Bivariate Decision Trees: Smaller, Interpretable, More Accurate

Rasul Kairgeldin Miguel Á. Carreira-Perpiñán

Dept. Computer Science & Engineering
University of California, Merced

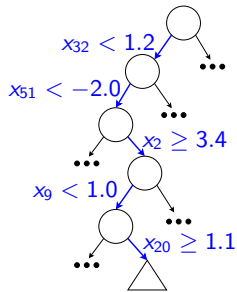
30th ACM SIGKDD Conference on
Knowledge Discovery and Data Mining

Introduction

Decision trees have several attractive properties in this context:

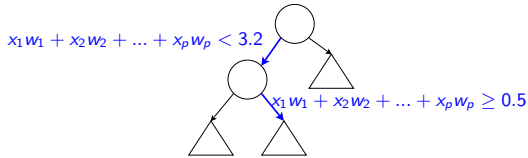
- Handle multiple classes directly.
- Conditional computation by design (input follows a single root-leaf path).
- Easy to train and to tune.
- As long as the number of nodes is not very large, they are **globally interpretable** by simple inspection of the nodes and the features they involve, without the need of any approximation or external explanation method.
- Each leaf can be described by a set of rules.

Modeling capacity: Axis-aligned trees



- Poor data model: only 5 features participate in the routing function of the above leaf.
- Resulting tree is much bigger - harder to interpret
- Recursive partitioning does not optimize global objective function

Oblique trees



- Each decision node is a function of all the features.
- Their non-linear combination is a much more complex order-D interaction.
- Better modeling capacity, but harder to interpret.

Proposed method

The goal is to design a decision trees that are more accurate than univariate trees while remain highly interpretable. **Bivariate decision trees**, where each decision node can have up to 2 features, strike a good tradeoff:

- More efficient in capturing feature correlation
- Significantly higher accuracy and much smaller compared to univariate trees
- Much more interpretable than oblique trees

How **interpretable** are bivariate trees?

- Captures pairwise interaction similar to GA^{2}M and Factorization Machine
- Much smaller number of rules to extract
- Bivariate split can be understood as a new,meaningful feature on its own
- Decision node can be shown as scatterplot in 2D (similar to **hierarchical 2D LDA**)

Illustration of a tree partitioning on a toy example

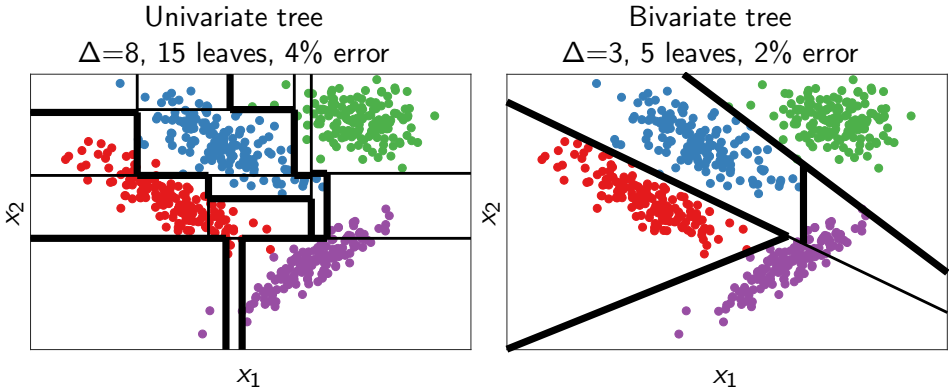
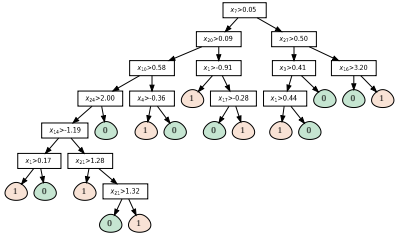


Figure: Partitioning by a univariate (left) and bivariate tree (right). By allowing some feature correlations, bivariate trees achieve better performance with much smaller trees.

Trees with extracted rules on breast cancer dataset

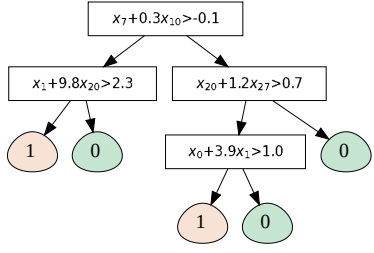
univariate CART tree

$\Delta = 7$, 16 leaves, $E_{\text{test}} = 6.4\%$



bivariate TAO tree

$\Delta = 3$, 5 leaves, $E_{\text{test}} = 2.0\%$



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

Learning bivariate trees

- Training set with K classes: $\{(\mathbf{x}_n, y_n)\}_{n=1}^N \subset \mathbb{R}^D \times \{1, \ldots, K\}$
- A set of decision nodes $\mathcal{N}_{\text{dec}}$, a set of leaves $\mathcal{N}_{\text{leaf}}$, and $\mathcal{N} = \mathcal{N}_{\text{dec}} \cup \mathcal{N}_{\text{leaf}}$
- a routing function in each decision node $i \in \mathcal{N}_{\text{dec}}$ as $f_i(\mathbf{x}; \boldsymbol{\theta}_i)$: $\mathbb{R}^D \to \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$ which sends a sample $\mathbf{x}$ to either its left or right child.
- $f_i(\mathbf{x}; \boldsymbol{\theta}_i) = \texttt{left}_i$ 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 $c_i \in \{1, \ldots, K\}$.

Learning bivariate trees. Optimization problem formulation

We collectively define the parameters of all nodes as $\mathbf{\Theta} = \{(\mathbf{w}_i, b_i)\}_{i \in \mathcal{N}_{\text{dec}}} \cup \{c_j\}_{j \in \mathcal{N}_{\text{leaf}}}$. $T(\mathbf{x}; \mathbf{\Theta})$ is a predictive function of the entire tree that guides a sample $\mathbf{x}$ to exactly one leaf. We minimize following objective function, where $L(\cdot, \cdot)$ is the 0/1 loss:

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

and we introduce the following, new type of regularization:

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

Regularization term imposes a cost of 0, 1 or C for each zero-, uni- or bivariate node (using 0, 1 or 2 features) in the tree, respectively.

Learning bivariate trees. Optimization

- We use a recent Tree Alternating Optimization (TAO) to learn bivariate trees because:
  - It can directly optimize the objective function (eq. (1)).
  - It can learn the structure of the tree and the parameters at the nodes.
  - It can take an initial tree and improve over it.
  - It is computationally efficient.
- The traditional, recursive partitioning algorithms, such as CART or C4.5, are inadequate because:
  - They grow a tree greedily from scratch rather than improving a given tree.
  - They are also quite suboptimal, particularly with oblique trees.
- "Optimal tree" algorithms (e.g. based on mixed-integer optimization and branch-and-bound) do not scale beyond toy datasets and tiny trees.

Learning bivariate trees. Optimization

- The underlying mechanism of TAO is to take a parametric tree of fixed structure (here, the one produced by CART), and perform optimization steps in turn over the parameters of a single node (decision node or leaf) while keeping the rest of the parameters fixed.
- It works quite similar to how one would optimize a neural network, but instead of gradients (which do not apply) TAO uses alternating optimization on a fixed tree structure.

TAO is based on two theorems:

- Eq. (1) **separates over any subset of non-descendant nodes** (e.g. all the nodes at the same depth); this follows from the fact that the tree makes hard decisions.
- Optimizing over the parameters of a single node i simplifies to a well-defined **reduced problem** over the instances that currently reach node i (the **reduced set** $\mathcal{R}_i \subset \{1, \dots, N\}$).

Learning bivariate trees. Optimization

The form of the reduced problem depends on the type of node:

Decision node: It is a **weighted 0/1 loss binary classification problem**:

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

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.

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 $\mathcal{R}_i$: $c_i = \text{argmax}_{k \in \{1,\ldots,K\}} \sum_{n \in \mathcal{R}_i} L(y_n, k)$.

Learning bivariate trees. Solving reduced problem over decision node

Problem (3) can be solved exactly in $\mathcal{O}(N^3 D^2)$ by enumerating every possible split over all $\binom{D}{2}$ combinations. Unfortunately, this is very costly. We propose a faster, approximate solution:

- Bivariate solution of eq. (3) s.t. $\|\mathbf{w}_i\|_0 = 2$, $b_i \in \mathbb{R}$ is achieved at $\boldsymbol{\theta}_i^{\text{biv}} = \{\mathbf{w}_i^{\text{biv}}, b_i^{\text{biv}}\}$
- Univariate solution of eq. (3) s.t. $\|\mathbf{w}_i\|_0 = 1$, $b_i \in \mathbb{R}$ is achieved at $\boldsymbol{\theta}_i^{\text{univ}} = \{(0, w_i^{\text{univ}})^T, b_i^{\text{univ}}\}$.
- Zero-variate solution of eq. (3) s.t. $\|\mathbf{w}_i\|_0 = 0$, $b_i \in \{-1, +1\}$ is achieved at $\boldsymbol{\theta}_i^0 = \{\mathbf{0}, b_i^0\}$.

Learning bivariate trees. Solving reduced problem over decision node

- Bivariate solution:
  - $\mathbf{W} \in \mathbb{R}^{2 \times H}$ 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.
  - Solution can be computed using thresholding over features of $\mathbf{X}_i^{\text{biv}}$.
- Univariate solution is computed simply by thresholding over original features.
- In zero-variate solution all samples in $\mathcal{R}_i$ are sent to the left ($b_i^0 = -1$) or the right ($b_i^0 = 1$).

Learning bivariate trees. Bivariate solution

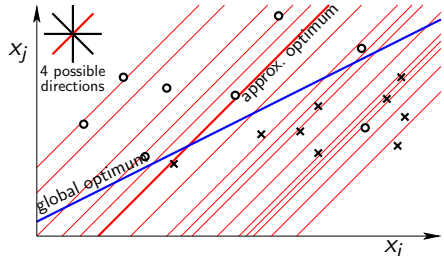


Figure: Illustration of our approximate solution of the reduced problem at a decision node assuming a selected pair of features (x_i, x_j). The instances in the reduced set of the node are labeled according to their pseudolabels (preferred child, left $\circ$ or right $\times$). The optimum (in 0/1 loss) 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.

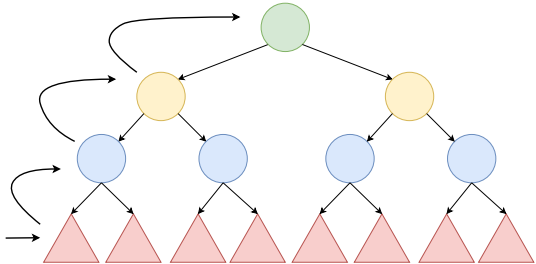
Learning bivariate trees. Solution to the reduced problem.

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.
Since bivariate split generally produces lower $0/1$ loss we typically set $C \geq 1$.
$\lambda\phi(\mathbf{w}_i)$ can be interpreted as the maximum allowed number of misclassified samples by a decision node. When this threshold is exceeded, the node is pruned.

Overview of Tree Alternating Optimization (TAO)



- Given an initial tree structure (typically produced with CART) with initial parameter values, the resulting algorithm repeatedly visits nodes in reverse breadth-first search order.
- Each iteration trains all nodes at the same depth (in parallel) from the leaves to the root, by solving either an eq. (3) or reduced problem at each leaf.

Pseudocode of bivariate TAO

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 $\mathcal{N}$
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

Bivariate CART

Our idea above of partial enumeration over the bivariate splits can be combined with greedy recursive partitioning (in particular CART).

- Does not anymore optimize any global objective function and it produces worse trees than bivariate TAO.
- Much faster to train.
- Can be implemented in 2 ways:
  - Modifying the CART split step (based on the Gini index) to use the partial enumeration
  - Constructing a new, augmented training set with $\leq D + \binom{D}{2}|H|$ features in advance and simply run the usual, univariate CART
- Works quite well on its own, can be used to initialize bivariate TAO.

Experiments: interaction of C and λ

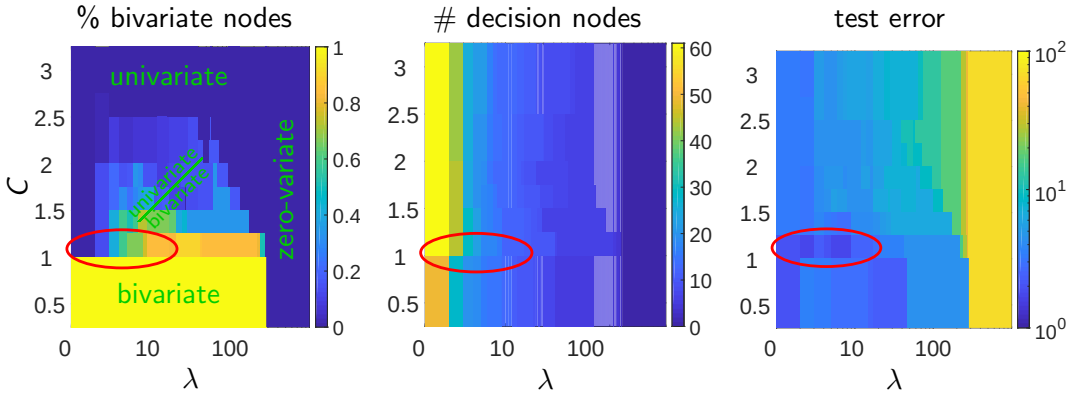


Figure: Phase diagram (λ, C) for the Segment dataset. We plot: the proportion of bivariate vs univariate decision nodes (indicating the regions of pure zero-, uni- and bivariate trees); the number of decision nodes; and the test error (%). The ellipse indicates the region of best-error trees.

Experiments: dependence on the training set size

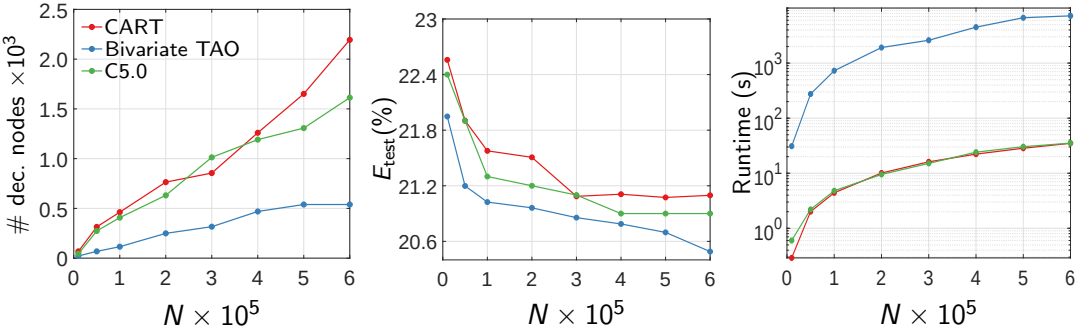


Figure: Number of nodes, test error and training time for univariate CART and C5.0 and bivariate TAO trees as a function of the sample size (subsampled from the SUSY dataset).

Experiments: comparison to oblique trees

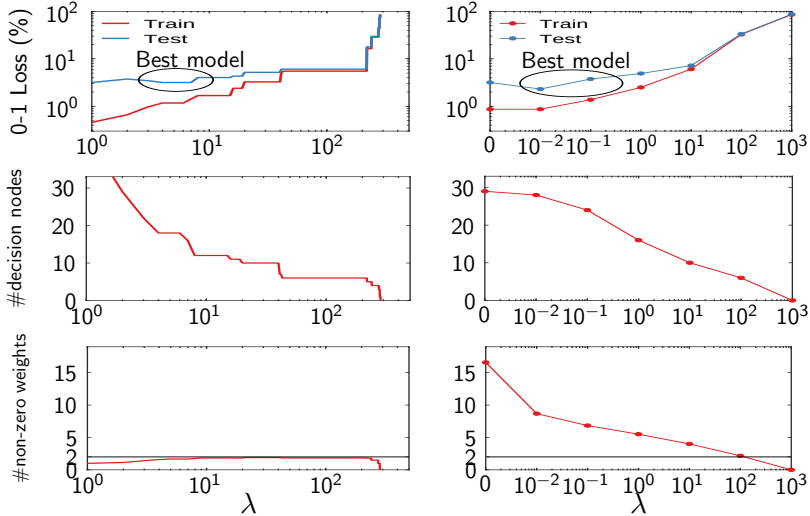
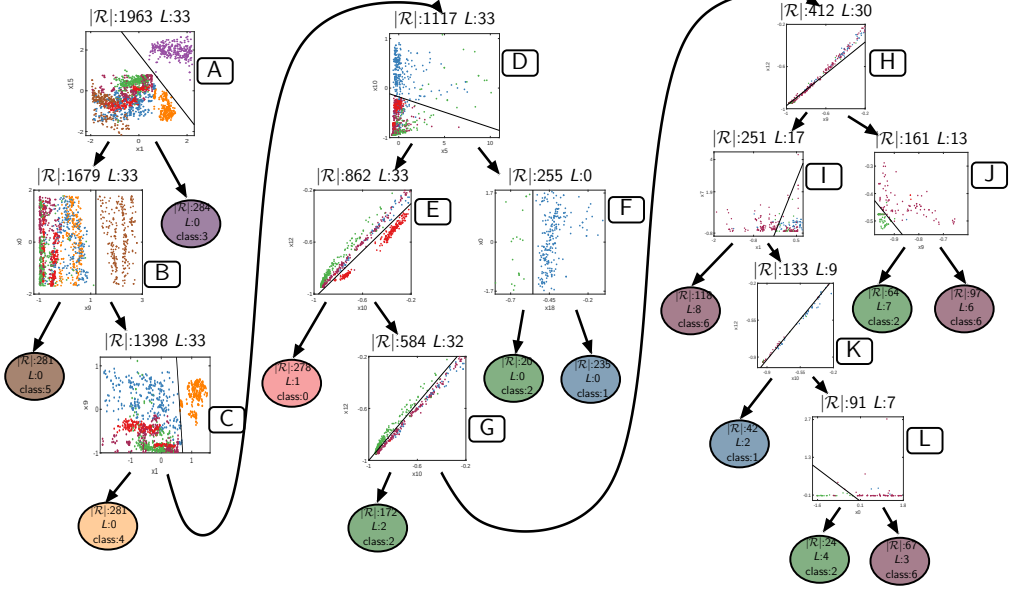


Figure: 0/1 loss, number of nodes and average number of features per decision node for bivariate (left) and oblique trees (right) over their regularization path (Segment dataset).

Experiments: quantitative comparison

Dataset	(N_{train}, D, K)		bivariate			univariate		oblique
			TAO	CART	BiDT	CART	C5.0	TAO
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
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

Experiments: interpretability



Conclusion

- We have proposed a new algorithm for training bivariate decision trees, 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.
- Bivariate trees 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.
- **Acknowledgments.** Work supported by NSF award IIS–2007147.