One Permutation Hashing

Ping Li
Department of Statistical Science
Cornell University

Art B Owen
Department of Statistics
Stanford University

Cun-Hui Zhang
Department of Statistics
Rutgers University

Abstract

Minwise hashing is a standard procedure in the context of search, for efficiently estimating set similarities in massive binary data such as text. Recently, b-bit minwise hashing has been applied to large-scale learning and sublinear time near-neighbor search. The major drawback of minwise hashing is the expensive preprocessing, as the method requires applying (e.g.,) $k = 200$ to 500 permutations on the data. This paper presents a simple solution called *one permutation hashing*. Conceptually, given a binary data matrix, we permute the columns once and divide the permuted columns evenly into k bins; and we store, for each data vector, the smallest nonzero location in each bin. The probability analysis illustrates that this one permutation scheme should perform similarly to the original (k-permutation) minwise hashing. Our experiments with training SVM and logistic regression confirm that one permutation hashing can achieve similar (or even better) accuracies compared to the k-permutation scheme. *See more details in arXiv:1208.1259.*

1 Introduction

Minwise hashing [4, 3] is a standard technique in the context of search, for efficiently computing set similarities. Recently, b-bit minwise hashing [18, 19], which stores only the lowest b bits of each hashed value, has been applied to sublinear time near neighbor search [22] and learning [16], on large-scale high-dimensional binary data (e.g., text). A drawback of minwise hashing is that it requires a costly preprocessing step, for conducting (e.g.,) $k = 200 \sim 500$ permutations on the data.

1.1 Massive High-Dimensional Binary Data

In the context of search, text data are often processed to be binary in extremely high dimensions. A standard procedure is to represent documents (e.g., Web pages) using w-shingles (i.e., w contiguous words), where $w \geq 5$ in several studies [4, 8]. This means the size of the dictionary needs to be substantially increased, from (e.g.,) 10^5 common English words to 10^{5w} "super-words". In current practice, it appears sufficient to set the total dimensionality to be $D = 2^{64}$, for convenience. Text data generated by w-shingles are often treated as binary. The concept of shingling can be naturally extended to Computer Vision, either at pixel level (for aligned images) or at Visual feature level [23].

In machine learning practice, the use of extremely high-dimensional data has become common. For example, [24] discusses training datasets with (on average) $n = 10^{11}$ items and $D = 10^9$ distinct features. [25] experimented with a dataset of potentially $D = 16$ trillion (1.6×10^{13}) unique features.

1.2 Minwise Hashing and b-Bit Minwise Hashing

Minwise hashing was mainly designed for binary data. A binary (0/1) data vector can be viewed as a set (locations of the nonzeros). Consider sets $S_i \subseteq \Omega = \{0, 1, 2, ..., D-1\}$, where D, the size of the space, is often set as $D = 2^{64}$ in industrial applications. The similarity between two sets, S_1 and S_2, is commonly measured by the *resemblance*, which is a version of the normalized inner product:

$$R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{a}{f_1 + f_2 - a}, \qquad \text{where } f_1 = |S_1|,\ f_2 = |S_2|,\ a = |S_1 \cap S_2| \tag{1}$$

For large-scale applications, the cost of computing resemblances exactly can be prohibitive in time, space, and energy-consumption. The minwise hashing method was proposed for efficient computing resemblances. The method requires applying k independent random permutations on the data.

Denote π a random permutation: $\pi : \Omega \to \Omega$. The hashed values are the two minimums of $\pi(S_1)$ and $\pi(S_2)$. The probability at which the two hashed values are equal is

$$\mathbf{Pr}\left(\min(\pi(S_1)) = \min(\pi(S_2))\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R \tag{2}$$

One can then estimate R from k independent permutations, π_1, ..., π_k:

$$\hat{R}_M = \frac{1}{k}\sum_{j=1}^{k} 1\{\min(\pi_j(S_1)) = \min(\pi_j(S_2))\}, \qquad \text{Var}\left(\hat{R}_M\right) = \frac{1}{k}R(1-R) \tag{3}$$

Because the indicator function $1\{\min(\pi_j(S_1)) = \min(\pi_j(S_2))\}$ can be written as an inner product between two binary vectors (each having only one 1) in D dimensions [16]:

$$1\{\min(\pi_j(S_1)) = \min(\pi_j(S_2))\} = \sum_{i=0}^{D-1} 1\{\min(\pi_j(S_1)) = i\} \times 1\{\min(\pi_j(S_2)) = i\} \tag{4}$$

we know that minwise hashing can be potentially used for training linear SVM and logistic regression on high-dimensional binary data by converting the permuted data into a new data matrix in $D \times k$ dimensions. This of course would not be realistic if $D = 2^{64}$.

The method of b-bit minwise hashing [18, 19] provides a simple solution by storing only the lowest b bits of each hashed data, reducing the dimensionality of the (expanded) hashed data matrix to just $2^b \times k$. [16] applied this idea to large-scale learning on the *webspam* dataset and demonstrated that using $b = 8$ and $k = 200$ to 500 could achieve very similar accuracies as using the original data.

1.3 The Cost of Preprocessing and Testing

Clearly, the preprocessing of minwise hashing can be very costly. In our experiments, loading the *webspam* dataset (350,000 samples, about 16 million features, and about 24GB in Libsvm/svmlight (text) format) used in [16] took about 1000 seconds when the data were stored in text format, and took about 150 seconds after we converted the data into binary. In contrast, the preprocessing cost for $k = 500$ was about 6000 seconds. Note that, compared to industrial applications [24], the *webspam* dataset is very small. For larger datasets, the preprocessing step will be much more expensive.

In the testing phrase (in search or learning), if a new data point (e.g., a new document or a new image) has not been processed, then the total cost will be expensive if it includes the preprocessing. This may raise significant issues in user-facing applications where the testing efficiency is crucial.

Intuitively, the standard practice of minwise hashing ought to be very "wasteful" in that all the nonzero elements in one set are scanned (permuted) but only the smallest one will be used.

1.4 Our Proposal: One Permutation Hashing

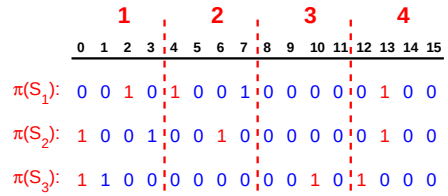

Figure 1: Consider $S_1, S_2, S_3 \subseteq \Omega = \{0, 1, ..., 15\}$ (i.e., $D = 16$). We apply one permutation π on the sets and present $\pi(S_1)$, $\pi(S_2)$, and $\pi(S_3)$ as binary (0/1) vectors, where $\pi(S_1) = \{2, 4, 7, 13\}$, $\pi(S_2) = \{0, 6, 13\}$, and $\pi(S_3) = \{0, 1, 10, 12\}$. We divide the space Ω evenly into $k = 4$ bins, select the smallest nonzero in each bin, and **re-index** the selected elements as: $[2, 0, *, 1]$, $[0, 2, *, 1]$, and $[0, *, 2, 0]$. For now, we use '*' for empty bins, which occur rarely unless the number of nonzeros is small compared to k.

As illustrated in Figure 1, the idea of *one permutation hashing* is simple. We view sets as 0/1 vectors in D dimensions so that we can treat a collection of sets as a binary data matrix in D dimensions. After we permute the columns (features) of the data matrix, we divide the columns evenly into k parts (bins) and we simply take, for each data vector, the smallest nonzero element in each bin.

In the example in Figure 1 (which concerns 3 sets), the sample selected from $\pi(S_1)$ is $[2, 4, *, 13]$, where we use '*' to denote an empty bin, for the time being. Since only want to compare elements with the same bin number (so that we can obtain an inner product), we can actually re-index the elements of each bin to use the smallest possible representations. For example, for $\pi(S_1)$, after re-indexing, the sample $[2, 4, *, 13]$ becomes $[2 - 4 \times 0, 4 - 4 \times 1, *, 13 - 4 \times 3] = [2, 0, *, 1]$.

We will show that empty bins occur rarely unless the total number of nonzeros for some set is small compared to k, and we will present strategies on how to deal with empty bins should they occur.

1.5 Advantages of One Permutation Hashing

Reducing k (e.g., 500) permutations to just one permutation (or a few) is much more computationally efficient. From the perspective of energy consumption, this scheme is desirable, especially considering that minwise hashing is deployed in the search industry. Parallel solutions (e.g., GPU [17]), which require additional hardware and software implementation, will not be energy-efficient.

In the testing phase, if a new data point (e.g., a new document or a new image) has to be first processed with k permutations, then the testing performance may not meet the demand in, for example, user-facing applications such as search or interactive visual analytics.

One permutation hashing should be easier to implement, from the perspective of random number generation. For example, if a dataset has one billion features ($D = 10^9$), we can simply generate a "permutation vector" of length $D = 10^9$, the memory cost of which (i.e., 4GB) is not significant. On the other hand, it would not be realistic to store a "permutation matrix" of size $D \times k$ if $D = 10^9$ and $k = 500$; instead, one usually has to resort to approximations such as universal hashing [5]. Universal hashing often works well in practice although theoretically there are always worst cases.

One permutation hashing is a better matrix sparsification scheme. In terms of the original binary data matrix, the one permutation scheme simply makes many nonzero entries be zero, without further "damaging" the matrix. Using the k-permutation scheme, we store, for each permutation and each row, only the first nonzero and make all the other nonzero entries be zero; and then we have to concatenate k such data matrices. This significantly changes the structure of the original data matrix.

1.6 Related Work

One of the authors worked on another "one permutation" scheme named *Conditional Random Sampling (CRS)* [13, 14] since 2005. Basically, CRS continuously takes the bottom-k nonzeros after applying one permutation on the data, then it uses a simple "trick" to construct a random sample for each pair with the effective sample size determined at the estimation stage. By taking the nonzeros continuously, however, the samples are no longer "aligned" and hence we can not write the estimator as an inner product in a unified fashion. [16] commented that using CRS for linear learning does not produce as good results compared to using b-bit minwise hashing. Interestingly, in the original "minwise hashing" paper [4] (we use quotes because the scheme was not called "minwise hashing" at that time), only one permutation was used and a sample was the first k nonzeros after the permutation. Then they quickly moved to the k-permutation minwise hashing scheme [3].

We are also inspired by the work on *very sparse random projections* [15] and *very sparse stable random projections* [12]. The regular random projection method also has the expensive preprocessing cost as it needs a large number of projections. [15, 12] showed that one can substantially reduce the preprocessing cost by using an extremely sparse projection matrix. The preprocessing cost of very sparse random projections can be as small as merely doing one projection. See `www.stanford.edu/group/mmds/slides2012/s-pli.pdf` for the experimental results on clustering/classification/regression using very sparse random projections.

This paper focuses on the "fixed-length" scheme as shown in Figure 1. The technical report (arXiv:1208.1259) also describes a "variable-length" scheme. Two schemes are more or less equivalent, although the fixed-length scheme is more convenient to implement (and it is slightly more accurate). The variable-length hashing scheme is to some extent related to the Count-Min (CM) sketch [6] and the Vowpal Wabbit (VW) [21, 25] hashing algorithms.

2 Applications of Minwise Hashing on Efficient Search and Learning

In this section, we will briefly review two important applications of the k-permutation b-bit minwise hashing: (i) sublinear time near neighbor search [22], and (ii) large-scale linear learning [16].

2.1 Sublinear Time Near Neighbor Search

The task of *near neighbor search* is to identify a set of data points which are "most similar" to a query data point. Developing efficient algorithms for near neighbor search has been an active research topic since the early days of modern computing (e.g, [9]). In current practice, methods for approximate near neighbor search often fall into the general framework of *Locality Sensitive Hashing (LSH)* [10, 1]. The performance of LSH largely depends on its underlying implementation. The idea in [22] is to directly use the bits from b-bit minwise hashing to construct hash tables.

Specifically, we hash the data points using k random permutations and store each hash value using b bits. For each data point, we concatenate the resultant $B = bk$ bits as a *signature* (e.g., $bk = 16$). This way, we create a table of 2^B buckets and each bucket stores the pointers of the data points whose signatures match the bucket number. In the testing phrase, we apply the same k permutations to a query data point to generate a bk-bit signature and only search data points in the corresponding bucket. Since using only one table will likely miss many true near neighbors, as a remedy, we independently generate L tables. The query result is the union of data points retrieved in L tables.

Index	Data Points
00 00	6, 110, 143
00 01	3, 38, 217
00 10	(empty)
11 01	5, 14, 206
11 10	31, 74, 153
11 11	21, 142, 329

Index	Data Points
00 00	8, 159, 331
00 01	11, 25, 99
00 10	3, 14, 32, 97
11 01	7, 49, 208
11 10	33, 489
11 11	6,15, 26, 79

Figure 2: An example of hash tables, with $b = 2$, $k = 2$, and $L = 2$.

Figure 2 provides an example with $b = 2$ bits, $k = 2$ permutations, and $L = 2$ tables. The size of each hash table is 2^4. Given n data points, we apply $k = 2$ permutations and store $b = 2$ bits of each hashed value to generate n (4-bit) signatures L times. Consider data point 6. For Table 1 (left panel of Figure 2), the lowest b-bits of its two hashed values are 00 and 00 and thus its signature is 0000 in binary; hence we place a pointer to data point 6 in bucket number 0. For Table 2 (right panel of Figure 2), we apply another $k = 2$ permutations. This time, the signature of data point 6 becomes 1111 in binary and hence we place it in the last bucket. Suppose in the testing phrase, the two (4-bit) signatures of a new data point are 0000 and 1111, respectively. We then only search for the near neighbors in the set $\{6, 15, 26, 79, 110, 143\}$, instead of the original set of n data points.

2.2 Large-Scale Linear Learning

The recent development of highly efficient linear learning algorithms is a major breakthrough. Popular packages include SVM$^{\text{perf}}$ [11], Pegasos [20], Bottou's SGD SVM [2], and LIBLINEAR [7].

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 logistic regression solves the following optimization problem (where $C > 0$ is the regularization parameter):

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathbf{T}\mathbf{w} + C\sum_{i=1}^{n} \log\left(1 + e^{-y_i \mathbf{w}^\mathbf{T}\mathbf{x_i}}\right), \tag{5}$$

The L_2-regularized linear SVM solves a similar problem:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathbf{T}\mathbf{w} + C\sum_{i=1}^{n} \max\left\{1 - y_i \mathbf{w}^\mathbf{T}\mathbf{x_i},\ 0\right\}, \tag{6}$$

In [16], they apply k random permutations on each (binary) feature vector $\mathbf{x}_i$ and store the lowest b bits of each hashed value, to obtain a new dataset which can be stored using merely nbk bits. At run-time, each new data point has to be expanded into a $2^b \times k$-length vector with exactly k 1's.

To illustrate this simple procedure, [16] provided a toy example with $k = 3$ permutations. Suppose for one data vector, the hashed values are $\{12013,\ 25964,\ 20191\}$, whose binary digits are respectively $\{010111011101101,\ 110010101101100, 100111011011111\}$. Using $b = 2$ bits, the binary digits are stored as $\{01, 00, 11\}$ (which corresponds to $\{1, 0, 3\}$ in decimals). At run-time, the (b-bit) hashed data are expanded into a new feature vector of length $2^b k = 12$: $\{0, 0, 1, 0,\ \ 0, 0, 0, 1,\ \ 1, 0, 0, 0\}$. The same procedure is then applied to all n feature vectors.

Clearly, in both applications (near neighbor search and linear learning), the hashed data have to be "aligned" in that only the hashed data generated from the same permutation are interacted. Note that, with our one permutation scheme as in Figure 1, the hashed data are indeed aligned.

3 Theoretical Analysis of the One Permutation Scheme

This section presents the probability analysis to provide a rigorous foundation for one permutation hashing as illustrated in Figure 1. Consider two sets S_1 and S_2. We first introduce two definitions,

for the number of "jointly empty bins" and the number of "matched bins," respectively:

$$N_{emp} = \sum_{j=1}^{k} I_{emp,j}, \qquad N_{mat} = \sum_{j=1}^{k} I_{mat,j} \tag{7}$$

where $I_{emp,j}$ and $I_{mat,j}$ are defined for the j-th bin, as

$$I_{emp,j} = \begin{cases} 1 & \text{if both } \pi(S_1) \text{ and } \pi(S_2) \text{ are empty in the } j\text{-th bin} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$I_{mat,j} = \begin{cases} 1 & \text{if both } \pi(S_1) \text{ and } \pi(S_1) \text{ are not empty and the smallest element of } \pi(S_1) \\ & \text{matches the smallest element of } \pi(S_2), \text{ in the } j\text{-th bin} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Recall the notation: $f_1 = |S_1|,\ f_2 = |S_2|,\ a = |S_1 \cap S_2|$. We also use $f = |S_1 \cup S_2| = f_1 + f_2 - a$.

Lemma 1

$$\mathbf{Pr}\left(N_{emp} = j\right) = \sum_{s=0}^{k-j} (-1)^s \frac{k!}{j!s!(k-j-s)!} \prod_{t=0}^{f-1} \frac{D\left(1 - \frac{j+s}{k}\right) - t}{D - t},\ \ 0 \le j \le k-1 \tag{10}$$

Assume $D\left(1 - \frac{1}{k}\right) \ge f = f_1 + f_2 - a$.

$$\frac{E\left(N_{emp}\right)}{k} = \prod_{j=0}^{f-1} \frac{D\left(1 - \frac{1}{k}\right) - j}{D - j} \le \left(1 - \frac{1}{k}\right)^f \tag{11}$$

$$\frac{E\left(N_{mat}\right)}{k} = R\left(1 - \frac{E\left(N_{emp}\right)}{k}\right) = R\left(1 - \prod_{j=0}^{f-1} \frac{D\left(1 - \frac{1}{k}\right) - j}{D - j}\right) \tag{12}$$

$$Cov\left(N_{mat},\ N_{emp}\right) \le 0 \qquad \square \tag{13}$$

In practical scenarios, the data are often sparse, i.e., $f = f_1 + f_2 - a \ll D$. In this case, the upper bound (11) $\left(1 - \frac{1}{k}\right)^f$ is a good approximation to the true value of $\frac{E(N_{emp})}{k}$. Since $\left(1 - \frac{1}{k}\right)^f \approx e^{-f/k}$, we know that the chance of empty bins is small when $f \gg k$. For example, if $f/k = 5$ then $\left(1 - \frac{1}{k}\right)^f \approx 0.0067$. For practical applications, we would expect that $f \gg k$ (for most data pairs), otherwise hashing probably would not be too useful anyway. This is why we do not expect empty bins will significantly impact (if at all) the performance in practical settings.

Lemma 2 shows the following estimator $\hat{R}_{mat}$ of the resemblance is unbiased:

Lemma 2

$$\hat{R}_{mat} = \frac{N_{mat}}{k - N_{emp}}, \qquad E\left(\hat{R}_{mat}\right) = R \tag{14}$$

$$Var\left(\hat{R}_{mat}\right) = R(1-R)\left(E\left(\frac{1}{k - N_{emp}}\right)\left(1 + \frac{1}{f-1}\right) - \frac{1}{f-1}\right) \tag{15}$$

$$E\left(\frac{1}{k - N_{emp}}\right) = \sum_{j=0}^{k-1} \frac{\mathbf{Pr}\left(N_{emp} = j\right)}{k - j} \ge \frac{1}{k - E(N_{emp})} \qquad \square \tag{16}$$

The fact that $E\left(\hat{R}_{mat}\right) = R$ may seem surprising as in general ratio estimators are not unbiased. Note that $k - N_{emp} > 0$, because we assume the original data vectors are not completely empty (all-zero). As expected, when $k \ll f = f_1 + f_2 - a$, N_{emp} is essentially zero and hence $Var\left(\hat{R}_{mat}\right) \approx \frac{R(1-R)}{k}$. In fact, $Var\left(\hat{R}_{mat}\right)$ is a bit smaller than $\frac{R(1-R)}{k}$, especially for large k.

It is probably not surprising that our one permutation scheme (slightly) outperforms the original k-permutation scheme (at merely $1/k$ of the preprocessing cost), because one permutation hashing, which is "sampling-without-replacement", provides a better strategy for matrix sparsification.

4 Strategies for Dealing with Empty Bins

In general, we expect that empty bins should not occur often because $E(N_{emp})/k \approx e^{-f/k}$, which is very close to zero if $f/k > 5$. (Recall $f = |S_1 \cup S_2|$.) If the goal of using minwise hashing is for data reduction, i.e., reducing the number of nonzeros, then we would expect that $f \gg k$ anyway.

Nevertheless, in applications where we need the estimators to be inner products, we need strategies to deal with empty bins in case they occur. Fortunately, we realize a (in retrospect) simple strategy which can be nicely integrated with linear learning algorithms and performs well.

Figure 3 plots the histogram of the numbers of nonzeros in the *webspam* dataset, which has 350,000 samples. The average number of nonzeros is about 4000 which should be much larger than k (e.g., 500) for the hashing procedure. On the other hand, about 10% (or 2.8%) of the samples have < 500 (or < 200) nonzeros. Thus, we must deal with empty bins if we do not want to exclude those data points. For example, if $f = k = 500$, then $N_{emp} \approx e^{-f/k} = 0.3679$, which is not small.

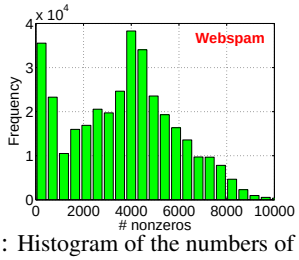

Figure 3: Histogram of the numbers of nonzeros in the *webspam* dataset (350,000 samples).

The strategy we recommend for linear learning is **zero coding**, which is tightly coupled with the strategy of hashed data expansion [16] as reviewed in Sec. 2.2. More details will be elaborated in Sec. 4.2. Basically, we can encode "*" as "zero" in the expanded space, which means N_{mat} will remain the same (after taking the inner product in the expanded space). This strategy, which is **sparsity-preserving**, essentially corresponds to the following modified estimator:

$$\hat{R}^{(0)}_{mat} = \frac{N_{mat}}{\sqrt{k - N^{(1)}_{emp}}\sqrt{k - N^{(2)}_{emp}}} \tag{17}$$

where $N^{(1)}_{emp} = \sum_{j=1}^{k} I^{(1)}_{emp,j}$ and $N^{(2)}_{emp} = \sum_{j=1}^{k} I^{(2)}_{emp,j}$ are the numbers of empty bins in $\pi(S_1)$ and $\pi(S_2)$, respectively. This modified estimator makes sense for a number of reasons.

Basically, since each data vector is processed and coded separately, we actually do not know N_{emp} (the number of *jointly* empty bins) until we see both $\pi(S_1)$ and $\pi(S_2)$. In other words, we can not really compute N_{emp} if we want to use linear estimators. On the other hand, $N^{(1)}_{emp}$ and $N^{(2)}_{emp}$ are always available. In fact, the use of $\sqrt{k - N^{(1)}_{emp}}\sqrt{k - N^{(2)}_{emp}}$ in the denominator corresponds to the normalizing step which is needed before feeding the data to a solver for SVM or logistic regression.

When $N^{(1)}_{emp} = N^{(2)}_{emp} = N_{emp}$, (17) is equivalent to the original $\hat{R}_{mat}$. When two original vectors are very similar (e.g., large R), $N^{(1)}_{emp}$ and $N^{(2)}_{emp}$ will be close to N_{emp}. When two sets are highly unbalanced, using (17) will overestimate R; however, in this case, N_{mat} will be so small that the absolute error will not be large.

4.1 The m-Permutation Scheme with $1 < m \ll k$

If one would like to further (significantly) reduce the chance of the occurrences of empty bins, here we shall mention that one does not really have to strictly follow "one permutation," since one can always conduct m permutations with $k' = k/m$ and concatenate the hashed data. Once the preprocessing is no longer the bottleneck, it matters less whether we use 1 permutation or (e.g.,) $m = 3$ permutations. The chance of having empty bins decreases exponentially with increasing m.

4.2 An Example of The "Zero Coding" Strategy for Linear Learning

Sec. 2.2 reviewed the data-expansion strategy used by [16] for integrating b-bit minwise hashing with linear learning. We will adopt a similar strategy with modifications for considering empty bins.

We use a similar example as in Sec. 2.2. Suppose we apply our one permutation hashing scheme and use $k = 4$ bins. For the first data vector, the hashed values are $[12013,\ 25964,\ 20191,\ *]$ (i.e., the 4-th bin is empty). Suppose again we use $b = 2$ bits. With the "zero coding" strategy, our procedure

is summarized as follows:

Original hashed values ($k = 4$) :	12013	25964	20191	$*$
Original binary representations :	010111011101101	110010101101100	100111011011111	$*$
Lowest $b = 2$ binary digits :	01	00	11	$*$
Expanded $2^b = 4$ binary digits :	0010	0001	1000	0000

New feature vector fed to a solver : $\frac{1}{\sqrt{4-1}} \times [0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]$

We apply the same procedure to all feature vectors in the data matrix to generate a new data matrix. The normalization factor $\frac{1}{\sqrt{k - N_{emp}^{(i)}}}$ varies, depending on the number of empty bins in the i-th vector.

5 Experimental Results on the Webspam Dataset

The *webspam* dataset has 350,000 samples and 16,609,143 features. Each feature vector has on average about 4000 nonzeros; see Figure 3. Following [16], we use 80% of samples for training and the remaining 20% for testing. We conduct extensive experiments on linear SVM and logistic regression, using our proposed one permutation hashing scheme with $k \in \{2^6, 2^7, 2^8, 2^9\}$ and $b \in \{1, 2, 4, 6, 8\}$. For convenience, we use $D = 2^{24} = 16,777,216$, which is divisible by k.

There is one regularization parameter C in linear SVM and logistic regression. Since our purpose is to demonstrate the effectiveness of our proposed hashing scheme, we simply provide the results for a wide range of C values and assume that the best performance is achievable if we conduct cross-validations. This way, interested readers may be able to easily reproduce our experiments.

Figure 4 presents the test accuracies for both linear SVM (upper panels) and logistic regression (bottom panels). Clearly, when $k = 512$ (or even 256) and $b = 8$, b-bit one permutation hashing achieves similar test accuracies as using the original data. Also, compared to the original k-permutation scheme as in [16], our one permutation scheme achieves similar (or even slightly better) accuracies.

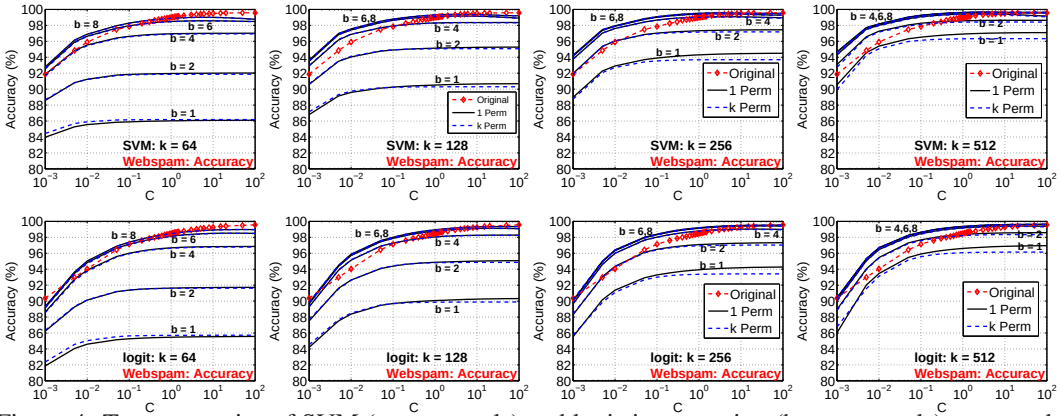

Figure 4: Test accuracies of SVM (upper panels) and logistic regression (bottom panels), averaged over 50 repetitions. The accuracies of using the original data are plotted as dashed (red, if color is available) curves with "diamond" markers. C is the regularization parameter. Compared with the original k-permutation minwise hashing (dashed and blue if color is available), the one permutation hashing scheme achieves similar accuracies, or even slightly better accuracies when k is large.

The empirical results on the *webspam* datasets are encouraging because they verify that our proposed one permutation hashing scheme performs as well as (or even slightly better than) the original k-permutation scheme, at merely $1/k$ of the original preprocessing cost. On the other hand, it would be more interesting, from the perspective of testing the robustness of our algorithm, to conduct experiments on a dataset (e.g., *news20*) where the empty bins will occur much more frequently.

6 Experimental Results on the News20 Dataset

The *news20* dataset (with 20,000 samples and 1,355,191 features) is a very small dataset in not-too-high dimensions. The average number of nonzeros per feature vector is about 500, which is also small. Therefore, this is more like a contrived example and we use it just to verify that our one permutation scheme (with the zero coding strategy) still works very well even when we let k be

as large as 4096 (i.e., most of the bins are empty). In fact, the one permutation schemes achieves noticeably better accuracies than the original k-permutation scheme. We believe this is because the one permutation scheme is "sample-without-replacement" and provides a better matrix sparsification strategy without "contaminating" the original data matrix too much.

We experiment with $k \in \{2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}\}$ and $b \in \{1, 2, 4, 6, 8\}$, for both one permutation scheme and k-permutation scheme. We use 10,000 samples for training and the other 10,000 samples for testing. For convenience, we let $D = 2^{21}$ (which is larger than 1,355,191).

Figure 5 and Figure 6 present the test accuracies for linear SVM and logistic regression, respectively. When k is small (e.g., $k \leq 64$) both the one permutation scheme and the original k-permutation scheme perform similarly. For larger k values (especially as $k \geq 256$), however, our one permutation scheme noticeably outperforms the k-permutation scheme. Using the original data, the test accuracies are about 98%. Our one permutation scheme with $k \geq 512$ and $b = 8$ essentially achieves the original test accuracies, while the k-permutation scheme could only reach about 97.5%.

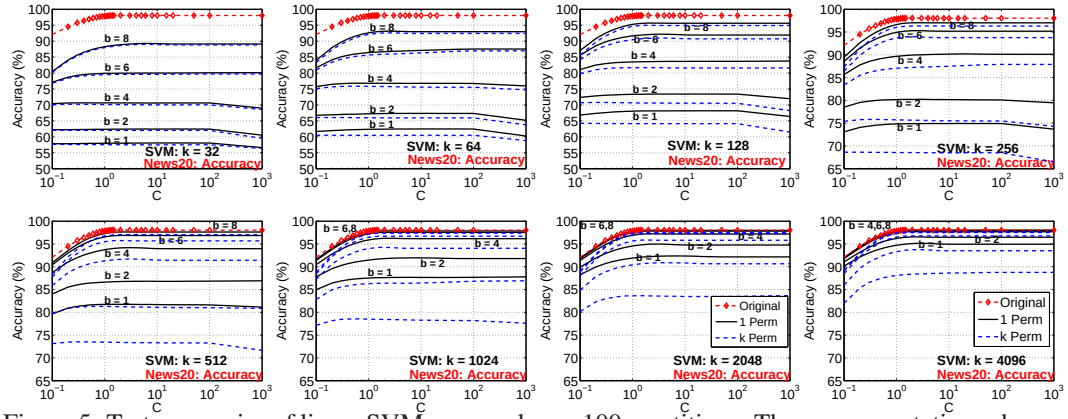

Figure 5: Test accuracies of linear SVM averaged over 100 repetitions. The one permutation scheme noticeably outperforms the original k-permutation scheme especially when k is not small.

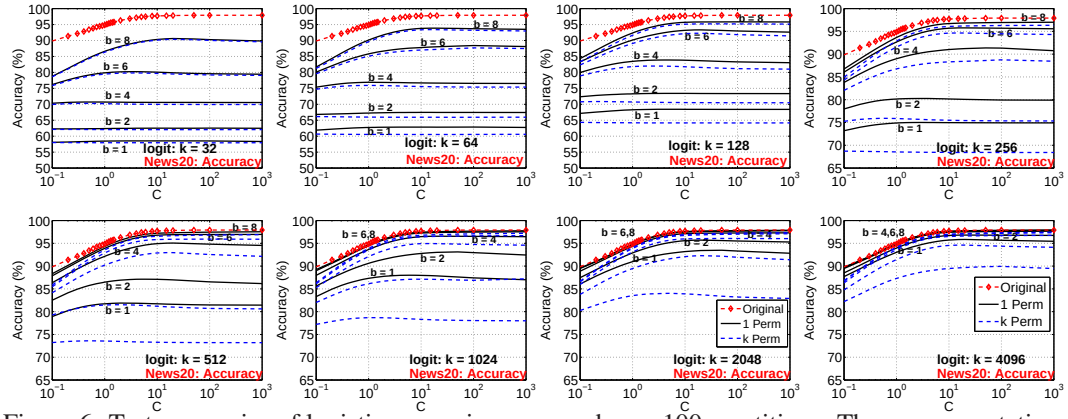

Figure 6: Test accuracies of logistic regression averaged over 100 repetitions. The one permutation scheme noticeably outperforms the original k-permutation scheme especially when k is not small.

7 Conclusion

A new hashing algorithm is developed for large-scale search and learning in massive binary data. Compared with the original k-permutation (e.g., $k = 500$) minwise hashing (which is a standard procedure in the context of search), our method requires only one permutation and can achieve similar or even better accuracies at merely $1/k$ of the original preprocessing cost. We expect that one permutation hashing (or its variant) will be adopted in practice. See more details in arXiv:1208.1259.

Acknowledgement: The research of Ping Li is partially supported by NSF-IIS-1249316, NSF-DMS-0808864, NSF-SES-1131848, and ONR-YIP-N000140910911. The research of Art B Owen is partially supported by NSF-0906056. The research of Cun-Hui Zhang is partially supported by NSF-DMS-0906420, NSF-DMS-1106753, NSF-DMS-1209014, and NSA-H98230-11-1-0205.

References

[1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Commun. ACM*, volume 51, pages 117–122, 2008.

[2] Leon Bottou. http://leon.bottou.org/projects/sgd.

[3] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, pages 327–336, Dallas, TX, 1998.

[4] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *WWW*, pages 1157 – 1166, Santa Clara, CA, 1997.

[5] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *STOC*, pages 106–112, 1977.

[6] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithm*, 55(1):58–75, 2005.

[7] 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.

[8] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. A large-scale study of the evolution of web pages. In *WWW*, pages 669–678, Budapest, Hungary, 2003.

[9] Jerome H. Friedman, F. Baskett, and L. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 24:1000–1006, 1975.

[10] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, Dallas, TX, 1998.

[11] Thorsten Joachims. Training linear svms in linear time. In *KDD*, pages 217–226, Pittsburgh, PA, 2006.

[12] Ping Li. Very sparse stable random projections for dimension reduction in l_α ($0 < \alpha \leq 2$) norm. In *KDD*, San Jose, CA, 2007.

[13] Ping Li and Kenneth W. Church. Using sketches to estimate associations. In *HLT/EMNLP*, pages 708–715, Vancouver, BC, Canada, 2005 (The full paper appeared in Commputational Linguistics in 2007).

[14] Ping Li, Kenneth W. Church, and Trevor J. Hastie. One sketch for all: Theory and applications of conditional random sampling. In *NIPS*, Vancouver, BC, Canada, 2008 (Preliminary results appeared in NIPS 2006).

[15] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *KDD*, pages 287–296, Philadelphia, PA, 2006.

[16] Ping Li, Anshumali Shrivastava, Joshua Moore, and Arnd Christian König. Hashing algorithms for large-scale learning. In *NIPS*, Granada, Spain, 2011.

[17] 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.

[18] Ping Li and Arnd Christian König. b-bit minwise hashing. In *WWW*, pages 671–680, Raleigh, NC, 2010.

[19] Ping Li, Arnd Christian König, and Wenhao Gui. b-bit minwise hashing for estimating three-way similarities. In *NIPS*, Vancouver, BC, 2010.

[20] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, pages 807–814, Corvalis, Oregon, 2007.

[21] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. Hash kernels for structured data. *Journal of Machine Learning Research*, 10:2615–2637, 2009.

[22] Anshumali Shrivastava and Ping Li. Fast near neighbor search in high-dimensional binary data. In *ECML*, 2012.

[23] Josef Sivic and Andrew Zisserman. Video google: a text retrieval approach to object matching in videos. In *ICCV*, 2003.

[24] Simon Tong. Lessons learned developing a practical large scale machine learning system. http://googleresearch.blogspot.com/2010/04/lessons-learned-developing-practical.html, 2008.

[25] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.

Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search

Anshumali Shrivastava ANSHU@CS.CORNELL.EDU
Dept. of Computer Science, Computing and Information Science (CIS), Cornell University, Ithaca, NY 14853, USA

Ping Li PINGLI@STAT.RUTGERS.EDU
Dept. of Statistics & Biostatistics, Dept. of Computer Science, Rutgers University, Piscataway, NJ 08854, USA

Abstract

The query complexity of *locality sensitive hashing (LSH)* based similarity search is dominated by the number of hash evaluations, and this number grows with the data size (Indyk & Motwani, 1998). 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, which requires applying a large number of permutations on the data. This is costly in computation and energy-consumption.

In this paper, we propose a hashing technique which generates all the necessary hash evaluations needed for similarity search, using one single permutation. The heart of the proposed hash function is a "rotation" scheme which densifies the sparse sketches of *one permutation hashing* (Li et al., 2012) in an unbiased fashion thereby maintaining the LSH property. This makes the obtained sketches suitable for hash table construction. This idea of rotation presented in this paper could be of independent interest for densifying other types of sparse sketches.

Using our proposed hashing method, the query time of a (K, L)-parameterized LSH 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. Our experimental evaluation on real data confirms that the proposed scheme significantly reduces the query processing time over minwise hashing without loss in retrieval accuracies.

1. Introduction

Near neighbor search is a fundamental problem with widespread applications in search, databases, learning, computer vision, etc. The task is to return a small number of data points from a dataset, which are most similar to a given input query. Efficient (sub-linear time) algorithms for near neighbor search has been an active research topic since the early days of computer science, for example, the *K-D tree* (Friedman et al., 1975). In this paper, we develop a new *hashing* technique for fast near neighbor search when the data are sparse, ultra-high-dimensional, and binary.

1.1. Massive, Sparse, High-Dimensional (Binary) Data

The use of ultra-high-dimensional data has become popular in machine learning practice. Tong (2008) discussed datasets with 10^{11} items and 10^9 features. Weinberger et al. (2009) experimented with a binary dataset of potentially 16 trillion unique features. Agarwal et al. (2011); Chandra et al. also described linear learning with massive binary data. Chapelle et al. (1999) reported good performance by using binary data for histogram-based image classification.

One reason why binary high-dimensional data are common in practice is due to the ubiquitous use of n-gram (e.g., n-contiguous words) features in text and vision applications. With $n \geq 3$, the size of the dictionary becomes very large and most of the grams appear at most once. More than 10 years ago, industry already used n-grams (Broder et al., 1997; Fetterly et al., 2003) with (e.g.,) $n \geq 5$.

1.2. Fast Approximate Near Neighbor Search and LSH

The framework of *locality sensitive hashing (LSH)* (Indyk & Motwani, 1998) is popular for sub-linear time near neighbor search. The idea of LSH is to bucket (group) the data points probabilistically in a hash table so that similar data points are more likely to reside in the same bucket.The performance of LSH depends on the data types and the implementations of specific underlying hashing algorithms. For binary data, minwise hashing is a popular choice.

1.3. Minwise hashing and b-Bit Minwise Hashing

For binary sparse data, it is routine to store only the locations of the nonzero entries. In other words, we can view binary vector in $\mathbb{R}^D$ equivalently as sets in $\Omega = \{0, 1, 2, ..., D-1\}$. Consider two sets $S_1, S_2 \subseteq \Omega$. A popular measure of similarity is the *resemblance* $R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$.

Consider a random permutation $\pi : \Omega \longrightarrow \Omega$. We apply π on S_1, S_2 and store the two minimum values under π:

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

$\min \pi(S_1)$ and $\min \pi(S_2)$, as the hashed values. By an elementary probability argument,

$$\mathbf{Pr}\left(\min \pi(S_1) = \min \pi(S_2)\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R \quad (1)$$

This method is known as *minwise hashing* (Broder, 1997; Broder et al., 1998; 1997). See Figure 1 for an illustration.

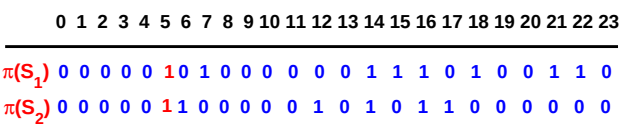

Figure 1. **(b-bit) minwise hashing**. Consider two binary vectors (sets), $S_1, S_2 \subseteq \Omega = \{0, 1, ..., 23\}$. The two permuted sets are: $\pi(S_1) = \{5, 7, 14, 15, 16, 18, 21, 22\}$, $\pi(S_2) = \{5, 6, 12, 14, 16, 17\}$. Thus, minwise stores $\min \pi(S_1) = 5$ and $\min \pi(S_2) = 5$. With b-bit minwise hashing, we store their lowest b bits. For example, if $b = 1$, we store 1 and 1, respectively.

Li & König (2010); Li et al. (2013) proposed b-bit minwise hashing by only storing the lowest b-bits (as opposed to, e.g., 64 bits) of the hashed values; also see Figure 1. This substantially reduces not only the storage but also the dimensionality. Li et al. (2011) applied this technique in large-scale learning. Shrivastava & Li (2012) directly used these bits to build hash tables for near neighbor search.

1.4. Classical LSH with Minwise Hashing

The crucial formula of the collision probability (1) suggests that minwise hashing belongs to the LSH family.

Let us consider the task of similarity search over a giant collection of sets (binary vectors) $\mathcal{C} = \{S_1, S_2, ..., S_N\}$, where N could run into billions. Given a query S_q, we are interested in finding $S \in \mathcal{C}$ with high value of $\frac{|S_q \cap S|}{|S_q \cup S|}$. The idea of linear scan is infeasible due to the size of $\mathcal{C}$. The idea behind LSH with minwise hashing is to concatenate K different minwise hashes for each $S_i \in \mathcal{C}$, and then store S_i in the bucket indexed by

$$B(S_i) = [h_{\pi_1}(S_i); h_{\pi_2}(S_i); ...; h_{\pi_K}(S_i)], \quad (2)$$

where $\pi_i : i \in \{1, 2, ..., K\}$ are K different random permutations and $h_{\pi_i}(S) = \min(\pi_i(S))$. Given a query S_q, we retrieve all the elements from bucket $B(S_q)$ for potential similar candidates. The bucket $B(S_q)$ contains elements $S_i \in \mathcal{C}$ whose K different hash values collide with that of the query. By the LSH property of the hash function, i.e., (1), these elements have higher probability of being similar to the query S_q compared to a random point. This probability value can be tuned by choosing appropriate value for parameter K. The entire process is repeated independently L times to ensure good recall. Here, the value of L is again optimized depending on the desired similarity levels.

The query time cost is dominated by $O(dKL)$ hash evaluations, where d is the number of nonzeros of the query vector. Our work will reduce the cost to $O(KL + dL)$. Previously, there were two major attempts to reduce query time (i.e., hashing cost), which generated multiple hash values from one single permutation: (i) *Conditional Random Sampling (CRS)* and (ii) *One permutation hashing (OPH)*.

1.5. Conditional Random Sampling (CRS)

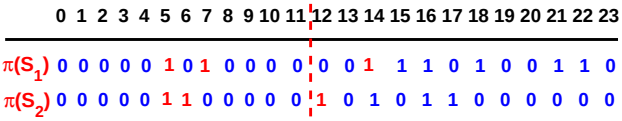

Figure 2. **Conditional Random Sampling (CRS)**. Suppose the data are already permuted under π. We store the smallest $k = 3$ nonzeros as "sketches": $\{5, 7, 14\}$ and $\{5, 6, 12\}$, respectively. Because $\min\{14, 12\} = 12$, we only look at up to the 12-th column (i.e., 13 columns in total). Because we have already applied a random permutation, any chunk of the data can be viewed as a random sample, in particular, the chunk from the 0-th column to the 12-th column. This looks we equivalently obtain a random sample of size 13, from which we can estimate any similarities, not just resemblance. Also, it clear that the method is naturally applicable to non-binary data. A careful analysis showed that it is (slightly) better NOT to use the last column, i.e., we only use up to the 11-th column (a sample of size 12 instead of 13).

Minwise hashing requires applying many permutations on the data. Li & Church (2007); Li et al. (2006) developed the *Conditional Random Sampling (CRS)* by directly taking the k smallest nonzeros (called *sketches*) from only one permutation. The method is unique in that it constructs an equivalent random sample pairwise (or group-wise) from these nonzeros. Note that the original minwise hashing paper (Broder, 1997) actually also took the smallest k nonzeros from one permutation. Li & Church (2007) showed that CRS is significantly more accurate. CRS naturally extends to multi-way similarity (Li et al., 2010; Shrivastava & Li, 2013) and the method is not restricted to binary data.

Although CRS requires only one permutation, the drawback of that method is that the samples (sketches) are not aligned. Thus, strictly speaking, CRS can not be used for linear learning or near neighbor search by building hash tables; otherwise the performance would not be ideal.

1.6. One Permutation Hashing

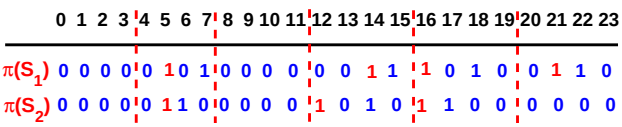

Figure 3. **One Permutation Hashing**. Instead of only storing the smallest nonzero in each permutation and repeating the permutation k times, we can use just one permutation, break the space evenly into k bins, and store the smallest nonzero in each bin. It is not directly applicable to near neighbor search due to empty bins.

Li et al. (2012) developed *one permutation hashing*. As illustrated in Figure 3, 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. When there are no empty bins, one permutation hashing estimates the resemblance without bias. However, if empty bins occur, we need a strategy to deal with them. The strategy adopted by Li et al. (2012) is to simply treat empty bins as "zeros" (i.e., *zero-coding*), which works well for linear learning. Note that, in Li et al. (2012), "zero" means zero in the "expanded metric space" (so that indicator function can be explicitly written as an inner product).

1.7. Limitation of One Permutation Hashing

One permutation hashing can not be directly used for near neighbor search by building hash tables because empty bins do not offer indexing capability. In other words, because of these empty bins, it is not possible to determine which bin value to use for bucketing.Using the LSH framework, each hash table may need (e.g.,) $10 \sim 20$ hash values and we may need (e.g.,) 100 or more hash tables. If we generate all the necessary (e.g., 2000) hash values from merely one permutation, the chance of having empty bins might be high in sparse data. For example, suppose on average each data vector has 500 nonzeros. If we use 2000 bins, then roughly $(1 - 1/2000)^{500} \approx 78\%$ of the bins would be empty.

Suppose we adopt the zero-coding strategy, by coding every empty bin with any fixed special number. If empty bins dominate, then two sparse vectors will become artificially "similar". On the other hand, suppose we use the "random-coding" strategy, by coding an empty bin randomly (and independently) as a number in $\{0, 1, 2, D-1\}$. Again, if empty bins dominate, then two sparse vectors which are similar in terms of the original resemblance may artificially become not so similar. We will see later that these schemes lead to significant deviation from the expected behavior. We need a better strategy to handle empty bins.

1.8. Our Proposal: One Permutation with Rotation

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 show empirically that its performance in near neighbor search is virtually identical to that of the original minwise hashing.

An interesting consequence is that our work supplies an unbiased estimate of resemblance regardless of the number of empty bins, unlike the original paper (Li et al., 2012).

2. Our Proposed Hashing Method

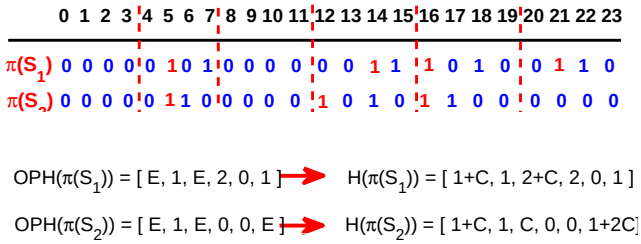

Figure 4. **One Permutation Hashing+Rotation**, with $D = 24$ (space size) and $k = 6$ bins (i.e., $D/k = 4$). For one permutation hashing (OPH), the hashed value is defined in (4). In particular, for set S_1, $OPH(\pi(S_1)) = [E, 1, E, 2, 0, 1]$ where E stands for "empty bins". The 0-th and 2-th bins are empty. In the 1-th bin, the minimum is 5, which becomes 1 after the modular operation: $5 \mod 4 = 1$. Our proposed hash method, defined in (5), needs to fill the two empty bins for S_1. For the 0-th bin, the first non-empty bin on the right is the 1-th bin (i.e., $t = 1$) with hashed value 1. Thus, we fill the 0-th empty bin with "$1 + C$". For the 2-th bin, the first non-empty bin on the right is the 3-th bin (i.e., $t = 1$ again) with hashed value 0 and hence we fill the 2-th bin with "C". The interesting case for S_2 is the 5-th bin, which is empty. The first non-empty bin on the right (circular) is the 1-th bin (as $5 + 2 \mod 6 = 1$). Thus, the 5-th bin becomes "1+2C".

Our proposed hashing method is easy to understand with an example as in Figure 4. Consider $S \subseteq \Omega = \{0, 1, ..., D-1\}$ and a random permutation $\pi : \Omega \longrightarrow \Omega$. As in Figure 3 (also in Figure 4), we divide the space evenly into k bins. For the j-th bin, where $0 \leq j \leq k-1$, we define the set

$$M_j(\pi(S)) = \left\{ \pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k} \right) \right\} \quad (3)$$

If $M_j(\pi(S))$ is empty, we denote the hashed value under this one permutation hashing by $\underset{j}{OPH}(\pi(S)) = E$, where E stands for "empty". If the set is not empty, we just take the minimum of the set (mod by D/k). In this paper, without loss of generality, we always assume D is divisible by k (otherwise we just increase D by padding zeros). That is, if $M_j(\pi(S)) \neq \phi$, we have

$$\underset{j}{OPH}(\pi(S)) = \min M_j(\pi(S)) \bmod \frac{D}{k} \quad (4)$$

We are now ready to define our hashing scheme, which is basically one permutation hashing plus a "rotation" scheme for filling empty bins. Formally, we define

$$\mathcal{H}_j(\pi(S)) = \begin{cases} \underset{j}{OPH}(\pi(S)) & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ \underset{(j+t) \bmod k}{OPH}(\pi(S)) + tC & \text{otherwise} \end{cases} \quad (5)$$

$$t = \min z, \quad s.t. \quad \underset{(j+z) \bmod k}{OPH}(\pi(S)) \neq E \quad (6)$$

Here $C \geq \frac{D}{k}+1$ is a constant, to avoid undesired collisions. Basically, we fill the empty bins with the hashed value of the first non-empty bin on the "right" (circular).

Theorem 1

$$\mathbf{Pr}\left(\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2))\right) = R \tag{7}$$

Theorem 1 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 4, another promising use of our work is for estimating resemblance using only linear estimator (i.e., an estimator which is an inner product); see Section 3.

3. Resemblance Estimation

Theorem 1 naturally suggests a linear, unbiased estimator of the resemblance R:

$$\hat{R} = \frac{1}{k}\sum_{j=0}^{k-1} 1\left\{\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2))\right\} \tag{8}$$

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 (Joachims, 2006; Shalev-Shwartz et al., 2007; Bottou; Fan et al., 2008) such as linear SVM.

The original one permutation hashing paper (Li et al., 2012) proved the following unbiased estimator of R

$$\hat{R}_{OPH,ub} = \frac{N_{mat}}{k - N_{emp}} \tag{9}$$

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

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

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, Li et al. (2012) provided a modified estimator

$$\hat{R}_{OPH} = \frac{N_{mat}}{\sqrt{k - N_{emp}^{(1)}}\sqrt{k - N_{emp}^{(2)}}} \tag{10}$$

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

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

This estimator, although is not unbiased (for estimating R), works well in the context of linear learning. In fact, the normalization by $\sqrt{k - N_{emp}^{(1)}}\sqrt{k - N_{emp}^{(2)}}$ is smoothly integrated in the SVM training procedure which usually requires the input vectors to have unit l_2 norms.

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 (8) and $\hat{R}_{OPH}$ in (10). The dataset, extracted from a chunk of Web crawl (with 2^{16} documents), is described in Table 1, which consists of 12 pairs of sets (i.e., total 24 words). Each set consists of the document IDs which contain the word at least once.

Table 1. 12 pairs of words used in Experiment 1. For example, "A" and "THE" correspond to the two sets of document IDs which contained word "A" and word "THE" respectively.

Word 1	Word 2	$\lvert S_1\rvert$	$\lvert S_2\rvert$	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

Figure 5 and Figure 6 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 original one permutation hashing paper 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 (Li & König, 2010), 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 well for resemblance estimation (and hence it is useful for training resemblance kernel SVM using a linear algorithm). The more exciting application of our work would be approximate near neighbor search.

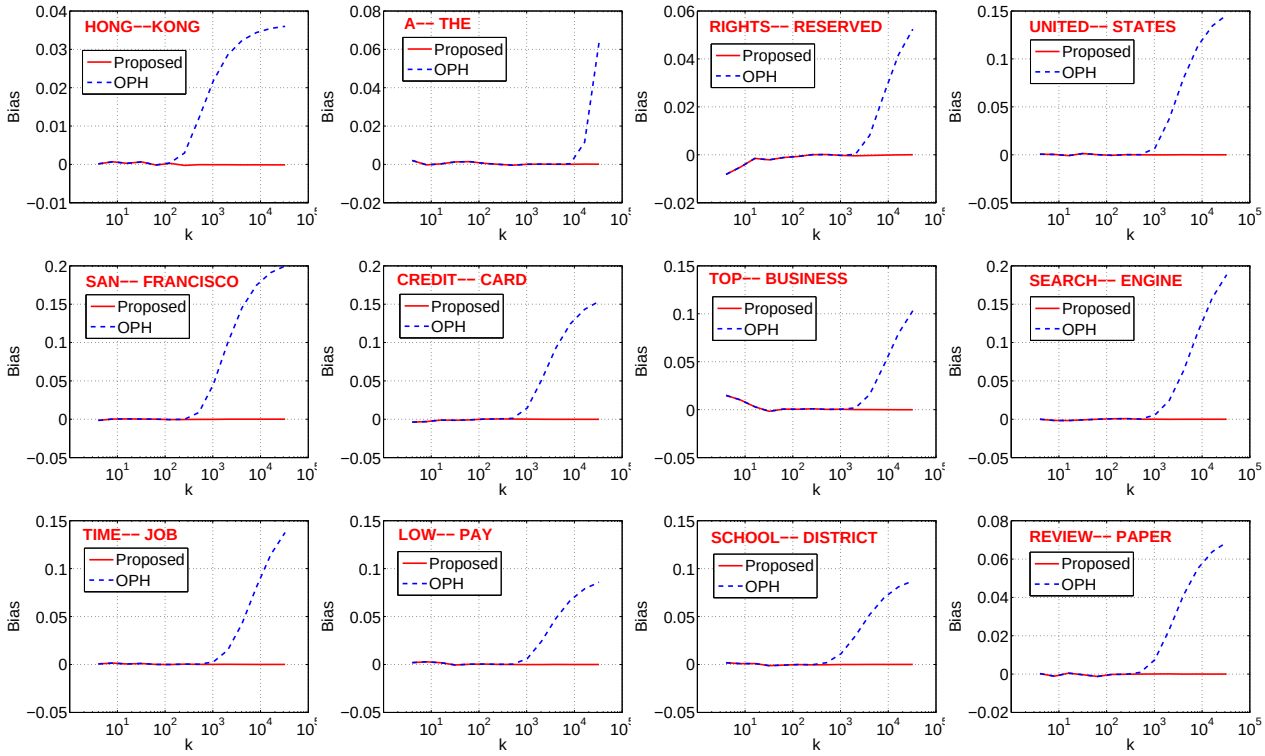

Figure 5. **Bias in resemblance estimation**. The plots are the biases of the proposed estimator $\hat{R}$ defined in (8) and the previous estimator $\hat{R}_{OPH}$ defined in (10) from the original one permutation hashing paper (Li et al., 2012). See Table 1 for a detailed description of the data. It is clear that the proposed estimator $\hat{R}$ is strictly unbiased regardless of k, the number of bins which ranges from 2^2 to 2^{15}.

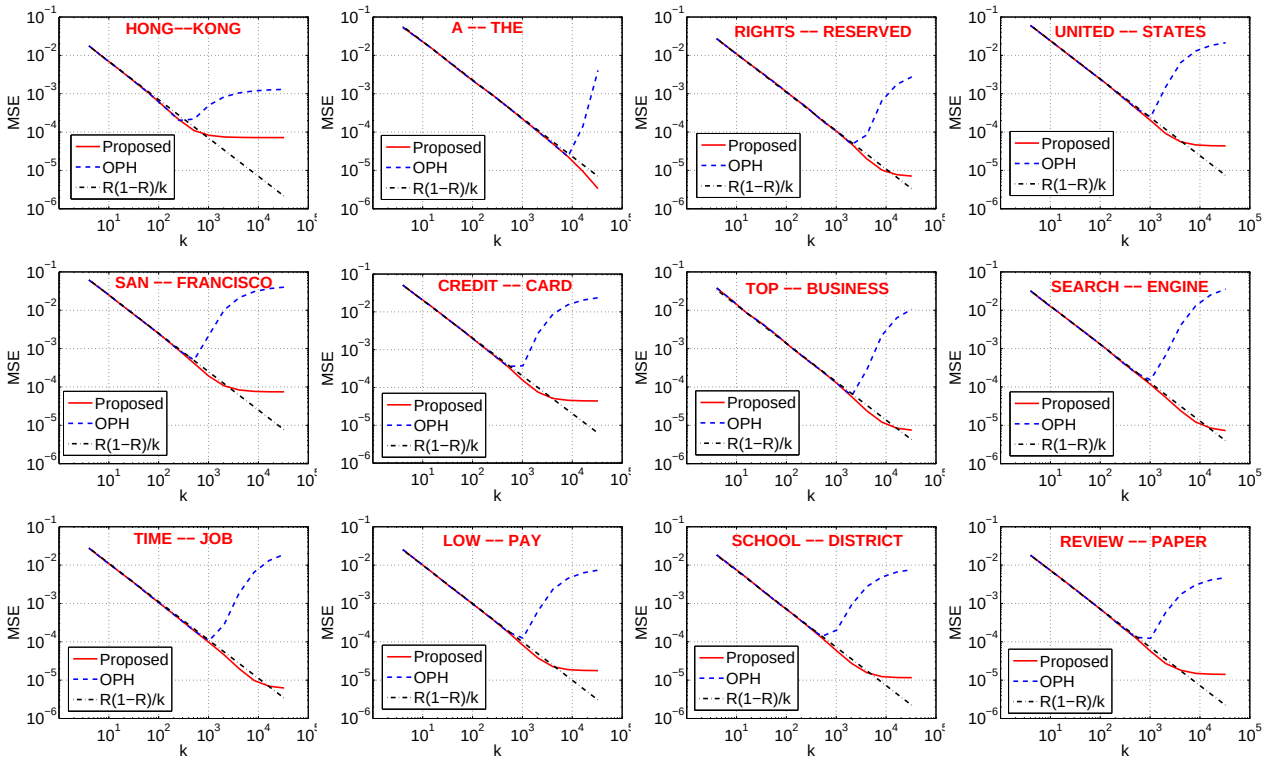

Figure 6. **MSE in resemblance estimation**. See the caption of Figure 5 for more descriptions. The MSEs of $\hat{R}$ essentially follow $R(1-R)/k$ which is the variance of the original minwise hashing, unless k is too large (i.e., when there are too many empty bins). The previous estimator $\hat{R}_{OPH}$ estimates R poorly when k is large, as it is not designed for estimating R.

4. Near Neighbor Search

We implement the classical LSH algorithm described in Section 1.4 using the standard procedures (Andoni & Indyk, 2004) with the following choices of hash functions:

- **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 (5) 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:

$$h_j^{EE}(S) = \begin{cases} \underset{j}{OPH}(\pi(S)), & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ \text{A fixed special symbol}, & \text{otherwise} \end{cases} \tag{11}$$

- **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

$$h_j^{ENE}(S) = \begin{cases} \underset{j}{OPH}(\pi(S)), & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ rand(\text{new seed}), & \text{otherwise} \end{cases} \tag{12}$$

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.

4.1. 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 4000 non-zeros on an 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 $K \times L$ independent 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 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 of K and L 10 times, and report the average over these 10 runs.

Figure 7 presents the plots of the fraction of points retrieved per query corresponding to $K = 10$ for all the three datasets with different choices of L. Due to space constraint, here we only show the recall plots for various values of L and T_0 corresponding to $K = 10$ in Figure 8. The plots corresponding to $K = \{6, 8, 12\}$ are very similar.

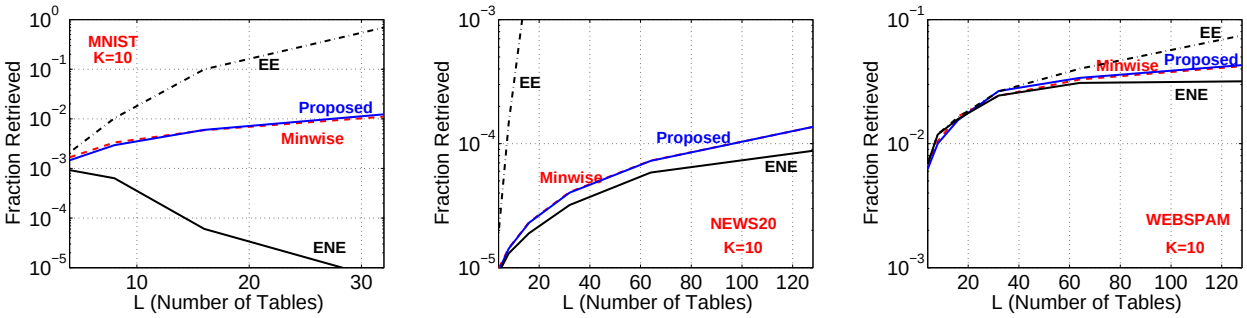

Figure 7. Fraction of points retrieved by the bucketing scheme per query corresponding to $K = 10$ shown over different choices of L. Results are averaged over 10 independent runs. Plots with $K = \{6, 8, 12\}$ are very similar in nature.

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 5).

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 $K \geq 10$.

5. Reduction in Computation Cost

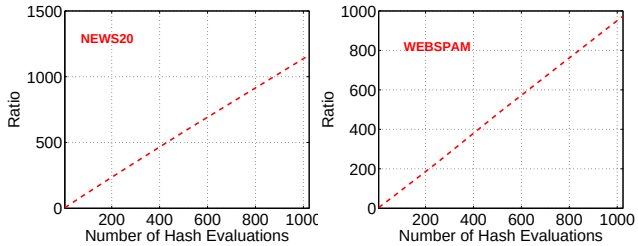

Figure 9. Ratio of time taken by minwise hashing to the time taken by our proposed scheme with respect to the number of hash evaluations, on the NEWS20 and WEBSPAM datasets.

5.1. 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 (Indyk & Motwani, 1998). 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" (see Figure 4). 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 9. As expected, with the increase in the number of hash evaluations minwise hashing is linearly more costly than our proposal.

Figure 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. Approximating these $K \times L$ permutations using D cheap universal hash functions (Carter & Wegman, 1977; Nisan, 1990; Mitzenmacher & Vadhan, 2008) 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.

5.2. Reduction in Total Query Time

The total query complexity of the LSH algorithm for retrieving approximate near neighbor using minwise hashing

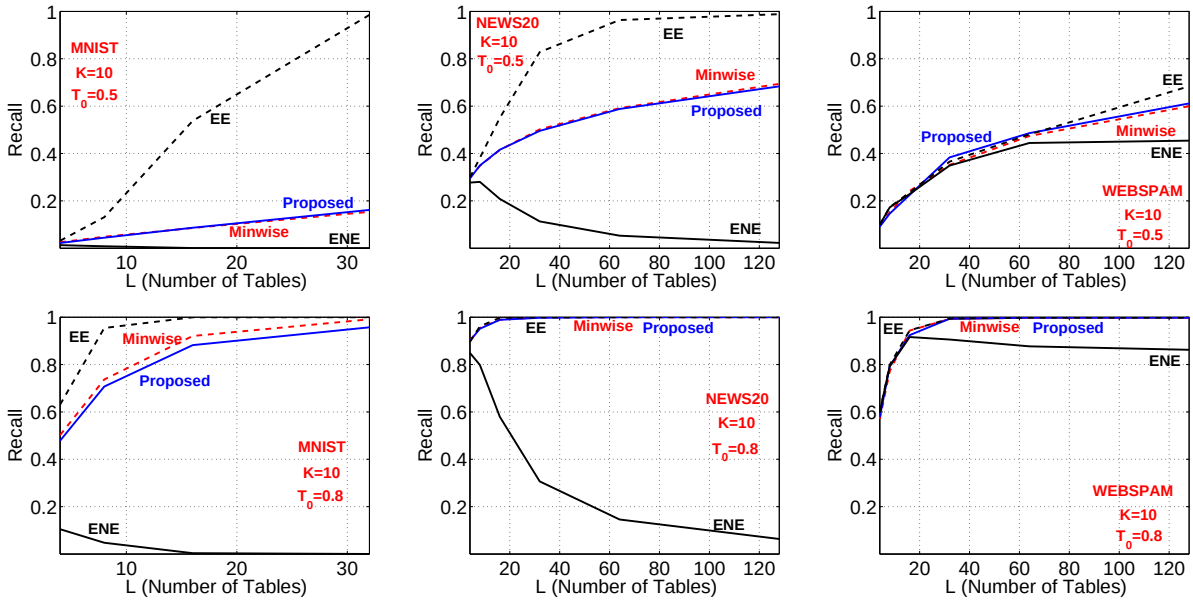

Figure 8. Recall values of points having similarity with the query greater than $T_0 = \{0.5, 0.8\}$ corresponding to $K = 10$ shown over different choices of L. Results are averaged over 10 independent runs. Plots with $K = \{6, 8, 12\}$ are very similar in nature.

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 (Indyk & Motwani, 1998; Dasgupta et al., 2011). In theory, the number of data points retrieved using the appropriate choice of LSH parameters (K, L), in the worst case, is $O(L)$ (Indyk & Motwani, 1998). 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 (Shrivastava & Li, 2012). This decreases the re-ranking cost further.

5.3. 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. Conclusion

With the explosion of data in modern applications, the brute force strategy of scanning all data points for searching for near neighbors is prohibitively expensive, especially in user-facing applications like search. LSH is popular for achieving sub-linear near neighbor search. Minwise hashing, which belongs to the LSH family, is one of the basic primitives in many "big data" processing algorithms. In addition to near-neighbor search, these techniques are frequently used in popular operations like near-duplicate detection (Broder, 1997; Manku et al., 2007; Henzinger, 2006), all-pair similarity (Bayardo et al., 2007), similarity join/record-linkage (Koudas & Srivastava, 2005), temporal correlation (Chien & Immorlica, 2005), etc.

Current large-scale data processing systems deploying indexed tables based on minwise hashing suffer from the problem of costly processing. In this paper, we propose a new hashing scheme based on a novel "rotation" technique to densify one permutation hashes (Li et al., 2012). The obtained hash function possess the properties of minwise hashing, and at the same time it is very fast to compute.

Our experimental evaluations on real data suggest that the proposed hash function offers massive savings in computation cost compared to minwise hashing. These savings come without any trade-offs in the performance measures.

Acknowledgement

The work is supported by NSF-III-1360971, NSF-Bigdata-1419210, ONR-N00014-13-1-0764, and AFOSR-FA9550-13-1-0137. We thank the constructive review comments from SIGKDD 2013, NIPS 2013, and ICML 2014, for improving the quality of the paper. The sample matlab code for our proposed hash function is available upon request.

References

Agarwal, Alekh, Chapelle, Olivier, Dudik, Miroslav, and Langford, John. A reliable effective terascale linear learning system. Technical report, arXiv:1110.4198, 2011.

Andoni, Alexandr and Indyk, Piotr. E2lsh: Exact euclidean locality sensitive hashing. Technical report, 2004.

Bayardo, Roberto J., Ma, Yiming, and Srikant, Ramakrishnan. Scaling up all pairs similarity search. In *WWW*, pp. 131–140, 2007.

Bottou, Leon. http://leon.bottou.org/projects/sgd.

Broder, Andrei Z. On the resemblance and containment of documents. In *the Compression and Complexity of Sequences*, pp. 21–29, Positano, Italy, 1997.

Broder, Andrei Z., Glassman, Steven C., Manasse, Mark S., and Zweig, Geoffrey. Syntactic clustering of the web. In *WWW*, pp. 1157 – 1166, Santa Clara, CA, 1997.

Broder, Andrei Z., Charikar, Moses, Frieze, Alan M., and Mitzenmacher, Michael. Min-wise independent permutations. In *STOC*, pp. 327–336, Dallas, TX, 1998.

Carter, J. Lawrence and Wegman, Mark N. Universal classes of hash functions. In *STOC*, pp. 106–112, 1977.

Chandra, Tushar, Ie, Eugene, Goldman, Kenneth, Llinares, Tomas Lloret, McFadden, Jim, Pereira, Fernando, Redstone, Joshua, Shaked, Tal, and Singer, Yoram. Sibyl: a system for large scale machine learning.

Chapelle, Olivier, Haffner, Patrick, and Vapnik, Vladimir N. Support vector machines for histogram-based image classification. *IEEE Trans. Neural Networks*, 10(5):1055–1064, 1999.

Chien, Steve and Immorlica, Nicole. Semantic similarity between search engine queries using temporal correlation. In *WWW*, pp. 2–11, 2005.

Dasgupta, Anirban, Kumar, Ravi, and Sarlós, Tamás. Fast locality-sensitive hashing. In *KDD*, pp. 1073–1081, 2011.

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, and Lin, Chih-Jen. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Fetterly, Dennis, Manasse, Mark, Najork, Marc, and Wiener, Janet L. A large-scale study of the evolution of web pages. In *WWW*, pp. 669–678, Budapest, Hungary, 2003.

Friedman, Jerome H., Baskett, F., and Shustek, L. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 24:1000–1006, 1975.

Henzinger, Monika Rauch. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, pp. 284–291, 2006.

Indyk, Piotr and Motwani, Rajeev. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pp. 604–613, Dallas, TX, 1998.

Joachims, Thorsten. Training linear svms in linear time. In *KDD*, pp. 217–226, Pittsburgh, PA, 2006.

Koudas, Nick and Srivastava, Divesh. Approximate joins: Concepts and techniques. In *VLDB*, pp. 1363, 2005.

Li, Ping and Church, Kenneth W. 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.

Li, Ping and König, Arnd Christian. b-bit minwise hashing. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 671–680, Raleigh, NC, 2010.

Li, Ping, Church, Kenneth W., and Hastie, Trevor J. Conditional random sampling: A sketch-based sampling technique for sparse data. In *NIPS*, pp. 873–880, Vancouver, BC, Canada, 2006.

Li, Ping, König, Arnd Christian, and Gui, Wenhao. b-bit minwise hashing for estimating three-way similarities. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2010.

Li, Ping, Shrivastava, Anshumali, Moore, Joshua, and König, Arnd Christian. Hashing algorithms for large-scale learning. In *NIPS*, Granada, Spain, 2011.

Li, Ping, Owen, Art B, and Zhang, Cun-Hui. One permutation hashing. In *NIPS*, Lake Tahoe, NV, 2012.

Li, Ping, Shrivastava, Anshumali, and König, Arnd Christian. b-bit minwise hashing in practice. In *Internetware*, Changsha, China, 2013.

Manku, Gurmeet Singh, Jain, Arvind, and Sarma, Anish Das. Detecting Near-Duplicates for Web-Crawling. In *WWW*, Banff, Alberta, Canada, 2007.

Mitzenmacher, Michael and Vadhan, Salil. Why simple hash functions work: exploiting the entropy in a data stream. In *SODA*, 2008.

Nisan, Noam. Pseudorandom generators for space-bounded computations. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC, pp. 204–212, 1990.

Shalev-Shwartz, Shai, Singer, Yoram, and Srebro, Nathan. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, pp. 807–814, Corvalis, Oregon, 2007.

Shrivastava, Anshumali and Li, Ping. Fast near neighbor search in high-dimensional binary data. In *ECML*, 2012.

Shrivastava, Anshumali and Li, Ping. Beyond pairwise: Provably fast algorithms for approximate k-way similarity search. In *NIPS*, Lake Tahoe, NV, 2013.

Tong, Simon. Lessons learned developing a practical large scale machine learning system. http://googleresearch.blogspot.com/2010/04/lessons-learned-developing-practical.html, 2008.

Weinberger, Kilian, Dasgupta, Anirban, Langford, John, Smola, Alex, and Attenberg, Josh. Feature hashing for large scale multitask learning. In *ICML*, pp. 1113–1120, 2009.

Improved Densification of One Permutation Hashing

Anshumali Shrivastava
Department of Computer Science
Computing and Information Science
Cornell University
Ithaca, NY 14853, USA
anshu@cs.cornell.edu

Ping Li
Department of Statistics and Biostatistics
Department of Computer Science
Rutgers University
Piscataway, NJ 08854, USA
pingli@stat.rutgers.edu

Abstract

The existing work on densification of one permutation hashing [24] reduces the query processing cost of the (K, L)-parameterized Locality Sensitive Hashing (LSH) algorithm with minwise hashing, from $O(dKL)$ to merely $O(d + KL)$, 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. While that is a substantial improvement, our analysis reveals that the existing densification scheme in [24] is sub-optimal. In particular, there is no enough randomness in that procedure, which affects its accuracy on very sparse datasets.

In this paper, we provide a new densification procedure which is provably better than the existing scheme [24]. This improvement is more significant for very sparse datasets which are common over the web. The improved technique has the same cost of $O(d + KL)$ for query processing, thereby making it strictly preferable over the existing procedure. Experimental evaluations on public datasets, in the task of hashing based near neighbor search, support our theoretical findings.

1 Introduction

Binary representations are common for high dimensional sparse data over the web [8, 25, 26, 1], especially for text data represented by high-order n-grams [4, 12]. Binary vectors can also be equivalently viewed as sets, over the universe of all the features, containing only locations of the non-zero entries. Given two sets S_1, $S_2 \subseteq \Omega = \{1, 2, ..., D\}$, a popular measure of similarity between sets (or binary vectors) is the *resemblance* R, defined as

$$R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{a}{f_1 + f_2 - a}, \tag{1}$$

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

It is well-known that minwise hashing belongs to the *Locality Sensitive Hashing (LSH)* family [5, 9]. The method applies a random permutation $\pi : \Omega \rightarrow \Omega$, on the given set S, and stores the minimum value after the permutation mapping. Formally,

$$h_\pi(S) = \min(\pi(S)). \tag{2}$$

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

$$Pr(h_\pi(S_1) = h_\pi(S_2)) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R. \tag{3}$$

The probability of collision (equality of hash values), under minwise hashing, is equal to the similarity of interest R. This property, also known as the *LSH property* [14, 9], makes minwise hash functions h_π suitable for creating hash buckets, which leads to sublinear algorithms for similarity search. Because of this same LSH property, minwise hashing is a popular indexing technique for a variety of large-scale data processing applications, which include duplicate detection [4, 13], all-pair similarity [3], fast linear learning [19], temporal correlation [10], 3-way similarity & retrieval [17, 23], graph algorithms [6, 11, 21], and more.

Querying with a standard (K, L)-parameterized LSH algorithm [14], for fast similarity search, requires computing $K \times L$ min-hash values per query, where K is the number of hashes in each hash table and L is the number of hash tables. In theory, the value of KL grows with the data size [14]. In practice, typically, this number ranges from a few hundreds to a few thousands. Thus, processing a single query, for near-neighbor search, requires evaluating hundreds or thousands of independent permutations π (or cheaper universal approximations to permutations [7, 22, 20]) over the given data vector. If d denotes the number of non-zeros in the query vector, then the query preprocessing cost is $O(dKL)$ which is also the bottleneck step in the LSH algorithm [14]. Query time (latency) is crucial in many user-facing applications, such as search.

Linear learning with b-bit minwise hashing [19], requires multiple evaluations (say k) of h_π for a given data vector. Computing k different min-hashes of the test data costs $O(dk)$, while after processing, classifying this data vector

(with SVM or logistic regression) only requires a single inner product with the weight vector which is $O(k)$. Again, the bottleneck step during testing prediction is the evaluation of k min-hashes. Testing time directly translates into the latency of on-line classification systems.

The idea of storing k contiguous minimum values after one single permutation [4, 15, 16] leads to hash values which do not satisfy the LSH property because the hashes are not properly aligned. The estimators are also not linear, and therefore they do not lead to feature representation for linear learning with resemblance. This is a serious limitation.

Recently it was shown that a "rotation" technique [24] for densifying sparse sketches from one permutation hashing [18] solves the problem of costly processing with minwise hashing (See Sec. 2). The scheme only requires a single permutation and generates k different hash values, satisfying the LSH property (i.e., Eq.(3)), in linear time $O(d+k)$, thereby reducing a factor d in the processing cost compared to the original minwise hashing.

Our Contributions: In this paper, we argue that the existing densification scheme [24] is not the optimal way of densifying the sparse sketches of one permutation hashing at the given processing cost. In particular, we provide a provably better densification scheme for generating k hashes with the same processing cost of $O(d+k)$. Our contributions can be summarized as follows.

- Our detailed variance analysis of the hashes obtained from the existing densification scheme [24] reveals that there is no enough randomness in that procedure which leads to high variance in very sparse datasets.
- We provide a new densification scheme for one permutation hashing with provably smaller variance than the scheme in [24]. The improvement becomes more significant for very sparse datasets which are common in practice. The improved scheme retains the computational complexity of $O(d+k)$ for computing k different hash evaluations of a given vector.
- We provide experimental evidences on publicly available datasets, which demonstrate the superiority of the improved densification procedure over the existing scheme, in the task of resemblance estimation and as well as the task of near neighbor retrieval with LSH.

2 Background

2.1 One Permutation Hashing

As illustrated in Figure 1, instead of conducting k independent permutations, *one permutation hashing* [18] uses only one permutation and partitions the (permuted) feature space into k bins. In other words, a single permutation π is used to first shuffle the given binary vector, and then the shuffled vector is binned into k evenly spaced bins. The k minimums, computed for each bin separately, are the k different hash values. Obviously, empty bins are possible.

	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 1: One permutation hashes [18] for vectors S_1 and S_2 using a single permutation π. For bins not containing any non-zeros, we use special symbol "E".

For example, in Figure 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 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

$$M_j(\pi(S)) = \left\{ \pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k} \right) \right\} \quad (4)$$

We assume for the rest of the paper 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

$$\underset{j}{OPH}(\pi(S)) = \begin{cases} E, & \text{if } \pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k} \right) = \phi \\ M_j(\pi(S)) \bmod \frac{D}{k}, & \text{otherwise} \end{cases} \quad (5)$$

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 1. If this intersection is null, i.e., $\pi(S) \cap \left[\frac{Dj}{k}, \frac{D(j+1)}{k} \right) = \phi$, then $\underset{j}{OPH}(\pi(S)) = E$.

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:

$$I^j_{emp} = \begin{cases} 1, & \text{if } \underset{j}{OPH}(\pi(S_1)) = \underset{j}{OPH}(\pi(S_2)) = E \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Simultaneously empty bins are only defined with respect to two sets S_1 and S_2. In Figure 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, [18] showed

$$Pr\left(\underset{j}{OPH}(\pi(S_1)) = \underset{j}{OPH}(\pi(S_2)) \middle| I^j_{emp} = 0\right) = R \tag{7}$$

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 (7). The simple strategy of ignoring empty bins leads to biased estimators of resemblance and shows poor performance [24]. Because of this same reason, one permutation hashing cannot be directly used to extract random features for linear learning with resemblance kernel.

2.2 Densifying One Permutation Hashing for Indexing and Linear Learning

[24] proposed 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 1 is shown in Figure 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.

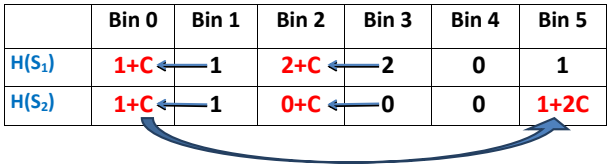

Figure 2: Densification by "rotation" for filling empty bins generated from one permutation hashing [24]. Every empty bin is assigned the value of the closest non-empty bin, towards right (circular), with an offset C. For the configuration shown in Figure 1, the above figure shows the new assigned values (in red) of empty bins after densification.

Given the configuration in Figure 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 would be 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:

$$\mathcal{H}_j(S) = \begin{cases} \underset{j}{OPH}(\pi(S)) & \text{if } \underset{j}{OPH}(\pi(S)) \neq E \\ \underset{(j+t) \bmod k}{OPH}(\pi(S)) + tC & \text{otherwise} \end{cases} \tag{8}$$

$$t = \min z, \quad s.t. \quad \underset{(j+z) \bmod k}{OPH}(\pi(S)) \neq E \tag{9}$$

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 $I^j_{emp} = 1$. [24] proved this fact as a theorem.

Theorem 1 *[24]*

$$\mathbf{Pr}\left(\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\right) = R \tag{10}$$

Theorem 1 implies that $\mathcal{H}$ satisfies the LSH property and hence it is suitable for indexing based sublinear similarity search. 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 [19] 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.

3 Variance Analysis of Existing Scheme

We first provide the variance analysis of the existing scheme [24]. Theorem 1 leads to an unbiased estimator of R between S_1 and S_2 defined as:

$$\hat{R} = \frac{1}{k}\sum_{j=0}^{k-1} \mathbf{1}\{\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\}. \tag{11}$$

Denote the number of simultaneously empty bins by

$$N_{emp} = \sum_{j=0}^{k-1} \mathbf{1}\{I^j_{emp} = 1\}, \tag{12}$$

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

$$M^N_j = \mathbf{1}\{I^j_{emp} = 0 \text{ and } \mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\} \tag{13}$$

$$M^E_j = \mathbf{1}\{I^j_{emp} = 1 \text{ and } \mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\} \tag{14}$$

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

$$\mathbb{E}(M_j^N | I_{emp}^j = 0) = \mathbb{E}(M_j^E | I_{emp}^j = 1) = \mathbb{E}(M_j^E + M_j^N) = R \ \forall j \quad (15)$$

It is not difficult to show that,

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

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

$$\hat{R} = \frac{1}{k}\sum_{j=0}^{k-1}\left[M_j^E + M_j^N\right] \quad (16)$$

We are interested in computing

$$Var(\hat{R}) = \mathbb{E}\left(\left(\frac{1}{k}\sum_{j=0}^{k-1}\left[M_j^E + M_j^N\right]\right)^2\right) - R^2 \quad (17)$$

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

$$f(m) = \mathbb{E}\left(\left(\frac{1}{k}\sum_{j=0}^{k-1}\left[M_j^E + M_j^N\right]\right)^2 \Big| m\right) \quad (18)$$

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,

$$\mathbb{E}\left[\sum_{j=0}^{k-1}\left[(M_j^N)^2 + (M_j^E)^2\right] \Big| m\right] = kR \quad (19)$$

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

Lemma 1

$$\mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^N \Big| m\right] = m(m-1)R\tilde{R} \quad (20)$$

Lemma 2

$$\mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^E \Big| m\right] = 2m(k-m)\left[\frac{R}{m} + \frac{(m-1)R\tilde{R}}{m}\right] \quad (21)$$

Lemma 3

$$\mathbb{E}\left[\sum_{i\neq j} M_i^E M_j^E \Big| m\right] = (k-m)(k-m-1) \times \left[\frac{2R}{m+1} + \frac{(m-1)R\tilde{R}}{m+1}\right] \quad (22)$$

Combining the expressions from the above 3 Lemmas and Eq.(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 2

$$Var(\hat{R}) = \frac{R}{k} + A\frac{R}{k} + B\frac{R\tilde{R}}{k} - R^2 \quad (23)$$
$$A = 2\mathbb{E}\left[\frac{N_{emp}}{k - N_{emp} + 1}\right]$$
$$B = (k+1)\mathbb{E}\left[\frac{k - N_{emp} - 1}{k - N_{emp} + 1}\right]$$

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 [18].

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

4 Intuition for the Improved Scheme

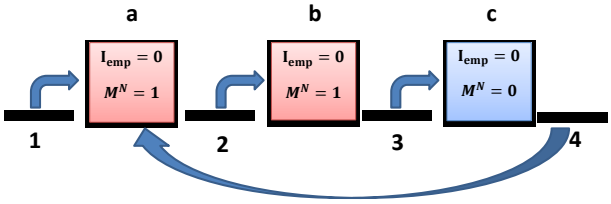

Figure 3: Illustration of the existing densification scheme [24]. The 3 boxes indicate 3 simultaneously non-empty bins. Any simultaneously empty bin has 4 possible positions shown by blank spaces. Arrow indicates the choice of simultaneous non-empty bins picked by simultaneously empty bins occurring in the corresponding positions. A simultaneously empty bin occurring in position 3 uses the information from Bin c. The randomness is in the position number of these bins which depends on π.

Consider a situation in Figure 3, 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 2.

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 3)

$$p = \frac{2}{m+1} \tag{24}$$

The value of p is high because there is no 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 3. 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.

5 The Improved Densification Scheme

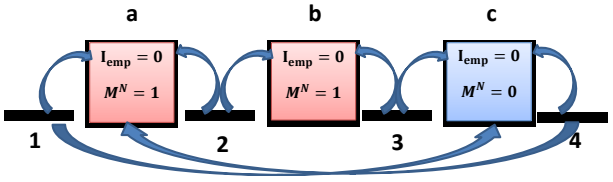

Figure 4: Illustration of the improved densification scheme. For every simultaneously empty bin, in the blank position, instead of always choosing the simultaneously non-empty bin from right, the new scheme randomly chooses to go either left or right. A simultaneously empty bin occurring at position 2 uniformly chooses among Bin a or Bin b.

Our proposal is explained in Figure 4. 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 1 is shown in Figure 5. 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.

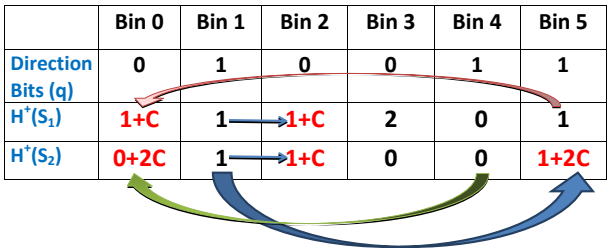

Figure 5: Assigned values (in red) of empty bins from Figure 1 using the improved densification procedure. Every empty Bin i uses the value of the closest non-empty bin, towards circular left or circular right depending on the random direction bit q_i, with offset C.

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 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 $2C$. It is 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_j $j = \{0, 1, 2, ..., k-1\}$ 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

$$\mathcal{H}^+_j(S) = \begin{cases} \underset{(j-t_1) \bmod k}{OPH}(\pi(S)) + t_1 C \\ \qquad \text{if } q_j = 0 \text{ and } \underset{j}{OPH}(\pi(S)) = E \\ \underset{(j+t_2) \bmod k}{OPH}(\pi(S)) + t_2 C \\ \qquad \text{if } q_j = 1 \text{ and } \underset{j}{OPH}(\pi(S)) = E \\ \underset{j}{OPH}(\pi(S)) \qquad \text{otherwise} \end{cases} \tag{25}$$

where

$$t_1 = \min z, \quad s.t. \quad \underset{(j-z) \bmod k}{OPH}(\pi(S)) \neq E \tag{26}$$

$$t_2 = \min z, \quad s.t. \quad \underset{(j+z) \bmod k}{OPH}(\pi(S)) \neq E \tag{27}$$

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 evaluations 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.

It is not difficult to show that $\mathcal{H}^+$ satisfies the LSH property for resemblance, which we state as a theorem.

Theorem 3

$$\mathbf{Pr}\left(\mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\right) = R \tag{28}$$

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

$$\hat{R}^+ = \frac{1}{k}\sum_{j=0}^{k-1} \mathbf{1}\{\mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\}. \tag{29}$$

6 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 Sec. 3, we denote

$$M_j^{N+} = \mathbf{1}\{I_{emp}^j = 0 \text{ and } \mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\} \tag{30}$$

$$M_j^{E+} = \mathbf{1}\{I_{emp}^j = 1 \text{ and } \mathcal{H}_j^+(S_1) = \mathcal{H}_j^+(S_2)\} \tag{31}$$

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 1 and Lemma 2 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 4

$$\mathbb{E}\left[\sum_{i\neq j} M_i^{E+}M_j^{E+}\middle| m\right] = (k-m)(k-m-1) \times \left[\frac{3R}{2(m+1)} + \frac{(2m-1)R\tilde{R}}{2(m+1)}\right] \tag{32}$$

The theoretical variance of the new estimator $\hat{R}^+$ is given by the following Theorem 4.

Theorem 4

$$Var(\hat{R}^+) = \frac{R}{k} + A^+\frac{R}{k^2} + B^+\frac{R\tilde{R}}{k^2} - R^2 \tag{33}$$

$$A^+ = \mathbb{E}\left[\frac{N_{emp}(4k - N_{emp} + 1)}{2(k - N_{emp} + 1)}\right]$$

$$B^+ = \mathbb{E}\left[\frac{2k^3 + N_{emp}^2 - N_{emp}(2k^2 + 2k + 1) - 2k}{2(k - N_{emp} + 1)}\right]$$

The new scheme reduces the value of p (see Eq.(24)) from $\frac{2}{m+1}$ to $\frac{1.5}{m+1}$. As argued in Sec. 4, this reduces the overall variance. Here, we state it as theorem that $Var(\hat{R}^+) \leq Var(\hat{R})$ always.

Theorem 5

$$Var(\hat{R}^+) \leq Var(\hat{R}) \tag{34}$$

More precisely,

$$Var(\hat{R}) - Var(\hat{R}^+) = \mathbb{E}\left[\frac{(N_{emp})(N_{emp}-1)}{2k^2(k - N_{emp} + 1)}[R - R\tilde{R}]\right] \tag{35}$$

The probability of simultaneously empty bins increases with increasing sparsity in dataset and the total number of bins k. We can see from Theorem 5 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.

7 Evaluations

Our first experiment concerns the validation of the theoretical variances of the two densification schemes. The second experiment focuses on comparing the two schemes in the context of near neighbor search with LSH.

7.1 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 webcrawl dataset which consists of word representation from 2^{16} documents. Every word is represented as a binary vector (or set) of $D = 2^{16}$ dimension, with a feature value of 1 indicating the presence of that word in the corresponding document. See Table 1 for detailed information of the data.

For all 12 pairs of words, we estimate the resemblance using the two estimators $\mathcal{R}$ and $\mathcal{R}^+$. We plot the empirical

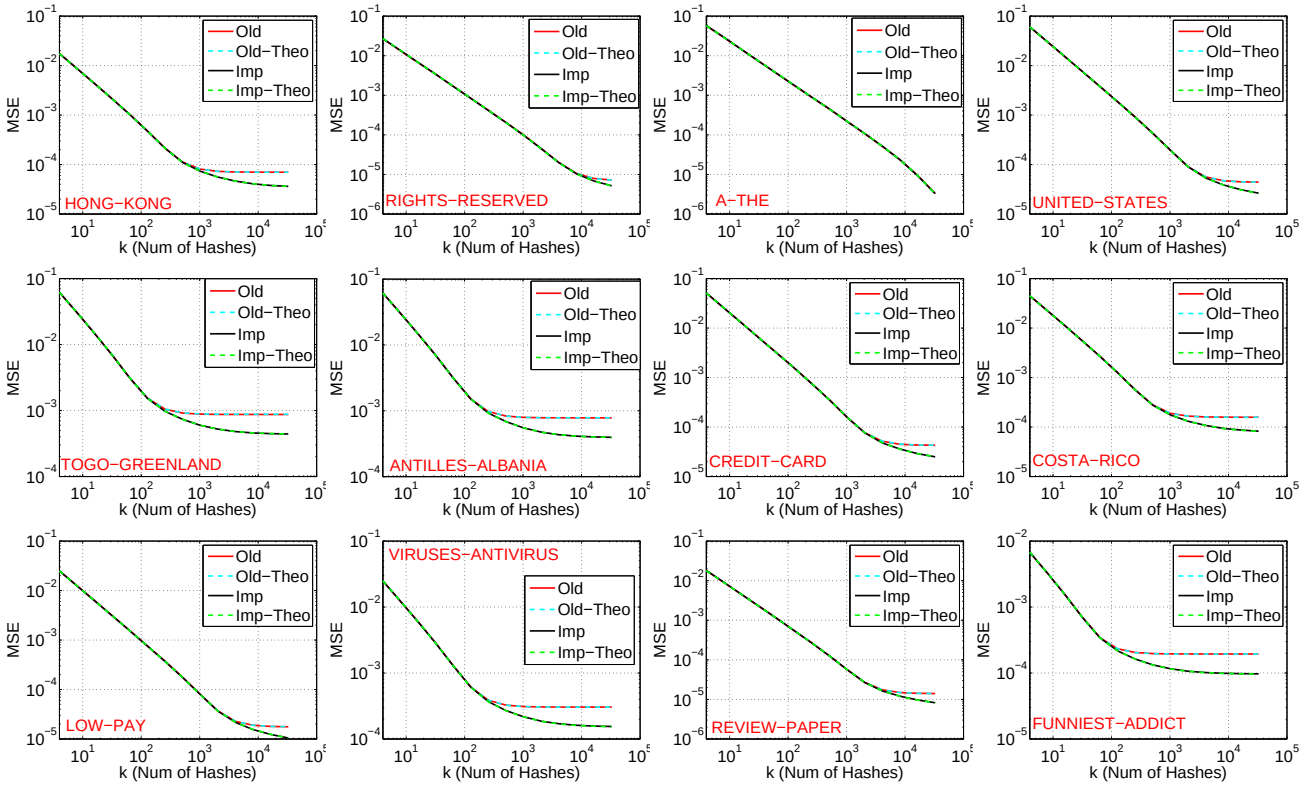

Figure 6: Mean Square Error (MSE) of the old scheme $\hat{R}$ and the improved scheme $\hat{R}^+$ along with their theoretical values on 12 word pairs (Table 1) from a web crawl dataset.

Table 1: Information of 12 pairs of word vectors. Each word stands for a set of documents in which the word is contained. For example, "A" corresponds to the set of document IDs which contained word "A".

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

Mean Square Error (MSE) 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 2 and Theorem 4. The results are summarized in Figure 6.

From the plots we can see that the theoretical and the empirical MSE values overlap in both the cases validating both Theorem 2 and Theorem 4. 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 5.

7.2 Near Neighbor Retrieval with LSH

In this experiment, we evaluate the two hashing schemes $\mathcal{H}$ and $\mathcal{H}^+$ on the standard (K, L)-parameterized LSH algorithm [14, 2] for retrieving near neighbors. Two publicly available sparse text datasets are described in Table 2.

Table 2: 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 (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

$$B_j(S) = [h_{j1}(S); h_{j2}(S); ...; h_{jK}(S)], \qquad (36)$$

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 [2] for details on the implementation of LSH. 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 training set are 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 recalls and the points retrieved per query are summarized in Figure 7.

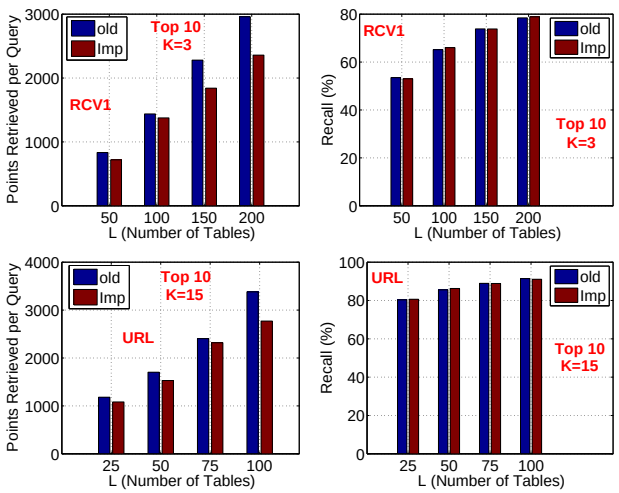

Figure 7: Average number of points scanned per query and the mean recall values of top 10 near neighbors, obtained from (K, L)-parameterized LSH algorithm, using $\mathcal{H}$ (old) and $\mathcal{H}^+$ (Imp). Both schemes achieve the same recall but $\mathcal{H}^+$ reports fewer points compared to $\mathcal{H}$. Results are averaged over 10 independent runs.

It is clear from Figure 7 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, a hash function which achieves a given recall and at the same time 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 $\mathcal{H}$.

7.3 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.(36)). 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 will be retrieved 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.

$$\mathbb{E}\big[1\{\mathcal{H}_{j1}(S_1) = \mathcal{H}_{j1}(S_2)\} \times 1\{\mathcal{H}_{j2}(S_1) = \mathcal{H}_{j2}(S_2)\}\big]$$
$$= \mathbb{E}\left[M_{j1}^N M_{j2}^N + M_{j1}^N M_{j2}^E + M_{j1}^E M_{j2}^N + M_{j1}^E M_{j2}^E\right]$$

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 $\mathbb{E}\left[M_{j1}^E M_{j2}^E\right]$ 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 $M_1^E M_2^E = 1$ is more likely. This happens in the case of sparse datasets which are common in practice.

8 Conclusion

Analysis of the densification scheme for one permutation hashing, which reduces the processing time of minwise hashes, reveals a sub-optimality in the existing procedure. We provide a simple improved procedure which adds more randomness in the current densification technique leading to a provably better scheme, especially for very sparse datasets. The improvement comes without any compromise with the computation and only requires $O(d+k)$ (linear) cost for generating k hash evaluations. We hope that our improved scheme will be adopted in practice.

Acknowledgement

Anshumali Shrivastava is a Ph.D. student partially supported by NSF (DMS0808864, III1249316) and ONR (N00014-13-1-0764). The work of Ping Li is partially supported by AFOSR (FA9550-13-1-0137), ONR (N00014-13-1-0764), and NSF (III1360971, BIGDATA1419210).

A Proofs

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 is equally likely. The proofs involve elementary combinatorial arguments of counting configurations.

A.1 Proof of Lemma 1

Given exactly m simultaneously non-empty bins, any two of them can be chosen in $m(m-1)$ ways (with ordering of i and j). Each term $M_i^N M_j^N$, for both simultaneously non-empty i and j, is 1 with probability $R\tilde{R}$ (Note, $\mathbb{E}\left(M_i^N M_j^N \middle| i \neq j, I_{emp}^i = 0, I_{emp}^j = 0\right) = R\tilde{R}$).

A.2 Proof of Lemma 2

The permutation is random and any sequence of simultaneously m non-empty and remaining $k-m$ empty bins are equal 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 simultaneously 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 simultaneously empty bin j, i.e., $I_{emp}^j = 1$, M_j^E replicates M_t^N corresponding to nearest non-empty Bin t which is towards the circular right. There are two cases we need to consider:

Case 1: $t = i$, which has probability $\frac{1}{m}$ and

$$\mathbb{E}(M_i^N M_j^E | I_{emp}^i = 0, I_{emp}^j = 1) = \mathbb{E}(M_i^N | I_{emp}^i = 0) = R$$

Case 2: $t \neq i$, which has probability $\frac{m-1}{m}$ and

$$\begin{aligned}&\mathbb{E}(M_i^N M_j^E | I_{emp}^i = 0, I_{emp}^j = 1)\\ =&\mathbb{E}(M_i^N M_t^N | t \neq i, I_{emp}^i = 0,\ I_{emp}^t = 0) = R\tilde{R}\end{aligned}$$

Thus, the value of $\mathbb{E}\left[\sum_{i\neq j} M_i^N M_j^E \middle| m\right]$ comes out to be

$$2m(k-m)\left[\frac{R}{m} + \frac{(m-1)R\tilde{R}}{m}\right]$$

which is the desired expression.

A.3 Proof of Lemma 3

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 have 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

$$(k-m)(k-m-1)\left[pR + (1-p)R\tilde{R}\right] \qquad (37)$$

because with probability $(1-p)$, it uses estimators from different simultaneous non-empty bins and in that case the $M_i^E M_j^E = 1$ with probability $R\tilde{R}$.

Consider Figure 3, where 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. 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 go to the same blank position they can be permuted. This adds extra term $(m+1)$.

If both i and j choose the 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.(37) leads to the desired expression.

A.4 Proof of Lemma 4

Similar to the proof of Lemma 3, 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 4, 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

$$p = \frac{\frac{2(m+2)}{2} + \frac{2m+4}{4}}{(m+1)(m+2)} = \frac{1.5}{m+1}.$$

Substituting p in Eq.(37) 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.

References

[1] A. Agarwal, O. Chapelle, M. Dudik, and J. Langford. A reliable effective terascale linear learning system. Technical report, arXiv:1110.4198, 2011.

[2] A. Andoni and P. Indyk. E2lsh: Exact euclidean locality sensitive hashing. Technical report, 2004.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140, 2007.

[4] A. Z. Broder. On the resemblance and containment of documents. In *the Compression and Complexity of Sequences*, pages 21–29, Positano, Italy, 1997.

[5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *STOC*, pages 327–336, Dallas, TX, 1998.

[6] G. Buehrer and K. Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, pages 95–106, Stanford, CA, 2008.

[7] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *STOC*, pages 106–112, 1977.

[8] T. Chandra, E. Ie, K. Goldman, T. L. Llinares, J. McFadden, F. Pereira, J. Redstone, T. Shaked, and Y. Singer. Sibyl: a system for large scale machine learning.

[9] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, Montreal, Quebec, Canada, 2002.

[10] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *WWW*, pages 2–11, 2005.

[11] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, pages 219–228, Paris, France, 2009.

[12] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of web pages. In *WWW*, pages 669–678, Budapest, Hungary, 2003.

[13] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, pages 284–291, 2006.

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

[15] P. Li and K. W. Church. Using sketches to estimate associations. In *HLT/EMNLP*, pages 708–715, Vancouver, BC, Canada, 2005.

[16] P. Li, K. W. Church, and T. J. Hastie. Conditional random sampling: A sketch-based sampling technique for sparse data. In *NIPS*, pages 873–880, Vancouver, BC, Canada, 2006.

[17] P. Li, A. C. König, and W. Gui. b-bit minwise hashing for estimating three-way similarities. In *Advances in Neural Information Processing Systems*, Vancouver, BC, 2010.

[18] P. Li, A. B. Owen, and C.-H. Zhang. One permutation hashing. In *NIPS*, Lake Tahoe, NV, 2012.

[19] P. Li, A. Shrivastava, J. Moore, and A. C. König. Hashing algorithms for large-scale learning. In *NIPS*, Granada, Spain, 2011.

[20] M. Mitzenmacher and S. Vadhan. Why simple hash functions work: exploiting the entropy in a data stream. In *SODA*, 2008.

[21] M. Najork, S. Gollapudi, and R. Panigrahy. Less is more: sampling the neighborhood graph makes salsa better and faster. In *WSDM*, pages 242–251, Barcelona, Spain, 2009.

[22] N. 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.

[23] A. Shrivastava and P. Li. Beyond pairwise: Provably fast algorithms for approximate k-way similarity search. In *NIPS*, Lake Tahoe, NV, 2013.

[24] A. Shrivastava and P. Li. Densifying one permutation hashing via rotation for fast near neighbor search. In *ICML*, Beijing, China, 2014.

[25] S. Tong. Lessons learned developing a practical large scale machine learning system. http://googleresearch.blogspot.com/2010/04/lessons-learned-developing-practical.html, 2008.

[26] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.