Bivariate Decision Trees: Smaller, Interpretable, More Accurate

Rasul Kairgeldin
Dept. Computer Science & Engineering
University of California, Merced
Merced, CA, USA 95343
rkairgeldin@ucmerced.edu

Miguel Á. Carreira-Perpiñán
Dept. Computer Science & Engineering
University of California, Merced
Merced, CA, USA 95343
mcarreira-perpinan@ucmerced.edu

Abstract

Univariate decision trees, commonly used since the 1950s, predict by asking questions about a single feature in each decision node. While they are interpretable, they often lack competitive predictive accuracy due to their inability to model feature correlations. Multivariate (oblique) trees use multiple features in each node, capturing high-dimensional correlations better, but sometimes they can be difficult to interpret. We advocate for a model that strikes a useful middle ground: bivariate decision trees, which use two features in each node. This typically produces trees that not only are more accurate than univariate trees, but much smaller, which offsets the small increase in node complexity and keeps them interpretable. They also help data mining by constructing new features that are useful for discrimination, and by providing a form of supervised, hierarchical 2D visualization that reveals patterns such as clusters or linear structure. We give two new algorithms to learn bivariate trees: a fast one based on CART; and a slower one based on alternating optimization with a feature regularization term, which produces the best trees while still scaling to large datasets.

1 Introduction

In many ways, decision trees stand alone in the statistical machine learning literature. They use conditional computation by design: an input instance follows a single root-leaf path, without using the rest of the tree parameters, which makes inference time extremely fast. They handle the multiclass case directly, without the need for one-vs-all or one-vs-one approaches, as each leaf can be labeled with a particular class. Finally, decision trees are perhaps most valued because of their interpretability, together with a handful of models (such as logistic regression, generalized additive models and scorecards). The way they achieve a prediction follows a sequence of simple questions about the input features which is quite close to human reasoning. A decision tree can be explained and audited at a global level by simple inspection.

The one serious disadvantage of decision trees is that they usually result in low predictive accuracy. There are two reasons for that. The first one is *a poor data model*. A typical decision tree is of the axis-aligned or univariate type. It means that each decision node or split operates on a single input feature (e.g. "if $x_7 > 10$ go right, otherwise go left"), which generally limits total number of features used by a tree. Moreover, correlated features cause tree to grow large, which complicates interpretability.

The second reason is *a poor optimization*. A decision tree can be seen as a piecewise constant approximation, which makes optimization of a loss function over the tree a nonconvex problem. A conventional greedy recursive partitioning does not optimize a global objective function, rather it searches for tree structures by optimizing local splits exactly. A recent algorithm, Tree Alternating

Interpretable AI: Past, Present and Future Workshop at NeurIPS 2024

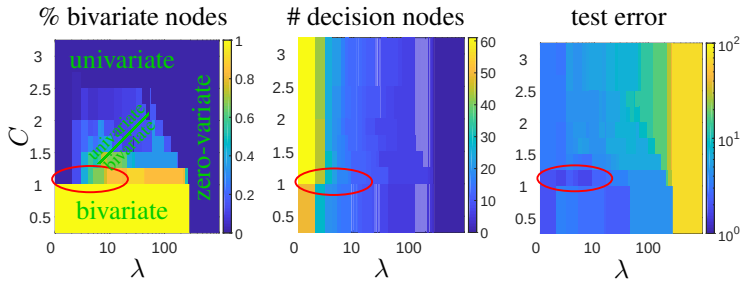


Figure 1: Phase diagram (λ, C) for the Segment dataset. We plot: the proportion of bivariate vs univariate decision nodes (indicating the regions of pure zero-, uni- and bivariate trees); the number of decision nodes; and the test error (%). The ellipse indicates the region of best-error trees.

Optimization (TAO) [7], has made it possible to train univariate and (sparse) oblique trees effectively (which were never successful with CART-like procedures). Unlike recursive partitioning, TAO operates on a well-defined parametric tree model and objective function. At each iteration, it updates each node's parameters so the objective decreases. However, a univariate tree (even trained by TAO) is still a poor data model in many cases, while a (sparse) oblique tree can sometimes be hard to interpret if it uses many features in each node.

Our goal here is to design decision trees that are more accurate than univariate trees while remaining highly interpretable and efficient to train. We propose *bivariate trees*, where each decision node has zero, one or two features at most. Allowing for two features rather than one significantly increases the predictive accuracy and decreases the tree size even more significantly, improving interpretability, as shown in our experiments.

We believe bivariate trees strike a good tradeoff that can make them very practical. *How interpretable are bivariate trees?* the ability to learn oblique splits even with two features greatly reduces the number of nodes and depth of the tree compared to a univariate one, while improving accuracy. It can also happen that the bivariate split can be understood as a new, meaningful feature on its own right. Finally, bivariate trees bring another advantage over univariate trees: we can show a 2D scatterplot at each decision node on its two selected features, which behaves like a *hierarchical 2D linear discriminant analysis*. This shows information between-class (how specific classes are separated from each other), as well as within-class (such as clustering or linear structure).

More detailed literature review on tree-based approaches and other pairwise-interaction models can be found in Appendix (sections A.2 and A.1). Our experiments in section 3 compare univariate, bivariate and oblique trees in terms of accuracy, model size and interpretability.

2 Learning bivariate trees

Consider a K-class problem with dataset of size N with D-dimensional input features $\{(\mathbf{x}_n, y_n)\}_{n=1}^N \subset \mathbb{R}^D \times \{1, \dots, K\}$. Consider a binary decision tree (each decision node has exactly 2 children) with a set of decision nodes $\mathcal{N}_{\text{dec}}$, a set of leaves $\mathcal{N}_{\text{leaf}}$, and $\mathcal{N} = \mathcal{N}_{\text{dec}} \cup \mathcal{N}_{\text{leaf}}$. We define a routing function in each decision node $i \in \mathcal{N}_{\text{dec}}$ as $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$ which sends a sample $\mathbf{x}$ to either its left or right child. We use bivariate decision nodes where routing function makes hard decisions $f_i(\mathbf{x}; \boldsymbol{\theta}_i) = \texttt{left}_i$ if $w_{ij}x_j + w_{ik}x_k + b_i < 0$, otherwise $\texttt{right}_i$, and the learnable parameters are $\boldsymbol{\theta}_i = \{\mathbf{w}_i, b_i\}$, where $\|\mathbf{w}_i\|_0 \leq 2$ ensures splits of no more than 2 features. Each leaf $i \in \mathcal{N}_{\text{leaf}}$ contains a constant label classifier that outputs a single class $c_i \in \{1, \dots, K\}$. We collectively define the parameters of all nodes as $\boldsymbol{\Theta} = \{(\mathbf{w}_i, b_i)\}_{i \in \mathcal{N}_{\text{dec}}} \cup \{c_j\}_{j \in \mathcal{N}_{\text{leaf}}}$. The predictive function of the entire tree $T(\mathbf{x}; \boldsymbol{\Theta})$ guides a sample $\mathbf{x}$ along a single path from the root through a sequence of bivariate decision nodes to exactly one leaf, which provides the classification output.

We consider the following objective function over all the parameters of a tree of given structure, where $L(\cdot, \cdot)$ is the 0/1 loss:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(y_n, T(\mathbf{x}_n; \boldsymbol{\Theta})) + \lambda \sum_{i \in \mathcal{N}_{\text{dec}}} \phi(\mathbf{w}_i) \quad \text{s.t.} \quad \begin{cases} \|\mathbf{w}_i\|_0 \leq 2, \\ i \in \mathcal{N}_{\text{dec}} \end{cases} \tag{1}$$

and we introduce the following, new type of regularization:

$$\phi(\mathbf{w}_i) = \begin{cases} C, & \text{if } \|\mathbf{w}_i\|_0 = 2 \\ \|\mathbf{w}_i\|_0, & \text{if } \|\mathbf{w}_i\|_0 < 2. \end{cases} \tag{2}$$

The *feature cost* regularization term (2) is critical in *allowing not just bivariate splits but also univariate ones and pruning nodes*. It imposes a cost of 0, 1 or C for each zero-, uni- or bivariate node (figure 1).

2.1 The better algorithm: bivariate TAO

In equation (1), the loss function is additively separable over the training samples, and the regularization term is additively separable over the nodes. This renders it suitable for alternating optimization over the tree nodes (on a tree of given structure), which results in monotonic descent and convergence in a finite number of iterations. The idea is based on two theorems: separability condition and reduced problem (RP) over a node. First, define the *reduced set* $\mathcal{R}_i \subset \{1, \dots, N\}$ as the training instances that reach the node $i \in \mathcal{N}$. **Separability condition** implies that equation (1) can be separated and optimized over parameters of any non-descendant nodes (located on the same depth) independently and in parallel. This is a result of tree making hard decisions ($\mathcal{R}_i \cap \mathcal{R}_j = \emptyset$ for any non-descendant nodes i and j). **Reduced problem over a node** states that optimizing equation (1) over parameters of the given node $i \in \mathcal{N}$ reduces to simpler, well-defined problem involving its reduced set $\mathcal{R}_i$. RP is different for decision nodes and leaves:

Leaf Equivalent to optimizing the top-level objective (1) over parameter c_i on $\mathcal{R}_i$. Exact solution: the majority class of the samples in $\mathcal{R}_i$: $c_i = \text{argmax}_{k \in \{1,\dots,K\}} \sum_{n \in \mathcal{R}_i} L(y_n, k)$.

Decision node The objective (1) can be equivalently reduced to the following $0/1$ *loss binary classification problem*:

$$E_i(\mathbf{w}_i, b_i) = \sum_{n \in \mathcal{R}_i} L(\bar{y}_n, f_i(\mathbf{x}_n; \mathbf{w}_i, b_i)) + \lambda \, \phi(\mathbf{w}_i) \text{ s.t. } \|\mathbf{w}_i\|_0 \leq 2 \tag{3}$$

where L is the $0/1$ loss and $\bar{y}_n \in \{\texttt{left}, \texttt{right}\}$ is a pseudolabel assigned to training instance x_n to indicate the child that yields a lower loss value. The loss is computed by propagating $\mathbf{x}_n$ down the corresponding child.

2.1.1 Solving the reduced problem over a decision node

The problem can be solved exactly in $\mathcal{O}(N^3 D^2)$ by enumerating every possible split (= linear dichotomy on the N instances) over all $\binom{D}{2}$ feature combinations. Unfortunately, this is very costly. We propose a faster, approximate solution which shows good results on practice.

Solution 1: L_{biv} (bivariate solution) of eq. (3) s.t. $\|\mathbf{w}_i\|_0 = 2$, $b_i \in \mathbb{R}$ is achieved at $\boldsymbol{\theta}_i^{\text{biv}} = \{\mathbf{w}_i^{\text{biv}}, b_i^{\text{biv}}\}$. Define $\mathbf{W} \in \mathbb{R}^{2 \times H}$ is a small set of line orientations sampled uniformly by rotating it around the origin. For each point in the reduced set we project all pairwise feature combinations onto line orientations $\mathbf{X}_i^{\text{biv}} = \mathbf{X}_i \mathbf{S} \mathbf{W}$ where $\mathbf{X}_i \in \mathbb{R}^{|\mathcal{R}_i| \times D}$ are points in the reduced set $\mathcal{R}_i$, and $\mathbf{S} \in \mathbb{R}^{D \times 2}$ is a selection matrix for feature combinations. Vectorization allows a GPU utilization for faster computation. The solution can be computed using thresholding over features of $\mathbf{X}_i^{\text{biv}}$. The resulting bias can be interpreted as an optimal split of the points projected onto selected orientation $\mathbf{w}_l$ to minimize number of misclassified instances in $\mathcal{R}_i$. We repeat this process for each pair of features and find best solution $\mathbf{w}_i^{\text{biv}}$, b_i^{biv}, where $\mathbf{w}_i^{\text{biv}}$ is a sparse vector. Pseudocode is in Fig. 7 and an illustration in Fig. 6.

Solution 2: L_{univ} (univariate solution) of eq. (3) s.t. $\|\mathbf{w}_i\|_0 = 1$, $b_i \in \mathbb{R}$ is achieved at $\boldsymbol{\theta}_i^{\text{univ}} = \{(0, w_i^{\text{univ}})^T, b_i^{\text{univ}}\}$. It is computed simply by thresholding over original features. For the decision node $i \in \mathcal{N}_{\text{dec}}$ algorithm selects one feature to split points in $\mathcal{R}_i$ in order to minimize eq. (3).

Solution 3: L_0 (zero-variate solution) of eq. (3) s.t. $\|\mathbf{w}_i\|_0 = 0$, $b_i \in \{-1, +1\}$ is achieved at $\boldsymbol{\theta}_i^0 = \{\mathbf{0}, b_i^0\}$. This indicates that all samples in $\mathcal{R}_i$ are sent to the left ($b_i^0 = -1$) or the right ($b_i^0 = 1$).

Table 1: Comparison between bivariate, univariate and oblique trees for classification. Each tree was selected by cross-validating its hyperparameter and this was repeated 3 times over random training/validation sets. We report training and test accuracy (% ±stdev); average depth Δ, node count n and number of features f per node (we omit $f = 1$ and 2 for univariate and bivariate trees, respectively); and average runtime (seconds or "timeout"). We indicate with color green best and blue second best result for test accuracy and node count over the univariate and bivariate trees (ignoring the oblique trees). Almost without exception across several datasets of varying types, our bivariate TAO tree is both most accurate and smallest compared to any other univariate tree or bivariate tree. *The reduction in depth and number of nodes, and the corresponding simplification of the tree, is drastic, typically several times fewer nodes* (17 times smaller on the MiniBooNE dataset).

Dataset		bivariate			univariate		oblique
		TAO	CART	BiDT	CART	C5.0	TAO
House 16H (11464,16,2)	training (%)	87.1±1.55	89.45±0.00	90.42±0.23	86.2±0.0	91.98±0.55	86.55±1.10
	test (%)	85.6±0.07	84.73±0.05	85.6±0.17	83.4±0.0	83.06±0.32	85.47±0.51
	Δ/#nodes/f	7/35	10/107	10/115	8/75	15.05/245	4/13/14.9
	runtime (s)	30	21	15	0.2	0.1	24
Electricity (32702,8,2)	training (%)	98.97±2.80	95.80±0.80	96.14±1.20	99.10±0.00	95.04±0.43	98.1±1.8
	test (%)	89.38±0.12	86.05±0.05	87.91±0.06	87.80±0.16	88.64±0.42	90.23±0.19
	Δ/#nodes/f	23.0/1083	24.0/1741	22.3/1881	30.0/6366	17.25/2615	10/249/6.8
	runtime (s)	300	81	393	0.9	0.9	134
MiniBooNE (62048,50,2)	training (%)	92.36±0.00	96.02±0.02	-	96.61±0.02	95.88±0.07	91.98±0.15
	test (%)	91.16±0.00	90.68±0.03	-	90.25±0.03	89.84±0.10	91.43±0.12
	Δ/#nodes/f	11.0/105	15/831	-	19.3/2012	15.65/1787	10/133/16.8
	runtime (s)	1200	1000	timeout	5.2	6.4	3000
SUSY (600000,18,2)	training (%)	80.71±0.00	81.35±0.00	-	81.45±0.00	80.90±0.00	81.10±0.00
	test (%)	79.51±0.00	79.01±0.00	-	78.90±0.00	79.10±0.00	80.3±0.00
	Δ/#nodes/f	17.0/1077	21/2780	-	24/4389	16.25/3227	12/983
	runtime (s)	≈2h	≈1h	timeout	40.2	35.2	≈2h

The solution of the RP can be summarized as follows:

$$\boldsymbol{\theta}_i^* = \begin{cases} \boldsymbol{\theta}_i^{\text{biv}}, & \text{if } L_{\text{biv}} + \lambda C < \min(L_{\text{univ}} + \lambda, L_0) \\ \boldsymbol{\theta}_i^{\text{univ}}, & \text{if } L_{\text{univ}} + \lambda < \min(L_{\text{biv}} + \lambda C, L_0) \\ \boldsymbol{\theta}_i^0, & \text{if } L_0 \leq \min(L_{\text{biv}} + \lambda C, L_{\text{univ}} + \lambda) \end{cases}$$

We break the ties always in favor of a model with lower number of parameters. The separability condition allows us to optimize E in parallel for the parameters of any group of nodes that do not have a parent-child relationship starting from deepest one and proceeding to the root in reverse BFS order, while keeping the parameters of the remaining nodes fixed. Some decision nodes will opt for not using a feature, hence being redundant (since it directs all input instances to the same child) and candidates for eventual removal in a postprocessing step at the end of the training. This means that bivariate TAO indeed achieves pruning, or equivalently learns the tree structure (subject to being a subset of the initial tree's structure).

2.1.2 Regularization path

For K-class classification problem with N training samples define $N_1 \geq N_2 \geq \cdots \geq N_K$, where N_1 is a number of samples in the most populous class. Considering that empty feature solution of the RP we can derive $\lambda\phi(\mathbf{w}_i) \geq L_0 - L_{\text{biv}} \in \{0, 1, \ldots, N - N_1\}$. The term $\lambda\phi(\mathbf{w}_i)$ can be interpreted as the maximum allowed number of misclassified samples by a decision node. Knowing the minimum difference in $0/1$ loss between empty feature solution and current best solution (univariate or bivariate) for each decision node we can calculate next significant λ in regularization path. This way full regularization path can be computed much faster.

We provide details on computational complexity of the algorithm as well as its extension for CART and regression task in Appendix (section A.3.1, section C.4 and section C.3)

3 Experiments

We provide all necessary details about hyperparameters, implementations of different algorithms and additional experiment results etc. in the appendix. We present quantitative analysis in Table 1, Table 2 and Table 3 as well as tree size and performance comparison relative to training set size in figure 2.s

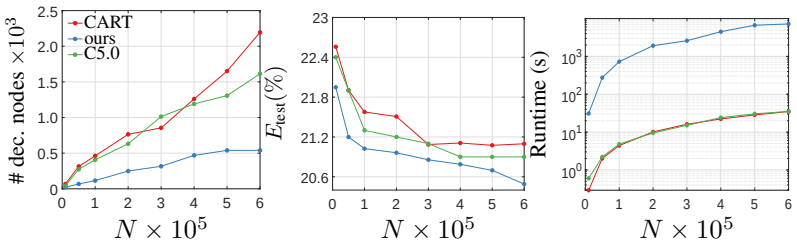


Figure 2: Number of nodes, test error and training time for univariate CART and C5.0 and bivariate TAO trees as a function of the sample size (subsampled from the SUSY dataset). Univariate trees (CART, C5.0) grow in size proportionally to the sample size, indicating a suboptimal training and pruning. Bivariate TAO tree slowly grows in size but eventually stops, indicating it has reached a sufficient model size

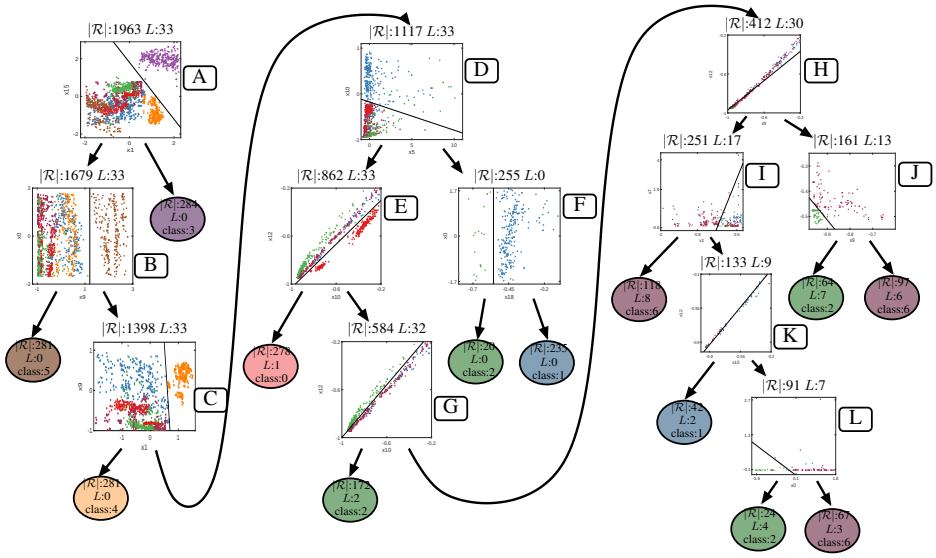


Figure 3: A bivariate TAO tree on the Segment dataset, plotted in 3 columns so it fits. Each decision node shows a scatterplot of its instances onto its two features. Each node shows the number of training instances reaching it ($|\mathcal{R}|$ and their 0/1 loss (L).

3.1 Segment dataset

Consider now fig. 3, which shows a bivariate TAO tree on the Segment dataset, a standard UCI benchmark for classification. It has 1963 instances training, 19 continuous features and 7 classes (balanced). Each class is an original color image and each feature is a local descriptor based on 3×3 patches (listed in the appendix, table 5). The bivariate tree has 25 nodes (12 decision nodes, 13 leaves), depth 10, and a training and test error of 1.53% and 2.59%, respectively. The univariate CART tree with lowest test error (not shown) is much bigger: 128 nodes, depth 15 and test error 3.99%. If we tune CART post-pruning to achieve 25 nodes, the error increases to 7.5%. Inspecting the bivariate tree reveals a wealth of information about the data, as we discuss next. Such insight would not be possible with black-box models such as forests or neural nets.

Global level structure The tree is heavily lopsided, having depth 10 but only 13 leaves. This means for most decision nodes one or both children is a leaf, so the IF-THEN-ELSE rules corresponding to the tree are almost perfectly tail-recursive. This makes them more interpretable, and faster at inference (many leaves have low depth).

A consequence of the lopsided structure is that several classes, accounting for over 50% of the dataset, are processed in a cascade way, and the tree makes early decisions for them (with very short rules). For example, class 3 is decided at the root (node A) after a single bivariate decision, with zero training error. Classes 5, 4 and 0 are also decided in a single root-leaf path each after 2, 3 and 5

decision nodes (one of them univariate), respectively, with near-zero training error. Classes 1, 2 and 6 are harder to separate and require deeper rules.

The tree has performed a significant *feature selection*: it uses only 10 features (0 1 5 6 7 9 10 12 15 18) out of the total 19. Since the tree has 12 decision nodes, it could use up to 24 distinct features, but it uses fewer. This is driven by the optimization we apply, which globally updates all the tree parameters. In contrast, a univariate tree with 12 decision nodes would be forced to use at most 12 features no matter what, because it can have only one in each node. In order to use sufficiently many features, a univariate tree has to be overly deep, which limits their interpretability.

Local level structure We show a scatterplot at each decision node projecting the instances reaching it onto its two features and showing the boundary of the split. The scatterplots show the power of bivariate splits (most obviously in nodes E, G, J, etc.). Using univariate splits would require a deep sequence of splits. More importantly, *the scatterplots provide a form of supervised dimensionality reduction, like a linear discriminant analysis but tree-structured, which is very helpful for visualization*. It is also directly interpretable because the projection is directly on two original features (rather than a complex function of all the features). For example, nodes C and E show a distinct cluster structure in classes 4 and 0, respectively. Although we were not able to obtain original images of Segment dataset, we conjecture that these clusters might correspond to objects of different color. Nodes E, G and K all use a bivariate split on features (10,12) and show a clear linear, parallel structure over multiple classes. Features 10 and 12 correspond to average amount of red and green color of the region. Bivariate split can be interpreted as a mixture of these two colors. Samples of class 0 in node E can be separated from other samples by measuring the amount of this mixture and thresholding it using bias.

All nodes use bivariate splits except nodes B and F, which use a univariate split. For plotting purposes only (so we can show a 2D scatterplot), we include feature 0 (region centroid column) as the vertical axis in those nodes. This has the serendipitous effect of showing that all classes show a linear structure along feature 0.

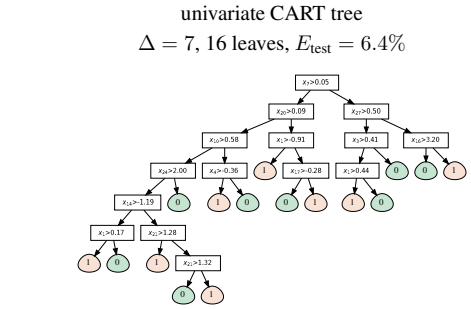
Note how some bivariate splits are able to separate out one class from the others cleanly (e.g. nodes A, B, C and E). But, in general, this is not possible with a linear decision boundary and it requires further partitioning down the tree. For example, nodes D and E show a sequence of two splits that separate class 0 from the rest. Thus, we should generally expect that a given decision node will have some classes straddling both sides of the boundary.

The bivariate decision functions can usually be understood as new, meaningful features. For example, split in node D can be interpreted as a combination of average amount of red color in the region and a contrast of adjacent pixels. We assume it can help to distinguish presence of vertical edges of specific color tone. Some features are repeatedly used (sometimes the same pair of features), indicating their relevance to separating some group of classes. For example, features 1 (row of the center pixel of the region) and 9 (average intensity of the region) are critical to separate cleanly first class 3 and then class 4 from the rest. This make sense since it indicates position of the 3 by 3 patch and its color the intensity. Some features are used only once, deep down the tree and affecting a small subset of instances, indicating they are needed to make a fine distinction. For example, features 0 and 6 appear only once and together in the deepest node (L). Some bivariate splits could be turned into univariate ones (eg nodes A and C) but at a higher loss. For example making univariate split at node A results in 15 more misclassified points.

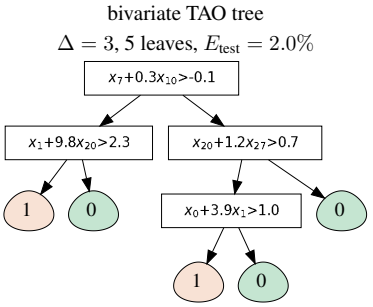
3.2 Breast Cancer

This is a binary classification task into malignant and bening tumors. Each input instance contains 30 features (listed in appendix table 4) extracted from a collection of cells, specifically, geometric features about size, shape, etc. (measured from snake-generated cell nuclei boundaries). Each such feature is very informative on its own and can be readily identified and understood by a radiologist.

Fig. 8 (appendix) shows how our bivariate TAO trees dominate the univariate CART trees by a significant margin over the range of tree sizes. Fig. 4 shows the best trees in terms of cross-validation. The univariate CART tree has depth 7, 16 leaves and 6.4% error. The bivariate TAO tree has depth 3, 5 leaves and 2% error. Not only is the bivariate much more accurate, but its small size makes it easier to understand by simple inspection. It results in only 5 rules vs 16 rules for the univariate tree.



1: **if** $(x_7 > 0.05)$ & $(x_{20} > 0.09)$ & $(x_{10} > 0.58)$ & $(x_{24} > 2.00)$ & $(x_{14} > -1.19)$ & $(x_1 > 0.17)$ **then** PREDICT 1
2: **if** $(x_7 > 0.05)$ & $(x_{20} > 0.09)$ & $(x_{10} > 0.58)$ & $(x_{24} > 2.00)$ & $(x_{14} > -1.19)$ & $(x_1 \leq 0.17)$ **then** PREDICT 0
3: **if** $(x_7 \leq 0.05)$ & $(x_{27} \leq 0.50)$ & $(x_{16} \leq 3.20)$ **then** PREDICT 1
4: **if** $(x_7 \leq 0.05)$ & $(x_{27} \leq 0.50)$ & $(x_{16} > 3.20)$ **then** PREDICT 0
5: ...//12 MORE RULES...



1: **if** $(x_7 + 0.3x_{10} > -0.1)$ & $(x_1 + 9.8x_{20} > 2.3)$ **then** PREDICT 1
2: **if** $(x_7 + 0.3x_{10} > -0.1)$ & $(x_1 + 9.8x_{20} \leq 2.3)$ **then** PREDICT 0
3: **if** $(x_7 + 0.3x_{10} \leq -0.1)$ & $(x_{20} + 1.2x_{27} \leq 0.7)$ **then** PREDICT 0
4: **if** $(x_7 + 0.3x_{10} \leq -0.1)$ & $(x_{20} + 1.2x_{27} > 0.7)$ & $(x_0 + 3.9x_1 > 1.0)$ **then** PREDICT 1
5: **else** PREDICT 0

Figure 4: Best univariate CART tree and bivariate TAO tree with their sets of rules (Breast Cancer dataset).

Global level structure The tree structure shows the bivariate tree can represent the whole dataset only in 5 very simple IF-THEN-ELSE rules. It is notable that the tree performs a significant feature selection. It uses only 6 features (out of 30). Three of them concern the *radius* measurements of nuclei boundaries (mean value, largest value and standard error), and obviously capture *size* information about the cells. Two of them concern *contour concavities* (mean value and standard error), and obviously capture *shape* information about the cells (essentially, whether they are circle-like or have indentations). The last feature is the texture (mean value) of the cell nuclei. The tree ignores features about smoothness of the contour, area of the cells, perimeter of the cell contour, fractal dimension, etc. This makes sense since some of those are likely correlated with size and shape as given by the radius and concavity, and thus are redundant.

Local level structure At a local level in the tree, we can look at specific nodes. The pairwise combination of features learned in each decision node can be seen as a new, constructed feature, which is quite meaningful and further improves interpretability. For example, the root of the tree (which uses as features the largest cell radius and the mean concavity) can be understood as detecting a cell collection where the cell nuclei boundaries are irregular or unusually large. The root's right child uses as features the standard error of radius and concavity, which means it detects high variance in the shape and size of the cell collection (as opposed to a collection that has uniform shape and size). Taking these two nodes together, the tree predicts malignancy (rightmost path). This can be done with the other 4 leaves, each of which is a population (malignant or benign) characterized by a short path or rule involving very few constructed features that are meaningful. As another example, the deepest decision node classifies the cell collection as malignant based on a combination of average size and texture variance irregularities.

4 Conclusion

We have proposed bivariate decision trees as a practically useful tradeoff between univariate trees and oblique trees. They are highly interpretable because they use two features at most in each decision node, unlike oblique trees, which use all or many features. Compared to univariate trees, bivariate trees are much smaller but significantly more accurate. They also can reveal insights about the data by constructing new, bivariate features that are useful for discrimination; and by providing a form of supervised, hierarchical 2D visualization at each decision node, which reveals patterns in the data such as clusters or linear structure. To learn a bivariate tree, we have given two algorithms. One is very fast, based on greedy recursive partitioning. The other is slower but produces much better trees, by using alternating optimization of the loss and the node features' cost globally over all the tree parameters.

References

[1] *Advances in Neural Information Processing Systems (NEURIPS)*, volume 35, 2022. MIT Press, Cambridge, MA.

[2] D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, July 2017.

[3] J. C. Bioch, O. van der Meer, and R. Potharst. Bivariate decision trees. In J. Komorowski and J. Zytkow, editors, *Proc. European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD 1997)*, pages 232–242, 1997.

[4] F. Bollwein and S. Westphal. A branch & bound algorithm to determine optimal bivariate splits for oblique decision tree induction. *Applied Intelligence*, 51(10):7552–7572, Oct. 2021.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001.

[6] L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.

[7] M. Á. Carreira-Perpiñán and P. Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pages 1211–1221. MIT Press, Cambridge, MA, 2018.

[8] M. Á. Carreira-Perpiñán and A. Zharmagambetov. Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting. In *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, pages 35–46, Seattle, WA, Oct. 19–20 2020.

[9] M. Á. Carreira-Perpiñán, M. Gabidolla, and A. Zharmagambetov. Towards better decision forests: Forest Alternating Optimization. In *Proc. of the 2023 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'23)*, pages 7589–7598, Vancouver, Canada, June 18–22 2023.

[10] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 785–794, San Francisco, CA, Aug. 13–17 2016.

[11] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, and P. J. Stuckey. MurTree: Optimal decision trees via dynamic programming and search. *J. Machine Learning Research*, 23(26):1–47, 2022.

[12] M. Gabidolla and M. Á. Carreira-Perpiñán. Pushing the envelope of gradient boosting forests via globally-optimized oblique trees. In *Proc. of the 2022 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'22)*, pages 285–294, New Orleans, LA, June 19–24 2022.

[13] M. Gabidolla and M. Á. Carreira-Perpiñán. Optimal interpretable clustering using oblique decision trees. In *Proc. of the 28th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2022)*, pages 400–410, Washington, DC, Aug. 14–18 2022.

[14] M. Gabidolla, A. Zharmagambetov, and M. Á. Carreira-Perpiñán. Improved multiclass AdaBoost using sparse oblique decision trees. In *Int. J. Conf. Neural Networks (IJCNN'22)*, Padua, Italy, July 18–22 2022.

[15] M. Gabidolla, A. Zharmagambetov, and M. Á. Carreira-Perpiñán. Beyond the ROC curve: Classification trees using Cost-Optimal Curves, with application to imbalanced datasets. In *Proc. of the 41th Int. Conf. Machine Learning (ICML 2024)*, Vienna, Austria, July 21–27 2024.

[16] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? In *Advances in Neural Information Processing Systems (NEURIPS)* neu [1], pages 507–520.

[17] T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Number 43 in Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York, 1990.

[18] R. Kairgeldin, M. Gabidolla, and M. Á. Carreira-Perpiñán. Adaptive softmax trees for many-class classification. In *Proc. of the 40th Conf. Uncertainty in Artificial Intelligence (UAI 2024)*, Barcelona, Spain, July 15–19 2024.

[19] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. In I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 3146–3154. MIT Press, Cambridge, MA, 2017.

[20] M. Lichman. UCI machine learning repository. `http://archive.ics.uci.edu/ml`, 2013.

[21] W.-Y. Loh. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, 12(2):361–386, Apr. 2002.

[22] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. Accurate intelligible models with pairwise interactions. In *Proc. of the 19th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2013)*, pages 623–631, Chicago, IL, Aug. 11–14 2013.

[23] D. Lubinsky. Classification trees with bivariate splits. *Applied Intelligence*, 4(3):283–296, July 1994.

[24] S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: A randomized algorithm for building oblique decision trees. In *Proc. of the 11th AAAI Conference on Artificial Intelligence (AAAI 1993)*, pages 322–327, Washington, DC, July 11–15 1993.

[25] N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva. Learning optimal decision trees with SAT. In *Proc. of the 18th Int. Joint Conf. Artificial Intelligence (IJCAI'03)*, pages 1362–1368, Acapulco, Mexico, Aug. 9–15 2003.

[26] S. Nijssen and E. Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1):9–51, July 2010.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830, Oct. 2011. Available online at `https://scikit-learn.org`.

[28] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[29] S. Rendle. Factorization machines. In *Proc. of the 10th IEEE Int. Conf. Data Mining (ICDM 2010)*, pages 995–1000, Sydney, Australia, Dec. 14–17 2010.

[30] L. Rokach and O. Maimon. *Data Mining With Decision Trees. Theory and Applications*. Number 81 in Series in Machine Perception and Artificial Intelligence. World Scientific, second edition, 2015.

[31] S. Verwer and Y. Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proc. of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 1625–1632, Honolulu, HI, Jan. 27 – Feb. 1 2019.

[32] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In H. Daumé III and A. Singh, editors, *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, pages 11398–11408, Online, July 13–18 2020.

[33] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Learning interpretable, tree-based projection mappings for nonlinear embeddings. In *Proc. of the 25th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2022)*, pages 9550–9570, Online, Mar. 28–30 2022.

[34] A. Zharmagambetov and M. Á. Carreira-Perpiñán. Semi-supervised learning with decision trees: Graph Laplacian tree alternating optimization. In *Advances in Neural Information Processing Systems (NEURIPS)* neu [1], pages 2392–2405.

[35] A. Zharmagambetov, S. S. Hada, M. Gabidolla, and M. Á. Carreira-Perpiñán. Non-greedy algorithms for decision tree optimization: An experimental comparison. In *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, July 18–22 2021.

A Appendix / supplemental material

A.1 Related work

Learning trees has long been dominated by greedy recursive partitioning, of which countless versions exist [6, 28, 21, 30]. This has been more successful with univariate trees than oblique ones [6, 24], which are rarely used. Indeed, popular types of decision forests use univariate trees, such as random forests [5] or gradient boosting forests [10, 19]. Many implementations of univariate trees exist, from `scikit-learn` to commercial packages such as SAS, SPSS, Matlab or even Excel. Most univariate trees have a constant prediction at the leaves (class label or regression output), but there also exist more complex leaf models, such as linear [28], although, again, these are rarely used.

We have found only three papers exploring the idea of bivariate trees, all of them in the context of greedy recursive partitioning, differing in how they adapt the univariate split search to a bivariate one. Two early approaches [23, 3] used a variation of CART's local search for oblique splits. A recent one [4] used an exhaustive and expensive search for the bivariate split using branch-and-bound. Because of the underlying greedy recursive partitioning, this still results in large, suboptimal trees.

A parallel line of work on tree learning has focused on non-greedy approaches using exact, brute-force search. This has taken different forms: mixed-integer optimization [2, 31], dynamic programming in various forms [11] and other forms of combinatorial optimization [26, 25]. While cleverly designed, all these approaches have the fundamental disadvantage that their worst-case complexity is exponential on the problem size, so that they become infeasible for a nontrivial tree depth and number of nodes, or a nontrivial sample size or dimensionality. None of these approaches have considered bivariate splits.

A final line of work is the use of alternating optimization over the tree nodes, the TAO algorithm [7]. This has made it possible to optimize a given objective function over all the parameters of a fixed-structure tree, for tasks such as clustering [13], dimensionality reduction [33], semi-supervised learning [34], extreme classification [18] or imbalanced classification [15]. The resulting trees are smaller but more accurate than traditional ones [35], and similar advantages carry over to the forest setting [8, 32, 14, 12, 9]. Optimally updating a decision node given all other nodes (the reduced problem) takes the form of a certain 0/1 loss binary classification problem. This is NP-hard for oblique splits, but can be approximated by a surrogate loss, although this can often produce inadequate results. Here, we capitalize on the possibility to use partial enumeration in a 2D space of line orientations to find a good bivariate split efficiently.

Finally, it is possible to construct a bivariate tree by first applying feature selection to the problem to select two features globally and then fitting a tree. This gives a bivariate tree that uses the *same* two features in each node, which will result in a poor accuracy. We seek bivariate trees where each node can use *any* two features.

A.2 Other pairwise-interaction models

Interpretable models based on pairwise interactions have a long history in machine learning, data mining and statistics. In fact, the idea of bivariate trees is not new (see related work), but it has never succeeded in practice, due to the lack of an effective optimization algorithm. We provide such algorithms in this paper and demonstrate how bivariate trees are indeed useful models.

Two widely used pairwise-interaction models are Generalized Additive Models with pairwise interactions (GA^{2}M) and Factorization Machines (FM). A GA^{2}M [17, sec. 9.5], [22] consists of a sum of univariate and bivariate component functions. A FM [29] is a low-rank bilinear model (the sum of linear and quadratic terms, where the latter's matrix is low-rank). These models are interpretable in that one can inspect the individual terms (e.g. with a 2D heatmap for each GA^{2}M pairwise term). The FM is especially useful with one-hot encoded categorical data, where each pairwise weight corresponds to the co-ocurrence of two categories.

In both cases, the total number of pairwise interactions with D features is $\mathcal{O}(D^2)$, so it is critical to restrict the actual number of pairwise interactions. This is for interpretability but also to keep the number of parameters small, so the model can learn from limited sample sizes, often a problem in practice. In FMs this is achieved by using a matrix of rank $K < D$ for the quadratic term; in GA^{2}M, by greedily selecting a small subset of interactions. In bivariate trees, we control this by

globally optimizing the loss plus a regularization term that penalizes the number of nodes, whose hyperparameter can be cross-validated or selected by hand to achieve a desired tree size. Note that, unlike FMs and GA^{2}Ms, bivariate trees are able to model more complex interactions due to the hierarchical structure of the tree (at the cost of making the tree deeper).

How interpretable are bivariate trees? On the one hand, each split involves now two features, so it is more complex. On the other hand, the ability to learn oblique splits (even with just two features) greatly reduces the number of nodes and depth of the tree compared to a univariate one, and hence the number of rules extracted from the tree. This can often make a bivariate tree *more interpretable* than a univariate one—a large univariate tree is not only complex, but examining it requires constant pan and zoom. A bivariate tree is also more accurate, particularly when the label depends on feature correlations. It can also happen that the bivariate split can be understood as a new, meaningful feature on its own right. Finally, bivariate trees bring another advantage over univariate trees: we can show a 2D scatterplot at each decision node on its two selected features, which behaves like a *hierarchical 2D linear discriminant analysis*. This shows information between-class (how specific classes are separated from each other), as well as within-class (such as clustering or linear structure). We convincingly demonstrate this our experiments by comparing bivariate and univariate trees in terms of accuracy and tree size, and by exploring in detail two case studies.

A.3 Oblique trees use many features

Here we show that one cannot achieve a bivariate tree by overpenalizing a sparse oblique tree. The latter, proposed in [7], minimizes the sum of the 0/1 loss plus an ℓ_1 penalty $\lambda \sum_i \|\mathbf{w}_i\|_1$ on the weight vector of each decision node i in the tree, with hyperparameter $\lambda \geq 0$. Fig. 9 shows the entire regularization path for a sparse oblique tree and a bivariate tree on the Segment dataset. While the bivariate tree has an average of at most 2 features per node throughout its path, the oblique one reaches an average of 2 only for very high λ, at which point the tree is very small and has an enormous error. The reason for this is that, while increasing λ does encourage sparsity of weight vectors over the whole tree, this sparsity results in some nodes being pruned (whenever $\mathbf{w}_i = \mathbf{0}$) while other nodes use more features. In this dataset, the best trees have about the same error ($\approx$2.5%) and number of nodes ($\approx$16) for both bivariate and oblique, but the latter using $\approx$7 features per node.

A.3.1 Regression

This requires just two modifications. First, the reduced problem over a leaf is solved by computing the mean of the samples in its reduced set. Second, the objective (1) in the decision node is now reduced to a weighted binary classification (instead of eq. (3)), where the weights are obtained by sending the sample down each child subtree and computing their respective loss.

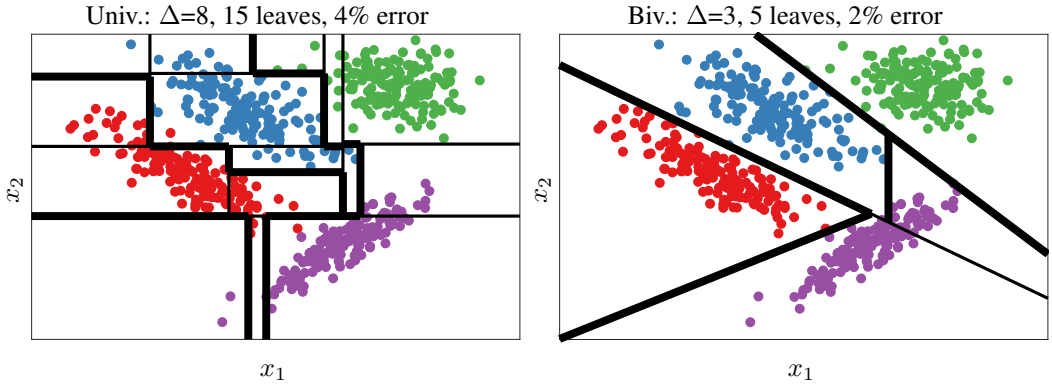


Figure 5: Partitioning by a univariate (left) and bivariate tree (right). By allowing some feature correlations, bivariate trees achieve better performance with much smaller trees.

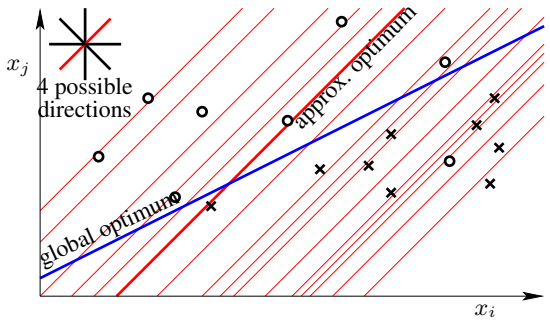


Figure 6: Illustration of our approximate solution of the reduced problem at a decision node assuming a selected pair of features (x_i, x_j). The instances in the reduced set of the node are labeled according to their pseudolabels (preferred child, left $\circ$ or right $\times$). The optimum (in 0/1 loss) linear classifier is the thick blue line (one misclassification). The approximate optimum found using the $H = 4$ possible directions (inset) is the thick red line (two misclassifications). The thin red lines are all the possible thresholds (passing through midpoints between projected instances) for the red orientation.

B Experiment setup

We use the CART Python implementation in `scikit-learn` [27]. We grow the tree to its maximum depth by setting the `minsplit` parameter to 1 and the `ccp alpha` complexity parameter to 0, and determine the optimal pruning parameter on the hold-out set.

We employ an open-source single-threaded Linux version of the C5.0 implementation in C (`https://rulequest.com/download.html`). To optimize model performance for each dataset, we conduct a grid search on the hold-out set to identify the most suitable parameters. We mainly fine-tune two parameters: `-c CF`, which governs the pruning severity, and `-m cases`, which specifies the minimum number of data points required to split a node. It's worth noting that we maintain default configurations for all other model parameters. Remarkably, our findings indicate that the tuned parameter values closely align with the default settings in many instances.

We use an open-source multithreaded implementation of BiDT by [4] written in C++ with Python interface (`https://github.com/fbollwein/OptimizedDecisionTrees`). We enable both univariate and bivariate splits. The resulting tree grows fully and is then pruned either using reduced error pruning or minimum cost-complexity pruning using the hold-out set.

We implement bivariate TAO in C++ with parallel processing and fine-tune regularization parameters C and λ. For each experiment we start with $C = 1$ and increase in with a step of 0.25 until the tree becomes fully univariate. Through empirical analysis, it was found that algorithm generalizes well when C falls within the range of 1 and 2.5 across most datasets. Since running algorithm for each value of λ is computationally costly we perform coarse search with finer steps closer to 0. As it was discussed in section 3.2 in the main paper, every next value of λ can be calculated for given C. We generate a fixed subset of line inclinations H uniformly between 0 and 180 degrees and for most experiments H is set between 30 and 90. We use save values of H for bivariate CART. The initial tree structure is obtained from fully grown CART tree (univariate or bivariate).

We run our experiments on datasets from UCI ML repository [20] and [16]. For datasets that does not have separate test set we randomly subsample 20% of the whole dataset for testing. We randomly select 10% of samples as hold-out set for validation. For experiments, we report average over 3 runs with stdev for train and test accuracy. We set a timeout of 2 hours for all algorithms on all datasets except for SUSY ($\approx$600k samples), for which it was increased to 4 hours. Our hardware setup is Intel Xeon CPU E5-2699 v3 @ 2.30GHz with 256 GB RAM.

C Additional experimental results

C.1 Interaction of C and λ: a phase diagram

Here we give a qualitative understanding of the effect of the feature cost C and the regularization hyperparameter λ on the size of the tree and the number of features it uses. It can be plotted as a

input training set $\{\mathbf{x}_n, y_n\}_{n=1}^N$,
binary axis-aligned tree T with given structure and
parameters $\boldsymbol{\Theta}$ at the nodes $\mathcal{N} = \mathcal{N}_{\text{dec}} \cup \mathcal{N}_{\text{leaf}}$
repeat
for $i \in \mathcal{N}$
$\mathcal{R}_i \leftarrow$ reduced set of node i
end if
for $d = \Delta$ **downto** 0 **do**
for $i \in$ nodes at depth d (can be done in parallel)
if $i \in \mathcal{N}_{\text{leaf}}$
$c_i \leftarrow$ majority class in $\mathcal{R}_i$
else
solution of reduced problem eq. (3) for decision node $i \in \mathcal{N}_{\text{dec}}$
end if
if $L_{\text{biv}} + \lambda C < \min(L_{\text{univ}} + \lambda, L_0)$
$\mathbf{w}_i, \mathbf{b}_i \leftarrow \boldsymbol{\theta}_i^{\text{biv}}$
else if $L_{\text{univ}} + \lambda < \min(L_{\text{biv}} + \lambda C, L_0)$
$\mathbf{w}_i, \mathbf{b}_i \leftarrow \boldsymbol{\theta}_i^{\text{univ}}$
else if $L_0 \leq \min(L_{\text{biv}} + \lambda C, L_{\text{univ}} + \lambda)$
$\mathbf{w}_i, \mathbf{b}_i \leftarrow \boldsymbol{\theta}_i^0$
end if
end for
end for
until $E(\boldsymbol{\Theta})$ does not strictly decrease
remove redundant nodes (empty features solution)
return trained T

input training set $\{\mathbf{x}_n, \bar{y}_n\}_{n \in \mathcal{R}_i}$ of decision node $i \in \mathcal{N}_{\text{dec}}$,
matrix of orientations $\mathbf{W} \in \mathbb{R}^{2 \times H}$
for each pair of features $j, k \in D$
for $\mathbf{w}_l \in \mathbf{W}$
$\mathbf{x}_l^{j,k} \leftarrow$ project selected features of
each sample in $\mathcal{R}_i$ onto w_l
$b_l^{j,k} \leftarrow$ optimal thresholding over $\mathbf{x}_l^{j,k}$
if $j, k, \mathbf{w}_l, b_l^{j,k}$ produce lowest value of eq. (3)
$\boldsymbol{\theta}_i^{biv} \leftarrow \{\mathbf{w}^*, b_l^{j,k}\}$, where $\mathbf{w}^*$ is a sparse vector
of all zeros with corresponding value of $\mathbf{w}_l$
at j, k
end if
end for
end for
return $\boldsymbol{\theta}_i^{biv}$

Figure 7: Pseudocode: bivariate TAO algorithm (top); solution to the reduced problem in a decision node i (bottom).

"phase diagram" in (λ, C)-space such as that in fig. 1. Inspection of the objective function shows that, since the loss is always less than N (where N is the sample size), there exists a critical value $0 < \lambda^* < N$ such that the optimal tree for $\lambda \geq \lambda^*$ consists of a single leaf node with a constant label (zero-variate tree). Likewise, for small enough λ, all nodes are bivariate if $C \leq 1$ (since univariate splits are no cheaper and less powerful); and there exists a critical value $1 < C^* \leq N$ such that all nodes are univariate if $C \geq C^*$ (since even a single bivariate node is too costly). Trees with both uni- and bivariate nodes exist for a region $(0, \lambda^*] \times (1, C^*]$, which contains the practically useful models, where we allow the objective function to determine the optimal uni/bivariate ratio. This also suggests a simplified training strategy where we fix C to a value a bit over one (on practice, $C \in [1.1, 1.5]$) and cross-validate λ.

C.2 Accuracy, tree size and number of features

Table 1 shows that, almost without exception across several datasets of varying types, our bivariate TAO tree is both most accurate and smallest compared to any other univariate tree (CART [6], C5.0

[28]) or bivariate tree (in particular, BiDT [4]). This is as expected: it uses the more flexible model and the better optimization. Next best is our other, more approximate algorithm, bivariate CART. The improvement in accuracy over univariate trees is consistent and varies depending on the dataset (usually a few percentage points). *The reduction in depth and number of nodes, and the corresponding simplification of the tree, is drastic, typically several times fewer nodes* (17 times smaller on the MiniBooNE dataset). In terms of training time, although not as lightning-fast as univariate trees, our bivariate trees scale well to large datasets (unlike BiDT, which times out, as expected from its brute-force search). When compared with the oblique trees, bivariate TAO trees are less accurate (as expected) but there is often little difference. However, oblique trees use quite a lot of features per node, which makes them more complex. Similar conclusions follow from the results for regression (table 3).

C.3 Computational complexity

Assume N training samples and a complete tree of depth Δ (having $2^\Delta - 1$ decision nodes). Call $\mathcal{R}_i$ the reduced set of node i. At the start of each iteration we update $\mathcal{R}_i$ for each node by propagating each training instance to its corresponding leaf, which is $\mathcal{O}(\Delta N)$ with space complexity of $\mathcal{O}(N)$ to store indices.

We optimize node parameters in reverse BFS order starting with leaves. For each leaf $i \in \mathcal{N}_{\text{leaf}}$ we compute label using majority vote in $\mathcal{O}(|\mathcal{R}_i|)$. For each decision node $i \in \mathcal{N}_{\text{dec}}$ we first compute pseudolabels by sending samples of reduces set to the left and right child at $\mathcal{O}(2\Delta|\mathcal{R}_i|)$. For univariate split, we sort instance feature values $\{x_{nd}\}_{n\in\mathcal{R}_i}$ before thresholding. In the final step, we determine the loss for every bias value similar to how the purest split is computed within recursive partitioning. It can be done exactly and efficiently through an incremental method, where we calculate the loss for the next bias value based on the loss of the current value. The total computational complexity for D features is $\mathcal{O}(D\Delta|\mathcal{R}_i|) + \Theta(D|\mathcal{R}_i|\log|\mathcal{R}_i|)$. For bivariate split the process is computationally similar to finding univariate split on $\mathbf{X}_i^{biv}$ (described in 2.1) for every pair of features. Since H is constant, total computations complexity is $\mathcal{O}(D^2\Delta|\mathcal{R}_i|^2) + \Theta(D^2|\mathcal{R}_i|^2\log|\mathcal{R}_i|)$.

The total cost of training is dominated by decision nodes. At any depth the union of all reduced sets is the whole training set. In this case the total computational cost is upper bounded by $\Theta(\Delta^2 D^2 N^2 + \Delta D^2 N^2 \log N)$. It is worth mentioning that decision nodes at each depth can be trained in parallel, which drastically reduces running time of the algorithm.

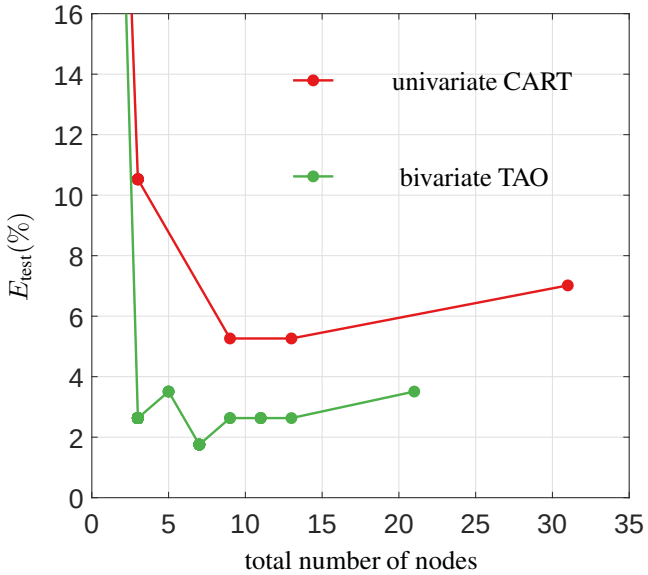


Figure 8: Test error (on the Breast Cancer dataset) of univariate CART and bivariate TAO trees as a function of the number of nodes (obtained from the regularization path over their hyperparameter).

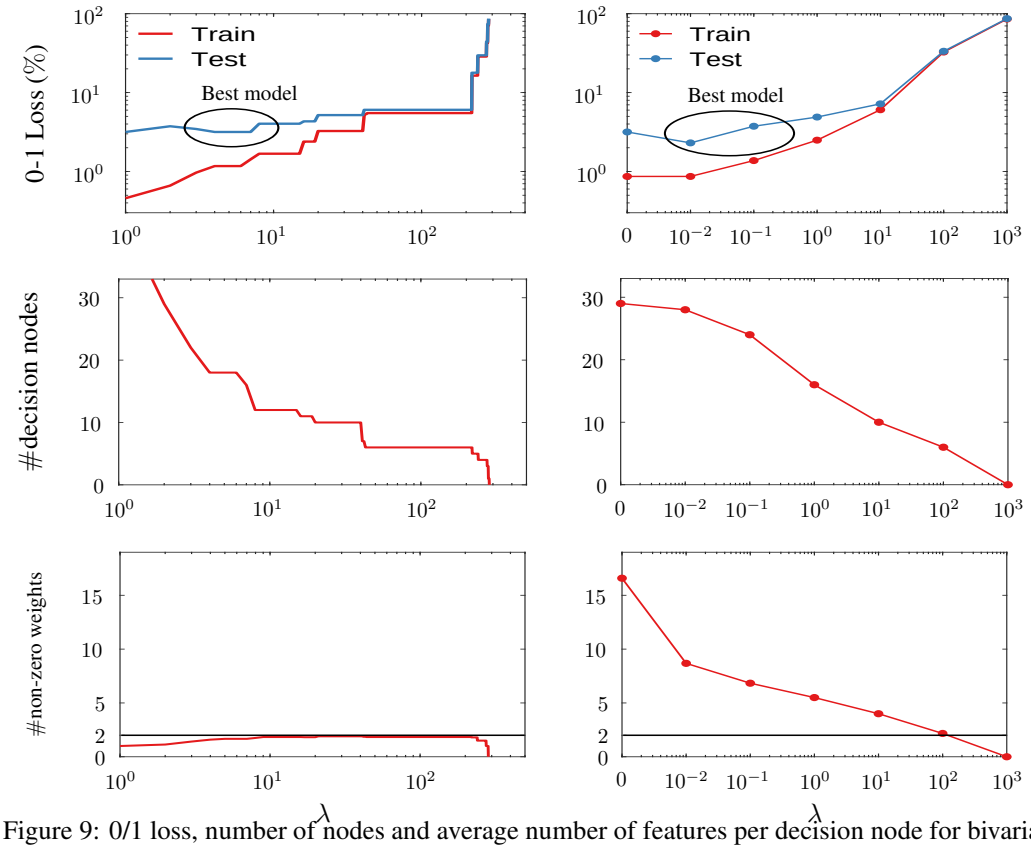


Figure 9: 0/1 loss, number of nodes and average number of features per decision node for bivariate (left) and oblique trees (right) over their regularization path (Segment dataset).

C.4 The faster algorithm: bivariate CART

Our idea above of partial enumeration over the bivariate splits can be combined with greedy recursive partitioning (in particular CART). This does not anymore optimize any global objective function and it produces worse trees than bivariate TAO, but it is much faster. It can be done in two ways. One, more efficient, is by modifying the CART split step (based on the Gini index) to use the partial enumeration, in a similar way to the TAO decision node reduced problem above. Another, less efficient (in memory), is to construct a new, augmented training set with $\leq D + \binom{D}{2}|H|$ features in advance and simply run the usual, univariate CART on it. This algorithm works quite well, as seen in our experiments, resulting in bivariate trees that are quite smaller and generalize better than univariate CART. Note this algorithm is different from previous work on bivariate trees.

Table 6 presents results of different initialization for bivariate TAO. We use initial tree structure produced by univariate and bivariate CART proposed in section 3.4. Initializing from bivariate CART generally results in smaller and better performing final trees. In the main paper we use both initializations and pick the best one on the validation set.

Table 2: Comparison between bivariate, univariate and oblique trees for classification. Each tree was selected by cross-validating its hyperparameter and this was repeated 3 times over random training/validation sets. We report training and test accuracy (% ±stdev); average depth Δ, node count n and number of features f per node (we omit $f = 1$ and 2 for univariate and bivariate trees, respectively); and average runtime (seconds or "timeout"). We indicate with color green best and blue second best result for test accuracy and node count over the univariate and bivariate trees (ignoring the oblique trees).

Dataset		bivariate			univariate		oblique
		TAO	CART	BiDT	CART	C5.0	TAO
Breast Cancer (455,30,2)	training (%)	96.04±1.53	99.12±0.00	96.99±0.10	98.61±0.41	98.9±0.48	98.21±0.79
	test (%)	98.25±0.43	98.00±0.00	97.66±0.10	94.73±0.00	95.6±0.67	97.71±1.04
	Δ/#nodes/f	1/3	3.0/9	1.6/5	4.0/16	5.3/12	3/15/10.3
	runtime (s)	4	6	2	0.1	0.1	5
Segment (1963,19,7)	training (%)	98.47±0.35	97.30±0.00	97.58±0.01	98.76±0.00	98.9±0.10	99.48±0.21
	test (%)	97.41±0.14	96.73±0.14	96.06±0.14	96.01±0.47	96.3±0.48	97.58±1.31
	Δ/#nodes/f	11.0/13	11.0/25	9.0/21	15.0/128	12.25/77	8/271/8.5
	runtime (s)	30	13	27	0.1	0.1	20
Optical recog. (3823,62,10)	training (%)	97.28±0.36	97.65±0.01	98.70±0.50	98.32±1.18	98.5±0.18	97.68±0.59
	test (%)	88.48±0.39	87.03±0.02	88.10±0.32	85.32±0.22	86.8±0.23	91.27±1.74
	Δ/#nodes/f	9.0/54	14.0/149	16.0/188	15.0/358	13.65/335	7/115.8/15.0
	runtime (s)	30	233	17	0.3	0.3	10
Spambase (3910,57,2)	training (%)	96.39±0.08	97.49±0.14	95.34±1.85	97.86±2.91	96.16±0.14	96.55±0.47
	test (%)	93.34±0.07	92.19±0.05	92.71±0.53	92.18±0.31	92.2±0.42	94.31±1.22
	Δ/#nodes/f	14/53	10.0/77	16/161	24.7/362	14.7/77	4/30/42.1
	runtime (s)	120	284	208	0.3	0.3	60
Pageblock (4652,10,5)	training (%)	98.39±0.00	98.41±0.00	99.19±0.40	99.74±0.01	97.7±0.09	95.96±0.12
	test (%)	97.93±00	96.83±0.01	96.38±0.06	96.52±0.17	96.54±0.13	96.35±0.41
	Δ/#nodes/f	8.0/25	18.0/159	4.3/14	18/268	10.05/90	6/39/7.5
	runtime (s)	30	21	15	0.2	0.1	39
House 16H (11464,16,2)	training (%)	87.1±1.55	89.45±0.00	90.42±0.23	86.2±0.0	91.98±0.55	86.55±1.10
	test (%)	85.6±0.07	84.73±0.05	85.6±0.17	83.4±0.0	83.06±0.32	85.47±0.51
	Δ/#nodes/f	7/35	10/107	10/115	8/75	15.05/245	4/13/14.9
	runtime (s)	30	21	15	0.2	0.1	24
Letter (16000,16,26)	training (%)	100±1.37	100±0.01	98.40±1.76	94.30±0.01	98.66±0.07	95.43±0.29
	test (%)	87.25±0.11	87.25±0.00	86.80±0.37	86.04±0.04	86.76±0.33	90.41±0.31
	Δ/#nodes/f	35/1314	35.0/2121	37.6/2596	28/3888	16.85/2817	11/2155/8.5
	runtime (s)	300	73	12*	0.3	0.9	77
Electricity (32702,8,2)	training (%)	98.97±2.80	95.80±0.80	96.14±1.20	99.10±0.00	95.04±0.43	98.1±1.8
	test (%)	89.38±0.12	86.05±0.05	87.91±0.06	87.80±0.16	88.64±0.42	90.23±0.19
	Δ/#nodes/f	23.0/1083	24.0/1741	22.3/1881	30.0/6366	17.25/2615	10/249/6.8
	runtime (s)	300	81	393	0.9	0.9	134
MiniBooNE (62048,50,2)	training (%)	92.36±0.00	96.02±0.02	-	96.61±0.02	95.88±0.07	91.98±0.15
	test (%)	91.16±0.00	90.68±0.03	-	90.25±0.03	89.84±0.10	91.43±0.12
	Δ/#nodes/f	11.0/105	15/831	-	19.3/2012	15.65/1787	10/133/16.8
	runtime (s)	1200	1000	timeout	5.2	6.4	3000
SUSY (600000,18,2)	training (%)	80.71±0.00	81.35±0.00	-	81.45±0.00	80.90±0.00	81.10±0.00
	test (%)	79.51±0.00	79.01±0.00	-	78.90±0.00	79.10±0.00	80.3±0.00
	Δ/#nodes/f	17.0/1077	21/2780	-	24/4389	16.25/3227	12/983
	runtime (s)	≈2h	≈1h	timeout	40.2	35.2	≈2h

Table 3: Like table 1 but for regression, reporting RMSE ±stdev over 3 runs.

Dataset	(N_{train}, D, K)	bivariate		univariate	oblique
		TAO	CART	CART	TAO
airfoil	(1002, 5, 1)	0.34±0.13	0.01±0.00	0.52±0.10	3.02±0.29
		2.46±0.43	2.72±0.02	2.75±0.62	3.13±0.38
		16/252	22/2133	15/479	8/147/3
		15	5	1.3	13
abalone	(2506, 8, 1)	2.14±0.03	0.39±0.12	2.32±0.11	2.11±0.02
		2.24±0.13	2.95±0.38	2.34±0.59	2.18±0.05
		2/7	23/903	7/39	6/58.6/6
		21	10	0.9	19
cpuact	(4915,21, 1)	1.84±0.53	1.74±0.43	2.10±0.71	2.47±0.07
		3.11±0.10	3.33±0.31	3.97±1.32	2.71±0.04
		10/191	14/327	13/373	6/52.7/13
		27	17	1.3	13
ailerons	(7154,40, 1)	1.88±0.03	0.71±0.00	1.81±0.12	1.65±0.02
		2.03±0.00	2.21±0.00	2.25±0.63	1.76±0.05
		5/21	15/50	21/105	6/60.2/13
		27	20	1.3	21

Table 4: List of features of Breast Cancer dataset. Description provided from the official UCI web page. Index column is used to indicate features in fig. 4 in the main paper.

radius1	radius2	radius3
texture1	texture2	texture3
perimeter1	perimeter2	perimeter3
area1	area2	area3
smoothness1	smoothness2	smoothness3
compactness1	compactness2	compactness3
concavity1	concavity2	concavity3
concave points1	concave points2	concave points3
symmetry1	symmetry2	symmetry3
fractal dimension1	fractal dimension2	fractal dimension3

Table 5: Feature description of Segment dataset. Description provided from the official UCI web page. Index column is used to indicate features in fig. 3 in the main paper.

Index	Description
x_0	the column of the center pixel of the region
x_1	the row of the center pixel of the region
x_2	# pixels in a region $= 9$
x_3	# lines of low contrast that go through the region
x_4	# lines of high contrast, greater than 5
x_5	contrast of horizontally adjacent pixels in the region
x_6	standard deviation of prev. feature
x_7	contrast of vertically adjacent pixels
x_8	standard deviation of prev. feature
x_9	average over the region of (R + G + B)/3
x_{10}	average over the region of the R
x_{11}	average over the region of the B
x_{12}	average over the region of the G
x_{13}	excess red: (2R - (G + B))
x_{14}	excess blue: (2B - (G + R))
x_{15}	excess green: (2G - (R + B))
x_{16}	Value of HSV
x_{17}	Saturation of HSV
x_{18}	Hue of HSV

Table 6: Comparison between bivariate CART and univariate CART initialization for bivariate TAO. We report train and test accuracy (%, ±stdev over 3 runs), average depth Δ of the tree and average node count.

Dataset	(N_{train}, D, K)		bivariate CART init.	univariate CART init.
Breast Cancer	(455,30, 2)	training (%)	96.04±1.53	99.12±0.01
		test (%)	98.25±0.43	97.37±0.12
		Δ/#nodes	1/3	3.1/9.5
Segment	(1963,19, 7)	training (%)	98.47±0.35	99.03±0.30
		test (%)	97.41±0.14	97.12±1.10
		Δ/#nodes	11.0/13	12.0/45.3
Optical recog.	(3823,62,10)	training (%)	97.28±0.36	97.17±0.05
		test (%)	88.48±0.39	87.65±0.11
		Δ/#nodes	14.0/61	11.0/123
Spambase	(3910,57, 2)	training (%)	96.78±0.08	96.39±0.08
		test (%)	92.32±0.07	93.34±0.07
		Δ/#nodes	11/28	14.0/53
Pageblock	(4652,10, 5)	training (%)	98.39±0.00	98.24±0.01
		test (%)	97.93±00	97.81±0.01
		Δ/#nodes	8.0/25	5.0/23
House 16H	(11464,16, 2)	training (%)	91.42±1.55	87.1±1.55
		test (%)	84.93±0.07	85.6±0.07
		Δ/#nodes	11/83	7/35
Letter	(16000,16,26)	training (%)	98.3±1.37	97.25±1.37
		test (%)	87.25±0.11	86.99±0.11
		Δ/#nodes	35/1314	28/2005
Electricity	(32702, 8, 2)	training (%)	98.97±2.80	96.80±0.80
		test (%)	89.38±0.12	87.05±0.05
		Δ/#nodes	23.0/1083	24.0/1141
MiniBooNE	(62048,50, 2)	training (%)	94.72±0.00	92.36±0.00
		test (%)	91.31±0.00	91.16±0.00
		Δ/#nodes	13.0/225	11/105