Locality Sensitive Hashing Revisited: Filling the Gap Between Theory and Algorithm Analysis

Hongya Wang Jiao Cao
School of Computer Science and Technology
Donghua University
Shanghai, China
{hywang@dhu.edu.cn, caojiao2011@yahoo.cn}

LihChyun Shu
College of Management
National Cheng Kung University
Taiwan
shulc@mail.ncku.edu.tw

Davood Rafiei
Department of Computing Science
University of Alberta
Edmonton, Canada
drafiei@cs.ualberta.ca

ABSTRACT

Locality Sensitive Hashing (LSH) is widely recognized as one of the most promising approaches to similarity search in high-dimensional spaces. Based on LSH, a considerable number of nearest neighbor search algorithms have been proposed in the past, with some of them having been used in many real-life applications. Apart from their demonstrated superior performance in practice, the popularity of the LSH algorithms is mainly due to their provable performance bounds on query cost, space consumption and failure probability.

In this paper, we show that a surprising gap exists between the LSH theory and widely practiced algorithm analysis techniques. In particular, we discover that a critical assumption made in the classical LSH algorithm analysis does not hold in practice, which suggests that using the existing methods to analyze the performance of practical LSH algorithms is a conceptual mismatch. To address this problem, a novel analysis model is developed that bridges the gap between the LSH theory and the method for analyzing the LSH algorithm performance. With the help of this model, we identify some important flaws in the commonly used analysis methods in the LSH literature. The validity of this model is verified through extensive experiments with real datasets.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Indexing Methods; F.0 [**Theory of Computation**]: General

Keywords

Locality Sensitive Hashing, Probabilistic Algorithm, Algorithm Analysis

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

http://dx.doi.org/10.1145/2505515.2505765.

1. INTRODUCTION

The nearest neighbor (NN) search in high dimensional spaces under some distance function is of great importance in areas such as database, information retrieval, data mining, pattern recognition and machine learning. When the dimensionality of the data is low, a number of access methods (*R-tree* [13], *kd-tree* [6]) have been shown to perform well in practice. As the dimensionality grows higher (e.g., larger than 50), however, it has long been recognized that these traditional tree-based indexing methods offer little improvement over a linear scan that compares a query to every point of the dataset [21]. This phenomenon is often called "the curse of dimensionality".

To overcome the search time bottleneck, there have been several proposals of approximation algorithms that trade precision for speed [1, 4]. In this paper, we focus on one of the most promising approximate NN search algorithms called Locality Sensitive Hashing, which has been widely applied in a variety of domains, including web clustering, computer vision and computational biology [3, 17].

We should point out that LSH does not solve approximate NN queries directly, and was originally proposed for a different problem called c-approximate r-near neighbor search (see Definition 1 in Section 2.2) in high-dimensional spaces [16]. The key idea behind LSH is that it is possible to devise hash functions such that points close to each other in some metric space collide with higher probability than do points that are far from one another. If a family $\mathcal{H}$ of hash functions satisfies the condition listed in Definition 2 (presented in Section 2.2), $\mathcal{H}$ is called locality sensitive. To date, several LSH families have been discovered for different similarity (dissimilarity) measures such as Hamming distance [16], l_s distance [9] with $s \in [0, 2]$, Jaccard coefficient [7], Cosine distance [8], etc.

Given a particular metric and the corresponding LSH family $\mathcal{H}$, an efficient approximate NN search algorithm can be easily devised, which will be discussed in detail shortly. Simply speaking, during a preprocessing phase, a number of hash tables or buckets are created by hashing all points in the dataset using independent hash functions randomly chosen from $\mathcal{H}$. To process a query, one only need to hash the query point and retrieve the elements stored in buckets containing that point.

Thanks to the soundness of the LSH theory, algorithms based on LSH are capable of providing, with some constant failure probability, excellent asymptotic performance

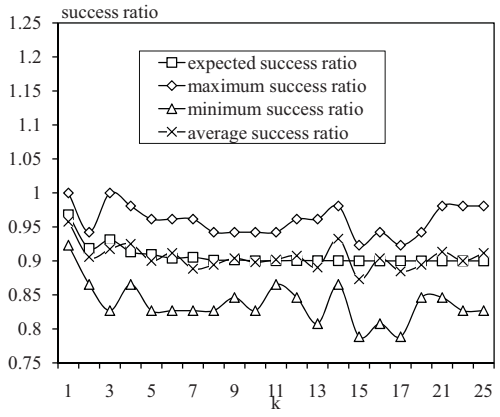

Figure 1: $SR_r^{k,L}$ **vs.** k $(r = 0.4, \delta = 0.1)$

in terms of space usage and query cost. Such theoretical results were originally proved in the influential paper by Indyk and Motwani for their classic LSH algorithm [16], and ever since then, the method of proof developed in their paper has been widely used by the vast majority of other LSH variants, e.g., LSH-forest [5], LSB-tree [20], Multi-probe LSH [18], and C2LSH [11], in order to obtain the corresponding theoretical guarantees.

Surprisingly, in our experimental study, we observe a significant deviation between the real and expected performance of the classic LSH algorithm proposed in [16]. For example, consider one typical experiment shown in Figure 1, where *the expected success ratio* represents the expected performance and the *maximum, minimum and average success ratios* are the actual best, worst and average performance, respectively[1]. In theory, the actual success ratios, including the maximum, minimum and average, should well match the expected performance [3], but the fact, shown in Figure 1, is that an obvious gap exists between the maximum/minimum and expected algorithm performance (12 percent maximum difference)[2], which contradicts the results obtained using the widely-accepted analysis method that will be reviewed in Section 2.3.

Clearly, such a significant difference cannot be simply attributed to "random errors" in the experiment. Instead, as will be discussed in the rest of this paper, this performance gap reveals a subtle problem that has been ignored for years. Motivated by this observation, we carry out a systematic in-depth investigation of LSH with the following important findings:

- We observe that a critical assumption used in the LSH algorithm analysis is incorrect from a practical perspective, which may lead to problematic theoretical results.
- A novel model is developed to bridge the gap between the expected theoretical performance and the observed performance of the LSH algorithms. Based on this model, we identify a fatal flaw in the proof of Theorem 4 in [16], which implies that, in theory, the performance guarantees of almost all LSH algorithm implementations are questionable. In particular, the constant probability, with which the LSH algorithms report the approximate r-near neighbors of a query as proved in [16] and many other LSH papers, may not exist at all.
- We also inspect the parameter selection approach for the classic LSH algorithm, and show that the actual failure probability for the randomized r-near neighbor reporting problem (Definition 4) fluctuates around, instead of being always equal to, the specified failure rate with the chosen values of k and L. A practical implication of this finding is that one can more precisely predict the behavior of LSH using the model proposed in this paper.
- To verify our arguments, extensive experiments have been performed with several real datasets. Experimental results confirm the validity of our model and demonstrate that the existing analysis model (although incorrect) is a good approximation of the accurate one proposed in this paper when the cardinality of dataset is sufficiently large.

The rest of this paper is organized as follows. A brief overview of LSH and the related work are provided in Section 2. Section 3 presents the main theoretical results of this paper. An Empirical study to validate our model is described in Section 4. Finally, we conclude this paper with a summary of our results and a brief discussion on future work in Section 5.

2. BRIEF OVERVIEW OF LSH AND THE RELATED WORK

2.1 The nearest neighbor search problem

Locality sensitive hashing is closely related to the problem of nearest neighbor (NN) search in high-dimensional spaces under some distance metric. Thus, before presenting the technical details of LSH, we first give the necessary notations and definitions as to NN search.

We use $\mathcal{O}$ to denote a set of data points, and assume all points $o_i \in \mathcal{O}$ belong to a d-dimensional space $\Re^d$. Let o^j denote the jth coordinate of o, $j = 1, 2, \ldots, d$. For any two points o and q in $\mathcal{O}$, the distance (l_s norm) between them is defined as follows:

$$\|q - o\|_s = (\sum_{j=1}^{d} |q^j - o^j|^s)^{1/s} \quad (1)$$

Given a dataset $\mathcal{O}$, the nearest neighbor $NN_q \in \mathcal{O}$ of a query point q under l_s norm is defined as follows:

$$\|q - NN_q\|_s \leq \|q - o\|_s, \forall o \in \mathcal{O} \quad (2)$$

To simplify notations, we often skip the subscript s in the sequel.

[1]To avoid blurring the main problem we want to explain, the discussion of the parameters such as k, L, r and δ in Figure 1 is deferred till Section 2.

[2]In Figure 1, one can also see a nice match between the actual average success ratio and the expected performance, which well confirms our theoretical analysis that will be discussed in Section 3.5.2.

NN search is aimed at, given a dataset and a query point, finding a way to (efficiently) report the nearest neighbor of the query. Essentially, the NN search problem is a concrete instance of an optimization problem since its goal is to find the point that minimizes a chosen objective function (the distance to the query point in this case) [3]. It has long been recognized that, for large enough d, most solutions to NN search offer little improvement over a linear scan that compares a query to every point from the dataset. This phenomenon is often called "the curse of dimensionality" [21].

In view of the difficulty in solving this problem directly, Indyk et al. propose to first deal with the r-near neighbor (rNN) search problem, which is originally termed as Point Location in Equal Ball (PLEB) in [16]. rNN search is the decision version of the NN search problem. A point o is called the r-near neighbor of query q if the distance between o and q is at most r. The purpose of rNN search is to report some r-near neighbor of q if there exists at least one rNN of q. Consider the running example depicted in Figure 2, o_1 is the NN of q and both o_1 and o_2 are r-near neighbors of q.

Algorithms supporting rNN search still suffer from the curse of dimensionality. Fortunately, in many areas, approximate nearest neighbors (ANN) are also acceptable [1]. Given a query point q, a data point, say ANN_q, is called an ANN of q under l_s norm if it satisfies $\|q - ANN_q\|_s \leq c\|q - NN_q\|_s$, where $c > 1$ is the approximate factor. The decision version of the ANN search problem, namely, the c-approximate r-near neighbor search problem (previously known as the ϵ-PLEB problem [16]), is defined as follows:

DEFINITION 1 ([3]). *Given a set $\mathcal{O}$ of points in a d-dimensional space $\Re^d$ and parameter $r > 0$, construct a data structure such that, given any query point q, if there exists an r-near neighbor of q in $\mathcal{O}$, it reports some c-approximate r-near neighbor of q in $\mathcal{O}$.*

In Figure 2, o_1 and o_2 are obviously the c-approximate r-near neighbors (crNN) of q since their distances to q are less than r. Besides, o_3 is a crNN of q as well because it falls in the outer circle.

It has been shown that the algorithm for crNN search can be used as a building block to solve the ANN search problem [14, 16]. Therefore, finding an efficient solution to crNN search is a pivotal step towards achieving the final goal. Intuitively, the crNN search problem is much weaker than the corresponding exact version, namely, the rNN search problem. The fact, however, is that devising an efficient deterministic algorithm for this problem remains difficult. To this end, by trading certainty for speed, Indyk et al. propose locality sensitive hashing, which provides attractive sub-linear search time at the expense of some constant failure probability.

2.2 Locality sensitive hashing

Locality sensitive hashing is one of the most popular (approximate) NN search algorithms in high-dimensional spaces. The rationale behind this algorithm is that, by using specific hashing functions, we can hash the points such that the probability of collision for data points which are close to each other is much higher than that for those which are far apart.

In the rest of this paper, we use $\mathcal{H}$ to denote a family of hash functions mapping $\Re^d$ to some universe $\mathcal{U}$. For any two points o and q, a hash function h is chosen from $\mathcal{H}$ uniformly at random. If the probability that these two points collide ($h(q) = h(o)$) satisfies the following condition, the family $\mathcal{H}$ is called locality sensitive.

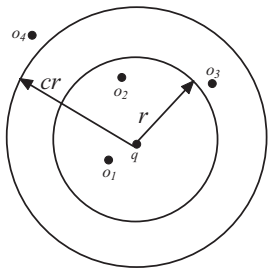

Figure 2: A running example for illustrating the concepts and definitions discussed in Section 2.

DEFINITION 2 (LOCALITY SENSITIVE HASHING [16]). *A family $\mathcal{H}$ of hash functions is called (r, cr, p_1, p_2)-sensitive if, for any two points o, $q \in \Re^d$ and $h \in \mathcal{H}$, it satisfies the following:*

- *if $\|q - o\| \leq r$ then $Pr_{\mathcal{H}}[h(q) = h(o)] \geq p_1$*
- *if $\|q - o\| \geq cr$ then $Pr_{\mathcal{H}}[h(q) = h(o)] \leq p_2$*

For an LSH family to be useful, it has to satisfy $p_1 > p_2$.

Given an LSH family $\mathcal{H}$, the classic LSH algorithm, denoted by LSH_C, works as follows. For parameters k and L, L functions $g_j(q) = (h_{1,j}(q), \ldots, h_{k,j}(q))$ are chosen, where $h_{i,j} (1 \leq i \leq k, 1 \leq j \leq L)$ is drawn independently and uniformly at random from $\mathcal{H}$ [16]. For each point $o \in \mathcal{O}$, the bucket $g_j(o)$ is computed, for $j = 1, \ldots, L$, and then o is inserted into the corresponding bucket. To process a query q, one has to compute $g_j(q), j = 1, \ldots, L$, first, and then retrieve all points that lie in at least one of these buckets. For all candidate points retrieved, a filtering procedure is performed to calculate the distance of each point to the query. Basically, there are two principal filtering strategies according to [3].

1. Interrupt the search after finding the first $\mathcal{T}$ points for some integer $\mathcal{T}$, and return the point with the minimum distance to the query point
2. Continue the search until all points that collide with the query are processed, and the points whose distances to the query are less than r are returned.

These two strategies, combined with the data structure discussed earlier, result in different behaviors of LSH_C. In particular, the first strategy is aimed at solving the randomized crNN search problem, which is defined as follows.

DEFINITION 3 ([3]). *Given a set $\mathcal{O}$ of points in a d-dimensional space $\Re^d$, and parameters $r > 0$, $\delta > 0$, construct a data structure such that, given any query point q, if there exists an r-near neighbor of q in $\mathcal{O}$, it reports some c-approximate r-near neighbor of q in $\mathcal{O}$ with probability $1 - \delta$.*

Unlike its deterministic counterpart, the solution to randomized crNN search may produce false positive. To be

specific, take, again, the scenario illustrated in Figure 2 as an example. Although o_4 falls outside the outer circle, it still has a chance to be reported as the crNN of q.

Strategy 1 is of great importance in theory as it owns attractive asymptotical query and space performance, provided that the values of parameters k and L are properly chosen. Despite its theoretical significance, Strategy 1 is rarely used in practice because of the undesirable quality of result compared to Strategy 2.

Strategy 2 enables us to solve the randomized r-near neighbor reporting problem, which is defined as follows.

DEFINITION 4 ([3]). *Given a set $\mathcal{O}$ of points in a d-dimensional space $\Re^d$, and parameters $r > 0$, $\delta > 0$, construct a data structure such that, given any query point q, reports each r-near neighbor of q in $\mathcal{O}$ with probability $1-\delta$.*

The problem defined above is a special case of the randomized version of the r-near neighbor search problem, where all (not just some) rNNs of q are returned. Note that Strategy 2 provides much better result quality since all data points in the answer set are rNNs of the query, and thus no false positive is returned. There, however, might still be false negative if some rNNs do not collide with the query. To see this, consider the example shown in Figure 2. While both o_1 and o_2 are rNNs of q, Strategy 2 might only report o_2 as the final result.

Instead of reporting all rNNs of the query, a practical implementation of Strategy 2 often sorts all rNNs in ascending order of their distances to the query, and only returns the nearest neighbor (the K-nearest neighbor) of the query by picking the first point (the first K points) from the sorted list. It is evident that Strategy 2, together with the sorting procedure, can provide much better quality of result compared to Strategy 1. The price, however, we have to pay is the much higher query time and the lack of theoretical guarantee on the running time of queries.

2.3 Theoretical performance of the classic LSH algorithm

The most appealing feature of the algorithms based on LSH is that, by setting proper values for parameter k and L, the query times of these algorithms are much lower than those of other competitors, either theoretically or practically. Next, we present some important theoretical results as to the classic LSH algorithm, under Strategy 1 and Strategy 2, respectively.

In Strategy 1, if we set $k = log_{1/p_2} n$ and $L = n^\rho$, where $\rho = \frac{ln 1/p_1}{ln 1/p_2}$, then for the randomized crNN search problem, the query time is dominated by $O(n^\rho)$ distance computations and $O(n^\rho log_{1/p_2} n)$ evaluations of hash functions from $\mathcal{H}$, with some constant failure probability $\delta < 1$ [12, 16, 9].

In Strategy 2, failure probability and query time also depend heavily on k and L. Different from Strategy 1, no theoretical guarantee exists for query time and, in the worst case, the cost to process a query might be as high as $O(n)$ [3]. Such a situation will occur if a huge amount of data points collide with the query point due to a bad choice of parameter values. Fortunately, for many real datasets, a careful choice of values for k and L leads to a sublinear query time [5, 20].

A commonly used parameter selection method for Strategy 2 is as follows [3, 15, 19]. Assume o is an r-near neighbor of q, and consider any parameter k. For any function g_j, the probability that $g_j(o) = g_j(q)$ is at least p_1^k. Therefore, for some $j = 1, 2, \ldots, L$, the probability that $g_j(o) = g_j(q)$ is at least $1 - (1 - p_1^k)^L$. By setting $L = log_{1-p_1^k} \delta$, we have $(1 - p_1^k)^L \leq \delta$. As a result, any r-near neighbor of q is returned by Strategy 2 with probability at least $1 - \delta$. How to choose a proper k is not a trivial issue and out of the scope of this paper, we would like refer interested readers to [3] for a detailed discussion.

3. REVISITING LOCALITY SENSITIVE HASHING

3.1 Preliminaries

In the rest of this paper, we use a slightly different description of locality sensitive hashing for our purpose, which is defined as follows.

DEFINITION 5. *A family $\mathcal{H}$ is called locality sensitive if for any two points o, $q \in \Re^d$, it satisfies*

$$Pr_{\mathcal{H}}[h(q) = h(o)] = f_{\mathcal{H}}(r) \tag{3}$$

where r is the distance between q and o, and $f_{\mathcal{H}}(r)$ is a monotonically decreasing function of r.

Please note that, although stated in a different form, Definition 5 is essentially equivalent to the original definition given in [16]. Actually, our definition is very similar to the LSH definition used in [8], which focuses on Jaccard and Cosine similarity measures.

For ANN search in a d-dimensional space under some l_s norm, the analytical expressions of $f_{\mathcal{H}}(r)$ have been discovered for $s \in [0, 2]$. Two particular instances ($s = 1, 2$) will be discussed shortly in Section 3.2 and Section 4.1, respectively.

By Definition 5, it is easy to see that $Pr_{\mathcal{H}}[h(q) = h(o)]$ actually depends only on the distance r between q and o, and has nothing to do with their specific locations in the d-dimensional space. In other words, each coordinate of both q and o can take arbitrary values as long as the distance between them remains to be r. Hence, we will use $Pr_{\mathcal{H}}[r]$ and $Pr_{\mathcal{H}}[h(q) = h(o)]$ ($\|q - o\| = r$) interchangeably in the following sections.

3.2 Some implication of using LSH in practice

In this section, we reexamine the rationale behind LSH and illustrate what is the actual implication for practitioners if a family of hash functions is locality sensitive.

Recall that the existence of the LSH families is the foundation for various different LSH algorithms. To make it easy to follow, we use the LSH family for Hamming space as an example to elaborate our main argument. It is worth noting that the results presented in this paper are not confined to this special case only, but applicable to all LSH families.

In Hamming space, the data points are binary, that is, each coordinate is either 0 or 1, and the dissimilarity between points is measured by the Hamming distance, which is defined as the number of positions at which the values are different. For binary data points, the Hamming distance is equivalent to the Manhattan distance (l_1 norm). Figure 3 depicts a simple example with two binary points, where $d = 10$ and the distance between q and o_1 is equal to 3.

A locality sensitive family of functions $\mathcal{H}_H$ exists for Hamming space. To be specific, $\mathcal{H}_H$ contains all functions h^i that map data points from $(0,1)^d$ to $(0,1)$ such that $h^i(o) = o^i$. Take Figure 3 as an example, in total there are 10 different hash functions in $\mathcal{H}_H$ since $d = 10$. If we choose h^2 from $\mathcal{H}_H$ to map these two points into a $(0,1)$ space, we get $h^2(q) = 1$ and $h^2(o_1) = 0$.

To see why $\mathcal{H}_H$ is locality sensitive, let's start with the concrete example depicted in Figure 3. Recall that the distance between o_1 and q is 3, and the dimensionality is 10. Therefore, if we choose a hash function h uniformly at random from $\mathcal{H}_H$ to hash o_1 and q, the probability that $Pr_{\mathcal{H}_H}[h(q) = h(o_1)]$ is equal to $1 - 3/10$.

To extend this observation to a more general case, consider two binary points q and o in a d-dimensional space under the Hamming metric. Assume the distance between q and o is r. It is not difficult to see that the probability that $Pr_{\mathcal{H}_H}[h(q) = h(o)]$ is equal to the proportion of coordinates on which p and o agree, that is, $Pr_{\mathcal{H}_H}[h(q) = h(o)] = 1 - \|q-o\|_1/d = 1 - r/d$. Since $1 - r/d$ decreases monotonically with r, it follows that $\mathcal{H}_H$ is locality sensitive according to Definition 5.

Having $\mathcal{H}_H$ at hand, we are now ready to scrutinize the real implication of LSH from a statistical perspective. According to the Law of Large Numbers (LLN), a probabilistic interpretation of $Pr_{\mathcal{H}_H}[r] = 1 - r/d$ is that, for any two points o and q with distance r, if we choose a large number, say $10,000$, of hash functions uniformly at random from $\mathcal{H}_H$, then the proportion of hash functions over which o and q collide will be very close to $1 - r/d$. Although this interpretation is easy to understand, it, however, does not lead to a feasible LSH algorithm directly.

To illustrate, consider a naïve (impractical) LSH algorithm LSH_N, which intends to solve the same target problem as LSH_C. For ease of presentation, we assume $k = 1$ and $L = 1$. Given a dataset $\mathcal{O}$ and a query q, in order to find the r-near neighbors of q, LSH_N works as follows: (1) generates a large number, say $10,000$, hash functions uniformly at random from $\mathcal{H}_H$ and inserts all data points in $\mathcal{O}$ into the $10,000$ hash tables; (2) for each data point o in $\mathcal{O}$, computes the collision ratio, i.e., the proportion of these $10,000$ hash functions over which o and q collide; if the collision ratio is greater than $1 - r/d$, then reports o as as an r-near neighbor of q.

The problems with LSH_N are obvious. First, to reduce the number of false negative caused by the probabilistic nature of this algorithm, one has to build a large number of hash tables, which may incur prohibitively huge storage cost. Second, even if the number of hash tables is sufficiently large, it is still difficult or even impossible to obtain a theoretical bound on failure probability, i.e., the probability of a false negative being produced.

While LSH_N is impractical, almost all LSH theoretical results are only valid for such an infeasible algorithm. Next, we will show the mismatch between the practical LSH algorithm and the widely practiced LSH analysis techniques.

3.3 Some of the problems with the existing analysis methods

After examining the exact meaning of LSH and the infeasibility of LSH_N, we focus on a practical LSH algorithm, i.e., LSH_C, in this section. For ease of presentation, we continue to assume $k = 1$ and $L = 1$.

Recall that, to build the core data structure, LSH_C first randomly chooses a hash function, say h, from $\mathcal{H}_H$, and then maps all data points in $\mathcal{O}$, using h, into a $(0,1)$ space. Given a radius r, to find all rNNs of a query q, the algorithm (under Strategy 2) scans all data points lying in the bucket $h(q)$, and reports points whose distances to q are less than r. As discussed in Section 2.3, the failure probability of this algorithm is no greater than $1 - Pr_{\mathcal{H}_H}[r] = r/d$.

Although the above analysis seems quite reasonable (given the fact that $\mathcal{H}_H$ is locality sensitive and $Pr_{\mathcal{H}_H}[r] = 1 - r/d$), we argue that this commonly used analysis method is problematic, and thus may lead to questionable results. Specifically, we notice that a critical assumption in this method is not correct, which is stated as follows.

Observation 1. *For any fixed hash function $h \in \mathcal{H}$ chosen by a concrete instance of LSH_C, the probability that a random pair of points with distance r collides over h, denoted by $Pr_h[r]$, may be not equal to $Pr_{\mathcal{H}}[r]$.*

To make this observation easy to comprehend, consider two datasets shown in Figure 4, where the distance of each pair of points is equal to 1. As discussed earlier, $Pr_{\mathcal{H}}[1] = 1 - 1/10 = 9/10$ for both datasets. Unlike $Pr_{\mathcal{H}}[1]$, however, $Pr_h[1]$ has different values for different datasets. For example, if the hash function $h^3(o) = o^3$ is chosen for use, then for dataset 1, the probability of collision for a random pair of points is 1, while for dataset 2, the probability of collision is only $3/4$. In fact, a more careful observation will reveal that, even for the same dataset, choosing different hash functions, i.e., different h^i, $i = 1, \ldots, 10$, may lead to different $Pr_h[1]$!

Observation 1 tells us that (1) $Pr_h[r]$, instead of $Pr_{\mathcal{H}}[r]$, should be used to analyze the performance of LSH_C because, for a specific LSH_C instance being able to work, the chosen hash functions have to be fixed rather than "randomly chosen", and (2) Unlike $Pr_{\mathcal{H}}[r]$, which is determined by the nature of the LSH family, $Pr_h[r]$ depends only on the specific hash function chosen from $\mathcal{H}$ and the characteristics of dataset. Consequently, in essence, $Pr_{\mathcal{H}}[r]$ and $Pr_h[r]$ are totally different. The difference, however, is so subtle that it has been ignored for years. Here we omit the subscript H for $\mathcal{H}_H$ intentionally to emphasize that our observation actually holds for all LSH families. Before giving a detailed explanation of this difference, we need to first review a few terms in probability theory.

An experiment is any procedure that can be infinitely repeated and has a well-defined set of results. The collection of all results is called the sample space of the experiment. An element of the power set of the sample space is called an event. If the results that actually occur fall in a given event, the event is said to have occurred.

In the case of LSH, an experiment, for both $Pr_{\mathcal{H}}[r]$ and $Pr_h[r]$, has two inputs, namely, a hash function h_i and a pair of points $p_{j,r}$ with distance r, and two possible results, that is, collision or no collision for a single trial, denoted by the binary-valued variable $Col_{i,j}$, where $Col_{i,j} = 1$ means collision, and 0 otherwise. Although sharing the same procedure of experiment, $Pr_{\mathcal{H}}[r]$ and $Pr_h[r]$ differ significantly in the way the corresponding events are defined. To be specific, the event associated with $Pr_{\mathcal{H}}[r]$ is formally defined as $\{Col_{i,j} \mid p_{j,r}$ is fixed and the hash functions are randomly chosen from $\mathcal{H}\}$, while the relevant event for $Pr_h[r]$ is the result set $\{Col_{i,j} \mid h_i$ is fixed and the pair of points is randomly chosen from $\mathcal{O}\}$.

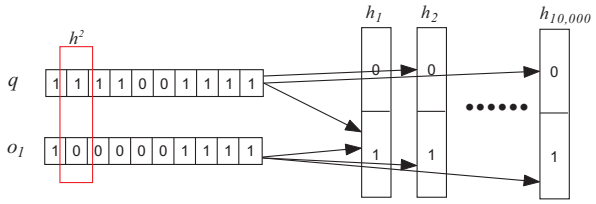

Figure 3: An illustration of the exact meaning of LSH. The probability of collision for q and o_1 can be estimated by the fraction of these 10,000 hash functions over which they collide.

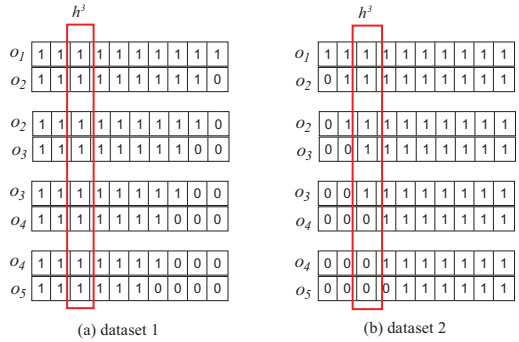

Figure 4: The probabilities of collision for different datasets using the same hash function are different. $Pr_{h^3}[1]$ for dataset 1 is equal to 1, whereas for dataset 2, it is only 3/4. Neither of them is equal to $Pr_{\mathcal{H}_H}[1]$, i.e., 9/10

Informally, to figure out $Pr_{\mathcal{H}}[r]$, the experiment is repeated by fixing the pair of points and randomly picking a hash function from $\mathcal{H}$, whereas for $Pr_h[r]$, the hash function is a constant factor and the randomness of the experiment depends only on the characteristics of dataset, e.g., the cardinality of dataset and data distribution.

In the LSH literature, the performance analysis of almost all LSH algorithms relies on the basic assumption $Pr_h[r] = Pr_{\mathcal{H}}[r]$, either explicitly or implicitly. The fact, however, is that while $Pr_{\mathcal{H}}[r]$ is identical for all pairs of points with distance r, $Pr_h[r]$ often varies with h. Consequently, the correctness of theoretical results obtained using the existing analysis method is questionable, and should be reinvestigated. Next, we will first give an important theorem to establish the quantitative relation between $Pr_h[r]$ and $Pr_{\mathcal{H}}[r]$. Then, some flaws in the performance analysis of LSH_C are identified.

3.4 Characterizing relation between $Pr_h[r]$ and $Pr_{\mathcal{H}}[r]$

THEOREM 1. *Given an LSH family $\mathcal{H}$ and a set $\mathcal{O}$ of d-dimensional data points, assume n hash functions are chosen uniformly at random from $\mathcal{H}$, and the number of pairs of points with distance r in $\mathcal{O}$ is m. $Pr_h[r]$ and $Pr_{\mathcal{H}}[r]$ are related in the following formula:*

$$Pr\Big[\lim_{m,n\to\infty} Pr_{\mathcal{H}}[r] = \sum_{i=1}^{n} Pr_{h_i}[r]/n\Big] = 1.$$

PROOF. For any hash function, say h_i, randomly chosen from $\mathcal{H}$, assume the number of pairs of points with distance r that collide over h_i is s_i, then the collision rate for h_i is s_i/m. On the other hand, for any pair of points with distance r, say $p_{j,r}$, in $\mathcal{O}$, assume the number of hash functions over which it collides is t_j, then the collision rate for $p_{j,r}$ is t_j/n. According to LLN, we have the following equations.

$$Pr\Big[\lim_{n\to\infty} Pr_{\mathcal{H}}[r] = t_j/n\Big] = 1, \forall j = 1, \ldots, m \tag{4}$$

$$Pr\Big[\lim_{m\to\infty} Pr_{h_i}[r] = s_i/m\Big] = 1, \forall i = 1, \ldots, n \tag{5}$$

By adding the system of equations together, we get

$$Pr\Big[\lim_{n\to\infty} mPr_{\mathcal{H}}[r] = \sum_{j=1}^{m} t_j/n\Big] = 1 \tag{6}$$

$$Pr\Big[\lim_{m\to\infty} \sum_{j=1}^{n} Pr_{h_i}[r] = \sum_{i=1}^{n} s_i/m\Big] = 1 \tag{7}$$

After simple equation transformation for (6) and (7) it follows

$$Pr\Big[\lim_{n\to\infty} nmPr_{\mathcal{H}}[r] = \sum_{j=1}^{m} t_j\Big] = 1 \tag{8}$$

$$Pr\Big[\lim_{m\to\infty} m\sum_{i=1}^{n} Pr_{h_i}[r] = \sum_{i=1}^{n} s_i\Big] = 1 \tag{9}$$

Since the total collision number of all pairs of points is a constant, we have

$$\sum_{j=1}^{m} t_j = \sum_{i=1}^{n} s_i \tag{10}$$

By substituting this equation into (9) we have

$$Pr\Big[\lim_{m,n\to\infty} Pr_{\mathcal{H}}[r] = \sum_{i=1}^{n} Pr_{h_i}[r]/n\Big] = 1 \tag{11}$$

We prove the theorem . □

Theorem 1 indicates that there exists an intrinsic relation between $Pr_h[r]$ and $Pr_{\mathcal{H}}[r]$ although they are defined from different angles. With Theorem 1, we are now ready to reexamine the methods for LSH_C performance analysis.

3.5 Some of the flaws in the classic LSH algorithm analysis

In this section, some flaws in the methods by which the theoretical performance of LSH_C was previously analyzed will be demonstrated. Please note that the problems to be identified are only genuine for the LSH_C analysis; for the impractical LSH algorithm (LSH_N), all existing theoretical results still hold. Before presenting our main results, we need to make an assumption that is necessary for the following discussion.

As shown in Theorem 1, although $Pr_h[r]$ and $Pr_{\mathcal{H}}[r]$ are intrinsically connected, the relation between them is not deterministic since $\sum_{i=1}^{n} Pr_{h_i}[r]/n$ only *converges almost surely* towards $Pr_{\mathcal{H}}[r]$ when m and n approach infinity. Thus, it is difficult to examine the existing methods merely under such an uncertain condition. As a workaround, we assume $Pr_{\mathcal{H}}[r] = \sum_{i=1}^{n} Pr_{h_i}[r]/n$ to ease our analysis. Please note that, according to LLN, this assumption is reasonable if n and m are sufficiently large.

Recall that Strategy 1 and Strategy 2 are two principal filtering approaches used in LSH_C, and their performance has been discussed in Section 2.3. Next, we will investigate what's wrong with the performance analysis for these two strategies.

3.5.1 Strategy 1

The theoretical performance of LSH_C under Strategy 1 was originally proved in Theorem 4 in [16], and the corresponding method of proof has been accepted as the standard analysis technique in the LSH literature thereafter. Being aware of the difference between $Pr_h[r]$ and $Pr_{\mathcal{H}}[r]$, we, however, point out the following observation:

FACT 1. *An important premise, based on which the theoretical results in Theorem 4 in [16] were derived, does not hold in practice actually.*

PROOF. A key step in the proof of Theorem 4 in [16] is to show that, given two data points o and q with distance less than r, $g_j(o) = g_j(q)$ holds for some $j = 1, \ldots, L$ with a constant probability if proper values of k and L are chosen.

In order to obtain such a constant probability, the authors use $Pr_{\mathcal{H}}[g_j(o) = g_j(q)] = Pr_{\mathcal{H}}[r]^k$ ($\|o - q\| = r$) as an indispensable premise in the proof. With the help of Observation 1, we know that this premise should be replaced by

$$Pr_{\mathcal{H}}[g_j(o) = g_j(q)] = \prod_{i=1}^{k} Pr_{h_i}[r]$$

Under the assumption that $Pr_{\mathcal{H}}[r] = \sum_{i=1}^{n} Pr_{h_i}[r]/n$, it is easy to see that $Pr_{\mathcal{H}}[r]^k \geq \prod_{i=1}^{k} Pr_{h_i}[r]$ according to Observation 2 below. As a result, it follows that $Pr_{\mathcal{H}}[g_j(o) = g_j(q)] \leq Pr_{\mathcal{H}}[r]^k$, which contradicts the widely-held belief that $Pr_{\mathcal{H}}[g_j(o) = g_j(q)] = Pr_{\mathcal{H}}[r]^k$ first used in [16][3]. □

[3]Without the assumption that $Pr_{\mathcal{H}}[r] = \sum_{i=1}^{n} Pr_{h_i}[r]/n$, $Pr_{\mathcal{H}}[g_j(o) = g_j(q)]$ will be less than, equal to or greater than $Pr_{\mathcal{H}}[r]^k$ with some unknown probabilities respectively, which still contradicts $Pr_{\mathcal{H}}[g_j(o) = g_j(q)] = Pr_{\mathcal{H}}[r]^k$.

OBSERVATION 2. *Given $\sum_{i=1}^{m} u_i = C$, where $u_i \in [0, 1], i = 1, 2, \ldots, m$ and C is a constant, $\prod_{i=1}^{m} u_i$ is maximized when the variance of u_i is zero, i.e., $u_1 = u_2 = \cdots = u_m$, and the value of $\prod_{i=1}^{m} u_i$ decreases as the variance of u_i increases.*

It is worth noting that Fact 1 does not rule out the existence of such a constant probability. The only implication of Fact 1 is that all theoretical results, which depend on the condition that $Pr_{\mathcal{H}}[g_j(o) = g_j(q)] = Pr_{\mathcal{H}}[r]^k$, are questionable. Considering the uncertain relation between $Pr_{\mathcal{H}}[r]$ and $\sum_{i=1}^{n} Pr_{h_i}[r]/n$, we conjecture that, for LSH_C, it might be difficult to obtain a constant failure probability like the one given in [16].

3.5.2 Strategy 2

Next, we look into LSH_C under Strategy 2. Recall that in Strategy 2, given k and δ, L should be at least equal to $log_{1-p_1^k}\delta$ in order to achieve a failure probability no greater than δ. In practice, L is often chosen such that $(1-p_1^k)^L = \delta$. Using our notations, L can be rewritten as

$$L = log_{1-Pr_{\mathcal{H}}[r]^k}\delta \tag{12}$$

By Observation 1, the probability of collision for a random pair of points with distance r is $P_{h_i}[r]$. As a result, given the L value calculated by Equation (12), the actual failure probability, denoted by $Pr_{\mathcal{A}}[r]$, should be $\prod_{j=1}^{L}(1 - \prod_{i=1}^{k} Pr_{h_i}[r])$ instead of δ. Since $Pr_h[r]$ varies with h, $Pr_{\mathcal{A}}[r]$ cannot be a constant as well. Interestingly enough, although $Pr_{\mathcal{A}}[r]$ itself is a random variable since all hash functions are randomly chosen, its expectation is exactly equal to δ.

THEOREM 2. $E\Big[Pr_{\mathcal{A}}[r]\Big] = \delta$

PROOF. Omitted due to space limitation. The full version is available on request. □

An immediate corollary of Theorem 2 is that the actual failure probability fluctuates around, rather than is always equal to, the specified failure rate δ, which will be justified experimentally in Section 4.4.2

4. EXPERIMENTAL VALIDATION

In this section, we conduct extensive experiments to verify the validity of our arguments presented in this paper. Section 4.1 describes the hash functions to be examined. Section 4.2 lists datasets with which the experiments are carried out. Section 4.3 elaborates on how data are preprocessed. Finally, the experimental results are reported in Section 4.4.

4.1 The LSH family

In [9], the authors propose a popular LSH family for l_2 norm, denoted by $\mathcal{H}_E$, based on the concept of p-stable distributions. The LSH function is defined as follows:

$$h(o) = \left\lfloor \frac{\vec{a} \cdot \vec{o} + b}{w} \right\rfloor \tag{13}$$

where $\vec{o}$ denotes the d-dimensional vector representation of a data point o; $\vec{a}$ is a d-dimensional vector, each component of which is drawn independently from Gaussian distribution defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$; $\vec{a} \cdot \vec{o}$ represents the inner product of $\vec{a}$ and $\vec{o}$; b is a real number chosen uniformly from $[0, w]$, and w is a constant that has to be specified a priori.

Under $\mathcal{H}_E$, the probability of collision for any two points with distance r is calculated as follows:

$$Pr_{\mathcal{H}_E}[r] = \int_0^w \frac{1}{r} f(\frac{t}{r})(1 - \frac{t}{w})dt \tag{14}$$

where $f(x) = 2g(x)$.

We choose $\mathcal{H}_E$ to verify our arguments for two reasons: (1) $\mathcal{H}_E$ is a very popular LSH family and has many successful applications in practice [20]; (2) A nice implementation of the LSH algorithm using $\mathcal{H}_E$, i.e., the E2LSH package, is publicly available [2]. Thus, it is easy for interested readers to repeat our experiments.

4.2 Datasets

We perform the experiments with several real datasets, including mnist1k, mnist10k, audio10k and color10k. All data are normalized such that each dimension has domain $[0, 1]$. The distance metric is Euclidean distance.

Mnist. The mnist dataset[4] consists of 60,000 handwritten pictures, and each picture has 28×28 pixels, each of which is represented by an integer in the range of $[0, 255]$. Thus, each data point is 784-dimensional. For our purpose, we use two subsets of the original mnist dataset, i.e., mnist10k and mnist1k, because the number of points in them is already large enough to verify our claim. Mnist10k is a test set of 10,000 points used in [10]. Mnist1k has only 1000 points and can be downloaded along with the E2LSH package.

Audio and Color. The audio dataset[5] contains more than 50,000 192-dimensional data points, which are extracted from the LDC SWITCHBOARD-1 collection. This collection consists of 2,400 two-sided telephone conversations among 543 speakers from different areas in the United States. Color is a 32-dimensional dataset[6] with more than 68,000 points, where each point describes the color histogram of an image in the Corel collection. For our purpose, we randomly choose 10,000 points from the audio and color datasets respectively to form the audio10k and color10k datasets, with which the experiments are performed.

4.3 Data preprocessing

The purpose of this experimental study is not to evaluate the performance of some NN search algorithm, but to verify the validity of our theoretical analysis. To this end, all datasets are preprocessed in the following way.

For each dataset, all 2-combinations, i.e., all possible pairs of points, are enumerated out of the dataset, and the distance of each pair is calculated. Then, all pairs of points are arranged in ascending order of their distances. Let d_{min} and d_{max} denote the minimum distances computed, respectively. In order to ensure comprehensive verification, starting at $r = d_{min}$, we choose a series of radii with spacing 0.1, i.e., $r = d_{min} + 0.1, d_{min} + 0.2, \ldots$, within the range of $[d_{min}, d_{max}]$. For each radius r, the pairs of points whose distances fall in the range of $[r - 0.01, r + 0.01]$ are selected to form the set S_r. Here we do not require the distance to be exactly equal to r because it is hard to find enough pairs of points satisfying this criterion, even for large datasets. In the empirical study to be discussed shortly, we experiment with a large number of S_rs produced from the several datasets to confirm our arguments given in earlier sections.

[4]http://yann.lecun.com/exdb/minist/

[5]http://www.cs.princeton.edu/cass/audio.tar.gz

[6]http://kdd.ics.uci.edu/databases/CorelFeatures/

4.4 Experimental Results and Analysis

4.4.1 Experiment 1

In this set of experiments, we first choose 10,000 hash functions uniformly at random from $\mathcal{H}_E$, and then choose 100 random pairs of points, denoted by $p_{j,r}$, $j = 1, \ldots, 100$, from each S_r[7]. For each $p_{j,r}$, the hash-function-based collision ratio, namely, the proportion of these 10,000 hash functions over which $p_{j,r}$ collides is calculated. Let $CR_r^{p_j}$ denote the collision ratio of $p_{j,r}$. The average hash-function-based collision ratio, denoted by CR_r^p, is the mean of all $CR_r^{p_j}$, namely, $\sum_{j=1}^{100} CR_r^{p_j}/100$.

Since all experimental results with respect to different datasets are very similar, we only report the collision ratios of 10 pairs of points chosen from the mnist1k dataset in Table 1. The probabilities of collision ($Pr_{\mathcal{H}_E}[r]$), which are calculated according to Equation (14), and the average collision ratios are also listed.

As shown in Table 1, the collision ratios of these 10 pairs of points are very close at each different radius value, and the average collision ratio is considerably close to the probability of collision, which confirms our interpretation of $Pr_{\mathcal{H}_E}[r]$, i.e., $Pr_{\mathcal{H}_E}[r]$ does not rely on any particular pair of data points; it is only determined by the nature of the given LSH family.

Table 1: Samples of the hash-function-based collision ratios

	r=0.4	r=0.5	r=0.6	r=0.7	r= 0.8
$CR_r^{p_1}$	0.6953	0.6216	0.5550	0.4896	0.4519
$CR_r^{p_2}$	0.6868	0.6139	0.5539	0.5019	0.4490
$CR_r^{p_3}$	0.6911	0.6145	0.5519	0.5025	0.4436
$CR_r^{p_4}$	0.6805	0.6126	0.5401	0.4795	0.4445
$CR_r^{p_5}$	0.6880	0.6126	0.5481	0.4925	0.4431
$CR_r^{p_6}$	0.6844	0.6064	0.5392	0.4940	0.4391
$CR_r^{p_7}$	0.6771	0.6084	0.5451	0.4953	0.4444
$CR_r^{p_8}$	0.6739	0.6007	0.5403	0.4907	0.4368
$CR_r^{p_9}$	0.6801	0.6024	0.5311	0.4867	0.4413
$CR_r^{p_{10}}$	0.6748	0.5981	0.5354	0.4813	0.4304
CR_r^p	0.6836	0.6092	0.5442	0.4889	0.4437
$Pr_{\mathcal{H}_E}[r]$	0.6825	0.6095	0.5451	0.4897	0.4426

Next, 100 hash functions h_i, $i = 1, \ldots, 100$, are randomly chosen from $\mathcal{H}_E$, and for each S_r in the mnist1k dataset, the dataset-based collision ratio, denoted by $CR_r^{h_i}$, is computed. $CR_r^{h_i}$ is defined as the proportion of all $p_{j,r}$ in S_r which

[7]For S_r with less than 100 pairs of points, all pairs of points in it are selected.

Table 2: Samples of the dataset-based collision ratios

	r=0.4	r=0.5	r=0.6	r=0.7	r= 0.8
$CR_r^{h_1}$	0.8627	0.8203	0.5619	0.5066	0.4715
$CR_r^{h_2}$	0.6078	0.4922	0.4834	0.4407	0.3752
$CR_r^{h_3}$	0.6078	0.6953	0.5015	0.4515	0.4009
$CR_r^{h_4}$	0.6473	0.6172	0.5257	0.4539	0.4819
$CR_r^{h_5}$	0.7451	0.5781	0.4532	0.4204	0.4699
$CR_r^{h_6}$	0.7843	0.4766	0.5982	0.5353	0.4105
$CR_r^{h_7}$	0.6275	0.7422	0.4743	0.5101	0.4093
$CR_r^{h_8}$	0.6863	0.5781	0.5680	0.4958	0.4434
$CR_r^{h_9}$	0.9216	0.7031	0.4924	0.4240	0.4093
$CR_r^{h_{10}}$	0.4902	0.4453	0.5408	0.4826	0.4289
CR_r^p	0.6836	0.6092	0.5442	0.4889	0.4437
stdev of CR_r^h	0.1208	0.0725	0.0334	0.0150	0.0070
$\|S_r\|$	52	129	332	836	2493

collide over h_i. Please note that the way in which $CR_r^{h_i}$ is calculated is totally different from $CR_r^{p_j}$. The average dataset-based collision ratio, denoted by CR_r^h, is defined as the mean of all $CR_r^{h_i}$, namely, $\sum_{i=1}^{100} CR_r^{h_i}/100$.

The collision ratios of 10 representative hash functions are illustrated in Table 2. The average hash-function-based collision ratios (CR_r^p), the standard deviations of $CR_r^{h_i}$ and the number of pairs of points in S_r are also listed. From Table 2 one can see significant deviation of $CR_r^{h_i}$ from CR_r^p, which justifies our claim in Observation 1 as $CR_r^{h_i}$ and CR_r^p are good approximations of $Pr_{h_i}[r]$ and $Pr_{\mathcal{H}_E}[r]$, respectively.

Another interesting fact that we can observe is that the standard deviation of $CR_r^{h_i}$ decreases as radius r becomes larger, which is mainly caused by the varying sizes of $S_r, r = 0.4, \ldots, 0.8$. The impact of the cardinality of dataset on $Pr_{h_i}[r]$ will be evaluated further in Experiment 2 and Experiment 3.

Recall that there is an intrinsic relation between $Pr_{h_i}[r]$ and $Pr_{\mathcal{H}}[r]$ as proved in Theorem 1. In order to support this claim, we plot the minimum and maximum dataset-based collision ratios, namely, $min(CR_r^{h_i}))$ and $max(CR_r^{h_i})$ $\forall i = 1, \ldots, 100$, CR_r^h and CR_r^p under different radii in Figure 5. As we can see from this figure, CR_r^h and CR_r^p almost coincide for all radii, which experimentally validates Theorem 1. Furthermore, the significant gap between the maximum and minimum $CR_r^{h_i}$ bears out Observation 1 again.

4.4.2 Experiment 2

Recall that, in Section 3.5, we argue that the actual failure probability will fluctuate around the value specified by users. In this set of experiments, we will experimentally confirm this argument with the dataset mnist1k.

Given a radius r, $r = d_{min}+0.1, d_{min}+0.2, \ldots$, for each k, $k = 1, 2, \ldots$, the corresponding L value is calculated using $L = log_{1-Pr_{\mathcal{H}}[r]^k}\delta$, where $Pr_{\mathcal{H}}[r]$ is computed using Equation (14) and δ is the pre-specified failure rate by users.

The measure, denoted by $SR_r^{k,L}$, is the success ratio, namely, the proportion of all pairs of points in S_r that collide over at least one $g_j, j = 1, 2, \ldots, L$. Please note that small $SR_r^{k,L}$ means high failure probability and vice versa.

For each k, we repeat the experiment 10 times and collect the minimum, maximum and average $SR_r^{k,L}$, respectively. Due to space limitation, only some representative results are reported in Figure 1, where the minimum, maximum and average $SR_r^{k,L}$ as a function of k are plotted, with $r = 0.4$ and $\delta = 0.1$. The values of $1-(1-Pr_{\mathcal{H}}[r]^k)^L$ (the expected $SR_r^{k,L}$) are also depicted since they may not be exactly equal to $1-\delta$ due to the rounding of L. In other words, the expected success ratio is $1-(1-Pr_{\mathcal{H}}[r]^k)^L$ rather than $1-\delta$ actually. As we can see, there exists a significant difference between the minimum and maximum $SR_r^{k,L}$ for all k values. While $SR_r^{k,L}$ fluctuates around $1-(1-Pr_{\mathcal{H}}[r]^k)^L$, its average, however, is considerably close to the expected success ratio, which justifies our claim in Theorem 2. A probabilistic interpretation of the above results is that the success ratio, with which LSH_C reports rNNs of a query, is a probabilistic event itself!

4.4.3 Experiment 3

In this set of experiments, we repeat the last experiments with more real datasets to show the impact of dataset characteristics on the performance of LSH_C.

Specifically, for each dataset (mnist10k, audio10k and color10k), we first form a subset by randomly choosing 10% out of this dataset, and then put another 10% out of the remaining points into the subset, and so on and so forth until the original dataset is exhausted. For each subset generated, we repeat Experiment 1 and collect the same statistics as those discussed in Section 4.4.1. Since the experimental results exhibit similar trends with respect to different radii, we only report some representative statistics here.

To demonstrate how $Pr_h[r]$ is affected by the characteristics of a dataset, the standard deviations of $CR_r^{h_i}$ are collected. Due to space limitation, only the result for minist10K is plotted in Figure 6. As we can see, the standard deviation at the same radius drops gradually as the proportion of points grows from 10% to 100%, which means that the larger the cardinality of a dataset, the closer $Pr_{h_i}[r]$ is to $Pr_{\mathcal{H}}[r]$. In addition, when the proportion of data points is fixed, the standard deviations at different radii are not identical, and vary in different trends for these three datasets. For example, when the proportion of points is set to 50% and $r = 1.2$, the standard deviation is equal to 3.52×10^{-4} for mnist10k, whereas for color10k, it is boosted to 2.47×10^{-2} drastically. This is because (1) for the same dataset, the numbers of pairs of points in different S_rs are not identical; (2) data distributions of these three datasets are different, which indicates that data distribution has an important impact on $Pr_{h_i}[r]$ as well.

In sum, for the same dataset, the more (less) pairs of points in an S_r, the smaller (larger) the variation in $Pr_{h_i}[r]$. When the standard deviation becomes very small, i.e., less than 10^{-3}, most $Pr_{h_i}[r]$ will be very close (even almost identical) to $Pr_{\mathcal{H}}[r]$. In that case, the actual performance of LSH_C will be considerably close to what is predicted by the existing analysis method (although it is not correct). These experimental results explain why the LSH algorithms work very well in practice, which we believe is one main cause for why such a gap exists for years.

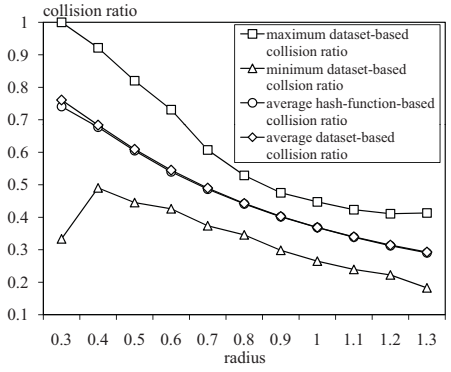

Figure 5: $min(CR_r^{h_i})$, $max(CR_r^{h_i})$, CR_r^h **and** CR_r^p **vs.** r **(mnist1k)**

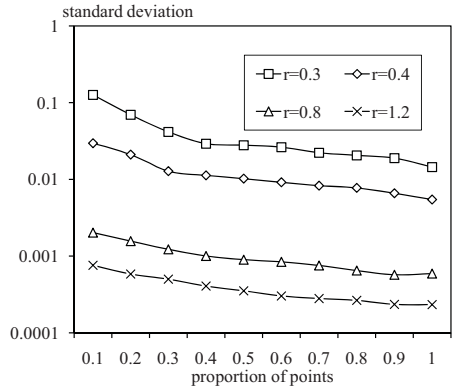

Figure 6: standard deviation vs. proportion of points (mnist10k)

5. CONCLUSION AND FUTURE WORK

In this paper, we observe that a fundamental gap exists between the LSH theory and the method for analyzing the LSH algorithm performance, and develop a novel analysis model to fill the gap between the LSH theory and practice. We believe that there are many interesting directions worthy of further investigation. A few quick examples include how to determine optimal parameters for the LSH algorithm under the proposed model, and is it possible to devise novel randomized (approximate) algorithms for NN search with provable constant failure probability.

Acknowledgements

The work reported in this paper is partially supported by NSFC under grant number 60903160, NSF of Shanghai under grant number 13ZR1400800 and NSC grant 101-2221-E-006-219.

6. REFERENCES

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *STOC*, pages 557–563, 2006.

[2] A. Andoni and P. Indyk. E2lsh manual. In *http://web.mit.edu/andoni/www/LSH*, 2004.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *SODA*, pages 573–582, 1994.

[5] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[6] J. L. Bentley. K-d trees for semidynamic point sets. In *SoCG*, pages 187–197, 1990.

[7] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, 1998.

[8] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[9] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.

[10] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD*, pages 301–312, 2003.

[11] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.

[12] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[13] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[14] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[15] J. He, S. Kumar, and S.-F. Chang. On the difficulty of nearest neighbor search. In *ICML*, pages 1127–1134, New York, NY, USA, July 2012.

[16] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[17] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *ACM Multimedia*, pages 869–876, 2004.

[18] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[19] M. Slaney, Y. Lifshits, and J. He. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE*, 100(9):2604–2623, 2012.

[20] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[21] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205. Morgan Kaufmann, 1998.