Tailored Bregman Ball Trees for Effective Nearest Neighbors

Frank Nielsen$^1$ Paolo Piro$^2$ Michel Barlaud$^2$

$^1$Ecole Polytechnique, LIX, Palaiseau, France  
$^2$CNRS / University of Nice-Sophia Antipolis, Sophia Antipolis, France

25th European Workshop on Computational Geometry  
March 16, 2009  
ULB, Brussels, Belgium
Outline

Introduction
  Bregman Nearest Neighbor search
  Bregman Ball Trees (BB-trees)

Improved Bregman Ball Trees
  Speeded-up construction
  Adaptive node degree
  Symmetrized Bregman divergences

Experiments
Nearest Neighbor (NN) search

Applications: computer vision, machine learning, data mining, etc.

Nearest neighbor $\text{NN}(q)$

Given:
- a set $S = \{p_1, ..., p_n\}$ of $n$ $d$-dimensional points
- a query point $q$
- a dissimilarity measure $D$

then

$$\text{NN}(q) = \arg \min_i D(q, p_i) \quad (1)$$

For asymmetric $D$ (like Bregman divergences):

$$\text{NN}_F^l(q) = \arg \min_i D(q, p_i) \quad \text{(left-sided)}$$

$$\text{NN}_F^r(q) = \arg \min_i D(p_i, q) \quad \text{(right-sided)}$$

$$\text{NN}_F(q) = \arg \min_i (D(p_i \| q) + D(q \| p_i))/2 \quad \text{(symmetrized)}$$
Bregman divergences $D_F$

$F(x) : \mathcal{X} \subset \mathbb{R}^d \mapsto \mathbb{R}$ strictly convex and differentiable generator

$$D_F(p\|q) = F(p) - F(q) - (p - q)^T \nabla F(q)$$ (2)

Bregman sided NN queries are related by Legendre conjugates:

$$D_{F^*}(\nabla F(q)\|\nabla F(p)) = D_F(p\|q) \quad \text{(dual divergence)}$$

Widely used as distortion measures between image features:

- **Mahalanobis** squared distances (symmetric)
  $$F(x) = \Sigma^{-1} x \quad (\Sigma \succ 0 \text{ is the covariance matrix})$$

- **Kullback-Leibler (KL)** divergence (asymmetric)
  $$F(x) = \sum_{j=1}^{d} x_j \log x_j$$
Naïve search methods

**Brute-force** linear search:

- exhaustive brute-force $O(dn)$
- randomized sampling $O(\alpha dn)$, $\alpha \in (0, 1)$

**Randomized sampling**

- keep a point with probability $\alpha$
- mean size of the sample: $\alpha n$
- speed-up: $\frac{1}{\alpha}$
- mean rank of the approximated NN: $\frac{1}{\alpha}$
Data structures for improved NN search

Two main sets of methods:
▶ mapping techniques (e.g. locality-sensitive hashing, random projections)
▶ tree-like space partitions with branch-and-bound queries (e.g. $kD$-trees, metric ball and vantage point trees)
  ▶ faster than brute-force (pruning sub-trees)
  ▶ approximate NN search

Extensions from the Euclidean distance to:
▶ arbitrary metrics: vp-trees [Yianilos, SODA 1993]
▶ Bregman divergences: $k$-means [Banerjee et al., JMLR 2005]

We focus on Bregman Ball trees [Cayton, ICML 2008]
Outline of BB-trees (I)

BB-tree construction
Recursive partitioning scheme

1. 2-means clustering (keep the two centroids $c_l$, $c_r$)

2. Bregman Balls $B(c_l, R_l)$ and $B(c_r, R_r)$ (possibly overlapping)

3. continue recursively until matching a stop criterion

Termination criteria:

- maximum number of points $l_0$ stored at a leaf
- maximum leaf radius $r_0$
Outline of BB-trees (II)

Branch-and-bound search

1. Descend the tree from the root to the leaves
   - At internal nodes, choose child whose ball is “closer” to $q$ (the sibling is temporarily ignored)
   - At leaves, search for the NN candidate $p'$ (brute force)

2. Traverse back up the tree (check ignored nodes)
   - project $q$ onto the ball $B(c, R)$ (bisection search):
     \[
     q_B = \arg\min_{x \in B} D_F(x \| q)
     \]
   - if $D_F(q_B \| q) > D_F(p' \| q)$ the node can be pruned out
Outline of BB-trees (III)

Bregman annuli

Lower/upper bounds to speed-up geodesic bisection search:

$$B(c, R, R') = \{ x | R \leq D_F(x||c) \leq R' \}$$

$$D_F(p'||q) < D_F(y||q) \text{ (prune out)}$$

$$D_F(p'||q) > D_F(x||q) \text{ (explore)}$$
Our main contributions

From BB-tree to BB-tree++:

▶ Speed up construction time (Bregman 2-means++)
▶ Learn the tree branching factor (G-means)
▶ Explore nearest nodes first (priority queue)
▶ Handle symmetrized/mixed Bregman divergences

We mainly focus on approximate NN queries (stop the search once a few leaves have been explored)
Speed up construction time

We replace Bregman 2-means by a careful light *initialization* of the two cluster centers [Arthur et al., SODA 2007]

**Bregman 2-means++**

1. pick the first seed $c_l$ uniformly at random
2. for each $p_i \in S$ compute $D_F(p_i||c_l)$
3. pick the second seed $c_r$ according to the distribution:

   $$\pi_i = \frac{D_F(p_i||c_l)}{\sum_{p_j \in S} D_F(p_j||c_l)} \quad (3)$$

- Good approximation guaranties [Nock et al., ECML 2008].
- Fast tree construction, nice splitting
Learning the tree branching factor (I)

Goal  Get as many as possible non-overlapping Bregman balls

Example  Three separated Gaussian samples.

<table>
<thead>
<tr>
<th></th>
<th>dataset</th>
<th>2–means</th>
<th>3–means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Method
adapt the branching factor $bf_i$ of each internal node
Learning the tree branching factor (II)

**G-means**

- assume Gaussian distribution of each group of points
- use Bregman 2-means++ initialization to split a set
- apply the Anderson-Darling normality test to the two clusters
- if the test returns true, we keep the center, otherwise we split it into two
- repeat for each new cluster

Ongoing work: generalization to *goodness-of-fit* tests for exponential family distributions (e.g. Stephens test).
Handling symmetrized Bregman divergences

Why?

- required by content-based information retrieval (CBIR) systems
- technically are not Bregman divergences

Example: $\text{SKL} \& \text{JS}(p; q) = \frac{1}{2} \text{KL}(p\|\frac{p+q}{2}) + \frac{1}{2} \text{KL}(q\|\frac{p+q}{2})$

Proposed solutions:

- symmetrized Bregman centroid of $B(c, R)$: *geodesic-walk* algorithm of [Nielsen et al., SODA 2007].
- mixed BB-trees: store two centers for each ball $B(l, r, R)$ mixed Bregman divergence [Nock et al., ECML 2008]

\[
D_{F,\alpha}(l\|x\|r) = (1 - \alpha)D_{F}(l\|x) + \alpha D_{F}(x\|r), \quad \alpha \in [0, 1] \quad (4)
\]

(for $\alpha = \frac{1}{2}, \ l = r$ we find the symmetrized Bregman div.)
Nearest neighbors for Image Retrieval

Task  find similar images to a query
  ▶  $S$ dataset of feature vectors (descriptors)
  ▶  $q$ descriptor of a query image
  ▶  retrieve the most similar descriptor (image) $\text{NN}(q)$

Example  SIFT descriptors:  [Lowe, IJCV 2005].

Dataset
10,000 images from PASCAL Visual Object Classes Challenge 2007
  ▶  10,000 database points (for building the tree)
  ▶  2,360 query points (for on-line search)
  ▶  dimension $d = 1111$
Performance evaluation

Approximate search
Find a “good” NN, i.e. a point close enough to the true NN
- explore a given amount of leaves
- from near-exact search to visiting one single leaf

<table>
<thead>
<tr>
<th>speed-up</th>
<th>number of divergence computations (ratio of brute-force over BB-tree++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{avg}$</td>
<td>average approximated NN rank</td>
</tr>
<tr>
<td>$NC$</td>
<td>number of points closer to the approximated NN ($NC = R_{avg} - 1$)</td>
</tr>
</tbody>
</table>
BB-tree construction performances

- **iter** number of $k$-means iterations
- **bs** maximum number of points in a leaf
- **depth** maximum tree depth
- **$depth_{avg}$** average tree depth
- **nLeaves** number of leaf nodes

<table>
<thead>
<tr>
<th>Method</th>
<th>Iter</th>
<th>Depth</th>
<th>$depth_{avg}$</th>
<th>nLeaves</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-means</td>
<td>10</td>
<td>53</td>
<td>28.57</td>
<td>594</td>
<td>1</td>
</tr>
<tr>
<td>2-means++</td>
<td>10</td>
<td>58.33</td>
<td>31.18</td>
<td>647</td>
<td>1.03</td>
</tr>
<tr>
<td>2-means+++</td>
<td>0</td>
<td>20</td>
<td>10.76</td>
<td>362</td>
<td>19.71</td>
</tr>
</tbody>
</table>
Asymmetric NN queries

**BB-tree vs BB-tree++**

![Graph showing speed-up comparison between BB-tree++ and BB-tree Cayton with SIFT dataset (KL)]
Symmetrized NN queries

BB-tree++ vs Randomized Sampling

SIFT dataset (SKL)

BB-tree++ (BF=4, bs=100)
Random Sampling

Average approximate NN rank

Speed-up wrt. brute-force
Conclusion

BB-tree++:

- adapted to the inner geometric characteristics of data
- speed up construction (\(k\)-means careful initialization)
- speed up search (priority queue)
- handle symmetrized Bregman divergences
- promising results for image retrieval (SIFT histograms)

Ongoing work:

- design the most appropriate divergence to a class of data
- extensive application to feature sets arising from image retrieval/classification