21st International Conference on Pattern Recognition (ICPR 2012)
November 11-15, 2012. Tsukuba, Japan

Soft Decision Trees

Ozan İrsoy[1], Olcay Taner Yıldız[2], Ethem Alpaydın[1]
[1]*Department of Computer Engineering, Boğaziçi University, 34342, İstanbul Turkey*
[2]*Department of Computer Engineering, Işık University, 34980, İstanbul Turkey*

Abstract

We discuss a novel decision tree architecture with soft decisions at the internal nodes where we choose both children with probabilities given by a sigmoid gating function. Our algorithm is incremental where new nodes are added when needed and parameters are learned using gradient-descent. We visualize the soft tree fit on a toy data set and then compare it with the canonical, hard decision tree over ten regression and classification data sets. Our proposed model has significantly higher accuracy using fewer nodes.

1. Introduction

A decision tree is an hierarchical structure composed of internal decision nodes and terminal leaves [3, 7]. For classification, the leaves carry the label of one of K classes, whereas for regression, the leaves carry a constant which is the numeric regression value. The input vector is composed of d attributes, $\boldsymbol{x} = [x_1, \ldots, x_d]^T$.

In the canonical *hard* binary decision tree, each decision node m applies a test $g_m(\boldsymbol{x})$ and chooses one of the children accordingly. Let $F_m(\boldsymbol{x})$ be the output generated by the subtree whose root is m and in a binary tree, let $F_m^L(\boldsymbol{x})$ and $F_m^R(\boldsymbol{x})$ denote respectively its left and right children and $g_m(\boldsymbol{x})$ hence has two outcomes:

$$F_m(\boldsymbol{x}) = \begin{cases} F_m^L(\boldsymbol{x}) & \text{if } g_m(\boldsymbol{x}) > 0 \text{ /* true */} \\ F_m^R(\boldsymbol{x}) & \text{otherwise /* false */} \end{cases} \quad (1)$$

Given an input, starting from the root node, one applies the test at each internal node and the input is forwarded to one of the two branches depending on the outcome. This process is repeated recursively until a leaf node is hit at which point the class label or the numeric regression value of the leaf constitutes the output. In the hard decision tree, therefore, a single path from the root to one of the leaves is traversed.

There are different decision tree architectures depending on the model they assume for $g_m(\boldsymbol{x})$: The most typical is the *univariate tree* where $g_m(\boldsymbol{x})$ uses a single input attribute and compares it against a threshold value [7]. In the *multivariate linear tree*, $g_m(\boldsymbol{x})$ defines a linear discriminant in the d-dimensional space [6, 10]. In the *multivariate nonlinear tree*, $g_m(\boldsymbol{x})$ can be a nonlinear discriminant, e.g., a multilayer perceptron [4]. In the *omnivariate tree*, $g_m(\boldsymbol{x})$ can be any of the above, chosen by a statistical model selection procedure [9].

In this paper, we discuss the soft decision tree where unlike the hard internal node, all children are selected but all with a certain probability. That is, we follow all the paths to all the leaves and all the leaves contribute to the final decision but with different probabilities. We discuss how such a model can be trained incrementally both for classification and regression, and also compare it experimentally with the hard version and show that it leads to trees that are both accurate and simpler in terms of the number of nodes.

This paper is organized as follows: In Section 2, we explain our proposed soft tree construction algorithm for classification and regression. We give our experimental results in Section 3 and conclude in Section 4.

2. Training a Soft Decision Tree

As opposed to the hard decision node which redirects instances to one of its children depending on the outcome of $g_m(\boldsymbol{x})$, a soft decision node redirects instances to all its children with probabilities calculated by a *gating function* $g_m(\boldsymbol{x})$. Let us consider a *binary node* where we have left and right children:

$$F_m(\boldsymbol{x}) = F_m^L(\boldsymbol{x}) g_m(\boldsymbol{x}) + F_m^R(\boldsymbol{x})(1 - g_m(\boldsymbol{x})) \quad (2)$$

and to choose among the two outcomes, we take $g_m(\boldsymbol{x}) \in [0, 1]$ as the *sigmoid function*:

$$g_m(\boldsymbol{x}) = \frac{1}{1 + \exp[-(\boldsymbol{w}_m^T \boldsymbol{x} + \boldsymbol{w}_{m0})]} \quad (3)$$

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: $P(L|\boldsymbol{x}) \equiv g_m(\boldsymbol{x})$ and $P(R|\boldsymbol{x}) \equiv 1 - g_m(\boldsymbol{x})$.

```
1  function F_m(x)
2     if m is leaf node
3        y = z_m /* leaf value at m */
4     else
5        g_m(x) = 1/(1 + exp(−(w_m^T x + w_m0)))
6        y = F_m^L(x) g_m(x) + F_m^R(x)(1 − g_m(x))
7     return y
```

Figure 1. Pseudocode for calculating the response of the subtree rooted at node m

Figure 1 shows the pseudocode for calculating the response of the subtree rooted at node m. If m is a leaf node (Line 2), the response is constant (Line 3). If m is an internal node (Line 4), the response of an internal node is the weighted sum of responses of its left and right subtrees (Line 6) weighted by the sigmoid gating (Line 5). Note that the function calls itself recursively both for its left and right children and returns the weighted average of the values they return. This implies that all the paths to all the leaves are traversed and all the leaves contribute to the overall output weighted by the product of the gating values on their paths.

Learning the tree is incremental and recursive, as with the hard decision tree. The algorithm starts with one node and fits a constant model. Then, as long as there is improvement, it replaces the leaf by a subtree. This involves optimizing the gating parameters and the values of its children leaf nodes by gradient-descent over an error function.

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 of the tree is filtered through a sigmoid at the root):

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

Figure 2 shows the pseudocode for finding the best split for node m using training set $\mathcal{X}$ and validation set $\mathcal{V}$. The gating parameters ($\boldsymbol{w}_m$) and the numeric leaf values of the children nodes (z_m^L, z_m^R) are set to small random values initially (Lines 3-6) and are then updated using gradient-descent. Note that only these last three nodes (current decision node and its leaf children) are updated and all the other nodes are fixed. But keep also in mind that since soft trees use a gating function, all the data points have an effect on these parameters (Line 8), whereas in a hard tree, only those data points that fall in the partition of the current node have an effect. Any instance should pass through all the intermediate decision nodes until it reaches the added node and its leaves and the error (Line 9) should be discounted by all the gating values along the way to find the "back-propagated error" for that instance, denoted by δ (Lines 10-17). This value is then used to update the gating parameters (Line 20) and the leaf values (Lines 21 and 22), where η is the step size of gradient-descent.

```
1  function LearnSoftTree(m, X, V)
2     E_before = ErrorOfTree(V)
3     for j = 0, ..., d
4        w_mj = rand(−0.01, 0.01)
5     z_m^L = rand(−0.01, 0.01)
6     z_m^R = rand(−0.01, 0.01)
7     repeat
8        for all (x, r) ∈ X
9           δ = F_root(x) − r
10          t = m
11          while t! = root do
12             p = t.parent
13             if t == p.left
14                δ = δ g_p(x)
15             else
16                δ = δ(1 − g_p(x))
17             t = p
18          β = δ(F_m^L(x) − F_m^R(x))
19          for j = 0, ..., d
20             w_mj = w_mj − η β v_m(x)(1 − v_m(x)) x_j
21          z_m^L = z_m^L − η δ v_m(x)
22          z_m^R = z_m^R − η δ(1 − v_m(x))
23    until convergence
24    E_after = ErrorOfTree(V)
25    if E_after < E_before
26       LearnSoftTree(m.left, X, V)
27       LearnSoftTree(m.right, X, V)
```

Figure 2. Pseudocode for finding the best split for node m using training set $\mathcal{X}$ and validation set $\mathcal{V}$.

As the tree grows deeper, the updates become smaller due to the multiplication of gating functions in the update rules. Hence to avoid very small updates, we adapt step size, starting from a fixed value and exponentially decreasing it by half at each time. This reduces the dependency of the algorithm on the step size and increases stability. The best step size is chosen for each split over the training set, and the decision to split that node is made by checking whether there is improve-

ment over the validation set (Lines 24-25). If so, the tree is updated with the new parameters, and tree construction continues recursively for the left and right children (Lines 26-27); otherwise, the node stays as a leaf.

Our initial experiments indicate that gradient descent with random initial point performs relatively poorly. Instead to initialize, we first find the best split as if it were a hard decision node and then we assign $w_{mj} = -1$ where j is the split dimension, take $w_{m0} = 1$ and we assign $w_{mk} = 0$ for all $k \neq j$. This initializes the oblique split by the axis-aligned split of the hard node except that it has a finite slope. The values of the children leaves are simply copied from their hard counterparts. Online gradient descent starts from this point on.

Similar models have been proposed in the past. In the hierarchical mixture of experts, Jordan and Jacobs [5] replace each expert with a complete system of mixture of experts in a recursive manner. This architecture is a soft decision tree where gating networks are the decision nodes. The difference is that in the former, the tree structure is fixed and the whole tree is learned using gradient-descent or expected maximization, whereas in our case, the tree is built incrementally.

One recent work by Ruta and Li [8] is the fuzzy regression tree which is different from our work in several aspects. First, their splits are defined over kernel responses, hence, are inherently one-dimensional, whereas our gating functions are defined directly over the input space. Second, they apply an exhaustive search to learn the parameters (as in the hard univariate tree) whereas we use gradient descent.

3. Experiments

The difference between a hard and soft fit is best seen in Figure 3 which shows a toy data sampled from a sinusoidal with added Gaussian noise and soft and hard tree fits. Soft tree achieves comparable accuracy using seven nodes (four leaves) compared to the hard tree with 37 nodes (19 leaves). Both the tree structure and the fits are shown; note that the sigmoid gating allows a smooth interpolation between neighboring leaves and lead to a smoother fit; this automatic interpolation leads to better generalization and also makes intermediary leaves redundant thereby simplifying the tree.

To compare the generalization error and model complexity of soft and hard trees, we use ten regression (*ABAlone, ADD10, BOSton, CALifornia, COMp, CONcrete, puma8FH, puma8FM, puma8NH, puma8NM*) and ten two-class classification data sets (*BREast, GERman, MAGic, MUSk2, PIMa, POLyadenylation, RINgnorm, SATellite47, SPAmbase, TWOnorm*) from the UCI repository [2]. We also compare soft trees with

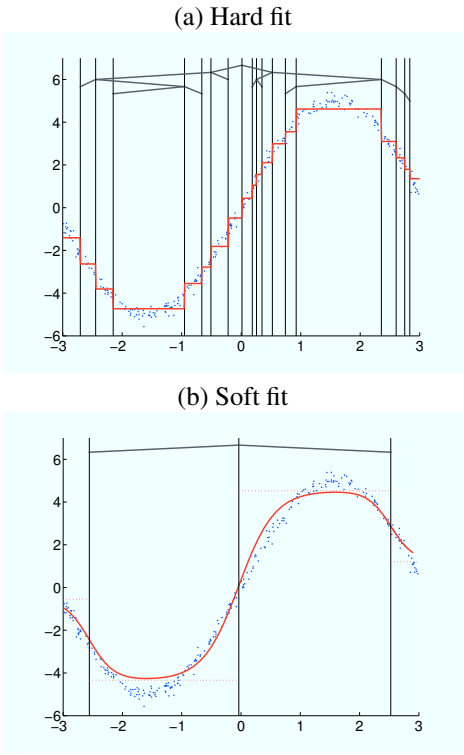

Figure 3. Hard and soft tree fits

a multivariate linear tree algorithm, namely linear discriminant trees (LDT) [10].

We first separate one third of the data set as the test set over which we evaluate and report the final performance. With the remaining two thirds, we apply 5×2-fold cross validation, which gives a total of ten folds for each data set. We use the 5×2 paired F-test [1] to compare the errors and Wilcoxon's rank sum test to compare tree complexities, as measured by the number of nodes; the best result is shown in boldface in the figures.

Table 1 shows the average error and tree size (number of nodes) of soft and hard regression trees. On all data sets, the soft tree has significantly fewer nodes and except four ties, soft tree has significantly smaller mean square error. Though the nodes in a soft tree are more multivariate and all paths are visited, the drastic reduction in tree size makes up for the added complexity.

Table 2 shows the comparison on classification data sets. In terms of accuracy, the soft tree has three wins (*twonorm*, *german*, *polyadenylation*), the hard tree has two wins (*spambase*, *magic*) and there are five ties—no overall significant difference. LDT is better than the soft tree on three datasets (*spambase*, *magic*, *pima*) and the hard tree on three datasets (*twonorm*, *polyadenylation*, *pima*). On the other hand, both hard and soft tree have a single common win over LDT, namely *ringnorm*.

Table 1. On the regression data sets, the average error and the number of nodes of soft and hard decision trees.

	Mean Square Error		Tree Size	
	Soft	Hard	Soft	Hard
ABA	**0.439**	0.557	**7**	32
ADD	**0.094**	0.267	**15**	202
BOS	0.271	0.344	**11**	18
CAL	**0.312**	0.326	**3**	201
COM	0.037	0.046	**5**	30
CON	0.264	0.286	**13**	69
8FH	**0.383**	0.418	**3**	40
8FM	**0.057**	0.074	**3**	92
8NH	0.388	0.416	**9**	52
8NM	**0.054**	0.084	**13**	144

Table 2. On the classification data sets, the average error and the number of nodes of soft, hard, and linear discriminant trees (LDT).

	Accuracy			Tree Size		
	Soft	Hard	LDT	Soft	Hard	LDT
BRE	95.34	93.80	95.09	17	47	**4**
GER	75.74	69.07	74.16	16	142	**7**
MAG	81.27	84.09	83.07	**17**	1072	40
MUS	92.25	94.62	93.59	22	202	15
PIM	70.85	69.41	**76.89**	26	111	**5**
POL	77.41	69.81	77.45	21	558	**5**
RIN	88.94	87.54	77.25	368	354	**4**
SAT	83.90	84.01	83.30	11	163	9
SPA	78.38	90.14	89.86	22	155	**13**
TWO	97.92	87.59	98.00	41	429	**3**

In terms of tree size, LDT is significantly better than soft tree which is also significantly better than the hard tree except on *magic* where soft tree is significantly better than LDT and hard tree, on *musk2* and *satellite47* where there is no difference between soft tree and LDT. LDT uses post-pruning whereas hard and soft trees use pre-pruning; post-pruning may be more aggresive than pre-pruning which can sometimes deteriorate performance.

4. Conclusions

We discuss a decision tree model with soft decisions, which makes use of a soft gating function to merge the decisions of the subtrees. The proposed model is visualized on a toy data set and evaluated on ten regression and ten classification data sets. The model is shown to have better or comparable performance to hard trees, while having fewer nodes.

Soft trees have several advantages: First, they provide a soft response whereas hard trees have discontinuous response at the leaf boundaries. This enables soft tree to have smoother fits and hence lower bias around the split boundaries. Second, a linear gating function enables soft trees to make oblique splits in contrast to the axis-orthogonal splits made by hard trees. In our experiments, we see that these two properties reduce the number of nodes required to solve a regression or a classification problem. In terms of accuracy, soft trees seem suited to regression problems where the gating function allows a smooth interpolation between its children.

One drawback of soft trees is gradient-descent which is prone to get stuck at local minima. In our experiments, initialization derived from the split which would have been made by a hard node work quite well.

Acknowledgments. This work is supported by TÜBİTAK 109E186 and Boğaziçi University Scientific Research Project BAP5701.

References

[1] E. Alpaydın. Combined 5×2 cv F test for comparing supervised classification learning classifiers. *Neural Computation*, 11:1975–1982, 1999.

[2] C. Blake and C. Merz. UCI repository of machine learning databases, 2000.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. John Wiley and Sons, 1984.

[4] H. Guo and S. B. Gelfand. Classification trees with neural network feature extraction. *IEEE Transactions on Neural Networks*, 3:923–933, 1992.

[5] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214, 1994.

[6] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[7] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Meteo, CA, 1993.

[8] A. Ruta and Y. Li. Learning pairwise image similarities for multi-classification using kernel regression trees. *Pattern Recognition*, 45:1396–1408, 2011.

[9] O. T. Yıldız and E. Alpaydın. Omnivariate decision trees. *IEEE Transactions on Neural Networks*, 12(6):1539–1546, 2001.

[10] O. T. Yıldız and E. Alpaydın. Linear discriminant trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):323–353, 2005.