Fast Nearest Neighbor Retrieval for Bregman Divergences

Lawrence Cayton LCAYTON@CS.UCSD.EDU
Department of Computer Science and Engineering, University of California, San Diego, CA 92093

Abstract

We present a data structure enabling efficient nearest neighbor (NN) retrieval for bregman divergences. The family of bregman divergences includes many popular dissimilarity measures including KL-divergence (relative entropy), Mahalanobis distance, and Itakura-Saito divergence. These divergences present a challenge for efficient NN retrieval because they are not, in general, metrics, for which most NN data structures are designed. The data structure introduced in this work shares the same basic structure as the popular metric ball tree, but employs convexity properties of bregman divergences in place of the triangle inequality. Experiments demonstrate speedups over brute-force search of up to several orders of magnitude.

1. Introduction

Nearest neighbor (NN) search is a core primitive in machine learning, vision, signal processing, and elsewhere. Given a database X, a dissimilarity measure d, and a query q, the goal is to find the $x \in X$ minimizing $d(x, q)$. Brute-force search is often impractical given the size and dimensionality of modern data sets, so many data structures have been developed to accelerate NN retrieval.

Most retrieval data structures are for the ℓ_2 norm and, more generally, metrics. Though many dissimilarity measures are metrics, many are not. For example, the natural notion of dissimilarity between probability distributions is the KL-divergence (relative entropy), which is not a metric. It has been used to compare histograms in a wide variety of applications, including text analysis, image classification, and content-based image retrieval (Pereira et al., 1993; Puzicha et al., 1999; Rasiwasia et al., 2007). Because the KL-divergence does not satisfy the triangle inequality, very little of the research on NN retrieval structures applies.

The KL-divergence belongs to a broad family of dissimilarities called *bregman divergences*. Other examples include Mahalanobis distance, used *e.g.* in classification (Weinberger et al., 2006); the Itakura-Saito divergence, used in sound processing (Gray et al., 1980); and ℓ_2^2 distance. Bregman divergences present a challenge for fast NN retrieval since they need not be symmetric or satisfy the triangle inequality.

This paper introduces *bregman ball trees* (bbtrees), the first NN retrieval data structure for general bregman divergences. The data structure is a relative of the popular metric ball tree (Omohundro, 1989; Uhlmann, 1991; Moore, 2000). Since this data structure is built on the triangle inequality, the extension to bregman divergences is non-trivial.

A bbtree defines a hierarchical space decomposition based on bregman balls; retrieving a NN with the tree requires computing bounds on the bregman divergence from a query to these balls. We show that this divergence can be computed exactly with a simple bisection search that is very efficient. Since only bounds on the divergence are needed, we can often stop the search early using primal and dual function evaluations.

In the experiments, we show that the bbtree provides a substantial speedup—often orders of magnitude—over brute-force search.

2. Background

This section provides background on bregman divergences and nearest neighbor search.

2.1. Bregman Divergences

First we briefly overview bregman divergences.

Definition 1 (Bregman, 1967)**.** *Let f be a strictly convex differentiable function.* [1] *The bregman diver-*

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

[1] Additional technical restrictions are typically put on

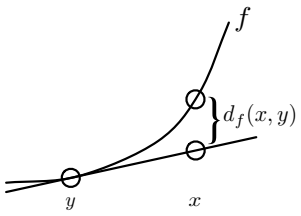


Figure 1. The bregman divergence between x and y.

gence based on f is

$$d_f(x,y) \equiv f(x) - f(y) - \langle \nabla f(y), x - y \rangle.$$

One can interpret the bregman divergence as the distance between a function and its first-order taylor expansion. In particular, $d_f(x,y)$ is the difference between $f(x)$ and the linear approximation of $f(x)$ centered at y; see figure 1. Since f is convex, $d_f(x,y)$ is always nonnegative.

Some standard bregman divergences and their base functions are listed in table 1.

A bregman divergence is typically used to assess similarity between two objects, much like a metric. But though metrics and bregman divergences are both used for similarity assessment, they do not share the same fundamental properties. Metrics satisfy three basic properties: non-negativity: $d(x,y) \geq 0$; symmetry: $d(x,y) = d(y,x)$; and, perhaps most importantly, the triangle inequality: $d(x,z) \leq d(x,y) + d(y,z)$. Bregman divergences are nonnegative, however they do not satisfy the triangle inequality (in general) and can be asymmetric.

Bregman divergences do satisfy a variety of geometric properties, a couple of which we will need later. The bregman divergence $d_f(x,y)$ is convex in x, but not necessarily in y. Define the bregman ball of radius R around μ as

$$B(\mu, R) \equiv \{x \ : \ d_f(x,\mu) \leq R\}.$$

Since $d_f(x,\mu)$ is convex in x, $B(\mu,R)$ is a convex set.

Another interesting property concerns means. For a set of points, the mean under a bregman divergence is well defined and, interestingly, is independent of the choice of divergence:

$$\mu_X \equiv \text{argmin}_\mu \sum_{x \in X} d_f(x,\mu) = \frac{1}{|X|} \sum_{x \in X} x.$$

This fact can be used to extend k-means to the family of bregman divergences (Banerjee et al., 2005).

f. In particular, f is assumed to be *Legendre*.

Table 1. Some standard bregman divergences.

	$f(x)$	$d_f(x,y)$
ℓ_2^2	$\frac{1}{2}\|x\|_2^2$	$\frac{1}{2}\|x-y\|_2^2$
KL	$\sum x_i \log x_i$	$\sum x_i \log \frac{x_i}{y_i}$
Mahalanobis	$\frac{1}{2} x^\top Q x$	$\frac{1}{2}(x-y)^\top Q (x-y)$
Itakura-Saito	$-\sum \log x_i$	$\sum \left(\frac{x_i}{y_i} - \log \frac{x_i}{y_i} - 1 \right)$

2.2. NN Search

Because of the tremendous practical and theoretical importance of nearest neighbor search in machine learning, computational geometry, databases, and elsewhere, many retrieval schemes have been developed to reduce the computational cost of finding NNs.

KD-trees (Friedman et al., 1977) are one of the earliest and most popular data structures for NN retrieval. The data structure and accompanying search algorithm provide a blueprint for a huge body of future work (including the present one). The tree defines a hierarchical space partition where each node defines an axis-aligned rectangle. The search algorithm is a simple branch and bound exploration of the tree. Though KD-trees are useful in many applications, their performance has been widely observed to degrade badly with the dimensionality of the database.

Metric ball trees (Omohundro, 1989; Uhlmann, 1991; Yianilos, 1993; Moore, 2000) extend the basic methodology behind KD-trees to metric spaces by using metric balls in place of rectangles. The search algorithm uses the triangle inequality to prune out nodes. They seem to scale with dimensionality better than KD-trees (Moore, 2000), though high-dimensional data remains very challenging. Some high-dimensional datasets are intrinsically low-dimensional; various retrieval schemes have been developed that scale with a notion of intrinsic dimensionality (Beygelzimer et al., 2006).

In many applications, an exact NN is not required; something nearby is good enough. This is especially true in machine learning applications, where there is typically a lot of noise and uncertainty. Thus many researchers have switched to the problem of *approximate* NN search. This relaxation led to some significant breakthroughs, perhaps the most important being locality sensitive hashing (Datar et al., 2004). Spill trees (Liu et al., 2004) are another data structure for approximate NN search and have exhibited very strong performance empirically.

The present paper appears to be the first to describe a general method for efficiently finding bregman NNs; however, some related problems have been examined. (Nielsen et al., 2007) explores the geometric properties of bregman voronoi diagrams. Voronoi diagrams are of course closely related to NN search, but do not lead to an efficient NN data structure beyond dimension 2. (Guha et al., 2007) contains results on *sketching* bregman (and other) divergences. Sketching is related to dimensionality reduction, which is the basis for many NN schemes.

We are aware of only one NN speedup scheme for KL-divergences (Spellman & Vemuri, 2005). The results in this paper are quite limited: experiments were conducted on only one dataset and the speedup is less than 3x. Moreover, there appears to be a significant technical flaw in the derivation of their data structure. In particular, they cite the pythagorean theorem as an *equality* for projection onto an arbitrary convex set, whereas it is actually an *inequality*.

3. Bregman Ball Trees

This section describes the bregman ball tree data structure. The data structure and search algorithms follow the same basic program used in KD-trees and metric trees; in place of rectangular cells or metric balls, the fundamental geometric object is a bregman ball.

A bbtree defines a hierarchical space partition based on bregman balls. The data structure is a binary tree where each node i is associated with a subset of the database $X_i \subset X$. Node i additionally defines a bregman ball $B(\mu_i, R_i)$ with center μ_i and radius R_i such that $X_i \subset B(\mu_i, R_i)$. Interior (non-leaf) nodes of tree have two child nodes l and r. The database points belonging to node i are split between child l and r; each point in X_i appears in exactly one of X_l or X_r.[2] Though X_l and X_r are disjoint, the balls $B(\mu_l, R_l)$ and $B(\mu_r, R_r)$ may overlap. The root node of the tree encapsulates the entire database. Each leaf covers a small fraction of the database; the set of all leaves cover the entirety.

3.1. Searching

This subsection describes how to retrieve a query's nearest neighbor with a bbtree. Throughout, $X = \{x_1, \ldots, x_n\}$ is the database, q is a query, and $d_f(\cdot, \cdot)$ is a (fixed) bregman divergence. The point we are searching for is the *left* NN

$$x_q \equiv \text{argmin}_{x \in X} d_f(x, q).$$

Finding the *right* NN ($\text{argmin}_{x \in X} d_f(q, x)$) is considered in section 5.

Branch and bound search locates x_q in the bbtree. First, the tree is descended; at each node, the search algorithm chooses the child for which $d_f(\mu, q)$ is smallest and *ignores* the sibling node (temporarily). Upon arriving at a leaf node i, the algorithm calculates $d_f(x, q)$ for all $x \in X_i$. The closest point is the candidate NN; call it x_c. Now the algorithm must traverse back up the tree and consider the previously ignored siblings. An ignored sibling j must be explored if

$$d_f(x_c, q) > \min_{x \in B(\mu_j, R_j)} d(x, q). \quad (1)$$

The algorithm computes the right side of (1); we come back that in a moment. If (1) holds, then node j and all of its children can be ignored since the NN cannot be found in that subtree. Otherwise, the subtree rooted at j must be explored. This algorithm is easily adjusted to return the k-nearest neighbors.

The algorithm hinges on the computation of (1)—the *bregman projection onto a bregman ball*. In the ℓ_2^2 (or arbitrary metric) case, the projection can be computed analytically with the triangle inequality. Since general bregman divergences do not satisfy this inequality, we need a different way to compute—or at least bound—the right side of (1). Computing this projection is the main technical contribution of this paper, so we discuss it separately in section 4.

3.2. Approximate Search

As we mentioned in section 2.2, many practical applications do not require an exact NN. This is especially true in machine learning applications, where there is typically a lot of noise and even the representation of points used is heuristic (*e.g.* selecting an appropriate kernel for an SVM often involves guesswork). This flexibility is fortunate, since exact NN retrieval methods rarely work well on high-dimensional data.

Following (Liu et al., 2004), a simple way to speed up the retrieval time of the bbtree is to simply stop after only a few leaves have been examined. This idea originates from the empirical observation that metric and KD-trees often locate a point very close to the NN quickly, then spend most of the execution time backtracking. We show empirically that the quality of the NN degrades gracefully as the number of leaves examined decreases. Even when the search procedure is stopped very early, it returns a solution that is among the nearest neighbors.

[2] The disjointedness of the two point sets is not essential.

3.3. Building

The performance of the search algorithm depends on how many nodes can be pruned; the more, the better. Intuitively, the balls of two siblings should be well-separated and compact. If the balls are well-separated, a query is likely to be much closer to one than the other. If the balls are compact, then the distance from a query to a ball will be a good approximation to the distance from a query to the nearest point within the ball. Thus at each level, we'd like to divide the points into two well-separated sets, each of which is compact. A natural way to do this is to use k-means, which has already been extended to bregman divergences (Banerjee et al., 2005).

The build algorithm proceeds from top down. Starting at the top, the algorithm runs k-means to partition the points into two clusters. This process is repeated recursively. The total build time is $O(n \log n)$. Clustering from the bottom-up might yield better results, but the $O(n^2 \log n)$ build time is impractical for large datasets.

4. Computing the Bound

Recall that the search procedure needs to determine if the bound

$$d_f(x_c, q) > \min_{x \in B(\mu, R)} d_f(x, q) \tag{2}$$

holds, where x_c is the current candidate NN. We first show that the right side can be computed to accuracy ϵ in only $O(\log \frac{1}{\epsilon})$ steps with a simple bisection search. Since we only actually need upper and lower bounds on the quantity, we then present a procedure that augments the bisection search with primal and dual bounds so that it can stop early.

The right of (2) is a convex program:

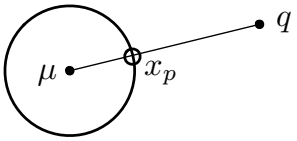
$$\begin{aligned} \min_x \quad & d_f(x, q) \\ \text{subject to:} \quad & d_f(x, \mu) \leq R. \end{aligned} \tag{P}$$

The search algorithm will need to solve (P) many times in the course of locating q's NN, so we need to be able to compute a solution very quickly.

Before considering the general case, let us pause to examine the ℓ_2^2 case. In this case, we can compute the projection x_p analytically:

$$x_p = \theta\mu + (1 - \theta)q,$$

where $\theta = \frac{\sqrt{2R}}{\|q - \mu\|}$.



What properties of this projection might extend to all of bregman divergences?

1. First, x_p lies on the line between q and μ; this drastically reduces the search space from a D-dimensional convex set to a one-dimensional line.
2. Second, x_p lies on the boundary of $B(\mu, R)$—*i.e* $d_f(x_p, \mu) = R$. Combined with property 1, this fact completely determines x_p: it is the point where the line between μ and q intersects the shell of $B(\mu, R)$.
3. Finally, since the ℓ_2^2 ball is spherically symmetric, we can compute this intersection analytically.

We prove that the first property is a special case of a fact that holds for all bregman divergences. Additionally, the second property generalizes to bregman divergences without change. The final property does not go through, so we will not be able to find a solution to (P) analytically.

Throughout, we use $q' \equiv \nabla f(q)$, $\mu' \equiv \nabla f(\mu)$, *etc.* to simplify notation. x_p denotes the optimal solution to (P).

Claim 2. *x_p' lies on the line between q' and μ'.*

Proof. The lagrange dual function of (P) is

$$\inf_x d_f(x, q) + \lambda(d_f(x, \mu) - R), \tag{3}$$

where $\lambda \geq 0$. Differentiating (3) with respect to x and setting it equal to 0, we get

$$\nabla f(x_p) - \nabla f(q) + \lambda \nabla f(x_p) - \lambda \nabla f(\mu) = 0.$$

We use the change of variable $\theta \equiv \frac{\lambda}{1+\lambda}$ and rearrange to arrive at

$$\nabla f(x_p) = \theta\mu' + (1 - \theta)q',$$

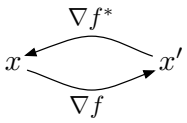
where $\theta \in [0, 1)$. □

Thus we see that property 1 of the ℓ_2^2 projection is a special case of a relationship between the gradients; it follows from claim 2 because $\nabla f(x) = x$ for the ℓ_2^2 divergence.

Since f is strictly convex, the gradient mapping is one-to-one. Moreover, the inverse mapping is given by the gradient of the *convex conjugate*, defined as

$$f^*(y) \equiv \sup_x \{\langle x, y\rangle - f(x)\}. \tag{4}$$

Symbolically:



Thus to solve (P), we can look for the optimal x' along $\theta\mu' + (1-\theta)q'$, and then apply ∇f^* to recover x_p.[3] To keep notation simple, we define

$$x'_\theta \equiv \theta\mu' + (1-\theta)q' \quad \text{and} \tag{5}$$
$$x_\theta \equiv \nabla f^*(x'_\theta). \tag{6}$$

Now onto the second property.

Claim 3. *$d_f(x_p, \mu) = R$—i.e. the projection lies on the boundary of $B(\mu, R)$.*

The claim follows from complementary slackness applied to (3). Claims 2 and 3 imply that finding the projection of q onto $B(\mu, R)$ is equivalent to

$$\begin{aligned} \text{find} \quad & \theta \\ \text{subject to:} \quad & d_f(x_\theta, \mu) = R \\ & \theta \in (0, 1] \\ & x_\theta = \nabla f^*(\theta\mu' + (1-\theta)q'). \end{aligned}$$

Fortunately, solving this program is simple.

Claim 4. *$d_f(x_\theta, \mu)$ is monotonic in θ.*

This claim follows from the convexity of f^*. Since $d_f(x_\theta, \mu)$ is monotonic, we can efficiently search for θ_p satisfying $d_f(x_{\theta_p}, \mu) = R$ using bisection search on θ. We summarize the result in the following theorem.

Theorem 5. *Suppose $\|\nabla^2 f^*\|_2$ is bounded around x'_p. Then a point x satisfying*

$$|d_f(x, q) - d_f(x_p, q)| \le \epsilon + O(\epsilon^2)$$

can be found in $O(\log 1/\epsilon)$ iterations. Each iteration requires one divergence evaluation and one gradient evaluation.

4.1. Stopping Early

Recall that the point of all this analysis is to evaluate whether

$$d_f(x_c, q) > \min_{x \in B(\mu,R)} d_f(x, q), \tag{7}$$

where x_c is the current candidate NN. If (7) holds, the node in question must be searched; otherwise it can be pruned. We can evaluate the right side of (7) exactly using the bisection method described previously, but an exact solution is not needed. Suppose we have bounds a and A satisfying

$$A \ge \min_{x \in B(\mu,R)} d_f(x, q) \ge a.$$

If $d_f(x_c, q) < a$, the node can be pruned; if $d_f(x_c, q) > A$, the node must be explored. We now describe upper and lower bounds that are computed at each step of the bisection search; the search proceeds until one of the two stopping conditions is met.

A lower bound is given by weak duality. The lagrange dual function is

$$\mathcal{L}(\theta) \equiv d_f(x_\theta, q) + \frac{\theta}{1-\theta}\Big(d_f(x_\theta, \mu) - R\Big). \tag{8}$$

By weak duality, for any $\theta \in [0, 1)$,

$$\mathcal{L}(\theta) \le \min_{x \in B(\mu,R)} d_f(x, q). \tag{9}$$

For the upper bound, we use the primal. At any θ satisfying $d_f(x_\theta, \mu) \le R$, we have

$$d_f(x_\theta, q) \ge \min_{x \in B(\mu,R)} d_f(x, q). \tag{10}$$

Let us now put all of the pieces together. We wish to evaluate whether (7) holds. The algorithm performs bisection search on θ, attempting to locate the θ satisfying $d_f(x_\theta, \mu) = R$. At step i the algorithm evaluates θ_i on two functions. First, it checks the lower bound bound given by the dual function $\mathcal{L}(\theta_i)$ defined in (8). If $\mathcal{L}(\theta_i) > d_f(x_c, q)$, then the node can be pruned. Otherwise, if $x_{\theta_i} \in B(\mu, R)$, we can update the upper bound. If $d_f(x_{\theta_i}, q) < d_f(x_c, q)$, then the node must be searched. Otherwise, neither bound holds, so the bisection search continues. See Algorithm 1 for pseudocode.

5. Left and Right NN

Since a bregman divergence can be asymmetric, it defines two NN problems:

- (lNN) return $\text{argmin}_{x \in X} d_f(x, q)$ and
- (rNN) return $\text{argmin}_{x \in X} d_f(q, x)$.

The bbtree data structure finds the left NN . We show that it can also be used to find the right NN.

[3] All of the base functions in table 1 have closed form conjugates.

Algorithm 1 CanPrune

Input: $\theta_l, \theta_r \in (0,1]$, $q, x_c, \mu \in \mathbb{R}^D$, $R \in \mathbb{R}$.
Set $\theta = \frac{\theta_l + \theta_r}{2}$.
Set $x_\theta = \nabla f^*(\theta\mu' + (1-\theta)q')$
if $\mathcal{L}(\theta) > d_f(x_c, q)$ **then**
  return YES
else if $x_\theta \in B(\mu, R)$ and $d_f(x_\theta, q) < d_f(x_c, q)$ **then**
  return NO
else if $d_f(x_\theta, \mu) > R$ **then**
  return CanPrune$(\theta_l, \theta, q, x_c, \mu)$
else if $d_f(x_\theta, \mu) < R$ **then**
  return CanPrune$(\theta, \theta_r, q, x_c, \mu)$
end if

Recall that the convex conjugate of f is defined as $f^*(y) \equiv \sup_x\{\langle x, y\rangle - f(x)\}$. The supremum is realized at a point x satisfying $\nabla f(x) = y$; thus

$$f^*(y') = \langle y, y'\rangle - f(y).$$

We use this identity to rewrite $d_f(\cdot,\cdot)$:

$$\begin{aligned} d_f(x,y) &= f(x) - f(y) - \langle y', x - y\rangle \\ &= f(x) + f^*(y') - \langle y', x\rangle \\ &= d_{f^*}(y', x'). \end{aligned}$$

This relationship provides a simple prescription for adapting the bbtree to the rNN problem: build a bbtree for the divergence d_{f^*} and the database $X' \equiv \{\nabla f(x_1), \ldots, \nabla f(x_n)\}$. On query q, $q' \equiv \nabla f(q)$ is computed and the bbtree finds $x' \in X'$ minimizing $d_{f^*}(x', q')$. The point x whose gradient is x' is then the rNN to q.

6. Experiments

We examine the performance benefit of using bbtrees for approximate and exact NN search. All experiments were conducted with a simple C implementation that is available from the author's website.

The results are for the KL-divergence. We chose to evaluate the bbtree for the KL-divergence because it is used widely in machine learning, text mining, and computer vision; moreover, very little is known about efficient NN retrieval for it. In contrast, there has been a tremendous amount of work for speeding up the ℓ_2^2 and Mahalanobis divergences—they both may be handled by standard metric trees and many other methods. Other bregman divergences appear much less often in applications. Still, examining the practical performance of bbtrees for these other bregman divergences is an interesting direction for future work.

We ran experiments on several challenging datasets.

- **rcv-D**. We used latent dirichlet allocation (LDA) (Blei et al., 2003) to generate topic histograms for 500k documents in the rcv1 corpus (Lewis et al., 2004). These histograms were generated by building a LDA model on a training set and then performing inference on 500k documents to generate their posterior dirichlet parameters. Suitably scaled, these parameters give a representation of the documents in the topic simplex (Blei et al., 2003). We generated data using this process for $D = 8, 16, \ldots, 256$ topics.
- **Corel histograms**. This dataset contains 60k color histograms generated from the Corel image dataset. Each histogram is 64-dimensional.
- **Semantic space**. This dataset is a 371-dimensional representation of 5000 images from the Corel Stock photo collection. Each image is represented as a distribution over 371 description keywords (Rasiwasia et al., 2007).
- **SIFT signatures**. This dataset contains 1111-dimensional representations of 10k images from the PASCAL 2007 dataset (Everingham et al., 2007). Each point is a histogram of quantized SIFT features as suggested in (Nowak et al., 2006).

Notice that most of these datasets are fairly high-dimensional.

We are mostly interested in approximate NN retrieval, since that is likely sufficient for machine learning applications. If the bbtree is stopped early, it is not guaranteed to return an exact NN, so we need a way to evaluate the quality of the point it returns. One natural evaluation metric is this: How many points from the database are closer to the query than the returned point? Call this value NC for "number closer". If NC is small compared to the size of the database, say 10 versus 100k, then it will likely share many properties with the true NN (*e.g.* class label).[4]

The results are shown in figure 2. These are strong results; it is shown that the bbtree is often orders of magnitude faster than brute-force search without a substantial degradation of quality. More analysis appears in the caption.

[4] A different evaluation criteria is the approximation ratio ϵ satisfying $d_f(x,q) \leq (1+\epsilon)d_f(x_q, q)$, where x_q is q's true NN. We did not use this measure because it is difficult to interpret. For example, suppose we find $\epsilon = .3$ approximate NNs from two different databases A and B. It could easily be the case that *all* points in A are 1.3-approximate NNs, whereas only the exact NN in database B is 1.3-approximate.

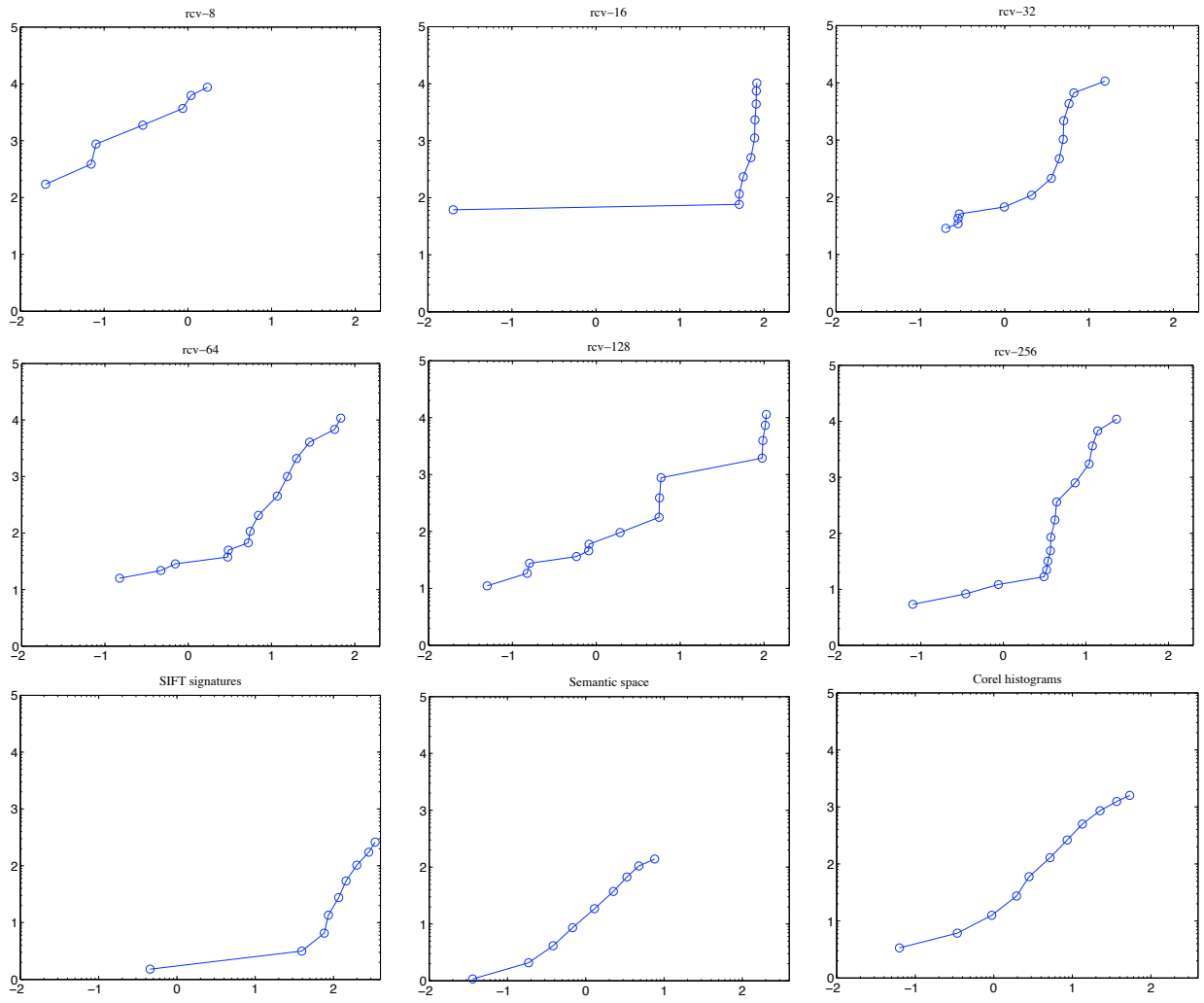


Figure 2. Log-log plots (base 10): y-axis is the exponent of the speedup over brute force search, x-axis is the exponent of the number of database points closer to the query than the reported NN. The y-axis ranges from 10^0 (no speedup) to 10^5. The x-axis ranges from 10^{-2} to 10^2. All results are averages over queries not in the database.

Consider the plot for rcv-128 (center). At $x = 10^0$, the bbtree is returning one of the two nearest neighbors (on average) out of 500k points at a 100x speedup over brute force search. At $x = 10^1$, the bbtree is returning one of the eleven nearest neighbors (again, out of 500k points) and yields three orders of magnitude speedup over brute force search.

The best results are achieved on the rcv-D datasets and the Corel histogram dataset. The improvements are less pronounced for the SIFT signature and Semantic space data, which may be a result of both the high dimensionality and small size of these two datasets. Even so, we are getting useful speedups on the semantic space dataset (10-100x speedup with small error). For the SIFT signatures, we are getting a 10x speedup while receiving NNs in the top 1%.

Table 2. Exact search

dataset	dimensionality	speedup
rcv-8	8	64.5
rcv-16	16	36.7
rcv-32	32	21.9
rcv-64	64	12.0
corel histograms	64	2.4
rcv-128	128	5.3
rcv-256	256	3.3
semantic space	371	1.0
SIFT signatures	1111	0.9

Finally, we consider exact NN retrieval. It is well known that finding a (guaranteed) exact NN in moderate to high-dimensional databases is very challenging. In particular, metric trees, KD-trees, and relatives typically afford a reasonable speedup in moderate dimensions, but the speedup diminishes with increasing dimensionality (Moore, 2000; Liu et al., 2004). When used for exact search, the bbtree reflects this basic pattern. Table 2 shows the results. The bbtree provides a substantial speedup on the moderate-dimensional databases (up through $D = 256$), but no speedup on the two databases of highest dimensionality.

7. Conclusion

In this paper, we introduced bregman ball trees and demonstrated their efficacy in NN search. The experiments demonstrated that bbtrees can speed up approximate NN retrieval for the KL-divergence by orders of magnitude over brute force search. There are many possible directions for future research. On the practical side, which ideas behind the many variants of metric trees might be useful for bbtrees? On the theoretical side, what is a good notion of intrinsic dimensionality for bregman divergences and can a practical data structure be designed around it?

Acknowledgements

Thanks to Serge Belongie, Sanjoy Dasgupta, Charles Elkan, Carolina Galleguillos, Daniel Hsu, Nikhil Rasiwasia, and Lawrence Saul. Support was provided by the NSF under grants IIS-0347646 and IIS-0713540.

References

Banerjee, A., Merugu, S., Dhillon, I. S., & Ghosh, J. (2005). Clustering with bregman divergences. *JMLR*.

Beygelzimer, A., Kakade, S., & Langford, J. (2006). Cover trees for nearest neighbor. *ICML*.

Blei, D., Ng, A., & Jordan, M. (2003). Latent dirichlet allocation. *JMLR*.

Bregman, L. (1967). The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, *7*, 200–217.

Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. *SCG 2004*.

Everingham, M., Gool, L. V., Williams, C. K., Winn, J., & Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 Results.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, *3(3)*, 209–226.

Gray, R. M., Buzo, A., Gray, A. H., & Matsuyama, Y. (1980). Distortion measures for speech processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*.

Guha, S., Indyk, P., & McGregor, A. (2007). Sketching information divergences. *COLT*.

Lewis, D. D., Yang, Y., Rose, T., & Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *JMLR*.

Liu, T., Moore, A. W., Gray, A., & Yang, K. (2004). An investigation of practical approximate neighbor algorithms. *NIPS*.

Moore, A. W. (2000). Using the triangle inequality to survive high-dimensional data. *UAI*.

Nielsen, F., Boissonnat, J.-D., & Nock, R. (2007). On bregman voronoi diagrams. *SODA* (pp. 746–755).

Nowak, E., Jurie, F., & Triggs, B. (2006). Sampling strategies for bag-of-features image classification. *ECCV*.

Omohundro, S. (1989). *Five balltree construction algorithms* (Technical Report). ICSI.

Pereira, F., Tishby, N., & Lee, L. (1993). Distributional clustering of English words. *31st Annual Meeting of the ACL* (pp. 183–190).

Puzicha, J., Buhmann, J., Rubner, Y., & Tomasi, C. (1999). Empirical evaluation of dissimilarity measures for color and texture. *ICCV*.

Rasiwasia, N., Moreno, P., & Vasconcelos, N. (2007). Bridging the gap: query by semantic example. *IEEE Transactions on Multimedia*.

Spellman, E., & Vemuri, B. (2005). Efficient shape indexing using an information theoretic representation. *International Conference on Image and Video Retrieval*.

Uhlmann, J. K. (1991). Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, *40*, 175–179.

Weinberger, K., Blitzer, J., & Saul, L. (2006). Distance metric learning for large margin nearest neighbor classification. *NIPS*.

Yianilos, P. N. (1993). Data structures and algorithms for nearest neighbor search in general metric spaces. *SODA*.