UNIVERSITY OF CALIFORNIA, SAN DIEGO

Bregman proximity search

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science and Engineering

by

Lawrence Cayton

Committee in charge:

Professor Sanjoy Dasgupta, Chair
Professor Virginia de Sa
Professor Charles Elkan
Professor Bhaskar Rao
Professor Lawrence Saul

2009

The dissertation of Lawrence Cayton is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I thank my adviser, Sanjoy Dasgupta, who has been patient and flexible over my years here. Thanks to Charles Elkan, Serge Belongie, and Lawrence Saul, for providing valuable advice whenever I needed it. Thanks to also to Bhaskar Rao and Virginia de Sa for being on my committee and making useful suggestions.

I am grateful to Sameer Agarwal, Kristin Branson, and Eric Wiewiora, who helped me through my first few years here. Ronald Graham, perhaps unknowingly, eased me through a few trying quarters with his cheerful, friendly temperament. Finally, thanks to my friends and family, who have been supportive throughout graduate school.

All results appearing in this dissertation are the sole work of the author, with the exception of chapter 8, which is joint work with Sanjoy Dasgupta. Some of this work has published or submitted:

- Portions of chapter 4 and all of chapter 5 appear in
  L. Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the International Conference on Machine Learning*, 2008.

- Most of chapter 8 appears in
  L. Cayton, S. Dasgupta. A learning framework for nearest neighbor search. In *Advances in Neural Information Processing Systems 20*, 2007.

- Portions of chapter 4 and all of chapter 6 are under review:
  L. Cayton. Efficient bregman range search. *Submitted,* 2009.

VITA

2003	B. S. in Computer Science; second major in Mathematics *magna cum laude*, Washington University in St. Louis
2005	M. S. in Computer Science, University of California, San Diego
2009	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

L. Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the International Conference on Machine Learning*, 2008.

L. Cayton, S. Dasgupta. A learning framework for nearest neighbor search. In *Advances in Neural Information Processing Systems 20*, 2007.

S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. Kriegman and S. Belongie. Generalized non-metric multidimensional scaling. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2007.

L. Cayton, S. Dasgupta. Robust Euclidean embedding. In *Proceedings of the International Conference on Machine Learning*, 2006.

ABSTRACT OF THE DISSERTATION

Bregman proximity search

by

Lawrence Cayton
Doctor of Philosophy in Computer Science and Engineering
University of California San Diego, 2009
Professor Sanjoy Dasgupta, Chair

In this dissertation, we study efficient solutions to proximity search problems where the notion of proximity is defined by a bregman divergence. Proximity search tasks are at the core of many machine learning algorithms and are a fundamental research topic in computational geometry, databases, and theoretical computer science. Perhaps the most basic proximity search problem is nearest neighbor search: on any input query, retrieve the most similar items from a (potentially large and complex) database efficiently, *i.e.* without performing a full linear scan. There is a massive body of work on proximity problems when the notion of distance is a metric, largely relying on the triangle inequality. In contrast, the tasks of efficient proximity search for the family of bregman divergences is essentially unstudied. This family includes standard Euclidean distance (squared), the Mahalanobis distance, the KL-divergence (relative entropy), the Itakura-Saito divergence, and many others. Bregman divergences need not satisfy the triangle inequality, nor do they need to be symmetric. Because these basic properties cannot be relied on, metric-based data structures are not immediately applicable.

The dissertation presents a data structure and accompanying search algorithms for nearest neighbor search and range search, the two most fundamental proximity tasks. The data structure is based on a hierarchical space decomposition based on simple convex bodies called bregman balls. The search algorithms work by repeatedly calling an extremely fast optimization procedure. These optimiza-

tion procedures rely on geometric properties of bregman divergences and notions of duality. We demonstrate that these search algorithms often provide orders of magnitude speedup over standard brute force search.

We also examine alternate approaches to bregman proximity problems. We show that two classical data structures can be adapted for bregman divergences, yielding some theoretical bounds on query time.

In the final part of the dissertation, we examine a novel approach to building nearest neighbor data structures based on learning. This approach yields theoretical guarantees akin to those in learning theory, which provides an alternative way to rigorously assess search performance. We explore the potential of this framework through several data structures.

Chapter 1

Introduction

Nearest neighbor search is a fundamental primitive of machine learning and a core topic of research in computational geometry, databases, theoretical computer science, information retrieval, and elsewhere. The basic problem is very easy to state: we have some database of points $X = \{x_1, x_2, \ldots, x_n\}$, often assumed to be residing in $\mathbb{R}^D$, and on a given query q, we wish to return the point most similar to q in the database. Similarity is encoded in some function $d(\cdot, \cdot)$ which takes a query, database point pair and maps them to a number that can be roughly interpreted as the distance between them. When the database and the query both reside in $\mathbb{R}^d$, for example, a standard choice is Euclidean distance

$$d(x, q) = \|x - q\|_2 \equiv \sqrt{\sum (x_i - q_i)^2}.$$

In this case, the task of nearest neighbor search is to find the $x \in X$ that is spatially closest to q.

Nearest neighbor (NN) search is used in a huge variety of applications; perhaps the most obvious is in information retrieval. Suppose we have some library of documents (*e.g.* web pages) and wish to search through them effectively. A standard approach to this problem is through the *vector space model*, in which each document is mapped to some high-dimensional vector [MRS08]. One reasonable vector representation is created by mapping each document to a list of its word counts; specifically position i will be the number of times word i occurs in the document, usually weighted to reflect the frequency of i across the corpus. A

natural way to search through the documents is to use some query phrase, such as one does on the internet. Using a mapping similar to the one used for documents, this query phrase is transformed into a high-dimensional vector. After the mapping, the documents and query lie in some high-dimensional Euclidean space, and hence similarity can be assessed using Euclidean distance or some other notion of distance between vectors. Finding the documents most relevant to the query is thus boiled down to a straightforward nearest neighbor search. When the document corpus is large, as is the case for web search, the efficiency of the NN search operation is of paramount importance.

In machine learning, NN search is most often used to approximate complex functions. In the standard learning setting, one must learn a mapping $f : \mathcal{X} \to \mathcal{Y}$ based on a set of examples of this mapping $\{(x_i, y_i) \mid i = 1, \ldots, n\} \subset \mathcal{X} \times \mathcal{Y}$. This example set is assumed to be drawn iid from some distribution over pairs and the goal is to find f which performs well on *any* pair drawn from this distribution. The two most common instances of this task are classification, where $\mathcal{Y} = \{-1, +1\}$, and regression, where $\mathcal{Y} = \mathbb{R}$. For simplicity, consider the case of classification. One approach to learning f is to assume that it has some parametric form. For example, one might assume that f is a linear function that is thresholded to determine the class of an example. Such an f can be written as $f(x) = \text{sign}(\langle w, x \rangle)$. The parameter vector w will be learned from the training examples. Support vector machines and neural networks both follow this parametric approach (though they can be used to learn non-linear functions). Challenges of using parametric approaches include choosing an effective parametrization of the problem and developing an effective optimization method to estimate the parameters. In contrast, NN search provides an immediate, almost trivial way to build f: store all the training examples, and to classify a test point x, simply return the class of the nearest x_i in the database. Using the NN classifier requires no guess of the functional form of f and requires no optimization procedure. Moreover, the nearest neighbor classifier has fairly strong theoretical guarantees: in the infinite sample limit, it has error no more than twice the optimal Bayes error [CH67].

These two examples illustrate the usefulness of NN search, but there are at

least two major challenges to the successful application of NN methods.

1. The performance of a NN search-based system depends on the notion of distance used. In many applications, it is not at all clear what the best notion of distance is; in fact, there is a fairly large body of work in which this selection procedure is cast as a learning problem itself [XNJR03, GRHS05, WBS06, DKJ+07].

2. The naive algorithm for computing a query's NN requires comparing the query to every point in the database. This search is often prohibitively expensive. For example, the database queried in web search is the set of all web pages. Similarly, many applications in machine learning have training sets containing millions or billions of examples.

The choice of distance function is largely application-specific, but the second problem is a somewhat more fundamental computer science issue. In a vast body of research, many data structures have been developed to store a database in such a way so that only a small portion of it need be examined to retrieve a NN. We review some highlights in the next chapter, but the essential idea behind many of these data structures is decompose the space in which the database points live into small cells, and then organize these cells using an abstract data structure (*e.g.* a tree). If the database consists of points in the plane, for example, a very simple way to decompose space is using a grid. See figure 1.1. The hope is that only the database points in a few grid squares will need to be examined, and the rest ignored.

Of course the applicability of a data structure to a particular application depends heavily on the choice of distance. A data structure making use of properties of the ℓ_2-norm, for example, is unlikely to be directly applicable to other notions of distance. In hopes of isolating the properties of an effective NN data structure that is broadly useful, researchers have turned efforts towards designing data structures for *families* of distance functions. Perhaps since ℓ_2 is the most intuitive notion of distance, most families are based on some abstraction of it. The

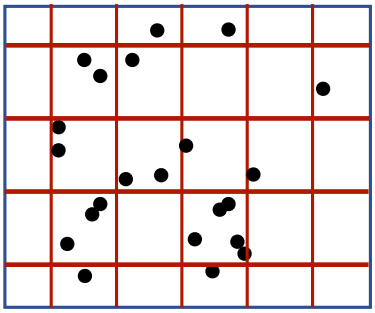


Figure 1.1: A simple decomposition of $\mathbb{R}^2$.

simplest generalization is to the family of ℓ_p norm distances

$$d(x, q) = \|x - q\|_p \equiv \left(\sum |x_i - q_i|^p\right)^{1/p},$$

where $p \in [1, \infty]$. A different way to generalize the ℓ_2 distance is to scale and rotate the axes with a positive definite matrix Q. The result is the Mahalanobis distance

$$d(x, q) = \sqrt{(x - q)^\top Q(x - q)},$$

defined for any $Q \succ 0$. Though these classes of distances are substantial generalizations of ℓ_2, many notions of distance used in computer science fall outside. One example is the edit distance, also called the *Levenshtein distance*, between two strings. It is defined as the minimum number of operations to transform one string into another, where an operation is a character substitution, deletion, or insertion. This distance cannot be characterized as an ℓ_p norm. Another example is the *shortest-path distance*. Suppose that the elements of the database are nodes of a graph, connected by edges with non-negative weights. A natural notion of distance between two nodes is the shortest path between them on the graph, efficiently computable via Dijkstra's algorithm. These metrics generated by graphs are not norms in general, though some special cases are [LLR95].

The shortest path, ℓ_p, and edit distances fall into the very general class of *metrics*, distance functions that are symmetric, non-negative, and which satisfy the triangle inequality. Perhaps the bulk of work on NN search is at this level

of generality, though much of it lies somewhere below. The triangle inequality provides a powerful-enough tool to effectively index many types of data for efficient NN retrieval.

The family of metrics is extremely broad, yet there are some important examples of distance notions that do not satisfy the metric axioms. These examples are not contrived abstractions, but rather important functions that are used extensively in data mining and elsewhere. Perhaps the best example of a distance-like notion that is not a metric is the KL-divergence

$$d_f(x, q) = \sum x_i \log \frac{x_i}{q_i},$$

which is the natural notion of distance between probability distributions [Csi91]. It has been used to compare histograms in a wide variety of applications, including text analysis, image classification, and content-based image retrieval [PTL93, PBRT99, RMV07]. 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 the Mahalanobis distance, used in classification [SP98]; the Itakura-Saito divergence, used in sound processing [GBGM80]; and ℓ_2^2 distance. The class of bregman divergences is quite different structurally from the class of metrics: whereas metrics are defined as any function satisfying the basic metric axioms, bregman divergences are defined via simple convexity notions. Thus working with bregman divergences requires a significantly different toolkit than the one employed for metrics. For NN retrieval in particular, bregman divergences present a challenge since they need not satisfy the triangle inquality, which is perhaps the workhorse of NN methods.

1.1 Overview of results

The central subject of this dissertation is efficient proximity search methods for bregman divergences. In addition to nearest neighbor search, we study another fundamental geometric problem called range search; we lump the two under the generic title of proximity search. In a sense, the work in this dissertation provides

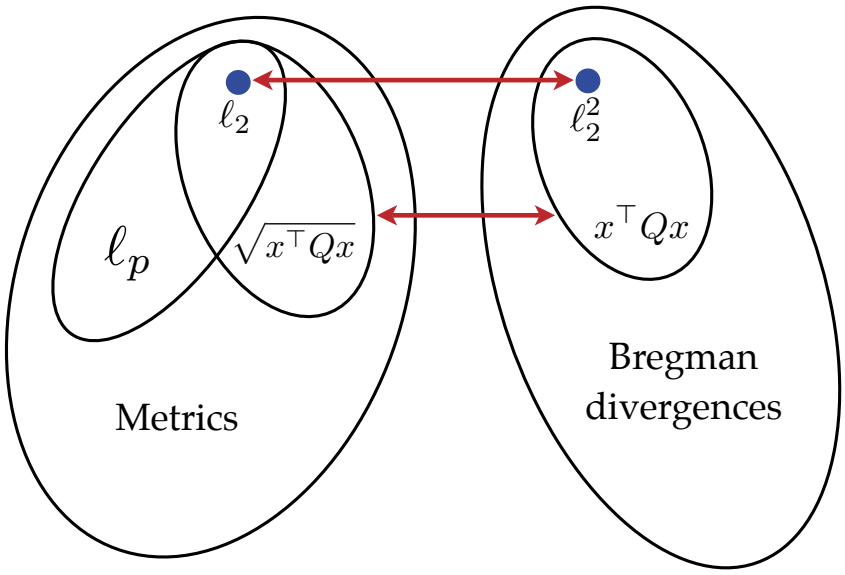


Figure 1.2: The space of metrics and bregman divergences are mostly disjoint. The only connection is through ℓ_2 and, more generally, Mahalanobis distances. The Mahalanobis distance is denoted $x^\top Qx$ above.

a parallel to the vast body of work on proximity search methods for metrics; but whereas that line of work builds from the triangle inequality, this line of work builds from convex optimization. See figure 1.2.

This dissertation presents the bregman ball tree (bb-tree), the first data structure for efficient proximity search with an arbitrary bregman divergence. Chapter 4 contains the bulk of the technical development of the data structure, including analyses of important optimization tasks associated with the bb-tree. In chapter 5, we discuss how to use the bb-tree for NN search and demonstrate empirically the effectiveness of the data structure. In chapter 6, we consider the other fundamental proximity task, range search. We show that the bb-tree can be used for this task as well, and provide an empirical evaluation on a variety of data sets. In chapter 7, we adapt classical data structures to bregman proximity problems. These developments give the only known theoretical query time bound for bregman range search and have other uses as well.

In chapter 8, we leave the main thread of the dissertation behind somewhat

to describe a novel approach to proximity search that may be applied to *any* type of distance. The basic idea is to treat a data structure as a function to be learned from sample queries. This approach provides a novel way to build data structures and also provides for a novel type of theoretical bound on query time based on generalization theory. This type of bound may be more useful in some situations than the worst-case bounds generally applied to NN data structures, such as in chapter 7. We apply the framework to two standard data structures, kd-trees and the rectilinear cell structures employed by locality sensitive hashing, and to the bb-tree.

Before entering the main body of this dissertation, we overview the vast body of work on metric methods for efficient proximity search in chapter 2, focusing mostly on developments that have been useful in the machine learning community. Chapter 3 is a self-contained overview of the basics of bregman divergences. The basics described in this chapter are known, but have not been previously collected.

Chapter 2

Efficient proximity search

This chapter gives a brief overview of efficient proximity search methods designed for metric spaces (or something more specific). The body of literature on this subject is absolutely enormous, so complete coverage is impossible. We will focus on the data structures that have been used in machine learning and data analysis.

Let us briefly survey other major sources of information on data structures for proximity search. The mammoth text of Samet [Sam06] contains details of many, many data structures, focusing mostly on classical data structures. The excellent paper of Clarkson [Cla06] surveys work on data structures with respect to intrinsic dimensionality. The survey paper of Indyk [Ind06] describes fairly recent developments in NN search for high-dimensional spaces. [AE99] provides an overview of algorithms for many types of range search. There are many other texts and surveys available.

There are two fundamental proximity search problems: nearest neighbor search and range search. In both problems, there is some database of n points $X = \{x_1, \ldots, x_n\}$ which we will assume for convenience to be a subset of $\mathbb{R}^D$. The nearest neighbor problem is to return the point nearest to a given input query q; the k-NN problem is to return the k nearest. For simplicity, we only work with the 1-NN problem, though the algorithms and data structures we describe can be easily adjusted to work for the k-NN problem. The general range search problem is to return all points in the database within a given query range Q, where Q is

typically some convex body. For machine learning applications, the typical range of interest is a ball range, *i.e.* the set of all points within a given distance γ of a given query q. Throughout this dissertation, *range search* refers to ball range search.

The quantity of interest is the query time. The build time is not explicitly considered, though data structures requiring more than roughly $O(n \log n)$ time to build are generally impractical. There are of course many important variations; for example, the *all-NN* problem is to find the NN for every $x_i \in X$ among the remaining database points. We focus on the static NN and range search problems, but many of the techniques we describe are more broadly applicable.

Most data structures for proximity search are based on the idea of a *space decomposition*. The basic idea is to break up the subset of $\mathbb{R}^D$ containing the database into small, simple cells. For example, a simple partition is a grid (refer back to figure 1.1, page 4). These cells are then used as the basis for a branch-and-bound search: first the cell containing the query is examined, then other cells are searched or pruned out as possible.

The following data structures are all tree-based organizations of some simple cells (spheres, cubes, rectangles, *etc.*), as is the bregman ball tree introduced later in this dissertation. The root node is associated with a cell containing the entirety of the database; nodes at lower levels are associated with smaller cells that contain some subset of the database. Data structures in this category all use the same branch-and-bound template algorithm for range search and another for nearest neighbor search.

We describe the algorithm for NN search on a query q. Pseudocode appears as Algorithm 1. First, the tree is descended; at each level the algorithm chooses which child to visit according to some heuristic criteria (*e.g.* the child whose cell is closer to q is visited). The other child is temporarily ignored. Once a leaf is reached, the distances from q to all of its database points are computed; the nearest point x_c is the *candidate* NN. Now the algorithm traverses back up the tree and evaluates which of the previously ignored siblings must be explored. A sibling node must be explored if it is possible that its cell contains a point closer to q than x_c;

Algorithm 1 NNSEARCH

Input: Tree node T (corresponding to cell $T.C$, points $T.X$), and query q.
if T is a leaf **then**
  Compute distances from q to all points in $T.X$
  return x_c, the closest point.
end if
Let l be the left child of T and r be the right child.
Let x_l = NNSEARCH(l,q).
Compute a lower bound $lb = \min_{x \in T.C} d(x, q)$.
if $lb > d(x_l, q)$ **then**
  return x_l. {Can skip the right node}
else
  Let x_r = NNSEARCH(l,q). {Exploring the right node.}
  return $\text{argmin}_{x \in \{x_l, x_r\}} d(x, q)$.
end if

otherwise it can be pruned. The algorithm proceeds up the tree, descending and exploring subtrees as necessary, until the root is reached.

The range search algorithm is similar, but has some important changes. Pseudocode appears as Algorithm 2. This basic algorithm (and the NN search algorithm) will be described in more detail later for the bregman ball tree (see chapter 5 and 6).

We now described several data structures with which the basic range and NN search algorithms can be used. Figure 2.1 shows examples of the space decompositions defined by these data structures.

The *quad-tree* [FB74] is perhaps the simplest proximity search structure. It is a tree-based organization of a regular grid over the database. Each cell is simply an axis-aligned hypercube. The root node of the tree corresponds to the hypercube containing the entire database. Each dimension of this hypercube is then split in half, forming 2^D children hypercubes. This partitioning is continued until each bucket only has one database point in it (or some small fixed number of points). A major drawback of this data structure is that the depth of the tree can

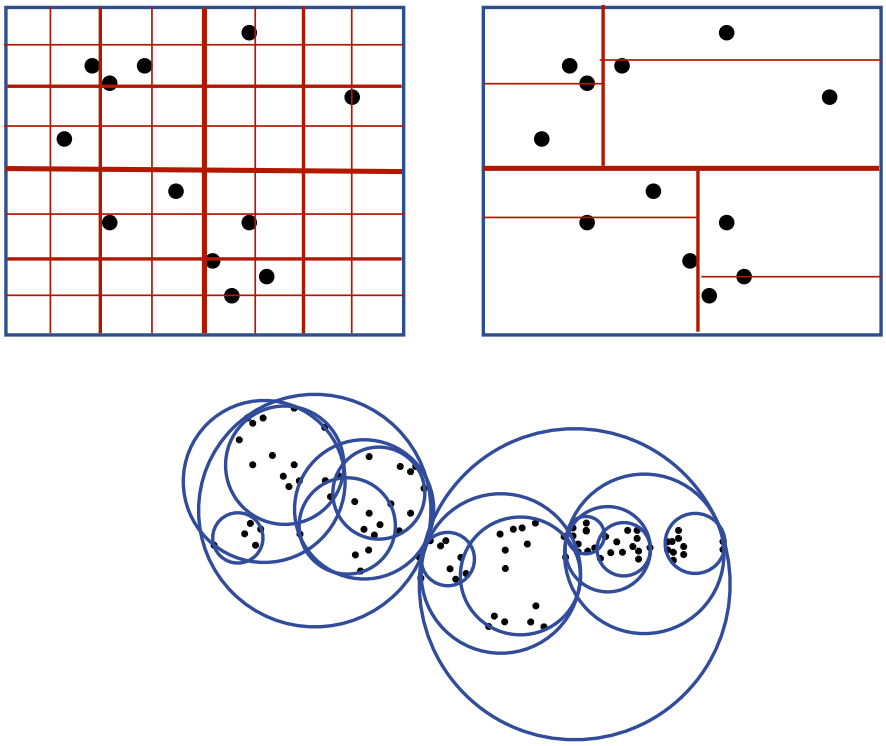


Figure 2.1: Three tree-based space decompositions. The line thickness indicates the depth of the split on the top two. **Top left:** quad-tree. **Top right:** kd-tree. **Bottom:** Metric tree.

Algorithm 2 RangeSearch

Input: Treenode T, query q, radius γ.
if $T.C \subset B(q, \gamma)$ **then**
  return $T.X$.
else if $T.C \cap B(q, \gamma) = \emptyset$ **then**
  return $\emptyset$.
else if T is a leaf **then**
  Compute distances from q to all points in leaf.
  return X_c, the set of points in T within distance γ of q.
else
  Let l (r) be the left (right) child of T.
  return RangeSearch(l,q,γ) $\cup$ RangeSearch(r,q,γ).
end if

be $O(n)$, which can hurt retrieval times. Nevertheless, this classical data structure has many applications and useful properties; for instance, it is provably efficient for some important variants of the NN problem [Cla85].

The kd-tree [FBF77] is a popular data structure based on a couple of tweaks to the quad-tree that seem to make it much more practical. First, the cells are axis-aligned *rectangles*, rather than cubes. Second, a node has only two children. A node is split into two children by picking a dimension (heuristically), ordering the points according the to their value in that dimension, and splitting them at the median. This second property ensures that the tree will have height $O(\log n)$. Though useful in many applications, the performance of kd-trees has been widely observed to degrade badly with the dimensionality of the database [Moo00]. Both the kd-tree and the quad-tree work for any ℓ_p norm.

Metric trees [Omo89, Uhl91, Yia93, Moo00] extend the basic ideas of the kd-trees to arbitrary metric spaces. The cell in this case is a metric ball. The search algorithm uses the triangle inequality to determine if a node can be pruned out. Metric trees seem to scale with dimensionality much better than kd-trees [Moo00, LMG06], though high-dimensional data is still a major challenge.

In general, the performance of these data structures is not *provably* sublinear

in the database size, but these data structures have performed well in a number of practical applications. Unfortunately, the worst-case behavior of *any* spatial data structure for exact NN search is widely believed to degrade to linear scan at a rate exponential in the dimension or require exponential space [Ind06]. Similar behavior has been demonstrated for range search [BCP93]. Moreover, this behavior has been observed in practice, though in varying levels of severity [WSB98]. For machine learning applications, this exponential dependence on dimensionality severely limits the applicability of these NN methods.

One commonly observed phenomenon in machine learning is that many apparently high-dimensional data sets are actually *intrinsically* low-dimensional [TdSL00, RS00]. For instance, a set of photographs of a fixed object taken from different angles will be extrinsically high-dimensional (the dimensionality of an image being the number of pixels), but will roughly lie on a three-dimensional manifold (the three dimensions that determine the placement of the camera). One major line of work capitalizes on this disparity with data structures and search algorithms that are exponential only in the intrinsic dimensionality of a dataset [Cla99, KR02, KL04, BKL06].

Still, it would be nice if there was no exponential dependence at all. Impelled by the difficulty of proving strong worst-case bounds for NN and range search, researchers turned to a relaxation of the problem. A $(1+\epsilon)$-NN of query q is defined as any point $y \in X$ such that

$$d(y,q) \leq (1+\epsilon)d(x,q)$$

for all $x \in X$. This relaxation led to some significant breakthroughs both in theory and in practice. [AMN+98] developed a data structure that is reminiscent of kd-tree which has a $O(\log n)$ query time, albeit with constant that is roughly $(\frac{D}{\epsilon})^D$. Though this is exponential in D, it is still a major result as it proves a worst-case bound on a tree-based space decomposition and because one can control the factor that is exponential in D by increasing ϵ.

A very different family of data structures is based on *locality sensitive hashing* (LSH) [IM98, DIIM04, AI06] (see also [KOR00, Har01]). Unlike the data structures we described previously, LSH-based schemes do not use a tree-based

organization of the data. These schemes are based on properties of random projections and embedding techniques; we will look at one such data structure in chapter 8. The LSH idea was a substantial breakthrough as these schemes have no exponential dependence on dimensionality, though they are exponential in the approximation factor.

Similarly, one can relax range search to approximate range search. An exact range query requires that all points within a range be returned and no points outside of that range; an approximate range query allows the points near the boundary to be returned or skipped. [AM00] extends the results of [AMN$^+$98] to approximate range searching, giving similar bounds using the same data structure. [CLM08] gives algorithms for approximate range search without any exponential factors using a reduction similar to the one in [KOR00].

For machine learning applications, an approximate nearest neighbor (or range) is often good enough. Consider using NN search for classification. In this case, the label of the query point is what one is after, not the NN itself. Rather, the class label of the NN provides a reasonable guess as to the class label of the query, as do other nearby points. This flexibility is fortunate, as approximate nearest neighbor algorithms have much lower theoretical time complexity and are much faster in practice [LMGY04].

In machine learning, the tree-based approaches still dominate; in particular, metric and kd-trees have found wide application. There are several reasons. First, these methods are flexible. Adapting them to solve approximate NN problem instead of the exact NN problem is trivial. More importantly, a wide variety of proximity problems can be solved using these methods with simple modifications, as we will discuss. The second reason is that many of the algorithms with theoretical guarantees are somewhat complex to implement and their complexity bounds often hide large constant factors. Thus there is still much work to be done in making these algorithms practical, though some LSH-based algorithms have appeared in applications [SVD03, CS07, JKDG08]. In contrast, the tree-based approaches are simple to implement and optimize. Finally, and most importantly, the tree-based approaches often perform very well in learning applications. For

example, the *spill tree* is a modified metric tree for approximate NN search that exhibited a dramatic speedup over brute-force search, even for high-dimensional data [LMGY04]. In that paper, the spill tree was compared to LSH-based scheme (empirically) and outperformed it handily.

Before closing this chapter, we note that there is a broad range of fundamental learning algorithms that require proximity search beyond the basic NN classifier/function estimator described before. Indeed, proximity search can be viewed as something of a data analysis primitive [GM00]. Let us mention a few examples. Locally weighted regression and kernel density estimation/regression both require retrieving points in a region around a test point. Neighborhood graphs—used in manifold learning, spectral algorithms, semisupervised algorithms, and elsewhere—can be built by connecting each point to all other points within a certain radius; doing so requires range search at each point. Computing point-correlation statistics, distance-based outliers/anomalies, and intrinsic dimensionality estimations also require range search. Metric and kd-trees have been applied to these and other learning problems; see, *e.g.* [Moo99, GM00, LMG06, SNS06].

Chapter 3

Bregman divergences

In this chapter, we give a self-contained overview on the basics of Bregman divergences. Though these results are mostly known, they do not appear to be collected anywhere.

Bregman divergences were introduced by Bregman in [Bre67] and have traditionally been studied by researchers in the optimization and numerical analysis communities. The book [CZ97] contains a detailed overview of Bregman divergences and their application in iterative optimization algorithms. Within machine learning, Bregman divergences first appear in work on iterative scaling [LPP97] and online learning, *e.g.* [GLS97, HW98, AW01] and were later identified with the k-means algorithm [BMDG05].

Definition 1. *Let $f : C \subset \mathbb{R}^D \to \mathbb{R}$ be a strictly convex function that is differentiable on the relative interior of C. The bregman divergence based on f is defined as*

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

Note that $d_f : C \times \mathrm{relint}(C) \longrightarrow \mathbb{R}_+$.

A bregman divergence measures the distance between f and its first-order taylor expansion. See figure 3.1. Table 3.1 lists some common bregman divergences.

It is often assumed that f is not only strictly convex but is also *Legendre*. A function $f : C \to \mathbb{R}$ is Legendre if

Table 3.1: Standard bregman divergences. S_{++}^D denotes the cone of positive definite $D \times D$ matrices.

Name	**Domain**	**Base function**	$d_f(x, y)$
ℓ_2^2	$\mathbb{R}^D$	$\frac{1}{2}\|x\|_2^2$	$\frac{1}{2}\|x - y\|_2^2$
Mahalanobis ($Q \succ 0$)	$\mathbb{R}^D$	$\frac{1}{2}x^\top Qx$	$\frac{1}{2}(x-y)^\top Q(x-y)$
KL-Divergence	$\mathbb{R}_+^D$	$\sum x_i \log x_i$	$\sum x_i \log \frac{x_i}{y_i} - x_i + y_i$
Itakura-Saito	$\mathbb{R}_+^D$	$-\sum \log x_i$	$\sum \left(\frac{x_i}{y_i} - \log \frac{x_i}{y_i} - 1\right)$
Exponential	$\mathbb{R}^D$	$\sum e^{x_i}$	$\sum e^{x_i} - (x_i - y_i + 1)e^{y_i}$
Bit Entropy	$[0,1]^D$	$\sum(x_i \log x_i + (1-x_i)\log(1-x_i))$	$\sum(x_i \log \frac{x_i}{y_i} + (1-x_i)\log(\frac{1-x_i}{1-y_i}))$
Hellinger	$[-1,1]^D$	$-\sum \sqrt{1-x_i^2}$	$\sum \frac{1-x_iy_i}{\sqrt{1-y_i^2}} - \sqrt{1-x_i^2}$
ℓ_p^p, $p \in [1,\infty]$	$\mathbb{R}^D$	$\|x\|_p^p$	$\sum \|x_i\|^p - px_i\mathrm{sgn}(y_i)\|y_i\|^{p-1} + (p-1)\|y_i\|^p$
Log-det	S_{++}^D	$-\log\det X$	$\langle X, Y^{-1}\rangle - \log\det XY^{-1} - N$
von Neumann entropy	S_{++}^D	$\mathrm{tr}\,(X\log X - X)$	$\mathrm{tr}\,(X(\log X - \log Y) - X + Y)$

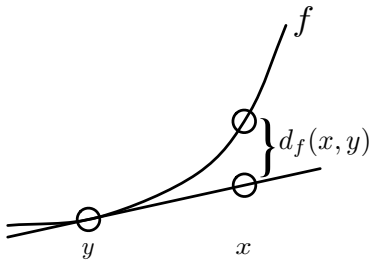


Figure 3.1: The bregman divergence between x and y.

- $C \neq \emptyset$ and its interior is convex;
- f is strictly convex and has continuous first partial derivatives; and
- if $x_1, x_2, \ldots \in C$ is a sequence converging to a boundary point of C, $\|\nabla f(x_n)\| \to \infty$.

These additional requirements aid in the analysis of certain iterative projection algorithms [CZ97].

The strict convexity of f implies that a bregman divergence is always non-negative and equal to zero only if its two arguments are identical. It is similarly easy to check that a bregman divergences is convex in its first argument, but not necessarily in its second.

3.1 Pythagorean theorem

The pythagorean theorem is a key property of bregman divergences that governs the geometry of bregman projections. This theorem underlies some of the bregman geometry we develop in chapter 4, even though it is not explicitly used.

Theorem 1. *Let C be a closed convex set and let $y = \operatorname{argmin}_{y \in C} d_f(y, z)$ be the bregman projection of z onto C. Further suppose that $x \in C$. Then*

$$d_f(x, z) \geq d_f(x, y) + d_f(y, z).$$

Proof. Define $G(x) = d_f(x, y) - d_f(x, z)$. Then

$$\begin{aligned} G(x) &= f(x) - f(y) - \langle \nabla f(y), x - y \rangle - (f(x) - f(z) - \langle \nabla f(z), x - z \rangle) \\ &= f(z) + \langle \nabla f(z), x - z \rangle - f(y) - \langle \nabla f(y), x - y \rangle, \end{aligned}$$

so G is linear in x. Moreover $G(y) = d_f(y, y) - d_f(y, z) = -d_f(y, z)$. Let $\alpha \in [0, 1]$ and $x_\alpha = \alpha x + (1 - \alpha)y$.

$$\begin{aligned} G(x_\alpha) = G(\alpha x + (1 - \alpha)y) &= \alpha G(x) + (1 - \alpha)G(y) \\ &= \alpha(d_f(x, y) - d_f(x, z) + d_f(y, z)) - d_f(y, z). \end{aligned}$$

Also,

$$G(x_\alpha) = d_f(x_\alpha, y) - d_f(x_\alpha, z),$$

so, when $\alpha > 0$,

$$d_f(x, y) + d_f(y, z) - d_f(x, z) = \frac{d_f(y, z) + d_f(x_\alpha, y) - d_f(x_\alpha, z)}{\alpha}.$$

We show that the right hand side is less than or equal to zero. Since $x_\alpha \in C$ and y is the projection of z onto C, $d_f(y, z) \leq d_f(x_\alpha, z)$. Thus we only need to show that $\frac{d_f(x_\alpha, y)}{\alpha} \leq 0$ for some α. We do this by taking the limit as $\alpha \to 0$ from the right:

$$\begin{aligned} \lim_{\alpha \to 0} \frac{d_f(x_\alpha, y)}{\alpha} &= \lim_{\alpha \to 0} \frac{f(\alpha x + (1 - \alpha)y) - f(y) - \langle \nabla f(y), \alpha x - \alpha y \rangle}{\alpha} \\ &= \lim_{\alpha \to 0} \frac{f(\alpha x + (1 - \alpha)y) - f(y)}{\alpha} - \lim_{\alpha \to 0} \frac{\langle \nabla f(y), \alpha x + (1 - \alpha)y - y \rangle}{\alpha} \\ &= \langle \nabla f(y), x - y \rangle - \lim_{\alpha \to 0} \langle \nabla f(y), x - y \rangle \quad \text{(by l'Hôpitals rule)} \\ &= 0. \end{aligned}$$

□

The previous inequality becomes an equality if C is an affine space.

Theorem 2. *Let C be an affine space and let $y = \operatorname{argmin}_{y \in C} d_f(y, z)$ be the bregman projection of z onto C. Let $x \in C$. Then*

$$d_f(x, z) = d_f(x, y) + d_f(y, z).$$

Proof. Let x' be any point in C, and set λ to satisfy $y = \lambda x + (1-\lambda)x'$. Define $x_\alpha = \alpha x + (1-\alpha)x'$, and let $G(x) = d_f(x,y) - d_f(x,z)$. Since $G(x_\alpha) = d_f(x_\alpha, y) - d_f(x_\alpha, z)$,

$$\frac{\partial G(x_\alpha)}{\partial \alpha} = \frac{\partial}{\partial \alpha} d_f(x_\alpha, y) - \frac{\partial}{\partial \alpha} d_f(x_\alpha, z). \tag{3.1}$$

Note that $x_\alpha \in C$; since y is the projection of z onto C and $x_\alpha = y$ when $\alpha = \lambda$, $d_f(x_\alpha, z)$ is minimized at $\alpha = \lambda$. Moreover, $d_f(x_\alpha, y)$ is also minimized at $\alpha = \lambda$ (it is zero there). Thus, both divergences on the right of (3.1) are minimized at $\alpha = \lambda$, implying $\left.\frac{\partial G(x_\alpha)}{\partial \alpha}\right|_{\alpha=\lambda} = 0$.

Using the linearity of G (see the proof of theorem 1), we can expand

$$\begin{aligned} G(x_\alpha) &= d_f(x_\alpha, y) - d_f(x_\alpha, z) \\ &= \alpha(d_f(x,y) - d_f(x,z)) + (1-\alpha)(d_f(x',y) - d_f(x',z)). \end{aligned}$$

Differentiating with respect to α, we get

$$\frac{\partial G(x_\alpha)}{\partial \alpha} = d_f(x,y) - d_f(x,z) - d_f(x',y) + d_f(x',z).$$

At $\alpha = \lambda$, $\frac{\partial G(x_\alpha)}{\partial \alpha} = 0$, so

$$d_f(x,y) - d_f(x,z) - d_f(x',y) + d_f(x',z) = 0.$$

This equality is true for all $x' \in C$, so picking $x' = y$, we get

$$d_f(x,z) = d_f(x,y) + d_f(y,z).$$

□

3.2 Conjugacy

Conjugacy is a central duality notion used in convex analysis [Roc70].

Definition 2. *The* convex conjugate *of a function* $f : \mathbb{R}^d \to \mathbb{R}$ *is*

$$f^*(x) = \sup_y \left\{ \langle x, y \rangle - f(y) \right\}.$$

The definition of a conjugate function immediately yields the *Fenchel-Young inequality*:

$$f(x) + f^*(y) \geq \langle x, y \rangle.$$

If f is differentiable, we can compute f^* by setting the derivative to zero:

$$\begin{aligned}
&\nabla_y \left(\langle x, y \rangle - f(y) \right) = x - \nabla f(y) = 0 \\
&\implies x = \nabla f(y) \\
&\implies y = (\nabla f)^{-1}(x).
\end{aligned}$$

Thus

$$f^*(x) = \langle x, (\nabla f)^{-1}(x) \rangle - f((\nabla f)^{-1}(x)). \tag{3.2}$$

A crucial property of the conjugate function is that its gradient is an inverse to the gradient of the original function. To keep notation clean, define $h(x) \equiv \nabla f(x)$. Then, following the above derivation, we have

$$\begin{aligned}
f^*(x) &= \langle x, h^{-1}(x) \rangle - f(h^{-1}(x)); \text{ implying that} \\
\nabla f^*(x) &= h^{-1}(x) + \nabla h^{-1}(x) \cdot x - \nabla h^{-1}(x) \nabla f(h^{-1}(x)) \\
&= h^{-1}(x) \quad (\text{since } \nabla f(x) = h(x)).
\end{aligned}$$

Thus ∇f and ∇f^* are inverses of one-another.

Let $x' \equiv \nabla f(x)$ and $y' \equiv \nabla f(y)$. Then plugging x' and y' into (3.2), we get the convenient relations

$$\begin{aligned}
f^*(x') &= \langle x', x \rangle - f(x) \quad \text{and} \\
f^*(y') &= \langle y', y \rangle - f(y).
\end{aligned}$$

Let us now consider the bregman divergence associated with f in light of the conjugate function. We show that $d_{f^*}(y', x') = d_f(x, y)$.

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

This equality is useful because it permits algorithms designed to optimize over the first argument to work over the second argument. It provides an elegant way to sidestep the asymmetry of bregman divergences. See section 5.2 for an example.

Another easily derived equality is

$$d_f(x, y) = f(x) + f^*(y') - \langle x, y' \rangle. \tag{3.3}$$

This equality reveals another interpretation interpretation of a bregman divergence: it is a measurement of the slackness in the Fenchel-Young inequality.

3.3 Centroidal property

Bregman divergences satisfy a centroidal property that makes them amenable to k-means style clustering [BMDG05]. This property states that the average divergence from a point to a set can be decomposed as the divergence from the point to the set's centroid plus the average divergence from the set to the centroid.

Theorem 3. *Let $S \subset \mathbb{R}^d$ be a set of size n and let $d_f(S, x)$ denote $\sum_{s \in S} d_f(s, x)$. Let $\mu = \frac{1}{n} \sum_{s \in S} s$ be the mean of S. Then for all $x \in \mathbb{R}^d$,*

$$d_f(S, x) = d_f(S, \mu) + n d_f(\mu, x).$$

Proof. Expanding the right side, we have

$$\begin{aligned}
&d_f(S, \mu) + n \cdot d_f(\mu, x) \\
&\quad = \sum_s f(s) - n f(\mu) - \sum_s \langle \nabla f(\mu), s - \mu \rangle + n f(\mu) - n f(x) - n \langle \nabla f(x), \mu - x \rangle \\
&\quad = \sum_s (f(s) - f(x)) - \sum_s \langle \nabla f(\mu), s - \mu \rangle - n \langle \nabla f(x), \mu - x \rangle \\
&\quad = \sum_s (f(s) - f(x)) - \left\langle \nabla f(\mu), \sum_s s - n\mu \right\rangle - \langle \nabla f(x), n\mu - nx \rangle \\
&\quad = \sum_s (f(s) - f(x)) - \left\langle \nabla f(x), \sum_s (s - x) \right\rangle \\
&\quad = \sum_s (f(s) - f(x) - \langle \nabla f(x), s - x \rangle) \\
&\quad = d_f(S, x).
\end{aligned}$$

□

3.4 Composability

One convenient property of bregman divergences is that they can by composed. Suppose that $\{f_i\}$ are strictly convex functions and $\{\alpha_i\}$ is a set of non-negative numbers. Then $g \equiv \sum \alpha_i f_i$ is also a strictly convex function. Thus we can define a bregman divergence based on g

$$\begin{aligned} d_g(x,y) &= \sum \alpha_i f_i(x) - \sum \alpha_i f_i(y) - \langle \nabla \left[\sum \alpha_i f_i(y) \right], x - y \rangle \\ &= \sum \alpha_i d_{f_i}(x,y). \end{aligned}$$

Similarly, one can define a D-dimensional multivariate bregman by summing D one-dimensional bregman divergences coordinate-wise. These properties are useful for working with mixed-type data. For example, a point x might have a histogram component of, say, the first 10 coordinates, and a vector coordinate of the next 5 coordinates. A reasonable way to compute the distance between two such points would be to measure the KL-divergence between the first 10 coordinates, and the ℓ_2^2 distance between the last 5. Any machinery defined for arbitrary bregman divergences, such as the fast NN algorithms developed in this dissertation, will be able to handle this mixed-type divergence as it is itself a bregman divergence.

3.5 Additional geometric facts

In the proof of theorem 1, we showed that the difference $G(x) = d_f(x,y) - d_f(x,z)$ is linear in x:

$$\begin{aligned} G(x) &= f(x) - f(y) - \langle \nabla f(y), x - y \rangle - (f(x) - f(z) - \langle f(z), x - z \rangle) \\ &= f(z) + \langle \nabla f(z), x - z \rangle - f(y) - \langle \nabla f(y), x - y \rangle. \end{aligned}$$

In particular, this means that the set of points equidistant from y and z lie on a hyperplane.

Theorem 4. *The set of points $\{x : d_f(x,y) = d_f(x,z)\}$ is an affine set.*

Bregman divergences do not in general satisfy the triangle inequality, but do satisfy the three point property

$$d_f(x,y) + d_f(y,z) = d_f(x,z) + \langle \nabla f(z) - \nabla f(y), x - y \rangle.$$

Finally, symmetrized bregman divergences have a particularly simple form:

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

3.6 Exponential families[1]

There is a remarkable connection between bregman divergences and exponential families. We sketch the basic details in this section.

Let ν be a σ-finite measure defined on the Borel subsets of $\mathbb{R}^D$. An exponential family consists of all distributions with probability density functions of the form

$$p_\eta(x) = \exp(\langle \eta, T(x) \rangle - G(\eta))\ \nu(x),$$

where $T(x)$ is a sufficient statistic. We focus on the case where $T(x) = x$; these are called *regular* exponential families. The set of all η such that

$$\int e^{\langle \eta, x \rangle} \nu(dx) < \infty$$

is called the *natural parameter space*. The function $G(\cdot)$ normalizes the probability distribution and is called the *log partition function*. It is equal to

$$G(\eta) = \log \int e^{\langle \eta, x \rangle} \nu(dx).$$

Many standard distributions are examples of regular exponential families. One example is the Bernoulli distribution, with pdf

$$p(x) = \begin{cases} \mu & \text{for } x = 1 \\ 1 - \mu & \text{for } x = 0 \end{cases} = \mu^x (1 - \mu)^{1-x}.$$

[1] The presentation of the material in this section is based partly on [BMDG05] and on a series of lectures given by Sanjoy Dasgupta in the fall of 2004.

Let $\eta = \log \frac{\mu}{1-\mu}$. Then

$$p(x) = \exp(x \cdot \eta - \log(1 + e^{\eta})),$$

so is an exponential family with log partition function $\log(1 + e^{\eta})$. Notice that the natural parameter space is all of $\mathbb{R}$, whereas the expectation μ is confined to $[0, 1]$.

Another example is the univariate Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, with pdf

$$p(\omega) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(\omega-\mu)^2/2\sigma^2}.$$

Re-parametrizing with $x \equiv (\omega, \omega^2)$ and $\eta \equiv (\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2})$ puts $p(x)$ into standard exponential form.

A third example is the Poisson distribution, with pdf

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!},$$

which is defined over $\mathcal{N}$ for $\lambda \in \mathbb{R}_+$. Setting $\nu(x) = \frac{1}{x!}$ and $\eta = \log(\lambda)$ recovers the familiar exponential form.

There are many other examples, including the binomial and exponential distributions, multidimensional Gaussian distribution, and the multinomial distribution.

The log-partition function has many useful properties. Here we need only a couple of essential ones. First, the gradient of $G(\eta)$ is the mean of the corresponding exponential distribution:

$$\nabla G(\eta) = \int x p_\eta(x) \nu(dx).$$

Additionally, the Hessian of $G(\eta)$ is equal to covariance matrix of p_η. Because the gradient is equal to the mean, the gradient can be thought of as providing a link between the natural parameter space and the expectation parameter space. Another fact of the log-partition function is that it is strictly convex (assuming p_η is minimal), and in fact is Legendre [BMDG05]. Thus the convex conjugate is defined, and $\nabla G^*(y)$ provides the inverse mapping from the space of expectation parameters to the space of natural parameters.

Suppose we wish to measure how similar two members of an exponential family are. A natural way to do this is to compute the KL-divergence between

them, and this is where the connection to bregman divergences becomes apparent. Let $p(x)$ and $q(x)$ be members of an exponential family with natural parameters η_1 and η_2 respectively. Then the KL-divergence between them is

$$\begin{aligned}
\int p(x) \log \frac{p(x)}{q(x)} dx &= \int p(x) \log \frac{\exp(\langle \eta_1, x \rangle - G(\eta_1))}{\exp(\langle \eta_2, x \rangle - G(\eta_2))} dx \\
&= \int p(x)(\langle \eta_1 - \eta_2, x \rangle + G(\eta_2) - G(\eta_1)) dx \\
&= \langle \nabla G(\eta_1), \eta_1 - \eta_2 \rangle + G(\eta_2) - G(\eta_1), \quad \text{since } \nabla G(\eta) \text{ is the mean} \\
&= d_G(\eta_2, \eta_1).
\end{aligned}$$

Thus the entropy between two members of an exponential family is the bregman divergence based on the log-partition function between their natural parameters. Moreover, we can write the pdf of any regular exponential family in terms of a bregman divergence:

$$\begin{aligned}
p_\eta(x) &= \exp(\langle x, \eta \rangle - G(\eta))\nu(x) \\
&= \exp(G^*(\eta') + \langle x - \eta', \eta \rangle - G^*(x) + G^*(x))\nu(x) \\
&= \exp(-d_{G^*}(x, \eta') + G^*(x))\nu(x).
\end{aligned}$$

Since η' is the mean of p_η, this equality shows that the pdf is a function of the divergence between x and the mean. Similarly,

$$p_\eta(x) = \exp(-d_G(\eta, x') + G^*(x))\nu(x),$$

where $x' \equiv \nabla G^*(x)$.

The connection between bregman divergences and exponential families goes a bit deeper. Here, we showed that for exponential family, there is a related bregman divergence. In [BMDG05], it is shown that there is a one-to-one correspondence between the regular exponential families and the so-called regular bregman divergences—bregman divergences defined over a restricted class of convex base functions.

3.7 Work related to bregman proximity search

This dissertation presents the first algorithms and data structures for bregman proximity search, but there has been some related work. [NBN07] explores the geometric properties of bregman voronoi diagrams. Voronoi diagrams are related to NN search, but do not lead to practical algorithms. [GIM07] 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 [SV05]. 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*.

Chapter 4

Bregman ball trees

In this chapter, we define the bregman ball tree (bb-tree) data structure. This is the first data structure for proximity problems for arbitrary bregman divergences. It defines a hierarchical space decomposition based on *bregman balls*, which are simple convex bodies. The reason for the use of these balls as the basis for the data structure is that we can compute bounds on the minimum and maximum divergence from a (query) point to a bregman ball very efficiently, as we show in this chapter.

This chapter describes the basics of the data structure, an algorithm for building it, and two optimization problems that are essential for using bb-trees. In later chapters, we will describe performing nearest neighbor search and range search using the bregman ball tree.

Throughout this chapter, $X = \{x_1, \ldots, x_n\}$ denotes the database, d_f notes some fixed bregman divergence, D denotes the dimensionality of the space, and n denotes the number of points in the database.

4.1 Structure

The bregman ball tree has a structure similar to the classical branch-and-bound data structure. It is a space partition that is organized into a tree. The challenge is that we must choose cells for which it will be easy to compute bounds

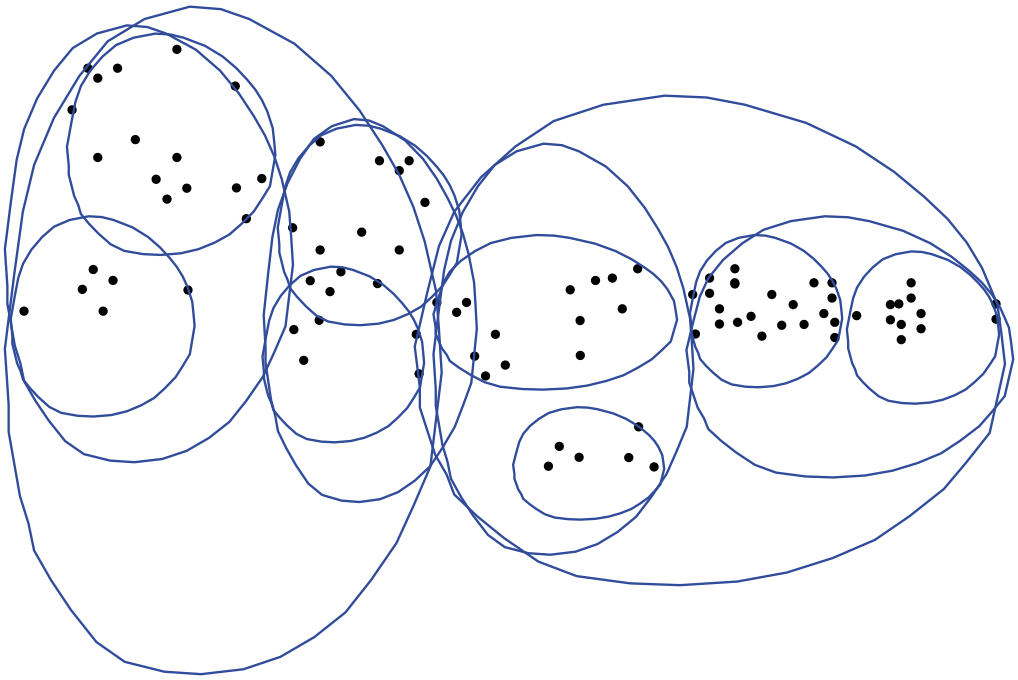


Figure 4.1: Example bb-tree.

on the divergence to. The cell type we choose is a bregman ball, defined as

$$B_f(\mu, R) \equiv \{x \mid d_f(x, \mu) \leq R\}.$$

Since Bregman divergences are convex in their first argument, bregman balls are convex bodies. We show later that we can efficiently bound the minimum and maximum divergence from a point to these types of cells.

A bb-tree 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 $B_f(\mu_i, R_i)$ with center μ_i and radius R_i such that $X_i \subset B_f(\mu_i, R_i)$. 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.[1] Though X_l and X_r are disjoint, the balls $B_f(\mu_l, R_l)$ and $B_f(\mu_r, R_r)$ may overlap. The root node of the tree encapsulates the entire database. Each leaf covers a

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

small fraction of the database; the set of all leaves cover the entirety. Figure 4.1 shows an example of a bregman ball tree.

4.2 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 [BMDG05].

The build algorithm proceeds from top down. Starting at the top, the algorithm runs k-means (with k=2) to partition the points into two clusters. This process is repeated recursively. If we assert that each node contains at least a fixed fraction of the points of its parent, and that k-means can be executed in linear time, then 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.

We note that the build algorithm is a reasonable heuristic that works fairly well in practice, but which can almost certainly be improved. To compare, there are many different methods for building KD-trees and metric trees, *e.g.* [AMN+98, Moo00]. Indeed, since the publication of some of the present work, other authors' have reported on more complex building algorithms for bregman ball trees, with some positive results [NPB09].

4.3 Minimization

The proximity search algorithms that make use of the bregman ball tree will need to efficiently lower bound the divergence from a query point to a bregman ball

(see chapter 5 for details). Here we study the core projection problem, developing the relevant bregman geometry and a very efficient algorithm. Some readers may prefer to skip ahead to chapters 5 and 6 to motivate the technical developments in the remainder of this chapter.

We are interested in solving the following convex program efficiently.

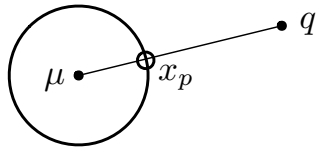
$$\begin{aligned} \min_{x} \quad & d_f(x, q) \\ \text{subject to:} \quad & d_f(x, \mu) \leq R. \end{aligned} \tag{minP}$$

We will need to be able to find a solution to this program extremely quickly, as such programs will be evaluated repeatedly in our nearest neighbor search algorithms. In particular, we will not be able to merely call some general purpose solver; we will need to find the solution in roughly the same amount of time as required for evaluating a D-dimensional analytic expression.

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 $\theta = 1 - \frac{\sqrt{2R}}{\|q-\mu\|}$.



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 $B_{\ell_2^2}(\mu, R)$—*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 $B_{\ell_2^2}(\mu, R)$.

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 (minP) 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 (minP).

Claim 1. *x_p' lies on the line between q' and μ'.*

Proof. The lagrangian of (minP) is

$$\inf_x d_f(x, q) + \lambda(d_f(x, \mu) - R), \tag{4.1}$$

where $\lambda \geq 0$. Differentiating (4.1) 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

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

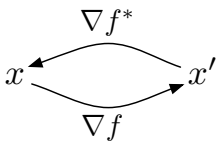
where $\theta \in [0, 1)$. □

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 1 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, which again is defined as

$$f^*(y) \equiv \sup_x \{\langle x, y \rangle - f(x)\}. \tag{4.2}$$

Symbolically:



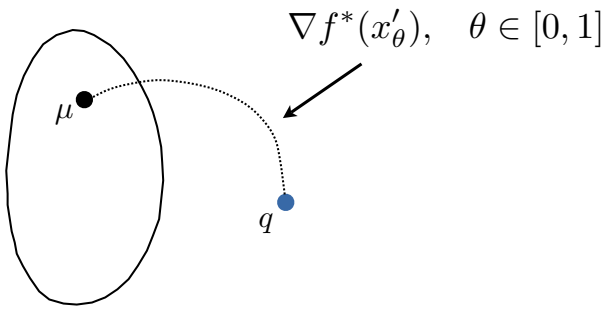


Figure 4.2: Illustration of Claim 1.

Refer back to section 3.2 for a review of the conjugate.

Thus to solve (minP), we can look for the optimal x' along $\theta\mu' + (1-\theta)q'$, and then apply ∇f^* to recover x_p.[2] To keep notation simple, we define

$$x'_\theta \equiv \theta\mu' + (1-\theta)q' \quad \text{and} \tag{4.3}$$

$$x_\theta \equiv \nabla f^*(x'_\theta). \tag{4.4}$$

Figure 4.2 illustrates Claim 1.

Now onto the second property.

Claim 2. *$d_f(x_p, \mu) = R$—i.e. the projection lies on the boundary of $B_f(\mu, R)$ (assuming that $q \notin B_f(\mu, R)$).*

Proof. Consider again the lagrangian of (minP):

$$\mathcal{L}(x, \lambda) = d_f(x, q) + \lambda(d_f(x, \mu) - R).$$

By claim 1, any optimal solution x_p can be written as

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

where $\theta^* = \frac{1}{1+\lambda^*}$ and λ^* is optimal for the dual. The complementary slackness condition is

$$\lambda^*(d_f(x_p, \mu) - R) = 0,$$

[2] All of the base functions in table 3.1 have closed form conjugates.

so either $\lambda^* = 0$ or $d_f(x_p, \mu) = R$. $\lambda^* = 0$ implies that $\theta^* = 1$; but then $x_p = q \notin B_f(\mu, R)$, so x_p is not feasible. We conclude that $d_f(x_p, \mu) = R$. □

Claims 1 and 2 imply that finding the projection of q onto $B_f(\mu, R)$ 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 3. *$d_f(x_\theta, \mu)$ is monotonic in θ.*

Proof.

$$\begin{aligned} \frac{d}{d\theta}\left[d_f(x_\theta, \mu)\right] &= \frac{d}{d\theta}\left[f(x_\theta) - \langle \mu', x_\theta \rangle\right] \\ &= \nabla f(x_\theta)^\top [\nabla^2 f^*(x'_\theta)](q' - \mu') - \mu'^\top [\nabla^2 f^*(x'_\theta)](q' - \mu') \\ &= (x'_\theta - \mu')^\top [\nabla^2 f^*(x'_\theta)](q' - \mu') \\ &= (1-\theta)(q' - \mu')^\top [\nabla^2 f^*(x'_\theta)](q' - \mu') \\ &\geq 0 \end{aligned}$$

since $\theta \in [0, 1]$ and f^* is convex. □

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*

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

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

4.4 Approximate minimization

Suppose we wish only to evaluate an inequality of the form

$$\min_{x \in B_f(\mu_j, R_j)} d_f(x, q) < C, \tag{4.5}$$

where C is some constant. We can solve the projection exactly and then check the inequality, but it may be much quicker to compute a series of approximations to the projection, and check the bound each time. The hope is that in many cases the algorithm will require many fewer iterations than if the exact projection is computed. We are thus interested in computing bounds

$$a \leq \min_{x \in B_f(\mu_j, R_j)} d_f(x, q) \leq A.$$

If $A < C$ (C as defined in (4.5)), then (4.5) holds. Alternatively, if $a > C$, then (4.5) fails. Either way, the exact solution need not be computed.

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

$$\mathcal{L}(\theta) \equiv d_f(x_\theta, q) + \frac{\theta}{1-\theta}\Big(d_f(x_\theta, \mu) - R\Big). \tag{4.6}$$

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

$$\mathcal{L}(\theta) \leq \min_{x \in B_f(\mu, R)} d_f(x, q). \tag{4.7}$$

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

$$d_f(x_\theta, q) \geq \min_{x \in B_f(\mu, R)} d_f(x, q). \tag{4.8}$$

Let us now put all of the pieces together. We wish to evaluate whether (4.5) 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 (4.6). If $\mathcal{L}(\theta_i) > C$, then the node can be pruned. Otherwise, if $x_{\theta_i} \in B_f(\mu, R)$, we can update the upper bound. If $d_f(x_{\theta_i}, q) < C$, then the node must be searched. Otherwise, neither bound holds, so the bisection search continues.

We note that the algorithm in this section does not improve the theoretical time bound in 5, but seemed to greatly improve the performance in practice.

4.5 Maximization

In this section, we consider the following problem

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

Unlike the minimization problem analyzed in section 4.3, this problem is not convex (nor is it concave). Nevertheless, we will need to be able to compute a solution to it—or at least bound it—for the range search algorithm presented in chapter 6. In general, non-convex problems are much more challenging to solve than convex ones. Interestingly, this particular non-convex problem contains a special structure that allows for efficient computation, as we show in this section. It will turn out that by simply extending the dual curve that was defined for the *minimization* problem (see the previous section), we can solve the maximization problem; see figure 4.3.

For the analysis below we need to ensure that the interior of the Bregman ball in question is not intersected by the domain boundary. For instance, in the generalized KL-divergence case, a KL-ball could be cut by the $x_i = 0$ hyperplane. This technicality might arise in practice, so we detail a solution for the KL-divergence in section 4.6.

Claim 4. *Suppose that the domain of f is C and that $B_f(\mu, R) \subset \text{relint}(C)$. Furthermore, assume that $\|\nabla^2 f^*(x')\|$ is lower-bounded for all x' such that $x \in B_f(\mu, R)$. Let x_p be the optimal solution to (maxP). Then x'_p lies on the dual curve $\{\theta\mu' + (1-\theta)q' \mid \theta \geq 0\}$.*

Proof. Though the program is not concave, the Lagrange dual still provides an upper bound on the optimal solution value (by weak duality). The Lagrangian is

$$\nu(x, \lambda) \equiv \; d_f(x, q) - \lambda(d_f(x, \mu) - R), \tag{4.9}$$

where $\lambda \geq 0$.

Differentiating (4.9) with respect to x and setting it equal to 0, we get

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

which implies that

$$\nabla f(x_p) = \frac{1}{1-\lambda}\left(\nabla f(q) - \lambda \nabla f(\mu)\right). \tag{4.10}$$

We need to check what type of extrema $\nabla f(x_p) = 0$ is:

$$\nabla_x^2 \nu(x, \lambda) = (1-\lambda)\nabla^2 f(x).$$

Thus for $\lambda > 1$, the x_p defined implicitly in (4.10) is a maximum. Setting $\theta \equiv -\frac{\lambda}{1-\lambda}$ gives

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

where $\theta \in (-\infty, 0) \cup (1, \infty)$; we restrict attention to $\theta \in (1, \infty)$ since that is where $\lambda > 1$ and hence x_p is a maximum. Let $x'_\theta \equiv \theta\mu' + (1-\theta)q'$ and $x_\theta \equiv \nabla f^*(x'_\theta)$. The Lagrange dual is

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

Then for any $\theta \in (1, \infty)$, we have

$$d_f(x_p, q) \leq \mathcal{L}(\theta) \tag{4.11}$$

by weak duality. We now show that there is a $\theta^* > 1$ satisfying $d_f(x_{\theta^*}, \mu) = R$. One can check that the derivative of $d_f(x_\theta, \mu)$ with respect to θ is

$$(\theta - 1)(\mu' - q')^\top \nabla^2 f^*(x'_\theta)(\mu' - q') \tag{4.12}$$

Since $\|\nabla^2 f^*\| > c$, for some positive c, (4.12) is at least $(\theta - 1)c$. We conclude that $d_f(x_\theta, \mu)$ is increasing at an increasing rate with θ. Thus there must be some $\theta^* > 1$ such that $d_f(x_{\theta^*}, \mu) = R$. Plugging this θ^* into the dual, we get

$$\begin{aligned}\mathcal{L}(\theta^*) &= d_f(x_{\theta^*}, q) + \frac{\theta^*}{1-\theta^*}(d_f(x_{\theta^*}, \mu) - R) \\ &= d_f(x_{\theta^*}, q).\end{aligned}$$

Combining with (4.11), we have

$$d_f(x_p, q) \leq d_f(x_{\theta^*}, \mu).$$

Finally, since (maxP) is a maximization problem and since x_{θ^*} is feasible, the previous inequality is actually an equality, giving the theorem. □

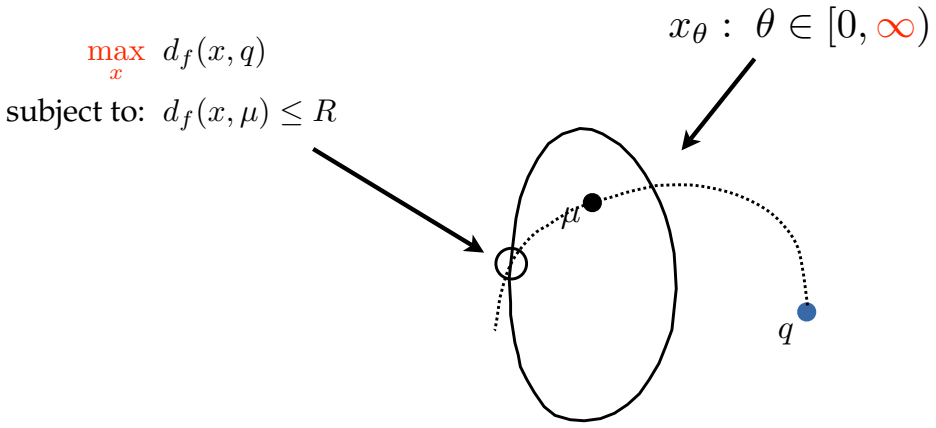


Figure 4.3: Illustration of Claim 4.

Again, figure 4.3 illustrates Claim 4. We note that the assumption of Claim 4 that $B_f(\mu, R) \subset \text{relint}(C)$ can actually be relaxed to the assumption that there exists a $\theta > 1$ such that $d_f(x_\theta, \mu) = R$. Put differently we only need to assume that the dual curve will reach the boundary of the Bregman ball.

4.6 Particulars for the KL-divergence

Using the bregman ball tree with the KL-divergence requires managing a few details and also provides some extra opportunities for speedup. Since the KL-divergence was the main motivation for developing the bb-tree, these details are worth explicating.

4.6.1 The normalized KL-divergence

Suppose we are interested in the case where all vectors in the database are histograms—*i.e.* for all $x \in X$, each $x_i \in [0, 1]$ and $\sum x_i = 1$. After building a bb-tree on X, using it will require one to compute solutions to the minimization and maximization problems described previously. Let us consider the minimization

problem

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

This is the standard minimization program that will be used extensively in nearest neighbor and range searching. Note that though all vectors in the database are histograms, there is no constraint that the projection onto the bregman ball in (minP) yields a histogram. Indeed, the bregman ball will contain many points that are not histograms, so, in this sense, the bb-tree is throwing away some of the structure of the database. In practice, this means that the lower bounds provided by solving (minP) will be much looser than they need to be. To tighten these bounds, we add the additional constraint that the point found sums to one:

$$\begin{aligned} \min_{x} \quad & d_f(x, q) \\ \text{subject to: } & d_f(x, \mu) \leq R, \\ & \sum_i x_i = 1. \end{aligned} \tag{minP2}$$

The Lagrangian of this program is

$$\nu(x, \lambda, \eta) \equiv d_f(x, q) + \lambda\big(d_f(x, \mu) - R\big) + \eta\left(\sum x_i - 1\right).$$

Differentiating with respect to x, we get

$$\nabla_x \nu = \nabla f(x) - \nabla f(q) + \lambda \nabla f(x) - \lambda \nabla f(\mu) + \eta \cdot \mathbf{1}.$$

Setting the gradient equal to zero and rearranging gives

$$\begin{aligned} \nabla f(x_p) &= \frac{1}{1+\lambda}\left(\nabla f(q) + \lambda f(\mu) - \eta \cdot \mathbf{1}\right) \\ &= \theta\mu' + (1-\theta)q' - (1-\theta)\eta \cdot \mathbf{1}, \text{ where } \theta \equiv \frac{\lambda}{1+\lambda}. \end{aligned}$$

In the case of the KL-divergence, we have

$$x_p = \mu^{\theta} q^{1-\theta} e^{-(1-\theta)\eta \cdot \mathbf{1} - \mathbf{1}}.$$

Applying the normalization constraint, we get

$$\begin{aligned} & \sum \mu_i^\theta q_i^{1-\theta} e^{-(1-\theta)\eta - 1} = 1 \\ \Longrightarrow \quad & e^{(1-\theta)\eta+1} = \sum \mu_i^\theta q_i^{1-\theta} \\ \Longrightarrow \quad & \eta = \frac{1}{1-\theta}\left(\log\left(\sum \mu_i^\theta q_i^{1-\theta}\right) - 1\right). \end{aligned}$$

Substituting this back into x_p, we see that

$$x_p = \mu^\theta q^{1-\theta} / \sum \mu_i^\theta q_i^{1-\theta}.$$

As a result, we can apply the same binary search algorithm described previously for solving (minP) to solve (minP2), except that at each evaluation of θ, one needs to normalize x_θ. Thus, though (minP2) yields tighter bounds and has an extra constraint, it is no harder to solve than (minP). A similar derivation applies to the maximization problem.

4.6.2 The generalized KL-divergence

We now consider an issue with the generalized KL-divergence, defined on all of $\mathbb{R}_+^D$. Suppose we wish to compute

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

where

$$d_f(x, y) \equiv \sum x_i \log \frac{x_i}{y_i} - x_i + y_i$$

One catch to implementing the simple dual-line search approach described abstractly before is that the KL-divergence is only defined on $\mathbb{R}_+^D$, which means that KL-ball might actually intersect the boundary. When this occurs, the line-search approach is not valid; the furthest point might be on the boundary of the domain, and not necessarily on the dual curve. Fortunately, we can easily check if the line search will fail. To do so, it suffices to evaluate whether $d_f(x_\theta, \mu) \geq R$ for $\theta \to \infty$. If $d_f(x_\theta, \mu) < R$ for arbitrarily large θ, the line search can fail. In practice, this

was not a significant problem; when it arose, we simply proceeded as if the node could not be pruned.

Portions of this chapter have been published or submitted:

- L. Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the International Conference on Machine Learning*, 2008.
- L. Cayton. Efficient bregman range search. *Submitted,* 2009.

Chapter 5

Bregman nearest neighbor search

In this chapter, we describe how to use the bregman ball tree for nearest neighbor search and provide experimental results for the KL-divergence. At present, this is the only known scheme for efficient NN search for an arbitrary bregman divergence and for the special case of the KL-divergence.

Throughout this chapter, we denote the database by $X = \{x_1, \ldots, x_n\}$, a query by q, and we assume that $d_f(\cdot, \cdot)$ is some fixed bregman divergence. The search algorithm described is for finding the *left* nearest neighbor

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

The data structure can be used for the *right* nearest neighbor problem also as we describe in section 5.2.

5.1 Basic algorithm

This section describes how to retrieve a query q's nearest neighbor with a bb-tree.

Branch and bound search locates the NN in the bb-tree. 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

$$d_f(x_c, q) > \min_{x \in B_f(\mu_j, R_j)} d_f(x, q). \tag{5.1}$$

The algorithm computes the right side of (5.1); we come back that in a moment. If (5.1) 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 (5.1)—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 (5.1). An algorithm for computing this projection was discussed previously in section 4.3. That algorithm can compute the projection (to accuracy ϵ) in time $O(D \log(1/\epsilon))$, which is not much worse than the time to evaluate a D-dimensional analytic expression. Moreover, we do not need to compute the exact projection, we merely need to check if the inequality (5.1) holds. Thus we can use the approximate projection technique described in section 4.4. We specialize the approximate projection technique to this case; see Algorithm 3 for pseudocode.

5.1.1 Approximate search

As we mentioned in chapter 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 [LMGY04], a simple way to speed up the retrieval time of the bb-tree 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 back-

Algorithm 3 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 $x_\theta \in B_f(\mu, R)$ 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

tracking. 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.

5.2 Left and right nearest neighbor

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 bb-tree data structure finds the left NN . We show that it can also be used to find the right NN.

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 bb-tree to the rNN problem: build a bb-tree 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 bb-tree finds $x' \in X'$ minimizing $d_{f^*}(x', q')$. The point x whose gradient is x' is then the rNN to q.

5.3 Experiments

We examine the performance benefit of using bb-trees for approximate and exact NN search. All experiments were conducted with a simple C implementation that is available for download.

The results are for the KL-divergence. We chose to evaluate the bb-tree 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 bb-trees 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) [BNJ03] to generate topic histograms for 500k documents in the rcv1 corpus [LYRL04]. 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

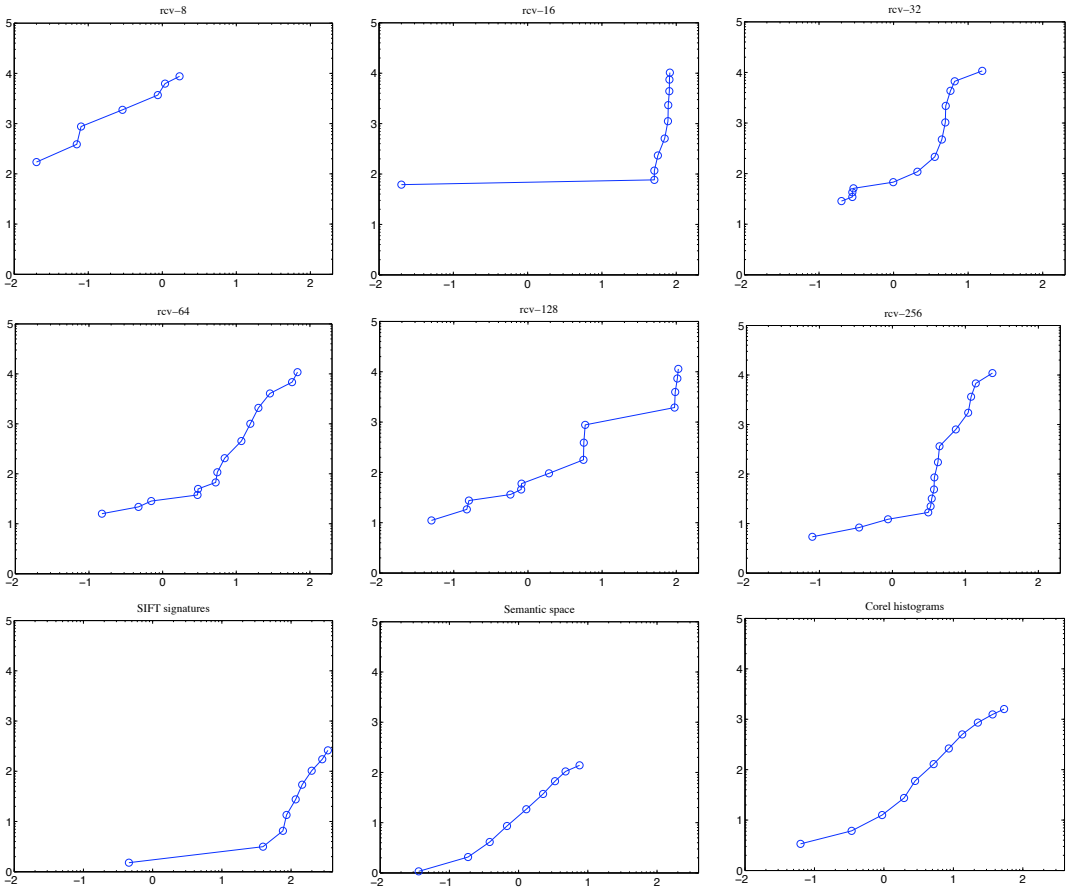


Figure 5.1: Log-log plots (base 10): y-axis is the exponent of the speedup over brute force search, x-axis is the exponent of the number of database points closer to the query than the reported NN. The y-axis ranges from 10^0 (no speedup) to 10^5. The x-axis ranges from 10^{-2} to 10^2. All results are averages over queries not in the database.

Consider the plot for rcv-128 (center). At $x = 10^0$, the bb-tree 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 bb-tree 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 one percent.

the documents in the topic simplex [BNJ03]. 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 [RMV07].

- **SIFT signatures**. This dataset contains 1111-dimensional representations of 10k images from the PASCAL 2007 dataset [EGW+07]. Each point is a histogram of quantized SIFT features as suggested in [NJT06].

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 bb-tree 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).[1]

The results are shown in figure 5.1. These are strong results; it is shown that the bb-tree is often orders of magnitude faster than brute-force search without a substantial degradation of quality. More analysis appears in the caption.

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

[1] 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.

Table 5.1: 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

speedup in moderate dimensions, but the speedup diminishes with increasing dimensionality [Moo00, LMGY04]. When used for exact search, the bb-tree reflects this basic pattern. Table 5.1 shows the results. The bb-tree provides a substantial speedup on the moderate-dimensional databases (up through $D = 256$), but no speedup on the two databases of highest dimensionality.

The results of this chapter were published in:

L. Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the International Conference on Machine Learning*, 2008.

Chapter 6

Bregman range search

In this chapter, we describe how to perform range search with a bregman ball tree. Again, the database is denoted by $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^D$, and d_f denotes a fixed bregman divergence. On query q with radius γ, we wish to return all database points $x \in X$ such that $d_f(x, q) \leq \gamma$. We note that the query is the right argument to the bregman divergence. If we wish to use the query as the left argument, the technique discussed in section 5.2 can be used.

6.1 Search algorithm

We now describe the search algorithm, which uses the branch-and-bound approach outlined in chapter 2. Computing bounds on an arbitrary bregman divergence requires novel bounding techniques based on geometric properties that we develop in the next section.

Suppose we are interested in returning all points within distance γ of a query q—*i.e.* we hope to retrieve all database points lying inside of $B_q \equiv B_f(q, \gamma)$. The search algorithm starts at the root node and recursively explores the tree. At a node i, the algorithm compares the node's bregman ball B_x to B_q. There are three possible situations, shown visually in figure 6.1. First, if B_x is contained in B_q, then all $x \in B_x$ are in the range of interest. We can thus stop the recursion and return all the points associated with the node without explicitly computing the divergence to any of them. This type of pruning is called *inclusion pruning*.

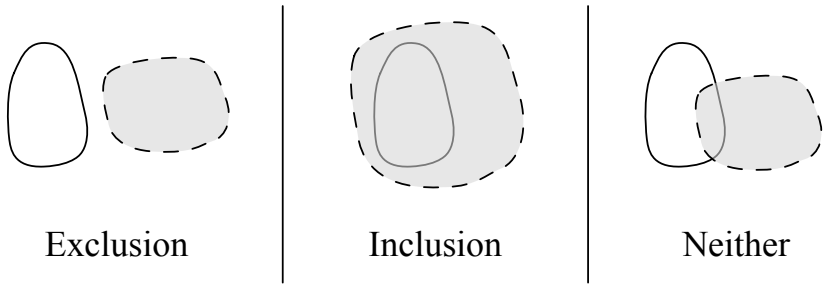


Figure 6.1: The three pruning scenarios. The dotted, shaded object is the query range and the other is the bregman ball associated with a node of the bb-tree.

Second, if $B_x \cap B_q = \emptyset$, the algorithm can prune out B_x and stop the recursion; none of these points are in range. This is *exclusion pruning*. All performance gains from using the algorithm come from these two types of pruning. The third situation is $B_x \cap B_q \neq \emptyset$ and $B_x \not\subset B_q$. In this situation, the algorithm cannot perform any pruning, so recurses on the children of node i. If i is a leaf node, then the algorithm computes the divergence to each database point associated with i and returns those elements within range.

The two types of pruning—inclusion and exclusion—have been applied to a variety of problems with metric and kd-trees, see *e.g.* [GM00, GM03, SNS06]. Thus though we focus on range search, we expect that this type of pruning will be useful in a broad range of problems.

As described in chapter 2, it has been widely observed that the performance of spatial decomposition data structures degrades with increasing dimensionality. In order to manage high-dimensional datasets, practitioners often use *approximate* proximity search techniques [DIIM04, LMG06, Cay08]. In the experiments, we explore one way to use the bb-tree in an approximate fashion.

Determining whether two bregman balls intersect, or whether one bregman ball contains another, is non-trivial. For the range search algorithm to be effective, it must be able to determine these relationships very quickly. In the case of metric balls, these determinations are trivially accomplished using the triangle inequality. Since we cannot rely on the triangle inequality for an arbitrary bregman divergence,

we must develop novel techniques.

6.2 Inclusion

In this subsection we derive an algorithm for determining if one bregman ball is a subset of another. Let $B_q \equiv B_f(\mu_q, R_q)$ and $B_x \equiv B_f(\mu_x, R_x)$. We wish to evaluate if $B_x \subset B_q$. This problem is equivalent to testing whether

$$d_f(x, \mu_q) \leq R_q \tag{6.1}$$

for all $x \in B_x$. One way to evaluate whether 6.1 holds is to find the $x \in B_x$ that is farthest from μ_q and compare the divergence between them to R_q. That is, we wish to solve

$$\begin{aligned} \max_x \quad & d_f(x, \mu_q) \\ \text{subject to: } & d_f(x, \mu_x) \leq R_x. \end{aligned} \tag{maxP}$$

We described an algorithm for solving this optimization problem in chapter 4.

6.3 Exclusion

We now demonstrate how to evaluate whether $B_q \cap B_x = \emptyset$. Interestingly, the solution again comes down to looking at the dual curve between two points. We will need to make use of the Pythagorean theorem, proved in chapter 3. It is restated here for convenience.

Theorem 6 (Pythagorean)**.** *Let $C \subset \mathbb{R}^D$ be a convex set and let $x \in C$. Then for all z, we have*

$$d_f(x, z) \geq d_f(x, y) + d_f(y, z),$$

where $y \equiv \text{argmin}_{y \in C} d_f(y, z)$ is the projection of z onto C.

At first glance, the Pythagorean theorem may appear to be a triangle inequality for bregman divergences. However, the inequality is actually the *reverse*

of the standard triangle inequality. Moreover, the theorem only applies to the very special case when y is the projection of z onto a convex set containing x. We now prove the main claim of this section.

Claim 5. *Suppose that $B_x \cap B_q \neq \emptyset$. Then there exists a w in the set*

$$\{\nabla f^*(\theta\mu'_x + (1-\theta)\mu'_q) \mid \theta \in [0,1]\}$$

such that $w \in B_q \cap B_x$.

Proof. Let $z \in B_x \cap B_q$. Define the dual curve

$$\ell = \{\nabla f^*(\theta\mu'_x + (1-\theta)\mu'_q) \mid \theta \in [0,1]\}.$$

Let x be the projection of μ_q onto B_x and let q be the projection of μ_x onto B_q. Both x and q are on the dual curve, so we are done if we can show that at least one of them lies in the intersection of B_x and B_q. Suppose towards contradiction that neither are in the intersection.

The projection of x onto B_q lies on the dual curve between x and μ_q; thus projecting x onto B_q yields q and similarly projecting q onto B_x yields x. By the Pythagorean theorem,

$$d_f(z,x) \geq d_f(z,q) + d_f(q,x), \tag{6.2}$$

since q is the projection of x onto B_q and since $z \in B_q$. Similarly,

$$d_f(z,q) \geq d_f(z,x) + d_f(x,q). \tag{6.3}$$

Inserting (6.2) into (6.3), we get

$$d_f(z,q) \geq d_f(z,q) + d_f(q,x) + d_f(x,q).$$

Rearranging, we get that $d_f(q,x) + d_f(x,q) \leq 0$. Thus both $d_f(q,x) = 0$ and $d_f(x,q) = 0$, implying that $x = q$. But since $x \in B_x$ and $q \in B_q$, we have that $x = q \in B_q \cap B_q$. This is the desired contradiction. □

The proceeding claim yields a simple algorithm for determining whether two balls B_x and B_q are disjoint: project μ_x onto B_q using the line search algorithm discussed previously. The projected point will obviously be in B_q; if it is also in B_x, the two balls intersect. Otherwise, they are disjoint and exclusion pruning can be performed.

6.4 Algorithm overview

Let us summarize the algorithms for evaluating containment and intersection.

In order to evaluate whether $B_x \subset B_q$ we solve the following (non-convex) program:

$$\begin{aligned} \max_x \quad & d_f(x, \mu_q) \\ \text{subject to: } & d_f(x, \mu_x) \leq R_x. \end{aligned} \tag{6.4}$$

Let x_p be the optimal solution to (6.4). Then

$$B_x \subset B_q \quad \Longleftrightarrow \quad d_f(x_p, \mu_q) < R_q$$

We can solve (6.4) by finding the $\theta > 1$ such that $x_\theta \equiv \nabla f^*(\theta\mu' + (1-\theta)q')$ satisfies $d_f(x_\theta, \mu_x) = R_x$. Such a θ can be found using Newton's method. Assuming each Newton iteration can be achieved in $O(D)$ time (true for the bregman divergences discussed in this paper), the solution can be found in $O(D \log\log(1/\epsilon))$ time. Moreover, the algorithm can usually be stopped early by comparing the dual $\mathcal{L}(\theta)$ to R_q, since $d_f(x_p, \mu_q) < \mathcal{L}(\theta)$.

In order to evaluate whether $B_x \cap B_q = \emptyset$, we solve the following convex program:

$$\begin{aligned} \min_x \quad & d_f(x, \mu_q) \\ \text{subject to: } & d_f(x, \mu_x) \leq R_x \end{aligned} \tag{6.5}$$

Let x_p be the optimal solution to (6.5). Then

$$B_x \cap B_q = \emptyset \quad \Longleftrightarrow \quad d_f(x_p, \mu_q) < R_q$$

Similar to the previous program, we can solve this using an efficient line search.

6.5 Experiments

We compare the performance of the search algorithm to standard brute force search on several datasets. We are particularly interested in text applications as

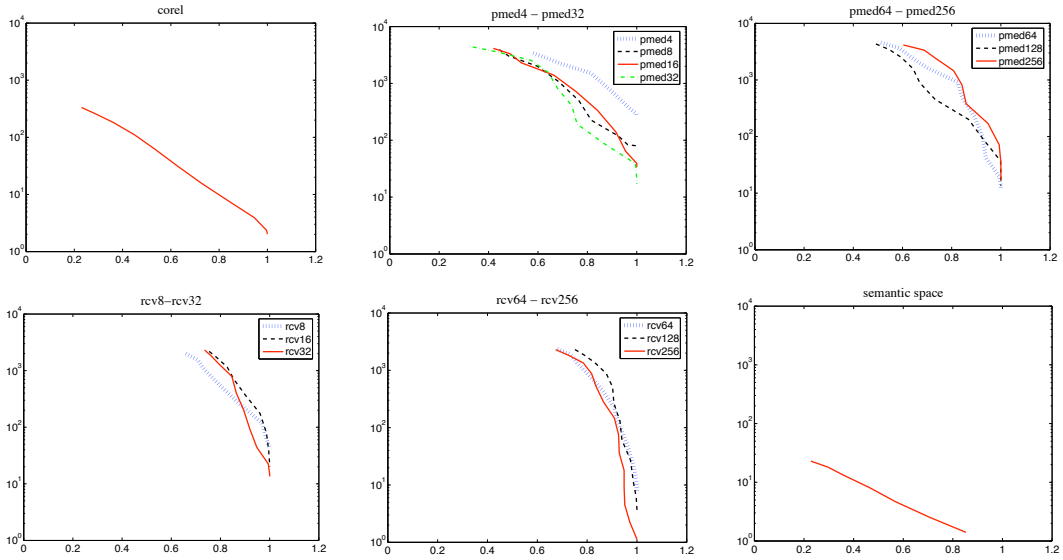


Figure 6.2: Approximate search results. The y-axis is on a logarithmic scale and is the speedup over brute force search. The x axis is a linear scale and is the average percentage of the points in range returned (*i.e.* the average recall).

histogram representations are common, datasets are often very large, and efficient search is broadly useful. We experimented with the following datasets.

- **pubmed-D.** We used one million documents from the pubmed abstract corpus (available from the UCI collection). We generated a *correlated topic model* (CTM) [BL07] with $D = 4, 8, \ldots, 256$ topics. For each D, we built a CTM using a training set and then performed inference on the 1M documents to generate the topic histograms.

- **Corel histograms.** This data set consists of 60k color histograms of dimensionality 64 generated from the Corel image datasets.

- **rcv-D.** Latent dirichlet allocation was applied to 500K documents from the rcv1 [LYRL04] corpus to generate topic histograms for each [BNJ03]. D is set to $8, 16, 32, \ldots 256$.

- **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 keywords [RMV07].

We note that many of these datasets are fairly high-dimensional.

All of our experiments are for the KL-divergence. Although the KL-divergence is widely used, little is known about efficient proximity techniques for it. In contrast, the ℓ_2^2 and Mahalanobis distances can be handled by metric methods, for which there is a huge literature.

There are two regimes where range search is particularly useful: when the radius γ is very small and when it is large. When γ is small, range search is useful in instance-based learning algorithms like locally weighted regression, which need to retrieve points close to each test point. It is also useful for generating neighborhood graphs. When γ is large enough that $B_f(q, \gamma)$ will contain most of the database, range search is potentially useful for applications like distance-based outlier detection and anomaly detection. We provide experiments for both of these regimes.

Table 6.1 shows the results for small γ. γ was chosen so that about 20 points would be inside the query ball (on average). On the pubmed datasets, we are getting one to two orders of magnitude speed-up across all dimensionalities. On the rcv datasets, the bb-tree range search algorithm is an order of magnitude faster than brute search except on the two datasets of highest dimensionality. The algorithm provides a useful speedup on corel, but no speedup on semantic space. We note that the semantic space dataset is both high-dimensional (371 dimensions) and quite small (5k), which makes it very hard for proximity search. The algorithm reflects the widely observed phenomenon that the performance of spatial decomposition data structures degrades with dimensionality, but still provides a useful speedup on several moderate-dimensional datasets.

Table 6.2 shows the results for large γ. In these experiments, γ was chosen so that all about about 100-300 points would be in range. The results here are more varied than for small γ, but we are still getting useful speedups across most of the datasets. Interestingly, the amount of speedup seems less dependent of the dimensionality in comparison to the small γ experiments.

Finally, we consider approximate search, which is the most likely use of this algorithm. There are many ways to use the bb-tree in an approximate way.

Table 6.1: Exact range search; small radius.

dataset	**dimensionality**	**speedup**
corel	64	2.53
pubmed4	4	371.6
pubmed8	8	102.7
pubmed16	16	37.3
pubmed32	32	18.6
pubmed64	64	13.26
pubmed128	128	15.0
pubmed256	256	18.9
rcv8	8	48.1
rcv16	16	23.0
rcv32	32	16.4
rcv64	64	11.4
rcv128	128	6.1
rcv256	256	1.1
semantic space	371	.7

Here, we follow [LMGY04] and simply cut-off the search process early. We are thus guaranteed to get only points within the specified range (perfect precision), but we may not get all of them (less than perfect recall). In instance-based learning algorithms, this loss of recall is often tolerable as long as a reasonable number of points are returned. Thus a practical way to deploy the range search algorithm is to run it until enough points are recovered. In this experiment, γ was set so that about 50 points would be returned. Figure 6.2 shows the results.

These are likely the most relevant results to practical applications. They demonstrate that the proposed algorithm provides a speedup of up to four orders of magnitude with a high recall.

The results of this chapter are under submission:

L. Cayton. Efficient bregman range search. *Submitted,* 2009.

Table 6.2: Exact range search; large radius.

dataset	**dimensionality**	**speedup**
corel	64	3.4
pubmed4	4	5.1
pubmed8	8	9.67
pubmed16	16	12.8
pubmed32	32	47.1
pubmed64	64	21.6
pubmed128	128	120.4
pubmed256	256	39.0
rcv8	8	8.9
rcv16	16	21.9
rcv32	32	16.4
rcv64	64	9.6
rcv128	128	3.1
rcv256	256	1.9
semantic space	371	1.0

Chapter 7

Alternative approaches to bregman proximity problems

In this chapter we examine approaches to bregman range and nearest neighbor search that are derived from known data structures. In the first section, we use linearization to transform the problem of bregman range search into a half-space range search. The result is the only known theoretical complexity bound on a bregman proximity problem. In the second section, we show that data structures based on axis-aligned splits (like the kd-tree and quad-tree) can be used for bregman proximity problems, as long as the bregman divergence is decomposable. This property is useful at least in part because there are many well-established software implementations of some of these classical data structures. Finally, we discuss bounding bregman balls with norm-balls and what that implies for proximity search.

7.1 Lifting to half-space search

In this section, we give a provably efficient algorithm for bregman range search by reducing the problem to a half-space range search. We note that the algorithms for half-space range search are primarily of theoretical interest. Thus though the developments in this section provide a theoretical bound on query time, whereas the bb-tree range search algorithm does not, the bb-tree-based algorithm

is more practical.

We show that a bregman range query reduces to a half-space range query using a lifting technique (also known as *linearization*). This technique is common in the context of ℓ_2 and has also been employed for bregman divergences [NBN07]. Suppose we wish to return all $x \in X$ that are divergence at most γ away from q; *i.e.* all points in $B_f(q, \gamma)$. A point x is in $B_f(q, \gamma)$ if

$$\begin{aligned} & d_f(x, q) \leq \gamma \\ \Leftrightarrow \quad & f(x) - f(q) - \langle \nabla f(q), x - q \rangle \leq \gamma \\ \Leftrightarrow \quad & f(x) + f^*(q') - \langle x, q' \rangle \leq \gamma, \end{aligned}$$

where again $q' \equiv \nabla f(q)$. The last equivalence was discussed in section 3.2, page 20. This form of a bregman divergence is useful because the two parameters (x and q) interact only through a inner product.

Map each $x \in X$ to

$$\widehat{x} \equiv (x_1, \ldots, x_D, f(x)) \tag{7.1}$$

and q to

$$\widehat{q} \equiv (q'_1, \ldots, q'_D, -1). \tag{7.2}$$

Then $x \in B_f(q, \gamma)$ if and only if

$$\langle \widehat{x}, \widehat{q} \rangle + \gamma - f^*(q') \geq 0,$$

which is a $(D+1)$-dimensional half-space range query.

Half-space range queries have been studied extensively in computational geometry, though many of the algorithms are primarily theoretical [AE99]. We do not survey the algorithms here, as they are rather intricate, but instead simply state the bounds. A D-dimensional half-space range queries can be answered in $O(\log n + k)$ time using $O(n^{\lfloor D/2 \rfloor} \text{polylog}(n))$ space, where k is the number of database points in range. If only counts are needed, the problem can solved in $O(\log n)$ time using $O(n^D / \log^D n)$ storage. Thus we get the following theorem.

Theorem 7. *Suppose that we have a database $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^D$, and, on query $B_f(q, \gamma)$, wish to return all $x \in B_f(q, \gamma) \cap X$. Queries of this form can be answered in time $O(\log n + k)$, where k is the cardinality of the intersection, using a data structure requiring space $O(n^{\lfloor (D+1)/2 \rfloor} \operatorname{polylog}(n))$.*

The query time is the best one can hope for proximity problems. Of course, the exponential dependence on D makes these methods unattractive for many applications. Unfortunately, work on lower bounds suggests that this exponential dependence is unavoidable [BCP93].

Relaxing half-space range search to *approximate* range search allows considerably better bounds. Suppose that H^+ is a half-space. Then a correct answer to a ϵ-approximate range query for H^+ contains all points $x \in H^+$ that are Euclidean distance at least ϵ from the boundary, and no points $x \notin H^+$ that are distance at least ϵ from the boundary. Points within distance ϵ of the boundary can be returned or not. Remarkably, this relaxation of the problem transforms the complexity from exponential to polynomial. In particular, using random projection techniques, [CLM08] give a data structure for approximate range search that uses $dn^{O(1/\epsilon^2)}$ storage and can answer queries in time $O(d/\epsilon^2 \operatorname{polylog}(d/\epsilon^2))$. We now show that these results can be used for bregman range search.

We define an ϵ-approximate bregman range for query $B_f(q, \gamma)$ as any subset of the database X containing all points $x \in X \cap B_f(q, \gamma(1-\epsilon))$ and no points $x \notin X \cap B_f(q, \gamma(1+\epsilon))$. Points in $B_f(q, \gamma(1+\epsilon)) \setminus B_f(q, \gamma(1-\epsilon))$ can be included or skipped.

Consider the halfspace $H^+ = \{x \mid \langle x, q \rangle \geq c\}$, with boundary hyperplane $H = \{x \mid \langle x, q \rangle = c\}$. Then as is easily verified, the ℓ_2 distance from a point y to H is given by

$$\|y - P_H(y)\|_2 = \left| \frac{\langle y, q \rangle - c}{\|q\|} \right|,$$

where $P_H(y)$ denotes the projection of y onto H.

Suppose we wish to return points $x \in X$ in an approximate bregman range of $B_f(q, \gamma)$. We need to show that an approximate half-space corresponds to a lifted approximate bregman ball. We use the same lifting map defined above in

equations (7.1) and (7.2), which defines a hyperplane boundary

$$H = \{\widehat{x} \mid \langle \widehat{x}, \widehat{q} \rangle = f^*(q') - \gamma\}. \tag{7.3}$$

Suppose that $x \in B_f(q, \gamma(1-\epsilon))$. From the derivation above, we know that x will lie on the correct side of the H; here, we check that it lies sufficiently far away from the H. We compute

$$\begin{aligned}
\|\widehat{x} - P_H(\widehat{x})\|_2 &= \left| \frac{\langle \widehat{x}, \widehat{q} \rangle - f^*(q') + \gamma}{\|\widehat{q}\|} \right| \\
&= \left| \frac{\gamma - d_f(x, q)}{\|\widehat{q}\|} \right| \\
&\geq \frac{\gamma - \gamma(1-\epsilon)}{\|\widehat{q}\|} \\
&= \frac{\gamma\epsilon}{\|\widehat{q}\|}.
\end{aligned}$$

Similarly, if $x \notin B_f(q, \gamma(1+\epsilon))$, we have

$$\begin{aligned}
\|\widehat{x} - P_H(\widehat{x})\|_2 &= \left| \frac{\langle \widehat{x}, \widehat{q} \rangle - f^*(q') + \gamma}{\|\widehat{q}\|} \right| \\
&\geq \frac{\gamma(1+\epsilon) - \gamma}{\|\widehat{q}\|} \\
&= \frac{\gamma\epsilon}{\|\widehat{q}\|}.
\end{aligned}$$

Thus an ϵ-approximate bregman range query lifts to a $\epsilon\gamma/\|\widehat{q}\|$ approximate halfspace query. Though this dependence on the query parameters is not ideal, it is reasonable to assume that the query center q can be restricted to bounded region $\|\widehat{q}\| < c$ and the query radius is bounded below by some constant. Under these assumptions, the derivation implies the following theorem.

Theorem 8. *Approximate bregman range queries can be answered in time* $O((D+1)/\epsilon^2 \operatorname{polylog}((D+1)/\epsilon^2))$ *time using a data structure requiring space* $(D+1)n^{O(1/\epsilon^2)}$*, assuming that all query centers are drawn from* $\{q \mid \|\widehat{q}\| < c_1\}$ *and query radii are drawn from* $\{\gamma \mid \gamma > c_2\}$*, where* c_1 *and* c_2 *are constants.*

7.2 Data structures based on axis-aligned rectangles

In this section, we show that data structures based on a axis-aligned rectangles can be used for the class of *decomposable* bregman divergences. The majority of bregman divergence are decomposable; refer back to table 3.1 on page 17. Examples of data structures using axis-aligned rectangles includes the quadtree, the kd-tree, and many variants such as the balanced box decomposition tree [AMN+98]. These type of data structures typically perform poorly on moderate to high-dimensional data. However, on very low-dimensional data, they often outperform ball-style decomposition schemes like the bb-tree and the metric tree. The reason is that the lower bounds on distances from a query to a rectangle can be computed more efficiently than the bounds from a query to a ball. Another advantage of these data structures is that there is well-developed software available for them, *e.g.* [MA].

Adapting these data structures is fairly simple. We need only show that the bregman projection onto a rectangle can be efficiently computed. We also need to show that the maximization problem—computing the maximum divergence from a point to a rectangle—is efficiently computable.

First, we define the class of bregman divergences that we will be working with.

Definition 3. *A bregman divergence d_f (defined on a subset of $\mathbb{R}^D$) is* decomposable *if it can be written as the sum of one-dimensional bregman divergences—i.e.*

$$d_f(x, y) = \sum_i \rho_{g_i}(x_i, y_i),$$

where $g_i : \mathbb{R} \to \mathbb{R}$ is a strictly convex function and ρ_{g_i} is the bregman divergence based on g_i.

7.2.1 Projection onto an axis-aligned rectangle

In this section, we analyze the problem of computing a bregman projection onto an axis-aligned rectangle and the associated maximization problem. We show

that we can compute this projection efficiently for decomposable bregman divergences. We only consider the case where the base function is the same for each dimension, though the more general case follows easily. The program we wish to solve is

$$\begin{aligned} \min_{x} \quad & d_f(x, q) \\ \text{subject to:} \quad & x_i \in [l_i, u_i] \ \text{ for } i \in [D]. \end{aligned} \tag{minBox}$$

The lagrangian of this program is

$$\begin{aligned} \nu(x, \lambda, \xi) &\equiv d_f(x, q) + \sum \lambda_i (x_i - u_i) + \sum \xi_i (l_i - x_i) \\ &= \sum \rho_g(x_i, q_i) + \sum \lambda_i (x_i - u_i) + \sum \xi_i (l_i - x_i). \end{aligned}$$

Differentiating with respect to x_i, we get

$$\frac{\partial \nu}{\partial x_i} = g'(x_i) - g'(q_i) + \lambda_i - \xi_i,$$

which implies that the optimal x^* satisfies

$$g'(x_i^*) = g'(q_i) + \xi_i - \lambda_i. \tag{7.4}$$

Applying complementary slackness,

$$\begin{aligned} \lambda_i > 0 \implies x_i = u_i \\ \xi_i > 0 \implies x_i = l_i. \end{aligned}$$

Assuming $l_i \neq u_i$,

$$\begin{aligned} \lambda_i > 0 \implies \xi_i = 0 \\ \xi_i > 0 \implies \lambda_i = 0 \end{aligned}$$

Setting $\eta_i = \xi_i - \lambda_i$, we get

$$\begin{aligned} \eta_i > 0 \implies x_i = u_i \\ \eta_i < 0 \implies x_i = l_i \\ \eta_i = 0 \implies g'(x_i) = g'(q_i), \end{aligned}$$

where the last line follows from (7.4). Since g is strictly convex, and hence one-to-one, this equality implies that $x_i = q_i$.

The result of all of this is that solving (minBox) is simple. For each i, x_i can be one of three values: l_i, u_i, or q_i (if $q_i \in [l_i, u_i]$). Thus the optimal x^* is given by

$$x_i^* = \begin{cases} \operatorname{argmin}_{x_i \in \{l_i, u_i, q_i\}} \ \rho_g(x_i, q_i) & \text{if } q_i \in [l_i, u_i] \\ \operatorname{argmin}_{x_i \in \{l_i, u_i\}} \ \rho_g(x_i, q_i) & \text{otherwise.} \end{cases}$$

Thus (minBox) can be solved in $O(D)$ time.

The *maximization* problem can be solved similarly. In particular, we will be maximizing a (univariate) convex function subject to box constraints, so the maximum must occur at one of the boundary points.

Chapter 8

A learning framework for proximity search

In this chapter, we leave the main thread of the dissertation behind somewhat and develop a broadly applicable, novel approach to proximity problems. This framework is not restricted to bregman proximity problems; we will work out the details of the framework for kd-trees and the structures employed by a locality sensitive hashing scheme.

The basic idea of the framework is to think of a proximity data structure as a function mapping queries to some (hopefully) subset of the database. Rather than building this function in the traditional way, we will instead attempt to *learn* it using a small sample query set, assumed to be drawn from the same (unknown) distribution as all of the queries.

This framework addresses several major shortcomings of standard approaches to proximity search:

1. The theoretical bounds for proximity search (see chapter 7 for examples) are often not very practical, typically because of exponential dependence on running time, and large constant factors.

2. The algorithms used for building proximity data structures (see sections 4.2 for example) are somewhat heuristic and seem suboptimal.

3. The data structures used for proximity search are, with a few exceptions,

not sensitive to the underlying distribution of queries. If, for example, part of the database is retrieved much more often than another, it's difficult to optimize the data structure to take advantage of this usage pattern.

We address the first shortcoming by providing a different type of theoretical guarantee: rather than focus on the worst case, generalization theory is employed to describe the performance of a data-structure on a distribution-by-distribution basis. As for the other shortcomings, the framework suggests explicitly optimizing the performance of a data structure over a test set, which addresses both.

8.1 Learning framework

In this section we formalize the learning framework for NN search. Again, this framework is quite general and it will hopefully be of use to algorithmic developments in NN searching beyond those presented here.

Let $X = \{x_1, \ldots, x_n\}$ denote the database and $\mathcal{Q}$ the space from which queries are drawn. A typical example is $X \subset \mathbb{R}^D$ and $\mathcal{Q} = \mathbb{R}^D$. We take a nearest neighbor data structure to be a mapping $f : \mathcal{Q} \to 2^X$; the interpretation is we compute distances only to $f(q)$, not all of X. For example, the structure underlying LSH partitions $\mathbb{R}^D$ into cells and a query is assigned to the subset of X that falls into the same cell.

What quantities are we interested in optimizing? We want to only compute distances to a small fraction of the database on a query; and, in the case of probabilistic algorithms, we want a high probability of success. More precisely, we hope to minimize the following two quantities for a data structure f:

- The fraction of X that we need to compute distances to:

$$\text{size}_f(q) \equiv \frac{|f(q)|}{n}.$$

- The fraction of a query's k nearest neighbors that are missed:

$$\text{miss}_f(q) \equiv \frac{|\Gamma_k(q) \setminus f(q)|}{k}$$

($\Gamma_k(q)$ denotes the k nearest neighbors of q in X).

In ϵ-approximate NN search, we only require a point x such that $d(q,x) \leq (1+\epsilon)d(q,X)$, so we instead use an approximate miss rate:

$$\epsilon\text{miss}_f(q) \equiv \mathbf{1}\left[\nexists x \in f(q) \text{ such that } d(q,x) \leq (1+\epsilon)d(q,X)\right].$$

None of the previously discussed data structures are built by explicitly minimizing these quantities, though there are known bounds for some. The reason is that research has typically focused on *worst-case* size_f and miss_f rates, which require minimizing these functions over *all* $q \in \mathcal{Q}$. $\mathcal{Q}$ is typically infinite.

In this work, we instead focus on *average-case* size_f and miss_f rates—*i.e.* we assume q is a draw from some unknown distribution $\mathcal{D}$ on $\mathcal{Q}$ and hope to minimize

$$\mathbb{E}_{q\sim\mathcal{D}}\left[\text{size}_f(q)\right] \quad \text{and} \quad \mathbb{E}_{q\sim\mathcal{D}}\left[\text{miss}_f(q)\right].$$

To do so, we assume that we are given a sample query set $Q = \{q_1, \ldots, q_m\}$ drawn iid from $\mathcal{D}$. We attempt to build f minimizing the empirical size and miss rates, then resort to generalization bounds to relate these rates to the true ones.

8.2 Learning algorithms

We propose two learning algorithms in this section. The first is based on a splitting rule for kd-trees designed to minimize a greedy surrogate for the empirical size_f function. The second is a algorithm that determines the boundary locations of the cell structure used in LSH that minimize a tradeoff of the empirical size_f and ϵmiss_f functions. Though they were already mentioned earlier in the dissertation, we review kd-trees and LSH in a bit more detail here.

8.2.1 kd-trees

kd-trees are a popular cell partitioning scheme for $\mathbb{R}^D$ for any ℓ_p metric. The data structure is built by picking a dimension, splitting the database along the median value in that dimension, and then recursing on both halves. See Algorithm 4 for pseudocode of the build procedure and Figure 8.1 for an example kd-tree.

Algorithm 4 BUILDKDTREE

Input: $S \subset X \subset \mathbb{R}^D$.

if $|S| < BucketSize$ **then**

return *Leaf*.

else

Pick an axis i.

Let $t = \text{median}(s_i : s \in S)$.

$LeftTree$ = BUILDKDTREE$(\{s \in S : s_i \leq t\})$.

$RightTree$ = BUILDKDTREE$(\{s \in S : s_i > t\})$.

return [*LeftTree*, *RightTree*, *median*, i].

end if

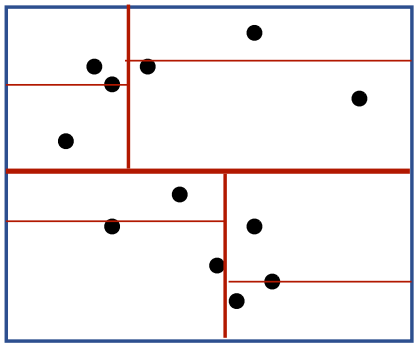


Figure 8.1: An example kd-tree.

To find a NN for a query q, one first computes distances to all points in the same cell, then traverses up the tree. At each parent node, the minimum distance between q and points already explored is compared to the distance to the split. If the latter is smaller, then the other child must be explored.

Explore right subtree:  Do not explore: 

Typically the cells contain only a few points; a query is expensive because it lies close to many of the cell boundaries and much of the tree must be explored.

Learning method

Rather than picking the median split at each level, we use the training queries q_i to pick a split that greedily minimizes the expected cost. A split s divides the sample queries (that are in the cell being split) into three sets: Q_{tc}, those q that are "too close" to s—*i.e.* nearer to s than $d(q, X)$; Q_r, those on the right of s but not in Q_{tc}; and Q_l, those on the left of s but not in Q_{tc}. Queries in Q_{tc} will require exploring both sides of the split. The split also divides the database points (that are in the cell being split) into X_l and X_r. The cost of split s is then defined to be

$$\text{cost}(s) \equiv |Q_l| \cdot |X_l| + |Q_r| \cdot |X_r| + |Q_{tc}| \cdot |X|.$$

$\text{cost}(s)$ is a greedy surrogate for $\sum_i \text{size}_f(q_i)$; evaluating the true average size would require a potentially costly recursion. In contrast, minimizing $\text{cost}(s)$ can be done painlessly since it takes on at most $2m+n$ possible values and each can be evaluated quickly. In fact, one can compute the optimal value with just a single pass over the database and query points. Using a sample set led us to a very simple, natural cost function that can be used to pick splits in a principled manner.

8.2.2 bb-trees

We show that the same algorithm described above for kd-trees can be applied to bregman ball trees. Recall the algorithm used recursively for splitting a subset of the database into two smaller subsets: the points are clustered into two groups using the 2-means algorithm and two bregman balls $B_f(\mu_l, R_l)$ and $B_f(\mu_r, R_r)$ are defined to cover the clusters. The learning algorithm will leave the centers of the balls untouched, but will refine the radii of the balls. Consider points x that are equidistant from μ_l and μ_r. They satisfy

$$\begin{aligned}
0 &= d_f(x, \mu_l) - d_f(x, \mu_r) \\
&= f(x) - f(\mu_l) - \langle \nabla f(\mu_l), x - \mu_l \rangle - f(x) + f(\mu_r) + \langle \nabla f(\mu_r), x - \mu_r \rangle \\
&= f(\mu_r) - f(\mu_l) + \langle x, \nabla f(\mu_r) - \nabla f(\mu_l) \rangle + \langle \nabla f(\mu_l), \mu_l \rangle - \langle \nabla f(\mu_r), \mu_r \rangle.
\end{aligned}$$

Algorithm 5 BuildRCS

Input: $X \subset \mathbb{R}^D$

Let $\mathbb{R} \in \mathbb{R}^{O(\log n) \times d}$ where R_{ij} is an iid draw from a p-stable distribution.

Project the database down to $O(\log n)$ dimensions: $x_i \mapsto Rx_i$.

Uniformly grid the space with B bins per direction.

Hence, the points equidistant from μ_l and μ_r form a hyperplane (refer back to section 3.5. Points satisfying $d_f(x, \mu_l) - d_f(x, \mu_r) = c$, for some constant c, will lie on a parallel hyperplane. Thus we can use the algorithm of the previous section to find an optimal value c for the split, and then set the radii accordingly.

8.2.3 Locality sensitive hashing

LSH was a tremendous breakthrough in NN search as it led to data structures with provably sublinear (in the database size) retrieval time for approximate NN searching. More impressive still, the bounds on retrieval are independent of the dimensionality of the database. We focus on the LSH scheme for the $\|\cdot\|_p$ norm ($p \in (0, 2]$), which we refer to as LSH_p. It is built on an extremely simple space partitioning scheme which we refer to as a *rectilinear cell structure* (RCS).

Pseudocode for building an RCS appears as Algorithm 5. The algorithm requires drawing samples from a p-stable distribution; a distribution $\mathcal{D}_p$ is p-stable if for any $v \in \mathbb{R}^d$ and $Z, X_1, \ldots, X_d$ drawn iid from $\mathcal{D}_p$, $\langle v, X \rangle \stackrel{\text{d}}{=} \|v\|_p Z$. For example, $\mathcal{N}(0, 1)$ is 2-stable.

See figure 8.5, left panel, for an example. On query q, one simply finds the cell that q belongs to, and returns the nearest x in that cell.

In general, LSH_p requires many RCSs, used in parallel, to achieve a constant probability of success; though in many situations one may suffice [DIIM04]. Note that LSH_p only works for distances at a single scale R: the specific guarantee is that LSH_p will return a point $x \in X$ within distance $(1 + \epsilon)R$ of q as long as $d(q, X) < R$. To solve the standard ϵ approximate NN problem, one must build $O(\log(n/\epsilon))$ LSH_p structures.

Learning method

We apply our learning framework directly to the class of RCSs since they are the core structural component of LSH_p. We consider a slightly wider class of RCSs where the bin widths are allowed to vary. Doing so potentially allows a *single* RCS to work at multiple scales if the bin positions are chosen appropriately. We give a simple procedure that selects the bin boundary locations.

We wish to select boundary locations minimizing the cost $\sum_i \epsilon\mathrm{miss}_f(q_i) + \lambda\mathrm{size}_f(q_i)$, where λ is a tradeoff parameter (alternatively, one could fix a miss rate that is reasonable, say 5%, and minimize the size). The optimization is performed along one dimension at a time. Fortunately, the optimal binning along a dimension can be found by dynamic programming. There are at most $m+n$ possible boundary locations; order them from left to right. The cost of placing the boundaries at p_1, p_2, p_{B+1} can be decomposed as $c[p_1, p_2] + \cdots + c[p_B, p_{B+1}]$, where

$$c[p_i, p_{i+1}] = \sum_{q \in [p_i, p_{i+1}]} \epsilon\mathrm{miss}_f(q) + \lambda \sum_{q \in [p_i, p_{i+1}]} \left|\{x \in [p_i, p_{i+1}]\}\right|.$$

Let D be our dynamic programming table where $D[p, i]$ is defined as the cost of putting the ith boundary at position p and the remaining $B + 1 - i$ to the right. Then

$$D[p, i] = \min_{p' \geq p} c[p, p'] + D[p', i - 1].$$

8.3 Generalization theory

In our framework, a nearest neighbor data structure is learned by specifically designing it to perform well on a set of sample queries. Under what conditions will this search structure have good performance on future queries?

Recall the setting: there is a database $X = \{x_1, \ldots, x_n\}$, sample queries $Q = \{q_1, \ldots, q_m\}$ drawn iid from some distribution $\mathcal{D}$ on $\mathcal{Q}$, and we wish to learn a data structure $f : \mathcal{Q} \to 2^X$ drawn from a function class $\mathcal{F}$. We are interested in the generalization of $\mathrm{size}_f(q) \equiv \frac{|f(q)|}{n}$, and $\mathrm{miss}_f(q) \equiv \frac{|\Gamma_k(q) \setminus f(q)|}{k}$, both of which have range $[0, 1]$ ($\epsilon\mathrm{miss}_f(q)$ can be substituted for $\mathrm{miss}_f(q)$ throughout this section).

Suppose a data structure f is chosen from some class $\mathcal{F}$, so as to have low empirical cost

$$\frac{1}{m}\sum_{i=1}^{m}\text{size}_f(q_i) \quad \text{and} \quad \frac{1}{m}\sum_{i=1}^{m}\text{miss}_f(q_i).$$

Can we then conclude that data structure f will continue to perform well for subsequent queries drawn from the underlying distribution on $\mathcal{Q}$? In other words, are the empirical estimates above necessarily close to the true expected values $\mathbb{E}_{q\sim\mathcal{D}}\text{size}_f(q)$ and $\mathbb{E}_{q\sim\mathcal{D}}\text{miss}_f(q)$?

There is a wide range of uniform convergence results which relate the difference between empirical and true expectations to the number of samples seen (in our case, m) and some measure of the complexity of the two classes $\{\text{size}_f : f \in \mathcal{F}\}$ and $\{\text{miss}_f : f \in \mathcal{F}\}$. The following is particularly convenient to use, and is well-known [BBL04, theorem 3.2].

Theorem 9. *Let $\mathcal{G}$ be a set of functions from a set $\mathcal{Z}$ to $[0,1]$. Suppose a sample $z_1,\ldots,z_m$ is drawn from some underlying distribution on $\mathcal{Z}$. Let $\mathcal{G}_m$ denote the restriction of $\mathcal{G}$ to these samples, that is,*

$$\mathcal{G}_m = \{(g(z_1), g(z_2), \ldots, g(z_m)) : g \in \mathcal{G}\}.$$

Then for any $\delta > 0$, the following holds with probability at least $1-\delta$:

$$\sup_{g\in\mathcal{G}}\left|\mathbb{E}g - \frac{1}{m}\sum_{i=1}^{m}g(z_i)\right| \;\leq\; 2\sqrt{\frac{2\log|\mathcal{G}_m|}{m}} + \sqrt{\frac{\log(2/\delta)}{m}}.$$

This can be applied immediately to the kind of data structure used by LSH.

Definition 4. *A $(u_1,\ldots,u_d,B)$-rectilinear cell structure (RCS) in $\mathbb{R}^D$ is a partition of $\mathbb{R}^D$ into B^d cells given by*

$$x \mapsto (h_1(x\cdot u_1),\ldots,h_d(x\cdot u_d)),$$

where each $h_i : \mathbb{R} \to \{1,\ldots,B\}$ is a partition of the real line into B intervals.

Theorem 10. *Fix any vectors $u_1, \ldots, u_d \in \mathbb{R}^D$, and, for some positive integer B, let the set of data structures $\mathcal{F}$ consist of all $(u_1, \ldots, u_d, B)$-rectilinear cell structures in $\mathbb{R}^D$. Fix any database of n points $X \subset \mathbb{R}^D$. Suppose there is an underlying distribution over queries in $\mathbb{R}^D$, from which m sample queries $q_1, \ldots, q_m$ are drawn. Then*

$$\sup_{f \in \mathcal{F}} \left| \mathbb{E}[\text{miss}_f] - \frac{1}{m} \sum_{i=1}^{m} \text{miss}_f(q_i) \right| \leq 2\sqrt{\frac{2d(B-1)\log(m+n)}{m}} + \sqrt{\frac{\log(2/\delta)}{m}}$$

and likewise for size_f.

Proof. Fix any $X = \{x_1, \ldots, x_n\}$ and any $q_1, \ldots, q_m$. In how many ways can these points be assigned to cells by the class of all $(u_1, \ldots, u_d, B)$-rectilinear data structures? Along each axis u_i there are $B-1$ boundaries to be chosen and only $m+n$ distinct locations for each of these (as far as partitioning of the x_i's and q_i's is concerned). Therefore there are at most $(m+n)^{d(B-1)}$ ways to carve up the points. Thus the functions $\{\text{miss}_f : f \in \mathcal{F}\}$ (or likewise, $\{\text{size}_f : f \in \mathcal{F}\}$) collapse to a set of size just $(m+n)^{d(B-1)}$ when restricted to m queries; the rest follows from theorem 9. □

This is good generalization performance because it depends only on the *projected* dimension, not the original dimension. It holds when the projection directions $u_1, \ldots, u_d$ are chosen randomly, but, more remarkably, even if they are chosen based on X (for instance, by running PCA on X). If we learn the projections as well (instead of using random ones) the bound degrades substantially.

Theorem 11. *Consider the same setting as Theorem 10, except that now $\mathcal{F}$ ranges over $(u_1, \ldots, u_d, B)$-rectilinear cell structures for all choices of $u_1, \ldots, u_d \in \mathbb{R}^D$. Then with probability at least $1-\delta$,*

$$\sup_{f \in \mathcal{F}} \left| \mathbb{E}[\text{miss}_f] - \frac{1}{m} \sum_{i=1}^{m} \text{miss}_f(q_i) \right| \leq 2\sqrt{\frac{2 + 2d(D+B-2)\log(m+n)}{m}} + \sqrt{\frac{\log(2/\delta)}{m}}$$

and likewise for size_f.

Proof. Again, fix any $X = \{x_1, \ldots, x_n\}$ and any $q_1, \ldots, q_m$; this time there are a lot more ways to partition these points into cells. For each $1 \leq i \leq d$, first a

direction u_i needs to be specified, then a binning along that direction. The $B-1$ bin boundaries are parallel hyperplanes; imagine first selecting one of them. By standard results, for instance [DGL96, page 233], there are a total of at most $2(m+n)^D$ ways to do this. Then pick the $B-2$ hyperplanes parallel to this first one: for each, there are $m+n$ choices. Thus there are at most $2(m+n)^{D+B-2}$ ways to select the binning for each i, and thus at most $2(m+n)^{d(D+B-2)}$ ways to divide the database points and queries into cells. □

kd-trees are slightly different than RCSs: the directions u_i are simply the coordinate axes, and the number of partitions per direction varies (*e.g.* one direction may have 10 partitions, another only 1).

Theorem 12. *Let $\mathcal{F}$ be the set of all depth η kd-trees in $\mathbb{R}^D$ and $X \subset \mathbb{R}^D$ be a database of points. Suppose there is an underlying distribution over queries in $\mathbb{R}^D$ from which $q_1, \ldots q_m$ are drawn. Then with probability at least $1-\delta$,*

$$\sup_{f \in \mathcal{F}} \left| \mathbb{E}[\text{miss}_f] - \frac{1}{m} \sum_{i=1}^{m} \text{miss}_f(q_i) \right| \leq 2\sqrt{\frac{(2^{\eta+1}-2)\log\left(D(3m+n)\right)}{m}} + \sqrt{\frac{\log(2/\delta)}{m}}$$

Proof. Consider a depth 1 kd-tree (single split). There are D possible directions to choose; and for each, there are at most $3m+n$ possible divider locations. Each database point either falls to the left or right of the boundary, giving the n. For the sample query points, each query either falls to the left or right, giving m, but we must add an additional $2m$ depending on whether the boundary falls close enough to each q (on the right or left) to put it in the "too close" set.

At the next level, we are choosing splits for two separate sets: the points that fell on the left of the first split and those on the right. We cannot treat these as two independent problems as the choice of split on the right could affect $\text{size}_f(q)$ for a query q on the left if it is close to the boundary. At depth two, then, there are at most $D(3m+n) \cdot D(3m+n) = (D(3m+n))^2$ possible *total* choices for the two boundaries. For a depth η tree we have at most

$$\prod_{i=1}^{\eta} \left(D(3m+n)\right)^{2^i}$$

total possibilities. Applying the uniform convergence bound then gives the result. □

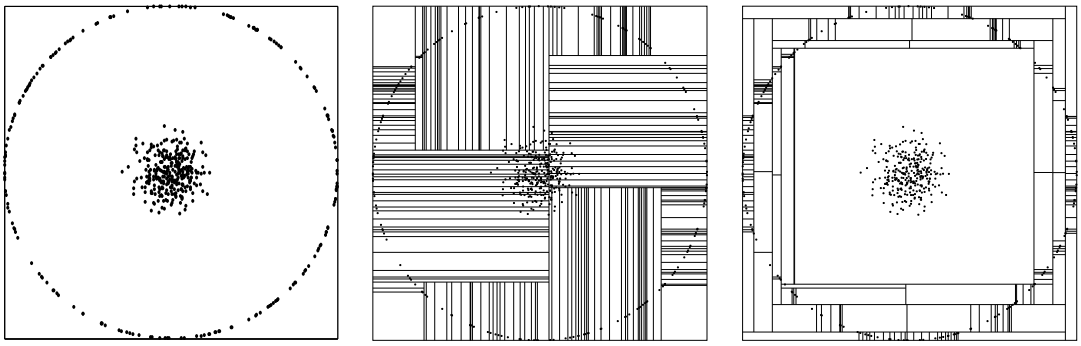


Figure 8.2: **Left**: Outer ring is the database; inner cluster of points are the queries. **Center**: kd-tree with standard median splits. **Right**: kd-tree with learned splits.

A kd-tree utilizing median splits has depth $\eta \leq \log n$. The depth of a kd-tree with learned splits can be higher, though we found empirically that the depth was always much less than $2 \log n$ (and can of course be restricted manually). kd-trees require significantly more samples than RCSs to generalize; the class of kd-trees is much more complex than that of RCSs.

The same proof goes through without change for bb-trees.

8.4 Experiments

8.4.1 kd-trees

First let us look at a simple example comparing the learned splits to median splits. Figure 8.2 shows a 2-dimensional dataset and the cell partitions produced by the learned splits and the median splits. The kd-tree constructed with the median splitting rule places nearly all of the boundaries running right through the queries. As a result, nearly the entire database will have to be searched for queries drawn from the center cluster distribution. The kd-tree with the learned splits places most of the boundaries right around the actual database points, ensuring that fewer leaves will need to be examined for each query. We compare the actual numbers below ("Toy" in table 8.2).

We now show results on several datasets from the UCI repository and 2004 KDD cup competition. We restrict attention to relatively low-dimensional datasets

($D < 100$) since that is the domain in which kd-trees are typically applied. These experiments were all conducted using a modified version of Mount and Arya's excellent kd-tree software [MA]. For the first set of experiments, we used a randomly selected subset of the dataset as the database and a separate small subset as the test queries. For the sample queries, we used the database itself—*i.e. no additional data was used to build the learned kd-tree.*

Table 8.1 shows the results. We compare performance in terms of the average number of database points we have to compute distances to on a test set.

The learned method outperforms the standard method on all of the datasets, showing a very large improvement on several of them. Note also that even the standard method exhibits good performance, often requiring distance calculations to less than one percent of the database. We are showing strong improvements on what are already quite good results.

What about when the query distribution is different than the database distribution? We took the datasets above that split into multiple classes and divided them into separate query and database sets. Table 8.2 shows the performance comparison.

Notice how the performance of kd-trees has degraded as compared to the previous experiments. We see a dramatic improvement for the Covertype dataset, and a reasonably strong improvement for the rest of the datasets.

We additionally experimented with the 'Corel50' image dataset. It is divided into 50 classes (*e.g.* air shows, bears, tigers, fiji) containing 100 images each. We used the 371-dimensional "semantic space" representation of the images recently developed in a series of image retrieval papers (see *e.g.* [RMV07]). This dataset allows us to explore the effect of differing query and database distributions in a natural setting. It also demonstrates that kd-trees with learned parameters can perform well on high-dimensional data.

Figure 8.3 shows the results of running kd-trees using median and learned splits. In each case, 4000 images were chosen for the database (from across all the classes) and images from select classes were chosen for the queries. The "All" queries were chosen from all classes; the "Animals" were chosen from the 11 animal

Table 8.1: Learned kd-tree vs standard kd-tree using the database itself as sample queries.

data set	DB size	test pts	dim	# distance calculations		%
				median split	learned split	improvement
Corel (UCI)	32k	5k	32	1035.7	403.7	61.0
Covertype (UCI)	100k	10k	55	20.8	18.4	11.4
Letter (UCI)	18k	2k	16	470.1	353.8	27.4
Pen digits (UCI)	9k	1k	16	168.9	114.9	31.9
Bio (KDD)	100k	10k	74	1409.8	1310.8	7.0
Physics (KDD)	100k	10k	78	1676.6	404.0	75.9

Table 8.2: Learned kd-tree vs standard kd-tree when the query distribution is different than the database distribution.

data set	query/DB	train pts	DB size	# distance calculations		%
				median split	learned split	improvement
Covertype	{1}/{2-8}	30k	100k	4387.0	31.5	99.3
Letter	{w-z}/{a-v}	2.5k	16k	2108.2	1529.6	27.4
Pen digits	{4}/{0-3, 5-9}	1k	9k	1860.6	1410.9	24.2
Toy	cluster/ring	250	200	162.9	78.4	51.9

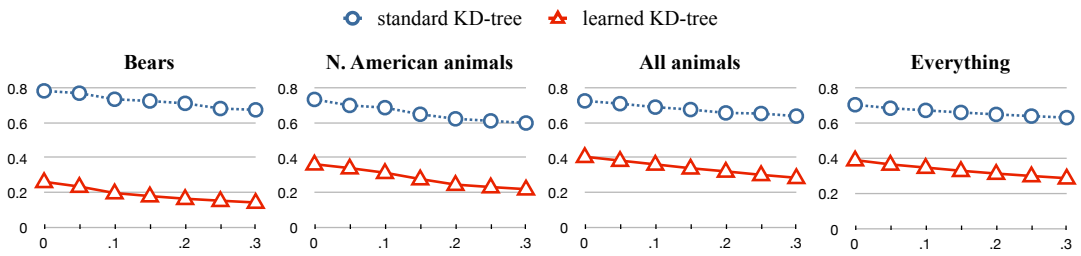


Figure 8.3: Percentage of DB examined as a function of ϵ (the approximation factor) for various query distributions.

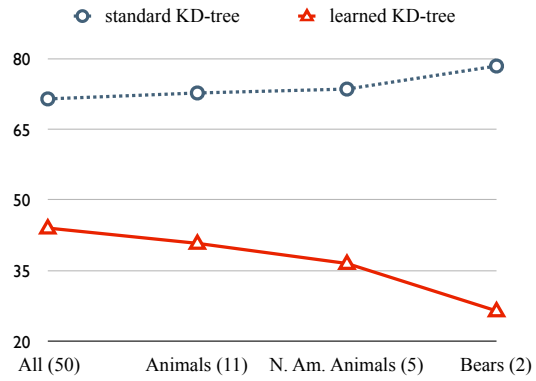


Figure 8.4: Percentage of DB examined for exact NN search as a function of the query distribution. Quantity in parentheses denotes the number of classes queries were drawn from.

classes; the "N. American animals" were chosen from 5 of the animal classes; and the "Bears" were chosen from the two bear classes. Standard kd-trees are performing somewhat better than brute force in these experiments. The learned kd-trees yield much faster retrieval times across a range of approximation errors.

Figure 8.4 shows the number of distance calculations versus the query distributions. The learning method is able to take advantage of simpler query distributions, whereas the performance of the standard method actually degrades.

8.4.2 RCS/LSH

Figure 8.5 is a simple example showing the effect of the learning algorithm. The queries and DB are drawn from the same distribution. Notice how the learning

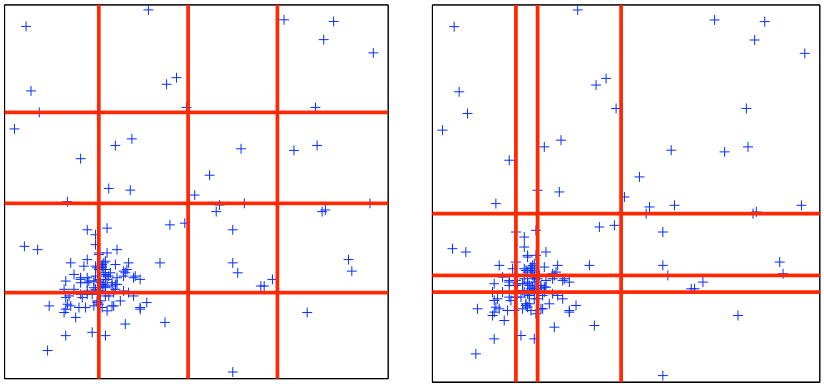


Figure 8.5: Example RCSs. **Left**: Standard RCS. **Right**: Learned RCS.

algorithm adjusts the bin boundaries to the regions of density.

Experimenting with RCS structures is somewhat challenging since there are two parameters to set (number of projections, number of boundaries), an approximation factor ϵ, and two quantities to compare (size and miss). We compare the size and miss rates of standard RCSs to tuned ones. We swept over the two parameters to get results for the standard RCSs. Results for learned RCSs were obtained using only a single (essentially unoptimized) setting of the parameters. Rather than minimizing a tradeoff between size_f and miss_f, we constrained the miss rate and optimized the size_f. The constraint was varied between runs (*e.g.* 2%, 4%, *etc.*) to get results comparable to the standard RCS runs.

Figure 8.6 shows the comparison on databases of 10k points drawn from the MNIST and Physics datasets (2.5k points were used as sample queries). We see a marked improvement for the Physics dataset and a small improvement for the MNIST dataset. We suspect that the learning algorithm helps substantially for the physics data because the one-dimensional projections are highly non-uniform whereas the MNIST one-dimensional projections are much more uniform.

8.5 Future work

There are many directions for future work using this framework. First, we expect that even stronger results may be obtained by using the framework to

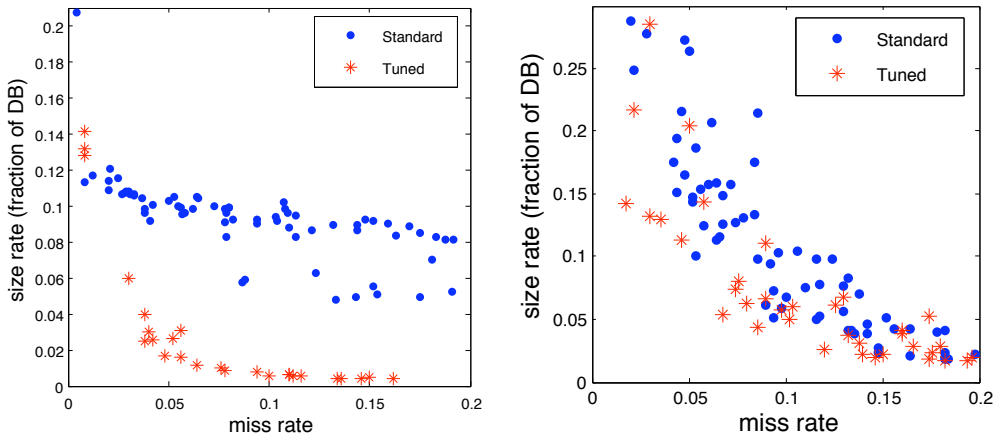


Figure 8.6: **Left**: Physics dataset. **Right**: MNIST dataset.

develop a novel data structure from the ground up, rather than to simply tune an existing data structure. Additionally, it would be interesting to implement the ideas of this framework for metric and ball trees. On the theoretical side, margin-based generalization bounds may allow the use of much richer classes of data structures.

The results of this chapter are joint work with Sanjoy Dasgupta. The dissertation author was the primary author and investigator.

L. Cayton, S. Dasgupta. A learning framework for nearest neighbor search. In *Advances in Neural Information Processing Systems 20*, 2007.

Bibliography

[AE99] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society, 1999.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the Symposium on Foundations of Computer Science*, 2006.

[AM00] Sunil Arya and David Mount. Approximate range searching. *Computational Geometry: Theory and Applications*, 2000.

[AMN+98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45(6):891–923, 1998.

[AW01] Katy Azoury and Manfred Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Machine Learning*, 43(3):211–246, 2001.

[BBL04] O. Bousquet, S. Boucheron, and G. Lugosi. Theory of classification: a survey of recent advances. *ESAIM: Probability and Statistics*, 9:323–375, 2004.

[BCP93] Hervé Brönnimann, Bernard Chazelle, and János Pach. How hard is half-space range searching? *Discrete and Computational Geometry*, 1993.

[BKL06] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.

[BL07] David Blei and John Lafferty. A correlated topic model of *Science*. *Annals of Applied Statistics*, 1(1):17–35, 2007.

[BMDG05] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *JMLR*, Oct 2005.

[BNJ03] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *JMLR*, 2003.

[Bre67] L.M. Bregman. 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(3):200–217, 1967.

[Cay08] Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *ICML*, 2008.

[Cay09] Lawrence Cayton. Efficient bregman range search, 2009. Submitted.

[CD08] Lawrence Cayton and Sanjoy Dasgupta. A learning framework for nearest neighbor search. In *Advances in Neural Information Processing Systems (NIPS) 20*, 2008.

[CH67] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[Cla85] Kenneth L. Clarkson. *Algorithms for Closest-point Problems*. PhD thesis, Stanford University, January 1985.

[Cla99] Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete and Computational Geometry*, 22:63–93, 1999.

[Cla06] K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[CLM08] Bernard Chazelle, Ding Liu, and Avner Magen. Approximate range searching in higher dimension. *Comput. Geom. Theory Appl.*, 39(1):24–29, 2008.

[CS07] M. Casey and M. Slaney. Fast recognition of remixed music audio. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2007.

[Csi91] Imre Csiszar. Why least squares and maximum entropy? an axiomatic approach to inference for linear inverse problems. *Annals of Statistics*, 1991.

[CZ97] Yair Censor and Stavros Zenios. *Parallel optimization: theory, algorithms, and applications*. Oxford University Press, 1997.

[DGL96] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, 2004.

[DKJ+07] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *ICML*, 2007.

[EGW+07] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 Results, 2007.

[FB74] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 1974.

[FBF77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[GBGM80] R. M. Gray, A. Buzo, A. H. Gray, and Y. Matsuyama. Distortion measures for speech processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1980.

[GIM07] Sudipto Guha, Piotr Indyk, and Andrew McGregor. Sketching information divergences. In *COLT*, 2007.

[GLS97] Adam J. Grove, Nick Littlestone, and Dale Schuurmans. General convergence results for linear discriminant updates. *Machine Learning*, pages 171–183, 1997.

[GM00] Alexander Gray and Andrew Moore. 'N-body' problems in statistical learning. In *NIPS*, 2000.

[GM03] Alexander Gray and Andrew Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining*, 2003.

[GRHS05] Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems 17*, 2005.

[Har01] Sariel Har-Peled. A replacement for voronoi diagrams of near linear size. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 94–103, 2001.

[HW98] Mark Herbster and Manfred K. Warmuth. Tracking the best regressor. In *COLT*, pages 24–31. ACM Press, 1998.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA, 1998. ACM Press.

[Ind06] P. Indyk. Nearest neighbors in high dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2006.

[JKDG08] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman. Online metric learning and fast similarity search. In *Advances in Neural Information Processing Systems 21*, 2008.

[KL04] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the 15th Annual Symposium on Discrete Algorithms (SODA)*, 2004.

[KOR00] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal of Computing*, 2000.

[KR02] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC)*, pages 741–750, 2002.

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[LMG06] Ting Liu, Andrew Moore, and Alexander Gray. New algorithms for efficient high-dimensional nonparametric classification. *JMLR*, 2006.

[LMGY04] T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate neighbor algorithms. In *NIPS*, 2004.

[LPP97] John Lafferty, Stephen Della Pietra, and Vincent Della Pietra. Statistical learning algorithms based on bregman distances. In *Proceedings of the Canadian Workshop on Information Theory*, 1997.

[LYRL04] D. D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 2004.

[MA] D. Mount and S. Arya. ANN library. `http://www.cs.umd.edu/~mount/ANN/`.

[Moo99] Andrew Moore. Very fast em-based mixture model clustering using multiresolution kd-trees. In *NIPS*, 1999.

[Moo00] A. W. Moore. Using the triangle inequality to survive high-dimensional data. In *UAI*, 2000.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[NBN07] Frank Nielsen, Jean-Daniel Boissonnat, and Richard Nock. On bregman voronoi diagrams. In *SODA*, pages 746–755, 2007.

[NJT06] Eric Nowak, Frederic Jurie, and Bill Triggs. Sampling strategies for bag-of-features image classification. In *ECCV*, 2006.

[NPB09] Frank Nielsen, Paolo Piro, and Michel Barlaud. Tailored bregman ball trees for effective nearest neighbors. In *European Workshop on Computational Geometry*, 2009.

[Omo89] S. Omohundro. Five balltree construction algorithms. Technical report, ICSI, 1989.

[PBRT99] Jan Puzicha, Joachim Buhmann, Yossi Rubner, and Carlo Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *ICCV*, 1999.

[PTL93] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *31st Annual Meeting of the ACL*, pages 183–190, 1993.

[RMV07] N. Rasiwasia, P. Moreno, and N. Vasconcelos. Bridging the gap: query by semantic example. *IEEE Transactions on Multimedia*, 2007.

[Roc70] R. Tyrrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 2000.

[Sam06] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[SNS06] Yirong Shen, Andrew Ng, and Matthias Seeger. Fast gaussian process regression using kd-trees. In *NIPS*, 2006.

[SP98] K.-K. Sung and T. Poggio. Example-based learning for view-based human face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1):39–51, Jan 1998.

[SV05] Eric Spellman and Baba Vemuri. Efficient shape indexing using an information theoretic representation. In *International Conference on Image and Video Retrieval*, 2005.

[SVD03] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2003.

[TdSL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 2000.

[Uhl91] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[WBS06] Kilian Weinberger, John Blitzer, and Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2006.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, 1998.

[XNJR03] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, 2003.

[Yia93] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, 1993.