Hashing Algorithms for Large-Scale Learning

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

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

Joshua Moore
Department of Computer Science
Cornell University
Ithaca, NY 14853
jlmo@cs.cornell.edu

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

Abstract

In this paper, we first demonstrate that b-bit minwise hashing, whose estimators are positive definite kernels, can be naturally integrated with learning algorithms such as SVM and logistic regression. We adopt a simple scheme to transform the nonlinear (resemblance) kernel into linear (inner product) kernel; and hence large-scale problems can be solved extremely efficiently. Our method provides a simple effective solution to large-scale learning in massive and extremely high-dimensional datasets, especially when data do not fit in memory.

We then compare b-bit minwise hashing with the Vowpal Wabbit (VW) algorithm (which is related the Count-Min (CM) sketch). Interestingly, VW has the same variances as random projections. Our theoretical and empirical comparisons illustrate that usually b-bit minwise hashing is significantly more accurate (at the same storage) than VW (and random projections) in binary data. Furthermore, b-bit minwise hashing can be combined with VW to achieve further improvements in terms of training speed, especially when b is large.

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, [34] experimented with a dataset with potentially 16 trillion (1.6×10^{13}) unique features. [32] discusses training sets with (on average) 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 increase in dataset sizes has made the storage and computational requirements for computing exact distances prohibitive, thus making data representations that allow compact storage and efficient approximate distance computation necessary.

The method of b-bit minwise hashing [26, 27, 25] is a very 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) [22, 30, 14, 19, 35] and logistic regression solvers. In [35], the authors addressed a critically important problem of training linear SVM

when the data can not fit in memory. In this paper, our work also addresses the same problem using a very different approach.

1.1 Ultra High-Dimensional Large Datasets and Memory Bottleneck

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 can be as large as 5 (or 7) in several studies [6, 7, 15]. 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 unlikely to occur more than once in a document. Interestingly, even when the data are not too high-dimensional, empirical studies [9, 18, 20] achieved good performance with binary-quantized data.

When the data can fit in memory, linear SVM 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 training the SVM [35]. A much more severe problem arises when the data can not fit in memory. This situation can be very common in practice. The publicly available *webspam* dataset needs about 24GB disk space (in LIBSVM input data format), which exceeds the memory capacity of many desktop PCs. Note that *webspam*, which contains only 350,000 documents represented by 3-shingles, is still a small dataset compared to the industry applications [32].

1.2 A Brief Introduction of Our Proposal

We propose a solution which leverages *b-bit minwise hashing*. Our approach assume the data vectors are binary, very 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 indeed 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 [12], or Vowpal Wabbit (VW) [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.

[35] proposed solving the memory bottleneck by partitioning the data into blocks, which are repeatedly loaded into memory as their approach updates the model coefficients. However, the computational bottleneck is still at the memory because loading the data blocks for many iterations consumes a large number of disk I/Os. Clearly, one should note that our method is not really a competitor of the approach in [35]. In fact, both approaches may work together to solve extremely large problems.

2 Review Minwise Hashing and b-Bit Minwise Hashing

Minwise hashing [6, 7] has been successfully applied to a very wide range of real-world problems especially in the context of search [6, 7, 4, 16, 10, 8, 33, 21, 13, 11, 17, 23, 29], 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 (normalized) measure of similarity is the *resemblance R*:

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

In this method, one applies a random permutation $\pi : \Omega \to \Omega$ on S_1 and S_2. The collision probability is simply

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

One can repeat the permutation k times: $\pi_1, \pi_2, ..., \pi_k$ to estimate R without bias, as

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

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

The common practice of minwise hashing is to store each hashed value, e.g., $\min(\pi(S_1))$ and $\min(\pi(S_2))$, using 64 bits [15]. The storage (and computational) cost will be prohibitive in truly large-scale (industry) applications [28].

b-bit minwise hashing [26, 27, 25] 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, we define the minimum values under π: $z_1 = \min\left(\pi\left(S_1\right)\right)$ and $z_2 = \min\left(\pi\left(S_2\right)\right)$, and define $e_{1,i} = i$th lowest bit of z_1, and $e_{2,i} = i$th lowest bit of z_2.

Theorem 1 *[26] Assume D is large.*

$$P_b = \mathbf{Pr}\left(\prod_{i=1}^{b} 1\left\{e_{1,i} = e_{2,i}\right\}\right) = C_{1,b} + (1 - C_{2,b})\, R \tag{4}$$

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

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

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

This (approximate) formula (4) is remarkably accurate, even for very small D. Some numerical comparisons with the exact probabilities are provided in Appendix A.

We can then estimate P_b (and R) from k independent permutations: $\pi_1, \pi_2, ..., \pi_k$,

$$\hat{R}_b = \frac{\hat{P}_b - C_{1,b}}{1 - C_{2,b}}, \qquad\qquad \hat{P}_b = \frac{1}{k}\sum_{j=1}^{k}\left\{\prod_{i=1}^{b} 1\{e_{1,i,\pi_j} = e_{2,i,\pi_j}\}\right\}, \tag{5}$$

$$\mathrm{Var}\left(\hat{R}_b\right) = \frac{\mathrm{Var}\left(\hat{P}_b\right)}{\left[1 - C_{2,b}\right]^2} = \frac{1}{k}\frac{[C_{1,b} + (1 - C_{2,b})R]\,[1 - C_{1,b} - (1 - C_{2,b})R]}{\left[1 - C_{2,b}\right]^2} \tag{6}$$

We will show that we can apply b-bit hashing for learning without explicitly estimating R from (4).

3 Kernels from Minwise Hashing and b-Bit Minwise Hashing

This section proves some theoretical properties of matrices generated by resemblance, minwise hashing, or b-bit minwise hashing, which are all positive definite matrices. Our proof not only provides a solid theoretical foundation for using b-bit hashing in learning, but also illustrates our idea behind the construction for integrating b-bit hashing with linear learning algorithms.

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_2, ..., $S_n \in \Omega = \{0, 1, ..., D-1\}$. Apply one permutation π to each set and define $z_i = \min\{\pi(S_i)\}$. The following three matrices are all 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 $\mathbf{M}^{(b)} \in \mathbb{R}^{n \times n}$: $M_{ij}^{(b)} = \prod_{t=1}^{b} 1\{e_{i,t} = e_{j,t}\}$, where $e_{i,t}$ is the t-th lowest bit of z_i.*

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

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

$$M_{ij} = 1\{z_i = z_j\} = \sum_{t=0}^{D-1} 1\{z_i = t\} \times 1\{z_j = t\}, \tag{7}$$

M_{ij} is the inner product of two D-dim vectors. Thus, $\mathbf{M}$ is PD.
Similarly, the b-bit minwise hashing matrix $\mathbf{M}^{(b)}$ is PD because

$$M_{ij}^{(b)} = \sum_{t=0}^{2^b-1} 1\{z_i = t\} \times 1\{z_j = t\}. \tag{8}$$

The resemblance matrix $\mathbf{R}$ is PD because $R_{ij} = \mathbf{Pr}\{M_{ij} = 1\} = E(M_{ij})$ and M_{ij} is the (i,j)-th element of the PD matrix $\mathbf{M}$. Note that the expectation is a linear operation. □

Our proof that the b-bit minwise hashing matrix $\mathbf{M}^{(b)}$ is PD provides us with a simple strategy to expand a nonlinear (resemblance) kernel into a linear (inner product) kernel. After concatenating the k vectors resulting from (8), the new (binary) data vector after the expansion will be of dimension $2^b \times k$ with exactly k ones.

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$^{\text{perf}}$ [22], Pegasos [30], Bottou's SGD SVM [5], and LIBLINEAR [14].

Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, 1\}$, the L_2-regularized linear SVM solves the following optimization problem:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w} + C\sum_{i=1}^{n} \max\{1 - y_i\mathbf{w}^{\mathbf{T}}\mathbf{x_i}, \ 0\}, \tag{9}$$

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

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w} + C\sum_{i=1}^{n} \log\left(1 + e^{-y_i\mathbf{w}^{\mathbf{T}}\mathbf{x_i}}\right). \tag{10}$$

Here $C > 0$ is an important penalty 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 independent 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.

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:

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

Clearly, this expansion is directly inspired by the proof that the b-bit minwise hashing matrix is PD in Theorem 2. Note that the total storage cost is still just nbk bits and each new data vector (of length $2^b \times k$) has exactly k 1's.

Also, note that in this procedure we actually do not explicitly estimate the resemblance R using (4).

5 Experimental Results on Webspam Dataset

Our experimental settings follow the work in [35] very closely. The authors of [35] conducted experiments on three datasets, of which 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 tool 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 for which 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 the same testing accuracy 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 the *webspam* dataset. We waited for **over one week**[1] but LIBSVM still had not output any results. Fortunately, using b-bit minwise hashing to estimate the resemblance kernels, we were able to obtain some results. For example, with $C = 1$ and $b = 8$, the training time of LIBSVM ranged from 1938 seconds ($k = 30$) to 13253 seconds ($k = 500$). In particular, when $k \geq 200$, the test accuracies essentially matched the best test results given by LIBLINEAR on the original *webspam* data.

Therefore, there is a significant benefit of *data reduction* provided by b-bit minwise hashing, for training nonlinear SVM. This experiment also demonstrates that it is very important (and fortunate) that we are able to transform this nonlinear problem into a linear problem.

5.2 Experimental Results on Linear SVM

Since there is an important tuning parameter C in linear SVM and logistic regression, we conducted our extensive experiments for a wide range of C values (from 10^{-3} to 10^2) with fine spacings in $[0.1, \ 10]$.

We mainly experimented with $k = 30$ to $k = 500$, and $b = 1, 2, 4, 8,$ and 16. Figures 1 (average) and 2 (std, standard deviation) provide the test accuracies. Figure 1 demonstrates that using $b \geq 8$ and $k \geq 150$ achieves about the same 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 when $C \geq 1$.

---

[1] We will let the program run unless it is accidentally terminated (e.g., due to power outage).
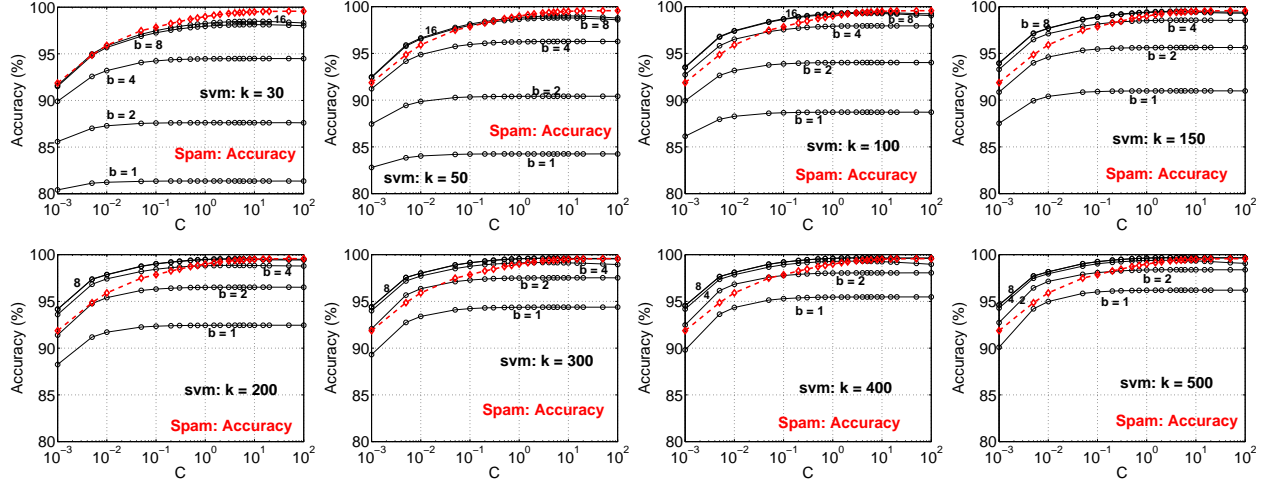
Figure 1: **Linear SVM test accuracy** (averaged over 50 repetitions). With $k \geq 100$ and $b \geq 8$. b-bit hashing (solid) achieves very similar accuracies as using the original data (dashed, red if color is available). Note that after $k \geq 150$, the curves for $b = 16$ overlap the curves for $b = 8$.
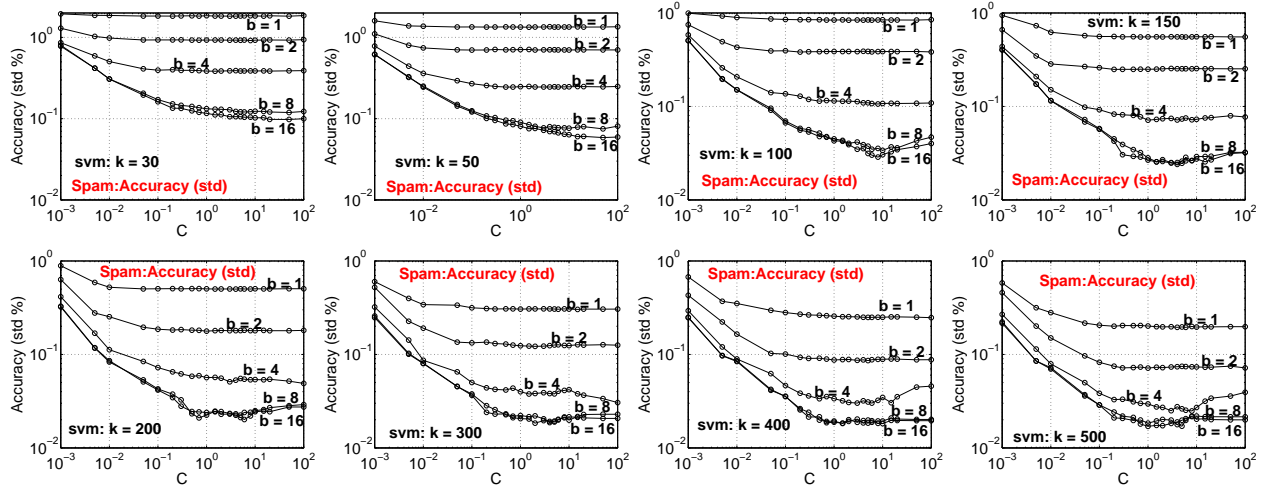

Figure 2: **Linear SVM test accuracy (std)**. The standard deviations are computed from 50 repetitions. When $b \geq 8$, the standard deviations become extremely small (e.g., 0.02%).

Compared with the original training time (about 100 seconds), we can see from Figure 3 that our method only need about $3 \sim 7$ seconds near $C = 1$ (about 3 seconds for $b = 8$). Note that here the training time did not include the data loading time. Loading the original data took about 12 minutes while loading the hashed data took only about 10 seconds. Of course, there is a cost for processing (hashing) the data, which we find is efficient, confirming prior studies [6]. In fact, data processing can be conducted during data collection, as is the standard practice in search. In other words, prior to conducting the learning procedure, the data may be already processed and stored by (b-bit) min-wise hashing, which can be used for multiple tasks including learning, clustering, duplicate detection, near-neighbor search, etc.

Compared with the original testing time (about $100 \sim 200$ seconds), we can see from Figure 4 that the testing time of our method is merely about 1 or 2 seconds. Note that the testing time includes both the data loading time and computing time, as designed by LIBLINEAR. The efficiency of testing may be very important in practice, for example, when the classifier is deployed in an user-facing application (such as search), while the cost of training or pre-processing (such as hashing) may be less critical and can often be conducted off-line.
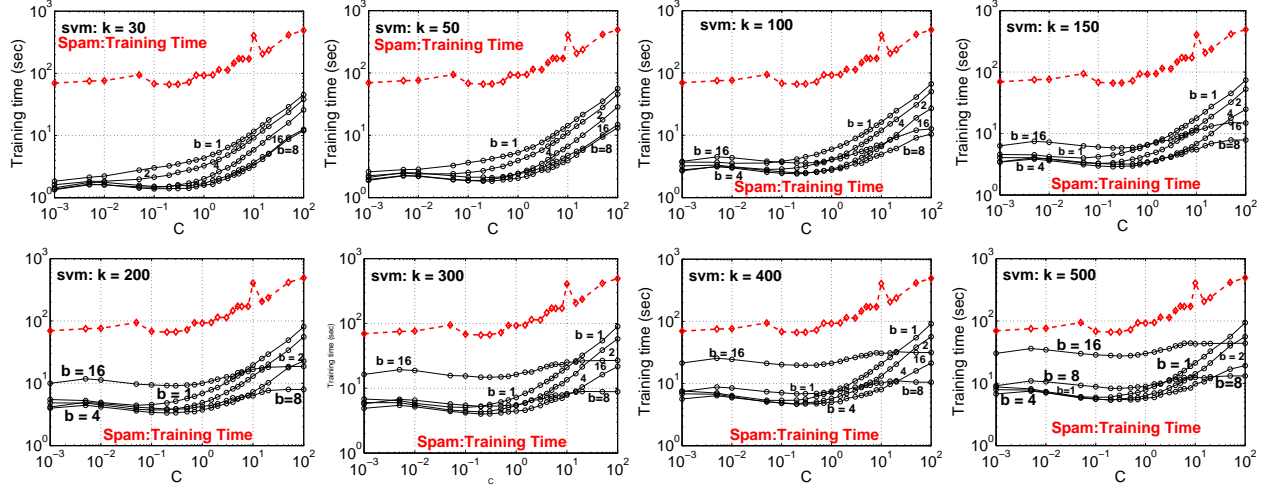
Figure 3: **Linear SVM Training time**. Compared with the training time of using the original data (dashed, red if color is available), we can see that our method with b-bit hashing only needs a very small fraction of the original cost.
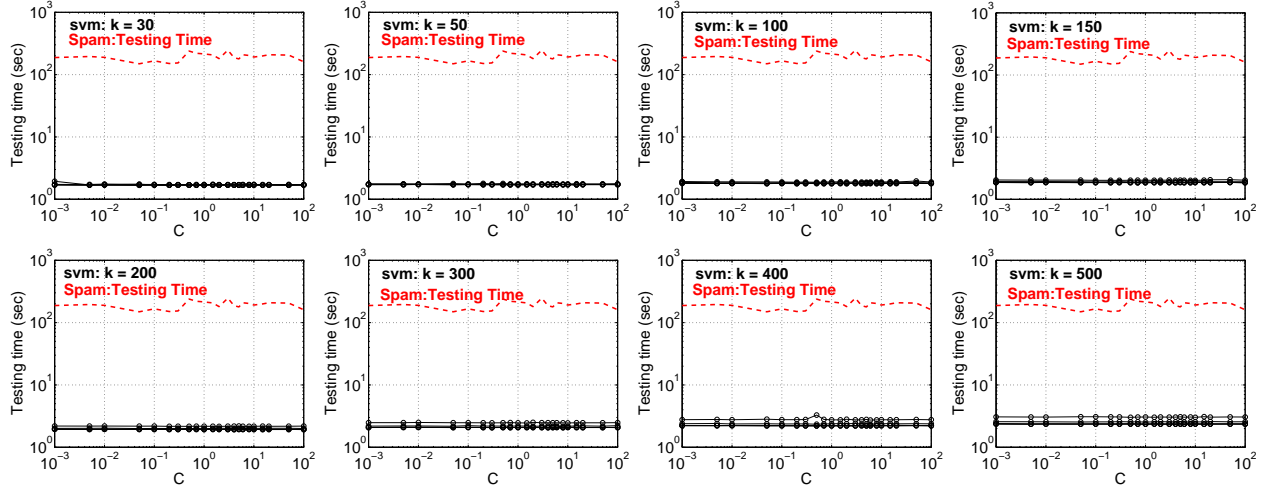

Figure 4: **Linear SVM testing time**. The original costs are plotted using dashed (red, if color is available) curves.

5.3 Experimental Results on Logistic Regression

Figure 5 presents the test accuracies and Figure 7 presents the training time using logistic regression. Again, with $k \geq 150$ (or even $k \geq 100$) and $b \geq 8$, b-bit minwise hashing can achieve the same test accuracies as using the original data. Figure 6 presents the standard deviations, which again verify that our algorithm produces stable predictions for logistic regression.

From Figure 7, we can see that the training time is substantially reduced, from about 1000 seconds to about $30 \sim 50$ seconds only (unless $b = 16$ and k is large).

In summary, it appears b-bit hashing is highly effective in reducing the data size and speeding up the training (and testing), for both (nonlinear and linear) 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 combined with *Vowpal Wabbit (VW)* [34] to further reduce the training time, especially when b is large.
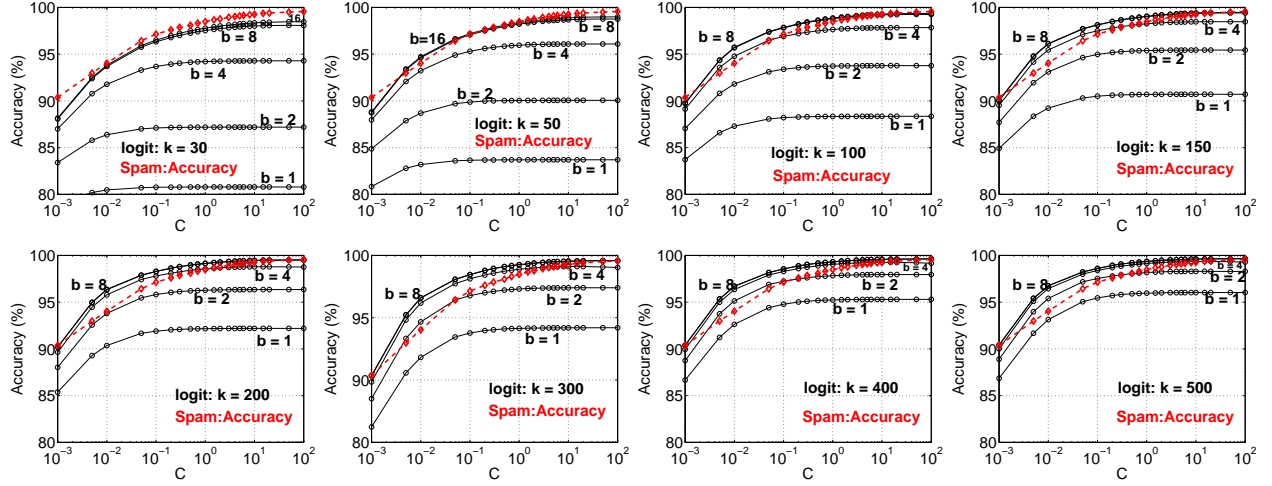
Figure 5: **Logistic regression test accuracy**. The dashed (red if color is available) curves represents the results using the original data.
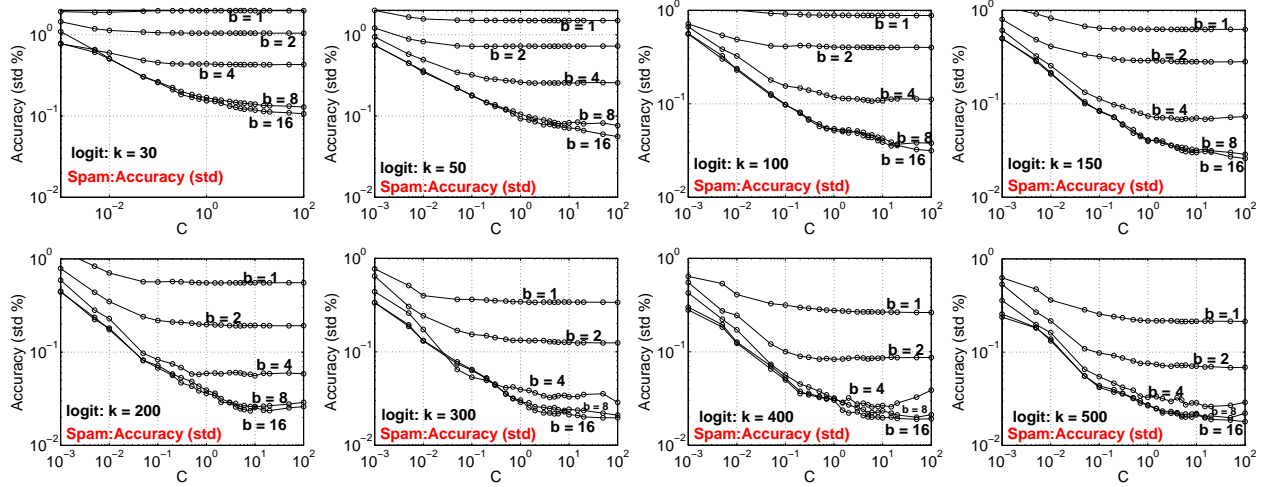

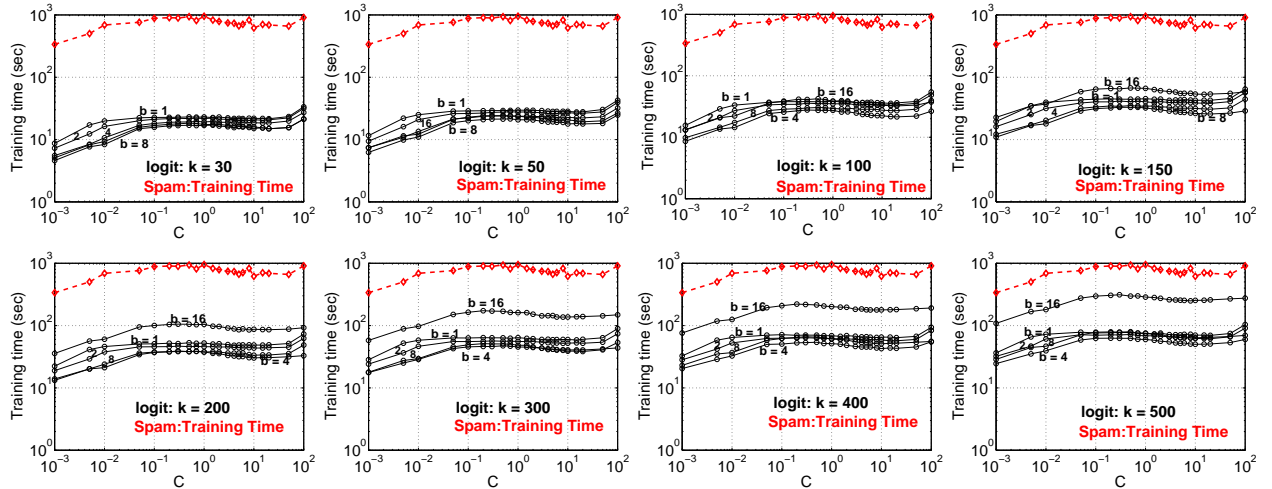Figure 6: **Logistic regression test accuracy (std)**. The standard deviations are computed from 50 repetitions.

Figure 7: **Logistic Regression Training time**. The dashed (red if color is available) curves represents the results using the original data.

6 Random Projections and Vowpal Wabbit (VW)

The two methods, random projections [1, 24] and Vowpal Wabbit (VW) [34, 31] are not limited to binary data (although for ultra high-dimensional used in the context of search, the data are often binary). The VW algorithm is also related to the Count-Min sketch [12]. In this paper, we use "VW" particularly for the algorithm in [34].

For convenience, we denote two D-dim data vectors by $u_1, u_2 \in \mathbb{R}^D$. Again, the task is to estimate the inner product $a = \sum_{i=1}^{D} u_{1,i} u_{2,i}$.

6.1 Random Projections

The general idea is to multiply the data vectors, e.g., u_1 and u_2, 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]

$$E(r_{ij}) = 0, \ \ Var(r_{ij}) = 1, \ \ E(r_{ij}^3) = 0, \ \ E(r_{ij}^4) = s, \ \ s \geq 1. \tag{11}$$

We must have $s \geq 1$ because $Var(r_{ij}^2) = E(r_{ij}^4) - E^2(r_{ij}^2) = s - 1 \geq 0$.

This generates two k-dim vectors, v_1 and v_2:

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

The general distributions which satisfy (11) includes the standard normal distribution (in this case, $s = 3$) and the "sparse projection" distribution specified as

$$r_{ij} = \sqrt{s} \times \begin{cases} 1 & \text{with prob. } \frac{1}{2s} \\ 0 & \text{with prob. } 1 - \frac{1}{s} \\ -1 & \text{with prob. } \frac{1}{2s} \end{cases} \tag{12}$$

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

$$\hat{a}_{rp,s} = \frac{1}{k} \sum_{j=1}^{k} v_{1,j} v_{2,j}, \ \ \ \ \ \ \ \ E(\hat{a}_{rp,s}) = a = \sum_{i=1}^{D} u_{1,i} u_{2,i}, \tag{13}$$

$$Var(\hat{a}_{rp,s}) = \frac{1}{k} \left[\sum_{i=1}^{D} u_{1,i}^2 \sum_{i=1}^{D} u_{2,i}^2 + \left(\sum_{i=1}^{D} u_{1,i} u_{2,i} \right)^2 + (s-3) \sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 \right] \tag{14}$$

which means $s = 1$ achieves the smallest variance. The only elementary distribution we know that satisfies (11) with $s = 1$ is the two point distribution in $\{-1, 1\}$ with equal probabilities, i.e., (12) with $s = 1$.

6.2 Vowpal Wabbit (VW)

Again, in this paper, "VW" always refers to the particular algorithm in [34]. VW may be viewed as a "bias-corrected" version of the Count-Min (CM) sketch algorithm [12]. In the original CM algorithm, the key step is to independently and uniformly hash elements of the data vectors to buckets $\in \{1, 2, 3, ..., k\}$ and the hashed value is the sum of the elements in the bucket. That is $h(i) = j$ with probability $\frac{1}{k}$, where $j \in \{1, 2, ..., k\}$. For convenience, we introduce an indicator function:

$$I_{ij} = \begin{cases} 1 & \text{if } h(i) = j \\ 0 & \text{otherwise} \end{cases}$$

which allow us to write the hashed data as

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

The estimate $\hat{a}_{cm} = \sum_{j=1}^{k} w_{1,j} w_{2,j}$ is (severely) biased for the task of estimating the inner products. The original paper [12] 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 not remove the bias and makes the analysis (such as variance) very difficult. Here we should mention that the bias of CM may not be a major issue in other tasks such as sparse recovery (or "heavy-hitter", or "elephant detection", by various communities).

[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, which corresponds to $s = 1$ in (12).

Here, we consider a more general situation by considering any $s \geq 1$. After applying multiplication and hashing on u_1 and u_2 as in [34], the resultant vectors g_1 and g_2 are

$$g_{1,j} = \sum_{i=1}^{D} u_{1,i} r_i I_{ij}, \qquad g_{2,j} = \sum_{i=1}^{D} u_{2,i} r_i I_{ij}, \quad j = 1, 2, ..., k \qquad (15)$$

where r_i is defined as in (11), i.e., $E(r_i) = 0$, $E(r_i^2) = 1$, $E(r_i^3) = 0$, $E(r_i^4) = s$. We have the following Lemma.

Lemma 1

$$\hat{a}_{vw,s} = \sum_{j=1}^{k} g_{1,j} g_{2,j}, \qquad E(\hat{a}_{vw,s}) = \sum_{i=1}^{D} u_{1,i} u_{2,i} = a, \qquad (16)$$

$$Var(\hat{a}_{vw,s}) = (s-1) \sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 + \frac{1}{k} \left[\sum_{i=1}^{D} u_{1,i}^2 \sum_{i=1}^{D} u_{2,i}^2 + \left(\sum_{i=1}^{D} u_{1,i} u_{2,i} \right)^2 - 2 \sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 \right] \qquad (17)$$

Proof: *See Appendix B.*□

Interestingly, the variance (17) says we do need $s = 1$, otherwise the additional term $(s-1) \sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2$ 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 (17) becomes identical to the variance of random projections (14).

7 Comparing b-Bit Minwise Hashing with VW

We implemented VW (which, in this paper, always refers to the algorithm developed in [34]) and tested it on the same webspam dataset. Figure 8 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). For example, 8-bit minwise hashing with $k = 200$ achieves about the same test accuracy as VW with $k = 10^6$. Note that we only stored the non-zeros of the hashed data generated by VW.
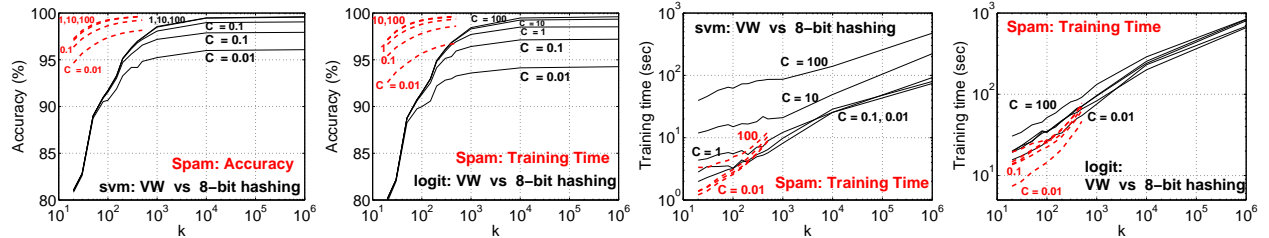

Figure 8: The dashed (red if color is available) curves represent b-bit minwise hashing results (only for $k \leq 500$) while solid curves represent VW. We display results for $C = 0.01, 0.1, 1, 10, 100$.

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 Appendix C, 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 requires 32 bits storage. Of course, even if VW only stores each sample using 16 bits, an improvement of 5- to 50-fold would still be very substantial. Note that this comparison makes sense for the purpose of *data reduction*, i.e., the sample size k is substantially smaller than the number of non-zeros in the original (massive) data.

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. When $\frac{c}{k} \geq 5$, then $\exp\left(-\frac{c}{k}\right) \approx 0$. In other words, if our goal is *data reduction* (i.e., $k \ll c$), then the hashed data by VW are dense.

In this paper, we mainly focus on *data reduction*. As discussed in the introduction, for many industry applications, the relatively sparse datasets are often massive in the absolute scale and we assume we can not store all the non-zeros. In fact, this is the also one of the basic motivations for developing minwise hashing.

However, the case of $c \ll k$ can also be interesting and useful in our work. This is because VW is an excellent tool for achieving *compact indexing* due to the *sparsity-preserving* property. Basically, we can let k be very large (like 2^{26} in [34]). As the original dictionary size D is extremely large (e.g., 2^{64}), even $k = 2^{26}$ will be a meaningful reduction of the indexing. Of course, using a very large k will not be useful for the purpose of *data reduction*.

8 Combining b-Bit Minwise Hashing with VW

In our algorithm, we reduce the original massive data to nbk bits only, where n is the number of data points. With (e.g.,) $k = 200$ and $b = 8$, our technique achieves a huge *data reduction*. In the run-time, we need to expand each data point into a binary vector of length $2^b k$ with exactly k 1's. If b is large like 16, the new binary vectors will be highly sparse. In fact, in Figure 3 and Figure 7, we can see that when using $b = 16$, the training time becomes substantially larger than using $b \leq 8$ (especially when k is large).

On the other hand, once we have expanded the vectors, the task is merely computing inner products, for which we can actually 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? Lemma 2 may provide some insights.

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 (on top of b-bit hashing). We denote the new estimator by $\hat{R}_{b,vw}$. Lemma 2 provides its theoretical variance.

Lemma 2

$$E\left(\hat{R}_{b,vw}\right) = R \tag{18}$$

$$Var\left(\hat{R}_{b,vw}\right) = Var\left(\hat{R}_b\right) + \frac{1}{m}\frac{1}{[1 - C_{2,b}]^2}\left(1 + P_b^2 - \frac{P_b(1 + P_b)}{k}\right), \tag{19}$$

$$= \frac{1}{k}\frac{P_b(1 - P_b)}{[1 - C_{2,b}]^2} + \frac{1}{m}\frac{1 + P_b^2}{[1 - C_{2,b}]^2} - \frac{1}{mk}\frac{P_b(1 + P_b)}{[1 - C_{2,b}]^2}$$

where $Var\left(\hat{R}_b\right) = \frac{1}{k}\frac{P_b(1 - P_b)}{[1 - C_{2,b}]^2}$ *is given by (6) and* $C_{2,b}$ *is the constant defined in Theorem 1.*

Proof: *The proof is quite straightforward, by following the conditional expectation formula:* $E(X) = E(E(X|Y))$, *and the conditional variance formula* $Var(X) = E(Var(X|Y)) + Var(E(X|Y))$.

Recall, originally we estimate the resemblance by $\hat{R}_b = \frac{\hat{P}_b - C_{1,b}}{[1 - C_{2,b}]}$, *where* $\hat{P}_b = \frac{T}{k}$ *and* T *is the number of matches in the two hashed data vectors of length* k *generated by* b-bit hashing. $E(T) = kP_b$ *and* $Var(T) = kP_b(1 - P_b)$.

Now, we apply VW (of size m and $s = 1$) on the hashed data vectors to estimate T (instead of counting it exactly). We denote this estimates by $\hat{T}$ and $\hat{P}_{b,vw} = \frac{\hat{T}}{k}$.

Because we know the VW estimate is unbiased, we have

$$E\left(\hat{P}_{b,vw}\right) = E\left(\frac{\hat{T}}{k}\right) = \frac{E(T)}{k} = \frac{kP_b}{k} = P_b.$$

Using the conditional variance formula and the variance of VW (17) (with $s = 1$), we obtain

$$
\begin{aligned}
Var\left(\hat{P}_{b,vw}\right) =& \frac{1}{k^2}\left[E\left(\frac{1}{m}\left[k^2 + T^2 - 2T\right]\right) + Var\left(T\right)\right] \\
=& \frac{1}{k^2}\left[\frac{1}{m}\left(k^2 + kP_b(1 - P_b) + k^2 P_b^2 - 2kP_b\right) + kP_b(1 - P_b)\right] \\
=& \frac{P_b(1 - P_b)}{k} + \frac{1}{m}\left(1 + P_b^2 - \frac{P_b(1 + P_b)}{k}\right)
\end{aligned}
$$

This completes the proof. □

Compared to the original variance $Var\left(\hat{R}_b\right) = \frac{1}{k}\frac{P_b(1-P_b)}{[1-C_{2,b}]^2}$, the additional term $\frac{1}{m}\frac{1+P_b^2}{[1-C_{2,b}]^2}$ in (19) can be relatively large if m is not large enough. Therefore, we should choose $m \gg k$ (to reduce the additional variance) and $m \ll 2^b k$ (otherwise there is no need to apply this VW step). If $b = 16$, then $m = 2^8 k$ may be a good trade-off, because $k \ll 2^8 k \ll 2^{16} k$.

Figure 8 provides an empirical study to verify this intuition. Basically, as $m = 2^8 k$, using VW on top of 16-bit hashing achieves the same accuracies at using 16-bit hashing directly and reduces the training time quite noticeably.
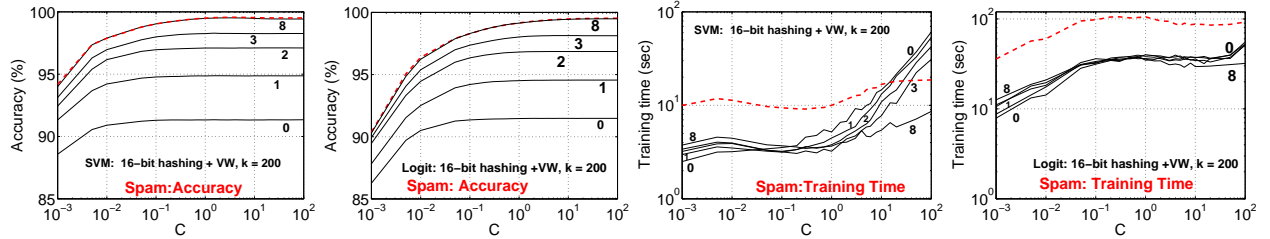

Figure 9: We apply VW hashing on top of the binary vectors (of length $2^b k$) generated by b-bit hashing, with size $m = 2^0 k, 2^1 k, 2^2 k, 2^3 k, 2^8 k$, for $k = 200$ and $b = 16$. The numbers on the solid curves (0, 1, 2, 3, 8) are the exponents. The dashed (red if color if available) curves are the results from only using b-bit hashing. When $m = 2^8 k$, this method achieves the same test accuracies (left two panels) while considerably reducing the training time (right two panels), if we focus on $C \approx 1$.

We also experimented with combining 8-bit hashing with VW. We found that we need $m = 2^8 k$ to achieve similar accuracies, i.e., the additional VW step did not bring more improvement (without hurting accuracies) in terms of training speed when $b = 8$. This is understandable from the analysis of the variance in Lemma 2.

9 Practical Considerations

Minwise hashing has been widely used in (search) industry and b-bit minwise hashing requires only very minimal (if any) modifications (by doing less work). Thus, we expect b-bit minwise hashing will be adopted in practice. It is also well-understood in practice that we can use (good) hashing functions to very efficiently simulate permutations.

In many real-world scenarios, the preprocessing step is not critical because it requires only one scan of the data, which can be conducted off-line (or on the data-collection stage, or at the same time as n-grams are generated), and it is trivially parallelizable. In fact, because b-bit minwise hashing can substantially reduce the memory consumption, it may be now affordable to store considerably more examples in the memory (after b-bit hashing) than before, to

avoid (or minimize) disk IOs. Once the hashed data have been generated, they can be used and re-used for many tasks such as supervised learning, clustering, duplicate detections, near-neighbor search, etc. For example, a learning task may need to re-use the same (hashed) dataset to perform many cross-validations and parameter tuning (e.g., for experimenting with many C values in SVM).

Nevertheless, there might be situations in which the preprocessing time can be an issue. For example, when a new unprocessed document (i.e. n-grams are not yet available) arrives and a particular application requires an immediate response from the learning algorithm, then the preprocessing cost might (or might not) be an issue. Firstly, generating n-grams will take some time. Secondly, if during the session a disk IO occurs, then the IO cost will typically mask the cost of preprocessing for b-bit minwise hashing.

Note that the preprocessing cost for the VW algorithm can be substantially lower. Thus, if the time for preprocessing is indeed a concern (while the storage cost or test accuracies are not as much), one may want to consider using VW (or *very sparse random projections* [24]) for those applications.

10   Conclusion

As data sizes continue to grow faster than the memory and computational power, machine-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 datasets based on b-bit minwise hashing. We show that the b-bit minwise hashing estimators are positive definite kernels and can be naturally integrated with learning algorithms such as SVM and logistic regression, leading to dramatic improvements in training time and/or resource requirements. We also compare b-bit minwise hashing with the Vowpal Wabbit (VW) algorithm, which has the same variances as random projections. Our theoretical and empirical comparisons illustrate that usually b-bit minwise hashing is significantly more accurate (at the same storage) than VW for binary data. Interestingly, b-bit minwise hashing can be combined with VW to achieve further improvements in terms of training speed when b is large (e.g., $b \geq 16$).

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

[19] 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.

[20] 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.

[21] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *WSDM*, pages 219–230, Palo Alto, California, USA, 2008.

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

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

[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*, To Appear.

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

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

[28] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting Near-Duplicates for Web-Crawling. In *WWW*, Banff, Alberta, Canada, 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, Barcelona, Spain, 2009.

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

[31] 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.

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

[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] 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.

A   Approximation Errors of the Basic Probability Formula

Note that the only assumption needed in the proof of Theorem 1 is that D is large, which is virtually always satisfied in practice. Interestingly, (4) is remarkably accurate even for very small D. Figure 10 shows that when $D = 20/200/500$, the absolute error caused by using (4) is $< 0.01/0.001/0.0004$. The exact probability, which has no closed-form, can be computed by exhaustive enumerations for small D.
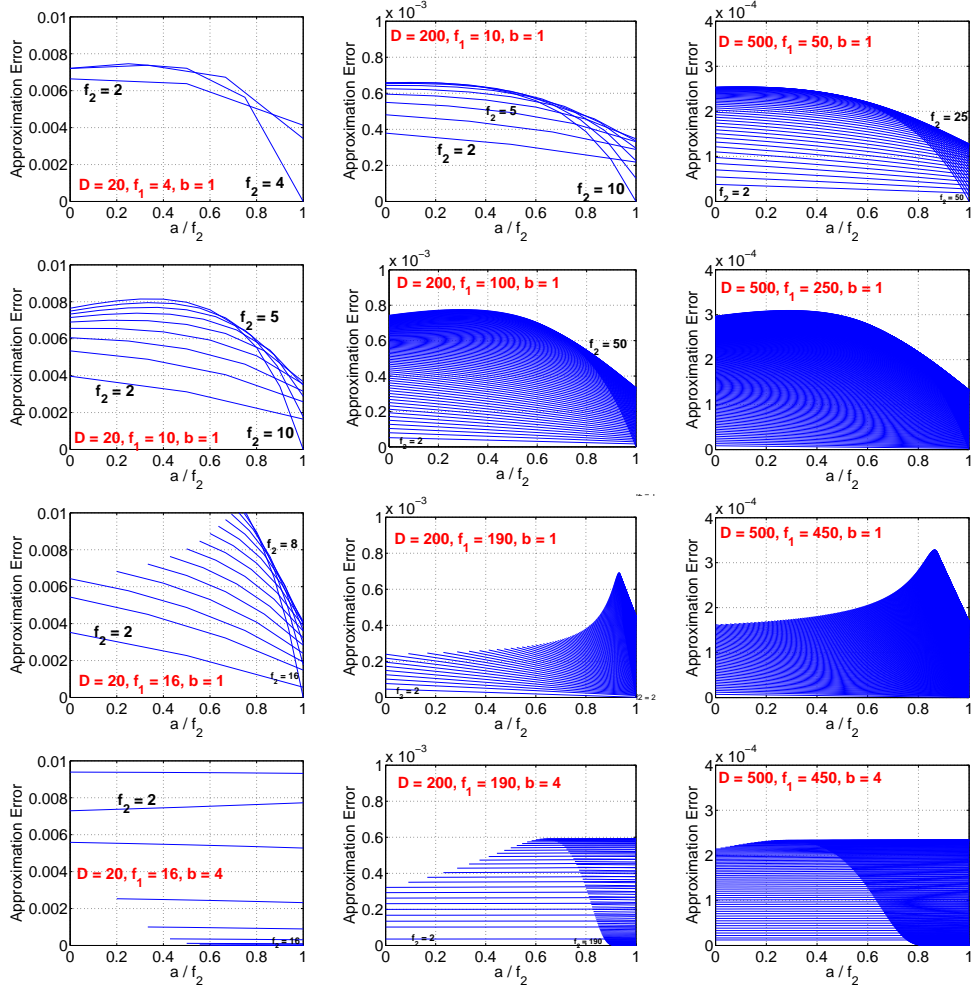

Figure 10: The absolute errors (approximate - exact) by using (4) are very small even for $D = 20$ (left panels), $D = 200$ (middle panels), and $D = 500$ (right panels). The exact probability can be numerically computed for small D (from a probability matrix of size $D \times D$). For each D, we selected three f_1 values. We always let $f_2 = 2, 3, ..., f_1$ and $a = 0, 1, 2, ..., f_2$.

B   Proof of Lemma 1

The VW algorithm [34] provides a bias-corrected version of the Count-Min (CM) sketch algorithm [12].

B.1   The Analysis of the CM Algorithm

The key step in CM is to independently and uniformly hash elements of the data vectors to $\{1, 2, 3, ..., k\}$. That is $h(i) = q$ with equal probabilities, where $q \in \{1, 2, ..., k\}$. For convenience, we introduce the following indicator

function:

$$I_{iq} = \begin{cases} 1 & \text{if } h(i) = q \\ 0 & \text{otherwise} \end{cases}$$

Thus, we can denote the CM "samples" after the hashing step by

$$w_{1,q} = \sum_{i=1}^{D} u_{1,i} I_{iq}, \qquad w_{2,q} = \sum_{i=1}^{D} u_{2,i} I_{iq}$$

and estimate the inner product by $\hat{a}_{cm} = \sum_{q=1}^{k} w_{1,q} w_{2,q}$, whose expectation and variance can be shown to be

$$E(\hat{a}_{cm}) = \sum_{i=1}^{D} u_{1,i} u_{2,i} + \frac{1}{k} \sum_{i \neq j} u_{1,i} u_{2,j} = a + \frac{1}{k} \sum_{i=1}^{D} u_{1,i} \sum_{i=1}^{D} u_{2,i} - \frac{1}{k} a \tag{20}$$

$$Var(\hat{a}_{cm}) = \frac{1}{k}\left(1 - \frac{1}{k}\right)\left[\sum_{i=1}^{D} u_{1,i}^2 \sum_{i=1}^{D} u_{2,i}^2 + \left(\sum_{i=1}^{D} u_{1,i} u_{2,i}\right)^2 - 2\sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2\right] \tag{21}$$

From the definition of I_{iq}, we can easily infer its moments, for example,

$$I_{iq} = \begin{cases} 1 & \text{if } h(i) = q \\ 0 & \text{otherwise} \end{cases}, \qquad E(I_{iq}^n) = \frac{1}{k}, \qquad E(I_{iq} I_{iq'}) = 0 \text{ if } q \neq q', \qquad E(I_{iq} I_{i'q'}) = \frac{1}{k^2} \text{ if } i \neq i'$$

The proof of the mean (20) is simple:

$$E(\hat{a}_{cm}) = \sum_{q=1}^{k}\sum_{i=1}^{D} u_{1,i} I_{iq} \sum_{i=1}^{D} u_{2,i} I_{iq} = \sum_{q=1}^{k}\left[\sum_{i=1}^{D} u_{1,i} u_{2,i} E\left(I_{iq}^2\right) + \sum_{i \neq j} u_{1,i} u_{2,j} E\left(I_{iq} I_{jq}\right)\right] = \sum_{i=1}^{D} u_{1,i} u_{2,i} + \frac{1}{k}\sum_{i \neq j} u_{1,i} u_{2,j}.$$

The variance (21 is more complicated:

$$Var(\hat{a}_{cm}) = E(\hat{a}_{cm}^2) - E^2(\hat{a}_{cm}) = \sum_{q=1}^{k} E(w_{1,q}^2 w_{2,q}^2) + \sum_{q \neq q'} E(w_{1,q} w_{2,q} w_{1,q'} w_{2,q'}) - \left(\sum_{i=1}^{D} u_{1,i} u_{2,i} + \frac{1}{k}\sum_{i \neq j} u_{1,i} u_{2,j}\right)^2$$

The following expansions are helpful:

$$\left[\sum_{i=1}^{D} a_i \sum_{i=1}^{D} b_i\right]^2 = \sum_{i=1}^{D} a_i^2 b_i^2 + \sum_{i \neq j} a_i^2 b_j^2 + 2a_i^2 b_i b_j + 2b_i^2 a_i a_j + 2a_i b_i a_j b_j$$

$$+ \sum_{i \neq j \neq c} a_i^2 b_j b_c + b_i^2 a_j a_c + 4a_i b_i a_j b_c + \sum_{i \neq j \neq c \neq t} a_i b_j a_c b_t$$

$$\left[\sum_{i \neq j} a_i b_j\right]^2 = \sum_{i \neq j} a_i^2 b_j^2 + a_i b_i a_j b_j + \sum_{i \neq j \neq c} a_i^2 b_j b_c + b_i^2 a_j a_c + 2a_i b_i a_j b_c + \sum_{i \neq j \neq c \neq t} a_i b_j a_c b_t$$

$$\sum_{i=1}^{D} a_i b_i \sum_{i \neq j} a_i b_j = \sum_{i \neq j} a_i^2 b_i b_j + b_i^2 a_i a_j + \sum_{i \neq j \neq c} a_i^2 b_j b_c$$

which, combined with the moments of I_{iq}, yield

$$\sum_{q=1}^{k} E(w_{1,q}^2 w_{2,q}^2) = \sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 + \frac{1}{k}\sum_{i \neq j} u_{1,i}^2 u_{2,j}^2 + 2u_{1,i}^2 u_{2,i} u_{2,j} + 2u_{2,i}^2 u_{1,i} u_{1,j} + 2u_{1,i} u_{2,i} u_{1,j} u_{2,j}$$

$$+ \frac{1}{k^2}\sum_{i \neq j \neq c} u_{1,i}^2 u_{2,j} u_{2,c} + u_{2,i}^2 u_{1,j} u_{1,c} + 4u_{1,i} u_{2,i} u_{1,j} u_{2,c} + \frac{1}{k^3}\sum_{i \neq j \neq c \neq t} u_{1,i} u_{2,j} u_{1,c} u_{2,t}$$

16

$$\sum_{q\neq q'} E(w_{1,q}w_{2,q}w_{1,q'}w_{2,q'})$$

$$=k(k-1)\left[\frac{1}{k^2}\sum_{i\neq j}u_{1,i}u_{2,i}u_{1,j}u_{2,j}+\frac{2}{k^3}\sum_{i\neq j\neq c}u_{1,i}u_{2,i}u_{1,j}u_{2,c}+\frac{1}{k^4}\sum_{i\neq j\neq c\neq t}u_{1,i}u_{2,j}u_{1,c}u_{2,t}\right]$$

$$=(k-1)\left[\frac{1}{k}\sum_{i\neq j}u_{1,i}u_{2,i}u_{1,j}u_{2,j}+\frac{2}{k^2}\sum_{i\neq j\neq c}u_{1,i}u_{2,i}u_{1,j}u_{2,c}+\frac{1}{k^3}\sum_{i\neq j\neq c\neq t}u_{1,i}u_{2,j}u_{1,c}u_{2,t}\right]$$

$$E(\hat{a}_{cm}^2)=\sum_{q=1}^{k}E(w_{1,q}^2w_{2,q}^2)+\sum_{q\neq q'}E(w_{1,q}w_{2,q}w_{1,q'}w_{2,q'})$$

$$=\sum_{i=1}^{D}u_{1,i}^2u_{2,i}^2+\sum_{i\neq j}u_{1,i}u_{2,i}u_{1,j}u_{2,j}+\frac{1}{k}\sum_{i\neq j}u_{1,i}^2u_{2,j}^2+2u_{1,i}^2u_{2,i}u_{2,j}+2u_{2,i}^2u_{1,i}u_{1,j}+u_{1,i}u_{2,i}u_{1,j}u_{2,j}+\frac{2}{k}\sum_{i\neq j\neq c}u_{1,i}u_{2,i}u_{1,j}u_{2,c}$$

$$+\frac{1}{k^2}\sum_{i\neq j\neq c}u_{1,i}^2u_{2,j}u_{2,c}+u_{2,i}^2u_{1,j}u_{1,c}+2u_{1,i}u_{2,i}u_{1,j}u_{2,c}+\frac{1}{k^2}\sum_{i\neq j\neq c\neq t}u_{1,i}u_{2,j}u_{1,c}u_{2,t}$$

$$=\left(\sum_{i=1}^{D}u_{1,i}u_{2,i}\right)^2+\left(\frac{1}{k}-\frac{1}{k^2}\right)\sum_{i\neq j}u_{1,i}^2u_{2,j}^2+u_{1,i}u_{2,i}u_{1,j}u_{2,j}+\frac{2}{k}\sum_{i=1}^{D}u_{1,i}u_{2,i}\sum_{i\neq j}u_{1,i}u_{2,j}+\frac{1}{k^2}\left(\sum_{i\neq j}u_{1,i}u_{2,j}\right)^2$$

Therefore,

$$Var(\hat{a}_{cm})=E(\hat{a}_{cm}^2)-\left(\sum_{i=1}^{D}u_{1,i}u_{2,i}+\frac{1}{k}\sum_{i\neq j}u_{1,i}u_{2,j}\right)^2$$

$$=\left(\frac{1}{k}-\frac{1}{k^2}\right)\sum_{i\neq j}u_{1,i}^2u_{2,j}^2+u_{1,i}u_{2,i}u_{1,j}u_{2,j}$$

$$=\frac{1}{k}\left(1-\frac{1}{k}\right)\left[\sum_{i=1}^{D}u_{1,i}^2\sum_{i=1}^{D}u_{2,i}^2+\left(\sum_{i=1}^{D}u_{1,i}u_{2,i}\right)^2-2\sum_{i=1}^{D}u_{1,i}^2u_{2,i}^2\right]$$

B.2 The Analysis of the More General Version of the VW Algorithm

The nice approach proposed in the VW paper [34] is to pre-element-wise-multiply the data vectors with a random vector r before taking the hashing operation. We denote the two resultant vectors (samples) by g_1 and g_2 respectively:

$$g_{1,q}=\sum_{i=1}^{D}u_{1,i}r_iI_{iq},\qquad g_{2,q}=\sum_{i=1}^{D}u_{2,i}r_iI_{iq}$$

where $r_i\in\{-1,1\}$ with equal probabilities. Here, we provide a more general scheme by sampling r_i from a sub-Gaussian distribution with parameter s and

$$E(r_i)=0,\ E(r_i^2)=1,\ E(r_i^3)=0,\ E(r_i^4)=s$$

which include normal (i.e., $s=3$) and the distribution on $\{-1,\ 1\}$ with equal probabilities (i.e., $s=1$) as special cases.

Let $\hat{a}_{vw,s} = \sum_{q=1}^{k} g_{1,q} g_{2,q}$. The goal is to show

$$E(\hat{a}_{vw,s}) = \sum_{i=1}^{D} u_{1,i} u_{2,i}$$

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

We can use the previous results and the conditional expectation and variance formulas:

$$E(X) = E(E(X|Y)), \qquad Var(X) = E(Var(X|Y)) + Var(E(X|Y))$$

$E(r_i) = 0$, $E(r_i^2) = 1$, $E(r_i^3) = 0$, $E(r_i^4) = s$.

$$E(\hat{a}_{vw,s}) = E(E(\hat{a}_{vw,s}|r)) = E\left(\sum_{i=1}^{D} u_{1,i}u_{2,i}r_i^2 + \frac{1}{k}\sum_{i\neq j} u_{1,i}u_{2,j}r_i r_j\right) = \sum_{i=1}^{D} u_{1,i}u_{2,i}$$

$$E(Var(\hat{a}_{vw,s}|r)) = \left(\frac{1}{k} - \frac{1}{k^2}\right) E\left(\sum_{i\neq j} u_{1,i}^2 u_{2,j}^2 r_i^2 r_j^2 + u_{1,i}u_{2,i}u_{1,j}u_{2,j}r_i^2 r_j^2\right) = \left(\frac{1}{k} - \frac{1}{k^2}\right)\sum_{i\neq j} u_{1,i}^2 u_{2,j}^2 + u_{1,i}u_{2,i}u_{1,j}u_{2,j}$$

As $Var(E(\hat{a}_{vw,s}|r)) = E(E^2(\hat{a}_{vw,s}|r)) - E^2(E(\hat{a}_{vw,s}|r))$, we need to compute

$$E(E^2(\hat{a}_{vw,s}|r)) = E\left(\sum_{i=1}^{D} u_{1,i}u_{2,i}r_i^2 + \frac{1}{k}\sum_{i\neq j} u_{1,i}u_{2,j}r_i r_j\right)^2$$

$$= s\sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 + \sum_{i\neq j} u_{1,i}u_{2,i}u_{1,j}u_{2,j} + \frac{1}{k^2}\sum_{i\neq j} u_{1,i}^2 u_{2,j}^2 + u_{1,i}u_{2,i}u_{1,j}u_{2,j}$$

Thus,

$$Var(E(\hat{a}_{vw,s}|r)) = (s-1)\sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 + \frac{1}{k^2}\sum_{i\neq j} u_{1,i}^2 u_{2,j}^2 + u_{1,i}u_{2,i}u_{1,j}u_{2,j}$$

$$Var(\hat{a}_{vw,s}) = (s-1)\sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2 + \frac{1}{k}\sum_{i\neq j} u_{1,i}^2 u_{2,j}^2 + u_{1,i}u_{2,i}u_{1,j}u_{2,j}$$

B.3 Another Simple Scheme for Bias-Correction

By examining the expectation (20), the bias of CM can be easily removed, because

$$\hat{a}_{cm,nb} = \frac{k}{k-1}\left[\hat{a}_{cm} - \frac{1}{k}\sum_{i=1}^{D} u_{1,i} \sum_{i=1}^{D} u_{2,i}\right] \tag{22}$$

is unbiased with variance

$$Var(\hat{a}_{cm,nb}) = \frac{1}{k-1}\left[\sum_{i=1}^{D} u_{1,i}^2 \sum_{i=1}^{D} u_{2,i}^2 + \left(\sum_{i=1}^{D} u_{1,i}u_{2,i}\right)^2 - 2\sum_{i=1}^{D} u_{1,i}^2 u_{2,i}^2\right], \tag{23}$$

which is essentially the same as the variance of VW.

C   Comparing b-Bit Minwise Hashing with VW Random Projections

We compare VW (and random projections) with b-bit minwise hashing for the task of estimating inner products on binary data. With binary data, i.e., $u_{1,i}, u_{2,i} \in \{0,1\}$, we have $f_1 = \sum_{i=1}^{D} u_{1,i}$, $f_2 = \sum_{i=1}^{D} u_{2,i}$, $a = \sum_{i=1}^{D} u_{1,i} u_{2,i}$. The variance (17) (by using $s = 1$) becomes

$$Var\left(\hat{a}_{vw,s=1}\right) = \frac{f_1 f_2 + a^2 - 2a}{k}$$

We can compare this variance with the variance of b-bit minwise hashing. Because the variance (6) is for estimating the resemblance, we need to convert it into the variance for estimating the inner product a using the relation:

$$a = \frac{R}{1+R}(f_1 + f_2)$$

We can estimate a from the estimated R,

$$\hat{a}_b = \frac{\hat{R}_b}{1 + \hat{R}_b}(f_1 + f_2)$$

$$\mathrm{Var}\left(\hat{a}_b\right) = \left[\frac{1}{(1+R)^2}(f_1 + f_2)\right]^2 \mathrm{Var}(\hat{R}_b)$$

For b-bit minwise hashing, each sample is stored using only b bits. For VW (and random projections), we assume each sample is stored using 32 bits (instead of 64 bits) for two reasons: (i) for binary data, it would be very unlikely for the hashed value to be close to 2^{64}, even when $D = 2^{64}$; (ii) unlike b-bit minwise hashing, which requires exact bit-matching in the estimation stage, random projections only need to compute the inner products for which it would suffice to store hashed values as (double precision) real numbers.

Thus, we define the following ratio to compare the two methods.

$$G_{vw} = \frac{Var\left(\hat{a}_{vw,s=1}\right) \times 32}{Var\left(\hat{a}_b\right) \times b} \tag{24}$$

If $G_{vw} > 1$, then b-bit minwise hashing is more accurate than binary random projections. Equivalently, when $G_{vw} > 1$, in order to achieve the same level of accuracy (variance), b-bit minwise hashing needs smaller storage space than random projections. There are two issues we need to elaborate on:

1. Here, we assume the purpose of using VW is for *data reduction*. That is, k is small compared to the number of non-zeros (i.e., f_1, f_2). We do not consider the case when k is taken to be extremely large for the benefits of *compact indexing* without achieving *data reduction*.

2. Because we assume k is small, we need to represent the sample with enough precision. That is why we assume each sample of VW is stored using 32 bits. In fact, since the ratio G_{vw} is usually very large (e.g., $10 \sim 100$) by using 32 bits for each VW sample, it will remain to be very large (e.g., $5 \sim 50$) even if we only need to store each VW sample using 16 bits.

Without loss of generality, we can assume $f_2 \le f_1$ (hence $a \le f_2 \le f_1$). Figures 11 to 14 display the ratios (24) for $b = 8, 4, 2, 1$, respectively. In order to achieve high learning accuracies, b-bit minwise hashing requires $b = 4$ (or even 8). In each figure, we plot G_{BP} for $f_1/D = 0.0001, 0.1, 0.5, 0.9$ and full ranges of f_2 and a. We can see that G_{vw} is much larger than one (usually 10 to 100), indicating the very substantial advantage of b-bit minwise hashing over random projections.

Note that the comparisons are essentially independent of D. This is because in the variance of binary random projection (24) the $-2a$ term is negligible compared to a^2 in binary data as D is very large. To generate the plots, we used $D = 10^6$ (although practically D should be much larger).

Conclusion: Our theoretical analysis has illustrated the substantial improvements of b-bit minwise hashing over the VW algorithm and random projections in binary data, often by 10- to 100-fold. We feel such a large performance difference should be noted by researchers and practitioners in large-scale machine learning.
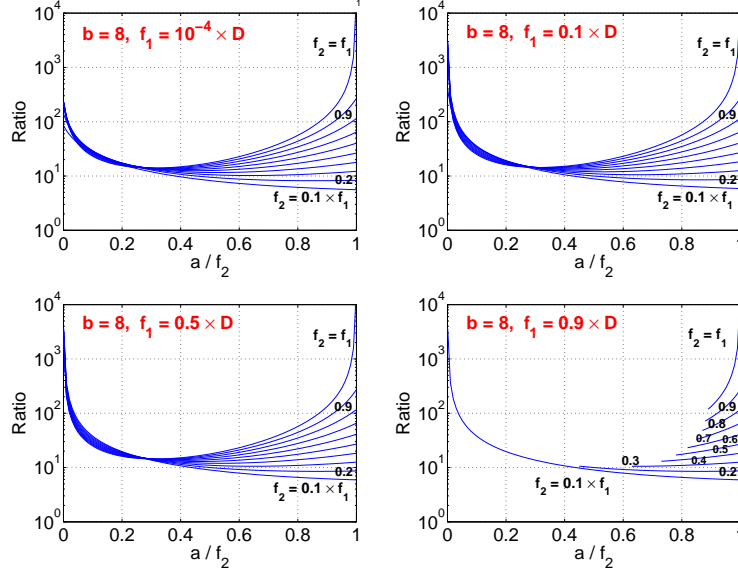
Figure 11: G_{vw} as defined in (24), for $b = 8$. $G_{vw} > 1$ means b-bit minwise hashing is more accurate than random projections at the same storage cost. We consider four f_1 values. For each f_1, we let $f_2 = 0.1f_1, 0.2f_1, ..., f_1$ and $a = 0$ to f_2.
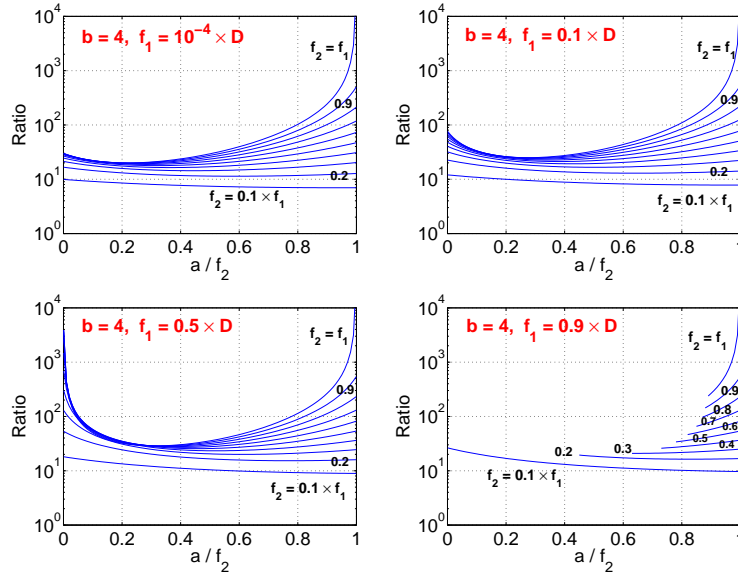

Figure 12: G_{vw} as defined in (24), for $b = 4$. They again indicate the very substantial improvement of b-bit minwise hashing over random projections.

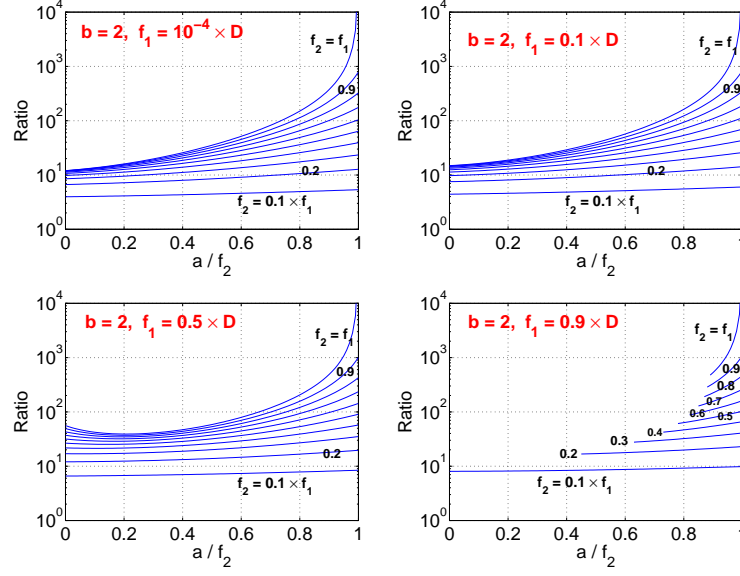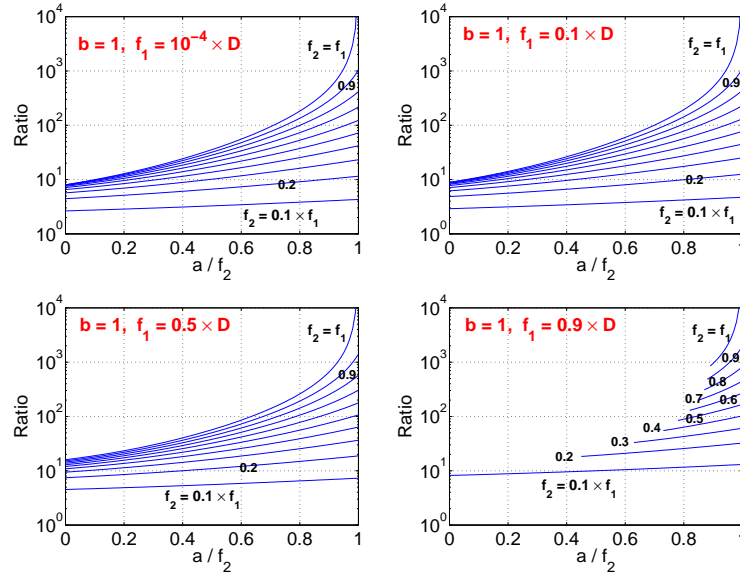Figure 13: G_{vw} as defined in (24), for $b = 2$.

Figure 14: G_{vw} as defined in (24), for $b = 1$.