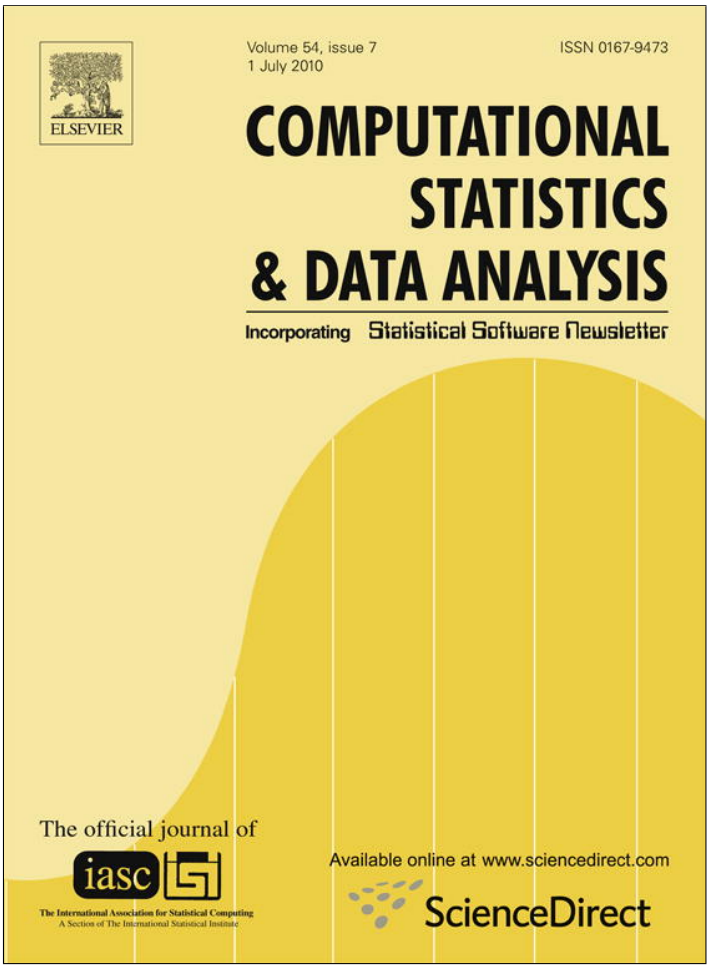

Computational Statistics and Data Analysis 54 (2010) 1707–1718

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Fast kernel conditional density estimation: A dual-tree Monte Carlo approach

Michael P. Holmes *, Alexander G. Gray, Charles Lee Isbell Jr.

College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA

ARTICLE INFO

Article history:
Received 26 February 2009
Received in revised form 7 January 2010
Accepted 8 January 2010
Available online 22 January 2010

Keywords:
Kernel conditional density estimation
Fast algorithms
Large datasets
Scalability
Dual-tree
Monte Carlo

ABSTRACT

We describe a fast, data-driven bandwidth selection procedure for kernel conditional density estimation (KCDE). Specifically, we give a Monte Carlo dual-tree algorithm for efficient, error-controlled approximation of a cross-validated likelihood objective. While exact evaluation of this objective has an unscalable $O(n^2)$ computational cost, our method is practical and shows speedup factors as high as 286,000 when applied to real multivariate datasets containing up to one million points. In absolute terms, computation times are reduced from months to minutes. This enables applications at much greater scale than previously possible. The core idea in our method is to first derive a standard deterministic dual-tree approximation, whose loose deterministic bounds we then replace with tight, probabilistic Monte Carlo bounds. The resulting Monte Carlo dual-tree algorithm exhibits strong error control and high speedup across a broad range of datasets several orders of magnitude greater in size than those reported in previous work. The cost of this high acceleration is the loss of the formal error guarantee of the deterministic dual-tree framework; however, our experiments show that error is still amply controlled by our Monte Carlo algorithm, and the many-order-of-magnitude speedups are worth this sacrifice in the large-data case, where cross-validated bandwidth selection for KCDE would otherwise be impractical.

1. Introduction

Conditional density estimation models the probability density $f(y|\mathbf{x})$ of a random variable y given a random vector $\mathbf{x}$. For example, in Fig. 1 each contour line perpendicular to the x axis represents a conditional density. This can be viewed as a generalization of regression: in regression we estimate the expectation $E[y|\mathbf{x}]$, while in conditional density estimation we model the full distribution. Fig. 1 illustrates a conditional bimodality such that $E[y|\mathbf{x}]$ is insufficiently descriptive for many tasks. Estimating conditional densities is much harder than regression, but having the full distribution is powerful because it allows one to extract almost any quantities of interest, including expectations, modes, prediction intervals, outlier boundaries, samples, expectations of non-linear functions of y, etc. Conditional densities also facilitate data visualization and exploration. Conditional density estimates are of fundamental utility, applicable to such problems as time series prediction, static regression with prediction intervals, learning in Bayes nets and other graphical models, and so on. The estimation problem is challenging, however, because the data from which $f(y|\mathbf{x})$ must be learned generally do not include any exact $\mathbf{x}$ for which $f(y|\mathbf{x})$ will be queried.

Nonparametric kernel techniques address this issue by interpolating between the points that have been seen, without strong assumptions on distributional forms. In nonparametric conditional density estimation, we make only minimal

* Corresponding author.
E-mail addresses: mpholmes@gmail.com (M.P. Holmes), agray@cc.gatech.edu (A.G. Gray), isbell@cc.gatech.edu (C.L. Isbell Jr.).

0167-9473/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2010.01.011

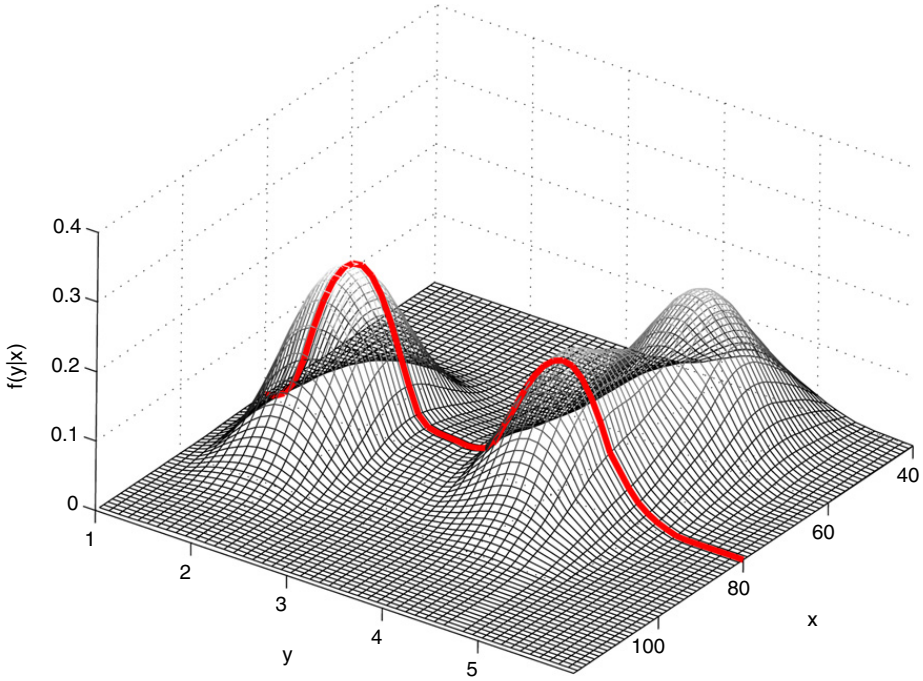

Fig. 1. Distribution $f(y, x)$ for which $f(y|x)$ can be either bimodal or unimodal, depending on x. The bold curve represents $f(y|x = 80)$.

assumptions about the smoothness of $f(y|\mathbf{x})$ without assuming any parametric form. Freedom from parametric assumptions is very often desirable when dealing with complex data, as we rarely have knowledge of true distributional structure. Nonparametric conditional density estimation has received some attention from statisticians and econometrics researchers (Gooijer and Zerom, 2003; Fan and Yim, 2004; Hansen, 2004; Bashtannyk and Hyndman, 2001; Hyndman et al., 1996; Rosenblatt, 1969), though relatively little when compared to nonparametric regression. Perhaps the main obstacle to wider adoption has been its computational cost, which is the problem addressed in this paper. Note that what we mean by nonparametric conditional density estimation is different from other techniques with similar names, such as conditional probability estimation (which refers to outputting class probabilities in the classification setting, also referred to as class-conditional probabilities).

In the present work, we consider the standard kernel conditional density estimator that first received serious attention in the work of Fan et al. (1996) and Hyndman et al. (1996), though it was originally proposed by Rosenblatt (1969). This is a direct kernel estimator of conditional densities, as opposed to approaches that separately estimate $f(y, \mathbf{x})$ and $f(\mathbf{x})$, which are combined to estimate $f(y|\mathbf{x}) = f(y, \mathbf{x})/f(\mathbf{x})$ (see Stender, 2006). Direct estimation of conditional densities allows parameter estimation to be formulated as the optimization of a single, unified objective function, whereas separate estimation of $f(y, \mathbf{x})$ and $f(\mathbf{x})$ optimizes two different objective functions that may not produce the highest-quality *conditional* densities.

Although the direct estimator we use is consistent given mild conditions on its bandwidths, practical use has been hampered by the lack of an efficient data-driven bandwidth selection procedure, upon which any kernel estimator depends critically. We propose a new method for efficiently selecting bandwidths to maximize cross-validated likelihood, an objective with some advantages over the squared-error criteria used in prior work. The speedup of this method is obtained by combining Monte Carlo techniques with a dual-tree-based approximation (see Gray and Moore, 2000) of the likelihood function. This approximation approach belongs to a new class of multi-tree Monte Carlo methods (Holmes, 2009). We present two versions of fast likelihood approximation, one analogous to previous dual-tree algorithms with deterministic error control, which gives speedups on the order of 1.5–10-fold in our experiments, and the other with a new, probabilistic Monte Carlo error control mechanism, which gives much larger speedups—as high as 286,000-fold on one million points.

With this fast learning procedure we can address datasets that are both higher in dimension and several orders of magnitude larger in size than those reported in previous work, which has been confined to bivariate datasets of size no greater than 1000 (Fan and Yim, 2004). We present results that validate the accuracy and speedup of our likelihood approximation on real datasets possessing a variety of sizes and dimensionalities. Most of these datasets were previously impractical to address with naively computed data-driven techniques. Thus, our fast bandwidth optimization method enables applications at scales that were previously unreachable. We conclude that kernel conditional density estimation is a powerful technique that is made substantially more efficient by our fast approximate optimization procedure, with many opportunities for application in a variety of statistical fields.

In the remainder of the paper we first describe the standard kernel conditional density estimator; this is followed by a discussion of the bandwidth selection problem and our choice of likelihood cross-validation as a bandwidth selection objective, at which point we derive our dual-tree approximation algorithms (both deterministic and Monte Carlo), show experimental performance, and conclude with a summary of results and implications.

2. Kernel conditional density estimation

We begin by drawing an analogy between the unconditional case of kernel density estimation (KDE), and the conditional case of KCDE. In the case of KDE with a single scalar bandwidth,[1] we estimate a density $f(\mathbf{x})$ from a dataset $\{\mathbf{x}_i\}$ by $\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_i K_h(\|\mathbf{x} - \mathbf{x}_i\|)$, where:

- $K_h(t) = \frac{1}{h^d}K\left(\frac{t}{h}\right)$,
- K is a kernel function, i.e., a compact, symmetric probability distribution such as the Gaussian or Epanechnikov kernels,
- d is the dimension of x,
- n is the number of data points,
- h is the bandwidth controlling the kernel widths (see Silverman, 1986).

Kernels allow us to interpolate between the data we have seen in order to predict the density at points we have not seen.

In kernel conditional density estimation, we estimate from a dataset $\{\mathbf{x}_i, y_i\}$ the set of all conditional distributions $f(y|\mathbf{x})$, rather than just $f(\mathbf{x})$. This means we have essentially a separate *unconditional KDE* problem for each possible vector $\mathbf{x}$. However, given a particular query point $\mathbf{x}^*$ for which we wish to know $f(y|\mathbf{x}^*)$, our dataset will most likely *not* contain any points with that precise value of $\mathbf{x}^*$. Thus, we have no direct training data for the individual KDE problem of that particular $f(y|\mathbf{x}^*)$. A solution to this problem is to borrow the y_i training values that are paired with values $\mathbf{x}_i$ other than the queried $\mathbf{x}^*$, and weight them according to how close their $\mathbf{x}_i$ are to $\mathbf{x}^*$. This is illustrated in Fig. 2. The borrowed y_i form a weighted training set on which something similar to KDE can be performed, finally yielding an estimate of $f(y|\mathbf{x}^*)$. Because interpolation occurs in both $\mathbf{x}$ and y, this leads to a double-kernel estimator:

$$\hat{f}(y|\mathbf{x}) = \frac{\sum_i K_{h_1}(y - y_i)K_{h_2}(\|\mathbf{x} - \mathbf{x}_i\|)}{\sum_i K_{h_2}(\|\mathbf{x} - \mathbf{x}_i\|)}. \tag{1}$$

Because of its similarity to the Nadaraya–Watson kernel regression estimator, this form is known as the Nadaraya–Watson (NW) conditional density estimator (Gooijer and Zerom, 2003). For a queried $\mathbf{x}$, it constructs a density by weighting each y_i proportionally to the proximity of the corresponding $\mathbf{x}_i$. Note that there are now two bandwidths, h_1 for the y kernel and h_2 for the $\mathbf{x}$ kernel.

The NW estimator is consistent provided $h_1 \to 0$, $h_2 \to 0$, and $nh_1h_2 \to \infty$ as $n \to \infty$ (Hyndman et al., 1996). Some work in the statistics and econometrics literature has produced extensions to the NW estimator, most notably by the addition of local polynomial smoothing (Fan et al., 1996; Fan and Yim, 2004; Gooijer and Zerom, 2003). For bandwidth selection, both reference rules and data-driven procedures have been proposed, but all applications appear to have been confined to the bivariate case, as has most of the theoretical analysis. This is understandable in light of the difficulty of selecting good bandwidths for large, high-dimensional datasets: theoretical analysis is difficult, reference rules and other closed-form methods tend to rely on unrealistic assumptions, and exact computation of data-driven objective functions is prohibitively expensive.

3. Bandwidth selection

As with all kernel estimators, the performance of the NW estimator depends critically on a suitable choice for the bandwidths h_1 and h_2. The aforementioned consistency conditions provide little guidance in the finite-sample setting. Bandwidth selection has always been a dilemma: on the one hand, asymptotic arguments and reference distributions lead to plug-in and reference rules whereby bandwidths can be efficiently calculated, but these perform poorly on finite samples and when reference distributions do not match reality; on the other hand, data-driven selection criteria give good bandwidths but are naively intractable on datasets of appreciable size. We propose a middle road that captures some of the advantage of each approach by being both efficient and data-driven.

The only data-driven bandwidth score to previously appear in the KCDE literature is the mean integrated squared error in the following form:

$$\mathrm{MISE}(h_1, h_2) = \int (f(y|\mathbf{x}) - \hat{f}(y|\mathbf{x}))^2 \mathrm{d}y f(\mathbf{x})\mathrm{d}\mathbf{x}. \tag{2}$$

As shown by Fan and Yim (2004), minimizing MISE is equivalent to minimizing

$$\int (\hat{f}(y|\mathbf{x}))^2 \mathrm{d}y f(\mathbf{x})\mathrm{d}\mathbf{x} - 2\int \hat{f}(y|\mathbf{x}) f(y, \mathbf{x})\mathrm{d}y\mathrm{d}\mathbf{x}\,. \tag{3}$$

[1] More generally, we can optimize a $d \times d$ matrix of bandwidths.

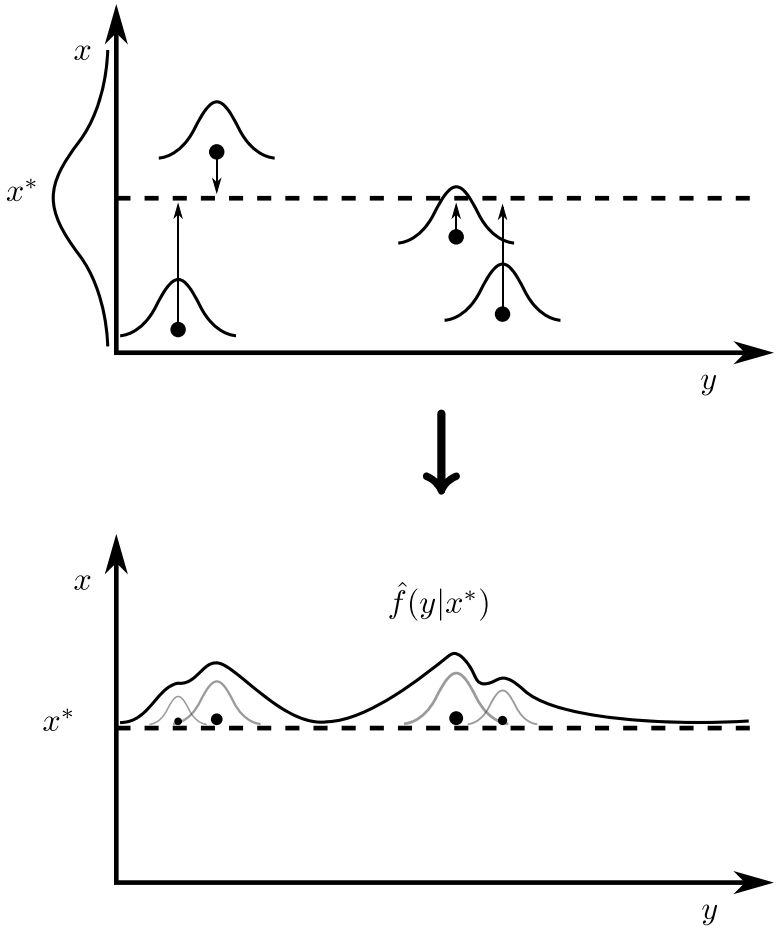

Fig. 2. KCDE requires interpolation in both x and y. To estimate $\hat{f}(y|x^*)$, we place a kernel on each data point (x_i, y_i), and each of these kernels is weighted by a second kernel centered on x^*. This gives a weighted set of kernels on the y_i, which we use to construct the conditional estimate $\hat{f}(y|x^*)$.

A consistent, cross-validated estimate of the MISE is obtained from

$$\widehat{\text{MISE}} = \frac{1}{n}\sum_i \int (\hat{f}^{-i}(y|\mathbf{x}_i))^2 \text{dy} - \frac{2}{n}\sum_i \hat{f}^{-i}(y_i|\mathbf{x}_i), \tag{4}$$

where $\hat{f}^{-i}$ denotes $\hat{f}$ evaluated with $(\mathbf{x}_i, y_i)$ left out.

Though conceptually and theoretically appealing, $\widehat{\text{MISE}}$ expands to include a triply-nested summation, giving a base computational cost of $O(n^3)$. While this could still be used as the starting point for an efficient approximation, we choose to start with another criterion that has lower base complexity: likelihood cross-validation.

Likelihood cross-validation is well known in standard kernel density estimation (see Silverman, 1986; Gray and Moore, 2003b), but has yet to be used for KCDE. In kernel density estimation, likelihood cross-validation has the important property of minimizing the Kullback–Leibler divergence between the estimated and true densities, which in many scenarios can be as good or better than minimizing a squared-error criterion, e.g., when smoother distributions are desired. A drawback to the likelihood function is its potential non-robustness to outliers, particularly in the presence of heavy-tailed distributions (Silverman, 1986); however, our empirical observation is that it generally performs quite well. Since the likelihood is also computationally cheaper than MISE, we therefore choose the likelihood as a less expensive (though, at $O(n^2)$, still unscalable) starting point, and focus the rest of the paper on developing a fast likelihood algorithm.

We define the cross-validated log-likelihood for KCDE as

$$L(h_1, h_2) = \frac{1}{n}\sum_i \log(\hat{f}^{-i}(y_i|\mathbf{x}_i)\hat{f}^{-i}(\mathbf{x}_i)), \tag{5}$$

where $\hat{f}(\mathbf{x})$ is the standard kernel density estimate over $\mathbf{x}$ using the bandwidth h_2 from $\hat{f}(y|\mathbf{x})$. As with the MISE, this criterion attempts to optimize the conditional density $\hat{f}(y|\mathbf{x})$, with each contribution being weighted by $\hat{f}(\mathbf{x})$ so that the more frequent conditional distributions have more influence on the choice of bandwidth. Although not formulated as such, $\hat{f}(y|\mathbf{x})\hat{f}(\mathbf{x})$ can also be interpreted as an estimate of $f(\mathbf{x}, y)$. Maximizing the KCDE likelihood will therefore also minimize the

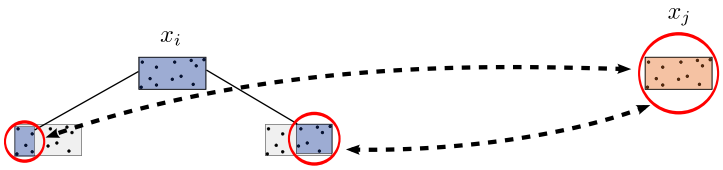

Fig. 3. In a dual-tree algorithm, we successively refine a tree-based partitioning of the data. At each point in the refinement, we test whether each pair of nodes can be approximated without further recursion. If the answer is yes, we use the approximation; otherwise we continue the refinement.

Kullback–Leibler divergence between this implicit $\hat{f}(\mathbf{x}, y)$ and the true $f(\mathbf{x}, y)$ (see Silverman, 1986). Since each leave-one-out term is $O(n)$, the likelihood score is naively computable in $O(n^2)$ time. Though less costly than the MISE, this remains unscalable in large-data settings. We take a dual-tree approach to make it fast.

3.1. Dual-tree approximation

Dual-tree recursion is a spatial-partitioning approach for error-controlled acceleration of computations that require the calculation of all pairwise distances. It has previously been used to accelerate a variety of computations such as finding all nearest neighbors, kernel density estimation, and two-point correlations. We present here a brief overview of the methodology, and refer the reader to the rich vein of work on dual-tree and multi-tree methods for greater detail (Gray and Moore, 2000; Moore et al., 2000; Gray and Moore, 2003a,b,c; Gray et al., 2004; Gray and Moore, 2004; Gray, 2004; de Freitas et al., 2005; Klaas et al., 2005a,b; Shen et al., 2006; Gray and Riegel, 2006; Klaas et al., 2006; March and Gray, 2007; Boyer et al., 2007; Riegel et al., 2008).

For the simple case of a double summation $\sum_i \sum_j g(\mathbf{x}_i, \mathbf{x}_j)$ over the data, the essential idea is that we can partition the set of pairs $(\mathbf{x}_i, \mathbf{x}_j)$ into subsets within which the values $g(\mathbf{x}_i, \mathbf{x}_j)$ are approximately constant. For each subset r, rather than explicitly computing $g(\mathbf{x}_i, \mathbf{x}_j)$ for every pair, we can simply approximate it by some estimate $\hat{g}_r$. Speedup is gained by the fact that we do a single estimate per subset, rather than evaluating all pairs; e.g., if there are s subsets, our computation becomes $O(s)$ rather than $O(n^2)$. Error is controlled for the global computation by asking locally, for each subset r, whether the error induced by approximating all of the $g(\mathbf{x}_i, \mathbf{x}_j)$ by $\hat{g}_r$ is sufficiently low. If the answer is no, the subset is divided and the error test is repeated recursively.

Suppose the data $\{\mathbf{x}_i\}$ are partitioned into subsets $r \in R$. We can write

$$\sum_i \sum_j g(\mathbf{x}_i, \mathbf{x}_j) = \sum_{r_i \in R} \sum_{r_j \in R} g(r_i, r_j), \tag{6}$$

where $g(r_i, r_j) = \sum_{i \in r_i} \sum_{j \in r_j} g(\mathbf{x}_i, \mathbf{x}_j)$. If, for a given pair (r_i, r_j), we can determine that $g(\mathbf{x}_i, \mathbf{x}_j)$ lies within sufficiently narrow bounds for all $\mathbf{x}_i \in r_i$ and $\mathbf{x}_j \in r_j$, then we can approximate by assuming all pairs in (r_i, r_j) have a value within those bounds, without calculating each term explicitly. The bounds on $g(\mathbf{x}_i, \mathbf{x}_j)$ can be used to bound the error induced by this approximation.

Algorithm 1 Generic dual-tree recursion.

DUALTREE
Input: nodes r_i and r_j; error tolerance ϵ
Output: approximate contribution from r_i and r_j to overall computation

1. **if** CANAPPROXIMATE(r_i, r_j, ϵ)
   (a) **return** APPROXIMATE(r_i, r_j, ϵ)
2. **if** ISLEAF(r_i) and ISLEAF(r_j)
   (a) **return** DUALTREEBASE(r_i, r_j)
3. **else**
   (a) **return** DUALTREE($r_i.lc$, $r_j.lc$, ϵ) $\oplus$ DUALTREE($r_i.lc$, $r_j.rc$, ϵ) $\oplus$ DUALTREE($r_i.rc$, $r_j.lc$, ϵ) $\oplus$ DUALTREE($r_i.rc$, $r_j.rc$, ϵ)

In a dual-tree recursion (see Algorithm 1) we produce partitions R_i and R_j over $\{\mathbf{x}_i\}$ and $\{\mathbf{x}_j\}$ by traversing two separate kd-trees over the data.[2] See Fig. 3 for an illustration. R_i and R_j contain the set of leaf nodes from each tree, initially consisting of only the two root nodes. Note that every kd-tree node provides a tight bounding box for the points it contains. Starting with the roots, the algorithm considers each node pair (r_i, r_j) to determine whether its contribution to the overall sum can

[2] A kd-tree is a type of binary space-partitioning tree used to speed up various kinds of computations. Trees of other types can also be used; see Moore et al. (2000) for details.

be approximated (step 1). If so, we make the approximation and prune that branch of the recursion (step 1a). If not, the two nodes are each replaced by their children, resulting in four recursive node–node comparisons (step 3a), unless both nodes are leaves (step 2), in which case further refinement is impossible and we are forced to do the brute-force computation for that pair (step 2a). Note that the mechanism for merging the results of recursive subdivisions, as represented by the $\oplus$ operator in the algorithm (step 3a), depends on the form of the function being approximated.

In order to apply the dual-tree methodology to a new problem, we must:

1. specify a node–node approximation function (APPROXIMATE in Algorithm 1),
2. specify the merge operator ($\oplus$ in Algorithm 1),
3. derive a "pruning rule" for deciding whether to approximate or recurse (CANAPPROXIMATE in Algorithm 1).

In the dual-tree framework, approximation error should be guaranteed to fall within a threshold ϵ set by the user. The approximation and pruning scheme must therefore be derived in such a way that local pruning decisions will lead to the desired global error guarantee.

3.2. The dual-tree approach for fast cross-validated likelihood

We now derive a dual-tree-based approximation to the cross-validated likelihood L. Note that we use Δ to denote the absolute error in the estimate of a quantity, e.g., $\Delta L \equiv |L - \widehat{L}|$. The goal will be to guarantee the following error bound:

$$\Delta L \leq \epsilon, \tag{7}$$

where the error tolerance ϵ is specified by the user. The main steps in deriving our algorithm are to specify the CANAPPROXIMATE and APPROXIMATE functions of Algorithm 1, along with an appropriate way of merging approximations from recursive subdivisions (step 3a of Algorithm 1). We begin by noting that, upon expansion of the $\hat{f}^{-i}$ terms, we can write

$$\begin{aligned}
L &= \frac{1}{n}\sum_i \log(\hat{f}^{-i}(y_i|\mathbf{x}_i)\hat{f}^{-i}(\mathbf{x}_i)) \\
&= \frac{1}{n}\sum_i \log\left[\left(\frac{\sum_{j\neq i} K_{h_1}(y_i - y_j)K_{h_2}(\|\mathbf{x}_i - \mathbf{x}_j\|)}{\sum_{j\neq i} K_{h_2}(\|\mathbf{x}_i - \mathbf{x}_j\|)}\right)\left(\sum_{j\neq i}\frac{K_{h_2}(\|\mathbf{x}_i - \mathbf{x}_j\|)}{n-1}\right)\right] \\
&= \frac{1}{n}\sum_i \log\left(\frac{\sum_{j\neq i} K_{h_1}(y_i - y_j)K_{h_2}(\|\mathbf{x}_i - \mathbf{x}_j\|)}{n-1}\right) \\
&= \frac{1}{n}\sum_i \log(A_i) - \log(n-1)\,,
\end{aligned} \tag{8}$$

where $A_i = \sum_{j\neq i} K_{h_1}(y_i - y_j)K_{h_2}(\|\mathbf{x}_i - \mathbf{x}_j\|)$. The $\log(n-1)$ term is constant and can be ignored. We will construct our approximation to L by first constructing a list of approximate A_i terms, whose logs we then sum to get an approximation $\widehat{L}$. In order to see how the various approximation errors propagate through to the total error in $\widehat{L}$, we will make use of the following error propagation bounds, which we state as a lemma.

Lemma 1. *Let f be any function $f(x_1, \ldots, x_m)$ of m scalar arguments. If for a given argument list $(x_1, \ldots, x_m)$ we substitute a list of approximate arguments $(\hat{x}_1, \ldots, \hat{x}_m)$ such that the absolute error $|x_i - \hat{x}_i|$ in each x_i is bounded by Δx_i, the following bounds hold for $\Delta f = |f(x_1, \ldots, x_m) - f(\hat{x}_1, \ldots, \hat{x}_m)|$:*

1. *If $f = \sum_i c_i x_i$, then $\Delta f \leq \sum_i c_i \Delta x_i$.*
2. *If $f = \log(x)$, then $\Delta f \leq \log(1 + \frac{\Delta x}{|x|})$.*

We proceed by constructing a series of conditions, each of which implies $\Delta L \leq \epsilon$, and the last of which will be satisfied by our algorithm. Combining Rule 1 from Lemma 1 with Eq. (8), we have $\Delta L \leq \frac{1}{n}\sum_i \Delta\log(A_i)$, so we can guarantee $\Delta L \leq \epsilon$ by enforcing $\frac{1}{n}\sum_i \Delta\log(A_i) \leq \epsilon$, which is implied by $\forall i,\ \Delta\log(A_i) \leq \epsilon$. Invoking Rule 2 and a bit of rearrangement, this becomes equivalent to

$$\forall i, \quad \frac{\Delta A_i}{A_i} < \mathrm{e}^{\epsilon} - 1. \tag{9}$$

Note that we drop the absolute value signs on A_i because, being a sum of kernel products, it is non-negative. Now consider a partitioning R of the data. We can write

$$A_i = \sum_{r\in R}\sum_{\substack{j\in r \\ j\neq i}} v(i,j) = \sum_{r\in R} S_{ir}, \tag{10}$$

where

$$v(i,j) \triangleq K_{h_1}(y_i - y_j)K_{h_2}(\|\mathbf{x}_i - \mathbf{x}_j\|) \tag{11}$$

$$S_{ir} \triangleq \sum_{\substack{j\in r \\ j\neq i}} v(i,j). \tag{12}$$

We therefore have $\Delta A_i \leq \sum_r \Delta S_{ir}$, and our enforcement condition holds if

$$\forall i, \quad \frac{\sum_r \Delta S_{ir}}{\sum_r S_{ir}} \leq \mathrm{e}^{\epsilon} - 1. \tag{13}$$

This condition is in turn implied by

$$\forall i, r, \quad \frac{\Delta S_{ir}}{S_{ir}} \leq \mathrm{e}^{\epsilon} - 1. \tag{14}$$

Note that this is a stronger condition than necessary to imply Eq. (13), but it is at least sufficient. We summarize this intermediate result as a lemma.

Lemma 2. *Let* $\widehat{L} = \frac{1}{n}\sum_i \log(\widehat{A}_i) - \log(n-1)$ *and let* $\widehat{A}_i = \sum_{r\in R} \widehat{S}_{ir}$ *for some partitioning* R *of the data and some approximator* $\widehat{S}_{ir}$ *of* S_{ir}. *If the approximations* $\widehat{S}_{ir}$ *satisfy* $\forall i, r, \frac{\Delta S_{ir}}{S_{ir}} \leq \mathrm{e}^{\epsilon} - 1$, *then* $\Delta L \leq \epsilon$.

3.3. Deterministic approximation

Now consider two partitionings R_{outer} and R_{inner} induced by dual kd-trees on the dataset. R_{outer} represents a partitioning of the outer summation index i, while R_{inner} corresponds to the inner index j (see Eq. (8)). Recall that in the dual-tree framework (Algorithm 1), each partitioning comes from the leaves of a tree in its current state of expansion. At any point in the expansion, we examine a pair of nodes r_i and r_j, each of which contains some subset of the indices. For each A_i with $i \in r_i$, the total contribution from all indices in r_j is $S_{ir_j} = \sum_{j\in r_j, j\neq i} v(i,j)$. We can bound the terms $v(i,j)$ for all $i \in r_i$ and $j \in r_j$ using the kd-tree bounding boxes of r_i and r_j. If the bounds on $v(i,j)$ are tight enough to permit an approximation satisfying Lemma 2, then we can trigger Can-approximate for these contributions while staying consistent with the global error bound; otherwise we must invoke the recursive comparison of the children of r_i and r_j.

We now introduce the specific approximator $\widehat{S}_{ir}$ that we will use in our algorithm. Given a node pair r_i and r_j whose contribution is to be approximated, let $v_{ij}^{\min}$ and $v_{ij}^{\max}$ be bounds on $v(i,j)$ (see Eq. (11)) given that $i \in r_i$ and $j \in r_j$. Specifically, using distance bounds derived from the kd-tree bounding boxes, we have

$$v_{ij}^{\min} = K_{h_1}(\mathrm{d}y_{ij}^{\max})K_{h_2}(\mathrm{d}\mathbf{x}_{ij}^{\max}) \tag{15}$$

$$v_{ij}^{\max} = K_{h_1}(\mathrm{d}y_{ij}^{\min})K_{h_2}(\mathrm{d}\mathbf{x}_{ij}^{\min}). \tag{16}$$

Define $\hat{v}_{ij} = \frac{v_{ij}^{\max} + v_{ij}^{\min}}{2}$ to be the midpoint estimator for $v(i,j)$ over r_i and r_j. Let n_{r_j} be the number of points in r_j. We estimate S_{ir_j} as follows:

$$\widehat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}. \tag{17}$$

This estimator corresponds to the Approximate function from Algorithm 1. We now give an intermediate lemma bounding the error of this approximation.

Lemma 3. *Let* $\widehat{S}_{ir_j}$, $v_{ij}^{\max}$, *and* $v_{ij}^{\max}$ *be defined as in Eqs. (15)–(17). The absolute error* $\Delta S_{ir_j} = |S_{ir_j} - \widehat{S}_{ir_j}|$ *satisfies* $\Delta S_{ir_j} \leq (n_{r_j} - 1)\frac{v_{ij}^{\max} - v_{ij}^{\min}}{2} + v_{ij}^{\max}$.

Proof. First note that setting $\hat{v}_{ij}$ to the midpoint of $v_{ij}^{\max}$ and $v_{ij}^{\min}$ means that no term $v(i,j)$ in S_{ir_j} can differ from $\hat{v}_{ij}$ by more than $\frac{v_{ij}^{\max} - v_{ij}^{\min}}{2}$. With $\widehat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$, we have

$$\begin{aligned}\Delta S_{ir_j} &= \left| (n_{r_j} - 1)\hat{v}_{ij} - \sum_{\substack{j\in r_j \\ j\neq i}} v(i,j) \right| \\ &\leq \max\left\{ (n_r - 1)\frac{v_r^{\max} - v_r^{\min}}{2} + v_r^{\max},\ (n_r - 1)\frac{v_r^{\max} - v_r^{\min}}{2} \right\}.\end{aligned}$$

The first term in the max handles the case where $i \notin r_j$, in which case $\widehat{S}_{ir_j}$ contains a $\hat{v}_{ij}$ for all but one of the $v(i,j)$ terms in S_{ir_j}. We can therefore consider each matched term to be estimated with error no greater than $\frac{v_{ij}^{\max} - v_{ij}^{\min}}{2}$, and the unmatched term to be estimated as 0 with error no greater than $v_r^{\max}$. The second term in the max handles $i \in r$, in which case all terms in S_{ir_j} are matched by terms in $\widehat{S}_{ir_j}$. The first term in the max is at least as large as the second, and the lemma follows. □

Lastly, we need to specify how to combine the answers from recursive subdivisions into a unified global approximation (i.e., the $\oplus$ operator from Algorithm 1). Here we diverge somewhat from the typical dual-tree form in that we will keep our approximations $\widehat{A}_i$ separate until the end, where the merge will happen by simply summing the $\log \widehat{A}_i$ terms. Nonetheless, we use the dual-tree form to share work across the $\widehat{A}_i$ in such a way as to guarantee the global error bound $\Delta L \leq \epsilon$. The resulting procedure DETLL is listed in Algorithm 2, and we now prove its correctness.

Algorithm 2 Deterministic dual-tree approximation of the cross-validated log-likelihood objective. Comments are denoted by //.

DETLL
Input: dataset D of points $(\mathbf{x}_i, y_i)$; bandwidths h_1 and h_2; error tolerance ϵ
Output: $\widehat{L}$ s.t. $\Delta L \leq \epsilon$

1. Let $\widehat{A} = (0, 0, \ldots, 0)$ contain initial estimates for all $\widehat{A}_i$
2. Let $\text{root}_i = \text{root}_j = \text{CONSTRUCTKDROOT}(D)$
3. DETDUALTREEAPPROX($\widehat{A}$, root_i, root_j, ϵ)
4. Let $\widehat{L} = -\log(n-1)$
5. **for** $i = 1$ to n
   (a) $\widehat{L} = \widehat{L} + \frac{1}{n} \log \widehat{A}_i$
6. **return** $\widehat{L}$

DETDUALTREEAPPROX
Input: list $\widehat{A}$ of partial $\widehat{A}_i$ values; nodes r_i and r_j; error tolerance ϵ
Output: estimated contributions to $\widehat{A}_i$ from $\forall i \in r_i,\ \forall j \in r_j$ (added to entries of $\widehat{A}$)

1. **if** $\frac{(n_{r_j}+1)v_{ij}^{\max}}{(n_{r_j}-1)v_{ij}^{\min}} \leq 2e^{\epsilon} - 1$
   (a) $\widehat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$
   (b) **for** $i \in r_i$: $\widehat{A}_i = \widehat{A}_i + \widehat{S}_{ir_j}$ // approximation of S_{ir_j}
2. **else if** NOTSPLITTABLE(r_i) and NOTSPLITTABLE(r_j)
   (a) **for** $i \in r_i$: $\widehat{A}_i = \widehat{A}_i + S_{ir_j}$ // exact evaluation of S_{ir_j}
3. **else**
   (a) **for** $(p, q) \in \{\{r_i.lc, r_i.rc\} \times \{r_j.lc, r_j.rc\}\}$
      i. DETDUALTREEAPPROX($\widehat{A}$, p, q, ϵ) // all pairs of child nodes

Theorem 1. *Given a dataset D, the bandwidths h_1 and h_2, and an error tolerance ϵ, the* DETLL *algorithm returns an approximation $\widehat{L}$ of the cross-validated log-likelihood L with respect to h_1 and h_2 over D such that the absolute error ΔL satisfies $\Delta L \leq \epsilon$.*

Proof. DETLL performs exact evaluation of the contributions to each A_i term in all cases except those where $\frac{(n_{r_j}+1)v_{ij}^{\max}}{(n_{r_j}-1)v_{ij}^{\min}} \leq 2e^{\epsilon} - 1$, in which case the contributions are approximated as $\widehat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$. The key to the proof is therefore to show that such approximation leads to $\Delta L \leq \epsilon$.

By Lemma 2, if all approximations $\widehat{S}_{ir_j}$ satisfy $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq e^{\epsilon} - 1$, then $\Delta L \leq \epsilon$. Since $S_{ir_j} \geq (n_{r_j} - 1)v_{ij}^{\min}$, we have $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq \frac{\Delta S_{ir_j}}{(n_{r_j}-1)v_{ij}^{\min}}$, so we can enforce the condition of Lemma 2 by ensuring that $\frac{\Delta S_{ir_j}}{(n_{r_j}-1)v_{ij}^{\min}} \leq e^{\epsilon} - 1$. By Lemma 3, approximating with $\hat{S}_{ir_j} = (n_{r_j} - 1)\hat{v}_{ij}$ gives $\Delta S_{ir_j} \leq (n_{r_j} - 1)\frac{v_{ij}^{\max} - v_{ij}^{\min}}{2} + v_{ij}^{\max}$. Substituting this bound and rearranging shows that the Lemma 2 condition is implied by enforcing $\frac{(n_{r_j}+1)v_{ij}^{\max}}{(n_{r_j}-1)v_{ij}^{\min}} \leq 2e^{\epsilon} - 1$. This exactly matches the way and conditions in which DETLL performs approximation. Thus, the approximation scheme of DETLL satisfies the conditions of Lemma 2, and the algorithm therefore guarantees $\Delta L \leq \epsilon$. □

3.4. Monte Carlo approximation

While the deterministic pruning rule of DETLL guarantees the desired error bound, in practice it operates with a high degree of overconservatism due to the looseness of the deterministic bounds used in its derivation. In particular, some of the overconservatism stems from the fact that $v_{ij}^{\min}$ and $v_{ij}^{\max}$ are extreme values based on the corners of kd-tree bounding boxes. These bounds may be very far from the majority of the $v(i,j)$ terms being summed. In order to reduce this overconservatism we therefore introduce a new Monte Carlo scheme that uses sampled $v(i,j)$ values to get tighter, probabilistic estimates of the bounds within which they lie. This scheme allows us to focus on the *typical* rather than the *extreme* case; it is also more robust to the presence of outliers. As a result, we end up with tighter bounds, higher prunes, and therefore larger speedups; however, this increase in speed comes at the cost of losing the formal, deterministic error guarantees of DETLL. Nevertheless, our experiments demonstrate that overall accuracy remains within tolerance, which indicates that the overconservatism of the rest of the error bounding framework compensates for the added error from the Monte Carlo estimates.

To derive the Monte Carlo scheme we go back to Lemma 2, which guarantees the $\Delta L \leq \epsilon$ error bound provided that $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq e^{\epsilon} - 1$. Rather than estimate S_{ir_j} and $\frac{\Delta S_{ir_j}}{S_{ir_j}}$ in terms of $v_{ij}^{\min}$ and $v_{ij}^{\max}$, we instead take a Monte Carlo sample from which S_{ir_j} and $\frac{\Delta S_{ir_j}}{S_{ir_j}}$ can be estimated. The Monte Carlo procedure comprises steps 1–4 of Algorithm 3, and it works as follows. For a given pair of nodes r_i and r_j whose contribution S_{ir_j} we wish to estimate, we sample m pairs $(i \in r_i, j \in r_j)$ of data indices, discarding any for which $i = j$. We then compute μ_v as the mean of $v(i,j)$ over the samples, f_{skip} as the fraction of samples discarded, and σ_μ, a standard error for μ_v, from B bootstrap resamplings of our sampled $v(i,j)$. We then construct a normal confidence interval for μ_v of width $2z$ standard errors. We estimate $\widehat{S}_{ir_j} = (1 - f_{\text{skip}}) n_{r_j} \mu_v$, which implies that if our confidence interval is correct, the required bound $\frac{\Delta S_{ir_j}}{S_{ir_j}} \leq e^{\epsilon} - 1$ from Lemma 2 is satisfied if $z\sigma_\mu / \mu_v \leq e^{\epsilon} - 1$. This last condition therefore becomes our Monte Carlo pruning rule.

Algorithm 3 Monte Carlo dual-tree approximation of the cross-validated log-likelihood objective. Comments are denoted by //.

MCLL
Input: dataset D of points $(\mathbf{x}_i, y_i)$; bandwidths h_1 and h_2; error tolerance ϵ; sample size m; number of sample replications B; number of std. errors z
Output: $\widehat{L}$, targeting approximation error $\Delta L \leq \epsilon$
$\Rightarrow$ identical to DETLL (Algorithm 2), with the exception of line 3:

3. MCDUALTREEAPPROX($\widehat{A}$, root$_i$, root$_j$, ϵ, m, B, z)

MCDUALTREEAPPROX
Input: list $\widehat{A}$ of partial $\widehat{A}_i$ values; nodes r_i and r_j; error tolerance ϵ; sample size m; number of sample replications B; number of std. errors z
Output: estimated contributions to $\widehat{A}_i$ from $\forall i \in r_i,\ \forall j \in r_j$ (added to entries of $\widehat{A}$)

1. Sample m pairs (i,j) uniformly from r_i and r_j, rejecting if $i = j$
2. $\mu_v =$ mean of $v(i,j)$ over sample, f_{skip} = fraction of pairs rejected due to $i = j$
3. σ_μ = standard error of μ_v from B bootstrap resamplings
4. **if** $z\sigma_\mu / \mu_v \leq e^{\epsilon} - 1$
   (a) $\widehat{S}_{ir_j} = (1 - f_{\text{skip}}) n_{r_j} \mu_v$
   (b) **for** $i \in r_i$: $\widehat{A}_i = \widehat{A}_i + \widehat{S}_{ir_j}$ // approximation of S_{ir_j}
5. **else if** NOTSPLITTABLE(r_i) and NOTSPLITTABLE(r_j)
   (a) **for** $i \in r_i$: $\widehat{A}_i = \widehat{A}_i + S_{ir_j}$ // exact evaluation of S_{ir_j}
6. **else**
   (a) **for** $(p,q) \in \left(\{r_i.lc, r_i.rc\} \times \{r_j.lc, r_j.rc\}\right)$
      i. MCDUALTREEAPPROX($\widehat{A}$, p, q, ϵ) // all pairs of child nodes

The full Monte Carlo dual-tree procedure, MCLL, is specified in Algorithm 3. Note that the sample size m, number of bootstrap resamplings B, and confidence interval half-width z are parameters of the algorithm. We have found the algorithm performance to not be highly sensitive to these values, and for all of our experiments we used $m = 25$, $B = 10$, and $z = 1.5$ as a good compromise between accuracy and speed. While one might be concerned that this number of samples is small relative to large dataset sizes, it is important to note that the number of data points in each node decreases exponentially as we move down the tree. The Monte Carlo estimates at higher nodes need only be good enough to indicate that the variance is large, leading to further node splits such that the final estimates (whose accuracy controls the final algorithm accuracy) are made in lower-level nodes. These final nodes are not only smaller, but also more tightly clustered, making good

small-sample estimation possible. One could also envision a scheme for sizing the samples adaptively; this may be a fruitful avenue for future work.

3.5. Optimization

Due to the noisiness of L over the bandwidth space, its gradient is of little utility. Thus, we adopt a similar scheme to that of Gray and Moore (2003b): evaluate the likelihood at all combinations of $h_1 \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$ and $h_2 \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$. Given standardized data, this allows us to cover all orders of magnitude within a range that, in practice, is more than wide enough to cover the optimal bandwidth pair. Evaluating over this grid allows us to gauge the composite performance of our likelihood approximations over the full range of bandwidths, from near-optimal to far-from-optimal. For applications, one might wish to evaluate on a grid with somewhat finer resolution, e.g., at intervals $\{0.25, 0.5, 0.75, 1\}$ within each order of magnitude.

4. Experiments

We present results from two sets of experiments. The first set is designed to test the efficiency and accuracy of our likelihood approximation algorithms across a broad array of datasets. In order to gauge the exact speedup and error performance of our methods, we restrict ourselves to medium-sized datasets (17,000–60,000 points) so that the baseline exact evaluations can be done in reasonable time. In the second set of experiments, we test the scalability of our Monte Carlo procedure by applying it to datasets on the order of hundreds of thousands to millions of points. Datasets of this size represent the largest reported applications of KCDE to date, by several orders of magnitude.

We note several details about the experimental methodology. First, while our method works with any kernel function, we used the Epanechnikov kernel in all cases (see Silverman, 1986). Second, all data were standardized (mean subtracted, divided by standard deviation) on a per-dimension basis. This allows us to search the same bandwidth range for all datasets. While not the same as individually optimizing per-dimension bandwidths, standardization means that the bandwidth h_2 effectively provides a scaled bandwidth $h_2\sigma_d$ for each dimension d of $\mathbf{x}$.

The nature of the cross-validated likelihood score L over a kernel with finite support (such as the Epanechnikov) is such that it can be pushed in value to $-\infty$. This happens in cases where one or more points are far enough from the others that they receive no leave-one-out probability density. The MCLL approximation occasionally returns a finite value for bandwidth pairs whose true score is $-\infty$. This is not problematic in practice, and, in fact, one could argue that this is a "feature" that makes MCLL more robust than exact cross-validated likelihood. In order not to skew the picture of accuracy, however, such instances were left out of the error averages. Both speedup and error measurements for the Monte Carlo approximation are averaged first across five invocations per evaluated bandwidth pair, then across all bandwidth pair averages. For deterministic approximation, we average only across all bandwidth pairs.

In order to compensate for the overconservatism of our approximation framework and to put the two approximation methods on equal grounds in terms of error performance, the error tolerances for the deterministic and Monte Carlo approximations were set to $\epsilon = 700$ and $\epsilon = 1$, respectively. These were chosen to consistently give actual errors of no more than 0.1. The additional parameter settings for the Monte Carlo algorithm were: $m = 25$ (sample size); $B = 10$ (resampling replications); $z = 1.5$ (number of standard errors for confidence interval).

All experiments were run sequentially on a two-processor Intel Xeon 3.8 GHz hyperthreading machine with 8 GB memory.

4.1. Error control and speedup on medium-scale datasets

In the first set of experiments, we ran the full optimization procedure described in Section 3.5 on seven medium-sized datasets ranging from 17,605 to 61,634 points in 3–16 dimensions. For each bandwidth pair evaluated during the optimization, we recorded the evaluation time and score for (1) exact evaluation, (2) deterministic dual-tree approximation (Algorithm 2), and (3) Monte Carlo dual-tree approximation (Algorithm 3). For both deterministic and Monte Carlo approximation, we compare the approximate likelihood scores to the exact scores and report the average error across all evaluated bandwidths. We also report the speedup in terms of total evaluation time over all bandwidths, with $\text{speedup}_{\text{approx}} = \text{time}_{\text{exact}}/\text{time}_{\text{approx}}$. Tables 1 and 2 give the numerical results (datasets names appear in the first column).

Several patterns are apparent in these results. First, both the deterministic and Monte Carlo dual-tree methods give speedup over the exact method while keeping error small. Speedups for the deterministic algorithm, however, are less than 10 in all cases, while the addition of Monte Carlo yields speedups from the mid-thousands to just over ten thousand. Adding Monte Carlo to the dual-tree methodology is therefore seen to give a boost of 2–3 orders of magnitude over the deterministic approach. Again, the reason for this is that Monte Carlo bounds are much tighter than the worst-case deterministic bounds, leading to higher prunes and far less overall work being performed.

The cost of increasing speedup via Monte Carlo is the loss of the formal error guarantee provided by the deterministic approximation. As we can see, however, error is still well controlled by the Monte Carlo algorithm, so this is a small sacrifice. Average absolute errors are below 0.1 in all cases, which generally corresponds to a relative error of less than 5%.

Table 1
Results for deterministic dual-tree likelihood approximation over a range of medium-sized datasets.

Dataset	n	d	Exact time (min)	Det. time (min)	Speedup	Error
sp500	17,605	6	296.2	30.9	9.6	0.03
census	20,640	9	203.9	136.4	1.5	0.02
spectral	46,420	5	818.8	374.8	2.2	0.02
sdss50k	50,000	3	621.5	110.0	5.7	0.04
galaxy50k	50,000	3	424.7	99.2	4.3	0.02
quasar50k	50,000	4	1337.2	384.7	3.5	0.009
corel	61,634	16	2533.0	1380.6	1.8	0.032

Table 2
Results for Monte Carlo dual-tree likelihood approximation over a range of medium-sized datasets.

Dataset	n	d	Exact time (min)	MC time (s)	Speedup	Error
sp500	17,605	6	296.2	2.2	8,265	0.05
census	20,640	9	203.9	2.6	4,745	0.01
spectral	46,420	5	818.8	5.7	8,606	0.01
sdss50k	50,000	3	621.5	6.5	5,746	0.05
galaxy50k	50,000	3	424.7	4.1	6,165	0.05
quasar50k	50,000	4	1337.2	4.9	16,233	0.0006
corel	61,634	16	2533.0	13.1	11,598	0.009

Table 3
Results for Monte Carlo dual-tree likelihood approximation over a set of large-scale datasets.

Dataset	n	d	Exact time	MC time	Speedup
sdss	389,353	3	26.4 days	33.7 s	67,611
galaxy1M	1000,000	3	197.4 days	1.7 min	169,572
quasar1M	1000,000	4	1.2 years	2.1 min	286,564

Another important pattern is the behavior of speedup as datasets increase in size. In the Monte Carlo case, speedup appears to trend upward with dataset size, and this is further validated by the larger-scale datasets addressed in the next section. In the deterministic case, the pattern is less clear, but may show a slight increasing trend. This is another strong reason for preferring the Monte Carlo algorithm in the large-data case, as its speedup level is more likely to be able to keep up with the $O(n^2)$ growth in computational cost of the exact algorithm.

All together, these medium-scale results indicate that both the deterministic and Monte Carlo dual-tree algorithms provide high-quality approximations, but the Monte Carlo approach is much faster. We therefore use the Monte Carlo algorithm exclusively as we examine the scalability of our likelihood approximation scheme.

4.2. Scalability on large datasets

We now address the large-data regime for which fast approximation is most important. In particular, we give results for our three largest datasets, each of which contains on the order of 10^5–10^6 points. Because the exact likelihood is too expensive to compute on datasets of this size, we instead extrapolated exact runtimes by fitting a quadratic curve to the runtimes on smaller subsamples of sizes $\{10, 25, 50, 75\} \times 10^4$. We then approximated likelihoods on the full-sized datasets using Monte Carlo dual-tree approximation. Table 3 summarizes the runtime performance.

We see in these results a continuation of the pattern of substantially increasing speedup as dataset size increases. Speedups range from around 67,000 to over 286,000. In absolute terms, the runtime reductions are quite significant, e.g., from 1.2 years to 2.1 min in the largest case. Relative to the medium-sized datasets, the Monte Carlo runtimes went from several seconds to several minutes, while the exact runtimes went from hundreds of minutes to tens and hundreds of days. This demonstrates excellent scalability. Our application of KCDE to datasets of size 10^6 is larger than any previously published application of KCDE by three orders of magnitude (Holmes et al., 2007). Combined with the good error control demonstrated in the previous section, these results would seem to make MCLL the algorithm of choice for KCDE likelihood optimization on large datasets.

5. Conclusion

We have presented a fast, scalable, approximate method for cross-validated, data-driven bandwidth selection in kernel conditional density estimation. This method injects Monte Carlo sampling into the dual-tree framework that has previously been used to produce state-of-the-art scalable bandwidth selection for other kernel estimators. Our Monte Carlo method is powerful because it enables KCDE to be used for datasets and applications that previously would have been computationally impractical. It optimizes a cross-validated likelihood objective that presents advantages over the squared-error losses used

in prior work. Because KCDE is such a fundamental and versatile method, having a fast, scalable version opens the door to being able to ask new questions, produce new forms of visualization and analysis, develop new methods that would previously have been too costly, and so on.

Experimentally, we have shown that our Monte Carlo dual-tree method controls error, produces speedups on the order of 10^3–10^5, and scales to datasets on the order of at least one million points. Absolute computation time reductions were as large as from 1.2 years to 2.1 min. These are very significant speedups, and represent an advance of at least several orders of magnitude in both speed and addressable dataset size over the prior state of the art.

The major drawback to this Monte Carlo dual-tree algorithm is the loss of formal error guarantees relative to the deterministic dual-tree method. Current and future work is focused on addressing this shortcoming by moving from the "Monte Carlo in trees" approach used here to a "trees in Monte Carlo" approach that puts the approximation on a fully Monte Carlo footing while still leveraging the power of tree-based spatial partitioning. The hope is that this will yield formal relative error control while also broadening the scope of applicability beyond likelihood-based KCDE to include fast Monte Carlo learning procedures for other kernel estimators, such as kernel density estimation and kernel regression, and other objective functions, such as MSE and MISE.

References

Bashtannyk, D.M., Hyndman, R.J., 2001. Bandwidth selection for kernel conditional density estimation. Computational Statistics & Data Analysis 36 (3), 279–298.

Boyer, G.F., Riegel, R.N., Gray, A.G., 2007. A parallel *N*-body data mining framework, in: NIPS Workshop on Efficient Machine Learning.

de Freitas, N., Wang, Y., Mahdaviani, M., Lang, D., 2005. Fast Krylov methods for *N*-body learning. Advances in Neural Information Processing Systems (NIPS) 18.

Fan, J., Yao, Q., Tong, H., 1996. Estimation of conditional densities and sensitivity measures in nonlinear dynamical systems. Biometrika 83 (1), 189–206.

Fan, J., Yim, T.H., 2004. A crossvalidation method for estimating conditional densities. Biometrika 91 (4), 819–834.

Gooijer, J.G.D., Zerom, D., 2003. On conditional density estimation. Statistica Neerlandica 57 (2), 159–176.

Gray, A.G., 2004. Fast kernel matrix–vector multiplication with application to Gaussian process regression. Tech. rep., Carnegie Mellon Computer Science Department.

Gray, A.G., Moore, A.W., 2000. *N*-body problems in statistical learning. Advances in Neural Information Processing Systems (NIPS) 13.

Gray, A.G., Moore, A.W., 2003a. Nonparametric density estimation: Toward Computational Tractability, in: SIAM International Conference on Data Mining, SDM.

Gray, A.G., Moore, A.W., 2003b. Rapid evaluation of multiple density models, in: Workshop on Artificial Intelligence and Statistics, AISTATS.

Gray, A.G., Moore, A.W., 2003c. Very fast multivariate kernel density estimation via computational geometry, in: Proceedings of the ASA Joint Statistical Meeting.

Gray, A.G., Moore, A.W., 2004. Fast computation of the pair correlation and *n*-point correlation functions, in: Conference on Computational Physics.

Gray, A.G., Moore, A.W., Nichol, R.C., Connolly, A.J., Genovese, C., Wasserman, L., 2004. Multi-tree methods for statistics on very large datasets in astronomy, in: Ochsenbein, F., Allen, M.G., Egret, D., Astronomical Data Analysis Software and Systems, ADASS, XIII.

Gray, A., Riegel, R., 2006. Large-scale kernel discriminant analysis with application to quasar discovery, in: Proceedings in Computational Statistics, CompStat.

Hansen, B.E., 2004. Nonparametric conditional density estimation, unpublished manuscript. URL: http://www.economics.ucr.edu/seminars/fall04/Bruce Hansen.pdf.

Holmes, M.P., 2009. Multi-tree Monte Carlo methods for fast, Scalable Machine Learning. Ph.D. Thesis, Georgia Institute of Technology.

Holmes, M.P., Gray, A.G., Isbell, Jr. C.L., 2007. Fast nonparametric conditional density estimation, in: Uncertainty in Artificial Intelligence, UAI.

Hyndman, R.J., Bashtannyk, D.M., Grunwald, G.K., 1996. Estimating and visualizing conditional densities. Journal of Computation and Graphical Statistics 5 (4), 315–336.

Klaas, M., Briers, M., de Freitas, N., Doucet, A., 2006. Fast particle smoothing: If I had a million particles, in: International Conference on Machine Learning, ICML.

Klaas, M., de Freitas, N., Doucet, A., 2005a. Toward practical N^2 Monte Carlo: The marginal particle filter. in: Uncertainty in Artificial Intelligence, UAI.

Klaas, M., Lang, D., de Freitas, N., 2005b. Fast maximum a posteriori inference in Monte Carlo state spaces, in: International Conference on Artificial Intelligence and Statistics, AISTATS.

March, W., Gray, A., 2007. Large-scale Euclidean MST and hierarchical clustering, in: NIPS Workshop on Efficient Machine Learning.

Moore, A., Connolly, A., Genovese, C., Gray, A., Grone, L., Kanidoris, II N., Nichol, R., Schneider, J., Szalay, A., Szapudi, I., Wasserman, L., 2000. Fast algorithms and efficient statistics: *N*-point correlation functions. in: Mining the Sky: Proceedings of the MPA/ESO/MPE Astrophysics Symposium.

Riegel, R., Gray, A., Richards, G., 2008. Massive-scale kernel discriminant analysis: Mining for quasars, in: SIAM International Conference on Data Mining.

Rosenblatt, M., 1969. Conditional probability density and regression estimators. In: Krishnaiah, P.R. (Ed.), Multivariate Analysis II. Academic Press, New York, pp. 25–31.

Shen, Y., Ng, A.Y., Seeger, M., 2006. Neural Information Processing Systems (NIPS).

Silverman, B.W., 1986. Density Estimation for Statistics and Data Analysis. In: Monographs on Statistics and Probability, vol. 26. Chapman & Hall.

Stender, N., 2006. Essays on marketing engineering. Ph.D. Thesis, University of Aarhaus, Denmark.