Scalable high-dimensional density estimation with kudzu density functions

(Pre-print for conferences, Northern Hemisphere summer 2024)

Robert L Grant*

September 11, 2024

Abstract

This paper introduces kudzu density functions, which have potential to be a flexible density estimator that is computationally efficient, and scalable to large numbers of observations and dimensions. Kudzu density functions are smoothed (convolved) density estimation trees, and are named after a climbing plant, which grows rapidly over trees and other obstacles. Bootstrap aggregated (bagged) ensembles of kudzu density functions show early evidence of improvements in fidelity for difficult target distributions.

Functions in R and Stata are described in terms of usage and programming design. These provide tree fitting, kudzu tuning, joint and marginal densities, cumulative probabilities, pseudo-random number generation, and output code to use the density in BUGS, JAGS, Stan and Stata `bayesmh`. Simulations provide assessments of fidelity and computational cost.

1 Background

This paper sets out the current status of kudzu density functions, ongoing development of their theory, and software implementations. The kudzu density function is a smoothed density estimation tree (DET), which presents a contribution to density estimation methods that is computationally efficient and scales well to larger datasets.

These functions are named after the climbing plant, kudzu[1], known for its extremely fast growth and capacity to grow over trees and other obstacles, smoothing

*bayescamp@protonmail.com

BayesCamp Ltd, 16 City Business Centre, Hyde Street, Winchester SO23 7TA, UK

I am grateful to Dr Gordon Hunter for mathematical advice, and Dr Lucio Morettini for chairing the workshop in 2020 where early versions of this work were discussed.

[1]From the Japanese, クズ (*Pueraria lobata*), also known as Japanese arrowroot, is most faithfully pronounced kuːdzuː with the same vowel sounds as "to do" (English RP), though it is often pronounced kʌdzuː in the USA like "must do". The Japanese word also means rubbish, perhaps because the plant can be found growing over waste ground, making something useful from something that is not.

out their shape. They are most clearly named kudzu density functions. We do not recommend shortening this to "kudzu densities", "kudzus", or "kudzudachi", as it introduces ambiguity with the output of the function, a probability density.

This section of the paper sets out the mathematical definitions for DETs, ramp functions, and kudzu density functions. Section 2 describes its implementation at present in R and Stata software. Section 2.8 presents results of assessing computational cost and fidelity of kudzu density functions, and Section 3 does the same with ensembles of kudzu density functions. Further sections, to follow through 2024, will add evaluation by repeated Bayesian updating and practical lessons from a case study. Code is available at **robertgrantstats.co.uk/kudzu**

1.1 Motivation and definitions

We will represent information from a dataset of n observations on p variables as a cloud of points in a p-dimensional Euclidean space[2] and a $n \times p$ matrix, $\boldsymbol{X}$. Each variable, x_j ($j \in \mathbb{N}, 1 \leq j \leq p$), is allocated one dimension, and its values are elements of its support set, $\mathcal{X}_j$. As it is not ambiguous, we refer to the support of all p variables as $\mathcal{X}$.

It is useful to have a probability density function with $\mathcal{X}$ as its domain, which is trained using $\boldsymbol{X}$. Such a function can be used to explore data, check assumptions, predict new observations, estimate modes, make probabilistic (Bayesian) inferences, and to infer relationships among variables (causal or predictive) without pre-specifying an algebraic formula [1].

In fact, in some applications, it is necessary for this function to be smooth, without instantaneous jumps in value (differentiable to at least the second derivative at all $\boldsymbol{X}$). For example, some Bayesian sampling algorithms require this, in order to provide a sample from the distribution, including Hamiltonian Monte Carlo, which is implemented in the popular open-source software Stan [2], and piecewise deterministic Markov processes [3], which show great potential for future development.

Our goal will be to partition this Euclidean space so as to allocate a constant predicted density to each partition, and then carry out a smoothing transformation on the predicted function so that the resulting function is at least twice differentiable throughout its domain.

This should be implemented in readily available software for data analysis, and should be scalable to large n and large p, as measured by memory use and execution time.

The result, $\hat{f}(\boldsymbol{x})$, should of course also be a probability density function. These are defined as a class of functions:

$$\begin{aligned} \hat{f} &: \mathcal{X} \mapsto \mathbb{R}^+ \\ \int_{\mathcal{X}} \hat{f}(\boldsymbol{x}) d\boldsymbol{x} &= 1 \end{aligned} \tag{1}$$

[2]The machine learning literature calls this "feature space" and refers to "features" rather than variables.

They represent the derivative of the probability with respect to the variables, and provide estimates of the probability of potential data lying in some interval via integration.

It is important to have in mind a clear distinction between probability and probability density. Consider the likelihood of some model given the (continuous-valued) data, which is, by definition, the probability of the data given the model. In practice, however, we compute the product of the probability densities at the observed data values.

That this works for the purpose of optimisation (as in maximum likelihood estimation) or Bayesian sampling is only so because those methods only require a value *proportional* to the probability. Any observed data is actually observed to some level of precision, and so is in a strict sense discrete. The probability of an observation $P(x = 1.234)$ to three decimal places is actually $P(1.2335 \leq x < 1.2345)$. Strictly, we should integrate the probability density function on this interval, but because the interval is small in comparison to the range of x values, the area under the curve is well approximated by a rectangle, the area of which is proportionate to the density function (height of the curve). Nevertheless, we must keep in mind this distinction.

The density function referred to so far requires a point $x \in \mathcal{X}$ as input, and is sometimes called the joint density. Marginal densities are functions calculated by integrating over some subset of the variables. Conditional densities are functions that hold some variables fixed at given values.

Note that we generally assume the space defined by the variables to be a metric space [4], and this is the case for continuous-valued variables (which can in theory take any real number, possibly above or below a limit, or between two limits). We restrict consideration in this paper, and in the software implementation, to continuous-valued variables, but it may be helpful to reflect on other forms of data.

Discrete-valued variables with their own distance metrics (such as counts) are also compatible with this concept. The only consideration is that *density* estimation is no longer the goal, and any density function would have to be integrated over an interval representing the discrete value(s) of interest. Ordinal variables can be treated as though they were discrete-valued and given their own Euclidean dimensions *in this setting*, but not in general, because they imply a topological, but not metric, space. However, this only holds if there are instances of every possible value present in the data[3]. Nominal variables only possess the indiscrete topology [4] and so, are not compatible with this work at all.

There is a lack of scalable density estimation techniques in the literature. Typical methods, such as kernels, become inaccurate with more than about 4 dimensions. There have been some attempts to correct for this with more complex and computationally demanding transformations of the kernels [1], and also work around optimal transport and normalising flows [5] or neural networks [6], but a simple technique called density estimation trees appears to have been overlooked since

[3] Otherwise, it might be unclear whether the missing value was on the left or right of a split.

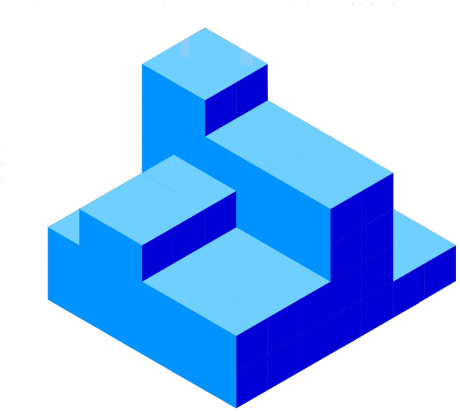

Figure 1: Illustrative diagram of a piecewise constant density in a two-dimensional space, such as DETs create.

2012.

1.2 Density estimation trees

Tree models are widely used for classification and regression problems [7, 8]. The common feature to all is that the *bounding box* of the data (the smallest hypercuboid that can contain $\boldsymbol{X}$) is recursively partitioned by splits parallel to the axes. Each subdivision of the bounding box is called a node, and those which remain at the end of the process are called terminal nodes or leaves.

Ram and Gray proposed a method called density estimation trees (DET) [9], which has not received as much attention as it deserves, having been cited 41 times between publication in 2012 and May 2024.

We must now define our notation for DETs. Density is predicted for any point $\boldsymbol{x} \in \mathcal{X}$, which is a p-vector with elements x_j where $1 \leq j \leq p$. There are L leaves, indexed by $1 \leq \ell \leq L$. Each has a bottom and a top in the jth dimension (these are $(p-1)$-dimensional hyperplanes) at $\mu_{b\ell j}$ and $\mu_{t\ell j}$ respectively, and a volume:

$$v_\ell = \prod_{j=1}^{p} (\mu_{t\ell j} - \mu_{b\ell j}) \tag{2}$$

Leaf ℓ contains n_ℓ observations. The probability density function for each leaf is $\hat{f}_{\text{DET}}(\boldsymbol{x})$, which is piecewise constant (Figure 1).

Ram and Gray's DET chooses the cutpoint for a split in a node as the one with biggest reduction in integrated squared error (ISE) [1, 10]. The ISE is an L2 criterion for model building: in theory, the difference between the true probability density and the estimate, squared and integrated over the support:

$$\int_{\mathcal{X}} \left(\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x}) \right)^2 d\boldsymbol{x} \tag{3}$$

In practice, we do not know the true $f(\boldsymbol{x})$, but to find the maximum reduction in ISE, we do not need it. First, we expand Formula 3:

$$\int_{\mathcal{X}} \left(\hat{f}^2(\boldsymbol{x}) - 2\hat{f}(\boldsymbol{x}) f(\boldsymbol{x}) + f^2(\boldsymbol{x}) \right) d\boldsymbol{x} \tag{4}$$

Next, we ignore the constant term $f^2(\boldsymbol{x})$, separate the integral of the sum into the sum of the integral:

$$\int_{\mathcal{X}} \hat{f}^2(\boldsymbol{x}) d\boldsymbol{x} - 2 \int_{\mathcal{X}} \hat{f}(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} \tag{5}$$

Note that Formula 5 can be described as a measure of *roughness* of the density estimate [1], minus two times the expectation of the density at the given data. This is a form of compensation for overfitting, valid for density estimates in general, though less relevant for DETs, where overfitting is also controlled by growth settings and pruning. Below, we will smooth this estimate, introducing more protection against over-fitting. Log-likelihood is not useful for density estimate fitting, unless there is a penalty term for roughness.

The second term can be replaced by a Monte Carlo estimator, using the observed data:

$$\int_{\mathcal{X}} \hat{f}^2(\boldsymbol{x}) d\boldsymbol{x} - \frac{2}{n} \sum_{i=1}^{n} \hat{f}(\boldsymbol{x}_i) \tag{6}$$

The piecewise constant estimate allows more simplification:

$$\begin{aligned}
&\sum_{\ell=1}^{L} \left(\int_{\boldsymbol{x} \in \ell} \left(\frac{n_\ell}{n v_\ell} \right)^2 d\boldsymbol{x} - \frac{2}{n} \frac{n_\ell}{n v_\ell} n_\ell \right) \\
=&\sum_{\ell=1}^{L} \left(\left(\frac{n_\ell}{n v_\ell} \right)^2 v_\ell - \frac{2}{n} \frac{n_\ell}{n v_\ell} n_\ell \right) \\
=&\sum_{\ell=1}^{L} -\frac{n_\ell^2}{n^2 v_\ell} \\
\propto&\sum_{\ell=1}^{L} -\frac{n_\ell^2}{v_\ell}
\end{aligned} \tag{7}$$

A useful way of thinking about this estimate is that there is a fixed count of data $n = \sum_\ell n_\ell$ to be allocated to leaves, but v_ℓ can be altered more. The tree-fitting task is to reduce the loss associated with large leaves by making sure they have small n_ℓ, and conversely to make the volume of data-containing leaves as small as possible; adding empty space to a leaf with $n_\ell > 0$ increases loss.

An important feature of ISE for DETs is that it is the sum of leaf-specific losses. This is in common with sum of squared errors for regression trees and cross-entropy for classification trees (among other loss functions).

DETs are only previously publicly implemented in a C++ library called MLPACK, along with many other machine learning methods [11]. There are interface functions in R, Python and Julia, none of which have much documentation for DETs or return very useful outputs for further manipulation of the tree. There is also a command line interface for Linux, which was used in previous work [12], and which exports an XML file requiring bespoke parsing programming. On this basis, we took the decision to implement a new DET program, suited to our needs.

1.3 Ramp functions

For smoothing, we will replace the discontinuous faces of each leaf with a ramp function $g(x)$, which rises smoothly from a minimum to a maximum value. This implies some necessary features of $g(\cdot)$. Its gradient must tend to zero at either end, so the minima and maxima are asymptotic limits for very low and high values of x respectively:

$$\lim_{x \leftarrow (-\infty)} \frac{dg}{dx} = \lim_{x \to \infty} \frac{dg}{dx} = 0 \tag{8}$$

Its gradient must rise monotonically, and then fall monotonically, which requires that its second derivative rises, falls, then rises again, making its first derivative upward concave in the middle of the function's shape, and downward concave on either side. This implies that its gradient has a shape like a symmetric probability density function, and hence the ramp function must be like a cumulative density function: integral of the probability density function, evaluated over $(-\infty, x]$.

The function we use is the inverse logistic function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{9}$$

For the function to be at least twice differentiable, and have two horizontal asymptotes at either end, will require it to be built from a power series which is convergent throughout the domain. The inverse logistic function achieves this and requires only one exponential function to be evaluated. This means that it is at least in the class of minimally computationally costly smooth ramps, requiring only one power series evaluation [4].

The inverse logistic function has minimum asymptote of 0 and maximum of 1. Its midpoint is when $z = 0$ and $g(z) = 0.5$. We can generalise its midpoint location to μ, its height to h, and increase or decrease its spread by a factor of σ, by replacing the standardised units of z with any variable x:

$$\frac{h}{1 + e^{-\frac{x-\mu}{\sigma}}} \tag{10}$$

[4] There may be a power series that enables the smooth ramp requirements and converges to required precision in fewer terms than the inverse logistic requires, though this will depend on the value of z and may not hold throughout the domain.

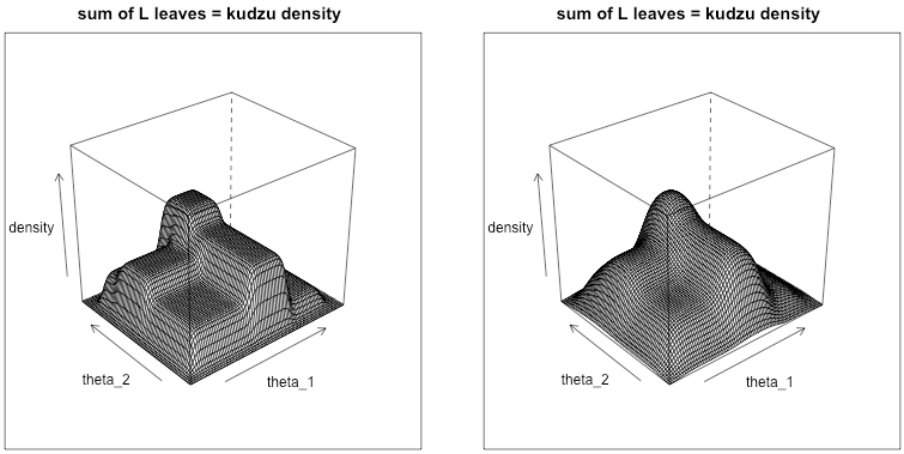

Figure 2: Illustrative diagram of a kudzu density function, applied to the DET from Figure 1, with different values of σ.

It is also possible to change the minimum asymptote, though that is not needed in this work.

The derivative of the inverse logistic function is the logistic probability density function. Further, the logistic probability density function is the product of two inverse logistic functions, one reversed with respect to x.

$$g'(z) = g(z)g(-z) \tag{11}$$

Viewed in terms of convolution, the kudzu density function is a sum of leaves, each of which is the product of its p dimensions, each of which is the convolution of the piecewise constant DET leaf with the logistic probability density function.

1.4 Kudzu density functions

The kudzu density function extends this idea by applying an inverse logistic function at each face of a DET leaf, with the lower asymptote and evaluating the leaf's contribution to density as the product of all the inverse logistic functions. The design of the kudzu density function was previously described in the context of the application to Bayesian updating [12]. It is given with full mathematical explanation here.

Considering one dimension of a DET leaf, with bottom and top edges at $\mu_{b\ell j}$ and $\mu_{t\ell j}$, and predicted density $\hat{f}_{\text{DET}}(x_j|\ell)$:

$$\hat{f}_{\text{kudzu}}(x_j|\ell) \propto \hat{f}_{\text{DET}}(x_j|\ell) \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) \tag{12}$$

Note that the top ramp function is reversed in direction; tops and bottoms must be treated separately in our calculations. Here, σ acts like the bandwidth in kernel

density estimates; a greater σ means more smoothing. At some distance $\pm\phi\sigma$ from the leaf faces, the leaf contribution can be regarded as negligible, as the product of the inverse logistic functions approaches the asymptote at zero.

For p dimensions, because each face is an inverse logistic function with only one variable, x_j, as argument, the dimensions are independent and uncorrelated, and the density is the product of them all:

$$\hat{f}_{\text{kudzu}}(\boldsymbol{x}|\ell) \propto \hat{f}_{\text{DET}}(\boldsymbol{x}|\ell) \prod_{j=1}^{p} \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) \tag{13}$$

The complete kudzu density function is proportional to the sum of the leaves:

$$\hat{f}_{\text{kudzu}}(\boldsymbol{x}) \propto \sum_{\ell=1}^{L} \hat{f}_{\text{DET}}(\boldsymbol{x}|\ell) \prod_{j=1}^{p} \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) \tag{14}$$

However, this will not in general integrate to one, and must be normalised by dividing by the sum of each leaf's definite integral over $[\mu_{b\ell j} - \phi\sigma, \mu_{t\ell j} + \phi\sigma]$ in all p dimensions. Note that each factor in the product in Formula 14 contains only one dimension of the infinitesimal dx_j, which means that the integral (repeated over p dimensions) of the product is the product of the unidimensional integrals:

$$\begin{aligned} &\int_{\mathcal{X}} \sum_{\ell=1}^{L} \hat{f}_{\text{DET}}(\boldsymbol{x}|\ell) \prod_{j=1}^{p} \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) d\boldsymbol{x} \\ \approx &\sum_{\ell=1}^{L} \hat{f}_{\text{DET}}(\boldsymbol{x}|\ell) \prod_{j=1}^{p} \int_{\mu_{b\ell j} - \phi\sigma}^{\mu_{t\ell j} + \phi\sigma} \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) dx_j \end{aligned} \tag{15}$$

Each combination of leaf and dimension is then:

$$\begin{aligned} &\approx \int \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) dx_j \\ &\approx \sigma \frac{e^{\frac{\mu_{t\ell j} - \mu_{b\ell j}}{\sigma}}}{e^{\frac{\mu_{t\ell j} - \mu_{b\ell j}}{\sigma}} - 1} \ln \left(\frac{e^{\frac{\mu_{b\ell j} - x_j}{\sigma}} + 1}{e^{\frac{\mu_{t\ell j} - x_j}{\sigma}} + 1} \right) + C \end{aligned} \tag{16}$$

which simplifies the definite integral to a sum of leaves, each being a product of terms for dimension j of leaf ℓ. Setting $w_{\ell j} = \mu_{t\ell j} - \mu_{b\ell j}$ to represent the width of the leaf ℓ in dimension j, and $u_{\ell j} = \frac{w_{\ell j}}{\sigma}$ the width in units of σ:

$$
\begin{aligned}
&\approx \quad \sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\left[\ln\left(\frac{e^{\frac{\mu_{b\ell j}-x_j}{\sigma}}+1}{e^{\frac{\mu_{t\ell j}-x_j}{\sigma}}+1}\right)\right]_{x_j=\mu_{b\ell j}-\phi\sigma}^{\mu_{t\ell j}+\phi\sigma} \\
&\approx \quad \sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\left(\ln\left(\frac{e^{\frac{\mu_{b\ell j}-(\mu_{t\ell j}+\phi\sigma)}{\sigma}}+1}{e^{\frac{\mu_{t\ell j}-(\mu_{t\ell j}+\phi\sigma)}{\sigma}}+1}\right)-\ln\left(\frac{e^{\frac{\mu_{b\ell j}-(\mu_{b\ell j}-\phi\sigma)}{\sigma}}+1}{e^{\frac{\mu_{t\ell j}-(\mu_{b\ell j}-\phi\sigma)}{\sigma}}+1}\right)\right) \\
&\approx \quad \sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\left(\ln\left(\frac{e^{\frac{-w_{\ell j}-\phi\sigma}{\sigma}}+1}{e^{\frac{\phi\sigma}{\sigma}}+1}\right)-\ln\left(\frac{e^{\frac{\phi\sigma}{\sigma}}+1}{e^{\frac{w_{\ell j}+\phi\sigma}{\sigma}}+1}\right)\right) \\
&\approx \quad \sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\left(\ln\left(\frac{e^{-(u_{\ell j}+\phi)}+1}{e^{\phi}+1}\right)-\ln\left(\frac{e^{\phi}+1}{e^{(u_{\ell j}+\phi)}+1}\right)\right)
\end{aligned}
\tag{17}
$$

which leads to:

$$
\therefore \int_{\mathcal{X}}\hat{f}_{\text{kudzu}}(\boldsymbol{x})d\boldsymbol{x} \approx \sum_{\ell=1}^{L}\hat{f}_{\text{DET}}(\boldsymbol{x}|\ell)\prod_{j=1}^{p}\sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\ln\left(\frac{e^{u_{\ell j}+\phi}+e^{-(u_{\ell j}+\phi)}+2}{e^{\phi}+e^{-\phi}+2}\right) \tag{18}
$$

When ϕ is large enough to approximate a plateau between the ramps in dimension j of leaf ℓ, the relevant factor in the product in Formula 18 simplifies to:

$$
\lim_{u_{\ell j}\to\infty,\phi\to\infty}\left(\sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\ln\left(\frac{e^{u_{\ell j}+\phi}+e^{-(u_{\ell j}+\phi)}+2}{e^{\phi}+e^{-\phi}+2}\right)\right) = \sigma u_{\ell j} = w_{\ell j} \tag{19}
$$

The complete kudzu density function is then Formula 14, divided by Formula 18:

$$
\hat{f}_{\text{kudzu}}(\boldsymbol{x}) = \frac{\sum_{\ell=1}^{L}\hat{f}_{\text{DET}}(\boldsymbol{x}|\ell)\prod_{j=1}^{p}\left(\frac{1}{1+e^{\frac{\mu_{b\ell j}-x_j}{\sigma}}}\right)\left(\frac{1}{1+e^{\frac{x_j-\mu_{t\ell j}}{\sigma}}}\right)}{\sum_{\ell=1}^{L}\hat{f}_{\text{DET}}(\boldsymbol{x}|\ell)\prod_{j=1}^{p}\sigma\frac{e^{u_{\ell j}}}{e^{u_{\ell j}}-1}\ln\left(\frac{e^{u_{\ell j}+\phi}+e^{-(u_{\ell j}+\phi)}+2}{e^{\phi}+e^{-\phi}+2}\right)} \tag{20}
$$

Evaluation of a joint density at some point $\boldsymbol{x}$ requires evaluation of $2Lp$ inverse logistic functions at most, plus trivial arithmetic operations and one-off calculation of the integral. The most costly part of each is one exponential function. Truncation of the Taylor series for the exponential function is not an option because of the substantial interest users are likely to have in the tails of the function. Nevertheless, this is relatively fast. It can be improved further by limiting evaluation to $L^* \leq L$ neighbouring leaves with non-negligible contributions to density. $\ell \in L^*$ iff:

$$
(\mu_{b\ell j}-\phi\sigma) \leq x_j \leq (\mu_{t\ell j}+\phi\sigma) \quad ,\forall j \tag{21}
$$

1.4.1 Integrated squared error for kudzu probability density functions

The ISE for kudzu is not as simple as for the piecewise constant DET. From Formula 6, we need to evaluate $\int_{\mathcal{X}} \hat{f}^2(\boldsymbol{x})d\boldsymbol{x}$.

An initial consideration is whether all kudzu density functions are square-integrable functions on $\mathbb{R}^p$ (that is, the integral of the square is finite). This can be proven by *reductio ad absurdum*.

Proof 1:
Because each such function is a sum of leaves, its square will contain $\frac{L(L+1)}{2}$ distinct terms. This is a finite summation, so can only be infinite if at least one of the terms has an infinite integral. Noting that each term approaches zero as $x_j \to -\infty$ or $x_j \to \infty$ $(\forall j)$, it follows that at least one ramp function would have to have an infinite maximum, meaning that DET would have to have set an infinite density in at least one of the leaves. The predicted density is the proportion of data (which cannot exceed 1), divided by the volume. The volume would have to be zero, but is the product of widths between cutpoints, and cutpoints are defined as either side of data. So, it would require: $\exists\{\ell, j\}$: $\mu_{b\ell j} < \mu_{t\ell j}$ **and** $\mu_{b\ell j} = \mu_{t\ell j}$. Therefore, the integral cannot be infinite.

The kudzu density function is proportional to a sum of leaves' contributions (Formula 20). If we denote the denominator from that formula as d, and set each leaf's unnormalised contribution from Formula 15 to:

$$\hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell) = \hat{f}_{\text{DET}}(\boldsymbol{x}|\ell) \prod_{j=1}^{p} \left(\frac{1}{1 + e^{\frac{\mu_{b\ell j} - x_j}{\sigma}}} \right) \left(\frac{1}{1 + e^{\frac{x_j - \mu_{t\ell j}}{\sigma}}} \right) \tag{22}$$

then the square is:

$$\hat{f}^2_{\text{kudzu}}(\boldsymbol{x}) = \frac{1}{d^2} \sum_{\ell=1}^{L} \sum_{\ell'=1}^{L} \hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell) \hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell') \tag{23}$$

which includes the sum of all the pairwise products of leaves' contributions. We can show the integrals as a symmetric $L \times L$ matrix:

$$\begin{bmatrix} \int_{\mathcal{X}} \hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell=1)\hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell'=1)d\boldsymbol{x} & \dots & \int_{\mathcal{X}} \hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell=1)\hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell'=L)d\boldsymbol{x} \\ \vdots & \ddots & \vdots \\ \int_{\mathcal{X}} \hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell=L)\hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell'=1)d\boldsymbol{x} & \dots & \int_{\mathcal{X}} \hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell=L)\hat{f}_{\text{leaf}}(\boldsymbol{x}|\ell'=L)d\boldsymbol{x} \end{bmatrix} \tag{24}$$

Row or column sums from this matrix will be useful in leafwise diagnosis of goodness of fit. Any pair of leaves that are not neighbours will have a negligible product of densities, and can be omitted from the calculation. The integral for each dimension is then the sum of a matrix of the same form. This matrix comprises L

squares down the diagonal (each leaf multiplied by itself, so there are two unique values to the edges), and off diagonal there are some pairs that share an edge and have three unique edge values, and some that have no shared edges and $\frac{L(L-1)}{2}$ products (each appearing twice in the sum). Among those that have three unique edge values, there are those that share a bottom edge, those that share a top edge, and those that are contiguous. Using integration by partial fractions, we obtain five different formulas for these five scenarios. Details are given in the appendix.

This presents no computational burden worse than exponential functions, and in fact, all the unique values of $e^{\frac{\mu_{k\ell j}}{\sigma}}$ can be stored and reused; the exponentials are arranged so as to capitalise on this.

The definite integral of $\int_{\mathcal{X}} \hat{f}^2_{\text{kudzu}}(\boldsymbol{x})d\boldsymbol{x}$ is combined with the Monte Carlo estimator, as in Formula 6, to give an ISE for kudzu, and this can be used for tuning σ as well as the possibility of displacing the leaf edges by $\delta_{k\ell j}$ to control variance inflation, which is discussed in Section 2.6.

1.4.2 Diagnostics and graphics

It is also important to be able to find *why* a particular DET leads to a suboptimal kudzu fit. A likely reason is that some leaves are inadequately smoothed while others are excessively smoothed, because their widths in various dimensions differ greatly, while we have one common σ. The possibility of adaptive σ, similar to adaptive kernel bandwidth, is to be considered in the future. Plotting $u_{\ell j}$ against v_ℓ may help to identify leaves which contribute a large volume to the density and have a wide range of uni-dimensional widths. Log-likelihood contributions from data in each leaf, or the contributions to ISE, can also be plotted.

2 Implementation in R and Stata

2.1 General programming style and priorities

Coding the DET and kudzu has prioritised, in descending order of importance: execution speed, ease of translation to other languages, human readability, and memory usage. Broadly, the style of programming is functional, though not all languages allow the use of functions as "first-class" objects [13]. Standardised lists are passed between functions, representing trees and kudzu density functions, but these are not (at present) formally defined as classes.

Where relevant, both American and British spellings are accommodated: for example, `kudzu::normalise()` and `kudzu::normalize()` are the same R function, `kudzu_normalise` and `kudzu_normalize` are identical Stata commands calling the same Mata[5] functions.

[5]Mata is a C-like internal language used inside Stata to manipulate varied objects and hold them in memory. The Stata imperative scripting language, which is familiar to all users, is more accurately called *ado code*, but this term is hardly known outside the Stata community.

Our R code uses only base R functions and is written in the "base" style (favouring nested brackets over left-to-right piping). The code will be speeded up in the future by combining R and C++ code, but a pure R version will be maintained for human readability and transparency.

Almost all the Stata functionality is written in Mata, making it much faster than the R implementation at present.

The most notable differences between the R and Mata implementations are that functions are not first-class objects[6] in Mata, and that operations on vectors and matrices in R become loops or elementwise operators (for example, `:*`) in Mata. In some places, we take advantage of Mata's accommodation of vectors or matrices with zero rows, and of the fact that missing values are taken to be the largest numbers. Other differences arise from the versions leapfrogging each other as they are updated.

2.2 Exposed and internal functions and objects

2.2.1 R

The following R functions are exposed:

- `expit()` — inverse logistic function
- `neighbour_leaves()` — given $\boldsymbol{x}$, σ and ϕ, return a vector of the index numbers of leaves that are neighbours (non-negligible density contributions); $\boldsymbol{x}$ can include $q < p$ missing values, which will lead to neighbours to the $(p-q)$-dimensional subspace
- `normalise()` — linear transformation of the data matrix to have any of: means 0, standard deviations 1, correlations 0
- `predict_kudzu()` — given a vector $\boldsymbol{x}$ and a kudzu list, return the kudzu density; if there are $0 < q < p$ missing values, the q dimensions will be integrated out, leading to a marginal $(p-q)$-dimensional distribution's density
- `predict_tree()` — given a vector $\boldsymbol{x}$ and a DET list, return the DET density; if there are $0 < q < p$ missing values, the q dimensions will be integrated out, leading to a marginal $(p-q)$-dimensional distribution's density
- `rkudzu()` — given a natural number n_r and a kudzu list, return n_r pseudo-random numbers from the kudzu density

[6] In a functional programming paradigm, functions can be supplied as input to other functions, and can be returned as output.

- `search_leaves()` — given a vector $\boldsymbol{x}$ and a DET list, return the index number of the leaf containing that point; if there are missing values in $\boldsymbol{x}$, return a vector of all the index numbers that intersect the subspace, as above
- `tree()` — given a data matrix and grow parameters, grow a DET; this function is intended to become more general-purpose, so at present you must specify `goal="density"` as an option, and `prune_parameters=NA`
- `tree2kudzu()` — given a DET list and a bandwidth (positive real number), return a kudzu list

There are a number of internal functions, which advanced users might find useful, are annotated thoroughly in the source code `.R` files, and can be adapted by the user under the CC-BY-4.0 license, provided that the text "©BayesCamp Ltd" is included.

Information on trees, nodes, and kudzu density functions is passed between functions as *list*s in R. Contents of these are described below. R can store these in single-object `.rds` files or multiple-object `.Rdata` files.

2.2.2 Stata / Mata

The Stata `rclass` commands are:

- `kudzu_normalise` — linear transformation of the data to have any of: means 0, standard deviations 1, correlations 0
- `kudzu_tree` — given a varlist and grow parameters, grow a DET; matrices of tops and bottoms, and a vector of densities, are in `r()` and optionally saved in a `.dta` file
- `kudzu_matrix` — given DET matrices, a bandwidth σ, and a neighbour distance ϕ, return a combined kudzu matrix including integral components, and optionally a `.dta` file
- `kudzu_predict` — given a vector $\boldsymbol{x}$ and a kudzu matrix, return the predicted density; if there are $0 < q < p$ missing values, the q dimensions will be integrated out, leading to a marginal $(p - q)$-dimensional distribution's density
- `kudzu_rng` — given a kudzu matrix and a number of desired draws, return a vector of real numbers drawn from the kudzu density function

There are two additional commands to facilitate storing kudzu matrices in `.dta` files: `kudzu_save` and `kudzu_load`. Many internal functions are well-documented in Mata and can be adapted by the user under the CC-BY-4.0 license, provided that the text "©BayesCamp Ltd" is included.

Information on trees, nodes, cutpoints, and kudzu density functions is passed between functions as *struct*s in Mata. Contents of these are described below. The

kudzu density function or DET is returned as a collection of matrices, vectors and scalars in `r()`, and can be written to a data (`.dta`) file using the `kudzu_save` Stata command, then loaded again using `kudzu_load`.

2.3 Example in R

```
# iris dataset
ir <- iris[,-5]
n <- NROW(ir)
p <- NCOL(ir)

# normalise (recommended)
nir <- normalise(ir) # returns a matrix

# round off (recommended)
nir <- round(nir,2)

# fit a maximal DET
nirtree <- tree(data=nir,
                goal="density",
                grow_parameters=list(min_n_l=3, max_nodes=150),
                prune_parameters=NA_integer_,
                dep_var=NA_integer_)

# predict density at a point from DET
predict_tree(c(-0.5, 0, 0.5, 0), nirtree)

# predict marginal density over a subspace from DET
predict_tree(c(-0.5, 0, NA, NA), nirtree)

# compact storage of kudzu density function
nirk <- tree2kudzu(nirtree, sigma=0.2, phi=6)

# predict marginal density over a subspace from kudzu density function
predict_kudzu(c(-0.5, 0, NA, NA), nirk)

# pseudo-random number generation
random_data <- rkudzu(10000, nirk)
```

2.4 Tree fitting

The `tree` function in R and `kudzu_tree` command in Stata carry out the CART algorithm[8, 14], with one notable difference from Breiman and colleagues' implementation: rather than recursively partitioning down each branch one at a time,

then moving to the next branch, we add a layer of growth to all branches where possible, then repeat.

Another difference is that our tree function requires sorting $2p$ times at the outset, and then not again. The contents of each child node in the tree are generated from the parent according to the split. This shifts computational impact from CPU to memory, which we feel is appropriate for the 21st century.

Within any node that is eligible for growth, each dimension is scanned in turn at all eligible cutpoints (midway between unique values of data), evaluating the ISE that would result from the split, and the biggest reduction in loss is selected. New child nodes are created, and the parent node is marked as no longer eligible for splitting. The user specifies some `grow_parameters` to control this, such the minimum number of observations per node.

The chosen split is the one with greatest reduction in integrated squared error, provided that it is permissible under the `grow_parameters`. The integrated squared error (ISE) loss function is used, as described by Ram & Gray [9], except that the constant term n^2 is omitted — see Formula 7. We calculate integrals of DET and kudzu leaves at the time of fitting, and store them for later use in evaluation of marginal densities.

The `tree` function/command will return a compact list/struct containing a $L \times p$ matrix of tops, another of bottoms, and a L-vector of densities. With the option `full_tree` in Stata or `full_tree=TRUE` in R, we obtain a much larger list/struct of all nodes (not just leaves), loss functions, cutpoints, and other facts allowing us to understand the tree growth in depth.

The kudzu list returned by `tree2kudzu` returns a list/struct containing $L \times p$ matrices of: tops, bottoms, `leaf_dim_integrals` univariate integral contributions of leaves, `eb` exponentiated bottoms divided by sigma, `et` exponentiated tops divided by sigma. There are also L-vectors of heights (to integrate to 1), leaf integral contributions and DET densities. There is a scalar for `sigma` and a kudzu list version number (not the software version) for future-proofing, in case the make-up of the list changes in future versions.

Other measures to reduce computation time for the DET are planned.

2.4.1 Pruning

In most implementations of trees, whether for density, classification or regression, the role of pruning is to control over-fitting. All binary tree growth algorithms are greedy algorithms, splitting each node at the locally best cutpoint (subject to some constraints, such as a minimum count of data per child node) without consideration of its value in the overall tree.

In order to be sure of having included important detail in all regions of the data space, it is necessary to let growth run through all branches, into a large number of nodes. Then, these can be reduced by comparing the loss function reduction obtained by each pair of leaves (terminal nodes) compared to their parent node, and removing those that are of least value overall. Various methods have been

proposed to do this; the natural choice in this context would be cost-complexity pruning [14, 8], with a penalised loss function:

$$\text{pISE}(\alpha) = \text{ISE} + \alpha L \tag{25}$$

where the penalty term is proportionate to the number of leaves, L, with coefficient α. Larger values of α favour trees with fewer leaves, and $\alpha = 0$ leads to the fully grown, unpruned tree. The value of α is chosen to minimise the penalised loss, using the golden section search algorithm [15]. Default initial values bracketing the solution are set to $\left\{0, \frac{\text{pISE}(\alpha=0)}{4L(\alpha=0)}, \frac{\text{pISE}(\alpha=0)}{L(\alpha=0)}\right\}$. We only consider values of α to a certain degree of precision, to avoid diminishing returns in the optimisation.

In kudzu, the smoothing and the ISE both also act to prevent overfitting, and so the role of pruning is less critical. In the software implementations at present, for this reason, only the last two "generations" of nodes are considered for removal, rather than tracing up through previous generations. We sort pairs of leaves by the loss reduction that they contribute, compared to their parent node. Then, we remove them in descending order of benefit (and with respect to the generations) and select the pruned tree from this list that minimises $\text{pISE}(\alpha)$.

Deeper pruning — removing more generations of nodes — could lead to a better tree, but has a computational impact, because details of previous generations of nodes must be either be stored in memory (including three $n \times p$ matrices, one of floating point elements, the others of integers), or recalculated after having been previously discarded during tree growth.

Because smoothing and pruning are very different processes, applied one after the other, it is not straightforward to quantitatively balance the two, and given the highly non-linear nature of trees, $\{\alpha, \sigma\}$ seem unlikely to jointly have a convex and unimodal loss surface. However, there may be scope to explore this as further methodological work using cross-validation.

2.5 Normalisation

In practice, tree-based models suffer from one major constraint to their model space: the requirement for splits to be parallel to an axis. This means that data distributions that are correlated will require more leaves (potentially *many* more leaves in high dimensions) to reach adequate loss function value, compared to the same data, rotated to be uncorrelated. For this reason, we suggest that users consider rotating all datasets prior to fitting a kudzu density function.

There are other computational challenges that can be overcome at the same time. If the values that the data assume vary greatly between dimensions, then the volumes and densities of leaves may take extremely large or small values and lead to digital rounding error, or even overflow errors and unexpected program exit. Datasets that are normalised to have mean 0 and standard deviation 1 in each dimension, and no correlations, will avoid these problems, and this is the procedure

implemented in the `normalise()` R function and `normalise` Stata command. Users can also carry out their own normalisation and bypass the process.

We do not attempt dimension reduction as part of the normalisation process, though users should consider doing this before starting if there are strong correlations in the data, and the potential for redundancy in the dimensions. There are settings where this is not possible, such as Bayesian updating; we cannot use a $(p' < p)$-dimensional density estimate for a p-dimensional prior probability density.

A ramification of normalising data is that observations which shared values, to some precision, in one dimension, will almost certainly not do so after rotation. This introduces many more potential cutpoints for tree-fitting to consider. Rounding to an acceptable level of precision is a simple way to control this.

Finally, we should consider the structure of associations among the dimensions in our data. In the context of Bayesian posteriors and priors, where the variables are unknowns in the model — this was the original motivation for this work — there are likely to be some dimensions uncorrelated with others, which can be estimated marginally with a univariate distribution (a smoothed histogram, a parametric distribution fit, or even a kernel density), and others which are correlated within a block of dimensions (a clique of mutual connections via non-negligible correlation), which should be estimated jointly by kudzu. This can massively reduce the dimensionality of the problem.

Association is not just correlation but also the shape of the pairwise bivariate marginal distribution. When data are convex and unimodal within each block of dimensions, it can be normalised to be uncorrelated. Kudzu may not even be needed for such blocks, but the difficult distributions will appear through non-linearity, and these will require the non-parametric approach.

2.6 Kudzu tuning

ISE prevents, to some extent, overfitting [1]. In kudzu density functions, ISE could be useful for selecting the bandwidth σ. There may be scope for other tuning parameters too, yet to be explored. Whether a semi-automatic tuning system using ISE would be useful will depend on the accessibility of any such software and the time for execution: ISE is computationally intensive in kudzu but very simple in DET.

In previous work, for Bayesian updating specifically, we mixed a small amount of large-variance background MVN density with the kudzu. This is a performance consideration to aid the computer in some uses of kudzu, and therefore should be chosen by the user rather than being fitted to the data at hand (training set). The same applies to the weight given to it in the overall kudzu density function. We suggest a reasonable but heuristic starting value to consider its covariance matrix might be a scalar multiple of the empirical covariance matrix of the data.

For example, in the context of predicting probabilities of future samples of data, it would be unrealistic to expect no data outside the current dataset's leaves ($\pm\phi\sigma$). Extending the outer faces of the tree (bounding box) by additional delta-translation

outward may suffice for this, or a background distribution could allow for about 1% of probability between h_j and $1.5h_j$, outside the bounding box.

When a kudzu density function is used for a prior distribution in MCMC sampling for Bayesian inference, on the other hand, the role of the background distribution is to provide gradient toward the leaves, to guide back the sampling algorithm if it strays, and so a much wider background distribution is justified.

2.7 Future developments

The code is shared on a personal website and will continue to be made available publicly, with a Creative Commons license. As the Stata code matures, it will be offered to the SSC repository. The tree code is structured with further generalisations in mind, notably extension to classification and regression problems. Kudzu smoothing may be useful in these settings too. It is also an ambition to allow a user-supplied loss function, and to explore the possibility of speeding up loss function evaluation, where the loss function is additive over the observed data's moments, by adjusting from one potential cutpoint to the next, rather than recalculating.

Translation to other languages should be relatively straightforward because of our use of basic functions only, and no third-party packages. Languages can follow either the R or Mata code closely, depending on their functional, object-oriented, statically or dynamically typed, and declarative preferences. Python and Julia are the other most popular open-source programming languages in the data science and statistics field [16]. A C++ implementation will allow for a faster (but less open) R version and a command line interface.

There may be utility in offering options for defining the bounding box. At present, it is the smallest hypercuboid whose extremes are equal to, or contain, the data.

Availability of RAM is particularly threatened during tree growth, where multiple versions of $n \times p$ matrices are stored. A sparse representation may be helpful for child nodes, where some entries are blanked out. Also, it may be sensible to store all unique values (floating point numbers) and then only refer to those by their indices (integers). A distributed version of DET building from databases might be useful, considering the relevance of these methods to large and rapidly arriving datasets.

2.8 Results: fidelity to simulated data

We see a good match good match to MVN at $p = 5$(Figure 3), which deteriorates gradually as p increases (in that it will require higher L and therefore higher n too. Note that correlated convex data like this can be rotated by the `normalise` function to be uncorrelated. A next phase of assessment will subjectively estimate the L and p required for an acceptable density function fit.

Multimodal mixtures of p MVNs with mean at $(1, 0, \ldots 0)$, $(0, 1, \ldots 0)$, ..., $(0, 0, \ldots 1)$ are also fitted very well (not shown here), because the arrangement of data on the axes is amenable to orthogonal tree modelling.

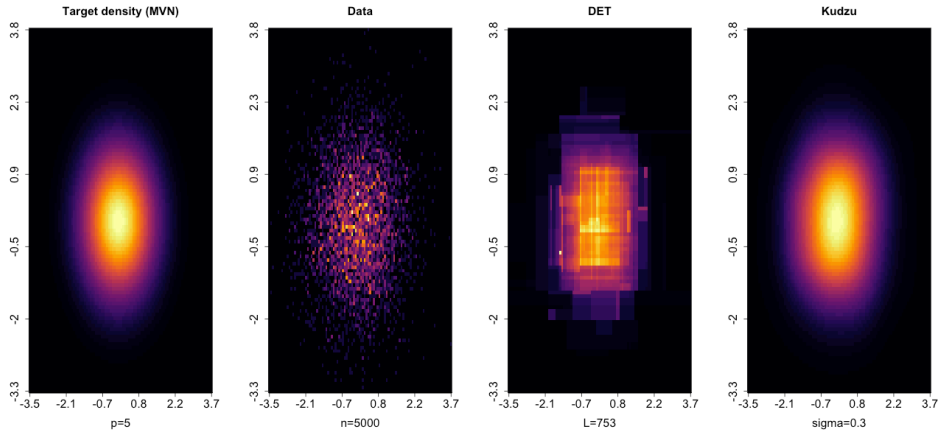

Figure 3: DET and kudzu marginal densities, evaluated on a 100×100 grid, for simulated multivariate normal (uncorrelated) data.

The banana distribution is another valuable test, as it is not amenable to rotation. We know that an orthogonal tree model will struggle with correlated data that lies diagonally to the axes, and curves. We can see that fidelity to the 5-dimensional banana in Figure 4 is not as good as for the MVN. This problem is addressed with a kudzu ensemble model in Section 3.

3 Assessment of kudzu ensembles

This section describes preliminary investigations at the time of writing. There is no simple function yet to fit and store kudzu ensembles, or estimate their densities and draw pseudo-random numbers from them. Bootstrap aggregation has been considered. The possibility of a boosted ensemble will be investigated next, but as density estimation is unsupervised learning, this will have to rely on some estimate of local error.

3.1 Bootstrap aggregated kudzu ensemble

We fit a set of 200 DETs and their kudzu densities to the banana data from Section 2.8. In each, n observations are sampled with replacement. Notably, we do not "decorrelate" these resamples as in the Random Forests algorithm (fit each tree considering only a random subset of dimensions for cutpoint candidates), though that could be investigated next.

The DETs must manually check whether a point for marginal density estimation is outside the bounding box of that particular resample, and if so, insert a density of zero.

We average all the predicted marginal densities at each point and plot them in Figure 5. Performance is much improved on Figure 4 with one single kudzu

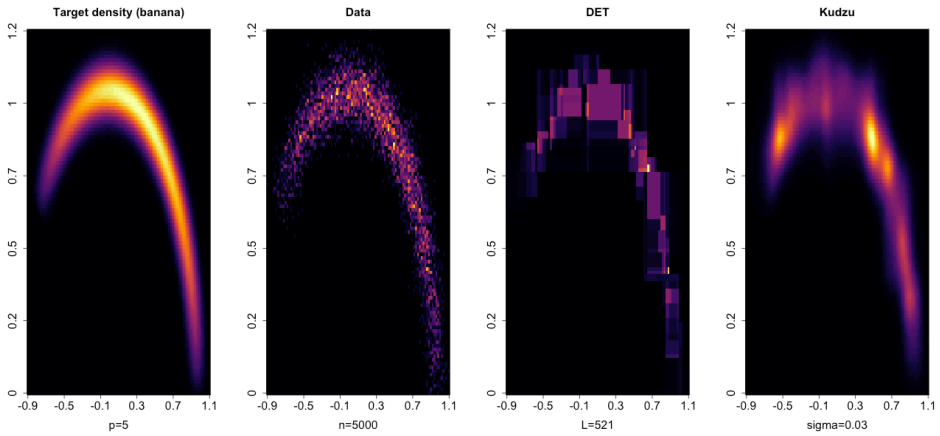

Figure 4: DET and kudzu marginal densities, evaluated on a 100×100 grid, for simulated multivariate data from a banana distribution.

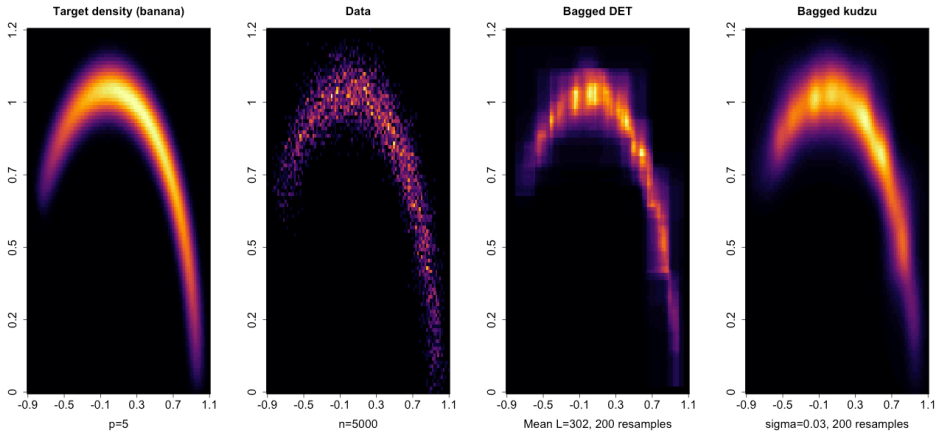

Figure 5: Averaged marginal densities over a bootstrap aggregated (bagged) DET and kudzu ensemble of 200 resamples, with the banana simulated data. Compare with Figure 4.

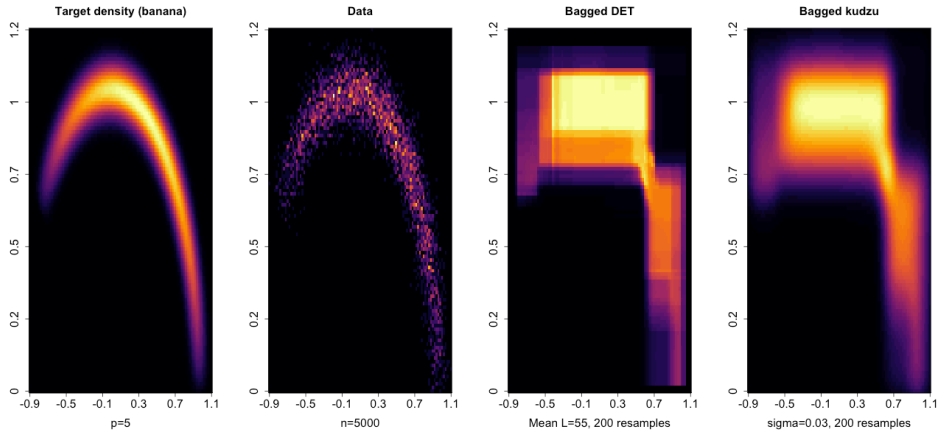

Figure 6: A bootstrap aggregated (bagged) DET and kudzu ensemble of 200 resamples. In contrast with Figure 5, this had a mean of 55 leaves per DET and kudzu density function.

density function. This ensemble contains slightly over 60,000 leaves, which is not challenging for storage or computation of the predicted densities, and yet provides good fidelity, seen here on a grid of 100 points per axis.

Importantly, the individual DETs must be maximally complex in order to achieve this fidelity. In Figure 6, we see the same ensemble construction, on the same data, but with DETs limited to be smaller, with an average of 55 leaves per DET / kudzu density function. The lesson of this is that the methods that are successful for ensembles of trees in classification and regression tasks are not necessarily transferable to the density estimation setting. The complexity of the DETs removes the benefit of "decorrelation".

Appendix: formulas for integrals of terms in the square of the kudzu density

Two unique edge values, including diagonal elements

Each leaf is a product of $2p$ factors, so its square becomes a product of $4p$ factors, separable into dimensions as:

$$
\begin{aligned}
&\int_{\mathcal{X}} \hat{f}^2_{\text{leaf}}(\boldsymbol{x}|\ell)d\boldsymbol{x} \\
=&\hat{f}_{\text{DET}}(\boldsymbol{x}|\ell)\hat{f}_{\text{DET}}(\boldsymbol{x}|\ell')\int_{\mathcal{X}}\prod_{j=1}^{p}\left(\frac{1}{1+e^{\frac{\mu_{b\ell j}-x_j}{\sigma}}}\right)^2\left(\frac{1}{1+e^{\frac{x_j-\mu_{t\ell j}}{\sigma}}}\right)^2 d\boldsymbol{x} \\
\propto&\prod_{j=1}^{p}\int_{x_j}\left(\frac{1}{1+e^{\frac{\mu_{b\ell j}-x_j}{\sigma}}}\right)^2\left(\frac{1}{1+e^{\frac{x_j-\mu_{t\ell j}}{\sigma}}}\right)^2 dx_j
\end{aligned}
\tag{26}
$$

which we can evaluate by setting $b = \frac{\mu_{b\ell j}}{\sigma}$, $t = \frac{\mu_{t\ell j}}{\sigma}$, and $z = \frac{x_j}{\sigma}$, for brevity, hence $\frac{dz}{dx} = \frac{1}{\sigma}$ and $dx = \sigma dz$, and considering the non-negligible domain, $b-\phi \leq z \leq t+\phi$:

$$
\begin{aligned}
&\int_{\mu_{b\ell j}-\phi\sigma}^{\mu_{t\ell j}+\phi\sigma}\left(\frac{1}{1+e^{\frac{\mu_{b\ell j}-x_j}{\sigma}}}\right)^2\left(\frac{1}{1+e^{\frac{x_j-\mu_{t\ell j}}{\sigma}}}\right)^2 dx_j \\
=&\,\sigma\int_{b-\phi}^{t+\phi}\left(\frac{1}{1+e^{b-z}}\right)^2\left(\frac{1}{1+e^{z-t}}\right)^2 dz \\
&\text{multiplying through to separate exponentials:} \\
=&\,\sigma e^{2t}\int_{b-\phi}^{t+\phi}\left(\frac{e^{2z}}{\left(e^z+e^b\right)^2\left(e^z+e^t\right)^2}\right)dz \\
=&\,\sigma e^{2t}\left[\frac{(e^t+e^b)}{(e^t-e^b)^3}\ln\left(\frac{e^z+e^b}{e^z+e^t}\right)+\frac{(e^te^z)+(e^be^z)+2(e^te^b)}{\left(e^t-e^b\right)^2\left(e^z+e^b\right)\left(e^z+e^t\right)}\right]_{z=b-\phi}^{t+\phi}
\end{aligned}
\tag{27}
$$

This formula also applies to any pairs of leaves that happen to share two edges in a particular dimension. We must now consider the off-diagonal products, where $\ell' \neq \ell$.

Four unique edge values

When there are no shared edges, hence four unique edge values, let:

$$
\begin{aligned}
b &= \mu_{b\ell j}/\sigma \\
t &= \mu_{t\ell j}/\sigma \\
b' &= \mu_{b\ell' j}/\sigma \\
t' &= \mu_{t\ell' j}/\sigma
\end{aligned}
\tag{28}
$$

The integral for one leaf and one dimension is:

$$
\begin{aligned}
&\sigma \int_{\min(b,b')-\phi}^{\max(t,t')+\phi} \left(\frac{1}{1+e^{b-z}}\right)\left(\frac{1}{1+e^{z-t}}\right)\left(\frac{1}{1+e^{b'-z}}\right)\left(\frac{1}{1+e^{z-t'}}\right) dz \\
&= \sigma e^t e^{t'} \int_{\min(b,b')-\phi}^{\max(t,t')+\phi} \left(\frac{e^{2z}}{\left(e^z+e^b\right)\left(e^z+e^t\right)\left(e^z+e^{b'}\right)\left(e^z+e^{t'}\right)}\right) dz \\
&= \sigma e^t e^{t'} \left[\frac{e^{t'}\ln\left(e^{t'}+e^z\right)}{\left(e^{t'}-e^b\right)\left(e^{t'}-e^t\right)\left(e^{t'}-e^{b'}\right)} + \frac{e^{b'}\ln\left(e^{b'}+e^z\right)}{\left(e^{b'}-e^b\right)\left(e^{b'}-e^t\right)\left(e^{b'}-e^{t'}\right)} \right. \\
&\quad \left. + \frac{e^{t}\ln\left(e^{t}+e^z\right)}{\left(e^{t}-e^b\right)\left(e^{t}-e^{b'}\right)\left(e^{t}-e^{t'}\right)} + \frac{e^{b}\ln\left(e^{b}+e^z\right)}{\left(e^{b}-e^t\right)\left(e^{b}-e^{b'}\right)\left(e^{b}-e^{t'}\right)} \right]_{z=\min(b,b')-\phi}^{\max(t,t')+\phi} \\
&= \sigma e^t e^{t'} \left[\sum_{k\in\{b,t,b',t'\}} \frac{e^k \ln\left(\frac{e^k}{e^z}+1\right)}{\prod_{k'\neq k}\left(e^k-e^{k'}\right)} \right]_{z=\min(b,b')-\phi}^{\max(t,t')+\phi}
\end{aligned}
\tag{29}
$$

Three unique edge values, sharing a top edge

When there are three unique edge values, and a shared top, replace t' with t:

$$
\begin{aligned}
&\sigma \int_{\min(b,b')-\phi}^{t+\phi} \left(\frac{1}{1+e^{b-z}}\right)\left(\frac{1}{1+e^{z-t}}\right)^2\left(\frac{1}{1+e^{b'-z}}\right) dz \\
&= \sigma e^{2t} \int_{\min(b,b')-\phi}^{t+\phi} \left(\frac{e^{2z}}{\left(e^z+e^b\right)\left(e^z+e^t\right)^2\left(e^z+e^{b'}\right)}\right) dz \\
&= \sigma e^{2t} \left[\frac{\left(e^{b'+b}-e^{2t}\right)\ln\left(e^z+e^t\right)}{\left(e^t-e^b\right)^2\left(e^t-e^{b'}\right)^2} + \frac{e^{b'}\ln\left(e^z+e^{b'}\right)}{\left(e^{b'}-e^b\right)\left(e^{b'}-e^t\right)^2} \right. \\
&\quad \left. + \frac{e^{b}\ln\left(e^z+e^b\right)}{\left(e^{b}-e^t\right)^2\left(e^{b}-e^{b'}\right)} + \frac{e^t}{\left(e^t-e^b\right)\left(e^t-e^{b'}\right)\left(e^z+e^t\right)} \right]_{z=\min(b,b')-\phi}^{\max(t,t')+\phi}
\end{aligned}
\tag{30}
$$

Three unique edge values, sharing a bottom edge

For three unique edge values, and a shared bottom, replace b' with b:

$$
\begin{aligned}
&\sigma \int_{b-\phi}^{\max(t,t')+\phi} \left(\frac{1}{1+e^{b-z}}\right)^2 \left(\frac{1}{1+e^{z-t}}\right)\left(\frac{1}{1+e^{z-t'}}\right) dz \\
&= \sigma e^t e^{t'} \int_{b-\phi}^{\max(t,t')+\phi} \left(\frac{e^{2z}}{\left(e^z+e^b\right)^2\left(e^z+e^t\right)\left(e^z+e^{t'}\right)}\right) dz \\
&= \sigma e^t e^{t'} \left[\frac{\left(e^{t'+t}-e^{2b}\right)\ln\left(e^z+e^b\right)}{\left(e^b-e^t\right)^2\left(e^b-e^{t'}\right)^2} + \frac{e^{t'}\ln\left(e^z+e^{t'}\right)}{\left(e^{t'}-e^t\right)\left(e^{t'}-e^b\right)^2} \right. \\
&\quad \left. + \frac{e^t \ln\left(e^z+e^t\right)}{\left(e^t-e^b\right)^2\left(e^t-e^{t'}\right)} + \frac{e^b e^t + e^b e^{t'}}{(e^t-e^b)^2(e^{t'}-e^b)^2(e^{t'}-e^t)(e^z+e^b)} \right]_{z=\min(b,b')-\phi}^{\max(t,t')+\phi}
\end{aligned} \tag{31}
$$

Three unique edge values, one top and one bottom coincide

Finally, for three unique edge values, where one top and one bottom coincide:

$$
\begin{aligned}
&\sigma \int_{\min(b,b')-\phi}^{\max(t,t')+\phi} \left(\frac{1}{1+e^{b-z}}\right)\left(\frac{1}{1+e^{z-t}}\right)\left(\frac{1}{1+e^{t-z}}\right)\left(\frac{1}{1+e^{z-t'}}\right) dz \\
&= \sigma e^t e^{t'} \int_{\min(b,b')-\phi}^{\max(t,t')+\phi} \left(\frac{e^{2z}}{\left(e^z+e^b\right)\left(e^z+e^t\right)^2\left(e^z+e^{t'}\right)}\right) dz \\
&= \sigma e^t e^{t'} \left[\frac{\left(e^{t'+b}-e^{2t}\right)\ln\left(e^z+e^t\right)}{\left(e^t-e^b\right)^2\left(e^t-e^{t'}\right)^2} + \frac{e^{t'}\ln\left(e^z+e^{t'}\right)}{\left(e^{t'}-e^b\right)\left(e^{t'}-e^t\right)^2} \right. \\
&\quad \left. + \frac{e^b \ln\left(e^z+e^b\right)}{\left(e^b-e^t\right)^2\left(e^b-e^{t'}\right)} + \frac{e^{b+t+t'}+e^{3t}}{(e^t-e^b)^2(e^{t'}-e^t)^2(e^z+e^t)} \right]_{z=\min(b,b')-\phi}^{\max(t,t')+\phi}
\end{aligned} \tag{32}
$$

References

[1] D. W. Scott, *Multivariate density estimation*. Wiley, 2e ed., 2015.

[2] S. D. Team, "Stan modeling language user's guide and reference manual, version 2.34," 2024.

[3] P. Fearnhead, J. Bierkens, M. Pollock, and G. O. Roberts, "Piecewise deterministic Markov processes for continuous-time Monte Carlo," *Statistical Science*, vol. 33, pp. 386–412, 2018.

[4] W. A. Sutherland, *Introduction to metric and topological spaces*. Oxford University Press, 1975.

[5] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," *Advances in Neural Information Processing Systems*, vol. 30, pp. 2335–2344, 2017.

[6] Q. Liu, J. Xu, R. Jiang, and W. H. Wong, "Density estimation using deep generative neural networks," *Proceedings of the National Academy of Sciences*, vol. 118, p. e2101344118, 2021.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009.

[8] L. Breiman, J. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Chapman & Hall, 1984.

[9] P. Ram and A. Gray, "Density estimation trees," *KDD '11: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 627–635, 2011.

[10] B. Silverman, *Density estimation*. Chapman & Hall, 1986.

[11] R. R. Curtin, M. Edel, O. Shrit, S. Agrawal, S. Basak, J. J. Balamuta, R. Birmingham, K. Dutt, D. Eddelbuettel, R. Garg, S. Jaiswal, A. Kaushik, S. Kim, A. Mukherjee, N. Gnanasai, N. Sharma, Y. S. Parihar, R. Swain, and C. Sanderson, "mlpack 4: a fast, header-only C++ machine learning library," *Journal of Open Source Software*, vol. 8, no. 82, p. 5026, 2023.

[12] R. L. Grant, "Non-parametric Bayesian updating and windowing with kernel densities and the kudzu algorithm," *International Journal of Computational Economics and Econometrics*, vol. 12, no. 4, pp. 405–428, 2022.

[13] J. Backfield, *Becoming Functional*. O'Reilly, 2014.

[14] Givens and Hoeting, *Computational Statistics*. Wiley, second ed., 2013.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes*. Cambridge University Press, third ed., 2007.

[16] R. Muenchen, "r4stats.com: Update to data science software popularity." https://r4stats.com/2023/06/07/update-to-data-science-software-popularity/, 2023.