Tailored Bregman Ball Trees for Effective Nearest Neighbors[*]

Frank Nielsen[†] Paolo Piro, Michel Barlaud[‡]

Abstract

Nearest Neighbor (NN) search is a crucial tool that remains critical in many challenging applications of computational geometry (e.g., surface reconstruction, clustering) and computer vision (e.g., image and information retrieval, classification, data mining). We present an effective Bregman ball tree [5] (Bb-tree) construction algorithm that *adapts* locally its internal node degrees to the inner geometric characteristics of the data-sets. Since symmetric measures are usually preferred for applications in content-based information retrieval, we furthermore extend the Bb-tree to the case of symmetrized Bregman divergences. Exact and approximate NN search experiments using high-dimensional real-world data-sets illustrate that our method improves significantly over the state of the art [5], sometimes by an *order* of magnitude.

1 Introduction and prior work

Finding nearest neighbors (NNs) is a very common task occurring in many applications ranging from computer vision to machine learning and data mining. Let $\mathcal{S} = \{p_1, ..., p_n\}$ be a set of n d-dimensional points (with d typically ranging up to a few thousand dimensions). Given a *query* point q, the nearest neighbor $\mathrm{NN}(q)$ is defined to be the "closest" point of $\mathcal{S}$ with respect to a dissimilarity measure D: $\mathrm{NN}(q) = \arg\min_i D(q, p_i)$. Instead of considering the closest neighbor, queries can be enlarged to report the first k "closest" points (useful for k-NN classification rule in machine learning). It is usually enough to get a *good* neighbor [6] by relaxing the exact search to get *near* and *fast* neighbors. Besides the theoretical puzzling questions related to the dreaded curse of dimensionality [6], practitioners make every endeavor to speed up applications by designing tailored schemes for improving over the naïve linear-time $O(dn)$ brute-force method. Among the flourishing literature of nearest neighbor techniques, we distinguish two main sets of methods: (1) those relying on tree-like space partitions with branch-and-bound queries, such as kD-trees, metric ball and vantage point trees [5, 8], and (2) those based on mapping techniques [3] (e.g., locality-sensitive hashing, random projections). The former tree-based methods improve over the brute force algorithm by *pruning* sub-trees whenever traversing the trees with queries (exact NN) or stopping after visiting a given "budget" of leaves (approximate NN). The latter methods concentrate on reducing dimensions while preserving as much as possible distances and controlling geometrically collisions of hash functions (only work for approximate NN). Since the Euclidean distance is often inappropriate for meaningfully measuring the proximity of feature points arising from, say, image applications, a wide range of distortion measures has been proposed and NN data-structures have been first extended to arbitrary metrics (e.g., vantage point trees [8]). However, these NN search methods rely fundamentally on the *triangle inequality* property, which is not satisfied by information-theoretic statistical distances such as the Kullback-Leibler divergence. Cayton [5] recently proposed the analogous of metric ball trees for the broad class of Bregman divergences. Bregman divergences D_F on vector set $\mathcal{X} \subset \mathbb{R}^d$ are defined for a strictly *convex* and *differentiable* generator $F(x) : \mathcal{X} \subset \mathbb{R}^d \mapsto \mathbb{R}$ as $D_F(p||q) = F(p) - F(q) - (p-q)^T \nabla F(q)$, where ∇F denotes the gradient of F. Those divergences (parameterized by a generator F) include all quadratic distances (also known as Mahalanobis squared distances, which are the only symmetric Bregman divergences and are obtained for $F(x) = \Sigma^{-1}x$, where $\Sigma \succ 0$ is the positive-definite variance-covariance matrix) and the asymmetric KL divergence ($F(x) = \sum_{j=1}^d x_j \log x_j$, the negative Shannon entropy), for which $D_F(p||q) \neq D_F(q||p)$. Many fundamental algorithms (e.g., k-means [4], PCA) and data-structures (e.g., Voronoi diagrams [10]) have been generalized to that class of divergences, thus offering *meta-algorithms* that can work for *any* Bregman divergence. For example, Banerjee et al. [4] showed that the celebrated Lloyd k-means algorithm extends to (and only to) the class of Bregman divergences unifying various algorithms. Since Bregman divergences are typically non-metric asymmetric distortion measures, we consider both left/right-sided and symmetrized NN searches defined as follows: $\mathrm{NN}^r_F(q) = \arg\min_i D_F(p_i||q)$ (right-sided), $\mathrm{NN}^l_F(q) = \arg\min_i D_F(q||p_i)$ (left-sided) and $\mathrm{NN}_F(q) = \arg\min_i (D_F(p_i||q) + D_F(q||p_i))/2$ (sym-

[*] Financially supported by ANR-07-BLAN-0328-01 GAIA (Computational Information Geometry and Applications) and DIGITEO GAS 2008-16D (Geometric Algorithms & Statistics)

[†] École Polytechnique, LIX, Palaiseau, France, nielsen@lix.polytechnique.fr

[‡] CNRS I3S, Sophia-Antipolis, France, {piro,barlaud}@i3s.unice.fr

metrized).

2 Bregman ball trees (Bb-trees)

Without loss of generality, we consider only *right-sided* NN queries, as left-sided NN queries can be handled similarly by considering the dual divergence $D_{F^*}(\nabla F(q)||\nabla F(p)) = D_F(p||q)$ arising from the Legendre-Fenchel conjugate F^* of F (with ∇F^* the functional inverse of ∇F^{-1}). See [10, 5] for details. Bregman generators come by pairs (F, F^*), e.g. the dual Legendre conjugates $F(x) = x \log x$ and $F^*(y) = \exp y$ (with $F'(x) = (F^{*\prime}(y))^{-1}$), see [10].

2.1 Outline of Bregman ball trees (Bb-trees) [5]

Similar to metric ball trees, a Bb-tree is built in a *top-down* fashion by applying recursively a partitioning scheme. First, the root is created to handle the source data-set $\mathcal{S}$. A 2-means (k-means [4] with $k = 2$) is computed, splitting $\mathcal{S}$ according to the two centroid points, say c_l and c_r, with respect to D_F. These two centroids defines a partition $\mathcal{S} = \mathcal{S}_l \cup \mathcal{S}_r$ that can be covered geometrically with corresponding Bregman balls $B(c_l, R_l)$ and $B(c_r, R_r)$ (possibly overlapping). This hierarchical decomposition of $\mathcal{S}$ is carried out recursively on $\mathcal{S}_l$ and $\mathcal{S}_r$ until a stopping criterion is eventually met. Two such typical termination criteria are: (1) a predefined maximum number of points l_0 (stored at leaves), or (2) a prescribed maximum radius (r_0, comes in handy for approximate NN). Note that the source points are stored *only* at leaves of the Bb-tree, and the sub-sets $\mathcal{S}_l$ and $\mathcal{S}_r$ may be theoretically unbalanced. Internal nodes store only two left/right Bregman balls covering the point sets stored at their left/right sub-trees.

In both *exact* and *approximate* NN queries, we perform a *branch-and-bound* search to answer queries. For a given query q, the tree is traversed in *depth-first-search* order from the root to the leaves. At an internal node, we choose to branch first on the sub-tree whose corresponding ball is "closer" to the query q (the sibling is temporarily ignored). Once a leaf node is reached, the closest point to q among the points stored at the leaf is computed using the brute-force method. This first visited leaf yields the very first NN point candidate p', thus giving an upper bound $D_F(p'||q)$ to the NN distance. In exact search, the tree traversal goes on through all formerly ignored subtrees. To decide whether a subtree must be explored or not, we check whether $D_F(p'||q) > \min_{x \in B(c,R)} D_F(x||q)$, where p' is the current NN candidate. This test is performed by projecting q onto the Bregman ball $B(c, R)$. The projection of a point q onto a Bregman ball B is the unique minimizer $q_B = \arg\min_{x \in B} D_F(x||q)$ [10]. Instead of projecting exactly the query point onto the ball, we rather make use of a *Bregman annulus* $B(c, R, R') = \{x \mid R \leq D_F(x||c) \leq R'\}$ that encloses the exact projection by construction: $q_B \in B(c, R, R')$. If $D_F(p'||q) < R$ then the node can be pruned; If $D_F(p'||q) > R'$ then the node must be explored. These lower/upper bounds are computed during the geodesic bisection search of the projection [5] (see Section 2.4). Overall, observe that the less the number of pairwise intersecting balls stored at nodes, the better the Bb-tree performance. Although exact NN retrieval on Bb-trees can often be achieved with much smaller computational cost than the brute-force search, the practical interest of Bb-trees is to get **significant** speed-up search when performing **approximate** NN queries. The approximate search allows one for large speed-up as it does not perform exhaustive branch-and-bound search. A common criterion to perform approximate search is to stop the branch-and-bound algorithm after exploring a *prescribed* number of leaves [5].

2.2 Speeding up construction time: `Bb-tree++`

In order to preprocess datasets efficiently by means of Bregman ball trees, instead of running the full regular Bregman k-means algorithm [4], we just perform a careful light initialization of the two cluster centers (seeds). Initialization turns out to be the crucial stage of k-means. That is, k-means locally optimizes the potential $P(\mathcal{C}) = \sum_{p \in \mathcal{S}} \min_{c \in \mathcal{C}} D_F(p||c)$, where $\mathcal{C}$ denotes the set of k centers. Each round assignment/cluster adjustment decreases this potential function so that monotonous convergence is guaranteed. It is striking to know that the worst-case running-time of k-means is theoretically exponential with the dimension although a recent smooth analysis yields polynomial amortized time [9]. A bad initialization may trap k-means into a local optimum. Initialization is thus all the more important, as k-means is called at each internal node of the Bb-tree to partition the data-sets into two sub-sets. Therefore, the idea is to replace the k-means local iterative algorithm by a well-chosen initialization that provides a guaranteed upper-bound on the optimal partition. This quantum leap for k-means was discovered by Arthur and Vassilvitskii [2] and later extended to Bregman divergences by Nock et al. [12]. We describe the initialization procedure of Bregman 2-`means++` for a set $\mathcal{S}$. First, draw uniformly at random the seed c_l (cluster "center") among the points of the data-set. Then compute the Bregman divergence of this point to all points of $\mathcal{S}$, and draw the second seed c_r according to the divergence distribution: $\pi_i = \frac{D_F(p_i||c_l)}{\sum_{p_j \in \mathcal{S}} D_F(p_j||c_l)}$. Thus c_r can never be c_l, since $D_F(c_l||c_l) = 0$ (the probability of drawing c_l is zero). This careful initialization yields an extremely fast tree construction, which statistically provides nice splitting and tends to balance sub-sets stored at leaves. Arthur and Vassilvitskii [2] proved that this initialization yields a k-means score $P(\mathcal{C})$ at most $8(2+\log k)$ of the optimal value. Similar bounds in $O(\log k)$ for generic Bregman divergences were later reported by Nock et al. [12]. Note that the two left/right Bregman balls stored at internal nodes tend to minimize the Bregman information [4] (i.e., variance for the square potential $F(x) = x^2$, mutual information for the negative Shannon entropy $F(x) = x \log x$, etc). Next, we improve the partition at each internal node by *learning* its degree (number of siblings) from the local data-sets.

2.3 Learning the tree branching factor

Answering queries may range from optimal logarithmic time (i.e., the shortest path to a leaf) up to linear time for a complete tree traversal. Therefore, it is important to partition the source data-sets into as many as possible non-overlapping Bregman balls. However, consider a data-set consisting of three separated Gaussian point samples. Forcing this set to be split into two will likely create overlapping balls. Thus we better learn the number of sub-sets when partitioning the data so that the induced Bregman balls better fit the intrinsic geometric characteristics of the set.[1] We adapt the branching factor bf_i (up to a maximum branching factor BF) of each internal node of the Bb-tree to the underlying distribution of the points by using the G-means strategy [7]. The underlying idea is to assume Gaussian distribution of each group of points (hence the name G-means).[2] Our use of G-means [7] algorithm starts by setting $k = 2$ and then test for Gaussian distribution of the points using the Anderson-Darling statistical test. Given a confidence level α, if the Anderson-Darling normality test returns `true`, we keep the center, otherwise we split it into two. Between two rounds, we simply run Bregman k-means++ initialization on the data-set and get all the new centers that hopefully refine the partitioning. We enforce a maximum degree to each internal node in order to strike a balance between the average tree depth and the overall ball tree shape. The initialization to k clusters follows the same principle: Namely, we draw the l-th seed from the data-set uniformly according to the distribution $\pi_i = \frac{D_F(p_i||\mathcal{C}_{l-1})}{\sum_{p_j \in \mathcal{S}} D_F(p_j||\mathcal{C}_{l-1})}$, where $\mathcal{C}_{l-1}$ denotes the formerly chosen $(l-1)$-th seed and $D_F(p||\mathcal{S}) = \min_{x \in \mathcal{S}} D_F(p||x)$.

When visiting the Bb-tree for answering nearest neighbor queries, we use a *priority queue.* To relax the exact NN to approximate NN queries, the criterion of stopping the search once a few leaves have been visited was proposed and successfully demonstrated by Cayton [5]. However, there was no guarantee to get a good approximation to the exact NN because leaves containing sub-sets that are close to the exact NN are not necessarily close in the tree. We improve this point by a careful **non-recursive** implementation of the tree traversal, which allows us to order the nodes to be explored by their divergence to the query point (i.e., the divergence of the query to the ball centers). Using a priority queue guarantees that nearest nodes are always explored first when traversing back the tree, by jumping appropriately to the most likely not yet visited sub-tree.

2.4 Bregman projection onto balls

The geodesic Γ_{pq} linking p to q is defined as $\Gamma_{pq} = \{\mathrm{LERP}(\lambda, p, q) \mid \lambda \in \mathbb{R}\}$, with $\mathrm{LERP}(\lambda, p, q) = \nabla F^{-1}((1-\lambda)\nabla F(p) + \lambda \nabla F(q))$; see [10, 5, 11]. To find the Bregman projection $q_B = \arg\min_{x \in B(c,R)} D_F(x||c)$ of a query point q onto a Bregman ball $B(c, R)$, we first check whether q is outside the ball or not: $D_F(q||c) > R$. If not, $D_F(q||c) \leq R$ and the projection is simply the point itself: $q_B = q$. Now consider the geodesic segment Γ_{cq}, the projection point q_B belongs necessarily to the geodesic Γ_{cq}: $q_B = \mathrm{LERP}(\lambda_q, c, q)$ for some $\lambda_q \in [0, 1]$. The value λ_q is approximated by a bisection search as follows. Initially, let $\lambda_m^{(0)} = 0$ and $\lambda_M^{(0)} = 1$ and consider a midpoint $m = \mathrm{LERP}(\lambda^{(0)}, c, q)$ obtained for $\lambda^{(0)} = \frac{\lambda_m^{(0)} + \lambda_M^{(0)}}{2}$. If $D_F(m||c) > R$ then recurse on $[\lambda_m^{(1)}, \lambda_M^{(1)}] = [\lambda_m^{(0)}, \lambda^{(0)}]$, otherwise recurse on $[\lambda_m^{(1)}, \lambda_M^{(1)}] = [\lambda^{(0)}, \lambda_M^{(0)}]$ and so on until the range size $\lambda_M^{(i)} - \lambda_m^{(i)}$ goes below a threshold ϵ (typically $\epsilon \in [10^{-10}, 10^{-5}]$). This yields a fine approximation of $q_B \sim \mathrm{LERP}(\lambda^{(i)}, c, q)$ by splitting the "λ" intervals a dozen of times. (Recall that the algorithm keeps a Bregman annulus, and refine the inner/outer radii to decide whether to prune or explore a sub-tree.)

2.5 Handling symmetrized Bregman divergences

Many content-based information retrieval systems (CBIRs) need *symmetrical* distortions measures. Except for generalized quadratic distances (Mahalanobis), the symmetrized Bregman divergence is technically not a Bregman divergence [10]. The symmetric KL divergence (SKL) goes by the name of Jensen-Shannon $\mathrm{JS}(p; q) = \frac{1}{2}\mathrm{KL}(p||\frac{p+q}{2}) + \frac{1}{2}\mathrm{KL}(q||\frac{p+q}{2})$. It turns out that these symmetrized Bregman divergences [11] are all generalizations of the Jensen remainder obtained for convex generators F: $\mathrm{JS}_F(p; q) = \frac{1}{2}(F(p) + F(q)) - F(\frac{p+q}{2})$. (Besides the Jensen-Shannon divergence for $F(x) = x \log x$, these so-called Burbea-Rao divergences also includes the important COSH distance used in speech/sound processing for the Burg entropy $F(x) = -\log x$.)

[1] Refer to Figure 1 of [7] for examples.

[2] This is not restrictive as any smooth density function may be arbitrarily well approximated using a Gaussian mixture model.

Bb-tree construction ($bs = 50$)					
method	*iter*	*depth*	$depth_{avg}$	*Leav.*	*speed-up*
2-m.	10	53	28.57	594	1
2-m.++	10	58.33	31.18	647	1.03
2-m.++	**0**	**20**	**10.76**	**362**	**19.71**

Table 1: Construction time of Bregman ball trees.

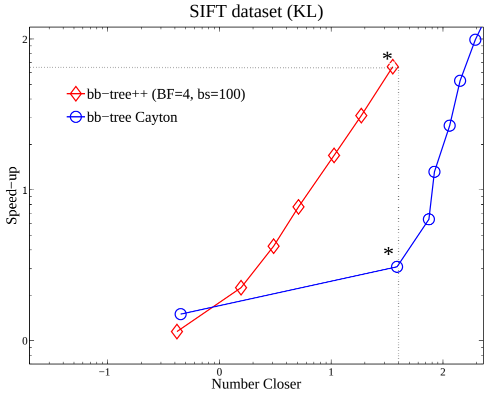

Figure 1: BB-tree versus BB-tree++ (log-log plot scale). The axis denotes the error rate expressed as the number of closer points to the approximated NN.

The symmetrized Bregman centroid can be explicitly computed following the geodesic-walk algorithm of [11].

3 Experiments

Table 1 compares different partitioning methods when building a Bb-tree on the SIFT dataset [13] (10,000 visual feature points extracted from images, encoded in dimension 1111). Bregman 2-`means++` with $iter = 0$ means that only the initialization step is carried out. The speed-up is by comparison with the most computationally expensive k-means method.

We used the maximum number of explored leaves as a parameter for stopping the branch-and-bound search. In each experiment of approximate search, we fixed a value of this parameter (from *near-exact* search to visiting only a *single* leaf), then we evaluated error rate and computational cost. Namely, the error rate is expressed as the number of closer points to the approximated NN (a quantity called Number Closer), while the speed-up is given by the ratio between the number of divergence computations in Bb-tree over the brute-force search. Figure 1 displays a comparison of Cayton's Bb-tree with our Bb-tree++ in approximate search on SIFT dataset (log-log plot).

Figure 2 presents the results of approximate search queries on SIFT dataset wrt. SKL divergence (symmetrized Bregman). The marked point shows a speed-up of 60.3 ($10^{1.78}$) when the Number Closer equals 14 ($10^{1.15}$), i.e. 0.14% of database points (log-log plot).

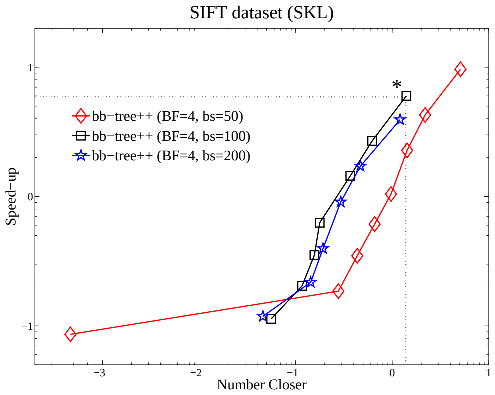

Figure 2: NN queries with respect to Jensen-Shannon divergence (symmetric Kullback-Leibler, SKL).

Acknowledgments

We thank L. Cayton for sharing with us his code and SIFT high-dimensional datasets.

References

[1] S.-I. Amari and N. Nagaoka. *Methods of Information Geometry*. Oxford University Press, 2000.

[2] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, pages 1027–1035, 2007.

[3] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, pages 327–336, 2008.

[4] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research (JMLR)*, 6:1705–1749, 2005.

[5] L. Cayton. Fast nearest neighbor retrieval for Bregman divergences. In *ICML*, pp. 112–119, 2008.

[6] B. Chazelle. Technical perspective: finding a good neighbor, near and fast. *Commun. ACM*, 51(1):115, 2008.

[7] G. Hamerly and C. Elkan. Learning the k in k-means. In *NIPS*, 2003.

[8] N. Kumar, L. Zhang, and S. K. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *ECCV*, pp. 364–378, 2008.

[9] B. Manthey and H. Röglin. Improved Smoothed Analysis of the k-Means Method. In *SODA* (arXiv:0809.1715), 2009.

[10] F. Nielsen, J.-D. Boissonnat, and R. Nock. On Bregman Voronoi diagrams. In *SODA*, pp. 746–755, 2007.

[11] F. Nielsen and R. Nock. Bregman sided and symmetrized centroids. In M. Ejiri, R. Kasturi, editor, *International Conference on Pattern Recognition (ICPR)*. IEEE CS Press, 2008. (arXiv:0711.3242)

[12] R. Nock, P. Luosto, and J. Kivinen. Mixed Bregman clustering with approximation guarantees. In *ECML/PKDD (2)*, pages 154–169, 2008.

[13] D. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, pages 91–111, Vol. 60, 2005.