b-Bit Minwise Hashing for Large-Scale Learning

Ping Li[*] Cornell University
Anshumali Shrivastava Cornell University
Joshua Moore Cornell University
Arnd Christian König Microsoft Research

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, [24] 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* [1], 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 [18–20] 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) [9, 13, 14, 23, 26] and logistic regression solvers.

2 Review of Minwise Hashing and b-Bit Minwise Hashing

Minwise hashing [4, 5] has been successfully applied to a wide range of real-world problems [2, 4–8, 11, 12, 22], 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 [10]. The storage (and computational) cost will be prohibitive in truly large-scale (industry) applications [21].

[*]A longer version will appear in the NIPS 2011 main conference. This work is supported by NSF (DMS-0808864), ONR (YIP-N000140910911), and a grant from Microsoft.

b-bit minwise hashing [18] provides a strikingly simple solution to this (storage and computational) problem by storing only the lowest b bits (instead of 64 bits) of each hashed value.

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

Theorem 1 *[18] Assume D is large.*

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

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

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

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

This (approximate) formula (3) is remarkably accurate, even for very small D; see Figure 1 in [16].

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.$\Box$

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$^{\text{perf}}$ [14], Pegasos [23], Bottou's SGD SVM [3], and LIBLINEAR [9]. 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):

$$\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^\mathbf{T}\mathbf{w} + C\sum_{i=1}^{n} \max\left\{1 - y_i \mathbf{w}^\mathbf{T}\mathbf{x_i},\ 0\right\}, \tag{5}$$

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

$$\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^\mathbf{T}\mathbf{w} + C\sum_{i=1}^{n} \log\left(1 + e^{-y_i \mathbf{w}^\mathbf{T}\mathbf{x_i}}\right). \tag{6}$$

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 [26]. 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 [26], 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 $C \geq 1$.

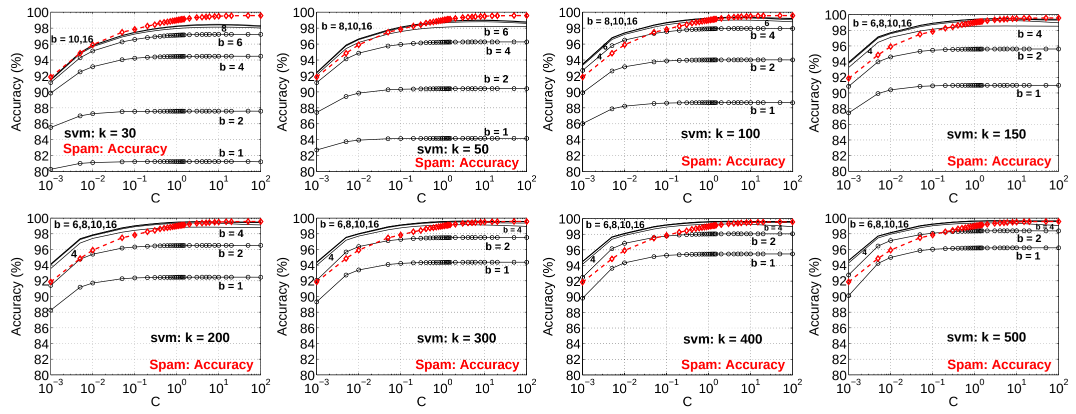

Figure 1: **SVM test accuracy** (averaged over 50 repetitions). With $k \geq 200$ and $b \geq 8$. b-bit hashing achieves very similar accuracies as using the original data (dashed, red if color is available).

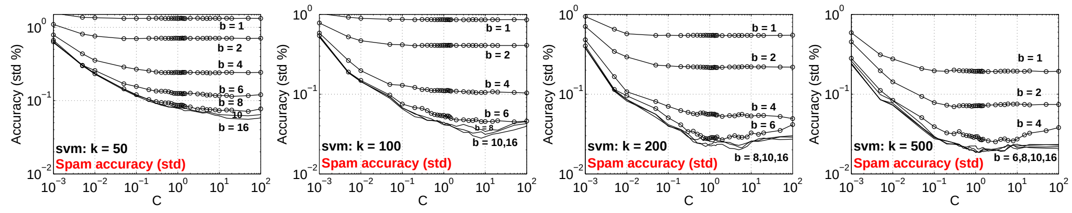

Figure 2: **SVM test accuracy (std)**. The standard deviations are computed from 50 repetitions. When $b \geq 8$, the standard deviations become extremely small (e.g., 0.02%).

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.

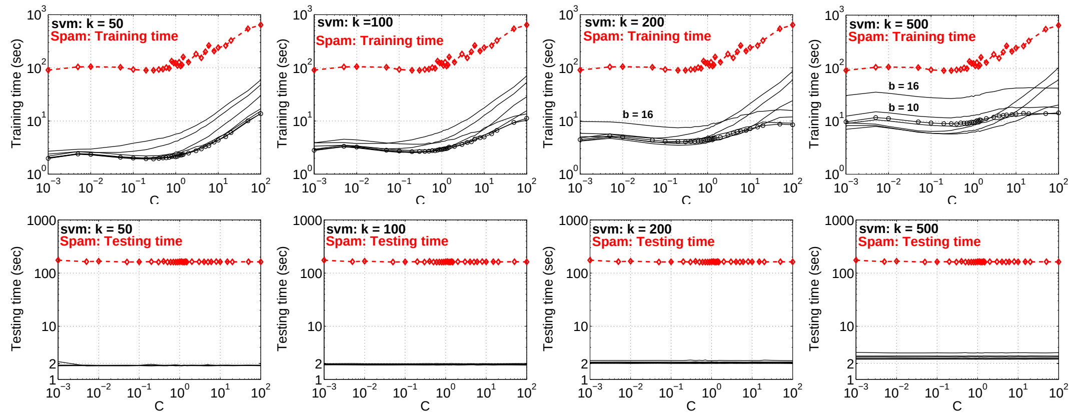

Figure 3: **SVM training time (upper panels) and testing time (bottom panels)**. The original costs are plotted using dashed (red, if color is available) curves.

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.

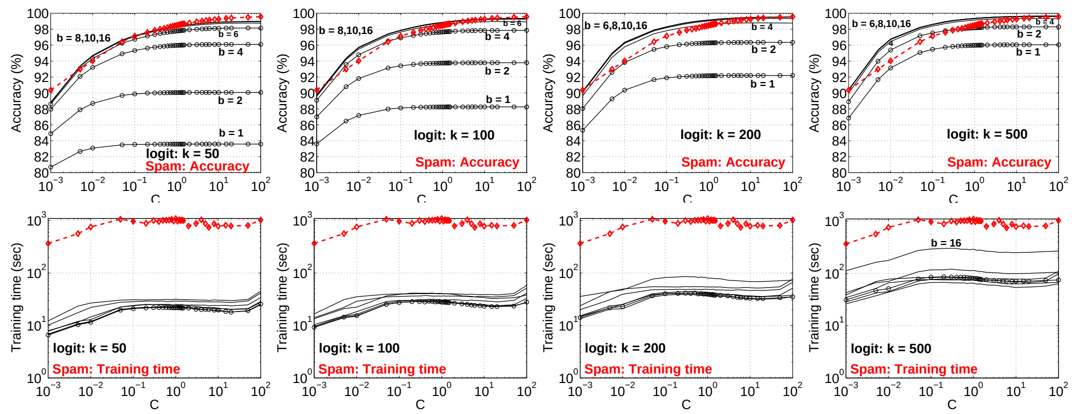

Figure 4: **Logistic regression test accuracy (upper panels) and training time (bottom panels)**.

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)* [25] to further reduce the training time when b is large.

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

We implemented the VW hashing algorithm [25] (not the VW online learning platform) 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 about the same test accuracy as VW with $k = 10^4 \sim 10^6$ (note that we only stored the non-zeros).

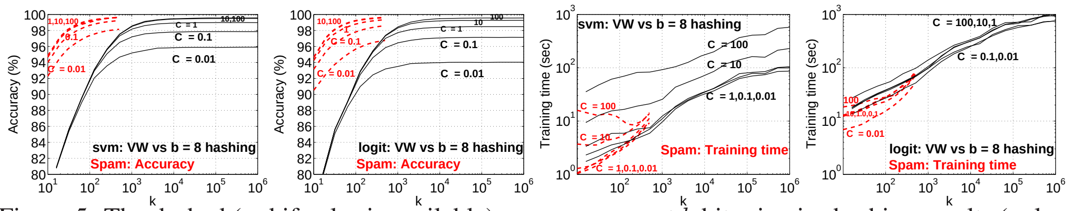

Figure 5: The dashed (red if color is available) curves represent b-bit minwise hashing results (only for $k \leq 500$) while solid curves for VW. We display results for $C = 0.01, 0.1, 1, 10, 100$.

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).

7 More Recent Work

A technical report [17] studied a number of key issues concerning the deployment of b-bit minwise hashing in the context of similarity computation and learning for large-scale industrial applications.

The first issue we studied was the effect of using practical hash functions in place of the fully random permutations assumed in the theoretical results. While this issue has been studied in theory (mostly for the derivation of *worst-case* results that do not necessarily reflect behavior in practice), this work examined the resulting effects on the accuracy of learning and similarity computation in practice. For this purpose, we conducted an extensive experimental evaluation using large-scale data sets and a number of different hash functions as well as full permutations. We first experimented with the original resemblance estimator of b-bit minwise hashing on English word vectors using 2U (2-universal) and 4U hash functions and observed that, as long as the data are not too dense, the estimates can match the performance predicted under the assumption of perfectly random permutations. Then we experimented with linear SVM and logistic regression on the *webspam* dataset as well as the (expanded) *rcv1* dataset (of about 200GB in LIBSVM format and 1 billion features). For small b (e.g., 1 or 2), the limited randomness has some noticeable (albeit quite small) impact on the test accuracies of the learning algorithms. 2U hashing and 4U hashing produced very similar accuracy results, while 4U requires substantially more computational overhead than 2U.

The second issue we explored was the fast computation of minwise hashes using graphics processors. As we were able to show, the minwise hashing algorithm itself is very well suited towards the properties of current GPU architectures, in particular their massive parallelism and SIMD instruction processing, while minimizing the impact of their constraints (most notably the limited bandwidth available for main memory data transfer). We observed that the new GPU-based implementation resulted in speed-ups of > 20 folds for the minwise hashing computation, thereby making the data loading time the main bottleneck. The observed gains are relevant for any large-scale instance of the numerous applications of (b-bit) minwise hashing.

The final issue we studied was the application of b-bit minwise hashing to online learning techniques, which are gaining momentum in the web context, as the size of training sets often exceed the available main memory capacity; we were able to show that b-bit minwise hashing provides an effective dimensionality reduction scheme, thereby allowing applications to store models that would otherwise not fit the available memory (note that the memory requirements of models are an important concern in search as the user-facing web servers typically have to host a number of different machine-learning models which are invoked for incoming queries). More importantly, the use of b-minwise hashing substantially reduced data loading time during training, thereby speeding up training times significantly (which is important, as many online learning algorithms require a large number of training epochs until they reach sufficient accuracy).

8 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 [15]. 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. Some of these issues have been studied in a very recent technical report [17].

References

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

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

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

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

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

[17] Ping Li, Anshumali Shrivastava, and Arnd Christian König. b-bit minwise hashing in practice: Large-scale batch and online learning and using gpus for fast preprocessing with simple hash functions. Technical report, 2011.

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

[19] Ping Li and Arnd Christian König. Accurate estimators for improving minwise hashing and b-bit minwise hashing. Technical report, 2011.

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

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

[22] 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.

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

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

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

[26] 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.