b-Bit Minwise Hashing

Ping Li[*]
Department of Statistical Science
Faculty of Computing and Information Science
Cornell University, Ithaca, NY 14853
pingli@cornell.edu

Arnd Christian König
Microsoft Research
Microsoft Corporation
Redmond, WA 98052
chrisko@microsoft.com

ABSTRACT

This paper establishes the theoretical framework of ***b*-bit minwise hashing**. The original *minwise hashing* method has become a standard technique for estimating set similarity (e.g., *resemblance*) with applications in information retrieval, data management, computational advertising, etc.

By only storing b bits of each hashed value (e.g., $b = 1$ or 2), we gain substantial advantages in terms of storage space. We prove the basic theoretical results and provide an unbiased estimator of the resemblance for any b. We demonstrate that, even in the least favorable scenario, using $b = 1$ may reduce the storage space at least by a factor of 21.3 (or 10.7) compared to $b = 64$ (or $b = 32$), if one is interested in resemblance ≥ 0.5.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Performance, Theory

1. INTRODUCTION

Computing the size of set intersections is a fundamental problem in information retrieval, databases, and machine learning. Given two sets, S_1 and S_2, where

$$S_1,\ S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\},$$

a basic task is to compute the joint size $a = |S_1 \cap S_2|$, which measures the (un-normalized) similarity between S_1 and S_2. The ***resemblance***, denoted by R, is a normalized similarity measure:

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

In large datasets encountered in information retrieval and databases, efficiently computing the joint sizes is often highly challenging [3,18]. Detecting duplicate web pages is a classical example [4,6].

Typically, each Web document can be processed as "a bag of shingles," where a shingle consists of w contiguous words in a document. Here w is a tuning parameter and was set to be $w = 5$ in several studies [4, 6, 12]. Clearly, the total number of possible shingles is huge. Considering merely 10^5 unique English words, the total number of possible 5-shingles should be $D = (10^5)^5 = O(10^{25})$. Prior studies used $D = 2^{64}$ [12] and $D = 2^{40}$ [4,6].

1.1 Minwise Hashing

In their seminal work, Broder and his colleagues developed *minwise hashing* and successfully applied the technique to duplicate Web page removal [4,6]. Since then, there have been considerable theoretical and methodological developments [5,8,19,21–23,26].

As a general technique for estimating set similarity, *minwise hashing* has been applied to a wide range of applications, for example, content matching for online advertising [30], detection of large-scale redundancy in enterprise file systems [14], syntactic similarity algorithms for enterprise information management [27], compressing social networks [9], advertising diversification [17], community extraction and classification in the Web graph [11], graph sampling [29], wireless sensor networks [25], Web spam [24,33], Web graph compression [7], and text reuse in the Web [2].

Here, we give a brief introduction to this algorithm. Suppose a random permutation π is performed on Ω, i.e.,

$$\pi : \ \Omega \longrightarrow \Omega, \qquad \text{where} \quad \Omega = \{0, 1, ..., D-1\}.$$

An elementary probability argument shows that

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

After k minwise independent permutations, π_1, π_2, ..., π_k, one can estimate R without bias, as a binomial:

$$\hat{R}_M = \frac{1}{k}\sum_{j=1}^{k} 1\{\min(\pi_j(S_1)) = \min(\pi_j(S_2))\}, \tag{2}$$

$$\text{Var}\left(\hat{R}_M\right) = \frac{1}{k}R(1-R). \tag{3}$$

Throughout the paper, we frequently use the terms "**sample**" and "**sample size**" (i.e., k). In *minwise hashing*, a sample is a hashed value, $\min(\pi_j(S_i))$, which may require e.g., 64 bits to store [12].

1.2 Our Main Contributions

In this paper, we establish a unified theoretical framework for ***b*-bit minwise hashing**. Instead of using $b = 64$ bits [12] or 40 bits [4, 6], our theoretical results suggest using as few as $b = 1$ or $b = 2$ bits can yield significant improvements.

In b-bit minwise hashing, a **sample** consists of b bits only, as opposed to e.g., 64 bits in the original minwise hashing. Intuitively, using fewer bits per sample will increase the estimation variance, compared to (3), at the same **sample size** k. Thus, we will have to increase k to maintain the same accuracy. Interestingly, our theoretical results will demonstrate that, when resemblance is not too small (e.g., $R \geq 0.5$, the threshold used in [4, 6]), we do not have to increase k much. This means our proposed b-bit minwise hashing can be used to improve estimation accuracy and significantly reduce storage requirements at the same time.

For example, when $b = 1$ and $R = 0.5$, the estimation variance will increase at most by a factor of 3. In this case, in order not to lose accuracy, we have to increase the sample size by a factor of

[*]Supported by Microsoft, NSF-DMS and ONR-YIP.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

3. If we originally stored each hashed value using 64 bits [12], the improvement by using $b = 1$ will be $64/3 = 21.3$.

Algorithm 1 illustrates the procedure of b-bit minwise hashing, based on the theoretical results in Sec. 2.

Algorithm 1 The b-bit minwise hashing algorithm, applied to estimating pairwise resemblances in a collection of N sets.

```
Input: Sets $S_n \in \Omega = \{0, 1, ..., D-1\}$, $n = 1$ to $N$.
Pre-processing:
1): Generate $k$ random permutations $\pi_j : \Omega \rightarrow \Omega$, $j = 1$ to $k$.
2): For each set $S_n$ and each permutation $\pi_j$, store the lowest $b$ bits of $\min(\pi_j(S_n))$, denoted by $e_{n,i,j}$, $i = 1$ to $b$.
Estimation: (Use two sets $S_1$ and $S_2$ as an example.)
1): Compute $\hat{E}_b = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{i=1}^{b} 1\{e_{1,i,\pi_j} = e_{2,i,\pi_j}\} = 1\right\}$.
2): Estimate the resemblance by $\hat{R}_b = \frac{\hat{E}_b - C_{1,b}}{1 - C_{2,b}}$, where $C_{1,b}$ and $C_{2,b}$ are from Theorem 1 in Sec. 2.
```

1.3 Comparisons with LSH Algorithms

Locality Sensitive Hashing (LSH) [8,20] is a set of techniques for performing approximate search in high dimensions. In the context of estimating set intersections, there exist LSH families for estimating the *resemblance*, the *arccosine* and the *Hamming distance* [1].

In [8, 16], the authors describe LSH hashing schemes that map objects to $\{0, 1\}$ (i.e., 1-bit schemes). The algorithms for the construction, however, are problem specific. Two discovered 1-bit schemes are the *sign random projections* (also known as *simhash*) [8] and the *Hamming distance LSH* algorithm proposed by [20].

Our b-bit minwise hashing proposes a new construction, which maps objects to $\{0, 1, ..., 2^b - 1\}$ instead of just $\{0, 1\}$. While our major focus is to compare with the original minwise hashing, we also conduct comparisons with the other two known 1-bit schemes.

1.3.1 Sign Random Projections

The method of sign (1-bit) random projections estimates the *arccosine*, which is $\cos^{-1}\left(\frac{a}{\sqrt{f_1 f_2}}\right)$, using our notation for sets S_1 and S_2. A separate technical report is devoted to comparing b-bit minwise hashing with sign (1-bit) random projections. See `www.stat.cornell.edu/~li/hashing/RP_minwise.pdf`. That report demonstrates that, unless the similarity level is very low, b-bit minwise hashing outperforms sign random projections.

The method of sign random projections has received significant attention in the context of duplicate detection. According to [28], a great advantage of *simhash* over *minwise hashing* is the smaller size of the fingerprints required for duplicate detection. The space-reduction of b-bit *minwise hashing* overcomes this issue.

1.3.2 The Hamming Distance LSH Algorithm

Sec. 4 will compare b-bit minwise hashing with the *Hamming distance LSH* algorithm developed in [20] (and surveyed in [1]):

- When the *Hamming distance LSH* algorithm is implemented naively, to achieve the same level of accuracy, its required storage space will be many magnitudes larger than that of b-bit minwise hashing in sparse data (i.e., $|S_i|/D$ is small).
- If we only store the non-zero locations in the *Hamming distance LSH* algorithm, then its required storage space will be about one magnitude larger (e.g., 10 to 30 times).

2. THE FUNDAMENTAL RESULTS

Consider two sets, S_1 and S_2,

$$S_1, S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\},$$
$$f_1 = |S_1|,\ f_2 = |S_2|,\ a = |S_1 \cap S_2|$$

Apply a random permutation π on S_1 and S_2: $\pi : \Omega \longrightarrow \Omega$. Define the minimum values under π to be z_1 and z_2:

$$z_1 = \min(\pi(S_1)), \qquad z_2 = \min(\pi(S_2)).$$

Define $e_{1,i}$ = ith lowest bit of z_1, and $e_{2,i}$ = ith lowest bit of z_2. Theorem 1 derives the main probability formula.

THEOREM 1. *Assume D is large.*

$$E_b = \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1\right) = C_{1,b} + (1 - C_{2,b})R$$

where (4)

$$r_1 = \frac{f_1}{D}, \qquad r_2 = \frac{f_2}{D},$$

$$C_{1,b} = A_{1,b}\frac{r_2}{r_1 + r_2} + A_{2,b}\frac{r_1}{r_1 + r_2}, \tag{5}$$

$$C_{2,b} = A_{1,b}\frac{r_1}{r_1 + r_2} + A_{2,b}\frac{r_2}{r_1 + r_2}, \tag{6}$$

$$A_{1,b} = \frac{r_1[1 - r_1]^{2^b - 1}}{1 - [1 - r_1]^{2^b}}, \qquad A_{2,b} = \frac{r_2[1 - r_2]^{2^b - 1}}{1 - [1 - r_2]^{2^b}}. \tag{7}$$

For a fixed r_j (where $j \in \{1, 2\}$), $A_{j,b}$ is a monotonically decreasing function of $b = 1, 2, 3, ...$

For a fixed b, $A_{j,b}$ is a monotonically decreasing function of $r_j \in [0, 1]$, reaching a limit:

$$\lim_{r_j \rightarrow 0} A_{j,b} = \frac{1}{2^b}. \tag{8}$$

***Proof**: See Appendix A.*□

Theorem 1 says that, for a given b, the desired probability (4) is determined by R and the ratios, $r_1 = \frac{f_1}{D}$ and $r_2 = \frac{f_2}{D}$. The only assumption needed in the proof of Theorem 1 is that D should be large, which is always satisfied in practice.

$A_{j,b}$ ($j \in \{1, 2\}$) is a decreasing function of r_j and $A_{j,b} \leq \frac{1}{2^b}$. As b increases, $A_{j,b}$ converges to zero very quickly. In fact, when $b \geq 32$, one can essentially view $A_{j,b} = 0$.

2.1 An Intuitive (Heuristic) Explanation

A simple heuristic argument may provide a more intuitive explanation of Theorem 1. Consider $b = 1$. One might expect that

$$\begin{aligned}\mathbf{Pr}(e_{1,1} = e_{2,1}) =& \mathbf{Pr}(e_{1,1} = e_{2,1} | z_1 = z_2)\mathbf{Pr}(z_1 = z_2) \\ &+ \mathbf{Pr}(e_{1,1} = e_{2,1} | z_1 \neq z_2)\mathbf{Pr}(z_1 \neq z_2) \\ \overset{??}{\approx}& R + \frac{1}{2}(1 - R) = \frac{1 + R}{2},\end{aligned}$$

because when z_1 and z_2 are not equal, the chance that their last bits are equal "may be" approximately $\frac{1}{2}$. This heuristic argument is actually consistent with Theorem 1 when $r_1, r_2 \rightarrow 0$. According to (8), as $r_1, r_2 \rightarrow 0$, we have $A_{1,1}, A_{2,1} \rightarrow \frac{1}{2}$, and $C_{1,1}, C_{2,1} \rightarrow \frac{1}{2}$ also; and hence the probability (4) approaches $\frac{1+R}{2}$.

In practice, when a very accurate estimate is not necessary, one might actually use this approximate formula to simplify the estimator. The errors, however, could be quite noticeable when r_1, r_2 are not negligible; see Sec. 5.2.

2.2 The Unbiased Estimator

Theorem 1 suggests an unbiased estimator $\hat{R}_b$ for R:

$$\hat{R}_b = \frac{\hat{E}_b - C_{1,b}}{1 - C_{2,b}}, \tag{9}$$

$$\hat{E}_b = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{i=1}^{b} 1\{e_{1,i,\pi_j} = e_{2,i,\pi_j}\} = 1\right\}, \tag{10}$$

where e_{1,i,π_j} (e_{2,i,π_j}) denotes the ith lowest bit of z_1 (z_2), under the permutation π_j. Following property of binomial distribution,

$$\text{Var}\left(\hat{R}_b\right) = \frac{\text{Var}\left(\hat{E}_b\right)}{\left[1 - C_{2,b}\right]^2} = \frac{1}{k}\frac{E_b(1 - E_b)}{\left[1 - C_{2,b}\right]^2}$$
$$= \frac{1}{k}\frac{\left[C_{1,b} + (1 - C_{2,b})R\right]\left[1 - C_{1,b} - (1 - C_{2,b})R\right]}{\left[1 - C_{2,b}\right]^2} \quad (11)$$

For large b, $\text{Var}\left(\hat{R}_b\right)$ converges to the variance of $\hat{R}_M$, the estimator for the original minwise hashing:

$$\lim_{b\rightarrow\infty} \text{Var}\left(\hat{R}_b\right) = \frac{R(1 - R)}{k} = \text{Var}\left(\hat{R}_M\right).$$

In fact, when $b \geq 32$, $\text{Var}\left(\hat{R}_b\right)$ and $\text{Var}\left(\hat{R}_M\right)$ are numerically indistinguishable for practical purposes.

2.3 The Variance-Space Trade-off

As we decrease b, the space needed for storing each "sample" will be smaller; the estimation variance (11) at the same sample size k, however, will increase. This variance-space trade-off can be precisely quantified by the ***storage factor*** $B(b; R, r_1, r_2)$:

$$B(b; R, r_1, r_2) = b \times \text{Var}\left(\hat{R}_b\right) \times k$$
$$= \frac{b\left[C_{1,b} + (1 - C_{2,b})R\right]\left[1 - C_{1,b} - (1 - C_{2,b})R\right]}{\left[1 - C_{2,b}\right]^2}. \quad (12)$$

Lower $B(b)$ is better. The ratio, $\frac{B(b_1;R,r_1,r_2)}{B(b_2;R,r_1,r_2)}$, measures the improvement of using $b = b_2$ (e.g., $b_2 = 1$) over using $b = b_1$ (e.g., $b_1 = 64$). Some algebra yields the following Theorem.

THEOREM 2. *If $r_1 = r_2$ and $b_1 > b_2$, then*

$$\frac{B(b_1; R, r_1, r_2)}{B(b_2; R, r_1, r_2)} = \frac{b_1}{b_2}\frac{A_{1,b_1}(1 - R) + R}{A_{1,b_2}(1 - R) + R}\frac{1 - A_{1,b_2}}{1 - A_{1,b_1}}, \quad (13)$$

is a monotonically increasing function of $R \in [0, 1]$.

If $R \rightarrow 1$ (which implies $r_1 \rightarrow r_2$), then

$$\frac{B(b_1; R, r_1, r_2)}{B(b_2; R, r_1, r_2)} \rightarrow \frac{b_1}{b_2}\frac{1 - A_{1,b_2}}{1 - A_{1,b_1}}. \quad (14)$$

If $r_1 = r_2$, $b_2 = 1$, $b_1 \geq 32$ (hence we treat $A_{1,b} = 0$), then

$$\frac{B(b_1; R, r_1, r_2)}{B(1; R, r_1, r_2)} = b_1 \frac{R}{R + 1 - r_1} \quad (15)$$

Proof: *We omit the proof due to its simplicity.*□

Suppose the original minwise hashing used 64 bits to store each sample, then the maximum improvement of b-bit minwise hashing would be 64-fold, attained when $r_1 = r_2 = 1$ and $R = 1$, according to (15). In the least favorable situation, i.e., $r_1, r_2 \rightarrow 0$, the improvement will still be $\frac{64}{3} = 21.3$-fold when $R = 0.5$.

Fig. 1 plots $\frac{B(64)}{B(b)}$, to directly visualize the relative improvements, which are consistent with what Theorem 2 predicts. The plots show that, when R is very large (which is the case in many practical applications), it is always good to use $b = 1$. However, when R is small, using larger b may be better. The cut-off point depends on r_1, r_2, R. For example, when $r_1 = r_2$ and both are small, it would be better to use $b = 2$ than $b = 1$ if $R < 0.4$, as shown in Fig. 1.

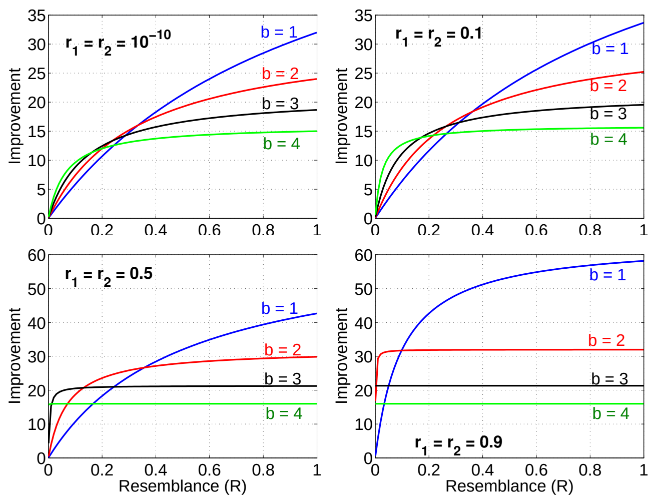


Figure 1: $\frac{B(64)}{B(b)}$, the relative storage improvement of using $b = 1, 2, 3, 4$ bits, compared to using 64 bits. $B(b)$ is defined in (12).

3. EXPERIMENTS

Experiment 1 is a sanity check, to verify: (A) our proposed estimator $\hat{R}_b$ in (9), is indeed unbiased; and (B) its variance follows the prediction by our formula in (11).

Experiment 2 is a duplicate detection task using a Microsoft proprietary collection of 1,000,000 news articles.

Experiment 3 is another duplicate detection task using 300,000 UCI NYTimes news articles.

3.1 Experiment 1

The data, extracted from Microsoft Web crawls, consists of 10 pairs of sets (i.e., total 20 words). Each set consists of the document IDs which contain the word at least once. Thus, this experiment is for estimating word associations.

Table 1: Ten pairs of words used in Experiment 1. For example, "KONG" and "HONG" correspond to the two sets of document IDs which contained word "KONG" and word "HONG" respectively.

Word 1	Word 2	r_1	r_2	R	$\frac{B(32)}{B(1)}$	$\frac{B(64)}{B(1)}$
KONG	HONG	0.0145	0.0143	0.925	15.5	31.0
RIGHTS	RESERVED	0.187	0.172	0.877	16.6	32.2
OF	AND	0.570	0.554	0.771	20.4	40.8
GAMBIA	KIRIBATI	0.0031	0.0028	0.712	13.3	26.6
UNITED	STATES	0.062	0.061	0.591	12.4	24.8
SAN	FRANCISCO	0.049	0.025	0.476	10.7	21.4
CREDIT	CARD	0.046	0.041	0.285	7.3	14.6
TIME	JOB	0.189	0.05	0.128	4.3	8.6
LOW	PAY	0.045	0.043	0.112	3.4	6.8
A	TEST	0.596	0.035	0.052	3.1	6.2

Table 1 summarizes the data and also provides the theoretical improvements, $\frac{B(32)}{B(1)}$ and $\frac{B(64)}{B(1)}$. The words were selected to include highly frequent word pairs (e.g., "OF-AND"), highly rare word pairs (e.g., "GAMBIA-KIRIBATI"), highly unbalanced pairs (e.g., "A-Test"), highly similar pairs (e.g, "KONG-HONG"), as well as word pairs that are not quite similar (e.g., "LOW-PAY").

We estimate the resemblance using the original minwise hashing estimator $\hat{R}_M$ and the b-bit estimator $\hat{R}_b$ ($b = 1, 2, 3$).

3.1.1 Validating the Unbiasedness

Figure 2 presents the estimation biases for the selected 2 word pairs. Theoretically, both estimators, $\hat{R}_M$ and $\hat{R}_b$, are unbiased (i.e., the y-axis in Figure 2 should be zero, after an infinite number of repetitions). Figure 2 verifies this fact because the empirical biases are all very small and no systematic biases can be observed.

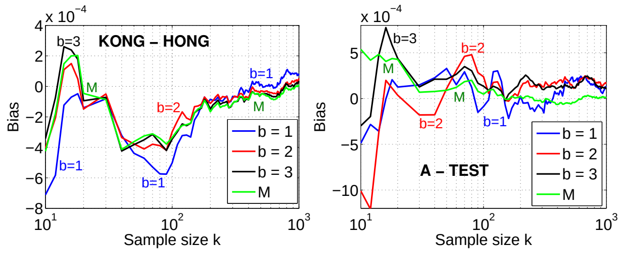


Figure 2: Empirical biases from 25000 simulations at each sample size k. "M" denotes the original minwise hashing.

3.1.2 *Validating the Variance Formula*

Figure 3 plots the empirical mean square errors (MSE = variance + bias^2) in solid lines, and the theoretical variances (11) in dashed lines, for 6 word pairs (instead of 10 pairs, due to the space limit).

All dashed lines are invisible because they overlap with the corresponding solid curves. Thus, this experiment validates that the variance formula (11) is accurate and $\hat{R}_b$ is indeed unbiased (otherwise, MSE will differ from the variance).

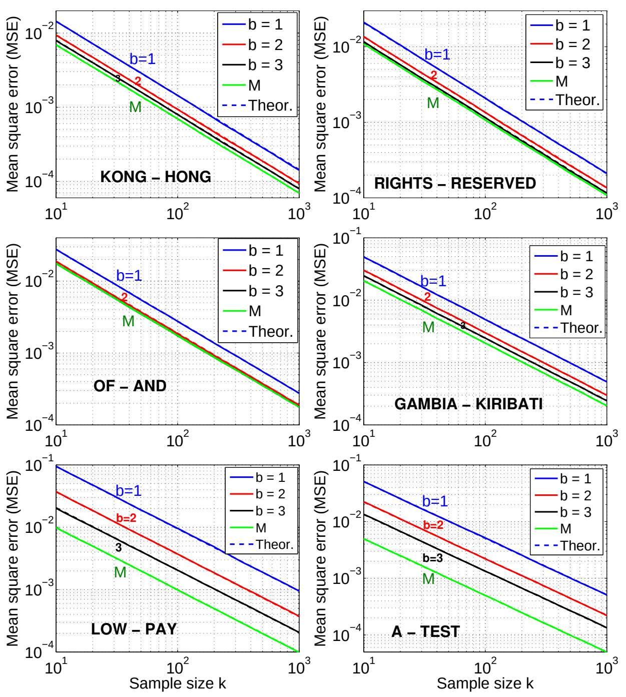


Figure 3: Mean square errors (MSEs). "M" denotes the original minwise hashing. "Theor." denotes the theoretical variances of Var($\hat{R}_b$)(11) and Var($\hat{R}_M$)(3). The dashed curves, however, are invisible because the empirical MSEs overlapped the theoretical variances. At the same k, Var($\hat{R}_1$) $>$ Var($\hat{R}_2$) $>$ Var($\hat{R}_3$) $>$ Var($\hat{R}_M$). However, $\hat{R}_1$ ($\hat{R}_2$) only requires 1 bit (2 bits) per sample, while $\hat{R}_M$ requires 32 or 64 bits.

3.2 Experiment 2: Microsoft News Data

To illustrate the improvements by the use of b-bit minwise hashing on a real-life application, we conducted a duplicate detection experiment using a corpus of 10^6 news documents. The dataset was crawled as part of the BLEWS project at Microsoft [15]. We computed pairwise resemblances for all documents and retrieved documents pairs with resemblance R larger than a threshold $\mathbf{R_0}$.

We estimate the resemblances using $\hat{R}_b$ with $b = 1, 2, 4$ bits, and the original minwise hashing (using 32 bits). Figure 4 presents the precision & recall curves. The recall values (bottom two panels in Figure 4) are all very high and do not differentiate the estimators.

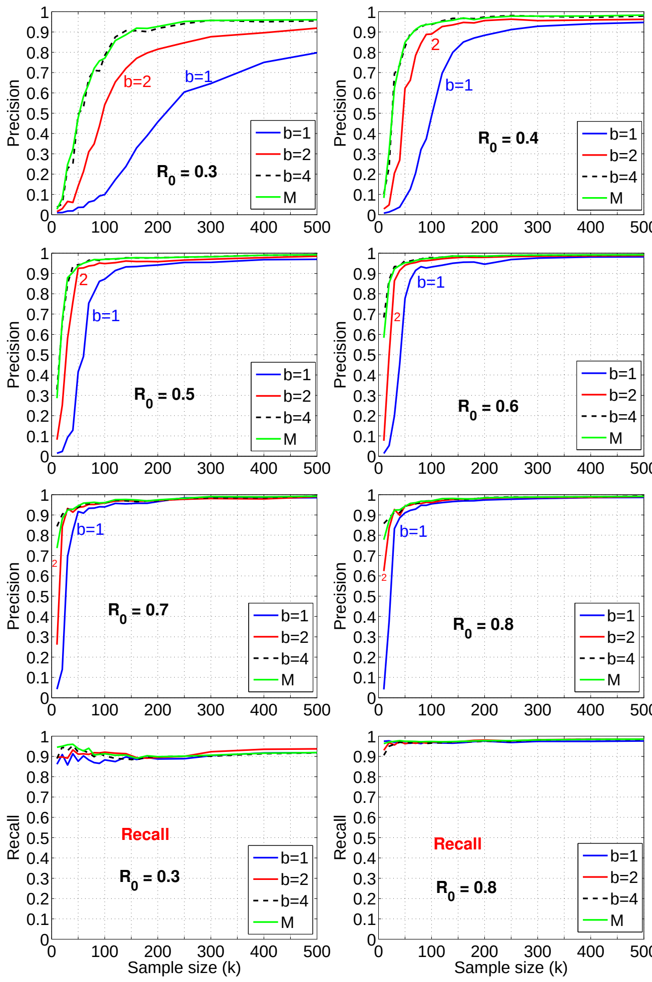


Figure 4: Microsoft collection of news data. The task is to retrieve news article pairs with resemblance $R \geq R_0$. The recall curves (bottom two panels) indicate all estimators are equally good (in recalls). The precision curves are more interesting for differentiating estimators. For example, when $R_0 = 0.4$ (top right panel), in order to achieve a precision = 0.80, the estimators $\hat{R}_M$, $\hat{R}_4$, $\hat{R}_2$, and $\hat{R}_1$ require $k = 50, 50, 75, 145$ samples, respectively, indicating $\hat{R}_4$, $\hat{R}_2$, and $\hat{R}_1$ respectively improve $\hat{R}_M$ by 8-fold, 10.7-fold, and 11-fold.

The precision curves for $\hat{R}_4$ (using 4 bits per sample) and $\hat{R}_M$ (using 32 bits per sample) are almost indistinguishable, suggesting a 8-fold improvement in space using $b = 4$.

When using $b = 1$ or 2, the space improvements are normally around 10-fold to 20-fold, compared to $\hat{R}_M$, especially for achieving high precisions (e.g., ≥ 0.9). This experiment again confirms the significant improvement of the b-bit minwise hashing using $b = 1$ (or 2). Table 2 summarizes the relative improvements.

In this experiment, $\hat{R}_M$ only used 32 bits per sample. For even larger applications, however, 64 bits per sample may be necessary [12]; and the improvements of $\hat{R}_b$ will be even more significant.

Note that in the context of (Web) document duplicate detection, in addition to shingling, a number of specialized hash-signatures have been proposed, which leverage properties of natural-language text (such as the placement of stopwords [31]). However, our approach is not aimed at any specific type of data, but is a general, domain-independent technique. Also, to the extent that other approaches rely on minwise hashing for signature computation, these may be combined with our techniques.

Table 2: Relative improvement (in space) of $\hat{R}_b$ (using b bits per sample) over $\hat{R}_M$ (32 bits per sample). For precision = 0.9, 0.95, we find the required sample sizes (from Figure 4) for $\hat{R}_M$ and $\hat{R}_b$ and use them to estimate the required storage in bits. The values in the table are the ratios of the storage costs. The improvements are consistent with the theoretical predictions in Figure 1.

R_0	Precision = 0.9			Precision = 0.95		
	$b = 1$	2	4	$b = 1$	2	4
0.3	—	5.7	8.8	—	—	7.1
0.4	9.2	10.0	8.3	—	10.0	8.2
0.5	10.8	12.7	8.4	8.2	10.1	7.7
0.6	12.9	11.7	8.6	10.5	12.4	8.5
0.7	16.0	14.8	9.6	15.4	12.7	7.6
0.8	17.4	10.3	8.0	18.7	14.2	7.7
0.9	16.6	14.0	10.7	23.0	17.6	9.7

3.3 Experiment 3: UCI NYTimes Data

We conducted another duplicate detection experiment on a public (UCI) collection of 300,000 NYTimes articles. The purpose is to ensure that our experiment will be repeatable by those who can not access the proprietary data in Experiment 2.

Figure 5 presents the precision curves for representative threshold R_0's. The recall curves are not shown because they could not differentiate estimators, just like in Experiment 1. The curves confirm again that using $b = 1$ or $b = 2$ bits, $\hat{R}_b$ could improve the original minwise hashing (using 32 bits per sample) by a factor of 10 or more. The curves for $\hat{R}_b$ with $b = 4$ almost always overlap with the curves for $\hat{R}_M$, verifying an expected 8-fold improvement.

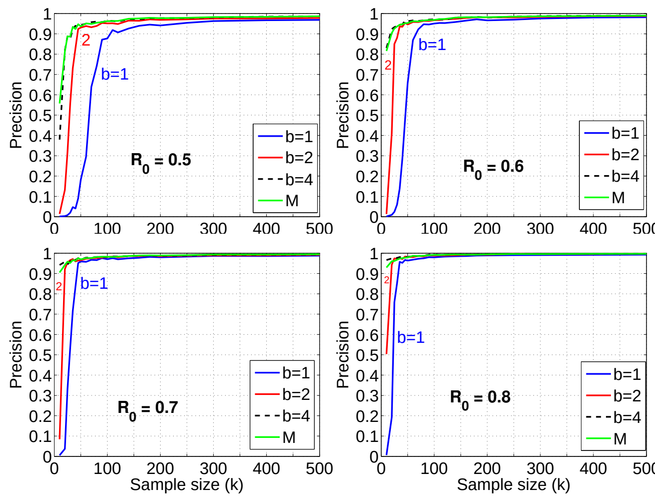


Figure 5: UCI collection of NYTimes data. The task is to retrieve news article pairs with resemblance $R \geq R_0$.

4. COMPARISONS WITH THE HAMMING DISTANCE LSH ALGORITHM

The *Hamming distance LSH* algorithm proposed in [20] is an influential 1-bit LSH scheme. In this algorithm, a set S_i, is mapped into a D-dimensional binary vector, y_i:

$$y_{it} = 1, \text{ if } t \in S_i; \quad y_{it} = 0, \text{ otherwise.}$$

k coordinates are randomly sampled from $\Omega = \{0, 1, ..., D-1\}$. We denote the samples of y_i by h_i, where $h_i = \{h_{ij}, j = 1 \text{ to } k\}$ is a k-dimensional vector. These samples will be used to estimate the Hamming distance H (using S_1, S_2 as an example):

$$H = \sum_{i=0}^{D-1} [y_{1i} \neq y_{2i}] = |S_1 \cup S_2| - |S_1 \cap S_2| = f_1 + f_2 - 2a.$$

Using the samples h_1 and h_2, an unbiased estimator of H is simply

$$\hat{H} = \frac{D}{k} \sum_{j=1}^{k} [h_{1j} \neq h_{2j}], \tag{16}$$

whose variance would be

$$\begin{aligned}\text{Var}\left(\hat{H}\right) =& \frac{D^2}{k^2} k \left[E\left([h_{1j} \neq h_{2j}]^2\right) - E^2\left([h_{1j} \neq h_{2j}]\right)\right] \\ =& \frac{D^2}{k}\left[\frac{\sum_{i=0}^{D-1}[y_{1i} \neq y_{2i}]^2}{D} - \left(\frac{\sum_{i=0}^{D-1}[y_{1i} \neq y_{2i}]}{D}\right)^2\right] \\ =& \frac{D^2}{k}\left[\frac{H}{D} - \frac{H^2}{D^2}\right].\end{aligned} \tag{17}$$

The above analysis assumes $k \ll D$ (which is satisfied in practice); otherwise one should multiply the Var $\left(\hat{H}\right)$ in (17) by $\frac{D-k}{D-1}$, the "finite sample correction factor." It would be interesting to compare $\hat{H}$ with b-bit minwise hashing. In order to estimate H, we need to convert the resemblance estimator $\hat{R}_b$ (9) to $\hat{H}_b$:

$$\hat{H}_b = f_1 + f_2 - 2\frac{\hat{R}_b}{1+\hat{R}_b}(f_1 + f_2) = \frac{1-\hat{R}_b}{1+\hat{R}_b}(f_1 + f_2). \tag{18}$$

The variance of $\hat{H}_b$ can be computed from Var $\left(\hat{R}_b\right)$ (11) using the "delta method" in statistics (note that $\left[\frac{1-x}{1+x}\right]' = \frac{-2}{(1+x)^2}$):

$$\begin{aligned}\text{Var}\left(\hat{H}_b\right) =& \text{Var}\left(\hat{R}_b\right)(f_1+f_2)^2\left(\frac{-2}{(1+R)^2}\right)^2 + O\left(\frac{1}{k^2}\right) \\ =& \text{Var}\left(\hat{R}_b\right)\frac{4(r_1+r_2)^2}{(1+R)^4}D^2 + O\left(\frac{1}{k^2}\right).\end{aligned} \tag{19}$$

Recall $r_i = f_i/D$. To verify the variances in (17) and (19), we conduct experiments using the same data as in Experiment 1. This time, we estimate H instead of R, using both $\hat{H}$ (16) and $\hat{H}_b$ (18).

Figure 6 reports the mean square errors, together with the theoretical variances (17) and (19). We can see that the theoretical variance formulas are accurate. When the data is not dense, the estimator $\hat{H}_b$ (18) given by b-bit minwise hashing is much more accurate than the estimator $\hat{H}$ (16). However, when the data is dense (e.g., "OF-AND"), $\hat{H}$ could still outperform $\hat{H}_b$.

We now compare the actual storage needed by $\hat{H}_b$ and $\hat{H}$. We define the following two ratios to make fair comparisons:

$$W_b = \frac{\text{Var}\left(\hat{H}\right) \times k}{\text{Var}\left(\hat{H}_b\right) \times bk}, \quad G_b = \frac{\text{Var}\left(\hat{H}\right) \times \frac{r_1+r_2}{2}64k}{\text{Var}\left(\hat{H}_b\right) \times bk}. \tag{20}$$

W_b and G_b are defined in the same spirit as the ratio of the *storage factors* introduced in Sec. 2.3. Recall each sample of b-bit minwise hashing requires b bits (i.e., bk bits per set). If we assume each sample in the *Hamming distance* LSH requires 1 bit, then W_b in (20) is a fair indicator and $W_b > 1$ means $\hat{H}_b$ outperforms $\hat{H}$.

However, as can be verified in Fig. 6 and Fig 7, when r_1 and r_2 are small (which is usually the case in practice), W_b tends to be very

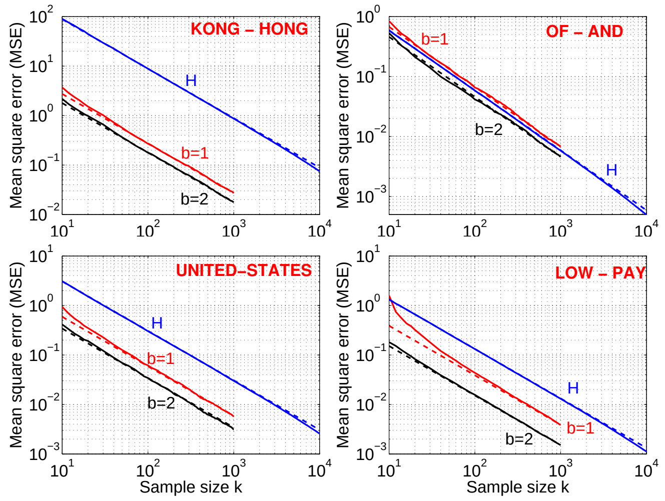


Figure 6: MSEs (normalized by H^2), for comparing $\hat{H}$ (16) with $\hat{H}_b$ (18). In each panel, three solid curves stand for $\hat{H}$ (labeled by "H"), $\hat{H}_1$ (by "b=1"), and $\hat{H}_2$ (by "b=2"), respectively. The dashed lines are the corresponding theoretical variances (17) and (19), which are largely overlapped by the solid lines. When the sample size k is not large, the empirical MSEs of $\hat{H}_b$ deviate from the theoretical variances, due to the bias caused by the nonlinear transformation of $\hat{H}_b$ from $\hat{R}_b$ in (18).

large, indicating a highly significant improvement of b-bit minwise hashing over the *Hamming distance LSH* algorithm in [20].

We consider in practice one will most likely implement the algorithm by only storing non-zero locations. In other words, for set S_i, only $r_i \times k$ locations need to be stored (each is assumed to use 64 bits). Thus, the total bits on average will be $\frac{r_1+r_2}{2}64k$ (per set). In fact, we have the following Theorem for G_b when $r_1, r_2 \rightarrow 0$.

THEOREM 3. *Consider $r_1, r_2 \rightarrow 0$, and G_b as defined in (20).*

$$\text{If } R \rightarrow 0, \quad \text{then } G_b \rightarrow \frac{8}{b}\left(2^b - 1\right). \tag{21}$$

$$\text{If } R \rightarrow 1, \quad \text{then } G_b \rightarrow \frac{64}{b}\frac{2^b - 1}{2^b}. \tag{22}$$

Proof: *We omit the proof due to its simplicity.* □

Figure 7 plots W_1 and G_1, for $r_1 = r_2 = 10^{-6}, 10^{-4}, 0.001, 0.01, 0{,}1$ (which are probably reasonable in practice), as well as $r_1 = r_2 = 0.9$ (as a sanity check). Note that, not all combinations of r_1, r_2, R are possible. For example, when $r_1 = r_2 = 1$, then R has to be 1.

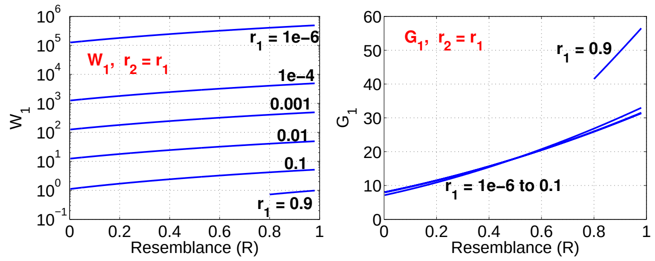


Figure 7: W_1 and G_1 as defined in (20). We consider $r_1 = 10^{-6}, 10^{-4}, 0.001, 0.01, 0.1, 0.9$. Note that not all combinations of (r_1, r_2, R) are possible. The plot for G_1 also verifies the theoretical limits proved in Theorem 3.

Figure 7 confirms our theoretical results. W_1 will be extremely large, when r_1, r_2 are small. However, when r_1 is very large (e.g., 0.9), it is possible that $W_1 < 1$, meaning that the *Hamming distance LSH* could still outperform b-bit minwise in dense data.

By only storing the non-zero locations, Figure 7 illustrates that b-bit minwise hashing will outperform the *Hamming distance LSH* algorithm, usually by a factor of 10 (for small R) to 30 (for large R and $r_1 \approx r_2$).

5. DISCUSSIONS

5.1 Computational Overhead

The previous results establish the significant reduction in storage requirements possible using b-bit minwise hashing. This section demonstrates that these also translate into significant improvements in computational overhead in the **estimation phrase**. The computational cost in the **preprocessing phrase**, however, will increase.

5.1.1 *Preprocessing Phrase*

In the preprocessing phrase, we need to generate minwise hashing functions and apply them to all the sets for creating fingerprints. This phrase is actually fairly fast [4] and is usually done off-line, incurring a one-time cost. Also, sets can be individually processed, meaning that this step is easy to parallelize.

The computation required for b-bit minwise hashes differs from the computation of traditional minwise hashes in two respects: (A) we require a larger number of (smaller-sized) samples, in turn requiring more hashing and (B) the packing of b-bit samples into 64-bit (or 32-bit) words requires additional bit-manipulation.

It turns out the overhead for (B) is small and the overall computation time scales nearly linearly with k; see Fig. 8. As we have analyzed, b-bit minwise hashing only requires increasing k by a small factor such as 3. Therefore, we consider the overhead in the preprocessing stage not to be a major issue. Also, it is important to note that b-bit minwise hashing provides the flexibility of trading storage with preprocessing time by using $b > 1$.

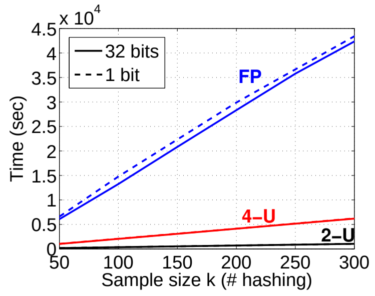


Figure 8: Running time in the preprocessing phrase on 100K news articles. 3 hashing functions were used: 2-universal hashing (labeled by "2-U"), 4-universal hashing (labeled by "4-U"), and full permutations (labeled by "FP"). Experiments with 1-bit hashing are reported in 3 dashed lines, which are only slightly higher (due to additional bit-packing) than their corresponding solid lines (the original minwise hashing using 32-bit).

The experiment in Fig. 8 was conducted on 100K articles from the BLEWS project [15]. We considered 3 hashing functions: first, 2-universal hash functions (computed using the fast universal hashing scheme described [10]); second, 4-universal hash-functions (computed using the `CWtrick` algorithm of [32]); and finally full random permutations (computed using the *Fisher-Yates shuffle* [13]).

5.1.2 *Estimation Phrase*

We have shown earlier that, when $R \geq 0.5$ and $b = 1$, we expect a storage reduction of at least a factor of 21.3, compared to using

64 bits. In the following, we will analyze how this impacts the computational overhead of the estimation.

Here, the key operation is the computation of the number of identical b-bit samples. While standard hash signatures that are multiples of 16-bit can easily be compared using a single machine instruction, efficiently computing the overlap between b-bit samples for small b is less straightforward. In the following, we will describe techniques for computing the number of identical b-bit samples when these are stored in a compact manner, meaning that individual b-bit samples $e_{1,i,j}$ and $e_{2,i,j}$, $i = 1, \ldots, b$, $j = 1, \ldots k$ are packed into arrays $A_l[1, \ldots, \frac{k \cdot b}{w}]$, $l = 1, 2$ of w-bit words. To compute the number of identical b-bit samples, we iterate through the arrays; for an each offset h, we first compute $v = A_1[h] \oplus A_2[h]$, where $\oplus$ denotes the bitwise-XOR. Subsequently, the h-th bit of v will be set if and only if the h-th bits in $A_1[h]$ and $A_2[h]$ are different. Hence, to compute the number of overlapping b-bit samples encoded in $A_1[h]$ and $A_2[h]$, we need to compute the number of b-bit blocks ending at offsets divisible by b that only contain 0s.

The case of $b = 1$ corresponds to the problem of counting the number of 0-bits in a word. We tested different methods suggested in [34] and found the fastest approach to be pre-computing an array $bits[1, \ldots, 2^{16}]$, such that $bits[t]$ corresponds to the number of 0-bits in the binary representation of t. Then we can compute the number of 0-bits in v (in case of $w = 32$) as

$$c = bits[v \ \& \ \text{0xffffu}] + bits[(v \gg 16) \ \& \ \text{0xffffu}].$$

Interestingly, we can use the same method for the cases where $b > 1$, as we only need to modify the values stored in $bits$, setting $bits[i]$ to the number of b-bit blocks that only contain 0-bits in the binary representation of i.

We evaluated this approach using a loop computing the number of identical samples in two signatures covering a total of 1.8 billion 32-bit words (using a 64-bit Intel 6600 Processor). Here, the 1-bit hashing requires 1.67x the time that the 32-bit minwise hashing requires.The results were essentially identical for $b = 2$.

Combined with the reduction in overall storage (for a given accuracy level), this means a significant speed improvement in the estimation phase: suppose in the original minwise hashing, each sample is stored using 64 bits. If we use 1-bit minwise hashing and consider $R > 0.5$, our previous analysis has shown that we could gain a storage reduction at least by a factor of 64/3 = 21.3 fold. The improvement in computational efficiency would be 21.3/1.67 = 12.8 fold, which is still significant.

5.2 Reducing Storage Overhead for r_1 and r_2

The unbiased estimator $\hat{R}_b$ (9) requires knowing $r_1 = \frac{f_1}{D}$ and $r_2 = \frac{f_1}{D}$. The storage cost could be a concern if r_1 (r_2) must be represented with a high accuracy (e.g., 64 bits).

This section illustrates that we only need to quantize r_1 and r_2 into Q levels, where $Q = 2^4$ is probably good enough and $Q = 2^8$ is more than sufficient. In other words, for each set, we only need to increase the total storage by 4 bits or 8 bits, which are negligible.

For simplicity, we carry out the analysis for $b = 1$ and $r_1 = r_2 = r$. In this case, $A_{1,1} = A_{2,1} = C_{1,1} = C_{2,1} = \frac{1-r}{2-r}$, and the correct estimator, denoted by $\hat{R}_{1,r}$ would be

$$\hat{R}_{1,r} = (2-r)\hat{E}_1 - (1-r),$$
$$Bias\left(\hat{R}_{1,r}\right) = E\left(\hat{R}_{1,r}\right) - R = 0,$$
$$\text{Var}\left(\hat{R}_{1,r}\right) = \frac{(1-r+R)(1-R)}{k}.$$

See the definition of $\hat{E}_1$ in (10). Now, suppose we only store an approximate value of r, denoted by $\tilde{r}$. The corresponding (approximate) estimator is denoted by $\hat{R}_{1,\tilde{r}}$:

$$\hat{R}_{1,\tilde{r}} = (2-\tilde{r})\hat{E}_1 - (1-\tilde{r}),$$
$$Bias\left(\hat{R}_{1,\tilde{r}}\right) = E\left(\hat{R}_{1,\tilde{r}}\right) - R = \frac{(\tilde{r}-r)(1-R)}{2-r},$$
$$\text{Var}\left(\hat{R}_{1,\tilde{r}}\right) = \frac{(1-r+R)(1-R)}{k}\frac{(2-\tilde{r})^2}{(2-r)^2}.$$

Thus, the (absolute) bias is upper bounded by $|\tilde{r}-r|$ (in the worst case, i.e., $R = 0$ and $r = 1$). Using $Q = 2^4$ levels of quantization, the bias is bounded by $1/16 = 0.0625$. In a reasonable situation, e.g., $R \geq 0.5$, the bias will be much smaller than 0.0625. Of course, if we increase the quantization levels to $Q = 2^8$, the bias ($< 1/256 = 0.0039$) will be negligible, even in the worst case.

Similarly, by examining the difference of the variances,

$$\left|\text{Var}\left(\hat{R}_{1,r}\right) - \text{Var}\left(\hat{R}_{1,\tilde{r}}\right)\right|$$
$$= \frac{|\tilde{r}-r|}{k}(1-r+R)(1-R)\frac{(4-\tilde{r}-r)}{(2-r)^2},$$

we can see that $Q = 2^8$ would be more than sufficient.

5.3 Combining Bits for Enhancing Performance

Our theoretical and empirical results have confirmed that, when the resemblance R is reasonably high, each bit per sample may contain strong information for estimating the similarity. This naturally leads to the conjecture that, when R is close to 1, one might further improve the performance by looking at a combination of multiple bits (i.e., "$b < 1$"). One simple approach is to combine two bits from two permutations using XOR ($\oplus$) operations.

Recall $e_{1,1,\pi}$ denotes the lowest bit of the hashed value under π. Theorem 1 has proved that

$$E_1 = \mathbf{Pr}\left(e_{1,1,\pi} = e_{2,1,\pi}\right) = C_{1,1} + \left(1 - C_{2,1}\right)R$$

Consider two permutations π_1 and π_2. We store

$$x_1 = e_{1,1,\pi_1} \oplus e_{1,1,\pi_2}, \qquad x_2 = e_{2,1,\pi_1} \oplus e_{2,1,\pi_2}$$

Then $x_1 = x_2$ either when $e_{1,1,\pi_1} = e_{2,1,\pi_1}$ and $e_{1,1,\pi_2} = e_{2,1,\pi_2}$, or, when $e_{1,1,\pi_1} \neq e_{2,1,\pi_1}$ and $e_{1,1,\pi_2} \neq e_{2,1,\pi_2}$. Thus

$$T = \mathbf{Pr}\left(x_1 = x_2\right) = E_1^2 + \left(1 - E_1\right)^2, \tag{23}$$

which is a quadratic equation with a solution

$$R = \frac{\sqrt{2T-1} + 1 - 2C_{1,1}}{2 - 2C_{2,1}}. \tag{24}$$

We can estimate T without bias as a binomial. The resultant estimator for R will be biased, at small sample size k, due to the nonlinearity. We will recommend the following estimator

$$\hat{R}_{1/2} = \frac{\sqrt{\max\{2\hat{T}-1, 0\}} + 1 - 2C_{1,1}}{2 - 2C_{2,1}}. \tag{25}$$

The truncation $\max\{\ .\ , 0\}$ will introduce further bias; but it is necessary and is usually a good bias-variance trade-off. We use $\hat{R}_{1/2}$ to indicate that two bits are combined into one. The asymptotic variance of $\hat{R}_{1/2}$ can be derived using the "delta method"

$$\text{Var}\left(\hat{R}_{1/2}\right) = \frac{1}{k}\frac{T(1-T)}{4(1-C_{2,1})^2(2T-1)} + O\left(\frac{1}{k^2}\right). \tag{26}$$

Note that each sample is still stored using 1 bit, despite that we use "$b = 1/2$" to denote this estimator.

Interestingly, as $R \to 1$, $\hat{R}_{1/2}$ does twice as well as $\hat{R}_1$:

$$\lim_{R\to 1} \frac{\text{Var}\left(\hat{R}_1\right)}{\text{Var}\left(\hat{R}_{1/2}\right)} = \lim_{R\to 1} \frac{2(1-2E_1)^2}{(1-E_1)^2 + E_1^2} = 2. \qquad (27)$$

(Recall, if $R = 1$, then $r_1 = r_2$, $C_{1,1} = C_{2,1}$, and $E_1 = C_{1,1} + 1 - C_{2,1} = 1$.) On the other hand, $\hat{R}_{1/2}$ may not be good when R is not too large. For example, one can numerically show that

$$\text{Var}\left(\hat{R}_1\right) < \text{Var}\left(\hat{R}_{1/2}\right), \qquad \text{if } R < 0.5774,\ r_1,\ r_2 \to 0$$

Figure 9 plots the empirical MSEs for four word pairs in Experiment 1, for $\hat{R}_{1/2}$, $\hat{R}_1$, and $\hat{R}_M$. For the highly similar pair, "KONG-HONG," $\hat{R}_{1/2}$ exhibits superior performance compared to $\hat{R}_1$. For the fairly similar pair, "OF-AND," $\hat{R}_{1/2}$ is still considerably better. For "UNITED-STATES," whose $R = 0.591$, $\hat{R}_{1/2}$ performs similarly to $\hat{R}_1$. For "LOW-PAY," whose $R = 0.112$ only, the theoretical variance of $\hat{R}_{1/2}$ is very large. However, owing to the truncation in (25) (i.e., the variance-bias trade-off), the empirical performance of $\hat{R}_{1/2}$ is not too bad.

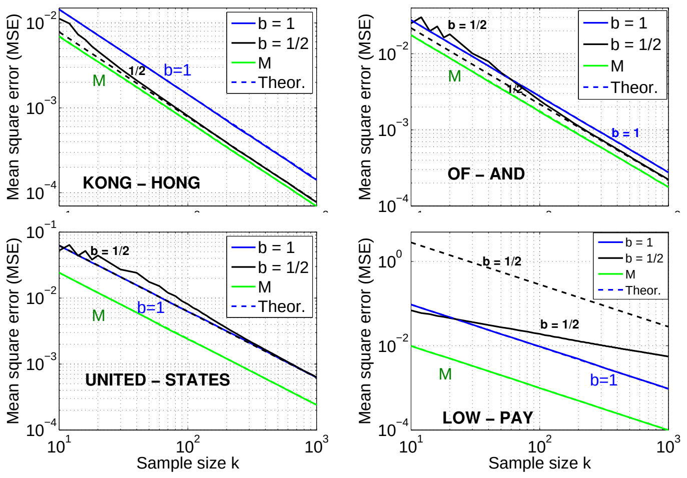


Figure 9: MSEs for comparing $\hat{R}_{1/2}$ (25) with $\hat{R}_1$ and $\hat{R}_M$. Due to the bias of $\hat{R}_{1/2}$, the theoretical variances Var $\left(\hat{R}_{1/2}\right)$, i.e., (26), deviate from the empirical MSEs when k is small.

In a summary, for applications which care about very high similarities, combining bits can reduce storage even further.

6. CONCLUSION

The ***minwise hashing*** technique has been widely used as a standard duplicate detection approach in the context of information retrieval, for efficiently computing set similarity in massive data sets. Prior studies commonly used 64 bits to store each hashed value.

This study proposes ***b-bit minwise hashing***, by only storing the lowest b bits of each hashed value. We theoretically prove that, when the similarity is reasonably high (e.g., resemblance ≥ 0.5), using $b = 1$ bit per hashed value can, even in the worst case, gain a 21.3-fold improvement in storage space, compared to storing each hashed value using 64 bits. We also discussed the idea of combining 2 bits from different hashed values, to further enhance the improvement, when the target similarity is very high.

Our proposed method is simple and requires only minimal modification to the original minwise hashing algorithm. We expect our method will be adopted in practice.

7. 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, 2009.

[2] Michael Bendersky and W. Bruce Croft. Finding text reuse on the web. In *WSDM*, pages 262–271, 2009.

[3] Sergey Brin, James Davis, and Hector Garcia-Molina. Copy detection mechanisms for digital documents. In *SIGMOD*, pages 398–409, 1995.

[4] Andrei Z. Broder. On the resemblance and containment of documents. In *Sequences*, pages 21–29, 1997.

[5] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer Systems and Sciences*, 60(3):630–659, 2000.

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

[7] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, pages 95–106, 2008.

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

[9] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.

[10] Dietzfelbinger, Martin and Hagerup, Torben and Katajainen, Jyrki and Penttonen, Martti A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.

[11] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2):1–36, 2009.

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

[13] R.A. Fisher and F. Yates. *Statistical Tables for Biological, Agricultural and Medical Research.* Oliver & Boyd, 1948.

[14] George Forman, Kave Eshghi, and Jaap Suermondt. Efficient detection of large-scale redundancy in enterprise file systems. *SIGOPS Oper. Syst. Rev.*, 43(1):84–91, 2009.

[15] Michael Gamon, Sumit Basu, Dmitriy Belenko, Danyel Fisher, Matthew Hurst, and Arnd Christian König. Blews: Using blogs to provide context for news articles. In *AAAI*, 2008.

[16] Aristides Gionis and Dimitrios Gunopulos and Nick Koudas. Efficient and Tunable Similar Set Retrieval. In *SIGMOD*, pages 247-258, 2001.

[17] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[18] Monika .R. Henzinge. Algorithmic challenges in web search engines. *Internet Mathematics*, 1(1):115–123, 2004.

[19] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithm*, 38(1):84–90, 2001.

[20] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[21] Toshiya Itoh, Yoshinori Takei, and Jun Tarui. On the sample size of k-restricted min-wise independent permutations and other k-wise distributions. In *STOC*, pages 710–718, 2003.

[22] P. Li and K. Church. A Sketch Algorithm for Estimating Two-way and Multi-way Associations *Computational Linguistics*, pages 305–354, 2007. (Preliminary results appeared in HLT/EMNLP 2005.)

[23] P. Li, K. Church and T. Hastie. One Sketch For All: Theory and Applications of Conditional Random Sampling. In *NIPS*, 2008.

[24] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *WSDM*, pages 219–230, 2008.

[25] Konstantinos Kalpakis and Shilang Tang. Collaborative data gathering in wireless sensor networks using measurement co-occurrence. *Computer Commu.*, 31(10):1979–1992, 2008.

[26] Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Algorithmica*, 55(1):113–133, 2009.

[27] Ludmila, Kave Eshghi, Charles B. Morrey III, Joseph Tucek, and Alistair Veitch. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, pages 1087–1096, 2009.

[28] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting Near-Duplicates for Web-Crawling. In *WWW*, 2007.

[29] Marc Najork, Sreenivas Gollapudi, and Rina Panigrahy. Less is more: sampling the neighborhood graph makes salsa better and faster. In *WSDM*, pages 242–251, 2009.

[30] Sandeep Pandey, Andrei Broder, Flavio Chierichetti, Vanja Josifovski, Ravi Kumar, and Sergei Vassilvitskii. Nearest-neighbor caching for content-match applications. In *WWW*, 441–450, 2009.

[31] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR*, pages 563–570, 2008.

[32] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *SODA*, 2004.

[33] Tanguy Urvoy, Emmanuel Chauveau, Pascal Filoche, and Thomas Lavergne. Tracking web spam with html style similarities. *ACM Trans. Web*, 2(1):1–28, 2008.

[34] Henry S. Warren. *Hacker's Delight*. Addison-Wesley, 2002.

APPENDIX

A. PROOF OF THEOREM 1

Consider two sets, $S_1, S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\}$. Denote $f_1 = |S_1|$, $f_2 = |S_2|$, and $a = |S_1 \cap S_2|$. Apply a random permutation π on S_1 and S_2: $\pi : \Omega \longrightarrow \Omega$. Define the minimum values under π to be z_1 and z_2:

$$z_1 = \min\left(\pi\left(S_1\right)\right), \qquad z_2 = \min\left(\pi\left(S_2\right)\right).$$

Define $e_{1,i} =$ ith lowest bit of z_1, and $e_{2,i} =$ ith lowest bit of z_2. The task is to derive $\mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1\right)$,

which can be decomposed to be

$$\begin{aligned}&\mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 = z_2\right)\\ &+\mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 \neq z_2\right)\\ &=\mathbf{Pr}\left(z_1 = z_2\right) + \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 \neq z_2\right)\\ &=R + \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1,\ z_1 \neq z_2\right).\end{aligned}$$

where $R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \mathbf{Pr}\left(z_1 = z_2\right)$ is the resemblance.

When $b = 1$, the task boils down to estimating

$$\begin{aligned}&\mathbf{Pr}\left(e_{1,1} = e_{2,1},\ z_1 \neq z_2\right)\\ =&\sum_{i=0,2,4,...}\left\{\sum_{j\neq i, j=0,2,4,...}\mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}\\ +&\sum_{i=1,3,5,...}\left\{\sum_{j\neq i, j=1,3,5,...}\mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}.\end{aligned}$$

Therefore, we need the following basic probability formula:

$$\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i \neq j\right).$$

We start with

$$\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i < j\right) = \frac{P_1 + P_2}{P_3}, \quad \text{where}$$

$$\begin{aligned}P_3 &= \binom{D}{a}\binom{D-a}{f_1-a}\binom{D-f_1}{f_2-a},\\ P_1 &= \binom{D-j-1}{a}\binom{D-j-1-a}{f_2-a-1}\binom{D-i-1-f_2}{f_1-a-1},\\ P_2 &= \binom{D-j-1}{a-1}\binom{D-j-a}{f_2-a}\binom{D-i-1-f_2}{f_1-a-1}.\end{aligned}$$

The expressions for P_1, P_2, and P_3 can be understood by the experiment of randomly throwing f_1+f_2-a balls into D locations, labeled $0, 1, 2, ..., D-1$. Those f_1+f_2-a balls belong to three disjoint sets: $S_1 - S_1\cap S_2$, $S_2 - S_1 \cap S_2$, and $S_1 \cap S_2$. Without any restriction, the total number of combinations should be P_3.

To understand P_1 and P_2, we need to consider two cases:

1. *The jth element is not in $S_1 \cap S_2$:* $\Longrightarrow P_1$.
   We first allocate the $a = |S_1 \cap S_2|$ overlapping elements randomly in $[j+1,\ D-1]$, resulting in $\binom{D-j-1}{a}$ combinations. Then we allocate the remaining f_2-a-1 elements in S_2 also randomly in the unoccupied locations in $[j+1, D-1]$, resulting in $\binom{D-j-1-a}{f_2-a-1}$ combinations. Finally, we allocate the remaining elements in S_1 randomly in the unoccupied locations in $[i+1, D-1]$, which has $\binom{D-i-1-f_2}{f_1-a-1}$ combinations.
2. *The jth element is in $S_1 \cap S_2$:* $\Longrightarrow P_2$.

After conducing expansions and cancelations, we obtain

$$\begin{aligned}&\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i < j\right) = \frac{P_1+P_2}{P_3}\\ =&\frac{\left(\frac{1}{a} + \frac{1}{f_2-a}\right)\frac{(D-j-1)!(D-i-1-f_2)!}{(a-1)!(f_1-a-1)!(f_2-a-1)!(D-j-f_2)!(D-i-f_1-f_2+a)!}}{\frac{D!}{a!(f_1-a)!(f_2-a)!(D-f_1-f_2+a)!}}\\ =&\frac{f_2(f_1-a)(D-j-1)!(D-f_2-i-1)!(D-f_1-f_2+a)!}{D!(D-f_2-j)!(D-f_1-f_2+a-i)!}\\ =&\frac{f_2(f_1-a)\prod_{t=0}^{j-i-2}(D-f_2-i-1-t)\prod_{t=0}^{i-1}(D-f_1-f_2+a-t)}{\prod_{t=0}^{j}(D-t)}\\ =&\frac{f_2}{D}\frac{f_1-a}{D-1}\prod_{t=0}^{j-i-2}\frac{D-f_2-i-1-t}{D-2-t}\prod_{t=0}^{i-1}\frac{D-f_1-f_2+a-t}{D+i-j-1-t}\end{aligned}$$

For convenience, we introduce the following notation:

$$r_1 = \frac{f_1}{D}, \qquad r_2 = \frac{f_2}{D}, \qquad s = \frac{a}{D}.$$

Also, we assume D is large (which is always satisfied in practice). Thus, we can obtain a reasonable approximation:

$$\begin{aligned}&\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i < j\right)\\ =&r_2(r_1 - s)\left[1-r_2\right]^{j-i-1}\left[1-(r_1+r_2-s)\right]^{i}\end{aligned}$$

Similarly, we obtain, for large D,

$$\begin{aligned}&\mathbf{Pr}\left(z_1 = i,\ z_2 = j,\ i > j\right)\\ =&r_1(r_2 - s)\left[1-r_1\right]^{i-j-1}\left[1-(r_1+r_2-s)\right]^{j}\end{aligned}$$

Now we have the tool to calculate the probability

$$\begin{aligned}&\mathbf{Pr}\left(e_{1,1} = e_{2,1},\ z_1 \neq z_2\right)\\ =&\sum_{i=0,2,4,...}\left\{\sum_{j\neq i, j=0,2,4,...}\mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}\\ +&\sum_{i=1,3,5,...}\left\{\sum_{j\neq i, j=1,3,5,...}\mathbf{Pr}\left(z_1 = i, z_2 = j\right)\right\}\end{aligned}$$

For example, (again, assuming D is large)

$$\begin{aligned}&\mathbf{Pr}\left(z_1 = 0, z_2 = 2, 4, 6, ...\right)\\ =&r_2(r_1 - s)\left(\left[1-r_2\right] + \left[1-r_2\right]^3 + \left[1-r_2\right]^5 + ...\right)\\ =&r_2(r_1 - s)\frac{1-r_2}{1-\left[1-r_2\right]^2}\end{aligned}$$

$$\begin{aligned}&\mathbf{Pr}\left(z_1=1,z_2=3,5,7,...\right)=r_2(r_1-s)[1-(r_1+r_2-s)]\\&\qquad\times\left([1-r_2]+[1-r_2]^3+[1-r_2]^5+...\right)\\=&r_2(r_1-s)[1-(r_1+r_2-s)]\frac{1-r_2}{1-[1-r_2]^2}.\end{aligned}$$

Therefore,

$$\begin{aligned}&\sum_{i=0,2,4,...}\left\{\sum_{i<j,j=0,2,4,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\&+\sum_{i=1,3,5,...}\left\{\sum_{i<j,j=1,3,5,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\=&r_2(r_1-s)\frac{1-r_2}{1-[1-r_2]^2}\times\\&\left(1+[1-(r_1+r_2-s)]+[1-(r_1+r_2-s)]^2+...\right)\\=&r_2(r_1-s)\frac{1-r_2}{1-[1-r_2]^2}\frac{1}{r_1+r_2-s}.\end{aligned}$$

By symmetry, we know

$$\begin{aligned}&\sum_{j=0,2,4,...}\left\{\sum_{i>j,i=0,2,4,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\&+\sum_{j=1,3,5,...}\left\{\sum_{i>j,i=1,3,5,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\=&r_1(r_2-s)\frac{1-r_1}{1-[1-r_1]^2}\frac{1}{r_1+r_2-s}.\end{aligned}$$

Combining the probabilities, we obtain

$$\begin{aligned}&\mathbf{Pr}\left(e_{1,1}=e_{2,1},\ z_1\neq z_2\right)\\=&\frac{r_2(1-r_2)}{1-[1-r_2]^2}\frac{r_1-s}{r_1+r_2-s}+\frac{r_1(1-r_1)}{1-[1-r_1]^2}\frac{r_2-s}{r_1+r_2-s}\\=&A_{1,1}\frac{r_2-s}{r_1+r_2-s}+A_{2,1}\frac{r_1-s}{r_1+r_2-s},\end{aligned}$$

where

$$A_{1,b}=\frac{r_1\left[1-r_1\right]^{2^b-1}}{1-\left[1-r_1\right]^{2^b}},\qquad A_{2,b}=\frac{r_2\left[1-r_2\right]^{2^b-1}}{1-\left[1-r_2\right]^{2^b}}.$$

Therefore, we can obtain the desired probability, for $b=1$,

$$\begin{aligned}&\mathbf{Pr}\left(\prod_{i=1}^{b=1}1\{e_{1,i}=e_{2,i}\}=1\right)\\=&R+A_{1,1}\frac{r_2-s}{r_1+r_2-s}+A_{2,1}\frac{r_1-s}{r_1+r_2-s}\\=&R+A_{1,1}\frac{f_2-a}{f_1+f_2-a}+A_{2,1}\frac{f_1-a}{f_1+f_2-a}\\=&R+A_{1,1}\frac{f_2-\frac{R}{1+R}(f_1+f_2)}{f_1+f_2-\frac{R}{1+R}(f_1+f_2)}+A_{2,1}\frac{f_1-a}{f_1+f_2-a}\\=&R+A_{1,1}\frac{f_2-Rf_1}{f_1+f_2}+A_{2,1}\frac{f_1-Rf_2}{f_1+f_2}\\=&C_{1,1}+(1-C_{2,1})R\end{aligned}$$

where

$$\begin{aligned}C_{1,b}=A_{1,b}\frac{r_2}{r_1+r_2}+A_{2,b}\frac{r_1}{r_1+r_2}\\C_{2,b}=A_{1,b}\frac{r_1}{r_1+r_2}+A_{2,b}\frac{r_2}{r_1+r_2}.\end{aligned}$$

To this end, we have proved the main result for $b=1$.

Next, we consider $b>1$. Due to the space limit, we only provide a sketch of the proof. When $b=2$, we need

$$\begin{aligned}&\mathbf{Pr}\left(e_{1,1}=e_{2,1},e_{1,2}=e_{2,2},\ \ z_1\neq z_2\right)\\=&\sum_{i=0,4,8,...}\left\{\sum_{j\neq i,j=0,4,8,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\&+\sum_{i=1,5,9,...}\left\{\sum_{j\neq i,j=1,5,9,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\&+\sum_{i=2,6,10,...}\left\{\sum_{j\neq i,j=2,6,10,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\\&+\sum_{i=3,7,11,...}\left\{\sum_{j\neq i,j=3,7,11,...}\mathbf{Pr}\left(z_1=i,z_2=j\right)\right\}\end{aligned}$$

We again use the basic probability formula $\mathbf{Pr}\left(z_1=i,z_2=j,i<j\right)$ and the sum of (different) geometric series, for example,

$$[1-r_2]^3+[1-r_2]^7+[1-r_2]^{11}+...=\frac{[1-r_2]^{2^2-1}}{1-[1-r_2]^{2^2}}.$$

Similarly, for general b, we will need

$$[1-r_2]^{2^b-1}+[1-r_2]^{2\times2^b-1}+[1-r_2]^{3\times2^b-1}+...=\frac{[1-r_2]^{2^b-1}}{1-[1-r_2]^{2^b}}.$$

After more algebra, we prove the general case:

$$\begin{aligned}&\mathbf{Pr}\left(\prod_{i=1}^{b}1\{e_{1,i}=e_{2,i}\}=1\right)\\=&R+A_{1,b}\frac{r_2-s}{r_1+r_2-s}+A_{2,b}\frac{r_1-s}{r_1+r_2-s}\\=&C_{1,b}+(1-C_{2,b})R,\end{aligned}$$

It remains to show some useful properties of $A_{1,b}$ (same for $A_{2,b}$). The first derivative of $A_{1,b}$ with respect to b is

$$\begin{aligned}\frac{\partial A_{1,b}}{\partial b}=&\frac{r_1[1-r_1]^{2^b-1}\log(1-r_1)\log2\left(1-[1-r_1]^{2^b}\right)}{\left(1-[1-r_1]^{2^b}\right)^2}\\&-\frac{-[1-r_1]^{2^b}\log(1-r_1)\log2\,r_1\left(1-[1-r_1]^{2^b-1}\right)}{\left(1-[1-r_1]^{2^b}\right)^2}\\\leq&0\qquad\qquad(\text{Note that }\log(1-r_1)\leq0)\end{aligned}$$

Thus, $A_{1,b}$ is a monotonically decreasing function of b. Also,

$$\lim_{r_1\to0}A_{1,b}=\lim_{r_1\to0}\frac{[1-r_1]^{2^b-1}-r_1\left(2^b-1\right)[1-r_1]^{2^b-2}}{2^b[1-r_1]^{2^b-1}}=\frac{1}{2^b},$$

$$\begin{aligned}\frac{\partial A_{1,b}}{\partial r_1}=&\frac{[1-r_1]^{2^b-1}-r_1\left(2^b-1\right)[1-r_1]^{2^b-2}}{\left(1-[1-r_1]^{2^b}\right)}\\&-\frac{2^b[1-r_1]^{2^b-1}r_1\left[1-r_1\right]^{2^b-1}}{\left(1-[1-r_1]^{2^b}\right)^2}\\=&\frac{[1-r_1]^{2^b-2}}{\left(1-[1-r_1]^{2^b}\right)^2}\left(1-2^br_1-[1-r_1]^{2^b}\right)\leq0.\end{aligned}$$

Note that $(1-x)^c\geq1-cx$, for $c\geq1$ and $x\leq1$. Therefore $A_{1,b}$ is a monotonically decreasing function of r_1.

b-Bit Minwise Hashing for Estimating Three-Way Similarities

Ping Li
Dept. of Statistical Science
Cornell University

Arnd Christian König
Microsoft Research
Microsoft Corporation

Wenhao Gui
Dept. of Statistical Science
Cornell University

Abstract

Computing[1] two-way and multi-way set similarities is a fundamental problem. This study focuses on estimating 3-way **resemblance** (Jaccard similarity) using b-bit minwise hashing. While traditional minwise hashing methods store each hashed value using 64 bits, b-bit minwise hashing only stores the lowest b bits (where $b \geq 2$ for 3-way). The extension to 3-way similarity from the prior work on 2-way similarity is technically non-trivial. We develop the precise estimator which is accurate and very complicated; and we recommend a much simplified estimator suitable for sparse data. Our analysis shows that b-bit minwise hashing can normally achieve a 10 to 25-fold improvement in the storage space required for a given estimator accuracy of the 3-way resemblance.

1 Introduction

The efficient computation of the similarity (or overlap) between sets is a central operation in a variety of applications, such as *word associations* (e.g., [13]), *data cleaning* (e.g., [40, 9]), *data mining* (e.g., [14]), *selectivity estimation* (e.g., [30]) or *duplicate document detection* [3, 4]. In machine learning applications, binary (0/1) vectors can be naturally viewed as sets. For scenarios where the underlying data size is sufficiently large to make storing them (in main memory) or processing them in their entirety impractical, probabilistic techniques have been proposed for this task.

Word associations (collocations, co-occurrences) If one inputs a query *NIPS machine learning*, all major search engines will report the number of pagehits (e.g., one reports 829,003), in addition to the top ranked URLs. Although no search engines have revealed how they estimate the numbers of pagehits, one natural approach is to treat this as a set intersection estimation problem. Each word can be represented as a set of document IDs; and each set belongs to a very large space Ω. It is expected that $|\Omega| > 10^{10}$. Word associations have many other applications in Computational Linguistics [13, 38], and were recently used for Web search query reformulation and query suggestions [42, 12].

Here is another example. Commercial search engines display various form of "vertical" content (e.g., *images, news, products*) as part of Web search. In order to determine from which "vertical" to display information, there exist various techniques to select verticals. Some of these (e.g., [29, 15]) use the number of documents the words in a search query occur in for different text corpora representing various verticals as features. Because this selection is invoked for all search queries (and the tight latency bounds for search), the computation of these features has to be very fast. Moreover, the accuracy of vertical selection depends on the number/size of document corpora that can be processed within the allotted time [29], i.e., the processing speed can directly impact quality.

Now, because of the large number of word-combinations in even medium-sized text corpora (e.g., the Wikipedia corpus contains $> 10^7$ distinct terms), it is impossible to pre-compute and store the associations for all possible multi-term combinations (e.g., $> 10^{14}$ for 2-way and $> 10^{21}$ for 3-way); instead the techniques described in this paper can be used for fast estimates of the co-occurrences.

Database query optimization Set intersection is a routine operation in databases, employed for example during the evaluation of conjunctive selection conditions in the presence of single-column indexes. Before conducting intersections, a critical task is to (quickly) estimate the sizes of the intermediate results to plan the optimal intersection order [20, 8, 25]. For example, consider the task of intersecting four sets of record identifiers: $A \cap B \cap C \cap D$. Even though the final outcome will be the same, the order of the join operations, e.g., $(A \cap B) \cap (C \cap D)$ or $((A \cap B) \cap C) \cap D$, can significantly affect the performance, in particular if the intermediate results, e.g., $A \cap B \cap C$, become too large for main memory and need to be spilled to disk. A good query plan aims to minimize

[1]This work is supported by NSF (DMS-0808864), ONR (YIP-N000140910911) and Microsoft.

the total size of intermediate results. Thus, it is highly desirable to have a mechanism which can estimate join sizes very efficiently, especially for the lower-order (2-way and 3-way) intersections, which could potentially result in much larger intermediate results than higher-order intersections.

Duplicate Detection in Data Cleaning: A common task in data cleaning is the identification of duplicates (e.g., duplicate names, organizations, etc.) among a set of items. Now, despite the fact that there is considerable evidence (e.g., [10]) that reliable duplicate-detection should be based on local properties of *groups* of duplicates, most current approaches base their decisions on pairwise similarities between items only. This is in part due to the computational overhead associated with more complex interactions, which our approach may help to overcome.

Clustering Most clustering techniques are based on pair-wise distances between the items to be clustered. However, there are a number of natural scenarios where the affinity relations are not pairwise, but rather triadic, tetradic or higher (e.g. [1, 43]). Again, our approach may improve the performance in these scenarios if the distance measures can be expressed in the form of set-overlap.

Data mining A lot of work in data mining has focused on efficient candidate pruning in the context of pairwise associations (e.g., [14]), a number of such pruning techniques leverage minwise hashing to prune pairs of items, but in many contexts (e.g., association rules with more than 2 items) multi-way associations are relevant; here, pruning based on pairwise interactions may perform much less well than multi-way pruning.

1.1 Ultra-high dimensional data are often binary

For duplicate detection in the context of Web crawling/search, each document can be represented as a set of w-shingles (w contiguous words); $w = 5$ or 7 in several studies [3, 4, 17]. Normally only the abscence/presence (0/1) information is used, as a w-shingle rarely occurs more than once in a page if $w \geq 5$. The total number of shingles is commonly set to be $|\Omega| = 2^{64}$; and thus the set intersection corresponds to computing the inner product in binary data vectors of 2^{64} dimensions. Interestingly, even when the data are not too high-dimensional (e.g., only thousands), empirical studies [6, 23, 26] achieved good performance using SVM with binary-quantized (text or image) data.

1.2 Minwise Hashing and SimHash

Two of the most widely adopted approaches for estimating set intersections are **minwise hashing** [3, 4] and **sign (1-bit) random projections** (also known as **simhash**) [7, 34], which are both special instances of the general techniques proposed in the context of locality-sensitive hashing [7, 24]. These techniques have been successfully applied to many tasks in machine learning, databases, data mining, and information retrieval [18, 36, 11, 22, 16, 39, 28, 41, 27, 5, 2, 37, 7, 24, 21].

Limitations of random projections The method of random projections (including simhash) is limited to estimating pairwise similarities. Random projections convert any data distributions to (zero-mean) multivariate normals, whose density functions are determined by the covariance matrix which contains only the pairwise information of the original data. This is a serious limitation.

1.3 Prior work on b-Bit Minwise Hashing

Instead of storing each hashed value using 64 bits as in prior studies, e.g., [17], [35] suggested to store only the lowest b bits. [35] demonstrated that using $b = 1$ reduces the storage space at least by a factor of 21.3 (for a given accuracy) compared to $b = 64$, if one is interested in resemblance ≥ 0.5, the threshold used in prior studies [3, 4]. Moreover, by choosing the value b of bits to be retained, it becomes possible to systematically adjust the degree to which the estimator is "tuned" towards higher similarities as well as the amount of hashing (random permutations) required.

[35] concerned only the pairwise resemblance. To extend it to the multi-way case, we have to solve new and challenging probability problems. Compared to the pairwise case, our new estimator is significantly different. In fact, as we will show later, estimating 3-way resemblance requires $b \geq 2$.

1.4 Notation

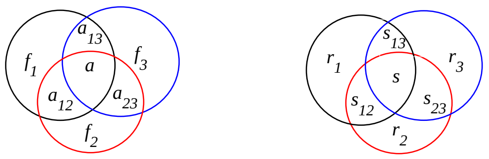


Figure 1: Notation for 2-way and 3-way set intersections.

Fig. 1 describes the notation used in 3-way intersections for three sets $S_1, S_2, S_3 \in \Omega,\ |\Omega| = D$.

- $f_1 = |S_1|,\ f_2 = |S_2|,\ \ f_3 = |S_3|$.
- $a_{12} = |S_1 \cap S_2|,\ a_{13} = |S_1 \cap S_3|,\ a_{23} = |S_2 \cap S_3|, a = a_{123} = |S_1 \cap S_2 \cap S_3|$.
- $r_1 = \frac{f_1}{D},\ r_2 = \frac{f_2}{D},\ r_3 = \frac{f_3}{D}.\ s_{12} = \frac{a_{12}}{D}, s_{13} = \frac{a_{13}}{D}, s_{23} = \frac{a_{23}}{D}, s = s_{123} = \frac{a}{D}$.
- $u = r_1 + r_2 + r_3 - s_{12} - s_{13} - s_{23} + s$.

We define three 2-way resemblances (R_{12}, R_{13}, R_{23}) and one 3-way resemblance (R) as:

$$R_{12} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|},\quad R_{13} = \frac{|S_1 \cap S_3|}{|S_1 \cup S_3|},\quad R_{23} = \frac{|S_2 \cap S_3|}{|S_2 \cup S_3|},\qquad R = R_{123} = \frac{|S_1 \cap S_2 \cap S_3|}{|S_1 \cup S_2 \cup S_3|}. \tag{1}$$

which, using our notation, can be expressed in various forms:

$$R_{ij} = \frac{a_{ij}}{f_i + f_j - a_{ij}} = \frac{s_{ij}}{r_i + r_j - s_{ij}},\ i \neq j, \tag{2}$$

$$R = \frac{a}{f_1 + f_2 + f_3 - a_{12} - a_{23} - a_{13} + a} = \frac{s}{r_1 + r_2 + r_3 - s_{12} - s_{23} - s_{13} + s} = \frac{s}{u}. \tag{3}$$

Note that, instead of a_{123}, s_{123}, R_{123}, we simply use a, s, R. When the set sizes, $f_i = |S_i|$, can be assumed to be known, we can compute resemblances from intersections and vice versa:

$$a_{ij} = \frac{R_{ij}}{1 + R_{ij}}(f_i + f_j), \qquad a = \frac{R}{1 - R}\left(f_1 + f_2 + f_3 - a_{12} - a_{13} - a_{23}\right).$$

Thus, estimating resemblances and estimating intersection sizes are two closely related problems.

1.5 Our Main Contributions

- We derive the basic probability formula for estimating 3-way resemblance using b-bit hashing. The derivation turns out to be significantly much more complex than the 2-way case. This basic probability formula naturally leads to a (complicated) estimator of resemblance.
- We leverage the observation that many real applications involve sparse data (i.e., $r_i = \frac{f_i}{D} \approx 0$, but $f_i/f_j = r_i/r_j$ may be still significant) to develop a much simplified estimator, which is desired in practical applications. This assumption of $f_i/D \rightarrow 0$ significantly simplifies the estimator and frees us from having to know the cardinalities f_i.
- We analyze the theoretical variance of the simplified estimator and compare it with the original minwise hashing method (using 64 bits). Our theoretical analysis shows that b-bit minwise hashing can normally achieve a 10 to 25-fold improvement in storage space (for a given estimator accuracy of the 3-way resemblance) when the set similarities are not extremely low (e.g., when the 3-way resemblance > 0.02). These results are particularly important for applications in which only detecting high resemblance/overlap is relevant, such as many data cleaning scenarios or duplicate detection.

The recommended procedure for estimating 3-way resemblances (in sparse data) is shown as Alg. 1.

Algorithm 1 The b-bit minwise hashing algorithm, applied to estimating 3-way resemblances in a collection of N sets. This procedure is suitable for sparse data, i.e., $r_i = f_i/D \approx 0$.

```
Input: Sets $S_n \in \Omega = \{0, 1, ..., D-1\}$, $n = 1$ to $N$.
Pre-processing phrase:
1) Generate $k$ random permutations $\pi_j : \ \Omega \rightarrow \Omega, j = 1$ to $k$.
2) For each set $S_n$ and permutation $\pi_j$, store the lowest $b$ bits of $\min\left(\pi_j\left(S_n\right)\right)$, denoted by $e_{n,t,\pi_j}$, $t = 1$ to $b$.
Estimation phrase: (Use three sets $S_1$, $S_2$, and $S_3$ as an example.)
1) Compute $\hat{P}_{12,b} = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{t=1}^{b} 1\{e_{1,t,\pi_j} = e_{2,t,\pi_j}\}\right\}$. Similarly, compute $\hat{P}_{13,b}$ and $\hat{P}_{23,b}$.
2) Compute $\hat{P}_b = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{t=1}^{b} 1\{e_{1,t,\pi_j} = e_{2,t,\pi_j} = e_{3,t,\pi_j}\}\right\}$.
3) Estimate $R$ by $\hat{R}_b = \frac{4^b\hat{P}_b - 2^b\left(\hat{P}_{12,b} + \hat{P}_{13,b} + \hat{P}_{23,b}\right) + 2}{(2^b-1)(2^b-2)}$.
4) If needed, the 2-way resemblances $R_{ij,b}$ can be estimated as $\hat{R}_{ij,b} = \frac{2^b\hat{P}_{ij,b} - 1}{2^b - 1}$.
```

2 The Precise Theoretical Probability Analysis

Minwise hashing applies k random permutations $\pi_j : \Omega \longrightarrow \Omega$, $\Omega = \{0, 1, ..., D-1\}$, and then estimates R_{12} (and similarly other 2-way resemblances) using the following probability:

$$\mathbf{Pr}\left(\min(\pi_j(S_1)) = \min(\pi_j(S_2))\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R_{12}. \tag{4}$$

This method naturally extends to estimating 3-way resemblances for three sets $S_1, S_2, S_3 \in \Omega$:

$$\mathbf{Pr}\left(\min(\pi_j(S_1)) = \min(\pi_j(S_2)) = \min(\pi_j(S_3))\right) = \frac{|S_1 \cap S_2 \cap S_3|}{|S_1 \cup S_2 \cup S_3|} = R. \tag{5}$$

To describe b-bit hashing, we define the minimum values under π and their lowest b bits to be:

$$z_i = \min\left(\pi\left(S_i\right)\right), \qquad e_{i,t} = t\text{-th lowest bit of } z_i.$$

To estimate R, we need to computes the empirical estimates of the probabilities $P_{ij,b}$ and P_b, where

$$P_{ij,b} = \mathbf{Pr}\left(\prod_{t=1}^{b} 1\{e_{i,t} = e_{j,t}\} = 1\right), \qquad P_b = P_{123,b} = \mathbf{Pr}\left(\prod_{t=1}^{b} 1\{e_{1,t} = e_{2,t} = e_{3,t}\} = 1\right).$$

The main theoretical task is to derive P_b. The prior work[35] already derived $P_{ij,b}$; see Appendix A. To simplify the algebra, we assume that D is large, which is virtually always satisfied in practice.

Theorem 1 *Assume D is large.*

$$P_b = \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i} = e_{3,i}\} = 1\right) = \frac{Z}{u} + R = \frac{Z+s}{u}, \tag{6}$$

where $u = r_1 + r_2 + r_3 - s_{12} - s_{13} - s_{23} + s$, and

$$\begin{aligned}
Z =& (s_{12} - s)A_{3,b} + \frac{(r_3 - s_{13} - s_{23} + s)}{r_1 + r_2 - s_{12}} s_{12} G_{12,b} + (s_{13} - s)A_{2,b} + \frac{(r_2 - s_{12} - s_{23} + s)}{r_1 + r_3 - s_{13}} s_{13} G_{13,b} \\
&+ (s_{23} - s)A_{1,b} + \frac{(r_1 - s_{12} - s_{13} + s)}{r_2 + r_3 - s_{23}} s_{23} G_{23,b} + \left[(r_2 - s_{23})A_{3,b} + (r_3 - s_{23})A_{2,b}\right] \frac{(r_1 - s_{12} - s_{13} + s)}{r_2 + r_3 - s_{23}} G_{23,b} \\
&+ \left[(r_1 - s_{13})A_{3,b} + (r_3 - s_{13})A_{1,b}\right] \frac{(r_2 - s_{12} - s_{23} + s)}{r_1 + r_3 - s_{13}} G_{13,b} \\
&+ \left[(r_1 - s_{12})A_{2,b} + (r_2 - s_{12})A_{1,b}\right] \frac{(r_3 - s_{13} - s_{23} + s)}{r_1 + r_2 - s_{12}} G_{12,b},
\end{aligned}$$

$$A_{j,b} = \frac{r_j(1 - r_j)^{2^b - 1}}{1 - (1 - r_j)^{2^b}}, \qquad G_{ij,b} = \frac{(r_i + r_j - s_{ij})(1 - r_i - r_j + s_{ij})^{2^b - 1}}{1 - (1 - r_i - r_j + s_{ij})^{2^b}}, \quad i, j \in \{1, 2, 3\},\ i \neq j.$$

Theorem 1 naturally suggests an iterative estimation procedure, by writing Eq. (6) as $s = P_b u - Z$.

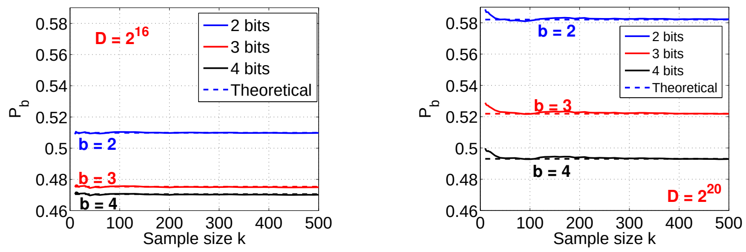


Figure 2: P_b, for verifying the probability formula in Theorem 1. The empirical estimates and the theoretical predictions essentially overlap regardless of the sparsity measure $r_i = f_i/D$.

A Simulation Study For the purpose of verifying Theorem 1, we use three sets corresponding to the occurrences of three common words ("OF", "AND", and "OR") in a chunk of real world Web crawl data. Each (word) set is a set of document (Web page) IDs which contained that word at least once. The three sets are not too sparse and $D = 2^{16}$ suffices to represent their elements. The $r_i = \frac{f_i}{D}$ values are 0.5697, 0.5537, and 0.3564, respectively. The true 3-way resemblance is $R = 0.47$.

We can also increase D by mapping these sets into a larger space using a random mapping, with $D = 2^{16}$, 2^{18}, 2^{20}, or 2^{22}. When $D = 2^{22}$, the r_i values are 0.0089, 0.0087, 0.0056.

Fig. 2 presents the empirical estimates of the probability P_b, together with the theoretical predictions by Theorem 1. The empirical estimates essentially overlap the theoretical predictions. Even though the proof assumes $D \to \infty$, D does not have to be too large for Theorem 1 to be accurate.

3 The Much Simplified Estimator for Sparse Data

The basic probability formula (Theorem 1) we derive could be too complicated for practical use. To obtain a simpler formula, we leverage the observation that in practice we often have $r_i = \frac{f_i}{D} \approx 0$, even though both f_i and D can be very large. For example, consider web duplicate detection [17]. Here, $D = 2^{64}$, which means that even for a web page with $f_i = 2^{54}$ shingles (corresponding to the text of a small novel), we still have $\frac{f_i}{D} \approx 0.001$. Note that, even when $r_i \to 0$, the ratios, e.g., $\frac{r_2}{r_1}$, can be still large. Recall the resemblances (2) and (3) are only determined by these ratios.

We analyzed the distribution of $\frac{f_i}{D}$ using two real-life datasets: the UCI dataset containing 3×10^5 NYTimes articles; and a Microsoft proprietary dataset with 10^6 news articles [19]. For the UCI-NYTimes dataset, each document was already processed as a set of single words. For the anonymous dataset, we report results using three different representations: single words (1-shingle), 2-shingles (two contiguous words), and 3-shingles. Table 1 reports the summary statistics of the $\frac{f_i}{D}$ values.

Table 1: Summary statistics of the $\frac{f_i}{D}$ values in two datasets

Data	Median	Mean	Std.
3×10^5 UCI-NYTimes articles	0.0021	0.0022	0.0011
10^6 Microsoft articles (1-shingle)	0.00027	0.00032	0.00023
10^6 Microsoft articles (2-shingle)	0.00003	0.00004	0.00005
10^6 Microsoft articles (3-shingle)	0.00002	0.00002	0.00002

For truly large-scale applications, prior studies [3, 4, 17] commonly used 5-shingles. This means that real world data may be significantly more sparse than the values reported in Table 1.

3.1 The Simplified Probability Formula and the Practical Estimator

Theorem 2 *Assume D is large. Let $T = R_{12} + R_{13} + R_{23}$. As $r_1, r_2, r_3 \to 0$,*

$$P_b = \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i} = e_{3,i}\} = 1\right) = \frac{1}{4^b}\left\{(2^b-1)(2^b-2)R + (2^b-1)T + 1\right\}. \quad (7)$$

Interestingly, if $b = 1$, then $P_1 = \frac{1}{4}(1+T)$, i.e., no information about the 3-way resemblance R is contained. Hence, it is necessary to use $b \geq 2$ to estimate 3-way similarities.

Alg. 1 uses $\hat{P}_b$ and $\hat{P}_{ij,b}$ to respectively denote the empirical estimates of the theoretical probabilities P_b and $P_{ij,b}$. Assuming $r_1, r_2, r_3 \to 0$, the proposed estimator of R, denoted by $\hat{R}_b$, is

$$\hat{R}_b = \frac{4^b\hat{P}_b - 2^b\left(\hat{P}_{12,b} + \hat{P}_{13,b} + \hat{P}_{23,b}\right) + 2}{(2^b-1)(2^b-2)}. \quad (8)$$

Theorem 3 *Assume D is large and $r_1, r_2, r_3 \to 0$. Then $\hat{R}_b$ in (8) is unbiased with the variance*

$$Var\left(\hat{R}_b\right) = \frac{1}{k}\frac{1}{(2^b-1)(2^b-2)}\left\{1 + (2^b-3)T + \left(4^b - 6\times 2^b + 10\right)R - (2^b-1)(2^b-2)R^2\right\}. \quad (9)$$

It is interesting to examine several special cases:

- $b = 1$: $Var(\hat{R}_1) = \infty$, i.e., one must use $b \geq 2$.
- $b = 2$: $Var(\hat{R}_2) = \frac{1}{6k}\left(1 + T + 2R - 6R^2\right)$.
- $b = \infty$: $Var(\hat{R}_\infty) = \frac{1}{k}R(1-R) = Var(\hat{R}_M)$. $\hat{R}_M$ is the original minwise hashing estimator for 3-way resemblance. In principle, the estimator $\hat{R}_M$ requires an infinite precision (i.e., $b = \infty$). Numerically, $Var(\hat{R}_M)$ and $Var(\hat{R}_{64})$ are indistinguishable.

3.2 Simulations for Validating Theorem 3

We now present a simulation study for verifying Theorem 3, using the same three sets used in Fig. 2.

Fig. 3 presents the resulting empirical biases: $E(\hat{R}_b) - R_b$. Fig. 4 presents the empirical mean square errors (MSE = bias^2+variance) together with the theoretical variances $Var(\hat{R}_b)$ in Theorem 3.

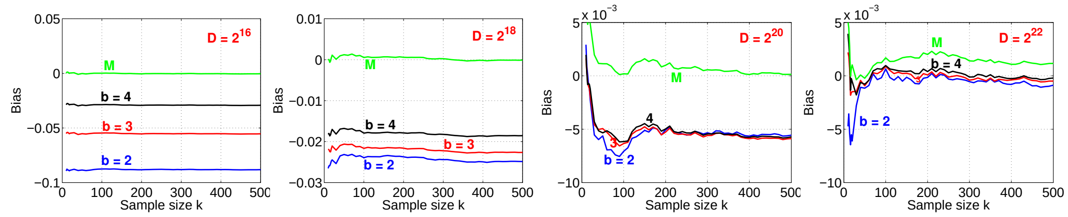


Figure 3: Bias of $\hat{R}_b$ (8). We used 3 (word) sets: "OF", "AND", and "OR" and four D values: 2^{16}, 2^{18}, 2^{20}, and 2^{22}. We conducted experiments using $b = 2$, 3, and 4 as well as the original minwise hashing (denoted by "M"). The plots verify that as r_i decreases (to zero), the biases vanish. Note that the set sizes f_i remain the same, but the relative values $r_i = \frac{f_i}{D}$ decrease as D increases.

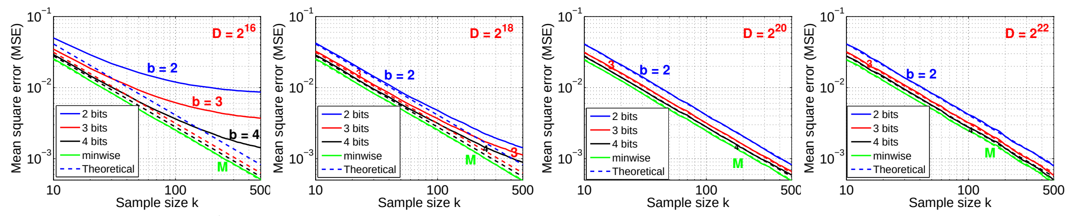


Figure 4: MSE of $\hat{R}_b$ (8). The solid curves are the empirical MSEs (=var+bias^2) and the dashed lines are the theoretical variances (9), under the assumption of $r_i \to 0$. Ideally, we would like to see the solid and dashed lines overlap. When $D = 2^{20}$ and $D = 2^{22}$, even though the r_i values are not too small, the solid and dashed lines almost overlap. Note that, at the same sample size k, we always have $Var(\hat{R}_2) > Var(\hat{R}_3) > Var(\hat{R}_4) > Var(\hat{R}_M)$, where $\hat{R}_M$ is the original minwise hashing estimator. We can see that, $Var(\hat{R}_3)$ and $Var(\hat{R}_4)$ are very close to $Var(\hat{R}_M)$.

We can summarize the results in Fig. 3 and Fig. 4 as follows:

- When the $r_i = \frac{f_i}{D}$ values are large (e.g., $r_i \approx 0.5$ when $D = 2^{16}$), the estimates using (8) can be noticeably biased. The estimation biases diminish as the r_i values decrease. In fact, even when the r_i values are not small (e.g., $r_i \approx 0.05$ when $D = 2^{20}$), the biases are already very small (roughly 0.005 when $D = 2^{20}$).
- The variance formula (9) becomes accurate when the r_i values are not too large. For example, when $D = 2^{18}$ ($r_i \approx 0.1$), the empirical MSEs largely overlap the theoretical variances which assumed $r_i \to 0$, unless the sample size k is large. When $D = 2^{20}$ (and $D = 2^{22}$), the empirical MSEs and theoretical variances overlap.
- For real applications, as we expect D will be very large (e.g., 2^{64}) and the r_i values (f_i/D) will be very small, our proposed simple estimator (8) will be very useful in practice, because it becomes unbiased and the variance can be reliably predicted by (9).

4 Improving Estimates for Dense Data Using Theorem 1

While we believe the simple estimator in (8) and Alg. 1 should suffice in most applications, we demonstrate here that the sparsity assumption of $r_i \to 0$ is not essential if one is willing to use the more sophisticated estimation procedure provided by Theorem 1.

By Eq. (6), $s = P_b u - Z$, where Z contains s, s_{ij}, r_i etc. We first estimate s_{ij} (from the estimated R_{ij}) using the precise formula for the two-way case; see Appendix A. We then iteratively solve for s using the initial guess provided by the estimator $\hat{R}_b$ in (8). Usually a few iterations suffice.

Fig. 5 reports the bias (left most panel, only for $D = 2^{16}$) and MSE, corresponding to Fig. 3 and Fig. 4. In Fig. 5, the solid curves are obtained using the precise estimation procedure by Theorem 1. The dashed curves are the estimates using the simplified estimator $\hat{R}_b$ which assumes $r_i \to 0$.

Even when the data are not sparse, the precise estimation procedure provides unbiased estimates as verified by the leftmost panel of Fig. 5. Using the precise procedure results in noticeably more accurate estimates in non-sparse data, as verified by the second panel of Fig. 5. However, as long as the data are reasonably sparse (the right two panels), the simple estimator $\hat{R}_b$ in (8) is accurate.

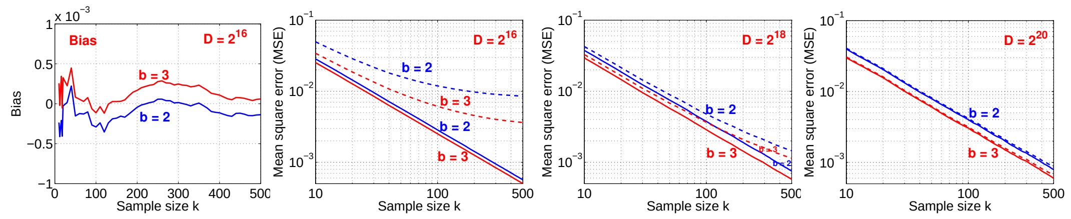


Figure 5: The bias (leftmost panel) and MSE of the precise estimation procedure, using the same data used in Fig. 3 and Fig. 4. The dashed curves correspond to the estimates using the simplified estimator $\hat{R}_b$ in (8) which assumes $r_i \to 0$.

5 Quantifying the Improvements Using b-Bit Hashing

This section is devoted to analyzing the improvements of b-bit minwise hashing, compared to using 64 bits for each hashed value. Throughout the paper, we use the terms "**sample**" and "**sample size**" (denoted by k). The original minwise hashing stores each "sample" using 64 bits (as in [17]). For b-bit minwise hashing, we store each "sample" using b bits only. Note that $Var(\hat{R}_{64})$ and $Var(\hat{R}_M)$ (the variance of the original minwise hashing) are numerically indistinguishable.

As we decrease b, the space needed for each sample will be smaller; the estimation variance at the same sample size k, however, will increase. This variance-space trade-off can be quantified by $B(b) = b \times \text{Var}\left(\hat{R}_b\right) \times k$, which is called the ***storage factor***. Lower $B(b)$ is more desirable. The ratio $\frac{B(64)}{B(b)}$ precisely characterizes the improvements of b-bit hashing compared to using 64 bits.

Fig. 6 confirms the substantial improvements of b-bit hashing over the original minwise hashing using 64 bits. The improvements in terms of the storage space are usually 10 (or 15) to 25-fold when the sets are reasonably similar (i.e., when the 3-way resemblance > 0.1). When the three sets are very similar (e.g., the top left panel), the improvement will be even 25 to 30-fold.

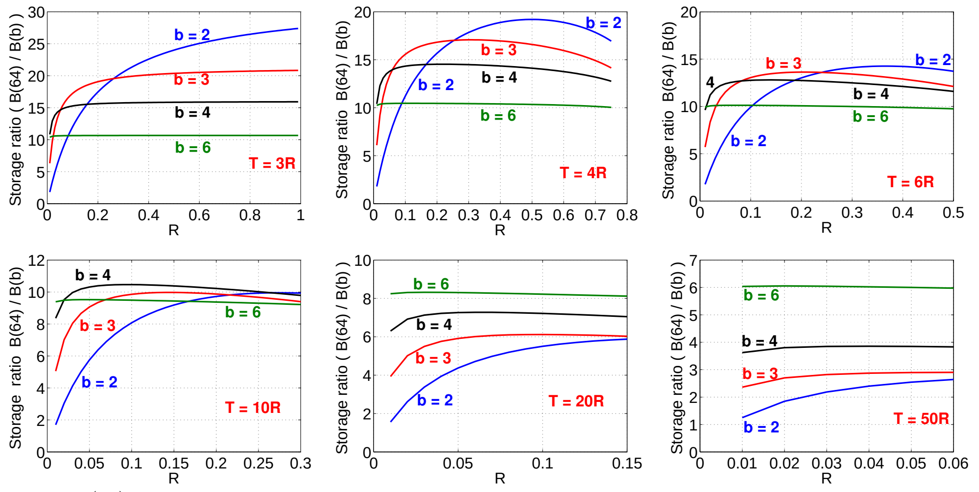


Figure 6: $\frac{B(64)}{B(b)}$, the relative storage improvement of using $b = 2, 3, 4, 6$ bits, compared to using 64 bits. Since the variance (9) contains both R and $T = R_{12} + R_{13} + R_{23}$, we compare variances using different T/R ratios. As $3R \leq T$ always, we let $T = \alpha R$, for some $\alpha \geq 3$. Since $T \leq 3$, we know $R \leq 3/\alpha$. Practical applications are often interested in cases with reasonably large R values.

6 Evaluation of Accuracy

We conducted a duplicate detection experiment on a public (UCI) collection of 300,000 NYTimes news articles. The task is to identify 3-groups with 3-way resemblance R exceeding a threshold $\mathbf{R_0}$. We used a subset of the data; the total number of 3-groups is about one billion. We experimented with $b = 2, 4$ and the original minwise hashing. Fig. 7 presents the precision curves for a representative set of thresholds R_0's. Just like in [35], the recall curves are not shown because they could not differentiate estimators. These curves confirm the significant improvement of using b-bit minwise hashing when the threshold R_0 is quite high (e.g., 0.3). In fact, when $R_0 = 0.3$, using $b = 4$ resulted in similar precisions as using the original minwise hashing (i.e., a 64/4=16-fold reduction in storage). Even when $R_0 = 0.1$, using $b = 4$ can still achieve similar precisions as using the original minwise hashing by only slightly increasing the sample size k.

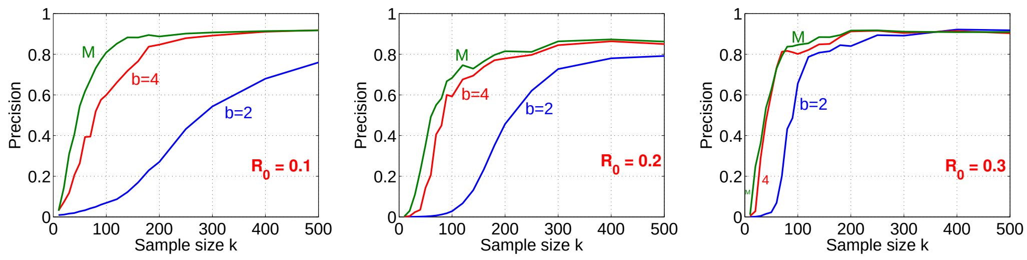


Figure 7: Precision curves on the UCI collection of news data. The task is to retrieve news article 3-groups with resemblance $R \geq R_0$. For example, consider $R_0 = 0.2$. To achieve a precision of at least 0.8, 2-bit hashing and 4-bit hashing require about $k = 500$ samples and $k = 260$ samples respectively, while the original minwise hashing (denoted by M) requires about 170 samples.

7 Conclusion

Computing set similarities is fundamental in many applications. In machine learning, high-dimensional binary data are common and are equivalent to sets. This study is devoted to simultaneously estimating 2-way and 3-way similarities using b-bit minwise hashing. Compared to the prior work on estimating 2-way resemblance [35], the extension to 3-way is important for many application scenarios (as described in Sec. 1) and is technically non-trivial.

For estimating 3-way resemblance, our analysis shows that b-bit minwise hashing can normally achieve a 10 to 25-fold improvement in the storage space required for a given estimator accuracy, when the set similarities are not extremely low (e.g., 3-way resemblance > 0.02). Many applications such as data cleaning and de-duplication are mainly concerned with relatively high set similarities.

For many practical applications, the reductions in storage directly translate to improvements in processing speed as well, especially when memory latency is the main bottleneck, which, with the advent of many-core processors, is more and more common.

Future work: We are interested in developing a b-bit version for *Conditional Random Sampling (CRS)* [31, 32, 33], which requires only one permutation (instead of k permutations) and naturally extends to non-binary data. CRS is also provably more accurate than minwise hashing for binary data. However, the analysis for developing the b-bit version of CRS appears to be very difficult.

A Review of b-Bit Minwise Hashing for 2-Way Resemblance

Theorem 4 ([35]) *Assume D is large.*

$$P_{12,b} = \mathbf{Pr}\left(\prod_{i=1}^{b} 1\{e_{1,i} = e_{2,i}\} = 1\right) = C_{1,b} + (1 - C_{2,b})\, R_{12}$$

where
$$C_{1,b} = A_{1,b}\frac{r_2}{r_1+r_2} + A_{2,b}\frac{r_1}{r_1+r_2}, \quad C_{2,b} = A_{1,b}\frac{r_1}{r_1+r_2} + A_{2,b}\frac{r_2}{r_1+r_2},$$
$$A_{1,b} = \frac{r_1\left[1-r_1\right]^{2^b-1}}{1-\left[1-r_1\right]^{2^b}}, \qquad A_{2,b} = \frac{r_2\left[1-r_2\right]^{2^b-1}}{1-\left[1-r_2\right]^{2^b}}.$$

If $r_1, r_2 \to 0$, $P_{12,b} = \frac{1+(2^b-1)R_{12}}{2^b}$ and one can estimate R_{12} by $\frac{2^b\hat{P}_{12,b}-1}{2^b-1}$, where $\hat{P}_{12,b}$ is the empirical observation of $P_{12,b}$. If r_1, r_2 are not small, R_{12} is estimated by $(\hat{P}_{12,b} - C_{1,b})/(1-C_{2,b})$.

References

[1] S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie. Beyond pairwise clustering. In *CVPR*, 2005.

[2] M. Bendersky and W. B. Croft. Finding text reuse on the web. In *WSDM*, pages 262–271, Barcelona, Spain, 2009.

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

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

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

[6] O. Chapelle, P. Haffner, and V. N. Vapnik. Support vector machines for histogram-based image classification. 10(5):1055–1064, 1999.

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

[8] S. Chaudhuri. An Overview of Query Optimization in Relational Systems. In *PODS*, pages 34–43, 1998.

[9] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operatior for similarity joins in data cleaning. In *ICDE*, 2006.

[10] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE*, pages 865–876, Tokyo, Japan, 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] K. Church. Approximate lexicography and web search. *International Journal of Lexicography*, 21(3):325–336, 2008.

[13] K. Church and P. Hanks. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16(1):22–29, 1991.

[14] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Trans. on Knowl. and Data Eng.*, 13(1), 2001.

[15] F. Diaz. Integration of News Content into Web Results. In *WSDM*, 2009.

[16] Y. Dourisboure, F. Geraci, and M. Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2):1–36, 2009.

[17] 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.

[18] G. Forman, K. Eshghi, and J. Suermondt. Efficient detection of large-scale redundancy in enterprise file systems. *SIGOPS Oper. Syst. Rev.*, 43(1):84–91, 2009.

[19] M. Gamon, S. Basu, D. Belenko, D. Fisher, M. Hurst, and A. C. König. Blews: Using blogs to provide context for news articles. In *AAAI Conference on Weblogs and Social Media*, 2008.

[20] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: the Complete Book*. Prentice Hall, New York, NY, 2002.

[21] A. Gionis, D. Gunopulos, and N. Koudas. Efficient and tunable similar set retrieval. In *SIGMOD*, pages 247–258, CA, 2001.

[22] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, Madrid, Spain, 2009.

[23] M. Hein and O. Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *AISTATS*, pages 136–143, Barbados, 2005.

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

[25] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.

[26] Y. Jiang, C. Ngo, and J. Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *CIVR*, pages 494–501, Amsterdam, Netherlands, 2007.

[27] N. Jindal and B. Liu. Opinion spam and analysis. In *WSDM*, pages 219–230, Palo Alto, California, USA, 2008.

[28] K. Kalpakis and S. Tang. Collaborative data gathering in wireless sensor networks using measurement co-occurrence. *Computer Communications*, 31(10):1979–1992, 2008.

[29] A. C. König, M. Gamon, and Q. Wu. Click-Through Prediction for News Queries. In *SIGIR*, 2009.

[30] H. Lee, R. T. Ng, and K. Shim. Power-law based estimation of set similarity join size. In *PVLDB*, 2009.

[31] P. Li and K. W. Church. A sketch algorithm for estimating two-way and multi-way associations. *Computational Linguistics*, 33(3):305–354, 2007 (Preliminary results appeared in HLT/EMNLP 2005).

[32] 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.

[33] P. Li, K. W. Church, and T. J. Hastie. One sketch for all: Theory and applications of conditional random sampling. In *NIPS*, Vancouver, BC, Canada, 2008.

[34] P. Li, T. J. Hastie, and K. W. Church. Improving random projections using marginal information. In *COLT*, pages 635–649, Pittsburgh, PA, 2006.

[35] P. Li and A. C. König. b-bit minwise hashing. In *WWW*, pages 671–680, Raleigh, NC, 2010.

[36] Ludmila, K. Eshghi, C. B. M. III, J. Tucek, and A. Veitch. Probabilistic frequent itemset mining in uncertain databases. In *KDD*, pages 1087–1096, Paris, France, 2009.

[37] G. S. Manku, A. Jain, and A. D. Sarma. Detecting Near-Duplicates for Web-Crawling. In *WWW*, Banff, Alberta, Canada, 2007.

[38] C. D. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.

[39] 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.

[40] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD*, pages 743–754, 2004.

[41] T. Urvoy, E. Chauveau, P. Filoche, and T. Lavergne. Tracking web spam with html style similarities. *ACM Trans. Web*, 2(1):1–28, 2008.

[42] X. Wang and C. Zhai. Mining term association patterns from search logs for effective query reformulation. In *CIKM*, pages 479–488, Napa Valley, California, USA, 2008.

[43] D. Zhou, J. Huang, and B. Schölkopf. Beyond pairwise classification and clustering using hypergraphs. 2006.

Hashing Algorithms for Large-Scale Learning

Ping Li
Cornell University
pingli@cornell.edu

Anshumali Shrivastava
Cornell University
anshu@cs.cornell.edu

Joshua Moore
Cornell University
jlmo@cs.cornell.edu

Arnd Christian König
Microsoft Research
chrisko@microsoft.com

Abstract

Minwise hashing is a standard technique in the context of search for efficiently computing set similarities. The recent development of b-bit minwise hashing provides a substantial improvement by storing only the lowest b bits of each hashed value. In this paper, we demonstrate that b-bit minwise hashing can be naturally integrated with linear learning algorithms such as linear SVM and logistic regression, to solve large-scale and high-dimensional statistical learning tasks, especially when the data do not fit in memory. We compare b-bit minwise hashing with the Count-Min (CM) and Vowpal Wabbit (VW) algorithms, which have essentially the same variances as random projections. Our theoretical and empirical comparisons illustrate that b-bit minwise hashing is significantly more accurate (at the same storage cost) than VW (and random projections) for binary data.

1 Introduction

With the advent of the Internet, many machine learning applications are faced with very large and inherently high-dimensional datasets, resulting in challenges in scaling up training algorithms and storing the data. Especially in the context of search and machine translation, corpus sizes used in industrial practice have long exceeded the main memory capacity of single machine. For example, [33] discusses training sets with 10^{11} items and 10^9 distinct features, requiring novel algorithmic approaches and architectures. As a consequence, there has been a renewed emphasis on scaling up machine learning techniques by using massively parallel architectures; however, methods relying solely on parallelism can be expensive (both with regards to hardware requirements and energy costs) and often induce significant additional communication and data distribution overhead.

This work approaches the challenges posed by large datasets by leveraging techniques from the area of *similarity search* [2], where similar increases in data sizes have made the storage and computational requirements for computing exact distances prohibitive, thus making data representations that allow compact storage and efficient approximate similarity computation necessary.

The method of b-bit minwise hashing [26–28] is a recent progress for efficiently (in both time and space) computing *resemblances* among extremely high-dimensional (e.g., 2^{64}) binary vectors. In this paper, we show that b-bit minwise hashing can be seamlessly integrated with linear Support Vector Machine (SVM) [13, 18, 20, 31, 35] and logistic regression solvers.

1.1 Ultra High-Dimensional Large Datasets and Memory Bottlenecks

In the context of search, a standard procedure to represent documents (e.g., Web pages) is to use w-shingles (i.e., w contiguous words), where $w \geq 5$ in several studies [6, 7, 14]. This procedure can generate datasets of extremely high dimensions. For example, suppose we only consider 10^5 common English words. Using $w = 5$ may require the size of dictionary Ω to be $D = |\Omega| = 10^{25} = 2^{83}$. In practice, $D = 2^{64}$ often suffices, as the number of available documents may not be large enough to exhaust the dictionary. For w-shingle data, normally only abscence/presence (0/1) information is used, as it is known that word frequency distributions within documents approximately follow a power-law [3], meaning that most single terms occur rarely, thereby making a w-shingle is unlikely to occur more than once in a document. Interestingly, even when the data are not too high-dimensional, empirical studies [8, 17, 19] achieved good performance with binary-quantized data.

When the data can fit in memory, linear SVM training is often extremely efficient after the data are loaded into the memory. It is however often the case that, for very large datasets, the data loading time dominates the computing time for solving the SVM problem [35]. A more severe problem arises when the data can not fit in memory. This situation can be common in practice. The publicly available *webspam* dataset (in LIBSVM format) needs about 24GB disk space, which exceeds the memory capacity of many desktop PCs. Note that *webspam*, which contains only 350,000 documents represented by 3-shingles, is still very small compared to industry applications [33].

1.2 Our Proposal

We propose a solution which leverages b-bit minwise hashing. Our approach assumes the data vectors are binary, high-dimensional, and relatively sparse, which is generally true of text documents represented via shingles. We apply b-bit minwise hashing to obtain a compact representation of the original data. In order to use the technique for efficient learning, we have to address several issues:

- We need to prove that the matrices generated by b-bit minwise hashing are positive definite, which will provide the solid foundation for our proposed solution.
- If we use b-bit minwise hashing to estimate the resemblance, which is nonlinear, how can we effectively convert this nonlinear problem into a linear problem?
- Compared to other hashing techniques such as random projections, Count-Min (CM) sketch [11], or Vowpal Wabbit (VW) [32, 34], does our approach exhibits advantages?

It turns out that our proof in the next section that b-bit hashing matrices are positive definite naturally provides the construction for converting the otherwise nonlinear SVM problem into linear SVM.

2 Review of Minwise Hashing and b-Bit Minwise Hashing

Minwise hashing [6, 7] has been successfully applied to a wide range of real-world problems [4, 6, 7, 9, 10, 12, 15, 16, 30], for efficiently computing set similarities. Minwise hashing mainly works well with binary data, which can be viewed either as 0/1 vectors or as sets. Given two sets, $S_1,\ S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\}$, a widely used measure of similarity is the *resemblance* R:

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

Applying a random permutation $\pi : \Omega \to \Omega$ on S_1 and S_2, the collision probability is simply

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

One can repeat the permutation k times: π_1, π_2, ..., π_k to estimate R without bias. The common practice is to store each hashed value, e.g., $\min(\pi(S_1))$ and $\min(\pi(S_2))$, using 64 bits [14]. The storage (and computational) cost will be prohibitive in truly large-scale (industry) applications [29]. *b-bit minwise hashing* [27] provides a strikingly simple solution to this (storage and computational) problem by storing only the lowest b bits (instead of 64 bits) of each hashed value.

For convenience, denote $z_1 = \min(\pi(S_1))$ and $z_2 = \min(\pi(S_2))$, and denote $z_1^{(b)}$ ($z_2^{(b)}$) the integer value corresponding to the lowest b bits of of z_1 (z_2). For example, if $z_1 = 7$, then $z_1^{(2)} = 3$.

Theorem 1 *[27] Assume D is large.*

$$P_b = \mathbf{Pr}\left(z_1^{(b)} = z_2^{(b)}\right) = C_{1,b} + (1 - C_{2,b})\,R \tag{3}$$

$$r_1 = \frac{f_1}{D},\quad r_2 = \frac{f_2}{D},\quad f_1 = |S_1|,\quad f_2 = |S_2|$$

$$C_{1,b} = A_{1,b}\frac{r_2}{r_1 + r_2} + A_{2,b}\frac{r_1}{r_1 + r_2}, \qquad C_{2,b} = A_{1,b}\frac{r_1}{r_1 + r_2} + A_{2,b}\frac{r_2}{r_1 + r_2},$$

$$A_{1,b} = \frac{r_1\left[1 - r_1\right]^{2^b - 1}}{1 - \left[1 - r_1\right]^{2^b}}, \qquad A_{2,b} = \frac{r_2\left[1 - r_2\right]^{2^b - 1}}{1 - \left[1 - r_2\right]^{2^b}}.\square$$

This (approximate) formula (3) is remarkably accurate, even for very small D; see Figure 1 in [25].

We can then estimate P_b (and R) from k independent permutations:

$$\hat{R}_b = \frac{\hat{P}_b - C_{1,b}}{1 - C_{2,b}}, \qquad \text{Var}\left(\hat{R}_b\right) = \frac{\text{Var}\left(\hat{P}_b\right)}{\left[1 - C_{2,b}\right]^2} = \frac{1}{k}\frac{\left[C_{1,b} + (1 - C_{2,b})R\right]\left[1 - C_{1,b} - (1 - C_{2,b})R\right]}{\left[1 - C_{2,b}\right]^2} \tag{4}$$

It turns out that our method only needs $\hat{P}_b$ for linear learning, i.e., no need to explicitly estimate R.

3 Kernels from Minwise Hashing b-Bit Minwise Hashing

Definition: A symmetric $n \times n$ matrix $\mathbf{K}$ satisfying $\sum_{ij} c_i c_j K_{ij} \geq 0$, for all real vectors c is called *positive definite (PD)*. Note that here we do not differentiate PD from *nonnegative definite*.

Theorem 2 *Consider n sets $S_1, ..., S_n \subseteq \Omega = \{0, 1, ..., D-1\}$. Apply one permutation π to each set. Define $z_i = \min\{\pi(S_i)\}$ and $z_i^{(b)}$ the lowest b bits of z_i. The following three matrices are PD.*

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

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

Proof: *A matrix $\mathbf{A}$ is PD if it can be written as an inner product $\mathbf{B^T B}$. Because*

$$M_{ij} = 1\{z_i = z_j\} = \sum_{t=0}^{D-1} 1\{z_i = t\} \times 1\{z_j = t\}, \tag{5}$$

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

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

Linear algorithms such as linear SVM and logistic regression have become very powerful and extremely popular. Representative software packages include SVM$^{\text{perf}}$ [20], Pegasos [31], Bottou's SGD SVM [5], and LIBLINEAR [13]. Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, 1\}$. The L_2-regularized linear SVM solves the following optimization problem):

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

and the L_2-regularized logistic regression solves a similar problem:

$$\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w^T w} + C\sum_{i=1}^{n} \log\left(1 + e^{-y_i\mathbf{w^T x_i}}\right). \tag{7}$$

Here $C > 0$ is a regularization parameter. Since our purpose is to demonstrate the effectiveness of our proposed scheme using b-bit hashing, we simply provide results for a wide range of C values and assume that the best performance is achievable if we conduct cross-validations.

In our approach, we apply k random permutations on each feature vector $\mathbf{x}_i$ and store the lowest b bits of each hashed value. This way, we obtain a new dataset which can be stored using merely nbk bits. At run-time, we expand each new data point into a $2^b \times k$-length vector with exactly k 1's.

For example, suppose $k = 3$ and the hashed values are originally $\{12013, 25964, 20191\}$, whose binary digits are $\{010111011101101,\ 110010101101100, 100111011011111\}$. Consider $b = 2$. Then the binary digits are stored as $\{01, 00, 11\}$ (which corresponds to $\{1, 0, 3\}$ in decimals). At run-time, we need to expand them into a vector of length $2^b k = 12$, to be $\{0, 0, 1, 0,\ \ 0, 0, 0, 1,\ \ 1, 0, 0, 0\}$, which will be the new feature vector fed to a solver such as LIBLINEAR. Clearly, this expansion is directly inspired by the proof that the b-bit minwise hashing matrix is PD in Theorem 2.

5 Experimental Results on Webspam Dataset

Our experiment settings closely follow the work in [35]. They conducted experiments on three datasets, of which only the *webspam* dataset is public and reasonably high-dimensional ($n = 350000$, $D = 16609143$). Therefore, our experiments focus on *webspam*. Following [35], we randomly selected 20% of samples for testing and used the remaining 80% samples for training.

We chose LIBLINEAR as the workhorse to demonstrate the effectiveness of our algorithm. All experiments were conducted on workstations with Xeon(R) CPU (W5590@3.33GHz) and 48GB

RAM, under Windows 7 System. Thus, in our case, the original data (about 24GB in LIBSVM format) fit in memory. In applications when the data do not fit in memory, we expect that b-bit hashing will be even more substantially advantageous, because the hashed data are relatively very small. In fact, our experimental results will show that for this dataset, using $k = 200$ and $b = 8$ can achieve similar testing accuracies as using the original data. The effective storage for the reduced dataset (with 350K examples, using $k = 200$ and $b = 8$) would be merely about 70MB.

5.1 Experimental Results on Nonlinear (Kernel) SVM

We implemented a new resemblance kernel function and tried to use LIBSVM to train an SVM using the *webspam* dataset. The training time well exceeded 24 hours. Fortunately, using b-bit minswise hashing to estimate the resemblance kernels provides a substantial improvement. For example, with $k = 150$, $b = 4$, and $C = 1$, the training time is about 5185 seconds and the testing accuracy is quite close to the best results given by LIBLINEAR on the original *webspam* data.

5.2 Experimental Results on Linear SVM

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

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

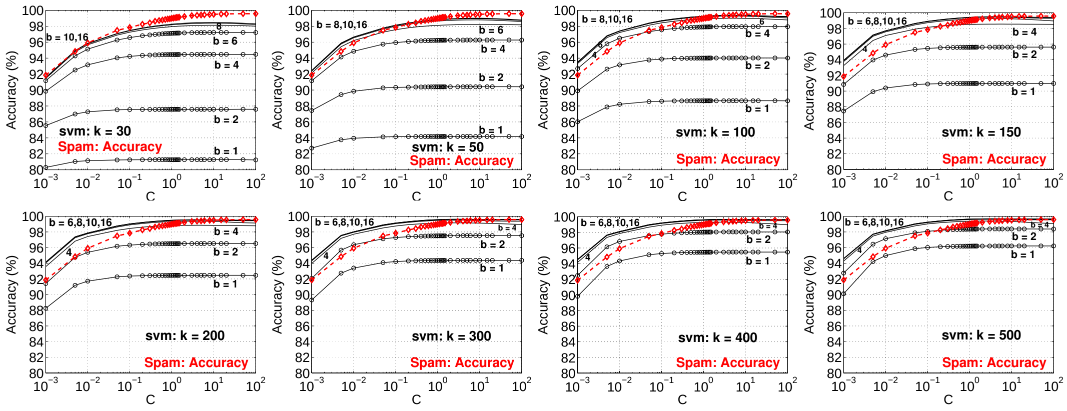


Figure 1: **SVM test accuracy** (averaged over 50 repetitions). With $k \geq 200$ and $b \geq 8$. b-bit hashing achieves very similar accuracies as using the original data (dashed, red if color is available).

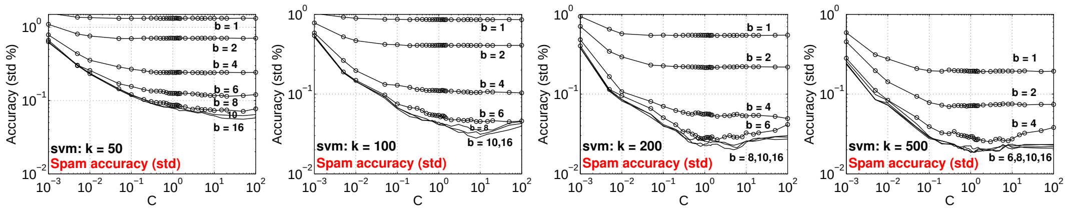


Figure 2: **SVM test accuracy (std)**. The standard deviations are computed from 50 repetitions. When $b \geq 8$, the standard deviations become extremely small (e.g., 0.02%).

Compared with the original training time (about 100 seconds), Figure 3 (upper panels) shows that our method only needs about 3 seconds (near $C = 1$). Note that our reported training time did not include data loading (about 12 minutes for the original data and 10 seconds for the hashed data).

Compared with the original testing time (about 150 seconds), Figure 3 (bottom panels) shows that our method needs merely about 2 seconds. Note that the testing time includes both the data loading time, as designed by LIBLINEAR. The efficiency of testing may be very important in practice, for example, when the classifier is deployed in a user-facing application (such as search), while the cost of training or preprocessing may be less critical and can be conducted off-line.

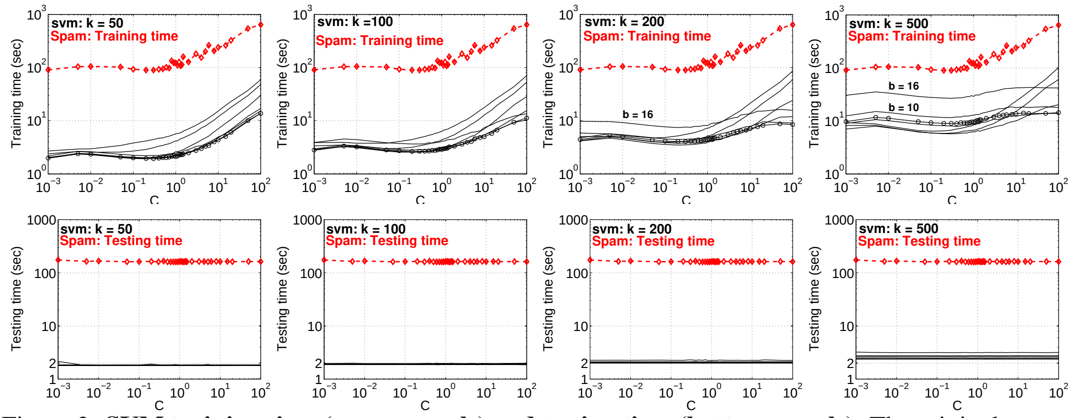


Figure 3: **SVM training time (upper panels) and testing time (bottom panels)**. The original costs are plotted using dashed (red, if color is available) curves.

5.3 Experimental Results on Logistic Regression

Figure 4 presents the test accuracies and training time using logistic regression. Again, with $k \geq 200$ and $b \geq 8$, b-bit minwise hashing can achieve similar test accuracies as using the original data. The training time is substantially reduced, from about 1000 seconds to about 30 seconds only.

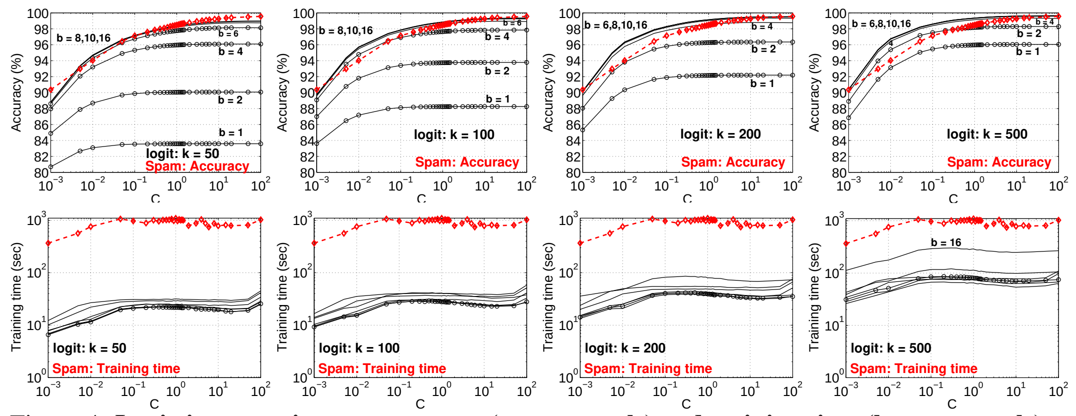


Figure 4: **Logistic regression test accuracy (upper panels) and training time (bottom panels)**.

In summary, it appears b-bit hashing is highly effective in reducing the data size and speeding up the training (and testing), for both SVM and logistic regression. We notice that when using $b = 16$, the training time can be much larger than using $b \leq 8$. Interestingly, we find that b-bit hashing can be easily combined with *Vowpal Wabbit (VW)* [34] to further reduce the training time when b is large.

6 Random Projections, Count-Min (CM) Sketch, and Vowpal Wabbit (VW)

Random projections [1, 24], Count-Min (CM) sketch [11], and Vowpal Wabbit (VW) [32, 34], as popular hashing algorithms for estimating inner products for high-dimensional datasets, are naturally applicable in large-scale learning. In fact, those methods are not limited to binary data. Interestingly, the three methods all have essentially the same variances. Note that in this paper, we use ”VW“ particularly for the hashing algorithm in [34], not the influential “VW” online learning platform.

6.1 Random Projections

Denote the first two rows of a data matrix by $u_1,\ u_2 \in \mathbf{R}^D$. The task is to estimate the inner product $a = \sum_{i=1}^D u_{1,i}u_{2,i}$. The general idea is to multiply the data vectors by a random matrix $\{r_{ij}\} \in \mathbb{R}^{D\times k}$, where r_{ij} is sampled i.i.d. from the following generic distribution with [24]

$$E(r_{ij}) = 0,\ \ Var(r_{ij}) = 1,\ \ E(r_{ij}^3) = 0,\ \ E(r_{ij}^4) = s,\ \ s \geq 1. \tag{8}$$

Note that $Var(r_{ij}^2) = E(r_{ij}^4) - E^2(r_{ij}^2) = s - 1 \geq 0$. This generates two k-dim vectors, v_1 and v_2:

$$v_{1,j} = \sum_{i=1}^D u_{1,i}r_{ij}, \qquad v_{2,j} = \sum_{i=1}^D u_{2,i}r_{ij}, \quad j = 1, 2, ..., k \tag{9}$$

The general family of distributions (8) includes the standard normal distribution (in this case, $s = 3$) and the "sparse projection" distribution specified as $r_{ij} = \sqrt{s} \times \begin{cases} 1 & \text{with prob. } \frac{1}{2s} \\ 0 & \text{with prob. } 1 - \frac{1}{s} \\ -1 & \text{with prob. } \frac{1}{2s} \end{cases}$

[24] provided the following unbiased estimator $\hat{a}_{rp,s}$ of a and the general variance formula:

$$\hat{a}_{rp,s} = \frac{1}{k}\sum_{j=1}^{k} v_{1,j}v_{2,j}, \qquad E(\hat{a}_{rp,s}) = a = \sum_{i=1}^{D} u_{1,i}u_{2,i}, \tag{10}$$

$$Var(\hat{a}_{rp,s}) = \frac{1}{k}\left[\sum_{i=1}^{D} u_{1,i}^2 \sum_{i=1}^{D} u_{2,i}^2 + a^2 + (s-3)\sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2\right] \tag{11}$$

which means $s = 1$ achieves the smallest variance. The only elementary distribution we know that satisfies (8) with $s = 1$ is the two point distribution in $\{-1, 1\}$ with equal probabilities.

[23] proposed an improved estimator for random projections as the solution to a cubic equation. Because it can not be written as an inner product, that estimator can not be used for linear learning.

6.2 Count-Min (CM) Sketch and Vowpal Wabbit (VW)

Again, in this paper, "VW" always refers to the hashing algorithm in [34]. VW may be viewed as a "bias-corrected" version of the Count-Min (CM) sketch [11]. In the original CM algorithm, the key step is to independently and uniformly hash elements of the data vectors to k buckets and the hashed value is the sum of the elements in the bucket. That is $h(i) = j$ with probability $\frac{1}{k}$, where $j \in \{1, 2, ..., k\}$. By writing $I_{ij} = \begin{cases} 1 & \text{if } h(i) = j \\ 0 & \text{otherwise} \end{cases}$, we can write the hashed data as

$$w_{1,j} = \sum_{i=1}^{D} u_{1,i}I_{ij}, \qquad w_{2,j} = \sum_{i=1}^{D} u_{2,i}I_{ij} \tag{12}$$

The estimate $\hat{a}_{cm} = \sum_{j=1}^{k} w_{1,j}w_{2,j}$ is (severely) biased for estimating inner products. The original paper [11] suggested a "count-min" step for positive data, by generating multiple independent estimates $\hat{a}_{cm}$ and taking the minimum as the final estimate. That step can reduce but can not remove the bias. Note that the bias can be easily removed by using $\frac{k}{k-1}\left(\hat{a}_{cm} - \frac{1}{k}\sum_{i=1}^{D} u_{1,i}\sum_{i=1}^{D} u_{2,i}\right)$.

[34] proposed a creative method for bias-correction, which consists of pre-multiplying (element-wise) the original data vectors with a random vector whose entries are sampled i.i.d. from the two-point distribution in $\{-1, 1\}$ with equal probabilities. Here, we consider the general distribution (8). After applying multiplication and hashing on u_1 and u_2, the resultant vectors g_1 and g_2 are

$$g_{1,j} = \sum_{i=1}^{D} u_{1,i}r_iI_{ij}, \qquad g_{2,j} = \sum_{i=1}^{D} u_{2,i}r_iI_{ij}, \quad j = 1, 2, ..., k \tag{13}$$

where $E(r_i) = 0,\ E(r_i^2) = 1,\ E(r_i^3) = 0,\ E(r_i^4) = s$. We have the following Lemma.

Theorem 3

$$\hat{a}_{vw,s} = \sum_{j=1}^{k} g_{1,j}g_{2,j}, \qquad E(\hat{a}_{vw,s}) = \sum_{i=1}^{D} u_{1,i}u_{2,i} = a, \tag{14}$$

$$Var(\hat{a}_{vw,s}) = (s-1)\sum_{i=1}^{D} u_{1,i}^2u_{2,i}^2 + \frac{1}{k}\left[\sum_{i=1}^{D} u_{1,i}^2 \sum_{i=1}^{D} u_{2,i}^2 + a^2 - 2\sum_{i=1}^{D} u_{1,i}^2u_{2,i}^2\right] \ \Box \tag{15}$$

Interestingly, the variance (15) says we do need $s = 1$, otherwise the additional term $(s - 1)\sum_{i=1}^{D} u_{1,i}^2u_{2,i}^2$ will not vanish even as the sample size $k \to \infty$. In other words, the choice of random distribution in VW is essentially the only option if we want to remove the bias by pre-multiplying the data vectors (element-wise) with a vector of random variables. Of course, once we let $s = 1$, the variance (15) becomes identical to the variance of random projections (11).

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

We implemented VW and experimented it on the same webspam dataset. Figure 5 shows that b-bit minwise hashing is substantially more accurate (at the same sample size k) and requires significantly less training time (to achieve the same accuracy). Basically, for 8-bit minwise hashing with $k = 200$ achieves similar test accuracies as VW with $k = 10^4 \sim 10^6$ (note that we only stored the non-zeros).

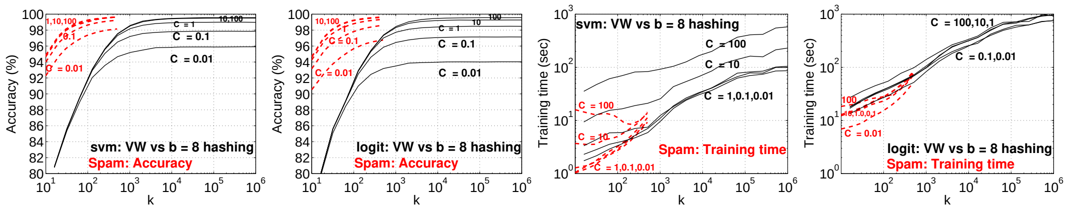


Figure 5: The dashed (red if color is available) curves represent b-bit minwise hashing results (only for $k \leq 500$) while solid curves for VW. We display results for $C = 0.01, 0.1, 1, 10, 100$.

This empirical finding is not surprising, because the variance of b-bit hashing is usually substantially smaller than the variance of VW (and random projections). In the technical report (arXiv:1106.0967, which also includes the complete proofs of the theorems presented in this paper), we show that, at the same storage cost, b-bit hashing usually improves VW by 10- to 100-fold, by assuming each sample of VW needs 32 bits to store. Of course, even if VW only stores each sample using 16 bits, an improvement of 5- to 50-fold would still be very substantial.

There is one interesting issue here. Unlike random projections (and minwise hashing), VW is a *sparsity-preserving* algorithm, meaning that in the resultant sample vector of length k, the number of non-zeros will not exceed the number of non-zeros in the original vector. In fact, it is easy to see that the fraction of zeros in the resultant vector would be (at least) $\left(1 - \frac{1}{k}\right)^c \approx \exp\left(-\frac{c}{k}\right)$, where c is the number of non-zeros in the original data vector. In this paper, we mainly focus on the scenario in which $c \gg k$, i.e., we use b-bit minwise hashing or VW for the purpose of *data reduction*.

However, in some cases, we care about $c \ll k$, because VW is also an excellent tool for *compact indexing*. In fact, our b-bit minwise hashing scheme for linear learning may face such an issue.

8 Combining b-Bit Minwise Hashing with VW

In Figures 3 and 4, when $b = 16$, the training time becomes substantially larger than $b \leq 8$. Recall that in the run-time, we expand the b-bit minwise hashed data to sparse binary vectors of length $2^b k$ with exactly k 1's. When $b = 16$, the vectors are very sparse. On the other hand, once we have expanded the vectors, the task is merely computing inner products, for which we can use VW.

Therefore, in the run-time, after we have generated the sparse binary vectors of length $2^b k$, we hash them using VW with sample size m (to differentiate from k). How large should m be? Theorem 4 may provide an insight. Recall Section 2 provides the estimator, denoted by $\hat{R}_b$, of the resemblance R, using b-bit minwise hashing. Now, suppose we first apply VW hashing with size m on the binary vector of length $2^b k$ before estimating R, which will introduce some additional randomness. We denote the new estimator by $\hat{R}_{b,vw}$. Theorem 4 provides its theoretical variance.

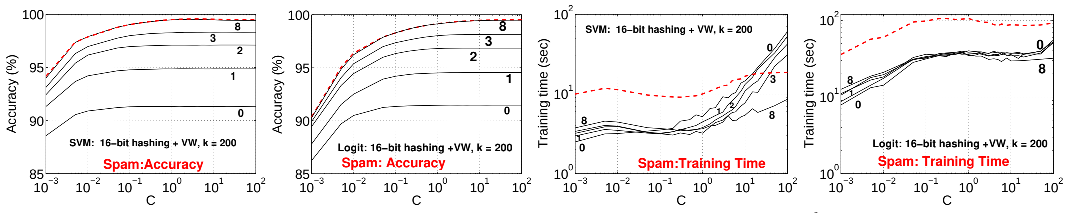


Figure 6: We apply VW hashing on top of the binary vectors (of length $2^b k$) generated by b-bit hashing, with size $m = 2^0 k, 2^1 k, 2^2 k, 2^3 k, 2^8 k$, for $k = 200$ and $b = 16$. The numbers on the solid curves (0, 1, 2, 3, 8) are the exponents. The dashed (red if color if available) curves are the results from only using b-bit hashing. When $m = 2^8 k$, this method achieves similar test accuracies (left panels) while substantially reducing the training time (right panels).

Theorem 4

$$Var\left(\hat{R}_{b,vw}\right) = Var\left(\hat{R}_b\right) + \frac{1}{m}\frac{1}{\left[1-C_{2,b}\right]^2}\left(1+P_b^2-\frac{P_b(1+P_b)}{k}\right), \tag{16}$$

where $Var\left(\hat{R}_b\right) = \frac{1}{k}\frac{P_b(1-P_b)}{[1-C_{2,b}]^2}$ *is given by (4) and* $C_{2,b}$ *is the constant defined in Theorem 1.* $\Box$

Compared to the original variance $Var\left(\hat{R}_b\right)$, the additional term in (16) can be relatively large, if m is small. Therefore, we should choose $m \gg k$ and $m \ll 2^b k$. If $b = 16$, then $m = 2^8 k$ may be a good trade-off. Figure 8 provides an empirical study to verify this intuition.

9 Limitations

While using b-bit minwise hashing for training linear algorithms is successful on the *webspam* dataset, it is important to understand the following **three** major limitations of the algorithm:

(A): Our method is designed for binary (0/1) sparse data. (B): Our method requires an expensive preprocessing step for generating k permutations of the data. For most applications, we expect the preprocessing cost is not a major issue because the preprocessing can be conducted off-line (or combined with the data-collection step) and is easily parallelizable. However, even if the speed is not a concern, the energy consumption might be an issue, especially considering (b-bit) minwise hashing is mainly used for industry applications. In addition, testing an new unprocessed data vector (e.g., a new document) will be expensive. *(C): Our method performs only reasonably well in terms of dimension reduction.* The processed data need to be mapped into binary vectors in $2^b \times k$ dimensions, which is usually not small. (Note that the storage cost is just bk bits.) For example, for the *webspam* dataset, using $b = 8$ and $k = 200$ seems to suffice and $2^8 \times 200 = 51200$ is quite large, although it is much smaller than the original dimension of 16 million. It would be desirable if we can further reduce the dimension, because the dimension determines the storage cost of the model and (moderately) increases the training time for batch learning algorithms such as LIBLINEAR.

In hopes of fixing the above limitations, we experimented with an implementation using another hashing technique named *Conditional Random Sampling (CRS)* [21, 22], which is not limited to binary data and requires only one permutation of the original data (i.e., no expensive preprocessing). We achieved some limited success. For example, CRS compares favorably to VW in terms of storage (to achieve the same accuracy) on the *webspam* dataset. However, so far CRS can not compete with b-bit minwise hashing for linear learning (in terms of training speed, storage cost, and model size). The reason is because even though the estimator of CRS is an inner product, the normalization factors (i.e, the effective sample size of CRS) to ensure unbiased estimates substantially differ pairwise (which is a significant advantage in other applications). In our implementation, we could not use fully correct normalization factors, which lead to severe bias of the inner product estimates and less than satisfactory performance of linear learning compared to b-bit minwise hashing.

10 Conclusion

As data sizes continue to grow faster than the memory and computational power, statistical learning tasks in industrial practice are increasingly faced with training datasets that exceed the resources on a single server. A number of approaches have been proposed that address this by either scaling out the training process or partitioning the data, but both solutions can be expensive.

In this paper, we propose a compact representation of sparse, binary data sets based on b-bit minwise hashing, which can be naturally integrated with linear learning algorithms such as linear SVM and logistic regression, leading to dramatic improvements in training time and/or resource requirements. We also compare b-bit minwise hashing with the Count-Min (CM) sketch and Vowpal Wabbit (VW) algorithms, which, according to our analysis, all have (essentially) the same variances as random projections [24]. Our theoretical and empirical comparisons illustrate that b-bit minwise hashing is significantly more accurate (at the same storage) for binary data. There are various limitations (e.g., expensive preprocessing) in our proposed method, leaving ample room for future research.

Acknowledgement

This work is supported by NSF (DMS-0808864), ONR (YIP-N000140910911), and a grant from Microsoft. We thank John Langford and Tong Zhang for helping us better understand the VW hashing algorithm, and Chih-Jen Lin for his patient explanation of LIBLINEAR package and datasets.

References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.

[2] 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.

[3] Harald Baayen. *Word Frequency Distributions*, volume 18 of *Text, Speech and Language Technology*. Kulver Academic Publishers, 2001.

[4] Michael Bendersky and W. Bruce Croft. Finding text reuse on the web. In *WSDM*, pages 262–271, Barcelona, Spain, 2009.

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

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

[7] 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.

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

[9] Ludmila Cherkasova, Kave Eshghi, Charles B. Morrey III, Joseph Tucek, and Alistair C. Veitch. Applying syntactic similarity algorithms for enterprise information management. In *KDD*, pages 1087–1096, Paris, France, 2009.

[10] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD*, pages 219–228, Paris, France, 2009.

[11] 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.

[12] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense implicit communities in the web graph. *ACM Trans. Web*, 3(2):1–36, 2009.

[13] 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.

[14] 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.

[15] George Forman, Kave Eshghi, and Jaap Suermondt. Efficient detection of large-scale redundancy in enterprise file systems. *SIGOPS Oper. Syst. Rev.*, 43(1):84–91, 2009.

[16] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, Madrid, Spain, 2009.

[17] Matthias Hein and Olivier Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *AISTATS*, pages 136–143, Barbados, 2005.

[18] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, ICML, pages 408–415, 2008.

[19] Yugang Jiang, Chongwah Ngo, and Jun Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *CIVR*, pages 494–501, Amsterdam, Netherlands, 2007.

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

[21] 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).

[22] Ping Li, Kenneth W. Church, and Trevor J. Hastie. Conditional random sampling: A sketch-based sampling technique for sparse data. In *NIPS*, pages 873–880, Vancouver, BC, Canada, 2006 (Newer results appeared in NIPS 2008.

[23] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Improving random projections using marginal information. In *COLT*, pages 635–649, Pittsburgh, PA, 2006.

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

[25] Ping Li and Arnd Christian König. Theory and applications b-bit minwise hashing. In *Commun. ACM*, 2011.

[26] Ping Li and Arnd Christian König. Accurate estimators for improving minwise hashing and b-bit minwise hashing. Technical report, 2011 (arXiv:1108.0895).

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

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

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

[30] Marc Najork, Sreenivas Gollapudi, and Rina Panigrahy. Less is more: sampling the neighborhood graph makes salsa better and faster. In *WSDM*, pages 242–251, Barcelona, Spain, 2009.

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

[32] 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.

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

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

[35] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. In *KDD*, pages 833–842, 2010.

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

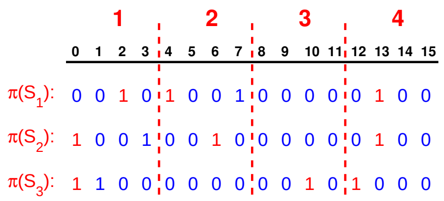

1
2
3
4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
π(S1): 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0
π(S2): 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0
π(S3): 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0

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.

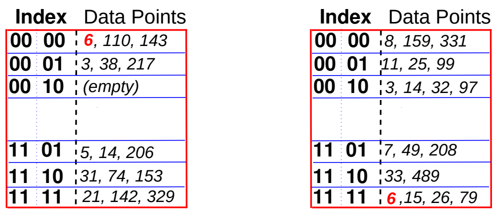
2 Applications of Minwise Hashing on Efficient Search and Learning

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):

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

The L_2-regularized linear SVM solves a similar problem:

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

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.

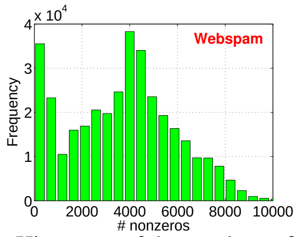


Figure 3: Histogram of the numbers of nonzeros in the *webspam* dataset (350,000 samples).

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

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

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.

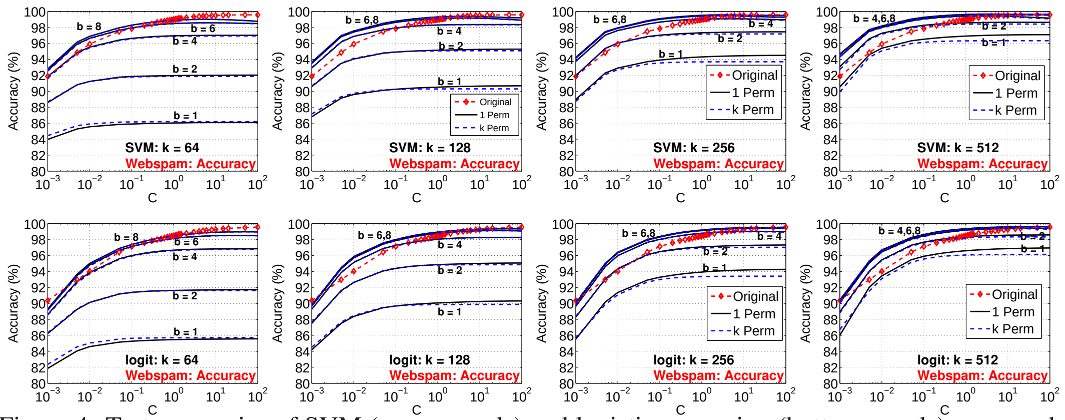


Figure 4: Test accuracies of SVM (upper panels) and logistic regression (bottom panels), averaged over 50 repetitions. The accuracies of using the original data are plotted as dashed (red, if color is available) curves with "diamond" markers. C is the regularization parameter. Compared with the original 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., $k \le 64$) both the one permutation scheme and the original k-permutation scheme perform similarly. For larger k values (especially as $k \ge 256$), 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 $k \ge 512$ and $b = 8$ essentially achieves the original test accuracies, while the k-permutation scheme could only reach about 97.5%.

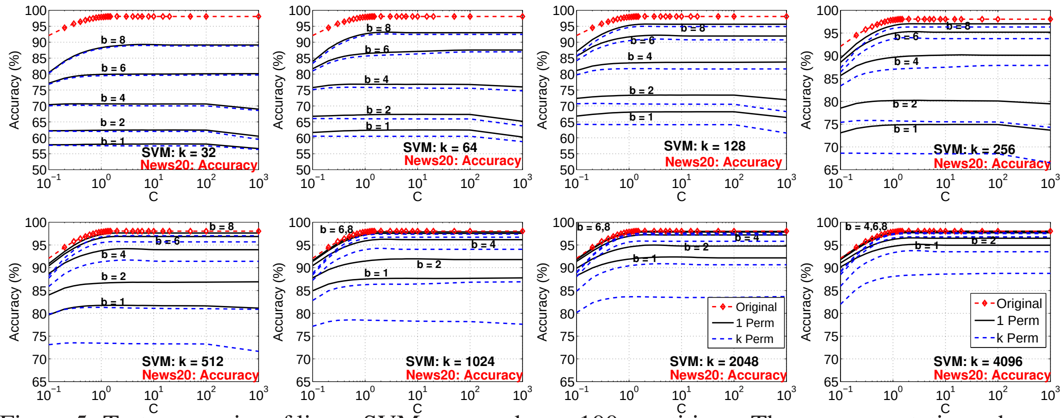


Figure 5: Test accuracies of linear SVM averaged over 100 repetitions. The one permutation scheme noticeably outperforms the original k-permutation scheme especially when k is not small.

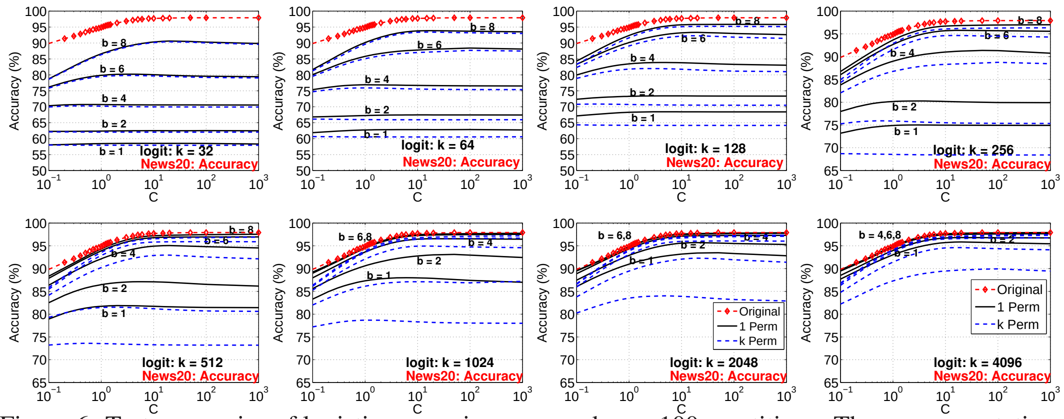


Figure 6: Test accuracies of logistic regression averaged over 100 repetitions. The one permutation scheme noticeably outperforms the original k-permutation scheme especially when k is not small.

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.

One Permutation Hashing for Efficient Search and Learning

Ping Li
Dept. of Statistical Science
Cornell University
Ithaca, NY 14853
pingli@cornell.edu

Art Owen
Dept. of Statistics
Stanford University
Stanford, CA 94305
owen@stanford.edu

Cun-Hui Zhang
Dept. of Statistics
Rutgers University
New Brunswick, NJ 08901
czhang@stat.rutgers.edu

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, the method of b-bit minwise hashing has been applied to large-scale linear learning (e.g., linear SVM or logistic regression) and sublinear time near-neighbor search. The major drawback of minwise hashing is the expensive preprocessing cost, as the method requires applying (e.g.,) $k = 200$ to 500 permutations on the data. The testing time can also be expensive if a new data point (e.g., a new document or image) has not been processed, which might be a significant issue in user-facing applications. While it is true that the preprocessing step can be parallelized, it comes at the cost of additional hardware & implementation and is not an energy-efficient solution.

We develop a very simple solution based on **one permutation hashing**. Conceptually, given a massive binary data matrix, we permute the columns only once and divide the permuted columns evenly into k bins; and we simply store, for each data vector, the smallest nonzero location in each bin. The interesting probability analysis (which is validated by experiments) reveals that our one permutation scheme should perform very similarly to the original (k-permutation) minwise hashing. In fact, the one permutation scheme can be even slightly more accurate, due to the "sample-without-replacement" effect.

Our experiments with training linear SVM and logistic regression on the *webspam* dataset demonstrate that this one permutation hashing scheme can achieve the same (or even slightly better) accuracies compared to the original k-permutation scheme. To test the robustness of our method, we also experiment with the small *news20* dataset which is very sparse and has merely on average 500 nonzeros in each data vector. Interestingly, our one permutation scheme noticeably outperforms the k-permutation scheme when k is not too small on the *news20* dataset. In summary, our method can achieve at least the same accuracy as the original k-permutation scheme, at merely $1/k$ of the original preprocessing cost.

1 Introduction

Minwise hashing [4, 3] is a standard technique for efficiently computing set similarities, especially in the context of search. Recently, b-bit minwise hashing [17], which stores only the lowest b bits of each hashed value, has been applied to sublinear time near neighbor search [21] and linear learning (linear SVM and logistic regression) [18], on large-scale high-dimensional binary data (e.g., text), which are common in practice. The major drawback of minwise hashing and b-bit minwise hashing is that they require an expensive preprocessing step, by conducting k (e.g., 200 to 500) permutations on the entire dataset.

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 seems sufficient to set the total dimensionality to be $D = 2^{64}$, for convenience. Text data generated by w-shingles are often treated as binary. In fact, for $w \geq 3$, it is expected that most of the w-shingles will occur at most one time in a document. Also, note that the idea of shingling can be naturally extended to images in Computer Vision, either at the pixel level (for simple aligned images) or at the Vision feature level [22].

In machine learning practice, the use of extremely high-dimensional data has become common. For example, [23] discusses training datasets with (on average) $n = 10^{11}$ items and $D = 10^9$ distinct features. [24] experimented with a dataset of potentially $D = 16$ trillion (1.6×10^{13}) unique features.

1.2 Minwise Hashing

Minwise hashing is mainly designed for binary data. A binary (0/1) data vector can be equivalently 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 to be $D = 2^{64}$ in industrial applications. The similarity between two sets S_1 and S_2 is commonly measured by the *resemblance*, which is a normalized version of the 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 the sets after applying the permutation π on S_1 and 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 [18]:

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

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 [17] provides a simple solution by storing only the lowest b bits of each hashed data. This way, the dimensionality of the expanded data matrix from the hashed data would be only $2^b \times k$ as opposed to $2^{64} \times k$. [18] applied this idea to large-scale learning on the *webspam* dataset (with about 16 million features) and demonstrated that using $b = 8$ and $k = 200$ to 500 could achieve very similar accuracies as using the original data. More recently, [21] directly used the bits generated by b-bit minwise hashing for building hash tables to achieve sublinear time near neighbor search. We will briefly review these two important applications in Sec. 2. Note that both applications require the hashed data to be "aligned" in that only the hashed data generated by the same permutation are interacted. For example, when computing the inner products, we simply concatenate the results from k permutations.

1.3 The Cost of Preprocessing and Testing

Clearly, the preprocessing step of minwise hashing can be very costly. For example, in our experiments, loading the *webspam* dataset (350,000 samples, about 16 million features, and about 24GB in Libsvm/svmlight format) used in [18] took about 1000 seconds when the data are stored in Libsvm/svmlight (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 (which is $\gg 150$). Note that, compared to industrial applications [23], 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 processed, then the cost will be expensive if it includes the preprocessing cost. 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

As illustrated in Figure 1, the idea of *one permutation hashing* is very 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.

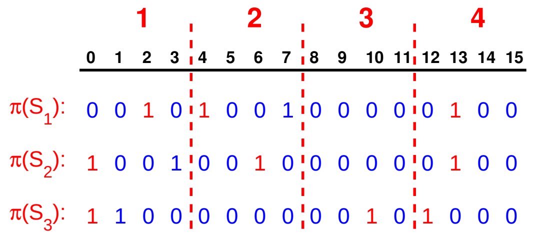

1
2
3
4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
π(S1): 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0
π(S2): 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0
π(S3): 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0

Figure 1: **Fixed-length hashing scheme**. Consider $S_1, S_2, S_3 \subseteq \Omega = \{0, 1, ..., 15\}$ (i.e., $D = 16$). We apply one permutation π on the three 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 three samples: $[2,\ 0,\ *,\ 1]$, $[0,\ 2,\ *,\ 1]$, and $[0,\ *,\ 2,\ 0]$. For now, we use '*' for empty bins, which occur rarely unless the number of nonzeros is small compared to k.

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 $[2-4\times 0, 4-4\times 1, *, 13-4\times 3] = [2, 0, *, 1]$. Similarly, for $\pi(S_2)$, the original sample $[0, 6, *, 13]$ becomes $[0, 6-4\times 1, *, 13-4\times 3] = [0, 2, *, 1]$, etc.

Note that, when there are no empty bins, similarity estimation is equivalent to computing an inner product, which is crucial for taking advantage of the modern linear learning algorithms [13, 19, 7, 11]. 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 Summary of the 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 highly desirable, especially considering that minwise hashing is deployed in the search industry.
- While it is true that the preprocessing can be parallelized, it comes at the cost of additional hardware and software implementation.
- 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.
- It should be much easier to implement the one permutation hashing than the original k-permutation scheme, 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 using universal hashing [5] to approximate permutations. Universal hashing often works well in practice although theoretically there are always worst cases. Of course, when $D = 2^{64}$, we have to use universal hashing, but it is always much easier to generate just one permutation.
- One permutation hashing is a better matrix sparsification scheme than the original k-permutation. In terms of the original binary data matrix, the one permutation scheme simply makes many nonzero entries be zero, without further "damaging" the original data matrix. With the original 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 will significantly change the structure of the original data matrix. As a consequence, we expect that our one permutation scheme will produce at least the same or even more accurate results, as later verified by experiments.

1.6 Related Work

One of the authors worked on another "one permutation" scheme named *Conditional Random Sampling (CRS)* [14, 15] since 2005. Basically, CRS works by continuously taking the first 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. In comparison, our new one permutation scheme works by first breaking the columns evenly into k bins and then taking the first nonzero in each bin, so that the hashed data can be nicely aligned.

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. After the authors of [4] realized that the estimators could not be written as an inner product and hence the scheme was not suitable for many applications such as sublinear time near neighbor search using hash tables, they quickly moved to the k-permutation minwise hashing scheme [3]. In the context of large-scale linear learning, the importance of having estimators which are inner products should become more obvious after [18] introduced the idea of using (b-bit) minwise hashing for linear learning.

We are also inspired by the work on "very sparse random projections" [16]. The regular random projection method also has the expensive preprocessing cost as it needs k projections. The work of [16] 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.[1]

Figure 1 presents the "fixed-length" scheme, while in Sec. 7 we will also develop a "variable-length" scheme. Two schemes are more or less equivalent, although we believe 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) [20, 24] hashing algorithms.

2 Applications of Minwise Hashing on Efficient Search and Learning

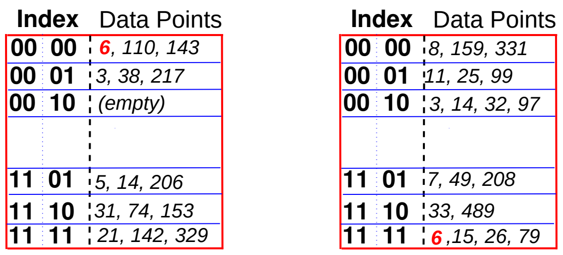
In this section, we will briefly review two important applications of the original (k-permutation) minwise hashing: (i) sublinear time near neighbor search [21], and (ii) large-scale linear learning [18].

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. Efficient algorithms for near neighbor search have numerous applications in the context of search, databases, machine learning, recommending systems, computer vision, etc. It 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)* [12, 1]. The performance of LSH solely depends on its underlying implementation. The idea in [21] is to directly use the bits generated by (b-bit) minwise hashing to construct hash tables, which allow us to search near neighbors in sublinear time (i.e., no need to scan all data points).

Specifically, we hash the data points using k random permutations and store each hash value using b bits (e.g., $b \leq 4$). For each data point, we concatenate the resultant $B = b \times k$ bits as a *signature*. The size of the space is $2^B = 2^{b\times k}$, which is not too large for small b and k (e.g., $bk = 16$). This way, we create a table of 2^B buckets, numbered from 0 to $2^B - 1$; 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 hash table will likely miss many true near neighbors, as a remedy, we generate (using independent random permutations) L hash tables. The query result is the union of the 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

[1]See `http://www.stanford.edu/group/mmds/slides2012/s-pli.pdf` for the experimental results on clustering/classification/regression using very sparse random projections [16].

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 $\{6, 15, 26, 79, 110, 143\}$, which is much smaller than the set of n data points.

The experiments in [21] confirmed that this very simple strategy performed well.

2.2 Large-Scale Linear Learning

The recent development of highly efficient linear learning algorithms (such as linear SVM and logistic regression) is a major breakthrough in machine learning. Popular software packages include SVM^{perf} [13], Pegasos [19], 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:

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

where $C > 0$ is the regularization parameter. The L_2-regularized linear SVM solves a similar problem:

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

In their approach [18], 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, [18] 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 vector of length $2^b k = 12$, to be $\{0, 0, 1, 0,\ \ 0, 0, 0, 1,\ \ 1, 0, 0, 0\}$, which will be the new feature vector fed to a solver such as LIBLINEAR. The procedure for this feature vector is summarized as follows:

Original hashed values $(k = 3)$:	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
New feature vector fed to a solver :		$[0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0] \times \frac{1}{\sqrt{k}}$	

The same procedure (with the same $k = 3$ permutations) is then applied to all n feature vectors. Very interestingly, we notice that the all-zero vector (0000 in this example) is never used when expanding the data. In our one permutation hashing scheme, we will actually take advantage of the all-zero vector to conveniently encode empty bins, a strategy which we will later refer to as the "**zero coding**" strategy.

The experiments in [18] confirmed that this simple procedure performed well.

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 compared with each other. With our one permutation scheme as presented in Figure 1, the hashed data are indeed aligned according to the bin numbers. The only caveat is that we need a practical strategy to deal with empty bins, although they occur rarely unless the number of nonzeros in one data vector is small compared to k, the number of bins.

3 Theoretical Analysis of the Fixed-Length One Permutation Scheme

While the one permutation hashing scheme, as demonstrated in Figure 1, is intuitive, we present in this section some interesting probability analysis to provide a rigorous theoretical foundation for this method. Without loss of generality, we 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

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

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

Later we will also use $I^{(1)}_{emp,j}$ (or $I^{(2)}_{emp,j}$) to indicate whether $\pi(S_1)$ (or $\pi(S_2)$) is empty in the j-th bin.

3.1 Expectation, Variance, and Distribution of the Number of Jointly Empty Bins

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

$$\frac{E\left(N_{emp}\right)}{k} = \prod_{j=0}^{f-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D - j} \leq \left(1 - \frac{1}{k}\right)^f \tag{10}$$

Assume $D\left(1-\frac{2}{k}\right) \geq f = f_1 + f_2 - a$,

$$\begin{aligned} \frac{Var\left(N_{emp}\right)}{k^2} =& \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1 - \frac{E(N_{emp})}{k}\right) \qquad (11) \\ &- \left(1-\frac{1}{k}\right)\left(\left(\prod_{j=0}^{f-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2 - \prod_{j=0}^{f-1} \frac{D\left(1-\frac{2}{k}\right)-j}{D-j}\right) \\ <& \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1 - \frac{E(N_{emp})}{k}\right) \qquad (12) \end{aligned}$$

Proof: *See Appendix A.* □

The inequality (12) says that the variance of $\frac{N_{emp}}{k}$ is smaller than its "binomial analog."

In practical scenarios, the data are often sparse, i.e., $f = f_1 + f_2 - a \ll D$. In this case, Lemma 2 illustrates that in (10) the upper bound $\left(1-\frac{1}{k}\right)^f$ 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 $\left(1-\frac{1}{k}\right)^f \approx 0.0067$; if $f/k = 1$, then $\left(1-\frac{1}{k}\right)^f \approx 0.3679$. 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 *Assume $D\left(1-\frac{1}{k}\right) \geq f = f_1 + f_2 - a$.*

$$\frac{E\left(N_{emp}\right)}{k} = \left(1-\frac{1}{k}\right)^f \exp\left(\frac{-D\log\frac{D+1}{D-f+1} + f\left(1-\frac{1}{2(D-f+1)}\right)}{k-1} + ...\right) \tag{13}$$

Under the reasonable assumption that the data are sparse, i.e., $f_1 + f_2 - a = f \ll D$, we obtain

$$\frac{E\left(N_{emp}\right)}{k} = \left(1-\frac{1}{k}\right)^f \left(1 - O\left(\frac{f^2}{kD}\right)\right) \tag{14}$$

$$\begin{aligned}\frac{Var\left(N_{emp}\right)}{k^2} =& \frac{1}{k}\left(1-\frac{1}{k}\right)^f\left(1-\left(1-\frac{1}{k}\right)^f\right) \\ &-\left(1-\frac{1}{k}\right)^{f+1}\left(\left(1-\frac{1}{k}\right)^f - \left(1-\frac{1}{k-1}\right)^f\right) + O\left(\frac{f^2}{kD}\right)\end{aligned} \tag{15}$$

Proof: *See Appendix B.* □

In addition to its mean and variance, we can also write down the distribution of N_{emp}.

Lemma 3

$$\mathbf{Pr}\left(N_{emp} = j\right) = \sum_{s=0}^{k-j}(-1)^s \frac{k!}{j!s!(k-j-s)!}\prod_{t=0}^{f-1}\frac{D\left(1-\frac{j+s}{k}\right)-t}{D-t} \tag{16}$$

Proof: *See Appendix C.* □

Because $E\left(N_{emp}\right) = \sum_{j=0}^{k-1} j\mathbf{Pr}\left(N_{emp}=j\right)$, this yields an interesting combinatorial identity:

$$k\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j} = \sum_{j=0}^{k-1} j \sum_{s=0}^{k-j}(-1)^s\frac{k!}{j!s!(k-j-s)!}\prod_{t=0}^{f-1}\frac{D\left(1-\frac{j+s}{k}\right)-t}{D-t} \tag{17}$$

3.2 Expectation and Variance of the Number of Matched Bins

Lemma 4 *Assume $D\left(1-\frac{1}{k}\right) \geq f = f_1 + f_2 - a$.*

$$\frac{E\left(N_{mat}\right)}{k} = R\left(1-\frac{E\left(N_{emp}\right)}{k}\right) = R\left(1-\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right) \tag{18}$$

Assume $D\left(1-\frac{2}{k}\right) \geq f = f_1 + f_2 - a$.

$$\begin{aligned}\frac{Var(N_{mat})}{k^2} =& \frac{1}{k}\left(\frac{E(N_{mat})}{k}\right)\left(1-\frac{E(N_{mat})}{k}\right) \\ &+\left(1-\frac{1}{k}\right)R\frac{a-1}{f-1}\left(1-2\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j} + \prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D-j}\right) \\ &-\left(1-\frac{1}{k}\right)R^2\left(1-\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2\end{aligned} \tag{19}$$

$$< \frac{1}{k}\left(\frac{E(N_{mat})}{k}\right)\left(1-\frac{E(N_{mat})}{k}\right) \tag{20}$$

Proof: *See Appendix D.* □

3.3 Covariance of N_{mat} and N_{emp}

Intuitively, N_{mat} and N_{emp} should be negatively correlated, as confirmed by the following Lemma:

Lemma 5 *Assume $D\left(1-\frac{2}{k}\right) \geq f = f_1+f_2-a$.*

$$\begin{aligned}\frac{Cov\left(N_{mat},\ N_{emp}\right)}{k^2} =& R\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j} - \prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\\ &-\frac{1}{k}R\left(1-\prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\end{aligned} \tag{21}$$

and

$$Cov\left(N_{mat},\ N_{emp}\right) \leq 0 \tag{22}$$

Proof: *See Appendix E.* □

3.4 An Unbiased Estimator of R and the Variance

Lemma 6 shows the following estimator $\hat{R}_{mat}$ of the resemblance is unbiased:

Lemma 6

$$\hat{R}_{mat} = \frac{N_{mat}}{k-N_{emp}}, \qquad E\left(\hat{R}_{mat}\right) = R \tag{23}$$

$$Var\left(\hat{R}_{mat}\right) = R(1-R)\left(E\left(\frac{1}{k-N_{emp}}\right)\left(1+\frac{1}{f-1}\right)-\frac{1}{f-1}\right) \tag{24}$$

$$E\left(\frac{1}{k-N_{emp}}\right) = \sum_{j=0}^{k-1}\frac{\mathbf{Pr}\left(N_{emp}=j\right)}{k-j} \geq \frac{1}{k-E(N_{emp})} \tag{25}$$

Proof: *See Appendix F. The right-hand side of the inequality (25) is actually a very good approximation (see Figure 8). The exact expression for $\mathbf{Pr}\left(N_{emp}=j\right)$ is already derived in Lemma 3.* □

The fact that $E\left(\hat{R}_{mat}\right) = R$ may seem surprising as in general ratio estimators are not unbiased. Note that $k - N_{emp} > 0$ always 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 somewhat smaller than $\frac{R(1-R)}{k}$, which can be seen from the approximation:

$$\frac{Var\left(\hat{R}_{mat}\right)}{R(1-R)/k} \approx g(f;k) = \frac{1}{1-\left(1-\frac{1}{k}\right)^f}\left(1+\frac{1}{f-1}\right)-\frac{k}{f-1} \tag{26}$$

Lemma 7

$$g(f;k) \leq 1 \tag{27}$$

***Proof**: See Appendix G.* □

It is probably not surprising that our one permutation scheme may (slightly) outperform the original k-permutation scheme (at merely $1/k$ of its preprocessing cost), because one permutation hashing can be viewed as a "sample-without-replacement" scheme.

3.5 Experiments for Validating the Theoretical Results

This set of experiments is for validating the theoretical results. The Web crawl dataset (in Table 1) consists of 15 (essentially randomly selected) pairs of word vectors (in $D = 2^{16}$ dimensions) of a range of similarities and sparsities. For each word vector, the j-th element is whether the word appeared in the j-th Web page.

Table 1: 15 pairs of English words. For example, "RIGHTS" and "RESERVED" correspond to the two sets of document IDs which contained word "RIGHTS" and word "RESERVED" respectively.

Word 1	Word 2	f_1	f_2	$f = f_1 + f_2 - a$	R
RIGHTS	RESERVED	12234	11272	12526	0.877
OF	AND	37339	36289	41572	0.771
THIS	HAVE	27695	17522	31647	0.429
ALL	MORE	26668	17909	31638	0.409
CONTACT	INFORMATION	16836	16339	24974	0.328
MAY	ONLY	12067	11006	17953	0.285
CREDIT	CARD	2999	2697	4433	0.285
SEARCH	WEB	1402	12718	21770	0.229
RESEARCH	UNIVERSITY	4353	4241	7017	0.225
FREE	USE	12406	11744	19782	0.221
TOP	BUSINESS	9151	8284	14992	0.163
BOOK	TRAVEL	5153	4608	8542	0.143
TIME	JOB	12386	3263	13874	0.128
REVIEW	PAPER	3197	1944	4769	0.078
A	TEST	39063	2278	2060	0.052

We vary k from 2^3 to 2^{15}. Although $k = 2^{15}$ is probably way too large in practice, we use it for the purpose of thorough validations. Figures 3 to 8 present the empirical results based on 10^5 repetitions.

3.5.1 $E(N_{emp})$ and $Var(N_{emp})$

Figure 3 and Figure 4 respectively verify $E(N_{emp})$ and $Var(N_{emp})$ as derived in Lemma 1. Clearly, the theoretical curves overlap the empirical curves.

Note that N_{emp} is essentially 0 when k is not large. Roughly when $k/f > 1/5$, the number of empty bins becomes noticeable, which is expected because $E(N_{emp})/k \approx \left(1 - \frac{1}{k}\right)^f \approx e^{-f/k}$ and $e^{-5} = 0.0067$. Practically speaking, as we often use minwise hashing to substantially reduce the number of nonzeros in massive datasets, we would expect that usually $f \gg k$ anyway. See Sec. 4 for more discussion about strategies for dealing with empty bins.

3.5.2 $E(N_{mat})$ and $Var(N_{mat})$

Figure 5 and Figure 6 respectively verify $E(N_{mat})$ and $Var(N_{mat})$ as derived in Lemma 4. Again, the theoretical curves match the empirical ones and the curves start to change shapes at the point where the occurrences of empty bins are more noticeable.

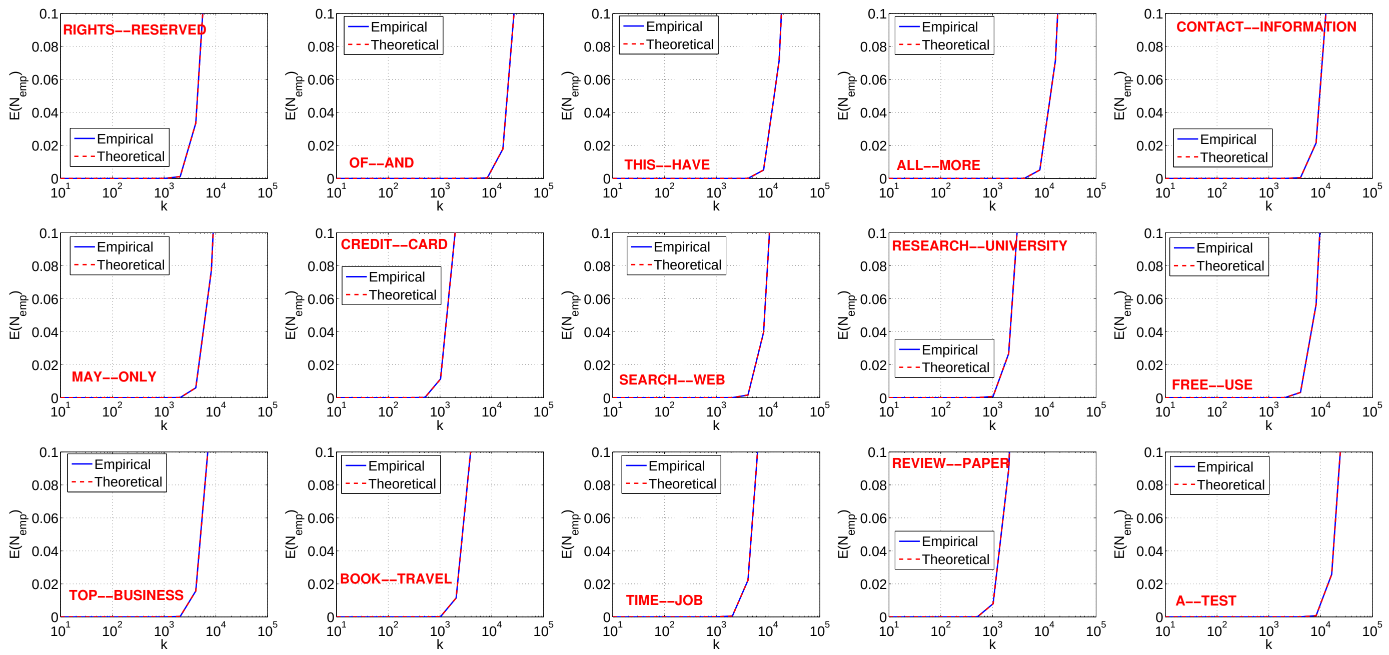


Figure 3: $E(N_{emp})/k$. The empirical curves essentially overlap the theoretical curves as derived in Lemma 1, i.e., (10). The occurrences of empty bins become noticeable only at relatively large sample size k.

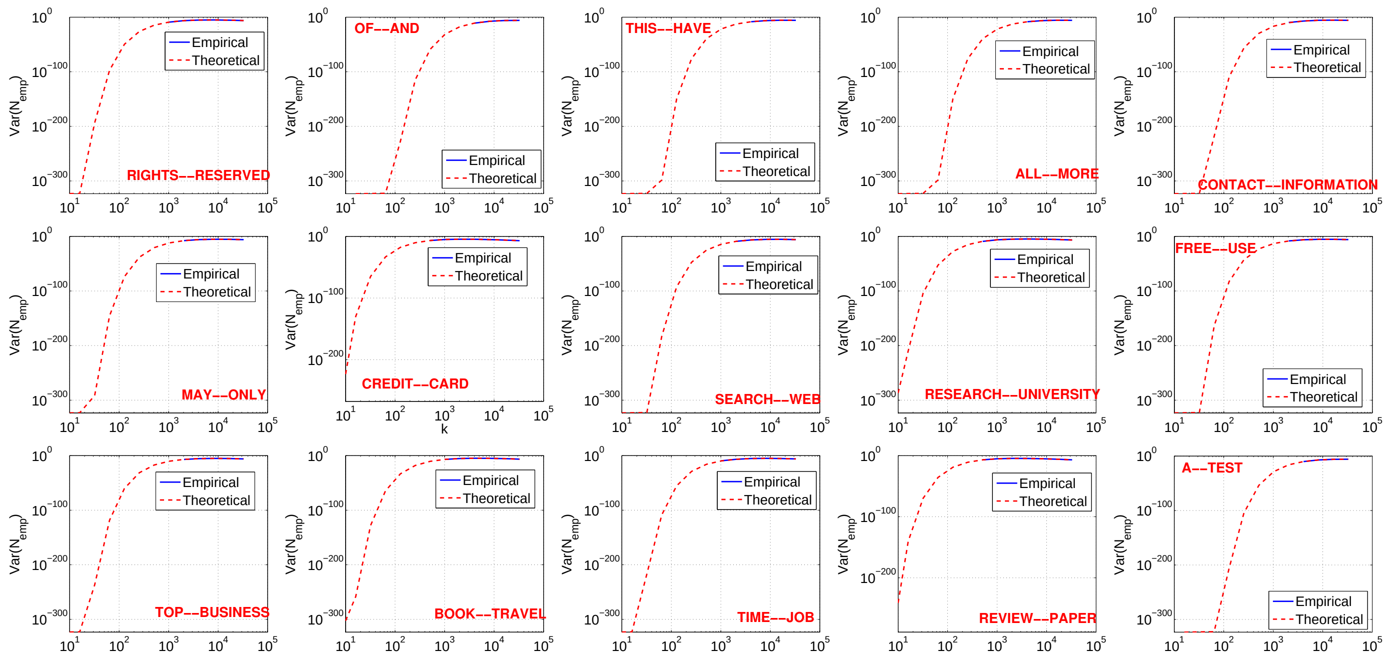


Figure 4: $Var(N_{emp})/k^2$. The empirical curves essentially overlap the theoretical curves as derived in Lemma 1, i.e., (11).

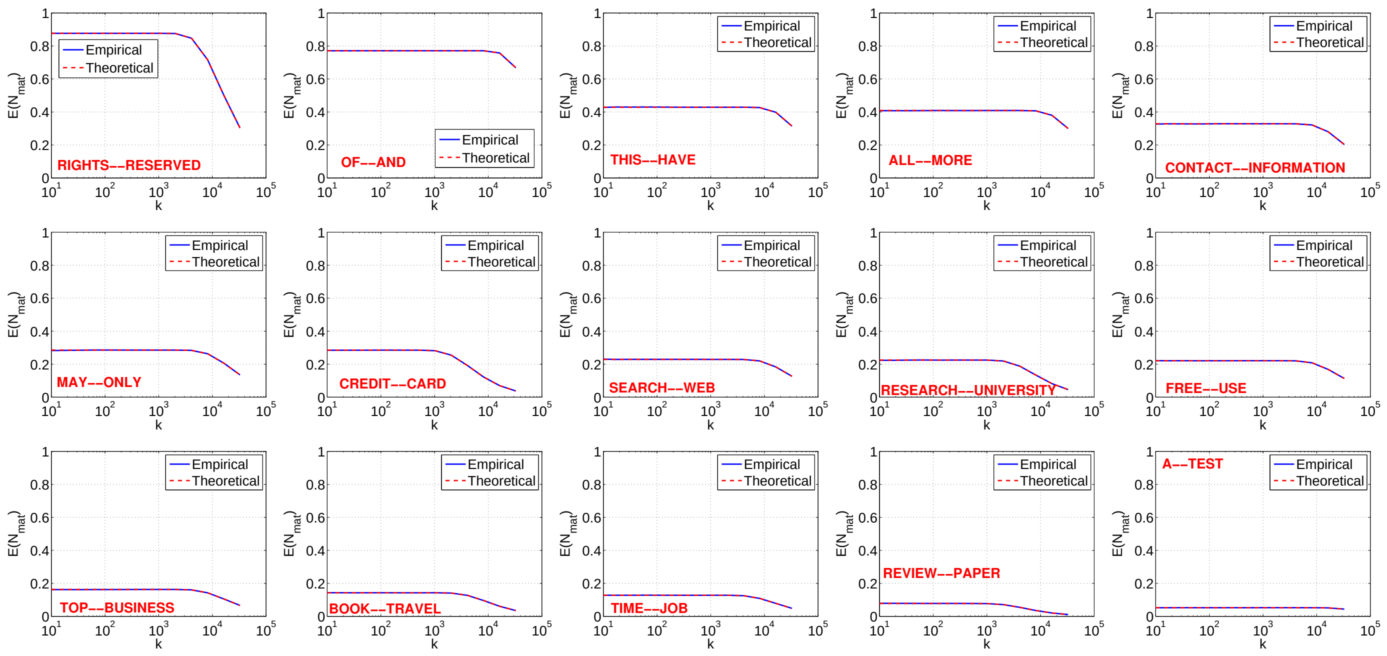


Figure 5: $E(N_{mat})/k$. The empirical curves essentially overlap the theoretical curves as derived in Lemma 4, i.e., (18).

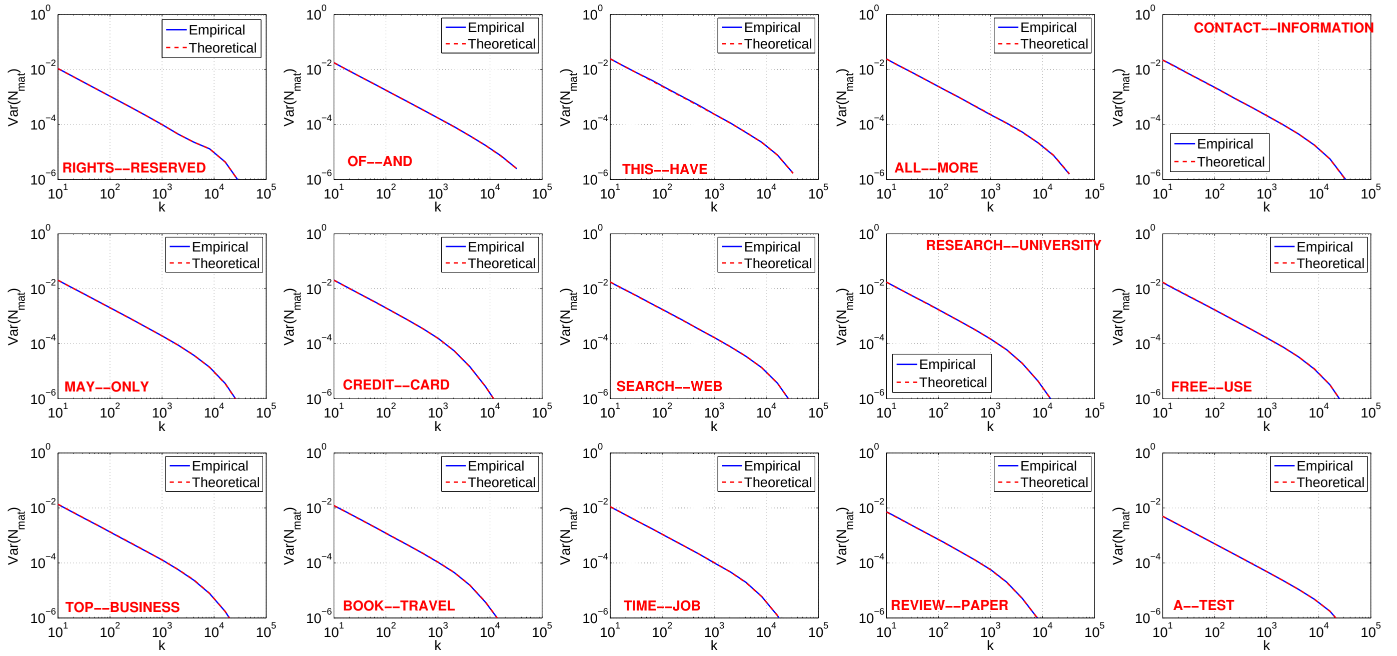


Figure 6: $Var(N_{mat})/k^2$. The empirical curves essentially overlap the theoretical curves as derived in Lemma 4, i.e., (19).

3.5.3 $Cov(N_{emp}, N_{mat})$

To verify Lemma 5, Figure 7 presents the theoretical and empirical covariances of N_{emp} and N_{mat}. Note that $Cov\left(N_{emp},\ N_{mat}\right) \leq 0$ as shown in Lemma 5.

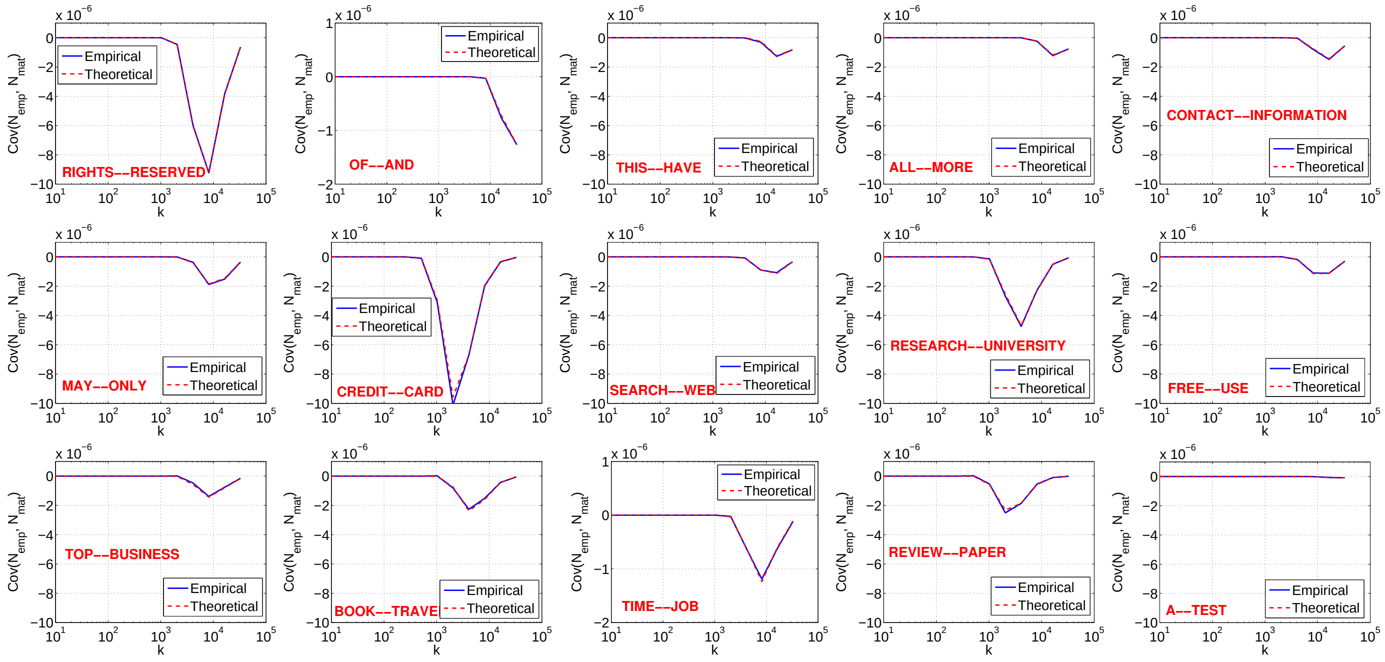


Figure 7: $Cov(N_{emp}, N_{mat})/k^2$. The empirical curves essentially overlap the theoretical curves as derived in Lemma 5, i.e., (21). The experimental results also confirm that the covariance is non-positive as theoretically shown in Lemma 5.

3.5.4 $E(\hat{R}_{mat})$ and $Var(\hat{R}_{mat})$

Finally, Figure 8 plots the empirical MSEs (MSE = bias^2 + variance) and the theoretical variances (24), where the term $E\left(\frac{1}{k-N_{emp}}\right)$ is approximated by $\frac{1}{k-E(N_{emp})}$ as in (25).

The experimental results confirm Lemma 6: (i) the estimator $\hat{R}_{mat}$ is unbiased; (ii) the variance formula (24) and the approximation (25) are accurate; (iii) the variance of $\hat{R}_{mat}$ is somewhat smaller than $R(1-R)/k$, which is the variance of the original k-permutation minwise hashing, due to the "sample-without-replacement" effect.

Remark: The empirical results presented in Figures 3 to 8 have clearly validated the theoretical results for our one permutation hashing scheme. Note that we did not add the empirical results of the original k-permutation minwise hashing scheme because they would simply overlap the theoretical curves. The fact that the original k-permutation scheme provides the unbiased estimate of R with variance $\frac{R(1-R)}{k}$ has been well-validated in prior literature, for example [17].

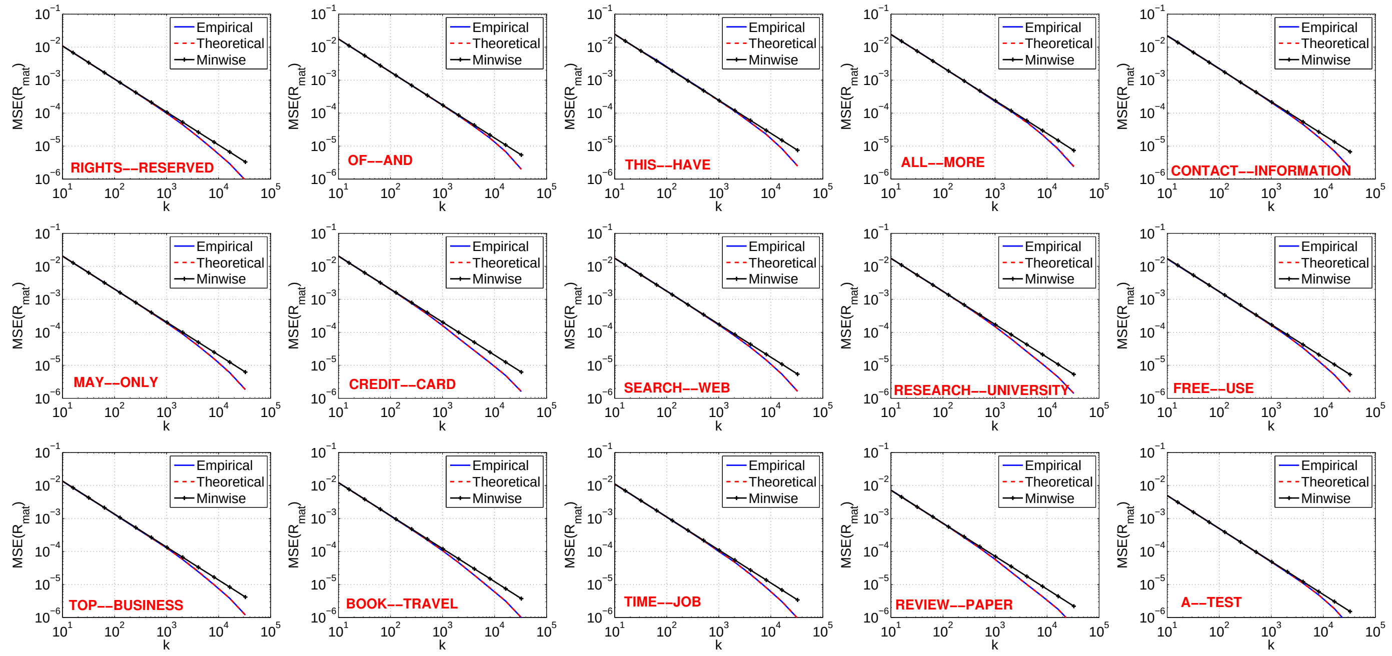


Figure 8: $MSE(\hat{R}_{mat})$, to verify the theoretical results of Lemma 6. Note that the theoretical variance curves use the approximation (25), for convenience. The experimental results confirm that: (i) the estimator $\hat{R}_{mat}$ is unbiased, (ii) the variance formula (24) and the approximation (25) are accurate; (iii) the variance of $\hat{R}_{mat}$ is somewhat smaller than $R(1-R)/k$, the variance of the original k-permutation minwise hashing.

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 very nicely integrated with linear learning algorithms and performs very well.

Figure 9 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 the k (e.g., 200 to 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.

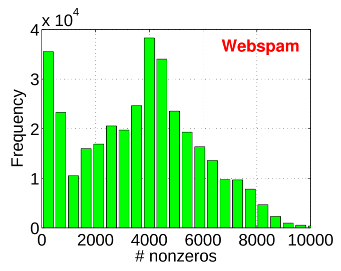


Figure 9: Histogram of the numbers of nonzeros in the *webspam* dataset (350,000 samples).

The first (obvious) idea is **random coding**. That is, we simply replace an empty bin (i.e., "*" as in Figure 1) with a random number. In terms of the original unbiased estimator $\hat{R}_{mat} = \frac{N_{mat}}{k-N_{emp}}$, the random coding scheme will almost not change the numerator N_{mat}. The drawback of random coding is that the denominator will effectively become k. Of course, in most practical scenarios, we expect $N_{emp} \approx 0$ anyway.

The strategy we recommend for linear learning is **zero coding**, which is tightly coupled with the strategy of hashed data expansion [18] 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). A very nice property of this strategy is that it is **sparsity-preserving**. This strategy essentially corresponds to the following modified estimator:

$$\hat{R}_{mat}^{(0)} = \frac{N_{mat}}{\sqrt{k - N_{emp}^{(1)}}\sqrt{k - N_{emp}^{(2)}}} \tag{28}$$

where $N_{emp}^{(1)} = \sum_{j=1}^{k} I_{emp,j}^{(1)}$ and $N_{emp}^{(2)} = \sum_{j=1}^{k} I_{emp,j}^{(2)}$ are the numbers of empty bins in $\pi(S_1)$ and $\pi(S_2)$, respectively. This modified estimator actually makes a lot of sense, after some careful thinking.

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, $N_{emp}^{(1)}$ and $N_{emp}^{(2)}$ are always available. In fact, the use of $\sqrt{k - N_{emp}^{(1)}}\sqrt{k - N_{emp}^{(2)}}$ in the denominator corresponds to the normalizing step which is usually needed before feeding the data to a solver. This point will probably become more clear in Sec. 4.2.

When $N_{emp}^{(1)} = N_{emp}^{(2)} = N_{emp}$, (28) is equivalent to the original $\hat{R}_{mat}$. When two original vectors are very similar (e.g., large R), $N_{emp}^{(1)}$ and $N_{emp}^{(2)}$ will be close to N_{emp}. When two sets are highly unbalanced, using (28) will likely overestimate R; however, in this case, N_{mat} will be so small that the absolute error will not be large. In any case, we do not expect the existence of empty bins will significantly affect the performance in practical settings.

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

In case some readers 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 has already reviewed the data-expansion strategy used by [18] 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

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

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 feature vector.

We believe zero coding is an ideal strategy for dealing with empty bins in the context of linear learning as it is very convenient and produces accurate results (as we will show by experiments). If we use the "random coding" strategy (i.e., replacing a "*" by a random number in $[0,\ 2^b - 1]$), we need to add artificial nonzeros (in the expanded space) and the normalizing factor is always $\frac{1}{\sqrt{k}}$ (i.e., no longer "sparsity-preserving").

We apply both the zero coding and random coding strategies on the *webspam* dataset, as presented in Sec. 5 Basically, both strategies produce similar results even when $k = 512$, although the zero coding strategy is slightly better. We also compare the results with the original k-permutation scheme. On the *webspam* dataset, our one permutation scheme achieves similar (or even slightly better) accuracies compared to the k-permutation scheme.

To test the robustness of one permutation hashing, we also experiment with the *news20* dataset, which has only 20,000 samples and 1,355,191 features, with merely about 500 nonzeros per feature vector on average. We purposely let k be as large as 4096. Interestingly, the experimental results show that the zero coding strategy can perform extremely well. The test accuracies consistently improve as k increases. In comparisons, the random coding strategy performs badly unless k is small (e.g., $k \leq 256$).

On the *news20* dataset, our one permutation scheme actually outperforms the original k-permutation scheme, quite noticeably when k is large. This should be due to the benefits from the "sample-without-replacement" effect. One permutation hashing provides a good matrix sparsification scheme without "damaging" the original data matrix too much.

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 9. Following [18], 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 $k \in \{2^5, 2^6, 2^7, 2^8, 2^9\}$ and $b \in \{1, 2, 4, 6, 8\}$. For convenience, we use $D = 2^{24}$, which is divisible by k and is slightly larger than 16,609,143.

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.

5.1 One Permutation v.s. k-Permutation

Figure 10 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 [18], our one permutation scheme achieves similar (or even very slightly better) accuracies.

5.2 Preprocessing Time and Training Time

The preprocessing cost for processing the data using $k = 512$ independent permutations is about 6,000 seconds. In contrast, the processing cost for the proposed one permutation scheme is only $1/k$ of the original cost, i.e., about 10 seconds. Note that *webspam* is merely a small dataset compared to industrial applications. We expect the (absolute) improvement will be even more substantial in much larger datasets.

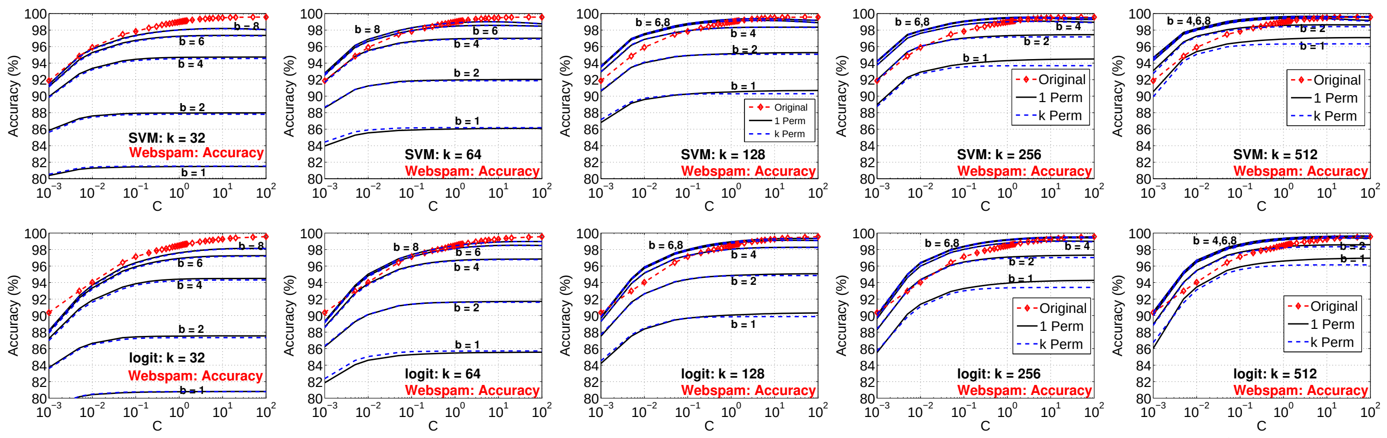


Figure 10: Test accuracies of SVM (upper panels) and logistic regression (bottom panels), averaged over 50 repetitions. The accuracies of using the original data are plotted as dashed (red, if color is available) curves with "diamond" markers. C is the regularization parameter. Compared with the original k-permutation minwise hashing scheme (dashed and blue if color is available), the proposed one permutation hashing scheme achieves very similar accuracies, or even slightly better accuracies when k is large.

The prior work [18] already presented the training time using the k-permutation hashing scheme. With one permutation hashing, the training time remains essentially the same (for the same k and b) on the *webspam* dataset. Note that, with the zero coding strategy, the new data matrix generated by one permutation hashing has potentially less nonzeros than the original minwise hashing scheme, due to the occurrences of empty bins. This phenomenon in theory may bring additional advantages such as slightly reducing the training time. Nevertheless, the most significant advantage of one permutation hashing lies in the dramatic reduction of the preprocessing cost, which is what we focus on in this study.

5.3 Zero Coding v.s. Random Coding for Empty Bins

The experimental results as shown in Figure 10 are based on the "zero coding" strategy for dealing with empty bins. Figure 11 plots the results for comparing zero coding with the random coding. When k is large, zero coding is superior to random coding, although the differences remain small in this dataset. This is not surprising, of course. Random coding adds artificial nonzeros to the new (expanded) data matrix, which would not be desirable for learning algorithms.

Remark: The empirical results on the *webspam* datasets are highly encouraging because they verify that our proposed one permutation hashing scheme works 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 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

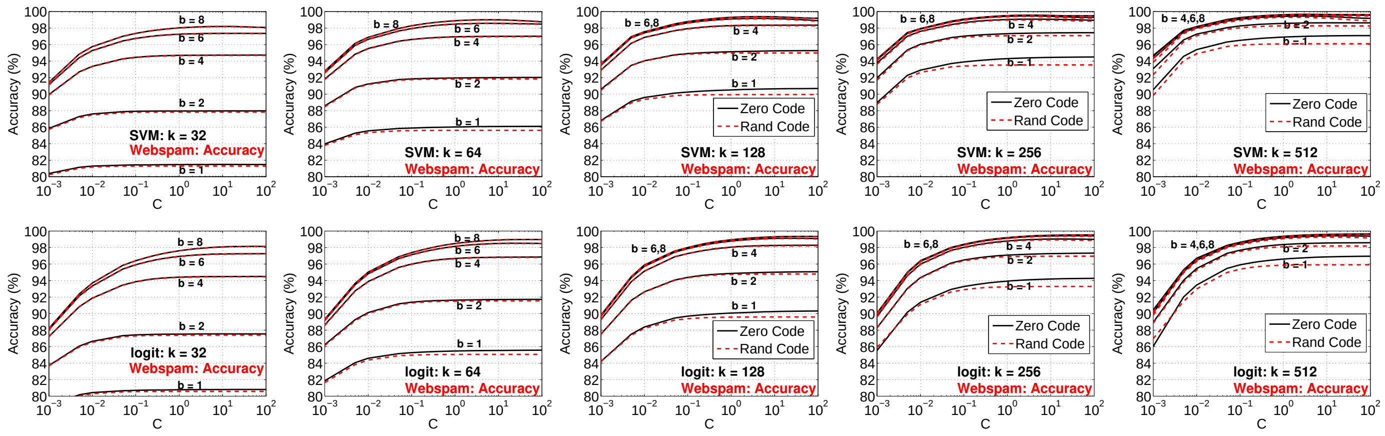


Figure 11: Test accuracies of SVM (upper panels) and logistic regression (bottom panels), averaged over 50 repetitions, for comparing the (recommended) zero coding strategy with the random coding strategy to deal with empty bins. We can see that the differences only become noticeable at $k = 512$.

original k-permutation scheme. We believe this is because the one permutation scheme is “sample-without-replacement” and provides a much better matrix sparsification strategy without “contaminating” the original data matrix too much.

6.1 One Permutation v.s. k-Permutation

We experiment with $k \in \{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}, 2^{12}\}$ 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 12 and Figure 13 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% even with $k = 4096$.

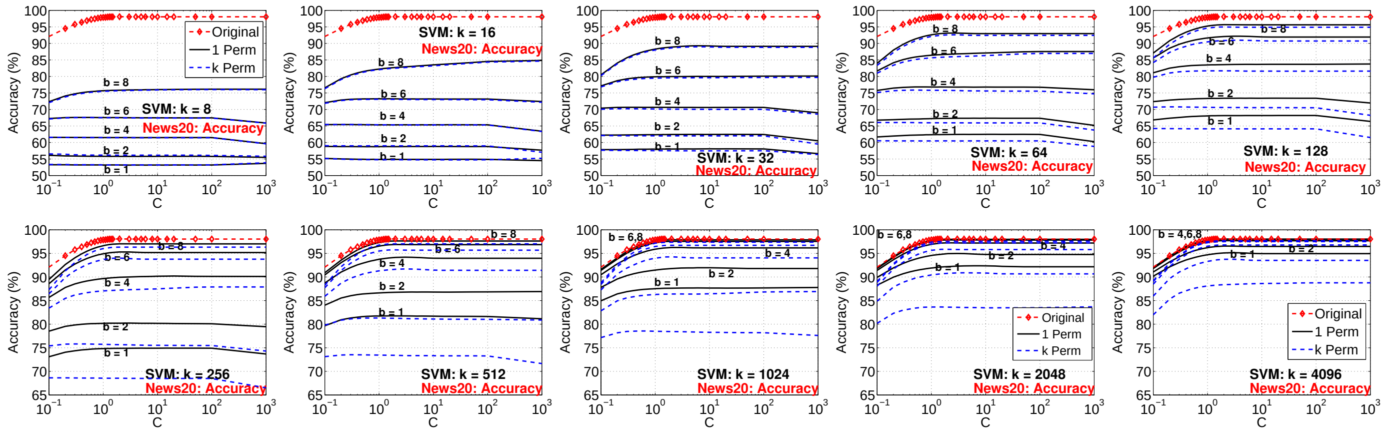


Figure 12: Test accuracies of linear SVM averaged over 100 repetitions. The proposed one permutation scheme noticeably outperforms the original k-permutation scheme especially when k is not small.

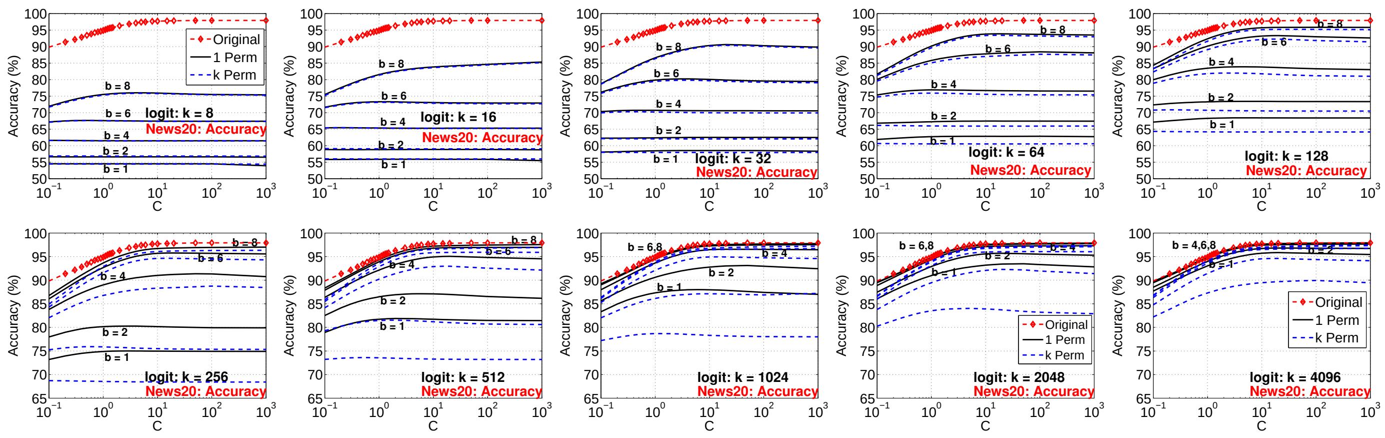


Figure 13: Test accuracies of logistic regression averaged over 100 repetitions. The proposed one permutation scheme noticeably outperforms the original k-permutation scheme especially when k is not small.

6.2 Zero Coding v.s. Random Coding for Empty Bins

Figure 14 and Figure 15 plot the results for comparing two coding strategies to deal with empty bins, respectively for linear SVM and logistic regression. Again, when k is small (e.g., $k \leq 64$), both strategies perform similarly. However, when k is large, using the random coding scheme may be disastrous, which is of course also expected. When $k = 4096$, most of the nonzero entries in the new expanded data matrix fed to the solver are artificial, since the original *news20* dataset has merely about 500 nonzero on average.

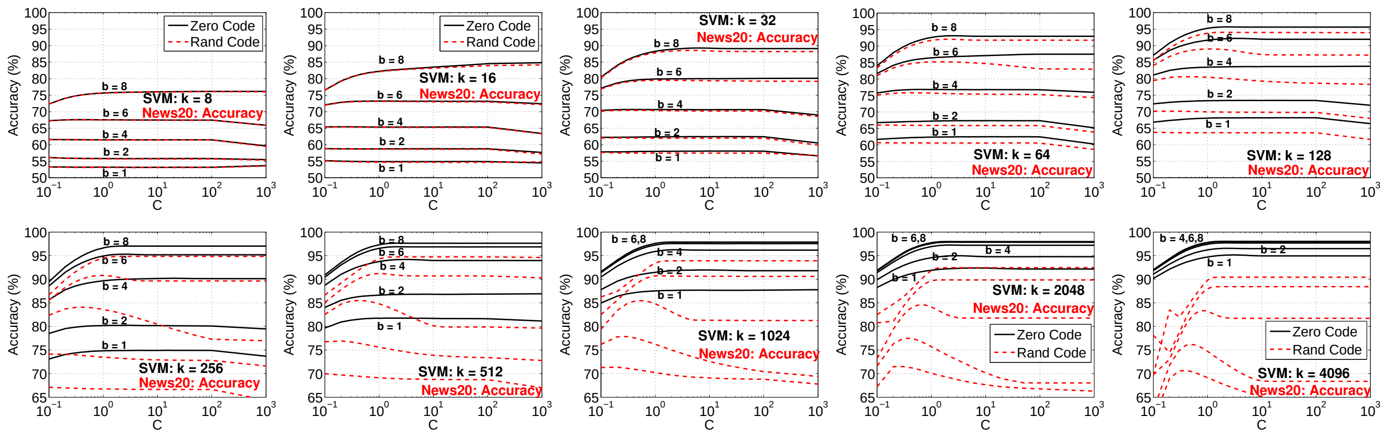


Figure 14: Test accuracies of linear SVM averaged over 100 repetitions, for comparing the (recommended) zero coding strategy with the random coding strategy to deal with empty bins. On this dataset, the performance of the random coding strategy can be bad.

Remark: We should re-iterate that the *news20* dataset is more like a contrived example, merely for testing the robustness of the one permutation scheme with the zero coding strategy. In more realistic industrial applications, we expect that numbers of nonzeros in many datasets should be significantly higher, and hence the performance differences between the one permutation scheme and the k-permutation scheme and the differences between the two strategies for empty bins should be small.

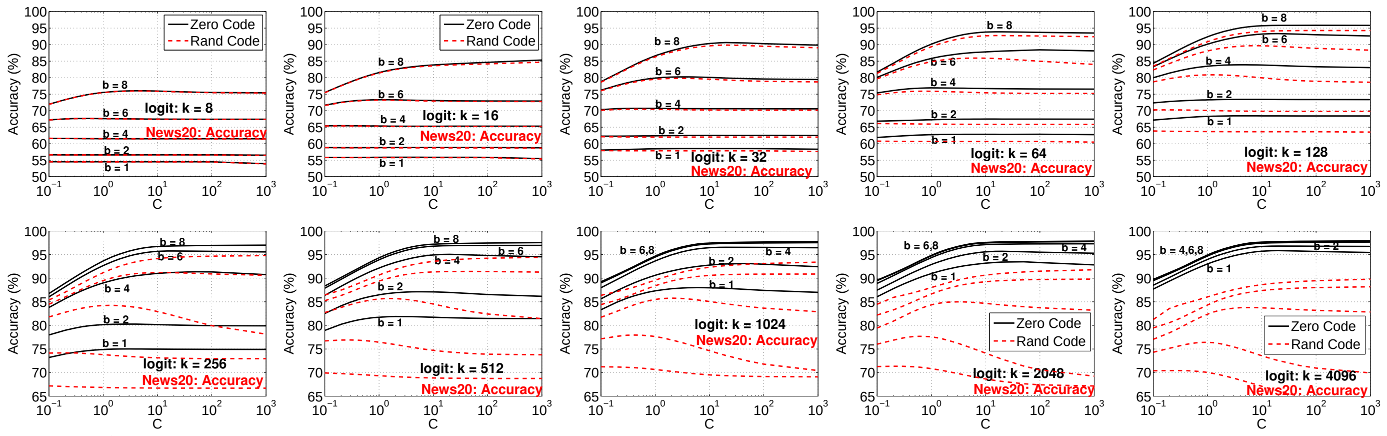


Figure 15: Test accuracies of logistic regression averaged over 100 repetitions, for comparing the zero coding strategy (recommended) with the random coding strategy to deal with empty bins. On this dataset, the performance of the random coding strategy can be bad.

7 The Variable Length One Permutation Hashing Scheme

While the fixed-length one permutation scheme we have presented and analyzed should be simple to implement and easy to understand, we would like to present a **variable-length** scheme which may more obviously connect with other known hashing methods such as the Count-Min (CM) sketch [6].

As in the fixed-length scheme, we first conduct a permutation $\pi : \Omega \rightarrow \Omega$. Instead of dividing the space evenly, we vary the bin lengths according to a multinomial distribution $mult\left(D, \frac{1}{k}, \frac{1}{k}, ..., \frac{1}{k}\right)$.

This variable-length scheme is equivalent to first uniformly grouping the original data entries into k bins and then applying permutations independently within each bin. The latter explanation connects our method with the Count-Min (CM) sketch [6] (but without the "count-min" step), which also hashes the elements uniformly to k bins and the final (stored) hashed value in each bin is the sum of all the elements in the bin. The bias of the CM estimate can be removed by subtracting a term. [20] adopted the CM sketch for linear learning. Later, [24] proposed a novel idea (named "VW") to remove the bias, by pre-multiplying (element-wise) the original data vectors with a random vector whose entries are sampled i.i.d. from the two-point distribution in $\{-1, 1\}$ with equal probabilities. In a recent paper, [18] showed that the variance of the CM sketch and variants are equivalent to the variance of random projections [16], which is substantially larger than the variance of the minwise hashing when the data are binary.

Since [18] has already conducted (theoretical and empirical) comparisons with CM and VW methods, we do not include more comparisons in this paper. Instead, we have simply showed that with one permutation only, we are able to achieve essentially the same accuracy as using k permutations.

We believe the fixed-length scheme is more convenient to implement. Nevertheless, we would like to present some theoretical results for the variable-length scheme, for better understanding the differences. The major difference is the distribution of N_{emp}, the number of jointly empty bins.

Lemma 8 *Under the variable-length scheme,*

$$\frac{E\left(N_{emp}\right)}{k} = \left(1 - \frac{1}{k}\right)^{f_1+f_2-a} \tag{29}$$

$$\frac{Var\left(N_{emp}\right)}{k^2} = \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1-\frac{E(N_{emp})}{k}\right) - \left(1-\frac{1}{k}\right)\left(\left(1-\frac{1}{k}\right)^{2(f_1+f_2-a)} - \left(1-\frac{2}{k}\right)^{f_1+f_2-a}\right) \tag{30}$$

$$< \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1-\frac{E(N_{emp})}{k}\right) \tag{31}$$

***Proof**: See Appendix H.* □

The other theoretical results for the fixed-length scheme which are expressed in terms N_{emp} essentially hold for the variable-length scheme. For example, $\frac{N_{mat}}{k-N_{emp}}$ is still an unbiased estimator of R and its variance is in the same form as (24) in terms of N_{emp}.

Remark: The number of empty bins for the variable-length scheme as presented in (29) is actually an upper bound of the number of empty bins for the fixed length scheme as shown in (10). The difference between $\prod_{j=0}^{f-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}$ and $\left(1-\frac{1}{k}\right)^f$ (recall $f = f_1 + f_2 - a$) is small when the data are sparse, as shown in Lemma 2, although it is possible that $\prod_{j=0}^{f-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j} \ll \left(1-\frac{1}{k}\right)^f$ in corner cases. Because smaller N_{emp} implies potentially better performance, we conclude that the fixed-length scheme should be sufficient and there are perhaps no practical needs to use the variable-length scheme.

8 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 algorithm (which is the 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 our proposed algorithm (or its variant) will be adopted in practice.

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] Izrail S. Gradshteyn and Iosif M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, New York, sixth edition, 2000.

[11] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, ICML, pages 408–415, 2008.

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

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

[14] 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).

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

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

[17] Ping Li and Arnd Christian König. Theory and applications b-bit minwise hashing. *Commun. ACM*, 2011.

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

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

[20] 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.

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

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

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

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

A Proof of Lemma 1

Recall $N_{emp} = \sum_{j=1}^{k} I_{emp,j}$, where $I_{emp,j} = 1$ if, in the j-th bin, both $\pi(S_1)$ and $\pi(S_2)$ are empty, and $I_{emp,j} = 0$ otherwise. Also recall $D = |\Omega|$, $f_1 = |S_1|$, $f_2 = |S_2|$, $a = |S_1 \cap S_2|$. Obviously, if $D\left(1-\frac{1}{k}\right) < f_1 + f_2 - a$, then none of the bins will be jointly empty, i.e., $E(N_{emp}) = Var(N_{emp}) = 0$. Next, assume $D\left(1-\frac{1}{k}\right) \geq f_1 + f_2 - a$, then by the linearity of expectation,

$$E\left(N_{emp}\right) = \sum_{j=1}^{k} \mathbf{Pr}\left(I_{emp,j} = 1\right) = k\mathbf{Pr}\left(I_{emp,1} = 1\right) = k\frac{\binom{D\left(1-\frac{1}{k}\right)}{f_1+f_2-a}}{\binom{D}{f_1+f_2-a}} = k\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}$$

To derive the variance, we first assume $D\left(1-\frac{2}{k}\right) \geq f_1 + f_2 - a$. Then

$$\begin{aligned}
Var\left(N_{emp}\right) =& E\left(N_{emp}^2\right) - E^2\left(N_{emp}\right)\\
=& E\left(\sum_{j=1}^{k} I_{emp,j}^2 + \sum_{i\neq j} I_{emp,i}I_{emp,j}\right) - E^2\left(N_{emp}\right)\\
=& k(k-1)\mathbf{Pr}\left(I_{emp,1} = 1, I_{emp,2} = 1\right) + E\left(N_{emp}\right) - E^2\left(N_{emp}\right)\\
=& k(k-1) \times \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right) - j}{D-j}\\
&+ k \times \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j} - \left(k \times \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}\right)^2
\end{aligned}$$

If $D\left(1-\frac{2}{k}\right) < f_1 + f_2 - a \leq D\left(1-\frac{1}{k}\right)$, then $\mathbf{Pr}\left(I_{emp,1} = 1, I_{emp,2} = 1\right) = 0$ and hence

$$Var\left(N_{emp}\right) = E\left(N_{emp}\right) - E^2\left(N_{emp}\right) = k \times \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j} - \left(k \times \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}\right)^2$$

Assuming $D\left(1-\frac{2}{k}\right) \geq f_1 + f_2 - a$, we obtain

$$\begin{aligned}
\frac{Var\left(N_{emp}\right)}{k^2} =& \frac{1}{k}\left(\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}\right)\left(1 - \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}\right)\\
&- \left(1-\frac{1}{k}\right)\left(\left(\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}\right)^2 - \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right) - j}{D-j}\right)\\
=& \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1 - \frac{E(N_{emp})}{k}\right)\\
&- \left(1-\frac{1}{k}\right)\left(\left(\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right) - j}{D-j}\right)^2 - \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right) - j}{D-j}\right)\\
<& \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1 - \frac{E(N_{emp})}{k}\right)
\end{aligned}$$

because

$$\left(\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2-\frac{D\left(1-\frac{2}{k}\right)-j}{D-j}>0$$
$$\Longleftrightarrow\left(D\left(1-\frac{1}{k}\right)-j\right)^2>(D-j)\left(D\left(1-\frac{2}{k}\right)-j\right)$$
$$\Longleftrightarrow\left(1-\frac{1}{k}\right)^2=1-\frac{2}{k}+\frac{1}{k^2}>1-\frac{2}{k}$$

This completes the proof.

B Proof of Lemma 2

The following expansions will be useful

$$\sum_{j=1}^{n-1}\frac{1}{j}=\log n+0.577216-\frac{1}{2n}-\frac{1}{12n^2}+\ldots \qquad \text{([10, 8.367.13])} \tag{32}$$

$$\log(1-x)=-x-\frac{x^2}{2}-\frac{x^3}{3}-\ldots \qquad (|x|<1) \tag{33}$$

Assume $D\left(1-\frac{1}{k}\right)\geq f_1+f_2-a$. We can write

$$\frac{E\left(N_{emp}\right)}{k}=\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}=\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\times\prod_{j=0}^{f_1+f_2-a-1}\left(1-\frac{j}{(k-1)(D-j)}\right)$$

Hence it suffices to study the error term

$$\prod_{j=0}^{f_1+f_2-a-1}\left(1-\frac{j}{(k-1)(D-j)}\right).$$

$$\log\prod_{j=0}^{f_1+f_2-a-1}\left(1-\frac{j}{(k-1)(D-j)}\right)=\sum_{j=0}^{f_1+f_2-a-1}\log\left(1-\frac{j}{(k-1)(D-j)}\right)$$
$$=\sum_{j=0}^{f_1+f_2-a-1}\left\{-\frac{j}{(k-1)(D-j)}-\frac{1}{2}\left(\frac{j}{(k-1)(D-j)}\right)^2-\frac{1}{3}\left(\frac{j}{(k-1)(D-j)}\right)^3+\ldots\right\}$$

Take the first term,

$$\sum_{j=0}^{f_1+f_2-a-1}-\frac{j}{(k-1)(D-j)}=\frac{1}{k-1}\sum_{j=0}^{f_1+f_2-a-1}\frac{D-j}{D-j}-\frac{D}{D-j}$$
$$=\frac{1}{k-1}\left(f_1+f_2-a-D\sum_{j=0}^{f_1+f_2-a-1}\frac{1}{D-j}\right)$$
$$=\frac{1}{k-1}\left(f_1+f_2-a-D\left(\sum_{j=1}^{D}\frac{1}{j}-\sum_{j=1}^{D-f_1-f_2+a}\frac{1}{j}\right)\right)$$
$$=\frac{1}{k-1}\left(f_1+f_2-a-D\left(\log(D+1)-\frac{1}{2(D+1)}-\log(D-f_1-f_2+a+1)+\frac{1}{2(D-f_1-f_2+a+1)}\right)+\ldots\right)$$

Thus, we obtain (by ignoring a term $\frac{D}{D+1}$)

$$\prod_{j=0}^{f_1+f_2-a-1}\left(1-\frac{j}{(k-1)(D-j)}\right)=\exp\left(\frac{-D\log\frac{D+1}{D-f_1-f_2+a+1}+(f_1+f_2-a)\left(1-\frac{1}{2(D-f_1-f_2+a+1)}\right)}{k-1}+...\right)$$

Assuming $f_1+f_2-a\ll D$, we can further expand $\log\frac{D+1}{D-f_1-f_2+a+1}$ and obtain a more simplified approximation:

$$\frac{E\left(N_{emp}\right)}{k}=\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\left(1-O\left(\frac{(f_1+f_2-a)^2}{kD}\right)\right)$$

Next, we analyze the approximation of the variance by assuming $f_1+f_2-a\ll D$. A similar analysis can show that

$$\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D-j}=\left(1-\frac{2}{k}\right)^{f_1+f_2-a}\left(1-O\left(\frac{(f_1+f_2-a)^2}{kD}\right)\right)$$

and hence we obtain, by using $1-\frac{2}{k}=\left(1-\frac{1}{k}\right)\left(1-\frac{1}{k-1}\right)$,

$$\begin{aligned}&\frac{Var\left(N_{emp}\right)}{k^2}\\=&\left(1-\frac{2}{k}\right)^{f_1+f_2-a}-\left(1-\frac{1}{k}\right)^{2(f_1+f_2-a)}+\frac{1}{k}\left(\left(1-\frac{1}{k}\right)^{f_1+f_2-a}-\left(1-\frac{2}{k}\right)^{f_1+f_2-a}\right)+O\left(\frac{(f_1+f_2-a)^2}{kD}\right)\\=&\frac{1}{k}\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\left(1-\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\right)\\&-\left(1-\frac{1}{k}\right)^{f_1+f_2-a+1}\left(\left(1-\frac{1}{k}\right)^{f_1+f_2-a}-\left(1-\frac{1}{k-1}\right)^{f_1+f_2-a}\right)+O\left(\frac{(f_1+f_2-a)^2}{kD}\right)\end{aligned}$$

C Proof of Lemma 3

Let $q(D,k,f)=\mathbf{Pr}\left(N_{emp}=0\right)$ and $D_{jk}=D(1-j/k)$. Then,

$$\begin{aligned}\mathbf{Pr}\left(N_{emp}=j\right)=&\binom{k}{j}P\{I_{emp,1}=\cdots=I_{emp,j}=1,I_{emp,j+1}=\cdots=I_{emp,k}=0\}\\=&\binom{k}{j}\frac{P_f^{D_{jk}}}{P_f^D}q(D_{jk},k-j,f).\end{aligned}$$

where P_f^D is the "permutation" operator: $P_f^D=D(D-1)(D-2)...(D-f+1)$.

Thus, to derive $\mathbf{Pr}\left(N_{emp}=j\right)$, we just need to find $q(D,k,f)$. By the union-intersection formula,

$$1-q(D,k,f)=\sum_{j=1}^{k}(-1)^{j-1}\binom{k}{j}E\prod_{i=1}^{j}I_{emp,i}.$$

From Lemma 1, we can infer $E\prod_{i=1}^{j} I_{emp,i} = P_f^{D_{jk}}/P_f^D = \prod_{t=0}^{f-1}\frac{D\left(1-\frac{j}{k}\right)-t}{D-t}$. Thus we find

$$q(D,k,f) = 1+\sum_{j=1}^{k}(-1)^j\binom{k}{j}\frac{P_f^{D_{jk}}}{P_f^D} = \sum_{j=0}^{k}(-1)^j\binom{k}{j}\frac{P_f^{D_{jk}}}{P_f^D}.$$

It follows that

$$\begin{aligned}\mathbf{Pr}\left(N_{emp}=j\right) =& \binom{k}{j}\sum_{s=0}^{k-j}(-1)^s\binom{k-j}{s}\frac{P_f^{D(1-j/k-s/k)}}{P_f^D}\\ =& \sum_{s=0}^{k-j}(-1)^s\frac{k!}{j!s!(k-j-s)!}\prod_{t=0}^{f-1}\frac{D\left(1-\frac{j+s}{k}\right)-t}{D-t}\end{aligned}$$

D Proof of Lemma 4

Define

$$\begin{aligned}&S_1\cup S_2 = \{j_1, j_2, ..., j_{f_1+f_2-a}\}\\ &J = \min \pi(S_1\cup S_2) = \min_{1\le i\le f_1+f_2-a}\pi(j_i)\\ &T = \underset{i}{argmin}\ \ \pi(j_i),\ \text{i.e.,}\ \pi(j_T)=J\end{aligned}$$

Because π is a random permutation, we know

$$\mathbf{Pr}\left(T=i\right) = \mathbf{Pr}\left(j_T=j_i\right) = \mathbf{Pr}\left(\pi(j_T)=\pi(j_i)\right) = \frac{1}{f_1+f_2-a},\ \ 1\le i\le f_1+f_2-a$$

Due to symmetry,

$$\mathbf{Pr}(T=i|J=t) = \mathbf{Pr}(\pi(j_i)=t|\min_{1\le l\le f_1+f_2-a}\pi(j_l)=t) = \frac{1}{f_1+f_2-a}$$

and hence we know that J and T are independent. Therefore,

$$\begin{aligned}E(N_{mat}) =& \sum_{j=1}^{k}\mathbf{Pr}(I_{mat,j}=1) = k\mathbf{Pr}(I_{mat,1}=1)\\ =& k\mathbf{Pr}\left(j_T\in S_1\cap S_2, 0\le J\le D/k-1\right)\\ =& k\mathbf{Pr}\left(j_T\in S_1\cap S_2\right)\mathbf{Pr}\left(0\le J\le D/k-1\right)\\ =& kR\mathbf{Pr}\left(I_{emp,1}=0\right)\\ =& kR\left(1-\frac{E\left(N_{emp}\right)}{k}\right)\end{aligned}$$

$$\begin{aligned}E(N_{mat}^2) =& E\left(\left(\sum_{j=1}^{k}I_{mat,j}\right)^2\right) = E\left(\sum_{j=1}^{k}I_{mat,j}+\sum_{i\ne j}^{k}I_{mat,i}I_{mat,j}\right)\\ =& E(N_{mat})+k(k-1)E(I_{mat,1}I_{mat,2})\end{aligned}$$

$$
\begin{aligned}
&E(I_{mat,1}I_{mat,2}) = \mathbf{Pr}\left(I_{mat,1}=1, I_{mat,2}=1\right)\\
=&\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(I_{mat,1}=1, I_{mat,2}=1|J=t\right)\mathbf{Pr}\left(J=t\right)\\
=&\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(j_T\in S_1\cap S_2, I_{mat,2}=1|J=t\right)\mathbf{Pr}\left(J=t\right)\\
=&\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(I_{mat,2}=1|J=t, j_T\in S_1\cap S_2\right)\mathbf{Pr}\left(j_T\in S_1\cap S_2\right)\mathbf{Pr}\left(J=t\right)\\
=&R\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(I_{mat,2}=1|J=t, j_T\in S_1\cap S_2\right)\mathbf{Pr}\left(J=t\right)
\end{aligned}
$$

Note that, conditioning on $\{J=t, j_T\in S_1\cap S_2\}$, the problem (i.e., the event $\{I_{mat,2}=1\}$) is actually the same as our original problem with f_1+f_2-a-1 elements whose locations are uniformly random on $\{t+1, t+2, ..., D-1\}$. Therefore,

$$
\begin{aligned}
&E(I_{mat,1}I_{mat,2})\\
=&R\sum_{t=0}^{D/k-1}\frac{a-1}{f_1+f_2-a-1}\left(1-\prod_{j=0}^{f_1+f_2-a-2}\frac{D\left(1-\frac{1}{k}\right)-t-1-j}{D-t-1-j}\right)\mathbf{Pr}\left(J=t\right)\\
=&R\frac{a-1}{f_1+f_2-a-1}\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(J=t\right)\left(1-\prod_{j=0}^{f_1+f_2-a-2}\frac{D\left(1-\frac{1}{k}\right)-t-1-j}{D-t-1-j}\right)\\
=&R\frac{a-1}{f_1+f_2-a-1}\left(\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(J=t\right)-\sum_{t=0}^{D/k-1}\mathbf{Pr}\left(J=t\right)\prod_{j=0}^{f_1+f_2-a-2}\frac{D\left(1-\frac{1}{k}\right)-t-1-j}{D-t-1-j}\right)
\end{aligned}
$$

By observing that

$$
\mathbf{Pr}(J=t)=\frac{\binom{D-t-1}{f_1+f_2-a-1}}{\binom{D}{f_1+f_2-a}}=\frac{f_1+f_2-a}{D}\prod_{j=0}^{t-1}\frac{D-f_1-f_2+a-j}{D-1-j}=\frac{f_1+f_2-a}{D}\prod_{j=1}^{f_1+f_2-a-1}\frac{D-t-j}{D-j}
$$

$$
\sum_{t=0}^{D/k-1}\mathbf{Pr}(J=t)=1-\mathbf{Pr}\left(I_{emp,1}=1\right)=1-\frac{E\left(N_{emp}\right)}{k}=1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}
$$

we obtain two interesting (combinatorial) identities

$$
\frac{f_1+f_2-a}{D}\sum_{t=0}^{D/k-1}\prod_{j=0}^{t-1}\frac{D-f_1-f_2+a-j}{D-1-j}=1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}
$$

$$
\frac{f_1+f_2-a}{D}\sum_{t=0}^{D/k-1}\prod_{j=1}^{f_1+f_2-a-1}\frac{D-t-j}{D-j}=1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}
$$

which helps us simplify the expression:

$$\begin{aligned}
&\sum_{t=0}^{D/k-1} \mathbf{Pr}\left(J=t\right) \prod_{j=0}^{f_1+f_2-a-2} \frac{D\left(1-\frac{1}{k}\right)-t-1-j}{D-t-1-j} \\
=&\sum_{t=0}^{D/k-1} \frac{f_1+f_2-a}{D} \prod_{j=1}^{f_1+f_2-a-1} \frac{D-t-j}{D-j} \prod_{j=0}^{f_1+f_2-a-2} \frac{D\left(1-\frac{1}{k}\right)-t-1-j}{D-t-1-j} \\
=&\sum_{t=0}^{D/k-1} \frac{f_1+f_2-a}{D} \prod_{j=1}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-t-j}{D-j} \\
=&\sum_{t=0}^{2D/k-1} \frac{f_1+f_2-a}{D} \prod_{j=1}^{f_1+f_2-a-1} \frac{D-t-j}{D-j} - \sum_{t=0}^{D/k-1} \frac{f_1+f_2-a}{D} \prod_{j=1}^{f_1+f_2-a-1} \frac{D-t-j}{D-j} \\
=&\left(1-\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right)-j}{D-j}\right) - \left(1-\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right) \\
=&-\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right)-j}{D-j} + \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}
\end{aligned}$$

Combining the results, we obtain

$$\begin{aligned}
&E(I_{mat,1}I_{mat,2}) \\
=&R\frac{a-1}{f_1+f_2-a-1}\left(1-\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j} + \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right)-j}{D-j} - \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right) \\
=&R\frac{a-1}{f_1+f_2-a-1}\left(1-2\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j} + \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right)-j}{D-j}\right)
\end{aligned}$$

And hence

$$\begin{aligned}
&Var(N_{mat}) = k(k-1)E(I_{mat,1}I_{mat,2}) + E(N_{mat}) - E^2(N_{mat}) \\
=&k(k-1)R\frac{a-1}{f_1+f_2-a-1}\left(1-2\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j} + \prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{2}{k}\right)-j}{D-j}\right) \\
&+kR\left(1-\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right) - k^2R^2\left(1-\prod_{j=0}^{f_1+f_2-a-1} \frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2
\end{aligned}$$

$$
\begin{aligned}
&\frac{Var(N_{mat})}{k^2}\\
=&\frac{1}{k}\left(\frac{E(N_{mat})}{k}\right)\left(1-\frac{E(N_{mat})}{k}\right)\\
&+\left(1-\frac{1}{k}\right)R\frac{a-1}{f_1+f_2-a-1}\left(1-2\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}+\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D-j}\right)\\
&-\left(1-\frac{1}{k}\right)R^2\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2\\
<&\frac{1}{k}\left(\frac{E(N_{mat})}{k}\right)\left(1-\frac{E(N_{mat})}{k}\right)\\
&+\left(1-\frac{1}{k}\right)R^2\left(1-2\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}+\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2\right)\\
&-\left(1-\frac{1}{k}\right)R^2\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2\\
=&\frac{1}{k}\left(\frac{E(N_{mat})}{k}\right)\left(1-\frac{E(N_{mat})}{k}\right)
\end{aligned}
$$

To see the inequality, note that $\frac{a-1}{f_1+f_2-a-1} < R = \frac{a}{f_1+f_2-a}$, and $\frac{D\left(1-\frac{2}{k}\right)-j}{D-j} < \left(\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)^2$ as proved towards the end of Appendix A. This completes the proof.

E Proof of Lemma 5

$$
\begin{aligned}
E\left(N_{mat}N_{emp}\right)=&E\left(\sum_{j=1}^{k}I_{mat,j}\sum_{j=1}^{k}I_{emp,j}\right)=\sum_{j=1}^{k}E\left(I_{mat,j}I_{emp,j}\right)+\sum_{i\neq j}E\left(I_{mat,i}I_{emp,j}\right)\\
=&0+\sum_{i\neq j}E\left(I_{mat,i}I_{emp,j}\right)=k(k-1)E\left(I_{emp,1}I_{mat,2}\right)
\end{aligned}
$$

$$
\begin{aligned}
E\left(I_{emp,1}I_{mat,2}\right)=&\mathbf{Pr}\left(I_{emp,1}=1,I_{mat,2}=1\right)=\mathbf{Pr}\left(I_{mat,2}=1|I_{emp,1}=1\right)\mathbf{Pr}\left(I_{emp,1}=1\right)\\
=&R\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}
\end{aligned}
$$

$$\begin{aligned}
&Cov\left(N_{mat},\ N_{emp}\right) = E\left(N_{mat}N_{emp}\right) - E\left(N_{mat}\right)E\left(N_{emp}\right)\\
=&k(k-1)R\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\\
&-kR\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)k\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\\
=&k^2R\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\\
&-kR\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\le 0
\end{aligned}$$

To see the inequality, it suffices to show that $g(k) < 0$, where

$$\begin{aligned}
g(k) =&k\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)-\left(1-\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\\
=&k\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)-1-(k-1)\left(\prod_{j=0}^{f_1+f_2-a-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)
\end{aligned}$$

Because $g(k=\infty)=0$, it suffices to show that $g(k)$ is increasing in k.

$$g(f;k) =k\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)-1-(k-1)\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)$$

$$\begin{aligned}
g(f+1;k) =&k\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\left(\frac{D\left(1-\frac{1}{k}\right)-f}{D-f}\right)-1-(k-1)\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\left(\frac{D\left(1-\frac{2}{k}\right)-f}{D\left(1-\frac{1}{k}\right)-f}\right)\\
=&g(f;k)-\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\left(\frac{D}{D-f}\right)+\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\left(\frac{D\left(1-\frac{1}{k}\right)}{D\left(1-\frac{1}{k}\right)-f}\right)
\end{aligned}$$

Thus, it suffices to show

$$\begin{aligned}
&-\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{1}{k}\right)-j}{D-j}\right)\left(\frac{D}{D-f}\right)+\left(\prod_{j=0}^{f-1}\frac{D\left(1-\frac{2}{k}\right)-j}{D\left(1-\frac{1}{k}\right)-j}\right)\left(\frac{D\left(1-\frac{1}{k}\right)}{D\left(1-\frac{1}{k}\right)-f}\right)\le 0\\
\Longleftrightarrow&h(f;k)=\left(\prod_{j=0}^{f-1}\frac{\left(D\left(1-\frac{2}{k}\right)-j\right)\left(D-j\right)}{\left(D\left(1-\frac{1}{k}\right)-j\right)^2}\right)\left(\frac{\left(1-\frac{1}{k}\right)\left(D-f\right)}{D\left(1-\frac{1}{k}\right)-f}\right)\le 1
\end{aligned}$$

$h(f;k)\le 1$ holds because one can check that $h(1;k)\le 1$ and $\frac{\left(D\left(1-\frac{2}{k}\right)-j\right)\left(D-j\right)}{\left(D\left(1-\frac{1}{k}\right)-j\right)^2}<1$.

This completes the proof.

F Proof of Lemma 6

We first prove that $\hat{R}_{mat} = \frac{N_{mat}}{k-N_{emp}}$ is unbiased,

$$\begin{aligned}
&I_{emp,j} = 1 \Rightarrow I_{mat,j} = 0\\
&E\Big(I_{mat,j}\Big|I_{emp,j} = 0\Big) = R\\
&E\Big(I_{mat,j}\Big|k - N_{emp} = m\Big) = (m/k)R,\ m > 0\\
&P\{k - N_{emp} > 0\} = 1\\
&E\Big(N_{mat}\Big|k - N_{emp}\Big) = R(k - N_{emp})\\
&E\Big(N_{mat}/(k - N_{emp})\Big|k - N_{emp}\Big) = R \qquad \text{independent of } N_{emp}\\
&E\left(\hat{R}_{mat}\right) = R
\end{aligned}$$

Next, we compute the variance. To simplify the notation, denote $f = f_1 + f_2 - a$ and $\tilde{R} = \frac{a-1}{f-1}$. Note that

$$\begin{aligned}
&E\Big(I_{mat,1}I_{mat,2}\Big|I_{emp,1} = I_{emp,2} = 0\Big) = R(a-1)/(f-1) = R\tilde{R}\\
&R^2 - R\tilde{R} = R\{a(f-1) - f(a-1)\}/\{f(f-1)\} = R(1-R)/(f-1)\\
&E\Big(I_{mat,1}I_{mat,2}\Big|I_{emp,1} + I_{emp,2} > 0\Big) = 0
\end{aligned}$$

By conditioning on $k - N_{emp}$, we obtain

$$\begin{aligned}
&E\Big(N_{mat}^2\Big|k - N_{emp} = m\Big)\\
&= kE\Big(I_{mat,1}\Big|k - N_{emp} = m\Big) + k(k-1)E\Big(I_{mat,1}I_{mat,2}\Big|k - N_{emp} = m\Big)\\
&= Rm + k(k-1)R\tilde{R}\mathbf{Pr}\Big(I_{emp,1} = I_{emp,2} = 0\Big|k - N_{emp} = m\Big)\\
&= Rm + k(k-1)R\tilde{R}\binom{m}{2}\Big/\binom{k}{2}\\
&= Rm + m(m-1)R\tilde{R}
\end{aligned}$$

and

$$\begin{aligned}
&E\Big(\hat{R}_{mat}^2\Big|k - N_{emp} = m\Big) = R\tilde{R} + (R - R\tilde{R})/m\\
&E\hat{R}_{mat}^2 = R\tilde{R} + (R - R\tilde{R})E(k - N_{emp})^{-1}
\end{aligned}$$

Combining the above results, we obtain

$$\begin{aligned}
Var\Big(\hat{R}_{mat}\Big) =& R\tilde{R} - R^2 + (R - R\tilde{R})E(k - N_{emp})^{-1}\\
=& R(1-R)E(k - N_{emp})^{-1} - (R^2 - R\tilde{R})(1 - E(k - N_{emp})^{-1})\\
=& R(1-R)E(k - N_{emp})^{-1} - R(1-R)(f-1)^{-1}(1 - E(k - N_{emp})^{-1})\\
=& R(1-R)\left\{E(k - N_{emp})^{-1} - (f-1)^{-1} + (f-1)^{-1}E(k - N_{emp})^{-1})\right\}
\end{aligned}$$

G Proof of Lemma 7

$$g(f;k) = \frac{1}{1-\left(1-\frac{1}{k}\right)^f}\left(1+\frac{1}{f-1}\right) - \frac{k}{f-1}$$

To show $g(f;,k) \leq 1$, it suffices to show

$$h(f;k) = (f+k-1)\left(1-\left(1-\frac{1}{k}\right)^f\right) - f \geq 0 \qquad \text{(note that } h(1;k)=0,\ h(2;k)>0\text{)}$$

for which it suffices to show

$$\frac{\partial h(f;k)}{\partial f} = \left(1-\left(1-\frac{1}{k}\right)^f\right) + (f+k-1)\left(-\left(1-\frac{1}{k}\right)^f \log\left(1-\frac{1}{k}\right)\right) - 1 \geq 0$$

and hence it suffices to show $-1-(f+k-1)\log\left(1-\frac{1}{k}\right) \geq 0$, which is true because $\log\left(1-\frac{1}{k}\right) < -\frac{1}{k}$. This completes the proof.

H Proof of Lemma 8

Recall we first divide the D elements into k bins whose lengths are multinomial distributed with equal probability $\frac{1}{k}$. We denote their lengths by L_j, $j=1$ to k. In other words,

$$(L_1, L_2, ..., L_k) \sim multinomial\left(D, \frac{1}{k}, \frac{1}{k}, ..., \frac{1}{k}\right)$$

and we know

$$E(L_j) = \frac{D}{k}, \quad Var(L_j) = D\frac{1}{k}\left(1-\frac{1}{k}\right), \quad Cov(L_i,\ L_j) = -\frac{D}{k^2}$$

Define

$$I_{i,j} = \begin{cases} 1 & \text{if the } i\text{-th element is hashed to the } j\text{-th bin} \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

We know

$$E(I_{i,j}) = \frac{1}{k}, \quad E(I_{i,j}^2) = \frac{1}{k}, \quad E(I_{i,j}I_{i,j'}) = 0, \quad E(I_{i,j}I_{i',j}) = \frac{1}{k^2},$$
$$E(1-I_{i,j}) = 1-\frac{1}{k}, \quad E(1-I_{i,j})^2 = 1-\frac{1}{k}, \quad E(1-I_{i,j})(1-I_{i,j'}) = 1-\frac{2}{k}$$

Thus

$$N_{emp} = \sum_{j=1}^{k}\prod_{i\in S_1\cup S_2}(1-I_{i,j})$$

$$E\left(N_{emp}\right)=\sum_{j=1}^{k}\prod_{i\in S_1\cup S_2}E\left(\left(1-I_{i,j}\right)\right)=k\left(1-\frac{1}{k}\right)^{f_1+f_2-a}$$

$$\begin{aligned}E\left(N_{emp}^2\right)=&\sum_{j=1}^{k}\prod_{i\in S_1\cup S_2}\left(1-I_{i,j}\right)^2+\sum_{j\neq j'}\prod_{i\in S_1\cup S_2}\left(1-I_{i,j}\right)\left(1-I_{i,j'}\right)\\=&k\left(1-\frac{1}{k}\right)^{f_1+f_2-a}+k(k-1)\left(1-\frac{2}{k}\right)^{f_1+f_2-a}\end{aligned}$$

$$Var\left(N_{emp}\right)=k\left(1-\frac{1}{k}\right)^{f_1+f_2-a}+k(k-1)\left(1-\frac{2}{k}\right)^{f_1+f_2-a}-k^2\left(1-\frac{1}{k}\right)^{2(f_1+f_2-a)}$$

Therefore,

$$\begin{aligned}\frac{Var\left(N_{emp}\right)}{k^2}=&\frac{1}{k}\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\left(1-\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\right)\\&-\left(1-\frac{1}{k}\right)\left(\left(1-\frac{1}{k}\right)^{2(f_1+f_2-a)}-\left(1-\frac{2}{k}\right)^{f_1+f_2-a}\right)\\<&\frac{1}{k}\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\left(1-\left(1-\frac{1}{k}\right)^{f_1+f_2-a}\right)\end{aligned}$$

This completes the proof of Lemma 8.