Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Alejandro Mesejo-León

Approximate Nearest Neighbor Search for the Kullback-Leibler divergence

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Eduardo Sany Laber

Rio de Janeiro
January 2018

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Alejandro Mesejo-León

Approximate Nearest Neighbor Search for the Kullback-Leibler divergence

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the undersigned Examination Committee.

Prof. Eduardo Sany Laber
Advisor
Departamento de Informática – PUC-Rio

Dr. Alexandre Roberto Rentería
Jobzi

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática – PUC-Rio

Prof. Márcio da Silveira Carvalho
Vice Dean of Graduate Studies
Centro Técnico Científico – PUC-Rio

Rio de Janeiro, January 9th, 2018

Daniel Alejandro Mesejo-León

Graduated in Computer Science from the University of Havana (UH), Havana - Cuba in 2013. Joined the Master Program in Informatics at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in 2015.

Bibliographic data

Mesejo-León, Daniel Alejandro

Approximate Nearest Neighbor Search for the Kullback-Leibler divergence / Daniel Alejandro Mesejo-León; advisor: Eduardo Sany Laber. – Rio de janeiro: PUC-Rio, Departamento de Informática, 2018.

v., 59 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Engenharia Informática – Teses. 2. Divergencia Kullback-Leiber;. 3. Busca de Vizinhos Mais Próximos;. 4. Índices Invertidos;. 5. Hash sensível a localidade;. 6. Árvores de Bregman.. I. Laber, Eduardo Sany. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD:004

A Yeyi y a mi familia

Acknowledgments

First and foremost I would like to thank my advisor Eduardo Sany Laber for giving me the freedom to choose my own research topic and guiding me during the whole process. Also to my co-workers at Jobzi, specially Paula Guedes and Alexandre Renteria for their continuous support and advice. To my family for always been there, in particular to my parents Angela and Alejandro. Last but no least, I would like to thanks my wife Haydée for everything she does.

Abstract

Mesejo-León, Daniel Alejandro; Laber, Eduardo Sany (Advisor). **Approximate Nearest Neighbor Search for the Kullback-Leibler divergence**. Rio de Janeiro, 2018. 59p. Dissertação de mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In a number of applications, data points can be represented as probability distributions. For instance, documents can be represented as topic models, images can be represented as histograms and also music can be represented as a probability distribution. In this work, we address the problem of the Approximate Nearest Neighbor where the points are probability distributions and the distance function is the Kullback-Leibler (KL) divergence. We show how to accelerate existing data structures such as the Bregman Ball Tree, by posing the KL divergence as an inner product embedding. On the practical side we investigated the use of two, very popular, indexing techniques: Inverted Index and Locality Sensitive Hashing. Experiments performed on 6 real world data-sets showed the Inverted Index performs better than LSH and Bregman Ball Tree, in terms of queries per second and precision.

Keywords

Kullback-Leibler divergence; Nearest Neighbor Search; Inverted Index; Bregman Ball Tree; Locality Sensitive Hashing.

Resumo

Mesejo-León, Daniel Alejandro; Laber, Eduardo Sany. **Busca Aproximada de Vizinhos mais Próximos para divergência de Kullback-Leibler**. Rio de Janeiro, 2018. 59p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Em uma série de aplicações, os pontos de dados podem ser representados como distribuições de probabilidade. Por exemplo, os documentos podem ser representados como modelos de tópicos, as imagens podem ser representadas como histogramas e também a música pode ser representada como uma distribuição de probabilidade. Neste trabalho, abordamos o problema do Vizinho Próximo Aproximado onde os pontos são distribuições de probabilidade e a função de distância é a divergência de Kullback-Leibler (KL). Mostramos como acelerar as estruturas de dados existentes, como a Bregman Ball Tree, em teoria, colocando a divergência KL como um produto interno. No lado prático, investigamos o uso de duas técnicas de indexação muito populares: Índice Invertido e Locality Sensitive Hashing. Os experimentos realizados em 6 conjuntos de dados do mundo real mostraram que o Índice Invertido é melhor do que LSH e Bregman Ball Tree, em termos de consultas por segundo e precisão.

Palavras-chave

Divergencia Kullback-Leiber; Busca de Vizinhos Mais Próximos; Índices Invertidos; Hash sensível a localidade; Árvores de Bregman.

Table of contents

List of figures

List of tables

1
Introduction

Nearest neighbor search (NNS) is one of the most important optimization problems, with applications in a wide range of fields including, but not limited to, machine learning, information retrieval and pattern recognition. Loosely speaking, the problem is that of finding the point in a given set that is closest (most similar) to a given point. The measure of closeness is, in general, a metric distance or a well defined similarity. One of the first mentions of this problem is attributed to Donald Knuth in vol. 3 of The Art of Computer Programming (1973), who called it the post-office problem, referring to an application of assigning the nearest post office to a residence. Concrete examples of NNS problems, in the above mentioned fields are k-nearest neighbors classification and regression, collaborative filtering, content-based image retrieval, coding theory, among others.

There are many versions of the NNS problem, one of the most well-known is the fixed-radius nearest neighbor (r-NN problem): given a query q, a set of points P, a distance $dist$ and a radius r the goal is to find $p_i \in P$ so that $dist(p, q) \leq r$. Another related version is the k nearest neighbor search (k-NNS), or in a more formal manner: let $\mathbb{U}$ be the universe of points, $P = \{p_1 \dots p_n\} \subset \mathbb{U}$, a query q and a distance function $dist : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$, the problem consists of finding k points p_i for which $dist(p_i, q)$ is minimum. If defined with respect to a similarity function $s : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$ the problems becomes the Top-k most similar problem (Top-k).

Obviously, these problems can be solved iterating over the entire dataset P and computing the distance from q to p_i for each p_i and keeping those points that are below the radius r or are in the set of k nearest neighbors. This results in a complexity of $O(mnd)$ where $m = |Q|$ is the cardinality of the set of queries, $n = |P|$ and d is the dimension of the points in $P \cup Q$. This complexity assumes that calculating the distance between two points is proportional to d. So the challenge is how to preprocess (index) P so it can support multiple queries in a sub-linear fashion.

The k-NNS and r-NNS problems are well solved for spaces of 2 or 3 dimensions. However extending these results to higher dimensions (of about 100), have proven to be a difficult task. This together with the fact that in most

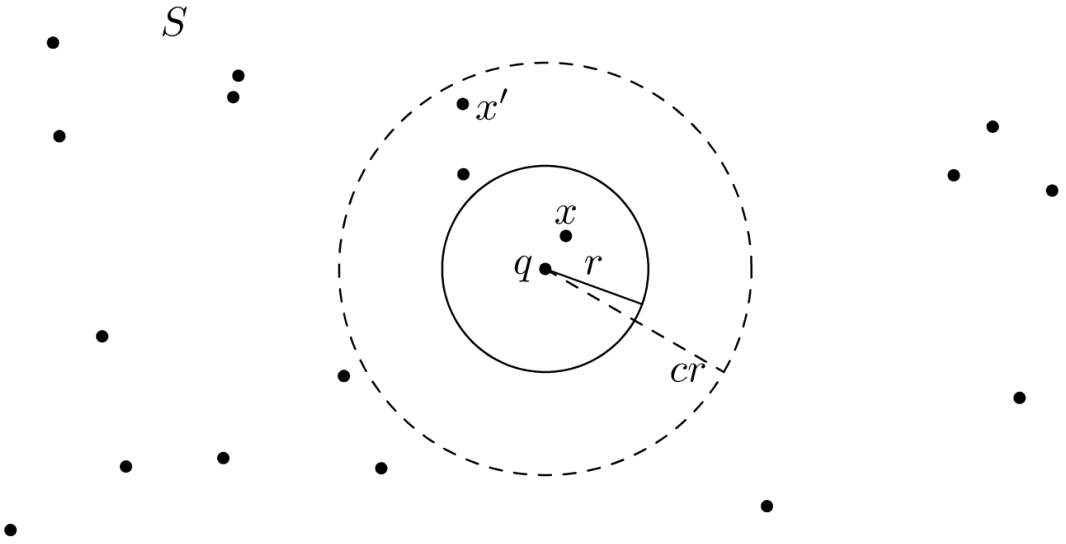


Figure 1.1: Representation of the r-NN and (c, r)-NN problems

practical scenarios the input data is noisy have motivated the focus on solving approximate versions of these problems where returning an approximate solution is acceptable.

A possible formalization for the approximate nearest neighbor search (ANNS) problem is given below:

Definition 1.1 *(ϵ-Near Neighbor Search Problem) Given a distance function dist, a set of points P in a space $(\mathbb{U}, dist)$ and $\epsilon > 0$, the problem consists of preprocessing P so as to efficiently return a point $p \in P$ for any given query point q, such that $dist(q, p) \leq (1+\epsilon)dist(q, p^*)$, where $dist(q, p^*)$ is the distance of q to its closest point in P.*

This easily generalizes to the k-NNS by setting p^* to be the k-th nearest neighbor. The approximate version of the r-NN problem is known as the (c, r)-NN problem, simply put, given a query q return all neighbors where $dist(q, p) \leq cr$. Figure 1.1 shows an example of r-NN and (c, r)-NN, in the r-NN version only x is allowed as an answer, while in (c, r)-NN both x and x' are possible answers. Finally both instances of the problem admit randomized versions, i.e. given a query q report an accurate answer with probability $1 - \delta$ where $0 \leq \delta \leq 1$.

1.1 Probability Distributions and the Kullback-Leibler divergence

Most indexing schemes in the literature implicitly or explicitly make use of metric properties like triangle inequality and symmetry to deliver good performance. However, it is often the case in real world applications, that the

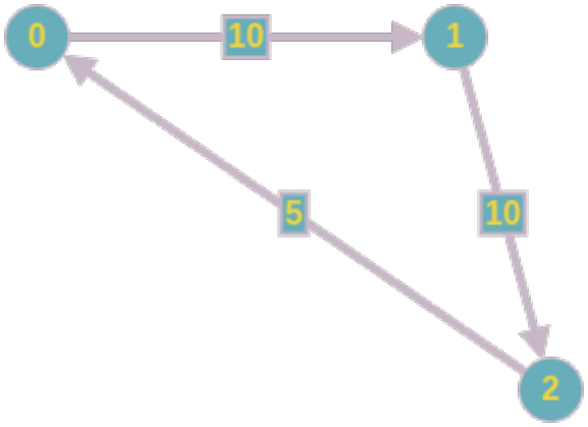


Figure 1.2: An image of simple graph where the distance between the nodes is not symmetric

distance measure of choice is not a metric. Such measures arise when comparing probability distributions, time series, images and matrices among others.

A very basic example is that of a directed graph D such as the one in Figure 1.2, where the distance from node 0 to node 2 is the sum of the edges $(0, 1), (1, 2)$ and the distance from node 2 to node 0 is given by the edge between them. The notion of distance in this context violates the triangular inequality, as well as the symmetry of metric distances. Moreover it illustrates the richness offered by such type of non-metric dissimilarity measures especially on complex data types.

Probability distributions can be naturally used to represent data in many applications. For instance, documents (20) (19), images and music (28) have been represented as probability distributions. A popular dissimilarity measure in information theoretic settings where probability distributions are evaluated in terms of their entropy or the amount of information they contain is the Kullback-Leibler (KL) divergence (26). This divergence measures the difference between the relative entropy and the self entropy of two distributions, moreover it has been extensively used in the context of information retrieval (30), texture similarity search (31), clustering (29) among others. In some tasks, it has proven its effectiveness over other dissimilarity measures such as the euclidean distance (30).

Given two (normalized) histograms $p = (p_1, p_2, \ldots, p_d)$ and $q = (q_1, q_2, \ldots, q_d)$ the Kullback-Leibler divergence between the two distributions is defined as:

Author	AI	Application	Bioinformatics	Database	Hardware	Software	System	Theory
Jim Gray	0.109	0.109	0.059	0.314	0.0987	0.091	0.123	0.093
R1	0.141	0.101	0.069	0.276	0.094	0.089	0.123	0.103
R2	0.1	0.1	0.1	0.299	0.1	0.1	0.1	0.1

Table 1.1: Each row represents an author as a probability vectors over major computer science topics. This table was originally presented in (9)

$$KL(p,q) = \sum_{i=1}^{d} p_i \ln \frac{p_i}{q_i} - p_i + q_i \tag{1-1}$$

$$= \sum_{i=1}^{d} p_i \ln \frac{p_i}{q_i} - \sum_{i=1}^{d} p_i + \sum_{i=1}^{d} q_i \tag{1-2}$$

$$= \sum_{i=1}^{d} p_i \ln \frac{p_i}{q_i} \tag{1-3}$$

To illustrate the effectiveness of KL-divergence, let us borrow a concrete example from Zhang et. al (9). In Table 1.1, three vectors generated from the DBLP data set are presented, with each representing an author. Each dimension of the vector represents the probability of the author publishing in a major field of computer science. From the record of Jim Gray, it is apparent that he is closely connected to database community.

If one were to use the Euclidean distance to search for an author similar to Jim Gray, the distances between Jim Gray and authors R1, R2 are exactly the same. However, when the KL-divergence is applied the distances from Jim Gray to R1 and R2 become 0.008 and 0.01 respectively implying that R1 is more similar to Jim Gray than R2. This can be explained by the fact that both Jim Gray and R1 have a higher probability of publication on System and lower probability of publication on Bioinformatics while R2 have equal probability to publish in almost all the fields except Database. KL-divergence generally captures such kinds of dissimilarities between distributions more accurately than Euclidean distance, since it gives higher weights to dimensions with larger probabilities.

1.2 Our Contributions

In this work, we address the ANNS problem where the points are probability distributions and the distance function is the KL divergence. Using the fact that the Kullback-Leibler can be expressed as an inner product (9) and the recent developments of Neyshabur and Srebro (8) we show that the (one sided) KL-divergence can be embedded into the Euclidean distance. Expressing the KL-divergence as an inner product provides a natural way of using indexing

methods like Locality Sensitive Hashing (1) (LSH) and Inverted Index (43). We explore these ideas here by comparing both LSH and Inverted Index data structures with Bregman Ball Trees (BBTree) over 6 real-world datasets. The experiments showed the Inverted Index performs better than LSH and Bregman Ball Tree, in terms of queries per second while achieving comparable precision measurements. On the theoretical side, we show how to obtain an ordinal embedding that leads to a theoretically faster search and construction procedures for the Bregman Ball Tree.

The organization of the dissertation is as follows. In Chapter 2, we describe the mathematical preliminaries that set the theoretical framework of this work. Chapter 3 discusses, in detail, all the three mentioned indexing methods and overviews related work. The theoretical analysis is presented in Chapter 4. In Chapter 5, we describe experimental evaluations on several real-world data sets. Finally, in Chapter 6, we summarize the main contributions and discuss some research directions concerned with a generalization of the KL divergence, the Bregman divergences.

2
Mathematical Preliminaries

In this chapter we introduce the main concepts needed to understand the rest of the work, discuss some properties that will be useful in the chapters ahead and develop the terminology and main notation that we will refer back throughout this dissertation.

2.1
Basic Notation

For vectors $p, q \in \mathbb{R}^d$ we use p_i (respectively q_i) to denote the value entry i of the vector, $\mathbf{p}$,$\mathbf{q}$ (bold) will be used to refer as a unique identifier of the vectors p, q. For a set of vectors we will use upper case letters such as P, S, M, in some cases it will be useful to think of these sets as matrices, for instance as $P^{n \times d}$ where $n = |P|$, i.e. the elements of the set are the rows of the matrix. The l^2 norm of the vector will be $\|p\|_2$ or more simply $\|p\|$, while the l^1 norm will be $|p|$. Unless otherwise stated we will assume the vectors are normalized with respect to the l^1 norm, and the term normalized histogram, or probability distribution will be used to refer to those vectors.

2.2
Some Properties of the Kullback-Leibler Divergence

The KL divergence belongs to the class of f-divergences and Bregman divergences (26). The Bregman divergence $D_f(x, y)$ associated with function f on points $x, y \in \mathbb{R}^d$, is the difference between the value of f at point x and the value of the first-order Taylor expansion of f around point y evaluated at point x:

$$D_f(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle$$

It follows that the generalized Kullback-Leibler divergence is generated by:

$$f(x) = \sum x_i \log x_i$$

Despite being positive and fulfilling the identity of indiscernibles ($KL(p, q) = 0 \Leftrightarrow p = q$), is not symmetric ($KL(p, q) \neq KL(q, p)$) and unbounded, i.e., for any given $c > 0$, one can construct histograms whose

Kullback-Leibler divergence exceeds c. In order to avoid these singularities, we assume that the histograms are β-constrained. In the discussion below we assume the histograms to be normalized.

Definition 2.1 *(α-constrained histogram). A histogram $p = (p_1, p_2, \ldots, p_d)$ is said to be α-constrained if $p_i \geq \frac{\alpha}{d}$ for $i = 1, 2, \ldots, d$*

For convenience, we will denote $\frac{\alpha}{d}$ by β. A d-dimensional β-constrained distribution will then imply a distribution that is α-constrained with $\alpha = \beta d$

Proposition 2.2 *Given two β-constrained histograms p, q, $0 \leq KL(p, q) \leq \ln \frac{1}{\beta}$*

Proof. The lower bound follows directly from Jensen inequality (40). For the upper bound, since we know that $\frac{p_i}{q_i} \leq \frac{1}{\beta}$ for all $i = 1, 2, \ldots, d$ we can write:

$$KL(p, q) = \sum_{i=1}^{d} p_i \ln \frac{p_i}{q_i} \leq \sum_{i=1}^{d} p_i \ln \frac{1}{\beta} = \ln \frac{1}{\beta}$$

For $\beta = O(\frac{1}{d})$, the upper bound is tight up to a constant factor

■

One of the major problems of the KL-divergence is the lack of symmetry, which induces the definition of two NN problems (10):

- (left-NN) return $\min_{p \in P} D_f(p, q)$
- (right-NN) return $\min_{p \in P} D_f(q, p)$

Luckily the Bregman divergences, including the KL-divergence, satisfy a duality property with respect to the convex conjugate. The convex conjugate of a function $f : X \to \mathbb{R} \cup \{+\infty\}$ can be defined defined in terms of the supremum by:

$$f^*(x^*) = \sup\{\langle x^*, x\rangle - f(x) | x \in X\}$$

where X is a real topological vector space, X^* is the dual space to X and $\langle \cdot, \cdot \rangle : X^* \times X \to \mathbb{R}$ is a dual pairing. In the case of the Bregman divergences the dual space $x^* \in X^*$ is given by $x^* = \nabla f(x)$. Now we can state the following proposition:

Proposition 2.3 *Let f^* the convex conjugate of f then:*

$$D_f(q, p) = D_{f^*}(p^*, q^*)$$

Proof. (10) The convex conjugate of f is defined as $f^* \equiv \sup_x\{\langle x, y\rangle - f(x)\}$. 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 $d_f(\cdot, \cdot)$:

$$\begin{aligned} D_f(x, y) &= f(x) - f(y) - \langle y', x - y\rangle && (2\text{-}1)\\ &= f(x) + f^*(y') - \langle y', x\rangle && (2\text{-}2)\\ &= D^*_f(y', x') && (2\text{-}3) \end{aligned}$$

■

This relationship can be used to construct data structures that only solve one of the problems of the Bregman divergences. Another interesting property of the Bregman divergence concerns centroids. For a set of points, the mean under any Bregman divergence is well defined and, interestingly, is independent of the choice of divergence:

$$\mu_X \equiv argmin_\mu \sum_{x\in X} D_f(x, \mu) = \frac{1}{|X|}\sum_{x\in X} x$$

This can be used to extend the well-known k-means clustering algorithm to the family of Bregman divergences (54).

2.3 Inner Product and Cosine Similarity

By inner product of two vectors $p = (p_1, \ldots, p_d), q = (q_1, \ldots q_d) \in \mathbb{R}^d$ we will be referring to the following scalar quantity:

$$\langle p, q\rangle = \sum_{i=1}^{d} p_i q_i$$

The above definition is equivalent to:

$$\langle p, q\rangle = cos(p, q)\|p\|\|q\|$$

This quantity takes into account both the orientation and the magnitude of the vectors. When the vectors are normalized ($\|p\| = 1$), the quantity represents the cosine similarity that measures the cosine of the angle between the vectors. For example, two vectors forming an angle of 0° have a similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1. In general cosine similarity is used in positive

space, where the outcome is bounded in $[0, 1]$. Simply, the cosine similarity is as follows:

$$cos(p, q) = \frac{\langle p, q \rangle}{\|p\|\|q\|}$$

The inner product and therefore the cosine similarity meet the following property, let p' be the prefix of length k of p and p'' the suffix of length $d - k$ i.e. $p' = (p_i, \ldots, p_k, 0, \ldots, 0)$ and $p'' = (0, \ldots, 0, p_{k+1}, \ldots, p_d)$

$$\langle q, p \rangle = \langle q, p' \rangle + \langle q, p'' \rangle$$

for every index k. As a final note, the version of the ANN for cosine similarity is known as Cosine Similarity Search (CSS), that is given $q \in \mathbb{R}$ one seeks for $p_i \in P$ such that $cos(p_i, q)$ is maximum. Similarly, the Maximum Inner Product Search (MIPS) problem is the version of the NN problem where the similarity function is the inner product. More formally, given a collection of vectors $\mathcal{S} \subset \mathbb{R}^d$ and a query $q \in R^d$, find a vector $p \in \mathcal{S}$ such that

$$p = \arg\max_{x \in \mathcal{S}} q^T x.$$

Notice that the while $cos(q, q) = 1$ is maximum, this do not imply that the inner product of q with itself is maximum. For example take the vectors $v = (1, 1)$ and $u = (2, 2)$ so we have that $cos(v, u) = cos(v, v) = 1$ but using inner product we have $\langle u, u \rangle = 1 < \langle u, v \rangle = 4$, this makes MIPS a considerably harder problem that CSS.

3
Overview of Indexing Methods

In this chapter, we overview the inner workings of three indexing methods: Bregman Ball Tree (10), Locality Sensitive Hashing (1) and Inverted Index (43). In the last part of the chapter we review previous work on indexing the Kullback-Leibler divergence, or the more general class of Bregman Divergences.

3.1
Bregman Ball Tree

In this section we detail the Bregman Ball Tree, to keep the discussion in agreement with the original paper we will review the method using the Bregman divergence, the analysis also sustains for the KL-divergence. The Bregman Ball Tree (BBTree) was proposed by Cayton (10), the method is an extension of Ball Tree for Bregman Divergences. It partitions the search space using a Bregman Ball. Recalling, a Bregman divergence can be defined as follows:

$$D_f(x, y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle$$

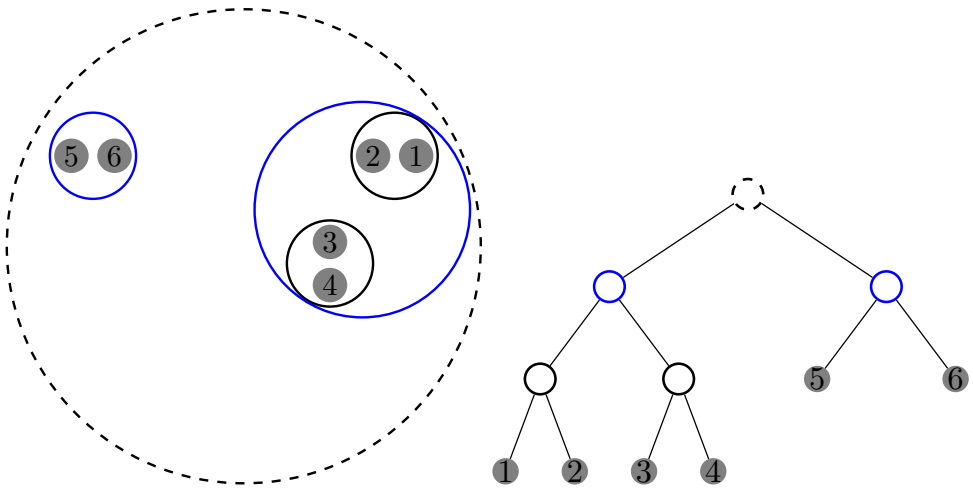
where f is a strictly convex differentiable function, in the case of the Kullback-Leibler divergence this function is $\sum x_i \log x_i$. A Bregman divergence $D_f(x, y)$ is convex in x, but not necessarily in y.

A Bregman Ball of radius R around some point μ is defined as:

$$B(\mu, R) = \{x : D_f(x, \mu) \leq R\}$$

Given that $D_f(x, \mu)$ is convex in x, $B(\mu, R)$ is a convex set. The search algorithm of the BBTree uses the same underlying principle as its metric counterparts, but instead of metric balls, the fundamental geometric object that creates the hierarchical space partition is a Bregman ball.

A space partition is a binary tree where each node n_i is associated with a subset of the database $P_i \subset P$. The subset P_i contains those elements that are inside a Bregman ball $B(\mu_i, R_i)$ with center μ_i and radius R_i. Each non-leaf node has two children nodes, left (l) and right (r). Each of the points $p \in P_i$ is in l or r and appears in exactly one of P_l or P_r.



3.1(a): Point Clusters 3.1(b): Tree Representation

Figure 3.1: On the left side an example of a hierarchical partition of the space by a Bregman Ball Tree, on the right side a tree representation of the hierarchical partition

Although the sets P_l and P_r are disjoint, the associated balls $B(\mu_l, R_l)$ and $B(\mu_r, R_r)$ may overlap. The entire set is encapsulated in the root node, each leaf is supposed to cover a small fraction of P and the set of all leaves covers P as a whole.

3.1.1 BBTree construction

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 there is a clear separation of the balls, a query is likely to be much closer to one than the other. Compactness means, that the distance from a query point q to a ball will be a good proxy of the distance from q to the nearest point within the ball. Therefore at each level we'd like to split the points into compact, well-separated sets.

A rather straightforward way to do this is to use k-means, which has already been extended to Bregman divergences (54). The algorithm for building the tree follows a top-down approach. Starting at the top, the algorithm runs k-means to partition the points into two clusters. This process is repeated recursively for each subset until a threshold t is reached on the numbers of points on a leaf node. For a fixed number of iterations r the complexity of k-means is $O(rnd)$, given the recursive fashion of the algorithm the total complexity is roughly $O(rnd \log n)$.

For instance, take as an example the points in Figure 3.1, the algorithm first creates a global cluster represented by the dashed ball, it then divides the

cluster in two smaller clusters, represented as the two blue clusters. In one case it reaches the minimum number of samples (the points 5 and 6) and stops the procedure. The other branch continues, and subdivide again the cluster again creating two leaf nodes. The right side of 3.1 is a graphical representation of the binary search tree generated from the hierarchical partition.

3.1.2 Search on a BBTree

This subsection describes the exact retrieval process of a query's nearest neighbor with a BBTree. Let $P = \{p_1, \ldots, p_n\}$ be the set of points in the database, q is a query and D_f is a (fixed) Bregman divergence, in our case the Kullback-Leibler divergence. All the analysis will be done with respect to the left NN $p = argmin_{p_i \in P} D_f(p, q)$.

The tree is traversed using a branch and bound routine; at each node, the search algorithm chooses the child for which $D_f(u, q)$ is smallest and temporarily ignores the sibling node. This is repeated until reaching a leaf node n_i. Once in n_i the algorithm then calculates $D_f(p, q)$ for all $p \in P_i$, the closest point is the candidate p_c. From this node the algorithm backtracks up the tree and considers the previously ignored sibling, it explores the sibling node n_j if

$$D_f(p_c, q) > min\{D_f(x, q) | x \in B(\mu_j, R_j)\},$$

otherwise the node n_j and all of its children can be ignored since the NN cannot be found in that subtree. The algorithm hinges on the computation of the Bregman projection onto a Bregman ball, since general Bregman divergences do not satisfy the triangle inequality, the main technical contribution of Cayton (10) is a fast way of computing the right side of the previous equation, which is a convex program:

$$\begin{array}{rll} \min\limits_{x} & D_f(x, q) & \\ \text{s.t.} & D_f(x, \mu) & \leq \quad R \end{array} \tag{3-1}$$

The search algorithm will need to solve (3-1) many times in the course of locating q's NN, so to compute a solution very quickly is an imperative requirement. Let x_p denote the optimal solution to 3-1 and $x'_p \equiv \nabla f(x_p)$. The main result of (10) is summarized in the following theorem, recalling that f^* is the convex conjugate of f, defined in Section 2.2:

Theorem 3.1 *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.

The details of the proof can be found in the original work of Cayton (10).

3.1.2.1 Approximate Search

To determine if a node must be explored during the backtracking phase we must solve (3-1). One can evaluate the right side of (3-1), using the bisection method described previously but an exact solution is not needed. Suppose we have bounds a and A satisfying:

$$a \leq \min_{x \in B(\mu,R)} D_f(x,q) \leq A$$

If $D_f(x_c,q) \leq a$ the node can be pruned; if $D_f(x_c,q) \geq A$ the node must be explored, the search proceeds until $D_f(x_c,q) \leq a$ or $D_f(x_c,q) \geq A$. We now describe how to compute the lower and upper bounds at each step of the bisection search. Throughout the discussion, we use $q' \equiv \nabla f(q)$, $\mu' \equiv \nabla f(\mu)$ to simplify notation. Let $x_\theta = \nabla f^*(\theta\mu' + (1-\theta)q')$ and define Lagrange dual function as

$$L(\theta) \equiv D_f(x_\theta, q) + \frac{\theta}{1-\theta}\Big(D_f(x_\theta,\mu) - R\Big) \tag{3-2}$$

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

$$L(\theta) \leq \min_{x \in B(\mu,R)} D_f(x,q) \tag{3-3}$$

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

$$D_f(x_\theta, q) \geq min_{x \in B(\mu,R)} D_f(x,q) \tag{3-4}$$

Let us now put all of the pieces together. We wish to evaluate whether (3-1) 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 given by the dual function $L(\theta_i)$ defined in (3-2). If $L(\theta_i) > D_f(x_c,q)$, then the node can be pruned. Otherwise, if $x_{\theta_i} \in B(\mu,R)$, we can update the upper bound. If $D_f(x_{\theta_i},q) < D_f(x_c,q)$, then the node must be searched. Otherwise, neither bound holds, so the bisection search continues. For a detailed pseudo-code of the algorithm, called *CanPrune*, see Algorithm 3.1.1. This algorithm is executed during the backtrack phase of the search algorithm of the BBTree. The first time the algorithm is called x_c is the best point found so far, q is the query, μ and R are the center and radius of the sibling node.

Algorithm 3.1.1: Can prune node (CanPrune)

input : $\theta_l, \theta_r \in [0, 1), q, x_c, \mu \in \mathbb{R}^d, R$
output: If the node must be explored or not

Set $\theta = \frac{\theta_l + \theta_r}{2}$
Set $x_\theta = \nabla f^*(\theta\mu' + (1-\theta)q')$
if $L(\theta) \geq D_f(x_c, q)$ **then**
  return YES
end
if $x_\theta \in B(\mu, R)$ *and* $D_f(x_\theta, q) \leq D_f(x_c, q)$ **then**
  return NO
end
if $D_f(x_\theta, \mu) \geq R$ **then**
  return $CanPrune(\theta_l, \theta, q, x_c, \mu)$
end
if $D_f(x_\theta, \mu) \leq R$ **then**
  return $CanPrune(\theta, \theta_r, q, x_c, \mu)$
end

The complexity of the exact search algorithm is $O(nd + od\log(1/\epsilon))$, where o is the number of nodes, so it follows that the complexity of the worst case for the search algorithm of BBTrees exceeds the simple linear search. A simple way to speed up the retrieval time of the BBTree is to simply stop after only a few leaves have been examined. This idea originates from the empirical observation that ball trees often locate a point very close to the NN quickly, then spend most of the execution time backtracking. This reduces the complexity significantly, at the cost of precision, therefore solving the ANNS version of the problem.

3.2 Locality Sensitive Hashing

In this section we discuss, in depth, the indexing method used by Locality Sensitive Hashing. The concept of LSH was introduced by Indyk et al. (5) with the motivation of solving ANNS problems. Here, we use a more general definition of LSH proposed in (8).

Definition 3.2 *Let $c > 1$ be a constant, let r be a threshold and let $0 \leq p_2 < p_1 \leq 1$ be some pair of probabilities. A pair (F, G) of families of functions h, g are said to be $(r, c \cdot r, p_1, p_2)$-LSH for a distance dist over the pair of spaces $\mathcal{X}$ and $\mathcal{Y}$ if for $q \in \mathcal{X}$ and $p \in \mathcal{Y}$*

- *if $dist(p, q) \leq r$ then $P_{(F,G)}[h(p) = g(q)] \geq p_1$*
- *if $dist(p, q) \geq c \cdot r$ then $P_{(F,G)}[h(p) = g(q)] \leq p_2$*

L	k	p_1	p_2
4	4	0.8784974493	0.0063846564
2	2	0.8704	0.0784
3	3	0.883785728	0.023808512
4	4	0.8784974493	0.0063846564
4	5	0.7956831789	0.0012793857
4	6	0.7035943671	0.0002559754

Table 3.1: Effect of amplification on probabilities, original probabilities are $p_1 = 0.8$ and $p_2 = 0.2$

This definition differs from the one in (5) because the distance function is defined over a pair of spaces rather than a single space. The motivation is to distinguish between the query space and the dataset space, which will be useful in the discussion of the Maximum Inner Product Search (MIPS) problem in Section 3.2.2.

The probabilities p_1, p_2 in the definition are considered with respect to the random choice of the pair of functions (h, g) from the families F and G. Clearly, the choice of the families of functions F and G depends on the distance function. A pair of families is useful when $p_1 > p_2$. Alternatively a LSH can be defined with respect to a universe of items U and a similarity function $sim : U \times U \to [0, 1]$ in the following manner:

Definition 3.3 *Let $0 \leq c < 1$ be a constant, let $S \in (0, 1]$ be a similarity threshold and let $0 \leq p_2 < p_1 \leq 1$ be thresholds. A pair of families F, G of functions h, g are said to be (S, cS, p_1, p_2)-LSH for a similarity sim over the pair of spaces $\mathcal{X}$ and $\mathcal{Y}$ if for $q \in \mathcal{X}$ and $p \in \mathcal{Y}$*

- *if $sim(p, q) \geq S$ then $P_F[h(p) = g(q)] \geq p_1$*
- *if $sim(p, q) \leq cS$ the $P_F[h(p) = g(q)] \leq p_2$*

Ideally one would expect to have p_1 high (close to 1) and p_2 very low (close to 0), but in general this is not possible, Figure 3.2 shows a comparison of the ideal versus the commonly obtained pair of probabilities. The ideal pairs of probabilities will be close to 1 for pairs of objects that are very close, and at the same time sharply fell for no so close objects, this is shown on the left side of Figure 3.2. In general, the design of such family of functions is elusive and one ends up with a pair of probabilities that resemble that of the right part of Figure 3.2. This problem is overcome composing an LSH family several times and using AND and OR constructions to amplify the original probabilities as shown in the next theorem.

Theorem 3.4 *Let $h \in H$ and $g \in G$ be two locality sensitive family of functions, $\mathbf{h} \in \mathbf{H}$ the family of functions obtained by concatenating k hash functions h and $\mathbf{g} \in \mathbf{G}$ equally for G, i.e. $\mathbf{h}(x) = [h_1(x)h_2(x)\dots h_k(x)]$ and $\mathbf{g}(x) = [g_1(x)g_2(x)\dots g_k(x)]$, where $\mathbf{h}(p) = \mathbf{g}(q)$ if and only if $h_i(x) = g_i(y) \forall i$. If H is $(r, c \cdot r, p_1, p_2)$ then $\mathbf{H}$ is $(r, c \cdot r, p_1^k, p_2^k)$*

Concatenating functions shrinks the probability of collision but choosing k properly can make the lower probability p_2 approaches 0 while the higher does not. Similarly, one could concatenate the functions and allow to be equals if any of the functions is equally, precisely;

Theorem 3.5 *Let $h \in H$ and $g \in G$ be two locality sensitive family of functions, $\mathbf{h} \in \mathbf{H}$ the family of functions obtained by concatenating L hash functions h and g, i.e. $\mathbf{h}(x) = [h_1(x)h_2(x)\dots h_L(x)]$ and $\mathbf{g}(x) = [g_1(x)g_2(x)\dots g_L(x)]$, where $\mathbf{h}(p) = \mathbf{g}(q)$ if and only if $h_i(x) = g_i(y)$ for some i. If H, G are $(r, c \cdot r, p_1, p_2)$ then the pair $\mathbf{H}, \mathbf{G}$ is $(r, c \cdot r, 1-(1-p_1)^L, 1-(1-p_2)^L)$*

The construction built in Theorem 3.5 makes all the probabilities grow, but by choosing L correctly p_1 can approach 1 while p_2 does not. It follows from Theorem 3.4 and Theorem 3.5 that each of the two probabilities p can be transformed into $1-(1-p^k)^L$. Simply take H and construct H' using Theorem 3.4 then from H', construct $\mathbf{H}'$ by Theorem 3.5.

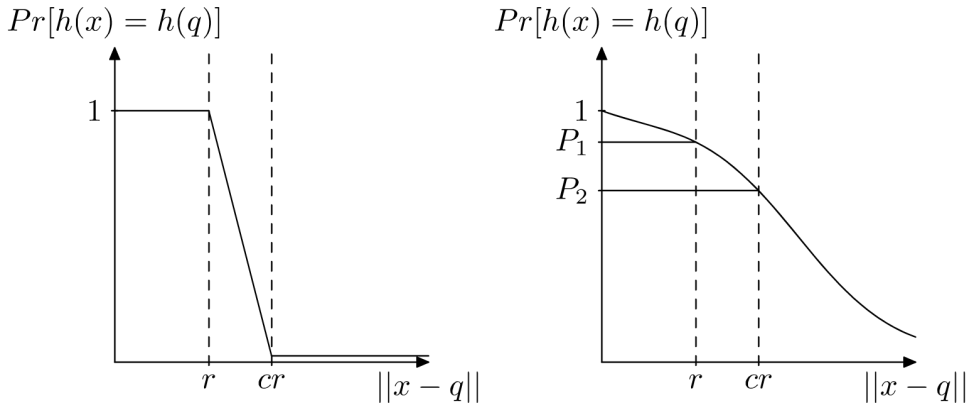
The effects of amplification can be seen in Table 3.1. Suppose we have a pair of families H, G such that $p_1 = 0.8$ if $dist(p, q) \leq r$ and $p_2 = 0.2$ for $dist(p, q) \geq c \cdot r$ for some c and r. The original probabilities in the example of Table 3.1 are $p_1 = 0.8$ and $p_2 = 0.2$. Applying theorems 3.4 and 3.5, with $k = 4$ and $L = 4$ for example, we have: ${p_1}^* = 1 - (1 - {p_1}^4)^4$, by substitution ${p_1}^* \approx 0.878$, analogously ${p_2}^* = 1 - (1 - {p_2}^4)^4 \approx 0.0063$. While the chosen pair of probabilities will be application dependent, one can perform some analysis to come out with good approximations for k and L.

Notice that we expect $np_{2,h} = np_2^k$ far points collisions, where n is the cardinality of $|P|$. So, we want $np_2^k = O(1)$. Thus we take $p_2^k = 1/n$ so that the expected number of far points encountered is 1. This implies:

$$p_2^k = 1/n \Rightarrow k \log(1/p_2) = \log n \Rightarrow k = \frac{\log n}{\log(1/p_2)}$$

Now for each table (hash function) we have a success probability of p_1^k, that is the probability of a good collision. If we have L tables in total, taking an union bound, we want to choose $L = O(1/p_1^k)$. For this choice we get:

$$p_1^k = p_1^{\frac{\log n}{\log(1/p_2)}} = n^{-\frac{\log 1/p_1}{\log 1/p_2}}$$



3.2(a): Ideal probabilities

3.2(b): Common probabilities

Figure 3.2: A comparison of the ideal pair of probabilities vs the commonly obtained. The pair of probabilities is defined with respect to some distance function.

Given a LSH function family one can build a data structure for efficient ANNS by using the algorithm specified in Algorithm 3.2.1. We first set k and L in line 1, and then for each l we build a hash table T_l to store the points. We then pick k hash functions from F, line 4. Based on the k hash functions we assign a hash code $\boldsymbol{h}$ to each point, lines 5-9. Finally we store each p in T_l using $\boldsymbol{h}$ as a key. Actually, the value of each key is a sub-collection of points p, given that two points p_i and p_j can have the same hash code.

Algorithm 3.2.1: Preprocessing of the set P

```
   Result: Preprocess P
 1 Set k and L ;
 2 for l = 1 ... L do
 3 |   Initialize a hash table T_l
 4 |   Pick k hashing functions h_i from F randomly
 5 |   for p ∈ P do
 6 |   |   Set h = []
 7 |   |   for h_{i=1...k} do
 8 |   |   |   h[i] = h_i(p)
 9 |   |   end
10 |   |   Store p in T_l using h as key
11 |   end
12 end
```

Once the set P is preprocessed, to answer a query q we simply retrieve the collection K of items in each of the $\boldsymbol{h}_1(q), \ldots, \boldsymbol{h}_L(q)$. An alternative ending criterion is to stop searching after finding the first $3L$ points (5), (including duplicates). Then to refine the results an exhaustive search is performed on S to retrieve the ϵ-Nearest Neighbors. We refer to the full search version of the

algorithm as **Strategy 1** and to the second as **Strategy 2**; the algorithm for **Strategy 1** is detailed in Algorithm 3.2.2.

Algorithm 3.2.2: Retrieval of ϵ-Nearest Neighbors

input : A query point q
output: The set K of the ϵ-Nearest Neighbors

```
S = ∅
for l = 1 ... L do
    Set h = []
    for h_{i=1...k} do
        h_l[i] = h_i(q)
    end
    K = S ∪ T_l[h_l]
end
Return the n nearest neighbors in K as the ϵ-Nearest Neighbors
```

Given an $(r, c\dot{r}, p_1, p_2)$-LSH, and using algorithm 3.2.1 it is possible to construct a data structure that can be used to separate, with high probability, pairs of points with distance at most r from those with distance at least $c \cdot r$. Such a data structure can be constructed in time $O(n^{\rho} \log n)$ and space $O(n^{1+\rho})$ (5), where $\rho = \frac{\log p_1}{\log p_2}$. For the search **Strategy 2**, we have a query time complexity of $O(n^{\rho} d)$, in expectation.

3.2.1 On LSH for solving Cosine Similarity Search

Due to the multiple applications of Cosine Similarity Search (CSS), the problem has received considerable attention. The first to propose the use of LSH for solving CSS was Charikar in (3). There the author proposed a family of functions F for dealing with the cosine similarity. The approach is based on the random hyper-plane rounding technique proposed in the seminal work of (2) for approximating the $MAX - CUT$ problem.

Given a collection of vectors in $\mathbb{R}^d$, the family of hash functions h_r is defined as follows:

$$h_r(u) = \begin{cases} 1 & \text{if } r \cdot u \geqslant 0 \\ 0 & \text{if } r \cdot u < 0 \end{cases}$$

where each vector r follows d-dimensional Gaussian distribution (i.e. each coordinate is drawn from the 1-dimensional Gaussian Distribution). The cardinality of the family F must be in the order of $O(\log^2 n)$ for the LSH to function properly (3).

Numerous works have built on random projections, in particular Super-bit LSH (39) aims to improve the above hashing functions for cosine similarity, by dividing the random projections into G groups then orthogonalizing B

random projections for each group, obtaining new GB random projections and thus G B-super bits. It is shown that the Hamming distance over the super bits is an unbiased estimation of the angular distance and the variance is smaller than the above random projection algorithm.

More recently, in 2015, Andoni et al. (38) showed the existence of a Locality-Sensitive Hashing (LSH) family for the euclidean distance over the unit sphere $S^{d-1} \subset \mathbb{R}^d$, this distance is known to be equivalent to the cosine similarity. Their new family of functions is based on cross-polytopes, basically it applies a random rotation to a point x and assigns the nearest vertex of the cross-polytope as the hash value. In detail, let $A \in R^{d \times d}$ be a random matrix of i.i.d Gaussian entries. To hash a point $x \in S^{d-1}$, their method computes $y = \frac{Ax}{\|Ax\|} \in S^{d-1}$ and then find the closest point to y from $\{\pm e_i\}_{1 \leq i \leq d}$, where e_i is the i-th standard basis vector of $\mathbb{R}^d$. The algorithm uses the closest neighbor as a hash of x. The collision probability of two points under the above family is bounded by the following theorem:

Theorem 3.6 *Suppose that $p, q \in S^{d-1}$ are such that $\|p - q\| = \tau$, where $0 < \tau < 2$. Then,*

$$\ln \frac{1}{P[h(p) = h(q)]} = \frac{\tau^2}{4 - \tau} \cdot \ln d + O_\tau(\ln \ln d)$$

The main implication of the cross-polytopes LSH is that it yields an ANNS algorithm with the asymptotically optimal running time exponent. Unlike earlier algorithms with this property, their algorithm is also practical, improving upon the well-studied hyperplane LSH (3) in practice.

3.2.2 On LSH for solving MIPS

Despite the relation between the cosine similarity and the inner product, providing a LSH for the later proved to be considerably harder. Both Shrivastava and Ping (7) and Neyshabur and Srebro (8) shown that there is no symmetric LSH over the entire space $\mathbb{R}^d$ for Maximum Inner Product Search (MIPS) problem. Recalling that the MIPS problem was defined in Section 2.3.

Theorem 3.7 *There is no LSH for Maximum Inner Product Search over $\mathbb{R}^d$*

Proof. Assume for contradiction h is an $(S, cS, p_1, p_2) - LSH$ (with $p1 > p2$). Let x be a vector such that $\|x\| = cS$. Let $q = x$ and $y = \frac{1}{c}x$. Therefore we have $q^T x = cS$ and $q^T y = S$. However, since $q = x$, we have $P[h(x) = h(q)] = 1 \leq p_2 \leq p_1 = P[h(q) = h(y)] \leq 1$ ■

With the aim of solving MIPS using LSH, Shrivastava and Ping (7) introduce the novel concept of Asymmetric Locality Sensitive Hashing (ALSH), that is the one we have used so far. The asymmetry refers to the difference between the random hash function over the query space and the dataset space, but even this approach can't be used to index the entire $\mathbb{R}^d$. Luckily, one does not needs $\mathbb{R}^d$ entirely to solve MIPS, for a given dataset.

It is possible to assume w.l.g. that every point in P has l^2 norm at most 1. In fact, each point in P can be divided by the maximum norm among the points in P without changing MIPS's solution. Moreover, we can also assume that q has norm 1 ($\|q\|_2 = 1$) because normalizing it does not affect the solution. These observation were made in (7).

Taking into account these considerations, Neyshabur and Srebro (8) approached MIPS by proposing a simple LSH for the inner product similarity over the spaces $\mathcal{X} = \{x \in R^d : |x| = 1\}$ and $\mathcal{Y} = \{y \in R^d : |y| \leq 1\}$. Here, $\mathcal{X}$ corresponds to the possible query points and $\mathcal{Y}$ to the database points.

The key observation here is that under the above conditions one can transform the MIPS problem into the cosine similarity search (CSS) problem. For that, it is employed a function $P : R^d \mapsto R^{d+1}$ that maps $v = (v_1, \ldots, v_d)$ into

$$P(v) = (v_1, \ldots, v_d, \sqrt{1 - |v|^2}).$$

It follows that if $v \in \mathcal{X} \cup \mathcal{Y}$, then $|P(v)| = 1$ and so the inner product becomes equivalent to the cosine similarity. This allows to use LSH for CSS such as the ones in (3) and (38). Notice that for this specific case where $\|q\| = 1$ the LSH family is symmetric, the same transformation is applied to both the query space and the dataset space. If one consider the case where the query is not normalized, the following transformation can be applied:

$$Q(v) = (v_1, \ldots, v_d, 0, \sqrt{1 - |v|^2})$$

for the dataset space (the points in P) an analogous mapping function is defined:

$$P(v) = (v_1, \ldots, v_d, \sqrt{1 - |v|^2}, 0)$$

This ALSH offers the same guarantees that the case where $\|q\| = 1$ and the norm of the dataset space is bounded.

3.3 Inverted Index

The inverted index is one of the core data structures used by search engines (43). Formally, an inverted index representation of a matrix P, is a set

of m lists, $I = \{I_1, \ldots, I_d\}$, one for each column (coordinate) of P, where list I_i contains pairs $(\mathbf{p}, p_i)$, subject to p is a row of (a vector in) P and $p_i > 0$. This data structure has been particularly successful for the problem of All-Pairs-Similarity Search problem (44), which is a generalization of the NNS problem.

The indexing phase is pretty straightforward and is described in the Algorithm 3.3.1

Algorithm 3.3.1: Inverted Indexing of P

```
   Result: Index P
 1 I = ∅
 2 for j = 1 ... d do
 3 |   I[j] = ∅
 4 end
 5 for i = 1 ... |P| do
 6 |   for j = 1 ... d do
 7 |   |   if sp_i[j] > 0 then
 8 |   |   |   I[j] = I[j] ∪ ⟨s_i, s_i[j]⟩
 9 |   |   end
10 |   end
11 end
```

Once the index is built one can search on it using two main classes of search strategies, as defined by Turtle and Flood (42).

- *Term-at-a-time (TAAT)*: strategies process query terms one by one and accumulate partial document scores as the contribution of each query term is computed.
- *Document-at-a-time (DAAT)*: strategies evaluate the contributions of every query term with respect to a single document before moving to the next document.

These search strategies arose in the information retrieval community, hence the names. In our scenario, term will correspond to coordinate and document to vector. Both TAAT and DAAT strategies can be optimized significantly by relaxing on the requirement that all document scores are complete and accurate. Optimization strategies have been studied extensively in the information retrieval literature. For a comprehensive overview of optimization techniques see Turtle and Flood (42). An important optimization technique for DAAT strategies is termed *max-score* by Turtle and Flood. Given a recall parameter k, it operates by keeping track of the top k scoring documents seen so far. Evaluation of a particular document is terminated as soon as it is clear

that this document will not place in the top k. As the focus of our research is not in evaluating both strategies, we only consider on a representative of TAAT, that is simple to implement. From this point on we will refer to TAAT as *Coordinate-at-a-Time* (CAAT) Search.

3.3.1 Coordinate at a Time Search

For CAAT strategies the basic idea behind such optimization techniques is to process query coordinates in some order that let the system identify the Top-k scoring documents without processing all query coordinates. This is achieved by maintaining $k+1$ candidates instead of k and computing an upper bound of the maximum achievable inner product value w by the $k+1$ candidate. If this score w is less than the score of the k candidate the process can be stopped. A detailed description of the search strategy can be found in Algorithm 3.3.2

Algorithm 3.3.2: Retrieval of Top k-Nearest Neighbors

input : A query point q
output: The set K of the k-Nearest Neighbors

```
1  Let I be the inverted index
2  Sort q_i in decreasing order
3  Initialize sim as an array of size |P|
4  Initialize top as an array of size k + 1
5  Set num_top = 0
6  Let C the list of indexes c | q_c > 0
7  Let C_v = ∅ the list of visited indexes
8  for c ∈ C do
9      Append c to C_v
10     for p_i, p ∈ I[c] do
11         sim[p] = sim[p] + p_i
12         UpdateTop(top, num_top, sim)
13     end
14     if num_top = k + 1 then
15         max_sim = 0
16         for c' ∈ C \ C_v do
17             max_sim = max_sim + max_c'
18         end
19         max_sim = sqrt(max_sim)
20         if sim[top[k + 1]] + max_sim < sim[top[k]] then
21             return top
22         end
23     end
24 end
```

The *UpdateTop* routine is described in Algorithm 3.3.3, basically it describes how to process each cumulative score for a single candidate p_i. We add a new candidate to *top* if we have fewer candidates that k or if this new candidate p has a greater cumulative score than the least candidates. Finally, we reorder *top* to keep the invariant than the minimum is found at the position num_{top}.

Algorithm 3.3.3: Algorithm for updating the Top k-Nearest Neighbors

```
input: The array top, num_top and the cumulative score sim
if num_top ≤ k or sim[p] > sim[top[num_top]] then
    if p ∉ top then
        if num_top < K + 1 then
            Set num_top = num_top + 1
        end
        Set top[num_top] = p
    end
    Restore top sorted order
end
```

In order to compute the upper bound of candidate $k+1$ we will make use of the Cauchy-Schwarz inequality as in (45), in conjunction with the properties of the previous chapter:

$$\langle u, v \rangle \leq \|u\|\|v\|$$

This inequality follows from the very definition of inner product. Moreover as normalizing the query q does not affect the order of the Top-k nearest neighbors we will consider q to be normalized. Therefore we can state the following

$$\langle q, p \rangle \leq \|p\| \mid \forall p \in P$$

Consider now a vector composed of the maximum values across all dimensions

$$v = (v_1, \ldots, v_d) \mid v_i = \max_{s \in S} s_i$$

Is clear from the definition of v that $\|v\| \geq \|p\| \mid \forall p \in P$ and also $\|v''\| \geq \|p''\| \mid \forall p \in P$ and a fixed k. Recall that $\|p''\|$ was defined in Section 2.3 as $p'' = (0, \ldots, 0, p_{k+1}, \ldots, p_d)$. Combining the two properties from above

we have that:

$$\langle q, p\rangle = \langle q, p'\rangle + \langle q, p''\rangle \tag{3-5}$$
$$\leq \langle q, p'\rangle + \|p''\| \tag{3-6}$$
$$\leq \langle q, p'\rangle + \|v''\| \tag{3-7}$$

So for MIPS it suffices to store for each coordinate c, $max_c = \max_{p\in P} p_c^2$. This bound is used in lines **15-22** of Algorithm 3.3.2. For each of the coordinates that still need to be visited we compute the bound, if the $k+1$ candidate cannot surpass the k candidate we terminate the search.

To clarify let use the following example suppose that the k-th candidate has a sim value of 10 and the $k+1$-th candidate has a sim value of 4, and we have two more dimensions remaining with max values of 2 and 1. Now the maximum sim value that the $k+1$-th candidate can obtain is $4+\sqrt{2^2+1} \leq 10$ so it can not become one of the k candidates. The rest of possible candidates can be divided into two sets: (1) those that have already been evaluated and (2) those that not. In the first case the sim value is less than the value of the $k+1$-th candidate therefore it also cannot be included in the set of k candidates. Now a point p that has not been evaluated can have a maximum of$\sqrt{2^2+1}$ as it has no previous coordinates in common with q, so it also cannot be included in the set K and the search can be terminated.

3.3.2 Sampling Techniques for Matrix Sparsification

The techniques discussed in the previous section, return the exact top-k (k-nearest neighbors) most similar neighbors, very efficiently (44). Unfortunately the efficiency only holds for sparse vectors, but in our application the vectors are dense. So we will use sampling techniques to approximate our matrix P of vectors, this will allow applying the CAAT algorithm at the cost of exactness. We study different techniques for approximating our vectors with sampling techniques, namely:

1. *Simple Sampling*: In this sampling for each $p \in P$ we sample p_i proportionally to $\frac{|p_i|}{|p|_1}$.

2. *CUR Sampling*: This scheme is inspired by the CUR decomposition algorithm, so we sample entire columns (rows) based on $p_i, i \in |C|$ where $|C|$ is the number of columns (rows), so we have three samplings: $RowSelect$, $ColumnSelect$ and both.

3. *Diamond Sampling*: Finally Diamond Sampling is inspired by the recently proposed sampling scheme by Ballard et. al. (46).

The result P' obtained from this sampling schemes has the same dimensions as P, only that the non-zero entries are those that we sampled, then P' is sparse. To the best of our knowledge we are the first to propose the *Simple Sampling* technique. We also applied it to select the entries from the query q. The CUR matrix approximation, of some matrix M is a set of three matrices C, U and R dependent on M, that when multiplied together, closely approximate M. A CUR approximation can be used in the same way as a low-rank approximation of the Singular Value Decomposition. C and R are made of columns and rows of M. The CUR matrix approximation is not unique and there are multiple algorithms for computing one. The algorithm by Mahoney and Drineas (47) samples rows and columns proportionally to the l^2 norm of the column (vector).

A problem related to Top-k MIPS retrieval is that of finding the Top-k largest values in a matrix-matrix multiplication $Q^T P$. Diamond Sampling (46) was proposed to avoid direct computation of inner product for all pairs, as a sampling approach that selects diamonds from the weighted tripartite representations of Q and P (46).

Algorithm 3.3.4: Diamond Sampling

input : A dense matrix $M \in \mathbb{R}^{m \times d}$
output: A sparse matrix A'
Let s be the number of possible entries in A'
forall $a_{ki} \neq 0$ **do**
  $w_{ki} = |a_{ki}| \|a_{*i}\|_1$
end
A' all-zero matrix of size $m \times d$
for $l = 1 \dots s$ **do**
  Sample (k, i) with probability $w_{ki} / \|W\|_1$
  Sample (k', i) with probability $a_{k'i} / \|a_{*i}\|_1$
  Set $A'[i, k] = A[i, k]$
  Set $A'[i, k'] = A[i, k']$
end
return A'

The general idea of *Diamond Sampling* is inspired by 4-vertex motif detection in large graphs. In their work the authors provide probabilistic bounds on the number of sufficient samples to guarantee a desired accuracy for the retrieval of the top-k inner products. To prove the bounds the authors assume that the set Q is known in advance, which is not the case in our

application. We merely study the algorithm as inspiration for a heuristic, the details of it are specified in Algorithm 3.3.4.

The sampling algorithms provide a sparse approximation P' of the matrix P, so the k nearest neighbors of q in P' may not be the same that the k nearest neighbors in P. In order to provide a better approximation instead of searching for the k nearest neighbors we search for the set B of the $b > k$ nearest neighbors, once the search is terminated we compute the actual similarity (distance) value of q with respect to $p_i \in B$ and return the k nearest neighbors.

3.4 Nearest Neighbor Search for Kullback-Leibler Divergence

The methods for constructing indexes for the efficient search under the KL divergence, or more generally, the Bregman divergence are relatively new (10). As in the rest of the literature of NNS for high dimensional data there are two relevant approaches: space partitioning methods and hashing.

3.4.1 Space Partitioning Methods

Space partitioning methods, often referred as hierarchical space partitioning algorithms are methods that create a decomposition of the space, commonly represented by a tree. These methods were developed by the computational geometry community as one of the first attempts at solving the NN problem in low dimensions. Examples of such methods are the KD-Tree, Vantage Point (VP)-Tree, and Ball Tree.

We already describe one of such approaches for Bregman Divergences, the BBTree of Cayton (10). Building on the work by Cayton, Coviello et al. (28) developed a new approach for deciding whether to explore nodes during backtracking, based on a variational approximation of the optimization problem. This scheme reduces the number of computations per node(28) which leads to a faster algorithm in higher dimensions.

In (36) Nielsen et al. proposed a generalization of VP-Trees for Bregman divergences. The main contribution of their work is to replace the triangle inequality by an analogous criterion based on the intersection of Bregman balls, which allows checking for pruning conditions. Zhang et al. (9), propose to map the points in the original space to an extended space and poses the problem as an inner product minimization problem. This formulation is similar to ours, but the work overlooks the potential to improve the Bregman Ball Tree and uses structures such as VA-file and R-tree.

3.4.2
Locality Sensitive Hashing for related distances

Additionally, several hashing schemes have been defined for alternative measures of probability distributions such as the Chi-squared distance (12), and the Hellinger distance (13). Also, Mu and Yan (25) proposed a family of LSH functions for dealing with non-metric distances, but they considered a symmetric version of the KL divergence. To the best of our knowledge, there hasn't been proposed an LSH scheme for the asymmetric KL divergence.

4
Embeddings for Kullback-Leibler Divergence

In this chapter we present the core of our theoretical results. The chapter is organized as follows: Section 4.1 overviews some negative results regarding the embedding of the Kullback-Leibler divergence in metric spaces such as the euclidean distance. Next, Section 4.2 describes how to embed (convert) the one sided Bregman divergence into an asymmetric inner product space. Section 4.3 builds on Section 4.2 and shows how to use the ideas of (8) to devise an ordinal asymmetric embedding into an euclidean space. Finally in Section 4.4 we combine the theoretical insights to propose provably efficient algorithms for the ANN problem under the Kullback-Leibler divergence

4.1
On Low Distortion Embeddings of the Kullback-Leibler divergence

We begin by defining a few preliminary concepts related to metric spaces and embeddings

Definition 4.1 *(Metric Space) Let X be a set and $dist : X \times X \to \mathbb{R}^+ \cup \{0\}$ a distance measure. A pair $M = (X, dist)$ is called a metric space provided that $dist$ satisfies the properties of the identity, symmetry and triangular inequality.*

Definition 4.2 *(D-embedding and Distortion) Given two metric spaces $(X, dist_1)$ and $(Y, dist_2)$, a mapping $f : X \to Y$ is called a D-embedding where $D \geq 1$, if there exists a number $r \geq 0$ such that for all $x, y \in X$,*

$$r \cdot dist_1(x, y) \leq dist_2(f(x), f(y)) \leq D \cdot r \cdot dist_1(x, y)$$

The infimum of all numbers D such that f is a D-embedding is called the distortion of f.

In general, the factor r is intended to allow the distances to be scaled by some constant factor and does not affect the definition of embedding. This notion of distortion can be naturally extended to non-metric spaces as well.

Bhattacharya et. al. (32) showed that low distortion embeddings for the Kullback-Leibler divergence into metric spaces cannot exist. The proof exploits

the fact that the Kullback-Leibler is not symmetric and violates the triangle inequality. We report the main result of Bhattacharya et. al., the proof can be found in the original work (32):

Theorem 4.3 *For sufficiently large d and small β, there exist a set S of d-dimensional β-constrained histograms such that any embedding of S into a metric space incurs a distortion of* $\Omega\left(\frac{\ln \frac{1}{d\beta}}{\ln(d \ln \frac{1}{\beta})}\right)$

It follows from the previous theorem that the distortion into any metric space can be made arbitrarily large. This result implies that there is no embedding for the KL divergence when considering both sides of the parameters of the function.

4.2 Inner Product Embedding of the One Sided Bregman divergence

Although the result from Bhattacharya et. al. can be discouraging, in order to retrieve the k nearest neighbors of a point we only need a transformation that maintains the order of the distances of the points $p \in P$ with respect to a query q. This type of embedding is known as an ordinal embedding (55), an asymmetric definition can be posed as follows:

Definition 4.4 *Let $(\mathbb{Q}^d, \mathbb{P}^d)$ be a pair of spaces equipped with some distance function $dist : \mathbb{Q}^d \times \mathbb{P}^d \to \mathbb{R}^+$. Moreover let $q \in Q$ and $p_i, p_j \in P$ be a triplet of points (q, p_i, p_j). An asymmetric ordinal embedding is a pair of functions (o, e), $o : \mathbb{Q}^d \to \mathbb{R}^m$ and $e : \mathbb{P}^d \to \mathbb{R}^m$ such that for all triplets (q, p_i, p_j):*

$$dist(q, p_i) \leq dist(q, p_j) \Rightarrow \|o(q) - e(p_i)\|_2 \leq \|o(q) - e(p_j)\|_2$$

We show that assuming the query can only be in one side e.g. right-side, and the data points in the left-side, we can devise such transformation even for a larger class of functions such as the Bregman divergences. First, recall that a Bregman divergence is defined as follows:

$$D_f(p, q) = f(p) - f(q) - \langle \nabla f(q), p - q \rangle ,$$

and that we want to minimize $D_f(p, q)$ for a given query q and $p \in P$. Lets define a pair of functions $h : \mathbb{R}^d \to \mathbb{R}^{d+2}$ and $g : \mathbb{R} \to \mathbb{R}^{d+2}$, in the following way:

$$h(p) = (f(p), p_1, \ldots, p_d, 1) \tag{4-1}$$

$$g(q) = (1, \nabla f(q)_1, \ldots, \nabla f(q)_d, \langle \nabla f(q), q \rangle - f(q)) \tag{4-2}$$

So,

$$D_f(p,q) = \langle h(p), g(q) \rangle$$

therefore

$$\min_{p \in P} D_f(p,q) = \min_{p \in P} \langle h(p), g(q) \rangle \tag{4-3}$$

$$= -\max_{p \in P} \langle h(p), g(q) \rangle \tag{4-4}$$

$$= \max_{p \in P} \langle -h(p), g(q) \rangle \tag{4-5}$$

both 4-4 and 4-5 follows from properties of addition and multiplication. This not only provides a natural way to use LSH, Inverted Index and other data structures (9) to solve the ANNS under the Bregman divergence, but also it can be used to accelerate the Bregman Ball Tree, which we show in the next section.

4.3 A simple ordinal embedding

Before further stepping into the creation of the ordinal embedding notice that multiplicative factors do not alter the order of the inner product similarity, therefore we can multiply $h(p)$ by $1/2$. From now on we call $h_2(p) = -1/2h(p)$. Assuming p and q are bounded normalized histograms and f (of D_f) is a continuous function we can state that there is some L that $\|h_2(p)^2\| \leq L$ and some C that $\|g(q)^2\| \leq C$. So we have:

$$h^+(p) = (h_2(p); \sqrt{L - \|h_2(p)\|^2}; 0) \tag{4-6}$$

$$g^+(q) = (g(q); 0; \sqrt{C - \|g(q)\|^2}) \tag{4-7}$$

where ; is the coordinate concatenation operator. So maximizing $\langle h^+(p), g^+(q) \rangle$ is equivalent to minimizing $D_f(p,q)$. Finally it turns out that:

$$\max_{p \in P} \left\langle h^+(p), g^+(q) \right\rangle = min_{p \in P} \|h^+(p) - g^+(q)\|_2 \tag{4-8}$$

$$= min_{p \in P} \sqrt{L + C + D_f(p,q)} \tag{4-9}$$

Now we are ready to prove the following, which is the main theoretical result of this work:

Theorem 4.5 *Given a ball* $B(\mu, R) = \{x : D_f(x, \mu) \leq R\}$ *the following inequality* $\min_{x \in B(\mu,R)} D_f(x,q) \leq t$ *holds if and only if* $min_{x \in B(\mu,R)} \|h^+(x) - g^+(q)\| \leq t'$ *holds, where* $t' = \sqrt{L + C + t}$

Proof. For the first part of the proof suppose that there is some x that $D_f(x, q) \leq t$, so:

$$D_f(x, q) \leq t \tag{4-10}$$
$$D_f(x, q) + L + C \leq t + L + C \tag{4-11}$$
$$\sqrt{D_f(x, q) + L + C} \leq \sqrt{t + L + C} \tag{4-12}$$
$$\|h^+(x) - g^+(q)\| \leq t' \tag{4-13}$$

The second part is analogous to the first one in the sense that one assumes some $h^+(x)$ exists such that $\|h^+(x) - g^+(q)\| \leq t'$ and then derive the inequality through algebraic transformations.

■

4.3.1 A faster pruning algorithm for Bregman Ball Trees

The previous result states that the pair (h^+, g^+) is an ordinal embedding. So we can substitute the *CanPrune* procedure detailed in Algorithm 3.1.1 by Algorithm 4.3.1.

Algorithm 4.3.1: Can prune node (FastCanPrune)

```
input  : g+(q), h+(μ) ∈ R^{d+4}, R', dist_{p_c}
output: If the node must be explored or not
if ‖h+(μ) − g+(q)‖_2 − R' ≥ dist_{p_c} then
    return YES
end
return NO
```

In Algorithm 4.3.1 $dist_{p_c}$ is the best distance found so far, μ is the centroid of the ball and $R' = \max_{p \in P} \|h^+(p) - h^+(\mu)\|$ is the radius of the euclidean ball. The value R' is computed during the construction and $\|h^+(\mu) - g^+(q)\|_2$ is computed while descending the tree so the cost of this verification is constant. The Algorithm simply uses the inequalities 4-14 and 4-15:

$$\|h^+(x) - g^+(q)\|_2 \geq \|h^+(\mu) - g^+(q)\|_2 - R' \tag{4-14}$$
$$\|h^+(x) - g^+(q)\|_2 \leq \|h^+(\mu) - g^+(q)\|_2 + R' \tag{4-15}$$

The previous inequalities make use of the triangle inequality (50). Given that the complexity of computing the previous inequalities is constant then the complexity of the Bregman Ball Tree querying algorithm becomes $O(nd + o)$, where o is the number of nodes, the original complexity was $O(nd + od \log(1/\epsilon))$,

this is a multiplicative improvement over the original Bregman Ball Tree querying algorithm. The new method is detailed in Algorithm 4.3.1

On a second note, notice that this transformation can be used to accelerate the construction phase of the Bregman Ball Tree. Simply transform all $p \in P$ up front with h^+, once the points are transformed any acceleration technique (52) for euclidean k-means can be used. Moreover, other construction techniques such as the one for $k-d$-trees can also be used resulting in faster versions of the construction algorithm.

4.4 Some notes on the embedding to the Kullback-Leibler divergence

The previous results apply to all Bregman divergences including the KL divergence but a closer look provides some insight on how to adapt the embedding methodology specifically for the Kullback-Leibler divergence. In particular, we deduce two results, one that saves computations and the other one simplifies the sampling strategies.

Theorem 4.6 *The nearest neighbor p^* in P of q under the KL divergence is the one that maximizes $\langle q, p^+ \rangle$ where $p^+ = (\ln p_1, \ldots, \ln p_d)$*

Proof.

First lets rewrite the KL divergence between q and p where q is on the left side of the divergence:

$$\begin{aligned} KL(q,p) &= \sum_{i=1}^{d} q_i \ln \frac{q_i}{p_i} \\ &= \sum_{i=1}^{d} q_i \ln q_i - \sum_{i=1}^{d} q_i \ln p_i \end{aligned} \tag{4-16}$$

Since $\sum_{i=1}^{d} q_i \ln q_i$ is constant, the maximization of the inner product corresponds to the minimization of the KL divergence.

■

Now we can instantiate the pairs of functions (f, g) for the embedding, in the following way $f(q) = q$ is the identity function and $g(p) = (\ln p_1, \ldots, \ln p_d)$ is a function that applies the natural logarithm to each individual coordinate. We can also express the KL divergence as the multiplication of the only positive vectors. This will be useful when using the sampling schemes. We show a simple transformation that keeps the ordering of the value of the inner products while keeping the coordinates positives.

Proposition 4.7 *Let S be a set of vectors $s \in \mathbb{R}^d$, and let m be the mapping $m : \mathbb{R}^d \to \mathbb{R}^d$, $m(s) = (s_1 + c, s_2 + c, \ldots, s_d + c)$ where c is the $|\min_{s,i} s_i| \forall s \in$*

$S, i \in [d]$, i.e. the minimum value across all coordinates for all vectors in S. Then f preserves the ordering of the inner products of S with respect to some query vector q

Proof. Let $u, v \in S$ and q a query vector, with $\langle u, q\rangle \leq \langle v, q\rangle$ moreover let $v' = m(v)$ and $u' = m(u)$ by construction of the mapping m all the coordinates of v', u' are positive. Also $\langle u', q\rangle = \langle u, q\rangle + c\sum q_i$ and $\langle v', q\rangle = \langle v, q\rangle + c\sum q_i$ then by properties of the sum we have $\langle u', q\rangle \leq \langle v', q\rangle$, which completes the proof ■

In order to prove Proposition 4.7 (the above proposition) we use the fact that in a given finite set there is minimum coordinate value. In our particular case we need not the set to be finite, due to our points p being β-constrained, this means that $\ln p_i$ is bounded, and we can use $c = \ln p_i$.

5
Experiments

In this section, we report our experimental study. In all the **nearest neighbor** was retrieved (1-NN) with respect to the left-NN problem.

5.1
Experimental Setup

All experiments were carried on an Intel i7-4500U CPU with 16 GB of RAM. We ran experiments on 6 of the originals datasets proposed by Cayton (10):

- **Corel Image Collection:** This dataset corresponds to roughly 60K 64-dimensional histograms corresponding to color images in HSV format.
- **Reuters Corpus:** We select a subset of nearly 400K documents of the RCV collection and for each document we used Latent Dirichlet Allocation (LDA) (19) to generate topic vectors of dimension $D \in \{64, 128, 256\}$ [1]. We refer to each individual dataset as LDA-D.
- **Semantic Space:** The SemSpace dataset consists of a sample of 4500 images of the Corel Stock photo collection. Each image is represented as a distribution over 371 description keywords (56).
- **SIFT signatures:** This dataset contains 10K histograms of quantized SIFT features. Each histogram has dimension 1111 and corresponds to an image in the PASCAL 2007 dataset (57).

In the original datasets of Cayton (10), the author also applied LDA for dimension 8 and 16, we left out those as, in general, a dimension d is considered high when d is close to 100. We used the same evaluation protocol as the one described in (51). For all datasets we randomly selected 500 query points and report the average number of queries per second as a performance measure and the average precision for the k-NN for $k = 1$. We define precision as:

$$p = \frac{|Re \cap Rel|}{|Re|}$$

[1] We use the lda library in http://pythonhosted.org/lda/

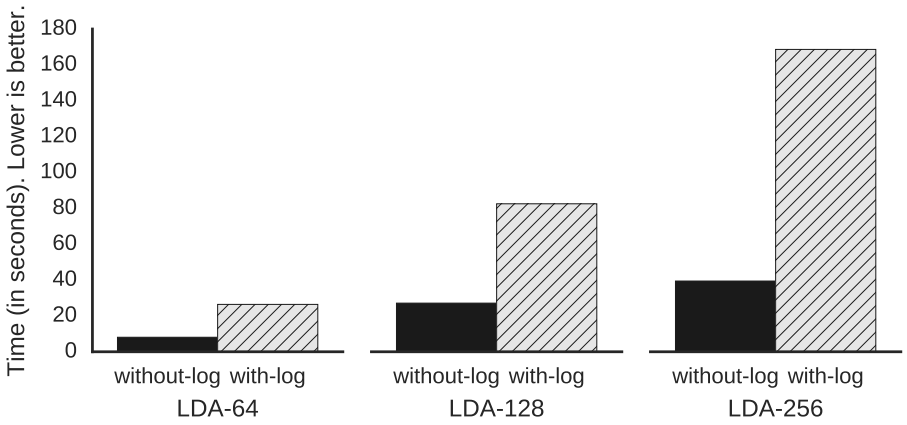


Figure 5.1: Effect of caching logarithm computations during search on a Bregman Ball Tree

where *Re* is the set of retrieved points and *Rel* is the set of relevant points. In our case as $k = 1$, the average precision amounts to the percentage of queries the method was able to retrieve the true nearest neighbor.

5.2 Implementation Details

We discuss the implementation details for reproducibility, but we remark that the implementation is not our primary contribution. Nevertheless, careful thought has gone into the process and we show that a clever implementation of the Bregman Ball Tree can improve its performance significantly. We implemented three methods:

- **BregmanBallTree**: The Bregman Ball Tree (BBTree) of Cayton (10), this is our baseline method.
- **FALCONN**: We used the LSH proposal of Andoni et. al (38) for cosine similarity using the transformations formulated in the previous section in conjunction with the adaptations of Neyshabur and Srebro (8) of LSH for MIPS. The method is named after the library that was used.
- **Inverted-Index**: A Maximum Inner Product Search using our proposal of inverted index.

All methods were implemented in Python, most of the algebraic computations were implemented using the SciPy stack (48).

5.2.1 Bregman Ball Tree Implementation

For the implementation of the BBTree we use the pattern "Structure of Arrays" (SoA), this pattern is the same used by the library Sklearn (49) for their implementation of the metric Ball Tree. In SoA the elements of each data point (record) are stored in a layout of parallel arrays, one array per field.

The implementation uses pre-allocated numpy arrays. The main advantage of pre-allocations is that the objects can be quickly iterated, this iterative interface gives more control over the heap, and leads to speed. In the downside readability of the code suffers and once the tree is built, augmenting or pruning it is not as straightforward. Also, the size of the tree must be known from the start, so there is not as much flexibility in building it.

During our experimentation, we found that pre-computing the value of the natural logarithm of the points in the data set could lead to an improvement of $4x$ in execution time. This is illustrated in Figure 5.1 over three datasets.

We found this discovery to be of some significance, the implementation provided by Lawrence Cayton, the author of the Bregman Ball Tree does not take into account this fact. Most of the comparisons performed don't take this into account. For instance, Zhang et. al. (9) compared an implementation of Bregman Ball Tree that did not store (preprocess) the values of the logarithmic function against a proposal that did store the values of the logarithmic function. The experiments were conducted on synthetic datasets of dimension $d \in [2, 4, 8, 16, 32]$ In their experiments, the BBTree consumes considerable less CPU time that the rest of proposed methods up to dimension 8. For dimension 16 and 32 the gain of the competitors is less than $4x$, so given that the methods of Zhang et. al. store the values of the computation, one could conclude that the speed of the methods came from storing this values and not from the methods themselves. Is difficult to provide an exact conclusion as Zhang et. al (9) does not report exact numbers and the comparison is made via plots.

5.2.2 LSH for KL implementation

As mentioned before we used the family of LSH functions proposed by Andoni et. al. (38) which is known to be theoretically optimal as well as practical. We used the implementation of the authors found in the FALCONN library. Besides the fact that the library is implemented in Python, one of the main motivations behind our choice was that this library implements a multi-probe version of LSH which leads to lower memory consumption.

5.2.3 Inverted Index Implementation

The Inverted Index together with the CAAT strategy is very simple to implement, which was one of the reasons why we chose this method. The only implementation aspect that shall be discussed is the method $UpdateTop$ given that there are two plausible implementations: one using a heap to keep sorted the array *top*, which results in a $O(n \log c)$ complexity where $c \geq k$ is the size of the set of candidates and k is the number of nearest neighbors required. The other strategy is to not maintain no *top* structure at all and at the final select the top k using the selection algorithm $QuickSelect$ (53), also known as Hoare's selection algorithm, this implies a complexity of $O(n)$ but with large constants, so if k is small we use the first version, otherwise the second version is used. As we performed all the experiments for $k = 1$ the first version was always the one chosen.

The best sampling method, among those discussed in Section 3.3.2, was Simple Sampling as illustrated in Figure 5.2. Each point in the curve corresponds to a set of parameters, for each set is plotted the precision vs the number of queries per second. The sampling by column inspired by CUR is the worst of the three methods. In particular the precision of this method is pretty low, it does not reach 0.6. On the other hand, Diamond Sampling reaches a precision of 0.9, but at a performance of 100 queries per second. The best of them was Simple Sampling, for a precision of 0.9, similar to the highest of Diamond Sampling, it performs at almost 900 queries per second, nine times more than Diamond Sampling; plus the highest precision is 0.95 at 200 queries per second.

5.2.4 Parameter Setting

Each of the methods has a different set of parameters, with little or no relation at all. The following list details the set of parameters for each method:

- **BregmanBallTree**: This method has two parameters the number of data points per leaf node and the number of leaf nodes to visit during the back-track part of the search algorithm.
- **FALCONN**: Locality Sensitive Hashing has two parameters L and k, but following the methodology discussed in Section 3.2, one can set L as a function of k. Also suitable thresholds p_1 and p_2 must be chosen.

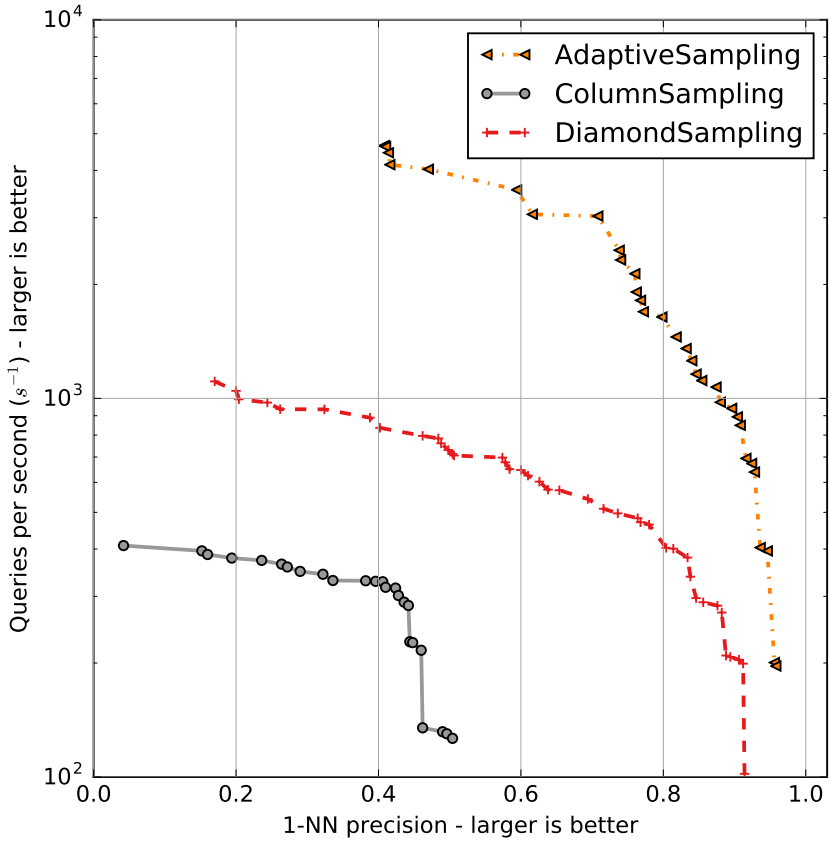


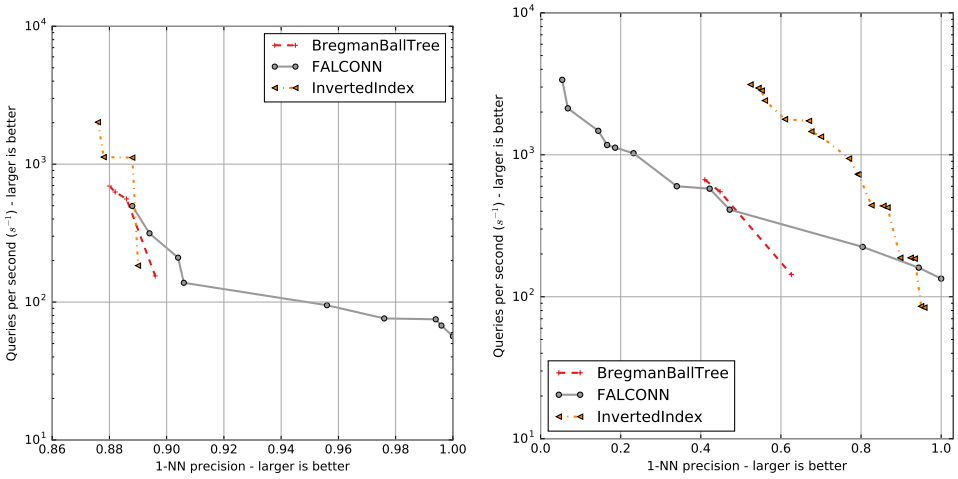
Figure 5.2: Performance of sampling methods on the SIFT dataset

– **Inverted-Index**: For this method one must set the number of coordinates to sample from the query, the number of samples for each data point and the number of candidates.

In the case of the **BregmanBallTree** method we choose the number of data points per leaf from $\{60, 100, 500\}$, the number of leaf nodes to be visited is chosen from $\{5, 10, 15\}$. For FALCONN we set k as a function of p_1 and p_2 as specified in Section 3.2 and derived L accordingly, the pair of probabilities was chosen from the cross product of $p_1 \in \{0.9, 0.8, 0.7, 0.6\}$ and $p_2 \in \{0.5, 0.4, 0.3, 0.2\}$. In the case of **Inverted-Index** we chose the number of samples for the query and the dataset from the power of 2 up to the half of d, i.e. $\{2, 4, 8, \ldots, 2^{\log d - 1}\}$, and the number of candidates was chosen from $\{5, 10, 50, 75, 100, 125, 150\}$.

5.3 Results

In this section we report our results using a plot of the Pareto frontier for each method. In the plots a curve represents the results of a method, each point represents a set of parameters, and for a given set of parameters we report the



5.3(a): LDA-64

5.3(b): Corel

Figure 5.3: Precision of each method vs number of queries per second. The legend is in the plot.

number of queries per seconds and the precision. We split the results into three categories according to the dimensionality of the data:

- **Low Dimensionality:** Corel, LDA-64
- **Medium Dimensionality:** LDA-128, LDA-256
- **High Dimensionality:** SemSpace, SIFT

The results for the **Low Dimensionality** are plotted in Figure 5.3. In the case of the Corel dataset, for the set of parameters chosen, the Bregman Ball Tree achieves a maximum of 0.6 precision. For the same performance of roughly 10^2 queries per second the LSH proposal reaches a precision of 1.0. The best performance, however, is achieved by the Inverted Index; for a precision of 0.6 its speed is an order of magnitude (a factor of 10) better than the two other proposals. This dominance is maintained up to a precision mark of 0.9 where it began to lose to FALCONN but the speed of both methods is in the same order of magnitude.

For the LDA-64 dataset, all the methods performed very well achieving a minimum precision above the mark of 0.8. And in terms of queries per second the performance is comparable, but FALCONN and Inverted Index are slightly better. In particular, Inverted Index obtains a precision of 0.9 at almost 4 times better than the others but then it loses for LSH right away. FALCONN starts a little slow in comparison to BBTree, but surpasses it for higher precision values.

In general for the **Low Dimensionality** datasets we consider that Inverted Index is the best overall, it manages to achieve good values of precision

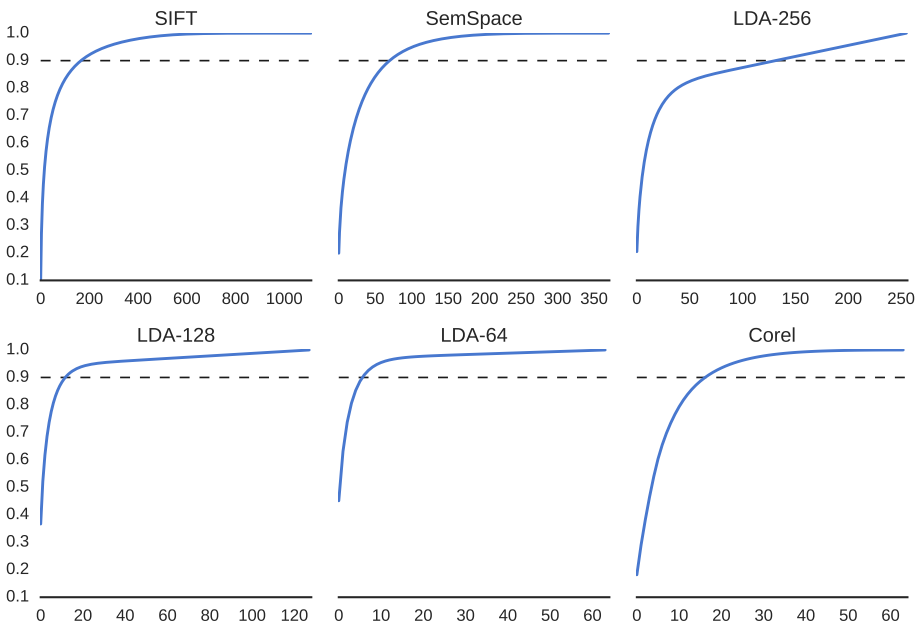


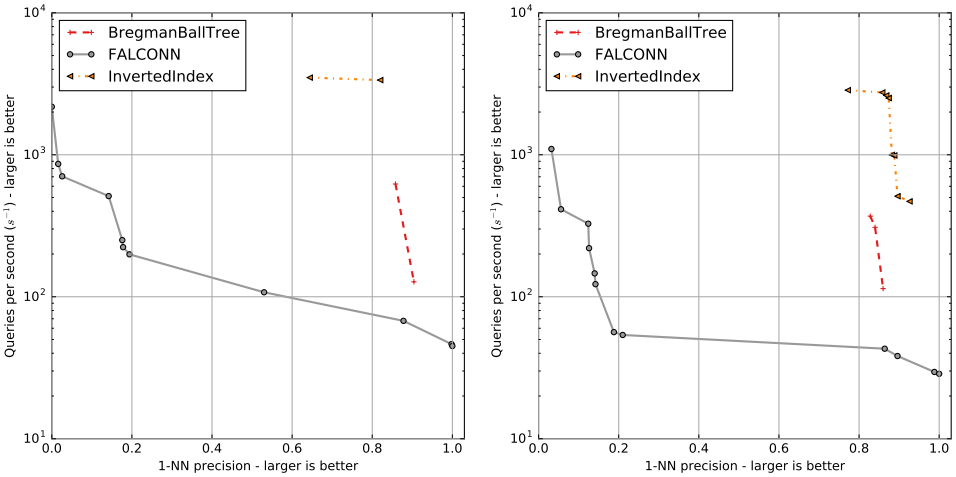
Figure 5.4: Mean cumulative sum of each of the probabilities sorted in non-increasing order. The line across each graph marks the value of 0.9

at a minimum of 0.5 for any set of parameters over the two datasets while, at some points achieving a speed of an order of magnitude better than the other two method proposals.

A possible explanation for this behavior can be found in Figure 5.4. In the graphs is shown the mean cumulative distribution of the values of the coordinates for each data point sorted in non increasing order. In other words, for each $p \in P$ we sorted the coordinates p_i from maximum to minimum, compute the cumulative sum and found the mean of the values of the cumulative sum across each dimension p_i.

The cumulative sum gives an idea of how many dimensions one will need to sample in order to get a good estimate of the norm of the vectors p which is a proxy for the value of their inner product. In the case of the LDA-64 dataset with nearly 10 dimensions we manage to obtain an approximation of 90 percent of the total sum, in the other hand, for Corel it needs almost the double, around 20 dimensions.

The number of dimensions to sample impacts the performance of Inverted Index, as more dimensions are sampled more candidates are added which translates not only in a bigger final set, but also in an increase in the number of operations to perform. Moreover the impact also extends to precision, as one needs a larger number of dimensions to have a good approximation value. All the above factors provide a plausible explanation of why the Inverted Index curve of LDA-64 in Figure 5.4 achieves a high precision at a high speed, also



5.5(a): LDA-128 5.5(b): LDA-256

Figure 5.5: Precision of each method vs number of queries per second. The legend is in the plot.

why in Corel the first value for precision is so low.

Notice that despite the values of queries per second are similar for both datasets, the number of data points in LDA-64 is almost 10 times the number of data points in Corel. Finally Figure 5.4 also gives a hint of why FALCONN (LSH) and BBTree also worked well for LDA-64, as it suggests that the data lies in a space of much lower dimensionality.

For both datasets of **Medium Dimensionality** FALCONN (LSH) performs poorly and is dominated by both BBTree and the Inverted Index. The results can be seen in Figure 5.5. In the case of the Inverted Index for the LDA-128 dataset the precision is not as good as in the **Low Dimensionality** datasets, but its performance in terms of queries per second is better than the other two methods for a similar precision level of 0.8 in particular is nearly two order of magnitude better than LSH. Bregman Ball Tree offers the best speed vs precision of the methods, it achieves a higher precision than Inverted Index and at the same time a better speed than LSH.

We plotted the distribution of the maximum inner product, once is normalized, to understand the behavior of FALCONN (LSH) . The results can be seen in Figure 5.6. For instance in the case of the LDA-256 dataset the value of the maximum similarity is below 0.08 while for LDA-128 is nearly 0.10, this low similarity values posed a problem for LSH, because setting the parameters is tricky, as L would need to be very large. Consider the scenario for LDA-128, the size of the dataset is 500000, if one choose $p_1 = 0.085$ and $p_2 = 0.035$, that is a gap of 0.05 and computes L and k as suggested in Section 3.2, this leads to $L = 15509$ and $k = 3$, leading to a blowout in storage and to

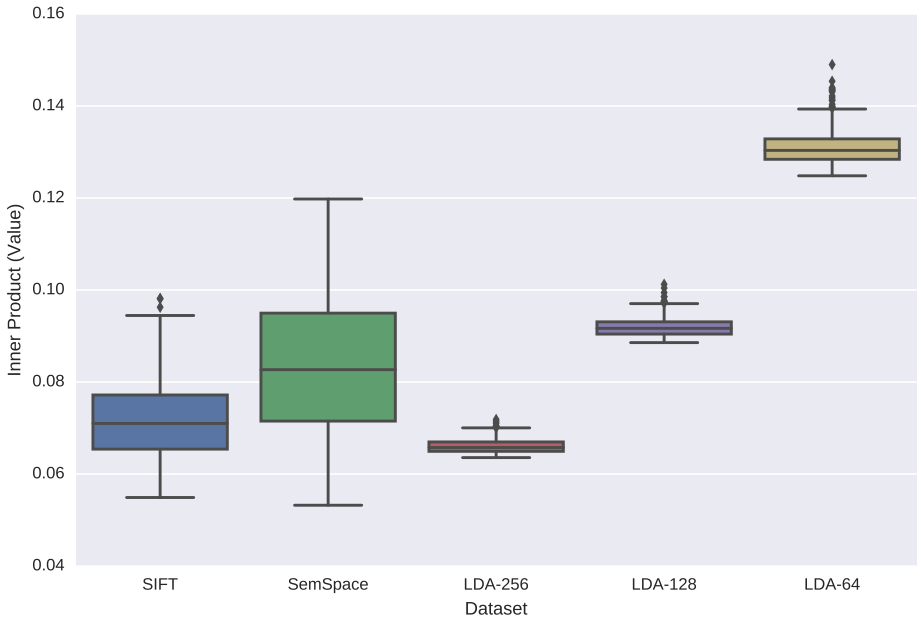


Figure 5.6: Distribution of values of the maximum inner product for the datasets SIFT, SemSpace, LDA-256, LDA-128 and LDA-64

a really large set of candidates.

This behavior is caused because for high dimensional data (dimension above 100) the maximum inner product is often small in comparison to the vector's norm, that is after normalizing. In fact it can be seen in Figure 5.6 the largest value is for LDA-64, also Corel is not shown as the magnitude was not the same as the others so it will cause a distortion in the data presentation, the mean value was around 0.4.

Figure 5.7 shows the norm distribution for each dataset. It is expected that if the norm of all vectors is roughly the same, the values of the inner product will be close to one. In Figure 5.7 is obvious that this is not the case for real world datasets.

Moreover, the norm distributions can help understand the causes behind the low similarity values. Before we proceed with the explanation, notice that the range of norms is normalized between $(0, 1]$, and that the maximum is always 1 and it corresponds to the maximum value found in the dataset. In the case of LDA-256 and LDA-128, it can be seen that the bulk of the norms is below 0.4 and nearly centered at 0.3 if one assumes a corresponding value of 0.3 for the queries the expected maximum inner product is in the range of 0.16 and 0.09 as shown in Figure 5.6. Also in the case of Corel the distribution is roughly centered at 0.6 which gives an expected value of 0.36 that is close to mean value found of 0.4.

Finally the results for the **High Dimensionality** datasets are shown

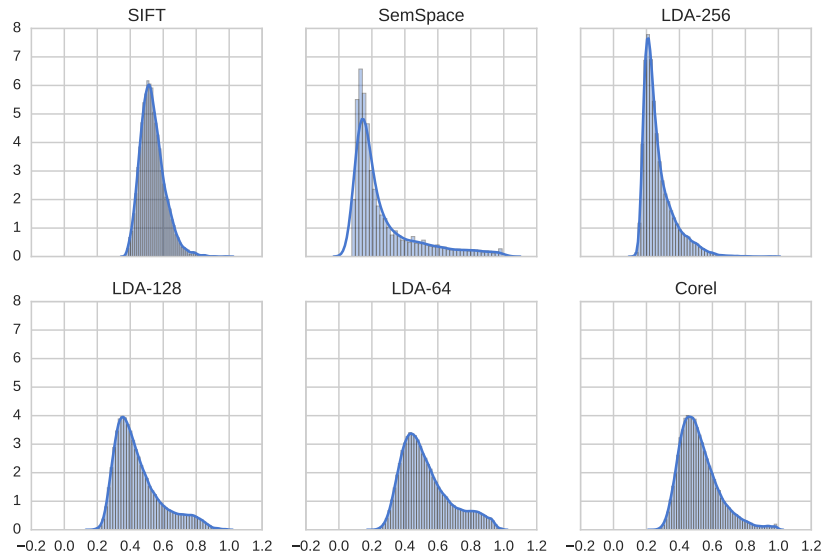
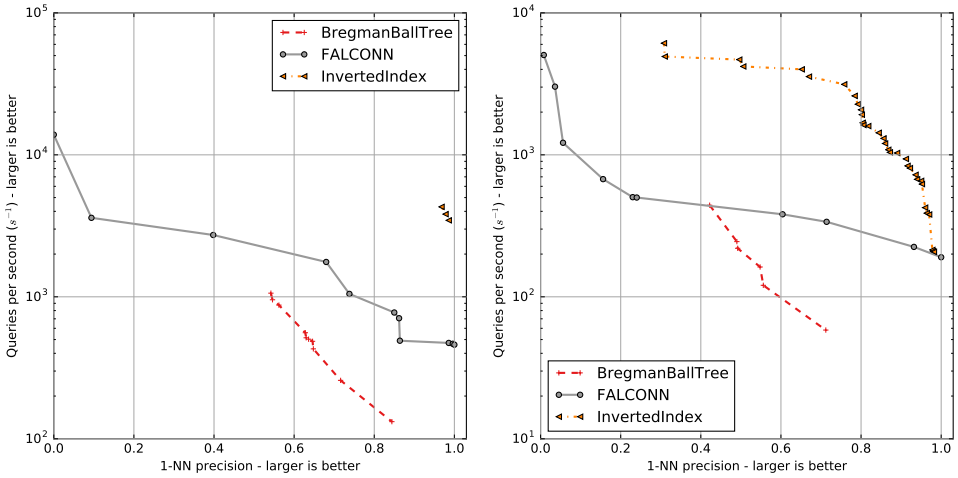


Figure 5.7: Distribution of norm values for each the datasets

in Figure 5.8. LSH and Inverted Index dominates Bregman Ball Tree in both datasets. In particular Inverted Index excels in SemSpace as the minimum precision threshold is above 0.9 and it manages to reach the maximum value of precision possible, at almost 6 times the speed of LSH for the same precision value. In the case of the SIFT dataset Inverted Index dominates both LSH and BBTree, the precision is almost 1, but in this case the performance is comparable to that of LSH. The BBTree performs very badly in this dataset, for maximum precision of around 0.7, Inverted Index is almost two orders of magnitudes faster than BBTree for a similar precision.

Summarizing our results, we implemented three different methods: BB-Tree, LSH (FALCONN) and Inverted Index. We found that for BBTree storing already calculated values of the logarithmic function leads to almost a 4 times improvement in terms of time consumption, over a naively implemented BBTree. Of the three sampling techniques, we tried for the Inverted Index, sampling the coordinates of each the data point according to it's value was the best, for all the set of parameters it dominated the alternatives both in terms of queries per second and precision.

The FALCONN based implementation was able to compete with the baseline proposal in some datasets, but it performed poorly in others. A possible explanation for this behavior could be the low values of the norm of the vectors, that make the approach unfeasible due to memory requirements.



5.8(a): SemSpace

5.8(b): Sift

Figure 5.8: Precision of each method vs number of queries per second. The legend is in the plot.

6
Conclusions

In this work we explored the approach of posing the problem of k-NN under the Kullback-Leibler divergence as a Maximum Inner Product Search. We showed in particular how this technique can be used to obtain better theoretical bounds for existing data structures such as the Bregman Ball Tree. On the practical side we investigated the use of two, very popular, indexing techniques: Inverted Index and Locality Sensitive Hashing. Experiments performed on 6 real world data-sets showed the Inverted Index performs better than LSH and Bregman Ball Tree, in terms of queries per second and precision.

Future research directions include an empirical comparison of our theoretical proposal to accelerate Bregman Ball Trees, and also it would be interesting to extend our result to the general class of Bregman divergences.

Bibliography

[1] INDYK, P.; MOTWANI, R.. **Approximate nearest neighbors: towards removing the curse of dimensionality**. In: PROCEEDINGS OF THE THIRTIETH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, p. 604–613. ACM, 1998.

[2] GOEMANS, M. X.; WILLIAMSON, D. P.. **Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming**. Journal of the ACM (JACM), 42(6):1115–1145, 1995.

[3] CHARIKAR, M. S.. **Similarity estimation techniques from rounding algorithms**. In: PROCEEDINGS OF THE THIRY-FOURTH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, p. 380–388. ACM, 2002.

[5] GIONIS, A.; INDYK, P.; MOTWANI, R. ; OTHERS. **Similarity search in high dimensions via hashing**. In: VLDB, volumen 99, p. 518–529, 1999.

[7] SHRIVASTAVA, A.; LI, P.. **Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 2321–2329, 2014.

[8] NEYSHABUR, B.; SREBRO, N.. **On symmetric and asymmetric lshs for inner product search**. In: PROCEEDINGS OF THE 32ND INTERNATIONAL CONFERENCE ON MACHINE LEARNING (ICML-15), p. 1926–1934, 2015.

[9] ZHANG, Z.; OOI, B. C.; PARTHASARATHY, S. ; TUNG, A. K.. **Similarity search on bregman divergence: Towards non-metric indexing**. Proceedings of the VLDB Endowment, 2(1):13–24, 2009.

[10] CAYTON, L.. **Fast nearest neighbor retrieval for bregman divergences**. In: PROCEEDINGS OF THE 25TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 112–119. ACM, 2008.

[12] GORISSE, D.; CORD, M. ; PRECIOSO, F.. **Locality-sensitive hashing for chi2 distance**. IEEE transactions on pattern analysis and machine intelligence, 34(2):402–409, 2012.

[13] KRSTOVSKI, K.; SMITH, D. A.; WALLACH, H. M. ; MCGREGOR, A.. **Efficient nearest-neighbor search in the probability simplex**. In: PROCEEDINGS OF THE 2013 CONFERENCE ON THE THEORY OF INFORMATION RETRIEVAL, p. 22. ACM, 2013.

[19] BLEI, D. M.. **Probabilistic topic models**. Communications of the ACM, 55(4):77–84, 2012.

[20] HOFMANN, T.. **Unsupervised learning by probabilistic latent semantic analysis**. Machine learning, 42(1-2):177–196, 2001.

[25] MU, Y.; YAN, S.. **Non-metric locality-sensitive hashing**. In: TWENTY-FOURTH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2010.

[26] KULLBACK, S.; LEIBLER, R. A.. **On information and sufficiency**. The annals of mathematical statistics, 22(1):79–86, 1951.

[28] COVIELLO, E.; MUMTAZ, A.; CHAN, A. B. ; LANCKRIET, G. R. G.. **That was fast! speeding up nn search of high dimensional distributions**. In: ICML, 2013.

[29] CHAUDHURI, K.; MCGREGOR, A.. **Finding metric structure in information theoretic clustering.** In: COLT, volumen 8, p. 10. Citeseer, 2008.

[30] JENSEN, J. H.; ELLIS, D. P.; CHRISTENSEN, M. G. ; JENSEN, S. H.. **Evaluation of distance measures between gaussian mixture models of mfccs**. In: INTERNATIONAL CONFERENCE ON MUSIC INFORMATION RETRIEVAL, p. 107–108, 2007.

[31] DO, M. N.; VETTERLI, M.. **Wavelet-based texture retrieval using generalized gaussian density and kullback-leibler distance**. IEEE transactions on image processing, 11(2):146–158, 2002.

[32] BHATTACHARYA, A.; KAR, P. ; PAL, M.. **On low distortion embeddings of statistical distance measures into low dimensional spaces**. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, p. 164–172. Springer, 2009.

[36] NIELSEN, F.; PIRO, P. ; BARLAUD, M.. **Bregman vantage point trees for efficient nearest neighbor queries**. In: MULTIMEDIA AND EXPO, 2009. ICME 2009. IEEE INTERNATIONAL CONFERENCE ON, p. 878–881. IEEE, 2009.

[38] ANDONI, A.; RAZENSHTEYN, I.. **Optimal data-dependent hashing for approximate near neighbors**. In: PROCEEDINGS OF THE FORTY-SEVENTH ANNUAL ACM ON SYMPOSIUM ON THEORY OF COMPUTING, p. 793–801. ACM, 2015.

[39] JI, J.; LI, J.; YAN, S.; ZHANG, B. ; TIAN, Q.. **Super-bit locality-sensitive hashing**. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, p. 108–116, 2012.

[40] DUDA, R. O.; HART, P. E. ; STORK, D. G.. **Pattern classification**. John Wiley & Sons, 2012.

[42] TURTLE, H.; FLOOD, J.. **Query evaluation: strategies and optimizations**. Information Processing & Management, 31(6):831–850, 1995.

[43] ZOBEL, J.; MOFFAT, A. ; RAMAMOHANARAO, K.. **Inverted files versus signature files for text indexing**. ACM Transactions on Database Systems (TODS), 23(4):453–490, 1998.

[44] BAYARDO, R. J.; MA, Y. ; SRIKANT, R.. **Scaling up all pairs similarity search**. In: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, p. 131–140. ACM, 2007.

[45] ANASTASIU, D. C.; KARYPIS, G.. **L2ap: Fast cosine similarity search with prefix l-2 norm bounds**. In: DATA ENGINEERING (ICDE), 2014 IEEE 30TH INTERNATIONAL CONFERENCE ON, p. 784–795. IEEE, 2014.

[46] BALLARD, G.; KOLDA, T. G.; PINAR, A. ; SESHADHRI, C.. **Diamond sampling for approximate maximum all-pairs dot-product (mad) search**. In: DATA MINING (ICDM), 2015 IEEE INTERNATIONAL CONFERENCE ON, p. 11–20. IEEE, 2015.

[47] MAHONEY, M. W.; DRINEAS, P.. **Cur matrix decompositions for improved data analysis**. Proceedings of the National Academy of Sciences, 106(3):697–702, 2009.

[48] WALT, S. V. D.; COLBERT, S. C. ; VAROQUAUX, G.. **The numpy array: a structure for efficient numerical computation**. Computing in Science & Engineering, 13(2):22–30, 2011.

[49] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. ; OTHERS. **Scikit-learn: Machine learning in python**. Journal of Machine Learning Research, 12(Oct):2825–2830, 2011.

[50] LIU, T.; MOORE, A. W. ; GRAY, A.. **New algorithms for efficient high-dimensional nonparametric classification**. Journal of Machine Learning Research, 7(Jun):1135–1158, 2006.

[51] AUMÜLLER, M.; BERNHARDSSON, E. ; FAITHFULL, A.. **Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms**. In: INTERNATIONAL CONFERENCE ON SIMILARITY SEARCH AND APPLICATIONS, p. 34–49. Springer, 2017.

[52] NEWLING, J.; FLEURET, F.. **Fast k-means with accurate bounds**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 936–944, 2016.

[53] HOARE, C. A.. **Algorithm 65: find**. Communications of the ACM, 4(7):321–322, 1961.

[54] BANERJEE, A.; MERUGU, S.; DHILLON, I. S. ; GHOSH, J.. **Clustering with bregman divergences**. Journal of machine learning research, 6(Oct):1705–1749, 2005.

[55] TERADA, Y.; LUXBURG, U.. **Local ordinal embedding**. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, p. 847–855, 2014.

[56] RASIWASIA, N.; MORENO, P. J. ; VASCONCELOS, N.. **Bridging the gap: Query by semantic example**. IEEE Transactions on Multimedia, 9(5):923–938, 2007.

[57] EVERINGHAM, M.; VAN GOOL, L.; WILLIAMS, C.; WINN, J. ; ZISSERMAN, A.. **Pascal visual object classes challenge results**. Available from www. pascal-network. org, 1(6):7, 2005.