Rapid Evaluation of Multiple Density Models

Alexander G. Gray
Department of Computer Science
Carnegie Mellon University
agray@cs.cmu.edu

Andrew W. Moore
Robotics Inst. and Dept. Comp. Sci.
Carnegie Mellon University
awm@cs.cmu.edu

Abstract

When highly-accurate and/or assumption-free density estimation is needed, nonparametric methods are often called upon - most notably the popular kernel density estimation (KDE) method. However, the practitioner is instantly faced with the formidable computational cost of KDE for appreciable dataset sizes, which becomes even more prohibitive when many models with different kernel scales (bandwidths) must be evaluated – this is necessary for finding the optimal model, among other reasons. In previous work we presented an algorithm for fast KDE which addresses large dataset sizes and large dimensionalities, but assumes only a single bandwidth. In this paper we present a generalization of that algorithm allowing multiple models with different bandwidths to be computed simultaneously, in substantially less time than either running the single-bandwidth algorithm for each model independently, or running the standard exhaustive method. We show examples of computing the likelihood curve for 100,000 data and 100 models ranging across 3 orders of magnitude in scale, in minutes or seconds.

1 KERNEL DENSITY ESTIMATION

Density estimation. In situations where the density of a dataset itself (rather than other inferences) is of importance, such as in exploratory scientific data analysis, nonparametric methods are used because they make minimal or no assumptions about the distribution of the data, while achieving high accuracy – making serious density estimation almost synonymous with nonparametric methods [14, 12]. For example, *kernel density estimation* (KDE), the most widely used and studied nonparametric density estimation method and thus our focus here, can be shown to converge to the true underlying density with probability 1 as more data are observed, with no distribution assumptions at all, requiring only mild conditions on the kernel function and scale [2]. Highly-accurate density estimation is also of use as a core engine in probabilistic learning tasks, from classification to regression to clustering (though arguably for some problems density estimation itself may be skipped and the problem better solved directly, *e.g.* in discriminative classification [16]). Unfortunately, the inherent flexibility which yields these benefits comes at a very high computational cost, which is our primary focus in this paper.

Kernel density estimation. The task is to estimate the density $\hat{p}(\underline{x}_q)$ for each point $\underline{x}_q$ in a query (test) dataset $\underline{X}_\mathcal{Q}$ (having size $N_\mathcal{Q}$), from which we can also compute the overall log-likelihood of the dataset $\widehat{L}_\mathcal{Q} = \sum_{q=1}^{N_\mathcal{Q}} \log \hat{p}(\underline{x}_q)$. The 'model' is the training dataset $\underline{X}_\mathcal{T}$ (having size $N_\mathcal{T}$) itself, in addition to a local kernel function $K(\cdot)$ centered upon each training datum, and its scale parameter h (the 'bandwidth'). The density estimate at the q^{th} test point $\underline{x}_q$ is

$$\hat{p}(\underline{x}_q) = \frac{1}{N_\mathcal{T}} \sum_{t=1}^{N_\mathcal{T}} \frac{1}{V_{Dh}} K\left(\frac{\|\underline{x}_q - \underline{x}_t\|}{h}\right) \tag{1}$$

where D is the dimensionality of the data and V_{Dh} is the volume encompassed by $K(\cdot)$. For good estimates, the exact form of $K(\cdot)$[1] turns out to be relatively unimportant, while the correct choice of h is critical [14, 12].

[1]Common choices are the spherical, Gaussian, or Epanechnikov kernels; our method works efficiently with any such standard kernel. The spherical kernel ($K(\underline{x}_q, \underline{x}_t) = 1$ if $\|\underline{x}_q - \underline{x}_t\| < h$, otherwise 0) is simplest. The Epanechnikov kernel ($K(\underline{x}_q, \underline{x}_t) = \frac{D+2}{2V'_{Dh}}(1 - \|\underline{x}_q - \underline{x}_t\|^2)$ if $\|\underline{x}_q - \underline{x}_t\| < h$, otherwise 0, where V'_{Dh} is the volume of the sphere in D dimensions) has the property of asymptotically-optimal estimation efficiency among all possible kernels [3]. The latter two are examples of kernels with compact, or bounded extent, a property we can exploit computationally.

Computational Cost. The first difficulty is that evaluating a density naively (by summing over each training point for each of the query points) is $O(N_{\mathcal{Q}} N_{\mathcal{T}})$ (or simply $O(N^2)$; for most of this paper it will turn out that $\underline{X}_{\mathcal{Q}} = \underline{X}_{\mathcal{T}}$, so for notational convenience we may also use $N = N_{\mathcal{Q}} = N_{\mathcal{T}}$). While univariate methods [13, 4] have been proposed to addressed this, [7] showed for the first time an algorithm yielding orders of magnitude in speedup over the default exhaustive method in the general multivariate case. High-accuracy KDE was demonstrated across dataset sizes ranging up to 3 million and dimensionalities up to 784, but the method was designed to compute a density for only one bandwidth. Consequently, computing B different densities with different bandwidths requires B independent computations. When B is large, say 100 or 1000, the total cost may overwhelm the efficiency gains of the individual computations. This is the issue we are addressing in this paper.

Overview. In the next section we'll review the reasons multiple density models need to be computed. After briefly reviewing the single-bandwith algorithm of [7], we'll present the multi-bandwidth generalization of the algorithm. Finally we will present the results of a performance study of the multi-bandwidth method.

2 THE PRACTICAL NEED TO COMPUTE MULTIPLE MODELS

Several common tasks demand the computation of models for the same data but B different bandwidths.

Scoring models for selection. As mentioned, the central issue of estimating a density optimally with KDE is selecting the optimal bandwidth h^*. Across statistical learning, *model selection* in current practice often amounts to evaluating a set of learned models (representing a finite set of parameter settings chosen from the set of all possible parameters) under a score function and a dataset (where the score may correspond to, for example, a Bayesian posterior, structural risk minimization, maximum entropy, maximum likelihood, least-squares, and so on).

Scoring models for combination. An alternative to strict model selection is *model combination*, in which the estimates of multiple learned models (again corresponding to some finite set of chosen parameter settings) are combined to form a final estimate, weighted by their score. Examples include Bayesian model combination and stacking. This methodology has been the focus of considerable attention in the learning literature in recent years, mainly for the task of classification – however, the same principle applies to density estimation, as noted by [15].

Cross-validation scoring for KDE. The two most widely-used scoring methods for bandwidth selection in KDE are derived from different motivations but both end up being a form of leave-one-out cross-validation. *Likelihood cross-validation* [8] is derived by minimization of the Kullback-Liebler information $\int p(\underline{x}) \log \frac{p(\underline{x})}{\hat{p}(\underline{x})} d\underline{x}$, yielding the score

$$CV(h) = \frac{1}{N_T} \sum_{t}^{N_T} \log \hat{p}_{-t}(\underline{x}_t) \tag{2}$$

where the $-t$ subscript denotes an estimate using all N_T points except the t^{th}. *Least-squares cross-validation* [11] minimizes an integrated squared error criterion $\int \hat{p}^2 - 2 \int \hat{p}p$, yielding the score

$$CV(h) = \frac{1}{hN_T(N_T - 1)} \sum_{t}^{N_T} \hat{p}_{-t}(\underline{x}_t) \tag{3}$$

where the density estimate uses the derived kernel $K^*(\cdot) = K(\cdot) * K(\cdot) - 2K(\cdot)$. Our algorithm is designed to allow both procedures. The computational implementation of either scoring procedure estimates the density for each of the N points of the dataset using the other $N-1$ points; this is done for each of the B bandwidths under consideration, yielding the cost $O(BN^2)$ by naive computation.

Exploratory visualization. It is often useful in exploratory data analysis to visualize the curve representing the score as a function of the bandwidth. Figure 1 shows an example of the kind of curve we would like to be able to generate quickly. Shown are the cross-validated likelihood scores for 1000 bandwidths ranging from 0.0001 to 0.1, along with the optimal bandwidth h^* (about .00774), for the astrophysics dataset described later in the empirical results.

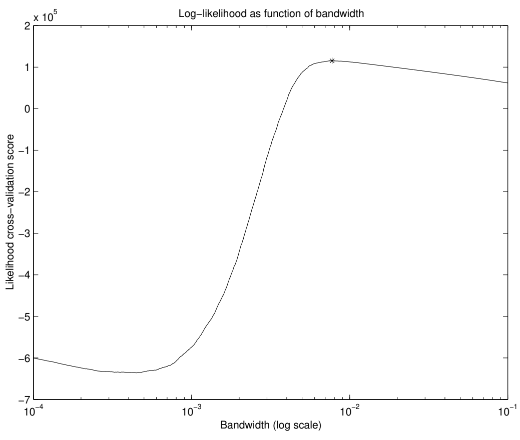

Figure 1: Example of bandwidth analysis.

3 PREVIOUS ALGORITHM: SINGLE-BANDWIDTH METHOD

Space-partitioning with ball-trees. First we partition the training set $\underline{X}_{\mathcal{T}}$ hierarchically using a space-partitioning tree. This allows us to pass a query point $\underline{x}_q$ down the tree, possibly ignoring or approximating entire chunks of the training data if they are far enough away to have provably small contribution to the density at $\underline{x}_q$, based on the stored boundaries of the sub-regions. Here we use ball-trees, which can be built efficiently and with high quality using the anchors hierarchy algorithm [9], and have been demonstrated to be effective in up to thousands of dimensions in [9]). Note that the tree need only be built on a dataset once, and does not need rebuilding for KDE evaluations using different bandwidths or for different queries. Thus its initial build cost (which is in the seconds typically) is amortized over all subsequent KDE computations.

Higher-order divide-and-conquer and dual-tree traversal. The first application of the principle of higher-order divide-and-conquer is that we will also build a space-partitioning tree for $\underline{X}_{\mathcal{Q}}$ (which is the same tree if $\underline{X}_{\mathcal{T}} = \underline{X}_{\mathcal{Q}}$). This allows us to now compare chunks of the query set with chunks of the training set, for added speed advantage. However, this necessitates generalizing the familiar tree traversal to *dual*-tree traversal. In the base case of the recursion, in which a leaf node of $\underline{X}_{\mathcal{Q}}$ is compared with a leaf node of $\underline{X}_{\mathcal{T}}$, the individual data points in the nodes Q and T are examined exhaustively. Through pruning at higher levels in the tree, we hope to minimize the number of leaf-leaf comparisons that are made. Pseudocode for the algorithm is shown [2].

Exclusion and inclusion. For each node-pair Q and T encountered during the traversal, using the boundary of the data $\underline{X}_Q$ in Q (having size N_Q) and the boundary of the data $\underline{X}_T$ in T (having size N_T), we can easily compute lower and upper bounds on the distance between any pair of query point and training point. Using this we can compute bounds on the mass contribution of $\underline{X}_T$ to the density at every query point in $\underline{X}_Q$. If the maximum density contribution of T is zero within the floating precision of the machine, it can be pruned from the search (*i.e.* we do not need to recurse on its children). Note that exclusion does not necessarily introduce any error: in the case of a finite-extent kernel (such as the spherical or optimal-efficiency Epanechnikov kernel), exclusion when the minimum distance is greater than the kernel extent preserves the result exactly. A similar property holds for inclusion with the spherical kernel. The opposite of exclusion is also possible: we can prune a node when its minimum possible mass contribution is 1 within machine-precision.

Dualtree($\mathcal{P}$,h)
`while` !empty($\mathcal{P}$),
$\{Q, T, dl, du, p\}$ = minpriority($\mathcal{P}$).
v = level(Q).
`if` $\frac{|U_v - L_v|}{|L_v|} < \epsilon$, `return`.
`if` $p < p_{\max}$,
$dl' = N_T K_h(\text{maxdist}(Q,T))$.
$du' = N_T K_h(\text{mindist}(Q,T))$.
$dl = dl'$, $du = du' - N_T$.
`if` $\frac{|du' - dl'|}{|l_Q + dl'|} < \delta$,
`foreach` $\underline{x}_q \in Q$, l_q += dl, u_q += du.
L_v -=$N_Q log(l_Q)$, U_v -= $N_Q log(u_Q)$.
l_Q += dl, u_Q += du.
L_v += $N_Q log(l_Q)$, U_v += $N_Q log(u_Q)$.
enqueue($\mathcal{P}$,$\{Q, T, dl, du$,priority$(Q,T)+p_{max}\}$).
`continue`.
`else`,
$dl = -dl$, $du = -du$.
`foreach` $\underline{x}_q \in Q$, l_q += dl, u_q += du.
L_v -=$N_Q log(l_Q)$, U_v -= $N_Q log(u_Q)$.
l_Q += dl, u_Q += du.
L_v += $N_Q log(l_Q)$, U_v += $N_Q log(u_Q)$.
`if` leaf(Q) `and` leaf(T), **Dualtree_base**(Q,T).
enqueue($\mathcal{P}$,Q.left,T.left,dl, du,priority(Q, T)).
enqueue($\mathcal{P}$,Q.left,T.right,dl, du,priority(Q, T)).
enqueue($\mathcal{P}$,Q.right,T.left,dl, du,priority(Q, T)).
enqueue($\mathcal{P}$,Q.right,T.right,dl, du,priority(Q, T)).

Dualtree_base(Q,T)
`foreach` $\underline{x}_q \in Q$,
`foreach` $\underline{x}_t \in T$,
$c = K_h(\|\underline{x}_q - \underline{x}_t\|)$, l_q += c, u_q += c.
u_q -= N_T.
L_v -= $N_Q log(l_Q)$, U_v -= $N_Q log(u_Q)$.
$l_Q = \min_{q \in Q} l_q$, $u_Q = \max_{q \in Q} u_q - N_T$.
L_v += $N_Q log(l_Q)$, U_v += $N_Q log(u_Q)$.

Dual-tree algorithm, single-bandwidth.

[2]For simplification, the algorithm shown does not explicitly add the centroid mass to the estimate, but simply tightens the lower and upper bounds; at the end of the computation, the estimate $\hat{p}(\underline{x}_q)$ is based on the midpoint between l_q and u_q. In the pseudocode `continue` has the same meaning as in C, skipping all subsequent code and sending execution directly to the next `while` loop iteration; similarly `break` escapes the inner-most enclosing loop as in C; also `a += b` means `a = a + b`. p_{max} is the maximum possible value of the priority function. If a node is a leaf, its left or right child is defined to be itself. The winnow() function removes bandwidths meeting the specified condition.

Constant-mass centroid approximation. If the percentage difference between these bounds is smaller than some predetermined small δ, we can prune the node by approximating its mass contribution by its centroid $\underline{\mu}_T$, *i.e.* we add $N_T K_h(\underline{\mu}_T)$. Exclusion and inclusion are actually both special cases of this more general pruning rule.

Guaranteed local and global bounds. We can maintain lower and upper bounds l_q and u_q on the density at each point $\underline{x}_q$, as well as global bounds L_v and U_v on the overall log-likelihood of the density estimate based on the local bounds available at level v in the tree. We begin with maximally pessimistic bounds and tighten them as we recurse and observe training points at increasingly finer granularity (we start by agnostically setting the lower bound to assume that no training points contribute any mass, and the upper bound to assume that all training points contribute maximum mass). We can stop the algorithm whenever we detect that $\frac{|U_v - L_v|}{|L_v|} < \epsilon$, corresponding to a user-set accuracy tolerance.

Priority search and reversibility. By controlling the search with a priority queue $\mathcal{P}$ (we use a Fibonacci heap), we can undo approximations made with the constant-mass pruning rule which as long as the ϵ tolerance has not been reached and there are still nodes on the queue. The simple priority function we use is based on the minimum distance between node boundaries.

4 GENERALIZED ALGORITHM: MULTI-BANDWIDTH METHOD

Higher-order divide-and-conquer and multiple bandwidths. We now generalize the algorithm to include a range of bandwidths indexed by b_{lo} and b_{hi} during consideration of each node-pair, which is recursively narrowed as search progresses toward the leaves. This corresponds to a second application of the principle of higher-order divide-and-conquer.

Vector generalizations of bounds. All the bounds, l_q,u_q,l_Q,u_Q,L_v and U_v, generalize from the scalar quantities of the single-bandwidth algorithm to vector quantities, containing bounds for each bandwidth.

Base case: sharing distance computations. In the base case, we reuse each distance computation d_{qt} for each of the $b_{hi} - b_{lo} + 1$ bandwidths that remain upon reaching the base case, just as the naive exhaustive algorithm can do if modified for the multi-bandwidth problem.

Multi-Dualtree$(\mathcal{P},\mathcal{H})$
`while` !empty$(\mathcal{P})$,
  $\{Q, T, dl, du, p, b_{lo}, b_{hi}\}$ = minpriority$(\mathcal{P})$.
  v = level(Q).
  `if` $\forall b \in [b_{lo}, b_{hi}]$, $\frac{|U_v^b - L_v^b|}{|L_v^b|} < \epsilon$, `return`.
  `else` $\{b_{lo}, b_{hi}\}$ = winnow$(b_{lo}, b_{hi}, \frac{|U_v^b - L_v^b|}{|L_v^b|} < \delta)$.
  `if` $p < p_{\max}$,
    $d_{\max}$ = maxdist(Q, T), $d_{\min}$ = mindist(Q, T).
    `foreach` $b \in [b_{lo}, b_{hi}]$,
      h = lookup$(\mathcal{H}, b)$.
      $dl' = N_T K_h(d_{\max})$, $du' = N_T K_h(d_{\min})$.
      $dl^b = dl'$, $du^b = du' - N_T$.
      `if` $\frac{|du' - dl'|}{|l_Q^b + dl'|} < \delta$,
        `foreach` $\underline{x}_q \in Q$, l_q^b += dl^b, u_q^b += du^b.
        L_v^b -= $N_Q log(l_Q^b)$, U_v^b -= $N_Q log(u_Q^b)$.
        l_Q^b += dl^b, u_Q^b += du^b.
        L_v^b += $N_Q log(l_Q^b)$, U_v^b += $N_Q log(u_Q^b)$.
    `if` $\forall b \in [b_{lo}, b_{hi}]$, $\frac{|du'_b - dl'_b|}{|l_Q^b + dl'_b|} < \delta$,
      enqueue$(\mathcal{P}, \{Q, T, dl, du, \text{priority}(Q, T) + p_{max}, b_{lo}, b_{hi}\})$.
      `continue`.
    `else` $\{b_{lo}, b_{hi}\}$ = winnow$(b_{lo}, b_{hi}, \frac{|du'_b - dl'_b|}{|l_Q^b + dl'_b|} < \delta)$.
  `else`,
    $dl = -dl$, $du = -du$.
    `foreach` $b \in [b_{lo}, b_{hi}]$,
      `foreach` $\underline{x}_q \in Q$, l_q^b += dl^b, u_q^b += du^b.
      L_v^b -= $N_Q log(l_Q^b)$, U_v^b -= $N_Q log(u_Q^b)$.
      l_Q^b += dl^b, u_Q^b += du^b.
      L_v^b += $N_Q log(l_Q^b)$, U_v^b += $N_Q log(u_Q^b)$.
  `if` leaf(Q) `and` leaf(T), **Multi-Dualtree_base**(Q,T).
  enqueue$(\mathcal{P}, \{Q.\text{left}, T.\text{left}, dl, du, \text{priority}(Q, T), b_{lo}, b_{hi}\})$.
  enqueue$(\mathcal{P}, \{Q.\text{left}, T.\text{right}, dl, du, \text{priority}(Q, T), b_{lo}, b_{hi}\})$.
  enqueue$(\mathcal{P}, \{Q.\text{right}, T.\text{left}, dl, du, \text{priority}(Q, T), b_{lo}, b_{hi}\})$.
  enqueue$(\mathcal{P}, \{Q.\text{right}, T.\text{right}, dl, du, \text{priority}(Q, T), b_{lo}, b_{hi}\})$.

Multi-Dualtree_base(Q, T, b_{lo}, b_{hi})
`foreach` $\underline{x}_q \in Q$,
  `foreach` $\underline{x}_t \in T$,
    $d_{qt} = \|\underline{x}_q - \underline{x}_t\|$.
    `foreach` $b \in [b_{lo}, b_{hi}]^{-}$,
      h = lookup$(\mathcal{H}, b)$, $c = K_h(d_{qt})$.
      `if` $c == 0.0$, `break`.
      `else` l_q^b += c, u_q^b += c.
  `foreach` $b \in [b_{lo}, b_{hi}]$, u_q^b -= N_T.
`foreach` $b \in [b_{lo}, b_{hi}]$,
  L_v^b -= $N_Q log(l_Q^b)$, U_v^b -= $N_Q log(u_Q^b)$.
  $l_Q^b = \min_{q \in Q} l_q^b$, $u_Q^b = \max_{q \in Q} u_q^b - N_T$.
  L_v^b += $N_Q log(l_Q^b)$, U_v^b += $N_Q log(u_Q^b)$.

Dual-tree algorithm, multi-bandwidth.

Exclusion and inclusion: exploitation of nesting. We compute the kernel evaluations for each bandwidth from highest to lowest, using the nesting property that if a bandwidth h is excludable, so are all bandwidths $h' < h$. In the finite-extent kernel case, we can also quickly perform an exclusion once we locate the smallest bandwidth still containing d_{qt}; this can be performed with binary search if B is very large. This also applies to inclusion in the spherical kernel case, in reverse. (This special case is not shown in the pseudocode.) Otherwise, the constant-mass approximation criterion can be tested for each bandwidth.

Recursive range-narrowing. After bounds updating and propagation for the appropriate sub-ranges, we narrow the range of bandwidths which still need to be considered and recurse. Note that with this procedure there is no loss of information nor pruning opportunity, with respect to the single-bandwidth algorithm.

5 PERFORMANCE

Empirical study. In our empirical study, we measure seconds of actual runtime of the multi-bandwidth scheme, independent single-bandwidth computations and the exhaustive method, on an Alpha-processor-based desktop workstation which is not as fast as the most recent Pentium-Pro-based workstations but has 14Gb of RAM. For brevity we report only likelihood cross-validation scoring, though performance is very similar for least-squares cross-validation scoring. The approximation parameter is set in all cases so that the maximum possible error in the overall log-likelihood was no more than 10^{-8}, *i.e.* one one-millionth of one percent away from the true value. The default kernel function used is the asymptotically-optimal Epanechnikov kernel. (Runtimes for the spherical kernel are very nearly identical, and very close to a factor of 2 greater for the infinite-extent Gaussian kernel.)

Datasets. Most experiments are on a segment of the Sloan Digital Sky Survey—a data collection of current scientific interest, and the active subject of ongoing nonparametric density estimation studies. It contains spatial coordinates in the first two dimensions - the dataset containing these attributes is called RA-Dec. The Sloan data includes an additional 20 color attributes from various instruments, which we test in a separate dataset called Colors. We also test a 5-dimensional biological screening dataset called BIO5.

Scaling with number of bandwidths, near optimum. We first examine the case in which the bandwidths fall on a scale ranging over one order of magnitude roughly centered around the optimum bandwidth h^*.

RA-Dec, N = 100K, $h \in [0.001, 0.01]$				
B	Multi Time	Indep Single Time	Naive Time	Speedup Over Naive
1	1.4	1.4	1204	889
10	7	25	3631	518
100	39	201	26859	671
1000	465	2170	374098	805

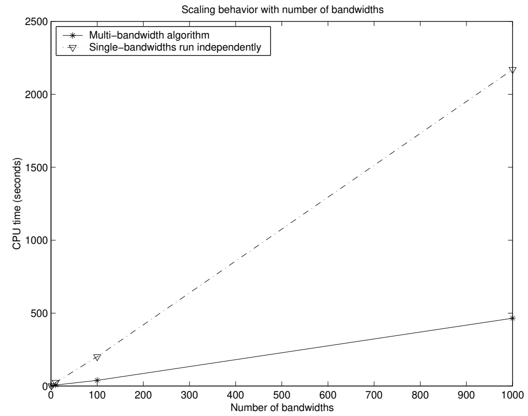

Figure 2: Simultaneous vs. separate computations.

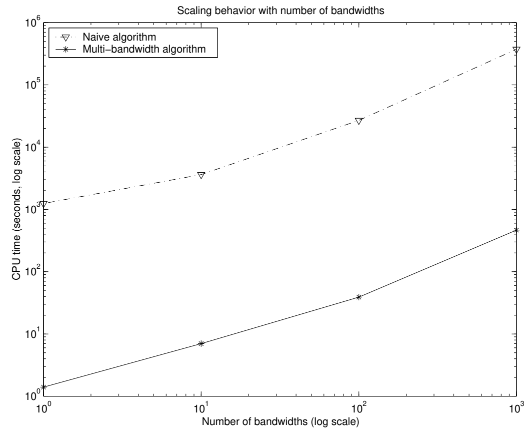

Figure 3: Comparison to naive exhaustive method.

Though the theoretical complexity of the scaling of the multi-bandwidth algorithm is $O(B)$, the log-log plot of the growth in actual CPU time shows a superlinearity. Though relatively mild in the range of B we are typically interested in (100 models is probably a reasonable number for most purposes), it is curious that the naive exhaustive method displays the same superlinearity.

This appears to be a side-effect of a limitation of the multi-bandwidth algorithm – it has a large memory footprint necessitated by the fact that it must store B entire densities (actually bounds on them), each of size $O(N)$. For the largest number of bandwidths, the large RAM of our test workstation was taxed near its limit, limiting the size of dataset that can be processed. Further, far below the point of swapping, the surprising effect of hardware cache-locality issues becomes significantly evident. Note that the naive multi-bandwidth method also shares this limitation. One advantage of independent single-bandwidth computations is that this memory consumption can be avoided if necessary.

Scaling with number of bandwidths, far from optimum. We next create difficulty for the algorithm by making it evaluate densities over a much broader range, covering 3 orders of magnitude. In this case we expect less sharing to be possible between the simultaneous computations.

RA-Dec, N = 100K, $h \in [0.0001, 0.1]$				
B	Tree Build Time	Multi Time	Naive Time	Speedup Over Naive
1	5	1.4	1204	889
10	5	110	3631	33
100	7	545	26859	49
1000	10	6240	374098	60

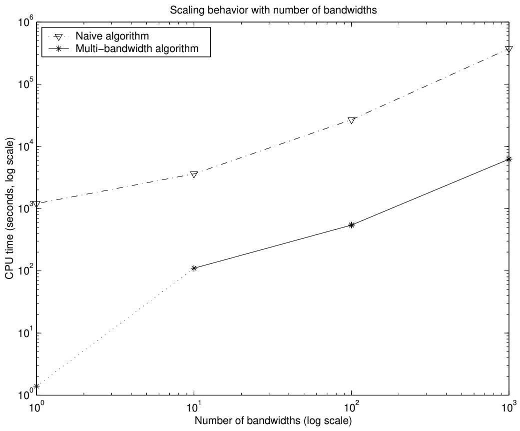

Figure 4: Performance far from optimal bandwidth.

We indeed observe a large degradation in the performance of the multi-bandwidth algorithm in this case. Over an order of magnitude of computational advantage is lost (note that the first data point in the plot of 4 is misleading).

Other datasets. To determine whether the scaling is relatively independent of the exact structure of the data, we test datasets other than the RA-Dec dataset, chosen to be significantly different kinds of measurements and in different dimensionalities.

N = 100K, near h^*				
B	RA-Dec (2-d)	BIO5 (5-d)	Colors (20-d)	Naive Time
1	1.4	14	20	1204
10	7	90	105	3631
100	39	449	413	26859
1000	465	5341	6675	374098

Indeed, the datasets appear to share a common scaling behavior. (Note that the naive time for RA-Dec only is shown; the naive time for the other datasets is nearly identical.)

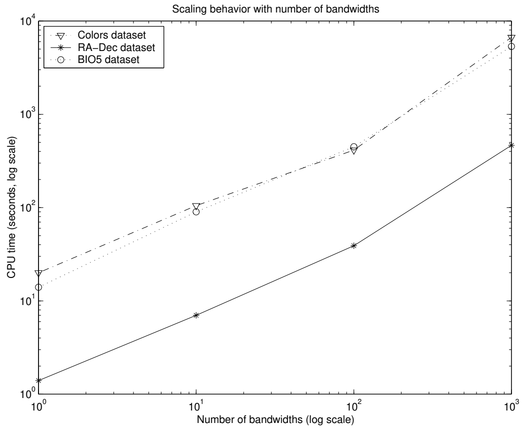

Figure 5: Scaling across datasets.

6 CONCLUSION AND FUTURE WORK

Multi-bandwidth algorithm for KDE. For the problem of multi-model multivariate kernel density estimation, we believe there exist no algorithms (for finite kernels) which are faster, nor any approximate algorithms (for all kernels) that are faster while providing hard-guarantees of high accuracy. The code is available for download at `http://www.cs.cmu.edu/~agray`.

Optimal bandwidth determination. Though the multi-bandwidth algorithm we presented can be used for finding the single optimum bandwidth for a dataset, we believe there exist opportunites to perform such a computation even more frugally. We plan to develop these ideas in future work.

Multi-recursion and generalized N-body problems. This algorithm represents a new extension along a continuing line of research in statistical algorithms [1, 10, 9, 5, 7]. The multi-bandwidth method is additionally an instance of a new algorithmic design principle we refer to as higher-order divide-and-conquer, or multi-recursion [6], which is appropriate for a wide range of problems which includes what we call generalized N-body problems: those involving distances or potentials between points in a multi-dimensional space. Future work will continue to apply this principle to problems in statistical learning.

Acknowledgments

The authors would like to thank Larry Wasserman of the CMU Statistics Dept. for helpful discussions regarding current statistical practice in density estimation. This work is supported by the NASA Graduate Research Fellowship.

References

[1] K. Deng and A. W. Moore. Multiresolution Instance-based Learning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 1233–1239, San Francisco, 1995. Morgan Kaufmann.

[2] L. Devroye and L. Gyorfi. *Nonparametric Density Estimation: The L_1 View.* Wiley, 1985.

[3] V. A. Epanechnikov. Nonparametric Estimation of a Multidimensional Probability Density. *Theory of Probability and its Applications*, 14:153–158, 1969.

[4] J. Fan and J. Marron. Fast Implementations of Nonparametric Curve Estimators. *Journal of Computational and Graphical Statistics*, 3:35–56, 1994.

[5] A. Gray and A. W. Moore. N-Body Problems in Statistical Learning. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13 (December 2000).* MIT Press, 2001.

[6] A. G. Gray. *Higher-Order Divide-and-Conquer and Generalized N-Body Problems: Algorithmic Advances in Statistical Learning, Computational Geometry, and Numerical Methods.* PhD. Thesis, Carnegie Mellon University, Computer Science Department, forthcoming.

[7] A. G. Gray and A. W. Moore. Nonparametric Density Estimation: Toward Computational Tractability. submitted to NIPS 2002, available from `http://www.cs.cmu.edu/~agray/kde.ps`, 2002.

[8] J. D. F. Habbema, J. Hermans, and K. van der Broek. A Stepwise Discrimination Program Using Density Estimation. In G. Bruckman, editor, *Computational Statistics*, pages 100–110. Physica Verlag, 1974.

[9] A. W. Moore. The Anchors Hierarchy: Using the Triangle Inequality to Survive High-Dimensional Data. In *Twelfth Conference on Uncertainty in Artificial Intelligence*. AAAI Press, 2000.

[10] A. W. Moore and M. S. Lee. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research*, 8, March 1998.

[11] M. Rudemo. Empirical Choice of Histograms and Kernel Density Estimators. *Scandinavian Journal of Statistics*, 9:65–78, 1982.

[12] D. W. Scott. *Multivariate Density Estimation.* Wiley, 1992.

[13] B. Silverman. Kernel Density Estimation using the Fast Fourier Transform. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 33, 1982.

[14] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall/CRC, 1986.

[15] P. Smyth and D. Wolpert. Linearly Combining Density Estimators via Stacking. *Machine Learning*, 36:59–83, 1999.

[16] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, 1995.