Structural, Syntactic, and Statistical Pattern Recognition
Joint IAPR International Workshops, S+SSPR 2022, Montreal, QC, Canada,
August 26–27, 2022,
LNCS, 13813, Springer, pp 152-162.

Distributed Decision Trees

Ozan İrsoy[1][0000−0002−7123−8361] and Ethem Alpaydın[2][0000−0001−7506−0321]

[1] Department of Computer Science, Cornell University,
Ithaca, NY 14853
oirsoy@cs.cornell.edu
[2] Department of Computer Science, Özyeğin University,
Çekmeköy 34794, Istanbul, Turkey
ethem.alpaydin@ozyegin.edu.tr

Abstract. In a budding tree, every node is part internal node and part leaf. This allows representing the tree in a continuous parameter space and training it with backpropagation, like a neural network. Unlike a traditional tree whose construction is composed of two distinct stages of growing and pruning, "bud" nodes grow into subtrees or are pruned back dynamically during learning. In this work, we extend the budding tree and propose the distributed tree where the children use different and independent splits; hence, multiple paths in a tree can be traversed at the same time. In a traditional tree, the learned representations are local, that is, activation makes a soft selection among all the root-to-leaf paths in a tree, but the ability to combine multiple paths of the distributed tree gives it the power of a distributed representation, as in a traditional perceptron layer. Our experimental results show that distributed trees perform comparably or better than budding and traditional hard trees.

Keywords: Decision trees · Hierarchical mixture of experts · Local vs distributed representations

1 Introduction

The decision tree is one of the most widely used models for supervised learning. Consisting of internal decision nodes and terminal leaf nodes, it implements a hierarchical decision function [3, 13, 14]. The decision nodes act as input-dependent gates and the response is read from the leaves.

Different decision tree architectures differ in the way the decision nodes are defined and/or how the tree is constructed. The basic decision tree is *hard* in the sense that a decision node chooses one of the branches. It is also *univariate*, that is, only one input attribute is used in making a decision.

In this work[3], we are going to discuss a number of extensions of the basic hard univariate tree; in Section 2, we will discuss the multivariate tree, the soft

[3] This submission is a revised version of our original report arXiv:1412.6388 from 2014, which has not been published in any conference/journal since then. Given the current need for interpretable/explainable alternatives to neural networks, we have decided to pick up this line of research again. O. İrsoy is now with Bloomberg L.P.

tree, and the budding tree. In Section 3, we will propose a new type of tree, namely the distributed budding tree. We give experimental results in Section 4 and conclude in Section 6.

2 Different Tree Architectures

2.1 Hard Decision Trees

Given input $\boldsymbol{x} = [1, x_1, \ldots, x_d]$, the response at node m is defined recursively:

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

If m is an internal node, the decision is forwarded to the left or right subtree depending on the outcome of the gating $g_m(x)$ (In this work, we assume that all input attributes are numeric and we build binary trees.) If m is a leaf node, for regression $\rho_m \in \mathbb{R}$ returns the scalar response value; for binary classification $\rho_m \in [0, 1]$ returns the probability of belonging to the positive class, and for tasks requiring multidimensional outputs (e.g. multiclass classification, vector regression), $\boldsymbol{\rho}_m$ is a vector.

In a univariate tree, the gating uses a single input dimension:

$$g_m(\boldsymbol{x}) = x_{j(m)} - c_m \tag{2}$$

where $j(m)$ denotes the attribute index $(1 \ldots d)$ used in node m and c_m is the corresponding threshold value against which the value of that attribute is compared. We choose one of the two branches depending on the sign of the difference. Decision tree induction algorithms, e.g., C4.5 [13], allow us find the best attribute and threshold at each node using the subset of the training data reaching that node (i.e., satisfying all the conditions on the nodes above m).

The *multivariate tree* [12, 16] is a generalization where all input attributes are used in gating. In a linear multivariate tree, we have

$$g_m(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \tag{3}$$

A univariate decision defines a split boundary that is orthogonal to one of the axes but the linear multivariate node allows arbitrary *oblique* splits. If we relax the linearity assumption on $g_m(\cdot)$, we get the *multivariate nonlinear tree*; for example, in [4], $g_m(\cdot)$ is defined as a multilayer perceptron. In an *omnivariate tree*, a node can be univariate, linear, or nonlinear multivariate [15].

Regardless of the gating and the leaf response functions, inducing the optimal decision tree is a difficult problem [14]. Finding the smallest decision tree that perfectly classifies a given set of input is NP-hard [5]. Thus typically, decision trees are constructed greedily.

Essentially, decision tree induction consists of two steps:

1. Growing the tree: Starting from the root node, at each node m, we search for the best decision function $g_m(\boldsymbol{x})$ that splits the data that reaches the node m. If the split provides an improvement in terms of a measure (e.g. entropy), we keep the split, and recursively repeat the process for the two children ml and mr. If the split does not provide any improvement, then m is kept as a leaf and ρ_m is assigned accordingly.
2. Pruning the tree: Once the tree is grown, we can check if reducing the tree complexity by replacing subtrees with leaf nodes leads to an improvement on a separate development set. This is done to avoid overfitting and improve generalization of the tree.

2.2 Soft Decision Trees

The hierarchical mixture of experts [10] define a soft decision tree where the gating nodes are linear multivariate and the splitting is soft; that is, instead of choosing one of the two children, both are chosen with different probabilities. This allows the overall output of the tree to be continuous and we can use backpropagation to update all of the tree parameters, in the gating nodes and the leaves.

More formally, a soft decision tree models the response as the following recursive definition:

$$y_m(\boldsymbol{x}) = \begin{cases} \rho_m & \text{if } m \text{ is a leaf} \\ g_m(\boldsymbol{x}) y_{ml}(\boldsymbol{x}) + (1 - g_m(\boldsymbol{x})) y_{mr}(\boldsymbol{x}) & \text{otherwise} \end{cases} \tag{4}$$

where $g_m(\boldsymbol{x}) = \sigma(\boldsymbol{w}_m^T \boldsymbol{x})$ where $\sigma(\cdot)$ is the sigmoid (logistic) function.

Soft decision trees are grown incrementally in a similar fashion to hard trees [8]. Every node is recursively split until a stopping criterion is reached, and we use gradient-descent to learn the splitting hyperplane of the parent and the response values of its two children.

In a previous study, we have shown how convolutional preprocessing can be added to soft decision trees thereby achieving an architecture that can compete with deep multilayer network [1].

2.3 Budding Trees

Budding trees [9] generalize soft trees further by softening the notion of *being a leaf* as well. Every node (called a *bud* node) is part internal node and part leaf. For a node m, the degree of how much of m is a leaf is defined by the *leafness* parameter $\gamma_m \in [0, 1]$. This allows the response function to be continuous with respect to the parameters, including the structure of a tree.

The response of bud node m is recursively defined:

$$y_m(\boldsymbol{x}) = (1 - \gamma_m)\big[g_m(\boldsymbol{x}) y_{ml}(\boldsymbol{x}) + (1 - g_m(\boldsymbol{x})) y_{mr}(\boldsymbol{x})\big] + \gamma_m \rho_m \tag{5}$$

The recursion ends when a node with $\gamma_m = 1$ is encountered. (In a traditional decision tree, γ_m is binary; it is 0 for an internal node and 1 for a leaf.)

The augmented error that we minimize is

$$E' = E + \lambda \sum_m (1 - \gamma_m) \quad \text{subject to } \gamma_m \in [0, 1], \forall m \tag{6}$$

where E is the original regression/classification error calculated using the output of the tree on a labelled training set, and the second term is a regularization/penalty term forcing nodes to be leaves.

Note that while backpropagating, not only we update all the decision node weights and the leaf values, but we also update γ_m, that is, we are also updating the structure of the tree. Initially, a node starts with its $\gamma_m = 1$ and if it is updated to be less, we physically split the node and grow a subtree with its gating function and children (with their own γ starting from 1). Similarly, a previously grown subtree that turned out be useless can be pruned back because of the regularization term.

Another difference is that any training instance updates all the parameters of all the tree and so the growth of any subtree affects the rest of the tree.

3 Distributed Budding Trees

Even though the soft tree (and also the budding tree) provide a means for hierarchical representation learning, the resulting representations are local. The overall response is a convex combination of all the leaves. This provides a soft selection among the leaves that is akin to a hierarchical softmax, and limits the representational power—If we use hard thresholds instead of soft sigmoidal thresholds, it essentially selects one of the leaves (paths). Thus, it partitions the input space into as many regions as the number of leaves in the tree, which results in a representation power that is linear in the number of nodes.

To overcome this limit, we extend the budding tree to construct a distributed tree where the two children of a node are selected by different and independent gating functions. Observe that the source of locality in a budding tree comes from the convexity among leaves and selecting one child more means selecting the other child less. We relax this constraint in the distributed budding tree by untying the gating function that controls the paths for the two subtrees:

$$y_m(\boldsymbol{x}) = (1 - \gamma_m)\big[g_{ml}(\boldsymbol{x}) y_{ml}(\boldsymbol{x}) + g_{mr}(\boldsymbol{x}) y_{mr}(\boldsymbol{x})\big] + \gamma_m \rho_m \tag{7}$$

where $g_{ml}(\boldsymbol{x}) = \sigma(\boldsymbol{w}_{ml}^T \boldsymbol{x})$ and $g_{mr}(\boldsymbol{x}) = \sigma(\boldsymbol{w}_{mr}^T \boldsymbol{x})$ are the conditions for the left and right children and $\boldsymbol{w}_{ml}$, $\boldsymbol{w}_{mr}$ respectively are the untied linear split parameters of the node m for the left and right splits. Hence the conditions for left and right subtrees are independent—we get the budding (and soft) tree if $g_{mr}(\boldsymbol{x}) \equiv 1 - g_{ml}(\boldsymbol{x})$.

With this definition, a tree no longer generates local representations, but distributed ones. The selection of one child is independent of the selection of the other, and for an input, multiple paths can be traversed (see Figure 1). Intuitively, a distributed tree becomes similar to a *hierarchical sigmoid* layer as

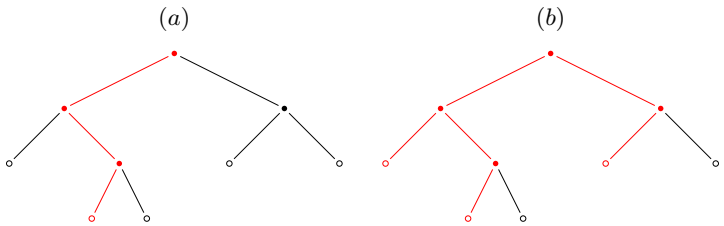

Fig. 1. (a) Operation of a budding tree which (softly) selects a single path across the tree. (b) Operation of a distributed tree where it can (softly) select multiple paths.

opposed to a hierarchical softmax. In the case of hard thresholds, more than one leaf node (path) can be selected for an instance, implying any one of 2^L possibilities where L is the number of leaves (as opposed to exactly one of L for the traditional tree). This results in a representation power exponential in the number of nodes.

The distributed budding tree still retains the hierarchy that exists in traditional hard/soft decision trees and budding trees. Essentially, each node m can still veto its entire downward subtree by not being activated. The activation of a node m means that there is at least one leaf in the subtree that is relevant to this particular input.

Previously we have also applied the same idea of continuous construction by softly adding/removing complexity to multilayer perceptrons and we have proposed two methods; in tunnel networks, it is done at the level of hidden units and in budding perceptrons, it works at the level of hidden layers [7].

4 Experiments

In this section, we report quantitative experimental results for regression, binary, and multiclass classification tasks.

We use ten regression (*ABAlone*, *ADD10*, *BOSton*, *CALifornia*, *COMp*, *CONcrete*, *puma8FH*, *puma8FM*, *puma8NH*, *puma8NM*), ten binary (*BREast*, *GERman*, *MAGic*, *MUSk2*, *PIMa*, *POLyadenylation*, *RINgnorm*, *SATellite47*, *SPAmbase*, *TWOnorm*) and ten multiclass (*BALance*, *CMC*, *DERmatology*, *ECOli*, *GLAss*, *OPTdigits*, *PAGeblock*, *PENdigits*, *SEGment*, *YEAst*) data sets from the UCI repository [2], as in [9]. We compare distributed trees with budding trees and the C4.5 for regression and classification tasks. Linear discriminant tree (LDT) which is a hard, multivariate tree [16] is used as an additional baseline for the classification tasks.

We adopt the following experimental methodology: 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. Hard trees (including the linear discriminant tree) use the validation set as a pruning set. Distributed and budding trees use the validation set to tune the

Table 1. Regression results.

	MSE			Size		
	C4.5	Bud	Dist	C4.5	Bud	Dist
ABA	54.13	41.61	41.79	44	35	**24**
ADD	24.42	4.68	**4.56**	327	35	**27**
BOS	34.21	21.85	**18.28**	19	19	33
CAL	31.18	24.01	**23.26**	300	94	**47**
COM	3.61	1.98	1.97	110	**19**	29
CON	26.89	15.62	**15.04**	101	38	43
8FH	41.69	37.83	37.89	47	**13**	21
8FM	6.89	5.07	**5.02**	164	17	15
8NH	39.46	**34.25**	34.53	77	**27**	43
8NM	6.69	3.67	3.61	272	37	37

learning rate 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 1 shows the mean squared errors and the number of nodes of the C4.5, budding tree, and distributed tree on the regression data sets. we see that the distributed trees perform significantly better on five data sets (*add10*, *boston*, *california*, *concrete*, *puma8fm*), whereas the budding tree performs better on one (*puma8nh*), the remainder four being ties. In terms of tree sizes, both the distributed tree and the budding tree have three wins (*abalone*, *add10*, *california* and *comp*, *puma8fh*, *puma8nh*, respectively), the remaining four are ties. Note that at the end of the training, because of the stochasticity of training, both distributed trees and budding trees have nodes that are almost leaf (having $\gamma \approx 1$). These nodes can be pruned to get smaller trees with negligible change in the overall response function.

Table 2 shows the percentage accuracy of C4.5, LDT, budding and distributed trees on binary classification data sets. In terms of accuracy, LDT has four wins (*german*, *pima*, *polyadenylation*, *twonorm*), distributed tree has five wins (*magick*, *musk2*, *ringnorm*, *satellite*, *spambase*) and the remaining one (*breast*) is a tie. On its four win, LDT produces very small trees (and on three, it produces the smallest trees). This suggests that with proper regularization, it is possible to improve the performance of budding and distributed trees. In terms of tree sizes, LDT has six wins (*breast*, *polyadenylation*, *ringnorm*, *satellite*, *spambase*, *twonorm*) and C4.5 has one (*german*) with the remaining three ties.

Table 3 shows the percentage accuracy of C4.5, LDT, budding and distributed trees on multiclass classification data sets. The distributed tree is significantly better on five data sets (*balance*, *dermatology*, *optdigits*, *pendigits*, *segment*), and the remaining five are ties. In terms of tree sizes, again LDT produces smaller trees with four wins (*balance*, *cmc*, *glass*, *optdigits*), and budding tree has one win (*pendigits*).

Table 2. Binary classification results

	Accuracy				Size			
	C4.5	LDT	Bud	Dist	C4.5	LDT	Bud	Dist
BRE	93.29	95.09	95.00	95.51	7	**4**	12	23
GER	70.06	**74.16**	68.02	70.33	**1**	3	42	39
MAG	82.52	83.08	86.39	**86.64**	53	38	122	40
MUS	94.54	93.59	97.02	**98.24**	62	11	15	35
PIM	72.14	**76.89**	67.20	72.26	8	5	68	35
POL	69.47	**77.45**	72.57	75.33	34	**3**	61	97
RIN	87.78	77.25	88.51	**95.06**	93	**3**	61	117
SAT	84.58	83.30	86.87	**87.91**	25	**9**	38	41
SPA	90.09	89.86	91.47	**93.29**	36	**13**	49	23
TWO	82.96	**98.00**	96.74	97.64	163	**3**	29	25

Table 3. Multiclass classification results

	Accuracy				Size			
	C4.5	LDT	Bud	Dist	C4.5	LDT	Bud	Dist
BAL	61.91	88.47	92.44	**96.36**	6	**3**	29	28
CMC	50.00	46.65	53.23	52.87	25	**3**	28	51
DER	94.00	93.92	93.60	**95.84**	16	11	11	20
ECO	77.48	81.39	83.57	83.83	10	11	24	58
GLA	56.62	53.38	53.78	55.41	21	**9**	21	55
OPT	84.85	93.73	94.58	**97.13**	121	**31**	40	92
PAG	96.72	94.66	96.52	96.63	24	29	37	27
PEN	92.96	96.60	98.14	**98.98**	170	66	**54**	124
SEG	94.48	91.96	95.64	**96.97**	42	33	33	76
YEA	54.62	56.67	59.32	59.20	25	22	41	42

5 Visualization

In this section, we visualize trees learned on a synthetic and real data.

Synthetic data is designed as a one-dimensional regression dataset where the response is a sinusoidal of the input with a small additive noise, and the input is limited to the interval $[-3, 3]$. 300 samples are drawn for both training and validation sets. Trees are displayed in Figure 2.

We observe that both type of trees use smoothly sloped sigmoid gating functions to interpolate between to endpoints to incrementally build up the sinusoidal shape. Since distributed trees have two (untied) gates, its decision nodes are able to (softly) split the space into not two but three regions. Therefore for some nodes we observe curves transitioning between three pieces, creating shapes with more than one break point, while using flat children.

In addition to synthetic data we use the MNIST handwritten digit database [11] to visualize trees as shown in Figure 3. Each node is visualized by the average value of all instances that fall into that node. In a hard tree, this is trivial to define, however in soft, budding and distributed trees every instance falls into every possible node with some nonzero value, however small. Therefore

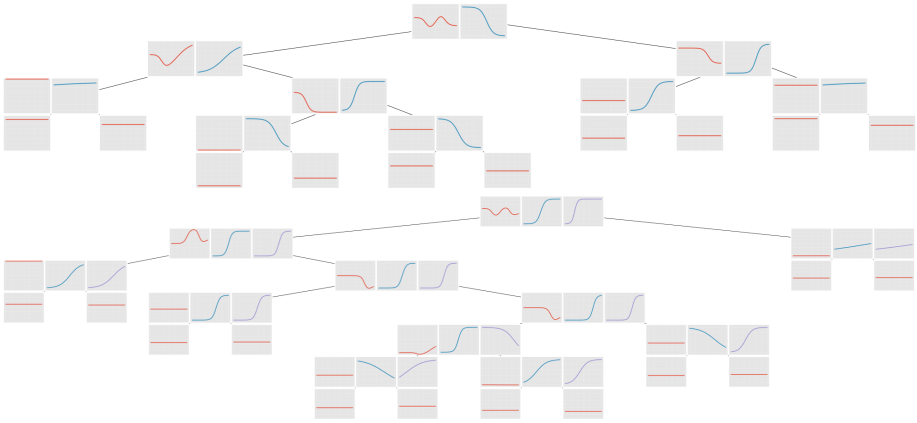

Fig. 2. Budding (top) and distributed (bottom) trees trained on toy data. Red curves display response by the subtree at that node. Blue curves display the gating function at that node. For distributed trees, blue and purple curves display (untied) left and right gating functions.

we compute a notion of a *soft membership* which is the multiplication of all gating values through the particular path, and use that to compute a weighted average instead. Additionally, distributed decision trees have a notion of *not taking either of the path to children*, finishing the computation at the given node as if it was a leaf. For each internal node, we measure this by multiplying the soft membership so far with $(1-g_{ml})(1-g_{mr})$, and display the resulting average right below each plot. Nodes also carry a transparency measured by the *non-leafness* of all the ancestors in the path $(1-\gamma_m)$ multiplied, since the more each ancestor tends to becoming a leaf the less impact the node has in the overall output.

For both trees we see interesting hierarchies: For instance, budding tree splits apart digit 4 from similar looking input, and then into two, digits of 7 and 9. Distributed tree has several variations of digit 3 spreading apart into branches from the same ancestor. Additionally, we see that distributed tree occasionally early stops at an internal node by shutting off both children, for instance digit 4 is recognized this way at the first level, below the root.

6 Conclusions

After reviewing soft and budding trees, we propose a distributed tree model that overcomes the locality of traditional trees and can learn distributed representations of the data. It does this by allowing multiple root-to-leaf path (soft) selections over a tree structure, as opposed to selection of a single path as done by budding (soft) and traditional (hard) trees. This increases their representational power from linear to exponential in the number of leaves. Quantitative

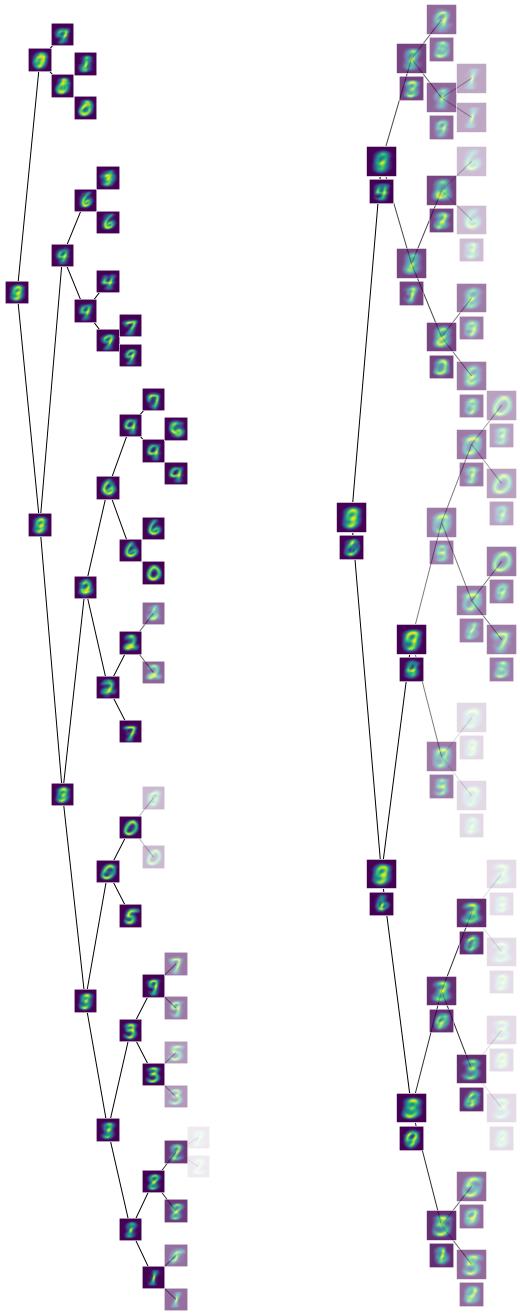

Fig. 3. Budding (left) and distributed (right) trees trained on MNIST, displayed vertically (root is at the left, growing towards the right). Each node displays the average input falling into that node (path), weighted by the soft memberships assigned by successive compositions of gating. Distributed trees show an additional image that belongs to neither left or right subtree, measured by how *closed* both gates are.

evaluation on several data sets shows that this increase is indeed helpful in terms of predictive performance.

Previously we have proposed using decision-tree autoencoders [6] and we believe that distributed trees too can be considered as alternative to layers of perceptrons for deep learning in that they can learn hierarchical distributed representations of the input in its different levels.

Even though the selection of left and right subtrees is independent in a distributed tree, it still preserves the hierarchy in its tree structure as in traditional decision trees and budding trees. This is because an activation close to zero for a node has the ability to veto its entire subtree, and an *active* node means that it believes that there is some relevant node down in that particular subtree. Visualizing the decisions and paths learned by soft and budding trees help towards the interpretation of the trained models.

References

1. Ahmetoğlu, A., İrsoy, O., Alpaydın, E.: Convolutional soft decision trees. In: Kurkova, V. (ed.) ICANN 2018, LNCS 11139, pp. 134–141. Springer (2018)
2. Bache, K., Lichman, M.: UCI machine learning repository (2013), http://archive.ics.uci.edu/ml
3. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth & Brooks, Pacific Grove, California (1984)
4. Guo, H., Gelfand, S.B.: Classification trees with neural network feature extraction. IEEE Transactions on Neural Networks **3**, 923–933 (1992)
5. Hancock, T., Jiang, T., Li, M., Tromp, J.: Lower bounds on learning decision lists and trees. Information and Computation **126**(2), 114–122 (1996)
6. İrsoy, O., Alpaydın, E.: Autoencoder trees. In: Asian Conference on Machine Learning (2015)
7. İrsoy, O., Alpaydın, E.: Continuously constructive deep neural networks. IEEE Transactions on Neural Networks and Learning Systems **31**(4), 1124–1133 (2020)
8. İrsoy, O., Yıldız, O.T., Alpaydın, E.: Soft decision trees. In: International Conference on Pattern Recognition (2012)
9. İrsoy, O., Yıldız, O.T., Alpaydın, E.: Budding trees. In: International Conference on Pattern Recognition (2014)
10. Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the EM algorithm. Neural computation **6**(2), 181–214 (1994)
11. LeCun, Y., Cortes, C.: The MNIST database of handwritten digits (1998)
12. Murthy, S.K., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Journal of Artificial Intelligence Research **2**, 1–32 (1994)
13. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
14. Rokach, L., Maimon, O.: Top-down induction of decision trees classifiers-a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **35**(4), 476–487 (2005)
15. Yıldız, O.T., Alpaydın, E.: Omnivariate decision trees. IEEE Transactions on Neural Networks **12**(6), 1539–1546 (2001)
16. Yıldız, O.T., Alpaydın, E.: Linear discriminant trees. International Journal of Pattern Recognition and Artificial Intelligence **19**(03), 323–353 (2005)