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 $\pi : \Omega \rightarrow \Omega$ 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 $z_1 = \min\left(\pi\left(S_1\right)\right)$ and $z_2 = \min\left(\pi\left(S_2\right)\right)$, 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.*

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

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

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

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

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.* □

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 $C \geq 1$.

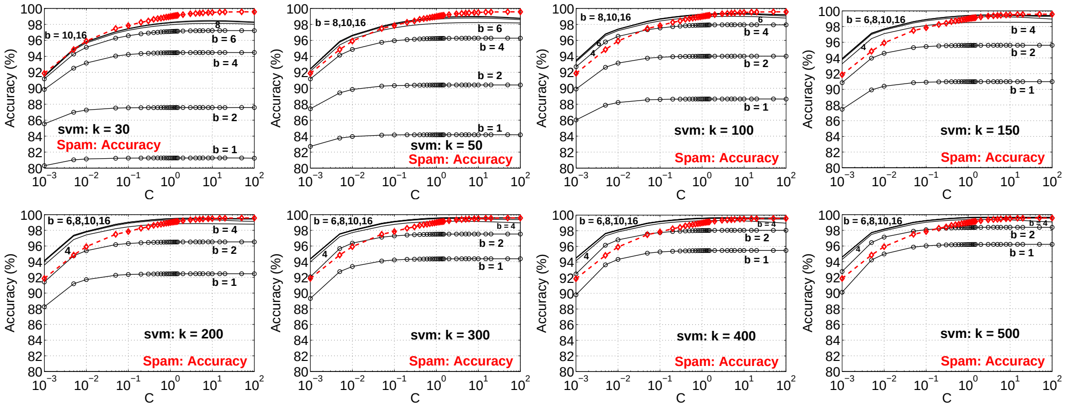

Figure 1: **SVM test accuracy** (averaged over 50 repetitions). With $k \geq 200$ and $b \geq 8$. b-bit hashing achieves very similar accuracies as using the original data (dashed, red if color is available).

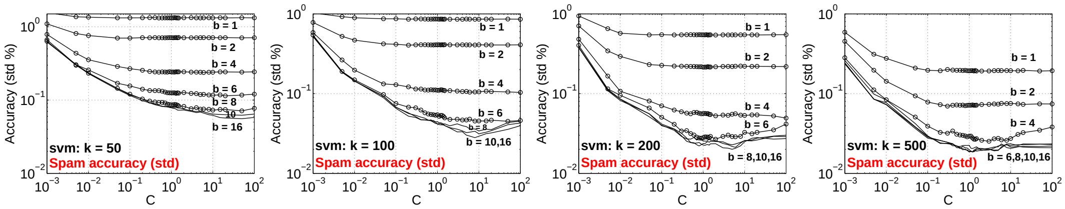

Figure 2: **SVM test accuracy (std)**. The standard deviations are computed from 50 repetitions. When $b \geq 8$, the standard deviations become extremely small (e.g., 0.02%).

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.

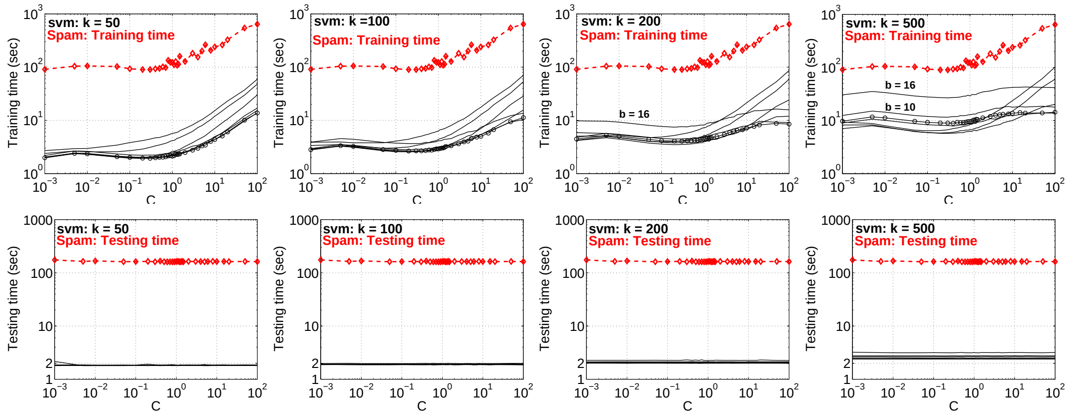

Figure 3: **SVM training time (upper panels) and testing time (bottom panels)**. The original costs are plotted using dashed (red, if color is available) curves.

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.

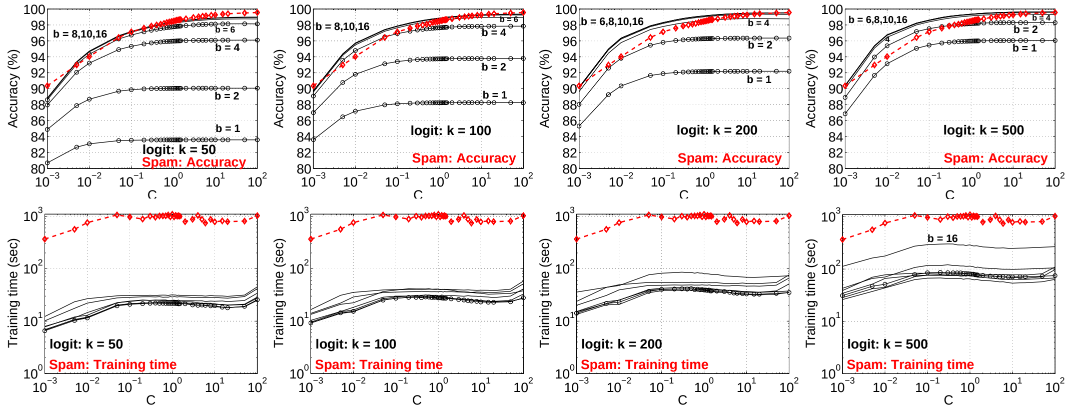

Figure 4: **Logistic regression test accuracy (upper panels) and training time (bottom panels)**.

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

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

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

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).

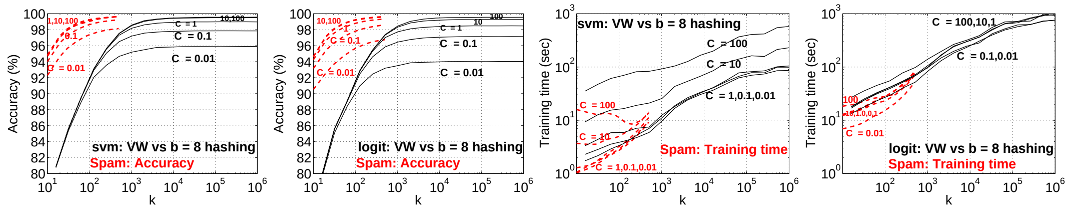

Figure 5: The dashed (red if color is available) curves represent b-bit minwise hashing results (only for $k \leq 500$) while solid curves for VW. We display results for $C = 0.01, 0.1, 1, 10, 100$.

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.

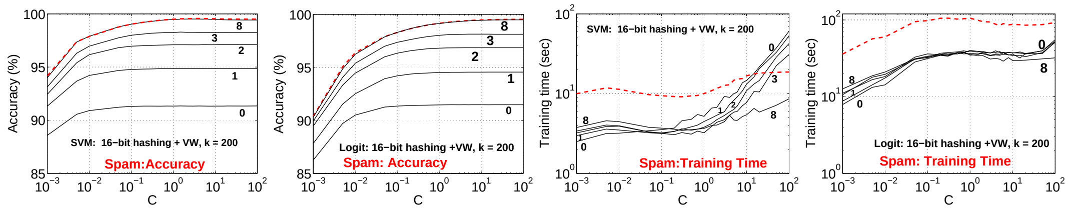

Figure 6: We apply VW hashing on top of the binary vectors (of length $2^b k$) generated by b-bit hashing, with size $m = 2^0 k, 2^1 k, 2^2 k, 2^3 k, 2^8 k$, for $k = 200$ and $b = 16$. The numbers on the solid curves (0, 1, 2, 3, 8) are the exponents. The dashed (red if color if available) curves are 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

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

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.* □

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 to 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.