1.2	1.9	16.67	3.1	5.6	
Lung	42.50	1.3	3.8	/	/	/	47.50	2.0	5.9	
LED10	†26.95	10.1	14.4	/	/	/	58.26	4.1	9.1	$32.89_{13.0}$ *a*
LEDeven	19.91	5.7	10.3	42.50	1.9	3.9	$^{\dagger,\triangle}$12.08	$^{\triangle}$2.0	$^{\triangle}$6.0	
Monk1	15.00	4.1	9.5	35.44	2.8	3.9	4.11	9.4	18.3	
Monk2	24.26	8.3	35.5	48.69	2.4	5.3	21.97	10.8	44.4	
Monk3	3.93	4.3	6.4	44.11	4.0	4.3	$^{\triangle}$3.57	$^{\triangle}$2.0	$^{\triangle}$2.0	
Pima	28.52	3.3	7.1	38.44	0.7	1.8	25.71	2.4	6.5	25.9 *b*
Vote0	7.70	2.9	4.4	7.73	1.4	1.7	$^{\triangle}$5.68	$^{\triangle}$1.0	$^{\triangle}$1.0	$4.3_{49.6}$ *c*
Vote1	9.95	4.5	10.8	18.86	2.4	3.2	$^{\triangle}$10.23	$^{\triangle}$1.0	$^{\triangle}$2.0	$12.79_{8.9}$ *a*
XD6	20.34	9.4	18.4	$^{\dagger,\triangle}$9.84	$^{\dagger,\triangle}$3.0	$^{\dagger,\triangle}$9.0	$^{\dagger,\triangle}$9.84	$^{\dagger,\triangle}$3.0	$^{\dagger,\triangle}$9.0	$22.06_{14.8}$ *a*

† : near-optimal or optimal results. △ : the same classifier is produced at each fold. $m_{\mathcal{F}}$: average number of monomials (see text). $l_{\mathcal{F}}$: average number of literals (see text). References: *a* [BN92], best reported results on DT induction. The small number indicates the least number of leaves (equiv. to a number of monomials). *b* [FW98, Qui96], C4.5's error. *c* [Dom98, CB91], various improved CN2's error (small numbers indicate the whole number of literals).

[NJ99] R. Nock and P. Jappy. A top-down and prune induction scheme for constrained decision committees. In *Proc. of the 3 rd International Symposium on Intelligent Data Analysis*, 1999. accepted.

[Qui96] J. R. Quinlan. Bagging, Boosting and C4.5. In *Proc. of AAAI-96*, pages 725–730, 1996.

[SS98] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual ACM Conference on Computational Learning Theory*, pages 80–91, 1998.