KERNEL DENSITY DECISION TREES

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose kernel density decision trees (KDDTs), a novel fuzzy decision tree (FDT, also called "soft" or "differentiable" decision tree) formalism based on kernel density estimation that achieves state-of-the-art prediction performance often matching or exceeding that of conventional tree ensembles. Ensembles of KDDTs achieve even better generalization. FDTs address the sensitivity and tendency to overfitting of decision trees by representing uncertainty through fuzzy partitions. However, compared to conventional, crisp decision trees, FDTs are generally complex to apply, sensitive to design choices, slow to fit and make predictions, and difficult to interpret. Moreover, finding the optimal threshold for a given fuzzy split is challenging, resulting in methods that discretize data, settle for near-optimal thresholds, or fuzzify crisp trees. Our KDDTs address these shortcomings using a fast algorithm for finding optimal partitions for FDTs with piecewise-linear splitting functions or KDDTs with piecewise-constant fitting kernels. Prediction can take place with or without fuzziness; without it, KDDTs are identical to standard decision trees, but with a more robust fitting algorithm. Using KDDTs simplifies the process of fitting a model, grounds design choices in the well-studied theory of density estimation, supports optional incorporation of expert knowledge about uncertainty in the data, and enables interpretation in the context of kernels. We demonstrate prediction performance against conventional decision trees and tree ensembles on 12 publicly available datasets.

1 INTRODUCTION

Decision trees are one of the oldest and most universally known model classes for classification and regression in machine learning. They have many benefits: they are fast to train and make predictions, relatively small in memory usage, easy to apply and tune, flexible in their non-parametric, variable-resolution structure, easy to verify for safety and robustness properties due to their piecewise-constant representation with linearly many pieces in the size of the model, and often considered to be interpretable to humans, both holistically as a hierarchical partition of the input space and on the individual prediction level as a list of simple rules.

However, decision trees in their most basic form are rarely used in practice due to certain weaknesses, the foremost of which is their sensitivity to the randomness innate to a set of training data sampled from an unknown underlying distribution and affected by the noise of real-world data collection, such as sensor noise. This results in severe overfitting in most scenarios. Moreover, the piecewise-constant nature of decision trees makes them less than ideally suited for regression.

Perhaps the most widely adopted way to address these issues is to use ensembles of trees, which have become standard repertoire in supervised learning of tabular data. These include random forests (Breiman, 2001), which employ bagging and feature subsampling to reduce overfitting; highly randomized ensembles, such as ExtraTrees (Geurts et al., 2006) that choose thresholds at random; and boosted ensembles, such as XGBoost (Chen & Guestrin, 2016), which uses a variety of strategies to improve generalization. Ensembles also improve the smoothness of regression because, while they are still piecewise-constant, the number of pieces becomes exponential in the size of the ensemble. The tradeoff of using ensembles is that the increased complexity results in longer training and prediction time, a larger memory footprint, reduced ease of interpretation, and much slower verification (shown to be NP-Complete by Katz et al. (2017)).

Another, less widely adopted way to address the overfitting issue of decision trees is by having the tree model uncertainty in the data directly. There are many realizations of this concept going by

names such as fuzzy decision trees, soft decision trees, differentiable decision trees, and neural trees. We use "fuzzy decision trees" (FDT) as an umbrella term for these kinds of models. Most operate by using soft partitions that, at each node, smoothly transition the allocation of the decision from one subtree to another based on a learned splitting function. They often support, and sometimes require, fuzzy data. They can be trained greedily by partitioning as in crisp trees, globally by algorithms resembling backpropagation as in neural networks, or some combination of the two. In particular, we focus on FDTs using greedy single-feature partitioning since they are the most like conventional, crisp tree-based models and thus retain some benefits such as flexible, non-parametric learning. The tradeoff of using this kind of FDTs is, overall, the complexity of using them. They generally exhibit shortcomings such as requiring more fitting steps and the additional design choices that come with them and losing some ease of interpretation compared to conventional, crisp decision trees. More importantly, selecting optimal partitions for continuous features given a fuzzy splitting function becomes a challenge because, unlike when fitting crisp trees, the loss is continuous in the threshold value and not easily minimized. Existing methods instead use strategies such as discretizing data, searching for near-optimal thresholds, or fuzzifying a fitted crisp tree.

We propose a new kind of FDT called kernel density decision tree (KDDT) that extends the CART methodology (Breiman et al., 1984) to model uncertainty based on the concept of kernel density estimation (KDE), a common approach to estimate the underlying distribution of a data sample. A KDDT is defined by three components: a decision tree, fitting kernels, and optional prediction kernels. The tree is identical in form to crisp trees, but instead of fitting to the data directly, it is fitted to the distribution estimated by KDE using the fitting kernels, which account for randomness from both the underlying process being modeled and from noise in the data collection process itself. More concretely, a KDDT is the decision tree that would be obtained by estimating the density of the data using the fitting kernel, sampling infinitely many points from the density estimate, then fitting a typical CART decision tree to these points. If an appropriate fitting kernel is used, this can result in better-generalizing selection both of splitting feature and threshold and of leaf values. If prediction kernels are used, then the tree's prediction is averaged over the prediction kernel at the given input to account for uncertainty in input values at test time. They can be the same or different from the fitting kernels, perhaps only modeling data collection uncertainty such as sensor noise. The kernels (and associated bandwidths), unlike standard practice for KDE, may be asymmetrical and defined differently over the domain. This could, for example, account for asymmetrical sensor noise that differs depending on the measured value. To address the problem of finding optimal splits, we also propose a fast optimal threshold search algorithm based on CART for FDTs with piecewise-linear splitting functions or KDDTs with piecewise-constant fitting kernels.

In some sense, trees and density-based classifiers are able to cover each other's weaknesses. Density-based classifiers generalize well near training data, but cannot make predictions far from it and suffer greatly from the curse of dimensionality. Trees, on the other hand, often overfit and fail to generalize in high-density regions, but their partitioning approach produces reasonable predictions even outside training data support, and they are more robust to the curse of dimensionality. KDDTs leverage the best of both. Our experiments comparing against scikit-learn models on public datasets show KDDT prediction performance sometimes matching or exceeding even random forests, and even better when used in ensembles. This hybridization also offers high utility, with the greatest strength over other FDT methods being its simplicity. The well-understood practice of density estimation provides a grounded framework for making design choices and incorporating expert knowledge about uncertainty in the data. Interpretation without a prediction kernel is identical to a conventional crisp tree, and with one, the kernel formalism at least allows a global interpretation as the expectation of the underlying crisp tree's prediction over the kernel. Other than these points, we retain all the aforementioned benefits of crisp trees, including fast fitting and prediction: we notably exploit the piecewise-constant fitting kernels such that we do not need to evaluate the full KDE function at multiple points, a computation that scales poorly with the number of training samples. In addition, a KDDT with a prediction kernel gains the utilities of smoothness and differentiability.

Our novel contributions can be summarized as:

- A new formalism for using KDE to model uncertainty in data when training and predicting with decision trees, which is unlike existing approaches that combine these concepts.
- A new FDT formulation that introduces an interdependence between splits at different levels of the tree to model the proposed KDE-DT formalism.

- A method to generate candidates for optimal splits in FDTs with piecewise-linear splitting functions, or KDDTs with piecewise-constant fitting kernels, and accompanying proof that one of the candidates is the optimal split.
- An extension of the CART algorithm to efficiently identify the optimal split candidate.

2 Related Work

Fuzzy decision trees, unlike standard decision trees, are based on soft partitions that allocate a decision partially to multiple subtrees rather than wholly to one or another. The allocation is typically based on a learned splitting function. Starting with Chang & Pavlidis (1977), many variations of the concept have been proposed over the years. We focus on those that build a tree greedily as with crisp trees and refer the reader to Chen et al. (2009), Altay & Cinar (2016), and Sosnowski & Gadomer (2019) for overviews of work in this area. In particular, we mention a few paradigms for fitting to highlight the difference in our approach. Most methods are based on fuzzy sets and only handle categorical features natively, so continuous data must be discretized (Mitra et al., 2002). Other approaches instead add fuzziness to partitions selected by algorithms for crisp trees (Chandra & Paul Varghese, 2009) or fuzzify a crisp tree after it is completely fitted (Crockett et al., 2000). Our approach uses kernels to naturally represent fuzziness of continuous features without need for discretization and efficiently finds optimal partitions for the estimated distributions of data.

There are a few cases where KDE has been integrated with decision trees. Smyth et al. (1995) propose a technique whereby prediction paths in a conventionally trained decision tree are used to choose features for KDE-based classification. This produces continuous predicted class probabilities (like KDE classifiers) and resists performance degradation due to the curse of dimensionality (like decision trees). Itani et al. (2020) use one-dimensional KDE as features for training decision trees for one-class classification, which is used for tasks like outlier or anomaly detection. Other works use KDE only in the leaves and not to determine partitions (Wang et al., 2006; Nowozin, 2012). Our approach, unlike these related works, modifies the basic decision tree to fit directly to the density estimate and optionally make kernel-smoothed predictions.

3 Methods

We first cover some preliminaries and notation relating to data, trees, and kernels. Let $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Y} \in \mathbb{R}^{n \times q}$ denote the training data. In the case of classification, each $Y_{i,j}$ is the probability that sample i has label j; for normal classification, each $\boldsymbol{Y}_{i,:}$ is a one-hot vector. We represent it in this way to support fuzzy labels and to unify the methods with those of regression. Similarly, categorical features can be represented in $\boldsymbol{X}$ as one-hot vectors or by probability of membership to each category. In the case of regression, $Y_{i,j}$ is target value j of sample i. In most regression applications, only one signal is predicted, or a separate model is used for each signal, so $\boldsymbol{Y}$ will have only one column. We notate a tree as a collection of nodes $\{m_1, m_2, \dots\}$, each of which has two children, a feature (attribute) index $a^{(m)} \in [p]$, and a threshold $t^{(m)} \in \mathbb{R}$ that form decision rule $x_{a^{(m)}} \leq t^{(m)}$.

A kernel function is a one-dimensional distribution used in kernel density estimation (KDE) to estimate a probability density given a sample. The typical form of multivariate KDE is $\hat{f}_{\boldsymbol{H}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} K_{\boldsymbol{H}}(\boldsymbol{x} - \boldsymbol{X}_{i,:}) = \frac{1}{n}|\boldsymbol{H}|^{-1/2}\sum_{i=1}^{n} K(\boldsymbol{H}^{-1/2}(\boldsymbol{x} - \boldsymbol{X}_{i,:}))$ where $\hat{f}$ is the predicted density, $\boldsymbol{H}$ is the bandwidth matrix, and K is the kernel function. Unlike the conventional form, we do not assume that the kernel function is symmetric, or that it is the same at all locations and over all features (dimensions) of $\boldsymbol{x}$. We *do* assume that the bandwidth matrix is diagonal such that $K_{\boldsymbol{H}}$ can be written as a product over the features. While this is somewhat restrictive, it enables the efficient computation of the probability that a point belongs to each partition defined by a tree with feature-aligned splitting, as shown in the next section. Thus, incorporating the bandwidth into the kernel function itself, we write the kernel at $\boldsymbol{x}'$ evaluated at $\boldsymbol{x}$ as $f(\boldsymbol{x}, \boldsymbol{x}') = \prod_{j=1}^{p} f_j(x_j, \boldsymbol{x}')$ with $f_j(\cdot, \boldsymbol{x}')$ the marginal distributions of the features. We also write $F_j(x_j, \boldsymbol{x}') = \int_{-\infty}^{x_j} f_j(z, \boldsymbol{x}')dz$ the cumulative distribution function (CDF) of f_j. The full KDE is accordingly $\hat{f}_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}, \boldsymbol{X}_{i,:})$. The following sections describe how KDDTs extend the decision tree formalism with kernels, how they are fitted, and how they make predictions.

3.1 FITTING

We fit a decision tree to the joint probability $p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{Y})$ as estimated using the KDE on $\boldsymbol{X}$.

$$\hat{f}_{\boldsymbol{X},\boldsymbol{Y}}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{\boldsymbol{Y}_{i,:}=\boldsymbol{y}} \prod_{j=1}^{p} f_j(x_j, \boldsymbol{X}_{i,:})$$

The fitting algorithm is a generalization of CART that recursively partitions the input space to minimize a loss, but rather than the raw data, we consider the distribution $\hat{f}_{\boldsymbol{X},\boldsymbol{Y}}$. In other words, the trained tree is equivalent to one trained by applying CART to infinitely many points sampled from $\hat{f}_{\boldsymbol{X},\boldsymbol{Y}}$. Of course, such an approach cannot actually be computed, but our fitting algorithm efficiently achieves the same result under the assumption that the $f_j(\cdot, \boldsymbol{X}_{i,:})$ are piecewise-constant.

To that end, for each node (including leaves) ℓ, we define membership function $u^{(\ell)} : \mathbb{R}^p \to [0, 1]$ that maps a point $\boldsymbol{x}$ to the probability that a random point $\mathbf{x} \sim f(\cdot, \boldsymbol{x})$ sampled from the kernel at $\boldsymbol{x}$ follows the path to ℓ.

$$u^{(\ell)}(\boldsymbol{x}) = P_{\mathrm{x} \sim f(\cdot, \boldsymbol{x})}(\mathbf{x} \text{ follows Path}(\ell))$$

Here Path(ℓ) is the set of internal nodes visited to reach a node ℓ. We further distinguish $\text{Path}_{\leq}(\ell) = \{m \in \text{Path}(\ell) \mid x_{a^{(m)}} \leq t^{(m)}\}$ the subset of the path where the left branch was taken, and similarly, $\text{Path}_{>}(m) = \{m' \in \text{Path}(\ell) \mid x_{a^{(m)}} > t^{(m)}\}$ the subset where the right branch was taken.

Because we assume we can write $f(\cdot, \boldsymbol{x}) = \prod_{j=1}^{p} f_j(\cdot, \boldsymbol{x})$, the features of $\mathbf{x}$ are independent. Let $\text{Path}_{\leq,j}(\ell) = \{m \in \text{Path}_{\leq}(\ell) \mid a^{(m)} = j\}$, and similarly for $\text{Path}_{>}$. Then we can write $u^{(\ell)}$ as

$$\begin{aligned} u^{(\ell)}(\boldsymbol{x}) &= \prod_{j \in [p]} P_{\mathrm{x}_j \sim f_j(\cdot, \boldsymbol{x})}(\max_{m \in \text{Path}_{>,j}(\ell)} t^{(m)} < \mathrm{x}_j \leq \min_{m \in \text{Path}_{\leq,j}(\ell)} t^{(m)}) \\ &= \prod_{j \in [p]} \max(0, b^{(\ell)}_{\leq,j}(\boldsymbol{x}) - b^{(\ell)}_{>,j}(\boldsymbol{x})) \end{aligned}$$

with upper and lower bound probabilities defined as

$$b^{(\ell)}_{\leq,j}(\boldsymbol{x}) = \begin{cases} 1 & \text{if Path}_{\leq,j}(\ell) \text{ is empty} \\ \min_{m \in \text{Path}_{\leq,j}(\ell)} F_j(t^{(m)}, \boldsymbol{x}) & \text{otherwise} \end{cases}$$

$$b^{(\ell)}_{>,j}(\boldsymbol{x}) = \begin{cases} 0 & \text{if Path}_{>,j}(\ell) \text{ is empty} \\ \max_{m \in \text{Path}_{>,j}(\ell)} F_j(t^{(m)}, \boldsymbol{x}) & \text{otherwise} \end{cases}.$$

In this way, the membership values for a given point can be easily computed for all nodes by traversing the tree and updating the bound probability values at each internal node.

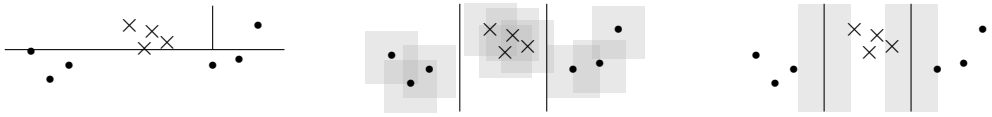
The value of a leaf ℓ is determined by expectation of $\boldsymbol{y}$ over $\hat{f}_{\boldsymbol{X},\boldsymbol{Y}}$ given the path constraints. For regression, the expectation may not be the minimizer for some loss functions, but for brevity, we limit our analysis to those where it is the minimizer, such as mean squared error (MSE).

$$\boldsymbol{v}^{(\ell)} = \mathbb{E}_{\mathbf{x},\mathbf{y} \sim \hat{f}_{\boldsymbol{X},\boldsymbol{Y}}}[\mathbf{y} \mid \mathbf{x} \text{ follows Path}(\ell)] = \frac{\sum_{i \in [n]} \boldsymbol{Y}_{i,:} u^{(\ell)}(\boldsymbol{X}_{i,:})}{\sum_{i \in [n]} u^{(\ell)}(\boldsymbol{X}_{i,:})}$$

The tree is grown from the root by recursively splitting the input space to greedily improve the purity of the leaf values as defined above. This results in a *different tree from one trained without fitting kernels*, both in the choice of splits and the values of the leaves. Figure 1 gives a simple illustrative example. In particular, at a node m, the feature $a^{(m)}$ and threshold $t^{(m)}$ of the split are chosen to maximize the gain

$$g(a^{(m)}, t^{(m)}) = L(\boldsymbol{v}^{(m)}) - \frac{\sum_{i \in [n]} u^{(m_L)}(\boldsymbol{X}_{i,:})}{\sum_{i \in [n]} u^{(m)}(\boldsymbol{X}_{i,:})} L(\boldsymbol{v}^{(m_L)}) - \frac{\sum_{i \in [n]} u^{(m_R)}(\boldsymbol{X}_{i,:})}{\sum_{i \in [n]} u^{(m)}(\boldsymbol{X}_{i,:})} L(\boldsymbol{v}^{(m_R)})$$

with m_L and m_R the children of m and $L : \mathbb{R}^p \to \mathbb{R}_{\geq 0}$ the loss function. For classification, the most common loss functions for trees are entropy $L(\boldsymbol{v}) = -\sum_{k \in [q]} v_k \log v_k$ and Gini impurity



(a) Data with two labels and the partition from a CART tree.

(b) The partition from fitting a KDDT with box kernels.

(c) Prediction with a box kernel transitions smoothly in the bands.

Figure 1: A simple example to illustrate how KDDTs differ from conventional decision trees.

$L(\boldsymbol{v}) = 1 - \sum_{k\in[q]} v_k^2$. For regression, the most common loss is mean squared error (MSE) summed over the q outputs. While it cannot be expressed as a function of $\boldsymbol{v}$ alone, it there is a loss function that is equivalent in terms of gain. Since the estimate is the mean, the MSE is the weighted variance $\mathbb{E}(y^2) - \mathbb{E}(y)^2$. When plugged into the gain, the $\mathbb{E}(y^2)$ terms sum to zero between parent and children, leaving loss $L(\boldsymbol{v}) = -\sum_{k\in q} v_k^2$.

Splitting always stops when there is zero gain, but zero gain may never occur when fitting with a kernel, so at least one additional stopping condition must be used. These may include a maximum depth, minimum permissible sample mass $\sum_{i\in[n]} u^{(\ell)}(\boldsymbol{X}_{i,:})$ for a leaf ℓ, or a pruning condition, such as cost-complexity pruning.

The key challenge is that, because of the smooth transition of decision allocation, algorithms for building crisp trees, such as CART, cannot be naively applied to find optimal splits. In fact, we are not aware of any previous work finding truly optimal splits for continuous features in fuzzy decision trees. In the following section, we show that by using piecewise-constant kernels, we can not only find the true optimal split, but incur only small additional computational cost over conventional CART, which is not true of strategies that heuristically search for good splits in fuzzy trees.

3.2 Optimal Splitting with Kernels

Choosing optimal threshold $t^{(m)}$ for given feature $a^{(m)}$ with an arbitrary kernel requires optimizing a one-dimensional non-convex function, and each candidate threshold tested costs $O(nq)$. To tackle this challenge, we introduce a more general form for FDTs and show how a KDDT maps to an equivalent FDT with a specific dependence between subsequent splits, then propose an efficient, optimal splitting algorithm for FDTs with piecewise-linear splitting functions, or KDDTs with piecewise-constant kernels. We introduce our splitting method this way both so that it can be extended to other kinds of FDTs and because the details are more easily explained in this general form.

3.2.1 General Fuzzy Decision Trees

In general, an FDT that uses single-feature threshold rules is defined by a decision tree and a splitting function $\sigma^{(m)} : \mathbb{R} \to [0, 1]$ that uses the distance to the threshold to allocate the decision between subtrees. Though not necessarily typical of FDTs, for compatibility with KDDTs, we must allow the splitting function to also depend on the feature index and value of the input. Thus we write prediction for internal nodes is computed recursively as

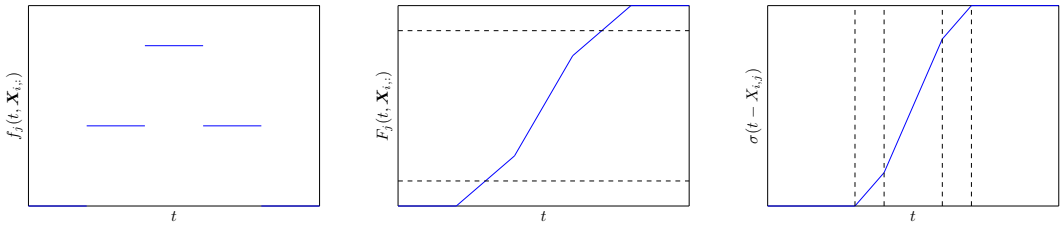
$$\hat{\boldsymbol{y}}^{(m)}(\boldsymbol{x}) = (1 - \sigma_{a^{(m)}}(t^{(m)}, \boldsymbol{x}))\hat{\boldsymbol{y}}^{(m_L)}(\boldsymbol{x}) + \sigma_{a^{(m)}}(t^{(m)}, \boldsymbol{x})\hat{\boldsymbol{y}}^{(m_R)}(\boldsymbol{x}),$$

and for leaf nodes ℓ, $\hat{\boldsymbol{y}}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{v}^{(\ell)}$. This leads to a membership function

$$u^{(\ell)}(\boldsymbol{x}) = \left(\prod_{m\in \text{Path}_{\leq}(\ell)} 1 - \sigma_{a^{(m)}}(t^{(m)}, \boldsymbol{x}) \right) \left(\prod_{m\in \text{Path}_{>}(\ell)} \sigma_{a^{(m)}}(t^{(m)}, \boldsymbol{x}) \right),$$

which is used to compute $\hat{\boldsymbol{y}}$ and $\boldsymbol{v}^{(\ell)}$ in the same way as for KDDTs. A KDDT with kernels f_j is equivalent to an FDT with splitting functions defined at each node as

$$\sigma^{(m)}(t^{(m)}, \boldsymbol{x}) = \max\left(0, \min\left(1, \frac{F_{a^{(m)}}(t^{(m)}, \boldsymbol{x}) - b^{(m)}_{>,a^{(m)}}(\boldsymbol{x})}{b^{(m)}_{\leq,a^{(m)}}(\boldsymbol{x}) - b^{(m)}_{>,a^{(m)}}(\boldsymbol{x})}\right)\right).$$



(a) A piecewise-constant kernel.

(b) Its CDF. Dashed lines show a possible $b_{\leq,j}$ and $b_{>,j}$.

(c) Its equivalent σ. Dashed lines show threshold candidates.

Figure 2: An illustration of how a KDDT kernel relates to an FDT splitting function.

The equivalence is proven in Appendix C and illustrated in Figure 2. Note that the equivalent FDT has an interdependence between splitting functions at different levels of the tree through the b terms; in principle, this is what distinguishes a KDDT from a typical application of fuzzy trees and allows it to fit to a kernel density estimate. Without this interdependence, an FDT can artificially sharpen a fuzzy split by successively splitting on the same feature and same (or similar) threshold value.

3.2.2 Efficient Optimal Splitting

We propose an efficient algorithm for finding optimal splits for FDT nodes when the splitting functions are piecewise-linear. The method a generalization of CART, which tests $O(n)$ candidate thresholds that bisect pairs of points in ascending order while keeping running totals for each side of the split so that computing loss requires constant time with respect to the number of training samples for each candidate. Similarly, a piecewise-linear splitting function allows us to generate $O(nr)$ candidate thresholds and keep running totals for each side of the split. The candidates are specified by Theorem 1, which is proven in Appendix B.

Theorem 1. *For given data $\boldsymbol{X}, \boldsymbol{Y}$ and feature index j, if the splitting function $\sigma_j(\cdot, \boldsymbol{X}_{i,:})$ is continuous and is linear on intervals $(-\infty, c_{i,1}), [c_{i,1}, c_{i,2}), \ldots, [c_{i,r_i}, \infty)$ for all $i \in [n]$ and the Hessian of the loss L is negative semidefinite everywhere, then the maximizer of the gain $g(j, \cdot)$ is in $\{c_{i,k} \mid i \in [n], k \in [r_i]\}$.*

Entropy, Gini impurity, and the weighted MSE as defined in Section 3.1 all satisfy the negative semidefinite Hessian condition. In the context of KDDTs, the threshold candidates correspond to the changepoints in the kernel values, which are conveniently also what must be iterated over to efficiently compute loss at different possible split thresholds. That is, between any pair of adjacent candidates, the membership functions resulting from the split are linear in the threshold value, and thus so are the candidate children's value vectors. By iterating over the candidates in ascending order, updating the rate of change of the child values at each candidate, and using that rate of change to compute the values themselves, we need only $O(q)$ at each candidate to update the value and compute loss. We can thus find the optimal feature and threshold in only $O(pqrn \log(rn))$ ($nr \log(nr)$ from sorting nr threshold candidates), where r is the maximum number of pieces in the splitting function. This is notably close to the $O(pqn \log n)$ to find the optimal split in CART; however, the complexity of fitting the entire tree of max depth d for CART is $O(pqdn \log n)$ since a point may only belong to a single path, whereas fitting an entire FDT is $O(pq \exp(d) nr \log(nr))$ since a point may belong to every path. The number of paths depends the kernel bandwidths, so the fitting process is fast in practice for smaller bandwidths, but slow for larger ones. If one were instead to compute the split fully at each candidate, without using the efficient scan, each candidate would cost $O(nq)$ (assuming the splitting function is evaluated in constant time), resulting in total $O(pqnc)$ for one split, with c the number of candidates, or $O(pqrn^2)$ when using the rn candidates specified by Theorem 1. One could use fewer candidates, but the resulting split may not be optimal.

Algorithm 1 in Appendix A shows the full fitting process for an FDT. It updates a membership vector $\boldsymbol{u} \in [0, 1]^n$ at each split. The change to KDDT is straightforward by using the stated equivalence for the splitting function and tracking bound probabilities $\boldsymbol{B}_{\leq} \in [0, 1]^{n \times p}$ and $\boldsymbol{B}_{>} \in [0, 1]^{n \times p}$, which are used to compute membership for each training sample. Omitted from Algorithm 1 for brevity the extreme threshold candidates where one leaf or the other has the minimum allowed sample mass.

3.3 PREDICTION

Prediction can be done with or without kernels. Prediction without kernels is the same process used with a conventional, crisp decision tree (though the tree itself is different due to training with fitting kernels), or equivalent to using a prediction kernel $f_j(x_j, \boldsymbol{x}') = \delta(x_j - x'_j)$ with δ the Dirac delta function. Prediction with kernels averages the prediction over the kernel, which is used to describe of the belief distribution of the true value of the input given a measurement, for example to account for imperfect sensors; the resulting prediction is the expectation of the prediction over that belief distribution. Prediction kernels need not be used if there is no uncertainty in the input value.

$$\hat{\boldsymbol{y}}_{\hat{f}}(\boldsymbol{x}) = \mathbb{E}_{\mathbf{x} \sim f(\cdot, \boldsymbol{x})}[\hat{\boldsymbol{y}}(\mathbf{x})] = \sum_{\ell \in \text{leaves}} u^{(\ell)}(\boldsymbol{x}) \boldsymbol{v}^{(\ell)}$$

The algorithm for computing the prediction updates $\boldsymbol{b}_{\leq}$ and $\boldsymbol{b}_{>}$ while progressing from root to leaf, then uses them to compute the leaf's membership value, that is, its weight in the sum. Similarly to fitting, prediction with a kernel is slowed depending on the kernel bandwidth because multiple paths may be visited. Prediction is fully specified in Algorithm 2 in Appendix A.

3.4 CHOOSING KERNELS

The choice of kernels ultimately defines the behavior of a KDDT. In the best case, the ability to define $f_j(\cdot, \boldsymbol{x}')$ differently depending on $\boldsymbol{x}'$ and j allows design of a density representation based on expert knowledge. However, such knowledge is not always available, but given a strategy to automatically make these choices, it still possible to benefit from using KDDTs. Though we limit fitting kernels to be piecewise-constant, it is conventional wisdom that, in kernel density estimation, the choice of bandwidth is more important than the choice of kernel, and the same holds true for KDDTs: if the bandwidth of the fitting kernel is too small, a KDDT behaves like a typical decision tree, and if too large, different data become overly blurred together, resulting in underfitting. Choosing prediction kernels is easier since candidates can be tested rapidly without refitting. Good defaults are to use the same kernels as fitting or use none at all.

We propose a few approaches for choosing bandwidths for fitting given a generic kernel, such as the box kernel or a piecewise-constant approximation of the Gaussian kernel. Since we assume that the bandwidth matrix is diagonal, bandwidth selection amounts to selecting a scalar bandwidth for each feature. Techniques from KDE such as Silverman's rule of thumb (Silverman, 1998) are simple, fast, and often effective, but do not consider the labels, so they are not always reliable for use with KDDTs. A more robust strategy is choosing the bandwidth h that maximizes the leave-one-out cross-validation likelihood $\mathcal{L}(h \mid \boldsymbol{X}, \boldsymbol{Y}) = \prod_i \hat{f}_{-i}(\boldsymbol{X}_{i,:} \mid \boldsymbol{Y}_{i,:}; h) = \prod_i \hat{f}_{-i}(\boldsymbol{X}_{i,:}, \boldsymbol{Y}_{i,:}; h) / \hat{f}_{-i}(\boldsymbol{X}_{i,:}; h)$ for each feature, where $\hat{f}_{-i}(\cdot; h)$ is the density estimate with bandwidth h fitted without sample i. This can be computed efficiently for each candidate h because of the piecewise-constant fitting kernels. Another possibility is to transform the data such that each feature has similar scale, then select a single bandwidth for all features by cross-validation of the KDDT itself. This has the advantage of accounting for the nature of the decision tree, but the disadvantages of selecting the same bandwidth for all features and of taking significantly more computation.

Our KDDT formalism can also allow a different kernel at each node, so these automatic bandwidth selection techniques (other than cross-validation of the whole KDDT) can be used to adapt the choice of bandwidth to the current partition of the training data. We find that, using rules of thumb or leave-one-out likelihood, it achieves excellent performance in some cases, but not very consistently compared to simply using the same set of kernels at all nodes. This and other more specialized bandwidth selection techniques are a topic for future work.

4 EXPERIMENTS

We compare the prediction accuracy of KDDTs against tree-based models from scikit-learn (Pedregosa et al., 2011) in the forms of decision trees, random forests, and ExtraTrees. We also compare against XGBoost (Chen & Guestrin, 2016) as a representative of boosted ensemble methods, but we do not implement boosted ensembles of KDDTs. The data are the 12 most popular tabular classification datasets with continuous features at the time of writing from the UCI Machine Learning

full name	short name	labels	features	samples
Iris	iris	3	4	150
Wine	wine	3	13	178
Glass Identification	glass	6	9	214
Optical Recognition of Handwritten Digits	optdigits	10	64	5620
Ionosphere	ion	2	34	351
Pen-Based Recognition of Handwritten Digits	pendigits	10	16	10992
Image Segmentation	segment	7	19	210
Letter Recognition	letter	26	16	20000
Yeast	yeast	10	8	1484
Spambase	spambase	2	57	4601
Connectionist Bench (Sonar, Mines vs. Rocks)	sonar	2	60	208
Statlog (Landsat Satellite)	satimage	6	36	6435

Table 1: Information about datasets.

	decision tree			random forest			ExtraTrees			boost
model	skl	ours	ours	skl	ours	ours	skl	ours	ours	xgb
p. kernel	-	yes	no	-	yes	no	-	yes	no	-
iris	94.0	96.7	**97.3**	94.0	94.7	**95.3**	95.3	95.3	**96.0**	94.0
wine	88.2	**97.7**	94.3	97.7	**98.9**	**98.9**	**98.3**	**98.3**	**98.3**	96.0
glass	71.0	**73.3**	71.0	78.6	**80.9**	77.6	**77.1**	76.7	76.3	78.1
optdigits	90.8	**97.7**	94.9	98.3	**98.6**	**98.6**	98.5	**98.6**	**98.6**	97.8
ion	87.8	92.3	**93.2**	93.7	**94.9**	**94.9**	**94.3**	**94.3**	**94.3**	93.2
pendigits	96.3	**98.9**	98.0	99.1	**99.3**	**99.3**	**99.4**	**99.4**	99.3	99.1
segment	88.6	**89.5**	89.0	91.9	**92.9**	92.4	91.4	**92.9**	91.9	87.6
letter	88.1	**94.5**	88.1	**96.8**	96.6	96.6	97.4	**97.5**	97.4	96.5
yeast	58.6	**61.3**	60.2	61.9	61.3	**62.3**	60.9	**61.5**	60.5	59.4
spambase	92.1	**93.5**	92.2	95.4	**95.6**	95.3	**96.0**	95.6	95.5	95.6
sonar	72.0	**83.1**	78.3	83.6	86.0	**89.4**	88.0	**89.4**	**89.4**	84.6
satimage	87.4	**90.6**	88.7	**92.0**	91.7	91.7	**91.8**	91.7	91.6	92.1

Table 2: Accuracy from 10-fold cross-validation compared to scikit-learn and XGBoost models. The best accuracy within each category is bold, and the best overall for each dataset is underlined.

Repository (Dua & Graff, 2017), summarized in Table 1. We normalize the features of each dataset to have mean 0 and standard deviation 1, then randomly split the dataset into 10 folds to compute accuracy by cross-validation. For the scikit-learn decision tree, we select a cost-complexity pruning α parameter from the log range of 10^{-5} to 10^{0} by 10-fold cross-validation. For our models, we select kernel bandwidth from the log range of 10^{-2} to 10^{0} by 10-fold cross-validation. This selects the same bandwidth for each feature, which is why we normalized the data. All KDDT-based models use a simple box kernel, and results are reported with and without using the same kernel for prediction. The stopping condition is a minimum sample mass of 1. All ensembles use the default settings of scikit-learn, with 100 trees, subsampling the features to the square root of the number of candidates at each node, and bootstrapping data samples only for random forests. The results are shown in Table 2.

5 Discussion

One of the foremost benefits of KDDTs is the simple, flexible framework for incorporating knowledge about uncertainty in the data. Kernels can be designed to account for measurement noise that depends on feature or measured value, known randomness in the process being observed, fuzzy features or labels, or another known source of uncertainty. Even without this knowledge, our experimental results show that automatic bandwidth selection with even a simple kernel can turn KDDTs into a low-cost enhancement to conventional trees, usually providing a significant boost to single-tree performance or a modest boost to random forest performance. There is less evidence that this minimal automatic approach is beneficial to ExtraTrees models, where the altered choice of threshold value is mostly lost due to random threshold selection. KDDTs outperform decision trees on every dataset and come close to or exceed the scikit-learn and XGBoost ensembles on several, a no-

table feat for single trees with feature-aligned splits. KDDTs outperform scikit-learn with random forests on 10 datasets. With ExtraTrees, they outperform on 6 and tie on 3, though close results should be viewed skeptically due to the particularly high innate randomness of ExtraTrees. A model using KDDTs outperforms all baseline models on 9 datasets and ties on 1.

There is still potential gain from further study of automatic bandwidth selection. Cross-validation can be somewhat expensive for ensembles used with large datasets, and limiting the search to use the same bandwidth for every feature, even after normalizing the features, is extremely restrictive. Benchmarking a wider range of bandwidth selection approaches, designing approaches better catered to KDDTs and their ensembles that consider different bandwidths for each feature, and incorporate pruning strategies for single-tree models are left to future work. However, ultimately, the best use case is when expert knowledge about the data and the underlying process it describes can be used to make choices about the kernels and bandwidth based on the needs of the application.

Even with better bandwidth selection, we are still limited to piecewise-constant fitting kernels. They can be designed to approximate any kernels, but the cost is that the complexity of fitting a KDDT is linear in the number of pieces. If there exists a class of piecewise-polynomial kernels with an analog to Theorem 1, they could also be used, with additional linear cost in the degree of the polynomial and the number of threshold candidates per piece. Regardless, kernels with support over a wide domain will lead to slower fitting and prediction since more points will belong to more paths. We are yet to study automatic kernel selection or how the choice of kernel shape affects performance.

The use of prediction kernels not only allows prediction with uncertain data, but also augments KDDTs with the utilities of smooth prediction and differentiability. A topic of our ongoing study is using this differentiability to train interpretable feature transformations for KDDTs, which addresses the weakness of decision trees in capturing certain relationships between features. Preliminary results show that this produces smaller and sometimes better-generalizing trees.

We also strongly suspect that, because kernels introduce an incentive to expand the margin between training points and the decision boundary, KDDTs will be more robust than conventional trees to input perturbation with magnitude up to the bandwidth. Appendix D includes visualizations of KDDTs and standard tree-based models on toy datasets to highlight the difference in decision boundary, particularly when the number of training samples is low. While we can verify KDDTs and ensembles without prediction kernels using existing methodology for trees and ensembles, a complete study of this topic will require a verification framework for KDDTs with prediction kernels. While we have previously developed such a framework for FDTs, it does not generalize to KDDTs as proposed because of the interdependence of splitting functions between layers of the tree. We are working to generalize this methodology so that we can provide a comprehensive robustness analysis of KDDTs in a future study.

Overall, KDDTs offer a simple and principled extension of conventional decision tree techniques to improve generalization under uncertainty. With appropriate bandwidth and stopping conditions, they can be incorporated into existing applications of trees, including ensembles, at little additional computational cost, and they offer additional utility such as smoothness, differentiability, and potentially higher robustness to input perturbation. With further study and improvements to come, we are hopeful that KDDTs will bring about wider use of fuzziness in the application of decision trees.

ACKNOWLEDGMENTS

Omitted for blind review.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, our experiments are performed on easily accessible datasets and compared against easily accessible implementations of baseline models. The procedure of the experiments is specified in Section 4. We plan to have a polished implementation of the proposed model publicly available by the time of publication.

REFERENCES

Ayca Altay and Didem Cinar. *Fuzzy Decision Trees*, pp. 221–261. Springer International Publishing, Cham, 2016. ISBN 978-3-319-39014-7. doi: 10.1007/978-3-319-39014-7_13. URL https://doi.org/10.1007/978-3-319-39014-7_13.

L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN 9780412048418.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

B. Chandra and P. Paul Varghese. Fuzzifying gini index based decision trees. *Expert Systems with Applications*, 36(4):8549–8559, 2009. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2008.10.053. URL https://www.sciencedirect.com/science/article/pii/S0957417408007537.

Robin LP Chang and Theodosios Pavlidis. Fuzzy decision tree algorithms. *IEEE Transactions on systems, Man, and cybernetics*, 7(1):28–35, 1977.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

Yi-lai Chen, Tao Wang, Ben-sheng Wang, and Zhou-jun Li. A survey of fuzzy decision tree classifier. *Fuzzy Information and Engineering*, 1(2):149–159, 2009.

K.A. Crockett, Z. Bandar, and A. Al-Attar. Soft decision trees: a new approach using non-linear fuzzification. In *Ninth IEEE International Conference on Fuzzy Systems. FUZZ- IEEE 2000 (Cat. No.00CH37063)*, volume 1, pp. 209–215, 2000. doi: 10.1109/FUZZY.2000.838660.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Sarah Itani, Fabian Lecron, and Philippe Fortemps. A one-class classification decision tree based on kernel density estimation. *Applied Soft Computing*, 91:106250, 2020. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2020.106250. URL https://www.sciencedirect.com/science/article/pii/S1568494620301903.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: an efficient SMT solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak (eds.), *Computer Aided Verification*, pp. 97–117, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

S. Mitra, K.M. Konwar, and S.K. Pal. Fuzzy decision tree, linguistic rules and fuzzy knowledge-based network: generation and evaluation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(4):328–339, 2002. doi: 10.1109/TSMCC.2002.806060.

Sebastian Nowozin. Improved information gain estimates for decision tree induction. *arXiv preprint arXiv:1206.4620*, 2012.

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 E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 1998. doi: https://doi.org/10.1201/9781315140919.

Padhraic Smyth, Alexander Gray, and Usama M Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In *Machine Learning Proceedings 1995*, pp. 506–514. Elsevier, 1995.

Zenon A. Sosnowski and Łukasz Gadomer. Fuzzy trees and forests—review. *WIREs Data Mining and Knowledge Discovery*, 9(6):e1316, 2019. doi: https://doi.org/10.1002/widm.1316. URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1316.

Li-Min Wang, Xiao-Lin Li, Chun-Hong Cao, and Sen-Miao Yuan. Combining decision tree and naive bayes for classification. *Knowledge-Based Systems*, 19(7):511–515, 2006. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2005.10.013. URL https://www.sciencedirect.com/science/article/pii/S0950705106000712. Creative Systems.

A ALGORITHMS

Algorithm 1 Fit an FDT with piecewise-linear splitting to $\boldsymbol{X}, \boldsymbol{Y}$.

function FIT($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{u}$)
  remove every row i where $u_i = 0$ from $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{u}$
  $\boldsymbol{s} \leftarrow \sum_{i\in[n]} u_i \boldsymbol{Y}_{i,:}$ ▷ weighted sum of labels
  $w \leftarrow \sum_{i\in[n]} u_i$ ▷ total sample weight
  $\boldsymbol{v} \leftarrow \boldsymbol{s}/w$
  if stopping conditions are met **then return** leaf m with $\boldsymbol{v}^{(m)} = \boldsymbol{v}$
  for $j \in \{1, \dots, p\}$ **do** ▷ iterate over features
    $\sigma_j \leftarrow$ splitting function for feature j at this node
    $c_{i,k} \leftarrow$ changepoints of $\sigma_j(\cdot, \boldsymbol{X}_{i,:})$ for all $i \in [n]$ ▷ see Theorem 1
    $\boldsymbol{t} \leftarrow (c_{i,k} \mid i \in [n], k \in [r_i - 1])$ ▷ threshold candidates
    $\boldsymbol{i} \leftarrow (i \mid i \in [n], k \in [r_i - 1])$ ▷ training data index corresponding to candidate
    $\sigma'' \leftarrow$ (change in slope of $\sigma(\cdot, \boldsymbol{X}_{i:,})$ at $c_{i,k} \mid i \in [n], k \in [r_i - 1]$)
    $\boldsymbol{s}_L \leftarrow \boldsymbol{0}$, $w_L \leftarrow 0$, $\boldsymbol{s}'_L \leftarrow \boldsymbol{0}$, $w'_L \leftarrow 0$
    $\boldsymbol{k} \leftarrow$ argsort($\boldsymbol{t}$)
    for k in $\boldsymbol{k}$ **do** ▷ iterate over candidate thresholds in increasing order
      $\delta_t \leftarrow t_k - t_{k-1}$ ▷ change in threshold; set to zero for first iteration
      $\boldsymbol{s}_L \leftarrow \boldsymbol{s}_L + \delta_t \boldsymbol{s}'_L$, $\boldsymbol{s}_R \leftarrow \boldsymbol{s} - \boldsymbol{s}_L$
      $w_L \leftarrow w_L + \delta_t w'_L$, $w_R \leftarrow w - w_L$
      $\boldsymbol{s}'_L \leftarrow \boldsymbol{s}'_L + u_{i_k} \sigma''_k \boldsymbol{Y}_{i_k,:}$
      $w'_L \leftarrow w'_L + u_{i_k} \sigma''_k$
      $\boldsymbol{v}_L \leftarrow \boldsymbol{s}_L / w_L$, $\boldsymbol{v}_R \leftarrow \boldsymbol{s}_R / w_R$
      gain $\leftarrow \frac{1}{n}(wL(\boldsymbol{v}) - w_L L(\boldsymbol{v}_L) - w_R L(\boldsymbol{v}_R))$ ▷ proportional to total sample weight
      if gain is best so far and w_L and w_R are large enough **then**
        update best gain
        if $\boldsymbol{s}'_L = \boldsymbol{0}$ **then** adjust best threshold to maximize margin
  if best gain is large enough **then**
    $j \leftarrow$ feature of best gain, $k \leftarrow$ candidate index of best gain
    $u_{L,i} \leftarrow (1 - \sigma_j(t_k, \boldsymbol{X}_{i,:}))u_i$ for all $i \in [n]$, $u_{R,i} \leftarrow \sigma_j(t_k, \boldsymbol{X}_{i,:})u_i$ for all $i \in [n]$
    $m_L \leftarrow$ FIT($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{u}_L$), $m_R \leftarrow$ FIT($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{u}_R$)
    return internal node m with $a^{(m)} = j$, $t^{(m)} = t_k$, and children m_L, m_R
  return leaf m with $\boldsymbol{v}^{(m)} = \boldsymbol{v}$
root $\leftarrow$ FIT($\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{1}$)

Algorithm 2 Predict with kernels.

function PREDICT($m, \boldsymbol{x}, \boldsymbol{b}_{\leq}, \boldsymbol{b}_{>}$)
  if m is a leaf **then**
    return $\boldsymbol{v}^{(m)} \prod_{j \in [p]} b_{\leq,j} - b_{>,j}$
  $\boldsymbol{b}'_{\leq} \leftarrow$ copy $\boldsymbol{b}_{\leq}$, $\boldsymbol{b}'_{>} \leftarrow$ copy $\boldsymbol{b}_{>}$
  $b'_{\leq,a^{(m)}} \leftarrow \min(b'_{\leq,a^{(m)}}, F_{a^{(m)}}(t^{(m)}, \boldsymbol{x}))$, $b'_{>,a^{(m)}} \leftarrow \max(b'_{>,a^{(m)}}, F_{a^{(m)}}(t^{(m)}, \boldsymbol{x}))$
  $\hat{\boldsymbol{y}} \leftarrow 0$
  if $b'_{\leq,a^{(m)}} > b_{>,a^{(m)}}$ **then** ▷ skip paths with 0 probability
    $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} +$ PREDICT(left child of $m, \boldsymbol{x}, \boldsymbol{b}'_{\leq}, \boldsymbol{b}_{>}$)
  if $b_{\leq,a^{(m)}} > b'_{>,a^{(m)}}$ **then** ▷ skip paths with 0 probability
    $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} +$ PREDICT(right child of $m, \boldsymbol{x}, \boldsymbol{b}_{\leq}, \boldsymbol{b}'_{>}$)
  return $\hat{\boldsymbol{y}}$
$\hat{\boldsymbol{y}} \leftarrow$ PREDICT(root, $\boldsymbol{x}, \mathbf{1}, \mathbf{0}$)

B PROOF OF OPTIMALITY OF CANDIDATE THRESHOLDS

Recall that we define the gain as

$$g(a^{(m)}, t^{(m)}) = L(\boldsymbol{v}^{(m)}) - \frac{\sum_{i \in [n]} u^{(m_L)}(\boldsymbol{X}_{i,:})}{\sum_{i \in [n]} u^{(m)}(\boldsymbol{X}_{i,:})} L(\boldsymbol{v}^{(m_L)}) - \frac{\sum_{i \in [n]} u^{(m_R)}(\boldsymbol{X}_{i,:})}{\sum_{i \in [n]} u^{(m)}(\boldsymbol{X}_{i,:})} L(\boldsymbol{v}^{(m_R)})$$

where L is the loss function. We will derive the second derivative with respect to the threshold t of the m_L term of the gain assuming linear splitting function. To ease notation, let $u_i = u^{(m_L)}(\boldsymbol{X}_{i,:})$, $w = \sum_i u_i^{(m_L)}$, and $\boldsymbol{v} = \boldsymbol{v}^{(m_L)}$, and let w_0 be defined similarly for the parent m.

$$\frac{d}{dt}\frac{w}{w_0} L(\boldsymbol{v}) = \frac{1}{w_0}\frac{dw}{dt} L(\boldsymbol{v}) + \frac{w}{w_0}\frac{d\boldsymbol{v}}{dt} \cdot \nabla L(\boldsymbol{v})$$

$$\frac{d^2}{dt^2}\frac{w}{w_0} L(\boldsymbol{v}) = \frac{1}{w_0}\frac{d^2 w}{dt^2} L(\boldsymbol{v}) + \left(\frac{2}{w_0}\frac{dw}{dt}\frac{d\boldsymbol{v}}{dt} + \frac{w}{w_0}\frac{d^2\boldsymbol{v}}{dt^2} \right) \cdot \nabla L(\boldsymbol{v}) + \frac{w}{w_0}\frac{d\boldsymbol{v}}{dt}^{\top} \nabla^2 L(\boldsymbol{v}) \frac{d\boldsymbol{v}}{dt}$$

We have $u_i = (1 - \sigma^{(m)}(t, \boldsymbol{X}_{i,:})) u^{(m)}(\boldsymbol{X}_{i,:})$. Since we assume $\sigma^{(m)}(\cdot, \boldsymbol{X}_{i,:})$ is piecewise-linear, u_i is linear with respect to t, so $d^2 w / dt^2 = 0$ and the first term vanishes. We next focus on the part in parentheses. Let $\boldsymbol{s} = \sum_i u_i \boldsymbol{Y}_{i,:}$ so that $\boldsymbol{v} = \boldsymbol{s}/w$.

$$\frac{d\boldsymbol{v}}{dt} = \frac{1}{w^2}\left(w\frac{d\boldsymbol{s}}{dt} - \frac{dw}{dt}\boldsymbol{s} \right)$$

$$\frac{d^2\boldsymbol{v}}{dt^2} = -\frac{2}{w^3}\frac{dw}{dt}\left(w\frac{d\boldsymbol{s}}{dt} - \frac{dw}{dt}\boldsymbol{s} \right) + \frac{1}{w^2}\left(w\frac{d^2\boldsymbol{s}}{dt^2} - \frac{d^2 w}{dt^2}\boldsymbol{s} \right)$$

Again because u_i is linear with respect to t, $d^2 w/dt^2 = 0$ and $d^2\boldsymbol{s}/dt^2 = 0$.

$$= -\frac{2}{w^3}\frac{dw}{dt}\left(w\frac{d\boldsymbol{s}}{dt} - \frac{dw}{dt}\boldsymbol{s} \right)$$

$$= -\frac{2}{w}\frac{dw}{dt}\frac{d\boldsymbol{v}}{dt}$$

Substitute this into the part of the earlier expression in parentheses.

$$\frac{2}{w_0}\frac{dw}{dt}\frac{d\boldsymbol{v}}{dt} + \frac{w}{w_0}\frac{d^2\boldsymbol{v}}{dt^2} = \frac{2}{w_0}\frac{dw}{dt}\frac{d\boldsymbol{v}}{dt} - \frac{2}{w_0}\frac{dw}{dt}\frac{d\boldsymbol{v}}{dt} = 0$$

Thus we have the simplified second derivative of the m_L term.

$$\frac{d^2}{dt^2}\frac{w}{w_0} L(\boldsymbol{v}) = \frac{w}{w_0}\frac{d\boldsymbol{v}}{dt}^{\top} \nabla^2 L(\boldsymbol{v}) \frac{d\boldsymbol{v}}{dt}$$

The same argument holds for m_R, where the only difference is $u_i = \sigma^{(m)}(t, \boldsymbol{X}_{i,:}) u^{(m)}(\boldsymbol{X}_{i,:})$. Thus, if $\nabla^2 L$ is negative semidefinite everywhere, the gain has positive second derivative with respect to t at all t. Moreover, this implies that on any given interval, the gain is maximized at one of the endpoints; for piecewise-linear splitting, it is accordingly maximized at one of the change points.

C PROOF OF EQUIVALENCE OF KDDT AND FDT

Recall that, for equivalence, we define the splitting function separately at each node as

$$\sigma^{(m)}(t^{(m)}, \boldsymbol{x}) = \max\left(0, \min\left(1, \frac{F_{a^{(m)}}(t^{(m)}, \boldsymbol{x}) - b^{(m)}_{>,a^{(m)}}(\boldsymbol{x})}{b^{(m)}_{\leq,a^{(m)}}(\boldsymbol{x}) - b^{(m)}_{>,a^{(m)}}(\boldsymbol{x})}\right)\right).$$

By induction over the path, we show that the membership function $u^{(\ell)}_{\text{FDT}}$ for an FDT with the above splitting function is equal to the KDDT membership function $u^{(\ell)}_{\text{KDDT}}$.

The base case is when the path is empty, that is, ℓ is the root. Then $u^{(\ell)}_{\text{FDT}}(\boldsymbol{x}) = u^{(\ell)}_{\text{KDDT}}(\boldsymbol{x}) = 1$.

For the inductive case, we assume that, for all $\boldsymbol{x}$, $u^{(\ell')}_{\text{FDT}}(\boldsymbol{x}) = u^{(\ell')}_{\text{KDDT}}(\boldsymbol{x})$, where ℓ' is the parent of ℓ. Assume ℓ is the left child of ℓ'.

$$u^{(\ell)}_{\text{FDT}}(\boldsymbol{x}) = (1 - \sigma^{(\ell')}(t^{(\ell')}, \boldsymbol{x}))u^{(\ell')}_{\text{FDT}}(\boldsymbol{x})$$

substitute for the splitting function and apply the inductive assumption.

$$\begin{aligned}
&= \left(1 - \max\left(0, \min\left(1, \frac{F_{a^{(m)}}(t^{(m)}, \boldsymbol{x}) - b^{(\ell')}_{>,a^{(m)}}}{b^{(\ell')}_{\leq,a^{(m)}} - b^{(\ell')}_{>,a^{(m)}}}\right)\right)\right) u^{(\ell')}_{\text{KDDT}}(\boldsymbol{x}) \\
&= \frac{b^{(\ell')}_{\leq,a^{(m)}} - \max(b^{(\ell')}_{>,a^{(m)}}, \min(b^{(\ell')}_{\geq,a^{(m)}}, F_{a^{(m)}}(t^{(m)}, \boldsymbol{x})))}{b^{(\ell')}_{\leq,a^{(m)}} - b^{(\ell')}_{>,a^{(m)}}} u^{(\ell')}_{\text{KDDT}}(\boldsymbol{x}) \\
&= \frac{\max(0, b^{(\ell)}_{\leq,a^{(m)}} - b^{(\ell)}_{>,a^{(m)}})}{b^{(\ell')}_{\leq,a^{(m)}} - b^{(\ell')}_{>,a^{(m)}}} u^{(\ell')}_{\text{KDDT}}(\boldsymbol{x})
\end{aligned}$$

In practice, a nonpositive denominator will not appear since the path would have already been stopped due to zero membership. Expand $u^{(\ell')}_{\text{KDDT}}(\boldsymbol{x})$.

$$= \frac{\max(0, b^{(\ell)}_{\leq,a^{(m)}} - b^{(\ell)}_{>,a^{(m)}})}{\max(0, b^{(\ell')}_{\leq,a^{(m)}} - b^{(\ell')}_{>,a^{(m)}})} \prod_{j \in [p]} \max(0, b^{(\ell')}_{\leq,j} - b^{(\ell')}_{>,j})$$

For any $j \neq a^{(m)}$, $b^{(\ell')}_{\leq,j} = b^{(\ell)}_{\leq,j}$ and $b^{(\ell')}_{>,j} = b^{(\ell)}_{>,j}$.

$$\begin{aligned}
&= \prod_{j \in [p]} \max(0, b^{(\ell)}_{\leq,j} - b^{(\ell')}_{>,j}) \\
&= u^{(\ell)}_{\text{KDDT}}(\boldsymbol{x})
\end{aligned}$$

The case where ℓ is the right child of ℓ' is nearly identical and thus omitted.

Since the membership function for an FDT using the specified splitting function is equivalent to a KDDT, and since both use the membership function in the same way to learn leaf values, determine splits, and make predictions, the two models are equivalent.

D VISUALIZATION ON TOY DATA

To help understand the practical difference between KDDTs and conventional trees, Figures 3 and 4 show some toy classification and regression datasets, respectively, each with 50, then 1000 samples, as well as the output of decision trees, random forests, and KDDTs with and without prediction kernels. Also reported is the test accuracy or R^2 with 10,000 test points. As in the main experiments, the scikit-learn models use default settings. The cost-complexity pruning parameter for the decision trees and the bandwidth for KDDTs are chosen by 10-fold cross-validation.

The KDDT classifiers tend to produce smoother boundaries with larger margins. Even without a prediction kernel, the tree is grown larger and produces smoother predicted probabilities (though

it can be pruned smaller without affecting the decision boundary if desired). In addition, both the KDDT classifiers and regressors have particularly good performance compared to the scikit-learn models on many of these toy datasets. We suspect that this might be due to kernels being especially effective when the number of samples and features is low, and because the noise used to generate these toy datasets is the same throughout the domain, resulting in the kernel describing the noise quite effectively. Even better results can be achieved by assuming that said noise is known and setting the kernel and bandwidth accordingly.

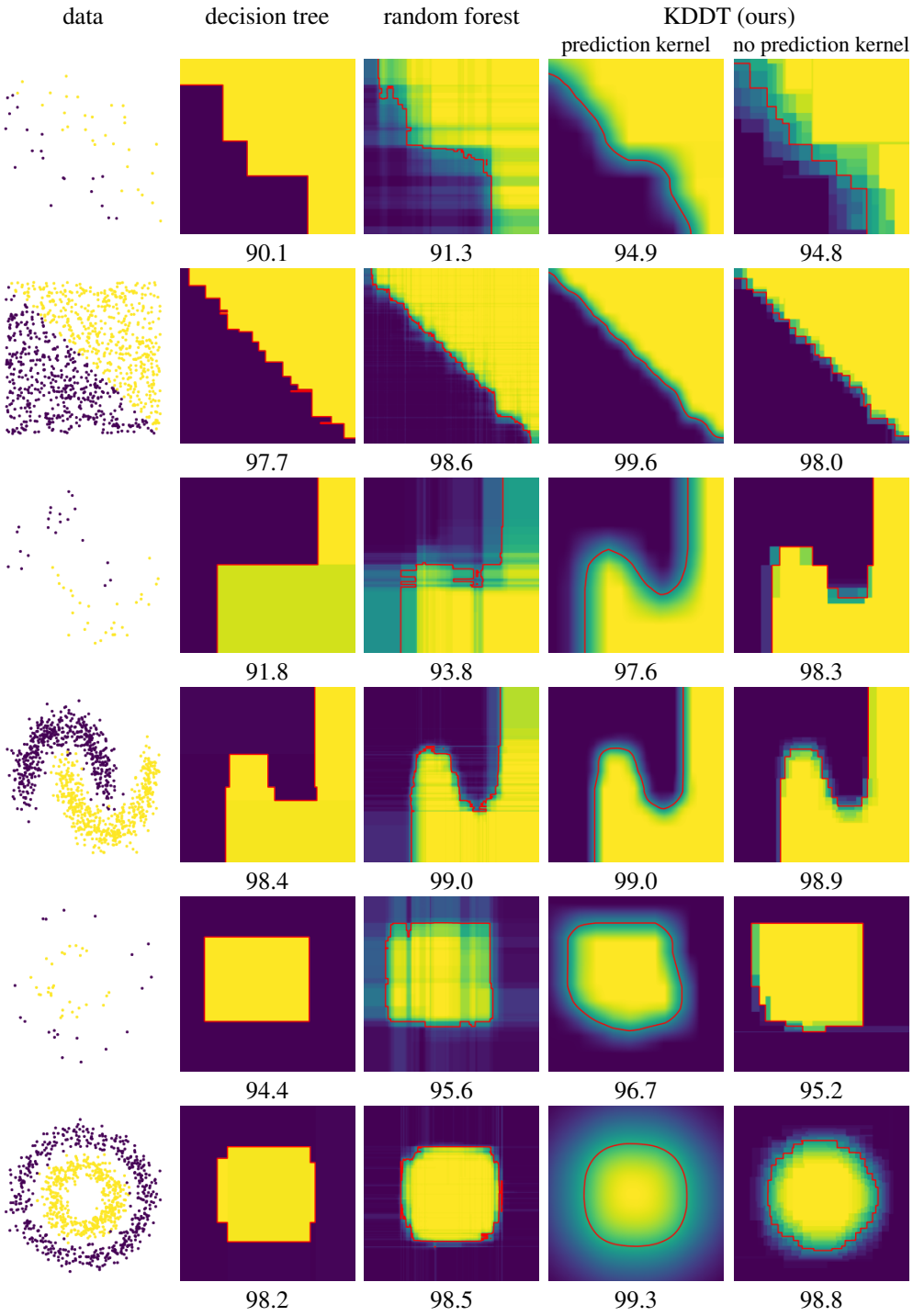


Figure 3: Visualization and test accuracy of classifiers fitted to toy data.

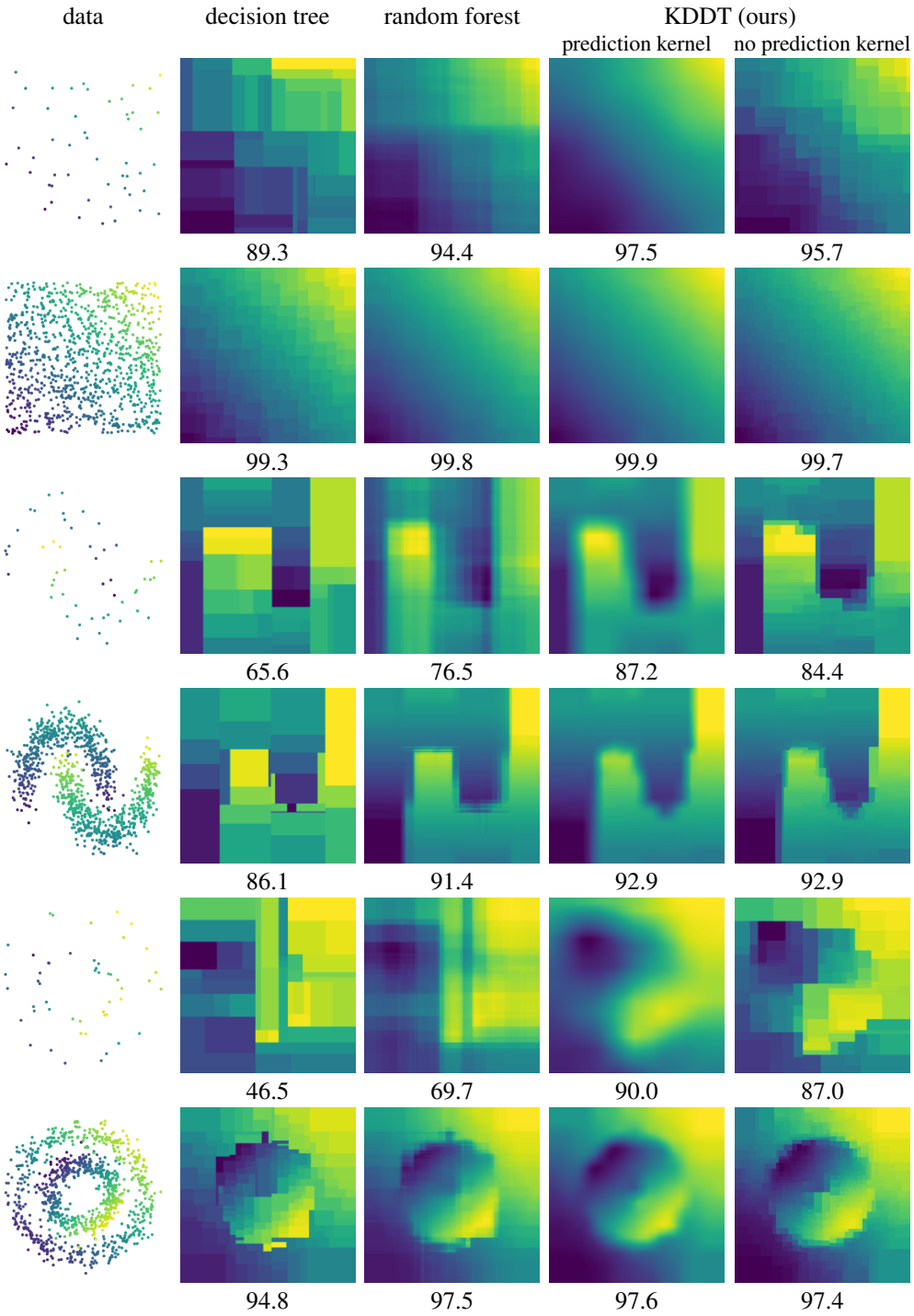


Figure 4: Visualization and test R^2 of regressors fitted to toy data.