

Bagging Soft Decision Trees

Olcay Taner Yıldız¹(✉), Ozan İrsoy², and Ethem Alpaydın³

¹ Department of Computer Engineering, Işık University, Şile, 34398 İstanbul, Turkey
olcaytaner@isikun.edu.tr

² Department of Computer Science, Cornell University, Ithaca, NY 14853-7501, USA
oirsoy@cs.cornell.edu

³ Department of Computer Engineering, Boğaziçi University,
Bebek, 34730 İstanbul, Turkey
alpaydin@boun.edu.tr

Abstract. The decision tree is one of the earliest predictive models in machine learning. In the soft decision tree, based on the hierarchical mixture of experts model, internal binary nodes take soft decisions and choose both children with probabilities given by a sigmoid gating function. Hence for an input, all the paths to all the leaves are traversed and all those leaves contribute to the final decision but with different probabilities, as given by the gating values on the path. Tree induction is incremental and the tree grows when needed by replacing leaves with subtrees and the parameters of the newly-added nodes are learned using gradient-descent. We have previously shown that such soft trees generalize better than hard trees; here, we propose to bag such soft decision trees for higher accuracy. On 27 two-class classification data sets (ten of which are from the medical domain), and 26 regression data sets, we show that the bagged soft trees generalize better than single soft trees and bagged hard trees. This contribution falls in the scope of research track 2 listed in the editorial, namely, machine learning algorithms.

Keywords: Decision trees · Regression trees · Regularization · Bagging

1 Introduction

Trees are frequently used in computer science to decrease search complexity from linear to log time. In machine learning too, decision trees are frequently used and unlike other non-parametric methods such as the k -nearest neighbor where an input test pattern needs to be compared with all the training patterns, the decision tree uses a sequence of tests at internal decision nodes to quickly find the leaf corresponding to the region of interest. In classification, a leaf carries the class label and in regression, it carries a constant which is the numeric regression value [1, 2].

In the canonical *hard* binary decision tree, each decision node applies a test and depending on the outcome, one of the branches is taken. This process is repeated recursively starting from the root node until a leaf node is hit at which point the class label or the numeric regression value stored at the leaf constitutes

the output. In the hard decision tree, therefore, a single path from the root to one of the leaves is traversed and the output is given by the value stored in that particular leaf.

There are different decision tree architectures depending on the way decision is made at a node: The most typical is the *univariate tree* where the test uses a single input attribute and compares it against a threshold value [2]. In the *multivariate linear tree*, the test defines a linear discriminant in the d -dimensional space [3, 4]. In the *multivariate nonlinear tree*, the test can use a nonlinear discriminant—for example, a multilayer perceptron [5]. In the *omni-variate tree*, the test can use any of the above, chosen by a statistical model selection procedure [6].

So from a geometrical point of view, in the d -dimensional input space, each univariate split defines a boundary that is orthogonal to one of the axes; a multivariate linear split defines a hyperplane of arbitrary orientation, and a multivariate nonlinear split can define a nonlinear boundary.

In the hierarchical mixture of experts, Jordan and Jacobs [7] replace each expert with a complete system of mixture of experts in a recursive manner. Though it can also be viewed as an ensemble method, this architecture defines a soft decision tree where gating networks act as decision nodes. The soft gating function in a binary decision node chooses both children, but with probabilities (that sum up to 1). Hence, the node merges the decision of its left and right subtrees unlike a hard decision node that chooses one of them.

This implies that in a soft tree for a test input, we are traversing all the paths to all the leaves and all those leaves contribute to the final decision, but with different probabilities, as specified by the gating values on each path. In our proposed extension [8], the tree structure is not fixed but is trained incrementally one subtree at a time, where the parameters of the node and the leaf values are learned using gradient-descent.

Because the soft decision tree is multivariate and uses all input attributes in all nodes, it may have high variance on small data sets. As a variance reduction procedure, in this paper, we use bagging [9] which has been used successfully to combine hard decision trees in many applications; in our case of soft decision trees too, the use of bagging corresponds to averaging over soft trees trained with different data splits and initial parameter values in gradient-descent, and hence leads to a more robust estimate.

This paper is organized as follows: In Sect. 3, we review the soft decision tree model and its training algorithm. We discuss bagging soft decision trees in Sect. 4. We give our experimental results in Sect. 5 and conclude in Sect. 6.

Our work on extensions of decision trees falls in the scope of research track 2 of the editorial, namely machine learning algorithms.

2 Glossary and Key Terms

Bagging is an ensemble method where from a single training set, we draw multiple training sets using bootstrapping, with each of these sets we train a different model, and then combine their predictions, for example, using voting.

Bootstrapping is a resampling method where we randomly draw from a set *with* replacement.

Decision tree is a hierarchical model composed of decision nodes applied to the input and leaves that contain class labels.

Ensemble contains multiple trained models that are trained separately. In bagging, each of these models is trained on a slightly different data sets and hence may fail on slightly different cases, so accuracy can be increased by combining these multiple predictions.

Multivariate model uses all of the input attributes in making a decision whereas a univariate model uses only one of the input attributes.

Soft decision is different from a hard decision in that if there are m outcomes, in a hard decision we choose one of the m and ignore the remaining $m - 1$; in a soft decision, we choose all m but with different probabilities—these probabilities sum up to 1.

3 Soft Decision Trees

3.1 The Model

As opposed to the hard decision node which directs instances to one of its children depending on the outcome of the test at node m , $g_m(\mathbf{x})$, a soft decision node directs instances to all its children with probabilities calculated by a *gating function* $g_m(\mathbf{x})$ [7]. Without loss of generalization, let us consider a *binary node* where we have left and right children:

$$F_m(\mathbf{x}) = F_m^L(\mathbf{x})g_m(\mathbf{x}) + F_m^R(\mathbf{x})(1 - g_m(\mathbf{x})) \quad (1)$$

This is a recursive definition where $F_m^L(\mathbf{x})$ for example corresponds to the value returned by the subtree whose root is the left child of node m . Recursion ends when the subtree is just a leaf, in which case the value stored in the leaf is returned.

In the case of a hard tree, the *hard* decision node returns $g_m(\mathbf{x}) \in \{0, 1\}$, whereas in a soft tree, $g_m(\mathbf{x}) \in [0, 1]$, as given by the *sigmoid function*:

$$g_m(\mathbf{x}) = \frac{1}{1 + \exp[-(\mathbf{w}_m^T \mathbf{x} + \mathbf{w}_{m0})]} \quad (2)$$

Separating the regions of responsibility of the left and right children can be seen as two-class classification problem and from that perspective, the gating model implements a discriminative (logistic linear) model estimating the posterior probability of the left child: $P(L|\mathbf{x}) \equiv g_m(\mathbf{x})$ and $P(R|\mathbf{x}) \equiv 1 - g_m(\mathbf{x})$.

In a hard tree, because $g_m(\mathbf{x})$ returns 0 or 1, in Eq. (1), the node copies the value of its left or right child, whereas in a soft tree because $g_m(\mathbf{x})$ returns a value between 0 and 1, the node returns a weighted average of its two children.

This allows a smooth transition at the decision boundary, leads to a smoother fit and hence better generalization. Because the tree is traversed recursively, Eq. (1) is defined recursively and as a result, all the paths to all the leaves are traversed and at the root node, we get a weighted average of all the leaves where the weight of each leaf is given by the product of the gating values on the path to each leaf.

Incidentally, this model can easily be generalized to m -ary nodes where each node has $m > 2$ children, by replacing Eq. (1) with a convex combination of the values of the m children and the sigmoid of Eq. (2) by the softmax.

3.2 Training

Learning the soft decision tree is incremental and recursive, as with the hard decision tree [8]. The algorithm starts with one node and fits a leaf. Then, as long as there is improvement, it replaces the leaf by a subtree of a node and its two children leaves. This involves optimizing the gating parameters at the node and the values of its children leaves.

The error function is cross-entropy for classification and square loss for regression (In classification, the final output should be a probability and that is why for a two-class task, the final output at root is filtered through a sigmoid):

$$E = \begin{cases} \sum_t (r^{(t)} - y^{(t)})^2 & \text{Regression} \\ \sum_t r^{(t)} \log y^{(t)} + (1 - r^{(t)}) \log(1 - y^{(t)}) & \text{Classification} \end{cases} \quad (3)$$

At each growth step, node m , which was previously a leaf is replaced by a decision node and its two children leaves. The gating parameters (\mathbf{w}_m) of the decision node and the numeric leaf values of the children nodes (z_m^L, z_m^R) are set to small random values initially and are then updated using gradient-descent:

$$\begin{aligned} \Delta w_{mi} &= -\eta \frac{\partial E}{\partial w_{mi}} = \eta(r - y)[F_m^L(\mathbf{x}) - F_m^R(\mathbf{x})]\alpha_m g_m(\mathbf{x})(1 - g_m(\mathbf{x}))x_i \\ \Delta z_m^L &= -\eta \frac{\partial E}{\partial z_m^L} = \eta(r - y)\alpha_m g_m(\mathbf{x}) \\ \Delta z_m^R &= -\eta \frac{\partial E}{\partial z_m^R} = \eta(r - y)\alpha_m(1 - g_m(\mathbf{x})) \end{aligned}$$

where η is the learning factor,

$$\alpha_m = \prod_{n=m, p=n.\text{parent}}^{n \neq \text{root}} \delta_{n,p.\text{left}} g_p(\mathbf{x}) + \delta_{n,p.\text{right}} (1 - g_p(\mathbf{x}))$$

and $\delta_{i,j}$ is the Kronecker delta.

Note that only the three nodes of the last added subtree (current decision node and the leaf values of its children) are updated and all the other nodes are fixed. But since soft trees use a soft gating function, all the data points have an effect

on these parameters, whereas in a hard tree, only those data points that fall in the partition of the current node have an effect. Any input instance should pass through all the intermediate decision nodes until it reaches the added node and its leaves and the error should be discounted by all the gating values along the way to find the “back-propagated error” for that instance (denoted by α above). This value is then used to update the gating parameters and the leaf values.

In the hierarchical mixture of experts [7], the tree structure is fixed and the whole tree is learned using gradient-descent or expectation-maximization, whereas in our case, the tree is built incrementally, one subtree at a time. One recent work by Ruta and Li [10] is the fuzzy regression tree which is different from our work in several aspects. First, their splits are defined over kernel responses, hence, are univariate (one-dimensional), whereas our gating functions are multivariate and defined directly over the input space. Second, they apply an exhaustive search to learn the parameters (as in the hard univariate tree, which is possible because the splits use a single dimension) whereas we use gradient-descent.

For cases where the input dimensionality is high, we have previously proposed to use L_1 and L_2 -norm regularization where we add a model complexity term to the usual misfit error of Eq. (3) to get an augmented error [11]:

$$E' = E + \lambda \begin{cases} \sum_{i=0}^d |w_{mi}| & L_1\text{-norm} \\ \sum_{i=0}^d w_{mi}^2 & L_2\text{-norm} \end{cases}$$

and then we use the partial derivative of the augmented error in gradient-descent. λ trades off data misfit and model complexity. w_{mi} are the gating parameters of all nodes m in the tree for all attributes $i = 1, \dots, d$.

Especially as we go down the tree, we localize in parts of the input space where certain dimensions may not be necessary or when certain dimensions are highly correlated; at the same time, as we go down the tree, we have fewer data that reach there; so, regularization helps.

4 Bagging Soft Decision Trees

Bagging, short for bootstrap aggregating, was introduced by Breiman [9]. The idea is to generate a set of training data from an initial data by bootstrapping, that is, drawing with replacement, then train a predictor on each training data, and then combine their predictions. Because drawing is done with replacement, certain instances may be drawn more than once, and certain instances not at all.

Different training data will differ slightly and the resulting trained predictors can be seen as noisy estimates to the ideal discriminant; combining them removes noise and leads to a smooth estimator with low variance, and hence better generalization [12].

Decision trees are frequently used in bagging and here we use soft decision trees to see if we also have the advantage due to bagging when we combine soft trees. The soft decision tree has multivariate splits and risks overfitting when the input dimensionality is high and data set is small; hence averaging by combination will have a regularizing effect.

Additional to the randomness due to data, there is also the randomness due to the initialization of parameters before gradient-descent; averaging over trees will also average this randomness out. A single soft tree may overfit due to noisy instances in the data or a bad initialization, but we expect the majority of the models to converge to good trees and hence by combining their predictions, we get an improved overall estimate.

Figure 1 shows the pseudocode of the algorithm BaggedSoftTree that creates B soft trees for a data set \mathcal{X} containing N instances. For each, first we build a bootstrap sample \mathcal{D}_i of size N by drawing with replacement from the original \mathcal{X} (Line 2). Because the new set also contains N instances and drawing is done with replacement, the new set may contain certain instances multiple times and certain instances may not appear at all. Therefore, \mathcal{D}_i will be similar to \mathcal{X} but also slightly different. Then on each \mathcal{D}_i , we learn a soft tree \mathcal{T}_i (Line 3).

These trees will be similar but also slightly different due to the randomness in training, both due to their sampled data and also the initialization of parameters. As the last step, for any new given test data, we combine the predictions of these soft trees using a committee-based procedure, such as voting.

```

BaggedSoftTree( $\mathcal{X}$ ,  $B$ )
1  for  $i = 1$  to  $B$ 
2     $\mathcal{D}_i = \text{BootStrap}(\mathcal{X})$ 
3     $\mathcal{T}_i = \text{LearnSoftTree}(\mathcal{D}_i)$ 
4  end for
5  Return prediction by aggregating classifiers  $\mathcal{T}_i$ 

```

Fig. 1. The pseudocode of the algorithm that creates bagged soft trees consisting of B soft trees for a data set \mathcal{X} .

5 Experiments

5.1 Setup

We compare single and bagged soft decision trees with single and bagged hard decision trees on classification and regression data sets. Our methodology is as follows: We first separate one-third of the data set as the test set over which we evaluate the final performance. With the remaining two-thirds, we apply 5×2 -fold cross-validation, i.e. we randomly separate the data into two stratified parts five times, and for each time, we interchange the roles of the parts as training set and validation set, which gives a total of 10 folds for each data set.

In bagging, we train and combine 100 models. In combining the output of the 100 trees to get the overall output, in regression we use the median of the 100 predictions, and in classification we take a vote over the 100 class predictions.

We compare soft and hard trees, single and bagged, in terms of their error on the left-out test set. We give a table where for each data set separately we show the average and standard deviation error for all compared tree algorithms. To compare the overall performance on all data sets, we use Nemenyi’s test in terms of average ranks on all data sets and check for statistically significant difference [13]: On each data set, we rank the methods in terms of their average error so the first one gets the rank of 1, the second rank 2, and so on. Then we calculate the average rank of each method and Nemenyi’s test tells us how much difference between ranks is significant.

5.2 Classification Data Sets

We compare the soft tree (Soft) with C4.5 tree (Hard), linear discriminant tree (Ldt) (which is a multivariate hard tree) [4], and the bagged versions of Soft, Hard, and Ldt trees (Soft_B, Hard_B, Ldt_B) on 27 two-class classification data sets from the UCI repository [14].

Ten of these classification data sets are from the medical domain: *Breast* is a breast cancer database obtained from the University of Wisconsin Hospitals, Madison, *Haberman* contains cases from a study on the survival of patients who had undergone surgery for breast cancer, *Heart* is a database concerning heart disease diagnosis, *Parkinsons* is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson’s disease, *Pima* contains patients with diabetes who are females at least 21 years old of Pima Indian heritage, *Promoters* contains E. coli promoter gene sequences (DNA) with associated imperfect domain theory, *Spect* describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. *Acceptors* and *Donors* are splice site detection data sets and the trained models should distinguish ‘GT’ and ‘AG’ sites occurring in the DNA sequence that function as splice sites and those that do not [15]. *Polyadenylation* datasets contains polyadenylation signals in human sequences [16].

Table 1 shows the average and standard deviation of test errors of Hard, Ldt, Soft, Hard_B, Ldt_B, and Soft_B on the separate data sets, where we see that bagged soft tree most of the time has the smallest error. Figure 2 shows the result of post-hoc Nemenyi’s test applied on the average ranks of these algorithms in terms of their error on all data sets.

We see that the bagged soft tree has the lowest average rank (slightly above 1) and is significantly better than all other tree variants. The bagged versions of Ldt and Hard are only as good as a single soft tree. The single soft tree is significantly more accurate than single Ldt or hard tree. Ldt is also multivariate but uses hard splits; the fact that the soft tree (bagged or single) is more accurate than Ldt shows that it is the softness of the split that leads to higher accuracy rather than whether the split is uni or multivariate.

Table 1. On two-class classification data sets, the average and standard deviation of test errors of Hard, Ldt, Soft, and their bagged versions, $Hard_B$, Ldt_B , and $Soft_B$.

Dataset	Hard	Ldt	Soft	$Hard_B$	Ldt_B	$Soft_B$
acceptors	16.1 ± 2.0	9.6 ± 0.8	8.7 ± 0.7	18.2 ± 0.1	8.7 ± 0.5	8.1 ± 0.5
artificial	1.1 ± 1.8	1.5 ± 1.9	1.1 ± 1.8	1.1 ± 1.8	0.7 ± 1.6	0.7 ± 1.6
breast	6.7 ± 1.1	4.9 ± 0.6	3.5 ± 0.7	4.7 ± 0.8	4.7 ± 0.7	3.1 ± 0.4
bupa	38.6 ± 4.1	39.1 ± 3.4	39.7 ± 4.2	35.4 ± 3.6	38.2 ± 2.3	36.5 ± 2.7
donors	7.7 ± 0.4	5.4 ± 0.3	5.7 ± 0.4	7.2 ± 0.4	5.4 ± 0.2	5.3 ± 0.3
german	29.9 ± 0.0	25.8 ± 2.0	24.0 ± 3.0	29.9 ± 0.0	27.0 ± 2.8	23.2 ± 0.8
haberman	26.6 ± 0.3	27.2 ± 1.5	25.9 ± 1.8	26.5 ± 0.0	26.5 ± 0.0	24.7 ± 1.6
heart	28.3 ± 4.7	18.4 ± 2.3	19.7 ± 3.4	24.7 ± 6.0	18.4 ± 2.2	15.7 ± 1.3
hepatitis	22.1 ± 4.4	20.4 ± 2.9	20.2 ± 2.4	20.8 ± 1.2	20.2 ± 1.6	18.7 ± 2.4
ironosphere	13.1 ± 1.9	12.3 ± 2.2	11.5 ± 2.0	9.4 ± 3.2	12.4 ± 1.9	11.6 ± 1.3
krvskp	1.2 ± 0.4	4.5 ± 0.7	1.8 ± 0.6	1.2 ± 0.5	4.7 ± 0.7	1.8 ± 0.2
magic	17.5 ± 0.6	16.9 ± 0.1	14.7 ± 0.5	16.4 ± 0.3	16.7 ± 0.2	13.9 ± 0.1
monks	12.8 ± 7.8	23.8 ± 8.2	0.0 ± 0.0	11.9 ± 4.6	24.0 ± 2.0	0.0 ± 0.0
mushroom	0.0 ± 0.1	1.8 ± 0.5	0.1 ± 0.0	0.1 ± 0.1	0.9 ± 0.2	0.1 ± 0.1
musk2	5.5 ± 0.6	6.4 ± 0.3	4.3 ± 0.7	5.3 ± 0.1	6.3 ± 0.2	3.8 ± 0.3
parkinsons	13.8 ± 2.3	13.5 ± 2.5	14.3 ± 2.7	14.0 ± 3.1	14.8 ± 4.1	10.9 ± 0.9
pima	27.9 ± 3.4	23.1 ± 1.4	24.9 ± 2.0	24.2 ± 1.2	22.6 ± 1.0	23.6 ± 1.0
polyaden	30.5 ± 1.3	22.6 ± 0.6	22.9 ± 0.5	29.2 ± 0.5	22.4 ± 0.4	22.1 ± 0.3
promoters	26.1 ± 9.9	34.4 ± 9.4	15.3 ± 6.7	14.7 ± 9.7	31.7 ± 5.9	10.8 ± 4.0
ringnorm	12.2 ± 1.1	22.8 ± 0.3	9.9 ± 1.7	7.2 ± 0.7	22.7 ± 0.3	5.1 ± 0.3
satellite47	15.4 ± 1.5	16.7 ± 1.4	12.4 ± 1.4	12.2 ± 0.5	16.7 ± 0.6	11.5 ± 0.6
spambase	9.9 ± 0.7	10.1 ± 0.7	7.5 ± 0.5	8.1 ± 0.4	9.8 ± 0.4	7.2 ± 0.3
spect	19.1 ± 2.8	20.1 ± 2.4	19.6 ± 2.4	20.4 ± 2.1	21.1 ± 0.0	17.4 ± 3.3
tictactoe	23.8 ± 2.2	31.9 ± 2.4	1.8 ± 0.3	22.1 ± 2.5	29.4 ± 1.0	1.6 ± 0.0
titanic	21.8 ± 0.5	22.4 ± 0.4	21.5 ± 0.2	22.1 ± 0.0	22.7 ± 0.2	21.5 ± 0.2
twonorm	17.0 ± 0.7	2.0 ± 0.1	2.1 ± 0.2	4.8 ± 0.7	2.0 ± 0.1	2.0 ± 0.1
vote	5.2 ± 0.7	6.7 ± 2.6	5.1 ± 0.9	4.9 ± 0.2	6.4 ± 1.1	4.6 ± 0.6

5.3 Regression Data Sets

We also compare soft regression trees (Soft) with the univariate regression tree (Hard) and their bagged versions, $Soft_B$ and $Hard_B$, on 26 regression data sets [17].

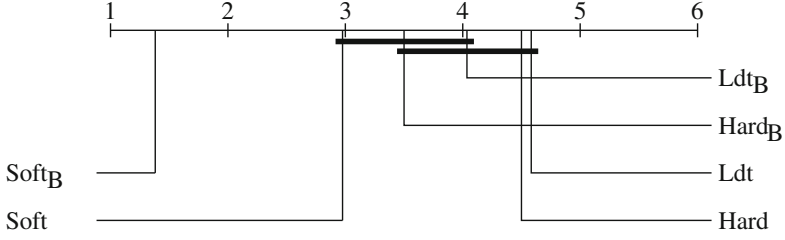


Fig. 2. On two-class classification data sets, the result of Nemenyi’s test applied on the ranks of Hard, Ldt, Soft, $Hard_B$, Ldt_B , and $Soft_B$ in terms of error. Indicated points are the average ranks and a thick underline implies no significant difference.

Table 2. On the regression data sets, the average and standard deviation of errors of Hard and Soft trees and their bagged versions, $Hard_B$, and $Soft_B$.

Dataset	Hard	Soft	$Hard_B$	$Soft_B$
abalone	0.53 ± 0.01	0.41 ± 0.01	0.50 ± 0.02	0.41 ± 0.01
add10	0.24 ± 0.01	0.08 ± 0.01	0.19 ± 0.00	0.05 ± 0.00
bank32fh	0.50 ± 0.01	0.40 ± 0.01	0.46 ± 0.01	0.40 ± 0.01
bank32fm	0.12 ± 0.00	0.04 ± 0.00	0.10 ± 0.00	0.04 ± 0.00
bank32nh	0.59 ± 0.01	0.45 ± 0.01	0.56 ± 0.01	0.43 ± 0.00
bank32nm	0.41 ± 0.02	0.20 ± 0.00	0.34 ± 0.01	0.19 ± 0.00
bank8fh	0.30 ± 0.01	0.26 ± 0.01	0.28 ± 0.01	0.26 ± 0.01
bank8fm	0.08 ± 0.00	0.04 ± 0.00	0.08 ± 0.01	0.04 ± 0.00
bank8nh	0.69 ± 0.02	0.56 ± 0.02	0.65 ± 0.02	0.56 ± 0.02
bank8nm	0.37 ± 0.03	0.12 ± 0.01	0.35 ± 0.02	0.10 ± 0.01
boston	0.34 ± 0.09	0.23 ± 0.03	0.27 ± 0.05	0.24 ± 0.02
comp	0.03 ± 0.00	0.02 ± 0.00	0.08 ± 0.00	0.02 ± 0.00
concrete	0.93 ± 0.05	0.23 ± 0.02	0.67 ± 0.03	0.22 ± 0.01
kin32fh	0.73 ± 0.03	0.32 ± 0.01	0.64 ± 0.01	0.32 ± 0.01
kin32fm	0.61 ± 0.02	0.08 ± 0.00	0.51 ± 0.01	0.07 ± 0.00
kin32nh	0.94 ± 0.02	0.75 ± 0.03	0.92 ± 0.03	0.75 ± 0.02
kin32nm	0.90 ± 0.01	0.62 ± 0.03	0.87 ± 0.01	0.60 ± 0.01
kin8fh	0.54 ± 0.02	0.26 ± 0.00	0.42 ± 0.02	0.26 ± 0.00
kin8fm	0.32 ± 0.01	0.03 ± 0.00	0.22 ± 0.01	0.03 ± 0.00
puma8fh	0.42 ± 0.01	0.38 ± 0.01	0.39 ± 0.01	0.38 ± 0.01
puma8nh	0.40 ± 0.02	0.36 ± 0.01	0.37 ± 0.01	0.35 ± 0.01
puma8fm	0.07 ± 0.00	0.05 ± 0.00	0.08 ± 0.00	0.05 ± 0.00
puma8nm	0.06 ± 0.01	0.05 ± 0.00	0.08 ± 0.00	0.04 ± 0.00
puma32fh	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01	0.59 ± 0.01
puma32fm	0.04 ± 0.00	0.07 ± 0.01	0.08 ± 0.01	0.06 ± 0.00
puma32nh	0.39 ± 0.01	0.43 ± 0.02	0.36 ± 0.01	0.41 ± 0.01

Table 2 shows the average and standard deviation of errors of Hard, Soft, Hard_B , and Soft_B on each data set separately. Figure 3 shows the result of Nemenyi’s test applied on the ranks of the error rates of these algorithms.

We see again that the bagged soft tree has the lowest rank; the bagged soft tree is significantly more accurate than the single soft tree and they are significantly better than both the hard tree and bagged hard tree. Bagging the hard tree leads to some improvement in terms of average rank but the difference is not significant here. Note that this does not mean bagging hard trees is useless, it is only with respect to the others that the difference between them seems insignificant—single and bagged hard trees rank mostly in 3rd and 4th ranks.

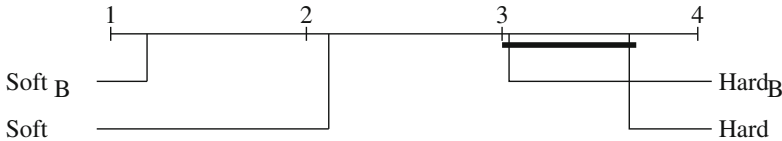


Fig. 3. On the regression data sets, the result of Nemenyi’s test applied on the ranks of errors of Hard and Soft trees and their bagged versions, Hard_B , and Soft_B .

6 Conclusions and Future Outlook

The soft tree has several advantages: First, it provides a continuous fit whereas the hard tree has a discontinuous response at the leaf boundaries. This enables the soft tree to have smoother fits and hence lower bias around the split boundaries. Second, the linear gating function enables the soft tree to make oblique splits in contrast to the axis-orthogonal splits made by the univariate hard tree.

In our previous experiments [8], we see that these two properties improve accuracy and also reduce the number of nodes required to solve a regression or a classification problem. Soft trees seem especially suited to regression problems where the gating function allows a smooth interpolation between the children of a node.

Here, we build on top of the soft decision tree model and show how its accuracy can be further improved by bagging. We see that on both classification and regression problems, we get significant improvement in terms of accuracy by bagging soft decision trees.

Bagging averages over both the randomness in sampling of data and the randomness in the initialization of parameters (before gradient-descent) and this leads to a smoother fit and better generalization.

Bagging is only one way to build an ensemble. We previously worked on methods for training and pruning an ensemble [12] and combining them to construct uncorrelated metaclassifiers [18] and these ensemble construction approaches can also use soft decision trees as the base learner.

Another possible future direction is in combining multiple sources: In some applications, there are multiple views or representations associated with each instance that complement each other and one possible future work is to train different soft trees with different views and then combine their predictions.

Even with a single representation, different soft trees can use different randomly chosen subsets of the features [19] and we can have soft random decision forests—these are possible future research directions.

Acknowledgments. This work is partially supported by Boğaziçi University Research Funds with Grant Number 14A01P4.

References

1. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: *Classification and Regression Trees*. John Wiley and Sons, New York (1984)
2. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo (1993)
3. Murthy, S.K., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. *J. Artif. Intell. Res.* **2**, 1–32 (1994)
4. Yıldız, O.T., Alpaydın, E.: Linear discriminant trees. *Int. J. Pattern Recogn. Artif. Intell.* **19**(3), 323–353 (2005)
5. Guo, H., Gelfand, S.B.: Classification trees with neural network feature extraction. *IEEE Trans. Neural Netw.* **3**, 923–933 (1992)
6. Yıldız, O.T., Alpaydın, E.: Omnivariate decision trees. *IEEE Trans. Neural Netw.* **12**(6), 1539–1546 (2001)
7. Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the EM algorithm. *Neural Comput.* **6**, 181–214 (1994)
8. İrsoy, O., Yıldız, O.T., Alpaydın, E.: Soft decision trees. In: *Proceedings of the International Conference on Pattern Recognition*, Tsukuba, Japan, pp. 1819–1822 (2012)
9. Breiman, L.: Bagging predictors. *Mach. Learn.* **26**, 123–140 (1996)
10. Ruta, A., Li, Y.: Learning pairwise image similarities for multi-classification using kernel regression trees. *Pattern Recogn.* **45**, 1396–1408 (2011)
11. Yıldız, O.T., Alpaydın, E.: Regularizing soft decision trees. In: *Proceedings of the International Conference on Computer and Information Sciences*, Paris, France (2013)
12. Ulaş, A., Semerci, M., Yıldız, O.T., Alpaydın, E.: Incremental construction of classifier and discriminant ensembles. *Inf. Sci.* **179**, 1298–1318 (2009)
13. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**, 1–30 (2006)
14. Blake, C., Merz, C.: *UCI repository of machine learning databases* (2000)
15. Kulp, D., Haussler, D., Reese, M.G., Eeckman, F.H.: A generalized hidden markov model for the recognition of human genes in dna. In: *International Conference on Intelligent Systems for Molecular Biology* (1996)
16. Liu, L., Han, H., Li, J., Wong, L.: An in-silico method for prediction of polyadenylation signals in human sequences. In: *International Conference on Genome Informatics* (2003)

17. Rasmussen, C.E., Neal, R.M., Hinton, G., van Camp, D., Revow, M., Ghahramani, Z., Kustra, R., Tibshirani, R.: Delve data for evaluating learning in valid experiments (1996)
18. Ulaş, A., Yıldız, O.T., Alpaydın, E.: Eigenclassifiers for combining correlated classifiers. *Inf. Sci.* **187**, 109–120 (2012)
19. Ho, T.K.: The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.* **20**, 832–844 (1998)

Machine Learning for Health Informatics

State-of-the-Art and Future Challenges

Holzinger, A. (Ed.)

2016, XXII, 481 p. 98 illus., Softcover

ISBN: 978-3-319-50477-3