2014 22nd International Conference on Pattern Recognition

Budding Trees

Ozan İrsoy
Department of Computer Science
Cornell University
Ithaca, NY 14853-7501, USA
Email: oirsoy@cs.cornell.edu

Olcay Taner Yıldız
Department of Computer Engineering
Işık University
Şile, İstanbul 34980 Turkey
Email: olcaytaner@isikun.edu.tr

Ethem Alpaydın
Department of Computer Engineering
Boğaziçi University
Bebek, İstanbul 34342 Turkey
Email: alpaydin@boun.edu.tr

***Abstract*—We propose a new decision tree model, named the budding tree, where a node can be both a leaf and an internal decision node. Each bud node starts as a leaf node, can then grow children, but then later on, if necessary, its children can be pruned. This contrasts with traditional tree construction algorithms that only grows the tree during the training phase, and prunes it in a separate pruning phase. We use a soft tree architecture and show that the tree and its parameters can be trained using gradient-descent. Our experimental results on regression, binary classification, and multi-class classification data sets indicate that our newly proposed model has better performance than traditional trees in terms of accuracy while inducing trees of comparable size.**

I. Introduction

A decision tree is a hierarchical structure for supervised learning tasks, composed of internal decision nodes and terminal label nodes [1]–[3]. Given an input vector (including a bias term) $x = [1, x_1, ..., x_d]^T$, the response at node m has the following recursive definition:

$$y_m(\boldsymbol{x}) = \begin{cases} \rho_m & \text{if } m \text{ is leaf} \\ y_{ml}(\boldsymbol{x}) & \text{else if } g_m(\boldsymbol{x}) > 0 \\ y_{mr}(\boldsymbol{x}) & \text{else if } g_m(\boldsymbol{x}) \leq 0 \end{cases} \quad (1)$$

If m is a leaf node, for binary classification, $\rho_m \in [0, 1]$ is the probabilistic response denoting the probability that the instance is positive; for regression $\rho_m \in \mathbb{R}$ is the numeric response. If m is not a leaf but an internal node, depending on the outcome of the test $g_m(\boldsymbol{x})$, we take the left or right branch, y_{ml} and y_{mr} respectively, and continue recursively.

Frequently [2], $g_m(\boldsymbol{x})$ uses only one of the input attributes:

$$g_m(\boldsymbol{x}) = x_j + w_0$$

and this is called the *univariate tree*. The *multivariate tree* [4], [5] is a generalization where we define

$$g_m(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

hence defining arbitrary *oblique* splits. The univariate tree is a special case where $w_{mi} = 1$ for some $i \in \{1, ..., d\}$ and $w_{mj} = 0$ for all $j \notin \{0, i\}$, and as such defines a split that is orthogonal to the axis x_i. If we relax the linearity assumption on $g_m(\cdot)$, we have the *multivariate nonlinear tree*. If the above constraints on $g_m(\cdot)$ are dependent on the node m itself, then we have the *omnivariate tree* [6].

Regardless of the type of the decision node, learning a tree is a difficult problem [3]. Finding the smallest decision tree that can classify all the instances in a training set is NP-hard [7]. That is why, decision tree induction algorithms are greedy—they do not guarantee finding the smallest decision tree but they learn in reasonable time.

Basically, a decision tree induction algorithm is composed of two steps:

1) *Growing the tree:* Starting from the root, at each node, given the data reaching that node, we look for the best decision function $g_m(\boldsymbol{x})$ (univariate or multivariate) that splits the data into two. If this split leads to an improvement (for example, in terms of entropy), the split is accepted, the node becomes a decision node with two children and tree generation continues at the two children recursively. If the split does not lead to any improvement, the node is not split further and remains as a leaf and a proper probability or numeric value is stored in it.

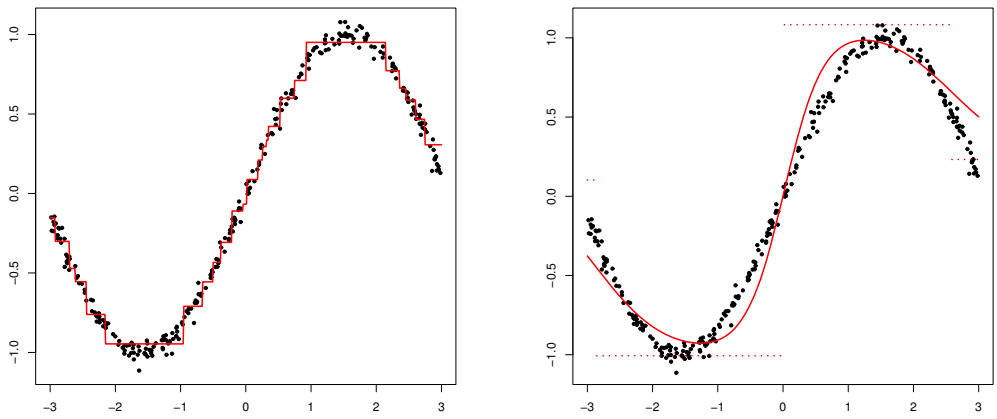
2) *Pruning the tree:* Once the tree is grown to its total length, we check if pruning a subtree, that is, replacing it with a leaf, leads to improvement over a separate pruning set. We are basically checking if the tree is overfitting at this stage and if so, by replacing a subtree with a leaf we are getting rid of variance.

That is, tree induction is composed of a first phase of greedily adding subtrees and the second phase of greedily replacing subtrees with leaves.

Our proposal in this work is to have a tree architecture where each node, which we call a *bud node*, can be an internal node and a leaf *at the same time*. We allow splitting and pruning at the same phase in tree learning, instead of having two separate phases each allowing only one type of change. A bud starts as a leaf, and if necessary may grow into a decision node and get children, but always retains its value as a leaf, and at any stage later on may lose its children and become a leaf again.

As we will see later, training a budding tree is an incremental process where small changes are done at each step and in Section II, we discuss the soft decision tree that is better suited for such small adjustments and as such forms the basis of the budding tree. In Section III, we discuss the budding tree model and how it is learned. We give experimental results in Section IV where we compare the budding tree with various univariate and multivariate tree models on many regression, binary, and multi-class classification data sets and we conclude in Section V.

1051-4651/14 $31.00 © 2014 IEEE
DOI 10.1109/ICPR.2014.616



(a) Hard fit

(b) Soft fit

Fig. 1. Hard and soft tree fits to sinusoidal toy data. The hard tree gives a piecewise constant approximation because of the hard splits whereas the soft gating function allows a smooth interpolation between values at the leaves (shown as dotted lines).

II. Soft Decision Tree

Equation (1) defines a *hard* split where in a decision node, depending on $g_m(\boldsymbol{x})$, we take one of the two branches—we either go to the left or to the right. In a *soft decision tree* [8], we have a *soft* split:

$$y_m(\boldsymbol{x}) = \begin{cases} \rho_m & \text{if } m \text{ is leaf} \\ g_m(\boldsymbol{x})y_{ml}(\boldsymbol{x}) + (1 - g_m(\boldsymbol{x}))y_{mr}(\boldsymbol{x}) & \text{otherwise} \end{cases} \tag{2}$$

where

$$g_m(\boldsymbol{x}) = \text{sigmoid}(\boldsymbol{w}_m^T\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}_m^T\boldsymbol{x})} \tag{3}$$

is the *gating function* that assigns input *softly* to the two children. Note that the splits are multivariate and hence oblique.

In contrast to a hard node which redirects the decision to only one of its children, a soft node propagates the decision to both children, weighted by the value of the gating function. Originally, this is the idea behind the hierarchical mixture of experts [9]. If we consider separating the regions of responsibility to the left and right subtrees as a binary classification problem, then the gating model implements a logistic linear model to estimate the posterior probability of the responsibilities: $\mathbb{P}_m(\text{left}|\boldsymbol{x}) := g_m(\boldsymbol{x})$ and $\mathbb{P}_m(\text{right}|\boldsymbol{x}) := 1 - g_m(\boldsymbol{x})$.

Soft decision trees are trained incrementally too. When necessary, nodes are split, children are added and the split parameters and the leaf values are learned using gradient-descent. At any split decision during training, every node in the tree is held fixed, besides the node currently being split and its two children.

An example is shown in Figure 1 that compares hard and soft trees. We see that though both uses leaf nodes that are constants, in the hard tree, hard splits cause a piecewise constant approximation that leads to a fit that looks like a ladder. In the case of a soft tree, because we take a weighted average where weights are given by the soft gating function, we get a smooth interpolation between leaves and hence a better fit with much fewer nodes. In this example, the hard tree uses 67 nodes whereas the soft tree uses only seven nodes.

III. The Budding Tree

A. The Model

We define a *bud node*, which further generalizes the soft node, by softening the notion of *being a leaf* as well:

$$\begin{aligned} y_m(\boldsymbol{x}) &= (1 - \gamma_m)\left[g_m(\boldsymbol{x})y_{ml}(\boldsymbol{x}) + (1 - g_m(\boldsymbol{x}))y_{mr}(\boldsymbol{x})\right] \\ &+ \gamma_m\rho_m \end{aligned} \tag{4}$$

where $\gamma_m \in [0, 1]$ is the *leafness* parameter. Recursion ends when a node having $\gamma = 1$ is encountered.

A budding tree has a response value in every bud and internal buds make a contribution to the overall response value as well. Note that budding trees do not have more expressive power than soft trees: We may convert any budding tree to a soft tree without altering the response, by recursively distributing internal bud contributions, $\gamma\rho$, downwards the tree to the leaves, such that $\rho = \gamma = 0$ for every internal bud and $\gamma = 1$ for every leaf bud. However, this extension will allow us to use a different training approach, as we discuss below.

Observe that any finite budding decision tree can be seen as a complete infinite binary tree with leaves having $\gamma = 1$ and the nodes below the leaves having arbitrary parameters, since they do not affect the overall response value. Thus, we can parametrize any tree by a countably infinite number of parameters $\{\boldsymbol{w}_m, \rho_m, \gamma_m\}_m$. With this parametrization, our training algorithm will optimize a loss function over this infinite dimensional space of all trees. In practice though, we will have nodes with $\gamma = 1$ and hence the actual physical size of the tree will be finite. This will provide a principled way to train the model without fixing any part of the tree.

Note that a similar parametrization can be made for multivariate linear trees, or soft decision trees, with the constraint that γ_m is either 0 or 1 for all m. However, by relaxing this constraint so that γ_m can take any value from the interval $[0, 1]$, our optimization problem becomes a continuous optimization problem, rather than a discrete one. This allows us to use any conventional box-constrained continuous nonlinear optimization method. Even though there is no formal necessity, we will use stochastic gradient-descent for ease of exposition and implementation.

B. Training

Training proceeds as follows: We start with a root node r having $\boldsymbol{w}_r = \boldsymbol{\epsilon}, \rho_r = \epsilon, \gamma_r = 1$, where ϵ is a small noise. This is the tree with a single node having a fixed response. At any iteration, we will compute the gradient of the error function with respect to all of the physical nodes we currently have (which is finitely but unboundedly many), and make a small update to every parameter of every node, as we will discuss below.

If any of the nodes make a transition from having $\gamma = 1$ to having $\gamma < 1$, we will physically split that node. We will initialize every node m with $\boldsymbol{w}_m = \boldsymbol{\epsilon}, \rho_m = \epsilon, \gamma_m = 1$. With this training scheme however, the tree has the possibility of exploding, and to avoid that, we add a regularization term to the error function to push all γ values towards 1. Hence, later on during training, γ_m can be pushed back to 1, effectively pruning the children.

The training process grows or shrinks the tree depending on the values of γ_m of all nodes. Note that while training, we do not fix any part of the tree, but update all parameters of all the nodes. This is different from traditional decision tree methods where all the nodes except the currently added one are fixed, which hence might suffer from fixing the upper nodes early on, without seeing how the lower nodes will do in the following iterations. The budding tree avoids this problem because it learns all the tree parameters jointly, rather than learning only the parameters of the last node.

The budding tree can explore a potential split by turning a leaf node into a partially internal node (decreasing γ from 1 to, say, 0.95), and then after failing to capture a signal for some time, it can turn that node into a leaf again (due to the regularization term). This implies that we have growing and pruning together; it implies a search with the possibility of backtracking.

More formally, for regression, given the training set $\mathcal{X} = \{\boldsymbol{x}^t, r^t\}_{t=1}^N$, where $r^t \in \mathbb{R}$ is the desired output for $\boldsymbol{x}^t$, we have

$$\begin{array}{ll} \text{minimize} & J = \sum_t \frac{1}{2}(y_r(\boldsymbol{x}^t) - r^t)^2 + \lambda \sum_m (1 - \gamma_m) \\ \text{subject to} & \gamma_m \in [0, 1] \quad \forall m \end{array} \tag{5}$$

where $y_r(\boldsymbol{x})$ is the output at the root node, calculated using Equation (4) recursively.

The first term in the objective function of Equation (5) is simply the sum of squared errors. The second term is the aforementioned regularization term on γ_m, where λ is the regularizing hyper-parameter. This second term is a decay term that penalizes γ_m values that are less than 1; hence, it pushes every node towards being a leaf, rather than an internal node.

Let us focus on stochastic gradient-descent (the error is computed with respect to a single instance $\boldsymbol{x}$). Define $\delta_m = \partial J^t / \partial y_m(\boldsymbol{x}^t)$, which is the *responsibility* of the node m. Back-propagating the error from the root towards the leaves, we have

$$\begin{aligned} \frac{\partial J^t}{\partial w_{mi}} &= \delta_m^t (1 - \gamma_m) g_m(\boldsymbol{x}^t)(1 - g_m(\boldsymbol{x}^t)) \\ &\quad (y_{ml}(\boldsymbol{x}^t) - y_{mr}(\boldsymbol{x}^t)) x_i^t, i = 0, \ldots, d \\ \frac{\partial J^t}{\partial \rho_m} &= \delta_m^t \gamma_m \\ \frac{\partial J^t}{\partial \gamma_m} &= \delta_m^t [-g_m(\boldsymbol{x}^t) y_{ml}(\boldsymbol{x}^t) \\ &\quad - (1 - g_m(\boldsymbol{x}^t)) y_{mr}(\boldsymbol{x}^t) + \rho_m] \\ &\quad - \lambda \end{aligned}$$

with

$$\delta_m^t = \begin{cases} y_r(\boldsymbol{x}) - r^t & \text{if } m \text{ is root } r \\ \delta_{pa(m)}^t (1 - \gamma_m) g_m(\boldsymbol{x}^t) & \text{if } m \text{ is a left child} \\ \delta_{pa(m)}^t (1 - \gamma_m)(1 - g_m(\boldsymbol{x}^t)) & \text{if } m \text{ is a right child} \end{cases}$$

where $pa(m)$ is the parent node of node m.

Whenever a γ_m value exceeds 1 from above or 0 from below, after an update, we restore it to 1 or 0, which effectively projects the gradient onto the feasible space. When a node with $\gamma_m < 1$ has $\gamma_m = 1$ after the update, we do not erase the parameter values of its children. This means if the node will be split again later, its children will continue from their previous set of parameter values.

Starting from a single node with a fixed response, the training starts to explore a potential split, when an error signal causes a decrease in γ_m. As the stochastic gradient-descent iterates, this causes small updates on the parameters of the children. If the split is actually helpful, γ of the parent continues to decrease smoothly (causing a smooth transformation from a leaf to an internal node), which will cause greater updates in the parameters of the children (which in turn can be split during the process).

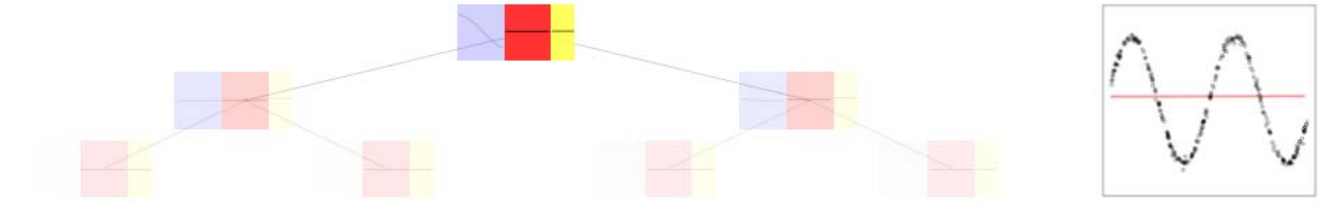
This can be considered as a redistribution of responsibility from the parent to its children: If the parent cannot explain the data as good as the children being explored, updates will slowly push the parent into being an internal node, increasing the effect of the children on the response. However, if the split is not helpful, regularization term will keep γ of the parent very close to 1, causing it to behave as a leaf.

We employ an adaptive learning rate using a diagonal variant of AdaGrad [10]: Let $g_\theta^{(\tau)}$ be the gradient at time step τ with respect to parameter $\theta \in \{\rho_m, \gamma_m, w_m\}_m$. The update rule is

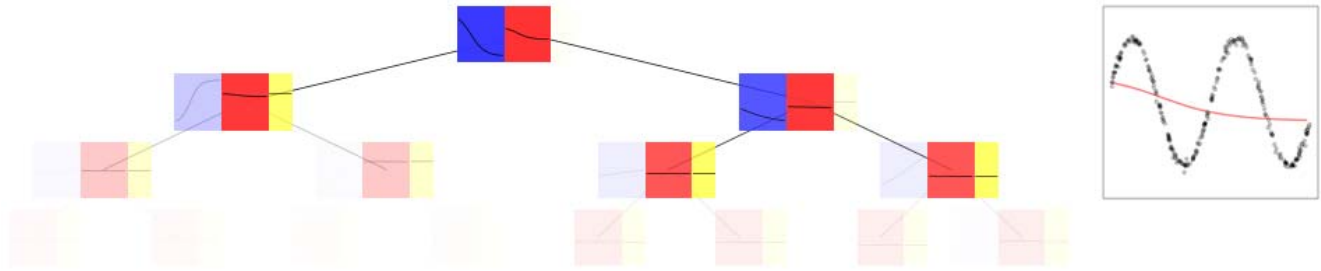
$$\theta^{(\tau)} = \theta^{(\tau-1)} - \phi \frac{g_\theta^{(\tau)}}{\sqrt{\sum_{l=1}^{\tau} (g_\theta^{(l)})^2}}$$

where ϕ is the learning rate.

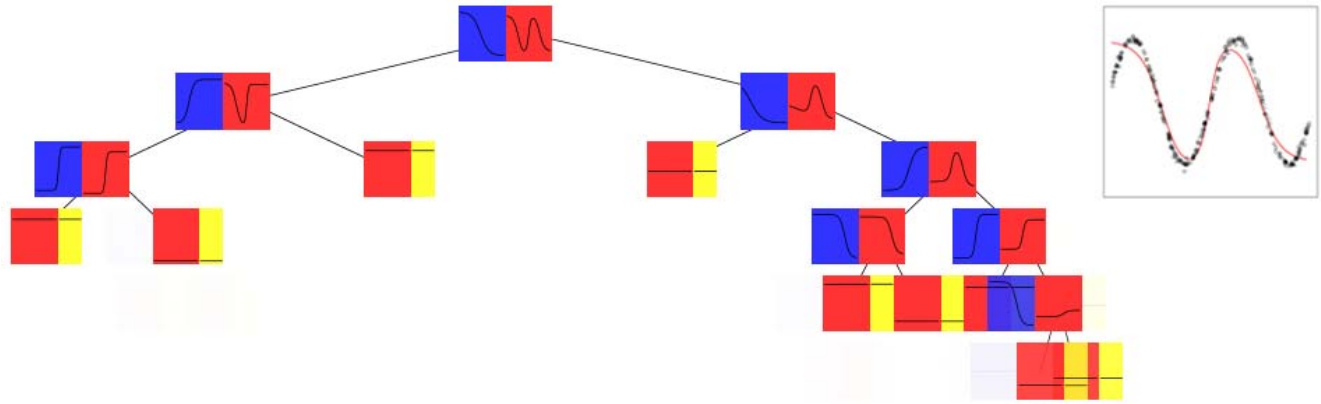
Intuitively, this causes rare parameters to get larger updates. This is very helpful in our case because, since higher nodes are



(a) Iteration 24



(b) Iteration 1329



(c) Iteration 2994

Fig. 2. The evolution of a budding tree during training. For every node in the tree, red plots (middle box) show the response function of the corresponding subtree, blue (left box) plots show the gating function (a higher value assigns more responsibility to the left subtree) and yellow plots (right box) show the ρ values. Overall transparency of a node denotes the overall effect on the response function, determined by the γ values of ancestors (e.g., a node closer to being leaf has more transparent children). The transparency of blue and yellow plots further depend on the γ value of the node itself, a higher γ results in a more transparent gating plot and more opaque plot of ρ. We see that early on, though the tree may look big, lower level nodes and are not really used; as learning continues, to decrease error, they move away from being a leaf, become a subtree and grow children and thus get a better fit.

updated more frequently than the lower nodes, we would like the nodes closer to the root to slow down and stabilize, and nodes away from root to update more aggressively to search for good splits and leaf values.

An example of how a budding tree evolves during training is shown in Figure 2. In the beginning, most nodes are leaves (their children are transparent in the figure) and hence the tree gives a fit that is too smooth. As we do more iterations, leafness of nodes decrease (they get more opaque in the figure), they become internal nodes and allow their children to be updated, which leads to better approximation.

C. Extension to classification

Extending the definition of budding trees to classification is simple. For binary classification, we pass the tree output from the root through a sigmoid to get a value between 0 and 1, which hence can be interpreted as the posterior probability of the positive class. In classification, the first term of Equation (5) becomes the cross-entropy instead of the squared difference. The gradient-descent update rules are calculated accordingly.

For $K > 2$ classes, we have a single tree but we store K values at the leaves for the classes, that is, $\boldsymbol{\rho}$ is a K-dimensional vector. We calculate the output for all classes using the same tree but for each using the corresponding ρ entry in the leaves. We then softmax the K values calculated at the root for the K classes. Again, cross-entropy is used and the update rules are derived accordingly.

IV. Experiments

In this section, we report experimental results for regression, binary, and multi-class classification tasks.

TABLE I. REGRESSION EXPERIMENTS

	MSE			Node count		
	Hard	Soft	Budding	Ha.	So.	Bu.
ABA	0.541	0.421	0.416	44	**21**	35
ADD	0.244	0.070	**0.046**	327	49	**35**
BOS	0.342	0.273	0.218	19	32	19
CAL	0.311	0.251	**0.240**	300	146	**94**
COM	0.036	0.023	**0.019**	110	31	**19**
CON	0.268	0.208	**0.156**	101	39	38
8FH	0.416	0.381	**0.378**	47	**5**	13
8FM	0.068	0.051	**0.050**	164	**9**	17
8NH	0.394	0.358	**0.342**	77	**24**	27
8NM	0.066	0.049	**0.036**	272	**23**	37

TABLE II. BINARY CLASSIFICATION EXPERIMENTS

	Accuracy				Node count			
	C4.5	LDT	Soft	Bud.	C.	L.	So.	Bu.
BRE	93.2	95.0	**96.5**	94.9	7	4	3	12
GER	70.0	74.1	**75.9**	68.7	**1**	3	8	56
MAG	82.5	83.0	85.3	**86.3**	53	38	56	122
MUS	94.5	93.5	95.6	**97.0**	62	11	33	15
PIM	72.1	**76.8**	75.0	67.1	8	5	7	68
POL	69.4	77.4	77.1	72.5	34	**3**	18	61
RIN	87.7	77.2	**90.1**	88.5	93	**3**	76	61
SAT	84.5	83.3	87.5	86.8	25	**9**	27	38
SPA	90.0	89.8	**92.4**	91.4	36	13	12	49
TWO	82.9	98.0	97.9	96.7	163	**3**	7	29

We use ten regression (*ABAlone*, *ADD10*, *BOSton*, *CALifornia*, *COMp*, *CONcrete*, *puma8FH*, *puma8FM*, *puma8NH*, *puma8NM*) and ten binary classification (*BREast*, *GERman*, *MAGic*, *MUSk2*, *PIMa*, *POLyadenylation*, *RINgnorm*, *SATellite47*, *SPAmbase*, *TWOnorm*) data sets from the UCI repository [11], as in [8]. We also use ten multiclass classification (*BALance*, *CMC*, *DERmatology*, *ECOli*, *GLAss*, *OPTdigits*, *PAGeblock*, *PENdigits*, *SEGment*, *YEAst*) data sets from the UCI repository.

We compare budding trees with the hard C4.5 tree and soft tree for regression, and with hard univariate C4.5 [2], hard multivariate LDT [5] and soft tree [8] on classification problems. Our experimental 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. Soft and hard trees (including the linear discriminant tree) use the validation set as a pruning set. Budding trees use the validation set to tune the hyper-parameters ϕ and λ. Statistical significance is tested with the paired t-test for the performance measures, and the Wilcoxon Rank Sum test for the tree sizes, both with significance level $\alpha = 0.05$. The values reported are results on the test set not used for training or validation (model selection). Significantly best results are shown in boldface in the figures.

Table I shows the mean square error and the number of nodes of hard, soft and budding trees on the regression data sets. Except two ties, the budding tree has significantly smaller mean square error. In terms of the number of nodes, budding tree has three wins (*add10*, *california*, *comp*), soft tree has five wins (*abalone*, *puma8fh*, *puma8fm*, *puma8nh*, *puma8nm*) and there are two ties. Note though that at the end of training, budding trees usually have leaves with parents having $\gamma \approx 1$—one can gain from node counts by pruning these subtrees starting from such nodes at negligible change in the response accuracy.

Table II shows the accuracy and the number of nodes of C4.5, LDT, soft and budding trees over binary classification data sets. In terms of accuracy, budding tree has two wins (*magic*, *musk2*), soft tree has four wins (*breast*, *german*, *ringnorm*, *spambase*), LDT has one win (*pima*), and there are three ties. In terms of tree sizes, C4.5 and LDT provide significantly smaller trees on one (*german*) and four (*polyadenylation*, *ringnorm*, *satellite47*, *twonorm*) data sets, respectively. Upon investigation, we attribute some of the dissatisfactory performance results of the budding tree to the small training/validation/test set sizes. For instance, on *pima*, *german* and *polyadenylation*, budding tree achieves good accuracy over validation sets, but fails to generalize well to the test set.

TABLE III. MULTI-CLASS CLASSIFICATION EXPERIMENTS

	Accuracy				Node count			
	C4.5	LDT	Soft	Budding	C4.5	LDT	So.	Bu.
BAL	61.91	88.46	89.85	**92.44**	5	**3**	10	29
CMC	50.00	46.64	52.03	53.23	24	**3**	21	28
DER	94.00	93.92	93.6	94.8	15	11	11	11
ECO	77.47	81.39	76.78	**83.56**	9	11	10	24
GLA	56.62	53.37	54.05	53.78	20	9	11	21
OPT	84.85	93.73	90.97	**94.57**	120	**31**	58	40
PAG	**96.71**	94.65	95.7	96.51	23	29	**16**	37
PEN	92.95	96.60	96.64	**98.13**	169	66	54	54
SEG	94.48	91.96	93.99	**95.63**	41	33	**22**	33
YEA	54.61	56.66	55.82	**59.31**	24	22	34	41

Table III shows the accuracy and the tree sizes of C4.5, LDT, soft and budding tree over multi-class classification data sets. Budding tree has significantly better performance than C4.5 and LDT over six data sets (*balance*, *ecoli*, *optdigits*, *pendigits*, *segment*, *yeast*), C4.5 has one win (*pageblock*) and there are three ties (*cmc*, *dermatology*, *glass*). In terms of three sizes, LDT has three wins (*balance*, *cmc*, *optdigits*), soft tree has two wins (*pageblock*, *segment*), and there are five ties (*dermatology*, *ecoli*, *glass*, *pendigits*, *yeast*).

V. CONCLUSIONS

We propose a new decision tree model where a node can be a leaf and an internal node at the same time. Our proposed budding trees have several advantages over soft trees.

Budding trees solve the optimization problem taking all the parameters into account, unlike traditional trees which solve incremental subproblems greedily, one subtree at a time. During training, as new nodes are added, the existing node parameters are not fixed but the whole tree is continuously updated so as to better take into account the changes in the model.

They retain the advantages of soft trees over hard trees: A soft response function provides smoother fits. This allows less bias near the decision boundaries for classification tasks, and provides a smooth interpolation between children, which better suits regression problems. Furthermore, a linear gating function enables oblique splits in contrast to axis-orthogonal splits of univariate trees.

Budding nodes are able to make a transition smoothly from being a leaf to being an internal node. This makes small updates possible, which allows true online learning without explicitly reconstructing the tree. It is also possible for an expanded node to go back to being a leaf again, hence incorporating pruning into the training, which normally used to be a separate phase. This can also be seen as backtracking

and is better than greedy hill-climbing that never backtracks as is traditionally used in tree construction.

Budding trees have a soft architecture and provide a continuous and differentiable response in terms of their parameters and hence they can be trained using a continuous optimization method like gradient-descent. With the utilization of chain rule, the parameters in all the layers and nodes can be trained together, each proportional to its responsibility.

Our experimental results on ten regression, ten binary classification and ten multi-class classification data sets have shown that budding trees, most of the time, lead to better approximation than well-known decision tree methods using trees of comparable sizes. Due to continuity and stochasticity of the optimization, most leaf nodes have parents that are almost leaves with their $\gamma \approx 1$, which increases the number of nodes with negligible change in the response. These extra nodes make it difficult to compare model complexities and if desired, can be pruned to reduce the number of nodes. As a future research direction, L1 or L2 type regularization on the splitting hyperplane parameters w can be employed as in [12], followed by a feature selection step. Another possible direction is to train multiple such budding trees and combine them, either by randomization and averaging, or through backfitting [13].

Of course, using gradient-descent has the disadvantage of a possibility of getting stuck in local minima but note that incremental tree construction methods are also far from finding the optimal decision tree and they investigate a much smaller portion of the state space.

In terms of representational power, the budding tree is not more powerful than a soft tree, that is, it cannot approximate any function that cannot be approximated by a soft tree. The novelty of the budding tree is in its construction, namely, that it implements a smooth change in the state space of trees. Whereas the traditional tree induction methods have explicit steps where a leaf is converted to a subtree and always stays that way, unless pruned in a separate pruning phase, in the budding tree, this state transition between leaf and decision node, or child/parent is done smoothly; a leaf may become the decision node of subtree, it may be both, and then it may go back to being a leaf again. In that sense, it can be said that the budding tree implements a floating search in the space of trees as opposed to the traditional tree induction methods that always go forward in training, and always backward in pruning. The budding tree allows smooth and retractable changes in both directions.

REFERENCES

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Pacific Grove, California: Wadsworth & Brooks, 1984.

[2] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[3] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 35, no. 4, pp. 476–487, 2005.

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

[5] O. T. Yıldız and E. Alpaydın, "Linear discriminant trees," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 03, pp. 323–353, 2005.

[6] ——, "Omnivariate decision trees," *Neural Networks, IEEE Transactions on*, vol. 12, no. 6, pp. 1539–1546, 2001.

[7] T. Hancock, T. Jiang, M. Li, and J. Tromp, "Lower bounds on learning decision lists and trees," *Information and Computation*, vol. 126, no. 2, pp. 114–122, 1996.

[8] O. İrsoy, O. T. Yıldız, and E. Alpaydın, "Soft decision trees," in *International Conference on Pattern Recognition*, 2012.

[9] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[11] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[12] O. T. Yıldız and E. Alpaydın, "Regularizing soft decision trees," in *Computer and Information Sciences III: 27th International Symposium on Computer and Information Sciences*, E. Gelenbe and R. Lent, Eds. Springer, 2013, pp. 15–21.

[13] H. A. Chipman, E. I. George, R. E. McCulloch *et al.*, "Bart: Bayesian additive regression trees," *The Annals of Applied Statistics*, vol. 4, no. 1, pp. 266–298, 2010.