Hashing Algorithms for Large-Scale Applications in Search, Learning, and Compressed Sensing

Ping Li

Department of Statistics and Biostatistics

Department of Computer Science

Rutgers University

Piscataway, NJ 08854, USA

Tutorial in 2014 Summer School & Workshop on Hashing

Outline

1. **Sign Cauchy Random Projections and χ^2 Kernels**

   An interesting & surprising story about hashing for search & learning.

   Ref: *Li, Samorodnitsky, and Hopcroft, NIPS 2013*

2. **Conditional Random Sampling (CRS)**

   Our very first work on hashing, also related to one permutation hashing.

   Ref: *Li and Church, EMNLP 2005*. Ref: *Li, Church, and Hastie, NIPS 2006*

3. **One Permutation Hashing and Densified One Permutation Hashing**

   A safe replacement of the standard & popular k-permutation minwise hashing.

   Ref: *Li, Owen, and Zhang, NIPS 2012*. Ref: *Shrivastava and Li, ICML 2014*.

4. **Very Sparse Random Projections and Compressed Sensing**

   Ref: *Li, Church, and Hastie, KDD 2006*. Also ongoing research.

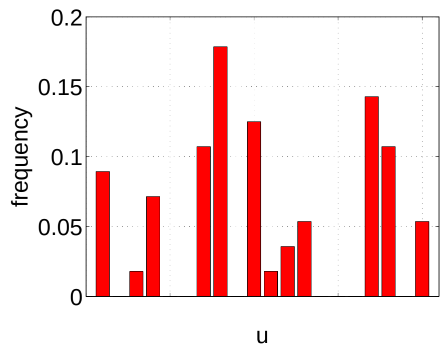
——————-

*References are mostly limited to own papers.

One Major Source of High-Dimensional Data: Histogram

Histogram-based features are very popular in practice, for example, natural language processing (NLP) and computer vision.

It can be viewed as high-dimensional vector: $\boxed{u_i \geq 0, \quad i = 1, 2, ..., D}$



The size of the space D can often be extremely large. For example, D can be the total number of words, or combinations of words (or characters, or visual words). In search industry, $D = 2^{64}$ is often used, for convenience.

An Example of Text Data Representation by n-grams

Each document (Web page) can be viewed as a set of n-grams.

For example, after parsing, a sentence "today is a nice day" becomes

- $n = 1$: $\{$"today", "is", "a", "nice", "day"$\}$
- $n = 2$: $\{$"today is", "is a", "a nice", "nice day"$\}$
- $n = 3$: $\{$"today is a", "is a nice", "a nice day"$\}$

It is common to use $n \geq 5$.

Using n-grams generates extremely high dimensional vectors, e.g., $D = (10^5)^n$. $(10^5)^5 = 10^{25} = 2^{83}$, although in current practice, it seems $D = 2^{64}$ suffices.

As a highly successful practice, n-gram representations have many variants, e.g., word n-grams, character n-grams, skip n-grams, etc.

Webspam: A Small Example of n-gram Data

Task: Classifying 350K documents into spam or non-spam (binary classification).

	Dim. (D)	Training Time	Accuracy
1-gram + linear SVM	254	20 sec	93.30%
3-gram + linear SVM	16,609,143	200 sec	99.6%
3-gram + kernel SVM	16,609,143	**About a Week**	99.6%

———————-

(Character) 1-gram: Frequencies of occurrences of single characters.

(Character) 3-gram: Frequencies of occurrences of 3-contiguous characters.

Challenges with Using Histogram-Based Features

- **High dimensionality** This may lead to large & costly (in both training and testing) statistical models and create large search space.
- **High storage cost** If the data are fully materialized, they might be way too large to store/maneuver, even for sparse data.
- **Streaming data** Histogram is a streaming model. How to compute summaries without storing/materializing the entire histograms, and how to update summary statistics without accessing the original histograms?
- **Binary vs. non-binary** While in NLP and search it is popular to use very high-dimensional and binary representations, the current mainstream practice in computer vision is to use non-binary features. In general, binary representations require a much large space (dimensionality).

A Side Note: Histogram and Data Stream

Turnstile data stream model: A data stream can be conceptually viewed as a long vector $u^{(t)}$ whose entries vary over time. At a time t, there is an incoming stream element (i_t, I_t), which updates the i_t-th entry in a linear fashion:

$$u_{i_t}^{(t)} = u_{i_t}^{(t-1)} + I_t$$

The process of histogram construction can be viewed as data streams ($I_t = 1$).

Outline

1. **Sign Cauchy Random Projections and χ^2 Kernels**

   An interesting & surprising story about hashing for search & learning.

   Ref: Li, Samorodnitsky, and Hopcroft, NIPS 2013

2. **Conditional Random Sampling (CRS)**

   Our very first work on hashing, also related to one permutation hashing.

   Ref: Li and Church, EMNLP 2005. *Ref: Li, Church, and Hastie, NIPS 2006*

3. **One Permutation Hashing and Densified One Permutation Hashing**

   A safe replacement of the standard & popular k-permutation minwise hashing.

   Ref: Li, Owen, and Zhang, NIPS 2012. *Ref: Shrivastava and Li, ICML 2014*.

4. **Very Sparse Random Projections and Compressed Sensing**

   Ref: Li, Church, and Hastie, KDD 2006. Also ongoing research.

Similarity of Histograms and Chi-Square Kernel

Given two high-dim nonnegative vectors $u, v \in \mathbb{R}^D$, the chi-square similarity is

$$\rho_{\chi^2} = \sum_{i=1}^{D} \frac{2u_iv_i}{u_i + v_i}, \qquad \sum_{i=1}^{D} u_i = \sum_{i=1}^{D} v_i = 1$$

The chi-square similarity is closely related to the chi-square distance d_{χ^2}:

$$d_{\chi^2} = \sum_{i=1}^{D} \frac{(u_i - v_i)^2}{u_i + v_i} = \sum_{i=1}^{D}(u_i + v_i) - \sum_{i=1}^{D} \frac{4u_iv_i}{u_i + v_i} = 2 - 2\rho_{\chi^2}$$

It is a “symmetric” version of the usual chi-square statistic.

Chi-square Kernels

1. χ^2-kernel: $K(u,v) = \rho_{\chi^2} = \sum_{i=1}^{D} \frac{2u_iv_i}{u_i+v_i}$

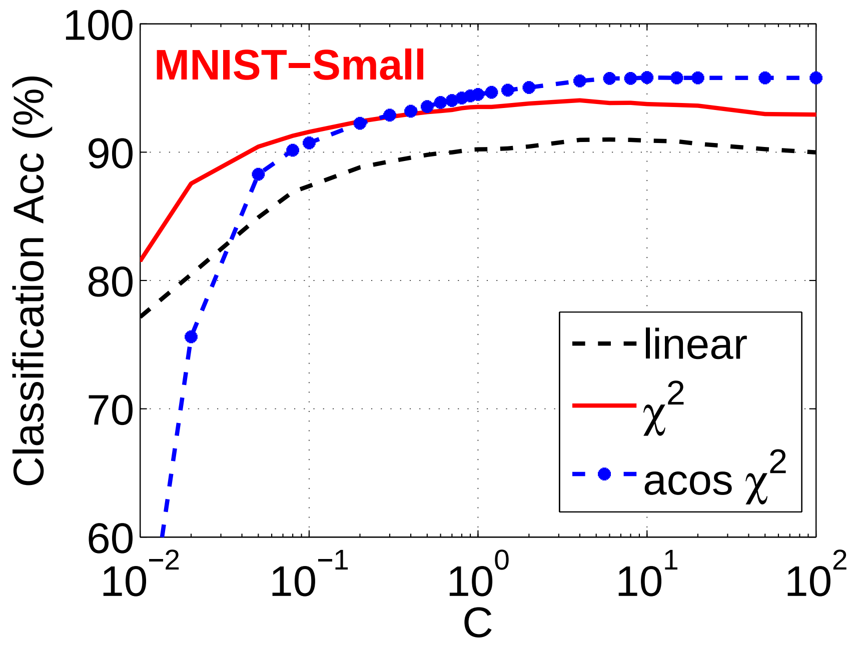
   It was proved to be positive definite in prior work.

2. acos-χ^2-kernel: $K(u,v) = 1 - \frac{1}{\pi}\cos^{-1}\rho_{\chi^2}$

   We have a short proof that it is also positive definite.

Advantage of Chi-Square Kernels: An Example

MNIST-Small: Chi-square kernels substantially improve linear kernel

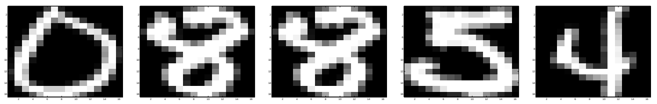


l_2-regularized kernel SVM with a regularization parameter C.

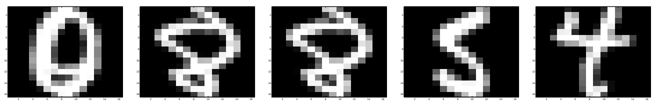
MNIST-small: original testing data and merely 1/6 of original training data

Classification on MNIST Data

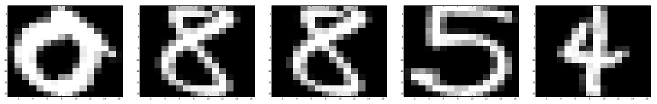
Person 1:



Person 2:



Person 3:



Review Logistic Regression and Linear SVM

For example, consider dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \{-1, 1\}$.

One can fit an L_2-regularized linear logistic regression:

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

or the L_2-regularized linear SVM:

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

where $\mathbf{C} > 0$ is the penalty (regularization) parameter.

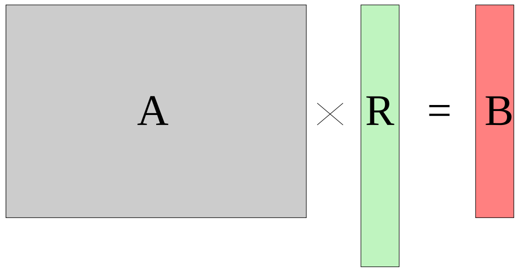
Challenges with Kernel Learning

Kernels were believed not very useful for large-scale applications:

1. Computing kernels is very expensive.
2. Computing a full kernel matrix is wasteful, because not all pairwise kernel values are used during training.
3. The kernel matrix does not fit in memory. The cost of storing the full kernel matrix in the memory is $O(n^2)$, which is not realistic for most PCs even for merely 10^5. Thus, kernel evaluations are often conducted on the fly, which means the computational cost is dominated by kernel evaluations.
4. In fact, evaluating kernels on-demand would encounter another serious (and often common) issue if the dataset itself is too big for the memory.

Sign Cauchy Random Projections

$\mathbf{A} \in \mathbb{R}^{n\times D}$: original data matrix, (e.g.,) generated from histograms.



$\mathbf{R} \in \mathbb{R}^{D\times k}$: random matrix with entries sampled from Cauchy $C(0,1)$.
$\mathbf{B} \in \mathbb{R}^{n\times k}$: resultant projected matrix, which is much smaller than $\mathbf{A}$.

Sign Cauchy Projections: only store the signs of the projected data in $\mathbf{B}$.

$u, v \in \mathbb{R}^{D}$: first two rows in $\mathbf{A}$. $x, y \in \mathbb{R}^{k}$: first two rows in $\mathbf{B}$.

$$x = u \times \mathbf{R}, \qquad y = v \times \mathbf{R}.$$

The Collision Probability

$$x = u \times \mathbf{R}, \qquad\qquad y = v \times \mathbf{R}.$$

The Collision Probability is related to the χ^2 similarity between u and v:

$$\mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right) \approx \frac{1}{\pi}\cos^{-1}\left(\rho_{\chi^2}(u,v)\right)$$

which might be a surprising finding.

—————————-

In general, the collision probability should be a monotone function of the similarity.

Applications of Sign Cauchy Projections

- **Efficient linear learning (e.g., linear SVM) for χ^2 kernel**.

  A negative sign can be coded as "01" and a positive sign as "10" (i.e., a vector of length 2). Concatenate k short vectors to form a vector of length $2k$.

- **Sub-linear time near-neighbor search in χ^2 similarity**.

  We can code a negative sign by "0" and positive sign by "1" and concatenate k such bits to form a hash table of 2^k buckets. In the query phase, one only searches for similar vectors in one bucket.

- **Other applications** requiring computing χ^2 similarity fast using small space.

Sign Cauchy Projections for Statistical Learning

Original data vectors: $u \in \mathbb{R}^D, \quad v \in \mathbb{R}^D$

Cauchy projection matrix: $\mathbf{R} \in \mathbb{R}^{D\times k}$ ($k = 4$ in this example)

Sign Cauchy random projections & expansions:

$x = u \times \mathbf{R}$:	-61.83	2.45	13.83	-1322.05
sgn(x) :	-1	$+1$	$+1$	-1
Expansion :	01	10	10	01

$y = v \times \mathbf{R}$:	-11.64	936.91	-343.43	-12.45
sgn(y) :	-1	$+1$	-1	-1
Expansion :	01	10	01	01

Output vectors: binary vectors in $2k = 8$ dimensions.

Fast Near Neighbor Search by Hash Tables

LSH: This is the standard practice in engineering (especially in search). Instead of scanning all data points to find the nearest neighbors (for an input query), we can partition the space into many bins by building hash tables.

For example, a table of $2^4 = 16$ partitions the data into 16 bins. The data point **8** is placed in bin 0000. To improve accuracy, we need to build many tables.

Index		Data Points
00	**00**	*8, 13, 251*
00	**01**	*5, 14, 19, 29*
00	**10**	*(empty)*
11	**01**	*7, 24, 156*
11	**10**	*33, 174, 3153*
11	**11**	*61, 342*

Index		Data Points
00	**00**	*2, 19, 83*
00	**01**	*17, 36, 129*
00	**10**	*4, 34, 52, 796*
11	**01**	*7, 198*
11	**10**	*56, 989*
11	**11**	*8 ,9, 156, 879*

The key is how to place the data points into bins. Sign cauchy projections provide a solution for search near neighbors in χ^2 similarity (at least approximately).

Sign Cauchy Projections for Fast Near Neighbor Search

Sign Cauchy random projections and encoding:

$$\begin{array}{rcccc}
x = u\times\mathbf{R}: & -61.83 & 2.45 & 13.83 & -1322.05\\
\text{sgn}(x): & -1 & +1 & +1 & -1\\
\text{Encoding}: & 0 & 1 & 1 & 0
\end{array}$$

$$\begin{array}{rcccc}
y = v\times\mathbf{R}: & -11.64 & 936.91 & -343.43 & -12.45\\
\text{sgn}(y): & -1 & +1 & -1 & -1\\
\text{Encoding}: & 0 & 1 & 0 & 0
\end{array}$$

Outcome: Data point x is placed in bin 0110 and y is placed in bin 0100.

The Challenging Issue

The collision probability of sign cauchy projections is not exactly a function of χ^2 similarity.

The Interesting Probability Problem

The collision probability $\mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right)$ is a not easy if we want a precise and simple answer.

$$x = u \times \mathbf{R} = \sum_{i=1}^{D} u_i R_i, \qquad y = v \times \mathbf{R} = \sum_{i=1}^{D} v_i R_i, \qquad R_i \sim C(0,1)$$

An upper bound

$$\mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right) \leq \frac{1}{\pi} \cos^{-1} \rho_1, \quad \text{where } \rho_1 = \left(\sum_{i=1}^{D} u_i^{1/2} v_i^{1/2}\right)^2$$

This upper bound is not tight.

Two Approximations

$$\mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right) \approx P_{\chi^2(1)} = \frac{1}{\pi}\cos^{-1}\left(\rho_{\chi^2}\right)$$

$$\mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right) \approx P_{\chi^2(2)} = \frac{1}{2} - \frac{2}{\pi^2}\int_0^{\frac{\pi}{2}} \tan^{-1}\left(\frac{\rho_{\chi^2}}{2-2\rho_{\chi^2}}\tan t\right) dt$$

The Approximation Is Very Accurate in Binary Data

When data are binary, we can compute the collision probability exactly.

In this case, the nonzoro coordinates of the data $(u,\ v)$ are simply

$$u_i = \frac{1}{a+c},\ \ \forall i \in I_a \cup I_c, \qquad v_i = \frac{1}{b+c},\ \ \forall i \in I_b \cup I_c$$

$$a = |I_a|,\ \ b = |I_b|,\ \ c = |I_c|$$

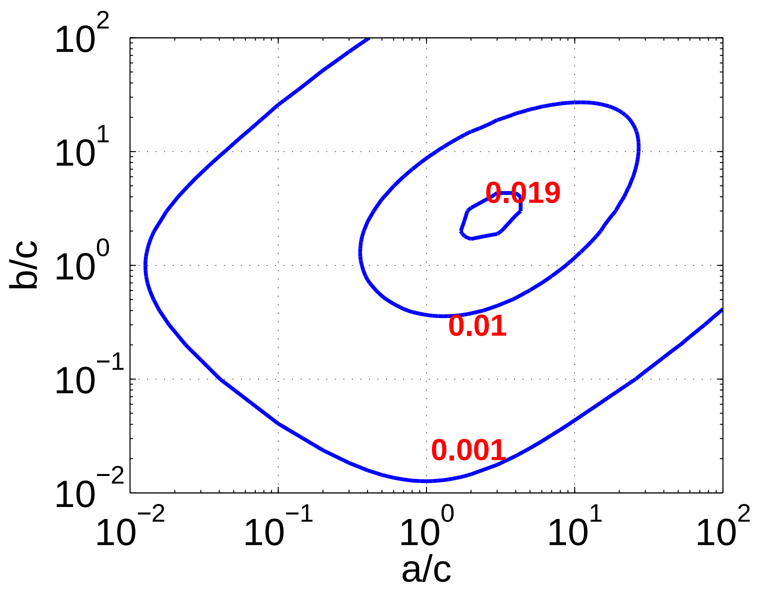
$$I_a = \{i|u_i > 0, v_i = 0\},\ \ I_b = \{i|v_i > 0, u_i = 0\},\ \ I_c = \{i|u_i > 0, v_i > 0\}$$

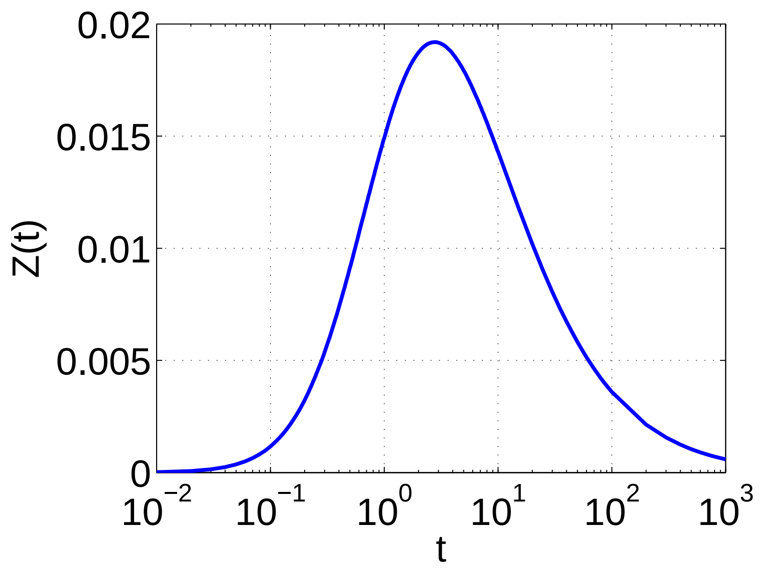
Lemma

$$Err = Z(a/c, b/c) = \mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right) - P_{\chi^2(2)}$$

$0 \leq Err \leq 0.01919$. The maximum error occurs at $a/c = b/c = 2.77935$.

Left panel: contour plot for the error $Z(a/c, b/c)$. The maximum error (which is < 0.01919) occurs along the diagonal line.



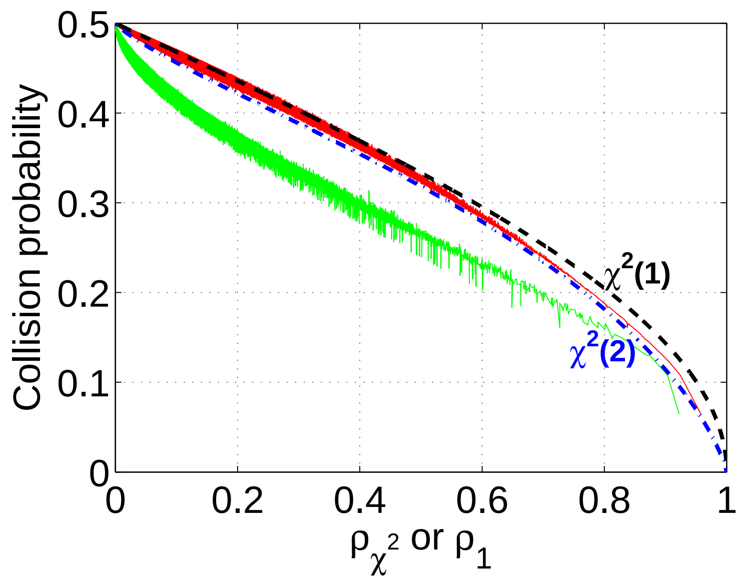


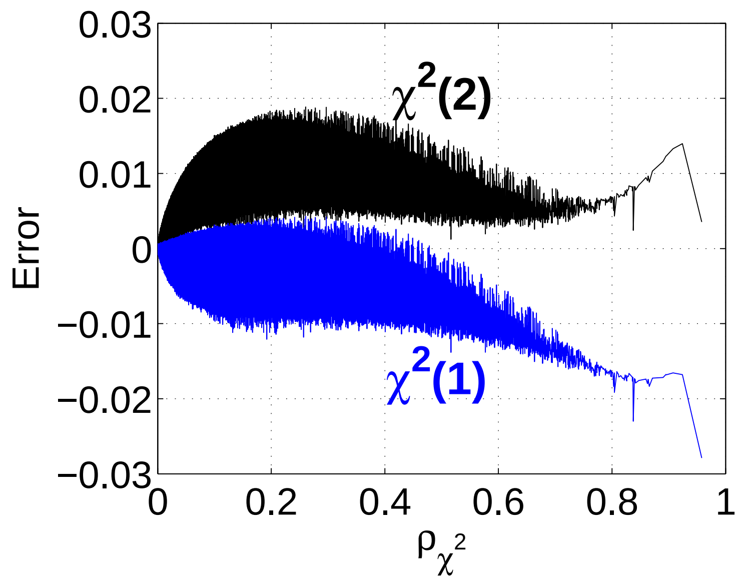
Right panel: the diagonal curve of $Z(a/c, b/c)$.

Experiment on 3.6 Million English Word Pairs

Typical sparse, non-binary data A word occurrences dataset (an example of histograms) with in total 2,702 words, i.e., 2,702 vectors and 3,649,051 word pairs. The entries of a vector are the occurrences of the word.

Both χ^2 approximations are accurate. The upper bound $\frac{1}{\pi}\cos^{-1}\rho_1$ is not tight.



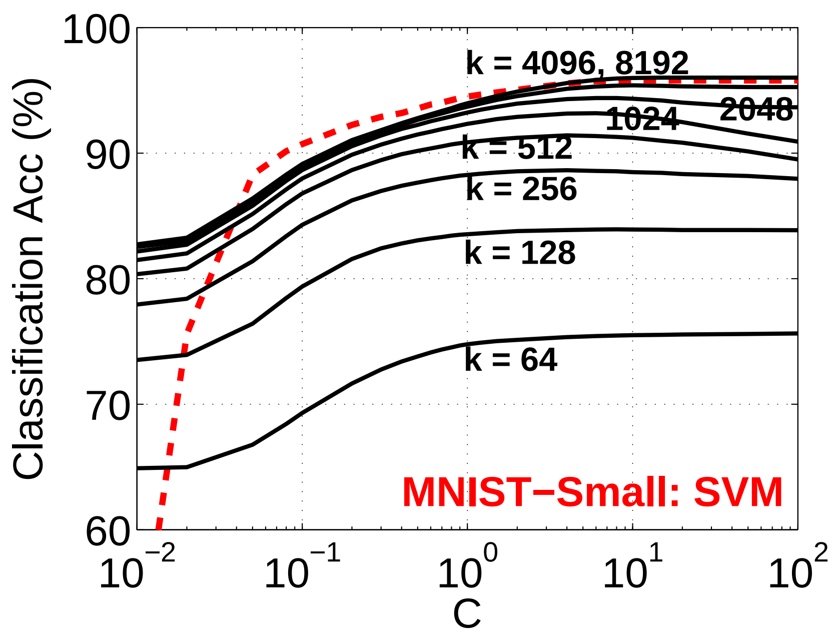


Linear SVM for Approximating Nonlinear Kernel

"negative sign" $\Longrightarrow$ "01", "positive sign" $\Longrightarrow$ "10".

k Cauchy projections $\Longrightarrow$ a binary vector of length $2k$

The inner product approximates the acos-χ^2-kernel: $1 - \frac{1}{\pi}\cos^{-1}\rho_{\chi^2}$.

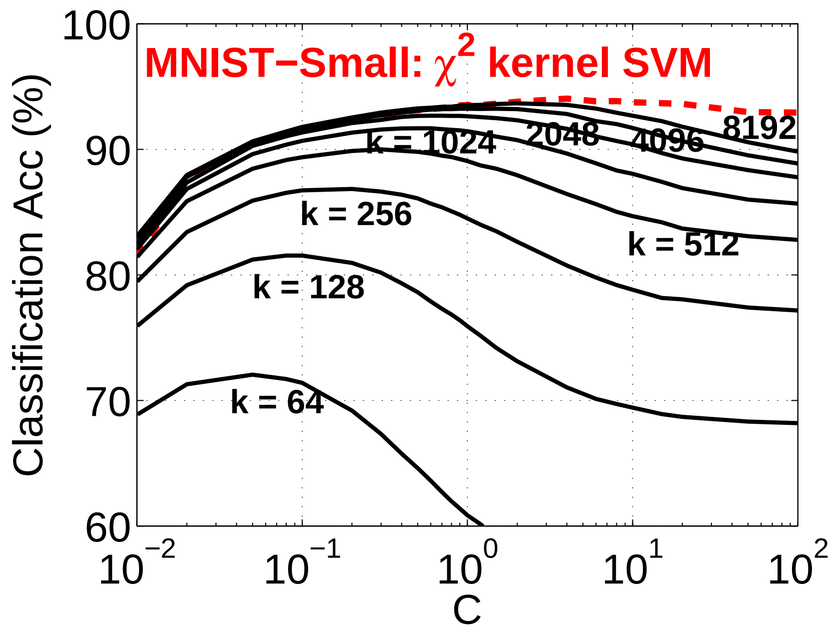


Red dashed curve: Classification accuracy of "acos-χ^2-kernel" using LIBSVM.

Solid curves: Classification accuracy of linear SVM with k sign Cauchy projections

Sign Cauchy Projections for Approximating χ^2-Kernel SVM

The inner product trick well approximates the "acos-χ^2-kernel", but not the χ^2-kernel. Nevertheless, this is a space-efficient way to estimate χ^2-kernel.



Red dashed curve: Classification accuracy of "χ^2-kernel" using LIBSVM.

Solid curves: Classification accuracy of χ^2-kernel estimated by k sign projections

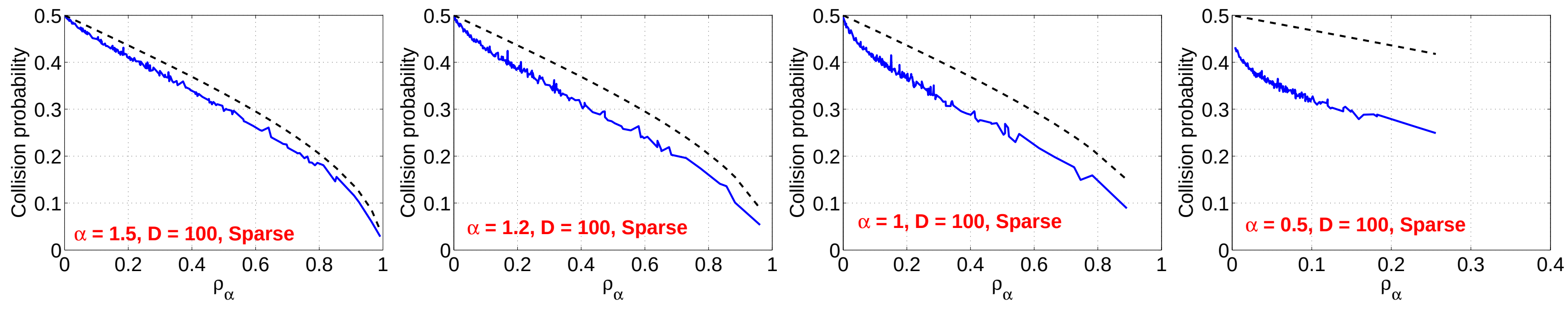
Generalization: Sign α-Stable Random Projections

The Cauchy distribution is a special instance of the α-stable (where $0 < \alpha \leq 2$) distribution family, with $\alpha = 1$. Note that Gaussian is $\alpha = 2$-stable.

Upper bound of sign α-stable collision probability

$$\mathbf{Pr}\left(\text{sign}(x) \neq \text{sign}(y)\right) \leq \frac{1}{\pi}\cos^{-1}\rho_\alpha, \quad \rho_\alpha = \left(\frac{\sum_{i=1}^D u_i^{\alpha/2}v_i^{\alpha/2}}{\sqrt{\sum_{i=1}^D u_i^\alpha \sum_{i=1}^D v_i^\alpha}}\right)^{2/\alpha}$$

For $\alpha = 2$, this bound is exact, known as "sim-hash". This bound, however, is not very accurate when α moves away from 2, as shown in simulations.



Approximate Nonlinear Kernels by Linear Kernels via Hashing

Sign Cauchy projection leads to an interesting line of research on kernels

- If chosen and used appropriately, nonlinear kernels can produce accurate results in learning tasks, often as good as sophisticated learning methods.
- For certain types of nonlinear kernels, there exist efficient hashing algorithms which can approximate nonlinear kernels by linear kernels.
- For example, **b-bit minwise hashing** for approximating **resemblance kernel**
- Interestingly, b-bit minwise hashing + random projections can approximate **CoRE kernel**.

CoRE Kernels

Consider data vectors $u, v \in \mathbb{R}^D$. One definition of CoRE Kernel is

CoRE Kernel: $K_C(u,v) = \rho \times R$

Correlation: $\rho = \rho(u,v) = \dfrac{\sum_{i=1}^D u_i v_i}{\sqrt{\sum_{i=1}^D u_i^2 \sum_{i=1}^D v_i^2}}$

Resemblance: $R = R(u,v) = \dfrac{a}{f_1 + f_2 - a}$

$f_1 = \sum_{i=1}^D 1\{u_i \neq 0\}, \quad f_2 = \sum_{i=1}^D 1\{v_i \neq 0\},$
$a = \sum_{i=1}^D 1\{u_i \neq 0\}1\{v_i \neq 0\}$

Classification with Kernels on Sparse Non-binary Data

CoRE Kernels perform very well compared to linear kernels

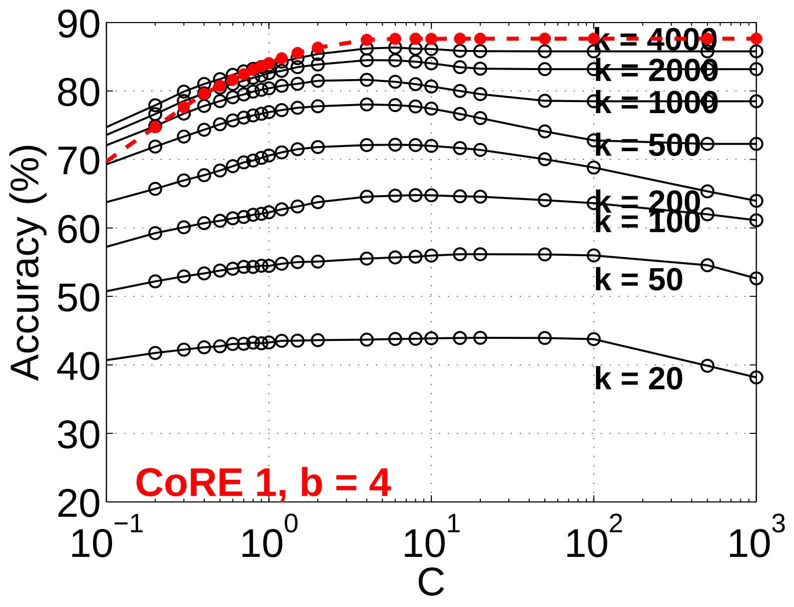
Classification accuracies (%)

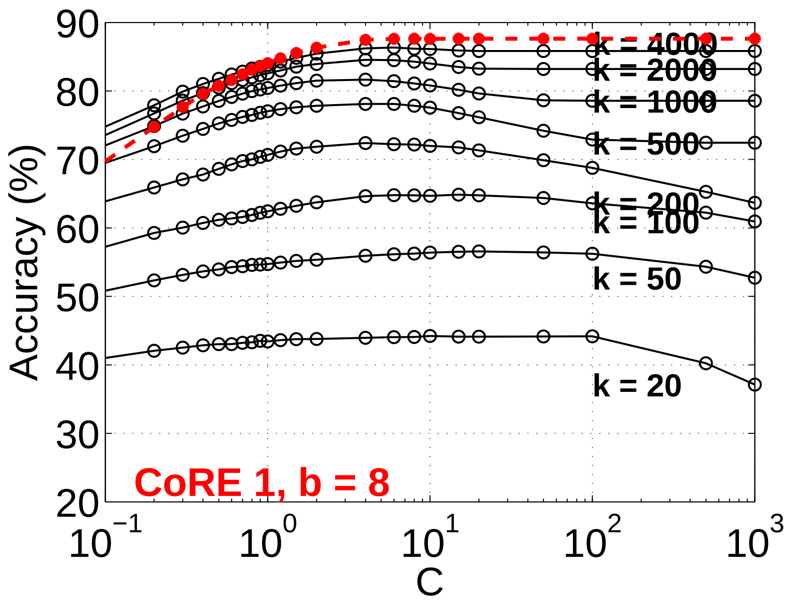
Dataset	Linear	Linear (binary)	Resemblance	CoRE
M-Basic	90.0	88.9	95.9	**97.0**
MNIST10k	90.0	88.8	95.5	**96.6**
M-Rotate	48.0	44.4	80.3	**87.6**
RCV1	96.3	95.6	96.5	**97.0**
USPS	91.8	87.4	92.5	**95.5**
Youtube	47.6	46.5	51.1	**53.1**

The results are almost as good as one can get with deep learning or boosting.

Hashing CoRE Kernels

M-Rotate: b-bit minwise hashing ($b = 4, 8$) + k random projections





Linear kernel can only achieve 48% accuracy, i..e, usual random projection methods (or variants) can only achieve at most 48%.

—————-

The plots will make more sense later in the talk.

Hint from the experiments with CoRE kernels: For sparse data, information about the locations of nonzero coordinates could be crucial.

Next, we focus on b-bit minwise hashing and one permutation hashing for search and learning with binary data.

The usual (Gaussian) random projections often do not work well for binary data.

Webspam: Binary Quantized Data

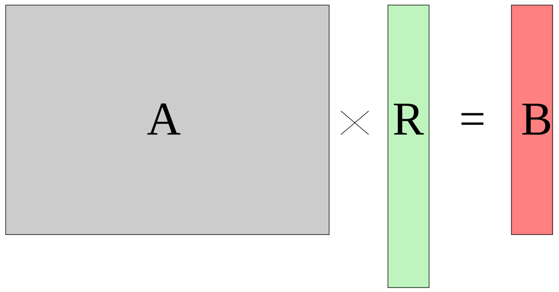
	Dim. (D)	Training Time	Accuracy
1-gram + linear SVM	254	20 sec	93.30%
Binary 1-gram + linear SVM	254	20 sec	87.78%
3-gram + linear SVM	16,609,143	200 sec	99.6%
Binary 3-gram + linear SVM	16,609,143	200 sec	99.6%
3-gram + kernel SVM	16,609,143	**About a Week**	99.6%

For this dataset, mainly presence/absence (i.e., binary) information matters.

It turns out the popular (normal) random projection method does not work well on this dataset.

Normal Random Projections

Random Projections: Replace original data matrix $\mathbf{A}$ by $\mathbf{B} = \mathbf{A} \times \mathbf{R}$



$\mathbf{R} \in \mathbb{R}^{D\times k}$: a random matrix, with i.i.d. entries sampled from $N(0,1)$.
$\mathbf{B} \in \mathbb{R}^{n\times k}$: projected matrix, also random.

$\mathbf{B}$ approximately preserves the Euclidean distance and inner products between any two rows of $\mathbf{A}$. In particular, $E\left(\mathbf{B}\mathbf{B}^\mathsf{T}\right) = \mathbf{A}E(\mathbf{R}\mathbf{R}^\mathsf{T})\mathbf{A}^\mathsf{T} = \mathbf{A}\mathbf{A}^\mathsf{T}$.

Therefore, we can simply feed $\mathbf{B}$ into (e.g.,) SVM or logistic regression solvers.

Very Sparse Random Projections

The projection matrix: $\mathbf{R} = \{r_{ij}\} \in \mathbb{R}^{D\times k}$. Instead of sampling from normals, we sample from a sparse distribution parameterized by $s \geq 1$:

$$r_{ij} = \begin{cases} g_{ij} & \text{with prob. } \frac{1}{s} \\ 0 & \text{with prob. } 1 - \frac{1}{s} \end{cases}$$

g_{ij} follows a chosen distribution, for example, Gaussian, $\{-1, 1\}$, stable, etc.

If $s = 100$, then on average, 99% of the entries are zero.

If $s = 1000$, then on average, 99.9% of the entries are zero.

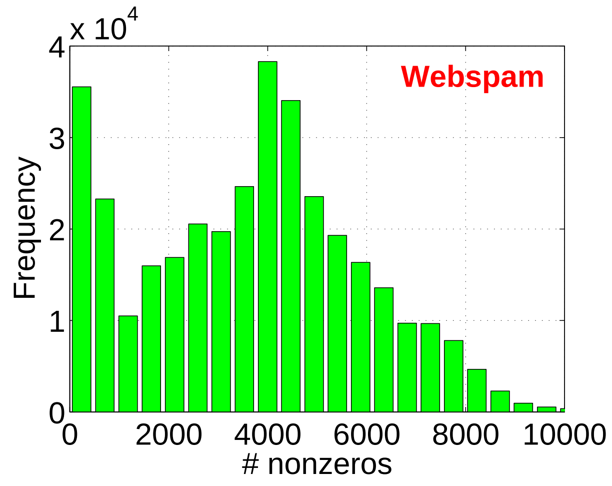
——————-

Ref: Li, Hastie, Church, Very Sparse Random Projections, KDD'06.

Ref: Li, Very Sparse Stable Random Projections, KDD'07.

A Running Example Using (Small) Webspam Data

Datset: 350K text samples, 16 million dimensions, about 4000 nonzeros on average, 24GB disk space.

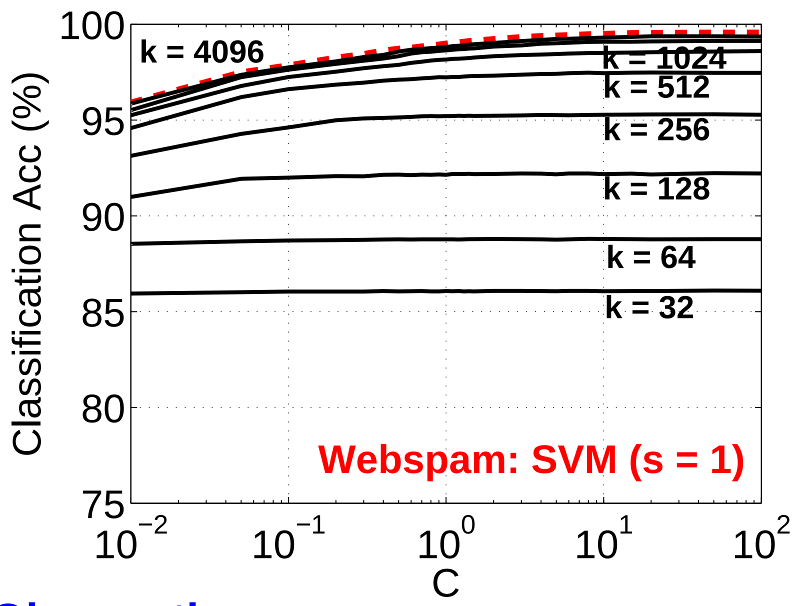


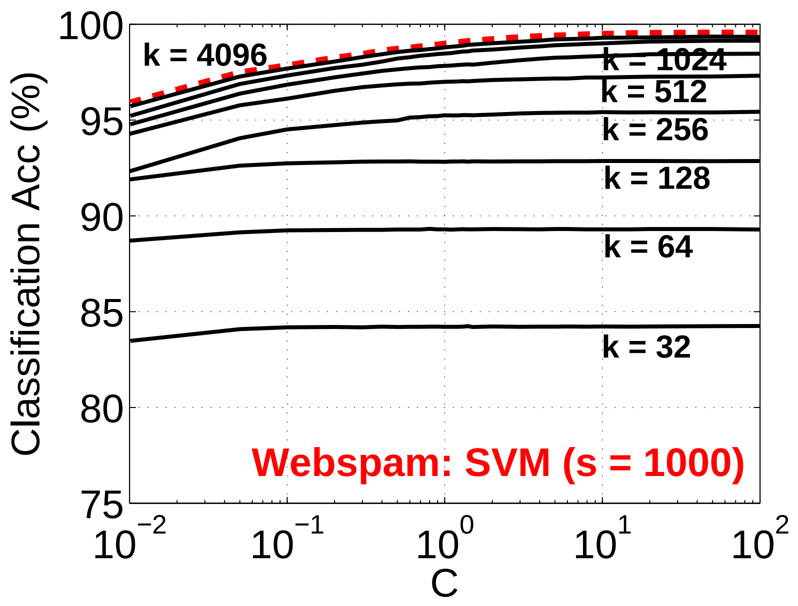
Task: Binary classification for spam vs. non-spam.

Data were generated using character 3-grams, i.e., every 3-contiguous characters.

Very Sparse Projections + Linear SVM on Webspam Data

Red dashed curves: results based on the original data





Observations:

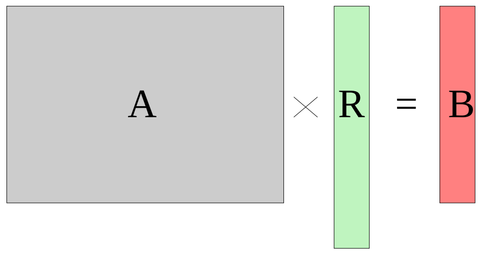
- We need a large number of projections (e.g.,$k > 4096$) for high accuracy.
- The sparsity parameter s matters little, i.e., matrix can be very sparse.

For more experiments on classification/clustering/regression with very sparse random projections, see pages 7 - 14 in a talk at MMDS 2012: `http://web.stanford.edu/group/mmds/slides2012/s-pli.pdf`

Disadvantages of Random Projections (and Variants)

Inaccurate, especially on binary data.

Variance Analysis for Inner Product Estimates



First two rows in **A**: $u_1, u_2 \in \mathbb{R}^D$ (D is very large):

$$u_1 = \{u_{1,1},\ u_{1,2},\ ...,u_{1,i},\ ...,u_{1,D}\}$$

$$u_2 = \{u_{2,1},\ u_{2,2},\ ...,u_{2,i},\ ...,u_{2,D}\}$$

First two rows in **B**: $v_1, v_2 \in \mathbb{R}^k$ (k is small):

$$v_1 = \{v_{1,1},\ v_{1,2},\ ...,v_{1,j},\ ...,v_{1,k}\}$$

$$v_2 = \{v_{2,1},\ v_{2,2},\ ...,v_{2,j},\ ...,v_{2,k}\}$$

$E\left(v_1^{\mathsf{T}} v_2\right) = u_1^{\mathsf{T}} u_2 = a$

$$\hat{a} = \frac{1}{k}\sum_{j=1}^{k} v_{1,j}v_{2,j}, \quad \text{(which is also an inner product)}$$

$$E(\hat{a}) = a$$

$$Var(\hat{a}) = \frac{1}{k}\left(m_1m_2 + a^2\right)$$

$$m_1 = \sum_{i=1}^{D}|u_{1,i}|^2, \quad m_2 = \sum_{i=1}^{D}|u_{2,i}|^2$$

———

The variance is dominated by marginal l_2 norms m_1m_2.

For real-world datasets, most pairs are often close to be orthogonal ($a \approx 0$).

——————–

Ref: Li, Hastie, and Church, Very Sparse Random Projections, KDD'06.

Ref: Li, Shrivastava, Moore, König, Hashing Algorithms for Large-Scale Learning, NIPS'11

Minwise Hashing

- In information retrieval and databases, efficiently computing similarities is often crucial and challenging, for example, duplicate detection of web pages.
- The method of minwise hashing is still the standard algorithm for estimating set similarity in industry, since the 1997 seminal work by Broder et. al.
- Minwise hashing has been used for numerous applications, for example:

  content matching for online advertising, detection of large-scale redundancy in enterprise file systems, syntactic similarity algorithms for enterprise information management, compressing social networks, advertising diversification, community extraction and classification in the Web graph, graph sampling, wireless sensor networks, Web spam, Web graph compression, text reuse in the Web, and many more.

Binary Data Vectors and Sets

A binary (0/1) vector in D-dim can be viewed a set $S \subseteq \Omega = \{0, 1, ..., D-1\}$.

Example: $S_1, S_2, S_3 \subseteq \Omega = \{0, 1, ..., 15\}$ (i.e., $D = 16$).

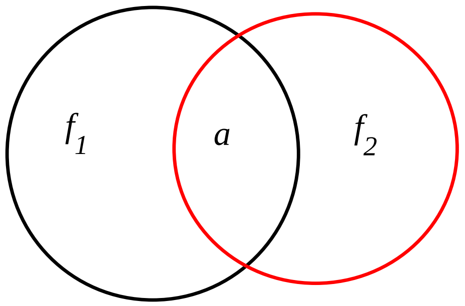
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1:	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
S_2:	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
S_3:	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0

$S_1 = \{1, 4, 5, 8\}, \quad S_2 = \{8, 10, 12, 14\}, \quad S_3 = \{3, 6, 7, 14\}$

Notation

A binary (0/1) vector $\Longleftrightarrow$ a set (locations of nonzeros).

Consider two sets $S_1, S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\}$ (e.g., $D = 2^{64}$)



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

The **resemblance** R is a popular measure of set similarity

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

(Is it more rational than $\frac{a}{\sqrt{f_1 f_2}}$?)

Minwise Hashing: Standard Algorithm in the Context of Search

The standard practice in the search industry:

Suppose a random permutation π is performed on Ω, i.e.,

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

An elementary probability argument shows that

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

An Example

$D = 5.\ S_1 = \{0,3,4\},\ S_2 = \{1,2,3\},\ R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{1}{5}$.

One realization of the permutation π can be

$$\begin{aligned} 0 &\Longrightarrow 3 \\ 1 &\Longrightarrow 2 \\ 2 &\Longrightarrow 0 \\ 3 &\Longrightarrow 4 \\ 4 &\Longrightarrow 1 \end{aligned}$$

$$\pi(S_1) = \{3,4,1\} = \{1,3,4\}, \qquad \pi(S_2) = \{2,0,4\} = \{0,2,4\}$$

In this example, $\min(\pi(S_1)) \neq \min(\pi(S_2))$.

Minwise Hashing in 0/1 Data Matrix

Original Data Matrix

$$\begin{array}{r|cccccccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
S_1: & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
S_3: & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

Permuted Data Matrix

$$\begin{array}{r|cccccccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
\pi(S_1): & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(S_2): & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(S_3): & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}$$

$$\min(\pi(S_1)) = 2, \quad \min(\pi(S_2)) = 0, \quad \min(\pi(S_3)) = 0$$

An Example with $k = 3$ Permutations

Input: sets S_1, S_2, ...,

Hashed values for S_1 :	113	264	1091
Hashed values for S_2 :	2049	103	1091
Hashed values for S_3 : ...			

....

Minwise Hashing Estimator

After k permutations, π_1, π_2, ..., π_k, one can estimate R without bias:

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

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

Issues with Minwise Hashing and Our Solutions

1. **Expensive storage (and computation)**: In the standard practice, each hashed value was stored using 64 bits.

   Our solution: b-bit minwise hashing by using only the lowest b bits.

2. **How to do linear kernel learning**:

   Our solution: We show that b-bit minwise hashing results in positive definite (PD) linear kernel matrix. The data dimensionality is reduced from 2^{64} to 2^b.

3. **Expensive and energy-consuming (pre)processing for k permutations**:

   Our solution: One permutation hashing, which is even more accurate.

Integrating b-Bit Minwise Hashing for (Linear) Learning

Very simple:

1. Apply k independent random permutations on each (binary) feature vector $\mathbf{x}_i$ and store the lowest b bits of each hashed value. The storage costs nbk bits.

2. At run-time, expand a hashed data point into a $2^b \times k$-length vector, i.e. concatenate k 2^b-length vectors. The new feature vector has exactly k 1's.

An Example with $k = 3$ Permutations

Input: sets S_1, S_2, ...,

Hashed values for S_1 :	113	264	1091
Hashed values for S_2 :	2049	103	1091
Hashed values for S_3 : ...			

....

An Example with $k = 3$ Permutations and $b = 2$ Bits

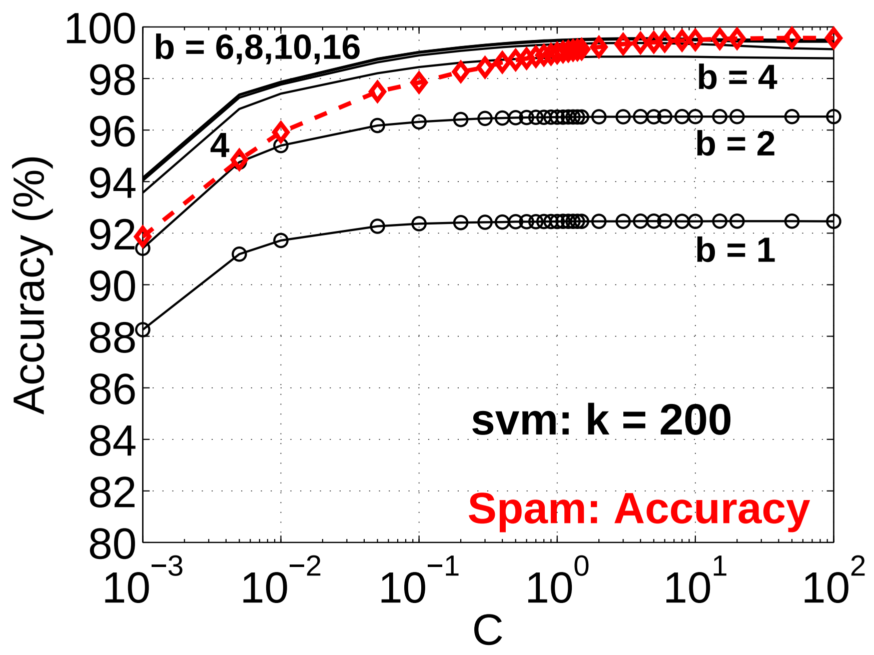
For set (vector) S_1: (Original high-dimensional binary feature vector)

Hashed values :	113	264	1091
Binary :	1110001	100001000	10001000011
Lowest $b = 2$ bits :	01	00	11
Decimal values :	1	0	3
Expansions (2^b) :	0100	1000	0001

New binary feature vector : $[0, 1, 0, 0, \quad 1, 0, 0, 0, \quad 0, 0, 0, 1] \times \dfrac{1}{\sqrt{3}}$

Same procedures on sets S_2, S_3, ...

Experiments on Webspam Data: Testing Accuracy



- Dashed: using the original data (**24GB** disk space).
- Solid: b-bit hashing. Using $b = 8$ and $k = 200$ achieves about the same test accuracies as using the original data. Space: **70MB** (350000×200)

A Good Practice

1. **Engineering:**

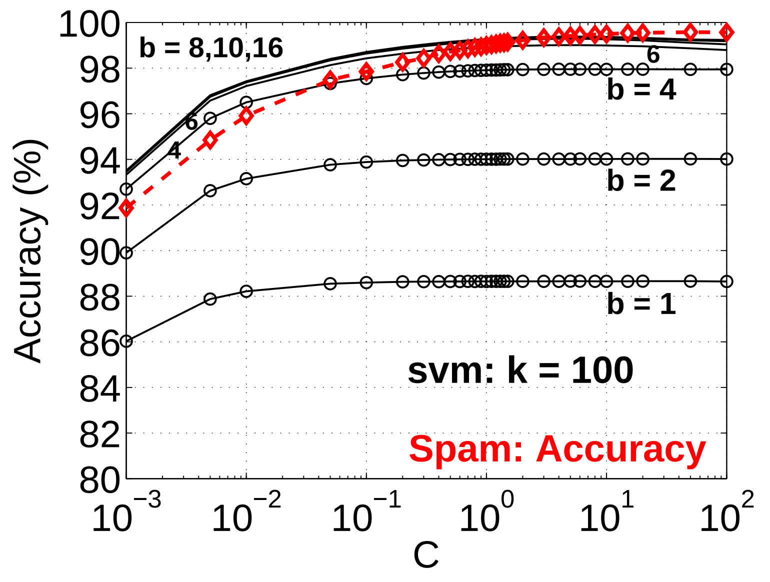
   Webspam, unigram, 254 dim $\Longrightarrow$3-gram, 16M dim, 4000 nonzeros per doc

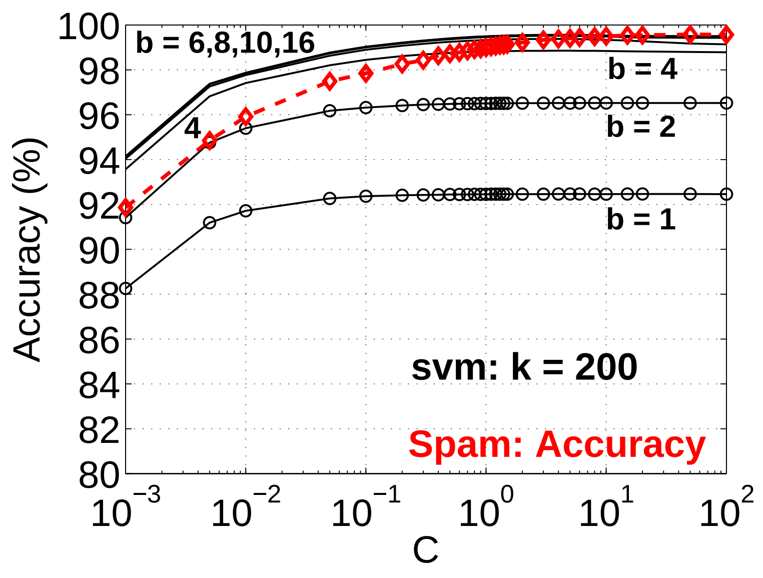
   Accuracy: $93.3\% \Longrightarrow 99.6\%$

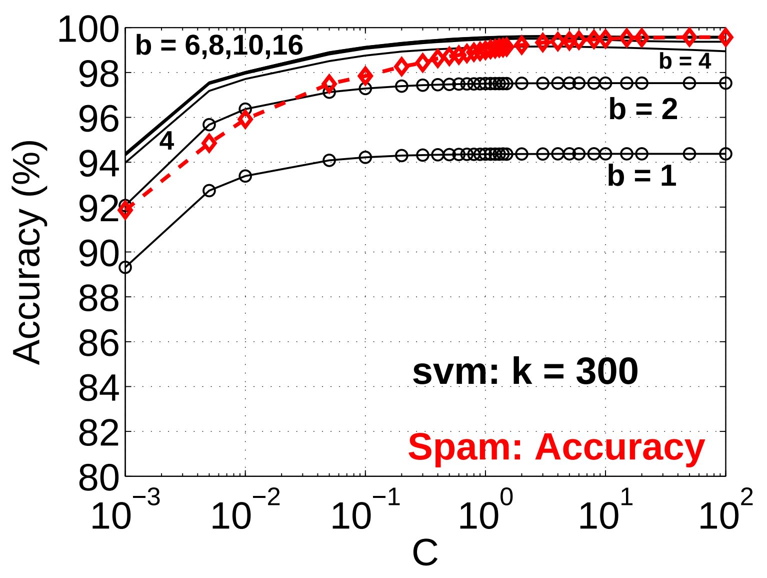
2. **Probability/Statistics:**

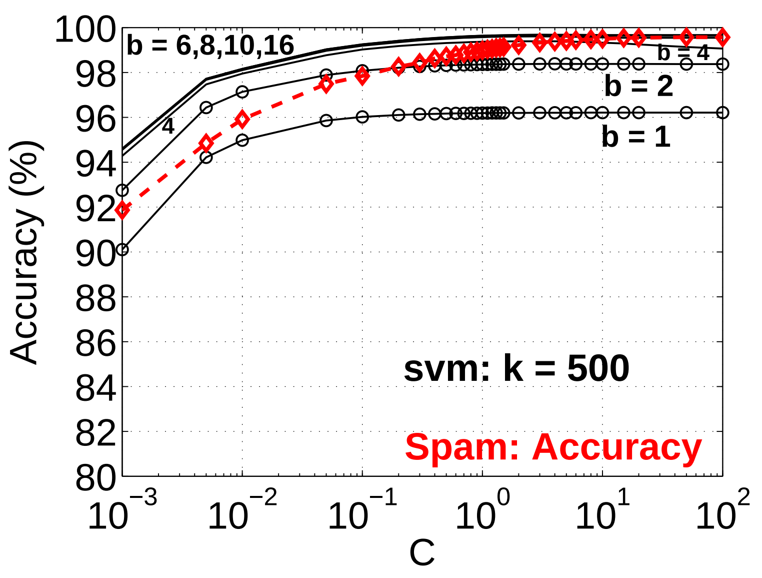
   16M dim, 4000 nonzeros $\Longrightarrow k = 200$ nonzeros, $2^b \times k = 51200$ dim

   Accuracy: $99.6\% \Longrightarrow 99.6\%$

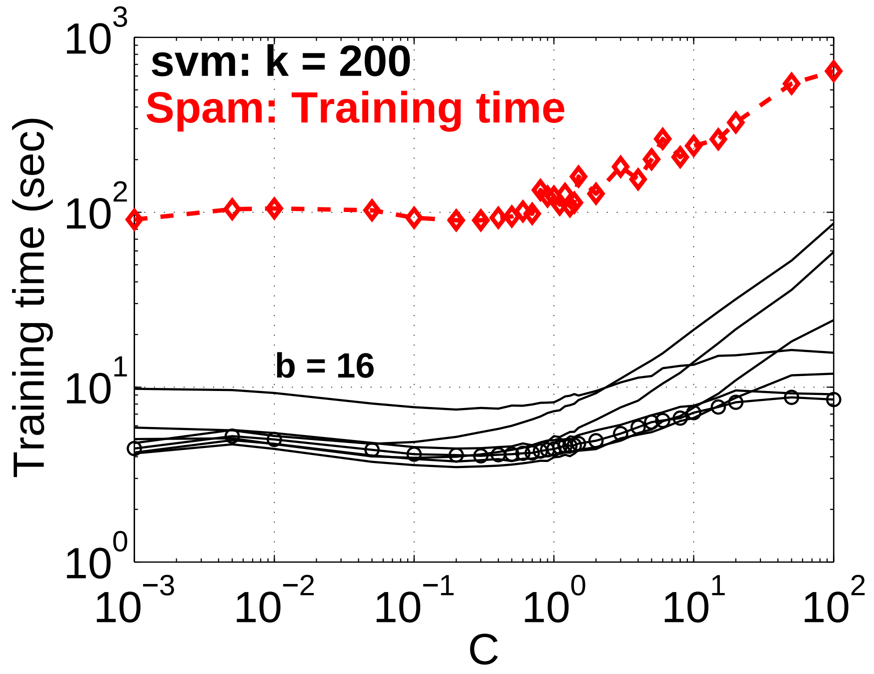

b = 8,10,16
6
4
b = 4
b = 2
b = 1
svm: k = 100
Spam: Accuracy
Accuracy (%)
C


b = 6,8,10,16
4
b = 4
b = 2
b = 1
svm: k = 200
Spam: Accuracy
Accuracy (%)
C


b = 6,8,10,16
b = 4
4
b = 2
b = 1
svm: k = 300
Spam: Accuracy
Accuracy (%)
C

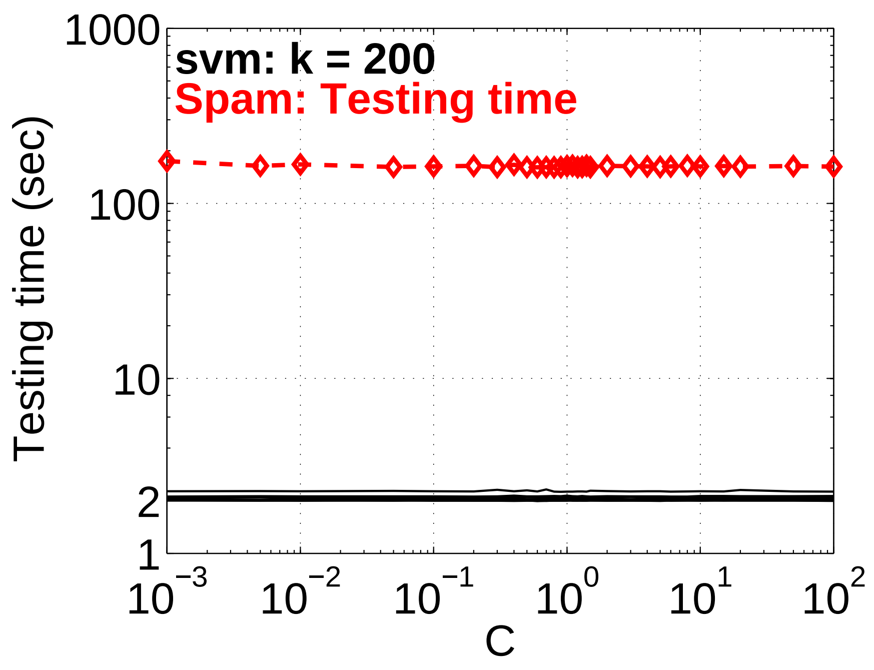

b = 6,8,10,16
b = 4
4
b = 2
b = 1
svm: k = 500
Spam: Accuracy
Accuracy (%)
C

Training Time



- They did not include data loading time (which is small for b-bit hashing)
- The original training time is about 200 seconds.
- b-bit minwise hashing needs about $3 \sim 7$ seconds (3 seconds when $b = 8$).

Testing Time



However, here we assume the test data have already been processed.

The Problem of Expensive Preprocessing

200 or 500 permutations (or even more for LSH) on the entire data can be very expensive. A serious issue when the new testing data have not been processed.

Two solutions:

1. **Conditional Random Sampling (CRS)**:

   Ref: Li and Church, Using Sketches to Estimate Associates, EMNLP 2005

   Ref: Li, Church, Hastie Conditional Random Sampling ..., NIPS 2006

2. **One Permutation Hashing**:

   Ref: Li, Owen, Zhang, One Permutation Hashing, NIPS 2012

   Ref: Shrivastava and Li, Densified One Permutation Hashing ... , ICML 2014

Intuition: Minwise Hashing Ought to Be Wasteful

Original Data Matrix

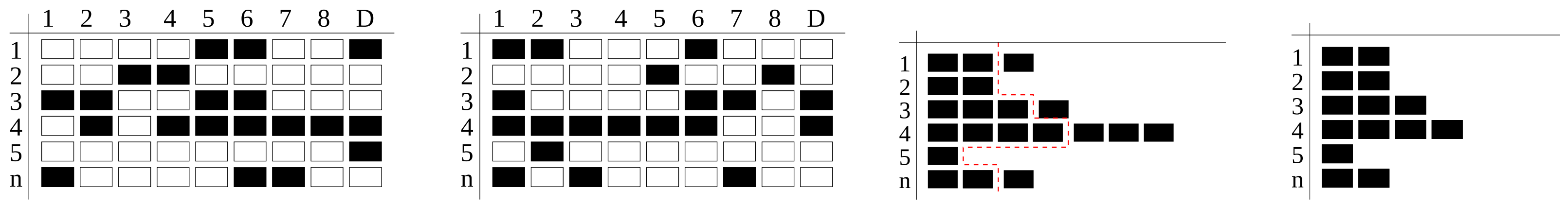
$$\begin{array}{c|cccccccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
S_1: & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
S_3: & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

Permuted Data Matrix

$$\begin{array}{c|cccccccccccccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
\pi(S_1): & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(S_2): & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(S_3): & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}$$

Only store the minimums and repeat the process k (e.g., 500) times.

Conditional Random Sampling (CRS)



(a) Original (b) Column-Permuted (c) Postings (d) Sketches

An example of CRS with two rows. Suppose the data are already permuted.

We can obtain a random sample of size 10 by taking the first 10 columns.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
u_1	0	1	0	2	0	1	0	0	1	2	1	0	1	0	2
u_2	1	3	0	0	1	2	0	1	0	0	3	0	0	2	1

K_1 :	2 (1)	4 (2)	6 (1)	9 (1)	10 (2)	
K_2:	1 (1)	2 (3)	5 (1)	6 (2)	8 (1)	11 (3)

With CRS, we use a sketch of size 6 for each row. If we do not use (11,3) of the second row, we would obtain exactly the same sample as if we had directly sampled the first 10 columns from the permuted data.

Once we have a random sample, we can estimate any summary statistics.

Comparing CRS with Broder's Original MinHash

Assuming $|K_1| = |K_2| = k$, Broder (and colleagues) originally used only one permutation and the following estimator of the resemblance by "throwing away" half of the samples (using k instead of $2k$ samples):

$$\frac{|\text{MIN}_k(K_1 \cup K_2) \cap K_1 \cap K_2|}{k}$$

The advantage of CRS: (Li and Church, 2005, Li, Church, and Hastie, 2006)

- CRS "throws away" much fewer samples and is thus (significantly) more accurate. See the analysis and experiments in the paper.
- CRS can trivially handle non-binary data and can conveniently adjust the numbers of samples according to data sparsity.

The Disadvantage of CRS

CRS (as well as Broder's original sketch) does not generate samples which are "aligned". In other words, the estimators are not linear and hence they can not be (rigorously) used for linear learning and LSH.

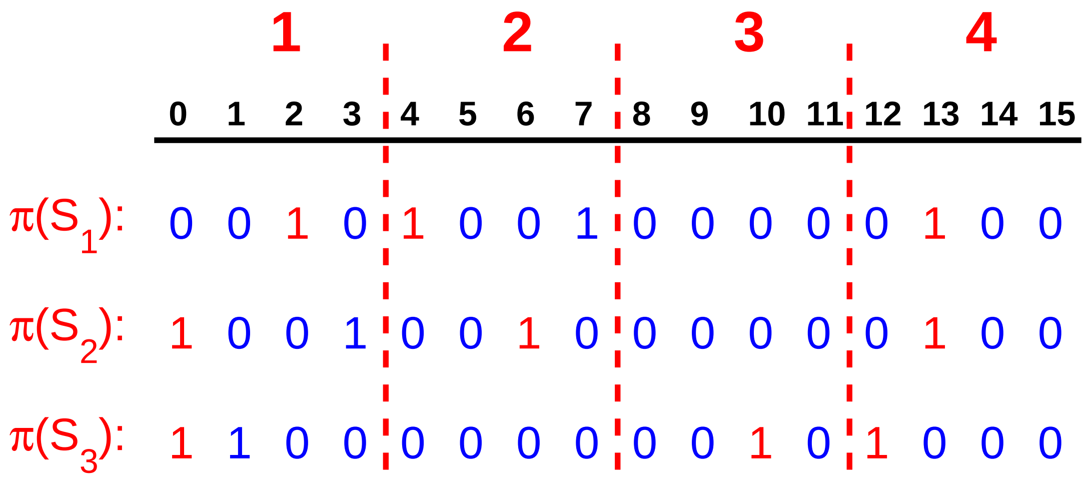
This is why we developed "one permutation hashing":

Ref: P. Li, A. Owen, C-H Zhang, One Permutation Hashing, NIPS'12

One Permutation Hashing

$S_1, S_2, S_3 \subseteq \Omega = \{0, 1, ..., 15\}$ (i.e., $D = 16$). The figure presents the permuted sets as three binary (0/1) vectors:

$$\pi(S_1) = \{2, 4, 7, 13\}, \quad \pi(S_2) = \{0, 6, 13\}, \quad \pi(S_3) = \{0, 1, 10, 12\}$$


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

One permutation hashing: divide the space Ω evenly into $k = 4$ bins and select the smallest nonzero in each bin.

Re-indexing the samples: only need to compare elements with the same bin #.

	1				2				3				4			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(S_1)$:	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
$\pi(S_2)$:	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$\pi(S_3)$:	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0

Sample from $\pi(S_1)$: $[2, 4, *, 13]$ Sample from $\pi(S_2)$: $[0, 6, *, 13]$

$*$ denotes empty bins, which occur rarely in reasonable scenarios.

Smallest representations:

$$[2, 4, *, 13] \Longrightarrow [2 - 4 \times 0, 4 - 4 \times 1, *, 13 - 4 \times 3] = [2, 0, *, 1]$$

$$[0, 6, *, 13] \Longrightarrow [0 - 4 \times 0, 6 - 4 \times 1, *, 13 - 4 \times 3] = [0, 2, *, 1]$$

Two major tasks:

- Theoretical analysis and estimators
- Practical strategies to deal with empty bins, should they occur. There are different strategies for different applications (linear or search).

Two Definitions

Recall: the space is divided evenly into k bins

# Jointly empty bins:
$$N_{emp} = \sum_{j=1}^{k} I_{emp,j}$$

# Matched bins:
$$N_{mat} = \sum_{j=1}^{k} I_{mat,j}$$

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

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

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

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

Results for the Number of Jointly Empty Bins N_{emp}

Notation:

$$f_1 = |S_1|, \quad f_2 = |S_2|, \quad a = |S_1 \cap S_2|, \quad f = |S_1 \cup S_2| = f_1 + f_2 - a.$$

Expectation:

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

Variance:

$$\frac{Var\left(N_{emp}\right)}{k^2} = \frac{1}{k}\left(\frac{E(N_{emp})}{k}\right)\left(1 - \frac{E(N_{emp})}{k}\right)$$

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

———————-

Perhaps too complicated!

Approximation in sparse data, i.e., $f = |S_1 \cup S_2| = f_1 + f_2 - a \ll D$.

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

Variance:
$$\frac{Var\left(N_{emp}\right)}{k^2} = \frac{1}{k}\left(1-\frac{1}{k}\right)^f\left(1-\left(1-\frac{1}{k}\right)^f\right)$$
$$-\left(1-\frac{1}{k}\right)^{f+1}\left(\left(1-\frac{1}{k}\right)^f - \left(1-\frac{1}{k-1}\right)^f\right) + O\left(\frac{f^2}{kD}\right)$$

Implication: $\frac{E(N_{emp})}{k} \approx \left(1-\frac{1}{k}\right)^f \approx e^{-f/k}$.

$f/k = 5 \Longrightarrow \frac{E(N_{emp})}{k} \approx 0.0067$ (negligible).

$f/k = 1 \Longrightarrow \frac{E(N_{emp})}{k} \approx 0.3679$ (noticeable).

Probability distribution function of N_{emp}:

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

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

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

which is otherwise not obvious at all how to prove.

Results for the Number of Matched Bins N_{mat}

Expectation:

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

Variance:

$$\frac{Var(N_{mat})}{k^2} = \frac{1}{k}\left(\frac{E(N_{mat})}{k}\right)\left(1-\frac{E(N_{mat})}{k}\right)$$

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

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

Covariance of N_{mat} and N_{emp}

Intuitively, N_{mat} and N_{emp} should be negatively correlated. Indeed

$$\frac{Cov\left(N_{mat},\ N_{emp}\right)}{k^2}$$

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

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

and

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

An Unbiased Estimator

Estimator : $$\hat{R}_{mat} = \frac{N_{mat}}{k - N_{emp}}$$

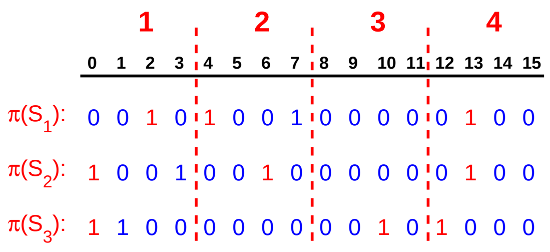
Expectation : $$E\left(\hat{R}_{mat}\right) = R$$ (slightly surprising)

Variance: $$Var\left(\hat{R}_{mat}\right)$$ (Recall $f = |S_1 \cup S_2|$)

$$= \frac{R(1-R)}{k}\left(E\left(\frac{1}{1-N_{emp}/k}\right)\left(1+\frac{1}{f-1}\right) - \frac{1}{f-1}\right)$$

$$< \frac{R(1-R)}{k}$$ (Variance of original minwise hashing)

Summary of Advantages of One Permutation Hashing


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

- Computationally much more efficient and energy-efficient.
- Parallelization-based solution requires additional hardware & implementation.
- Testing unprocessed data is much faster, crucial for user-facing applications.
- Implementation is easier, from the perspective of random number generation, e.g., storing a permutation vector of length $D = 10^9$ (4GB) is not a big deal.
- One permutation hashing is a much better matrix sparsification scheme.

One Permutation Hashing for Linear Learning

Almost the same as k-permutation hashing, but we must deal with empty bins *.

Zero coding: Encode empty bins as (e.g.,) 00000000 in the expanded space.

This simple strategy often works well for linear learning (but not for LSH).

Zero Coding Example

One permutation hashing with $k = 4$ and $b = 2$

For set (vector) S_1:

Original hashed values $(k = 4)$: $12013 \quad 25964 \quad 20191 \quad *$

Original binary representations :

$010111011101101 \quad 110010101101100 \quad 100111011011111 \quad *$

Lowest $b = 2$ binary digits : $01 \quad 00 \quad 11 \quad *$

Corresponding decimal values : $1 \quad 0 \quad 3 \quad *$

Expanded $2^b = 4$ binary digits : $0010 \quad 0001 \quad 1000 \quad 0000$

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

Same procedures on sets S_2, S_3, ...

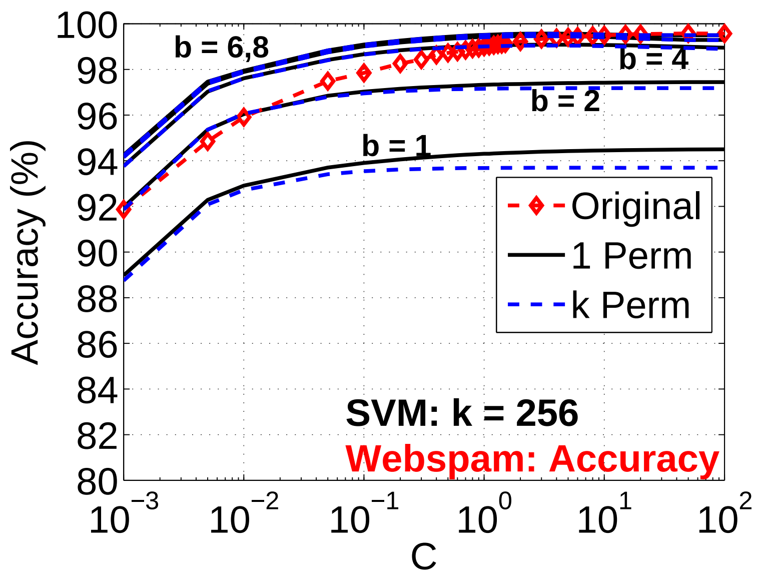
New feature vector : $[0, 0, 1, 0,\ \ 0, 0, 0, 1,\ \ 1, 0, 0, 0,\ \ 0, 0, 0, 0] \times \frac{1}{\sqrt{4-1}}$

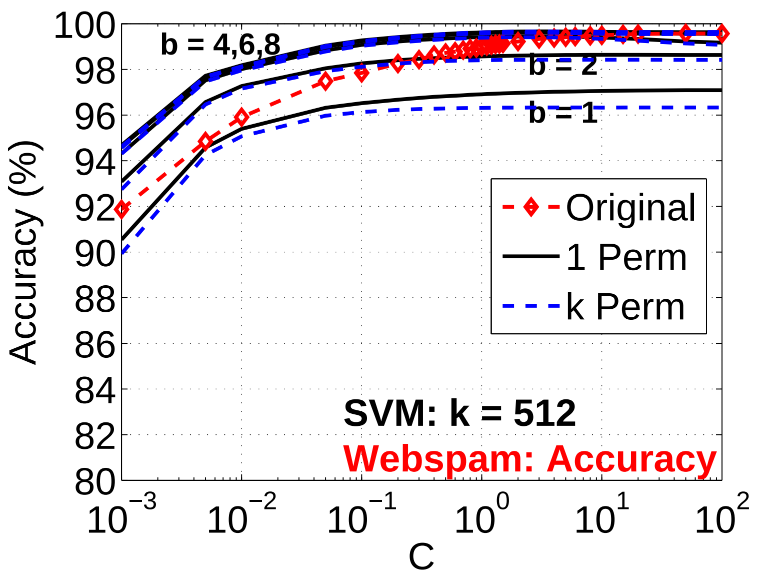
- New feature vector naturally has unit norm, good for SVM solvers.
- # nonzeros is always less than the original, i.e., sparsity-preserving.
- Interestingly, the scheme naturally replaces the original unbiased estimator

$$\hat{R} = \frac{N_{mat}}{k - N_{emp}} \quad \text{by} \quad \hat{R}^{(0)} = \frac{N_{mat}}{\sqrt{k - N_{emp}^{(1)}}\sqrt{k - N_{emp}^{(2)}}}$$

where $N_{emp}^{(1)}$ and $N_{emp}^{(2)}$ of empty bins in S_1 and S_2, respectively.

Experimental Results on Webspam Data



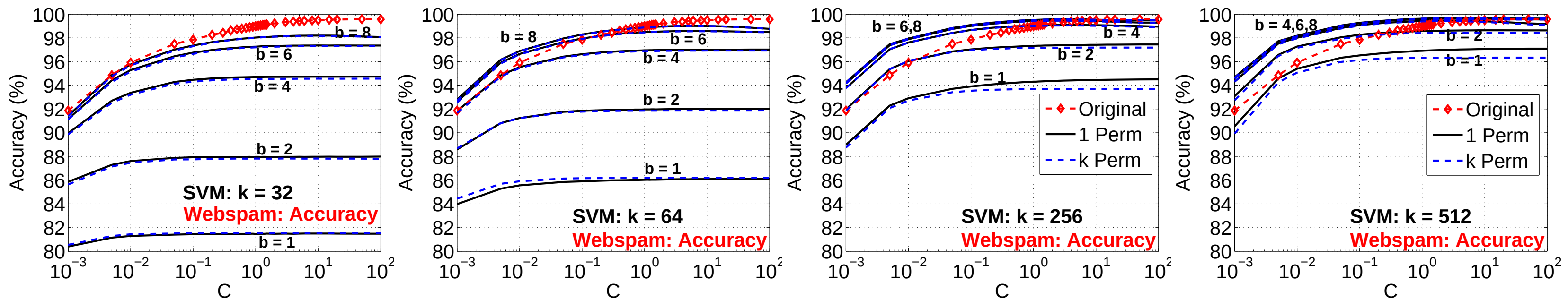


When $k = 512$ (or even 256) and $b = 8$, b-bit one permutation hashing achieves similar test accuracies as using the original data.

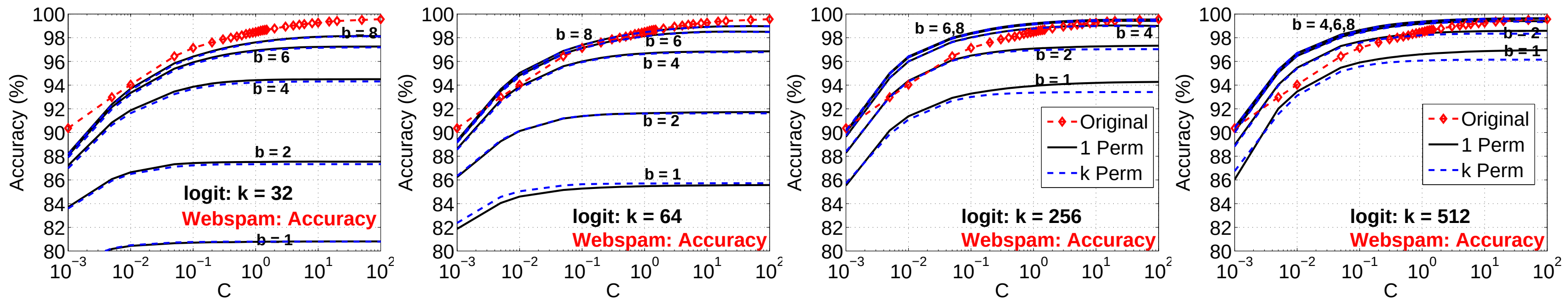
One permutation hashing (zero coding) is even slightly more accurate than k-permutation hashing (at merely $1/k$ of the original cost).

One Permutation v.s. k-Permutation

Linear SVM



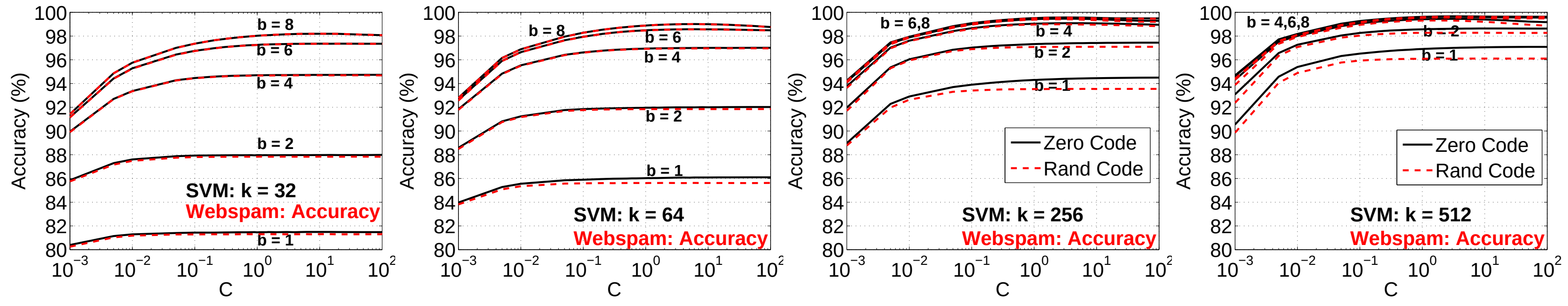
Logistic Regression



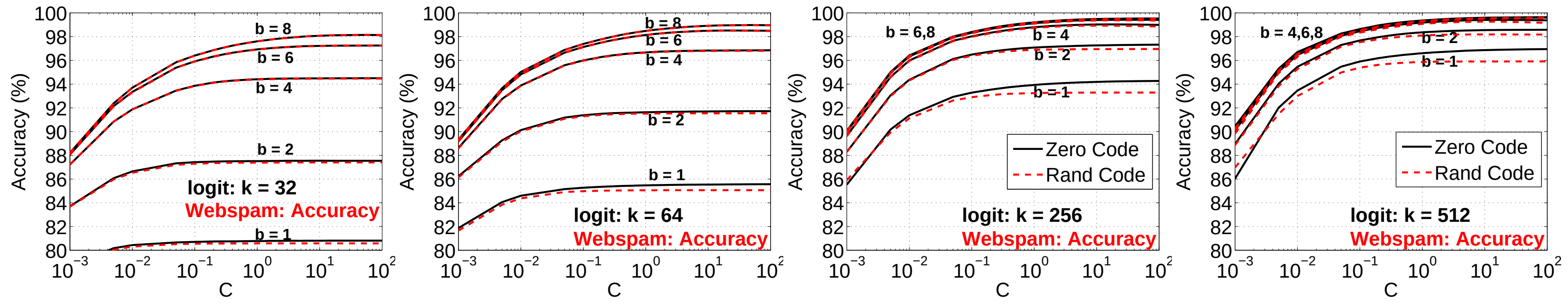
The influence of empty bins is not that obvious, for this dataset and task.

Zero Coding v.s. Random Coding

Linear SVM



Logistic Regression



Zero coding is better when k is large. Random coding replaces empty bins with artificial nonzeros, which are undesirable.

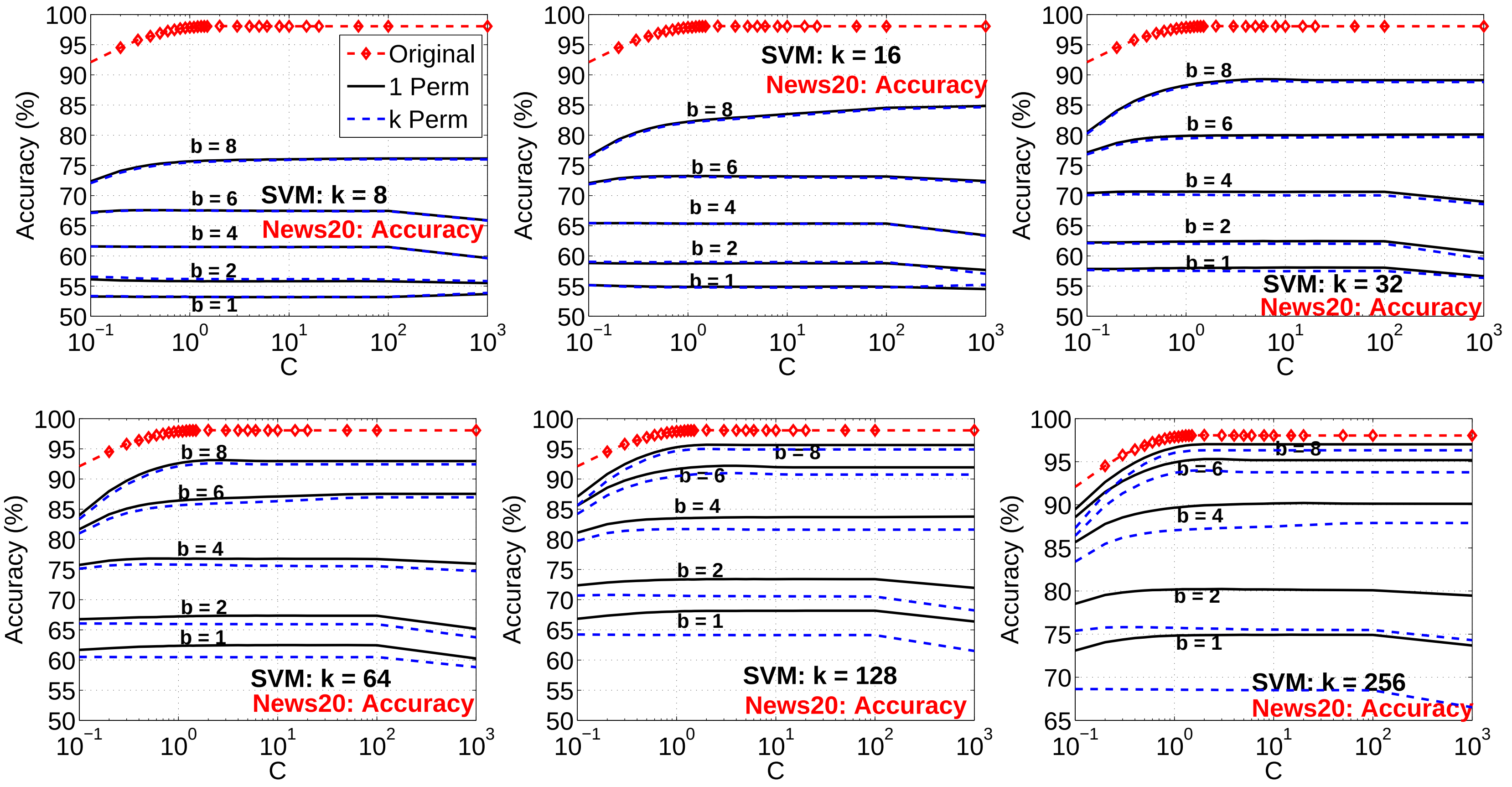
Testing the Robustness of Zero Coding on 20Newsgroup Data

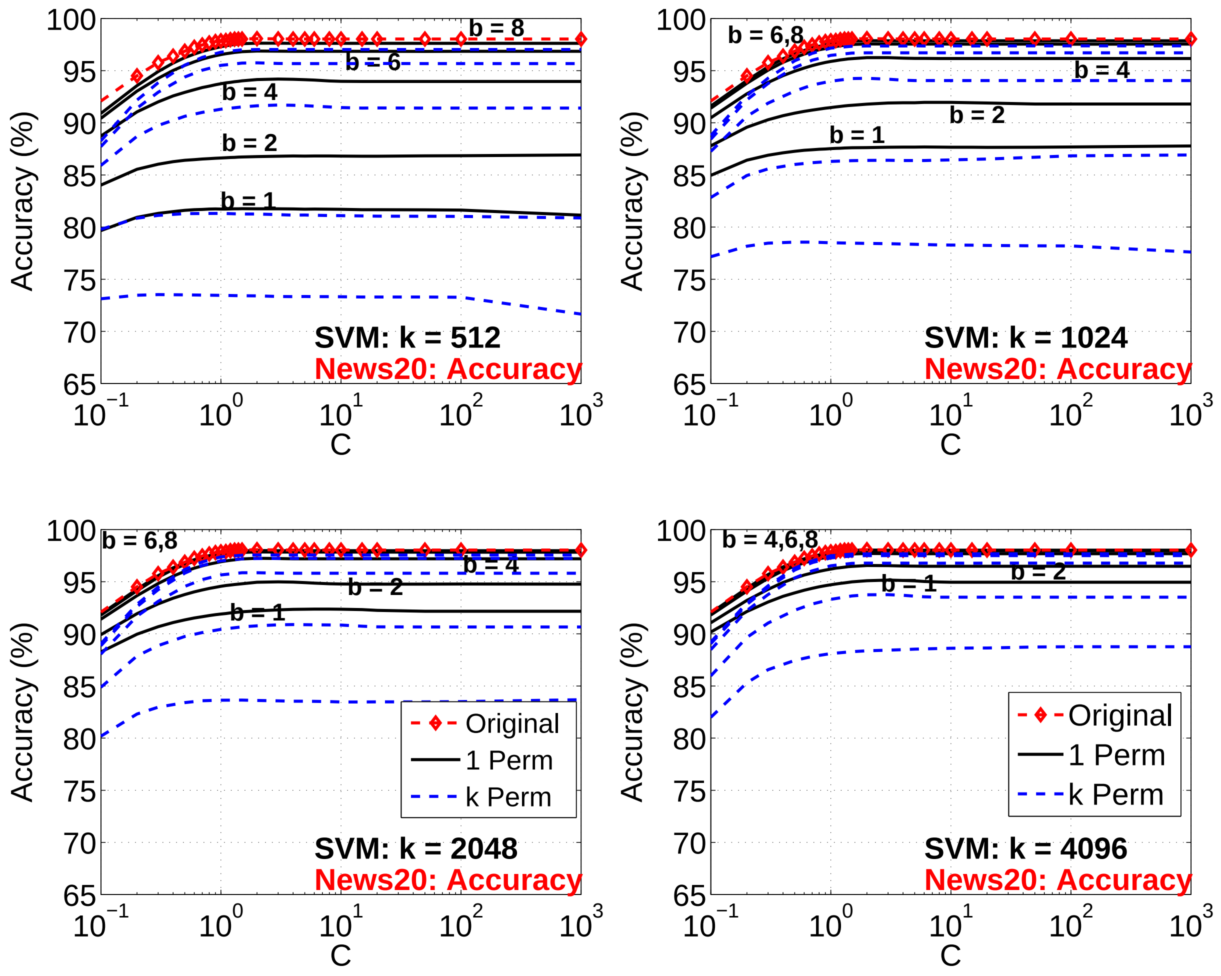
For webspam data, the # empty bins may be too small to make the algorithm fail.

The 20Newsgroup (News20) dataset has merely $20,000$ samples in about one million dimensions, with on average about 500 nonzeros per sample.

Therefore, News20 is merely a toy example to test the robustness of our algorithm. In fact, we let k as large as $k = 4096$, i.e., most of bins are empty.

20Newsgroups: One Permutation v.s. k-Permutation (SVM)



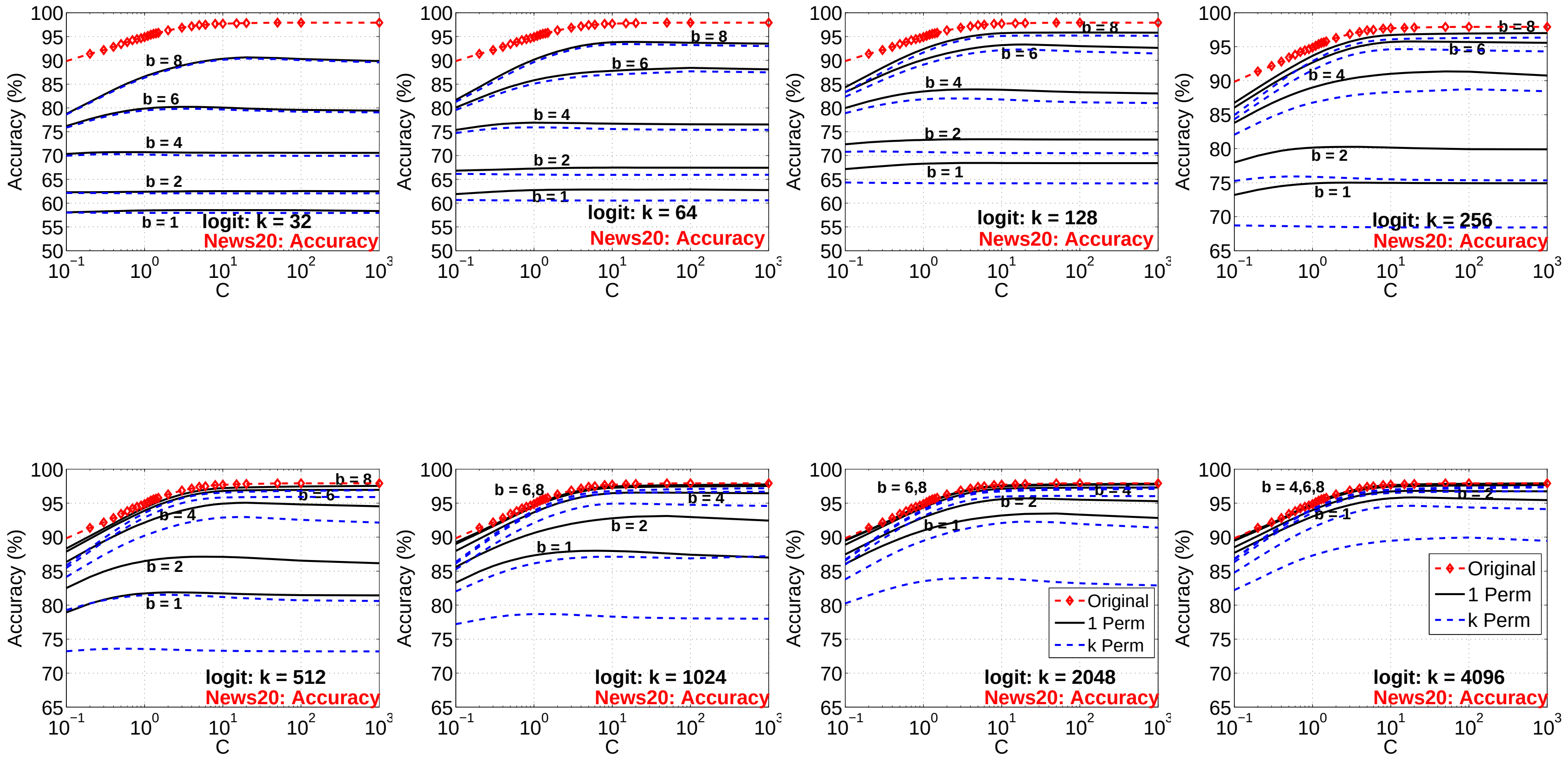


One permutation with zero coding can reach the original accuracy 98%.

The original k-permutation can only reach 97.5% even with $k = 4096$.

20Newsgroups: One Permutation v.s. k-Permutation (Logit)

Logistic Regression



Limitation of One Permutation Hashing

One permutation hashing can not be directly used for near neighbor search by building hash tables because **empty bins** do not offer indexing capability.

In other words, because of these empty bins, it is not possible to determine which bin value to use for bucketing.

Using the LSH framework, each hash table may need (e.g.,) $10 \sim 20$ hash values and we may need (e.g.,) 100 or more tables. If we generate all the necessary (e.g., 2000) hash values from merely one permutation, the chance of having empty bins might be very high in sparse data.

For example, suppose on average each data vector has 500 nonzeros. If we use 2000 bins, then roughly $(1 - 1/2000)^{500} \approx 78\%$ of the bins would be empty.

Index		Data Points
00	00	*8, 13, 251*
00	01	*5, 14, 19, 29*
00	10	*(empty)*
11	01	*7, 24, 156*
11	10	*33, 174, 3153*
11	11	*61, 342*

Index		Data Points
00	00	*2, 19, 83*
00	01	*17, 36, 129*
00	10	*4, 34, 52, 796*
11	01	*7, 198*
11	10	*56, 989*
11	11	*8 ,9, 156, 879*

Neither Zero-coding nor Random-Coding Would Work

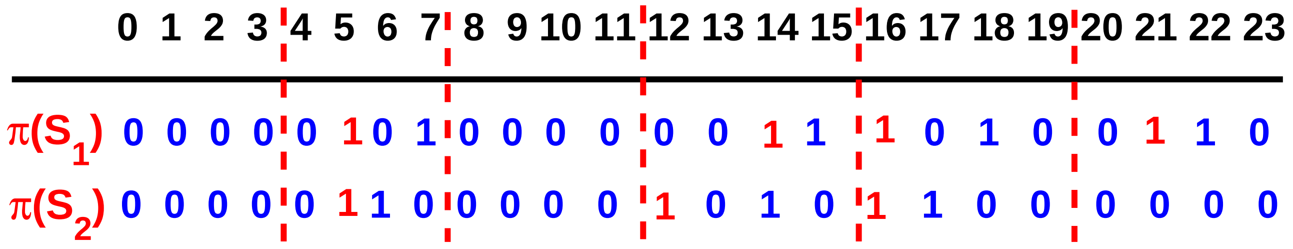
Zero-coding, or “empty-equal” (EE), scheme: If empty bins dominate, then two sparse vectors will become artificially “similar”.

Random-coding, or “empty-not-equal” (ENE), scheme: By coding an empty bin randomly, again if empty bins dominate, then two sparse vectors which are similar in terms of the original resemblance may artificially become not so similar.

Why zero-coding seems to work with linear learning? It works because in the worst case (when the number of bins is the same as the number of columns), we get back the original inner product (which is not necessarily bad).

Our Proposal: One Permutation with Rotation

The original one permutation hashing (**OPH**) is densified to become **H**):


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
π(S1) 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 1 0
π(S2) 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0

$OPH(\pi(S_1)) = [\, E, 1, E, 2, 0, 1 \,] \rightarrow H(\pi(S_1)) = [\, 1{+}C, 1, 2{+}C, 2, 0, 1 \,]$

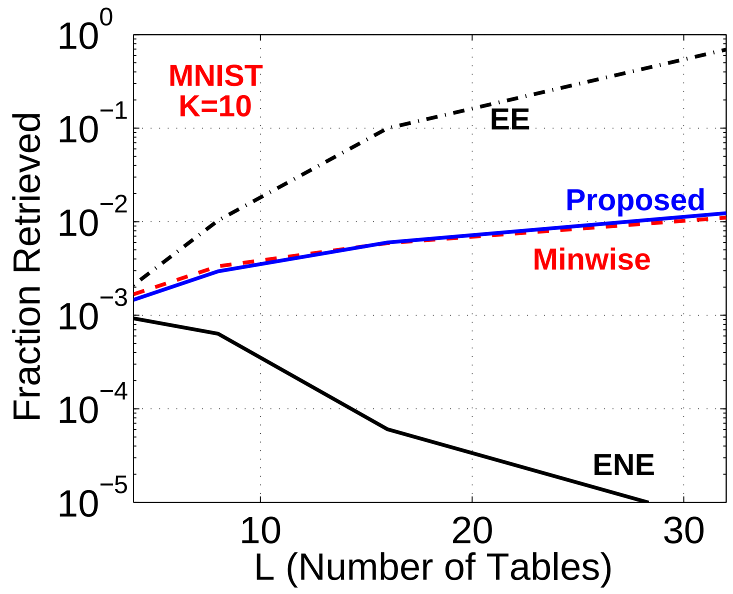
$OPH(\pi(S_2)) = [\, E, 1, E, 0, 0, E \,] \rightarrow H(\pi(S_2)) = [\, 1{+}C, 1, C, 0, 0, 1{+}2C$

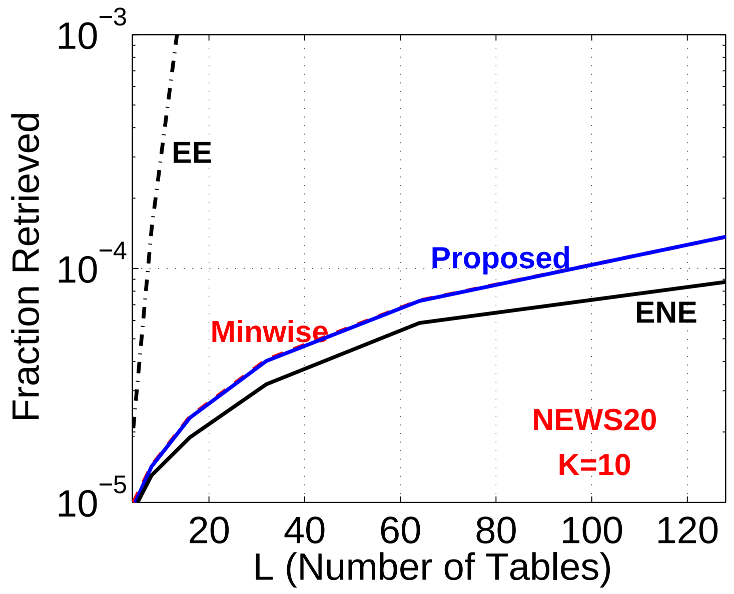
$C \geq D/k + 1$ is a constant for avoiding undesired collision.

Theorem

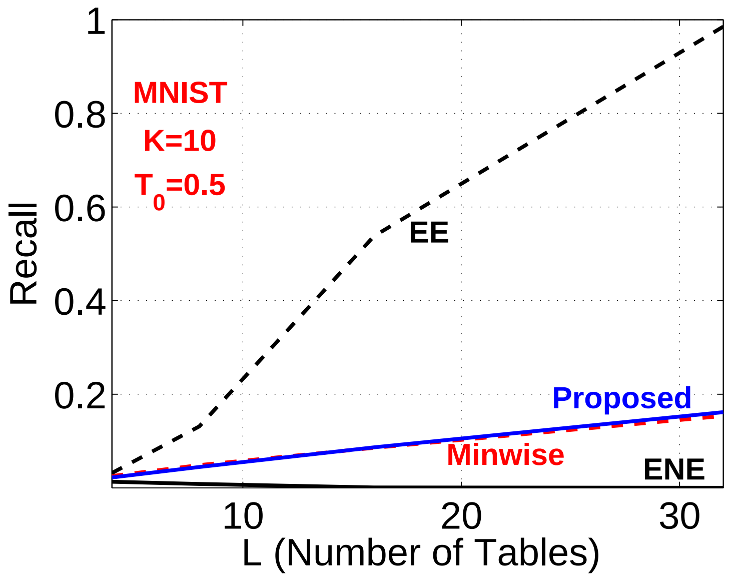
$$\mathbf{Pr}\left(\mathcal{H}_j(\pi(S_1)) = \mathcal{H}_j(\pi(S_2))\right) = R$$

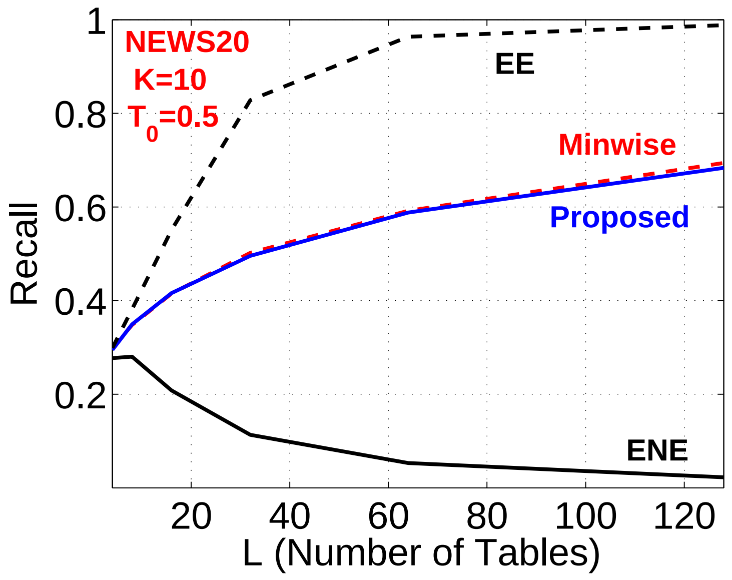
LSH Experiments for Near Neighbor Search





Our proposal matches the standard minwise hashing.


MNIST
K=10
T_0=0.5
EE
Proposed
Minwise
ENE
Recall
L (Number of Tables)


NEWS20
K=10
T_0=0.5
EE
Minwise
Proposed
ENE
Recall
L (Number of Tables)

Outline

1. **Sign Cauchy Random Projections and χ^2 Kernels**

   An interesting & surprising story about hashing for search & learning.

   Ref: Li, Samorodnitsky, and Hopcroft, NIPS 2013

2. **Conditional Random Sampling (CRS)**

   Our very first work on hashing, also related to one permutation hashing.

   Ref: Li and Church, EMNLP 2005. *Ref: Li, Church, and Hastie, NIPS 2006*

3. **One Permutation Hashing and Densified One Permutation Hashing**

   A safe replacement of the standard & popular k-permutation minwise hashing.

   Ref: Li, Owen, and Zhang, NIPS 2012. *Ref: Shrivastava and Li, ICML 2014*.

4. **Very Sparse Random Projections and Compressed Sensing**

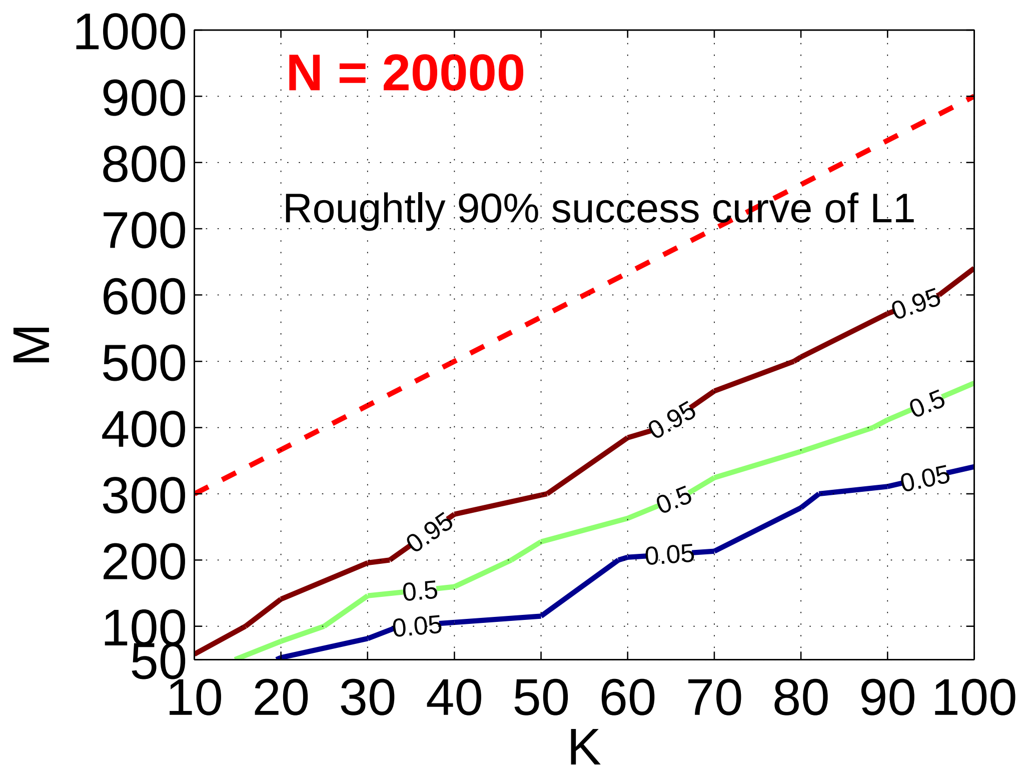
   Ref: Li, Church, and Hastie, KDD 2006. Also ongoing research.

Compressed Sensing with Very Sparse Random Projections

Compressed sensing (sparse recovery) has been a very popular topic of research. There were prior well-known papers on using sparse matrices for sparse recovery, for example, count-sketch and SMP.

This wiki page `http://groups.csail.mit.edu/toc/sparse/wiki/index.php?title=Sparse_Recovery_Experiments` shows that count-sketch needs about 10 (or 15) times more measurements than L1 decoding, and SMP needs about half of the measurements as count-sketch.

Our method for compressed sensing with very sparse projections



The 90% success curve for L1 decoding is drawn according to the wiki page.

Our method, which requires essentially one scan (like count-min sketch), produces surprisingly good results. A technical report will be posted soon.

Conclusions and Lessons Learned

- Probabilistic hashing can be extremely powerful in practice.
- Choosing a good randomization scheme is an art. Whether a strategy can be successful will depend on data types, applications, and statistical estimations.
- In theory, it is necessary to combine expertise from theoretical CS, system, statistics, and applied math, to develop hashing algorithms in a “single loop” (i.e., identify engineering problem, develop algorithm, conduct theoretical analysis, implement the method and integrate it to system).
- In reality, it is difficult for a single personnel (from academia) to do all of these. The good news is that many schools have realized the importance of interdisciplinary nature of “big data” research and many new hires are jointly with multiple departments (CS, Statistics, EE, Math, etc).