Design and Analysis of LSH Based Techniques for Inner Product

By

Venkatesh Guntakindapalli

Under the Supervision of Dr Debajyoti Bera

Indraprastha Institute of Information Technology,Delhi

May, 2017

Design and Analysis of LSH Based Techniques for Inner Product

By

Venkatesh Guntakindapalli

submitted
in partial fulfillment of the requirements for the degree of
Master of Technology

Indraprastha Institute of Information Technology,Delhi
May, 2017

CERTIFICATE

This is to certify that the thesis titled "Design and Analysis of LSH based techniques for Inner Product" being submitted by Venkatesh Guntakindapalli to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by him under my supervision. In my opinion the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

May, 2017

Debajyoti Bera
Department of Computer Science
Indraprastha Institute of Information Technology Delhi
New Delhi 110 020

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my advisor Dr.Debajyoti Bera for providing an excellent guidance and being supportive throughout the span of this thesis. Without his patience, critiques and thoughtful insights this work would never have been completed. A very special thanks to IIITD for providing an excellent infrastructure, environment and a flexible curriculum to carry out my work at a suitable pace. I am deeply thankful to all the faculty and admin staff here for there extremely supportive attitude. I am thankful to all my friends for being so supportive and helping me out with the problems whenever I got stuck. Fun time with you people always made me feel fresh and enthusiastic. I dedicate this work to my parents, who have always been there for me and provided me support both emotionally and financially.

Abstract

Locality sensitive hashing(LSH) is a hashing scheme that hashes objects into several buckets such that objects that are similar hashes into the same bucket and objects that are dissimilar hashes into different bucket with high probability. LSH provides sub-linear query time for near neighbor search problem. The nearness of two objects is measured using different similarity measures, e.g. cosine similarity, jaccard similarity. Inner product similarity is an ubiquitous measure in many data mining and machine learning problems. Even though inner product similarity appears in many important problems it is hard to get locality sensitive hashing for this similarity. There are few transformations that alleviate this problem and provide locality sensitive hashing for inner product similarity. In this thesis study we analyze the performance of locality sensitive hashing for inner product similarity. As of our knowledge there is no proper study that estimates the number of hash function evaluations required to achieve particular performance measures(true positive rate and false positive rate). This analysis provides a guideline to anyone who wants to use locality sensitive hashing for inner product similarity search related problems. We also investigated and proposed a new technique that reduces the required number of hash function evaluations. This technique bands the hash values in a hierarchical tree like structure. The idea behind this technique is that the locality sensitive hashing technique performs better if the collision probability difference is more.

TABLE OF CONTENTS

List of Tables

LIST OF FIGURES

FIGURE **Page**

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 thesis study we analyze the performance of one of such hashing algorithms called locality sensitive hashing.

1.1 Thesis Structure

In this chapter we explain the importance of hashing algorithms and briefly review our work

In chapter 2 we explain the definition of locality sensitive hashing and review few well known hashing schemes for different similarity measures. We also explain the importance of inner product similarity and show why it is hard to get a naive locality sensitive hashing schemes for inner product similarity. We also review few asymmetric transformations and explain how these asymmetric transformations provide locality sensitive hashing schemes for inner product similarity.
In chapter 3 we analyze the performance of different locality sensitive hashing schemes for inner product similarity. This analysis gives an estimate on the performance of locality sensitive hashing schemes for inner product similarity. In this chapter we provide a hierarchical banding technique that improves the performance of traditional locality sensitive techniques for inner product similarity.
In chapter 4 we use locality sensitive hashing to solve frequent itemset mining problem. We also show that locality sensitive hashing schemes give better results with hierarchical banding technique than naive or single level banding technique.

1.2 Contribution

The main goal of this thesis study is to give a guideline on the performance of locality sensitive schemes for inner product similarity search. Following are major contributions of this thesis study

1. We provide detailed analysis on the performance of different locality sensitive hashing schemes for inner product similarity.

2. We introduced a new technique called hierarchical banding that improves the performance of naive locality sensitive hashing performance.

3. We explored the usage of locality sensitive hashing schemes to solve frequent item set mining problem. We show that locality sensitive hashing schemes performs better with hierarchical banding.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter we review different similarity measures and discuss few well known Locality Sensitive Hashing(LSH) schemes for those similarity measures. Even though LSH provide elegant solution for searching near neighbor problems[1], we discuss a known result that shows that a naive LSH does not exist for inner product similarity. We review few transformation techniques to workaround this negative result that leads to LSH schemes for inner product similarity.

2.1 Locality Sensitive Hashing

The locality sensitive hashing technique was introduced by Indyk et. al. in their seminal work to solve "Near-Neighbor Search" problem [1]. LSH is a family of functions with the property that the function values of similar input objects in the domain of these functions have a higher probability of colliding than non-similar ones. Formally, consider a family of hash functions $\mathscr{H}$ that maps $\mathbb{R}^d$ to some set S. Let Sim denotes similarity measure that maps pairs of objects from $\mathbb{R}^d$ to {0,1}; a case $Sim(x,y) = 1$ indicates that x, y are identical.

Definition 1: Locality Sensitive Hashing(LSH) family A family $\mathscr{H}$ is called (s_1, s_2, p_1, p_2) - sensitive for any two points $x, y \in \mathbb{R}^d$ and a hash function h chosen uniformly at random from $\mathscr{H}$ satisfies the following properties:

1. If $Sim(x,y) \geq s_1$ then $Pr_h[h(x) = h(y)] \geq p_1$

2. If $Sim(x,y) \leq s_2$ then $Pr_h[h(x) = h(y)] \leq p_2$

Here $0 \leq s_1, s_2, p_1, p_2 \leq 1$. In the following subsection we review different similarity measures and discuss corresponding LSH schemes.

2.1.1 Minhash

Minhash is an LSH for Jaccard similarity (JS) or Jaccard index. It is a statistic that measures the similarity of two finite sets. The Jaccard similarity is defined as the size of intersection divided by the size of union of the input sets, i.e.,

$$JS(x,y) = \frac{|x \cap y|}{|x \cup y|} = \frac{|x \cap y|}{|x| + |y| - |x \cap y|}, \tag{2.1}$$

where $|x|$ denotes the size of x.

Minhash: Minhash is a technique for converting a large set, let say X, to its signature representation. It randomly selects a permutation of the set and searches for a non zero element in that permutation. The minhash value of a set X is the index of first non-zero element of that randomly chosen permutation.

$$minhash(X) = min\{\Pi_k(X)\} \tag{2.2}$$

Here $\Pi_k(X)$ denotes the k^{th} permutation of set X. It is computationally expensive to compute all permutations of a set. A common workaround is using any method that simulates the effect of a random permutation. In this workaround, a hash function is selected, all values of X are hashed into the interval { 1,2,... |X| } and minimum of these hash values is selected. Minhash is now defined as

$$minhash(X) = min\{h_k(X)\} \tag{2.3}$$

Using a hash function to simulate the permutation behavior is a reasonable choice. It is not computationally expensive and provides random permutation of the input set.

Broder et al.[2] showed that if we compute $minhash$ values for two sets(x and y) then the probability of getting the same minhash values for both the sets is equal to their Jaccard similarity. i.e.,

$$Pr[h(x) = h(y)] = \frac{|x \cap y|}{|x| + |y| - |x \cap y|} = JS(x,y) \tag{2.4}$$

Therefore, if two sets have high Jaccard similarity then the probability of getting the same hash values is close to one and this probability will decrease as Jaccard similarity decreases. This makes minhash an efficient LSH for Jaccard similarity.

2.1.2 Simhash

Simhash [3] is an LSH for cosine similarity(CS). Cosine similarity is a measure of similarity between two non zero vectors from an inner product space that represents the cosine of angle between them. If $x, y \in \mathbb{R}^d$ are two non zero vectors then the cosine similarity is defined as

$$CS(x,y) = 1 - \frac{\arccos\left(\frac{x^T * y}{||x||_2 * ||y||_2}\right)}{\pi}, \tag{2.5}$$

where $||x||_2$ denotes the 2-norm of vector x. Observe that if x, y are perpendicular(i.e., not similar) then $CS(x,y)$ is 0, on the other hand if $x = y$ then $CS(x,y)$ is equal to 1.

Simhash is based on the concept of sign random projection introduced by Goemans et al. Given a vector x, simhash of x is defined as below

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

Here, w is a vector of length d and each component of w is generated from standard normal distribution, i.e., $w_i \sim N(0,1)$.

$$h_w(x) \equiv \begin{cases} 1, & \text{if } w^T * x \geq 0. \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

It was shown by Charikar et al.[3] that collision under sign random projection satisfy property that,

$$Pr[h_w(x) = h_w(y)] = 1 - \frac{\theta}{\pi} = CS(x,y), \tag{2.8}$$

where θ is the cosine of angle between x and y, i.e. $\theta = \arccos\left(\frac{x^T * y}{||x||_2 * ||y||_2}\right)$. The collision probability under this hash function is proportional to cosine similarity so this hash function family form an LSH for cosine similarity.

2.1.3 Hamhash

Indyk et al.[1] introduced an LSH hash family for Hamming similarity which we refer to as Hamhash. For any data point $x \in \{0,1\}^d$ its Hamhash value is defined as

$$h(x) = x_i, \tag{2.9}$$

where i is a uniformly chosen random index with $1 \leq i \leq d$

Hamhash is an LSH for Hamming similarity(HS). The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. For two binary strings $x, y \in \{0,1\}^d$, the Hamming distance is equal to the number of ones in $x \bigwedge y$

$$HD(x,y) = x \bigwedge y = \Sigma_{i=1}^{d} x_i \oplus y_i \tag{2.10}$$

The following equation expresses Hamming distance in terms of set operations which we use more frequently than Eq 2.10. If x, y are set representations of two binary strings(X,Y) of length d, then Hamming distance between x and y is defined as

$$HD(x,y) = |x \cup y| - |x \cap y| \tag{2.11}$$

The Hamming similarity(HS) is an inverse of Hamming distance and it is defined as

$$HS(x,y) = 1 - \frac{HD(x,y)}{d} = 1 - \frac{|x \cup y| - |x \cap y|}{d} \tag{2.12}$$

It is easy to see that for any two vectors x, y the probability of getting the same hash value under Hamhash is equal to their Hamming similarity. i.e.,

$$Pr[h(x) = h(y)] = 1 - \frac{|x \cup y| - |x \cap y|}{d} = HS(x,y) \tag{2.13}$$

2.1.4 L2LSH

The L_2 distance or Euclidean distance between two points $x, y \in \mathbb{R}^d$ is defined as the length of the straight line between x and y.

$$L_2(x,y) = \sqrt{||x||_2^2 + ||y||_2^2 - 2 * x^T * y} \tag{2.14}$$

Here, $||v||_2$ denotes the 2-norm of vector v. The 2-norm of any vector $v \in \mathbb{R}^n$ is defined as

$$||v||_2 = \sqrt{v^T * v} = \sqrt{v_1^2 + v_2^2 + ... + v_n^2} \tag{2.15}$$

L2LSH is an LSH for the L_2 distance. Datar et al. presented a novel LSH family for all L_p ($p \in (0,2]$) distance measures [4], when p = 2 it provides an LSH for L_2 distance. For any point $x \in \mathbb{R}^d$ its hash value is defined as

$$h_{w,b,r}(x) = \lfloor \frac{w^T * x + b}{r} \rfloor \tag{2.16}$$

Here r is a random number, b is a uniformly chosen random integer from [0,r] and w is a vector of size d, where each component is generated from the standard normal distribution, i.e., $w_i \sim N(0,1)$. The collision probability under this scheme was shown to be[4],

$$Pr[h_{w,b,r}(x) = h_{w,b,r}(x)] = F_r(l) \tag{2.17}$$

where

$$F_r(l) = 1 - 2 * \Phi(-r/l) - \frac{2}{\sqrt{2\pi} r/l}(1 - e^{-r^2/(2l^2)}) \tag{2.18}$$

in which $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ is the cumulative density function(cdf) of the standard normal distribution and $l = L_2(x,y)$. The collision probability $F_r(l)$ is a monotonically decreasing function of the distance l. Hence $h_{w,b,r}(x)$ form an LSH for L_2 distance.

2.2 Inner Product Similarity

Inner product similarity(IP) of two points(or vectors) $x, y \in \mathbb{R}^d$ is defined as

$$< x, y >= \Sigma_{i=1}^{d} x_i * y_i = x^T * y \tag{2.19}$$

Example 1: Let x = [1,0,1,0] and y = [1,1,0,0] then <x, y> = 1*1+0*1+1*0+0*0 = 1
Example 2: Let x = [1.2, 3, 1, 3.5] and y = [1,2.5, 3, 1] then <x, y> = $x^T * y$= 1.2*1+3*2.5+1*3+3.5*1 = 12.6
If the domain of the data points is $\{0,1\}^d$ and the data is represented as sets(where each element of the set represents index of non zero element) then the inner product is equal to the cardinality of their set intersection.

Inner product similarity is a ubiquitous measure in many data mining and machine learning problems. Srivastava et. al.[5] defined Maximum Inner Product Search(MIPS) problem , a variation of inner product similarity search problem, and described its applications in various domains. Despite having so many applications of inner product

similarity the following result[6] explains why it is hard to get our LSH for IP.

Lemma: There can not exist any LSH family for inner product similarity.

Proof: Lets assume there is a hash function h that is an LSH for IP. If $x \in R^d$ then the inner product of x with itself is $< x,x > = x^T * x = ||x||_2^2$. It is also possible to find another vector $y \in R^d$ such that the $< x,y >$ is greater than $< x,x >$. As per LSH definition if the similarity of x with y, $< x,y >$, is greater than with x itself, $< x,x >$, then $Pr[h(x) = h(y)]$ $\geq Pr[h(x) = h(x)]$. Since $Pr[h(x) = h(x)]$ is always one $Pr[h(x) = h(y)]$ can not be greater than 1. It contradicts the LSH definition and the assumption that h is an LSH for IP.

2.2.1 Asymmetric Transformations

The above proof is a very discouraging result because it indicates that sub-linear similarity search algorithm for inner product can not be obtained using LSH. In this subsection we discuss a technique proposed by Srivastava et al. [5] and Neyshaber et al. [7] that avoid this result and provide an LSH scheme for inner product.

They used transformation functions before applying LSH technique on the data. These transformation functions avoid the negative result in Lemma 1 [5][7]. These transformations have two functions, one function is used while preprocessing the data and the other function is used while querying the data. The usage of different functions while preprocessing and querying is called asymmetric transformation[6]. Asymmetric transformations provide Locality sensitive hashing framework for inner product similarity.

Binary-Transformation

Binary transformation is introduced by Srivastava el al. [5]. We use this technique with different LSH schemes to get an LSH framework for inner product similarity. Suppose D is a binary data set($D \subseteq \{0,1\}^d$), $|D|$= n (n is the number of points in D) and M denote the maximum number of ones in any data point $x_i \in D$, i.e.,

$$M = max_{i=1}^{n} |x_i| \tag{2.20}$$

The preprocessing transformation P is a function that maps d dimensional domain to d+M dimensional range by appending M-f_x ones followed by f_x zeros, f_x denotes the

number of ones in x. Formally,

$$P : \{0,1\}^d \rightarrow \{0,1\}^{d+M} \tag{2.21}$$

For $x \in D$, P(x) is defined as

$$P(x) = [x;1;1;1;.....1;1;0;0;.....0] \tag{2.22}$$

The query transformation Q is a function that maps d dimensional domain to d+M dimensional range by appending M zeros at the end of x. Formally,

$$Q : \{0,1\}^d \rightarrow \{0,1\}^{d+M} \tag{2.23}$$

For x ∈ D, Q(x) is defined as

$$Q(x) = [x;0;0.....0;0] \tag{2.24}$$

Simple-Transformation

This technique is introduced by Neyshabur et al.[7] and this can be used with any real data set. This technique assumes that norms of both data and query points are less than one.

The preprocessing transformation P is a function that maps d dimensional data to d+2 dimensional data by appending $\sqrt{1-||x_i||_2^2}$ and 0 at the end of x. Formally,

$$P : \mathbb{R}^d \rightarrow \mathbb{R}^{d+2} \tag{2.25}$$

$$P(x) = [x;\sqrt{1-||x_i||_2^2};0] \tag{2.26}$$

The query transformation Q is a function that maps d dimensional domain to d+2 dimensional range by appending 0 and $\sqrt{1-||x_i||_2^2}$ at the end of x. Formally,

$$Q : \mathbb{R}^d \rightarrow \mathbb{R}^{d+2} \tag{2.27}$$

$$Q(x) = [x;0;\sqrt{1-||x_i||_2^2}] \tag{2.28}$$

2.2.2 LSH for IP

In this section we use different asymmetric transformations, discussed in the previous subsection, to express various similarity measures in terms of inner product similarity. These reductions also construct LSH schemes for inner product similarity. Figure 2.1

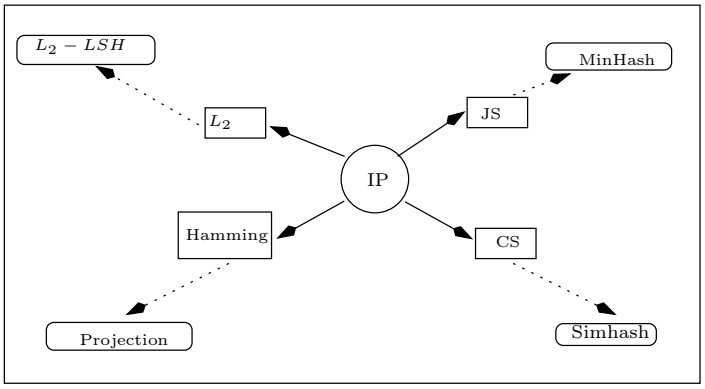

Figure 2.1: IP to Similarity Reductions

shows various possible reductions from inner product similarity to different similarity measures. The names in oval shaped boxes represent the names of hash functions that form LSH framework for corresponding similarity measures. Here **Projection** is an LSH for Hamming similarity but we use the name **Hamhash** to refer this family of hash functions.

Cosine Similarity-IP

The cosine similarity of two points x and q is defined as

$$CS(x,q) = 1 - \frac{\arccos\left(\frac{x^T * q}{|x| * |q|}\right)}{\pi} \tag{2.29}$$

By using binary transformations we can convert x and q to $x^{'}$ and $q^{'}$ where

$$x^{'} = Q(P(x)) = [x;1;1...1;1;0.....0;0], \tag{2.30}$$

$$q^{'} = P(Q(q)) = [q;0;0;....0;0;1;1.....1;0;0....0] \tag{2.31}$$

From Eq. 2.29, 2.30 and 2.31 the cosine similarity between x' and q' is

$$CS(x^{'},q^{'}) = 1 - \frac{\arccos\left(\frac{x^T * q}{M}\right)}{\pi} = 1 - \frac{\arccos\left(\frac{<x,q>}{M}\right)}{\pi} \tag{2.32}$$

If we design a simhash family to estimate the cosine similarity between $x^{'}$ and $q^{'}$ then it will also form an LSH for inner product between x and q.

If x and q are real vectors(particularly not binary vectors) then we need to use simple transformation. Simple transformation will convert x and q into $x^{'}$ and $q^{'}$ where

$$x^{'} = P(x) = [x;\sqrt{1 - ||x||_2^2};0], \tag{2.33}$$

$$q^{'} = Q(q) = [q; 0; \sqrt{1 - ||q||_2^2}] \tag{2.34}$$

From Eq. 2.29, 2.33 and 2.34 the cosine similarity between $x^{'}$ and $q^{'}$ is defined as

$$CS(x^{'}, q^{'}) = 1 - \frac{\arccos\left(x^T * q\right)}{\pi} = 1 - \frac{\arccos(< x, q >)}{\pi} \tag{2.35}$$

If the norm of any point is greater than one then simple transformation can not be applied. First the norm of points should be bounded and it can be done by dividing all points with largest possible norm. Suppose M denotes the largest possible norm then the cosine similarity is equal to

$$CS(x^{'}, q^{'}) = 1 - \frac{\arccos\left(\frac{x^T * q}{M^2}\right)}{\pi} = 1 - \frac{\arccos\left(\frac{< x, q >}{M^2}\right)}{\pi} \tag{2.36}$$

Hamming Similarity-IP

In this work we use Hamming similarity to measure the similarity between binary data. Suppose x and q are set representations of two binary data points then the Hamming similarity between x and q is defined as

$$HS(x, q) = 1 - \frac{|x \cup q| - 2 * |x \cap q|}{d} = 1 - \frac{|x \cup q| - 2 * < x, q >}{d} \tag{2.37}$$

If we use binary mappings to convert x and q into $x^{'}$ and $q^{'}$, then

$$x^{'} = Q(P(x)) = [x; 1; 1...1; 1; 0.....0; 0]] \tag{2.38}$$

$$q^{'} = P(Q(q)) = [q; 0; 0; ....0; 0; 1; 1.....1; 0; 0....0] \tag{2.39}$$

from Eq. 2.37, 2.38, 2.39

$$HS(x^{'}, q^{'}) = 1 - \frac{|x \cup q| - |x \cap q|}{2M + d} = 1 - \frac{2 * M - 2 * < x, q >}{2M + d} \tag{2.40}$$

If we design Hamhash family to estimate the Hamming similarity then it will also estimate the inner product similarity. This Hamhash family form an LSH scheme for inner product similarity.

Jaccard Similarity-IP

The Jaccard similarity between two sets x, q is defined as

$$JS(x, q) = \frac{|x \cap q|}{|x \cup q|} = \frac{|x \cap q|}{|x| + |q| - |x \cap q|} \tag{2.41}$$

If we use binary transformation to convert x and q into $x^{'}$ and $q^{'}$, then

$$x^{'} = Q(P(x)) = [x;1;1...1;1;0.....0;0] \tag{2.42}$$

$$q^{'} = P(Q(q)) = [q;0;0;....0;0;1;1.....1;0;0....0] \tag{2.43}$$

From Eq. 2.41, 2.42, 2.43

$$JS(x^{'},q^{'}) = \frac{|x^{'} \cap q^{'}|}{|x^{'} \cup q^{'}|} = \frac{|x \cap q|}{|x^{'}| + |q^{'}| - |x \cap q|} = \frac{|x \cap q|}{2 * M - |x \cap q|} = \frac{< x,q >}{2 * M - < x,q >} \tag{2.44}$$

If we design Minhash family to estimate the Jaccard similarity it will also estimate the inner product similarity. This Minhash family form an LSH scheme for IP.

L_2 distance-IP

The L_2 distance between two points x and q is defined as

$$L_2(x,q) = \sqrt{||x||_2^2 + ||q||_2^2 - 2 * x^T * q} \tag{2.45}$$

Suppose x and q are two binary points ($x,q \in \{0,1\}^d$) then we can use binary transformation to convert them into x' and $q^{'}$, i.e.,

$$x^{'} = Q(P(x)) = [x;1;1...1;1;0.....0;0], \tag{2.46}$$

$$q^{'} = P(Q(q)) = [q;0;0;....0;0;1;1.....1;0;0....0] \tag{2.47}$$

From Eq. 2.45, 2.46, 2.47

(2.48)

$$L_2(x^{'},q^{'}) = \sqrt{||x^{'}||_2^2 + ||q^{'}||_2^2 - 2 * x^{'T} * q^{'}} = \sqrt{M + M - 2 * < x,q >} = \sqrt{2 * M - 2 * < x,q >}$$

If the norm of any point is greater than one then simple transformation can not be applied. First the norm of points should be bounded and it can be done by dividing all points with largest possible norm. In this case the L_2 distance will be

(2.49)

$$L_2(x^{'},q^{'}) = \sqrt{||x^{'}||_2^2 + ||q^{'}||_2^2 - 2 * x^{'T} * q^{'}} = \sqrt{1 + 1 - 2 * \frac{< x,q >}{M^2}} = \sqrt{2 - 2 * < x,q > /M^2}$$

If we design an L_2LSH family for L_2 distance then it will also estimate the inner product similarity. This L_2LSH family form an LSH scheme for IP.

CHAPTER 3

LSH Analysis

In the previous chapter we reviewed various well known locality sensitive hashing schemes for different similarity measures and discussed how to construct LSH schemes for inner product similarity. In this chapter we analyze the performance of different LSH schemes for inner product similarity. We also introduce a two level banding technique that improves the performance of LSH schemes for inner product similarity.

In section 4.1 we review standard LSH bucketing technique. In section 4.2 we use this technique to analyze the performance of different LSH schemes. This analysis provides how many hash function evaluations are required to construct (s_1, s_2, tp, fp)-sensitive hash family from (s_1, s_2, p_1, p_2)-sensitive hash family. In section 4.3 we introduce two level banding scheme that reduces the number of hash function evaluations. The key observation is that the LSH family with high probability difference $(p_1 - p_2)$ perform better than the LSH family with low probability difference.

3.1 LSH Banding Technique

Locality sensitive hashing gives sub linear query time capability for many problems[1][6]. One way of improving the performance of LSH based algorithms is banding hash values of data points. The classical (K, L)-banding technique , which is also called and-or banding,

is defined as follows: Lets assume we have access to a locality sensitive hash family H

1. It randomly selects $K * L$ hash functions from H.
2. Any two points x, q are hashed into the same location iff

$$\exists j \text{ s.t. } b_j(x) \equiv b_j(q) \text{ for } 1 \leq j \leq L,$$
$$\text{here } b_j(x) \equiv b_j(q) \text{ iff } \forall i \; h_{ij}(x) = h_{ij}(q) \text{ for } 1 \leq i \leq K$$

Here we divide $K * L$ hash functions into L bands where each band has size K. b_j denotes the j^{th} band and h_{ij} denotes the i^{th} hash function in j^{th} band.

3.2 True Positive Rate(tp) and False Positive Rate(fp)

In this section we define true positive rate(tp) and false positive rate(fp) in locality sensitive hashing context. Given a dataset D, inner product threshold θ, error margin ϵ and query point q, tp and fp are defined as :

True Positive Rate(tp): It is the fraction of points which have inner product(with query q) greater than the required threshold and LSH scheme also hash them into the same location as query point q.

$$tp = \frac{|pred\_true\_positive \cap actual\_true\_positive|}{|actual\_true\_positive|} \tag{3.1}$$

False Positive Rate(fp): It is the fraction of points which have inner product(with query q) less than the required threshold but the LSH scheme hash them into the same location as query point q.

$$fp = \frac{|pred\_true\_positive \cap actual\_false\_positive|}{|actual\_flase\_positive|} \tag{3.2}$$

pred_true_positive is a set of points where each point get the same hash value as query point q. actual_true_positive is a set of points where each point in this set has inner product(with query point q) greater than required threshold. actual_false_positive is a

set of points where each point in this set has inner product(with query point q) less than required threshold.

For any query point q, if we guarantee tp = 0.9 and fp = 0.1 then the candidates that any LSH scheme generates are more likely to be good candidates, i.e. 90% of the actual true positives are retrieved at the cost of 10% extra false positive candidates. In the next section we compare the required number of hash function evaluations for different LSH schemes to achieve given tp and fp values.

3.3 LSH Analysis

In this section we analyze the performance of different LSH functions for inner product similarity. If we have (s_1, s_2, p_1, p_2)-sensitive hash family H for inner product then it gives the following guarantees(as per the definition of LSH), Let $x, q \in \mathbb{R}^d$

1. If $< x, q > \geq s_1$ then $Pr_{h \in H}[h(x) = h(q)] \geq p_1$
2. If $< x, q > \leq s_2$ then $Pr_{h \in H}[h(x) = h(q)] \leq p_2$

LSH	s_1	s_2	p_1	p_2
Minhash	$\frac{\theta}{2M-\theta}$	$\frac{(1-\epsilon)\theta}{2M-(1-\epsilon)\theta}$	$\frac{\theta}{2M-\theta}$	$\frac{(1-\epsilon)\theta}{2M-(1-\epsilon)\theta}$
Simhash	$1-\frac{\arccos\left(\frac{\theta}{M}\right)}{\pi}$	$1-\frac{\arccos\left(\frac{(1-\epsilon)\theta}{M}\right)}{\pi}$	$1-\frac{\arccos\left(\frac{\theta}{M}\right)}{\pi}$	$1-\frac{\arccos\left(\frac{(1-\epsilon)\theta}{M}\right)}{\pi}$
Hamhash	$\frac{d+2\theta}{d+2M}$	$\frac{d+2(1-\epsilon)\theta}{d+2M}$	$\frac{d+2\theta}{d+2M}$	$\frac{d+2(1-\epsilon)\theta}{d+2M}$
L2LSH	$d_1 = \sqrt{2M-2\theta}$	$d_2 = \sqrt{2M-2(1-\epsilon)\theta}$	$F_r(d_1)$	$F_r(d_2)$
Simple Simhash($\|x\| \leq 1$)	$1-\frac{\arccos(\theta)}{\pi}$	$1-\frac{\arccos((1-\epsilon)\theta)}{\pi}$	$1-\frac{\arccos(\theta)}{\pi}$	$1-\frac{\arccos((1-\epsilon)\theta)}{\pi}$

Table 3.1: List of LSH Functions

In Table 4.2 we listed all LSH family of hash functions for inner product similarity as per this definition. Here θ denotes the inner product threshold and ϵ denotes the error margin. LSH bucketing algorithm will construct $(\theta, (1-\epsilon)\theta, tp, fp)$-sensitive hash family from H. Here

$$tp = 1-(1-p_1^K)^L \tag{3.3}$$

and

$$fp = 1-(1-p_2^K)^L \tag{3.4}$$

We can get different values for tp, fp by applying different values to K, L. We got the following expressions for K, L by solving Eq. 3.3 and Eq. 3.4.

(3.5) $$K = \log_{\rho}^{\delta}$$

Here $\delta = \log_{(1-fp)}^{(1-tp)}$ and $\rho = \frac{p_1}{p_2}$

(3.6) $$L = -\frac{\ln(1-tp)}{p_1^k}$$

The product $K * L$ value denotes the number of hash evaluations required for any hash family H to get given tp and fp values.

Probability Difference($p_1 - p_2$):

When we analyzed the performance of different locality sensitive hash families we found that the probability difference($0 \leq p_1 - p_2 \leq 1$) affect the performance. For any hash family if the probability difference is high then that hash family requires less number of hash function evaluations($K * L$) to achieve required tp and fp values.

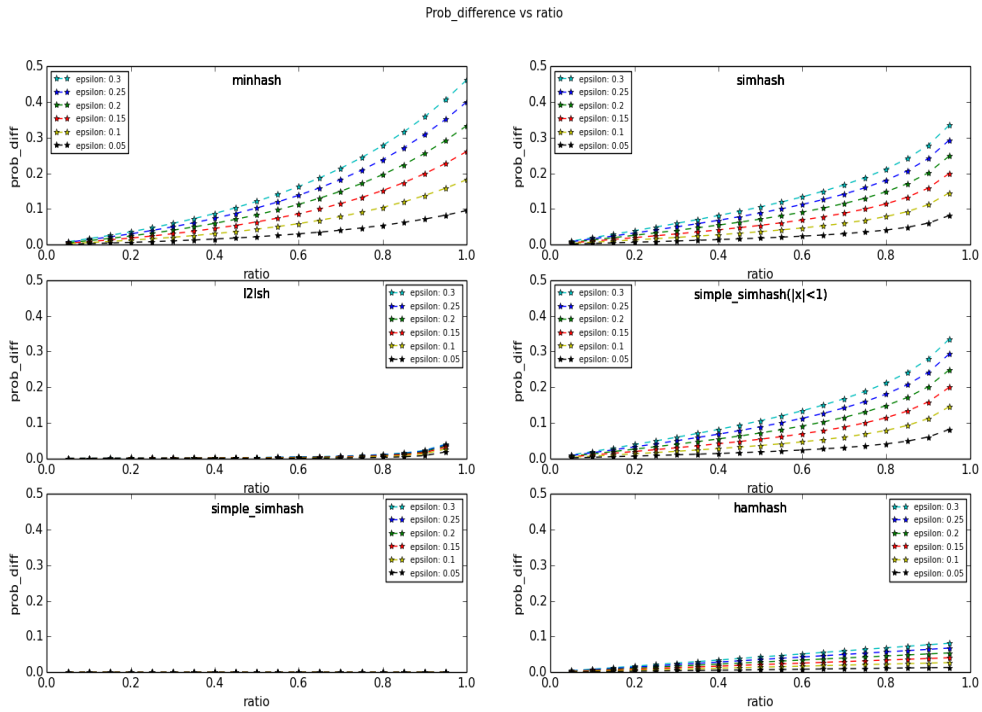

Figure 3.1: Prob_Diff vs Ratio(θ /M)

Figure 3.1 shows probability differences of different hash families at different levels of error ϵ. X-axis represent the ratio θ /M. In Figure 3.1 the probability difference decreases with decrease in ratio value. Among all LSH families Minhash and Simhash have better probability differences so we focus more on these hash function families.

3.3.1 Theoretical Analysis:

In this subsection we show how many hash function evaluations are required for Sinhash and Minhash to achieve required tp and fp values. Figure 3.2 and 3.3 shows the number of hash functions required for Minhash, Simhash to achieve various tp and fp respectively.

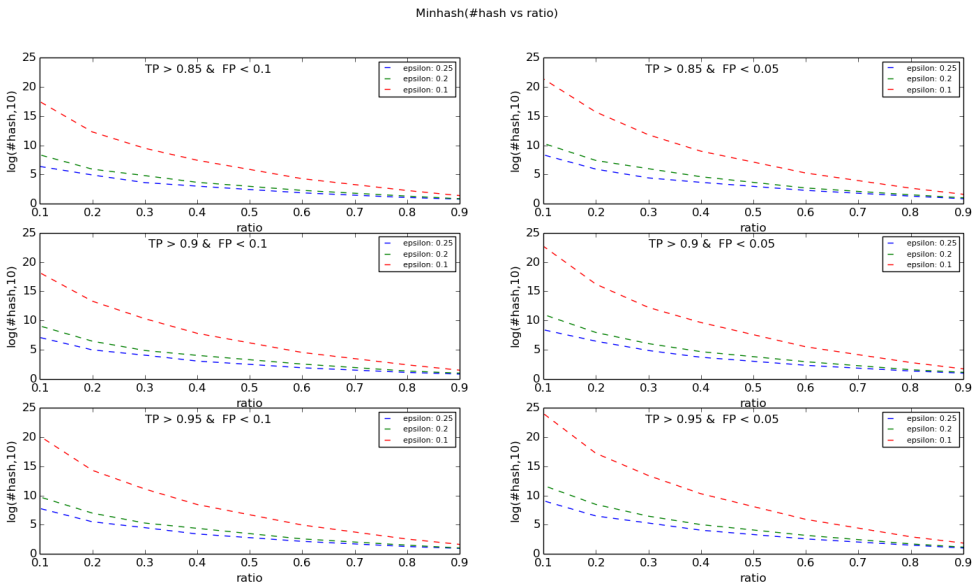

Figure 3.2: no of hash vs ratio(θ/M)

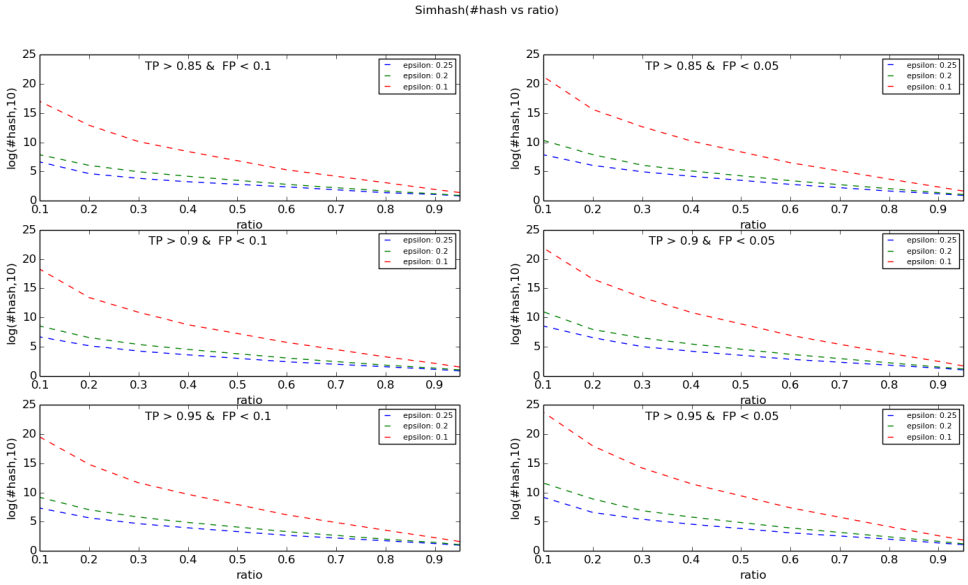

Figure 3.3: no of hash vs ratio(θ/M)

Y-axis values are represented on log scale with base 10. From the graphs it is clear that if $tp \approx 1$ and $fp \approx 0$ then the required number of hash function evaluations($K * L$) are large. If we compare 3.2 and 3.3 it is clear that Minhash takes relatively less number of hash values compare to Simhash to achieve the same tp and fp guarantees. This implies that Minhash is a reasonable choice among all other hash families to achieve the required tp and fp values.In the experiment section we show this observation on two datasets.

3.3.2 Experimental Results

In this section we analyze the performance of Minhash and Simhash on the two real datasets.

Datasets:

We used two publicly available high dimensional sparse datasets: EP2006 and MNIST. EP2006 is a binary "BOW" representation of 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.

Dataset	#Train	#Query	#Dim	nonzeros(mean,std)	M
EP2006	16088	3309	4,272,227	(6072,3208)	27,299
MNIST	50000	10000	784	(150,41)	306

Table 3.2: Datasets

Experiment Details:

In this experiment we compare the theoretical number of hash function evaluations to practical number of hash function evaluations. We followed the following procedure to find the theoretical number of hash function evaluations($K * L$) to achieve required tp and fp values.

1. Select a hash function h randomly from Simhash(or Minhash) hash family.

2. Randomly select a query point q from query set, and find its hash value $h(q)$

3. For each item x in the training set, if $h(x) = h(q)$ and $< x, q >$ greater than threshold θ then add x to pred_true_postive and act_true_positive.

4. For each item x in the training set, if $h(x) = h(q)$ and $< x, q >$ less than (1-ϵ)θ then add x to pred_true_postive and act_false_positive.

5. For each item x in the training set, if $h(x)! = h(q)$ and $< x, q >$ greater than θ then add x to act_true_positive

6. For each item x in the training set, if $h(x)! = h(q)$ and $< x, q >$ less than (1-ϵ)θ then add x to act_false_positive.

7. From Eq. 3.1 calculate tp and consider it as p_1

8. From Eq. 3.2 calculate fp and consider it as p_2.

9. Using K, L formulas compute their values. This product value ($K * L$) is the required theoretical number of hash function evaluations.

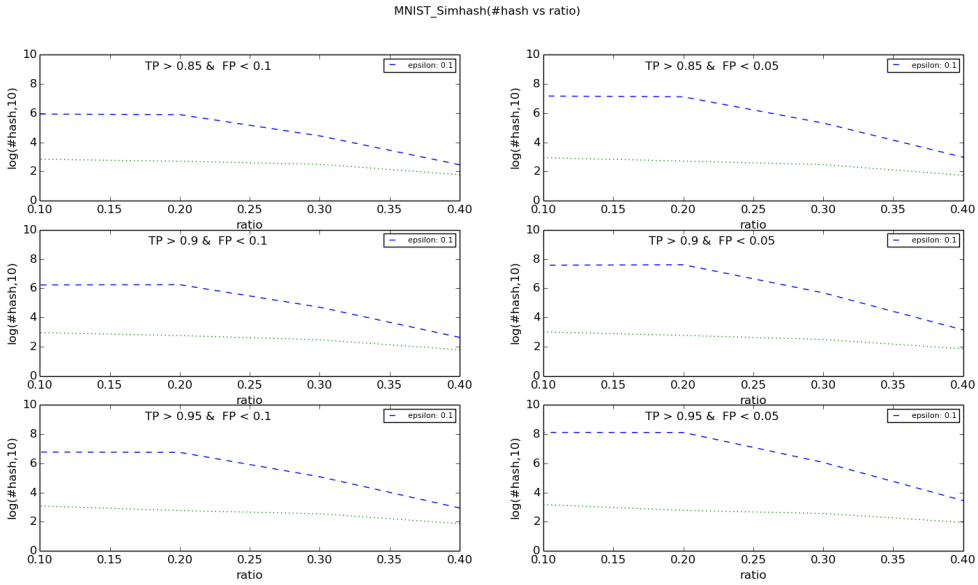

Figure 3.4: Simhash performance on MNIST Dataset

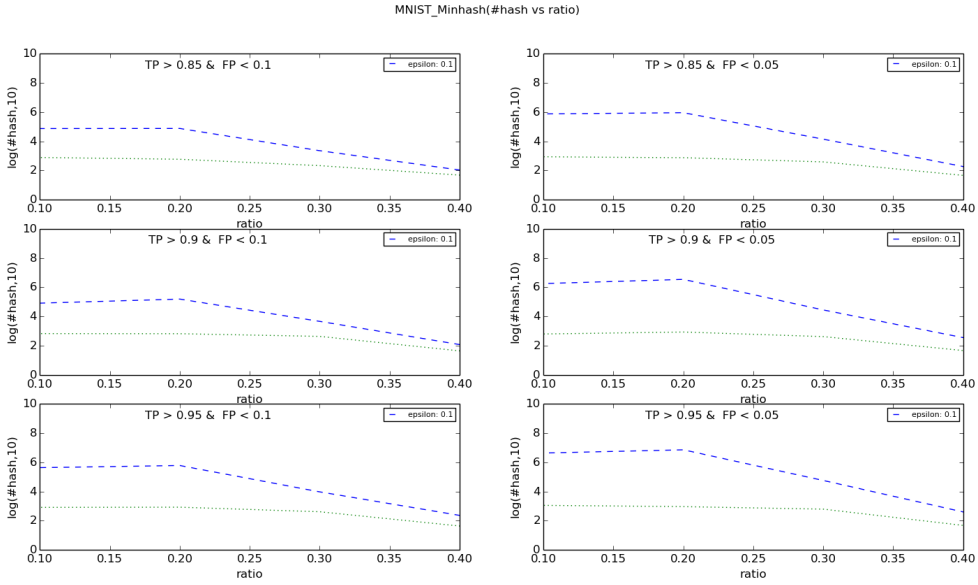

Figure 3.5: Minhash performance on MNIST Dataset

Figure 3.4 and Figure 3.5 show the number of hash function evaluations(#hash) required for Simhash and Minhash on MNIST dataset to achieve required tp and fp. Values on Y-axis are represented on log scale with base 10. Dotted lines(. .) represent the practical number of hash function evaluations required to achieve given tp and fp rate and

line with dashes(- -) represent theoretical number computed using above procedure. We computed the practical numbers($K*L$) by running the simulation with different K,L values. The goal of this approach is to find the practical K,L values that are better than theoretical values and at the same time will give the same tp and fp. We can see that the practical values are far less than theoretical values.

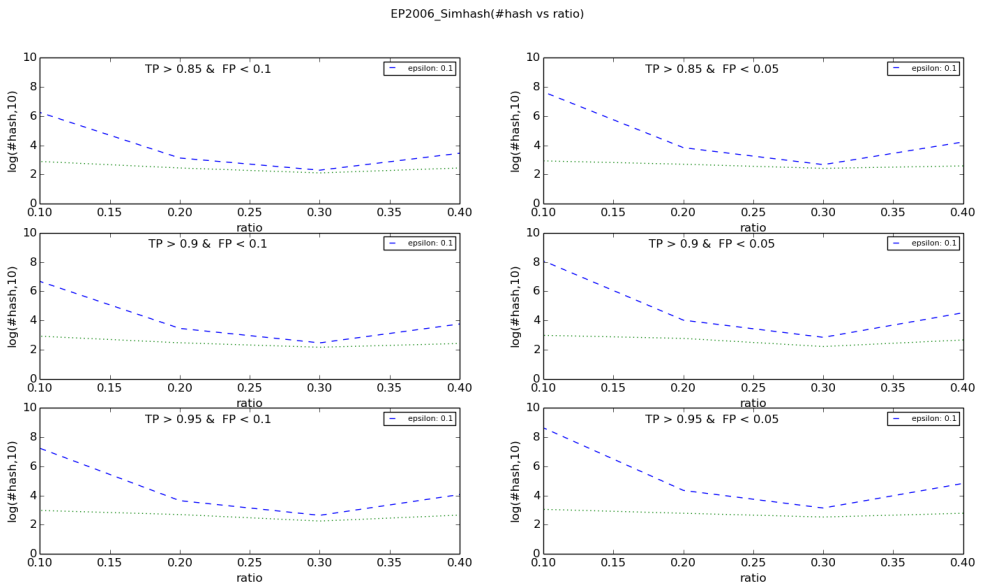

Figure 3.6: Simhash performance on EP2006 Dataset

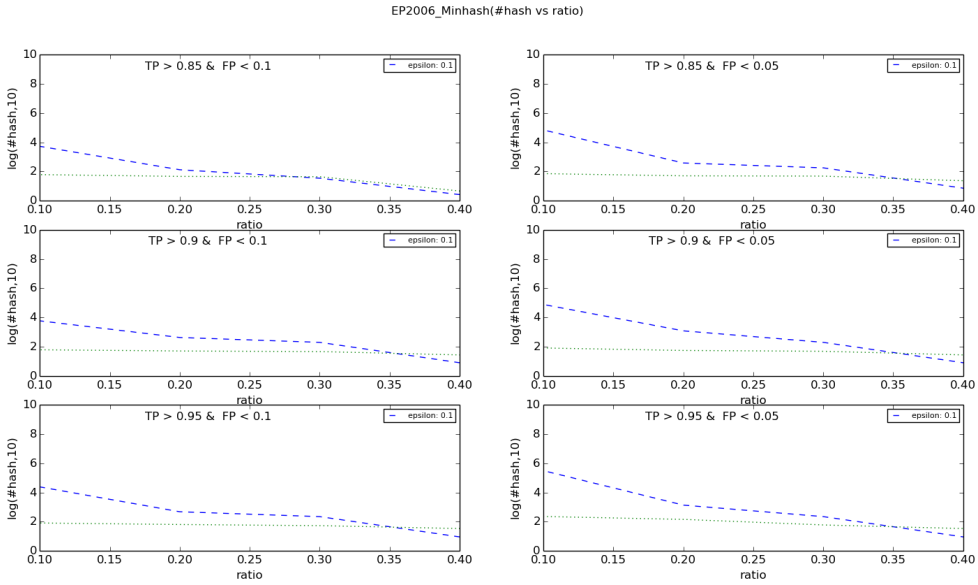

Figure 3.7: Minhash performance on EP2006 Dataset

Figure 3.6 and Figure 3.7 show the number of hash function evaluations(#hash) required for Simhash and Minhash on EP2006 dataset to achieve required tp and fp. Values on Y-axis are represented on log scale with base 10. Dotted lines(. .) represent the practical number of hash function evaluations required to achieve given tp and fp rate and line with dashes(- -) represent theoretical number computed using above procedure. We computed the practical values($K * L$) by running the simulation with different K,L values. The goal of this approach is to find the practical K,L values that are better than theoretical values and at the same time will give the same tp and fp. We can see that the practical values are far less than theoretical values.

3.4 Two Level Banding

In the previous section we analyzed the performance of Simhash and Minhash with single level (K,L)-banding technique. In the previous section we observed a pattern that shows if the probability difference ($p_1 - p_2$) is high then that LSH family of hash functions require less number of hash evaluation to achieve required tp and fp values.

In this section we leverage this pattern and reduce the number of hash function evaluations with two level banding scheme. In this two level banding scheme, first level is used

to boost the probability difference and second level works as standard (K,L) banding technique. In the first level we use Or-banding. Suppose we have access to a LSH scheme H then the two level banding technique is defined as:

1. It randomly selects $K_1 * K * L$ hash functions from H. K_1 is the band size at first level.

2. At first level it divide $K_1 * K * L$ hash functions into $K * L$ bands where each band size is K_1.

3. Any two points x, q are said to be hashed into the same location if the following conditions are satisfied

$$\begin{gathered}\exists j \text{ s.t. } b_j(x) \equiv b_j(q)\ 1 \leq j \leq L,\\ b_j(x) \equiv b_j(q) \text{ iff } \forall i\ O_{ij}(x) \equiv O_{ij}(q)\ 1 \leq i \leq K,\\ O_{ij}(x) \equiv O_{ij}(q) \text{ iff } \exists k \text{ s.t } h_{ijk}(x) = h_{ijk}(q)\ 1 \leq k \leq K_1\end{gathered}$$

b_j denotes the j^{th} band in the second level. O_{ij} denotes the $(i * j)^{th}$ band in first level. h_{ijk} denotes the $(i * j * k)^{th}$ hash function.

If H is a $(s_1, s_2, p_1, p_2)-$sensitive hash family then the first level convert it into $(s_1, s_2, P_1, P_2)-$sensitive hash family. Second level convert this first level output into $(s_1, s_2, tp, fp)-$sensitive hash family. Here

$$\begin{aligned}P_1 &= 1 - (1 - p_1)^{K_1},\\ P_2 &= 1 - (1 - p_2)^{K_1}\end{aligned}$$

and

$$\begin{aligned}tp &= 1 - (1 - P_1^K)^L\\ fp &= 1 - (1 - P_2^K)^L\end{aligned}$$

3.4.1 Theoretical Analysis

In this subsection we analyze the number of hash function evaluations required for Simhash and Minhash with two level banding scheme. Using this theoretical analysis we show that the required number of hash function evaluations for two level banding technique are far less than the single level banding technique.

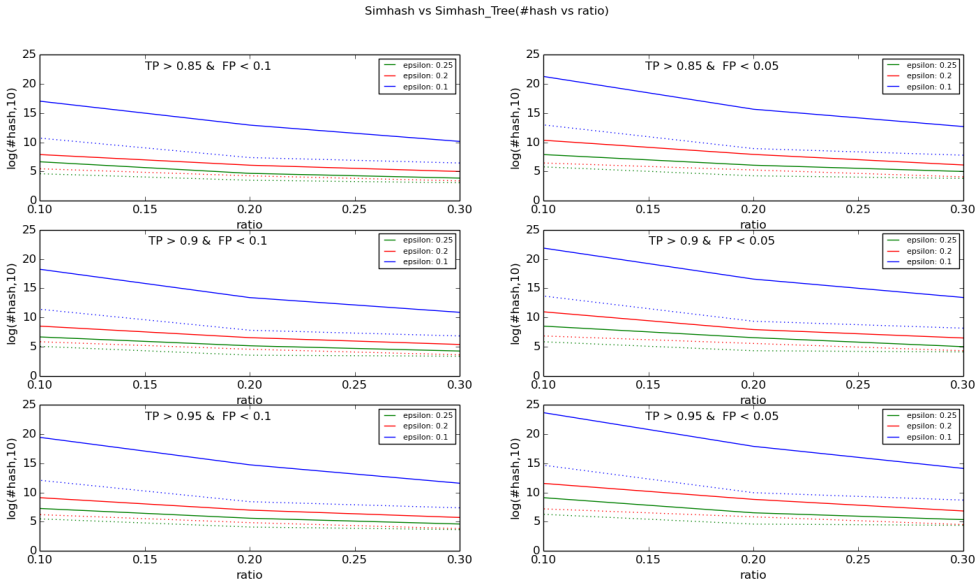

Figure 3.8: Simhash : Single vs Two Level Banding

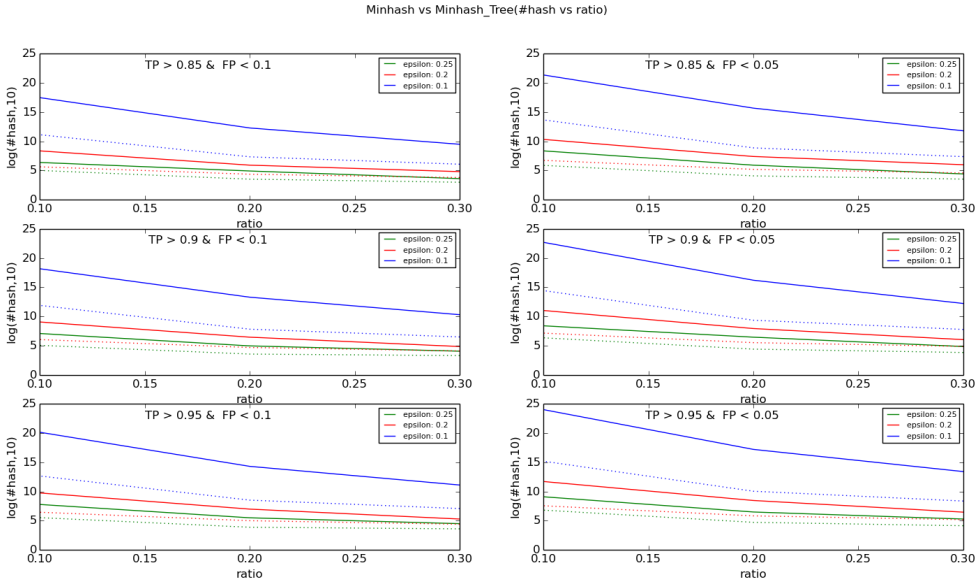

Figure 3.9: Minhash : Single vs Two Level Banding

Figure 3.8 and Figure 3.9 shows the performance improvement or decrease in number of hash function evaluations for Simhash and Minhash. Thick lines represent single level banding and dotted lines represent two level banding, lines with same color compare single and two level banding. We did not solve for optimal K_1 value, we randomly took

K_1 value such that $P_1 - P_2$ is greater than 0.25. Using this P_1, P_2, we found K, L. Total number of hash function evaluations required are $K_1 * K * L$. Figure 3.8 and Figure 3.9 represent these values on log scale with base 10.

Since we did not use optimal K_1 value, we strongly believe that the actual $K_1 * K * L$ value required to achieve given tp and fp values using this technique is far less than the values represented in the above Figures.

3.4.2 Experimental Results

In the above subsection we showed theoretically that using two level banding technique the performance of an LSH schemes can be improved. In this section we observe the same on two real datasets.

Datasets:
We used two publicly available high dimensional sparse datasets: EP2006 and MNIST. EP2006 is a binary "BOW" representation of 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.

Dataset	#Train	#Query	#Dim	nonzeros(mean,std)	M
EP2006	16088	3309	4,272,227	(6072,3208)	27,299
MNIST	50000	10000	784	(150,41)	306

Table 3.3: Datasets

Experiment Details
To compute the number of hash evaluations required for single level banding we followed the procedure that we described in the previous subsection. In two level banding scheme, we randomly selected K_1 value to increase the probability difference more than 0.25, i.e., $(P_1 - P_2) \geq 0.25$, with these new P_1 and P_2 values we computed K, L values. Therefore the total number hash evaluations in two level banding scheme are $K_1 * K * L$.

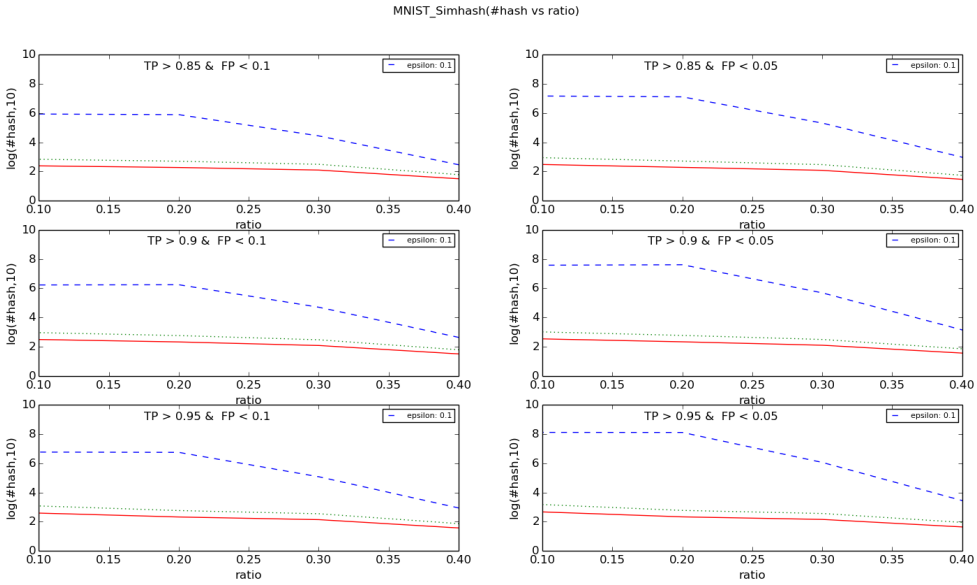

Figure 3.10: Simhash : Single vs Multi Level Banding(MNIST)

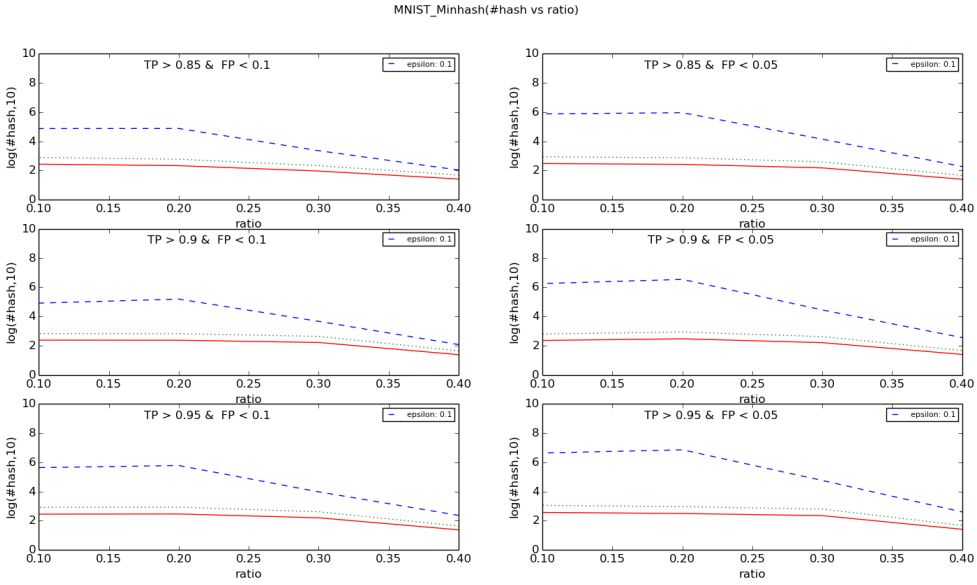

Figure 3.11: Minhash : Single vs Multi Level Banding(MNIST)

Figure 3.10 and 3.11 shows the performance of Simhash and Minhash on MNIST dataset. Lines with dashes represent theoretical number of hash function evaluations required for single level banding. Dotted lines represent practical number of hash evaluations required for single level banding. Lines with red color represent practical

number of hash evaluations required for hierarchical banding scheme. It is clear that hierarchical banding scheme need less number of hash function evaluations to achieve the same tp and fp values.

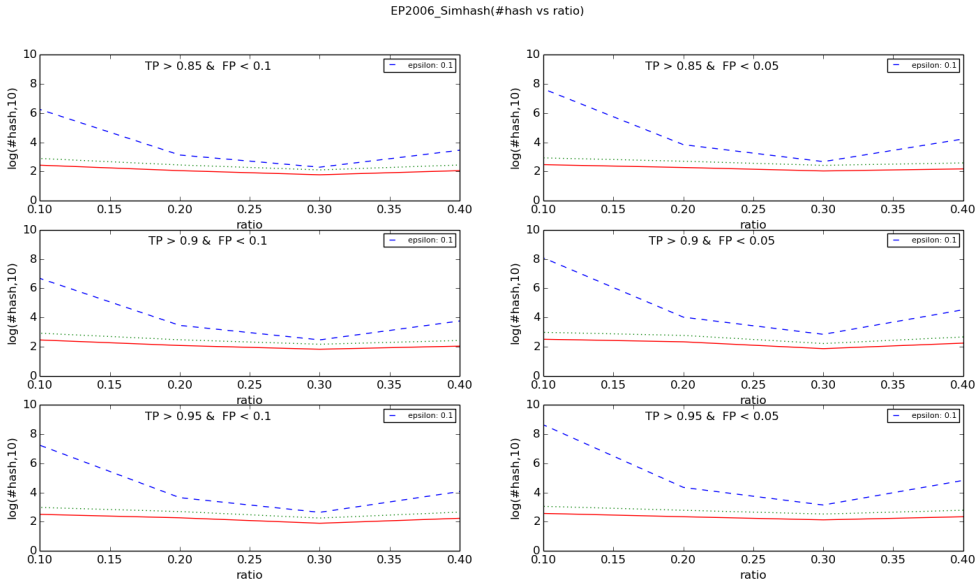

Figure 3.12: Simhash : Single vs Multi Level Banding(EP2006)

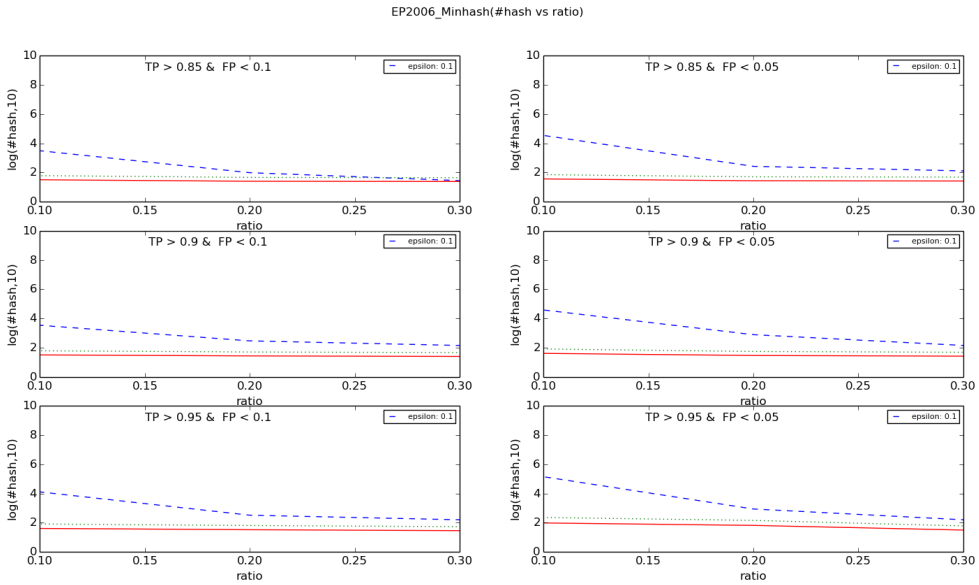

Figure 3.13: Minhash : Single vs Multi Level Banding(EP2006)

Figure 3.12 and 3.13 shows the performance of Simhash and Minhash on EP2006 dataset.It is clear that hierarchical banding scheme need less number of hash values to achieve the same tp and fp rate.

3.5 Conclusion

From the theoretical experiments and practical results we can conclude the following results:

1. The hash family with better probability difference($p_1 - p_2$) need less number of hash function evaluations to achieve the required tp and fp rate.
2. The required practical number of hash function evaluations are far less than the theoretical number .
3. The number of hash function evaluations required to achieve given tp and fp rate can be reduced using two level banding.

Finding the optimal K_1 value for first level is an interesting problem for future work. For two level banding scheme we can define the problem as:

Problem:
Given (s_1, s_2, p_1, p_2)-sensitive LSH, tp and fp,

$$\text{Minimize f}(K_1, K, L) = K_1 * K * L$$
$$\text{s.t } tp = 1 - (1 - P_1^K)^L$$
$$fp = 1 - (1 - P_2^K)^L$$
$$\text{where } P_1 = 1 - (1 - p_1)^{K_1}$$
$$P_2 = 1 - (1 - p_2)^{K_1}$$

We solved for K and L values, i.e.,

$$K = \log_{\rho}^{\delta}$$

Here $\delta = \log_{(1-fp)}^{(1-tp)}$ and $\rho = \frac{P_1}{P_2}$

$$L_2 = -\frac{\ln(1-tp)}{P_1^K}$$

CHAPTER 4

LSH IN FREQUENT PATTERN MINING

In the previous chapter we designed and analyzed the performance of different LSH techniques with respect to inner product similarity(IP). In this chapter we explore the usage of locality sensitive hashing technique in frequent itemset(FI) mining.

Frequent itemset mining is the problem of extracting all itemsets that occur with required frequency in a transactional database. The Apriori (and its variants) is a famous algorithm to solve frequent itemset mining problem. The overhead in the Apriori algorithm is the number of candidates it generates at every step and the number of I/O operations it requires to filter these candidates. In this chapter we investigate the usage of LSH framework, developed in the previous chapters, to overcome these problems. In this chapter we incorporate LSH framework into traditional Apriori algorithm such that the candidate generation step will be randomized to filter more candidates(which do not have the required frequency).

In section 6.1 we review frequent itemset mining problem and different techniques to solve this problem. In section 6.2 we explain various notations and concepts required to understand Apriori, LSH-Apriori algorithms. In section 6.3 we introduce LSH-Apriroi algorithm and explain how it uses locality sensitive hashing technique. In section 6.4 we show the performance of LSH-Apriori on real datasets. In this section we show that LSH-Apriori retrieves good percentage of frequent itmes at each support level with less number of candidates than traditional Apriori algorithm. In section 6.5 we conclude this

chapter with important observations.

4.1 Introduction

Frequent itemset mining problem deals with the extraction of all itemsets that occur frequently(the notion of frequency is defined in the later part of this section) in the transactional databases. The frequent itemset mining in a transactional databases was first appeared in retail industry for discovering association rules among all items in the retail store. Association rule defines the dependency of one item's sale on another item. However, the frequent itemset mining problem has lot of application in different domain like finding correlation[8], finding episodes[9] and clustering[10].

Mathematically, each transaction can be formulated as a subset of set of items in corresponding retail store or a transactional database D. Given a database D and frequency threshold(called support) s $\in$ (0,1) the main goal of frequent itemset mining algorithms is to extract all items whose support value is greater than s. The support of two item sets x and y is defined as

$$\text{Support(s)} = \frac{|X \cap Y|}{|D|}$$

Here X, Y denote the set of transactions where each transaction contains corresponding itemset.
Frequent itemset mining is a non-trivial problem in data mining. There are so many approaches to solve this problem but the problem of checking whether there exist any itemset of size k whose support is greater than s is a NP-complete problem[11]. One more problem is checking for the support requirement of an itemset- Whole database has to be read to check its support value which requires more I/O effort. The current algorithms that solve frequent itemset mining problem follow either incrementally generating the candidates or not generating candidates at all. One of the famous algorithm that follow former approach is Apriori algorithm[12].

Apriori algorithm is explained in section 3. It is based on anti-monotonicity property of itemsets which says that the subset of any infrequent itemset is also infrequent. Apriori algorithm works in bottom-up fashion where it finds itemsets in FI in increasing order of their size. The bottleneck in Apriori algorithm is that it generates lot of candidates at every stage, which requires more I/O and time to check whether they satisfy the support

requirement.

To overcome this bottleneck the algorithm has to generate candidates such that majority of them will satisfy the support requirement. Our work investigates this approach using locality sensitive hashing technique.

Previous work:

There are considerable number of solutions based on Apriori approach but very few based on Hashing. PCY is one of the techniques that uses hashing but its approach is entirely different [13]. PCY's main motivation is to use main memory more efficiently but LSH-Apriori's approach differs with it entirely.

Cohen et.al. uses similar approach as LSH-Apriori but they intentionally did not consider support requirement[14]. They developed a set of techniques to find interesting association among items without any support pruning.

LSH-Apriori extracts frequent itemsets by taking support threshold into consideration. LSH-Apriori reduces the number of candidates by mapping similar items to same location with high probability. In this chapter LSH-Apriori uses Jaccard similarity and cosine similarity as a similarity measures and minhash, simhash as corresponding LSH hash families.

4.2 Background

In this section we introduce technical background required to understand the LSH-Apriori algorithm. We will use the following notations in this chapter

1. **I (Itemset) :** I denotes the set of items i.e., I = $\{i_1, i_2, ...i_m\}$.
2. **D (Database):** Each item in D is a transaction t_i. Each transaction $t_i \in I^m$. D can be represented as m x n matrix. The row r represent presence of item i_r in the transactions $\{t_1, t_2, ..., t_n\}$.
3. **m:** Number of items in I
4. **n:** Number of transactions in D

5. **l:** l denotes the level at which the algorithm is progressing.
6. ϵ **:** Error margin
7. **TP:** True positive rate, It is defined formally in chapter 4.
8. **FP:** False positive rate, It is also defined formally in chapter 4.

Support:
Let x,y be two item sets and X, Y denote the corresponding rows in m x n data matrix, then

$$\text{Support(x,y)} = \frac{|X \cap Y|}{|D|} = \frac{<x,y>}{n}$$

To use LSH hash functions for estimating the support value between two item sets, we need to establish relation between similarity measures and support. The following theory will describe this approach.

Jaccard Similarity and Support:

From section 3.2.3, we know that the Jaccard similarity of two item sets x, y (after applying binary mapping) is

$$\text{JS}(x', y') = \frac{<x,y>}{2M - <x,y>}$$

From the support equation

$$< x, y > = \text{support}(x, y)\text{*n} = \text{s*n}$$

From these two equations we can represent JS in terms of support, i.e.,

$$\text{JS}(x', y') = \frac{s*n}{2M - s*n}$$

Here, $0 \leq s * n \leq M$

Cosine Similarity and Support:
From section 3.2.1, we know that the cosine similarity of two items sets x, y (after applying binary mapping) is

$$CS(x', y') = 1 - \frac{\arccos\left(\frac{x^T * y}{M}\right)}{\pi} = 1 - \frac{\arccos\left(\frac{<x,y>}{M}\right)}{\pi}$$

By combining the support equation and the above equation, we get

$$CS(x^{'},q^{'}) = 1 - \frac{\arccos(\frac{s*n}{M})}{\pi}$$

Here, $0 \leq s*n \leq M$
Using these two similarity equations for Jaccard similarity and cosine similarity we get the following (s_1, s_2, p_1, p_2)-sensitive LSH families for support.

LSH	s_1	s_2	p_1	p_2
Minhash	$\frac{s*n}{2M-s*n}$	$\frac{(1-\epsilon)s*n}{2M-(1-\epsilon)s*n}$	$\frac{s*n}{2M-\theta}$	$\frac{(1-\epsilon)s*n}{2M-(1-\epsilon)s*n}$
Simhash	$1 - \frac{\arccos(\frac{s*n}{M})}{\pi}$	$1 - \frac{\arccos\left(\frac{(1-\epsilon)s*n}{M}\right)}{\pi}$	$1 - \frac{\arccos(\frac{s*n}{M})}{\pi}$	$1 - \frac{\arccos\left(\frac{(1-\epsilon)s*n}{M}\right)}{\pi}$

Table 4.1: LSH Functions for Support

These LSH family of hash functions will be incorporated into standard Apriori algorithm to approximate its candidates generation step, so that every candidate in the candidates list will satisfy the support requirement with high probability.

Apriori Algorithm :

Apriori algorithm is one of the famous algorithms to solve frequent itemset mining problem. It works level by level such that at each level l it takes all frequent itemsets from the previous level $l-1$ and join each item with remaining all other items to form candidates for the next level. Once the candidates list C has been generated, all items in C will be checked for their support requirement.

Algorithm 1 Aprioiri Algorithm(D,s)

1: l = 1
2: $F = x | x\ is\ s-frequent\ in\ D$
3: $Output\ F$;
4: **while** $F \neq \phi$ **do**
5: $l = l + 1$;
6: **for** $I_a \in F$ **do**
7: **for** $I_b \in F$ **do**
8: add $I_a \cup I_b$ to C if $|I_a \cup I_b| = l$
9: **end for**
10: **end for**
11: $F = \phi$
12: **for** $I \in C$ **do**
13: *Add I to F if support of* $I \geq s$
14: **end for**
15: $Output\ F$;
16: **end while**

In this algorithm joining every item with remaining items will generate many candidates for the next level. Checking for the support requirement of these candidates need many I/O operations. Since locality sensitive hashing schemes estimate similarity of two objects, we explore the usage of LSH technique to join items together. This method avoid joining two items such that the union of these items has less support value than required threshold. In the next section we explore the usage of LSH in frequent itemset mining by approximating candidates generation step of Apriori algorithm.

4.3 LSH-Apriori

In the standard Apriori algorithm candidates are generated(steps from 6 to 10) by simply joining every item with other item at the same level. This way of generating candidates is a bottleneck because it adds every itemset to potential candidates set. If we estimate the support value of this itemset before adding it to potential candidate set it will avoid unnecessary support pruning. In this section we elaborate LSH-Apriori algorithm that implement this idea using locality sensitive hashing.

Algorithm 2 LSH-Apriori(D,s,ϵ,tp,fp)

```
l = 1
F = x|x is s − frequent in D
Output F;
while F ≠ φ do
    l = l + 1;
    for x ∈ F do
        x' ← Q(P(x))
        add x' to F_p
    end for
    for x ∈ F do
        x' ← P(Q(x))
        add x' to F_q
    end for
    Compute K, L values based on support, ϵ, tp, fp values
    for x ∈ F_p do
        for i = 1 to L do
            store x at H_i(x)
        end for
    end for
    F = φ
    for q ∈ F_q do
        cand ← φ
        for i = 1 to L do
            cand ← cand ∪ items at H_i(q)
        end for
        for x ∈ cand do
            I ← x ∪ q
            Add I to F if support of I ≥ s
        end for
    end for
    Output F;
end while
```

LSH-Apriori algorithm approximate the candidate generation step. It adds items to the potential candidates set C (with high probability) if the support value of the item is high. This approach prune items that have low support without reading the whole database to check its support value. There is one drawback in this approach it may not generate items that actually have support value greater than the required threshold. This can be avoided by taking more hash functions. Accuracy and time(required to find hash values of different data points) form a trade off, with high L value one can get high accuracy but it take more time to evaluate the hash functions. In the experimental

section we show the performance of LSH-Apriori on two real datasets.

4.4 Experimental Results

In this section we show the performance of LSH-Apriori algorithm on two real world datasets- BMS1 and BMS2. These two datasets are click-stream datasets of two different websites, each transaction in these datasets represent the sequence of pages that any user visited at an instance. The following table lists the characteristics of these datasets

Dataset	# Items	# Transactions	M	avg_nonzeros
BMS1	497	59603	3658	301
BMS2	3340	77513	3766	107

Table 4.2: Datasets for Frequent Itemset Mining

We use the following parameters in this section to evaluate the performance of LSH-Apriori with respect to traditional Apriori algorithm.

1. Candidate Ratio = $\frac{candidates\_of\_LSH-Apriori}{candidates\_of\_Apriori}$

2. Accuracy = $\frac{Items\_of\_LSH-Apriori}{Items\_of\_Apriori} * 100$

Candidate ratio varies between 0 and 1, accuracy varies between 0 and 100. Candidate ratio and accuracy form a tradeoff, the goal here is to get good accuracy with less candidate ratio. First we show the performance of LSH-Apriori where minhash, simhash are used as LSH families. Next, we improve these results with two level banding technique. With these experiments we can observe that two level banding technique achieves the better accuracy with less candidate ratio.

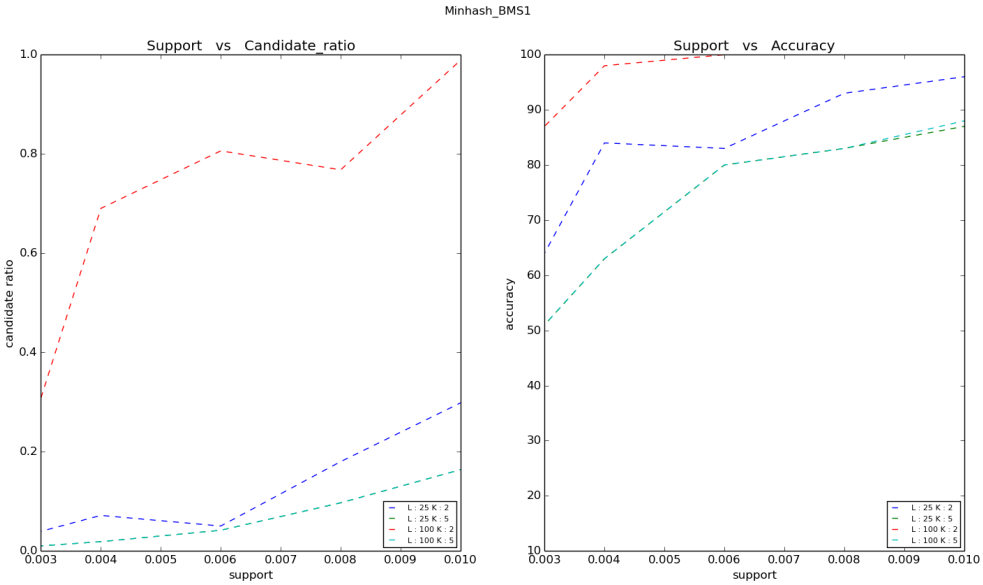

Figure 4.1: Minhash-Apriori performance on BMS1 dataset

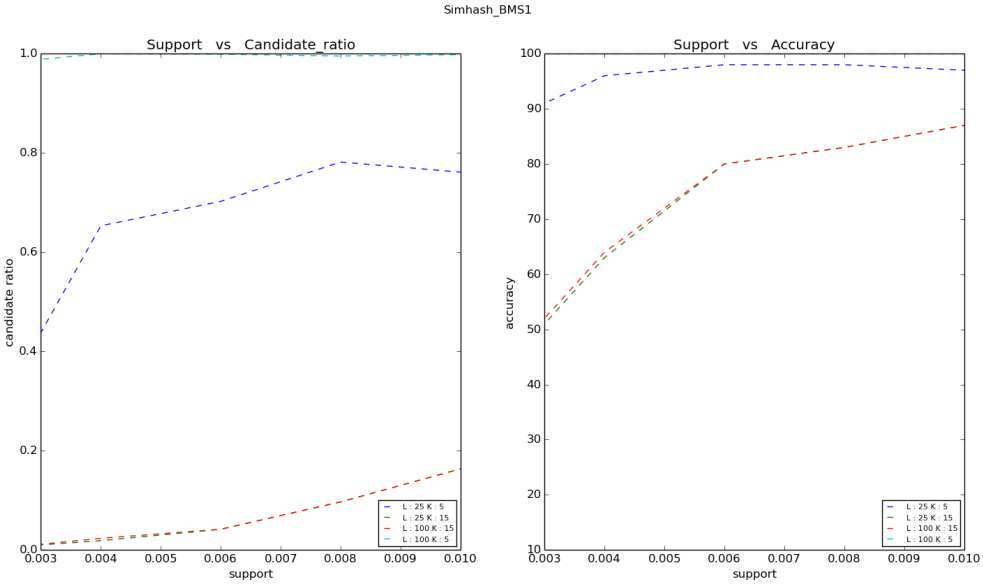

Figure 4.2: Simhash-Apriori performance on BMS1 dataset

Figure 4.1 and 4.2 shows the performance of Minhash and Simhash respectively. Minhhash gives accuracy ≥ 60 with candidate ratio ≤ 0.2 when the number of hash functions are 60 (L = 25, K = 2). Simhash gives the same performance when number of hash functions are 375 (L = 25, K = 15).

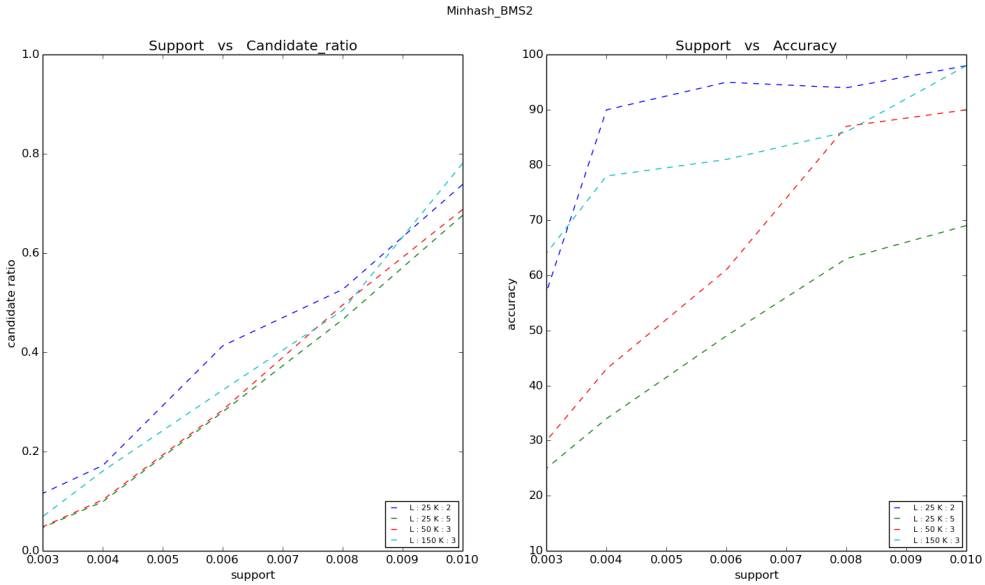

Figure 4.3: Minhash-Apriori performance on BMS2 dataset

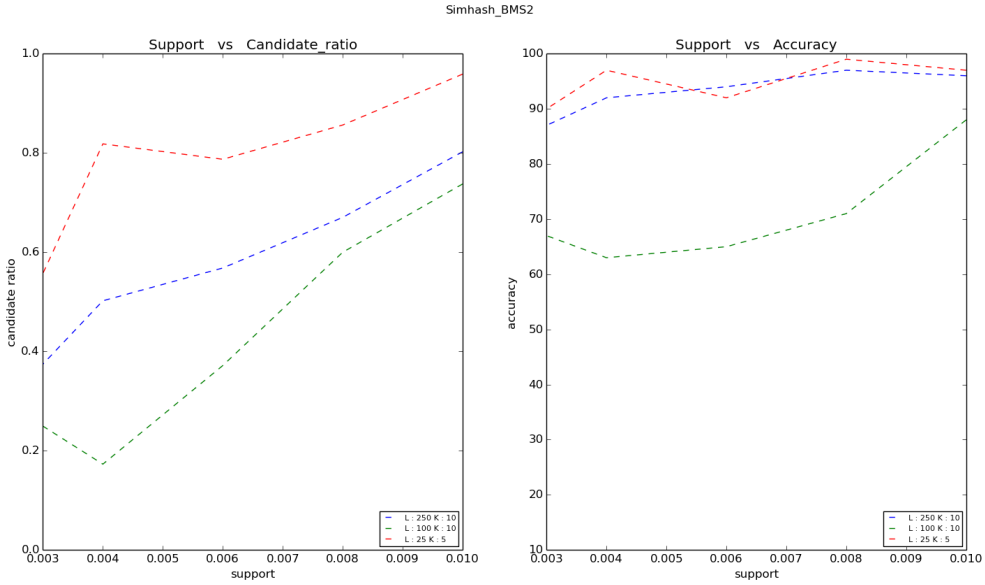

Figure 4.4: Simhash-Apriori performance on BMS2 dataset

Figure 4.3 and 4.4 show the performance of minhash and simhash on BMS2 dataset. We can see that minhash needs less number of hash functions to get the same performance compared to simhash.

Two Level Banding:

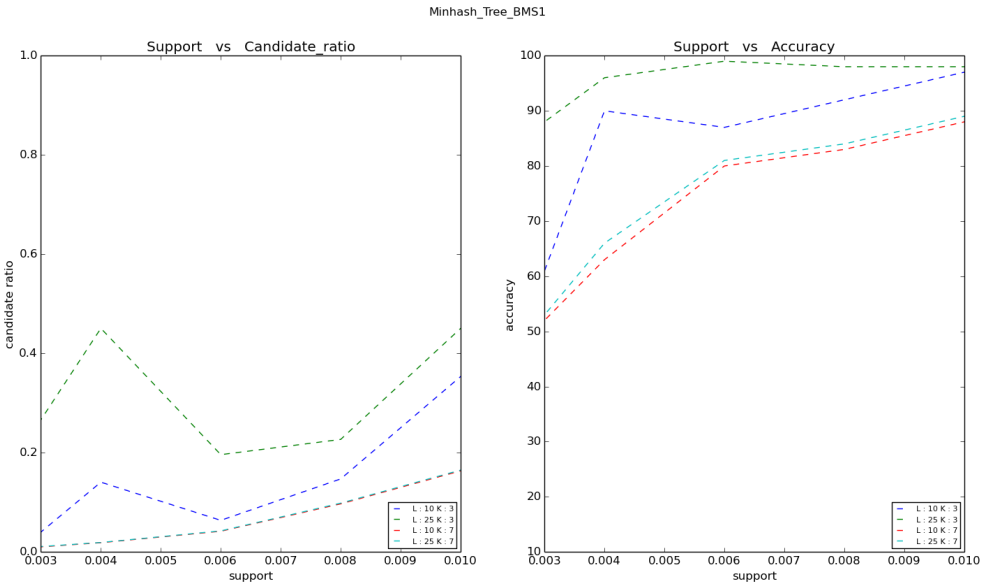

Figure 4.5: Minhash_Tree performance on BMS1 dataset

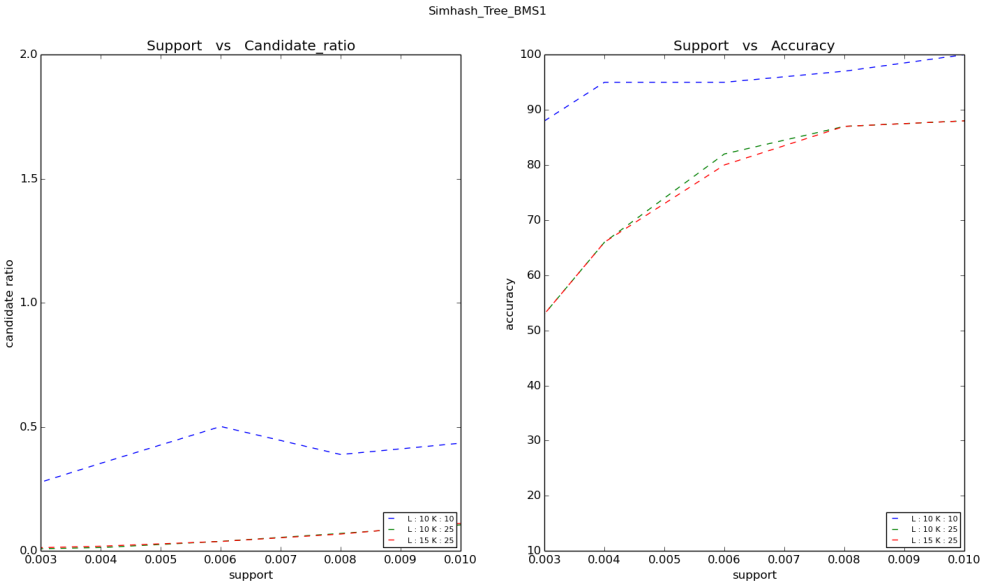

Figure 4.6: Simhash_Tree performance on BMS1 dataset

Figure 4.5 and 4.6 show the performance of two level banding on BMS1 dataset. We

took $K_1 = 2$ in the lower level. It is clear from these graphs that two level banding gives better accuracy with low candidate ratio. We can observe the same pattern on BMS2 dataset also.

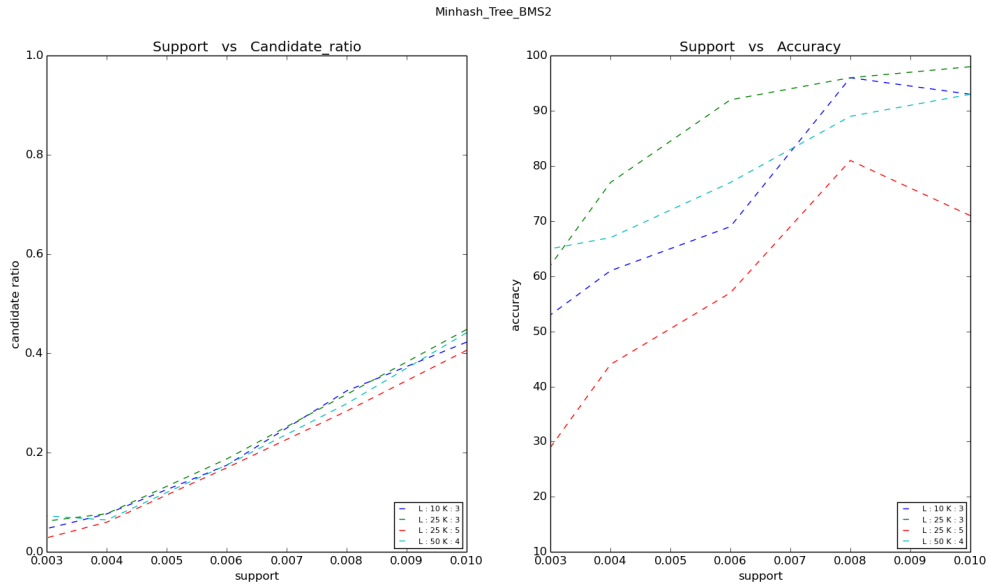

Figure 4.7: Minhash_Tree performance on BMS2 dataset

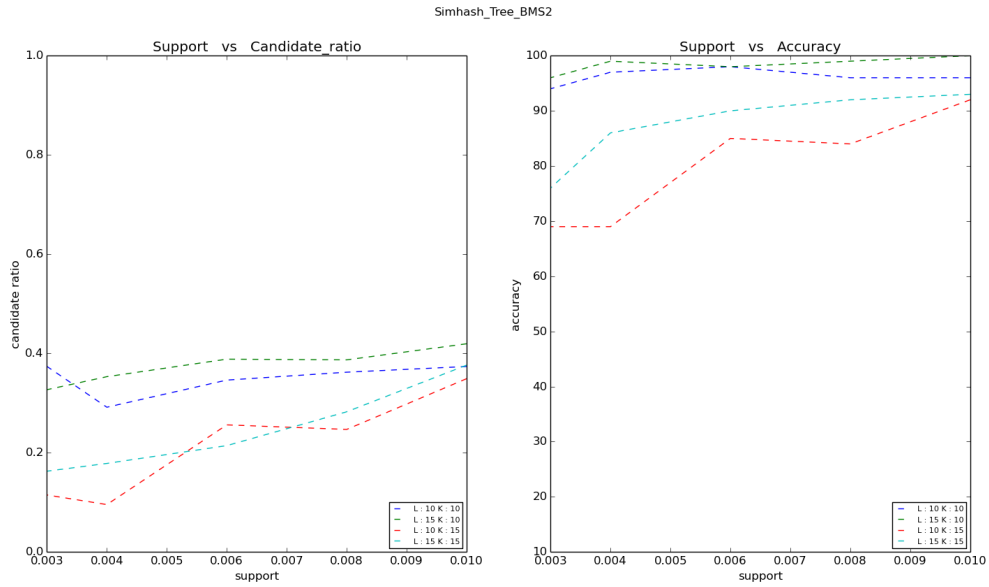

Figure 4.8: Simhash_Tree performance on BMS2 dataset

Figure 4.7 and 4.8 show the performance of two level banding on BMS2 dataset. We took K_1 = 2 in the lower level. Two level banding gives better results compared to single level banding.

4.5 Conclusion

From the experiment section we can observe the following patterns

1. We can improve the performance of LSH-Apriori using two level banding technique.
2. In single level banding, if K is increased with constant L value then candidate ratio will be decreased.
3. In single level banding, if L is increased with constant K value then accuracy will be increased.
4. LSH-Apriori performs better for optimal values of its parameters.

BIBLIOGRAPHY

[1] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," *STOC*, pp. 601–613, 1998.

[2] A. Z. Broder, "On the resemblance and containment of documents," *the Compression and Complexity of Sequences*, p. 21,Äì29, 1997.

[3] M. S. Charikar, "Similarity estimation techniques from rounding algorithms.," *In Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388, (2002, May).

[4] P. I. M. Datar, N. Immorlica and V. S. Mirrokn, "Locality-sensitive hashing scheme based on p-stable distributions," *SCG*, p. 253 ,Äì 262, 2004.

[5] A. Shrivastava and P. Li, "Asymmetric minwise hashing for indexing binary inner products and set containment," *WWW*, 2015.

[6] A. Shrivastava and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)," *NIPS*, 2014.

[7] B. Neyshabur and N. Srebro., "On symmetric and asymmetric lshs for inner product search.," *arXiv preprint arXiv:1410.5518*, (2014).

[8] S. B. C. Silverstein and R. Motwani., "Beyond market baskets:generalizing association rules to dependence rules.," *Data Min. Knowl.Discov.*, p. 2(1):3968, 1998.

[9] H. T. H. Mannila and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Min. Knowl. Discov*, p. 1(3):259289, 1997.

[10] J. Y. H. Wang, W. Wang and P. S. Yu, "Clustering by pattern similarity in large data sets," *In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin*, June 3-6, 2002.

[11] H. M. S. S. H. T. D. Gunopulos, R. Khardon and R. S.Sharma, "Discovering all most specific sentences. acm trans. database syst," p. 28(2):140174, 2003.

[12] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," *In Proceedings of 20th International Conference on Very Large Data Bases*, September 12-15, 1994, Santiago de Chile, Chile.

[13] M. C. J. S. Park and P. S. Yu, "An effective hash based algorithm for mining association rules," *In Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, California*, May 22-25,1995.

[14] S. F. A. G. P. I. R. M. J. D. U. E. Cohen, M. Datar and C. Yang, "Finding interesting associations without support pruning," *IEEE Trans. Knowl. Data Eng*, 2001.