On the Practice of Branching Program Boosting

Tapio Elomaa and Matti Kääriäinen

Department of Computer Science, P.O. Box 26
FIN-00014 University of Helsinki, Finland
{elomaa,matti.kaariainen}@cs.helsinki.fi

Abstract. Branching programs are a generalization of decision trees. From the viewpoint of boosting theory the former appear to be exponentially more efficient. However, earlier experience demonstrates that such results do not necessarily translate to practical success. In this paper we develop a practical version of Mansour and McAllester's [13] algorithm for branching program boosting. We test the algorithm empirically with real-world and synthetic data. Branching programs attain the same prediction accuracy level as C4.5. Contrary to the implications of the boosting analysis, they are not significantly smaller than the corresponding decision trees. This further corroborates the earlier observations on the way in which boosting analyses bear practical significance.

1 Introduction

The *weak learning model* or *boosting* theory [16,6] has been able to offer an analytical explanation for the practical success of top-down induction of decision trees (subsequently DTs for short) [9,12]. Earlier attempts to explain the success of DT learning in theoretical models have not been successful. Even though the weak learning framework may better suit analyzing and designing practical learning algorithms than the PAC model and its variants, one must exercise care in drawing conclusions about the practical implications of boosting analyses [4]. In this paper we provide further evidence to support meticulous consideration.

A *branching program* (BP) is an abstract model of computation that takes the form of a directed acyclic graph (DAG). BPs have been well-studied in theoretical computer science. In this paper we view them as classifiers. In empirical machine learning a similar representation formalism — *decision graphs* — has been studied to some extent [14,10,11]. BPs are a strict generalization of DTs. Thus, their learning in computational learning frameworks is hard [5,1].

Mansour and McAllester [13] devised a boosting algorithm for BPs. The main advantage obtained by using BPs rather than DTs is that their training error is guaranteed to decline exponentially in the square root of the size of the program. When DT learning algorithms are viewed as boosting algorithms, the training error declination is only polynomial in the size of the tree [9,12]. The learning algorithm for BPs is basically very similar to the algorithms used in top-down induction of DTs [3,15]. It greedily searches for good splits of the nodes in the last level of the evolving program. The central difference between a BP and a

L. De Raedt and P. Flach (Eds.): ECML 2001, LNAI 2167, pp. 133–144, 2001.

DT is that in the former branches may grow together, while in the latter two separate branches never unite.

In this paper we experiment with a practical learning algorithm based on the results of Mansour and McAllester [13]. We clarify the algorithm and test it with data sets from the UCI Repository [2] and some synthetic data. In the experiments BPs attain the same overall prediction accuracy level as unpruned DTs. Domain specific differences, though, are observed; in particular, on some synthetic domains the differences are clear. BPs appear to be slightly smaller than unpruned DTs, but there is no clear difference.

In Section 2 we recapitulate weak learning and boosting, introducing at the same time the framework used subsequently. We also review Kearns and Mansour's [9] analysis of DT learning as a boosting algorithm. Section 3 presents the boosting algorithm for BPs and the motivation behind its details. We also briefly touch the related work of learning decision graphs. In Section 4 empirical experiments with the BP algorithm are reported. The results are contrasted with those obtained by C4.5. Lessons learned from the experimentation are reflected upon in Section 5. Finally, we present the concluding remarks of this paper.

2 Weak Learning, Boosting, and Decision Tree Induction

Let f be a boolean target function over an input space X. A set $\mathcal{H}$ of base classifiers or predicates on X fulfills the *β-weak learning hypothesis* (with respect to f), where $\beta \in (0, 1/2]$, if it contains, for any distribution D on X, a classifier $h \in \mathcal{H}$ such that

$$\mathbf{Pr}_D\{\, h(x) \neq f(x) \,\} \leq \frac{1}{2} - \beta.$$

In other words, the weak learning hypothesis guarantees the existence of a predicate with a predictive *advantage* of at least β on f over random guessing.

Boosting algorithms exploit the weak learning hypothesis by combining many different predicates from $\mathcal{H}$ on different filtered distributions of the original sample. Thus, they amplify the small predictive advantages over random guessing iteratively to obtain a combined function, whose training error is less than any desired error threshold [16,6]. A natural type of boosting algorithm is a *voting* algorithm, which uses some number of iterations to assemble a collection of weak classifiers — e.g., decision stumps — and at the end uses weighted voting to determine the classification of future instances. For example, AdaBoost [8] is such an algorithm.

By interpreting the node predicates as weak classifiers, one can apply the boosting framework also to DT learning. In top-down induction of DTs predicates are assigned to the leaves of the evolving tree. Usually the predicates test the value of a single attribute. They filter subsets of the original sample forward in the tree. A *splitting criterion* is used to rank candidate predicates. It favors predicates that reduce the impurity of the class distribution in the subsets that result from splitting the data. Since the filtered distributions gradually lose impurity, they usually become easier as we go down in the tree. The final

predictor is combined from the node predicates. Thus, viewing node predicate selection as a form of weak learning enables to explain the learning behavior of such successful DT learners as CART [3] and C4.5 [15].

Under the weak learning hypothesis, choosing a suitable predicate class and using an appropriate splitting criterion, the training error of the DT is driven down polynomially in its size [9]. This bound, which is close to optimal for any DT learning algorithm, is exponentially worse than that of AdaBoost [8]. However, empirically the error reduction with respect to the classifier size in AdaBoost is not essentially any better than that in C4.5 [4,7].

Let T be a binary DT constructed on the basis of sample S. We denote the set of leaves of T by $L(T)$. Throughout this paper we assume that leaves are labeled by the majority class of the examples reaching them. For a node $v \in T$ let $S_v \subseteq S$ be the set of examples reaching it. By $S_v^1 \subseteq S_v$ we denote the set of positive training examples reaching node v. The fraction of the sample reaching node v is denoted by $\hat{p}_v = |S_v|/|S|$ and the proportion of positive examples from those that reach v is denoted by $\hat{q}_v = |S_v^1|/|S_v|$.

A splitting criterion is a mapping $F : [0,1] \to [0,1]$. It assigns an impurity value to an observed class frequency distribution. Pure distributions, where $\hat{q} \in \{0,1\}$, have no impurity; i.e., for them $F(\hat{q}) = 0$. The most mixed distribution is the one in which $\hat{q} = 1/2$ and it has the maximal impurity, $F(1/2) = 1$. In addition to these properties, Kearns and Mansour [9] required a *permissible* splitting criterion F to be symmetric about $1/2$, $F(x) = F(1-x)$ for any $x \in [0,1]$, and to be concave. Commonly-used impurity functions fulfilling these properties include the binary entropy of C4.5 [15] and the Gini function of the CART algorithm [3]. Also the criterion that is used in the BP learning algorithm, $G(\hat{q}) = 2\sqrt{\hat{q}(1-\hat{q})}$ [9], is permissible.

Let the *index* of the tree T be

$$F(T) = \sum_{\ell \in L(T)} \hat{p}_\ell F(\hat{q}_\ell).$$

For any $h \in \mathcal{H}$ let T_h be the DT consisting of a single internal node labeled with h and two leaves corresponding to its values 0 and 1. Let $F(T_h, S)$ denote the index of T_h as measured with respect to S. Then, the change in the index, when node v is split using predicate h, is $\Delta(S_v, h) = F(\hat{q}_v) - F(T_h, S_v)$. Selecting a split to replace a node entails choosing the attribute that gives the most decrease to the value of the splitting criterion. Evaluation of attributes, on its part, entails determining the best binary partition for the domain of the attribute.

The *empirical error* of decision tree T is the weighted sum of the error of its leaves

$$\hat{\epsilon}(T) = \sum_{\ell \in L(T)} \hat{p}_\ell \min\{\, \hat{q}_\ell, 1 - \hat{q}_\ell \,\}.$$

It is bounded by the index $F(T)$, because $F(\hat{q}_\ell) \geq \min\{\, \hat{q}_\ell, 1 - \hat{q}_\ell \,\}$ by the properties required from a permissible splitting criterion.

Kearns and Mansour [9] showed that when the splitting criterion $G(\hat{q}) = 2\sqrt{\hat{q}(1-\hat{q})}$ is used in connection with DT learning, then assuming the β-weak

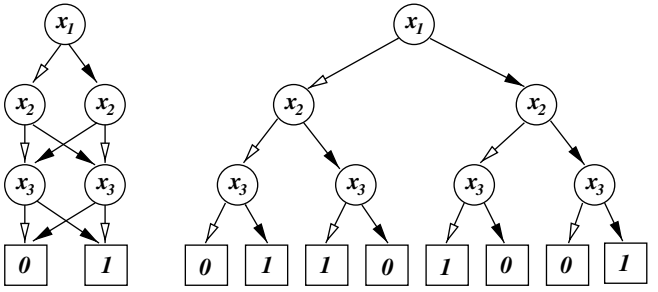

Fig. 1. Minimal BP and DT for computing the exclusive-or of three bits, $x_1 \oplus x_2 \oplus x_3$. Arcs with a black arrow head correspond to value 1 and those with a white head correspond to value 0.

learning hypothesis for $\mathcal{H}$, for any sample S_v from X there exists an $h \in \mathcal{H}$ such that $\Delta(S_v, h) \geq (\beta^2/16)G(\hat{q}_v)$. Based on this result Mansour and McAllester [12] defined that $\mathcal{H}$ and F satisfy the *γ-weak index reduction hypothesis* if for any sample S_v from X there exists an $h \in \mathcal{H}$ such that $\Delta(S_v, h) \geq \gamma F(\hat{q}_v)$.

By approximating — according to the weak learning hypothesis — the reduction of the index obtained by growing the size of the tree, one can bound the empirical error of the tree as a polynomial of its size:

$$\hat{\epsilon}(T) \leq F(T) \leq |T|^{-\gamma}.$$

The best known bound for γ with respect to β is obtained by using the splitting criterion G.

3 Branching Program Boosting

The polynomial reduction of a DT's error is inefficient from the analytical point of view [4]. Mansour and McAllester [13] showed that using the more compact DAG representation of BPs one can exploit the γ-weak index reduction hypothesis more efficiently than when DTs are used.

We are concerned with *leveled* BPs in the following. Let P_d be such a BP of depth d. The nodes of P_d form a two-dimensional lattice. There are $d+1$ levels $L_0, L_1, \ldots, L_d$ and each L_j has its individual width. All the arcs from the nodes of L_j go to the nodes of L_{j+1}. The first level consists of the root of the program and the nodes at level d are leaves. Leaves may already appear at earlier levels.

Internal nodes of a BP contain a predicate. Leaves are labeled by one of the two classes. Each instance x has a unique path from the root of the program to one of its leaves: At an internal node containing the predicate h decide for x whether $h(x) = 0$ or $h(x) = 1$ and follow the corresponding arc to the next level. Finally, after at most d internal nodes a leaf is reached. The label of the leaf determines the class prediction for x. Fig. 1 gives an example of a BP and a DT recognizing the same function.

3.1 The BP Learning Algorithm

Let us start by sketching the main points of the learning algorithm. We fill in the missing details and describe the analytical background subsequently.

The analysis of Mansour and McAllester [13] is based on a constant γ over all levels. No such constant is available in practice. In the following algorithm, instead, each level j has its own γ_j. Together with the current index, its value determines the width w_j of a grid of *potential nodes*. From those the ones that get connected to nodes of the previous level, will make up L_j. Potential nodes help to merge leaves of P_j whose class distribution is close enough to each other. We discuss the intuition behind potential nodes in more detail in the next subsection.

Algorithm LearnBP(S)
Initialize: Let L_0 consist of the root node of the BP. Associate the whole sample S with the root. Let P_0 be this single node program.
Main Loop: For j in $\{0, \ldots, d-1\}$ define L_{j+1} as follows.

1. Let $leaves_j$ consist of the pure nodes of L_j and set $L'_j = L_j \setminus leaves_j$. If L'_j is empty, then terminate.
2. **Predicate Selection:** For each $v \in L'_j$ choose the predicate h_v that most reduces the value of the splitting criterion F. In other words, for each $v \in L'_j$ check all attributes and determine their optimal binary splits with respect to S_v using F. Choose the best attribute and its optimal binary split to v. Technically, we choose v as the predicate h that maximizes $\Delta(S_v, h) = F(\hat{q}_v) - F(T_h, S_v)$.
3. **Program Expansion:** Let P'_{j+1} be the BP obtained from P_j by splitting the leaves in L'_j. Determine the average reduction of index, $\gamma_{j+1} > 0$, obtained by P'_{j+1}. That is,
$$\gamma_{j+1} = \frac{F(P_j) - F(P'_{j+1})}{F(P_j)}. \tag{1}$$
4. **Potential Node Grid Choosing:** Using γ_{j+1} and $F(P_j)$ determine the width w_{j+1} (the number of subintervals) of a one-dimensional grid of potential nodes each corresponding to an interval in $[0, 1]$. Determine the widths of the intervals.
5. **Leaf Merging:** Associate each leaf v of P'_{j+1} with the potential node that corresponds to the subinterval of $[0, 1]$ that contains the proportion, $\hat{q}_v$, of positive examples within v.
6. Those potential nodes that have at least one incoming arc make up L_{j+1}. Other potential nodes are discarded. The resulting program is P_{j+1}.

The parts that are not fully elaborated in the above algorithm are steps 3–6. They also comprise the theoretically most demanding points in this algorithm. The next subsection will review the analytical motivation behind these steps.

3.2 Analytical Background

A key difficulty in learning BPs is to have a sufficiently finely spaced grid so that when subsets from two or more nodes get merged, the loss in index reduction is

not too large. On the other hand, in order to restrict the growth of the hypothesis size, one must limit the number of potential nodes in the grid. Next, we go through those parts of the analysis of Mansour and McAllester [13] that motivate the choice of the grid of potential nodes used in LearnBP.

Consider, first, growing a leveled BP, where the nodes never get merged. This leveled DT growing yields the same polynomial error declination with respect to the tree size as other DT learning algorithms. Hence, to improve the error reduction with respect to the classifier size, nodes must be merged in the BP.

In the following we consider what happens to the index of a program, when P_j is expanded to P_{j+1} using the two-phase process, where all nodes $v \in L'_j$ are first split in two to produce P'_{j+1}. By Eq. 1, the index of the expanded program is $(1-\gamma_{j+1})F(P_j)$. After expansion the grid of potential nodes is chosen. It helps to merge together leaves of P'_{j+1} to produce the final nodes of P_{j+1}.

The potential nodes $n_1, \ldots, n_{w_{j+1}}$ correspond to a division of $[0,1]$ into consecutive intervals $I_i = [u_{i-1}, u_i)$, where $0 = u_0 < u_1 < \cdots < u_{w_{j+1}} = 1$. Let N_i denote those leaves v of P'_{j+1} for which $\hat{q}_v \in I_i$. The nodes belonging to N_i are merged together to potential node n_i. If $N_i = \emptyset$, the node n_i is discarded.

Let us see how the index of the BP changes due to merging of nodes. Before any such operations

$$F(P'_{j+1}) = \sum_{v \in L(P'_{j+1})} \hat{p}_v F(\hat{q}_v) = \sum_{i=1}^{w_{j+1}} \hat{p}_i F_i,$$

where $\hat{p}_i = \sum_{v \in N_i} \hat{p}_v$ and $F_i = \sum_{v \in N_i} (\hat{p}_v / \hat{p}_i) F(\hat{q}_v)$. After the merging, node n_i receives all the examples that arrived in P'_{j+1} to the leaves belonging to N_i. Hence, $\hat{p}_i$ is the proportion of examples received by n_i out of all examples. Let $\hat{q}_i$ denote, as usual, the fraction of positive examples out of those reaching n_i. Then, the index after the mergings is

$$F(P_{j+1}) = \sum_{i=1}^{w_{j+1}} \hat{p}_i F(\hat{q}_i).$$

The change in the index of the BP can, thus, be expressed as

$$F(P_{j+1}) - F(P'_{j+1}) = \sum_{i=1}^{w_{j+1}} \hat{p}_i (F(\hat{q}_i) - F_i).$$

Now, $\inf_{x \in I_i} F(x) \leq F_i \leq F(\hat{q}_i) \leq \sup_{x \in I_i} F(x)$, because $\hat{q}_i \in I_i$ and F is concave (see Fig. 2). Thus, the index increases, but the increase is small provided that F maps the points of each interval I_i close to each other.

In order not to lose the whole index reduction obtained by P'_{j+1}, the increase of index due to mergings has to be small with respect to γ_{j+1} and $F(P_j)$. This can be obtained by setting $u_m = 1/2$, where $m = w_{j+1}/2$, and $u_{m \pm k} = F_{\pm}^{-1}(1/(1 + \gamma_{j+1}/3)^k)$, for $0 < k < m$. Above, F_- is F restricted to $[0, 1/2]$ and F_+ to $[1/2, 1]$. Now, $\sup_{x \in I_i} F(x) = (1 + \gamma_{j+1}/3) \inf_{x \in I_i} F(x)$, whenever $1 < i < w_{j+1}$.

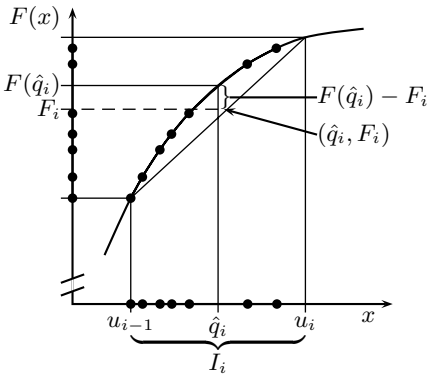

Fig. 2. Leaves with a fraction of positive examples in the interval $I_i = [u_{i-1}, u_i)$ make up N_i and are gathered to the potential node n_i. Combined these leaves have index $\hat{p}_i F_i$ and proportion $\hat{q}_i$ of positive examples within them. The point $(\hat{q}_i, F_i)$ is a convex combination of points $(\hat{q}_v, F(\hat{q}_v))$, $v \in N_i$ and, thus, falls within the region bounded by $F(x)$ in between end points u_{i-1} and u_i and the line connecting these end points. The index value assigned to n_i, $\hat{p}_i F(\hat{q}_i)$, is higher than $\hat{p}_i F_i$.

To keep the width of the grid under control n_1 and $n_{w_{j+1}}$ are handled as special cases. By setting

$$w_{j+1} = 2 + 2 \left\lceil \frac{\ln(6/(\gamma_{j+1} F(P_j)))}{\ln(1 + \gamma_{j+1}/3)} \right\rceil,$$

it holds that $1/(1 + \gamma_{j+1}/3)^{w_{j+1}/2} \leq (\gamma_{j+1}/6) F(P_j)$. Hence, the first and the last potential node represent (a subset of) those leaves $v \in L(P'_{j+1})$ for which $F(\hat{q}_v) \leq (\gamma_{j+1}/6) F(P_j)$. When the grid is chosen as presented above, then

$$\begin{aligned} F(P_{j+1}) - F(P'_{j+1}) &\leq \frac{\gamma_{j+1}}{3} \sum_{i=2}^{w_{j+1}-1} \hat{p}_i \inf_{x \in I_i} F(x) + (\hat{p}_1 + \hat{p}_{w_{j+1}}) \frac{\gamma_{j+1}}{6} F(P_j) \\ &\leq \left(\frac{\gamma_{j+1}}{3} + \frac{\gamma_{j+1}}{6} \right) F(P_j). \end{aligned}$$

On the other hand, by Eq. 1, $F(P'_{j+1}) = (1 - \gamma_{j+1}) F(P_j)$. Thus, $F(P_{j+1}) \leq (1 - \gamma_{j+1}/2) F(P_j)$. In other words, at least half of the index reduction obtained by P'_{j+1} is retained also after merging nodes together.

It remains to show that $|P_j|$ grows slowly enough. Let us first assume that γ_j equals a fixed $\gamma > 0$ and that $F(P_{j+1}) = (1 - \gamma/2) F(P_j)$ for each j. This is the least index reduction conforming to the above analysis. Now, the index of P_j is at most $(1 - \gamma/2)^j$. The increase in width, $w_{j+1} - w_j$, depends only on γ, not on j. Therefore, $|P_j|$ grows only quadratically in j. Together these observations imply that in this case the error reduction of a BP is exponential in the square root of its size. When it is also allowed that $F(P_{j+1}) < (1 - \gamma/2) F(P_j)$ —the analysis becomes more complicated, because $w_{j+1} - w_j$ may now vary. Mansour

and McAllester [13], nevertheless, show that the same error declination holds in general for a fixed γ. This analysis can be extended to show that for a program P produced by LearnBP, with γ being a uniform lower bound for γ_j, it holds that

$$\hat{\epsilon}(P) \leq F(P) \leq e^{-\Omega(\gamma\sqrt{|P|})}.$$

3.3 Related Work: Decision Graph Induction

Even though a DAG is an obvious generalization of a tree, learning decision graphs (DGs) is just an isolated strand in empirical machine learning. The DG induction algorithms were developed to solve, in particular, the subtree replication and data fragmentation problems, which are inherent to top-down induction of DTs. DGs resemble BPs, but are not exactly alike.

Oliver's [14] iterative hill-climbing algorithm uses a Minimum Message Length (MML) criterion to determine whether to split a leaf or to join a pair of leaves in the evolving DG. In experiments the algorithm attained the same accuracy level as MML-based DT learners and C4.5. DGs were observed to give particularly good results in learning disjunctive concepts.

Kohavi [10], originally, proposed constructing DGs in a bottom-up manner. Despite some empirical success, the algorithm was not able to cope with numerical attributes and lacked methods for dealing with irrelevant attributes. Subsequently Kohavi and Li [11] presented a method that post-processes a DT top-down into a DG. Special requirements were put to the initial DT; it was required to be *oblivious*, that is, test the same variable at each node in the same level of the tree. This approach was proposed as an alternative to the bottom-up pruning of DTs. The reported experiments exhibit classification accuracies comparable to those of C4.5, but with smaller classifier sizes.

Despite these few approaches, learning DAG-formed classifiers has not yet been thoroughly studied. Neither has sufficient analysis been presented for them.

4 Empirical Evaluation

We now test LearnBP in practice. As splitting criterion we use Kearns and Mansour's [9] function G. The analysis above only covers two class problems. Also, the algorithm LearnBP can only handle predicates. Therefore, we restrict our attention to domains with two classes. For nominal variables with more than two values, we search their optimal binary grouping with respect to G and use it. Numerical value ranges are binarized.

The classifier sizes are compared to see whether the BPs with zero empirical error are smaller than the corresponding DTs like they should be by the analysis (assuming the index reduction hypothesis). In order to evaluate the practical applicability of BPs, we compare the prediction accuracies of BPs and DTs.

It suffices to test LearnBP against C4.5, because the relation of C4.5 and AdaBoost is known [4,7]. In order to make the comparison fair to BPs, we contrast them against unpruned DTs built by C4.5. Moreover, we force C4.5 to

drive the empirical error to zero, if possible, like LearnBP does.[1] In order to make the comparison fair to DTs, we avoid repetitive counting of leaves, and measure classifier sizes by the number of internal nodes.

As test domains we use well-known binary data sets from the UCI Repository [2] and some synthetic domains. From the UCI Repository also relatively large domains were included. (Adult has approx. 32,500 and Connect some 67,500 examples; the latter was changed into a binary problem by uniting classes draw and lost). Some data sets were manipulated to rid the effects of missing attribute values. The synthetic data sets used are the majority function on 10 and 50 bits (denoted by MAJ10 and MAJ50, respectively), the multiplexor function with 3 address bits (MPLX3) [15], and the exclusive-or function on 6 and 8 bits (XOR6 and XOR8). For all synthetic data sets two different-sized random samples were generated. Our test strategy is 10-fold cross-validation repeated 10 times.

Table 1 gives the average prediction accuracies and sizes for BPs and DTs. It also records whether the observed differences are statistically significant as measured by the two-tailed Student's t-test at significance level 0.01 (99%).

4.1 Results on the UCI Data

On "real-world" data sets the accuracies of the learning algorithms are close to each other. In seven out of the total fifteen UCI domains statistically significant differences in the prediction accuracies are observed. On five of these cases the differences are in favor of LearnBP and on two cases of the unpruned DTs of C4.5. On some domains BPs may benefit from the fact that due to subset mergings the split decisions are often based on a larger population of training examples than the corresponding decisions in DTs. The obtained accuracy levels are lower than when using pruning, but not dramatically.

In hypothesis sizes the fact that pruning is disabled is, of course, observed. Some of the classifier average sizes are very large. On ten domains BPs are smaller than unpruned DTs and in the remaining five UCI domains unpruned DTs are smaller than the corresponding BPs. The algorithm that produces smaller hypotheses tends to have the better prediction accuracy. There are only three exceptions to this rule.

4.2 Results on Synthetic Data

On synthetic data, BPs are systematically more accurate on the majority problems. In addition to those there is only one further significant difference (in favor of C4.5). The results on the majority domains can be explained by the utility of merging leaves in the evolving program. Consider two leaves that are merged. Their fractions of positive examples have to be close to each other. Thus, even though different bits were tested en route to the leaves, the subsets associated

[1] It is not possible to set the parameters of Release 8 of C4.5 to produce perfect unpruned trees. Therefore, in this comparison we use Release 5 [15].

Table 1. Average classification accuracies and classifier sizes (with standard deviations) for LearnBP and C4.5 on the test domains. Statistically significant differences between the two learning algorithms are indicated by a double asterisk.

Data set		LearnBP	C4.5	BP Size	DT Size
Adult		$81.6 \pm 0.2^{**}$	81.3 ± 0.1	$3,959.6 \pm 50.0^{**}$	$4,870.1 \pm 13.5$
Breast W		94.6 ± 0.6	93.9 ± 0.6	$26.0 \pm 0.5^{**}$	29.6 ± 1.1
Chess		99.5 ± 0.1	$99.7 \pm 0.1^{**}$	46.1 ± 0.9	$44.3 \pm 0.3^{**}$
Connect		82.5 ± 0.1	$84.3 \pm 0.1^{**}$	$11,012.0 \pm 30.7$	$9,058.0 \pm 9.8^{**}$
Diabetes		70.3 ± 1.5	69.4 ± 1.0	$119.9 \pm 2.1^{**}$	131.8 ± 1.7
Euthyroid		$91.0 \pm 0.5^{**}$	90.1 ± 0.5	$135.6 \pm 0.7^{**}$	161.4 ± 2.3
German		$69.4 \pm 0.7^{**}$	67.6 ± 0.6	$157.7 \pm 2.0^{**}$	167.1 ± 1.6
Glass2		79.9 ± 1.6	80.8 ± 1.7	18.4 ± 0.3	$14.8 \pm 0.6^{**}$
Heart H		72.7 ± 2.3	72.8 ± 1.2	$43.9 \pm 0.9^{**}$	52.6 ± 0.8
Hepatitis		75.7 ± 2.0	78.0 ± 1.9	$13.6 \pm 0.3^{**}$	15.0 ± 0.5
Ionosphere		88.4 ± 1.1	89.8 ± 1.3	18.0 ± 0.3	$15.4 \pm 0.8^{**}$
Liver		65.4 ± 1.8	63.4 ± 1.5	$70.7 \pm 1.7^{**}$	77.6 ± 1.4
Sonar		$77.6 \pm 1.6^{**}$	74.5 ± 1.5	$15.5 \pm 0.3^{**}$	16.5 ± 0.5
Tic-tac-toe		$93.7 \pm 0.9^{**}$	86.8 ± 1.0	$81.6 \pm 2.9^{**}$	107.5 ± 2.1
Voting		86.6 ± 0.7	86.2 ± 0.6	41.0 ± 1.2	$35.7 \pm 0.3^{**}$
Maj10	200	$82.3 \pm 2.5^{**}$	78.7 ± 2.5	$40.2 \pm 1.2^{**}$	42.8 ± 0.8
	400	87.0 ± 1.9	84.8 ± 1.5	$67.2 \pm 1.9^{**}$	74.6 ± 1.4
Maj50	1000	$66.2 \pm 1.1^{**}$	61.4 ± 1.2	$159.7 \pm 1.7^{**}$	191.1 ± 2.2
	2000	$70.2 \pm 1.3^{**}$	63.8 ± 0.9	$296.3 \pm 2.1^{**}$	368.3 ± 2.6
Mplx3	400	82.8 ± 1.8	84.4 ± 1.3	101.2 ± 3.6	$96.5 \pm 2.6^{**}$
	800	95.2 ± 1.0	94.6 ± 0.7	109.2 ± 5.4	113.8 ± 5.0
Xor6	300	98.1 ± 0.8	98.4 ± 0.7	71.5 ± 2.1	$59.7 \pm 0.1^{**}$
	600	99.0 ± 0.8	$100.0 \pm 0.1^{**}$	75.6 ± 2.8	$63.0 \pm 0.1^{**}$
Xor8	500	81.2 ± 2.2	81.6 ± 0.8	257.7 ± 5.5	$207.6 \pm 0.5^{**}$
	1000	96.6 ± 0.6	96.7 ± 0.7	321.5 ± 4.8	$245.6 \pm 0.9^{**}$

with the leaves are likely to consist of examples with similar numbers of positive and negative bits. On the other hand, sometimes mergings are disadvantageous.

Over the ten synthetic domains the race for the smaller hypothesis is tied. This time there is no exception to the rule that the algorithm that produces the smaller hypothesis also has the better prediction accuracy.

In summary, the overall performance of BPs and unpruned DTs is very similar in both measured parameters. Nevertheless, BPs seem to perform slightly better than DTs.

5 Discussion

The empirical observations in the comparison between AdaBoost and C4.5 were explained with their different *advantage sequences* [4], i.e., how the weak learning parameter β changes from round to round. This depends on the algorithm and the data. The parameter characterizes the difficulty of the classification problems

posed to the weak learners. While C4.5 produces increasingly refined partitions of the sample, thus obtaining increasing advantage, AdaBoost concentrates on the examples that are falsely classified by earlier weak classifier, thus producing harder and harder filtered distributions and losing advantage.

Advantage sequences cannot directly be used in analyzing LearnBP, since it chooses many weak classifiers at a time. However, there is reason to believe that the γ-sequences of BPs are worse than those of DTs, because merging may make the task of separating the two classes more difficult. The advantage in LearnBP grows as the original sample is refined, but subset mergings lead to slow growth. Consider, e.g., the exclusive-or on eight bits. Assume that LearnBP has grown P'_8. Let u and v be two leaves that will be merged in P_8. If seven different bits have been tested on the paths from the root to u and v, splitting the leaves by testing the remaining bit produces pure leaves and, thus, gives a large reduction in index. On the other hand, when u and v are merged, the examples reaching them get mixed and it is impossible to separate the positive and negative examples by testing a single bit. We obtain a smaller value for γ_9, and also a larger BP since all information lost has to be gathered anew.

We may note that the width of the grid of potential nodes explodes with respect to the width of the corresponding program level. While the program levels typically are up to 30 nodes wide, the width of the grid of potential nodes may be close to 30,000 intervals. In the analysis of Mansour and McAllester [13] potential nodes determine the size of a program, which is much larger than the actual program size. Quite often the programs produced by LearnBP are actually DTs or almost such, i.e., they have very few node mergings. Sometimes the BP consists of several consecutive trees; i.e., there are nodes that gather all the remaining examples and the program construction starts anew.

We have compared BPs with unpruned DTs to better relate their sizes. However, DTs can be easily pruned to enhance classifier compactness. In this respect our empirical setting was not fair; with pruning C4.5 would easily beat LearnBP in classifier size. On the other hand, we wanted to obtain an understanding of the basic properties of a new interesting learning approach and test how the boosting analysis is reflected to practice.

LearnBP can be seen to work in the same explicit merging approach as some DG learning algorithms do, where all leaves of the evolving program are first expanded and then merged heuristically. Using the grid of potential nodes is an analytically motivated heuristic. Other heuristics could be used as well.

6 Conclusion

Constructing DAG-formed classifiers rather than tree-formed ones is a natural idea. However, since the former are harder to interpret and prune, they have not become more popular. Moreover, no evident advantage for using BPs has been known. The new analysis [13] gives a promise of a concrete advantage that could bear fruit in practice. However, our empirical evaluation indicates that this theoretical advantage does not directly materialize in experiments. The results,

though, are comparable to those obtained using unpruned DTs. Altogether, learning BPs (or DGs) is an interesting new idea which might deserve more empirical attention.

References

1. Bergadano, F., Bshouty, N.H., Tamon, C., Varricchio, S.: On learning branching programs and small depth circuits. In: Ben-David, S. (ed.): Computational Learning Theory: Proc. Third European Conference. Lecture Notes in Artificial Intelligence, Vol. 1208, Springer-Verlag, Berlin Heidelberg New York (1997) 150–161
2. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine (1998). `http://www.ics.uci.edu/~mlearn/MLRepository.html`
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and regression trees. Wadsworth, Pacific Grove, CA (1984)
4. Dietterich, T., Kearns, M., Mansour, Y.: Applying the weak learning framework to understand and improve C4.5. In: Saitta, L. (ed.): Proc. Thirteenth Internationl Conference on Machine Learning, Morgan Kaufmann, San Francisco (1996) 96–104
5. Ergün, F., Kumar, R.S., Rubinfeld, R.: On learning bounded-width branching programs. In: Proc. Eighth Annual Conference on Computational Learning Theory, ACM Press, New York (1995) 361–368
6. Freund, Y.: Boosting a weak learning algorithm by majority. Inf. Comput. **121** (1995) 256–285
7. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Saitta, L. (ed.): Proc. Thirteenth International Conference on Machine Learning. Morgan Kaufmann, San Francisco (1996) 148–156
8. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55** (1997) 119–139
9. Kearns, M., Mansour, Y.: On the boosting ability of top-down decision tree learning algorithms. J. Comput. Syst. Sci. **58** (1999) 109–128
10. Kohavi, R.: Bottom-up induction of oblivious read-once decision graphs. In: Bergadano, F., De Raedt, L. (eds.): Machine Learning: Proc. Seventh European Conference. Lecture Notes in Artificial Intelligence, Vol. 784, Springer-Verlag, Berlin Heidelberg New York (1994) 154–169
11. Kohavi, R., Li, C.-H.: Oblivious decision trees, graphs, and top-down pruning. In: Proc. Fourteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Francisco, CA (1995) 1071–1079
12. Mansour, Y., McAllester, D.: Boosting with multi-way branching in decision trees. In: Solla, S.A., Leen, T.K., Müller, K.-R. (eds.): Advances in Neural Information Processing 12. MIT Press, Cambridge, MA (2000) 300–306
13. Mansour, Y., McAllester, D.: Boosting using branching programs. In: Cesa-Bianchi, N., Goldman, S. (eds): Proc. Thirteenth Annual Conference on Computational Learning Theory. Morgan Kaufmann, San Francisco, CA (2000) 220–224
14. Oliver, J.J.: Decision graphs — an extension of decision trees. In: Proc. Fourth International Workshop on Artificial Intelligence and Statistics. Society for Artificial Intelligence and Statistics (1993) 343–350
15. Quinlan, J.R.: C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, CA (1993)
16. Schapire, R.E.: The strength of weak learnability. Mach. Learn. **5** (1990) 197–227