

Massive-Scale Kernel Discriminant Analysis: Mining for Quasars*

Ryan Riegel[†]

Alexander Gray[‡]

Gordon Richards[§]

Abstract

We describe a fast algorithm for kernel discriminant analysis, empirically demonstrating asymptotic speed-up over the previous best approach. We achieve this with a new pattern of processing data stored in hierarchical trees, which incurs low overhead while helping to prune unnecessary work once classification results can be shown, and the use of the Epanechnikov kernel, which allows additional pruning between portions of data shown to be far apart or very near each other. Further, our algorithm may share work between multiple simultaneous bandwidth computations, thus facilitating a rudimentary but nonetheless quick and effective means of bandwidth optimization. We apply a parallelized implementation of our algorithm to a large data set (40 million points in 4D) from the Sloan Digital Sky Survey, identifying approximately one million quasars with high accuracy. This exceeds the previous largest catalog of quasars in size by a factor of ten.

Keywords: Kernel Discriminant Analysis, Scalable Algorithm, Classification, Bandwidth Learning, Astrophysics

1 Classification Via Density Estimation

Kernel discriminant analysis (KDA) [3], also called kernel density classification [8] or nonparametric Bayes classification, is a nonparametric method for predicting the class of query observations given a set of labeled reference observations. It is particularly useful in scientific applications due to its balance of properties:

1. It is highly accurate, owing to its nonparametric form.
2. It is easy to use in that it requires little understanding of the problem’s underlying model but still offers an intuitive means of incorporating prior knowledge.
3. If desired, its classifications are also accompanied by highly accurate probabilities of inclusion in each class, which can be useful in their own right.

We are motivated by a high-profile application in astrophysics pertaining to the identification of quasars. Believed to be active galactic nuclei, quasars are the brightest objects in the universe, typically out-shining their entire host galaxy by several orders of magnitude. As a result, they are the most distant and thus oldest objects we can see, making knowledge of their locations invaluable to the verification of theories such as dark energy. Prior to work with our astrophysicist collaborators [14], the largest available catalog of quasars listed fewer than one hundred thousand examples; massive sky surveys such as the Sloan Digital Sky Survey (SDSS), on the other hand, are expected to contain millions. Our goal is then to predict which of the unlabeled objects in the survey are quasars given a comparatively small (though still prohibitively large) sample of hand-identified objects.

Until recently, there was no feasible means of performing exact KDA when both the number of queries and number of references were large; traditionally, KDA is $O(N^2)$ when there are $O(N)$ queries and $O(N)$ references. Further, existing approximate methods do not provide hard bounds on introduced error. A new algorithmic approach presented in [7], however, offers marked improvement to running-time with *no* introduction of error. In this paper, we describe a new algorithm similar to that of [7] which achieves dramatic asymptotic speed-up.

The remainder of this section details the mathematics of KDA. Following that, we examine what it takes to make a fast algorithm for KDA, juxtaposing previous work with the primary contributions of this paper. Sections 3 and 4 establish a framework for our algorithm and subsequently fill in the details. Section 5 then considers accelerations to KDA’s learning phase and Section 6 discusses the possibility of parallelization. In Section 7, we demonstrate our algorithm’s superior performance on two data sets.

1.1 Bayes’ Rule in Classification. The optimal classifier on M classes assigns observation $x \in \mathcal{X}$ to the class C_k , $1 \leq k \leq M$, that has the greatest posterior probability $P(C_k|x)$ [13]. Applying Bayes’ rule,

$$(1.1) \quad P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

*We would also like to acknowledge astrophysicists Robert Nichol and Robert Bruner and code developer Garrett Boyer

[†]Georgia Institute of Technology

[‡]Georgia Institute of Technology

[§]Johns Hopkins University

we assign x to C_k if, for all $l \neq k$, $1 \leq l \leq M$, we have

$$(1.2) \quad f(x|C_k)P(C_k) > f(x|C_l)P(C_l),$$

where $f(x|C)$ denotes the probability density of data sampled from C and $P(C)$ is the prior probability of C . (Note that Bayes' rule's denominator $f(x)$ cancels from either side.) It is typically given that $\sum_{k=1}^M P(C_k) = 1$, i.e. that there are no unexplained classes, and accordingly, that $f(x) = \sum_{k=1}^M f(x|C_k)P(C_k)$. In the case that $M = 2$, this implies that it is equivalent to classify x as C_1 when $P(C_1|x) =$

$$(1.3) \quad \frac{f(x|C_1)P(C_1)}{f(x|C_1)P(C_1) + f(x|C_2)P(C_2)} > 0.5.$$

1.2 Kernel Discriminant Analysis. In place of the typically unknown values of $f(x|C)$, KDA uses kernel density estimates of the form

$$(1.4) \quad \hat{f}_h(x|C) = \frac{1}{N} \sum_{i=1}^N K_h(x, z_i),$$

trained with bandwidth h on a size N set of reference observations $z_i \in \mathcal{X}$ of class C . Kernel K_h may take many forms but must be non-negative and have

$$(1.5) \quad \int_{\mathcal{X}} K_h(x, z) dx = 1$$

for all z , i.e. it must be a probability density function. For $\mathcal{X} = \mathbb{R}^D$, a popular choice of kernel is the Gaussian,

$$(1.6) \quad K_h(x, z) = \frac{1}{(h\sqrt{2\pi})^D} \exp\left(\frac{-\|x - z\|^2}{2h^2}\right),$$

though the optimal kernel for density estimation is in fact the Epanechnikov kernel [15],

$$(1.7) \quad K_h(x, z) = \max\left\{\frac{D+2}{2V_D(h)}\left(1 - \frac{\|x - z\|^2}{h^2}\right), 0\right\},$$

where $V_D(h)$ is the volume of a D -dimensional sphere with radius h . Observe that the Epanechnikov kernel is a parabolic function of distance for $\|x - z\| < h$ and zero otherwise; we will later exploit this property to accelerate computation. Also, note that many kernels are actually functions of distance $d(x, z) = \|x - z\|$; later in this paper, K_h of one argument is understood to work on the distance term directly.

Proper selection of bandwidths h_k , $1 \leq k \leq M$, is critical for accurate classification results. Theory suggests that the optimal bandwidth for density estimation is related to a data set's variance and the inverse fifth-root of its size [15], but in practice, it is most effective to choose bandwidths that minimize KDA's cross-validation error or some other measure of loss.

1.3 Extensions to the Classifier. It is not uncommon to have additional information $y \in \mathcal{Y}$ available for each observation x that is nonetheless difficult to incorporate into density estimates $\hat{f}_h(x|C)$. For instance, corresponding information in \mathcal{Y} may not be available for all of the references, or \mathcal{Y} may not be favorable to kernels (e.g. may have no workable interpretation as \mathbb{R}^D). If it is reasonable to assume that y has negligible effect on class-conditional densities, we may still capture its effect in the class' priors with

$$(1.8) \quad P(C|x, y) \propto f(x|C, y)P(C|y) \approx f(x|C)P(C|y).$$

Similarly, for fine-tuned control over density estimates, we may provide weight w_i with each reference z_i , giving

$$(1.9) \quad \hat{f}_h(x|C) = \sum_{i=1}^N w_i K_h(x, z_i),$$

where $\sum_{i=1}^N w_i = 1$.

Another modification possible in the $M = 2$ case replaces the 0.5 in (1.3) with an alternate confidence threshold t . Rather than performing optimal classification, this has the effect of identifying points with very high (or low) probability of being in C_1 , which can be a useful surrogate for fully computed class probabilities. Simple algebra and incorporation of the above gives us

$$(1.10) \quad (1 - t)\hat{f}_{h_1}(x|C_1)P(C_1|y) > t\hat{f}_{h_2}(x|C_2)P(C_2|y);$$

we will use this as our classification test for the remainder of this paper. This modification may seem trivial from the mindset of exhaustive computation, which provides fully computed class probabilities for each query, but it is useful under the new algorithmic approach. Our algorithm deals only in bounds on probabilities and aims to terminate computation on a query as soon as it is possible to demonstrate (1.10) or its converse. While we will know that the query's probability is definitely above or below the threshold, we have no guarantee of being able to show by how far.

2 Computational Considerations

An obvious means of performing KDA is to compute the full kernel summation for each query. For $O(N)$ queries and $O(N)$ references, this naïve algorithm is $O(N^2)$ overall. While this approach is trivial to implement even in parallel, it does not scale to large N . In the best of conditions, a compute cluster with ten thousand nodes would only be able to handle a problem two orders of magnitude larger than is feasible on a single machine.

In order to make asymptotic improvement to running time, it is necessary to avoid a great deal of the

explicit kernel evaluations. A new computational approach developed in [7] achieves this by rearranging work in a manner that permits consideration of subsets of queries and references at an abstract level. It is then hoped that bounds on results obtained at abstract levels can demonstrate that further work is unnecessary and can thus be *pruned*. For example, as suggested above, we may prune all remaining work on a subset of queries if density bounds prove (1.10) or its converse for each of the queries.

2.1 Previous Work. The algorithm presented in [7] is best described as a high-order divide-and-conquer method for the $M = 2$ case of KDA. In a nutshell, it constructs two binary spatial trees, one on the query data and one on the reference data, with pairs of nodes from either tree representing abstracted computations. It then iteratively refines bounds on class-conditional density estimates $\hat{f}_{h_1}(x|C_1)$ and $\hat{f}_{h_2}(x|C_2)$ until each query’s classification can be shown. Given bounding boxes for either node in a pair, bounds on the pair’s contribution to densities are computed using kernel evaluations at the pair’s minimum and maximum distances. Bounds are refined by, at each iteration, heuristically selecting an unrefined pair and replacing its contribution with the combined contributions of its four child pairs. If (1.10) becomes provably true or false for a query node, i.e. one class’ lower-bound probability becomes greater than the other’s upper-bound probability, then the queries represented by the node are appropriately labeled and further work on those queries is removed from consideration. Ideally, bounds give a query’s classification long before all references are visited exhaustively, thereby abbreviating computation while still yielding the exact result.

2.2 This Paper. In this paper, we directly extend the work done in [7]. Our new algorithm is the same in spirit as the old one and is also implemented specifically for the $M = 2$ case¹, but it benefits from several key differences:

- It uses an improved work order that better favors pruning and is implemented with reduced overhead and improved numerical stability.
- It uses the Epanechnikov kernel, which enables a new kind of pruning based on bounded distances between node pairs.
- It can compute results for multiple bandwidth combinations simultaneously, sharing work to acceler-

ate bandwidth optimization.

- It can parallelize computation over the queries, making use of spatial information to reduce each processor’s overall workload.

We achieve empirically asymptotic improvement over the previous best algorithm, met with multiple orders of magnitude improvement on the data sets we examined.

3 Foundations

Previous work [7] does not formalize the manner of rearranging work it uses to make efficient computation possible. We include the underlying mathematics here for the purposes of rigor and helping to clarify how the algorithm presented in this paper works.

In all of the following, X is a subset of the full query set $X^{\text{root}} \subset \mathcal{X} \times \mathbb{R}$. Queries are given by pairs (x, π) , where x is used in density estimation and $\pi = P(C_1|y)$. Symbol Z_k represents a subset of the weighted references $Z_k^{\text{root}} \subset \mathcal{Z} \times \mathbb{R}$ available for for class C_k . References are pairs (z, w) , where z is used in kernel computations and w is the weight. Note that the sum of weights of Z_k^{root} equals 1, but the same of Z_k generally does not. We use “root” to denote full sets because they are semantically equivalent to the roots of trees used by our algorithm.

3.1 Recursive Formulation. We represent density estimation with a set of key-value pairs $\phi_k(X, Z_k)$, where

$$(3.11) \quad \phi_k(X, Z_k) = \left\{ \left(x, \sum_{(z,w) \in Z_k} w K_{h_k}(x, z) \right) \mid (x, \pi) \in X \right\}.$$

Classification is then a task of comparing density estimates for matching keys in $\phi_1(X, Z_1)$ and $\phi_2(X, Z_2)$ in accordance with (1.10).

For partitions $X^L \cup X^R = X$ and $Z_k^L \cup Z_k^R = Z_k$, observe that

$$(3.12) \quad \phi_k(X, Z_k) = \phi_k(X^L, Z_k) \cup \phi_k(X^R, Z_k)$$

$$(3.13) \quad = \phi_k(X, Z_k^L) + \phi_k(X, Z_k^R),$$

where addition on sets of key-value pairs is understood to add values for matching keys. This immediately suggests a recursive alternative to the naïve algorithm in which the query set and reference sets are repeatedly split until they become singleton, whereupon ϕ_k may be computed directly. Note that this reformulation by itself does nothing to help asymptotic running time because, without pruning, each of the $O(N^2)$ kernel evaluations will ultimately occur. What it does offer

¹Note, however, that no inherent properties of the algorithm prevent it from being extended to $M > 2$.

is the occurrence of $\phi_k(X, Z_k)$ for nontrivial subsets $X \subseteq X^{\text{root}}$ and $Z_k \subseteq Z_k^{\text{root}}$, instrumental to bounds computation and pruning.

Our algorithm cannot afford to spend much time deciding splits $X^L \cup X^R = X$, etc., so we use space partitioning trees to decide these splits in advance. This has the effect of forcing the same partitions to occur throughout computation, though this is not necessary in the underlying math.

3.2 Bounds Computation. It is typically possible to bound kernel results between points in X and Z_k if we know bounding boxes or bounding radii for these sets. In the case of the Gaussian and Epanechnikov kernels, bounds are functions of upper- and lower-bound distances $d^u(X, Z_k)$ and $d^l(X, Z_k)$ between the regions containing X and Z_k , with

$$(3.14) \quad K_{h_k}^u(X, Z_k) = K_{h_k}(d^l(X, Z_k))$$

and vice versa. Bounded contributions to density then assume that all references are either maximally near to or maximally far from the queries, and are given by

$$(3.15) \quad \phi_k^u(X, Z_k) = W(Z_k)K_{h_k}^u(X, Z_k)$$

and similarly for $\phi_k^l(X, Z_k)$, where

$$(3.16) \quad W(Z_k) = \sum_{(z, w) \in Z_k} w.$$

Note that bounds have zero width when X and Z_k are singleton and are thus equivalent to exact computation.

To make use of these bounds in classification, we must compose them to form bounds on the full density estimate, i.e. such that they bound a kernel summation over the full reference set. For a query $(x, \pi) \in X$, this is achieved by combining the bounds of P components of computation with

$$(3.17) \quad \Phi_k^u(x) \leftarrow \sum_{p=i}^P \phi_k^u(X^p, Z_k^p)$$

and similarly for $\Phi_k^l(x)$, where Z_k^p , $1 \leq p \leq P$, forms a partition of Z_k^{root} and $(x, \pi) \in X^p$ for each p . We further define

$$(3.18) \quad \Phi_k^u(X) \leftarrow \max_{(x, \pi) \in X} \Phi_k^u(x), \quad \Phi_k^l(X) \leftarrow \min_{(x, \pi) \in X} \Phi_k^l(x),$$

which can be computed directly if $X \subseteq X^p$ for all p .

3.3 Iterative Refinement. Computation under the new algorithmic approach implicitly constructs two binary expression trees, one for either class, in accordance with the recursive formulation given in (3.12) and

(3.13). The nodes of these trees represent partial density estimations $\phi_k(X, Z_k)$, with interior nodes symbolically composing their children via \cup or $+$. Operation begins with trees initialized to single nodes for $\phi_1(X^{\text{root}}, Z_1^{\text{root}})$ and $\phi_2(X^{\text{root}}, Z_2^{\text{root}})$, and at each iteration *expands* a selected, non-singleton leaf into two children reflecting having split either its queries or references. After expansion, the algorithm reassesses bounds on the full density estimates for the involved queries as in (3.17), using bounded results computed for the newly created leaves in conjunction with bounds already available from other leaves.² We stop expanding nodes $\phi_k(X, Z_k)$ for which we can show either of

$$(3.19) \quad (1-t)\Phi_1^l(X)\Pi^l(X) > t\Phi_2^u(X)(1-\Pi^l(X)),$$

$$(3.20) \quad t\Phi_2^l(X)(1-\Pi^u(X)) > (1-t)\Phi_1^u(X)\Pi^u(X),$$

where

$$(3.21) \quad \Pi^u(X) = \max_{(x, \pi) \in X} \pi, \quad \Pi^l(X) = \min_{(x, \pi) \in X} \pi,$$

which decides (1.10) for all queries in X .

It is easy to see that this procedure terminates. For finite X and Z_k , recursion will ultimately reach singleton sets $X = \{(x, \pi)\}$ and $Z_k = \{(z, w)\}$, where upon contribution bounds computations become exact, i.e. the upper and lower bound become equal, and recursion along that branch ceases. If all branches for some x reach singleton leaves, then bounds on the full density estimate are also exact and thus one of (3.19) or (3.20) must be true³ for each singleton X . If branches do not reach leaves then it is only because x has already been pruned. Thus, all queries x will eventually prune and iterative refinement will stop. Correctness is more difficult to show rigorously; we omit the proof because it is nonetheless fairly intuitive.

4 The Algorithm

In the previous section, we did not impose any requirements on how one chooses partitions $X^L \cup X^R = X$, etc., or on the order of leaves expanded during computation. These details have no impact upon correctness but are nonetheless critical with regards to running time. An additional concern is the practical storage of computed bounds in a manner that assists with finding bounds on the full density estimate. Addressing these issues, we also describe a new pruning opportunity and the bookkeeping that facilitates it, ultimately arriving at the algorithm in Figure 1.

²It is easy to prove by induction that, for any $(x, \pi) \in X$, we can find leaves that satisfy the requirements of (3.17).

³Or either side is exactly equal, but this is degenerate and may be thought of as a third, nonclassifying prune.

4.1 Spatial Trees. Both the algorithm in [7] and the one presented in this paper use *kd*-trees [12], a kind of spatially-informed tree, to facilitate the splitting of query and reference sets. We construct two trees, one for X^{root} and one for the combined⁴ references Z_1^{root} and Z_2^{root} , in which each interior node represents a partitioning $X^L \cup X^R = X$, etc.; we will reuse these partitions any time we need to split data later in computation. Points stored beneath each node are represented by their bounding box and child nodes are formed by splitting the widest dimension of the parent’s bounding box along its midpoint. Leaves are formed when the number of represented points drops below a specified threshold; the algorithm stops partitioning a set once it reaches a leaf and computes $\phi_k(X, Z_k)$ exhaustively if both X and Z_k are leaves. Trees also store useful statistics on represented queries and references, such as bounds on prior probability $[\Pi^l(X), \Pi^u(X)]$ and summed reference weight $W(Z_k)$, which may be computed rapidly in a bottom-up manner. Lastly, the query tree provides a convenient location to store sums of computed bounds; [7] makes heavy use of this, but the algorithm we propose does not.

The primary motivation for spatial trees is that they tend to tighten bounds rapidly as we expand $\phi_k(X, Z_k)$ even though splits are made without any knowledge of interaction between X and Z_k . In other words, we would not expect to see gigantic improvement in bounds from choosing custom partitions for each expansion, a comparatively expensive procedure. Tree building incurs $O(N \log N)$ computations, but the cost of this step is small compared to the rest of the algorithm except for very large data sets. It is also possible to reuse existing spatial trees on the data, perhaps computed because some prior algorithm needed them as well.

Due to the conjoined nature of our reference tree, matching simultaneous expansions are made to $\phi_1(X, Z_1)$ and $\phi_2(X, Z_2)$. Bounds and pruning on these computations are informed by separate bounding boxes and statistics stored for either class; if ever a reference node ceases to contain points from one of the classes, the node is removed from consideration when bounding that class’ kernel summation. We will use $Z_{1,2}$ to denote nodes of the combined tree, from which both Z_1 and Z_2 are available.

Because sets X and Z_k occurring in computation constitute nodes from *kd*-trees, we will henceforth refer to components of computation $\phi_k(X, Z_k)$ as *node-pairs*.

4.2 Expansion Pattern. To guide computation, [7] uses a heuristic favoring the expansion of node-pairs

that provide the most bounds tightening over the contribution of their parents. This may sound desirable with regards to testing (3.19) and (3.20), but it does not guarantee that further expansion will provide much improvement and is prone to missing node-pairs that are just about to make a “breakthrough.” A further problem of this kind of expansion is its dependence upon priority queues (implemented, for instance, as a heap), which in our experience impose significant computational overhead and cache inefficiency. Also, because potentially any query node may follow the present computation, it is necessary to propagate and later undo all computed bounds to all levels of the query tree to which they apply, possibly impacting numerical stability.

Our algorithm uses a non-heuristic expansion pattern best thought of as a hybrid between depth- and breadth-first. In short, queries are handled depth-first while references are handled breadth-first. This pattern’s associated data structure is a stack of lists of node-pairs, where each list pertains to one query node X and all reference nodes $Z_{1,2}^p$, $1 \leq p \leq P$, at some level of the reference tree, i.e. with $\bigcup_{p=1}^P Z_{1,2}^p = Z_{1,2}^{\text{root}}$. Bounds computation over a list consists of accumulating the partial bounds found for all node-pairs $\phi_1(X, Z_1^p)$ and $\phi_2(X, Z_2^p)$ as per (3.17). Query splits are performed by replacing the list at the top of the stack with two equal-length lists pertaining to either child of X and the same nodes $Z_{1,2}^p$. Reference splits instead replace the list with an at most double-length list pertaining to the same X and all of the children of nodes $Z_{1,2}^p$. Leaf nodes $Z_{1,2}^p$ are left unsplit until X becomes a leaf, at which point they are processed exhaustively and removed, their contribution added to exact results stored for each query point. In practice, we always split queries and references simultaneously if possible. Processing lists in this manner results in a logarithmically deep stack containing lists of sizes doubling from 1 to $O(N)$, thus consuming $O(N)$ space overall. This is more overhead than depth-first’s $O(\log N)$ space, but significantly less than breadth-first’s $O(N^2)$ (pessimistically assuming that no prunes occur).

Because bounds computed for one set of queries do not influence the results of another, the order in which nodes are processed from the query tree does not have any effect on pruning. The new expansion pattern exploits this to minimize overhead while still offering tightened bounds from all parts of the reference tree, crucial in demonstrating queries’ classifications. Further, each list in the new expansion pattern offers sufficient information to find the most current bounds on X ’s full kernel summation: the respective sums of upper and lower bounds obtained for the represented node-pairs. This eliminates any need to propagate or

⁴Alternately, separate trees could be used for each class.

undo bound contributions throughout the query tree, saving time as well as minimizing the effect of limited floating-point precision.

4.3 Pruning. We have already motivated *classification pruning*, where one class' lower-bound probability exceeds the other's upper-bound probability. Each time the hybrid expansion pattern forms a new work list, it finds bounds on the full density estimates and tests (3.19) and (3.20); if either of these are true, it labels the represented queries appropriately and pops the list from the stack.

The Epanechnikov kernel makes possible two other very important pruning opportunities. As mentioned before, this kernel is parabolic for distances smaller than bandwidth h and zero otherwise. Thus, if the lower-bound distance between X and Z_k is greater than the bandwidth, those points' contribution to the kernel sum will be exactly zero. This makes for a convenient *exclusion prune*: node-pairs $\phi_{1,2}(X, Z_{1,2})$ with $d^l(X, Z_{1,2}) \geq \max\{h_1, h_2\}$ may be removed from the hybrid expansion pattern's work lists.

Similarly, the Epanechnikov kernel's parabolic interior enables *inclusion pruning*, though this is somewhat more involved. Because a sum of parabolas is a parabola, we may compute the exact kernel summation over a set of references Z_k at once if we can prove it is entirely within bandwidth h of some query x . By extension, if $d^u(X, Z_{1,2}) \leq \min\{h_1, h_2\}$, we may exactly compute $\phi_{1,2}(X, Z_{1,2})$ in $O(N)$ or find tighter than normal bounds in constant time. The derivation of this prune observes that square distance may be computed

$$(4.22) \quad (x - z) \cdot (x - z) = x \cdot x - 2x \cdot z + z \cdot z,$$

ultimately yielding the expression

$$(4.23) \quad W(Z_k) \frac{D+2}{2V_D(h_k)} \left(1 - \frac{\delta(x, Z_k)}{h_k^2} \right)$$

in place of the original kernel summation, where

$$(4.24) \quad \delta(x, Z_k) = x \cdot x - 2x \cdot \frac{S(Z_k)}{W(Z_k)} + \frac{T(Z_k)}{W(Z_k)},$$

$$(4.25) \quad S(Z_k) = \sum_{(z,w) \in Z_k} wz, \quad T(Z_k) = \sum_{(z,w) \in Z_k} wz \cdot z;$$

note that $S(Z_k)$ and $T(Z_k)$ may be precomputed at the same time as $W(Z_k)$. This constitutes a parabola centered at $\frac{S(Z_k)}{W(Z_k)}$. To employ this in pruning, each work list must record respective sums W_k , S_k , and T_k of $W(Z_k)$, $S(Z_k)$, and $T(Z_k)$ for pruned nodes Z_k , replicating these sums into future lists formed via expansion. These terms will eventually provide exact density contributions to individual queries once X is

```

init  $X = X^{\text{root}}$ ,  $\text{work} = \{Z_{1,2}^{\text{root}}\}$ 
init  $W_{1,2} = 0$ ,  $S_{1,2} = 0$ ,  $T_{1,2} = 0$ 
function  $\text{kda}(X, \text{work}, W_{1,2}, S_{1,2}, T_{1,2})$ 
  init  $\Phi_1^l = 0$ ,  $\Phi_1^u = 0$ ,  $\Phi_2^l = 0$ ,  $\Phi_2^u = 0$ 
  init  $\text{next\_work} = \{\}$  // filled below
  for each  $Z_{1,2} \in \text{work}$ 
    if  $d^u(X, Z_{1,2}) \leq \min\{h_1, h_2\}$ ,
      // Inclusion prune; add to parabola
       $W_1 += W(Z_1)$ ;  $W_2 += W(Z_2)$ 
      ... // Similar for  $S_{1,2}$  and  $T_{1,2}$ 
    elif  $d^l(X, Z_{1,2}) < \max\{h_1, h_2\}$ ,
      // No prune; record contribution
       $\Phi_1^l += W(Z_1)K_{h_1}(d^u(X, Z_1))$ 
      ... // Similar for  $\Phi_1^u$ ,  $\Phi_2^l$ , and  $\Phi_2^u$ 
      add  $Z^L$  and  $Z^R$  to  $\text{next\_work}$ 
    else // Exclusion prune; do nothing
      // Account for inclusion prunes
       $\Phi_1^l += W_1 K_{h_1}(\delta^u(X, W_1, S_1, T_1))$ 
      ... // Similar for  $\Phi_1^u$ ,  $\Phi_2^l$ , and  $\Phi_2^u$ 
      if  $(1-t)\Phi_1^l \Pi^l(X) > t\Phi_2^u(1-\Pi^l(X))$ ,
        label  $X$  as  $C_1$ ; return
      if  $t\Phi_2^l(1-\Pi^u(X)) > (1-t)\Phi_1^u \Pi^u(X)$ ,
        label  $X$  as  $C_2$ ; return
       $\text{kda}(X^L, \text{next\_work}, W_{1,2}, S_{1,2}, T_{1,2})$ 
       $\text{kda}(X^R, \text{next\_work}, W_{1,2}, S_{1,2}, T_{1,2})$ 

```

Figure 1: Pseudocode for kernel discriminant analysis. Each recursion begins by testing all nodes from a level of the reference tree against the query node, pruning those it can while simultaneously accumulating the contributions of others and building the next level's work list. Pruned contributions are then applied to computed density bounds and bounds are tested to see if they demonstrate a classification. Handling of base-case omitted.

a leaf, but before expansion reaches that point, we compute tighter than normal bounds with

$$(4.26) \quad W_k K_{h_k}(\delta^l(X, W_k, S_k, T_k))$$

and vice versa, where

$$(4.27) \quad \delta^u(X, W_k, S_k, T_k) = d^u\left(X, \frac{S_k}{W_k}\right) - \frac{S_k}{W_k} \cdot \frac{S_k}{W_k} + \frac{T_k}{W_k}$$

and similarly for $\delta^l(X, W_k, S_k, T_k)$.

5 Bandwidth Learning

Class-dependent bandwidths h_k have a significant impact on classification accuracy, making the determination of optimal bandwidths a critical phase in any application of KDA. In practice, the best approach to bandwidth optimization is minimizing cross-validation classification error. Our algorithm is capable of leave-one-out cross-validation (LOO CV) at negligible additional cost. Further, a specialized multi-bandwidth version of

our algorithm shares work between multiple bandwidth combinations, greatly reducing the time necessary to explore the space of bandwidth settings.

Fast bandwidth optimization is arguably the most significant contribution of this paper. Often times, full density estimates are more useful than just classifications and existing fast algorithms can approximate these with high accuracy [5, 6, 10, 9]. Regardless, these estimators must undergo the same learning as KDA before their results can be trusted. Classification pruning, available in KDA but not regular kernel density estimation, can greatly accelerate this preliminary phase. Further, the results of the KDA learning phase may be used to restrict computations of full density estimates to just those points identified as members of some class of interest.

5.1 Leave-one-out Cross-validation. In cross-validation situations, queries X^{root} and references $Z_{1,2}^{\text{root}}$ represent the same data. It is handy to denote data as a set $X_{1,2} \subset \mathcal{X} \times \mathbb{R} \times \mathbb{R}$, separable into sets X_1 and X_2 for either class and with elements (x, π, w) : the observation, its data-dependent prior, and its reference weight.

Our goal is to find the results for each (x, π, w) that would be obtained from normal computation if $Z_{1,2} = X_{1,2} \setminus (x, \pi, w)$. It is sufficient and safe to relax density bounds computed for $X_{1,2}$ to reflect having left out one point from the appropriate classes. In the case of the Gaussian and Epanechnikov kernels, a point certainly contributes the maximum kernel value to itself. Leaving a point out then results in subtracting the maximum kernel value at that point’s weight from both the upper and lower bounds for its class, renormalizing bounds such that weights again sum to one. Because any changes we make to bounds must apply to all points represented by $X_{1,2}$, we must find a lower bound on changes made to the lower bounds and an upper bound on changes made to the upper bounds. If $X_{1,2}$ contains points of either class, then a suitable modification to the lower bounds is to subtract the maximum kernel value at the maximum represented weight but *not* renormalize, and a suitable change to the upper bounds is to renormalize *without* having subtracted anything. The new bounds are then

$$(5.28) \quad \Phi_k^l(X_{1,2}) - w^u(X_k)K_{h_k}(0), \quad \frac{\Phi_k^u(X_{1,2})}{1 - w^u(X_k)},$$

where

$$(5.29) \quad w^u(X_k) = \max_{(x, \pi, w) \in X_k} w.$$

Clearly, if $X_{1,2}$ contains only points from one class (i.e. one of X_1 or X_2 is empty), then tighter modifica-

tions can be made. This is imperative in the base-case because a classification may never be found otherwise.

5.2 Multi-bandwidth Computation. Classification accuracy is nondifferentiable, preventing the use of most rapidly converging optimization techniques to find favorable bandwidths. Indeed, it is not uncommon for researchers to test bandwidths by hand until accuracy seems to have peaked, a time-consuming process prone to error and wasted effort. Here, we describe an extension to our algorithm that swiftly and systematically considers all pairs of two sets of bandwidths.

Given sorted lists of bandwidths h_1^i , $1 \leq i \leq I$, and h_2^j , $1 \leq j \leq J$, we modify the standard algorithm such that it maintains vectors of corresponding length for computed density bounds $\vec{\Phi}_k^{l,u}$ and pruning information \vec{W}_k , \vec{S}_k , and \vec{T}_k . The modified algorithm iterates over the bandwidths from largest to smallest when considering each item from the work list, reusing distance computations and shortcutting computation on smaller bandwidths when a larger one prunes via exclusion. We annotate work items Z_k with $\text{lo}(Z_k)$ and $\text{hi}(Z_k)$, the lowest and highest indices of bandwidths not yet pruned via inclusion or exclusion. Iteration is constrained to between these indices, and the work item is removed from future lists if no unpruned bandwidths remain. The algorithm then considers all pairs of bandwidths for classification pruning. We also annotate query nodes X with $\text{lo}_k(X)$ and $\text{hi}_k(X)$ for either class, the lowest and highest indices of bandwidths for which classifications are yet to be made. Iteration for both density computation and classification is constrained to within these indices, and the entire work list is popped when the query node has been classified for all bandwidth pairs. Pseudocode for this algorithm is shown in Figure 2.

There is only a factor of $O(I+J)$ more work needed to compute the bandwidths’ associated densities, but there are $O(IJ)$ classification tests to perform at each level of recursion. The classification test is, however, very fast in comparison to density estimation. We should then expect to see running time grow roughly linearly for bounded values of I and J . The sharing of tree building, distance computations, and exclusion pruning also provide speed-up over running the standard algorithm on all of the bandwidth combinations.

We may use this multi-bandwidth version of the algorithm in a rudimentary but effective means of bandwidth optimization. Starting from theoretic approximations of the optimal bandwidths [15] or some simpler measure such as the average k th-nearest-neighbor distance⁵, test ten or so bandwidths ranging from an order

⁵A fast algorithm for this problem is suggested in [5].

of magnitude larger to an order of magnitude smaller. Then, iteratively narrow ranges to contain just the few bandwidths found to yield the greatest accuracy, or extend the range if the greatest accuracy bandwidths lie on the edge. This can only hope to offer linear convergence upon a local optimum, but it is conceptually similar to the well-known Nelder-Mead method of optimization and in practice finds reasonable bandwidths in just a few iterations. It is also possible to reuse trees between iterations.

6 Parallelization

The naïve algorithm for KDA is highly parallelizable, but parallelized implementations still suffer from the method’s inherent poor scalability. Even assuming communication cost to be negligible, it is likely uneconomical to purchase one-hundred-fold more processors to tackle a problem just one order of magnitude larger than was previously feasible.

The algorithm we propose is also parallelizable. There are no read/write dependencies between sibling recursions; thus, it is safe for these tasks to be handled by different processors. For P processors and grain size parameter G , we perform median (rather than midpoint) splits in the query tree up to depth $\lceil \log PG \rceil$, enqueueing each node at that depth in a global work list. This ensures that there are at least G grains per processor and that grains are all about the same size. As they become available, processors are assigned computations $\phi_{1,2}(X, Z_{1,2}^{\text{root}})$ for successive nodes X in the global work list. The use of multiple grains per processor helps alleviate inevitable irregularities in the amount of time it takes to classify different regions of the queries; using too many grains, however, both incurs communication overhead and reduces the degree to which work can be shared between queries.

This method of parallelization should be more advantageous than simply breaking the queries into chunks before forming trees. The initial query nodes assigned to processors are more compact than they would otherwise be, thus encouraging the earlier pruning of much of the reference set. This has the effect of restricting the amount of the reference tree needed by individual processors, which can be exploited by an advanced data distribution system to reduce overall communication. We must note, however, that this method presupposes the formation of the query tree before parallelization can begin. At present, tree building is performed in serial, but future work aims to address whether that step can also be parallelized.

```

init  $X = X^{\text{root}}$ ,  $\text{work} = \{Z_{1,2}^{\text{root}}\}$ 
init  $\vec{W}_{1,2} = 0$ ,  $\vec{S}_{1,2} = 0$ ,  $\vec{T}_{1,2} = 0$ 
function  $\text{multi}(X, \text{work}, \vec{W}_{1,2}, \vec{S}_{1,2}, \vec{T}_{1,2})$ 
  init  $\vec{\Phi}_1^l = 0$ ,  $\vec{\Phi}_1^u = 0$ ,  $\vec{\Phi}_2^l = 0$ ,  $\vec{\Phi}_2^u = 0$ 
  init  $\text{next\_work} = \{\}$  // filled below
  for each  $Z_{1,2} \in \text{work}$ 
    for  $i$  from  $\text{hi}(Z_1)$  to  $\text{lo}(Z_1)$ 
      if  $d^u(X, Z_1) \leq h_1^i$ 
        // Inclusion prune; add to parabolas
         $\vec{W}_{1(i)} += W(Z_1)$ 
        ... // Similar for  $\vec{S}_1$  and  $\vec{T}_1$ 
      else break
    for  $i$  continuing to  $\text{lo}(Z_1)$ 
      if  $d^l(X, Z_1) < h_1^i$ ,
        // No prune; record contributions
         $\vec{\Phi}_{1(i)}^l += W(Z_1)K_{h_1^i}(d^u(X, Z_1))$ 
         $\vec{\Phi}_{1(i)}^u += W(Z_1)K_{h_1^i}(d^l(X, Z_1))$ 
      else break
    // No work for exclusion prunes
    update  $\text{lo}(Z_1)$  and  $\text{hi}(Z_1)$ 
    ... // Similar for  $Z_2$ 
    if  $\text{lo}(Z_1) \leq \text{hi}(Z_1)$  or  $\text{lo}(Z_2) \leq \text{hi}(Z_2)$ 
      add  $Z^L$  and  $Z^R$  to  $\text{next\_work}$ 
    for  $i$  from  $\text{lo}_1(X)$  to  $\text{hi}_1(X)$ 
      // Account for inclusion prunes
       $\vec{\Phi}_{1(i)}^l += \vec{W}_{1(i)}K_{h_1^i}(\delta^u(X, \vec{W}_{1(i)}, \vec{S}_{1(i)}, \vec{T}_{1(i)}))$ 
       $\vec{\Phi}_{1(i)}^u += W_{1(i)}K_{h_1^i}(\delta^l(X, W_{1(i)}, S_{1(i)}, T_{1(i)}))$ 
    ... // Similar for  $\vec{\Phi}_2^l$  and  $\vec{\Phi}_2^u$ 
    for  $i$  from  $\text{lo}_1(X)$  to  $\text{hi}_1(X)$ 
      for  $j$  from  $\text{lo}_2(X)$  to  $\text{hi}_2(X)$ 
        if  $(1-t)\vec{\Phi}_1^{l(i)}\Pi^l(X) > t\vec{\Phi}_2^{u(j)}(1-\Pi^l(X))$ ,
          label  $X$  as  $C_1$  for  $(i, j)$ 
        if  $t\vec{\Phi}_2^{l(j)}(1-\Pi^u(X)) > (1-t)\vec{\Phi}_1^{u(i)}\Pi^u(X)$ ,
          label  $X$  as  $C_2$  for  $(i, j)$ 
      update  $\text{lo}_1(X)$ ,  $\text{hi}_1(X)$ ,  $\text{lo}_2(X)$ , and  $\text{hi}_2(X)$ 
    if  $\text{lo}_1(X) \leq \text{hi}_1(X)$  and  $\text{lo}_2(X) \leq \text{hi}_2(X)$ 
       $\text{multi}(X^L, \text{next\_work}, \vec{W}_{1,2}, \vec{S}_{1,2}, \vec{T}_{1,2})$ 
       $\text{multi}(X^R, \text{next\_work}, \vec{W}_{1,2}, \vec{S}_{1,2}, \vec{T}_{1,2})$ 

```

Figure 2: Pseudocode for the multi-bandwidth algorithm. For either class, the algorithm checks for and handles inclusion prunes followed by normal bounds computations until the first exclusion prune. Pruned contributions are then applied for each bandwidth and bandwidth combinations are tested for classification. Handling of base-case omitted.

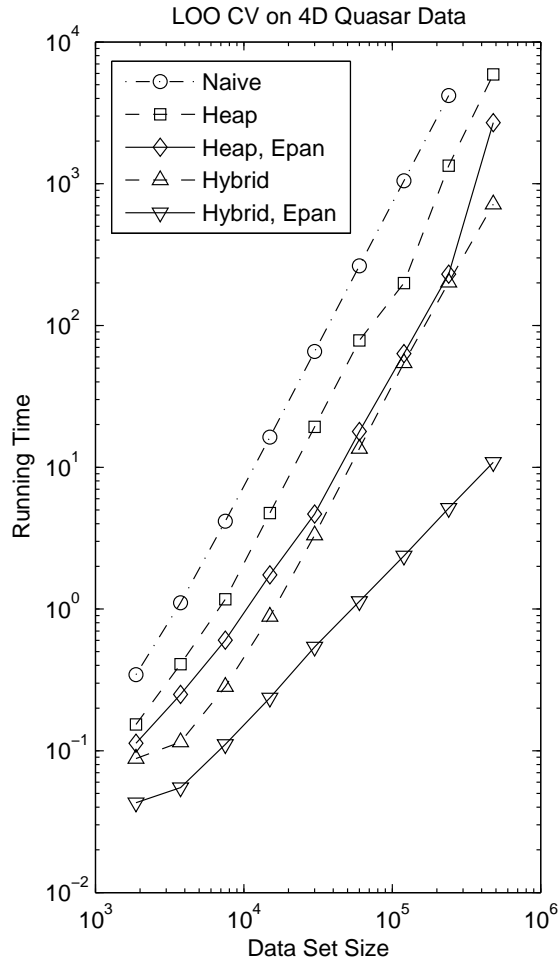


Figure 3: Running times of LOO CV on one bandwidth combination for the naïve algorithm, the algorithm from [7] (Heap), the improved expansion pattern (Hybrid) and pruning opportunities (Epan) separately, and finally the improvements together. Bandwidths were determined to maximize classification accuracy on the largest set and scaled via the theoretic inverse relationship to $N^{\frac{1}{5}}$. Only one processor used.

7 Results

We applied our algorithms to two data sets: a large selection of quasars and stars from the SDSS and the Cover Type data set available from the UCI Machine Learning Repository [1].

All experiments are conducted on code compiled by gcc 3.4.6 in Linux 2.6.9 on dual NetBurst-class Intel Xeon 3.0GHz computers with 8GB RAM.

7.1 Quasar Identification. We demonstrate timing results for KDA classifiers on 4D color spectra data pertaining to deep-space objects. The largest reference set we use contains 80k known quasars and 400k non-quasars; our query set consists of 40m unidentified ob-

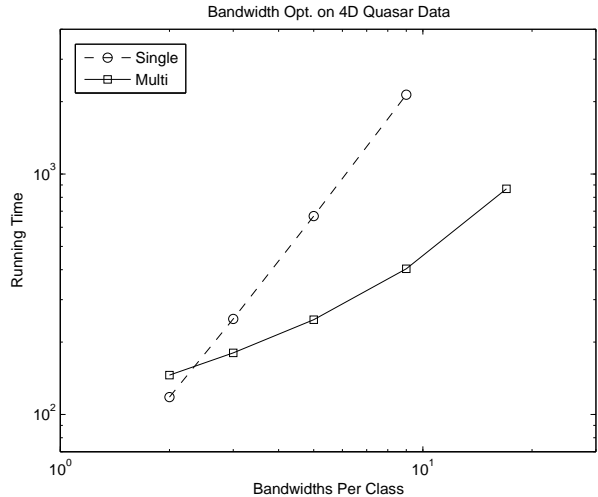


Figure 4: Running times for the multi-bandwidth algorithm and the single-bandwidth algorithm run on all bandwidth combinations. Bandwidth range is constant between runs and bandwidths are tested at linear intervals. Only one processor used.

jects. In Figure 3, it is apparent that both the hybrid expansion pattern and inclusion/exclusion pruning are crucial to asymptotic performance. Our new algorithm empirically scales at about $O(N^{1.1})$ and has an extremely low cross-over point with the naïve algorithm.

Three multi-bandwidth runs with 10 bandwidths per class, successively narrowing the tested bandwidth ranges about the optimum, take 334, 127, and 86 seconds when parallelized over two processors. Decreasing running time is expected both because bandwidths are easier to prune together if they are more similar to each other and because larger bandwidths, which diminish as ranges tighten, tend to prune the least. Figure 4 demonstrates that the multi-bandwidth algorithm is indeed much faster for a bounded number of bandwidths, which is all that is necessary for optimization. In comparison, the estimated naïve running time for the above procedure is 140 hours even if the algorithm utilizes the same density-sharing technique employed by the multi-bandwidth algorithm. Optimal bandwidths yield within-class of accuracies of 95% for quasars and 98% for stars.

Finally, we parallelize over multiple computers to tackle classification of the full data set. Using the a cosmologically informed prior of 4% for the quasar class, three dual-processor machines discover just shy of one million quasars in 456 seconds. Unfortunately, our parallelization methods see poor speed-up due to serial data access and tree building, as shown in Figure 5. These steps alone take about 310 seconds, the better portion of computation; the whole serial computation

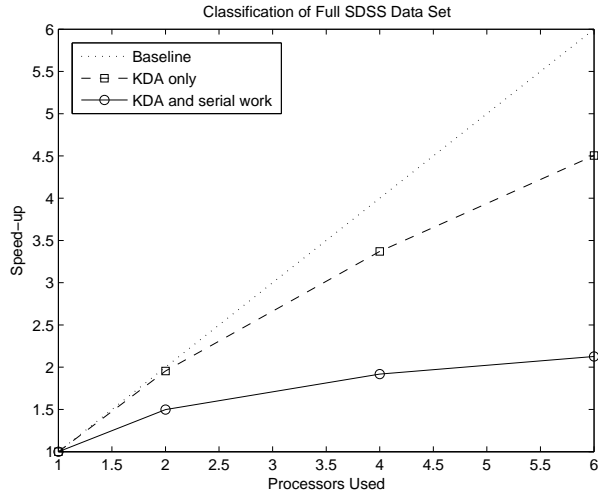


Figure 5: Speed-up for parallelized classification of the full 40m SDSS data set. Excluding data access and tree building shows that the underlying computation has workable speed-up. Future parallelization of tree building may then make this feasible.

takes only 640 seconds. An important future avenue of research is parallelizing tree building, but at current we are forced to conclude that parallelization is not worthwhile at this scale. In any case, we far improve upon both the estimated naïve running time of 380 hours, not to mention classifying the entire unknown data in a tenth the time it takes the old algorithm to test one bandwidth combination.

Using careful tuning of parameters and separate classifiers for separate regions of data, our collaborators have extracted more than one million quasars from the SDSS data. Hand-verification of subsets of this data predicts that identified quasars are 85.6% accurate and, further, that we have identified 93.4% of the quasars to be found.

7.2 Forest Cover-Type Data. Originally used in [2], the forest cover-type data is a 55-dimensional (10 continuous, 44 binary, 1 nominal classification) data set with 580k observations describing the relationship between cartographic variables and seven classes of forest cover-type, i.e. what kind of trees grow there. Because KDA is not particularly good at qualitative features (these would require scaling or a specialized kernel), some preprocessing is required. We normalize each dimension to have unit variance and then perform PCA to reduce the dimensionality to 8. Further, as implemented, our algorithm only supports two-class problems, so we will train our classifiers to distinguish one class from the rest of the data; results are shown for distinguishing the largest class, lodgepole pines, which

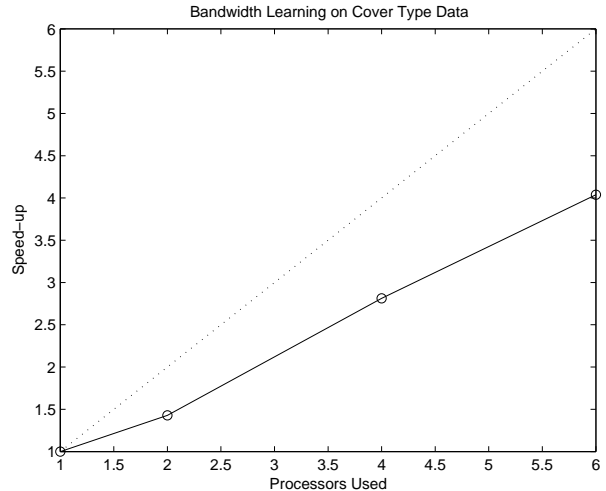


Figure 6: Speed-up for parallelized classification of LOO CV with the multi-bandwidth algorithm. This is more favorable because the multi-bandwidth algorithm performs a larger proportion of parallelizable work than does single-bandwidth classification.

constitutes nearly half of the data, and the smallest class, cottonwoods and willows, which constitutes only 0.5% of the data.

For lodgepole pines, we optimized bandwidths with three 10-by-10 runs of the multi-bandwidth algorithm, taking 609, 135, and 92 seconds when parallelized over two-processors. The resulting classification accuracy is 90% both within and outside the class. For cottonwoods and willows, the same optimization procedure takes 1037, 248, and 213 seconds and results in a within-class accuracy of 99.4% and 97.7% outside. The estimated running time of the naïve algorithm for a single bandwidth combination for the identification of either class is 10.6 hours, suggesting an overall time commitment of 320 hours for the above work, around seven orders of magnitude slower.

Parallelization of LOO CV with the multi-bandwidth algorithm was much more favorable than with the large SDSS query set. This makes sense given Amdahl’s law because the ratio of serial work to parallel work is significantly reduced: more parallel kernel summations and classification tests are performed for the serial act of tree-building. Figure 6 shows workable speed-up, suggesting that iterations of the multi-bandwidth algorithm, especially the long-running ones seen at higher dimensions, still benefit greatly from parallelization.

8 Conclusion

We have demonstrated numerous enhancements to the previous best algorithm for KDA. The hybrid expan-

sion pattern and inclusion/exclusion pruning have resulted in very clear gains in asymptotic running time. Further, the multi-bandwidth version of the algorithm and parallelization enables rapid bandwidth learning for use in classification or determination of accurate class probabilities. We have significantly extended the size of problems now tractable under this analysis, arriving at results in minutes where previously it would have taken weeks.

Our algorithm’s orders of magnitude speed-up is obtained with a minimal degree of parallelization. Through the use of a more advanced data distribution and tree building method, we can expect parallelization to yield even more impressive gains. Also, for higher-dimensional problems, techniques such as logistic regression may be used in combination with KDA [4].

Already, an earlier rendition of our quasar catalog has been used to confirm the cosmic magnification effect predicted by the theory of relativity [11]. The new, larger catalog that we have found will undoubtedly lead to many other impressive discoveries. Nonetheless, these are results from one data set in one scientific application. KDA has no special dependence upon quasar identification, and in fact, the general ideas behind our fast algorithm have no special dependence on KDA [5]. Methods like those demonstrated in this paper have potential to accelerate a wide variety of computations across science and data mining.

N	Naïve	Heap	Hp.Ep.	Hybrid	Hy.Ep.
1875	.344	.153	.113	.088	.043
3750	1.11	.408	.250	.115	.055
7500	4.17	1.17	.603	.283	.111
15k	16.3	4.75	1.74	.884	.236
30k	65.5	19.3	4.68	3.32	.540
60k	264	78.4	17.9	13.6	1.13
120k	1050	199	63.2	54.5	2.38
240k	4200	1340	230	202	5.15
480k	—	5900	2690	716	10.8

Table 1: Running times in seconds of LOO CV on one bandwidth combination for the various algorithms. Plotted on log-log axes in Figure 3.

References

- [1] A. Asuncion and D. Newman. UCI machine learning repository, 2007.
- [2] J. A. Blackard. *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. PhD dissertation, Colorado State University, Department of Forest Sciences, 1998.
- [3] E. Fix and J. L. Hodges. Discriminatory analysis, Non-parametric regression: consistency properties. Technical report, USAF School of Aviation Medicine, 1951.
- [4] A. Gray, P. Komarek, T. Liu, and A. Moore. High-dimensional probabilistic classification for drug discovery. In *COMPSTAT 2004*, 2004.
- [5] A. Gray and A. 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. Gray and A. Moore. Nonparametric Density Estimation: Toward Computational Tractability. In *SIAM International Conference on Data Mining 2003*, 2003.
- [7] A. Gray and R. Riegel. Large-Scale Kernel Discriminant Analysis with Application to Quasar Discovery. In *Proceedings of Computational Statistics*, 2006.
- [8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.
- [9] D. Lee and A. Gray. Faster gaussian summation: Theory and experiment. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, Arlington, Virginia, 2006. AUAI Press.
- [10] D. Lee, A. Gray, and A. Moore. Dual-tree fast gauss transforms. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 747–754. MIT Press, Cambridge, MA, 2006.
- [11] M. Peplow. Quasars Reveal Cosmic Magnification. *Nature*, April 27 2005.
- [12] F. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, 1985.
- [13] C. R. Rao. *Linear Statistical Inference*. Wiley, 1973.
- [14] G. Richards, R. Nichol, A. Gray, R. Brunner, and R. Lupton. Efficient Photometric Selection of Quasars from the Sloan Digital Sky Survey: 100,000 $z > 3$ Quasars from Data Release One. *Astrophysical Journal Supplement Series*, 155:257–269, 2004.
- [15] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1986.