PROBABILISTIC HASHING TECHNIQUES FOR BIG DATA

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Anshumali Shrivastava

August 2015

PROBABILISTIC HASHING TECHNIQUES FOR BIG DATA

Anshumali Shrivastava, Ph.D.

Cornell University 2015

We investigate probabilistic hashing techniques for addressing computational and memory challenges in large scale machine learning and data mining systems. In this thesis, we show that the traditional idea of hashing goes far beyond near-neighbor search and there are some striking new possibilities. We show that hashing can improve state of the art large scale learning algorithms, and it goes beyond the conventional notions of pairwise similarities. Despite being a very well studied topic in literature, we found several opportunities for fundamentally improving some of the well know textbook hashing algorithms. In particular, we show that the traditional way of computing minwise hashes is unnecessarily expensive and without loosing anything we can achieve an order of magnitude speedup. We also found that for cosine similarity search there is a better scheme than SimHash.

In the end, we show that the existing locality sensitive hashing framework itself is very restrictive, and we cannot have efficient algorithms for some important measures like inner products which are ubiquitous in machine learning. We propose asymmetric locality sensitive hashing (ALSH), an extended framework, where we show provable and practical efficient algorithms for Maximum Inner Product Search (MIPS). Having such an efficient solutions to MIPS directly scales up many popular machine learning algorithms.

We believe that this thesis provides significant improvements to some of the heavily used subroutines in big-data systems, which we hope will be adopted.

BIOGRAPHICAL SKETCH

VITA

2014 Best Paper Award in Neural Information Processing System (NIPS) Conference.

2014 Best Paper Award in IEEE/ACM International Conference on Advances in Social Network Analysis and Mining (ASONAM)

2008 Integrated B.Sc.(H) and M.Sc., Mathematics and Computing

2008 Indian Institute of Technology (IIT) Kharagpur Silver Medal

2007 J.C. Gosh Memorial Endowment Prize from IIT Kharagpur

2003 DMMS Scholarship from IIT Kharagpur

PUBLICATIONS

Shrivastava, A. and Li, P. *Improved Asymmetric Locality Sensitive Hashing (ALSH) for Maximum Inner Product Search (MIPS)*

International Conference on Uncertainty in Artificial Intelligence *UAI* 2015

Shrivastava, A. and Li, P. *Asymmetric Minwise Hashing for Indexing Binary Inner Products and Set Containment*

International World Wide Web Conference *WWW* 2015

Shrivastava, A. and Li, P. *Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS).*

Annual Conference on Neural Information Processing Systems *NIPS* 2014

Shrivastava, A. and Li, P. *A New Space for Comparing Graphs.*

International Conference on Advances in Social Network Analysis and Mining IEEE/ACM ASONAM 2014

Shrivastava, A. and Li, P. *Improved Densification of One Permutation Hashing.*

International Conference on Uncertainty in Artificial Intelligence *UAI* 2014 .

Shrivastava, A. and Li, P. *Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search.*

International Conference on Machine Learning *ICML* 2014.

Li, P., Mitzenmacher, M., and Shrivastava, A. *Codings for Random Projections.*

International Conference on Machine Learning *ICML* 2014.

Shrivastava, A. and Li, P. *In Defense of Minhash over Simhash.*

International Conference on Artificial Intelligence and Statistics *AISTATS* 2014.

Shrivastava, A. and Li, P. *Beyond Pairwise: Provably Fast Algorithms for Approximate k-Way Similarity Search.*

Annual Conference on Neural Information Processing Systems *NIPS* 2013.

Shrivastava, A. and Li, P. *Fast Near Neighbor Search in High-Dimensional Binary Data.*

European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases ECML/PKDD 2012.

Sun, X., Shrivastava, A. and Li, P. *Fast multi-task learning for query spelling correction.*

International Conference on Information and Knowledge Management *CIKM* 2012.

Li, P., Shrivastava, A. and Konig, A. C. *GPU-based minwise hashing.*

International World Wide Web Conference *WWW* (Companion Volume)

2012.

Sun, X., Shrivastava, A. and Li, P. *Query spelling correction using multi-task learning*.

International World Wide Web Conference *WWW* (Companion Volume) 2012.

Li, P., Shrivastava, A., Moore, J. L. and Konig, A. C. *Hashing Algorithms for Large-Scale Learning*.

Annual Conference on Neural Information Processing Systems *NIPS* 2011.

To Nutan and Abhaishanker Srivastava.

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Ping Li for his continuous support throughout my PhD. Ping introduced me to the beautiful area of hashing and further provided constant motivation which kept me going. I am very thankful to him for his patience during my early years. His constant encouragement, ever after series of rejections, never let me down. I was indeed fortunate to have Ping as my advisor.

I would like to thank my committee members Johannes Gehrke and Gennady Samorodnitsky for their invaluable suggestions and constructive feedbacks on this work. In addition, I would like to thank Thorsten Joachims for multiple discussions and suggestions on related topics. All of them were also very supportive and helpful in making me decide my future career plans.

I had a pleasure of collaborating with and learning from Misha Bilenko and Christian Konig as a part of my internship at Microsoft Research. I would like to thank them for an amazing internship experience.

I had a fortune of interacting with faculties at the department of computer science at Cornell University, which has inspired me in many ways. The faculties, colleagues and staffs were extremely helpful and supportive throughout the course of my PhD. It was indeed a pleasure to be a part of such an exciting community.

Part of the work in this thesis was completed at Rutgers University. I am thankful to the wonderful faculties, staffs and colleagues at Rutgers University for their help. I am grateful to the statistics and computer science department at Rutgers University for providing me space, funding and computing resources to support my research.

My family Abhaishanker Srivastava, Nutan Srivastava and Vishal Srivastava made my PhD journey quite smooth with their continuous support and encouragements. I am greatly indebted to them.

My PhD life, both personally and professionally, would not have been the same without Tushar Gulati, Vikram Thapar, Javad Ahmed Raheel, Shouvik Chatterjee, Sudip Roy, Amit Sharma, Siddharth Karkare, Hema Koppula, Arunima Singh, Ritu Sahore, Poornima Gadamsetty, Karthik Raman, Abhishiek Anand, Martin Slawski, James DeBoer and Marian Thorpe. I would also like to thank all my PhD colleagues and friends from Cornell University for making grad school a wonderful experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Modern applications are constantly dealing with datasets at tera-byte scale and the anticipation is that very soon it will reach peta-byte levels. Being able to utilize this enormous amounts of data is the key factor behind recent breakthroughs in vision, speech and natural language, search, social networks, etc. It is beyond doubt that the unprecedented opportunity provided by learning algorithms in improving our life is bottlenecked by our ability to exploit large amounts of data. As we continuously collect more data scaling up existing machine learning and data mining algorithms becomes increasingly challenging. Simple data mining operations such as search, learning, clustering, etc., become hard at this scale. A brute force linear scan of the whole data for search is impractical because of latency. Even with all the parallelism, it is infeasible to run any algorithm quadratic in the size of the data.

Hashing algorithms are becoming popular for modern big data systems. These algorithms trade a very small amount of certainty, which typically is insignificant for most purposes, with a huge, often exponential gains, in the computations and the memory. In this dissertation we present several fundamental insights and improvements in the classical textbook hashing algorithms in the literature. Some of these frequently encountered problems, which can be solved using the techniques proposed in this dissertation were known to be hard, and in fact cannot be accomplished otherwise. We demonstrate the direct consequences of the obtained fundamental improvements in many real applications which include search, query matching, classification and regression, collaborative filtering for large scale recommendations, and Google sets.

1.1 Results

We introduce the following techniques and paradigms, which address the scaling challenges common in large scale learning systems. We improve few decade-old textbook hashing algorithms and also provide solutions to some known problems in literature. We summarize the contributions and their significance in this section. We discuss them, in details, in the respective chapters.

1) Asymmetric Locality Sensitive Hashing (ALSH) Paradigm [80, 85, 86]: Maximum inner product search (MIPS) occurs as a subroutine in numerous machine learning algorithms [80]. Finding hashing-based sub-linear algorithms for MIPS was a known hard problem and it is not difficult to show that it is in fact impossible in the classical hashing paradigm. We introduced the ALSH paradigm, a strict extension of the existing framework, where we can carefully apply asymmetric transformations and eventually construct practical and provable sub-linear algorithms for MIPS.

Fundamental Contributions: A sub-linear solution to MIPS will directly translate into efficient algorithms for many machine learning subroutines which include training massive deep networks, object detection, fast classification, active learning for optimization, orthogonal matching pursuit, collaborative filtering, cutting plane methods, etc. Apart from solving MIPS, asymmetric transformations add a new dimension, yet to be fully explored, to the existing hashing literature. With asymmetric transformations, we can reduce the search problem in one similarity measure to search problem in another similarity measure which is known to be efficient. This allows us to exploit good hash functions in one domain for efficiently solving search problem in another domain which may not have any known efficient algorithm. With this paradigm, we also im-

prove classical minwise hashing algorithm for retrieving set containment [85].

Demonstrated Practical Impact: As a direct consequence, we show huge computational savings, compared to the state-of-the-art algorithms, in the task of top-k item recommendations with collaborative filtering approaches [80, 83]. We also demonstrate computational improvements in searching with set containment [85]. Set containment is a similarity measure commonly used for record matching and plagiarism detection in practice.

2) A new randomized technique called "Densification" [81, 82]: Large scale data processing applications require computing several minwise hashes of the data matrix as one of the primitive operations. This is computationally very expensive. We found a new way of generating multiple minwise hashes by first binning the data, computing local minwise like hashes in each bin, and finally densifying the bins via random rotation. As a result, we can generate thousands (even more) different hashes, with the required property, for the same cost as one vanilla minwise hash. This procedure leads to algorithmic query time improvement over classical widely adopted search algorithms. For very sparse data, adding coin flips during the densification step provably improves the variance due to extra randomization [82].

Fundamental Contributions: Sparsity in hashes makes indexing for sub-linear search impossible. The "densification" scheme is of independent interest in itself and is directly applicable for densifying any general sparse hashes where sparsity is not desirable. In addition to algorithmic query time improvement, this densification procedure naturally leads to an order of magnitude faster kernel features for the resemblance kernel [65].

Demonstrated Practical Impact: In the task of similarity search [81], we obtain

1000 folds savings in the hashing time, for generating 1000 hashes, without losing any accuracy. The densification procedure makes training and testing time of SVMs and logistic regression, with resemblance kernel features, around 100 times faster

3) *MinHash* or *SimHash* ? A paradigm for comparing hash functions across similarity measures [84, 78]: In the existing literature, it was taken for granted that the two popular hashing schemes, *MinHash* and *SimHash*, are theoretically incomparable and the choice between them is decided based on whether the desired notion of similarity is cosine similarity or resemblance. We found the rather surprising fact that for cosine similarity search with sparse binary data, *MinHash* is provably superior to the usual *SimHash* which was the only known hash function for this task in the literature. We also provide the theoretical quantification of the advantages.

Fundamental Contributions: *MinHash* and *SimHash* are widely popular hash functions in both theory as well as in industrial practice. This work provides precise recommendations for practitioners about the choice between them. Apart from theoretical comparisons between *MinHash* and *SimHash*, this work also provides conditions under which hash functions for different similarity measures can be compared. In particular, it turns out that if two similarity measures are distortions of each other then we can compare the corresponding locality sensitive hash functions associated with them. This gives a generic paradigm to compare two hash functions meant for different similarity measures. This is the first formal comparison of two locality sensitive hash function meant for different similarity measures.

Demonstrated Practical Impact: For searching with cosine similarity, we

demonstrate that *MinHash* leads to significant computational savings compared to SimHash which is very valuable for practitioners.

4) Hashing framework for k-way similarities [79]: There are notions of similarity which are not pairwise, for instance the k-way resemblance which is a simultaneous similarity function between k elements. Such similarity measures naturally model many real scenarios like group recommendations, clustering, etc. K-way similarity measures lead to a blow up in the computation complexity because of the possible number of combinations. It turns out that for searching with the k-way resemblance, we can do a lot better than brute force scan.

Fundamental Contributions: We formalized the notions of search with k-way similarity measures and found provably fast hashing algorithms to solve them. This is the first non-trivial fast hashing algorithm for k-way similarity search in the literature. In addition, any probabilistic combination of k-way resemblance naturally admits such fast algorithms, leading to a new class of k-way similarity measures which admits efficient solutions.

Demonstrated Practical Impact: We show that k-way resemblance improves conventional similarity search, and it naturally models the "Google sets" application, leading to better performance compared to other methods.

5) Hash-based kernel features for large scale classification and regression [65, 61, 63, 64, 58, 78, 59] The computational complexity of kernel-based classifiers is quadratic in the size of the dataset because they require computation of the costly kernel matrix. We show how to convert a locality sensitive hashing scheme into efficient features for kernel learning in linear time. To be able to

compute these features we need the range of the hash function to be small. We propose b-bit minwise hashing [65, 61, 63, 64, 78] and efficient quantizations of random projections [58, 59] for reducing the range space. These two quantizations lead to efficient kernel features for learning with kernels in linear time.

Fundamental Contributions: Finding the right hash quantization naturally translates into the fundamental problem of information theory, which deals with the amount of information content in the hash codes. The new quantizations for random projections theoretically improve the classical known quantization for the euclidean distance. The b-bit minwise hashing quantizations provably improve over the popular random projections based hash kernels.

Demonstrated Practical Impact: Experimental results show that such quantizations are very effective and outperform other state-of-the-art hash kernels including the hashing scheme of VW (Vowpal Wabbit) [65, 61]. We obtain huge reduction in training and testing time with SVMs and logistic regression on benchmark datasets. The new quantizations can also be used for indexing, leading to improvements over the state-of-the-art hashing algorithm for L_2 distance. [65, 61].

1.2 Organization

The dissertation is organized as follows. We start by reviewing the existing notions of Locality Sensitive Hashing (LSH) for near-neighbor search problem and related work in Chapter 2. LSH schemes and related algorithms will be the common background needed for the rest of this thesis. In Chapter 3 we show how the same hashing framework, originally meant for sub-linear search, can be converted into kernel features leading to fast linear time algorithms for

learning with kernels. In Chapter 4 we show a very counter intuitive result that for binary datasets MinHash is a superior LSH for cosine similarity compared to SimHash, which goes against the traditional belief of always using SimHash for cosine similarity. We later show in Chapter 5 that MinHash based algorithms can be extended for beyond pairwise similarities and we can solve a class of k-way similarity search problem very efficiently using MinHash.

After showing many striking and new advantages of MinHash, we argue in Chapter 6 that the decade-old standard scheme for computing MinHash is very inefficient. We provide a one pass and one permutation solution leading to an order of magnitude faster hashing scheme. Finally in Chapter 7 we show that the existing LSH framework itself is very restrictive and there are many similarities, e.g. inner products, which cannot be hashed. We extend the framework to allow asymmetry and derive the first sub-linear provable and practical algorithm for maximum inner product search (MIPS). MIPS is ubiquitous subroutine in machine learning application. We later show the power of the new framework by deriving many new improved hashing algorithms which provably improve the state-of-the-art.

Proofs of theorems are deferred to appendix except at few places, for instance Chapter 5, where the proofs are constructive and convey the main idea.

CHAPTER 2

BACKGROUND: CLASSICAL LOCALITY SENSITIVE HASHING (LSH)

In this chapter, we briefly review Locality Sensitive Hashing (LSH) families and their applications in sub-linear time near neighbor search. We refer readers to [36] for detailed description of existing works in LSH literature. This thesis provides several new fundamental results and improvements in the LSH domain. The concepts described in this chapter will be heavily referred throughout the course of this thesis.

2.1 The Near-Neighbor Search Problem

Near-neighbor search or similarity search is one of the fundamental problems in computer science. In this problem, we are typically given a giant collection $C \subset \mathbb{R}^D$ and a query $q \in \mathbb{R}^D$. The task is to search for point $x \in C$ which minimizes (or maximizes) the distance (or similarity) with the query q, i.e., we are interested in

$$x = \arg\min_{x \in C} Dist(x, q), \tag{2.1}$$

where $Dist$ is the distance metric of interest. To work with similarities we can change to $\arg\max$ and replace distance by the appropriate similarity. Near-neighbor search algorithms have been one of the basic building blocks in numerous applications including search, databases, learning, recommendation systems, computer vision, etc. It is one of the most heavily used subroutine in almost all applications dealing with data.

With the explosion of data in recent times, typically, the size of the collection C blows up. For instance, documents (or items) over the web. A linear scan

is very expensive at this scale. Moreover, querying is a frequent and latency critical operation. Therefore, it is infeasible to scan the whole collection for answering every query. To mitigate this problem there are plenty of literatures that create smart data structures which can answer near neighbor queries efficiently. These data structures require one time costly preprocessing of the collection C during their construction. After this pre-processing querying is very fast and is sub-linear in the size of the collection.

Early techniques for sub linear time near-neighbor search were based on deterministic space partitioning methods like trees and Voronoi cells [26]. It was later found that techniques based on space partitioning suffer from the curse of dimensionality and the runtime guarantees typically scale exponentially in the dimensions. For example, it was shown in [89] (both empirically and theoretically) that all current techniques (based on space partitioning) degrade to linear search, even for dimensions as small as 10 or 20.

Locality Sensitive Hashing (LSH) [42] based randomized techniques are common and successful in industrial practice for efficiently solving near-neighbor search. Unlike space partitioning techniques, both the running time as well as the accuracy guarantees of LSH based algorithms are independent of the dimensionality of the data. This makes LSH suitable for large scale processing systems dealing with ultra-high dimensional data which are common these days. Furthermore, LSH based schemes are massively parallelizable, which makes them ideal for modern "Big" datasets. The prime focus of this thesis will be on hashing based algorithms for large scale search and learning, which do not suffer from the curse of dimensionality and works well in practice.

2.2 Definitions

Finding efficient (sub-linear time) algorithms for exact near neighbor search turned out to be a disappointment with the massive dimensionality of current datasets [31, 89]. Approximate versions of the near-neighbor search problem were proposed [42, 36] to break the linear query time bottleneck. One widely adopted formalism is the c-approximate near neighbor or c-NN.

Definition 1 *(c-**Approximate Near Neighbor or** c-**NN**). Given a set of points in a d-dimensional space $\mathbb{R}^d$, and parameters $S_0 > 0$, $\delta > 0$, construct a data structure which, given any query point q, does the following with probability $1 - \delta$: if there exist an S_0-near neighbor of q in P, it reports some cS_0-near neighbor of q in P.*

The usual notion of S_0-near neighbor is in terms of the distance function. While dealing with similarities, we can equivalently define S_0-near neighbor of point q as a point p with $Sim(q, p) \geq S_0$, where Sim is the desired similarity.

2.2.1 Locality Sensitive Hashing (LSH)

A popular technique for c-NN, uses the underlying theory of *Locality Sensitive Hashing* (LSH) [42]. LSH is a family of functions, with the property that similar input objects in the domain of these functions have a higher probability of colliding in the range space than non-similar ones. In formal terms, consider $\mathcal{H}$ a family of hash functions mapping $\mathbb{R}^D$ to some set $\mathcal{S}$.

Definition 2 Locality Sensitive Hashing (LSH) Family *A family $\mathcal{H}$ is called (S_0, cS_0, p_1, p_2)-sensitive if for any two point $x, y \in \mathbb{R}^d$ and h chosen uniformly from $\mathcal{H}$ satisfies the following:*

- *if $Sim(x, y) \geq S_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \geq p_1$*
- *if $Sim(x, y) \leq cS_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \leq p_2$*

For approximate nearest neighbor search typically, $p_1 > p_2$ and $c < 1$ is needed. Since we will be mostly dealing with similarities we need $c < 1$. To get distance analogy we can use the transformation $D(x, y) = 1 - Sim(x, y)$ with a requirement of $c > 1$. The definition of LSH family $\mathcal{H}$ is tightly linked with the similarity function of interest Sim. An LSH allows us to construct data structures that give provably efficient query time algorithms for c-NN problem.

One sufficient condition for a hash family $\mathcal{H}$ to be a locality sensitive hashing family is that the collision probability $Pr_{\mathcal{H}}(h(x) = h(y))$ is monotonically increasing function of the similarity Sim. We can easily see that if $Pr_{\mathcal{H}}(h(x) = h(y)) = f(Sim(x, y))$, where f is a monotonically increasing function, then the two required conditions in the Definition 2.2.1 is always satisfied for any S_0 and $c < 1$.

It was shown [42] that having an LSH family for a given similarity measure is sufficient for efficiency solving c-NN instances. Formally,

Fact 1 *Given a family of (S_0, cS_0, p_1, p_2) -sensitive hash functions, one can construct a data structure for c-NN with $O(n^{\rho} \log_{1/p_2} n)$ query time, where $\rho = \frac{\log p_1}{\log p_2}$.*

The quantity $\rho < 1$ measures the efficiency of a given LSH, the smaller the better. In theory, in the worst case, the number of points scanned by a given

LSH to find a c-approximate near neighbor is $O(n^{\rho})$ [42], which is dependent on ρ. Thus given two LSHs, for the same c-NN problem, the LSH with smaller value of ρ will achieve the same approximation guarantee and at the same time will have faster query time. LSH with lower value of ρ will report fewer points from the database as the potential near neighbors, which is desirable. It should be noted that the efficiency of an LSH scheme, the ρ value, is dependent on many things. It depends on the similarity threshold S_0 and the value of c the approximation factor.

2.2.2 Some Popular LSHs

Minwise Hashing (MinHash)

One of the most popular measure of similarity between web documents is **resemblance (or Jaccard Similarity)** $\mathcal{R}$ [12]. This similarity measure is only defined over sets which can be equivalently thought of as binary vectors over the universe, with non-zeros indicating the elements belonging to the given set. The resemblance similarity between two given sets $S_1, S_2 \subseteq \Omega = \{1, 2, ..., D\}$ is defined as

$$\mathcal{R} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{a}{f_1 + f_2 - a}, \tag{2.2}$$

where $f_1 = |S_1|$, $f_2 = |S_2|$, and $a = |S_1 \cup S_2|$.

Minwise hashing [13] is the LSH for resemblance similarity. The minwise hashing family applies a random permutation $\pi : \Omega \rightarrow \Omega$, on the given set S, and stores only the minimum value after the permutation mapping. Formally

MinHash is defined as:

$$h_{\pi}^{min}(S) = \min(\pi(S)). \tag{2.3}$$

Given sets S_1 and S_2, it can be shown by elementary probability argument that

$$Pr_{\pi}(h_{\pi}^{min}(S_1) = h_{\pi}^{min}(S_2)) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \mathcal{R}. \tag{2.4}$$

It follows from Equation 2.4 that minwise hashing is (S_0, cS_0, S_0, cS_0) sensitive family of hash function when the similarity function of interest is resemblance i.e $\mathcal{R}$. From Fact 1 it has efficiency

$$\rho = \frac{\log S_0}{\log cS_0}, \tag{2.5}$$

for approximate resemblance based search.

SimHash

SimHash is another popular LSH for the cosine similarity measure, which originates from the concept of **Signed Random Projections (SRP)** [18]. Given a vector x, SRP utilizes a random w vector with each component generated from i.i.d. normal, i.e., $w_i \sim N(0, 1)$, and only stores the sign of the projection. Formally SimHash is given by

$$h_w^{Sign}(x) = sign(w^T x). \tag{2.6}$$

It was shown in the seminal work [33] that collision under SRP satisfies the following equation:

$$Pr_w(h_w^{Sign}(x) = h_w^{Sign}(y)) = 1 - \frac{\theta}{\pi}, \tag{2.7}$$

where $\theta = cos^{-1}\left(\frac{x^T y}{\|x\|_2\|y\|_2}\right)$. The term $\frac{x^T y}{\|x\|_2\|y\|_2}$, is the cosine similarity.

For sets (or binary vectors) S_1 and S_2, the **Cosine Similarity** reduces to

$$\mathcal{S} = \frac{a}{\sqrt{f_1 f_2}}, \tag{2.8}$$

with $f_1 = |S_1|$, $f_2 = |S_2|$, and $a = |S_1 \cup S_2|$.

There is a variant of SimHash where, instead of $w_i \sim N(0,1)$, we choose each w_i independently as either +1 or -1 with probability $\frac{1}{2}$. It is known that this variant performs similar to the one with $w \sim N(0,1)$ [75, 39].

Since $1 - \frac{\theta}{\pi}$ is monotonic with respect to cosine similarity $\mathcal{S}$. Equation 2.7 implies that SimHash is a $\left(S_0, cS_0, \left(1 - \frac{cos^{-1}(S_0)}{\pi}\right), \left(1 - \frac{cos^{-1}(cS_0)}{\pi}\right)\right)$ sensitive hash function with efficiency

$$\rho = \frac{\log\left(1 - \frac{cos^{-1}(S_0)}{\pi}\right)}{\log\left(1 - \frac{cos^{-1}(cS_0)}{\pi}\right)}, \tag{2.9}$$

when the similarity function of interest is the cosine similarity $\mathcal{S}$.

LSH for L_2 Distance (L2-LSH)

[25] presented a novel LSH family for all L_p ($p \in (0, 2]$) distances. In particular, when $p = 2$, this scheme provides an LSH family for L_2 distances. Formally, given a fixed (real) number r, we choose a random vector a with each component generated from i.i.d. normal, i.e., $a_i \sim N(0,1)$, and a scalar b generated uniformly at random from $[0, r]$. The hash function is defined as:

$$h^{L2}_{a,b}(x) = \left\lfloor \frac{a^T x + b}{r} \right\rfloor \tag{2.10}$$

where $\lfloor\rfloor$ is the floor operation. The collision probability under this scheme can be shown to be

$$Pr(h^{L2}_{a,b}(x) = h^{L2}_{a,b}(y)) = F_r(d); \qquad F_r(d) = 1 - 2\Phi(-r/d) - \frac{2}{\sqrt{2\pi}(r/d)}\left(1 - e^{-(r/d)^2/2}\right) \tag{2.11}$$

where $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ is the cumulative density function (cdf) of standard normal distribution and $d = \|x - y\|_2$ is the Euclidean distance between the vectors x and y. This collision probability $F_r(d)$ is a monotonically decreasing function of the distance d and hence $h^{L2}_{a,b}$ is an LSH for L_2 distances. This scheme is also the part of LSH package [6]. Here r is a parameter. As argued previously, $\|x - y\|_2 = \sqrt{(\|x\|_2^2 + \|y\|_2^2 - 2x^T y)}$ is not monotonic in the inner product $x^T y$ unless the given data has a constant norm. Hence, $h^{L2}_{a,b}$ is not suitable for MIPS.

Except minwise hashing, most of the popular LSH schemes, such as SimHash and L2-LSH, are based on smart quantizations of random projections. These random projections based hashing schemes can be computed efficiently by utilizing faster variants of matrix multiplication, see [54, 4, 24] for more details. In chapter 6, we will show that with a the novel idea of "densification" we can even make minwise hashing faster.

2.3 Classical LSH Algorithm

To be able to answer queries in sub-linear time, the idea behind the LSH algorithm is to create hash tables from the given collection C which we are interested in searching. The hash table construction is one time costly operation which makes further querying efficient. The hash tables are generated using the locality sensitive hash family.

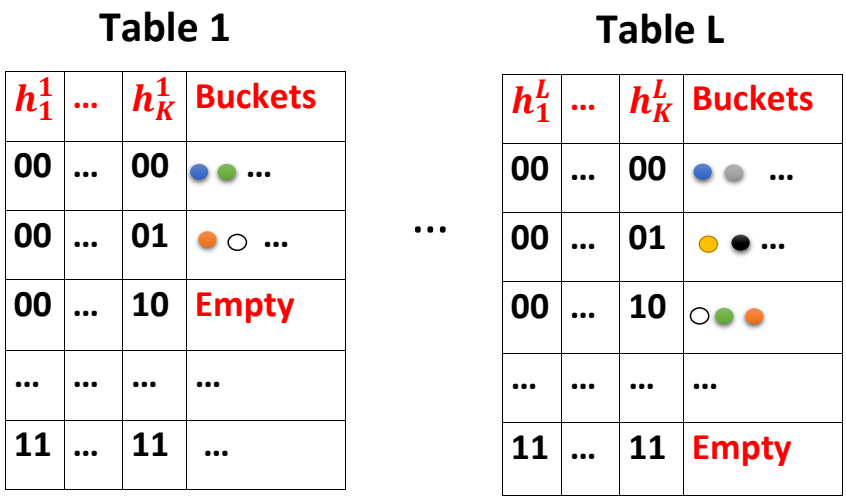

Figure 2.1: Illustration of hash table for the classical LSH algorithm

We assume that we have an access to the appropriate locality sensitive family $\mathcal{H}$ for the similarity search problem of interest. The classical (K, L) parameterized LSH algorithm, which we also refer to as *b*ucketing algorithm, works as follows. We generate L different meta-hash functions. Each of these meta-hash functions is formed by concatenating K sampled hash values from $\mathcal{H}$.

$$B_j(x) = [h_{j1}(x); h_{j2}(x); ...; h_{jK}(x)], \tag{2.12}$$

where $h_{ij}, i \in \{1, 2, ..., K\}$ and $j \in \{1, 2, ..., K\}$, are KL different hash evaluations of the locality sensitive hash function under consideration.

The algorithm works in two phases:

1) **Preprocessing Phase:** We construct L hash tables from the data by storing element $x \in C$, at location $B_j(x)$ in hash-table j (See Figure 2.1 for an illustration). We do not store the whole data vectors in the hash tables, which will be very inefficient from the memory perspective. We only store pointers or references to the vector.

2) **Query Phase:** Given a query Q, we report the union of all the points in the

buckets $B_j(Q)\ \forall\, j \in \{1, 2, ..., L\}$, where the union is over L hash tables. It should be noted that we do not scan all the elements in C, we only probe L different buckets, one from each hash tables.

The probability of finding a point x is a function of K, L and $Sim(x, Q)$ which is the similarity of the point with the query Q. Therefore, for a given instance of c-NN problem we need the appropriate values of K and L to obtain query time algorithmic guarantees given by fact 1. We refer readers to [6] for more details on choices of K and L. Optimal choice of K and L for a given c-NN instance leads to the guarantee given by Fact 1.

CHAPTER 3

NON-LINEAR LEARNING IN LINEAR TIME VIA HASHING

With the advent of the Internet, many machine learning applications are faced with very large and inherently high-dimensional datasets, resulting in challenges in scaling up training algorithms and storing the data. Especially in the context of search and machine translation, corpus sizes used in industrial practice have long exceeded the main memory capacity of single machine. For example, [88] discusses training sets with 10^{11} items and 10^9 distinct features, requiring novel algorithmic approaches and architectures. As a consequence, there has been a renewed emphasis on scaling up machine learning techniques.

In this chapter we show that locality sensitive hashing techniques which are used for efficient indexing, for sub-linear time near-neighbor search, also lead to very efficient and practical large scale learning algorithms. In particular, we show that a locality sensitive hashing family directly leads to kernel features for the associated similarity function. These kernel features can then be used for approximate learning with (non-linear) kernels in linear time, which otherwise is quadratic and prohibitively expensive.

3.1 Approximate Kernel Learning in Linear Time with Hashing

Linear algorithms such as linear SVM and logistic regression have become very powerful and extremely popular. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, 1\}$. The L_2-regularized linear SVM solves the following optimization problem):

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathbf{T}\mathbf{w} + C\sum_{i=1}^{n} \max\left\{1 - y_i\mathbf{w}^\mathbf{T}\mathbf{x_i},\ 0\right\}, \tag{3.1}$$

The above optimization learns a linear boundary between positive and negative examples as shown in Figure 3.1 on left hand side. Owing to a series of work, the optimization in Equation 3.1 can be solved very efficiently and can be accomplished in linear time. Representative software packages include SVM$^{\text{perf}}$ [45], Pegasos [77], Bottou's SGD SVM [10], and LIBLINEAR [29].

An equivalent formulation of optimization problem, given by Equation 3.1, in dual form is

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x_j^T x_k}) \tag{3.2}$$
$$\text{s.t} \quad 0 \leq \alpha_i \leq C \ \ \forall i \quad \text{and} \quad \sum_i \alpha_i y_i = 0$$

In many scenarios we are interested in learning a non-linear boundary, as shown on the right panel of Figure 3.1. This can be achieved using an appropriate positive semidefinite kernel function $k(x, y)$ and solving the optimization given by

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{k(x_j, x_k)}) \tag{3.3}$$
$$\text{s.t} \quad 0 \leq \alpha_i \leq C \ \ \forall i \quad \text{and} \quad \sum_i \alpha_i y_i = 0.$$

It relies on the observation that any positive definite function $k(x, y)$ defines a generalized inner product and the optimization algorithms can still be solved. The cost of this convenience is that algorithms access the data only through evaluations of $k(x, y)$, or through the kernel matrix. This incurs quadratic computational and storage costs.

Recently, the idea of kernel features [73] has become very popular for approximate learning with kernels in linear time. Observe that the only difference between Equation 3.2 and Equation 3.3 is that the inner product $x_j^T x_k$ is

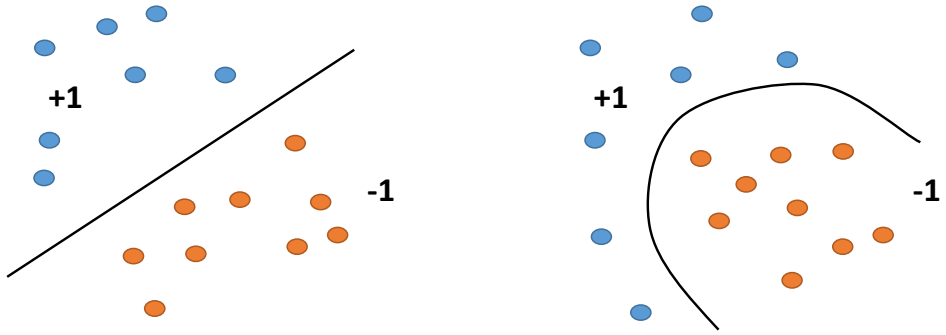

Figure 3.1: Linear SVMs (left) and Kernel SVMs (right) with non-linear boundary.

replaced by the kernel $k(x_j, x_k)$. This motivates an efficient approximate optimization using random kernel features. The idea is to generate a transformation $\phi : \mathbb{R}^D \mapsto \mathbb{R}^m$ s.t. $\phi(x)^T \phi(y) \simeq k(x, y)$. After having such a transformation ϕ, simply using traditional fast linear SVM, i.e. optimizing Equation 3.2, with $\{(\phi(x)_i, y_i)\}_{i=1}^n$ instead of $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ will approximately solve Equation 3.3. Thus we can learn a non-linear boundary by solving Equation 3.2 which is the dual of the optimization given by Equation 3.1 and therefore is solvable in linear time. Thus we avoid the infeasible quadratic cost. In next section, we show that LSH scheme for a given similarity (or kernel) automatically implies generation of such mapping (or kernel features) $\phi(x)$.

3.2 From LSH to Kernel Features

Let the range of the randomized hash functions be $h : x \to [0, D-1]$. The key observation is that given a locality sensitive hash family $\mathcal{H}$, such that $Pr_{h \in \mathcal{H}}(h(x) = h(y)) = k(x, y)$, then if we define $\phi(x)_i = 1\{h(x) = i\}$ for $i = \{0, 1, ..., D-1\}$. We can

easily see that

$$\mathbb{E}\left[\phi(x)^T\phi(y)\right] = \mathbb{E}\left[1\{h(x) = h(y)\}\right] = k(x, y).$$

Thus, we obtain kernel feature mapping ϕ, such that $\phi(x)^T\phi(y)$ in expectation is $k(x, y)$. Just like [73], we have to concatenate sufficiently many independent $\phi(x)$s to get the required concentrations of inner product around $k(x, y)$. In fact, the implications are stronger. Any randomized hashing scheme leads to a kernel given by $k(x, y) = Pr_{h\in\mathcal{H}}(h(x) = h(y))$. This kernel automatically leads to corresponding kernel features with features give by $\phi(x)_i = 1\{h(x) = i\}$. Formally, we can state

Theorem 1 *(***Hashability implies Learnability***) Given any randomized hashing family $\mathcal{H}$ and a function h drawn uniformly from $\mathcal{H}$. Then the function $f(x, y) = Pr_{h\in\mathcal{H}}(h(x) = h(y))$ is positive semidefinite and hence a valid kernel. Moreover, there exist a construction of randomized kernel features $\phi : \mathbb{R}^D \mapsto \mathbb{R}^m$ s.t.*

$$\mathbb{E}[\phi(x)^T\phi(y)] = f(x, y).$$

Proof: A function is positive semidefinite if it can be written as an inner product. Observe that the collision event can be written as an inner product,

$$1\{h(x) = h(y)\} = \sum_{t=0}^{R-1} 1\{h(x) = t\} \times 1\{h(y) = t\}, \tag{3.4}$$

where R is the range of the hash function h. Thus $1\{h(x) = h(y)\}$ is a kernel. The proof of $f(x, y)$ being positive semidefinite follows from taking the expectation. Note that expectation is a linear operator and hence preserve positive semidefinite property. The construction of random kernel features is immediate. □

The dimensionality of $\phi(x)$ blows up if the range of h is large. A simple fix is to randomly rehash $h(x)$ to a smaller range. Let us assume that we have a random 2-independent universal hashing scheme $H : x \to [0, B]$ where B is

sufficiently small. It is not difficult to show that if h is locality sensitive then so is $H(h(.))$, but with a smaller range.

We illustrate the above concept with b-bit minwise hashing which we later show outperforms the popular Vowpal Wabbit for large scale learning. There are few more recent works [49, 50] where different hashing schemes have been used as kernel features for efficient (linear time) large scale learning with some important newer kernels. Such derived kernel features lead to drastic improvements in classification accuracy.

Corollary 1 *Consider n sets S_1, ..., $S_n \subseteq \Omega = \{0, 1, ..., D-1\}$. Apply one permutation π to each set. Define $z_i = \min\{\pi(S_i)\}$ and $z_i^{(b)}$ the lowest b bits of z_i. The following three matrices are PD.*

1. *The* resemblance matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$*, whose (i, j)-th entry is the resemblance between set S_i and set S_j: $R_{ij} = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} = \frac{|S_i \cap S_j|}{|S_i|+|S_j|-|S_i \cap S_j|}$.*
2. *The* minwise hashing matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$*: $M_{ij} = 1\{z_i = z_j\}$.*
3. *The* b-bit minwise hashing matrix $\mathbf{M}^{(b)} \in \mathbb{R}^{n\times n}$*: $M_{ij}^{(b)} = 1\left\{z_i^{(b)} = z_j^{(b)}\right\}$.*

Consequently, consider k independent permutations and denote $\mathbf{M}_{(s)}^{(b)}$ the b-bit minwise hashing matrix generated by the s-th permutation. Then $\sum_{s=1}^{k} \mathbf{M}_{(s)}^{(b)}$ is also PD.

Proof: *The proof is similar to the proof of Theorem 1. A matrix $\mathbf{A}$ is PD if it can be written as an inner product $\mathbf{B^T B}$. Because*

$$M_{ij} = 1\{z_i = z_j\} = \sum_{t=0}^{D-1} 1\{z_i = t\} \times 1\{z_j = t\}, \tag{3.5}$$

M_{ij} is the inner product of two D-dim vectors. Thus, $\mathbf{M}$ is PD. Similarly, $\mathbf{M}^{(b)}$ is PD because $M_{ij}^{(b)} = \sum_{t=0}^{2^b-1} 1\{z_i^{(b)} = t\} \times 1\{z_j^{(b)} = t\}$. $\mathbf{R}$ is PD because $R_{ij} = \mathbf{Pr}\{M_{ij} = 1\} = E\left(M_{ij}\right)$ and M_{ij} is the (i, j)-th element of $\mathbf{M}$. Note that the expectation is a linear operation. □

3.3 Integrating b-Bit Minwise Hashing with (Linear) Learning Algorithms

We demonstrate the effectiveness of kernel features using b-bit hashing. To simplify the arguments, we use standard C-SVM (Equation 3.1) and provide all results for a wide range of C values. We assume that the best performance is achievable if we conduct cross-validations.

In our approach, we apply k random permutations on each feature vector $\mathbf{x}_i$ and store the lowest b bits of each hashed value. This way, we obtain a new dataset which can be stored using merely nbk bits. At run-time, we expand each new data point into a $2^b \times k$-length vector with exactly k 1's. We will see in Section 6 that we can achieve this in one pass over the feature vector and we can do everything with just one permutation.

For example, suppose $k = 3$ and the minwise hashed values are originally $\{12013, 25964, 20191\}$, whose binary digits are

$$\{010111011101101,\ 110010101101100, 100111011011111\}.$$

Consider $b = 2$. Then the binary digits are stored as $\{01, 00, 11\}$ (which corresponds to $\{1, 0, 3\}$ in decimals). At run-time, we need to expand them into a vector of length $2^b k = 12$, to be

$$\{0, 0, 1, 0,\ \ 0, 0, 0, 1,\ \ 1, 0, 0, 0\},$$

which will be the new feature vector fed to a solver such as LIBLINEAR. Clearly, this expansion is directly inspired by the proof that the b-bit minwise hashing matrix is PD in Corollary 1.

3.4 Experiments

Our experiment settings closely follow the work in [92]. They conducted experiments on three datasets, of which only the *webspam* dataset is public and reasonably high-dimensional ($n = 350000$, $D = 16609143$). Therefore, our experiments focus on *webspam*. Following [92], we randomly selected 20% of samples for testing and used the remaining 80% samples for training.

We chose LIBLINEAR [29] as the workhorse to demonstrate the effectiveness of our algorithm. All experiments were conducted on workstations with Xeon(R) CPU (W5590@3.33GHz) and 48GB RAM, under Windows 7 System. Thus, in our case, the original data (about 24GB in LIBSVM format) fit in memory. In applications when the data do not fit in memory, we expect that b-bit hashing will be even more substantially advantageous, because the hashed data are relatively very small. In fact, our experimental results will show that for this dataset, using $k = 200$ and $b = 8$ can achieve similar testing accuracies as using the original data. The effective storage for the reduced dataset (with 350K examples, using $k = 200$ and $b = 8$) would be merely about 70MB.

3.4.1 Evaluations with SVMs

We implemented a new resemblance kernel function and tried to use LIBSVM to train an SVM using the *webspam* dataset. The training time well exceeded 24 hours. Fortunately, using b-bit minswise hashing as kernel features to estimate the resemblance kernels provides a substantial improvement as expected from Section 3.2. For example, with $k = 150$, $b = 4$, and $C = 1$, the training time

is about 5185 seconds and the testing accuracy is quite close to the best results given by LIBLINEAR on the original *webspam* data.

There is an important tuning parameter C. To capture the best performance and ensure repeatability, we experimented with a wide range of C values (from 10^{-3} to 10^2) with fine spacings in $[0.1,\ 10]$.

We experimented with $k = 10$ to $k = 500$, and $b = 1$, 2, 4, 6, 8, 10, and 16. Figure 3.2 (average) and Figure 3.3 (std, standard deviation) provide the test accuracies. Figure 3.2 demonstrates that using $b \geq 8$ and $k \geq 200$ achieves similar test accuracies as using the original data. Since our method is randomized, we repeated every experiment 50 times. We report both the mean and std values. Figure 3.3 illustrates that the stds are very small, especially with $b \geq 4$. In other words, our algorithm produces stable predictions. For this dataset, the best performances were usually achieved at $C \geq 1$.

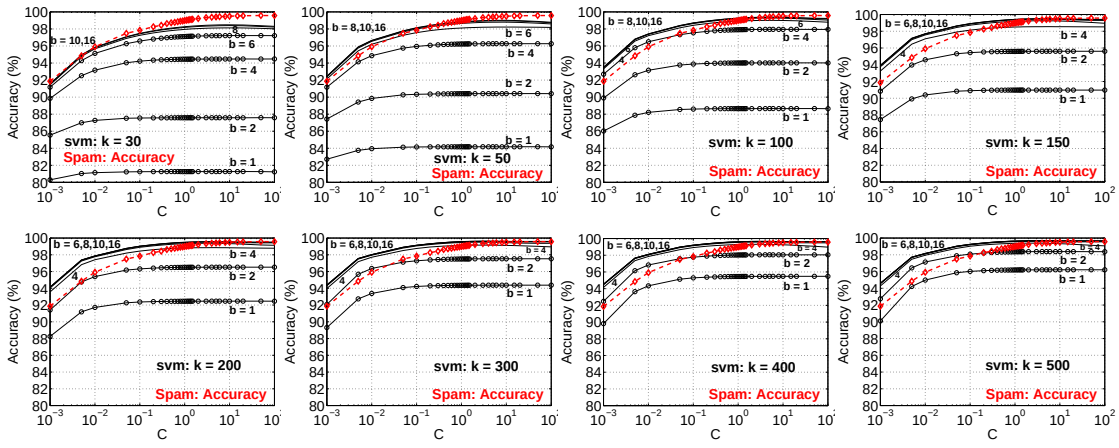

Figure 3.2: **SVM test accuracy** (averaged over 50 repetitions). With $k \geq 200$ and $b \geq 8$. b-bit hashing achieves very similar accuracies as using the original data (dashed, red if color is available).

Compared with the original training time (about 100 seconds), Figure 3.4 (upper panels) shows that our method only needs about 3 seconds (near $C = 1$).

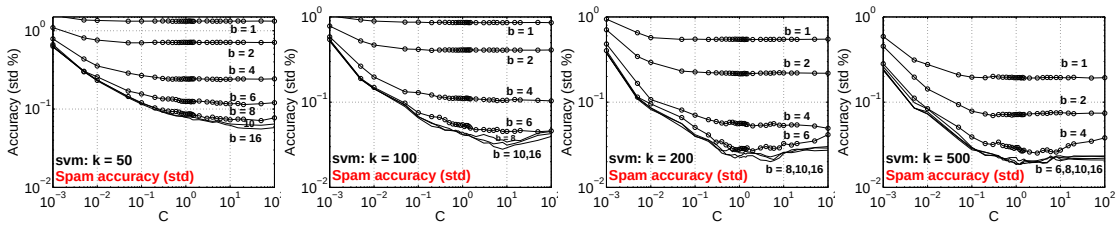

Figure 3.3: **SVM test accuracy (std)**. The standard deviations are computed from 50 repetitions. When $b \geq 8$, the standard deviations become extremely small (e.g., 0.02%).

Note that our reported training time did not include data loading (about 12 minutes for the original data and 10 seconds for the hashed data). Of course, there is a cost for processing (hashing) the data. We show in Chapter 6 that in this case this cost can be made as small as the data reading cost.

Compared with the original testing time (about 150 seconds), Figure 3.4 (bottom panels) shows that our method needs merely about 2 seconds. Note that the testing time includes both the data loading time, as designed by LIBLINEAR. The efficiency of testing may be very important in practice, for example, when the classifier is deployed in a user-facing application (such as search), while the cost of training or preprocessing may be less critical and can be conducted offline.

3.4.2 Evaluations with Logistic Regression

Figure 3.5 presents the test accuracies and training time using logistic regression. Again, with $k \geq 200$ and $b \geq 8$, b-bit minwise hashing can achieve similar test accuracies as using the original data. The training time is substantially reduced, from about 1000 seconds to about 30 seconds only.

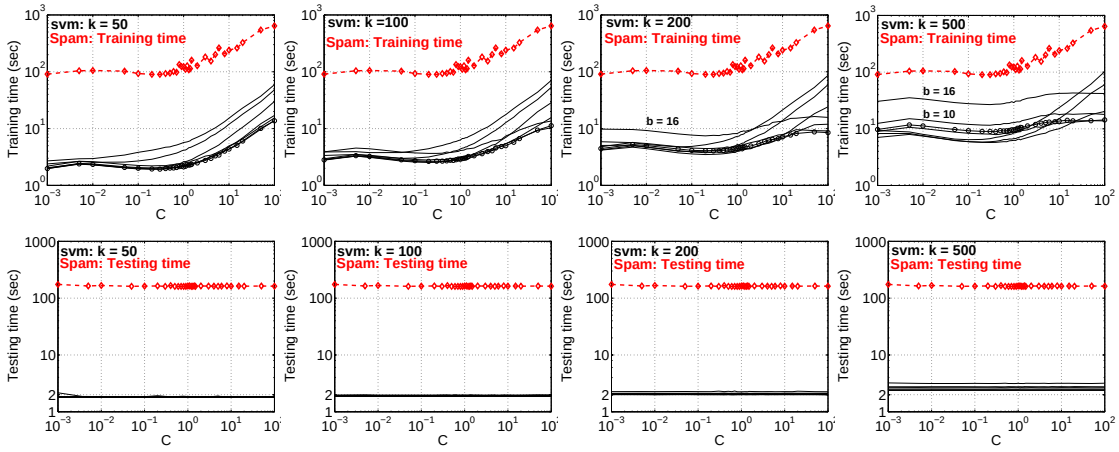

Figure 3.4: **SVM training time (upper panels) and testing time (bottom panels)**. The original costs are plotted using dashed (red, if color is available) curves.

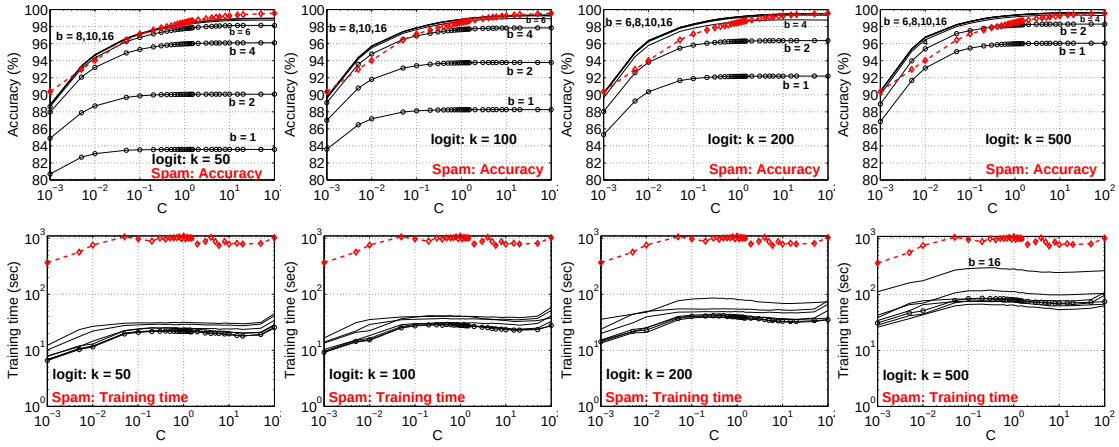

Figure 3.5: **Logistic regression test accuracy (upper panels) and training time (bottom panels)**.

In summary, it appears b-bit hashing is highly effective in reducing the data size and speeding up the training (and testing), for both SVM and logistic regression. We notice that when using $b = 16$, the training time can be much larger than using $b \leq 8$.

3.5 Comparing b-Bit Minwise Hashing with VW (and Random Projections)

We implemented the Vowpal Wabbit (VW) algorithm [] and experimented it on the same webspam dataset. Figure 3.6 shows that b-bit minwise hashing is substantially more accurate (at the same sample size k) and requires significantly less training time (to achieve the same accuracy). Basically, for 8-bit minwise hashing with $k = 200$ achieves similar test accuracies as VW with $k = 10^4 \sim 10^6$ (note that we only stored the non-zeros).

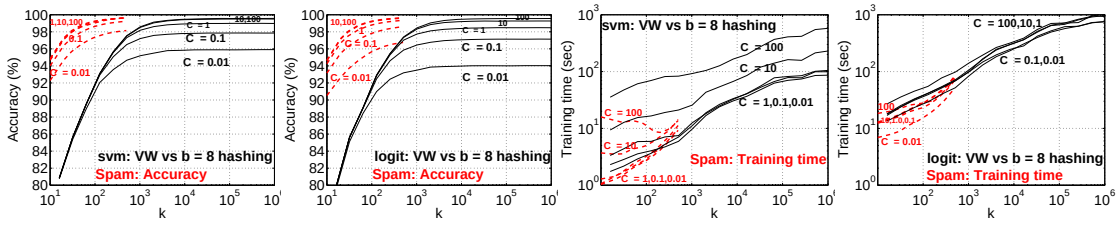

Figure 3.6: Comparison of b-bit minwise hashing (dashed red) with Vowpal Wabbit (VW) (solid black)

This empirical finding is not surprising, because the variance of b-bit hashing is usually substantially smaller than the variance of VW (and random projections) [65].

3.6 Discussions

It is known that randomized hashing makes approximate near-neighbor search problem easier. In particular, existence of locality sensitive hashing scheme, for a given similarity measure, automatically guarantees existence of provable sub-linear time search algorithms for the corresponding similarity measure. In

this chapter we have seen that existence of locality sensitive hashing implies something more. It also implies existence of fast kernel learning algorithms with the corresponding similarity measure as a kernel. In particular, there is a generic construction of random kernel features, using the underlying locality sensitive hashing family, which leads to scalable and efficient linear time kernel learning algorithms.

The generic construction of random kernel features, using LSH, will make various kernel learning with newer kernels feasible at large scale. Kernel learning, in general, is a quadratic time optimization which is prohibitive for modern big-data. We have demonstrated the usefulness of kernel feature constructions using b-bit minwise hashing. The superiority of b-bit minwise hashing features over popular random features like VW [90] is very exciting, and in near future we hope to see many interesting kernels and their corresponding features based on hashing.

CHAPTER 4

MINHASH OR SIMHASH ?

Locality Sensitive Hashing (LSH) (Section 2.2.1) scheme is tightly coupled with the underlying hash function which in turn is defined with respect to a similarity measure. An LSH scheme for one similarity measure cannot be used in general for a different similarity measure. Therefore, it is taken for granted that the two popular hashing schemes MinHash and SimHash, defined in Section 2.2.2, are incomparable and the choice between them is based on whether the desired notion of similarity is resemblance or cosine similarity.

In this chapter, we show that for sparse binary data, which is usually of interest over the web, there is actually a fixed choice among these two hashing schemes. Our theoretical and empirical results show a counter-intuitive fact that for sparse data MinHash is the preferred choice even when the desired measure is cosine similarity. Although the decade old concept of LSH comes with a rich theoretical analysis, there is no machinery to mathematically compare two LSH schemes for different similarity measures. By showing that MinHash is provably superior to SimHash for retrieval with cosine similarity, we provide the first evidence that two LSHs for different similarity measures can be compared.

4.1 MinHash and SimHash for Sparse Binary Data

Current web datasets are typically very sparse and extremely high dimensional, mainly due to the wide adoption of the "Bag of Words" (BoW) representations for documents and images. In BoW or shingle representations, it is known that

the word frequency within a document follows power law [8], indicating that most of the words occur rarely in the document. In "BoW" representations most of the higher order shingles in the document occur only once. It is often the case that just the presence or absence information suffices in practice [16, 38, 44, 65]. Thus, high dimensional web data are almost always binary or binary like. The most information is in the sparsity structure of a vector rather than the magnitude of its components. Many leading search companies use only sparse binary representations in their large data systems [15]. Furthermore, there are many empirical studies which suggest that binary quantizations of datasets achieve good performance in practice [16, 38, 44]. All these factors lead to an increased emphasis on techniques which are capable of exploiting binary datasets.

MinHash is an LSH for resemblance similarity which is only defined over binary vectors. On the other hand SimHash is an LSH for cosine similarity which works for general vectors with real valued components. With the abundance of binary data over the web, it has become a practically important question: *which LSH should be preferred in binary data?*. This question has not been adequately answered in existing literature. In particular it is not even clear if there is a mathematical answer. There were prior empirical evidences. For instance, [78] empirically demonstrated that b-bit minwise hashing [56], a small space variant of MinHash, outperformed SimHash and spectral hashing [91].

4.1.1 Few Notations

Binary data can be viewed as sets (locations of nonzeros). Consider two sets $S_1, S_2 \subseteq \Omega = \{1, 2, ..., D\}$. The cosine similarity ($\mathcal{S}$) is

$$\mathcal{S} = \frac{a}{\sqrt{f_1 f_2}}, \quad \text{where} f_1 = |S_1|, \quad f_2 = |S_2|, \quad a = |S_1 \cap S_2| \tag{4.1}$$

The resemblance similarity, denoted by $\mathcal{R}$, is

$$\mathcal{R} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{a}{f_1 + f_2 - a} \tag{4.2}$$

Clearly these two similarities vary only in the normalization. To better illustrate the connection, we re-write $\mathcal{R}$ as

$$\mathcal{R} = \frac{a/\sqrt{f_1 f_2}}{\sqrt{f_1/f_2} + \sqrt{f_2/f_1} - a/\sqrt{f_1 f_2}} = \frac{S}{z - S} \tag{4.3}$$

$$\text{where } z = z(r) = \sqrt{r} + \frac{1}{\sqrt{r}} \geq 2, \quad r = \frac{f_2}{f_1} \geq 0 \tag{4.4}$$

4.1.2 An Important Inequality

There are two degrees of freedom: f_2/f_1 and a/f_2 between cosine similarity and resemblance in Equation 4.3. This makes it inconvenient for analysis. Fortunately, in the current case, we show in Theorem 2 that we can bound $\mathcal{R}$ by purely functions of $\mathcal{S}$. This is not usually true for other similarity measures. In fact we are not aware of any popular similarity pairs which admit such a relation.

Theorem 2

$$\mathcal{S}^2 \leq \mathcal{R} \leq \frac{\mathcal{S}}{2 - \mathcal{S}} \tag{4.5}$$

Tightness *Without making assumptions on the data, neither the lower bound $\mathcal{S}^2$ or the upper bound $\frac{\mathcal{S}}{2-\mathcal{S}}$ can be improved in the domain of continuous functions.*

Data dependent bound: *If the data satisfy $z \leq z^*$, where z is defined in (4.4), then*

$$\frac{\mathcal{S}}{z^* - \mathcal{S}} \leq \mathcal{R} \leq \frac{\mathcal{S}}{2 - \mathcal{S}} \tag{4.6}$$

Proof: See Appendix A.0.1 □

Figure 4.1 plots the upper and the lower bounds from Theorem 2. It is evident that in high similarity region (e.g., when $S > 0.8$), the upper and lower bounds essentially overlap. Note that, in order for S to be close to 1, we must have $f_1 \approx f_2$ (i.e., $z \approx 2$).

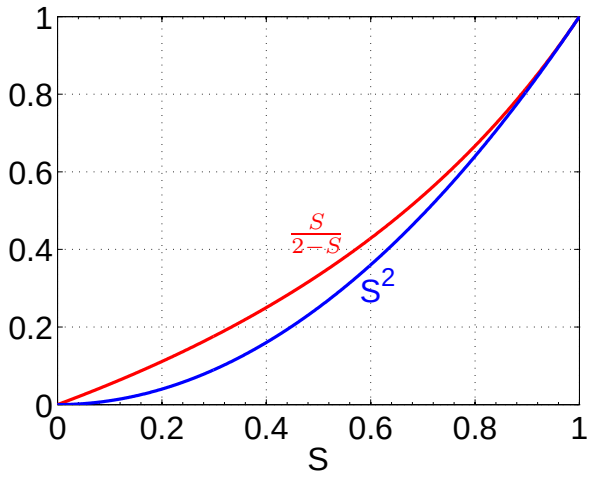

Figure 4.1: Plot of upper and lower bounds in Theorem 2

The inequality in Theorem 2 suggests that any methodology that performs well with respect to resemblance $\mathcal{R}$ will also perform good with respect to $\mathcal{S}$. We make this argument more formal and show that MinHash is also an LSH for cosine similarity and in fact it should be the preferred choice.

4.2 MinHash an LSH for Cosine Similarity

We would like to highlight that the well known ρ values for MinHash and SimHash from Equations 2.5 and 2.9 respectively, which determine the worst case query complexity of these algorithms, are not directly comparable because they are in the context of different similarity measures i.e. resemblance and cosine similarity. To make the comparison possible, we fix our gold standard similarity measure to be the cosine similarity $Sim = \mathcal{S}$. Theorem 2 leads to two simple corollaries:

Corollary 2 *If $\mathcal{S}(x, y) \geq S_0$, then we have*

$$Pr\left(h_\pi^{min}(x) = h_\pi^{min}(y)\right) = \mathcal{R}(x, y) \geq \mathcal{S}(x, y)^2 \geq S_0^2$$

Corollary 3 *If $\mathcal{S}(x, y) \leq cS_0$, then we have*

$$Pr\left(h_\pi^{min}(x) = h_\pi^{min}(y)\right) = \mathcal{R}(x, y) \leq \frac{c\mathcal{S}(x, y)}{2 - c\mathcal{S}(x, y)} \leq \frac{cS_0}{2 - cS_0}$$

Immediate consequence of these two corollaries combined with the definition of LSH is the following:

Theorem 3 *For binary data, MinHash is $(S_0, cS_0, S_0^2, \frac{cS_0}{2-cS_0})$ sensitive family of hash function for cosine similarity with $\rho = \frac{\log S_0^2}{\log \frac{cS_0}{2-cS_0}}$.*

Note we need one more condition that $p_1 \geq p_2$, i.e. $c \leq \frac{2S_0}{1+S_0^2}$, for the above definition to be useful because we want $\rho \leq 1$. We will see that, for almost all interesting region, this condition is always satisfied.

4.2.1 1-bit Minwise Hashing

SimHash generates a single bit output (only the signs) whereas MinHash generates an integer value. Recently proposed b-bit minwise hashing [56] provides simple strategy to generate an informative single bit output from MinHash, by using the parity of MinHash values:

$$h_\pi^{min,1-bit}(S_1) = \begin{cases} 1 & \text{if } h_\pi^{min}(S_1) \text{ is odd} \\ 0 & otherwise \end{cases} \tag{4.7}$$

For 1-bit MinHash and very sparse data (i.e., $\frac{f_1}{D} \to 0$, $\frac{f_2}{D} \to 0$), we have the following collision probability[1]

$$Pr(h_\pi^{min,1bit}(S_1) = h_\pi^{min,1-bit}(S_2)) = \frac{\mathcal{R}+1}{2} \tag{4.8}$$

There are empirical evidence [78] suggesting that for the task of near neighbor search this scheme outperforms SRP. There is no theoretical justification that explains such empirical findings. Our analysis presented in previous sections allow us to theoretically analyze this new scheme. The inequality in Theorem 2 can be modified for $\frac{\mathcal{R}+1}{2}$ as well, which leads to

Theorem 4 *For binary data, 1-bit MH (minwise hashing) is* $(S_0, cS_0, \frac{S_0^2+1}{2}, \frac{1}{2-cS_0})$ *sensitive family of hash function for cosine similarity with* $\rho = \frac{\log \frac{2}{S_0^2+1}}{\log (2-cS_0)}$.

Similar to Theorem 3 we need one condition $\frac{S_0^2+1}{2} \geq \frac{1}{2-cS_0}$.

[1]The exact collision probability for 1-bit MinHash is slightly different but we can always randomly rehash to 0 or 1 and obtain this exact collision probability if necessary

4.2.2 Theoretical Comparisons

Worst Case

Since we fixed our standard similarity measure to be the cosine similarity and re-derive the ρ values from the definitions of LSH. These ρ values are now directly comparable.

$$\text{SimHash: } \rho = \frac{\log\left(1 - \frac{cos^{-1}(\mathcal{S}_0)}{\pi}\right)}{\log\left(1 - \frac{cos^{-1}(c\mathcal{S}_0)}{\pi}\right)} \tag{4.9}$$

$$\text{MinHash: } \rho = \frac{\log S_0^2}{\log \frac{cS_0}{2-cS_0}} \tag{4.10}$$

$$\text{1-bit MH: } \rho = \frac{\log \frac{2}{S_0^2+1}}{\log\left(2 - cS_0\right)} \tag{4.11}$$

This is a worst case analysis. Nevertheless, for high similarity region, the comparisons of the ρ values indicate that MinHash significantly outperforms SimHash as shown in Figure 4.2, at least for $\mathcal{S}_0 \geq 0.8$.

Restricted Worst Case

The worst case analysis does not make any assumption on the data. Although, there is not much hope to improve the lower bound in theory because of the tightness result, we argue that the bound $\mathcal{S}^2 \leq \mathcal{R}$ in Equation 4.5 is usually very conservative in practice. We empirically verify this. We show that for six different real datasets (Table 4.1) covering a wide range of variations, we often have $\mathcal{R} = \frac{\mathcal{S}}{z-\mathcal{S}}$ with z only being slightly larger than 2, as shown in Figure 4.3.

For each dataset, we compute both cosine and resemblance for every query-train pair (e.g., 10000×60000 pairs for MNIST dataset). For each query point,

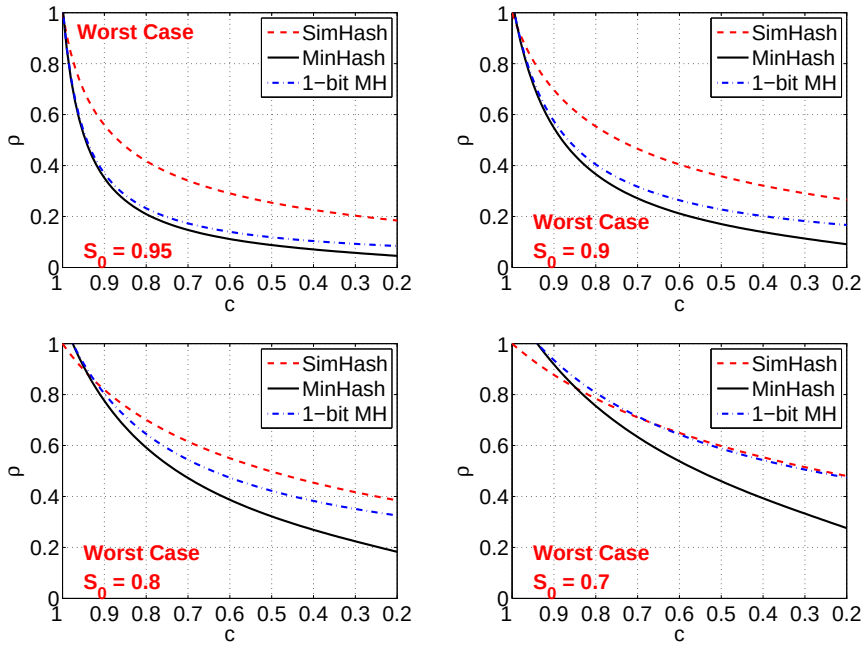

Figure 4.2: Worst case gap (ρ) analysis between MinHash and SimHash for high similarity region.

Table 4.1: Datasets Used for Comparing MinHash and SimHash

Dataset	# Query	# Train	# Dim
MNIST	10,000	60,000	784
NEWS20	2,000	18,000	1,355,191
NYTIMES	5,000	100,000	102,660
RCV1	5,000	100,000	47,236
URL	5,000	90,000	3,231,958
WEBSPAM	5,000	100,000	16,609,143

we rank its similarities to all training points in descending order. We examine the top-1000 locations as in Figure 4.4. For every top location, we plot the median (among all query points) of the similarities, separately for cosine (dashed)

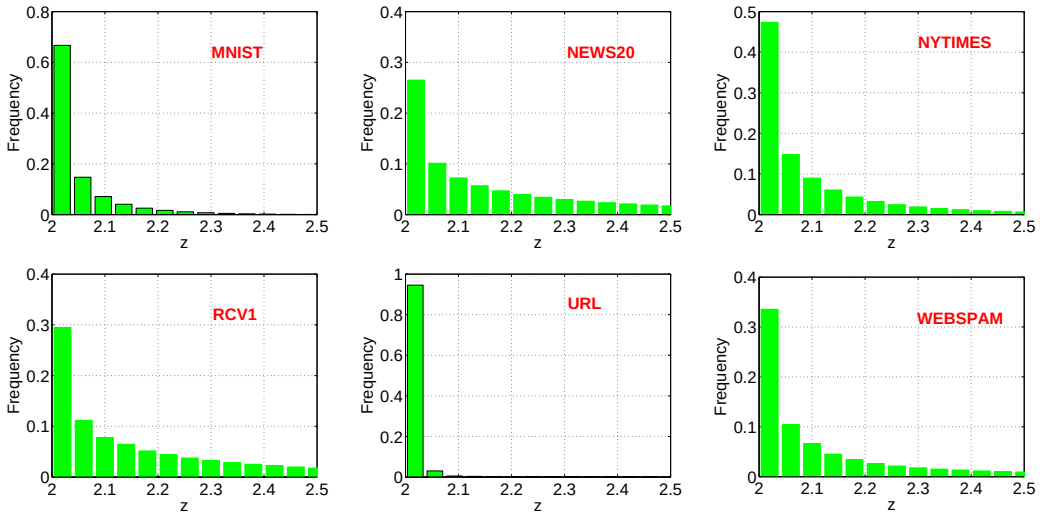

Figure 4.3: Frequencies of the z (Eq 4.4) values for different datasets.

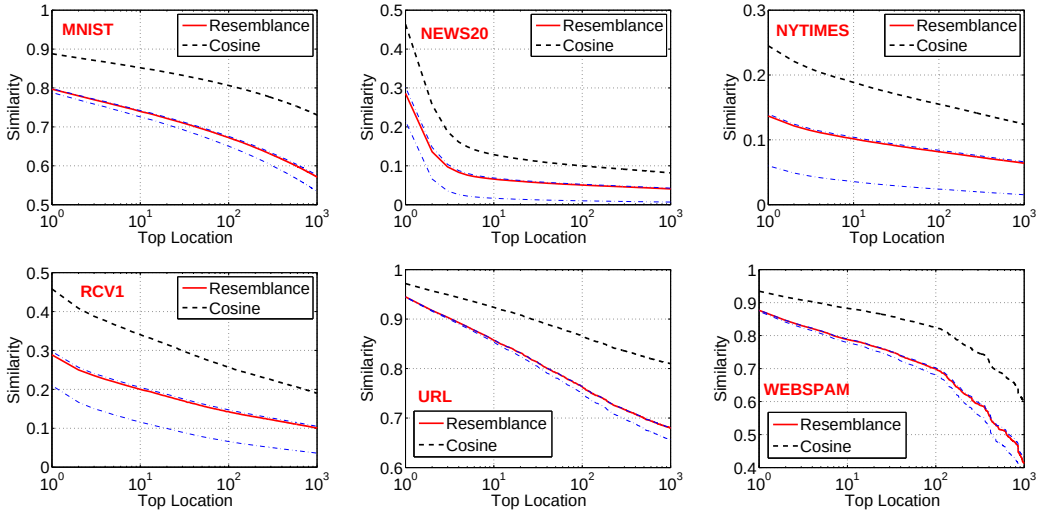

Figure 4.4: Plot of Median Cosine Similarity (dashed), Resemblance (bold) from different real datasets along with theoretical upper and lower bounds (dot dashed)

and resemblance (solid), together with the lower and upper bounds of $\mathcal{R}$ (dot-dashed). Interestingly, for all six datasets, $\mathcal{R}$ matches fairly well with the upper bound $\frac{\mathcal{S}}{2-\mathcal{S}}$. In other words, the lower bound $\mathcal{S}^2$, although tight, can be very conservative in practice.

We know that if the data satisfy $z \leq z^*$, where z is defined in (4.4), then we have better bound from Theorem 2. Figure 4.3 has demonstrated that in real data, we can fairly safely replace the lower bound $\mathcal{S}^2$ with $\frac{\mathcal{S}}{z-\mathcal{S}}$ for some z which, defined in (4.4), is very close to 2 (for example, 2.1). If we are willing to make this assumption, then we can go through the same analysis for MinHash as an LSH for cosine and compute the corresponding ρ values:

$$\text{MinHash: } \rho = \frac{\log \frac{\mathcal{S}_0}{z-\mathcal{S}_0}}{\log \frac{c\mathcal{S}_0}{2-c\mathcal{S}_0}} \tag{4.12}$$

$$\text{1-bit MH: } \rho = \frac{\log \frac{2(z-\mathcal{S}_0)}{z}}{\log\left(2 - c\mathcal{S}_0\right)} \tag{4.13}$$

Note that this is still a worst case analysis (and hence can still be very conservative). Figure 4.5 presents the ρ values for this restricted worst case gap analysis, for two values of z (2.1 and 2.3) and $\mathcal{S}_0$ as small as 0.2. The results confirms that MinHash still significantly outperforms SimHash theoretically even in low similarity region.

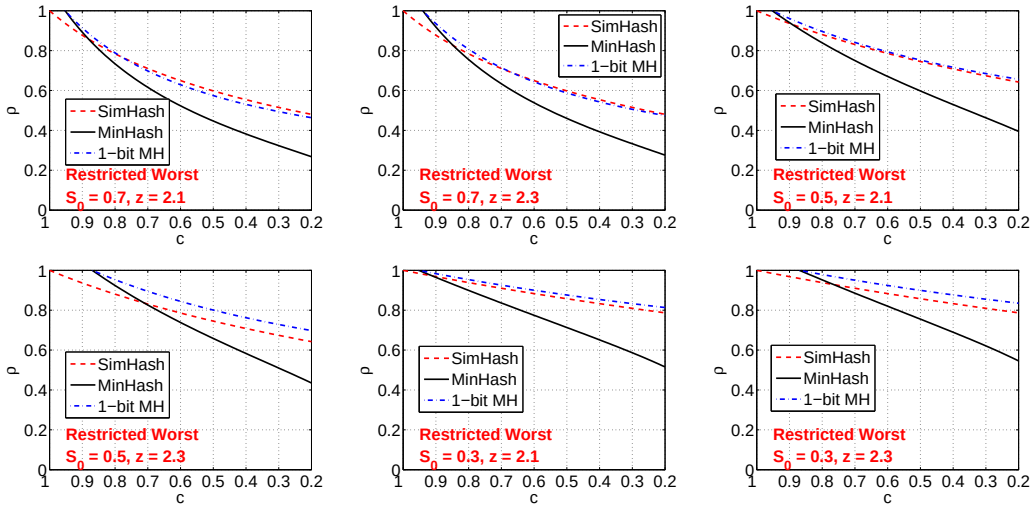

Figure 4.5: Restricted worst case gap (ρ) analysis, between MinHash and SimHash, by assuming the data satisfy $\frac{\mathcal{S}}{z-\mathcal{S}} \leq R \leq \frac{\mathcal{S}}{2-\mathcal{S}}$, where z is defined in (4.4).

Both Figure 4.2 and Figure 4.5 show that 1-bit MinHash can be less competitive when the similarity is not high. This is expected as analyzed in [55]. The remedy is to use more bits.

Idealized Case Gap Analysis

The restricted worst case analysis can still be very conservative and may not fully explain the stunning performance of MinHash in our experiments on datasets of low similarities. The problem with the existing analysis is that we require bounding the p_1 and p_2 (LSH Definition 2) for MinHash. On the other hand for SimHash we have exact values and no bounds. This makes the ρ value comparisons favorable for SimHash. Therefore to mitigate this factor, we also provide an analysis based on fixed z value. That is, we only analyze the gap ρ by assuming $\mathcal{R} = \frac{\mathcal{S}}{z-\mathcal{S}}$ for a fixed z. We call this idealized gap analysis. Not surprisingly, Figure 4.6 confirms that, with this assumption, MinHash significantly outperform SimHash even for extremely low similarity. We should keep in mind that this idealized gap analysis can be somewhat optimistic and should only be used as some side information.

4.3 Benefits In Practice

We evaluate both MinHash and SimHash in the actual task of retrieving top-k near neighbors. We implemented the standard (K, L) parameterized LSH [42] algorithm with both MinHash and SimHash. That is, we concatenate K hash functions to form a new hash function for each table, and we generate L such tables. We used all the six binarized datasets with the query and training parti-

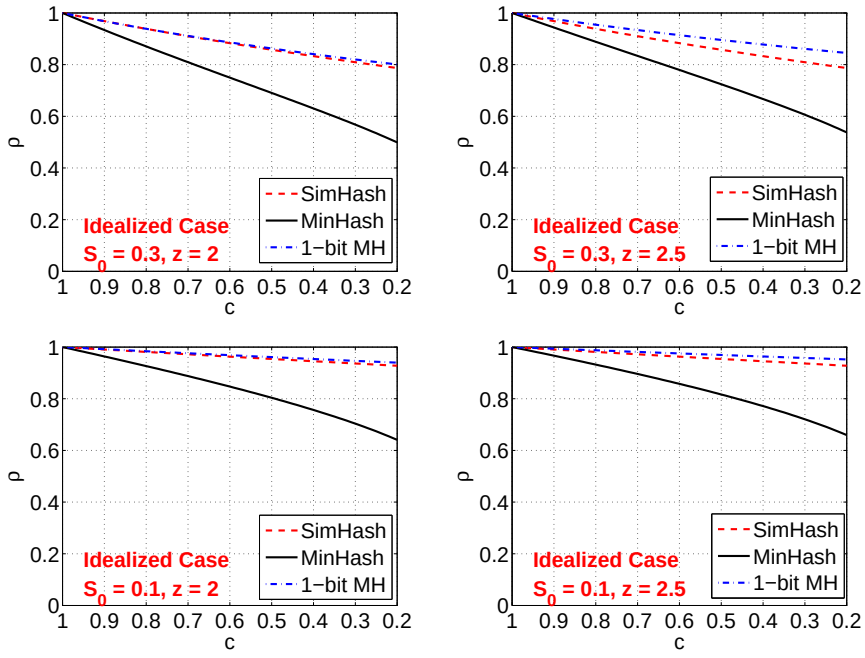

Figure 4.6: Idealized case gap (ρ) analysis, between MinHash and SimHash, by assuming $\mathcal{R} = \frac{\mathcal{S}}{z-\mathcal{S}}$ for a fixed z ($z = 2$ and $z = 2.5$ in the plots).

tion size as shown in Table 4.1. For each of the datasets, elements from training partition were used for constructing hash tables, while the elements of the query partition was used as query for top-k neighbor search. For every query, we compute the gold standard top-k near neighbors using the **cosine similarity**.

In standard (K, L) parameterized bucketing scheme the choice of K and L is dependent on the similarity thresholds and the hash function under consideration. In the task of top-k near-neighbor retrieval, the similarity thresholds vary with the datasets. Hence, the actual choice of ideal K and L is hard to determine. To ensure that this choice does not affect our evaluations, we implemented all the combination $K \in \{1, 2, ..., 30\}$ and $L \in \{1, 2, ..., 200\}$. This combination includes all the reasonable choices for both the hash functions.

For each combinations of (K, L) and for both of the hash functions, we com-

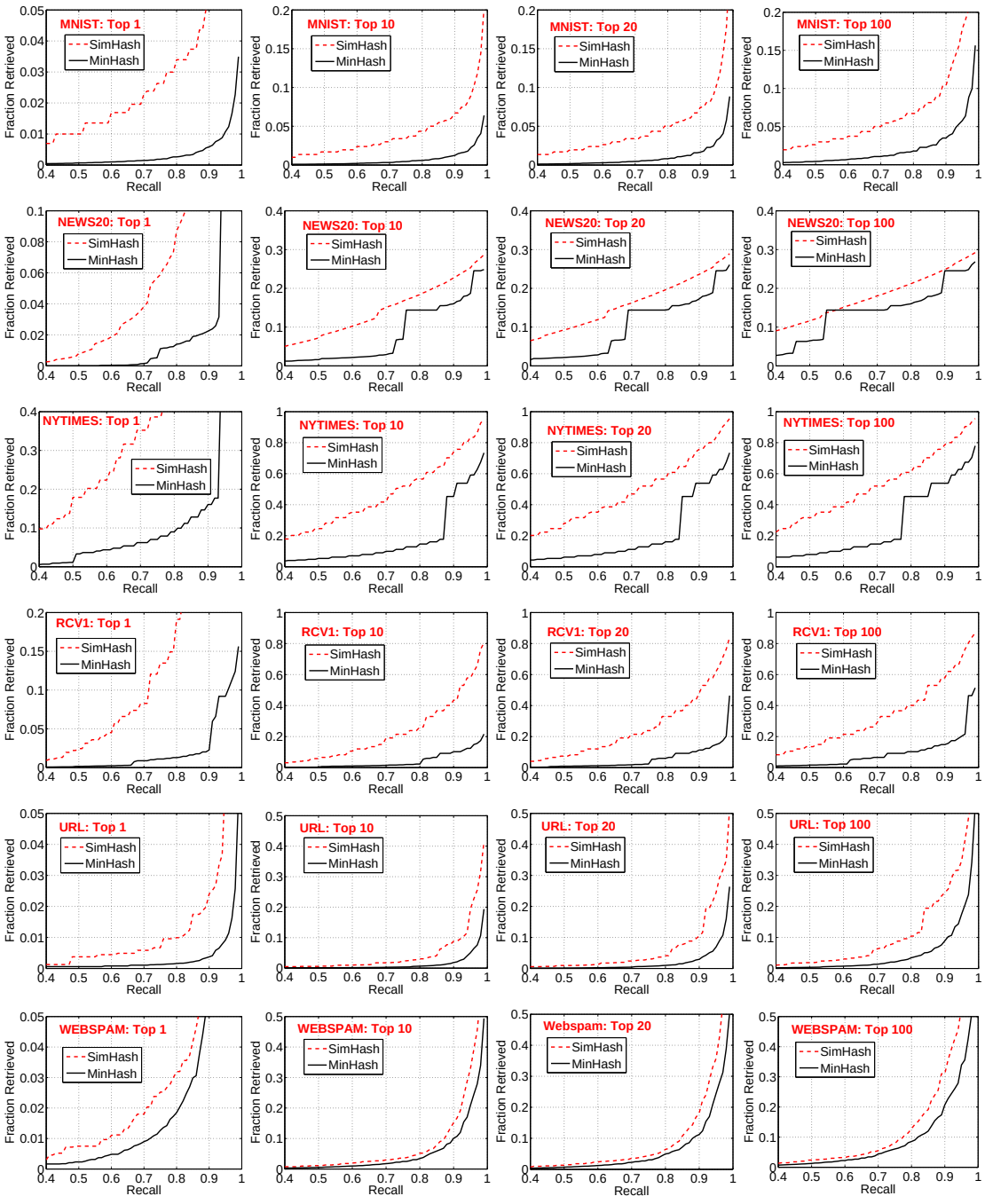

Figure 4.7: MinHash and SimHash comparisons on binary data.

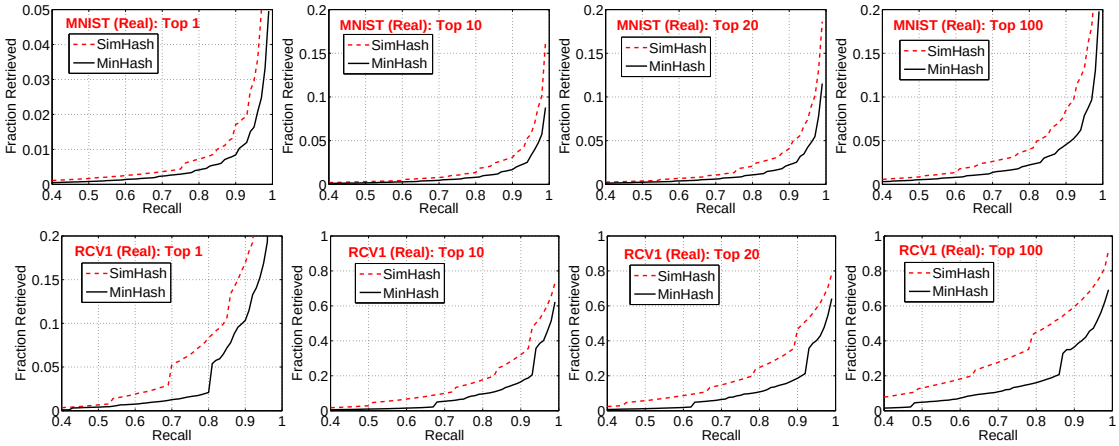

Figure 4.8: MinHash and SimHash comparison on sparse real-valued data. MinHash does not uses value information but still outperforms SimHash

puted the mean recall of the top-k gold standard neighbors along with the average number of points reported per query. We then compute the least number of points needed, by each of the two hash functions, to achieve a given percentage of recall of the gold standard top-k, where the least was computed over the choices of K and L. We are therefore ensuring the best over all the choices of K and L for each hash function independently. This eliminates the effect of K and L, if any, in the evaluations. The plots of the fraction of points retrieved at different recall levels, for $k = 1,\ 10,\ 20,\ 100$ are in Figure 4.7.

A good hash function, at a given recall should retrieve less number of points. MinHash needs to evaluate significantly less fraction of the total data points to achieve a given recall compared to SimHash. MinHash is consistently better than SimHash, in most cases very significantly, irrespective of the choices of dataset and k. It should be noted that our gold standard measure for computing top-k neighbors is cosine similarity. This should favor SimHash because it was the only known LSH for cosine similarity. Despite this "disadvantage",

MinHash still outperforms SimHash in top near neighbor search with cosine similarity. This nicely confirms our theoretical gap analysis.

To conclude this section, we also add a set of experiments using the original (real-valued) datasets, for MNIST and RCV1. We apply SimHash on the original data and MinHash on the binarized data. We also evaluate the retrieval results based on the cosine similarities of the original data. This set-up places MinHash in a very disadvantageous place compared to SimHash. Nevertheless, we can see from Figure 4.8 that MinHash still noticeably outperforms SimHash, although the improvements are not as significant, compared to the experiments on binarized data (Figure 4.7). The supports the fact that for very sparse dataset binary information is sufficient for most purposes.

4.4 Discussions

The fact that, for binary data, MinHash is provably superior to SimHash, even for retrieving with cosine similarity, was quite unexpected. It might appear as orange to apple comparison at first, but closer investigation reveals something different, as shown in this chapter. We believe that the conclusions presented here provide sufficient evidence for revisiting existing recommended algorithm for different measures of interest, for example different kernels in machine learning. A relation between cosine similarity and resemblance made all the comparisons possible. This motivates the study of relations between different similarity measures, which can be helpful in finding better algorithms for the problem at hand. We hope to see many results along these lines in future.

CHAPTER 5

HASHABILITY FOR K-WAY SIMILARITIES

Existing notions of similarity in search problems mainly work with pairwise similarity functions. In this chapter, we go beyond this notion and look at the problem of k-way similarity search, where the similarity function of interest involves k arguments ($k \geq 2$). An example of higher order similarity is the **3-way Jaccard** similarity which is defined as:

$$\mathcal{R}^{3way} = \frac{|S_1 \cap S_2 \cap S_3|}{|S_1 \cup S_2 \cup S_3|}, \tag{5.1}$$

$S_1, S_2, S_3 \in C$, where C is a size n collection of sets (or binary vectors). In this chapter, we investigate the usefulness of higher order similarity search in many practical applications and furthermore show some fundamental results on the efficiency of these search problems. In particular, we show that a class of approximate $\mathcal{R}^{3way}$ Jaccard similarity search problems admit fast algorithms with provable guarantees, analogous to the pairwise case. Our analysis and speedup guarantees naturally extend to k-way resemblance [52, 57] defined over k sets $\{S_1, S_2, ..., S_k\}$ as

$$\mathcal{R}^{k-way} = \frac{|S_1 \cap S_2 \cap ... \cap S_k|}{|S_1 \cup S_2 \cup ... \cup S_k|}. \tag{5.2}$$

In the process, we extend traditional framework of *locality sensitive hashing (LSH)* to handle higher-order similarities, which could be of independent theoretical interest.

5.1 Importance of k-way Resemblance

We list **four** practical scenarios where k-way resemblance search would be a natural choice and in the later section provide some empirical support.

(i) Google Sets: (http://googlesystem.blogspot.com/2012/11/google-sets-still-available.html) *Google Sets* is among the earliest google projects, which allows users to generate list of similar words by typing only few related keywords. For example, if the user types "mazda" and "honda" the application will automatically generate related words like "bmw", "ford", "toyota", etc. This application is currently available in google spreadsheet. If we assume the term document binary representation of each word w in the database, then given query w_1 and w_2, we show that $\frac{|w_1 \cap w_2 \cap w|}{|w_1 \cup w_2 \cup w|}$ turns out to be a very good similarity measure for this application (see Section 5.1.1).

(ii) Joint recommendations: Users A and B would like to watch a movie together. The profile of each person can be represented as a sparse vector over a giant universe of attributes. For example, a user profile may be the set of actors, actresses, genres, directors, etc, which she/he likes. On the other hand, we can represent a movie M in the database over the same universe based on attributes associated with the movie. If we have to recommend movie M, jointly to users A and B, then a natural measure to maximize is $\frac{|A \cap B \cap M|}{|A \cup B \cup M|}$. The problem of group recommendation [5] is applicable in many more settings such as recommending people to join circles, etc.

(iii) Improving retrieval quality: We are interested in finding images of a particular type of object, and we have two or three(possibly noisy) representative images. In such a scenario, a natural expectation is that retrieving images simultaneously similar to all the representative images should be more refined than just retrieving images similar to any one of them. In Section 5.1.1, we demonstrate that in cases where we have more than one elements to search for, we can refine our search quality using k-way resemblance search. In a dynamic

feedback environment [11], we can improve subsequent search quality by using k-way similarity search on the pages already clicked by the user.

(iv) Beyond pairwise clustering: While machine learning algorithms often utilize the data through pairwise similarities (e.g., inner product or resemblance), there are natural scenarios where the affinity relations are not pairwise, but rather triadic, tetradic or higher [2, 93]. The computational cost, of course, will increase exponentially if we go beyond pairwise similarity.

5.1.1 Empirical Observations

In this section, we empirically demonstrate the usefulness of 3-way and higher-order similarity search using (i) Google Sets, and (ii) Improving retrieval quality.

Google Sets: Generating Semantically Similar Words

The task is to retrieve words which are "semantically" similar to the given set of query words. We collected 1.2 million random documents from Wikipedia and created a standard term-doc binary vector representation of each term present in the collected documents after removing standard stop words and punctuation marks. More specifically, every word is represented as a 1.2 million dimension binary vector indicating its presence or absence in the corresponding document. The total number of terms (or words) was around 60,000 in this experiment.

Since there is no standard benchmark available for this task, we show qualitative evaluations. For querying, we used the following four pairs of semantically related words: (i) "jaguar" and "tiger"; (ii) "artificial" and "intelligence";

(iii) "milky" and "way" ; (iv) "finger" and "lakes". Given the query words w_1 and w_2, we compare the results obtained by the following four methods.

- **Google Sets:** We use Google's algorithm and report 5 words from Google spreadsheets [1]. This is Google's algorithm which uses its own data.
- **3-way Resemblance (3-way):** We use 3-way resemblance $\frac{|w_1 \cap w_2 \cap w|}{|w_1 \cup w_2 \cup w|}$ to rank every word w and report top 5 words based on this ranking.
- **Sum Resemblance (SR):** Another intuitive method is to use the sum of pairwise resemblance $\frac{|w_1 \cap w|}{|w_1 \cup w|} + \frac{|w_2 \cap w|}{|w_2 \cup w|}$ and report top 5 words based on this ranking.
- **Pairwise Intersection (PI):** We first retrieve top 100 words based on pairwise resemblance for each w_1 and w_2 independently. We then report the words common in both. If there is no word in common we do not report anything.

The results in Table 5.1 demonstrate that using 3-way resemblance retrieves reasonable candidates for these four queries. An interesting query is "finger" and "lakes". Finger Lakes is a region in upstate New York. Google could only relate it to New York, while 3-way resemblance could even retrieve the names of cities and lakes in the region. Also, for query "milky" and "way", we can see some (perhaps) unrelated words like "dance" returned by Google. We do not see such random behavior with 3-way resemblance. For the query "Jaguar" and "Tiger" all the results given by 3-way resemblance belong to the cat family where as Google retrieves dog which seems suboptimal. Although we are not aware of the algorithm and the dataset used by Google, we can see that 3-way resemblance appears to be a right measure for this application.

Table 5.1: Retrieval with various measure on Google Sets queries

"JAGUAR" AND " TIGER"				"ARTIFICIAL" AND "INTELLIGENCE"			
GOOGLE	3-WAY	SR	PI	GOOGLE	3-WAY	SR	PI
LION	LEOPARD	CAT	—	COMPUTER	COMPUTER	SECURITY	—
LEOPARD	CHEETAH	LEOPARD	—	PROGRAMMING	SCIENCE	WEAPONS	—
CHEETAH	LION	LITRE	—	SCIENCE	INTELLIGENT	SECRET	—
CAT	PANTHER	BMW	—	ROBOT	HUMAN	ATTACKS	—
DOG	CAT	CHASIS	—	ROBOTICS	TECHNOLOGY	HUMAN	—

"MILKY" AND " WAY"				"FINGER" AND "LAKES"			
GOOGLE	3-WAY	SR	PI	GOOGLE	3-WAY	SR	PI
DANCE	GALAXY	EVEN	—	NEW	SENECA	RIVERS	—
STARS	STARS	ANOTHER	—	YORK	CAYUGA	FRESHWATER	—
SPACE	EARTH	STILL	—	NY	ERIE	FISH	—
THE	LIGHT	BACK	—	PARK	ROCHESTER	STREAMS	—
UNIVERSE	SPACE	TIME	—	CITY	IROQUOIS	FORESTED	—

The results clearly show the problem with the sum of pairwise similarity method. The similarity value with one of the words dominate the sum and hence we see for queries "artificial" and "intelligence" that all the retrieved words are mostly related to the word "intelligence". Same is the case with query "finger" and "lakes" as well as "jaguar" and "tiger". Note, "jaguar" is also a car brand. Also, for all 4 queries, there was no common word in the top 100 words similar to the each query word individually and so PI method gives no answer.

The results are not surprising because 3-way resemblance seems the most reasonable metric which clearly models co-occurrences. We should note the importance of the denominator term in 3-way resemblance, without which frequent words will be blindly favored. 3-way resemblance seems a very natural and useful measure. The most exciting part , which we show later, is that 3-way resemblance similarity search admits provable sub-linear guarantees making it an ideal choice. On the other hand, no such known provable guarantees exist

for SR and other such heuristic based search methods.

Improving Retrieval Quality in Similarity Search

In traditional near neighbor similarity search, we have a single query, and we are interested in searching a database for candidates very similar to the given query. In many situations it is possible to get a very small set of representative object instead of just one, for instance we might be interested in searching for an image of an animal or an object and we have two or three representative images instead of just one. Even in the case when we just have one query, we can search for few similar objects and then filter those retrieved points to generate a few representative candidate to refine the search. This filtering of retrieved candidates could be application dependent, for instance it could be the click feedback of initial search query results. In such scenario a natural expectation is that searching for objects simultaneously similar to the set of representative candidates should be of better compared to any single query search.

We demonstrate how the retrieval quality of traditional similarity search can be boosted by utilizing more query candidates instead of just one. For the evaluations we choose two public datasets: MNIST and WEBSPAM. The two datasets reflect diversity both in terms of task and scale that is encountered in practice. The MNIST dataset consists of handwritten digit samples. Each sample is an image of 28×28 pixel yielding a 784 dimension vector with the associated class label (digit $0-9$). We binarize the data by settings all non zeros to be 1. We used the standard partition of MNIST, which consists of 10,000 samples in one set and 60,000 in the other. The WEBSPAM dataset, with 16,609,143 features, consists of sparse vector representation of emails labeled as spam or not. We randomly

sample 70,000 data points and partitioned them into two independent sets of size 35,000 each.

Table 5.2: Percentage of top candidates with the same labels as that of query retrieved using k-way (k = 2, 3 and 4) resemblance.

	MNIST				WEBSPAM			
TOP	1	10	20	50	1	10	20	50
Pairwise	94.20	92.33	91.10	89.06	98.45	96.94	96.46	95.12
3-way	96.90	96.13	95.36	93.78	99.75	98.68	97.80	96.11
4-way	**97.70**	**96.89**	**96.28**	**95.10**	**99.90**	**98.87**	**98.15**	**96.45**

For evaluation, we need to generate potential similar search query candidates for k-way search. It makes no sense in trying to search for object simultaneously similar to two very different objects. To generate such query candidates, we took one independent set of the data and partition it according to the class labels. We then run a cheap k-mean clustering on each class, and randomly sample triplets $< x_1, x_2, x_3 >$ from each cluster for evaluating 2-way, 3-way and 4-way similarity search. For MNIST dataset, the standard 10,000 test set was partitioned according to the labels into 10 sets, each partition was then clustered into 10 clusters, and we choose 10 triplets randomly from each cluster. In all we had 100 such triplets for each class, and thus 1000 overall query triplets. For WEBSPAM, which consists only of 2 classes, we choose one of the independent set and performed the same procedure. We selected 100 triplets from each cluster. We thus have 1000 triplets from each class making the total number of 2000 query candidates.

The above procedures ensures that the elements in each triplets $< x_1, x_2, x_3 >$ are not very far from each other and are of the same class label. For each triplet

$< x_1, x_2, x_3 >$, we sort all the points x in the other independent set based on:

- **Pairwise:** We only use x_1 and rank x based on resemblance $\frac{|x_1 \cap x|}{|x_1 \cup x|}$.
- **3-way NN:** We rank x based on 3-way resemblance $\frac{|x_1 \cap x_2 \cap x|}{|x_1 \cup x_2 \cup x|}$.
- **4-way NN:** We rank x based on 4-way resemblance $\frac{|x_1 \cap x_2 \cap x_3 \cap x|}{|x_1 \cup x_2 \cup x_3 \cup x|}$.

We look at the top 1, 10, 20 and 50 points based on orderings described above. Since, all the query triplets are of the same label, The percentage of top retrieved candidates having same label as that of the query items is a natural metric to evaluate the retrieval quality. This percentage values accumulated over all the triplets are summarized in Table 5.2.

We can see that top candidates retrieved by 3-way resemblance similarity, using 2 query points, are of better quality than vanilla pairwise similarity search. Also 4-way resemblance, with 3 query points, further improves the results compared to 3-way resemblance similarity search. This clearly demonstrates that multi-way resemblance similarity search is more desirable whenever we have more than one representative query in mind. Note that, for MNIST, which contains 10 classes, the boost compared to pairwise retrieval is substantial. The results follow a consistent trend.

In is evident from experiments that searching with k-way resemblance can be significantly beneficial in practice. With the data explosion in modern applications, the brute force way of scanning all the data for searching is prohibitively expensive, specially in user-facing applications like search. The need for k-way similarity search can only be fulfilled if it admits efficient algorithms. As we shown later, it turns out that searching with k-way resemblance is efficient.

5.2 Problem Formulation

Our focus will remain on binary vectors which can also be viewed as sets. We illustrate our method using 3-way resemblance similarity function $Sim(S_1, S_2, S_3) = \frac{|S_1 \cap S_2 \cap S_3|}{|S_1 \cup S_2 \cup S_3|}$. The algorithm and guarantees naturally extends to k-way resemblance. Given a size n collection $C \subseteq 2^{\Omega}$ of sets (or binary vectors), we are particularly interested in the following three problems:

1. Given two query sets S_1 and S_2, find $S_3 \in C$ that maximizes $Sim(S_1, S_2, S_3)$.
2. Given a query set S_1, find two sets $S_2, S_3 \in C$ maximizing $Sim(S_1, S_2, S_3)$.
3. Find three sets $S_1, S_2, S_3 \in C$ maximizing $Sim(S_1, S_2, S_3)$.

The brute force way of enumerating all possibilities leads to the worst case query time of $O(n)$, $O(n^2)$ and $O(n^3)$ for problem 1, 2 and 3, respectively. In a hope to break this barrier, just like the case of pairwise near neighbor search, we define the c-approximate ($c < 1$) versions (Definition 1) of the above three problems. As in the case of c-NN, we are given two parameters $R_0 > 0$ and $\delta > 0$. For each of the following cases, the guarantee is with probability at least $1 - \delta$:

1. (**3-way c-Near Neighbor or 3-way c-NN**) Given two query sets S_1 and S_2, if there exist $S_3 \in C$ with $Sim(S_1, S_2, S_3) \geq R_0$, then we report some $S_3' \in C$ so that $Sim(S_1, S_2, S_3') \geq cR_0$.
2. (**3-way c-Close Pair or 3-way c-CP**) Given a query set S_1, if there exist a pair of set $S_2, S_3 \in C$ with $Sim(S_1, S_2, S_3) \geq R_0$, then we report sets $S_2', S_3' \in C$ so that $Sim(S_1, S_2', S_3') \geq cR_0$.

3. (**3-way c-Best Cluster or 3-way c-BC**) If there exist sets $S_1, S_2, S_3 \in C$ with $Sim(S_1, S_2, S_3) \geq R_0$, then we report sets $S_1', S_2', S_3' \in C$ so that $Sim(S_1', S_2', S_3') \geq cR_0$.

5.3 Fast Algorithms via Hashing

In this section we show that using hashing we can obtain provably fast algorithms for the c-approximate search problem formalized in the previous Section.

5.3.1 Sub-linear Algorithm for 3-way c-NN

The basic philosophy behind sub-linear search is bucketing, which allows us to preprocess dataset in a fashion so that we can filter many bad candidates without scanning all of them. LSH-based techniques rely on randomized hash functions to create buckets that probabilistically filter bad candidates. This philosophy is not restricted for binary similarity functions and is much more general. Here, we first focus on 3-way c-NN problem for binary data.

Theorem 5 *For $\mathcal{R}^{3way}$ c-NN one can construct a data structure with $O(n^{\rho} \log_{1/cR_0} n)$ query time and $O(n^{1+\rho})$ space, where $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$.* □

Minwise hashing idea for 2-way resemblance can be naturally extended to k-way resemblance. Specifically, given three sets $S_1, S_2, S_3 \subseteq \Omega$ and an independent random permutation $\pi : \Omega \rightarrow \Omega$, we have with MinHash (Section 2.4):

$$Pr\left(\min(\pi(S_1)) = \min(\pi(S_2)) = \min(\pi(S_3))\right) = \mathcal{R}^{3way}. \tag{5.3}$$

Equation 5.3 shows that minwise hashing, although it operates on sets individually, preserves all 3-way (in fact k-way) similarity structure of the data. The existence of such a hash function is the key requirement behind the existence of efficient approximate search. For the pairwise case, the probability event was a simple hash collision, and the min-hash itself serves as the bucket index. In case of 3-way (and higher) c-NN problem, we have to take care of a more complicated event to create an indexing scheme. In particular, during preprocessing we need to create buckets for each individual S_3, and while querying we need to associate the query sets S_1 and S_2 to the appropriate bucket. We need extra mechanisms to manipulate these minwise hashes to obtain a bucketing scheme. Later in Chapter 7, we will see that this idea, of having a mechanism for assigning buckets to query and a different mechanism for preprocessing the elements in the collection, is very powerful and a strict generalization of existing LSH framework. The proof of Theorem 5 is constructive and conveys the major idea.

Proof of Theorem 5: We use two additional functions: $f_1 : \Omega \rightarrow N$ for manipulating $\min(\pi(S_3))$ and $f_2 : \Omega \times \Omega \rightarrow N$ for manipulating both $\min(\pi(S_1))$ and $\min(\pi(S_2))$. Let $a \in \mathbb{N}^+$ be such that $|\Omega| = D < 10^a$. We define $f_1(x) = (10^a + 1) \times x$ and $f_2(x, y) = 10^a x + y$. This choice ensures that given query S_1 and S_2, for any $S_3 \in C$, $f_1(\min(\pi(S_3)) = f_2(\min(\pi(S_1)), \min(\pi(S_2)))$ holds if and only if $(\min(\pi(S_1)) = \min(\pi(S_2)) = \min(\pi(S_2))$, and thus we get a bucketing scheme. To complete the proof, we introduce two integer parameters K and L. Define a new hash function by concatenating K events. To be more precise, while preprocessing, for every element $S_3 \in C$ create buckets $g_1(S_3) = [f_1(h_1(S_3)); ...; f_1(h_K(S_3))]$ where h_i is chosen uniformly from minwise hashing family. For given query points S_1 and S_2, retrieve only points in the bucket $g_2(S_1, S_2) = [f_2(h_1(S_1), h_1(S_2)); ...; f_2(h_K(S_1), h_K(S_2))]$. Repeat this process L

times independently. For any $S_3 \in C$, with $Sim(S_1, S_2, S_3) \geq R_0$, is retrieved with probability at least $1 - (1 - R_0^K)^L$. Using $K = \lceil \frac{\log n}{\log \frac{1}{cR_0}} \rceil$ and $L = \lceil n^\rho \log(\frac{1}{\delta}) \rceil$, where $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$, the proof can be obtained using standard concentration arguments used to prove Fact 1, see [42, 36]. It is worth noting that the probability guarantee parameter δ gets absorbed in the constants as $\log(\frac{1}{\delta})$. Note, the process is stopped as soon as we find some element with $\mathcal{R}^{3way} \geq cR_0$. □

Theorem 5 can be easily extended to k-way resemblance with same query time and space guarantees. Note that k-way c-NN is at least as hard as k^*-way c-NN for any $k^* \leq k$, because we can always choose $(k - k^* + 1)$ identical query sets in k-way c-NN, and it reduces to k^*-way c-NN problem. With this observation, and noting that the expression for ρ is similar with the ρ for the pairwise case, we can infer that the guarantees are hard to improve. Any improvements in $\mathcal{R}^{3way}$ c-NN will imply improvements in the classical min-hash LSH for Jaccard similarity.

One Universal Data Structure for All: An interesting consequence is that it possible to also perform the traditional pairwise c-NN search using the same hash tables deployed for 3-way c-NN. In the query phase we have an option, if we have two different queries S_1, S_2, then we retrieve from bucket $g_2(S_1, S_2)$ and that is usual 3-way c-NN search. If we are just interested in pairwise near neighbor search given one query S_1, then we will look into bucket $g_2(S_1, S_1)$, and we know that the 3-way resemblance between S_1, S_1, S_3 boils down to the pairwise resemblance between S_1 and S_3. So, the same hash tables can be used for both the purposes. This property generalizes, and hash tables created for k-way c-NN can be used for any k^*-way similarity search for $k^* \leq k$. The approximation guarantees still holds. This flexibility makes k-way bucketing strictly

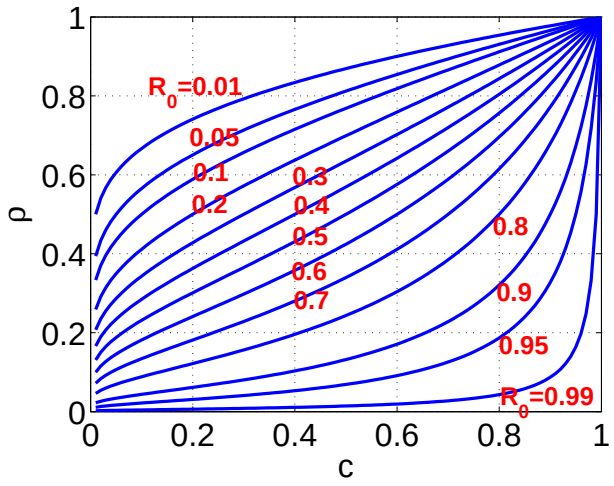

Figure 5.1: $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$ values for 3-way similarity search.

advantageous over the pairwise scheme.

One of the peculiarity of LSH based techniques is that the query complexity exponent $\rho < 1$ is dependent on the choice of the threshold R_0 we are interested in and the value of c which is the approximation ratio that we will tolerate. Figure 5.1 plots $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$ with respect to c, for selected R_0 values from 0.01 to 0.99. For instance, if we are interested in highly similar pairs, i.e. $R_0 \approx 1$, then we are looking at near $O(\log n)$ query complexity for c-NN problem as $\rho \approx 0$. On the other hand, for very lower threshold R_0, there is not much of hope of time-saving because ρ is close to 1.

5.3.2 Other Efficient k-way Similarities

We refer to the k-way similarities for which there exist sub-linear algorithms for c-NN search, with query and space complexity exactly as given in Theorem 5,

as **efficient** . We have demonstrated existence of one such example of efficient similarity, which is the k-way resemblance. This leads to a natural question: "Are there more of them?".

[20] analyzed all the transformations on similarities that preserve existence of efficient LSH based search. In particular, they showed that if $\mathcal{S}$ is a similarity for which there exists an LSH family, then there also exists an LSH family for any similarity which is a *probability generating function (PGF)* transformation on $\mathcal{S}$. PGF transformation on $\mathcal{S}$ is defined as $PGF(\mathcal{S}) = \sum_{i=1}^{\infty} p_i \mathcal{S}^i$, where $\mathcal{S} \in [0, 1]$ and $p_i \geq 0$ satisfies $\sum_{i=1}^{\infty} p_i = 1$. Similar theorem can also be shown in the case of 3-way resemblance.

Theorem 6 *Any PGF transformation on* 3*-way resemblance* $\mathcal{R}^{3way}$ *is* efficient. □

Recall in the proof of Theorem 5, we created hash assignments $f_1(\min(\pi(S_3))$ and $f_2(\min(\pi(S_1)), \min(\pi(S_2)))$, which lead to a bucketing scheme for the 3-way resemblance search, where the collision event $E = \{f_1(\min(\pi(S_3)) = f_2(\min(\pi(S_1)), \min(\pi(S_2)))\}$ happens with probability $Pr(E) = \mathcal{R}^{3way}$. To prove the above Theorem 6, we will need to create hash events having probability $PGF(\mathcal{R}^{3way}) = \sum_{i=1}^{\infty} p_i (\mathcal{R}^{3way})^i$. Note that $0 \leq PGF(\mathcal{R}^{3way}) \leq 1$. We will make use of the following simple lemma.

Lemma 1 $(\mathcal{R}^{3way})^n$ *is* efficient *for all* $n \in \mathbb{N}$.

Proof: Define new hash assignments $g_1^n(S_3) = [f_1(h_1(S_3)); ...; f_1(h_n(S_3))]$ and $g_2^n(S_1, S_2) = [f_2(h_1(S_1), h_1(S_2)); ...; f_2(h_n(S_1), h_n(S_2))]$. The collision event $g_1^n(S_3) = g_2^n(S_1, S_2)$ has probability $(\mathcal{R}^{3way})^n$. We now use the pair $< g_1^n, g_2^n >$ instead of $< f_1,$

$f_2 >$ and obtain same guarantees, as in Theorem 5, for $(\mathcal{R}^{3way})^n$ as well. □

Proof of Theorem 6: From Lemma 1, let $< g_1^i, g_2^i >$ be the hash pair corresponding to $(\mathcal{R}^{3way})^i$ as used in above lemma. We sample one hash pair from the set $\{< g_1^i, g_2^i >: i \in \mathbb{N}\}$, where the probability of sampling $< g_1^i, g_2^i >$ is proportional to p_i. Note that $p_i \geq 0$, and satisfies $\sum_{i=1}^{\infty} p_i = 1$, and so the above sampling is valid. It is not difficult to see that the collision of the sampled hash pair has probability exactly $\sum_{i=1}^{\infty} p_i(\mathcal{R}^{3way})^i$. □

Theorem 6 can be naturally extended to k-way similarity for any $k \geq 2$. Thus, we now have infinitely many k-way similarity functions admitting efficient sublinear search. One, that might be interesting, because of its radial basis kernel like nature, is shown in the following corollary.

Corollary 4 $e^{\mathcal{R}^{k-way}-1}$ *is* efficient.

Proof: Use the expansion of $e^{\mathcal{R}^{k-way}}$ normalized by e to see that $e^{\mathcal{R}^{k-way}-1}$ is a PGF on $\mathcal{R}^{k-way}$. □

5.3.3 Fast Algorithms for 3-way c-CP and 3-way c-BC Problems

For 3-way c-CP and 3-way c-BC problems, using bucketing scheme with minwise hashing family will save even more computations.

Theorem 7 *For $\mathcal{R}^{3way}$ c-Close Pair Problem (or c-CP) one can construct a data structure with $O(n^{2\rho} \log_{1/cR_0} n)$ query time and $O(n^{1+2\rho})$ space, where $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$.* □

Note that we can switch the role of f_1 and f_2 in the proof of Theorem 5. We are thus left with a c-NN problem with search space $O(n^2)$ (all pairs) instead of n. A bit of analysis, similar to Theorem 5, will show that this procedure achieves the required query time $O(n^{2\rho} \log_{1/cR_0} n)$, but uses a lot more space, $O(n^{2(1+\rho)})$, than shown in the above theorem. It turns out that there is a better way of doing c-CP that saves us space.

Proof of Theorem 7: We again start with constructing hash tables. For every element $S_c \in C$, we create a hash-table and store S_c in bucket $B(S_c) = [h_1(S_c); h_2(S_c); ...; h_K(S_c)]$, where h_i is chosen uniformly from minwise independent family of hash functions $\mathcal{H}$. We create L such hash-tables. For a query element S_q we look for all pairs in bucket $B(S_q) = [h_1(S_q); h_2(S_q); ...; h_K(S_q)]$ and repeat this for each of the L tables. Note, we do not form pairs of elements retrieved from different tables as they do not satisfy Equation 2.4. If there exist a pair $S_1, S_2 \in C$ with $Sim(S_q, S_1, S_2) \geq R_0$, using Equation 2.4, we can see that we will find that pair in bucket $B(S_q)$ with probability $1 - (1 - R_0^K)^L$. Here, we cannot use traditional choice of K and L, similar to what we did in Theorem 5, as there are $O(n^2)$ instead of $O(n)$ possible pairs. We instead use $K = \lceil \frac{2 \log n}{\log \frac{1}{cR_0}} \rceil$ and $L = \lceil n^{2\rho} \log(\frac{1}{\delta}) \rceil$, with $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$. With this choice of K and L, the result follows. Note, the process is stopped as soon as we find pairs S_1 and S_2 with $Sim(S_q, S_1, S_2) \geq cR_0$. The key argument that saves space from $O(n^{2(1+\rho)})$ to $O(n^{1+2\rho})$ is that we hash n points individually. Eq. (2.4) makes it clear that hashing all possible pairs is not needed when every point can be processed individually, and pairs formed within each bucket itself filter out most of the unnecessary combinations. □

Theorem 8 *For $\mathcal{R}^{3way}$ c-Best Cluster Problem (or c-BC) there exist an algorithm with*

running time $O(n^{1+2\rho}\log_{1/cR_0} n)$, *where* $\rho = 1 - \frac{\log 1/c}{\log 1/c + \log 1/R_0}$. □

The argument similar to one used in proof of Theorem 7 leads to the running time of $O(n^{1+3\rho}\log_{1/cR_0} n)$ as we need $L = O(n^{3\rho})$, and we have to processes all points at least once.

Proof of Theorem 8: Repeat c-CP problem n times for every element in collection C acting as query once. We use the same set of hash tables and hash functions every time. The preprocessing time is $O(n^{1+2\rho}\log_{1/cR_0} n)$ evaluations of hash functions and the total querying time is $O(n \times n^{2\rho}\log_{1/cR_0} n)$, which makes the total running time $O(n^{1+2\rho}\log_{1/cR_0} n)$. □

For k-way c-BC Problem, we can achieve $O(n^{1+(k-1)\rho}\log_{1/cR_0} n)$ running time. If we are interested in very high similarity cluster, with $R_0 \approx 1$, then $\rho \approx 0$, and the running time is around $O(n \log n)$. This is a huge saving over the brute force $O(n^k)$. In most practical cases, specially in big data regime where we have enormous amount of data, we can expect the k-way similarity of good clusters to be high and finding them should be efficient. We can see that with increasing k, hashing techniques save more computations.

5.4 Practical Implementation

For theoretical analysis just existence of hash functions guaranteeing the required collision probability suffices, and we can assume bucket probing as a constant time operation. There are two major practical concerns that arise when constructing tables using minwise hashes: 1) The range of hash values can

be potentially huge. Moreover, we need to concatenate K of these signatures, which makes construction of constant time lookup tables impossible. 2) The hash functions do not map uniformly into the bucket space leading to many buckets being empty. For the pairwise case, both of these issues were handled by employing universal hashing scheme to remap the theoretical buckets uniformly to manageable size addresses space, see [32]. The basic idea being trading practicality with a very small random collision probability. We will follow the same route.

Suppose for 3-way c-NN problem, we want to create hash tables of size $\mathbb{S}$ for lookup. While preprocessing, the theoretical bucket $g_1(S_3) = [f_1(h_1(S_3)); ...; f_1(h_k(S_3))]$, corresponding to an element $S_3 \in C$, is associated with the actual bucket

$$B(S_3) = ((\sum_{i=1}^{i=k} r_i \times f_1(h_i(S_3)) \text{mod } P) \text{mod } \mathbb{S}),$$

where r_is are random integers and P is any prime number sufficiently larger than $\mathbb{S}$ to ensure uniformness. After we have preprocessed all the elements of C, during the query stage, for query elements S_1 and S_2, we retrieve from hash tables only the elements stored at the location (or index)

$$B(S_1, S_2) = ((\sum_{i=1}^{i=k} r_i \times f_2(h_i(S_1), h_i(S_2)) \text{mod } P) \text{mod } \mathbb{S})$$

This scheme incurs an addition $\frac{1}{\mathbb{S}}$ probability of retrieving a random point. This probability is negligible and there are more ways to reduce this probability, for example, use another hash function for chaining, see [6].

In case when $f_1(h_i(S_3))$ is very large, which is possible for k-way resemblance with large k, we can rehash $f_1(h_i(S_3))$ itself to a manageable size and then create buckets, an idea very similar to recently proposed b-bit minwise hashing [78].

5.5 Computation Savings in Practice

In this section, we will demonstrate the savings in computation obtained via the proposed bucketing (Theorem 5) scheme in the context of 3-way resemblance search. We chose the same query candidates generated in section 5.1.1. From each triplet (x_1, x_2, x_3), we use the 1st and the 2nd data-point x_1 and x_2 as the 3-way c-NN query.

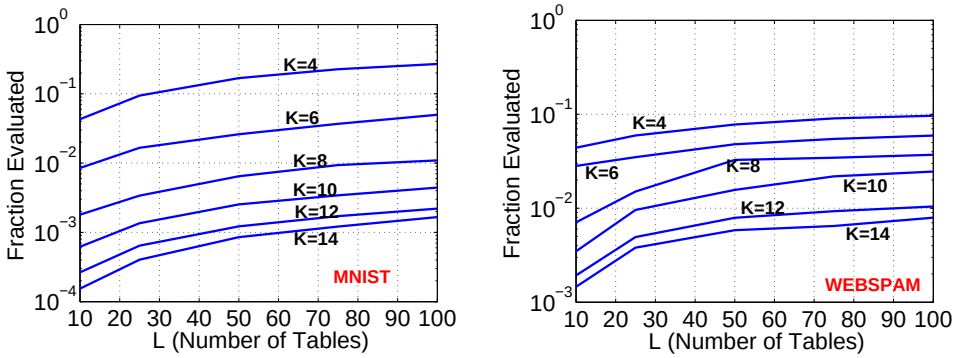

Figure 5.2: Fraction of the total points retrieved by the bucketing scheme for fast 3-way resemblance search with various K and L

Traditional evaluations of bucketing scheme need a similarity threshold R_0 as input. The values of parameters K and L is then determined on the basis of this R_0. We follow a more rigorous evaluation. We implement the bucketing procedure, for 3-way c-NN search, exactly as described in Section 5.4. We chose a fixed table-size of $\mathbb{S} = 2^{24}$, i.e., 24 bits addresses. We evaluate the bucketing procedure over a set of values of K and L. In particular, we use the following choices: $K \in \{4, 6, 8, 10, 12, 14\}$ and $L \in \{10, 20, 50, 75, 100\}$. For each combination of K and L, we compute the total number of points retrieved by the bucketing scheme. Figure 5.2 summarizes the plot of the ratio of total number of points scanned by the bucketing scheme to the total number of points that would be

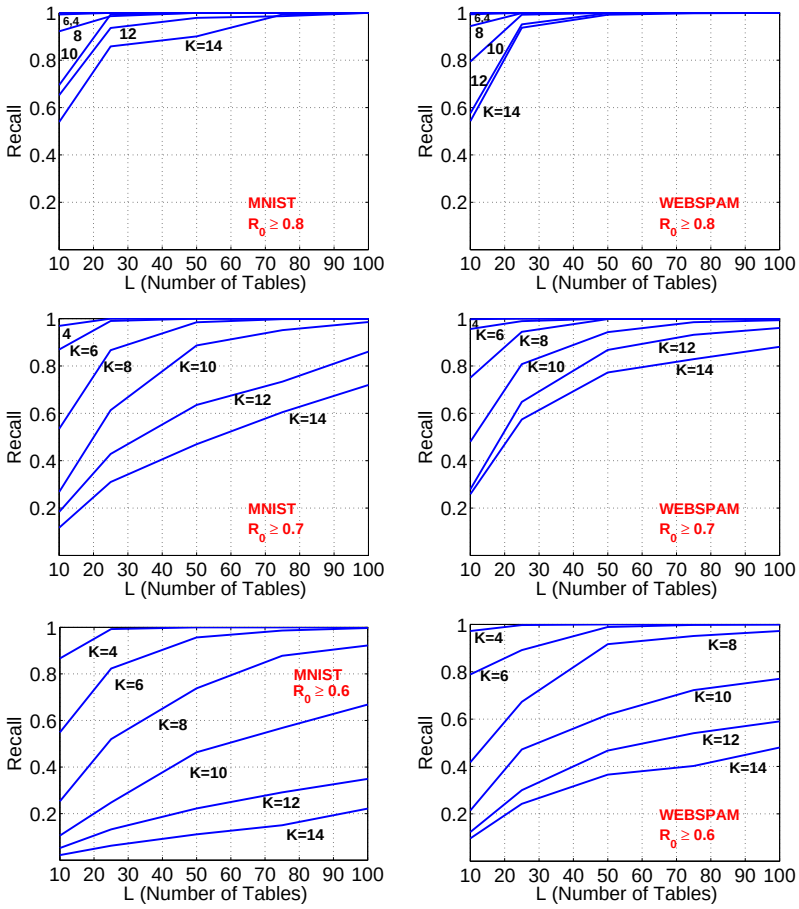

Figure 5.3: Recall of the points with 3-way resemblance greater than R_0 for various combinations of parameters K and L

required by a brute force linear scan. The y-axis is on the log scale.

Figure 5.2 gives a nice overview of how the values of parameters K and L affect the number of points retrieved by the bucketing scheme. It should be noted that the number of points retrieved also depends on the similarity distribution of data. For example, if the whole data are just duplicates, of the given query, then any hashing scheme will retrieve all the elements. The speedup obtained is very much dependent on the similarity value distribution of data, and it is usually much better than the worst case theoretical guarantees.

To evaluate the accuracy of this bucketing scheme, we report the recall of all the points with similarity greater than R_0. In particular, we chose $R_0 \in \{0.8, 0.7, 0.6\}$. Figure 5.3 plots the recall, given a value of R_0, over all choices of K and L for both the datasets.

Few things that we can observe in the plots is that the recall values goes down with similarity level R_0, which is what one should expect from any probabilistic bucketing scheme. To get an estimate of how much computation we save, for simplicity, let us assume that we are interested in almost perfect recall. For MNIST data if we are interested in retrieving all 3-way neighbors with $R_0 \geq 0.8$, then a good combination is $K = 14$ and $L = 75$. This combination evaluates on an average of 72 points per query. For $R_0 \geq 0.7$, perfect recall can be achieved with $K = 10$ and $L = 100$ which scans around 265 points per query. While for $R_0 \geq 0.6$, we need $K = 6$ and $L = 75$, and that scans around 2200 points per query. Compare this with brute force way of 60000 points per query. This is a massive saving. As pointed out before, the savings are more if our interest is in retrieving very similar neighbors.

In the case of WEBSPAM dataset, achieving almost perfect recall for $R_0 \geq 0.8$ can be done using $K = 14$ and $L = 50$. This combination scans an average of around 205 points per query. For $R_0 \geq 0.7$, the average points retrieved per query is 762 with $K = 10$ and $L = 75$. Perfect recall for $R_0 \geq 0.6$ with $K = 6$ and $L = 50$ scans around 1678 points per query. The brute force way is scanning all of the 35000 points. The savings are even more prominent when the total number of data points, i.e. n, is large.

In practice, especially in the "Big Data" regime, we will be mostly interested in high similarity thresholds. And therefore, we expect this scheme to be much

more beneficial. For simplicity, we restricted our arguments to the case where we are interested in perfect recall. In most practical cases, we are interested only in retrieving few elements from dataset with high similarity, which should be much more efficient. These are some of the many crucial reasons why probabilistic bucketing scheme is popular for pairwise similarity search.

5.6 Discussions

We believe that we have presented the first evidence of usefulness of k-way similarity search and also demonstrated its feasibility in practice. While the work presented in this Chapter is promising for efficient 3-way and k-way similarity search in binary high-dimensional data, there are numerous interesting and useful research directions. We mention two important such examples.

Limited Randomness and k-way Hashing: A lot is known about the effect of limited randomness on minwise hashing for pairwise estimation and retrieval [40, 41, 71, 62]. Studying the effect of limited randomness for k-way search and estimation is wide open. The analysis may be combinatorially interesting and challenging.

k-way sign random projections: It would be very useful to develop theory for k-way sign random projections for non-binary data. Under usual (real-valued) random projections, it is known that the volume (which is related to the determinant) is approximately preserved [66, 48]. We speculate that the collision probability of k-way sign random projections might be also a (monotonic) function of the determinant. In case the collision probability is monotonic, the ideas presented to create buckets for the k-way cases are directly applicable.

CHAPTER 6

AN ORDER OF MAGNITUDE FASTER MINHASH

MinHash is a decade old algorithm which has made innumerable applications over the web faster. We have further shown in chapter 4 that MinHash is arguably the best hash function for the web by showing its superiority over another very popular hashing scheme SimHash. In chapter 5, we show that the properties of MinHash go beyond pairwise similarity, and we can use MinHash signatures for efficient querying with k-way similarity measures, which is a very promising direction in many real applications.

The query complexity of MinHash based similarity search is dominated by the number of hash evaluations, and this number grows with the data size [42]. For large scale learning with kernel feature for resemblance kernel, we need multiple (hundreds to thousands) MinHash signatures. As discussed in Chapter 3, the bottleneck cost is feature generation because it requires computation of multiple hashes. This is true with any application using MinHash. In industrial applications such as search where the data are often high-dimensional and binary (e.g., text n-grams), *minwise hashing* is widely adopted. Traditional scheme for computing MinHash requires computing hundreds or thousands of MinHash signatures of the data. This multiple hash computation is the bottleneck step in many applications, which is costly both in terms of computation and memory. We argue that this cost in the decade old algorithm is unnecessary.

In this chapter, we propose an order of magnitude faster variant of minwise hashing (MinHash) which generates all the necessary hash evaluations (for similarity search or linear learning), using one single permutation and one pass over the data. The heart of the proposed hash function is a "rotation" scheme

which densifies the sparse sketches of *one permutation hashing* [60] in an unbiased fashion thereby achieving the LSH property. This LSH property is required for hash table construction and kernel feature generation. This idea of rotation presented in this chapter could be of independent theoretical interest for densifying other types of sparse sketches in an unbiased manner.

With the new hashing method, the query time of a (K, L)-parameterized LSH (Section 2.3) is reduced from the typical $O(dKL)$ complexity to merely $O(KL+dL)$, where d is the number of nonzeros of the data vector, K is the number of hashes in each hash table, and L is the number of hash tables. This is an algorithmic improvement over existing (K, L)-parameterized LSH for resemblance similarity search which has innumerable applications. The proposed hashing scheme also leads to linear estimator for resemblance. Linear estimator leads to kernel features. Thus speedup in hash computation automatically implies fast kernel features for learning with resemblance kernel (See Chapter 3 for details).

6.1 Problems with Classical Minwise Hashing Scheme

Minwise hashing is one of the widely adopted hash function that is used to construct index structures for resemblance based similarity search. In hashing based methods, the total number of hash evaluations needed to perform sublinear similarity search is $O(n^{\rho} log n)$ [42] per query, which in practice can run into hundreds and even in thousands. Thus, processing a single query with minwise hashing requires to store and process hundreds or even thousands of giant permutations of size equal to the dimensionality of the datasets D which in practice can go to $D = 2^{64}$. Querying with minwise hashing, even though parallelizable [62], is computational expensive and hence not energy efficient.

Why not Conditional Random Sampling Sketches ? The idea of *Conditional Random Sampling (CRS)* sketches, that generates multiple hashes (or sketches) from a single permutation, has been there in literature for a while [12, 51, 52, 53], also referred to as "bottom-k Sketches" [21]. It turns out that CRS sketches are only suitable for estimation of similarity, and they cannot be used to construct hash buckets for fast similarity search. Minwise hashing satisfies the **LSH property** of having the "collision probability" equal to the similarity value. This property makes minwise hashing an ideal candidate for generating index structures [42]. CRS sketches do not satisfy this LSH property and hence are unsuitable for LSH based table construction. See Appendix B for more details and experimental comparisons.

6.2 One Permutation Hashing

The idea of binning has proven quite powerful in streaming applications for generating efficient data sketches [17, 22]. It involves partitioning the data vector into multiple bins, and then storing appropriate summary statistics computed over each of the bins. Based on this idea, [60] developed *one permutation hashing*. As illustrated in Figure 6.1, the method breaks the space equally into k bins after one permutation and stores the smallest nonzero of each bin. In this way, the samples are naturally aligned. Note that if $k = D$, where $D = |\Omega|$, then we get back the (permuted) original data matrix. One of the major concerns with one permutation hashing is the occurrence of empty bins.

For example, in Figure 6.1, $\pi(S_1)$ and $\pi(S_2)$ denote the state of the binary vectors S_1 and S_2 after applying permutation π. These shuffled vectors are then

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(\Omega)$	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15	16 17 18 19	20 21 22 23
	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

Figure 6.1: **One Permutation Hashing**. Instead of only storing the smallest nonzero in each permutation and repeating the permutation k times, we can use just one permutation, break the space evenly into k bins, and store the smallest nonzero in each bin.

divided into 6 bins of length 4 each. We start the numbering from 0. We look into each bin and store the corresponding minimum non-zero index. For bins not containing any non-zeros, we use a special symbol "E" to denote empty bins. We also denote

$$M_j(\pi(S)) = \left\{\pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k}\right)\right\} \tag{6.1}$$

We assume for the rest of the chapter that D is divisible by k, otherwise we can always pad extra dummy features. We define $\underset{j}{OPH}$ ("OPH" for one permutation hashing) as

$$\underset{j}{OPH}(\pi(S)) = \begin{cases} \text{"}E\text{"}, & \text{if } \pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k}\right) = \phi \\ M_j(\pi(S)) \mod \frac{D}{k}, & \text{otherwise} \end{cases} \tag{6.2}$$

i.e., $\underset{j}{OPH}(\pi(S))$ denotes the minimum value in Bin j, under permutation mapping π, as shown in the example in Figure 6.1. If this intersection is null, i.e., $\pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k}\right) = \phi$, then $\underset{j}{OPH}(\pi(S)) =$ "E".

One permutation hashing generate many values, including the empty value 'E', of the data using a single permutation. It was shown that one permutation, was enough to estimate the resemblance with lesser variance than the original minwise hashing. Although, it turns out that it is not suitable for the task of similarity search because of the empty bins.

6.2.1 The Problem of Empty Bins

Consider the events of "simultaneously empty bin" $I^j_{emp} = 1$ and "simultaneously non-empty bin" $I^j_{emp} = 0$, between given vectors S_1 and S_2, defined as:

$$I^j_{emp} = \begin{cases} 1, & \text{if } \underset{j}{OPH}(\pi(S_1)) = \underset{j}{OPH}(\pi(S_2)) = E \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

Simultaneously empty bins are only defined with respect to two sets S_1 and S_2. In Figure 6.1, $I^0_{emp} = 1$ and $I^2_{emp} = 1$, while $I^1_{emp} = I^3_{emp} = I^4_{emp} = I^5_{emp} = 0$. Bin 5 is only empty for S_2 and not for S_1, so $I^5_{emp} = 0$.

Given a bin number j, if it is not simultaneously empty ($I^j_{emp} = 0$) for both the vectors S_1 and S_2, [60] showed

$$Pr\left(\underset{j}{OPH}(\pi(S_1)) = \underset{j}{OPH}(\pi(S_2)) \middle| I^j_{emp} = 0\right) = R. \tag{6.4}$$

On the other hand, when $I^j_{emp} = 1$, no such guarantee exists. When $I^j_{emp} = 1$ collision does not have enough information about the similarity R. Since the event $I^j_{emp} = 1$ can only be determined given the two vectors S_1 and S_2 and the materialization of π, one permutation hashing cannot be directly used for indexing, especially when the data are very sparse. In particular, $\underset{j}{OPH}(\pi(S))$

does not lead to a valid LSH hash function because of the coupled event $I^j_{emp} = 1$ in (6.4). The simple strategy of ignoring empty bins leads to biased estimators of resemblance and shows poor performance [81]. Because of this same reason, one permutation hashing cannot be directly used to extract random features for linear learning with resemblance kernel.

6.3 Densifying One Permutation Hashing

We propose a "rotation" scheme that assigns new values to all the empty bins, generated from one permutation hashing, in an unbiased fashion. The rotation scheme for filling the empty bins from Figure 6.1 is shown in Figure 6.2. The idea is that for every empty bin, the scheme borrows the value of the closest non-empty bin in the clockwise direction (circular right hand side) added with offset C.

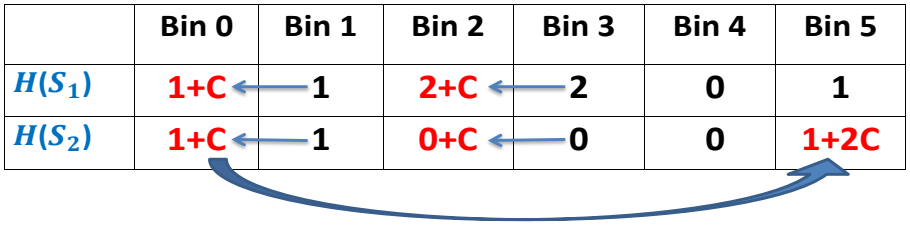

Figure 6.2: Densification by "rotation" for filling empty bins generated from one permutation hashing [81]. Every empty bin is assigned the value of the closest non-empty bin, towards right (circular), with an offset C. For the configuration shown in Figure 6.1, the above figure shows the new assigned values (in red) of empty bins after densification.

Our proposed hashing method is simple and effective. After one permutation hashing, we collect the hashed values for each set. If a bin is empty, we

"borrow" the hashed value from the first non-empty bin on the right (circular). Due to the circular operation, the scheme for filling empty bins appears like a "rotation". We can show mathematically that we obtain a valid LSH scheme. Because we generate all necessary hash values from merely one permutation, the query time of a (K, L)-parameterized LSH scheme is reduced from $O(dKL)$ to $O(KL + dL)$, where d is the number of nonzeros of the query vector. We will later show empirically that its performance in near neighbor search is virtually identical to that of the original MinHash.

Given the configuration in Figure 6.1, for Bin 2 corresponding to S_1, we borrow the value 2 from Bin 3 along with an additional offset of C. Interesting is the case of Bin 5 for S_2, the circular right is Bin 0 which was empty. Bin 0 borrows from Bin 1 acquiring value $1 + C$, Bin 5 borrows this value with another offset C. The value of Bin 5 finally becomes $1 + 2C$. The value of $C = \frac{D}{k} + 1$ enforces proper alignment and ensures no unexpected collisions. Without this offset C, Bin 5 which was not simultaneously empty, after reassignment, will have value 1 for both S_1 and S_2. This is an error as initially there was no collision (note $I^5_{emp} = 0$). Multiplication by the distance of the non-empty bin, from where the value was borrowed, ensures that the new values of simultaneous empty bins ($I^j_{emp} = 1$), at any location j for S_1 and S_2, never match if their new values come from different bin numbers.

Formally the hashing scheme with "rotation", denoted by $\mathcal{H}$, is defined as:

$$\mathcal{H}_j(S) = \begin{cases} \underset{j}{OPH}(\pi(S)) & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ \\ \underset{(j+t) \bmod k}{OPH}(\pi(S)) + tC & \text{otherwise} \end{cases} \tag{6.5}$$

$$t = \min z, \quad s.t. \quad \underset{(j+z) \bmod k}{OPH}(\pi(S)) \neq E \tag{6.6}$$

Here $C = \frac{D}{k} + 1$ is a constant.

This densification scheme ensures that whenever $I^j_{emp} = 0$, i.e., Bin j is simultaneously empty for any two S_1 and S_2 under considerations, the newly assigned value mimics the collision probability of the nearest simultaneously non-empty bin towards right (circular) hand side making the final collision probability equal to R, irrespective of whether $I^j_{emp} = 0$ or $I^j_{emp} = 1$.

Theorem 9

$$\mathbf{Pr}\left(\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\right) = R \tag{6.7}$$

Proof: See Appendix A.0.2 □

Theorem 9 proves that our proposed method provides a valid hash function which satisfies the LSH property from one permutation. While the main application of our method is for approximate neighbor search, which we will elaborate in Section 6.3.3, another promising use of our work is for estimating resemblance using only a linear estimator (i.e., an estimator which is an inner product); see Section 6.3.1. Generating KL different hash values of $\mathcal{H}$ only requires $O(d + KL)$, which saves a factor of d in the query processing cost compared to the cost of $O(dKL)$ with traditional minwise hashing. For fast linear

learning [65] with k different hash values the new scheme only needs $O(d + k)$ testing (or prediction) time compared to standard b-bit minwise hashing which requires $O(dk)$ time for testing.

6.3.1 Resemblance Estimation

Theorem 9 naturally suggests a linear, unbiased estimator of the resemblance R:

$$\hat{R} = \frac{1}{k} \sum_{j=0}^{k-1} 1\left\{\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2))\right\} \tag{6.8}$$

Linear estimator is important because it implies that the hashed data form an inner product space, which allows us to take advantage of the modern linear learning algorithms [45, 77, 10, 29] such as linear SVM.

The original one permutation hashing paper [60] proved the following unbiased estimator of R

$$\hat{R}_{OPH,ub} = \frac{N_{mat}}{k - N_{emp}} \tag{6.9}$$

$$N_{emp} = \sum_{j=0}^{k-1} 1\left\{\underset{j}{OPH}(\pi(S_1)) = E \text{ and } \underset{j}{OPH}(\pi(S_2)) = E\right\}$$

$$N_{mat} = \sum_{j=0}^{k-1} 1\left\{\underset{j}{OPH}(\pi(S_1)) = \underset{j}{OPH}(\pi(S_2)) \neq E\right\}$$

which unfortunately is not a linear estimator because the number of "jointly empty" bins N_{emp} would not be known until we see both hashed sets. To address

this issue, [60] provided a modified estimator

$$\hat{R}_{OPH} = \frac{N_{mat}}{\sqrt{k - N_{emp}^{(1)}}\sqrt{k - N_{emp}^{(2)}}} \tag{6.10}$$

$$N_{emp}^{(1)} = \sum_{j=0}^{k-1} 1\left\{ \underset{j}{OPH}(\pi(S_1)) = E \right\},$$

$$N_{emp}^{(2)} = \sum_{j=0}^{k-1} 1\left\{ \underset{j}{OPH}(\pi(S_2)) = E \right\}$$

This estimator, is not unbiased (for estimating R). It is easy to see that as k increases (to D), $\hat{R}_{OPH}$ estimates the original (normalized) inner product, not resemblance. From the perspective of applications (in linear learning), this is not necessarily a bad choice, of course.

Here, we provide an experimental study to compare the two estimators, $\hat{R}$ in (6.8) and $\hat{R}_{OPH}$ in (6.10). The dataset, extracted from a chunk of Web crawl (with 2^{16} documents), is described in Table 6.2, which consists of 12 pairs of sets (i.e., total 24 words). Each set consists of the document IDs which contain the word.

Figure 6.3 and Figure 6.4 summarizes the estimation accuracies in terms of the bias and mean square error (MSE). For $\hat{R}$, bias = $E\left(\hat{R} - R\right)$ and MSE = $E\left(\hat{R} - R\right)^2$. The definitions for $\hat{R}_{OPH}$ is analogous. The results confirm that our proposed hash method leads to an unbiased estimator regardless of the data sparsity or number of bins k. The estimator in the one permutation hashing [60] can be severely biased when k is too large (i.e., when there are many empty bins). The MSEs suggest that the variance of $\hat{R}$ essentially follows $\frac{R(1-R)}{k}$, which is the variance of the original minwise hashing estimator [55], unless k is too large. But even when the MSEs deviate from $\frac{R(1-R)}{k}$, they are not large, unlike $\hat{R}_{OPH}$.

This experimental study confirms that our proposed hash method works

Table 6.1: 12 pairs of words vectors used for compare MSE.

Word 1	Word 2	$\|S_1\|$	$\|S_2\|$	R
HONG	KONG	940	948	0.925
RIGHTS	RESERVED	12234	11272	0.877
A	THE	39063	42754	0.644
UNITED	STATES	4079	3981	0.591
SAN	FRANCISCO	3194	1651	0.456
CREDIT	CARD	2999	2697	0.285
TOP	BUSINESS	9151	8284	0.163
SEARCH	ENGINE	14029	2708	0.152
TIME	JOB	12386	3263	0.128
LOW	PAY	2936	2828	0.112
SCHOOL	DISTRICT	4555	1471	0.087
REVIEW	PAPER	3197	1944	0.078

well for resemblance estimation and hence it is useful for, e.g., training resemblance kernel SVM using a linear algorithm.

6.3.2 Variance Analysis

We provide the variance analysis of the existing scheme. Theorem 9 leads to an unbiased estimator of R between S_1 and S_2 defined as:

$$\hat{R} = \frac{1}{k} \sum_{j=0}^{k-1} \mathbf{1}\{\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\}. \tag{6.11}$$

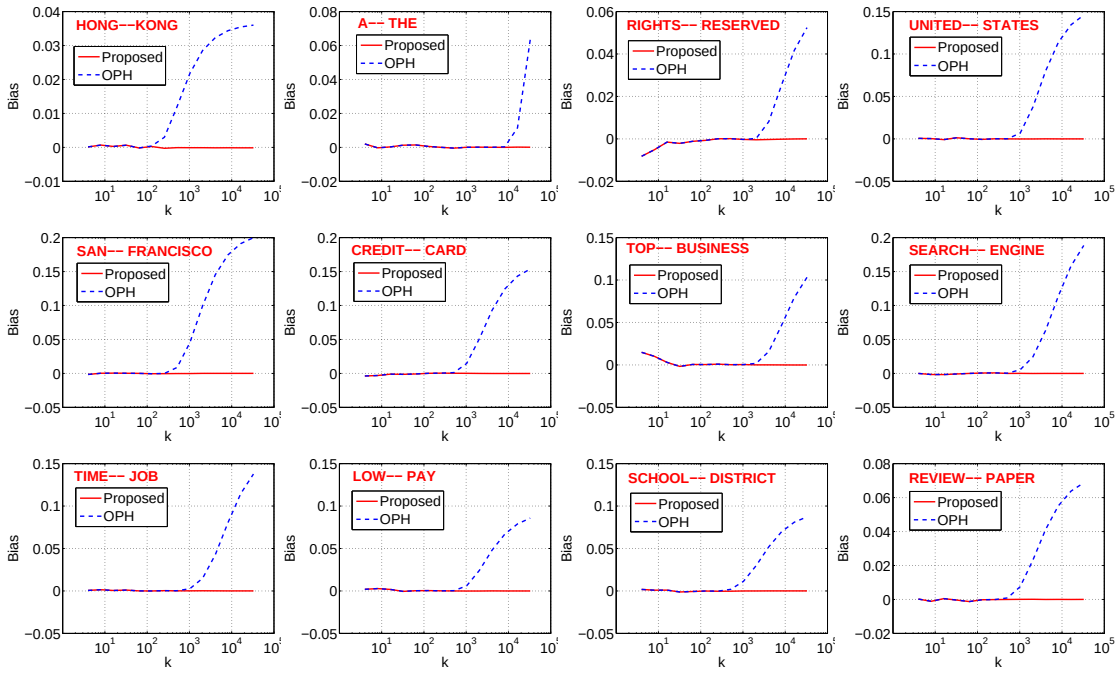

Figure 6.3: **Bias in resemblance estimation**. The plots are the biases of the proposed estimator $\hat{R}$ defined in (6.8) and the previous estimator $\hat{R}_{OPH}$ defined in (6.10) with respect to number of bins k

Denote the number of simultaneously empty bins by

$$N_{emp} = \sum_{j=0}^{k-1} \mathbf{1}\{I_{emp}^{j} = 1\}, \tag{6.12}$$

where $\mathbf{1}$ is the indicator function. We partition the event $\left(\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\right)$ into two cases depending on I_{emp}^{j}. Let M_j^N (**Non-empty Match at** j) and M_j^E (**Empty Match at** j) be the events defined as:

$$M_j^N = \mathbf{1}\{I_{emp}^{j} = 0 \text{ and } \mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\} \tag{6.13}$$

$$M_j^E = \mathbf{1}\{I_{emp}^{j} = 1 \text{ and } \mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\} \tag{6.14}$$

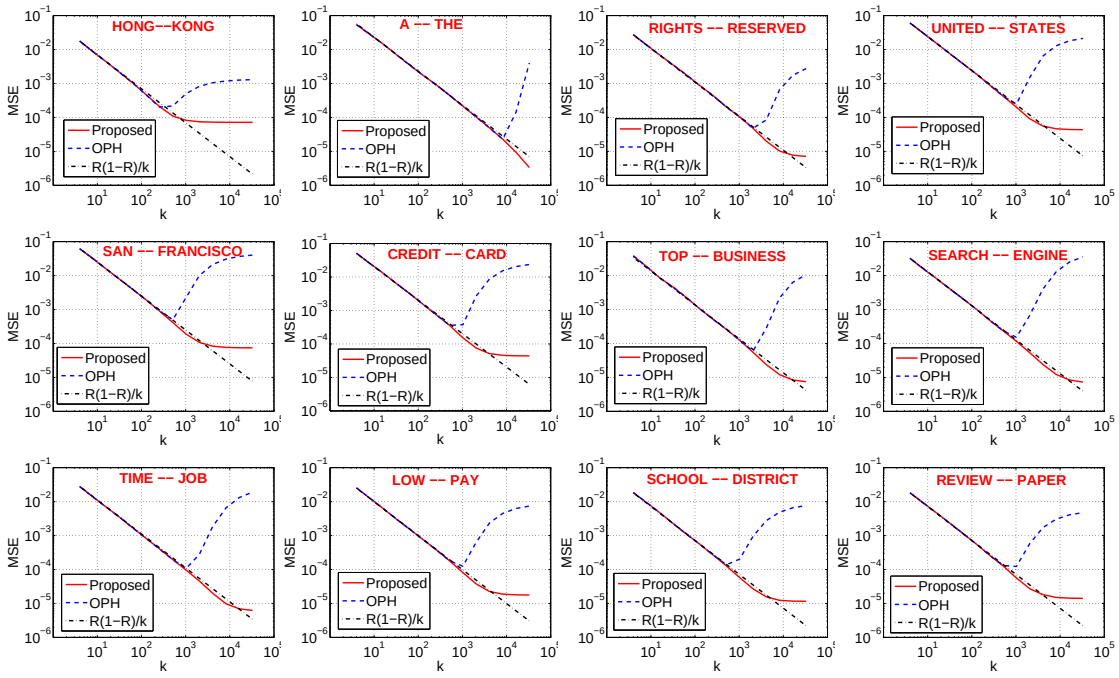

Figure 6.4: **MSE in resemblance estimation**. The MSE of the proposed estimator $\hat{R}$ defined in (6.8) and the previous estimator $\hat{R}_{OPH}$ defined in (6.10) with respect to number of bins k

Note that, $M_j^N = 1 \implies M_j^E = 0$ and $M_j^E = 1 \implies M_j^N = 0$. This combined with Equation 6.4 implies,

$$\begin{aligned} \mathbb{E}(M_j^N | I_{emp}^j = 0) &= \mathbb{E}(M_j^E | I_{emp}^j = 1) \\ &= \mathbb{E}(M_j^E + M_j^N) = R \ \ \forall j \end{aligned} \tag{6.15}$$

It is not difficult to show that,

$$\mathbb{E}\left(M_j^N M_i^N \middle| i \neq j, I_{emp}^j = 0 \text{ and } I_{emp}^i = 0\right) = R\tilde{R},$$

where $\tilde{R} = \frac{a-1}{f1+f2-a-1}$. Using these new events, we have

$$\hat{R} = \frac{1}{k} \sum_{j=0}^{k-1} \left[M_j^E + M_j^N \right] \tag{6.16}$$

We are interested in computing

$$Var(\hat{R}) = \mathbb{E}\left(\left(\frac{1}{k}\sum_{j=0}^{k-1}\left[M_j^E + M_j^N\right]\right)^2\right) - R^2 \tag{6.17}$$

For notational convenience we will use m to denote the event $k - N_{emp} = m$, i.e., the expression $\mathbb{E}(.|m)$ means $\mathbb{E}(.|k - N_{emp} = m)$. To simplify the analysis, we will first compute the conditional expectation

$$f(m) = \mathbb{E}\left(\left(\frac{1}{k}\sum_{j=0}^{k-1}\left[M_j^E + M_j^N\right]\right)^2 \Bigg| m\right) \tag{6.18}$$

By expansion and linearity of expectation, we obtain

$$k^2 f(m) = \mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^N \Big| m\right] + \mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^E \Big| m\right]$$
$$+\mathbb{E}\left[\sum_{i\neq j} M_i^E M_j^E \Big| m\right] + \mathbb{E}\left[\sum_{i=1}^{k}\left[(M_j^N)^2 + (M_j^E)^2\right] \Big| m\right]$$

$M_j^N = (M_j^N)^2$ and $M_j^E = (M_j^E)^2$ as they are indicator functions and can only take values 0 and 1. Hence,

$$\mathbb{E}\left[\sum_{j=0}^{k-1}\left[(M_j^N)^2 + (M_j^E)^2\right] \Big| m\right] = kR \tag{6.19}$$

The values of the remaining three terms are given by the following 3 Lemmas; See the proofs in the Appendix.

Lemma 2

$$\mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^N \Big| m\right] = m(m-1)R\tilde{R} \tag{6.20}$$

Proof: See Appendix A.0.3 □

Lemma 3

$$\mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^E \Big| m\right] = 2m(k-m)\left[\frac{R}{m} + \frac{(m-1)R\tilde{R}}{m}\right] \tag{6.21}$$

Proof: See Appendix A.0.3 □

Lemma 4

$$\mathbb{E}\left[\sum_{i\neq j} M_i^E M_j^E \middle| m\right] = (k-m)(k-m-1) \times \left[\frac{2R}{m+1} + \frac{(m-1)R\tilde{R}}{m+1}\right] \tag{6.22}$$

Proof: See Appendix A.0.3 □

Combining the expressions from the above 3 Lemmas and Eq.(6.19), we can compute $f(m)$. Taking a further expectation over values of m to remove the conditional dependency, the variance of $\hat{R}$ can be shown in the next Theorem.

Theorem 10

$$Var(\hat{R}) = \frac{R}{k} + A\frac{R}{k} + B\frac{R\tilde{R}}{k} - R^2 \tag{6.23}$$
$$A = 2\mathbb{E}\left[\frac{N_{emp}}{k - N_{emp} + 1}\right]$$
$$B = (k+1)\mathbb{E}\left[\frac{k - N_{emp} - 1}{k - N_{emp} + 1}\right]$$

The theoretical values of A and B can be computed using the probability of the event $Pr(N_{emp} = i)$, denoted by P_i, which is given by Theorem 3 in [60].

$$P_i = \sum_{s=0}^{k-i} \frac{(-1)^s k!}{i!s!(k-i-s)!} \prod_{t=0}^{f_1+f_2-a-1} \frac{D\left(1 - \frac{i+s}{k}\right) - t}{D - t}$$

6.3.3 Experimental Comparisons

We implement the classical LSH algorithm described in Section 2.3 using the standard procedures with the following choices of hash functions:

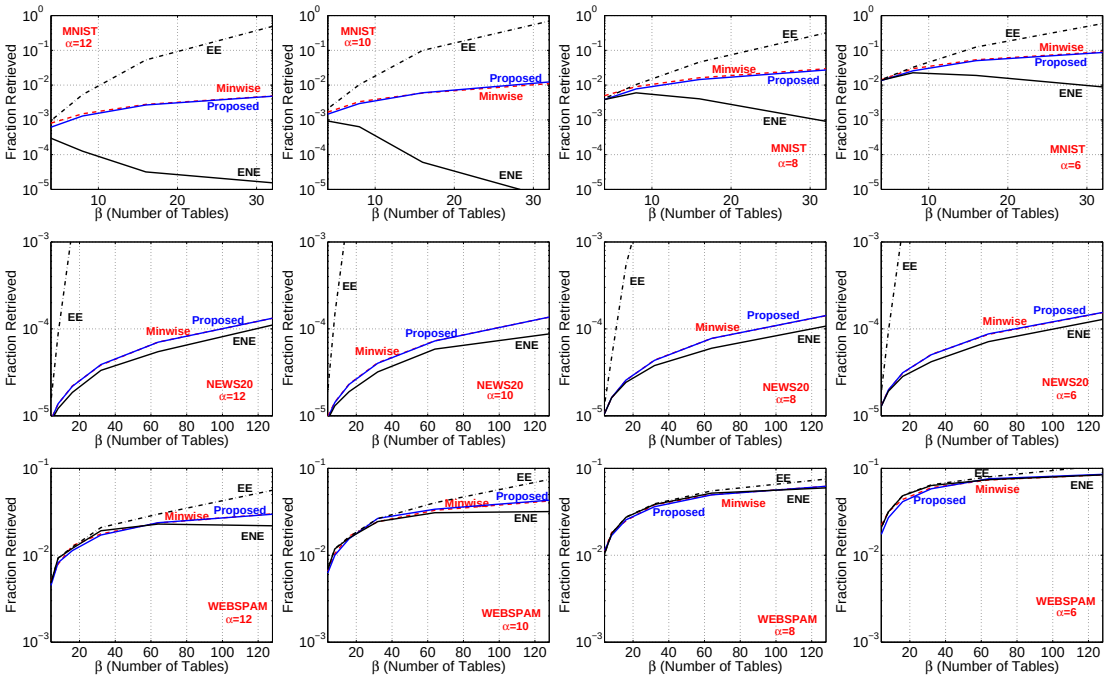

Figure 6.5: Fraction of points retrieved by the bucketing scheme per query corresponding to $K = 10$ shown over different choices of L. Results are averaged over 10 independent runs.

- **Traditional Minwise Hashing** We use the standard minwise hashing scheme $h_i(S) = \min(\pi_i(S))$ which uses $K \times L$ independent permutations.

- **The Proposed Scheme** We use our proposed hashing scheme $\mathcal{H}$ given by (6.27) which uses only one permutation to generate all the $K \times L$ hash evaluations.

- **Empty Equal Scheme (EE)** We use a heuristic way of dealing with empty bins by treating the event of simultaneous empty bins as a match (or hash collision). This can be achieved by assigning a fixed special symbol to all the empty bins. We call this **Empty Equal (EE)** scheme. This can be

formally defined as:

$$h_j^{EE}(S) = \begin{cases} \underset{j}{OPH}(\pi(S)), & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ \text{A fixed special symbol}, & \text{otherwise} \end{cases} \tag{6.24}$$

- **Empty Not Equal Scheme (ENE)** Alternatively, one can also consider the strategy of treating simultaneous empty bins as a mismatch of hash values referred to as **Empty Not Equal (ENE)**. ENE can be reasonably achieved by assigning a new random number to each empty bin independently. The random number will ensure, with high probability, that two empty bins do not match. This leads to the following hash function

$$h_j^{ENE}(S) = \begin{cases} \underset{j}{OPH}(\pi(S)), & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ rand(\text{new seed}), & \text{otherwise} \end{cases} \tag{6.25}$$

Our aim is to compare the performance of minwise hashing with the proposed hash function $\mathcal{H}$. In particular, we would like to evaluate the deviation in performance of $\mathcal{H}$ with respect to the performance of minwise hashing. Since $\mathcal{H}$ has the same collision probability as that of minwise hashing, we expect them to have similar performance. In addition, we would like to study the performance of simple strategies (EE and ENE) on real data.

Datasets

To evaluate the proposed bucketing scheme, we chose the following three publicly available datasets.

- **MNIST** Standard dataset of handwritten digit samples. The feature dimension is 784 with an average number of around 150 non-zeros. We use

the standard partition of MNIST, which consists of 10000 data points in one set and 60000 in the other.

- **NEWS20** Collection of newsgroup documents. The feature dimension is 1,355,191 with 500 non-zeros on an average. We randomly split the dataset in two halves having around 10000 points in each partition.
- **WEBSPAM** Collection of emails documents. The feature dimension is 16,609,143 with about 4000 non-zeros on average. We randomly selected 70000 data points and generated two partitions of 35000 each.

These datasets cover a wide variety in terms of size, sparsity and task. In the above datasets, one partition, the bigger one in case of MNIST, was used for creating hash buckets and the other partition was used as the query set. All datasets were binarized by setting non-zero values to 1.

We perform a rigorous evaluation of these hash functions, by comparing their performances, over a wide range of choices for parameters K and L. In particular, we want to understand if there is a different effect of varying the parameters K and L on the performance of $\mathcal{H}$ as compared to minwise hashing. Given parameters K and L, we need $K \times L$ number of hash evaluations per data point. For minwise hashing, we need $K \times L$ permutations while for the other three schemes we bin the data into $k = K \times L$ bins using only one permutation.

For both WEBSPAM and NEWS20, we implemented all the combinations for $K = \{6, 8, 10, 12\}$ and $L = \{4, 8, 16, 32, 64, 128\}$. For MNIST, with only 784 features, we used $K = \{6, 8, 10, 12\}$ and $L = \{4, 8, 16, 32\}$.

We use two metrics for evaluating retrieval: (i) the fraction of the total number of points retrieved by the bucketing scheme per query, (ii) the recall at a

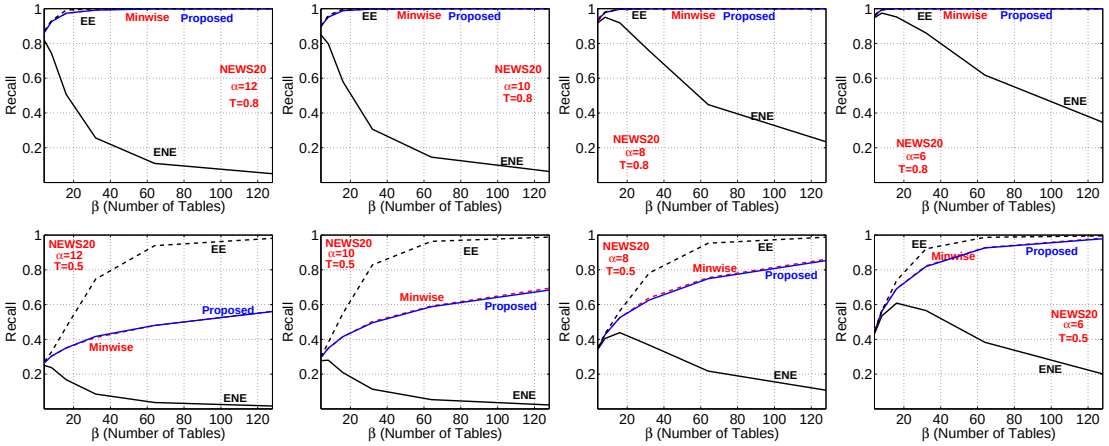

Figure 6.6: *NEWS20*: Recall values of all the points having similarity with the query greater than T, shown over all the combinations.

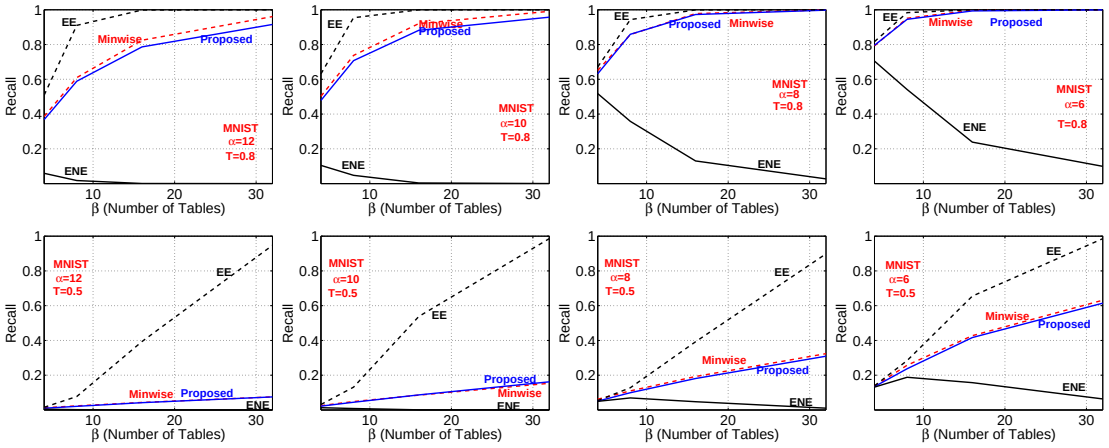

Figure 6.7: *MNIST*: Recall values of all the points having similarity with the query greater than T, shown over all the combinations.

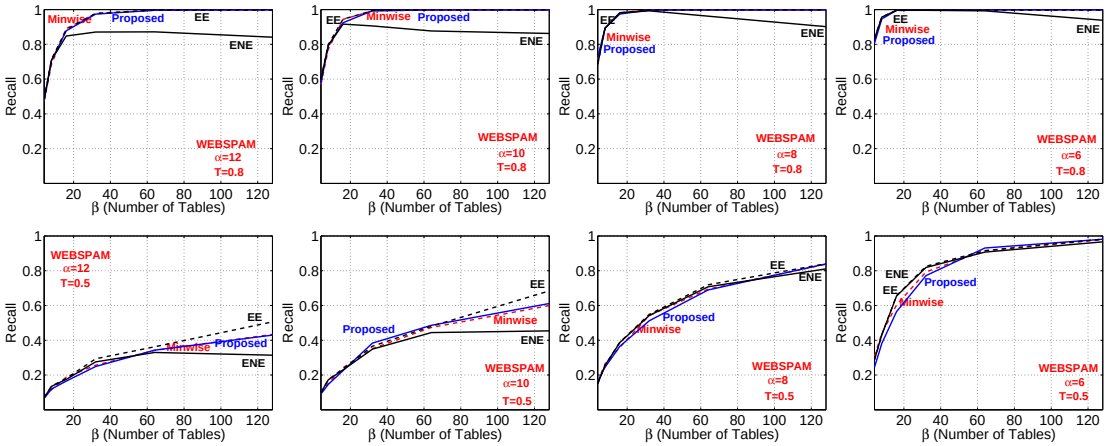

Figure 6.8: *WEBSPAM*: Recall values of all the points having similarity with the query greater than T, shown over all the combinations.

given threshold T_0, defined as the ratio of retrieved elements having similarity, with the query, greater than T_0, to the total number of elements having similarity, with the query, greater than T_0. It is important to balance both of them, for instance in linear scan we retrieve everything, and so we always achieve a perfect recall. For a given choice of K and L, we report both of these metrics independently. For reporting the recall, we choose two values of threshold $T_0 = \{0.5,\ 0.8\}$. Since the implementation involves randomizations, we repeat the experiments for each combination 10 times, and report the average.

Figure 6.5 presents the plots of the fraction of points retrieved per query corresponding to $K = \{6, 8, 10, 12\}$ for all the three datasets with different choices of L. We show the recall plots for various values of L and T_0 corresponding to these K in Figure 6.6, 6.7 and 6.8. One can see that the performance of the proposed hash function $\mathcal{H}$ is indistinguishable from minwise hashing, irrespective of the sparsity of data and the choices of K, L and T_0. Thus, we conclude that minwise hashing can be replaced by $\mathcal{H}$ without loss in the performance and with a huge

gain in query processing time (see Section 6.3.4).

Except for the WEBSPAM dataset, the EE scheme retrieves almost all the points, and so its not surprising that it achieves a perfect recall even at $T_0 = 0.5$. EE scheme treats the event of simultaneous empty bins as a match. The probability of this event is high in general for sparse data, especially for large k, and therefore even non-similar points have significant chance of being retrieved. On the other hand, ENE shows the opposite behavior. Simultaneous empty bins are highly likely even for very similar sparse vectors, but ENE treats this event as a rejection, and therefore we can see that as $k = K \times L$ increases, the recall values starts decreasing even for the case of $T_0 = 0.8$. WEBSPAM has significantly more nonzeros, so the occurrence of empty bins is rare for small k. Even in WEBSPAM, we observe an undesirable deviation in the performance of EE and ENE with $K \geq 10$.

6.3.4 Reduction in Computation Cost

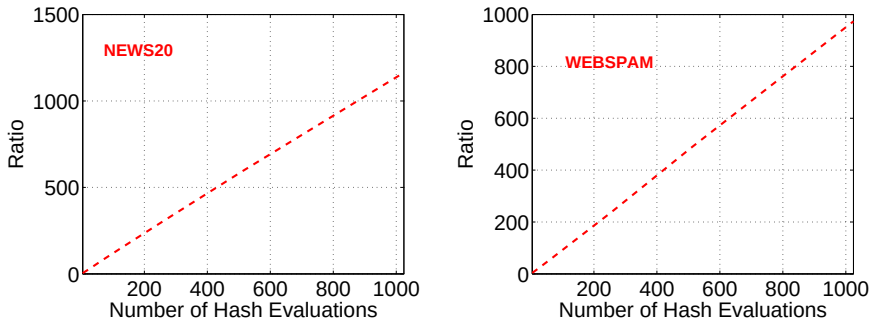

Figure 6.9: Ratio of time taken by minwise hashing to the time taken by our proposed scheme with respect to the number of hash evaluations, on the NEWS20 and WEBSPAM datasets.

Query Processing Cost Reduction

Let d denote the average number of nonzeros in the dataset. For running a (K, L)-parameterized LSH algorithm, we need to generate $K \times L$ hash evaluations of a given query vector [42]. With minwise hashing, this requires storing and processing $K \times L$ different permutations. The total computation cost for simply processing a query is thus $O(dKL)$.

On the other hand, generating $K \times L$ hash evaluations using the proposed hash function $\mathcal{H}$ requires only processing a single permutation. It involves evaluating $K \times L$ minimums with one permutation hashing and one pass over the $K \times L$ bins to reassign empty values via "rotation". Overall, the total query processing cost is $O(d + KL)$. This is a massive saving over minwise hashing.

To verify the above claim, we compute the ratio of the time taken by minwise hashing to the time taken by our proposed scheme $\mathcal{H}$ for NEWS20 and WEBSPAM dataset. We randomly choose 1000 points from each dataset. For every vector, we compute 1024 hash evaluations using minwise hashing and the proposed $\mathcal{H}$. The plot of the ratio with the number of hash evaluations is shown in Figure 6.9. As expected, with the increase in the number of hash evaluations minwise hashing is linearly more costly than our proposal.

Figure 6.9 does not include the time required to generate permutations. Minwise hashing requires storing $K \times L$ random permutation in memory while our proposed hash function $\mathcal{H}$ only needs to store 1 permutation. Each fully random permutation needs a space of size D. Thus with minwise hashing storing $K \times L$ permutations take $O(KLD)$ space which can be huge, given that D in billions is common in practice and can even run into trillions. Approximat-

ing these $K \times L$ permutations using cheap universal hash functions [14, 70, 68] reduces the memory cost but comes with extra computational burden of evaluating $K \times L$ universal hash function for each query. These are not our main concerns as we only need one permutation.

Reduction in Total Query Time

The total query complexity of the LSH algorithm for retrieving approximate near neighbor using minwise hashing is the sum of the query processing cost and the cost incurred in evaluating retrieved candidates. The total running cost of LSH algorithm is dominated by the query processing cost [42, 24]. In theory, the number of data points retrieved using the appropriate choice of LSH parameters (K, L), in the worst case, is $O(L)$ [42]. The total cost of evaluating retrieved candidates in brute force fashion is $O(dL)$. Thus, the total query time of LSH based retrieval with minwise hashing is $O(dKL + dL) = O(dKL)$. Since, both scheme behaves very similarly for any given (K, L), the total query time with the proposed scheme is $O(KL + d + dL)$ which comes down to $O(KL + dL)$. This analysis ignores the effect of correlation in the LSH algorithm but we can see from the plots that the effect is negligible.

The need for evaluating all the retrieved data points based on exact similarity leads to $O(dL)$ cost. In practice, an efficient estimate of similarity can be obtained by using the same hash evaluations used for indexing [78]. This decreases the re-ranking cost further.

Pre-processing Cost Reduction

The LSH algorithm needs a costly pre-processing step which is the bucket assignment for all N points in the collection. The bucket assignment takes in total $O(dNKL)$ with minwise hashing. The proposed hashing scheme $\mathcal{H}$ reduces this cost to $O(NKL + Nd)$.

6.4 A Strictly Better Densification Scheme

The densification scheme shown in the previous section was the first attempt to obtain k hashes with collision probability $\mathcal{R}$ in time O(d +k). There is a sub-optimality in that scheme which hurts performance for very sparse datasets. In this section we provide a strictly better densification scheme.

6.4.1 Intuition for the Improved Scheme

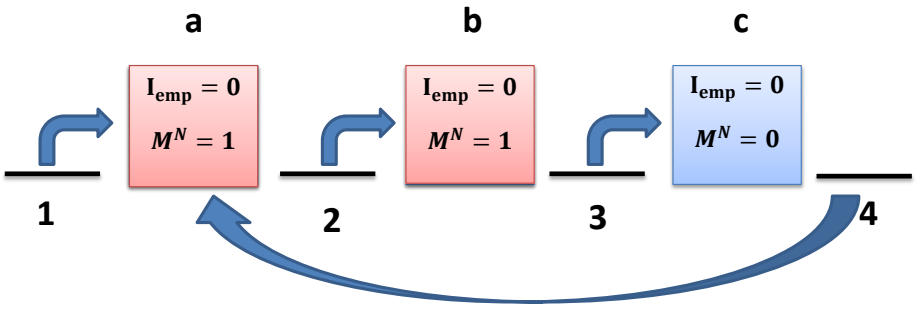

Figure 6.10: Illustration of the old densification scheme [81] from previous section. The randomness is in the position number of these bins which depends on π.

Consider a situation in Figure 6.10, where there are 3 simultaneously non-empty bins ($I_{emp} = 0$) for given S_1 and S_2. The actual position numbers of these simultaneously non-empty bins are random. The simultaneously empty bins ($I_{emp} = 1$) can occur in any order in the 4 blank spaces. The arrows in the figure show the simultaneously non-empty bins which are being picked by the simultaneously empty bins ($I_{emp} = 1$) located in the shown blank spaces. The randomness in the system is in the ordering of simultaneously empty and simultaneously non-empty bins.

Given a simultaneously non-empty Bin t ($I^t_{emp} = 0$), the probability that it is picked by a given simultaneously empty Bin i ($I^i_{emp} = 1$) is exactly $\frac{1}{m}$. This is because the permutation π is perfectly random and given m, any ordering of m simultaneously non-empty bins and $k - m$ simultaneously empty bins are equally likely. Hence, we obtain the term $\left[\frac{R}{m} + \frac{(m-1)R\tilde{R}}{m}\right]$ in Lemma 3.

On the other hand, under the given scheme, the probability that two simultaneously empty bins, i and j, (i.e., $I^i_{emp} = 1$, $I^j_{emp} = 1$), both pick the same simultaneous non-empty Bin t ($I^t_{emp} = 0$) is given by (see proof of Lemma 4)

$$p = \frac{2}{m+1} \tag{6.26}$$

The value of p is high because there is not enough randomness in the selection procedure. Since $R \leq 1$ and $R \leq R\tilde{R}$, if we can reduce this probability p then we reduce the value of $[pR + (1-p)R\tilde{R}]$. This directly reduces the value of $(k-m)(k-m-1)\left[\frac{2R}{m+1} + \frac{(m-1)R\tilde{R}}{m+1}\right]$ as given by Lemma 4. The reduction scales with N_{emp}.

For every simultaneously empty bin, the current scheme uses the information of the closest non-empty bin in the right. Because of the symmetry in the arguments, changing the direction to left instead of right also leads to a valid densification scheme with exactly same variance. This is where we can infuse

randomness without violating the alignment necessary for unbiased densification. We show that randomly switching between left and right provably improves (reduces) the variance by making the sampling procedure of simultaneously non-empty bins more random.

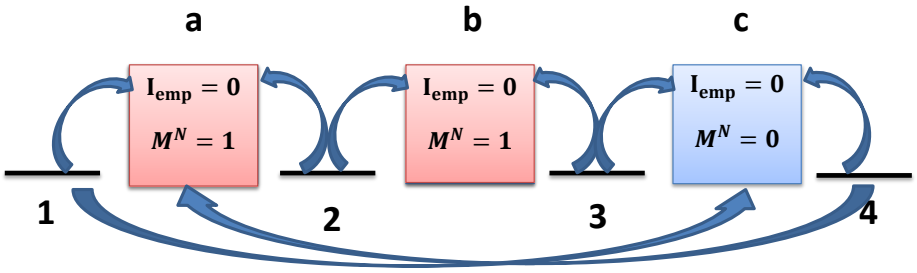

Figure 6.11: Illustration of the improved densification scheme. We infuse more randomness in choosing the directions.

Our proposal is explained in Figure 6.11. Instead of using the value of the closest non-empty bin from the right (circular), we will choose to go either left or right with probability $\frac{1}{2}$. This adds more randomness in the selection procedure.

In the new scheme, we only need to store 1 random bit for each bin, which decides the direction (circular left or circular right) to proceed for finding the closest non-empty bin. The new assignment of the empty bins from Figure 6.1 is shown in Figure 6.12. Every bin number i has an i.i.d. Bernoulli random variable q_i (1 bit) associated with it. If Bin i is empty, we check the value of q_i. If $q_i = 1$, we move circular right to find the closest non-empty bin and use its value. In case when $q = 0$, we move circular left.

For S_1, we have $q_0 = 0$ for empty Bin 0, we therefore move circular left and borrow value from Bin 5 with offset C making the final value $1 + C$. Similarly for

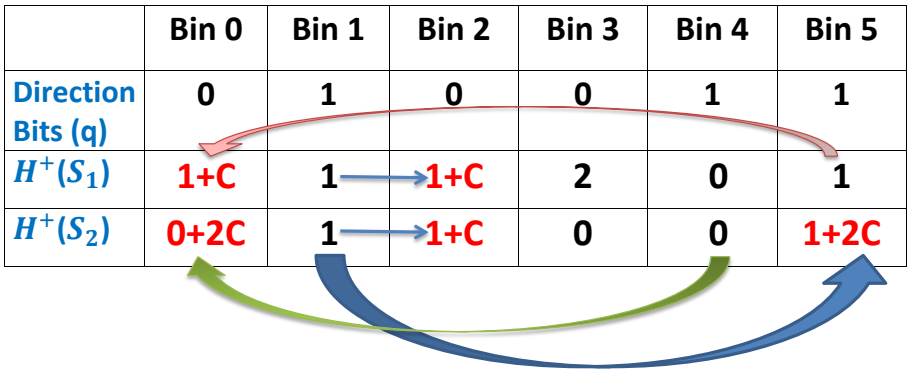

Figure 6.12: Assigned values (in red) of empty bins from Figure 6.1 using the improved densification procedure.

empty Bin 2 we have $q_2 = 0$ and we use the value of Bin 1 (circular left) added with C. For S_2 and Bin 0, we have $q_0 = 0$ and the next circular left bin is Bin 5 which is empty so we continue and borrow value from Bin 4, which is 0, with offset $2C$. A factor of 2 because we traveled 2 bins to locate the first non-empty bin. For Bin 2, again $q_2 = 0$ and the closest circular left non-empty bin is Bin 1, at distance 1, so the new value of Bin 2 for S_2 is $1 + C$. For Bin 5, $q_5 = 1$, so we go circular right and find non-empty Bin 1 at distance 2. The new hash value of Bin 5 is therefore $1 + 2C$. Note that the non-empty bins remain unchanged.

Formally, let $q_h\ j = \{0, 2, ..., k - 1\}$ be k i.i.d. Bernoulli random variables such

that $q_j = 1$ with probability $\frac{1}{2}$. The improved hash function $\mathcal{H}^+$ is given by

$$\mathcal{H}_j^+(S) = \begin{cases} \underset{(j-t_1) \bmod k}{OPH}(\pi(S)) + t_1 C \\ \qquad \text{if } q_j = 0 \text{ and } \underset{j}{OPH}(\pi(S)) = E \\ \underset{(j+t_2) \bmod k}{OPH}(\pi(S)) + t_2 C \\ \qquad \text{if } q_j = 1 \text{ and } \underset{j}{OPH}(\pi(S)) = E \\ \underset{j}{OPH}(\pi(S)) \qquad \text{otherwise} \end{cases} \tag{6.27}$$

where

$$t_1 = \min z, \quad s.t. \quad \underset{(j-z) \bmod k}{OPH}(\pi(S)) \neq E \tag{6.28}$$

$$t_2 = \min z, \quad s.t. \quad \underset{(j+z) \bmod k}{OPH}(\pi(S)) \neq E \tag{6.29}$$

with same $C = \frac{D}{k} + 1$. Computing k hash evaluations with $\mathcal{H}^+$ requires evaluating $\pi(S)$ followed by two passes over the k bins from different directions. The total complexity of computing k hash evaluation is again $O(d+k)$ which is the same as that of the existing densification scheme. We need an additional storage of the k bits (roughly hundreds or thousands in practice) which is practically negligible.

Similar to the arguments used for Theorem 9, it is not difficult to show that $\mathcal{H}^+$ satisfies the LSH property for resemblance.

Theorem 11

$$\mathbf{Pr}\left(\mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\right) = R \tag{6.30}$$

$\mathcal{H}^+$ leads to an unbiased estimator of resemblance $\hat{R}^+$

$$\hat{R}^+ = \frac{1}{k} \sum_{j=0}^{k-1} \mathbf{1}\{\mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\}. \tag{6.31}$$

6.4.2 Variance Analysis of Improved Scheme

When $m = 1$ (an event with prob $\left(\frac{1}{k}\right)^{f_1+f_2-a} \simeq 0$), i.e., only one simultaneously non-empty bin, both the schemes are exactly same. For simplicity of expressions, we will assume that the number of simultaneous non-empty bins is strictly greater than 1, i.e., $m > 1$. The general case has an extra term for $m = 1$, which makes the expression unnecessarily complicated without changing the final conclusion.

Following the notation as in Section 6.3.2, we denote

$$M_j^{N+} = \mathbf{1}\{I_{emp}^j = 0 \text{ and } \mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\} \tag{6.32}$$

$$M_j^{E+} = \mathbf{1}\{I_{emp}^j = 1 \text{ and } \mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\} \tag{6.33}$$

The two expectations $\mathbb{E}\left[\sum_{i\neq j} M_i^{N+}M_j^{N+}\middle|m\right]$ and $\mathbb{E}\left[\sum_{i\neq j} M_i^{N+}M_j^{E+}\middle|m\right]$ are the same as given by Lemma 2 and Lemma 3 respectively, as all the arguments used to prove them still hold for the new scheme. The only change is in the term $\mathbb{E}\left[\sum_{i\neq j} M_i^{E}M_j^{E}\middle|m\right]$.

Lemma 5

$$\mathbb{E}\left[\sum_{i\neq j} M_i^{E+}M_j^{E+}\middle|m\right] = (k-m)(k-m-1) \times \left[\frac{3R}{2(m+1)} + \frac{(2m-1)R\tilde{R}}{2(m+1)}\right] \tag{6.34}$$

Proof: See Appendix A.0.4. □

The theoretical variance of the new estimator $\hat{R}^+$ is

Theorem 12

$$Var(\hat{R}^+) = \frac{R}{k} + A^+\frac{R}{k^2} + B^+\frac{R\tilde{R}}{k^2} - R^2 \tag{6.35}$$
$$A^+ = \mathbb{E}\left[\frac{N_{emp}(4k - N_{emp} + 1)}{2(k - N_{emp} + 1)}\right]$$
$$B^+ = \mathbb{E}\left[\frac{2k^3 + N_{emp}^2 - N_{emp}(2k^2 + 2k + 1) - 2k}{2(k - N_{emp} + 1)}\right]$$

The new scheme reduces the value of p (see Eq.(6.26)) from $\frac{2}{m+1}$ to $\frac{1.5}{(m+1)}$. As argued in Section 6.4.1, this reduces the overall variance.

Theorem 13

$$Var(\hat{R}^+) \leq Var(\hat{R}) \tag{6.36}$$

More precisely,

$$\begin{aligned} &Var(\hat{R}) - Var(\hat{R}^+) \\ =&\mathbb{E}\left[\frac{(N_{emp})(N_{emp} - 1)}{2k^2(k - N_{emp} + 1)}[R - R\tilde{R}]\right] \geq 0 \end{aligned} \tag{6.37}$$

Proof: See Appendix A.0.4. □

The probability of simultaneously empty bins increases with increasing sparsity in dataset and the total number of bins k. We can see from Theorem 13 that with more simultaneously empty bins, i.e., higher N_{emp}, the gain with the improved scheme $\mathcal{H}^+$ is higher compared to $\mathcal{H}$. Hence, $\mathcal{H}^+$ should be significantly better than the existing scheme for very sparse datasets or in scenarios when we need a large number of hash values.

6.4.3 Comparisons of Mean Square Errors

We empirically verify the theoretical variances of $\mathcal{R}$ and $\mathcal{R}^+$ and their effects in many practical scenarios. To achieve this, we extracted 12 pairs of words (which cover a wide spectrum of sparsity and similarity) from the web-crawl dataset. The detailed information about the data is summarized in Table 6.2.

Table 6.2: 12 pairs of words used in Experiment 1. Every word is a set of documents in which the word is contained. For example, 'HONG" correspond to the sets of document IDs which contained word "HONG".

Word 1	Word 2	f_1	f_2	R
HONG	KONG	940	948	0.925
RIGHTS	RESERVED	12,234	11,272	0.877
A	THE	39,063	42,754	0.644
UNITED	STATES	4,079	3,981	0.591
TOGO	GREENLAND	231	200	0.528
ANTILLES	ALBANIA	184	275	0.457
CREDIT	CARD	2,999	2,697	0.285
COSTA	RICO	773	611	0.234
LOW	PAY	2,936	2,828	0.112
VIRUSES	ANTIVIRUS	212	152	0.113
REVIEW	PAPER	3,197	1,944	0.078
FUNNIEST	ADDICT	68	77	0.028

For all 12 pairs of words, we estimate the resemblance using the two estimators $\mathcal{R}$ and $\mathcal{R}^+$. We plot the empirical *Mean Square Error (MSE)* of both of both estimators with respect to k which is the number of hash evaluations. To validate the theoretical variances (which is also the MSE because the estimators are

unbiased), we also plot the values of the theoretical variances computed from Theorem 10 and Theorem 12. The results are summarized in Figure 6.13.

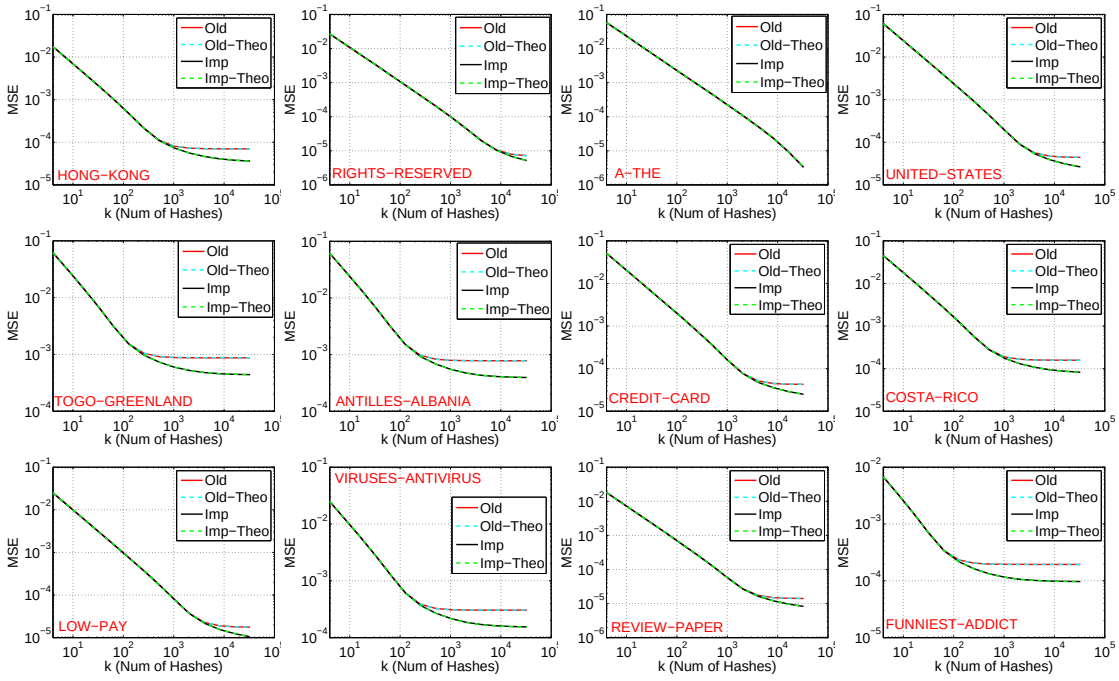

Figure 6.13: Mean Square Error (MSE) of the old scheme $\hat{R}$ and the improved scheme $\hat{R}^+$ along with their theoretical values on 12 word pairs (Table 6.2) from web-crawl dataset.

From the plots we can see that the theoretical and the empirical MSE values overlap in both the cases validating both Theorem 10 and Theorem 12. When k is small both the schemes have similar variances, but when k increases the improved scheme always shows better variance. For very sparse pairs, we start seeing a significant difference in variance even for k as small as 100. For a sparse pair, e.g., "TOGO" and "GREENLAND", the difference in variance, between the two schemes, is more compared to the dense pair "A" and "THE". This is in agreement with Theorem 13.

6.4.4 Near Neighbor Retrieval Comparison

Table 6.3: Dataset information.

Data	# dim	# nonzeros	# train	# query
RCV1	47,236	73	100,000	5,000
URL	3,231,961	115	90,000	5,000

In this experiment, we evaluate the two hashing schemes $\mathcal{H}$ and $\mathcal{H}^+$ on the standard (K, L) parameterized LSH algorithm [42, 6] for retrieving near neighbors. Two publicly available sparse text datasets, *RCV1* and *URL*, are described in Table 7.3.

In (K, L) parameterized LSH algorithm for near neighbor search, we generate L different meta-hash functions. Each of these meta-hash functions is formed by concatenating K different hash values as

$$B_j(S) = [h_{j1}(S); h_{j2}(S); ...; h_{jK}(S)], \tag{6.38}$$

where $h_{ij}, i \in \{1, 2, ..., K\}$, $j \in \{1, 2, ..., L\}$, are KL realizations of the hash function under consideration. The (K, L) parameterized LSH works in two phases:

1. **Preprocessing Phase:** We construct L hash tables from the data by storing element S, in the train set, at location $B_j(S)$ in hash-table j.
2. **Query Phase:** Given a query Q, we report the union of all the points in the buckets $B_j(Q)$ $\forall j \in \{1, 2, ..., L\}$, where the union is over L hash tables.

For every dataset, based on the similarity levels, we chose a K based on standard recommendation. For this K we show results for a set of values of L

depending on the recall values. Please refer to [6] for details on the implementation and the choice of K and L. Since both $\mathcal{H}$ and $\mathcal{H}^+$ have the same collision probability, the choice of K and L is the same in both cases.

For every query point, the gold standard top 10 near neighbors from the train set were computed based on actual resemblance. We then compute the recall of these gold standard neighbors and the total number of points retrieved by the (K, L) bucketing scheme. We report the mean computed over all the points in the query set. Since, the experiments involve randomization the final results presented are averaged over 10 independent runs. The recall and the points retrieved per query is summarized in Figure 6.14.

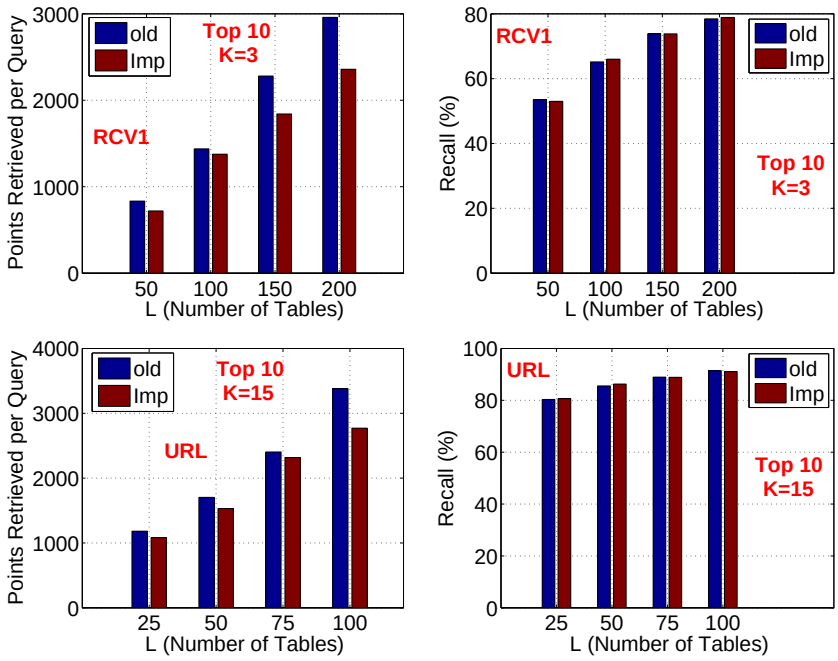

Figure 6.14: Average number of points scanned per query and the mean recall values of top 10 near neighbors, obtained from (K, L) parameterized LSH algorithm, using $\mathcal{H}$ (old) and $\mathcal{H}^+$ (Imp).

It is clear from Figure 6.14 that the improved hashing scheme $\mathcal{H}^+$ achieves

the same recall but at the same time retrieves less number of points compared to the old scheme $\mathcal{H}$. To achieve 90% recall on URL dataset, the old scheme retrieves around 3300 points per query on an average while the improved scheme only needs to check around 2700 points per query. For RCV1 dataset, with $L = 200$ the old scheme retrieves around 3000 points and achieves a recall of 80%, while the same recall is achieved by the improved scheme after retrieving only about 2350 points per query. A good hash function provides a right balance between recall and number of points retrieved. In particular, hash function which achieves a given recall and retrieves less number of points is desirable because it implies better precision. The above results clearly demonstrate the superiority of the indexing scheme with improved hash function $\mathcal{H}^+$ over the indexing scheme with $\mathcal{H}$.

Why $\mathcal{H}^+$ retrieves less number of points than $\mathcal{H}$?

The number of points retrieved, by the (K, L) parameterized LSH algorithm, is directly related to the collision probability of the meta-hash function $B_j(.)$ (Eq.(7.46)). Given S_1 and S_2 with resemblance R, the higher the probability of event $B_j(S_1) = B_j(S_2)$, under a hashing scheme, the more number of points it will retrieve per table.

The analysis of the variance (second moment) about the event $B_j(S_1) = B_j(S_2)$ under $\mathcal{H}^+$ and $\mathcal{H}$ provides some reasonable insight. Recall that since both estimators under the two hashing schemes are unbiased, the analysis of the first

moment does not provide information in this regard.

$$\mathbb{E}[\mathbf{1}\{\mathcal{H}_{j1}(S_1) = \mathcal{H}_{j1}(S_2)\} \times \mathbf{1}\{\mathcal{H}_{j2}(S_1) = \mathcal{H}_{j2}(S_2)\}]$$
$$= \mathbb{E}\left[M^N_{j1}M^N_{j2} + M^N_{j1}M^E_{j2} + M^E_{j1}M^N_{j2} + M^E_{j1}M^E_{j2}\right]$$

As we know from our analysis that the first three terms inside expectation, in the RHS of the above equation, behaves similarly for both $\mathcal{H}^+$ and $\mathcal{H}$. The fourth term $\mathbb{E}\left[M^E_{j1}M^E_{j2}\right]$ is likely to be smaller in case of $\mathcal{H}^+$ because of smaller values of p. We therefore see that $\mathcal{H}$ retrieves more points than necessary as compared to $\mathcal{H}^+$. The difference is visible when empty bins dominate and $M^E_1 M^E_2 = 1$ is more likely. This happens in case of sparse datasets which are common in practice.

6.5 Discussions

MinHash is a decade old algorithm which has made numerous online algorithms faster. In most of these algorithms, the bottleneck step is the computations of many MinHashes of the data. We have found a way to reduced this hashing time drastically. Although, we have only demonstrated the benefit in the task of near-neighbor search, the improvement in hashing time that we have obtained can be directly use to speedup most of the randomized algorithms over the web drastically, which use MinHash as a subroutine.

The idea of densification of one permutation hashing is general, and it can be used to densify any sparse sketches where sparsity is not desired. This could be of independent theoretical interest in itself.

CHAPTER 7

ASYMMETRIC LSH : INNER PRODUCTS ARE HASHABLE

The focus of this chapter is on the problem of *Maximum Inner Product Search (MIPS)*. In this problem, we are given a giant data vector collection $\mathcal{S}$ of size N, where $\mathcal{S} \subset \mathbb{R}^D$, and a given query point $q \in \mathbb{R}^D$. We are interested in searching for $p \in \mathcal{S}$ which maximizes (or approximately maximizes) the **inner product** $q^T p$. Formally, we are interested in efficiently computing

$$p = \arg\max_{x \in \mathcal{S}} \quad q^T x \tag{7.1}$$

The MIPS problem is related to *near neighbor search (NNS)*, which instead requires computing

$$p = \arg\min_{x \in \mathcal{S}} \|q - x\|_2^2 = \arg\min_{x \in \mathcal{S}} (\|x\|_2^2 - 2q^T x) \tag{7.2}$$

These two problems are equivalent if the norm of every element $x \in \mathcal{S}$ is constant. Note that the value of the norm $\|q\|_2$ has no effect as it is a constant throughout and does not change the identity of arg max or arg min. There are many scenarios in which MIPS arises naturally at places where the norms of the elements in $\mathcal{S}$ have very significant variations [46] and can not be controlled. We list few (among many) practical scenarios where MIPS is solved as a subroutine.

1. Recommender Systems: Recommender systems are often based on collaborative filtering which relies on past behavior of users, e.g., past purchases and ratings. Latent factor modeling based on matrix factorization [47] is a popular approach for solving collaborative filtering. In a typical matrix factorization model, a user i is associated with a latent user characteristic vector u_i, and similarly, an item j is associated with a latent item characteristic vector v_j. The rating

$r_{i,j}$ of item j by user i is modeled as the **inner product** between the corresponding characteristic vectors.

In this setting, given a user i and the corresponding learned latent vector u_i finding the right item j, to recommend to this user, involves computing

$$j = \arg\max_{j'} \quad r_{i,j'} = \arg\max_{j'} \quad u_i^T v_{j'} \tag{7.3}$$

which is an instance of the standard MIPS problem. It should be noted that we do not have control over the norm of the learned vector, i.e., $\|v_j\|_2$, which often has a wide range in practice [46].

If there are N items to recommend, solving (7.3) requires computing N inner products. Recommendation systems are typically deployed in on-line application over web where the number N is huge. A brute force linear scan over all items, for computing arg max, would be prohibitively expensive.

2. Large-Scale Object Detection with DPM: Deformable Part Model (DPM) based representation of images is the state-of-the-art in object detection tasks [30]. In DPM model, first a set of part filters are learned from the train dataset. During detection, these learned filter activations over various patches of the test image are used to score the test image. The activation of a filter on an image patch is an inner product between them. Typically, the number of possible filters are large (e.g., millions) and so scoring the test image is costly. Very recently, it was shown that scoring based only on filters with high activations performs well in practice [28]. Identifying filters, from a large collection of possible filters, having high activations on a given image patch requires computing top inner products. Consequently, an efficient solution to the MIPS problem will benefit large scale object detections based on DPM.

3. Multi-Class Label Prediction: The models for multi-class SVM (or logistic regression) learn a weight vector w_i for each of the class label i. After the weights are learned, given a new test data vector x_{test}, predicting its class label is basically an MIPS problem:

$$y_{test} = \arg\max_{i \in \mathcal{L}} \quad x_{test}^T \, w_i \tag{7.4}$$

where $\mathcal{L}$ is the set of possible class labels. Note, the vectors $\|w_i\|_2$ are learned as a part of an optimization and therefore we have no control over norms.

The size, $|\mathcal{L}|$, of the set of class labels differs in applications. Classifying with large number of possible class labels is common in fine grained object classification, for instance, prediction task with 100,000 classes [28] (i.e., $|\mathcal{L}| = 100,000$). Computing such high-dimensional vector multiplications for predicting the class label of a single instance can be expensive in, e.g., user-facing applications.

4. Large Scale Deep Learning Systems: Modern deep learning systems with millions or billions of nodes in every hidden layer are becoming more and more popular. To regularize such a humongous parameter space various algorithms, such as, Maxout [34] networks, max product networks [72] and networks with adaptive dropouts [7], have shown significant success. These algorithms require computation of maximum inner products as a subroutine for computing activation (or max activations), which is invoked during every iteration of training. Even during testing which is latency critical the same operations are needed. Computing all possible inner products, for finding the maximum or large inner products, is very inefficient from energy perspective. A faster subroutine for MIPS will significantly make these networks more scalable.

In this chapter, we present the first provably sublinear time hashing algorithm for approximate *Maximum Inner Product Search* (MIPS) [80, 85, 86]. Search-

ing with (un-normalized) inner product as the underlying similarity measure was a known difficult problem and finding hashing schemes for MIPS was considered hard. While the existing Locality Sensitive Hashing (LSH) framework is insufficient for solving MIPS, in this Chapter, we extend the LSH framework to allow asymmetric hashing schemes. In the extended framework we show that MIPS admits efficient algorithms. In the new framework, we provide explicit constructions of different hash functions which leads to provable sublinear algorithms for MIPS. These obtained algorithms are also practical. The ideas used provide a generic recipe for constructing new hashing schemes for many similarity measures, which could be of independent theoretical interest in itself.

7.1 Impossibility of LSH for Inner Products

We first start by observing a negative result that it is mathematically impossible to obtain a locality sensitive hashing for MIPS. In [76, 46], the authors also argued that finding locality sensitive hashing for inner products could be hard, but to the best of our knowledge we have not seen a formal proof.

Theorem 14 *There can not exist any LSH family for MIPS.*

Proof: *Suppose, there exists such hash function h. For un-normalized inner products the self similarity of a point x with itself is $Sim(x, x) = x^T x = \|x\|_2^2$ and there may exist another points y, such that $Sim(x, y) = y^T x > \|x\|_2^2 + M$, for any constant M. Under any single randomized hash function h, the collision probability of the event $\{h(x) = h(x)\}$ is always 1. So if h is an LSH for inner product then the event $\{h(x) = h(y)\}$ should have higher probability compared to the event $\{h(x) = h(x)\}$, since we can always choose y with $Sim(x, y) = S_0 + \delta > S_0$ and $cS_0 > Sim(x, x)\ \forall S_0$ and $c < 1$. This is not possible because the probability can not be greater than 1. This completes the proof.* □

Note that in [18] it was shown that we can not have a hash function where the collision probability is equal to the inner product, because "1 - inner product" does not satisfy the triangle inequality. This does not totally eliminates the existence of LSH. For instance, under L2-LSH, the collision probability is a monotonic function of distance and not the distance itself.

7.2 New Framework: Asymmetric LSH (ALSH)

The basic idea of LSH is probabilistic bucketing and it is more general than the requirement of having a single hash function h. In the LSH algorithm (Section 2.3), we use the same hash function h for both the preprocessing step and the query step. We assign buckets in the hash table to all the candidates $x \in \mathcal{S}$ using h. We use the same h on the query q to identify relevant buckets. The only requirement for the proof, of Fact 1, to work is that the collision probability of the event $\{h(q) = h(x)\}$ increases with the similarity $Sim(q, x)$. The theory [36] behind LSH still works if we use hash function h_1 for preprocessing $x \in \mathcal{S}$ and a different hash function h_2 for querying, as long as the probability of the event $\{h_2(q) = h_1(x)\}$ increases with $Sim(q, x)$, and there exist p_1 and p_2 with the required property. The traditional LSH definition does not allow this asymmetry but it is not a required condition in the proof. For this reason, we can relax the definition of c-NN without losing runtime guarantees. As the first step we define Asymmetric locality sensitive hashing, which will be useful later.

Definition 3 *(Asymmetric Locality Sensitive Hashing (ALSH)) A family $\mathcal{H}$, along with the two vector functions $Q : \mathbb{R}^D \mapsto \mathbb{R}^{D'}$ (**Query Transformation**) and $P : \mathbb{R}^D \mapsto \mathbb{R}^{D'}$ (**Preprocessing Transformation**), is called (S_0, cS_0, p_1, p_2)-sensitive if*

for a given c-NN instance with query q, and the hash function h chosen uniformly from $\mathcal{H}$ satisfies the following:

- *if $Sim(q, x) \geq S_0$ then $Pr_{\mathcal{H}}(h(Q(q))) = h(P(x))) \geq p_1$*
- *if $Sim(q, x) \leq cS_0$ then $Pr_{\mathcal{H}}(h(Q(q)) = h(P(x))) \leq p_2$*

Here x is any point in the collection $\mathcal{S}$. When $Q(x) = P(x) = x$, we recover the vanilla LSH definition with $h(.)$ as the required hash function. Coming back to the problem of MIPS, if Q and P are different, the event $\{h(Q(x)) = h(P(x))\}$ will not have probability equal to 1 in general. Thus, $Q \neq P$ can counter the fact that self similarity is not highest with inner products. We just need the probability of the new collision event $\{h(Q(q)) = h(P(y))\}$ to satisfy the conditions of Definition of c-NN for $Sim(q, y) = q^T y$. Note that the query transformation Q is only applied on the query and the pre-processing transformation P is applied to $x \in \mathcal{S}$ while creating hash tables. It is this asymmetry which will allow us to solve MIPS efficiently. In Section 7.5.2, we explicitly show a construction (and hence the existence) of asymmetric locality sensitive hash function for solving MIPS. The source of randomization h for both q and $x \in \mathcal{S}$ is the same. Formally, it is not difficult to show a result analogous to Fact 1.

Theorem 15 *Given a family of hash function $\mathcal{H}$ and the associated query and pre-processing transformations P and Q, which is (S_0, cS_0, p_1, p_2) -sensitive, one can construct a data structure for c-NN with $O(n^{\rho} \log n)$ query time and space $O(n^{1+\rho})$, where $\rho = \frac{\log p_1}{\log p_2}$.*

7.3 The First Construction of ALSH for MIPS

In this section, we provide the first construction of hashing scheme for MIPS. In later sections, we show few improved versions.

7.3.1 From MIPS to Near Neighbor Search (NNS)

Without loss of any generality, let $U < 1$ be a number such that $\|x_i\|_2 \leq U < 1, \ \forall x_i \in \mathcal{S}$. If this is not the case then define a scaling transformation,

$$S(x) = \frac{U}{M} \times x; \qquad M = max_{x_i \in \mathcal{S}} \|x_i\|_2; \tag{7.5}$$

Note, that we are allowed one time preprocessing and asymmetry, S is the part of asymmetric transformation. For simplicity of arguments, let us assume that $\|q\|_2 = 1$, the arg max in Equation 7.1 is independent of the query. Later we show in Section 7.3.3 that it can be easily removed.

We are now ready to describe the key step in our algorithm. First, we define two vector transformations $P : \mathbb{R}^D \mapsto \mathbb{R}^{D+m}$ and $Q : \mathbb{R}^D \mapsto \mathbb{R}^{D+m}$ as follows:

$$P(x) = [x; \|x\|_2^2; \|x\|_2^4; ....; \|x\|_2^{2^m}]; \qquad Q(x) = [x; 1/2; 1/2; ....; 1/2], \tag{7.6}$$

where [;] is the concatenation. $P(x)$ appends m scalers of the form $\|x\|_2^{2^i}$ at the end of the vector x, while Q(x) simply appends m "1/2" to the end of the vector x.

By observing that

$$Q(q)^T P(x_i) = q^T x_i + \frac{1}{2}(\|x_i\|_2^2 + \|x_i\|_2^4 + ... + \|x_i\|_2^{2^m}); \quad \|P(x_i)\|_2^2 = \|x_i\|_2^2 + \|x_i\|_2^4 + ... + \|x_i\|_2^{2^{m+1}}$$

we obtain the following key equality:

$$\|Q(q) - P(x_i)\|_2^2 = (1 + m/4) - 2q^T x_i + \|x_i\|_2^{2^{m+1}} \tag{7.7}$$

Since $\|x_i\|_2 \leq U < 1$, $\|x_i\|^{2^{m+1}} \to 0$, at the tower rate (exponential to exponential). The term $(1 + m/4)$ is a fixed constant. As long as m is not too small (e.g., $m \geq 3$ would suffice), we have

$$\arg\max_{x \in \mathcal{S}} q^T x \simeq \arg\min_{x \in \mathcal{S}} \|Q(q) - P(x)\|_2 \tag{7.8}$$

This gives us connection between solving un-normalized MIPS and approximate near neighbor search. Transformations P and Q, when norms are less than 1, provide correction to the L2 distance $\|Q(q) - P(x_i)\|_2$ making it rank correlate with the (un-normalized) inner product. This works only after shrinking the norms, as norms greater than 1 will instead blow the term $\|x_i\|_2^{2^{m+1}}$.

7.3.2 L2-ALSH

Eq. (7.8) shows that MIPS reduces to the standard approximate near neighbor search problem which can be efficiently solved. As the error term $\|x_i\|_2^{2^{m+1}} < U^{2^{m+1}}$ goes to zero at a tower rate, it quickly becomes negligible for any practical purposes. In fact, from theoretical perspective, since we are interested in guarantees for c-approximate solutions, this additional error can be absorbed in the approximation parameter c. Formally, we can state the following theorem.

Theorem 16 *Given a c-approximate instance of MIPS, i.e., $Sim(q, x) = q^T x$, and a query q such that $\|q\|_2 = 1$ along with a collection $\mathcal{S}$ having $\|x\|_2 \leq U < 1$ $\forall x \in \mathcal{S}$. Let P and Q be the vector transformations defined in Eq. (7.16), respectively. We have the following two conditions for hash function $h^{L2}_{a,b}$ (defined by Eq. (2.10))*
1) if $q^T x \geq S_0$ then $Pr[h^{L2}_{a,b}(Q(q)) = h^{L2}_{a,b}(P(x))] \geq F_r(\sqrt{1 + m/4 - 2S_0 + U^{2^{m+1}}})$

2) if $q^T x \leq cS_0$ *then* $Pr[h^{L2}_{a,b}(Q(q)) = h^{L2}_{a,b}(P(x))] \leq F_r(\sqrt{1 + m/4 - 2cS_0})$
where the function F_r *is defined in Eq. (2.11).*

Proof: See Appendix A.0.5. □

Thus, we have obtained $p_1 = F_r(\sqrt{(1 + m/4) - 2S_0 + U^{2^{m+1}}})$ and $p_2 = F_r(\sqrt{(1 + m/4) - 2cS_0})$. Applying Theorem 15, we can construct data structures with worst case $O(n^\rho \log n)$ query time guarantees for c-approximate MIPS, where

$$\rho = \frac{\log F_r(\sqrt{1 + m/4 - 2S_0 + U^{2^{m+1}}})}{\log F_r(\sqrt{1 + m/4 - 2cS_0})} \tag{7.9}$$

We also need $p_1 > p_2$ in order for $\rho < 1$. This requires us to have $-2S_0 + U^{2^{m+1}} < -2cS_0$, which boils down to the condition $c < 1 - \frac{U^{2^{m+1}}}{2S_0}$. Note that $\frac{U^{2^{m+1}}}{2S_0}$ can be made arbitrarily close to zero with the appropriate value of m. For any given $c < 1$, there always exist $U < 1$ and m such that $\rho < 1$. This way, we obtain a sublinear query time algorithm for MIPS. The guarantee holds for any values of U and m satisfying $m \in \mathbb{N}^+$ and $U \in (0, 1)$. We also have one more parameter r for the hash function $h_{a,b}$. Recall the definition of F_r in Eq. (2.11): $F_r(d) = 1 - 2\Phi(-r/d) - \frac{2}{\sqrt{2\pi}(r/d)}\left(1 - e^{-(r/d)^2/2}\right)$.

Given a c-approximate MIPS instance, ρ is a function of 3 parameters: U, m, r. The algorithm with the best query time chooses U, m and r, which minimizes the value of ρ. For convenience, we define

$$\rho^* = \min_{U,m,r} \frac{\log F_r(\sqrt{1 + m/4 - 2S_0 + U^{2^{m+1}}})}{\log F_r(\sqrt{1 + m/4 - 2cS_0})} \quad s.t. \quad \frac{U^{2^{m+1}}}{2S_0} < 1 - c, \; m \in \mathbb{N}^+, \; 0 < U < 1. \tag{7.10}$$

See Figure 7.6 for the plots of ρ^*. With this best value of ρ, we can state our main result in Theorem 20.

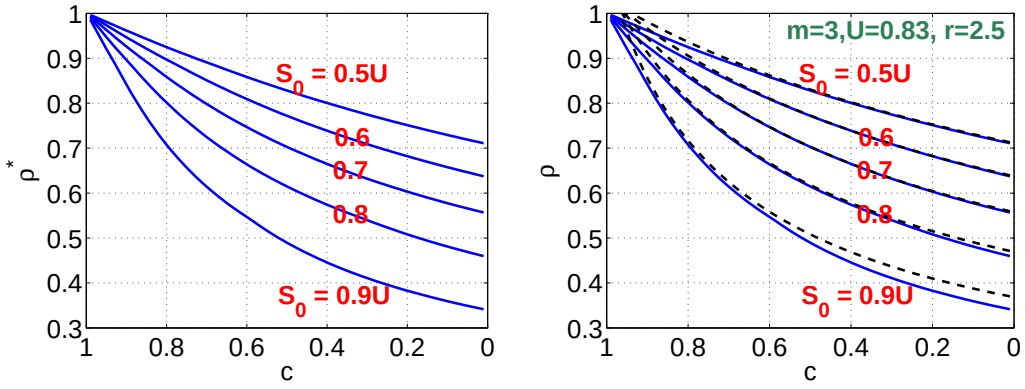

Figure 7.1: **Left**: Optimal values of ρ^* with respect to approximation ratio c for different S_0. **Right**: ρ values (dashed curves) for $m = 3$, $U = 0.83$ and $r = 2.5$. The solid curves are ρ^* values.

Theorem 17 *(**Approximate MIPS is Efficient**) For the problem of c-approximate MIPS with $\|q\|_2 = 1$, one can construct a data structure having $O(n^{\rho^*} \log n)$ query time and space $O(n^{1+\rho^*})$, where $\rho^* < 1$ is the solution to constraint optimization (7.23).*

Just like in the typical LSH framework, the value of ρ^* in Theorem 20 depends on the c-approximate instance we aim to solve, which requires knowing the similarity threshold S_0 and the approximation ratio c. Since, $\|q\|_2 = 1$ and $\|x\|_2 \leq U < 1$, $\forall x \in \mathcal{S}$, we have $q^t x \leq U$. A reasonable choice of the threshold S_0 is to choose a high fraction of U, for example, $S_0 = 0.9U$ or $S_0 = 0.8U$.

The computation of ρ^* and the optimal values of corresponding parameters can be conducted via a grid search over the possible values of U, m and r, as we only have 3 parameters. We compute the values of ρ^* along with the corresponding optimal values of U, m and r for $S_0 \in \{0.9U,\ 0.8U,\ 0.7U,\ 0.6U,\ 0.5U\}$ for different approximation ratios c ranging from 0 to 1. The plot of the optimal ρ^* is shown in Figure 7.6, and the corresponding optimal values of U, m and r are shown in Figure 7.2.

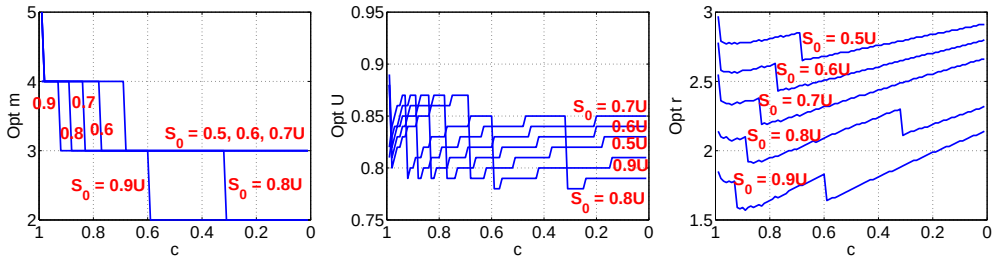

Figure 7.2: Optimal values of parameters m, U and r with respect to the approximation ratio c for high similarity threshold S_0.

Practical Recommendation of Parameters In practice, the actual choice of S_0 and c is dependent on the data and is often unknown. Figure 7.2 illustrates that $m \in \{2, 3, 4\}$, $U \in [0.8,\ 0.85]$, and $r \in [1.5,\ 3]$ are reasonable choices. For convenience, we recommend to use $m = 3$, $U = 0.83$, and $r = 2.5$. With this choice of the parameters, Figure 7.6 (Right Panel) shows that the ρ values using these parameters are very close to ρ^* values.

7.3.3 Removing the condition $\|q\|_2 = 1$

Changing norms of the query does not affect the $\arg\max_{x \in C} q^T x$, and hence, in practice for retrieving top-k, normalizing the query should not affect the performance. But for theoretical purposes, we want the runtime guarantee to be independent of $\|q\|_2$. We are interested in the c-approximate instance which being a threshold based approximation changes if the query is normalized.

Transformations P and Q were precisely meant to remove the dependency of euclidean distance on the norms of x. Realizing the fact that we are allowed asymmetry, we can use the same idea to get rid of the norm of q. Let M be the upper bound on all the norms or the radius of the space as defined in equa-

tion 7.5. Let the transformations $P, Q : \mathbb{R}^D \to \mathbb{R}^{D+m}$ and $S : \mathbb{R}^D \to \mathbb{R}^D$ be the ones defined in Eq (7.16), (7.17) and (7.5) respectively.

Given the query q and data point x, our new asymmetric transformations are $P(Q(S(q)))$ and $Q(P(S(x)))$ respectively. Observe that

$$||P(Q(S(q))) - Q(P(S(x)))||_2 = \frac{m}{2} + ||S(x)||_2^{2^{m+1}} + ||S(q)||_2^{2^{m+1}} - 2q^t x \times \left(\frac{U^2}{M^2}\right) \quad (7.11)$$

$P(Q(S(q)))$ appends first m 1/2s to $S(q)$ and then m components of the form $||S(q)||^{2^i}$. $Q(P(T(q)))$ does the same thing but in a different order. Now we are working in $D+2m$ dimensions. The transformations are asymmetric in ordering but we know that asymmetry is necessary. Both $||S(x)||_2^{2^{m+1}}, ||S(q)||_2^{2^{m+1}} \leq U^{2^{m+1}} \to 0$. Using exactly same arguments as before, we get

Theorem 18 *(**Unconditional Approximate MIPS is Efficient**) For the problem of c-approximate MIPS in a bounded space, one can construct a data structure having $O(n^{\rho_u^*} \log n)$ query time and space $O(n^{1+\rho_u^*})$, where $\rho_u^* < 1$ is the solution to constraint optimization (7.23).*

$$\rho_u^* = \min_{0<U<1, m\in N, r} \frac{\log F_r(\sqrt{m/2 - 2S_0\left(\frac{U^2}{M^2}\right) + 2U^{2^{m+1}}})}{\log F_r(\sqrt{m/2 - 2cS_0\left(\frac{U^2}{M^2}\right)})} \quad s.t. \quad \frac{U^{(2^{m+1}-2)}M^2}{S_0} < 1 - c, \quad (7.12)$$

Again, for any c-approximate MIPS instance, with S_0 and c, we can always choose m big enough such that the constraint is always satisfied leading to $\rho_u^* < 1$. Thus we obtain first sub-linear solutions to c-approximate MIPS. The theoretical guarantee only depends on the radius of the space M.

7.3.4 Computational Savings in Collaborative Filtering

We evaluate our proposed hash function for the MIPS problem on two popular collaborative filtering datasets (on the task of item recommendations):
1. Movielens. We choose the largest available Movielens dataset, the *Movielens 10M*, which contains around 10 million movie ratings from 70,000 users over 10,000 movie titles. The ratings are between 1 to 5, with increments of 0.5 (i.e., 10 possible ratings in total).
2. Netflix. This dataset contains 100 million movie ratings from 480,000 users over 17,000 movie titles. The ratings are on a scale from 1 to 5 (integer).

Each dataset forms a sparse **user-item matrix** R, where the value of $R(i, j)$ indicates the rating of user i for movie j. Given the user-item ratings matrix R, we follow the standard PureSVD procedure described in [23] to generate user and item latent vectors. This procedure generates latent vectors u_i for each user i and vector v_j for each item j, in some chosen fixed dimension f. The PureSVD method returns top-ranked items based on the inner products $u_i^T v_j, \ \forall j$.

The PureSVD procedure, despite its simplicity, outperforms other popular recommendation algorithms for the task of top-k recommendations (see [23]) on these two datasets. Following [23], we use the same choices for the latent dimension f, i.e., $f = 150$ for Movielens and $f = 300$ for Netflix.

Baseline Heuristics: Our proposal is the first provable hashing scheme in the literature for retrieving inner products and hence there is no existing baseline. Since, our hash function uses Hashing for L2 distance after asymmetric transformation P and Q (7.16), we would like to know if such transformations are even needed and furthermore get an estimate of the improvements obtained

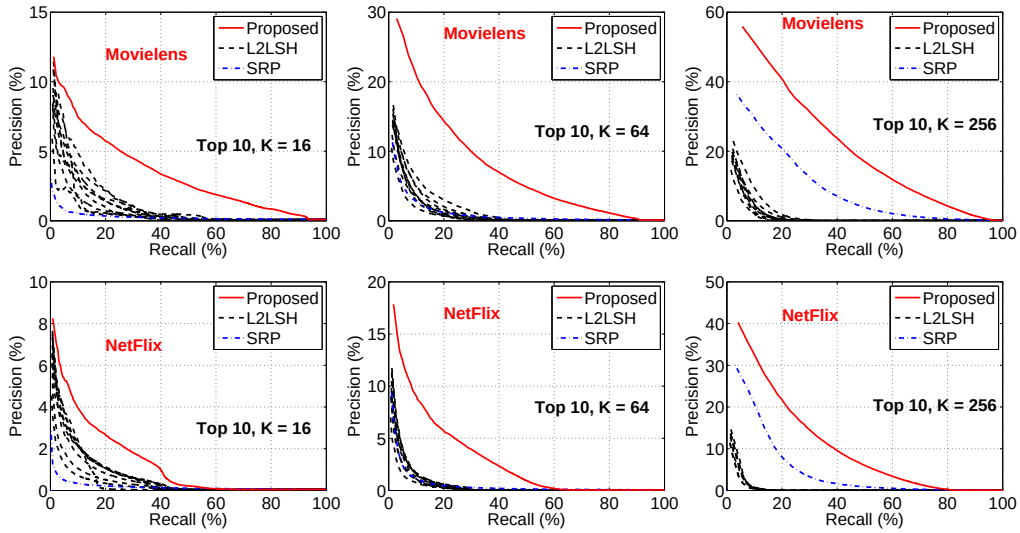

Figure 7.3: Precision-Recall curves (higher is better), of the rankings based on hash collisions for top-10 items.

using these transformations. We therefore compare our proposal with **L2-LSH** the hashing scheme $h^{L2}_{a,b}$ described by Eq. (2.10). It is implemented in the LSH package for near neighbor search with Euclidean distance. In addition, we compare with **SRP** which is the other popular LSH in literature.

Hash Code Quality Evaluations

We are interested in knowing, how the two hash functions correlate with the top-T inner products. For this task, given a user i and its corresponding user vector u_i, we compute the top-T gold standard items based on the actual inner products $u_i^T v_j$, $\forall j$. We then compute K different hash codes of the vector u_i and all the item vectors v_js. For every item v_j, we then compute the number of times its hash values matches (or collides) with the hash values of query which is user u_i, i.e., we compute $Matches_j = \sum_{t=1}^{K} \mathbf{1}(h_t(u_i) = h_t(v_j))$ where $\mathbf{1}$ is the indicator

function. Based on $Matches_j$ we rank all the items. Here, we use standard hash functions for SRP and L2-LSH. For our proposed asymmetric hash function we have $h(u_i) = h^{L2}_{a,b}(Q(u_i))$, since u_i is the query, and $h(v_j) = h^{L2}_{a,b}(P(v_j))$ for all the items. We compute the precision and recall of the top-T items for $T \in \{1, 5, 10\}$, obtained from the sorted list based on *Matches* for different hash functions. We use our recommended setting of $m = 3$, $U = 0.83$, and $r = 2.5$ for both datasets. We plot all combinations of r in $\{1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\}$ for L2-LSH. SRP does not have any parameter. To compute this precision and recall, we start at the top of the ranked item list and walk down in order. Suppose we are at the k^{th} ranked item, we check if this item belongs to the gold standard top-T list. If it is one of the top-T gold standard item, then we increment the count of *relevant seen* by 1, else we move to $k+1$. By k^{th} step, we have already seen k items, so the *total items seen* is k. The precision and recall at that point is then computed as:

$$Precision = \frac{\text{relevant seen}}{k}, \qquad Recall = \frac{\text{relevant seen}}{T} \tag{7.13}$$

We vary a large number of k values to obtain continuously-looking *precision-recall* curves. Note that it is important to balance both precision and recall. Methodology which obtains higher precision at a given recall is superior. Higher precision indicates higher ranking of the relevant items. We average this value of precision and recall over different users. The results for $K = 16$, 64 and 256 hash codes is summarized in Figure 7.8.

We can clearly see, that the proposed hashing scheme is significantly better than SRP and L2-LSH. This is not surprising because we know from Theorem 16 that collision under the new hash function is a direct indicator of high inner product. The suboptimal performance of SRP and L2-LSH clearly indicates that the norms of the item characteristic vectors do play a significant role in item recommendation task.

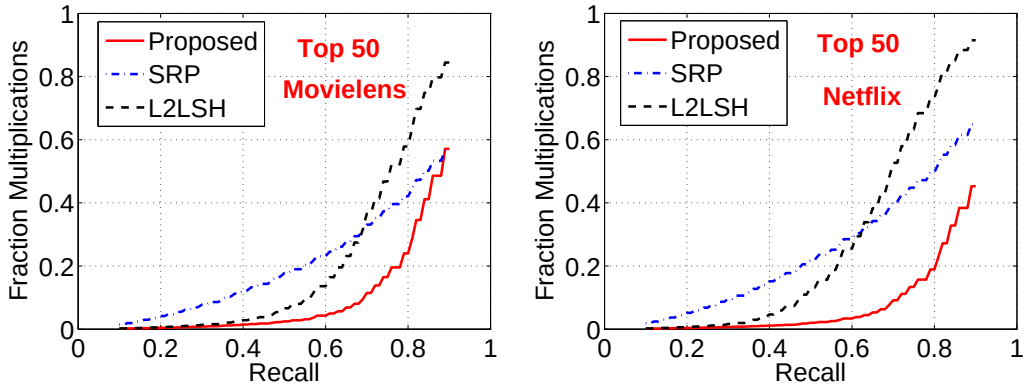

Figure 7.4: Mean number of inner products per query, relative to a linear scan, evaluated by different hashing schemes at different recall levels, for generating top-50 recommendations. The plot includes the additional inner product computations while hashing the query. The best performing K and L at a given recall value, for all the three hashing schemes, is shown.

Actual Savings in Top Items Retrieval

We implemented the standard (K, L) parametrized bucketing algorithm as described in Section 2.3 for retrieving top-50 items based on PureSVD procedure using the proposed hash function and the two heuristics hash function SRP and L2-LSH. We plot the recall, with the mean ratio of inner product required to achieve a particular recall with the number of inner products required in a linear scan. This ratio include the inner products required while hashing queries. The results are summarized in Figure 7.11. Again, we use $m = 3$, $U = 0.83$, and $r = 2.5$ for our hashing scheme. For L2-LSH, we observe that the recommended setting [25] of $r = 4$ usually performs the best and so we show $r = 4$.

It is evident from the plots that the proposed hashing scheme leads to significant savings compared to other hash functions in the task of top-50 recommendations. In order to correctly identify the top-50 item, with 80% accuracy, the

proposed scheme only needs to evaluate 20% of the total inner products, while L2-LSH roughly need 60%. SRP needs to evaluate 40% of the total inner product. For the Movielens dataset, this translates roughly into an average of 2k, 6k and 4k inner product evaluations per query with the proposed scheme, SRP and L2-LSH respectively. On the Netflix dataset the numbers are 3.4k, 13.6k, and 8.5k respectively. Clearly L2-LSH and SRP heuristics are suboptimal for retrieving inner products.

7.4 A Generic Framework for ALSH Construction

We are allowed any asymmetric transformation on x and q. This gives us a lot of flexibility to construct ALSH for new similarities $\mathcal{S}$ that we are interested in. The generic idea is to take a particular similarity $Sim(x, q)$ for which we know an existing LSH or ALSH. Then we construct transformations P and Q such $Sim(P(x), Q(q))$ is monotonic in the similarity $\mathcal{S}$ that we are interested in. The other observation that makes it easier to construct P and Q is that LSH based guarantees are independent of dimensions, thus we can expand the dimensions like we did for P and Q. In the next two Sections, we demonstrate the power of this framework by constructing provably superior ALSH schemes.

7.5 Sign-ALSH: Better ALSH for MIPS

It was argued in [58] that the quantizations used in traditional L2-LSH is suboptimal and it hurts the variance of the hashes. This raises a natural question that L2-ALSH, developed in previous Section, which uses L2-LSH as a subrout-

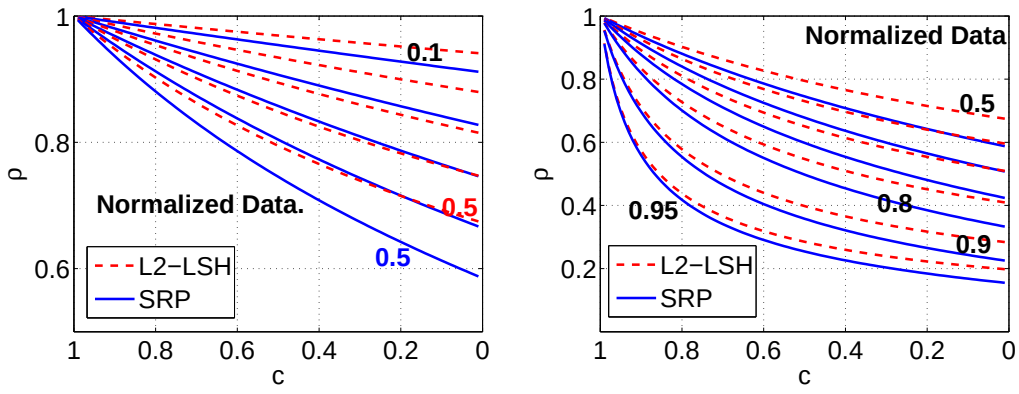

Figure 7.5: Values of ρ_{SRP} and ρ_{L2-LSH} (Lower is better) for normalized data. It is clear that SRP is more suited for retrieving inner products when the data is normalized

ing for solving MIPS could be suboptimal and there may be a better hashing scheme. We provide such a scheme in this Section.

7.5.1 Intuition for the Better Scheme : Why Signed Random Projections (SRP)

Recently [58], it was observed that the quantization of random projections used by traditional L2-LSH scheme is not desirable when the data is normalized and in fact the shift b in Equation 2.10 hurts the variance leading to less informative hashes. The sub-optimality of L2-LSH hints towards existence of better hashing functions for MIPS.

As argued before when the data is normalized then both L2-NNS and correlation-NNS are equivalent to MIPS. Therefore, for normalized data we can use either L2-LSH which is popular LSH for L2 distance or SRP which is popular LSH for correlation to solve MIPS directly. This raises a natural question

"Which will perform better ?".

If we assume that the data is normalized, i.e., all the norms are equal to 1, then both SRP and L2-LSH are monotonic in the inner product and their corresponding ρ values for retrieving max inner product can be computed as

$$\rho_{SRP} = \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}(S_0)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}(cS_0)\right)} \tag{7.14}$$

$$\rho_{L2-LSH} = \frac{\log\left(F_r(\sqrt{2 - 2S_0})\right)}{\log\left(F_r(\sqrt{2 - 2cS_0})\right)} \tag{7.15}$$

where the function $F_r(.)$ is given by Equation 2.11. The values of ρ_{SRP} and ρ_{L2-LSH} for different $S_0 = \{0.1, 0.2, .., 0.9, 0.95\}$ with respect to approximation ratio c is shown in Figure 7.5. We use standard recommendation of $r = 2.5$ for L2-LSH. We can clearly see that ρ_{SRP} is consistently better than ρ_{L2-LSH} given any S_0 and c. Thus, for MIPS with normalized data L2-LSH type of quantization given by equation 2.10 seems suboptimal. It is clear that when the data is normalized then SRP is always a better choice for MIPS as compared to L2-LSH. This motivates us to explore the possibility of better hashing algorithm for general (unnormalized) instance of MIPS using SRP, which will have impact in many applications as pointed out in [80].

Asymmetric transformations give us enough flexibility to modify norms without changing inner products. The transformations provided in [80] used this flexibility to convert MIPS to standard near neighbor search in L_2 space for which we have standard hash functions. Signed random projections are popular hash functions widely adopted for correlation or cosine similarity. We use asymmetric transformations to convert approximate MIPS into approximate maximum correlation search and thus we avoid the use of sub-optimal L2-LSH. The collision probability of the hash functions is one of the key constituents

which determine the efficiency of the obtained ALSH algorithm. We show that our proposed transformation with SRP is better suited for ALSH compared to the existing L2-ALSH for solving general MIPS instance.

7.5.2 From MIPS to Correlation-NNS

We assume for simplicity that $\|q\|_2 = 1$ as the norm of the query does not change the ordering, we show in the next section how to get rid of this assumption. Without loss of generality let $\|x_i\|_2 \leq U < 1, \ \forall x_i \in \mathcal{S}$ as it can always be achieved by scaling the data by large enough number. We define two vector transformations $P : \mathbb{R}^D \mapsto \mathbb{R}^{D+m}$ and $Q : \mathbb{R}^D \mapsto \mathbb{R}^{D+m}$ as follows:

$$P(x) = [x; 1/2 - \|x\|_2^2; 1/2 - \|x\|_2^4; ....; 1/2 - \|x\|_2^{2^m}] \tag{7.16}$$

$$Q(x) = [x; 0; 0; ....; 0], \tag{7.17}$$

Using $\|Q(q)\|_2^2 = \|q\|_2^2 = 1$, $Q(q)^T P(x_i) = q^T x_i$, and

$$\begin{aligned}
&\|P(x_i)\|_2^2 \\
&= \|x_i\|_2^2 + 1/4 + \|x_i\|_2^4 - \|x_i\|_2^2 + 1/4 + \|x_i\|_2^8 - \|x_i\|_2^4 + ... \\
&+ 1/4 + \|x_i\|_2^{2^{m+1}} - \|x_i\|_2^{2^m} \\
&= m/4 + \|x_i\|_2^{2^{m+1}}
\end{aligned}$$

we obtain the following key equality:

$$\frac{Q(q)^T P(x_i)}{\|Q(q)\|_2 \|P(x_i)\|_2} = \frac{q^T x_i}{\sqrt{m/4 + \|x_i\|_2^{2^{m+1}}}} \tag{7.18}$$

The term $\|x_i\|^{2^{m+1}} \to 0$, again vanishes at the tower rate. This means we have approximately

$$\arg\max_{x \in \mathcal{S}} q^T x \simeq \arg\max_{x \in \mathcal{S}} \frac{Q(q)^T P(x_i)}{\|Q(q)\|_2 \|P(x_i)\|_2} \tag{7.19}$$

This provides another solution for solving MIPS using known methods for approximate correlation-NNS. Asymmetric transformations P and Q provide a lot of flexibility. It should be noted that transformations P and Q are not unique for this task and there are other possibilities [9, 69]. Transformations P and Q allow us to use signed random projections and thereby making it possible to avoid suboptimal L2-LSH.

7.5.3 Fast Algorithms for MIPS Using Sign Random Projections

Eq. (7.19) shows that MIPS reduces to the standard approximate near neighbor search problem which can be efficiently solved by sign random projections, i.e., h^{Sign} (defined by Eq. (2.6)). Formally, we can state the following theorem.

Theorem 19 *Given a c-approximate instance of MIPS, i.e., $Sim(q, x) = q^T x$, and a query q such that $\|q\|_2 = 1$ along with a collection $\mathcal{S}$ having $\|x\|_2 \leq U < 1 \; \forall x \in \mathcal{S}$. Let P and Q be the vector transformations defined in Eq. (7.16) and Eq. (7.17), respectively. We have the following two conditions for hash function h^{Sign} (defined by Eq. (2.7))*

- *if $q^T x \geq S_0$ then*

$$Pr[h^{Sign}(Q(q)) = h^{Sign}(P(x))] \geq 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0}{\sqrt{m/4 + U^{2^{m+1}}}}\right)$$

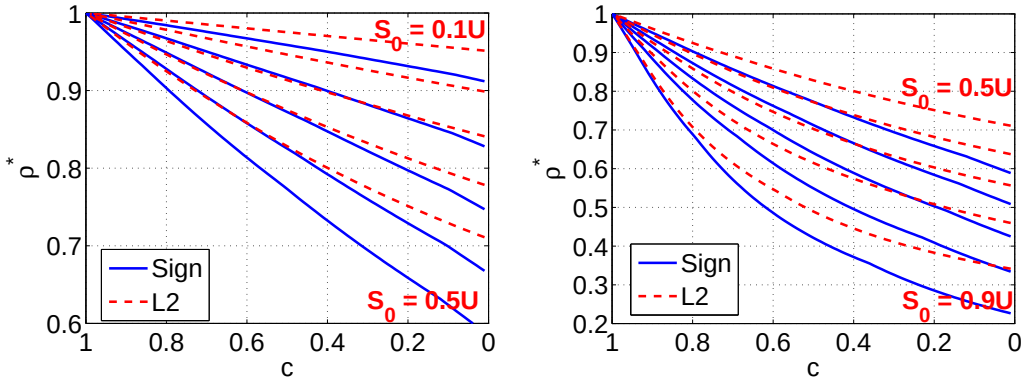

Figure 7.6: Optimal values of ρ^* (lower is better) with respect to approximation ratio c for different S_0, obtained by a grid search over parameters U and m, given S_0 and c.

- *if* $q^T x \leq cS_0$ *then*

$$Pr[h^{Sign}(Q(q)) = h^{Sign}(P(x))] \leq 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\min\{cS_0, z^*\}}{\sqrt{m/4 + (\min\{cS_0, z^*\})^{2^{m+1}}}}\right)$$

where $z^* = \left(\frac{m/2}{2^{m+1}-2}\right)^{2^{-m-1}}$.

Proof: *See Appendix A.0.6.* □

Therefore, we have obtained, in LSH terminology,

$$p_1 = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0}{\sqrt{m/4 + U^{2^{m+1}}}}\right) \tag{7.20}$$

$$p_2 = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\min\{cS_0, z^*\}}{\sqrt{m/4 + (\min\{cS_0, z^*\})^{2^{m+1}}}}\right), \tag{7.21}$$

$$z^* = \left(\frac{m/2}{2^{m+1}-2}\right)^{2^{-m-1}} \tag{7.22}$$

Theorem 15 allows us to construct data structures with worst case $O(n^\rho \log n)$ query time guarantees for c-approximate MIPS, where $\rho = \frac{\log p_1}{\log p_2}$. For any given

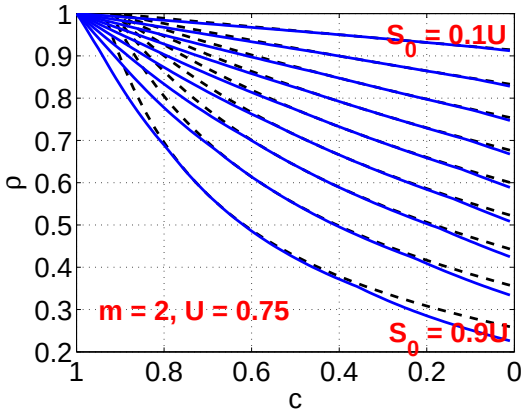

Figure 7.7: Optimal ρ values of Sign-ALSH and the ρ values for fixed parameters: $m = 2$ and $U = 0.75$.

$c < 1$, there always exist $U < 1$ and m such that $\rho < 1$. This way, we obtain a sublinear query time algorithm for MIPS. Because ρ is a function of 2 parameters, the best query time chooses U and m, which minimizes the value of ρ. For convenience, we define

$$\rho^* = \min_{U,m} \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0}{\sqrt{m/4 + U^{2^{m+1}}}}\right)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\min\{cS_0, z^*\}}{\sqrt{m/4 + (\min\{cS_0, z^*\})^{2^{m+1}}}}\right)\right)} \tag{7.23}$$

See Figure 7.6 for the plots of ρ^*, which also compares the optimal ρ values for L2-ALSH in the prior work [80]. Sign-ALSH is noticeably better.

Parameter Selection

Figure 7.7 presents the ρ values for a selected parameters: $(m, U) = (2, 0.75)$ We can see that even if we use fixed parameters, the performance would not degrade much. This frees practitioners from the burden of choosing parameters.

7.5.4 Removing the Dependence on the Norm of the Query.

We can use the same idea used in Section 7.3.3 to get rid of the norm of q. Let M be the upper bound on all the norms i.e. $M = max_{x \in C} \|x\|_2$. In other words M is the radius of the space.

Let $U < 1$, define the transformations, $T : \mathbb{R}^D \to \mathbb{R}^D$ as

$$T(x) = \frac{Ux}{M} \tag{7.24}$$

and transformations $P, Q : \mathbb{R}^D \to \mathbb{R}^{D+m}$ are the same for the Sign-ALSH scheme as defined in Eq (7.16) and (7.17).

Given the query q and any data point x, observe that the inner products between $P(Q(T(q)))$ and $Q(P(T(x)))$ is

$$P(Q(T(q)))^T Q(P(T(x))) = q^T x \times \left(\frac{U^2}{M^2}\right) \tag{7.25}$$

$P(Q(T(q)))$ appends first m zeros components to $T(q)$ and then m components of the form $1/2 - \|q\|^{2^i}$. $Q(P(T(q)))$ does the same thing but in a different order. Now we are working in $D + 2m$ dimensions. It is not difficult to see that the norms of $P(Q(T(q)))$ and $Q(P(T(q)))$ is given by

$$\|P(Q(T(q)))\|_2 = \sqrt{\frac{m}{4} + \|T(q)\|_2^{2^{m+1}}} \tag{7.26}$$

$$\|Q(P(T(x)))\|_2 = \sqrt{\frac{m}{4} + \|T(x)\|_2^{2^{m+1}}} \tag{7.27}$$

The transformations are very asymmetric but we know that it is necessary.

Therefore the correlation or the cosine similarity between $P(Q(T(q)))$ and $Q(P(T(x)))$ is

$$Corr = \frac{q^T x \times \left(\frac{U^2}{M^2}\right)}{\sqrt{\frac{m}{4} + \|T(q)\|_2^{2^{m+1}}} \sqrt{\frac{m}{4} + \|T(x)\|_2^{2^{m+1}}}} \tag{7.28}$$

Note $\|T(q)\|_2^{2^{m+1}}, \|T(x)\|_2^{2^{m+1}} \leq U < 1$, therefore both $\|T(q)\|_2^{2^{m+1}}$ and $\|T(x)\|_2^{2^{m+1}}$ converge to zero at a tower rate and we get approximate monotonicity of correlation with the inner products. We can apply sign random projections to hash $P(Q(T(q)))$ and $Q(P(T(q)))$.

Using the fact $0 \leq \|T(q)\|_2^{2^{m+1}} \leq U$ and $0 \leq \|T(x)\|_2^{2^{m+1}} \leq U$, it is not difficult to get p_1 and p_2 for Sign-ALSH, without any conditions on any norms. Simplifying the expression, we get the following value of optimal ρ_u (u for unrestricted).

$$\rho_u^* = \min_{U,m,} \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0 \times \left(\frac{U^2}{M^2}\right)}{\frac{m}{4} + U^{2^{m+1}}}\right)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{cS_0 \times \left(\frac{4U^2}{M^2}\right)}{m}\right)\right)} \tag{7.29}$$
$$s.t. \;\; U^{2^{m+1}} < \frac{m(1-c)}{4c}, \; m \in \mathbb{N}^+, \text{ and } 0 < U < 1.$$

With this value of ρ_u^*, we can state our main theorem.

Theorem 20 *For the problem of c-approximate MIPS in a bounded space, one can construct a data structure having $O(n^{\rho_u^*} \log n)$ query time and space $O(n^{1+\rho_u^*})$, where $\rho_u^* < 1$ is the solution to constraint optimization (7.29).*

Note, for all $c < 1$, we always have $\rho_u^* < 1$ because the constraint $U^{2^{m+1}} < \frac{m(1-c)}{4c}$ is always true for big enough m. The only assumption for efficiently solving MIPS that we need is that the space is bounded, which is always satisfied for any finite dataset. ρ_u^* depends on M, the radius of the space, which is expected.

7.6 Random Space Partitioning with ALSH

In this section, we show that due to the nature of the new transformations P and Q from there is one subtle but surprising advantage of Sign-ALSH over

L2-ALSH.

One popular application of LSH (Locality Sensitive Hashing) is random partitioning of the data for large scale clustering, where similar points map to the same partition (or bucket). Such partitions are very useful in many applications [37]. With classical LSH, we simply use $h(x)$ to generate partition for x. Since $Pr_{\mathcal{H}}(h(x) = h(y))$ is high if $sim(x, y)$ is high, similar points are likely to go into the same partition under the usual LSH mapping. For general ALSH, this property is lost because of asymmetry.

In case of ALSH, we only know that $Pr(h(P(x)) = h(Q(y))$ is high if $sim(x, y)$ is high. Therefore, given x we cannot determine whether to assign partition using $h(P(.))$ or $h(Q(.))$. Neither $Pr(h(P(x)) = h(P(y))$ nor $Pr_{\mathcal{H}}(h(Q(x)) = h(Q(y))$ strictly indicates high value of $sim(x, y)$ in general. Therefore, partitioning property of classical LSH does not hold anymore with general ALSHs. However for the case of inner products using Sign-ALSH, there is a subtle observation which allows us to construct the required assignment function, where pairs of points with high inner products are more likely to get mapped in the same partition while pairs with low inner products are more likely to map into different partitions.

In case of Sign-ALSH for MIPS, we have the transformations $P(Q(T(x)))$ and $Q(P(T(x)))$ given by

$$P(Q(T(x))) = [x; 1/2 - \|T(x)\|_2^2; ....; 1/2 - \|T(x)\|_2^{2^m}, 0, ..., 0]$$

$$Q(P(T(x))) = [x; 0, ..., 0, 1/2 - \|T(x)\|_2^2; ....; 1/2 - \|T(x)\|_2^{2^m}].$$

After this transformation, we multiply the generated $D + 2m$ dimensional vector by a random vector $a \in \mathbb{R}^{D+2m}$ whose entries are i.i.d. Gaussian followed by taking the sign. For illustration let $a = [w; s_1, ...s_m, t_1, ...t_m]$ where $w \in \mathbb{R}^D$ b_i and c_i are numbers. All components of a are i.i.d. from $N(0, 1)$. With this notation, we

can write the final Sign-ALSH as

$$h^{Sign}(P(Q(T(x)))) = Sign(w^T T(x) + \sum_{i=1}^{m} s_i(1/2 - ||T(x)||_2^{2^i}))$$

$$h^{Sign}(Q(P(T(x)))) = Sign(w^T T(x) + \sum_{i=1}^{m} t_i(1/2 - ||T(x)||_2^{2^i}))$$

The key observation here is that $h^{Sign}(P(Q(T(x))))$ does not depend on t_is and $h^{Sign}(Q(P(T(x))))$ does not depend on s_is. If we define a mapping (it is not exactly a mapping as it involve independent random α_i for every evaluation)

$$h_w(x) = Sign(w^T T(x) + \sum_{i=1}^{m} \alpha_i(1/2 - ||T(x)||_2^{2^i})) \tag{7.30}$$

where α_i are sampled i.i.d. from $N(0, 1)$ for every x independently of everything else. Then, **under the randomization of** w, it is not difficult to show that

$$Pr_w(h_w(x) = h_w(y)) = Pr(h^{Sign}(P(x)) = h^{Sign}(Q(y)))$$

for any x and y. The term $Pr(h^{Sign}(P(x)) = h^{Sign}(Q(y)))$ satisfies the LSH like property and therefore, in any partitions using h_w, points with high inner products are more likely to be together. Thus, $h_w(x)$ is the required assignment. Note, h_w is not technically an LSH because we are randomly sampling α_i for all x independently. The construction of h_w using independent randomizations could be of separate interest in itself. To the best of our knowledge, this is the first example of LSH like partition using hash function with independent randomization for every data point.

The function h_w is little subtle here, we sample w i.i.d from Gaussian and use the same w for all x, but while computing h_w we use α_i independent of everything for every x. The probability is under the randomization of w and independence of all α_i ensures the asymmetry. We are not sure if such construction is possible with L2-ALSH. For LSH partitions with binary data, a very same idea,

used here, can be applied to asymmetric minwise hashing which is discussed later in Section 7.7.1.

7.6.1 Ranking Experiments

In Section 7.6.1, the L2-ALSH scheme was shown to outperform other reasonable heuristics in retrieving maximum inner products. Since our proposal is an improvement over L2-ALSH, we first focus on comparisons with L2-ALSH. In this section, we compare L2-ALSH with Sign-ALSH based on ranking.

7.6.2 Datasets

We use three publicly available dataset MNIST, WEBSPAM and RCV1 for evaluations. For each of the three dataset we generate two independent partitions, the query set and the train set. Each element in the query set is used for querying, while the training set serves as the collection C that will be searched for MIPS. The statistics of the dataset and the partitions are summarized in Table 7.1

Dataset	Dimension	Query size	Train size
MNIST	784	10,000	60,000
WEBSPAM	16,609,143	5,000	100,000
RCV1	47,236	5,000	100,000

Table 7.1: Datasets used for evaluations.

In this section, we show how the ranking of the two ALSH schemes, L2-ALSH and Sign-ALSH, correlates with inner products. Given a query vector q, we compute the top-10 gold standard elements based on the actual inner products $q^T x$, $\forall x \in C$, here our collection is the train set. We then generate K

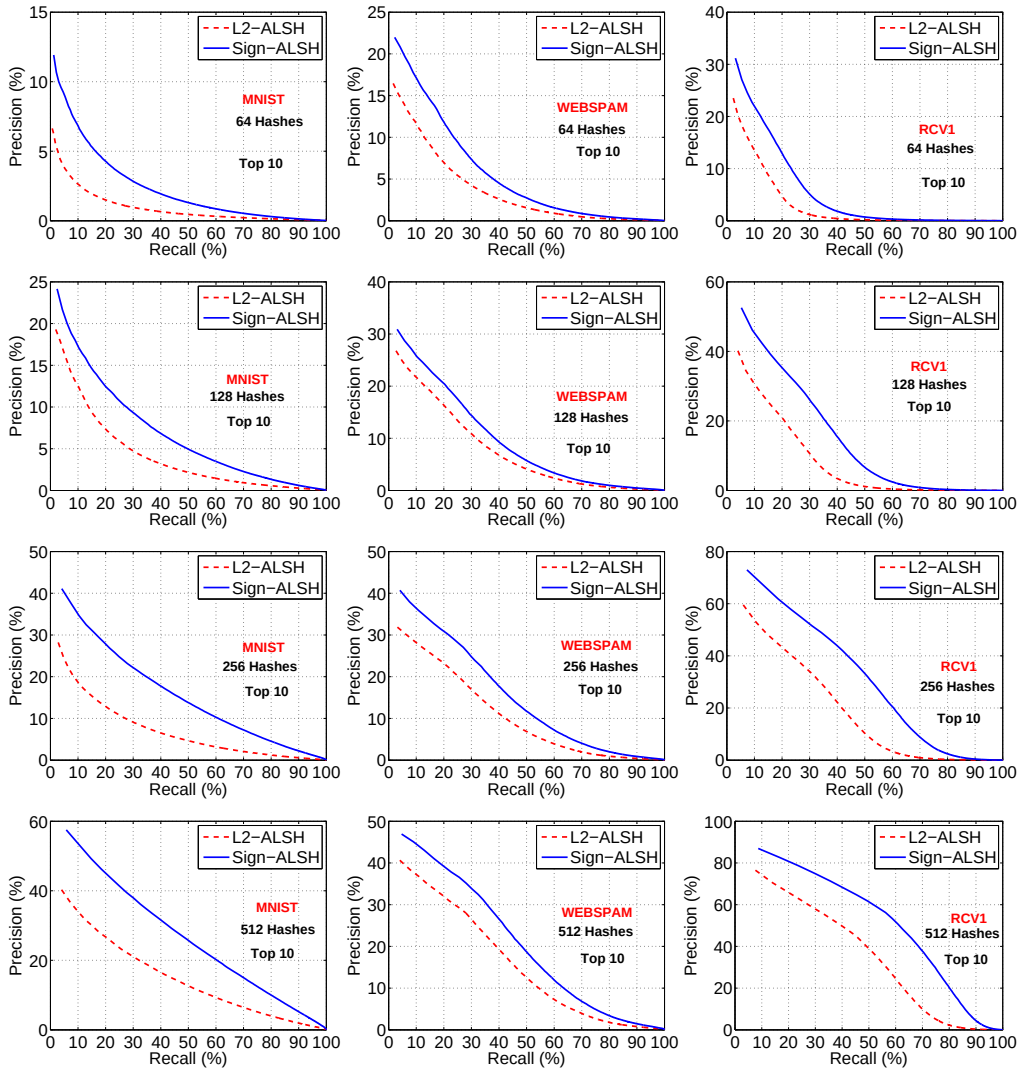

Figure 7.8: Precision-Recall curves for L2-ALSH and the improved Sign-ALSH for retrieving top-10 elements.

different hash codes of the query q and all the elements $x \in C$ and then compute

$$Matches_x = \sum_{t=1}^{K} \mathbf{1}(h_t(Q(q)) = h_t(P(x))), \tag{7.31}$$

where $\mathbf{1}$ is the indicator function and the subscript t is used to distinguish independent draws of h. Based on $Matches_x$ we rank all the elements x. Ideally, for a better hashing scheme, $Matches_x$ should be higher for element x having higher inner products with the given query q. This procedure generates a sorted list

of all the items for a given query vector q corresponding to the each of the two asymmetric hash functions under consideration.

For L2-ALSH, we used the same parameters used and recommended in [80]. For Sign-ALSH, we used the recommended choice shown in Section 7.5.3, which is $U = 0.75$, $m = 2$. Note that Sign-ALSH does not have parameter r.

We compute precision and recall of the top-10 gold standard elements, obtained from the sorted list based on $Matches_x$. To compute this precision and recall, we start at the top of the ranked item list and walk down in order. Suppose we are at the k^{th} ranked item, we check if this element belongs to the gold standard top-10 list. If it is one of the top-10 gold standard elements, then we increment the count of *relevant seen* by 1, else we move to $k + 1$. By k^{th} step, we have already seen k elements, so the *total items seen* is k. The precision and recall at that point is then computed as:

$$Precision = \frac{\text{relevant seen}}{k}, \qquad Recall = \frac{\text{relevant seen}}{10}$$

We show performance for $K \in \{64, 128, 256, 512\}$. Note that it is important to balance both precision and recall. The method which obtains higher precision at a given recall is superior. Higher precision indicates higher ranking of the top-10 inner products which is desirable. We report averaged precisions and recalls.

The plots for all the three datasets are shown in Figure 7.8. We can clearly see, that our proposed Sign-ALSH scheme gives significantly higher precision recall curves than the L2-ALSH scheme, indicating better correlation of top inner products with Sign-ALSH compared to L2-ALSH. The results are consistent across datasets.

7.6.3 Comparisons of Hashing Based and Tree Based Methods

	Sign-ALSH	L2-ALSH [80]	Cone Trees [76]
MNIST	**7,944**	9,971	11,202
WEBSPAM	**2,866**	3,813	22,467
RCV1	**9,951**	11,883	38,162

Table 7.2: Mean number of inner products evaluated per query by different algorithms for MIPS.

We have shown in the previous Section that Sign-ALSH outperforms L2-ALSH in ranking evaluations. In this Section, we consider the actual task of finding the maximum inner product. Our aim is to estimate the computational saving, in finding the maximum inner product, with Sign-ALSH compared to the existing scheme L2-ALSH. In addition to L2-ALSH which is a hashing scheme, there is an another tree based space partitioning method [76] for solving MIPS. Although, in theory, it is know that tree based methods perform poorly [89] due to their exponential dependence on the dimensionality, it is still important to understand the impact of such dependency in practice. Unfortunately no empirical comparison between hashing and tree based methods exists for the problem of MIPS in the literature. To provide such a comparison, we also consider tree based space partitioning method [76] for evaluations. We use the same three datasets as described in Section 7.6.2.

Tree based and hashing based methodologies are very different in nature. The major difference is in the stopping criteria. Hashing based methods create buckets and stop the search once they find a good enough point, they may not succeed with some probability. On the other hand, tree based methods use branch and bound criteria to stop exploring further. So it is possible that a tree based algorithm finds the optimal point but continues to explore further requiring more computations. The usual stopping criteria thus makes tree based

methods unnecessarily expensive compared to hashing based methods where the criteria is to stop after finding a good point. Therefore, to ensure fair comparisons, we allow the tree based method to stop the evaluations immediately once the algorithm finds the maximum inner product and prevent it from exploring further. Also, in case when hashing based algorithm fails to find the best inner product we resort to the full linear scan and penalize the hashing based algorithm for not succeeding. All this is required to ensure that tree based algorithm is not at any disadvantage compare to hashing methods.

We implemented the bucketing scheme with Sign-ALSH and L2-ALSH. The bucketing scheme requires creating many hash tables during the preprocessing stage. During query phase, given a query, we compute many hashes of the query and probe appropriate buckets in each table. Please refer [6] for more details on the process. We use the same fixed parameters for all the evaluations, i.e., (m=2, U=0.75) for Sign-ALSH and (m=3, U=0.83, r=2.5) for L2-ALSH as recommended in [80]. The total number of inner products evaluated by a hashing scheme, for a given query, is the total number of hash computation for the query plus the total number of points retrieved from the hash tables. In rare cases, with very small probability, if the hash tables are unable to retrieve the gold standard maximum inner product, we resort to linear scan and also include the total number of inner products computed during the linear scan. We stop as soon as we reach the gold standard point.

We implemented Algorithm 5 from [76], which is the best performing algorithm as shown in the evaluations. For this algorithm, we need to select one parameter which is the minimum number of elements in the node required for splitting. We found that on all the three datasets the value of 100 for this pa-

rameter works the best among $\{500,\ 200,\ 100, 50\}$. Therefore, we use 100 in all our experiments. The total number of inner products evaluated by tree based algorithm is the total number of points reported plus the total number of nodes visited, where we compute the branch and bound constraint. Again we stop the search process as soon as we reach the point with gold standard maximum inner product. As argued, we need this common stopping condition to compare with hashing based methods, where we do not have any other stopping criteria [42].

For every query we compute the number of inner products evaluated by different methods for MIPS. We report the mean of the total number of inner products evaluated per query in Table 7.2. We can clearly see that hashing based methods are always better than the tree based algorithm. Except on MNIST dataset, hashing based methods are significantly superior, which is also not surprising because MNIST is an image dataset having low intrinsic dimensionality. Among the two hashing schemes Sign-ALSH is always better than L2-ALSH, which verifies our theoretical findings and supports our arguments in favor of Sign-ALSH over L2-ALSH for MIPS.

7.7 Even Better ALSH for Binary MIPS

We know that binary representations for web data are common, owing to the wide adoption of the "bag of words (n-gram)" representations for documents and images. In this Section, we show that for binary data there is even a better ALSH than Sign-ALSH. The new ALSH makes use of MinHash.

MinHash is one of the widely used indexing scheme for search and record matching. The underlying similarity measure of interest with MinHash is the

resemblance (also known as the Jaccard similarity). The resemblance similarity between two sets $x, y \subseteq \Omega = \{1, 2, ..., D\}$ is

$$\mathcal{R} = \frac{|x \cap y|}{|x \cup y|} = \frac{a}{f_x + f_y - a}, \tag{7.32}$$
$$\text{where } f_x = |x|, \quad f_y = |y|, \quad a = |x \cap y|.$$

While the resemblance similarity is convenient and useful in numerous applications, there are also many scenarios where the resemblance is not the desired similarity measure [3, 19]. For instance, consider text descriptions of two restaurants:

1. "Five Guys Burgers and Fries Downtown Brooklyn New York"
2. "Five Kitchen Berkley"

Shingle (n-gram) based representations for strings are common in practice. Typical (first-order) shingle based representations of these names will be (i) {five, guys, burgers, and, fries, downtown, brooklyn, new, york } and (ii) {five, kitchen, berkley}. Now suppose the query is "Five Guys" which in shingle representation is {Five, Guys}. Suppose we hope to match and search the records, for this query "Five Guys", based on resemblance. Observe that the resemblance between query and record (i) is $\frac{2}{9} = 0.22$, while that with record (ii) is $\frac{1}{4} = 0.25$. Thus, simply based on resemblance, record (ii) is a better match for query "Five Guys" than record (i), which however should not be correct in this content.

Clearly the issue here is that resemblance penalizes the sizes of the sets involved. Shorter sets are unnecessarily favored over longer ones, which hurts the performance in (e.g.,) record matching [3]. There are other scenarios where such

penalization is undesirable. For instance, in plagiarism detection, it is typically immaterial whether the text is plagiarized from a long or a short document.

To counter the often unnecessary penalization of the sizes of the sets with resemblance, a modified measure, the *set containment* (or Jaccard containment) was adopted [12, 3, 19]. Containment of set x and y with respect to x is defined as

$$\mathcal{J}_C = \frac{|x \cap y|}{|x|} = \frac{a}{f_x}. \tag{7.33}$$

In the above example with query "Five Guys", the set containment with respect to query for record (i) will be $\frac{2}{2} = 1$ and with respect to record (ii) it will be $\frac{1}{2}$, leading to the desired ordering. It should be noted that for any fixed query x, the ordering under set containment with respect to the query, is the same as the ordering with respect to the intersection a (or binary inner product). Thus, near neighbor search problem with respect to $\mathcal{J}_C$ is equivalent to the maximum inner product search problem or MIPS.

Both L2-ALSH and Sign-ALSH construccted in previous Sections work for any general inner products over $\mathbb{R}^D$. For sparse and high-dimensional binary dataset which are common over the web, we showed that MinHash is typically the preferred choice of hashing over random projection based hash functions [84]. Therefore it is worth investigating if we can make use of MinHash and asymmetric transformations to come up with a better ALSH for MIPS. In this section, we show an affirmative result that we can actually derive better ALSH for binary MIPS using MinHash. We call the new scheme asymmetric minwise hashing (*MH-ALSH*), which is more suitable for indexing binary inner products compared to the existing ALSHs for general inner products.

In Theorem 14, we showed that there cannot exist any LSH for general un-

normalized inner product. The argument relies on the fact that we can have pairs x and y, s.t. $x^T y \gg x^T x$. This argument does not hold true for binary inner products and we always have $x^T y \leq x^T x \; \forall$ binary x and y. But, using a slightly different argument it can still be shown that even for binary MIPS we cannot have any LSH scheme. For binary inner product we can have x, y and z such that $x^T y >> z^T z$. Now, $Pr(h(x) = h(y)) > Pr(h(z) = h(z)) = 1$ is again impossible.

7.7.1 Asymmetric Minwise Hashing

In this section, we provide a very simple asymmetric fix to MinHash, named *asymmetric minwise hashing (MH-ALSH)*, which makes the overall collision probability monotonic in the original inner product a.

Following the general ALSH framework, we define the new preprocessing and query transformations $P' : [0,1]^D \rightarrow [0,1]^{D+M}$ and $Q' : [0,1]^D \rightarrow [0,1]^{D+M}$ as:

$$P'(x) = [x; 1; 1; 1; ...; 1; 0; 0; ...; 0] \tag{7.34}$$

$$Q'(x) = [x; 0; 0; 0; ...; 0], \tag{7.35}$$

For $P'(x)$ we append $M - f_x$ 1s and rest f_x zeros, while in $Q'(x)$ we simply append M zeros.

At this point we can already see the power of asymmetric transformations. The original inner product between $P'(x)$ and $Q'(x)$ is unchanged and its value is $a = x^T y$. Given the query q, the new resemblance R' between $P'(x)$ and $Q'(q)$ is

$$R' = \frac{|P'(x) \cap Q'(q)|}{|P'(x) \cup Q'(q)|} = \frac{a}{M + f_q - a}. \tag{7.36}$$

If we define our new similarity as $Sim(x, y) = \frac{a}{M+f_q-a}$, then the near neighbors in this new similarity are the same as near neighbors with respect to either set

intersection a or set containment $\frac{a}{f_q}$. Thus, we can instead compute near neighbors in $\frac{a}{M+f_q-a}$ which is also the resemblance between $P'(x)$ and $Q'(q)$. We can therefore use MinHash on $P'(x)$ and $Q'(q)$.

Observe that now we have $M + f_q - a$ in the denominator, where M is the maximum nonzeros seen in the dataset (the cardinality of largest set), which for very sparse data is likely to be much smaller than D. Thus, asymmetric MinHash is a better scheme than $\mathcal{H}_S$ with collision probability $\frac{a}{D}$ for very sparse datasets where we usually have $M \ll D$.

From theoretical perspective, to obtain an upper bound on the query and space complexity of c-approximate near neighbor with binary inner products, we want the collision probability to be independent of the quantity f_q. This is not difficult to achieve. The asymmetric transformation used to get rid of f_x in the denominator can be reapplied to get rid of f_q.

Formally, we can define $P'' : [0,1]^D \to [0,1]^{D+2M}$ and $Q'' : [0,1]^D \to [0,1]^{D+2M}$ as :

$$P''(x) = Q'(P'(x)); \qquad Q''(x) = P'(Q'(x)); \tag{7.37}$$

where in $P''(x)$ we append $M - f_x$ 1s and rest $M + |f_x|$ zeros, while in $Q''(x)$ we append M zeros, then $M - f_q$ 1s and rest zeros

Again the inner product a is unaltered, and the new resemblance then becomes

$$R'' = \frac{|P''(x) \cap Q''(q)|}{|P''(x) \cup Q''(q)|} = \frac{a}{2M - a}. \tag{7.38}$$

which is independent of f_q and is monotonic in a. This allows us to achieve a formal upper bound on the complexity of c-approximate maximum inner product search with the new asymmetric MinHash.

From the collision probability expression, i.e., Eq. (7.38), we have

Theorem 21 *Minwise hashing along with Query transformation Q'' and Preprocessing transformation P'' defined by Equation 7.37 is a $(S_0, cS_0, \frac{S_0}{2M-S_0}, \frac{cS_0}{2M-cS_0})$ sensitive asymmetric hashing family for set intersection.*

This leads to an important corollary.

Corollary 5 *There exists an algorithm for c-approximate set intersection, with bounded sparsity M, that requires $O(n^{1+\rho_{MH-ALSH}})$ space and $O(n^{\rho}_{MH-ALSH} \log n)$, where*

$$\rho_{MH-ALSH} = \frac{\log \frac{S_0}{2M-S_0}}{\log \frac{cS_0}{2M-cS_0}} < 1 \tag{7.39}$$

Given query q and any point $x \in C$, the collision probability under traditional MinHash is $R = \frac{a}{f_x+f_q-a}$. This penalizes sets with high f_x, which in many scenarios is not desirable. To balance this negative effect, asymmetric transformation penalizes sets with smaller f_x. Note, that $M - f_x$ ones added in the transformations $P'(x)$ gives additional chance in proportion to $M - f_x$ for MinHash of $P'(x)$ not to match with the MinHash of $Q'(x)$. This asymmetric probabilistic correction balances the penalization inherent in MinHash. This is a simple way of correcting the probability of collision which could be of independent interest in itself. We will show in our evaluation section, that despite this simplicity such correction leads to significant improvement over plain MinHash.

7.7.2 Efficient Sampling

Our transformations P'' and Q'' always create sets with $2M$ nonzeros. In case when M is big, hashing might take a lot of time. We can use (improved) con-

sistent weighted sampling [67, 43] for efficient generation of hashes. We can instead use transformations P''' and Q''' that makes the data non-binary as follows

$$P'''(x) = [x; M - f_x; 0] \tag{7.40}$$
$$Q'''(x) = [x; 0; M - f_x]$$

It is not difficult to see that the weighted resemblance (or weighted Jaccard similarity) between $P'''(x)$ and $Q'''(q)$ for given query q and any $x \in C$ is

$$\mathcal{R}_W = \frac{\sum_i \min(P'''(x)_i, Q'''(q)_i)}{\sum_i \max(P'''(x)_i, Q'''(q)_i)} = \frac{a}{2M - a}. \tag{7.41}$$

Therefore, we can use fast consistent weighted sampling for weighted resemblance on $P'''(x)$ and $Q'''(x)$ to compute the hash values in time constant per nonzero weights, rather than maximum sparsity M. In practice we will need many hashes for which we can utilize the recent line of work that make MinHash and weighted MinHash significantly much faster [60, 81, 82, 35].

7.7.3 Theoretical Comparisons

For solving the MIPS problem in general data types, we already know two asymmetric hashing schemes, *L2-ALSH* and *Sign-ALSH*, as described in Section 7.3 and Section 7.5 respectively. In this section, we provide theoretical comparisons of the two existing ALSH methods with the proposed asymmetric minwise hashing (*MH-ALSH*).

Before we formally compare various asymmetric LSH schemes for maximum inner product search, we argue why asymmetric MinHash should be advantageous over traditional MinHash for retrieving inner products. Let q be the

binary query vector, and f_q denotes the number of nonzeros in the query. The $\rho_{MH-ALSH}$ for asymmetric MinHash in terms of f_q and M is straightforward:

$$\rho^q_{MH-ALSH} = \frac{\log \frac{S_0}{f_q+M-S_0}}{\log \frac{cS_0}{f_q+M-cS_0}} \tag{7.42}$$

For MinHash, we have $\rho^q_{min} = \frac{\log \frac{S_0}{f_q+M-S_0}}{\log \frac{cS_0}{f_q}}$ (see [85] for details). Since M is the upper bound on the sparsity and cS_0 is some value of inner product, we have $M - cS_0 \geq 0$. Using this fact, the following theorem immediately follows

Theorem 22 *For any query q, we have $\rho^q_{MH-ALSH} \leq \rho^q_{min}$.*

This result theoretically explains why asymmetric MinHash is better for retrieval with binary inner products, compared to plain MinHash.

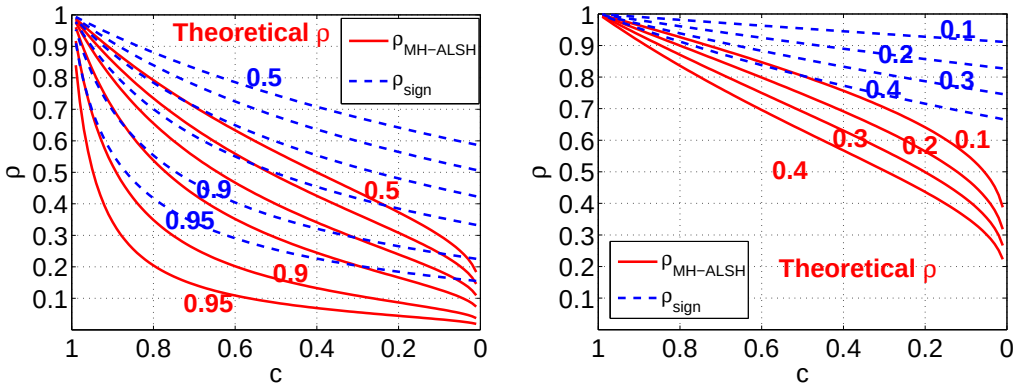

Figure 7.9: Comparisons of $\rho_{MH-ALSH}$ and ρ_{sign} (lower is better) with respect to approximation ratio c for different $\frac{S_0}{M}$.

For comparing asymmetric MinHash with ALSH for general inner products, we compare $\rho_{MH-ALSH}$ with the ALSH for inner products based on sign random projections. In Section 7.5 it was shown that Sign-ALSH has better theoretical ρ values compared to L2-ALSH. Therefore, it suffices to show that asymmetric

MinHash outperforms sign random projection based ALSH. Both $\rho_{MH-ALSH}$ and ρ_{sign} can be rewritten in terms of ratio $\frac{S_0}{M}$ as follows. Note that for binary data we have $M = \max_{x \in C} \|x\|^2 = V^2$

$$\rho_{MH-ALSH} = \frac{\log \frac{S_0/M}{2-S_0/M}}{\log \frac{cS_0/M}{2-cS_0/M}}; \quad \rho_{Sign} = \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0}{M}\right)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{cS_0}{M}\right)\right)} \tag{7.43}$$

Observe that M is also the upper bound on any inner product. Therefore, we have $0 \leq \frac{S_0}{M} \leq 1$. We plot the values of $\rho_{MH-ALSH}$ and ρ_{sign} for $\frac{S_0}{M} = \{0.1,\ 0.2, ..., 0.8,\ 0.9,\ 0.95\}$ with c. The comparison is summarized in Figure 7.9. Note that here we use ρ_{Sign} based on the slightly more convenient transformation from [9, 69] instead of $\rho_{Sign-ALSH}$ for convenience although the two schemes perform essentially the same.

Clearly, irrespective of the choice of threshold $\frac{S_0}{M}$ or the approximation ratio c, asymmetric MinHash outperforms sign random projection based ALSH in terms of the theoretical ρ values. This is not surprising, because it is known that MinHash based methods are often significantly powerful for binary data compared to SRP (or SimHash) [84]. Therefore ALSH based on MinHash outperforms ALSH based on SRP as shown by our theoretical comparisons. Our proposal thus leads to an algorithmic improvement over state-of-the-art hashing techniques for retrieving binary inner products.

7.7.4 Evaluations

In this section, we compare the different hashing schemes on the actual task of retrieving top-ranked elements based on set Jaccard containment. The experiments are divided into two parts. In the first part, we show how the ranking

based on various hash functions correlate with the ordering of set containment. In the second part, we perform the actual LSH based bucketing experiment for retrieving top-ranked elements and compare the computational saving obtained by various hashing algorithms.

Datasets

We chose four publicly available high dimensional sparse datasets: *EP2006*, *MNIST*, *NEWS20*, and *NYTIMES*. (Note that "EP2006" is a short name for "E2006LOG1P" from LIBSVM web site.) Except for MNIST, the other three datasets are high dimensional binary "BoW" representation of the corresponding text corpus. MNIST is an image dataset consisting of 784 pixel image of handwritten digits. Binarized versions of MNIST are commonly used in literature. The pixel values in MNIST were binarized to 0 or 1 values. For each of the four datasets, we generate two partitions. The bigger partition was used to create hash tables and is referred as the *training partition*. The small partition which we call the *query partition* is used for querying. The statistics of these datasets are summarized in Table 7.3. The datasets cover a wide spectrum of sparsity and dimensionality.

Table 7.3: Datasets

Dataset	# Query	# Train	# Dim	nonzeros (mean ± std)
EP2006	2,000	17,395	4,272,227	6072 ± 3208
MNIST	2,000	68,000	784	150 ± 41
NEWS20	2,000	18,000	1,355,191	454 ± 654
NYTIMES	2,000	100,000	102,660	232 ± 114

Competing Hash Functions

We consider the following hash functions for evaluations:

1. **Asymmetric minwise hashing (Proposed):** This is our proposal, the asymmetric MinHash described in Section 7.7.1.
2. **Traditional minwise hashing (MinHash):** This is the usual minwise hashing, the popular heuristic described in Section 2.2.2. This is a symmetric hash function, we use h_{π}^{min} as define in Eq.(2.4) for both query and the training set.
3. **L2 based Asymmetric LSH for Inner products (L2-ALSH):** This is the first asymmetric LSH described in Section 7.3 for general inner products based on LSH for L2 distance.
4. **SRP based Asymmetric LSH for Inner Products (Sign-ALSH):** This is the imporved asymmetric hash function defined in Section 7.5 for general inner products based on SRP.

Ranking Experiment: Hash Quality Evaluations

We are interested in knowing, how the orderings under different competing hash functions correlate with the ordering of the underlying similarity measure which in this case is the set containment. For this task, given a query q vector, we compute the top-100 gold standard elements from the training set based on the set containment $\frac{a}{f_q}$. Note that this is the same as the top-100 elements based on binary inner products. Give a query q, we compute K different hash codes of the vector q and all the vectors in the training set. We then compute the number

of times the hash values of a vector x in the training set matches (or collides) with the hash values of query q defined by

$$Matches_x = \sum_{t=1}^{K} \mathbf{1}(h_t(q) = h_t(x)), \tag{7.44}$$

where $\mathbf{1}$ is the indicator function. t subscript is used to distinguish independent draws of the underlying hash function. Based on $Matches_x$ we rank all elements in the training set. This procedure generates a sorted list for every query for every hash function. For asymmetric hash functions, in computing total collisions, on the query vector we use the corresponding Q function (query transformation) followed by underlying hash function, while for elements in the training set we use the P function (preprocessing transformation) followed by the corresponding hash function.

We compute the precision and the recall of the top-100 gold standard elements in the ranked list generated by different hash functions. To compute precision and recall, we start at the top of the ranked item list and walk down in order, suppose we are at the p^{th} ranked element, we check if this element belongs to the gold standard top-100 list. If it is one of the top 100 gold standard elements, then we increment the count of *relevant seen* by 1, else we move to $p + 1$. By p^{th} step, we have already seen p elements, so the *total elements seen* is p. The precision and recall at that point is then computed as:

$$Precision = \frac{\text{relevant seen}}{p}, \quad Recall = \frac{\text{relevant seen}}{100} \tag{7.45}$$

It is important to balance both. Methodology which obtains higher precision at a given recall is superior. Higher precision indicates higher ranking of the relevant items. We finally average these values of precision and recall over all elements in the query set. The results for $K \in \{32,\ 64,\ 128\}$ are summarized in Figure 7.10.

We can clearly see, that the proposed hashing scheme always achieves better, often significantly, precision at any given recall compared to other hash functions. The two ALSH schemes are usually always better than traditional minwise hashing. This confirms that fact that ranking based on collisions under minwise hashing can be different from the rankings under set containment or inner products. This is expected, because MinHash in addition penalizes the number of nonzeros leading to a ranking very different from the ranking of inner products. Sign-ALSH usually performs better than L2-LSH, this is in line with the results obtained in Section 7.5.

It should be noted that ranking experiments only validate the monotonicity of the collision probability. Although, better ranking is certainly a good indicator of good hash function, it does not always mean that we will achieve faster sub-linear LSH algorithm. For bucketing the probability sensitivity around a particular threshold is the most important factor, see [74] for more details. What matters is the **gap** between the collision probability of good and the bad points. In the next subsection, we compare these schemes on the actual task of near neighbor retrieval with set containment.

LSH Bucketing Experiment: Computational Savings in Near Neighbor Retrieval

In this section, we evaluate the four hashing schemes on the standard (K, L)-parameterized bucketing algorithm [6] for sub-linear time retrieval of near neighbors based on set containment. In (K, L)-parameterized LSH algorithm, we generate L different meta-hash functions. Each of these meta-hash functions

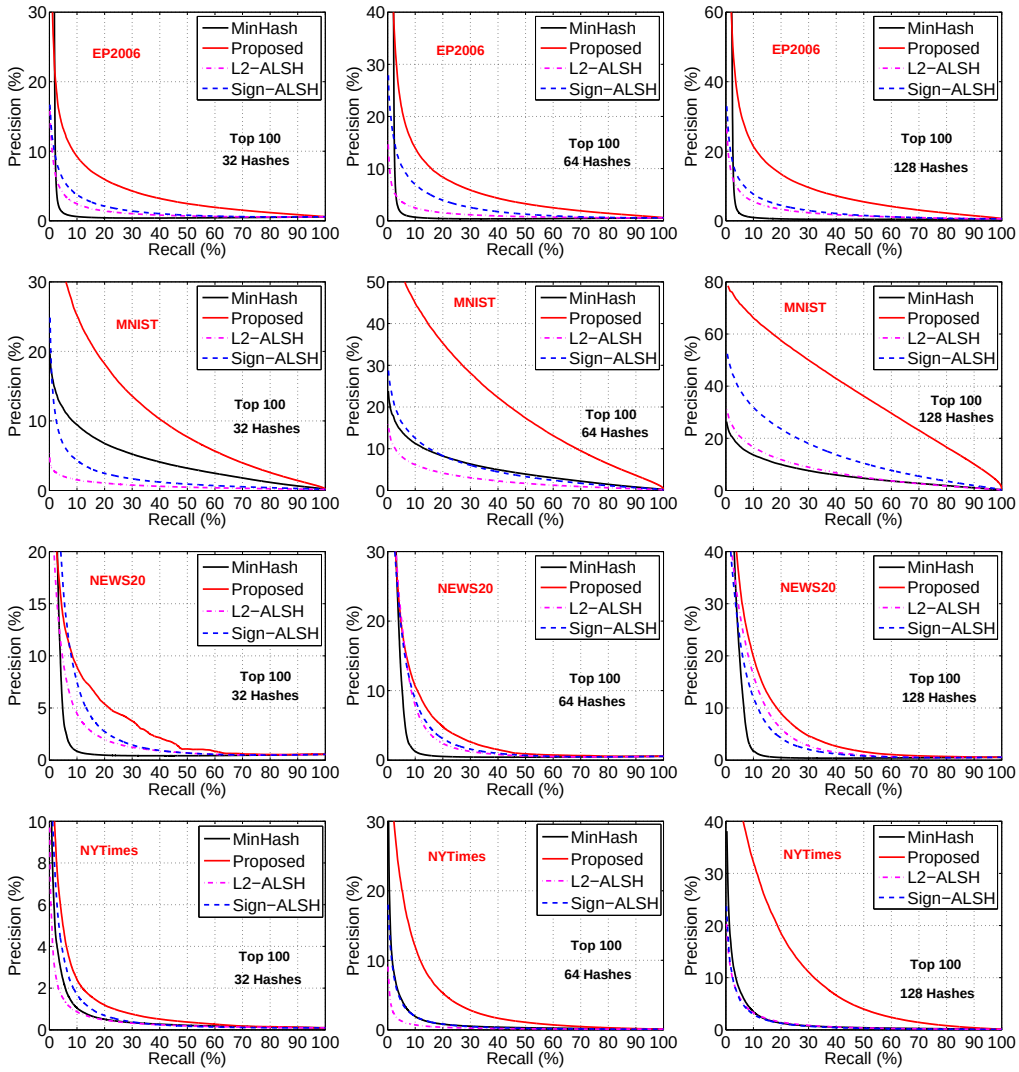

Figure 7.10: **Ranking Experiments for Asymmetric Minwise Hashing**. Precision Vs Recall curves for retrieving top-100 items, for different hashing schemes on 4 chosen datasets.

is formed by concatenating K different hash values as

$$B_j(x) = [h_{j1}(x); h_{j2}(x); ...; h_{jK}(x)], \tag{7.46}$$

where $h_{ij}, i \in \{1, 2, ..., K\}$ and $j \in \{1, 2, ..., L\}$, are KL different independent evaluations of the hash function under consideration. Different competing scheme uses its own underlying randomized hash function h.

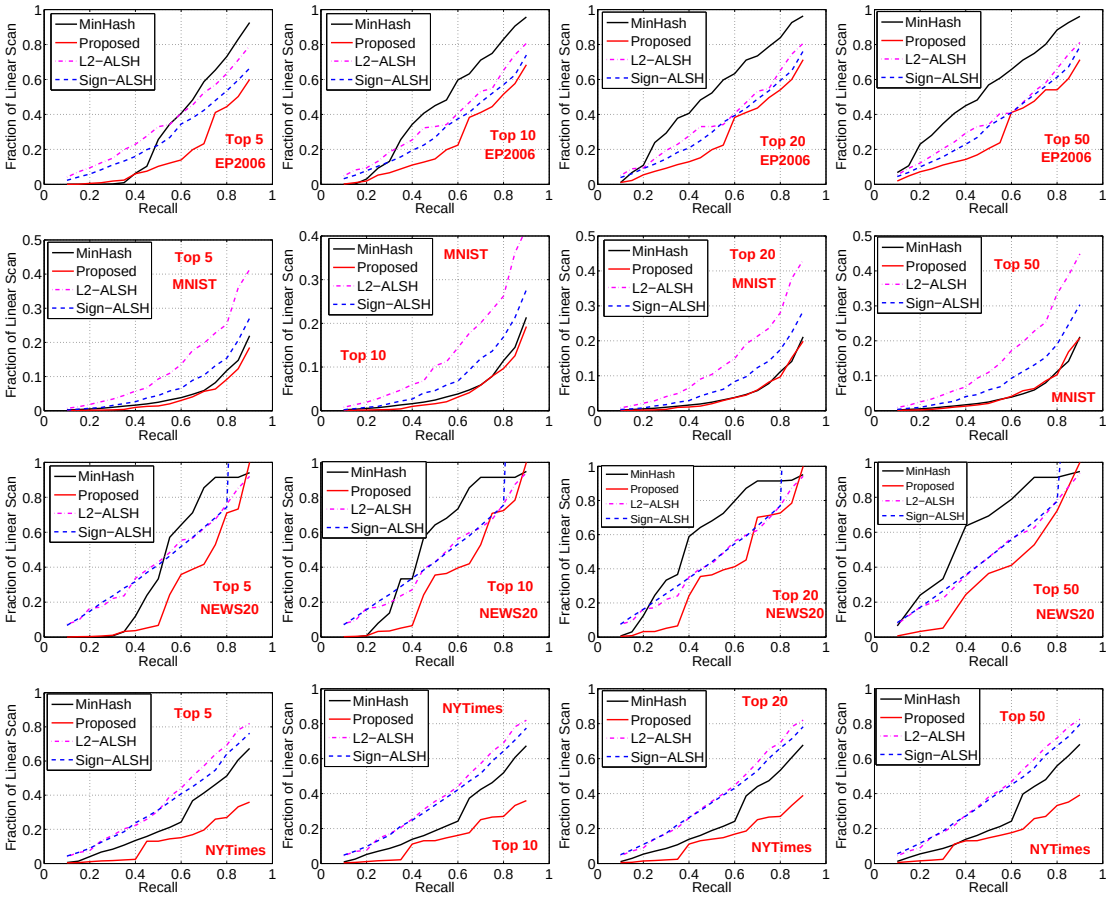

Figure 7.11: **LSH Bucketing Experiments for Asymmetric Minwise Hashing.** Average number of points retrieved per query (lower is better), relative to linear scan, evaluated by different hashing schemes at different recall levels, for top-5, top-10, top-20, top-50 nearest neighbors based on set containment (or equivalently inner products), on four datasets.

In general, the (K, L)-parameterized LSH works in two phases:

1. **Preprocessing Phase:** We construct L hash tables from the data by storing element x, in the training set, at location $B_j(P(x))$ in the hash-table j. Note that for vanilla MinHash which is a symmetric hashing scheme $P(x) = x$. For other asymmetric schemes, we use their corresponding P functions. Preprocessing is a one time operation, once the hash tables are created

they are fixed.

2. **Query Phase:** Given a query q, we report the union of all the points in the buckets $B_j(Q(q))\ \forall\, j \in \{1, 2, ..., L\}$, where the union is over L hash tables. Again here Q is the corresponding Q function of the asymmetric hashing scheme, for MinHash $Q(x) = x$.

Typically, the performance of a bucketing algorithm is sensitive to the choice of parameters K and L. Ideally, to find best K and L, we need to know the operating threshold S_0 and the approximation ratio c in advance. Unfortunately, the data and the queries are very diverse and therefore for retrieving top-ranked near neighbors there are no common fixed threshold S_0 and approximation ratio c that work for all the queries.

Our objective is to compare the four hashing schemes and minimize the effect of K and L, if any, on the evaluations. This is achieved by finding best K and L at every recall level. We run the bucketing experiment for all combinations of $K \in \{1, 2, 3, ...40\}$ and $L \in \{1, 2, 3, ..., 400\}$ for all the four hash functions independently. These choices include the recommended optimal combinations at various thresholds. We then compute, for every K and L, the mean recall of Top-T pairs and the mean number of points reported, per query, to achieve that recall. The best K and L at every recall level is chosen independently for different Ts. The plot of the mean fraction of points scanned with respect to the recall of top-T gold standard near neighbors, where $T \in \{5, 10, 20, 50\}$, is summarized in Figure 7.11.

The performance of a hashing based method varies with the variations in the similarity levels in the datasets. It can be seen that the proposed asymmetric MinHash always retrieves much less number of points, and hence requires sig-

nificantly less computations, compared to other hashing schemes at any recall level on all the four datasets. Asymmetric MinHash consistently outperforms other hash functions irrespective of the operating point. The plots clearly establish the superiority of the proposed scheme for indexing set containment (or inner products).

L2-ALSH and Sign-ALSH perform better than traditional MinHash on EP2006 and NEWS20 datasets while they are worse than plain MinHash on NYTIMES and MNIST datasets. If we look at the statistics of the dataset from Table 7.3, NYTIMES and MNIST are precisely the datasets with less variations in the number of nonzeros and hence MinHash performs better. In fact, for MNIST dataset with very small variations in the number of nonzeros, the performance of plain MinHash is very close to the performance of asymmetric MinHash. This is of course expected because there is negligible effect of penalization on the ordering. EP2006 and NEWS20 datasets have huge variations in their number of nonzeros and hence MinHash performs very poorly on these datasets. What is exciting is that despite these variations in the nonzeros, asymmetric MinHash always outperforms other ALSH for general inner products.

The difference in the performance of plain MinHash and asymmetric MinHash clearly establishes the utility of our proposal which is simple and does not require any major modification over traditional MinHash implementation. Given the fact that MinHash is widely popular, we hope that our proposal will be adopted.

7.8 Discussions

We have questioned the symmetry in the classical locality sensitive hashing definition, and, as a consequence, found a wide room for new opportunities. We found an extended ALSH framework which is a strict generalization of the classical LSH framework. We show that searching for maximum inner product admits sub-linear algorithm. Such an efficient solution is not possible in the classical LSH framework. MIPS is a common subroutine in a variety of machine learning algorithms, and all these algorithms will directly benefit from faster MIPS routines shown in this chapter.

In the ALSH framework, we show construction of three hashing algorithms for MIPS. We believe, what we have found is just the tip of an iceberg. The implications of ALSH framework, presented in this chapter, are far deeper. In particular, as a direct consequence, now it is possible to reduce one similarity search problem into another, by smartly engineering asymmetric transformations. There is now a hope of finding faster algorithms for many new search problems by reducing them to some know efficient similarity search problem. It is also an interesting direction to explore the properties of such similarities which can be reduced to one another. We believe that many exciting results will come in this directions.

CHAPTER 8

MOVING FORWARD

With the fundamental improvements in the hashing techniques and the new paradigms proposed in this thesis, we see further enormous opportunities to improve current large scale learning systems to a very significant extent. Following are some of the interesting future directions.

Web-scale deep networks via hashing: Scaling up deep networks is becoming an important research direction. One of the our motivations in finding efficient solutions to maximum inner product search (Chapter 7) was to scale up existing deep learning architectures. Recent results have shown that for dealing with massive datasets we need to train massive networks with the number of parameters potentially running into billions [27]. Regularizing such huge networks with techniques like adaptive dropouts [7] and maxouts [34] have proved to be the key reason behind their success. These regularization techniques only use few nodes, out of potentially billions, having large activations. Finding these nodes with large activations directly reduces to a maximum inner product search (MIPS) instance. We have shown first practical sub-linear solutions to MIPS in Chapter 7, an immediate follow up is to scale up the training and testing of giant deep networks using asymmetric hashing framework. Training and testing operations require identifying nodes with maximum activations, which can now be done in sub-linear time. Being able to train deep nets with an order of magnitude more parameters will be a big leap.

Joint or group recommendations: Current algorithms for group recommenda-

tion recommend items which are usually high in some weighted additive relevance with respect to everyone in the group. This technique has problems, because some relevance may dominate the others, and the item with maximum relevance may not be a balanced choice for all. It is more natural and useful to make use of k-way resemblance as shown in Chapter 5. Having found efficient sub-linear search algorithms for k-way resemblance, it is worth revisiting the problem of group recommendations with this new perspective.

A new privacy framework with hashing: Hashes themselves provide a randomized representation of the dataset which can be directly used for search, estimation and also learning as shown in Chapter 3. This totally eliminates the need for storing (or transmitting) the data anywhere, and it is possible to only work with hashes. The hashes are typically based on random projections or minwise hashing. These hashes are randomized and do not reveal direct information about the variables in the datasets. Thus, hashing provides a naturally reasonably encrypted interface. This leads to some natural questions "What kind of queries can we reasonably answer just on the basis of hashes ?" and "How much private are these random hashes?". Both of these questions are worthy of further explorations given the utility of hashing and the need for privacy. This is also a very promising domain because there are few works showing positive results that random hashes have privacy properties such as k-anonymity [87].

Hashing with limited randomness: In this thesis, we have developed many new hashing schemes for big data systems. An interesting and useful line of

work is to study the effect of randomness on these new hash functions. This will open up further possibilities of making these hash functions even faster using very cheap randomizations. There are lot of interesting works on tabulation-based hash functions [71] that are cheap to compute and at the same time behave like fully random hash functions. Design and analysis of these cheap randomizations-based on tabulations, for the newly developed hash functions, is an important and interesting direction to proceed.

This is the most exciting time to work on big data. The scale and the variability of the current datasets have made us rethink about the conventional assumptions prevailing in the existing literature. The modified settings open enormous opportunities for researchers to build new foundations by revisiting old assumptions. This opens a wide room for fundamental improvements in the existing well studied algorithms, some of which are shown in this thesis. The encouraging results that we have found with probabilistic hashing make us feel very optimistic and confident about the future of hashing algorithms for large scale search and learning.

APPENDIX A

PROOFS

A.0.1 Proof of Theorem 2

The proof of inequality is not difficult given $r \leq \frac{1}{\mathcal{S}^2}$ and using Equations 4.3.

Proof of tightness: The proof requires a bit of analysis. Let a continuous function $f(\mathcal{S})$ be a sharper upper bound i.e., $\mathcal{R} \leq f(\mathcal{S}) \leq \frac{\mathcal{S}}{2-\mathcal{S}}$. For any rational $\mathcal{S} = \frac{p}{q}$, with $p, q \in \mathbb{N}$ and $p \leq q$, choose $f_1 = f_2 = q$ and $a = p$. Note that f_1, f_2 and a are positive integers. This choice leads to $\frac{\mathcal{S}}{2-\mathcal{S}} = \mathcal{R} = \frac{p}{2q-p}$. Thus, the upper bound is achievable for all rational $\mathcal{S}$. Hence, it must be the case that $f(\mathcal{S}) = \frac{\mathcal{S}}{2-\mathcal{S}} = \mathcal{R}$ for all rational values of $\mathcal{S}$. For any real number $c \in [0, 1]$, there exist a Cauchy sequence of rational numbers $\{r_1, r_2, ...r_n, ...\}$ such that $r_n \in \mathbb{Q}$ and $\lim_{n\to\infty} r_n = c$. Since all r_n's are rational, $f(r_n) = \frac{r_n}{2-r_n}$. From the continuity of both f and $\frac{\mathcal{S}}{2-\mathcal{S}}$, we have $f(\lim_{n\to\infty} r_n) = \lim_{n\to\infty} \frac{r_n}{2-r_n}$ which implies $f(c) = \frac{c}{2-c}$ implying $\forall c \in [0, 1]$.

For proving tightness of $\mathcal{S}^2$, let $\mathcal{S} = \sqrt{\frac{p}{q}}$, choosing $f_2 = a = p$ and $f_1 = q$ gives an infinite set of points having $\mathcal{R} = \mathcal{S}^2$. We can now use similar arguments used to prove the tightness of upper bound. All we need is the existence of a Cauchy sequence of square root of rational numbers converging to any real c, which is just the square root of rational Cauchy sequence converging to c^2 (we used continuity of square root function). □

A.0.2 Proof of Theorem 9

The proof is a simple case based analysis. Note, C is always greater than the value of any non empty bin.

CASE I: $(I^j_{emp} = 0)$

Without loss of generality, let $\min_j \pi(S_1) \neq E$. If we have $\min_j \pi(S_2) \neq E$, then both of these values are untouched and we get

$$\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2)) \iff \min_j \pi(S_1) = \min_j \pi(S_2).$$

In case if $\min_j \pi(S_2) = E$, then by the choice of C ,

$$\mathcal{H}_j(\pi(S_2)) > C > \min_j \pi(S_1) = \mathcal{H}_j(\pi(S_1)).$$

Therefore, either way we have

$$\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2)) \iff \min_j \pi(S_1) = \min_j \pi(S_2).$$

$$Pr(\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2))|I^j_{emp} = 0) = Pr(\min_j(\pi(S_1)) = \min_j(\pi(S_2))|I^j_{emp} = 0) = R$$

.

CASE II: $(I^j_{emp} = 1)$

Let

$$t_1 = \min x \quad \text{s.t} \quad \min_{(j+x)\text{mod k}} (\pi(S_1)) \neq E.$$

$$t_2 = \min x \quad \text{s.t} \quad \min_{(j+x)\text{mod k}} (\pi(S_2)) \neq E.$$

Let $m = \min(t_1, t_2)$ and $t = (j + m)\text{mod k}$. The definition of t_1 and t_2 implies $I^t_{emp} = 0$.

Subcase I: $(t_1 = t_2 = m)$

We have

$$\mathcal{H}_j(\pi(S_1)) = \min_t(\pi(S_1)) + mC \quad \text{and} \quad \mathcal{H}_j(\pi(S_2)) = \min_t(\pi(S_2)) + mC.$$

$$\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2)) \iff \min_t \pi(S_1) = \min_t \pi(S_2).$$

Subcase II: ($t_1 \neq t_2$) Without loss of generality $t_1 > t_2 = m$. Clearly, by definition of t, t_1 and t_2, we have

$$\min_t(\pi(S_1)) = E; \quad \text{and} \quad \min_t(\pi(S_2)) \neq E.$$

Also,

$$\mathcal{H}_j(\pi(S_1)) = \min_{(j+t_1)\bmod k}(\pi(S_1)) + t_1 C > \min_{(j+t_2)\bmod k}(\pi(S_2)) + t_2 C = \mathcal{H}_j(\pi(S_2)).$$

Thus, from the two subcases we can write,

$$Pr(\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2))|I^j_{emp} = 1) = Pr(\min_t(\pi(S_1)) = \min_t(\pi(S_2))|I^t_{emp} = 0) = R.$$

A.0.3 Proof of Theorem 10

For the analysis, it is sufficient to consider the configurations, of empty and non-empty bins, arising after throwing $|S_1 \cup S_2|$ balls uniformly into k bins with exactly m non-empty bins and $k - m$ empty bins. Under uniform throwing of balls any ordering of m non-empty and $k - m$ empty bins are equally likely. The proofs involve elementary combinatorial arguments of counting configurations.

Proof of Lemma 2

Given exactly m simultaneous non empty bins, any two of them can be chosen in $m(m-1)$ ways (with ordering of i and j). Each term $M^N_i M^N_j$, for both simultaneously non-empty i and j, is 1 with probability $R\tilde{R}$ (Note, $\mathbb{E}\left(M^N_i M^N_j \middle| i \neq j, I^i_{emp} = 0, I^j_{emp} = 0\right) = R\tilde{R}$).

Proof of Lemma 3

The permutation is random and any sequence of simultaneous m non-empty and remaining $k-m$ empty bins are equally likely. This is because, while randomly throwing $|S_1 \cup S_2|$ balls into k bins with exactly m non-empty bins every sequence of simultaneous empty and non-empty bins has equal probability. Given m, there are total $2m(k-m)$ different pairs of empty and non-empty bins (including the ordering). Now, for every simultaneous empty bin j, i.e., $I^j_{emp} = 1$, M^E_j replicates M^N_t corresponding to nearest non-empty Bin t which is towards the circular right. There are two cases,

Case 1: $t = i$, which has probability $\frac{1}{m}$ and

$$\mathbb{E}(M^N_i M^E_j | I^i_{emp} = 0, I^j_{emp} = 1) = \mathbb{E}(M^N_i | I^i_{emp} = 0) = R$$

Case 2: $t \neq i$, which has probability $\frac{m-1}{m}$ and

$$\begin{aligned}&\mathbb{E}(M^N_i M^E_j | I^i_{emp} = 0, I^j_{emp} = 1)\\ =&\mathbb{E}(M^N_i M^N_t | t \neq i, I^i_{emp} = 0,\ I^t_{emp} = 0) = R\tilde{R}\end{aligned}$$

Thus, the value of $\mathbb{E}\left[\sum_{i\neq j} M^N_i M^E_j \middle| m\right]$ comes out to be

$$2m(k-m)\left[\frac{R}{m} + \frac{(m-1)R\tilde{R}}{m}\right]$$

which is the desired expression.

Proof of Lemma 4

Given m, we have $(k-m)(k-m-1)$ different pairs of simultaneous non-empty bins. There are two cases, if the closest simultaneous non-empty bins towards

their circular right are identical, then for such i and j, $M_i^E M_j^E = 1$ with probability R, else $M_i^E M_j^E = 1$ with probability $R\tilde{R}$. Let p be the probability that two simultaneously empty bins i and j has the same closest bin on the right. Then $\mathbb{E}\left[\sum_{i\neq j} M_i^E M_j^E \middle| m\right]$ is given by

$$(k-m)(k-m-1)\left[pR + (1-p)R\tilde{R}\right] \tag{A.1}$$

because with probability $(1-p)$, it uses estimators from different simultaneous non-empty bin and in that case the $M_i^E M_j^E = 1$ with probability $R\tilde{R}$.

Consider Figure 6.10, here we have 3 simultaneous non-empty bins, i.e., $m = 3$ (shown by colored boxes). Given any two simultaneous empty bins Bin i and Bin j (out of total $k-m$) they will occupy any of the $m+1 = 4$ blank positions. The arrow shows the chosen non-empty bins, for filling the empty bins, corresponding each of the positions of the non-empty bins. There are $(m+1)^2 + (m+1) = (m+1)(m+2)$ different ways of fitting two simultaneous non-empty bins i and j between m non-empty bins. Note, if both i and j goes to the same blank positions they can be permuted. This adds extra term $(m+1)$.

If both i and j chooses same blank space or the first and the last blank space, then both the simultaneous empty bins, Bin i and Bin j, corresponds to the same non-empty bin. The number of ways in which this happens is $2(m+1) + 2 = 2(m+2)$. So, we have

$$p = \frac{2(m+2)}{(m+1)(m+2)} = \frac{2}{m+1}.$$

Substituting p in Eq.(A.1) leads to the desired expression.

A.0.4 Proof of Theorem 12

Proof of Lemma 5

Similar to the proof of Lemma 4, we need to compute p which is the probability that two simultaneously empty bins, Bin i and Bin j, use information from the same bin. As argued before, the total number of positions for any two simultaneously empty bins i and j, given m simultaneously non-empty bins is $(m+1)(m+2)$. Consider Figure 6.11, under the improved scheme, if both Bin i and Bin j choose the same blank position then they choose the same simultaneously non-empty bin with probability $\frac{1}{2}$. If Bin i and Bin j choose consecutive positions (e.g., position 2 and position 3) then they choose the same simultaneously non-empty bin (Bin b) with probability $\frac{1}{4}$. There are several boundary cases to consider too. Accumulating the terms leads to

$$p = \frac{\frac{2(m+2)}{2} + \frac{2m+4}{4}}{(m+1)(m+2)} = \frac{3}{2(m+1)}.$$

Substituting p in Eq.(A.1) yields the desired result.

Note that $m = 1$ (an event with almost zero probability) leads to the value of $p = 1$. We ignore this case because it unnecessarily complicates the final expressions. $m = 1$ can be easily handled and does not affect the final conclusion.

Proof of Theorem 13

Immediate from Lemma 4 and Lemma 5 and the fact that $R \leq R\tilde{R}$. For computing gain all we need is to subtract the two expressions from Lemma 3 and Lemma 4 respectively. All other components involved in the variance remain the same for both $\hat{R}$ and $\hat{R}^{+}$.

A.0.5 Proof of Theorem 16

From Equation 2.11, we have

$$Pr[h^{L2}_{a,b}(Q(q)) = h^{L2}_{a,b}(P(x))] = F_r(||Q(q) - P(x)||_2)$$
$$=F_r(\sqrt{1 + m/4 - 2q^T x + ||x||_2^{2^{m+1}}}) \geq F_r(\sqrt{1 + m/4 - 2S_0 + U^{2^{m+1}}})$$

The last step follows from the monotonically decreasing nature of F combined with inequalities $q^T x \geq S_0$ and $||x||_2 \leq U$. We have also used the monotonicity of the square root function. The second inequality similarly follows using $q^T x \leq cS_0$ and $||x||_2 \geq 0$.. This completes the proof

A.0.6 Proof of Theorem 19

Proof: When $q^T x \geq S_0$, we have, according to Equation 2.7

$$Pr[h^{Sign}(Q(q)) = h^{Sign}(P(x))] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{q^T x}{\sqrt{m/4 + ||x||_2^{2^{m+1}}}}\right)$$
$$\geq 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{q^T x}{\sqrt{m/4 + U^{2^{m+1}}}}\right)$$

When $q^T x \leq cS_0$, by noting that $q^T x \leq ||x||_2$, we have

$$Pr[h^{Sign}(Q(q)) = h^{Sign}(P(x))] = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{q^T x}{\sqrt{m/4 + ||x||_2^{2^{m+1}}}}\right)$$
$$\leq 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{q^T x}{\sqrt{m/4 + (q^T x)^{2^{m+1}}}}\right)$$

For this one-dimensional function $f(z) = \frac{z}{\sqrt{a+z^b}}$, where $z = q^T x$, $a = m/4$ and $b = 2^{m+1} \geq 2$, we know

$$f'(z) = \frac{a - z^b\,(b/2 - 1)}{(a + z^b)^{3/2}}$$

One can also check that $f''(z) \leq 0$ for $0 < z < 1$, i.e., $f(z)$ is a concave function. The maximum of $f(z)$ is attained at $z^* = \left(\frac{2a}{b-2}\right)^{1/b} = \left(\frac{m/2}{2^{m+1}-2}\right)^{2^{-m-1}}$ If $z^* \geq cS_0$, then we need to use $f(cS_0)$ as the bound.

APPENDIX B

CRS (CONDITIONAL RANDOM SAMPLING) SKETCHES AND DENSIFIED ONE PERMUTATION HASHING

In their seminal work [12, 51, 52, 53], another sketching scheme was proposed for estimating resemblance similarity between two sets or binary vectors. Instead of using only the minimum under a permutation π, as in case of minwise hashing, the idea was to use the k minimums (for some $k \geq 1$) as a sketch.

Sticking to the old notation of [12], we define a new operator $MIN_k(S)$ operating over a set S as

$$MIN_k(S) = \begin{cases} \text{Set of smallest } k \text{ elements of S,} & \text{if } |S| > k \\ S, & \text{otherwise} \end{cases}$$

Also, define $M_i(S)$ to be the i^{th} smallest elements of set S.

Formally, the CRS sketches under a given permutation π is $MIN_k(\pi(S))$. This generates k sketches simultaneously for a given data S if $|S| > k$, else it generates only $|S|$ sketches. It was shown that one can obtain an unbiased estimator of resemblance between sets S_1 and S_2 as follows:

$$\hat{R}^{Bot-k} = \frac{|MIN_k(\pi(S_1 \cup S_2)) \cap MIN_k(\pi(S_1)) \cap MIN_k(\pi(S_2))|}{|MIN_k(\pi(S_1 \cup S_2))|}$$

The CRS sketches with $k = 8$ for the sets S_1 and S_2 is shown in Figure B.1. CRS sketches are not aligned and do not satisfy the LSH property, except when $i = 1$, in which case we recover minwise hashing. The expression of the form

$$\frac{1}{k} \sum_{i=1}^{k} \mathbf{1}(M_i(\pi(S_1) = M_i(\pi(S_2))),$$

$\pi(\Omega)$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
$\pi(S_1)$	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0
$\pi(S_2)$	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0

Botk(S_1)	6	8	15	16	17	19	22	23
Botk(S_2)	6	7	8	13	15	17	18	-

Figure B.1: Example CRS sketches of example vectors S_1 and S_2 used in Figure 1 of the main Chapter

is not an unbiased estimator of resemblance. In particular, the event $M_i(\pi(S_1))$ = $M_i(\pi(S_2))$, unlike minwise hashing is not purely a function of similarity R between S_1 and S_2 . It does depend on other information about S_1 and S_2. Thus, other factors adversely affects the collision probability, and hence the bucket assignment Equation 7.46 does not guarantee similar points in same buckets. In formal terms, CRS sketches do not satisfy LSH property.

It is reasonable to expect that $M_i(\pi(S_1)) = M_i(\pi(S_2)$ is a positive indication of high similarity between S_1 and S_2. CRS sketches, for small values of k, behaves like minwise hashing and in limit when $k = 1$ they both are the same. Owing to this, it may seem reasonable to reduce the processing time of minwise hashing by using CRS sketches, for some small k, by generating k hash evaluations in a single permutation. This, reduces the processing time over minwise hashing by a factor of k. Unfortunately, this scheme will only work for small k (such as $k < 10$). This is not much improvement over minwise hashing. As k increases this scheme leads to large deviations. It should be noted that such usage of CRS

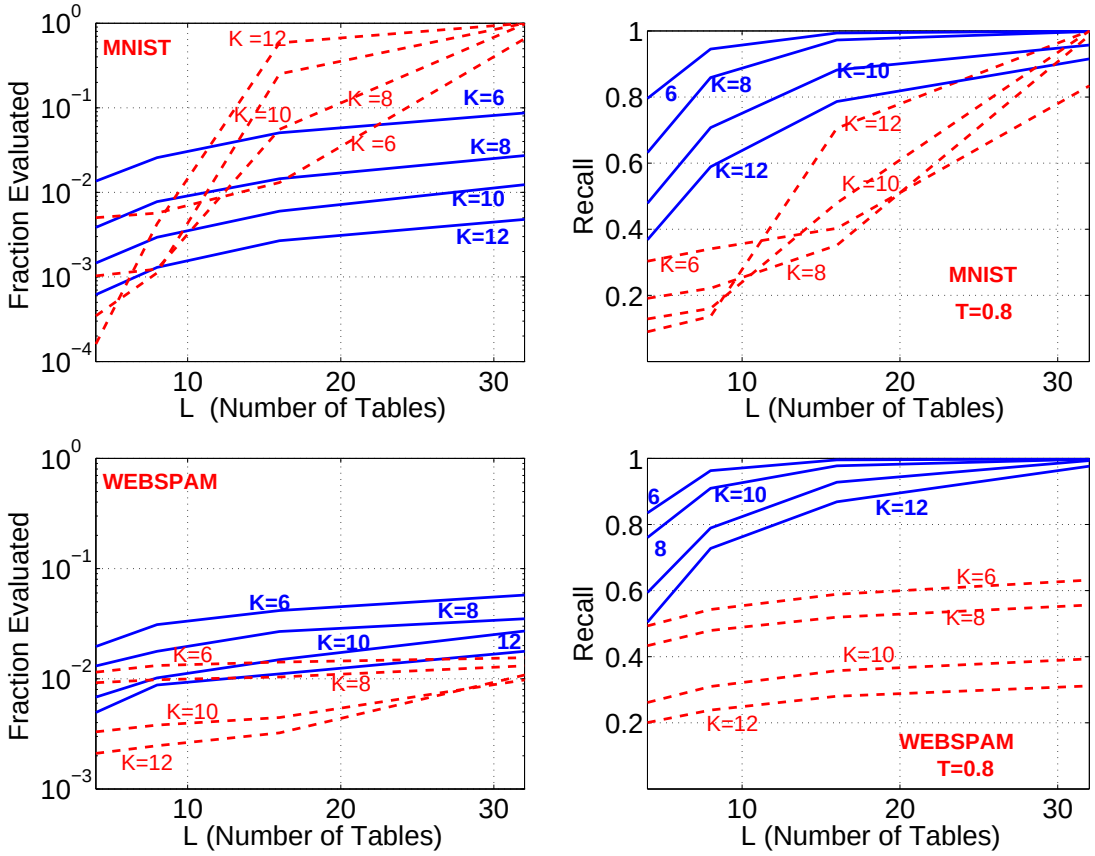

Figure B.2: Comparisons of CRS sketches with the proposed hashing scheme in retrieving all the points having similarity with the query greater than $T = 0.8$. CRS sketches are in dashed red

for reducing the cost of minwise hashing is just a heuristic and not a principled way to do near neighbor search.

To compete with our scheme in terms of computation, CRS sketches need to set the value of k to be $k = K \times L$. This high value of k comes with two major problems: 1) $K \times L$ is sufficiently large and hence the collision of CRS sketches will not be an indicator of high similarity. 2) The current web datasets are very sparse. For datapoints with less than $K \times L$ non-zeros, we will not have the necessary number of hash evaluations. Since there is a lot of variation in the

number of non-zeros in the given datasets, this problem is critical.

To provide experimental evidence, we perform experiments on the MNIST and WEBSPAM datasets used in Chapter 6. We repeat the experimental evaluation procedure described in Chapter 6 with high similarity threshold $T = 0.8$. We use the same combinations of K and L and implement the bucketing scheme with both CRS sketches and $\mathcal{H}$. Out aim is to evaluate the performance of LSH based retrieval using CRS sketches with only one permutation and compare it with the performance of the proposed scheme. We set $k = K \times L$. We also need some mechanism to handle exceptionally sparse data points. We handle them by setting $M_i(\pi(S)) = 0, \text{if } |S| < i$. The evaluation results are summarized in Figure B.2.

It is evident from the plots that using a single permutation with CRS sketches performs poorly even in retrieving elements with high similarity ($T > 0.8$). For the WEBSPAM data, our proposed scheme can achieve a perfect recall with only evaluating about $1/100$ fraction of the data. On the other hand, CRS sketches can achieve only a recall of around 50% after retrieving $1/100$ fraction of data points. The trends with variation in K and L clearly suggest the inferiority of CRS sketches in retrieving highly similar points, which is expected because the collision event with CRS sketches is not perfectly co-related with the resemblance.

In case of MNIST dataset we see a very unpredictable trend with CRS sketches. This is because the average number of non-zeros in this dataset is around 150 and as the number of hash evaluation exceed this number, CRS sketches becomes undefined. We can see that $M_i(\pi(S)) = 0, \text{if } |S| < i$ causes a jump in the number of points retrieved when $K \times L$ is large. Even with small

$K \times L$, where we have well defined CRS sketches, the recall is very bad compared to the prosed hash function. Despite this sparsity problem of MNIST dataset we have seen in Chapter 6 that the proposed hashing scheme still behaves exactly like minwise hashing, which is very exciting.

BIBLIOGRAPHY

[1] http://www.howtogeek.com/howto/15799/how-to-use-autofill-on-a-google-docs-spreadsheet-quick-tips/.

[2] S. Agarwal, Jongwoo Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie. Beyond pairwise clustering. In *CVPR*, 2005.

[3] Parag Agrawal, Arvind Arasu, and Raghav Kaushik. On indexing error-tolerant set containment. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 927–938. ACM, 2010.

[4] Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *STOC*, pages 557–563, Seattle, WA, 2006.

[5] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. Group recommendation: semantics and efficiency. *Proc. VLDB Endow.*, 2(1):754–765, 2009.

[6] Alexandr Andoni and Piotr Indyk. E2lsh: Exact euclidean locality sensitive hashing. Technical report, 2004.

[7] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

[8] Harald Baayen. *Word Frequency Distributions*, volume 18 of *Text, Speech and Language Technology*. Kulver Academic Publishers, 2001.

[9] Yoram Bachrach, Yehuda Finkelstein, Ran Gilad-Bachrach, Liran Katzir, Noam Koenigstein, Nir Nice, and Ulrich Paquet. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, 2014.

[10] Leon Bottou. http://leon.bottou.org/projects/sgd.

[11] Christina Brandt, Thorsten Joachims, Yisong Yue, and Jacob Bank. Dynamic ranked retrieval. In *WSDM*, pages 247–256, 2011.

[12] Andrei Z. Broder. On the resemblance and containment of documents. In *the Compression and Complexity of Sequences*, pages 21–29, Positano, Italy, 1997.

[13] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. In *STOC*, pages 327–336, Dallas, TX, 1998.

[14] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. In *STOC*, pages 106–112, 1977.

[15] Tushar Chandra, Eugene Ie, Kenneth Goldman, Tomas Lloret Llinares, Jim McFadden, Fernando Pereira, Joshua Redstone, Tal Shaked, and Yoram Singer. Sibyl: a system for large scale machine learning.

[16] Olivier Chapelle, Patrick Haffner, and Vladimir N. Vapnik. Support vector machines for histogram-based image classification. 10(5):1055–1064, 1999.

[17] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[18] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, Montreal, Quebec, Canada, 2002.

[19] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operatior for similarity joins in data cleaning. In *ICDE*, 2006.

[20] Flavio Chierichetti and Ravi Kumar. Lsh-preserving functions and their applications. In *SODA*, pages 1078–1094, 2012.

[21] Edith Cohen and Haim Kaplan. Summarizing data using bottom-k sketches. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 225–234. ACM, 2007.

[22] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[23] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

[24] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. Fast locality-sensitive hashing. In *KDD*, pages 1073–1081, 2011.

[25] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokn. Locality-sensitive hashing scheme based on *p*-stable distributions. In *SCG*, pages 253 – 262, Brooklyn, NY, 2004.

[26] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational geometry*. Springer, 2000.

[27] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[28] Thomas Dean, Mark A Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1814–1821. IEEE, 2013.

[29] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[30] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

[31] Jerome H. Friedman, F. Baskett, and L. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 24:1000–1006, 1975.

[32] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, 1999.

[33] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[34] Ian Goodfellow, David Warde-farley, Mehdi Mirza, Aaron Courville, and

Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1319–1327, 2013.

[35] Bernhard Haeupler, Mark Manasse, and Kunal Talwar. Consistent weighted sampling made fast, small, and easy. Technical report, arXiv:1410.4266, 2014.

[36] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(14):321–350, 2012.

[37] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web. In *WebDB*, pages 129–134, 2000.

[38] Matthias Hein and Olivier Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *AISTATS*, pages 136–143, Barbados, 2005.

[39] Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291. ACM, 2006.

[40] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

[41] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of ACM*, 53(3):307–323, 2006.

[42] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, Dallas, TX, 1998.

[43] Sergey Ioffe. Improved consistent sampling, weighted minhash and L1 sketching. In *ICDM*, pages 246–255, Sydney, AU, 2010.

[44] Yugang Jiang, Chongwah Ngo, and Jun Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *CIVR*, pages 494–501, Amsterdam, Netherlands, 2007.

[45] Thorsten Joachims. Training linear svms in linear time. In *KDD*, pages 217–226, Pittsburgh, PA, 2006.

[46] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. Efficient retrieval of recommendations in a matrix factorization framework. In *CIKM*, pages 535–544, 2012.

[47] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems.

[48] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. Technical report, arXiv:1207.6083, 2013.

[49] Ping Li. Core kernels. In *UAI*, Quebec, CA, 2014.

[50] Ping Li. 0-bit consistent weighted sampling. In *KDD*, 2015.

[51] Ping Li and Kenneth W. Church. Using sketches to estimate associations. In *HLT/EMNLP*, pages 708–715, Vancouver, BC, Canada, 2005.

[52] Ping Li and Kenneth W. Church. A sketch algorithm for estimating two-way and multi-way associations. *Computational Linguistics (Preliminary results appeared in HLT/EMNLP 2005)*, 33(3):305–354, 2007.

[53] Ping Li, Kenneth W. Church, and Trevor J. Hastie. Conditional random sampling: A sketch-based sampling technique for sparse data. In *NIPS*, pages 873–880, Vancouver, BC, Canada, 2006.

[54] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *KDD*, pages 287–296, Philadelphia, PA, 2006.

[55] Ping Li and Arnd Christian König. b-bit minwise hashing. In *Proceedings of the 19th International Conference on World Wide Web*, pages 671–680, Raleigh, NC, 2010.

[56] Ping Li and Arnd Christian König. Theory and applications b-bit minwise hashing. *Commun. ACM*, 2011.

[57] Ping Li, Arnd Christian König, and Wenhao Gui. b-bit minwise hashing for estimating three-way similarities. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2010.

[58] Ping Li, Michael Mitzenmacher, and Anshumali Shrivastava. Coding for random projections. In *ICML*, 2014.

[59] Ping Li, Michael Mitzenmacher, and Anshumali Shrivastava. Coding for random projections and approximate near neighbor search. Technical report, arXiv:1403.8144, 2014.

[60] Ping Li, Art B Owen, and Cun-Hui Zhang. One permutation hashing. In *NIPS*, Lake Tahoe, NV, 2012.

[61] Ping Li, Anshumali Shrivastava, and Arnd Christian König. b-bit minwise hashing in practice: Large-scale batch and online learning and using GPUs for fast preprocessing with simple hash functions. Technical report, 2011.

[62] Ping Li, Anshumali Shrivastava, and Arnd Christian König. GPU-based minwise hashing. In *WWW (Poster)*, 2012.

[63] Ping Li, Anshumali Shrivastava, and Arnd Christian König. b-bit minwise hashing in practice. In *Internetware*, Changsha, China, 2013.

[64] Ping Li, Anshumali Shrivastava, and Christian A. Konig. Gpu-based minwise hashing: Gpu-based minwise hashing. In *WWW*, pages 565–566, 2012.

[65] Ping Li, Anshumali Shrivastava, Joshua Moore, and Arnd Christian König. Hashing algorithms for large-scale learning. In *NIPS*, Granada, Spain, 2011.

[66] Avner Magen and Anastasios Zouzias. Near optimal dimensionality reductions that preserve volumes. In *APPROX / RANDOM*, pages 523–534, 2008.

[67] Mark Manasse, Frank McSherry, and Kunal Talwar. Consistent weighted sampling. Technical Report MSR-TR-2010-73, Microsoft Research, 2010.

[68] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *SODA*, 2008.

[69] Behnam Neyshabur and Nathan Srebro. On symmetric and asymmetric lshs for inner product search. Technical report, arXiv:1410.5518, 2014.

[70] Noam Nisan. Pseudorandom generators for space-bounded computations. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC, pages 204–212, 1990.

[71] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. In *STOC*, 2011.

[72] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

[73] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2007.

[74] Anand Rajaraman and Jeffrey Ullman. *Mining of Massive Datasets*. http://i.stanford.edu/ ullman/mmds.html.

[75] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[76] Parikshit Ram and Alexander G Gray. Maximum inner-product search using cone trees. In *KDD*, pages 931–939, 2012.

[77] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, pages 807–814, Corvalis, Oregon, 2007.

[78] Anshumali Shrivastava and Ping Li. Fast near neighbor search in high-dimensional binary data. In *ECML*, Bristol, UK, 2012.

[79] Anshumali Shrivastava and Ping Li. Beyond pairwise: Provably fast algorithms for approximate k-way similarity search. In *NIPS*, Lake Tahoe, NV, 2013.

[80] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*, Montreal, CA, 2014.

[81] Anshumali Shrivastava and Ping Li. Densifying one permutation hashing via rotation for fast near neighbor search. In *ICML*, Beijing, China, 2014.

[82] Anshumali Shrivastava and Ping Li. Improved densification of one permutation hashing. In *UAI*, Quebec, CA, 2014.

[83] Anshumali Shrivastava and Ping Li. An improved scheme for asymmetric lsh. *arXiv preprint arXiv:1410.5410*, 2014.

[84] Anshumali Shrivastava and Ping Li. {In Defense of Minhash over

Simhash}. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 886–894, 2014.

[85] Anshumali Shrivastava and Ping Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, Florence, Italy, 2015.

[86] Anshumali Shrivastava and Ping Li. Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips). In *UAI*, Amsterdam, Netherlands, 2015.

[87] Xiaoxun Sun, Min Li, Hua Wang, and Ashley Plank. An efficient hash-based algorithm for minimal k-anonymity. In *Proceedings of the thirty-first Australasian conference on Computer science-Volume 74*, pages 101–107. Australian Computer Society, Inc., 2008.

[88] Simon Tong. Lessons learned developing a practical large scale machine learning system. http://googleresearch.blogspot.com/2010/04/lessons-learned-developing-practical.html, 2008.

[89] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[90] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.

[91] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, 2008.

[92] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. In *KDD*, pages 833–842, 2010.

[93] D. Zhou, J. Huang, and B. Schölkopf. Beyond pairwise classification and clustering using hypergraphs. In *NIPS*, 2006.