# Decision Trees in Computational Graph: Representation, Optimization and Extension

Zhang Jinxiong

November 27, 2019

**Abstract**

Decision tree looks like a simple computational graph without cycles, where only the leaf nodes specify the output values and the non-terminals specify their tests or computation. We will express decision trees in the language of computational graph. As shown, the decision tree is a shallow binary network.

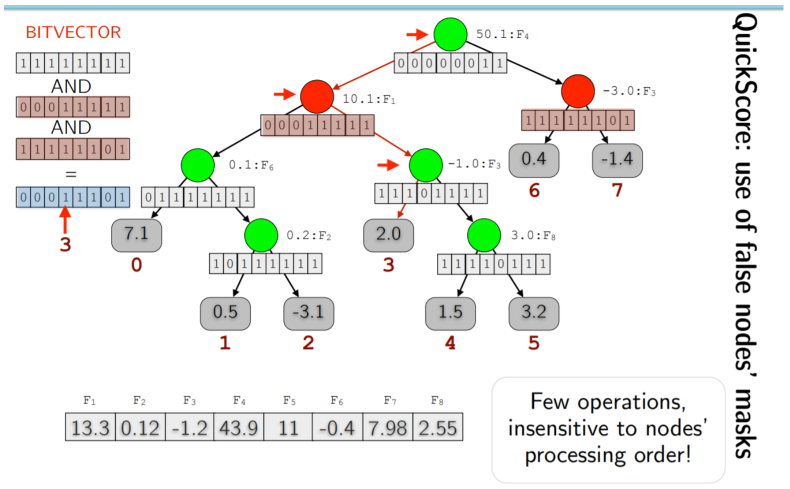
## 1 Introduction

Computational graph is the descriptive language of deep learning models. Christopher Olah summarized : "To create a computational graph, we make each of these operations, along with the input variables, into nodes. When the value of one node is the input to another node, an arrow goes from one to another. These sorts of graphs come up all the time in computer science, especially in talking about functional programs. They are very closely related to the notions of dependency graphs and call graphs. They are also the core abstraction behind the popular deep learning framework Theano."

In deep neural network, the operation of non-terminals is a smooth activation function such as sigmoid or 'ReLU' and it acts on every input in the element-wise sense. All inputs of the deep neural network share *the same depth and activation function* in computational graph. More complex architecture can include some 'feedback structure'.

In decision trees, the operation of non-terminal is a test function such as typical 'statistical test' and it determines the input next state - for example terminated or not. Different inputs have different depth and test functions in decision trees. The test function depends on the 'splitting criteria' when building a decision tree. In vanilla binary decision tree, the test function does not change the inputs and the final outputs of terminals depend on the labels of its instances.

The bitvector is originally adopted to improve the efficiency of the tree-based ranking model. It is first time to illustrate how to perform an interleaved traversal of the ensemble by means of simple logical bitwise operations as following according to the QuickScorer[1, 3, 4, 5] algorithm.



BITVECTOR
AND
AND
=
QuickScore: use of false nodes' masks
Few operations, insensitive to nodes' processing order!

Figure 1: Tree Traversal Example [1]

As shown, this traversal is of fewer operations and insensitive to nodes' processing order. It is the fewer operations that matters to improve the performance of software. Our motivation of the representation (4) is to describe the decision tree model in the language of computational graph so that we can take advantages of deep learning software packages to implement tree-based models. The nodes' processing order is our concern. We convert the logical bitwise operations to the matrix-vector multiplication. And we find the false nodes via direct mathematical operations in our novel representation. It is end-to-end as deep neural network. What is more, our representation is compatible with oblique decision tree as shown later. In theory, we introduce the bitvector matrix $B$. It is delicate that there is no columns with all elements equal to 1 in the bitvector matrix $B$. It is an open problem to find all the bitvector matrices of the given size decision trees. And we can use the bitvector matrix $B$ to describe the structure of decision tree.

In Deep Neural Decision Trees[8] soft binning function is used to make the split decisions. Typically, a binning function takes as input a real scalar $x$ and produces an index of the bins to which $x$ belongs. Yongxin Yang et al[8] propose a differentiable approximation of this function. Given the differentiable approximate binning function, it is to construct the decision tree via Kronecker product $\otimes$. They asserted that they can exhaustively find all final nodes by

$$f_1(x_1) \otimes f_2(x_2) \otimes \cdots \otimes f_D(x_D)$$

where each feature $x_d$ is binned by its own neural network $f_d(x_d)$. Our representation shares the same properties with that work: the model can be easily

[1]This figure is copied from the slides presented in SIGIR15 by Raffaele Perego.

implemented in neural networks tool-kits, and trained with gradient descent rather than greedy splitting. However, our representation is to describe the decision trees in the computational graph language precisely. It is not hybrid or analogous of deep neural network, which makes it different from others. And our representation is compatible with categorical feature as shown in the section *Categorical Features*. Even they are equivalent, our model is optimized in the way like binarized neural networks as shown later. We formulate the training of decision trees as a constrained continuous optimization problem although it is not entirely or elegantly solved.

In the following sections, a new representation of decision tree is proposed in the language of computational graph. It is shown that the decision tree is a shallow binary network. Thus we extend the field of computational graph beyond deep learning and acceleration techniques in deep learning may help to accelerate the tree-based learning algorithms. A specific example is to illustrate this representation. More characters of this representation are discussed in the decision tree structure matrix. The training of decision trees are formulated as constrained continuous optimization problem. We also propose the generalized decision trees and a novel ensemble way to composite generalized decision trees.

## 2 Parameterized Decision Trees: A New Representation

Different from building a decision tree, prediction of a decision tree is the tree traversal in essence. Inspired by QuickScorer, we split such prediction to the following stages. We only consider the numerical features for simplicity and categorical features are discussed later.

The first stage is to find the false nodes in the decision tree with respect to the input $x$:

$$h = \frac{\mathrm{Sign}(Sx - t) + 1}{2} = D(Sx - t) \tag{1}$$

where so-called 'selection matrix' $S \in \mathbb{R}^{n_L \times n}$ consists of one-hot row vector in $\mathbb{R}^n$ representing which feature is tested in a node; the elements of threshold vector $t \in \mathbb{R}^{n_L}$ are the optimal splitting points of each node associated with one feature; $Sign(\cdot)$ is the element-wise sign function and $Sign(0) = -1$;$D(x)$ is the binarized ReLU function defined by

$$D(x_i) = \begin{cases} 1, & \text{if } x_i > 0 \\ 0, & \text{otherwise} \end{cases}.$$

Here $n_L$ is the number of the non-terminals and $n$ is the dimension of input space. For convenience, we define that the node is a true node if its feature is greater than the splitting point otherwise the node is 'false' node.

The second stage is to find bitvector[2]of false nodes

$$H = B\ \mathrm{Diag}(h) \tag{2}$$

where $Diag(\cdot)$ maps a vector to a diagonal matrix; $B \in \mathbb{B}^{L\times n_L}$ is the bitvector matrix of the decision tree. Every column of $B$ is a bit-vector of node; the matrix $B\ \mathrm{Diag}(h)$ is the matrix multiplication product of matrix $B$ and $\mathrm{Diag}(h)$. Here $L$ is the number of terminal nodes(leaves).

The final stage is to determine the output

$$v[i], \quad i = \arg\max(Hp) \tag{3}$$

where $p = (1, 2, \cdots, n_L - 1, n_L)^T \in \mathbb{R}^{n_L}$ and $n_L$ is the number of the non-terminal nodes; $v[i]$ are the $i$th elements of vector $v$; $\arg\max(v)$ returns the index of the first maximum in the vector $v$. In fact the vector $p$ can be any positive vector such as $p = (1, 1, \cdots, 1, 1)^T \in \mathbb{R}^{n_L}$.

Note that the key is the index of the first maximum of the hidden state

$$\{B\,\mathrm{Diag}[\underbrace{D(Sx-t)}_{\text{binary vector}}]\}p$$

where if the test return true, the element of the binary vector is zero; otherwise, the element of the binary vector is 1. The 1s select the bitvectors of false nodes and 0s rule out the bitvectors of true nodes. If we change 1 to any positive number, the index of the first maximum of the hidden state does not change. Thus binarized ReLU is not the unique activation choice. For example, ReLU is an alternative to binarized ReLU in our representation. We will show more alternatives of binarized ReLU later. We can rewrite the decision tree in a more compact form:

$$T(x) = v[\arg\max(B\,\sigma(Sx-t))] \tag{4}$$

where the element $v[i]$ are the $i$th element of vector $v$; the function $\arg\max(x)$ returns the index of the first maximum in the vector $x$; the matrix $B$ is the bitvector matrix of the decision tree; the element-wise transformation $\sigma(\cdot)$ is the ReLU; the matrix $S$ is the selection matrix of the decision tree; the vector $x$ is the input data.

The function ReLU is used to express the regression trees and adaptive splines for a continuous response by statisticians such as Heping Zhang.

Note that $v[i] = \left\langle v, \vec{1}_i \right\rangle = \sum_{n=1}^{L} v[n]1_i(n)$ where $\vec{1}_i = (0, 0, \cdots, 1, 0, \cdots, 0)$ and $1_i(n) = 1$ if $n = i$ otherwise $1_i(n) = 0$. The operator $\arg\max$ is equivalent to choose an one-hot vector. Here we do not specify the data type or any requirement on the value vector $v$. The element $v[i]$ can be anything even function. Usually it is numerical for regression and categorical/discrete for classification.

Here is the hierarchical structure of decision tree for regression:

$$\begin{array}{lll} x & \to h & \to v[i] \\ \mathbb{R}^n & \to \mathbb{R}_+^{n_L} & \to \mathbb{R}. \end{array}$$

[2]Every non-terminal node $n$ is associated with a node bitvector (of the same length), acting as a bitmask that encodes (with 0's) the set of leaves to be removed from the subtree whenever $n$ is a false node. It is declared that Domenico Dato et al are the first to represent the non-terminal nodes as bitvectors.

Additionally, the element of value vector $v$ is categorical for classification trees.

It is really a shallow model. And its hidden layers are sparse binary.

Now there is nothing other than expressing the decision tree in the language of computational graph. It looks far from a step function. However, note that

- the matrices $S$ and $B$ are binary, i.e., their elements are 0 or 1.
- $\arg\max()$ only selects one element in this case.

All 'if-then' tests are transformed to numerical computation. And the test phase and the traversal phase are split and separated. In test phase, we obtain the hidden state $h$ dependent on the selection matrix $S$ and threshold vector $t$. In traversal phase, we get the hidden state $H$ dependent on the bitvector matrix $B$ and then arrive at the terminal node by multiplying the constant positive vector $p$. These two phases play different roles while they are connected and associated. All the input samples share the same computational procedure. This makes it more computation dense in prediction. The good news is that we can apply the high performance technology such as fast matrix multiplication to decision tree.

The tuple $(S, t, B, v)$ really determines a decision tree.

**Theorem 1** *The mapping $M : T \mapsto (S, t, B, v)$ is bijective.*

**Proof 2.1** *It is only necessary to prove that it is equivalent to the theorem in QuickScorer. The key idea is that the leftmost non-zero element in the result of logical 'AND' of bitvectors is the one which is not zero in every bitvector of its false nodes.*

This also provides some hints why the binary neural networks work if you believe in the reasonability of decision trees. In another hand, this representation tells us that the decision trees only have hidden layer with the binarized ReLU activation function. Why does it work so well? Where is the reasonability of decision trees? All reasonability may be our illusion as George Box's quotation "All models are wrong, but some are useful". Even so we shall not cease from exploration to truth for the honor of the human mind.

Can we find some invariants of the bitvector matrix $B$? Is there other efficient matrix representation of the decision tree structure? Is there an intuitive or physical interpretation of the bitvector matrix $B$? Is it just another example of unreasonable effectiveness of mathematics in data science? Why do we constraint the bitvector matrix in these constraints? How can we optimize the decision tree structure? There are many problems to solve.

## 3 An Example of Binary Decision Tree

Suppose the input variable $x$ has 4 numerical features, the learned decision tree is as shown in the following figure 2.

The threshold vector is $t = (1, 4, 3, 2, 5)^T$. The value vector is categorical or numerical, such as $v = (v_1, v_2, v_3, v_4, v_5, v_6)^T$. The selection matrix $S$ and bit-vector matrix $B$ is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

respectively.

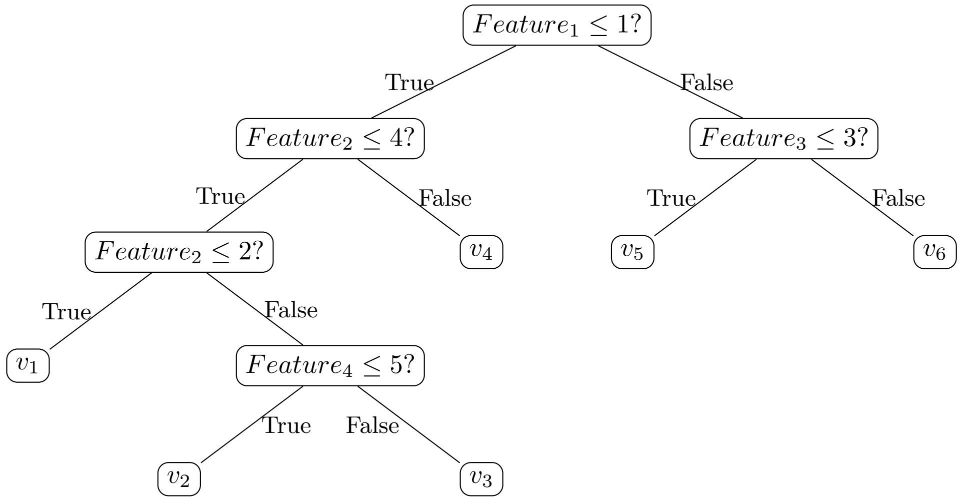


Figure 2: A Binary Decision Tree

Suppose input vector $x = (2, 1, 2, 2)^T$, then the output is given by

$$T(x) = v[\arg\max(B\,\sigma(Sx - t))] = v_5.$$

Here we do not specify the data types of the elements in the value vector. For regression, the value vector $v$ is numerical; for classification, it is categorical [3].

The first element of bitvector always corresponds to the leftmost leaf in the decision tree so that the algorithm return the first value when there is no false node for some inputs such as $(1, 1, 2, 3)^T$. And the order of bitvector is left to right, top to bottom.

[3]See more differences of classification trees and regression trees in Classification and regression trees.

## 4 Structures of Bitvector Matrices

Not all 0-1 matrix is the bitvector matrix of a decision tree. For example, the identity matrix cannot be a bitvector matrix of a decision tree because there are at most one 0 element 0f some columns of the bitvector matrix.

It is obvious that there are some inherent structures in the bitvector matrix $B$. There are four rules:

1. there are the most 0s in the bitvector of the root node.

2. there is only one 0 in the bitvectors of left-leaf-parent nodes.

3. the left nodes inherit the 1s from their ancestors.

4. the right nodes inherit the 0s from their parents and shift these 0s to 1s.

Here the node $n$ is defined as left-leaf-parent node if its left child node is a terminal node/leaf. And we can infer that there is at least one 0 in the bitvector of the root combining the first rule and the second rule. The number of zeroes in the bitvectors of the nodes tell how many terminal nodes/leaves are in the left-subtree of this node. The first and the second rule are intuitive and naive. The forth and fifth rules are to verify or prove.

If the bitvector matrix is given as following:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

we can recover the structure of the decision tree:

- from the first rule, we know the first column corresponds to the root node;

- from the second rule, we know the third, forth and fifth column correspond to left-leaf-parent nodes;

- from the third rule, we know the second, forth and fifth column correspond to left nodes of root;

- from the forth rule, we know the third column corresponds to the right node of root.

What is more, we can infer that the forth node is the left child of the second node because they share the same last three 1s; the fifth node is the right child of the fourth node because it inherits the first 1s from the fourth node and the fourth node is a left-leaf-parent node.

**Lemma 1** *It is a bijective mapping from the structure of the decision tree to the bitvector matrix $B$.*

Thus we also call the matrix $B$ as the structure matrix of the decision tree.

The number of columns $m$ is less than the number of rows in bitvector matrix $n$. For above example, $n-m=1$. In another word, the number of non-terminal nodes is less than the terminal nodes. In fact, this difference is always equal to 1 for all binary decision trees. Additionally, we will know the sizes of $S, t, B, v$ if the input data dimension and the number of terminal nodes are given.

As the decision tree shown in figure 2, the first node must be false and the third node must be true if arriving at the sixth leaf.

$$\begin{pmatrix}0\\0\\0\\0\\1\\1\end{pmatrix} \wedge \begin{pmatrix}1\\1\\1\\1\\0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\\0\\1\\1\end{pmatrix} - \{ \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}1\\1\\1\\1\\0\\1\end{pmatrix} \} = \begin{pmatrix}0\\0\\0\\0\\0\\1\end{pmatrix}$$

And the first node must be false and the third node must be true if arriving at the fifth leaf.

$$\begin{pmatrix}0\\0\\0\\0\\1\\1\end{pmatrix} \wedge \{ \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}1\\1\\1\\1\\0\\1\end{pmatrix} \} = \begin{pmatrix}0\\0\\0\\0\\1\\1\end{pmatrix} - \{ \begin{pmatrix}1\\1\\1\\1\\0\\1\end{pmatrix} - \{ \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}0\\0\\0\\0\\1\\1\end{pmatrix} \}\} = \begin{pmatrix}0\\0\\0\\0\\1\\0\end{pmatrix}$$

The first/second/forth node must be true if arriving at the first leaf.

$$\{ \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}0\\0\\0\\0\\1\\1\end{pmatrix} \} \wedge \{ \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}0\\0\\0\\1\\1\\1\end{pmatrix} \} \wedge \{ \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}0\\1\\1\\1\\1\\1\end{pmatrix} \} = \begin{pmatrix}1\\1\\1\\1\\1\\1\end{pmatrix} - \begin{pmatrix}0\\1\\1\\1\\1\\1\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\\0\\0\end{pmatrix}$$

It is observed that the logical and $\wedge$ is equivalent to rule out the leaves where we cannot arrive at some node. For example, if the root node is true, we cannot arrive at the fifth and sixth leaves which corresponds to the first term in the last equation.

As bitvector defined, we observe that every non-terminal node $n$ is associated with a node bitvector (of the same length) $c$, acting as a 'complement bitmask' that encodes (with 1's) the set of leaves potential to reach at its subtree whenever $n$ is a true node.

**Definition 1** *The complement $c$ of the bitmask bitvector $b$ is defined by*

$$b + c = \vec{1}.$$

Here $\vec{1}$ is a vector of given shape and type, filled with ones such as $(1,1,1)^T$.

**Definition 2** *The left reachable leaf set of the non-terminal node $n$ is defined as the leaves potential to reach whenever the node is false, denoted as $L(n)$*

**Lemma 2** *The left reachable leaf set of the non-terminal node $n$ only rely on its and its parent's bitvectors.*

**Proof 4.1** *Let $n, p, b_n, p_n$ denotes the node $n$, the parent node $p$, bitvector of $n$ and $p$, respectively. We prove it by induction.*

*If $n$ is the root, it is trivial.*

*If $n$ is the left child of the root, it is first to rule out the leaves which its parent cannot reach when its parent is true, i.e.,*

$$h = \vec{1} - p_n.$$

*Then it is time to remove the nodes which can reach when it is false, i.e.,*

$$h - \{b_n - p_n\} = \vec{1} - p_n - \{b_n - p_n\} = \vec{1} - b_n.$$

*If $n$ is the right child of the root, it is first to rule out the left reachable leaf set $L(p)$ of the root, denoted by $h$. Then it is time to remove the nodes when the node cannot reach when it is false, i.e.,*

$$h - \{b_n - p_n\}.$$

*Here $h$ is determined by this procedure recursively.*

Thus we can find the sets leaves potential to reach whenever the node is true or false.

**Lemma 3** *The augmented matrix $(B\ \vec{1})$ is invertible.*

**Proof 4.2** *It is only necessary to prove that we can arrive at any leaf/terminal node by elementary column transformation rather than the logical 'And' $\wedge$ operation. The difference of two bitvector of the matrix $B$ means the intersection when the given bitvectors are false nodes. Because each leaf belongs to its parent's left reachable leaf set or corresponds to its parent's complement vector. According to the above lemma, it is equivalent to differences of the bitvectors and $\vec{1}$.*

**Lemma 4** *The bitvector matrix is column full-rank as well as its complement bitvector matrix.*

It is easy to prove the following lemma by contradiction.

**Lemma 5** *The bitvector $b$ cannot coincide with its complement bitvector $c$ in the bitvector matrix of a decision tree.*

We can arrange the bitvector of non-terminals in the pre-order as in the binary tree traversal for clarity.

It is not supersizing that there is an example that the following 0-1 matrix is not a structure matrix of a decision tree:

$$\begin{pmatrix}0&0&0&0&1\\0&0&0&0&0\\1&1&0&0&1\\1&1&0&0&1\\1&0&1&0&1\\1&1&1&1&1\end{pmatrix}.$$

Given the number of non-terminals $n_L$, we can compute the number of the structure matrices of decision trees denoted by $\mathcal{M}(n)$. If $n_L = 2$, $\mathcal{M}(2) = 2$. If $n_L = 3$, $\mathcal{M}(3) = 5 = 2 \times 3 - 1$. If $n_L = 4$, $\mathcal{M}(4) = 18 = 4 \times 5 - 2$. If $n_L = 5$, $\mathcal{M}(5) =?$. It is clear that $\mathcal{M}(n_L + 1) < \mathcal{M}(n_L) \times (n_L + 1)$.

In general, the bitvector matrix as well as its complement is the representation of ordered binary tree data structure. Binary tree is referred to a tree if every node has only two children. Ordered tree means that the tree is not oriented.

The tree traversal is in the language of matrix computation. There are more works to translate the operator of binary tree into matrix computation. The adjacency matrix of graph cannot describe the ordered structure of decision tree. Dekel Tsur gives a data structure that stores an ordered tree $T$ with $n$ nodes and degree distribution. The literatures mentioned there is a good guide to design a succinct data structure that stores a tree. RapidScorer introduces a modified run length encoding called **epitome** to the bitvector representation of the tree nodes. Guofang Wei coach a team study the invertibility probability of binary matrices Thomas Voigt and Gunter M. Ziegler show that the edges of the hyperplanes spanned by random 0/1-vectors are equivalent to bounds on e the probability that a random 0/1-matrix. Seth Pettie applies the forbidden 0-1 matrices to search Tree and path compression-based data structures. For more information on 0-1 matrix . see the conference Coimbra Meeting on 0-1 Matrix Theory and Related Topics.

Given the number of internal nodes, can we find all the possible structure of decision trees? Given a bitvector matrix $B$, can we find the selection matrix $S$ and threshold vector $t$?

If the answers to above questions are all 'Yes', we only need to specify the number of internal nodes and then all the structure of decision tree can be found by a greedy way.

What is the relation of this matrix and the methods to train a decision tree?

## 5 Optimization of Decision Trees

Even we can find all the structure of the decision trees given the number of internal nodes, there are still selection matrix $S$, threshold vector $t$ and value vector $v$ to optimize.

Ishwar K. Sethiand Jae H. YOO use a neural learning scheme combining backpropagation and soft competitive learning to simultaneously determine the splits for each node of a tree structure in fixed size without any splitting function. binarized neural networks are neural networks with binary weights and activations at run-time. XNOR-net[7] shows some methods to training binary-weights-networks. Coarse gradient descent method is designed for learning sparse weight binarized activation neural networks. Blended coarse gradient descent[9] is designed for full quantization of deep neural networks. Payne proposed a way to construct optimal binary decision trees in 1977. Optimal decision trees mix optimization methods and decision trees. Dimitris Bertsimas and Romy Shioda Sicco Verwer and Yingqian Zhang Dimitris Bertsimas and Jack Dunn Hèlène Verhaeghe et al Oktay et al Kristin P. Bennettand Jennifer A. Blue Sanjeeb Dash, et al Sanjeeb Dash et al Sicco Verwer and Yingqian Zhang Murat Firat et al formulate the optimization of decision tree as constraint optimization problem. Nina Narodytska et al develop a SAT based model for computing smallest-size decision trees given training data. And these methods can find optimal linear-combination (oblique) splits[4]for decision trees. Mohammad Norouzi et al formulate a convex-concave upper bound on the tree's empirical loss and use stochastic gradient descent to train decision trees. Alternative to greedy splitting including almost all Optimal Classification Trees is based on the classic representation of decision tree, i.e., the sum of indicator functions. It is almost impossible to check all the non-greedy training methods of decision trees. There is the conference CPAIOR – Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming–interested in such topic.

Different from our representation of decision tree, Mohammad Norouzi et al introduce a tree navigation function rather than the bitvector matrix $B$. And optimal classification tree encodes the information of structure into Mixed-Integer-Optimization formulation. As known, it is first to describe the structure of decision tree via the bitvector matrix.

[4]In our representation, linear-combination (oblique) splits correspond to real rows of the selection matrix $S$. In another word, the selection matrix $S$ can be any matrix in $\mathbb{R}^{N_L \times p}$ for oblique decision trees.

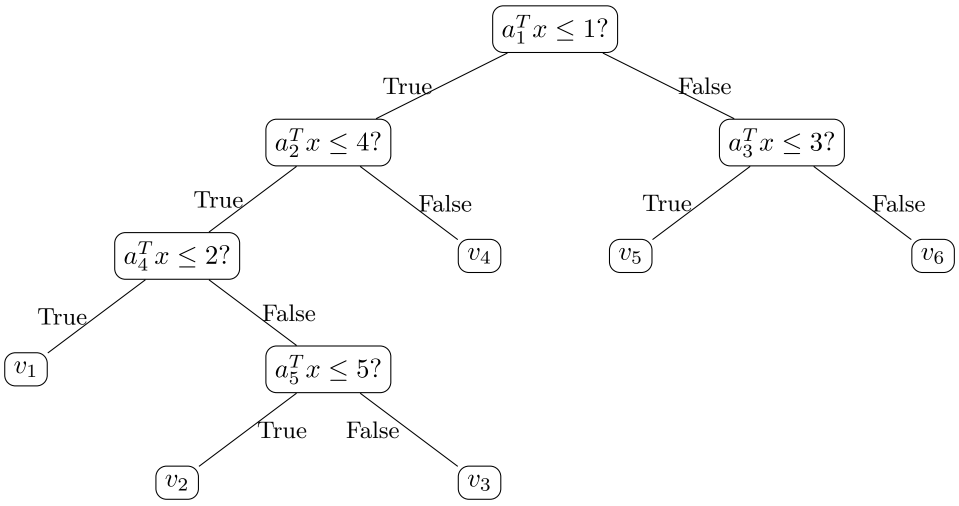
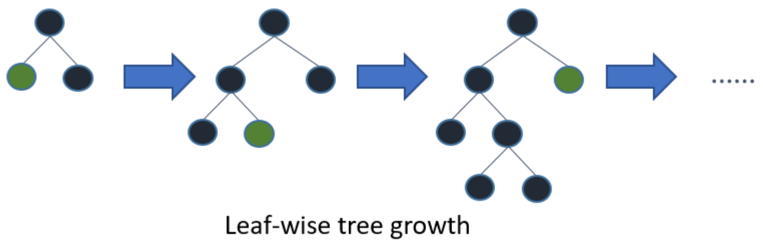


Figure 3: A Toy Example of Oblique Decision Trees

Our aim is to build a binary decision tree in the case where only the number of internal nodes and the dimension of input data are specified. So that the structure and splitting points are learnt from the input samples. In another word, the representation tuple $(S, t, B, v)$ is learnt from the input samples. The difficult is the restriction of bitvector matrix. The bitvector matrix $B$ is not only '0-1' valued but also subject to the four rules in above section.

A basic idea is to limit the bitvector matrix $B$ to some special form such as oblivious decision trees. In another word, we optimize the splitting points $S, t, v$ of the decision tree with pre-specified structure $B$. And we can optimize the traversal parameters $B, v$ when the test parameters $S, t$ are given. The problem is how to find an alternative minimization methods to find the optimal tree. Another idea is to find the whole space that the four rules describe. It is clear that the space governed by the four rules is not a linear space so that it is not possible to find the bases to generate all the bitvector matrices. What is worse, it is not convex.

A leaf-wise tree growth is proposed in LightGBM[2].



It inspires us to optimize only one node in each step in our framework.

First, we initialize a non-zero bitvector of the root node according to the first rule and the other bitvectors are zero vectors as in the coordinate optimization method. In this step, we only can split the data to the left or the right of the root node. And then we can optimize the first column vector of bitvector matrix $B$, the first row of selection matrix $S$ and the first element of the threshold vector $t$ by minimizing the 'splitting criteria' as ordinary decision tree growth. For example, if the number of terminal nodes is 6 and input samples have 4 numerical features, the first step is given by

$$\arg\min_{B,S,t,v} \sum_{i=1}^{N} \ell(y_i, T(x_i)) \tag{5}$$

where $\ell(\cdot,\cdot)$ is the loss function;$\{(x_i, y_i) \mid i = 1, 2, \cdots, N\}$ is the training data set;

$$T(x) = v[\arg\max(\{B \ \ \mathrm{Diag}[D(Sx - t)]\}p)];$$

and

$$S = \begin{pmatrix} s_1 & s_2 & s_3 & s_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ b_2 & 0 & 0 & 0 & 0 \\ b_3 & 0 & 0 & 0 & 0 \\ b_4 & 0 & 0 & 0 & 0 \\ b_5 & 0 & 0 & 0 & 0 \\ b_6 & 0 & 0 & 0 & 0 \end{pmatrix}, t = \begin{pmatrix} t_1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{pmatrix}.$$

Here $s_1, \cdots, s_4, t_1, v_1, \cdots, v_6 \in \mathbb{R}$ and $b_2, \cdots, b_6 \in \{0, 1\}$. Note that the first element of bitvector always corresponds to the leftmost leaf in the decision tree so that there is one 0 in the first column of the matrix $B$. We take advantages of numerical optimization methods to solve the problem (5).

Second, we fix the optimal bitvector of root node and initialize the second column of the bivector according to four rules, which correspond to the first left node of the root node. Other bitvectors are set to be zeroes. At the same time, we optimize the second row of selection matrix $S$ and the second element of the threshold vector $t$ by minimizing the 'splitting criteria'. In this step, the data split into the left nodes of root can be split into the right/left nodes of second node. We continue the example in the first step. It is still given by (5) but we fix the optimal parameters $s_1^*, \cdots, s_4^*, b_2^*, \cdots, b_6^*, t_1^*$ returned by the first step:

$$S = \begin{pmatrix} s_1^* & s_2^* & s_3^* & s_4^* \\ s_1 & s_2 & s_3 & s_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 0 & b_1 & 0 & 0 & 0 \\ b_2^* & b_2 & 0 & 0 & 0 \\ b_3^* & b_3 & 0 & 0 & 0 \\ b_4^* & b_4 & 0 & 0 & 0 \\ b_5^* & b_5 & 0 & 0 & 0 \\ b_6^* & b_6 & 0 & 0 & 0 \end{pmatrix}, t = \begin{pmatrix} t_1^* \\ t_2 \\ 0 \\ 0 \\ 0 \end{pmatrix}, v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{pmatrix}.$$

It is a greedy way to test all legal inheritors of the root node $(0, b_2^*, b_3^*, b_4^*, b_5^*, b_6^*)^T$ according to the four rules. So we can elect only one legal inheritors of the root node.

**Algorithm 1** Optimization of Oblique Decision Trees

```
1: Input training data set $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathbb{R}, n = 1, 2, \cdots, N\}$
   and the number of terminal nodes $L$
2: for $n = 1, 2, \ldots, L - 1$ do
3:     Elect a legal bitvector of $n$th non-terminal node according to the four
       rules: $b_n \in \mathbb{R}^L$.
4:     Initial the associated row of slection matrix, the $n$th element of threshold
       vector $t$, the value vector, i.e., $s_n$, $t_n$, $v$.
5:     Fix the $b_1, \cdots, b_{n-1}$ of bitvetor matrix $B$, $s_1, \cdots, s_{n-1}$ of slection matrix,
       $t_1, \cdots, t_{n-1}$ of threshold vector $t$ and set others equal to 0s.
6:     Update the parameters
           $$b_n, s_n, t_n, v = \arg\min_{B,S,t,v} \sum_{i=1}^{N} \ell(y_i, T(x_i)).$$
7: end for
8: Output $B, S, t, v$.
```

Other steps are similar to the second step until all the parameters are found.

Note that the value vector $v$ is determined in the last step. We can optimize this procedure by using the outputs of each step. For example, we have known that input samples are split in the left or the right of the root, we only use the input samples in the left of the root to optimize the parameters of the second node in the second step. In another word, we can reduce the computation of cost function $\sum_{i=1}^{N} \ell(y_i, T(x_i))$. It is not always necessary to optimize the decision tree according to all the training data at each step like in random forests. composite decision tree.

It is mentioned 'as only one-step optimal and not overall optimal'. It is 'block coordinate descent'. Such tree growth method is a modification of the top-down growth method. The main drawback of this optimization is that we cannot optimize bitvector matrix $B$. We elect legal bitvector of non-terminal nodes according to the four rules rather than update the column vectors of bitvector matrix $B$ by gradient-based optimization method. However, we can run the algorithm 1 many times to find the suboptimal structure. Another drawback is that the cost function is not regularized.

## 6 Categorical Features

James Dougherty et al found that the performance of the Naive-Bayes algorithm significantly improved when features were discretized using an entropy-based method. Hybrid Decision Tree simulates human reasoning by using symbolic learning to do qualitative analysis and using neural learning to do subsequent quantitative analysis. Cheng Guo and Felix Berkhahn embed the categorical features to Euclidean spaces. CatBoost[6] is a new open-sourced gradient boosting library that successfully handles categorical features, where "we perform a random permutation of the data set and for each example we compute average label value for the example with the same category value placed before the given one in the permutation". CORELS (Certifiable Optimal RulE ListS) is a custom discrete optimization technique for building rule lists over a categorical feature space. This algorithm provides the optimal solution, with a certificate of optimality. It is always to take different measures to deal with qualitative and quantitative features.

Categorical features are usually stored in different data format from the numerical features in computer, which can provide some qualitative information for prediction. The domain/value set of categorical feature is discrete and usually finite. Note that there are no arithmetic operations of categorical features. The only function we can act on categorical features is to test whether it is of one specific type. That is a 'yes-no' question.

Because we cannot add up the numerical features and categorical features directly, we should split embedding representation of categorical features and numerical features in oblique decision tree while there is no problem to test categorical or numerical features in vanilla decision tree [5].

After embedding the categorical features into real vectors, we solve the following optimization problem in vanilla decision trees with categorical features:

$$\arg\min_{B,S,t,v}\sum_{i=1}^{N}\ell(y_i,T(x_i))$$

where $S$ is the one-hot row vector matrix and $B$ is the bitvector column matrix constrained by the the four rules . The threshold value of categorical feature is also limited in the embedded (discrete) space.

As mentioned in the previous section, it is the constraints on the bitvector matrix $B$ that are not convex. Note that the non-negative vector $s \succeq 0$ is one-hot if and only if $\|s\|_2^2 = \langle s, 1\rangle = 1$. We add such convex constraints to the cost function in the optimization problem (5) and obtain:

$$\begin{aligned}&\arg\min_{B,S,t,v}\sum_{i=1}^{N}\ell(y_i,T(x_i))\\&\text{subject to } s_i\in\mathbb{R}^n_+, \|s_j\|_2^2 = \left\langle s_j, \vec{1}\right\rangle = 1 \quad \forall j\in\{1,\cdots,n_L\}.\end{aligned}\tag{6}$$

Similar to the algorithm [1], we propose the algorithm [2] to train decision trees with categorical features.

Now we define the oblique decision trees with categorical features. As mentioned above, it is not possible to add up the numerical features and categorical features directly. For example, there is no meaning of the sum that equals to your age and gender. They introduce the dummy variables(sometimes called

[5]The combination of categorical features are discussed in CatBoost. However, it is Cartesian product of some domains rather than linear combination of different features.

**Algorithm 2** Optimization of Decision Trees with Categorical Features

```
1: Input training data set $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathbb{R}, n = 1, 2, \cdots, N\}$
   and the number of terminal nodes $L$
2: for $n = 1, 2, \ldots, L-1$ do
3:     Elect a legal bitvector of $n$th non-terminal node according to the four
       rules
           $$b_n \in \mathbb{R}^L.$$
4:     Initial the associated row of slection matrix, the $n$th element of threshold
       vector $t$, the value vector, i.e., $s_n$, $t_n$, $v$.
5:     Fix the $b_1, \cdots, b_{n-1}$ of bitvetor matrix $B$, $s_1, \cdots, s_{n-1}$ of slection matrix,
       $t_1, \cdots, t_{n-1}$ of threshold vector $t$ and set others equal to 0s.
6:     Solve the optimization problem (6).
7: end for
8: Output $B, S, t, v$.
```

indicator variable) in statistics and quantitative economy community such as Dummy-Variable Regression. Economics Wiki explains that "This indicator variable takes on the value of 1 or 0 to indicate the availability or lack of some effect that would change the outcome of whatever is being tested." If we regard the linear combination of numerical features as classic linear regression to test some properties of input sample, we generalize the linear regression to dummy-variable regression or regression discontinuity design(RDD). Thus we can deal with the categorical features in oblique decision trees without embedding:

$$T(x) = v[\arg\max(B\,\sigma(f(x)))]$$

where $f(x) = (f_1(x), f_2(x), \cdots, f_{n_L}(x))^T$ and $f_i(x)$ is the dummy-variable regression.

In classification, the element of the value vector $v$ is categorical. It is not suitable to find the optimal $v$ via gradient-based methods. Like in classic decision tree growth, we use the label of the instances of $v_i$ to find the best value of $v_i$. We can apply the loss function for multi-label classification to decision tree if we adjust the form of output $v$.

## 7 Relaxed Optimization of Decision Trees

We have the following necessary conditions of bitvector matrix $B$:

1. $b_{m,i}^2 = b_{m,i} \in \mathbb{R}$, $b_m = (b_{m,1}, \cdots, b_{m,L})^T$;
2. $\|b_m\|_0 = \|b_m\|_1 = \|b_m\|_2^2 = \left\langle b_m, \vec{1} \right\rangle = f(m)$;
3. $f(1) \leq f(m) \in \{1, 2, \cdots, n_L\} \quad \forall m$;

4. $1 \leq \langle b_m, b_n \rangle \leq n_L - 1, \|b_m - b_n\|_2^2 > 0 \quad \forall m \neq n.$

Here $b_m, b_n$ are the columns of bitvector matrix and $m, n \in \{1, 2, \cdots, n_L\}$. And we assume that $b_1$ is the bitvector of the root node. The first condition constraint the elements in the binary set $\{0, 1\}$. The second condition is equivalent to the first condition. The last conditions are according to the four rules. And we require that the bitvector matrix $B$ satisfies the following condition:

$$Rank(B) = n_L, Rank(B\ \vec{1}) = L = n_L + 1.$$

There are some redundancies in these conditions. For example, the second condition is the necessary for the first one. The problem is the rank condition. The invertible 0-1 matrix may have some inherent characteristics.

We formulate decision tree training as the following optimization problem:

$$\begin{aligned}
&\min_{B,S,t,v} \sum_{i=1}^{N} \ell(y_i, T(x_i)) \\
&\text{subject to} \\
&s_j \in \mathbb{R}_+^n, \|s_j\|_2^2 = \left\langle s_j, \vec{1} \right\rangle = 1 \quad \forall j \in \{1, \cdots, n_L\}; \\
&b_{m,i}^2 = b_{m,i}, \forall i \in \{1, 2, \cdots, L\},\ \|b_m\|_1 = \|b_m\|_2^2 = \left\langle b_m, \vec{1} \right\rangle \leq f(m), \\
&\ f(m) \in \{1, 2, \cdots, n_L\},\ \forall m \in \{1, 2, \cdots, n_L\}; \\
&1 \leq \langle b_m, b_n \rangle \leq n_L - 1,\ \forall m \neq n \in \{1, 2, \cdots, n_L\}; \\
&Rank(B) = n_L, Rank(B\ \vec{1}) = n_L + 1; \\
&t \in \mathbb{D}_1 \times \cdots \mathbb{D}_{n_L - c} \times \mathbb{R}^c \text{ if } Sx \in \mathbb{D}_1 \times \cdots \mathbb{D}_{n_L - c} \times \mathbb{R}^c.
\end{aligned} \tag{7}$$

Here $\ell(\cdot, \cdot)$ is the loss function; $T(x) = v[\arg\max(B\,\sigma(Sx - t))](4)$; $\sigma(\cdot)$ is the ReLU function. $S = (s_1, \cdots, s_{L-1})^T$, $B = (b_1, \cdots, b_{L-1})$ and $\mathbb{D}_1, \cdots \mathbb{D}_{n_L - c}$ are embedded (discrete) spaces. The last condition is data type constraints. The training data set is $\{(x_n, y_n) \mid x_n \in \mathrm{X} \subset \mathbb{R}^p, y_n \in \mathrm{Y} \in \mathbb{R}\}$ for $n = 1, 2, \cdots, N$.

It is really a hybrid optimization: equality, inequality, rank constrained. It is too complex and weird to describe the structure matrix space governed by the four rules.

Can we apply blended coarse gradient descent to update all the parameters of the decision tree? If the matrix $B$ is not a bitvector matrix of a decision tree, does this expression make some sense?

## 8 Super Models of Decision Trees

Before answering the question in the previous section, we revisit the generalization of decision trees. Probal Chaudhuri et al propose generalized regression trees by blending tree-structured nonparametric regression and adaptive recursive partitioning with maximum likelihood estimation. Torsten Hothorn et al embed tree-structured regression models into a well defined theory of conditional inference procedures. Xiaogang Su et al propose a method of constructing regression trees within the framework of maximum likelihood. Wei-Yin Loh surveys the developments and briefly reviews the key ideas behind some of the modern major decision tree algorithms. Richard A. Berk overviews the decision tree algorithms in the monograph *Statistical Learning from A Regression Perspective*. Recently, Lisa Schlosser et al unify three popular unbiased recursive partitioning approaches. These generalization focus on the statistical inference.

In computer science community, hybrid decision trees integrate decision trees with other algorithms such as Hybrid Decision Tree, A hybrid decision tree classifier, A Hybrid Decision Tree/Genetic Algorithm. Archana Panhalkar and Dharmpal Doye elaborate the various approaches of converting decision tree to hybridized decision tree.

In this section, we will show how to generalize the parameterized decision trees.

There are many activation functions in deep neural network such as ReLU, parametric ReLu, Swish, Mish. There are a partial list of activation functions in deep neural network at Simon's blog and a more comprehensive list of activation functions at Wikipedia. They are motivated initially to solve the vanishing gradient problem and boost the stability and robust generalization. In our models, the activation function is non-differential binarized ReLU, a modified sign function. If we want to diversify the activation functions of decision trees, these activation functions may help. Any cumulative distribution function $\sigma$ is an approximation of the binarized ReLU if its support is $(0, \infty)$ so that if $x \leq 0$ then $\sigma(x) = 0$ otherwise $\sigma(x) > 0$ and $\lim_{x\to\infty} \sigma(x) = 1$.

Now we generalize the bitvector matrices to all 0-1 matrices and the vector $p$ to a real vector variable ,i.e., $p \in \mathbb{R}^{n_L}$, and use a softmax function to output the leaf distributions. For regression task, we obtain the following generalized decision tree:

$$T(x) = \langle v, \operatorname{softmax} B\, \sigma(Sx - t)\rangle \tag{8}$$

where the vector $v$ is the numerical value vector; the function softmax returns a discrete probability distribution as approximation to one-hot vector; the matrix $B$ is a 0-1 matrix; the element-wise transformation $\sigma(\cdot)$ is the activation function; the matrix $S$ is a real matrix associated with the decision tree; the vector $x$ is the input data; the vector $t$ is real.

This representation is the minimal extension of decision trees, where we set the bitvector free from the four rules. If the bitvector is extended to real vector space, the decision tree is sparse in the new representation. If we still keep the bitvector in binary space, the decision tree is still common in our new representation. The softmax function is used to normalize the outputs as one probability distribution. That is also we do not restrict $p$ in the positive real vector set. We convert the tree traversal to its stochastic version. This extension includes the decision tree. In another word, our new representation is the super set of decision tree.

Eric Jang et al. present an efficient gradient estimator that replaces the non-differentiable sample from a categorical distribution with a differentiable sample from a novel Gumbel-Softmax distribution. Such method helps us to design alternatives of the softmax.

In this setting, it is analogous to the shallow neural network. If the bitvector matrix is the gene of decision tree, this setting is the revolutionary mutation and numerical optimization is genome editing to train these models. It is a dilemma of dimension: more outputs nodes, higher accuracy and less outputs nodes, less training cost.

All the operators are common in deep learning: matrix multiplication, ReLU and softmax. There are one affine layer, a nonlinear layer and a softmax layer. In some sense, decision trees are special type neural networks. We call those models 'generalized decision trees'. This representation is 'mixed-precision' because only $B$ is 0-1 valued [6] and other parameters are real.

We focus on the update formula of binary matrix $B$. Now we take a thought experiment, what does it mean when one bit changes into 1 from 0 when updating the binary matrix? If the gradient is 0, nothing change after updating the binary matrix $B$. In anther word, the gradient is not 0 if one bit changes into 1 from 0 when updating the binary matrix. The problem is how to binarize the gradients. If the gradient is close to 0, it is not supposed to update one bit from 1 to 0 or reverse. If the gradient is larger than 1, it is supposed to update the bit from 1 to 0 or reverse. Another scheme is to train the model via gradient-based or other numerical optimization methods, and after that binarize the bitvector matrix $B$. It is also the case when we binarize the deep neural network models. It is possible to apply the coarse gradient descent method to update the binary matrix $B$.

Our framework can include more hybrid trees. For example, the tests are still linear functions, univariate or multivariate. We can replace them with nonlinear functions $f = (f_1, \cdots, f_{n_L})^T$:

$$T(x) = v[\arg\max(B\,\sigma(f(x)))].$$

If we still obey the four rules, it is still a decision tree type algorithms. If we set the bitvector free from the four rules, it is a generalized decision tree with nonlinear test function. Thus we could apply kernel tricks to decision tree.

The output also can substitute with nonlinear functions $f = (f_1, \cdots, f_L)^T$.

$$\begin{aligned} T(x) &= \langle \mathrm{softmax}(B\,\sigma(Sx - b)), f(x)\rangle \\ &= \sum_{i=1}^{L} e_i f_i(x). \end{aligned}$$

Here $(e_1, \cdots, e_l)^T = \mathrm{softmax}(B\,\sigma(Sx - b))$. These are "Multivariate Adaptive Regression Splines".

[6] In vanilla binary decision tree, the selection matrix is also 0-1 valued. Here we focus on oblique decision trees.

## References

[1] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *Acm Transactions on Information Systems*, 35(2):1–31.

[2] Guolin Ke, Qi Meng, Thomas William Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tieyan Liu. Lightgbm: a highly efficient gradient boosting decision tree. pages 3149–3157, 2017.

[3] Francesco Lettich, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Gpu-based parallelization of quickscorer to speed-up document ranking with tree ensembles. 1653, 2016.

[4] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Quickscorer: A fast algorithm to rank documents with additive ensembles of regression trees. pages 73–82, 2015.

[5] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Quickscorer: Efficient traversal of large ensembles of decision trees. pages 383–387, 2017.

[6] Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *arXiv: Learning*, 2017.

[7] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv: Computer Vision and Pattern Recognition*, 2016.

[8] Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv: Learning*, 2018.

[9] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Blended coarse gradient descent for full quantization of deep neural networks. *arXiv: Learning*, 2018.