Statistics for BigData: Compressed Sensing, Search & Learning

Ping Li

Department of Statistics & Biostatistics

Department of Computer Science

Rutgers, the State University of New Jersey

Piscataway, NJ 08854

September 19, 2013

A Simple Interesting Story about Compressed Sensing

Sparse Signal: $x_1 = x_2 = 1, \qquad x_i = 0, \quad 3 \leq i \leq N$

But we know neither the locations nor the magnitudes of the nonzero coordinates. In fact, the entries of $\mathbf{x}$ can be time-varying (e.g., data streams).

Task: Recover $\mathbf{x}$ from a small number of linear nonadaptive **Measurements:**

$$\boxed{y_j = \sum_{i=1}^N x_i s_{ij}} = x_1 s_{1j} + x_2 s_{2j} = s_{1j} + s_{2j}, \;\; j = 1, 2, ..., M.$$

for this particular example.

The matrix $\{s_{ij}\}$ is called the **design matrix**, which can be manually generated and/or implemented by hardware (such as cameras).

A 3-Iteration 3-Measurement Scheme

$\{y_j\}$ is the measurement vector and $\{s_{ij}\}$ is the design matrix.

For this example, $\boxed{y_j = s_{1j} + s_{2j}}$, $j = 1, 2, ..., M$.

Ratio Statistics:

$$z_{1,j} = y_j/s_{1j} = 1 + \frac{s_{2j}}{s_{1j}}$$

$$z_{2,j} = y_j/s_{2j} = 1 + \frac{s_{1j}}{s_{2j}}$$

$$z_{i,j} = y_j/s_{ij} = \frac{s_{1j}}{s_{ij}} + \frac{s_{2j}}{s_{ij}}, \ i \geq 3$$

Ideal Design: s_{2j}/s_{1j} is either 0 or $\pm\infty$, i.e., $z_{1,j} = 1$ or $\pm\infty$.

Suppose we use $M = 3$ measurements.

First coordinate $z_{1,j} = y_j/s_{1j} = 1 + \frac{s_{2j}}{s_{1j}}$

Second coordinate $z_{2,j} = y_j/s_{2j} = 1 + \frac{s_{1j}}{s_{2j}}$

Suppose s_{2j}/s_{1j} is either 0 or $\pm\infty$:

$j = 1$: If $\frac{s_{2j}}{s_{1j}} = 0$, then $z_{1,1} = 1$ (truth), $z_{2,1} = \pm\infty$ (useless)

$j = 2$: If $\frac{s_{2j}}{s_{1j}} = \pm\infty$, then $z_{1,2} = \pm\infty$, $z_{2,2} = 1$

$j = 3$: If $\frac{s_{2j}}{s_{1j}} = 0$, then $z_{1,3} = 1$, $z_{2,3} = \pm\infty$

With 3 measurements, we see $z_{1,j} = 1$ twice and we safely estimate $\hat{x}_1 = 1$.

In the second iteration, we compute the residuals and update the ratio statistics:

$$r_j = y_j - \hat{x}_1 s_{1j} = s_{2j}$$

$$z_{2,j} = r_j / s_{2j} = 1, \quad j = 1, 2, 3$$

Therefore, we can correctly estimate $\hat{x}_2 = 1$.

In the third iteration, we update the residual and ratio statistics:

$$r_j = 0$$

$$z_{i,j} = r_j/s_{ij} = 0, \; i \geq 3$$

This means, all zeros are identified.

An important (and perhaps surprising) consequence:

$\boxed{M = 3 \text{ measurements suffice for } K = 2\text{, regardless of } N.}$

This in a sense contradicts the classical compressed sensing result that $O(K\log N)$ measurements are needed.

Realization of the Ideal Design

It suffices to sample s_{ij} from α-stable distribution: $s_{ij} \sim S(\alpha, 1)$ with $\alpha \rightarrow 0$.

The standard procedure: $w \sim exp(1)$, $u \sim unif(-\pi/2, \pi/2)$, w and u are independent. Then

$$\frac{\sin(\alpha u)}{(\cos u)^{1/\alpha}}\left[\frac{\cos(u-\alpha u)}{w}\right]^{(1-\alpha)/\alpha} \sim S(\alpha, 1)$$

which can be practically replaced by $\pm\frac{1}{[unif(0,1)]^{1/\alpha}}$.

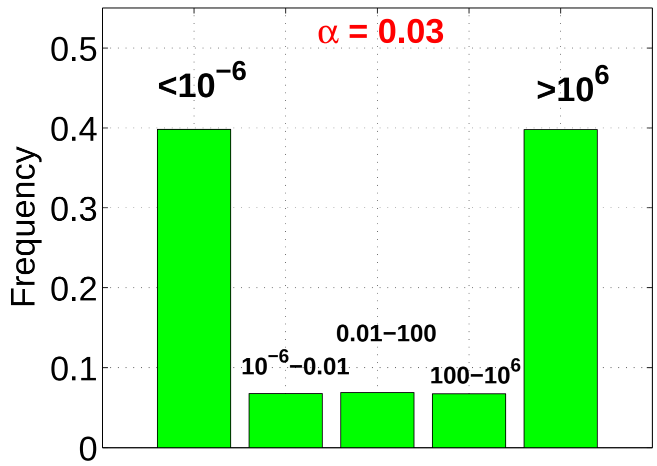
Stability: If $S_1, S_2 \sim S(\alpha, 1)$ i.i.d., then for any constants C_1, C_2,

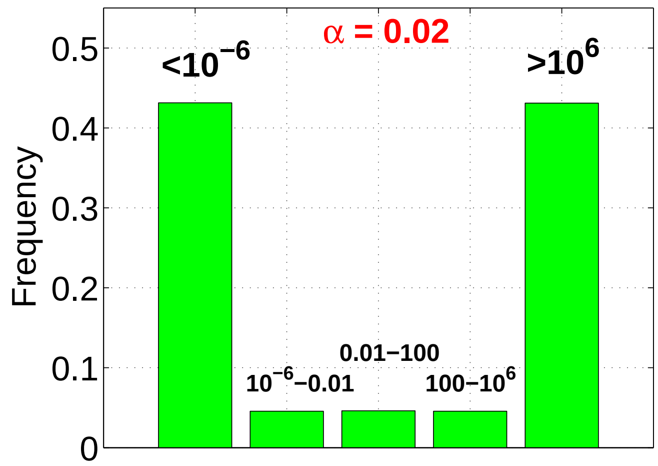
$$C_1S_1 + C_2S_2 = S \times \left(|C_1|^\alpha + |C_2|^\alpha\right)^{1/\alpha}, \qquad S \sim S(\alpha, 1)$$

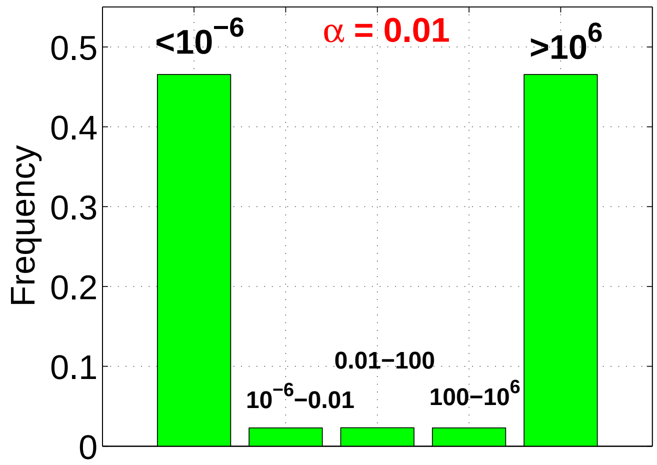
$$\sum_{i=1}^{N} x_iS_i = S \times \left(\sum_{i=1}^{N} |x_i|^\alpha\right)^{1/\alpha}$$

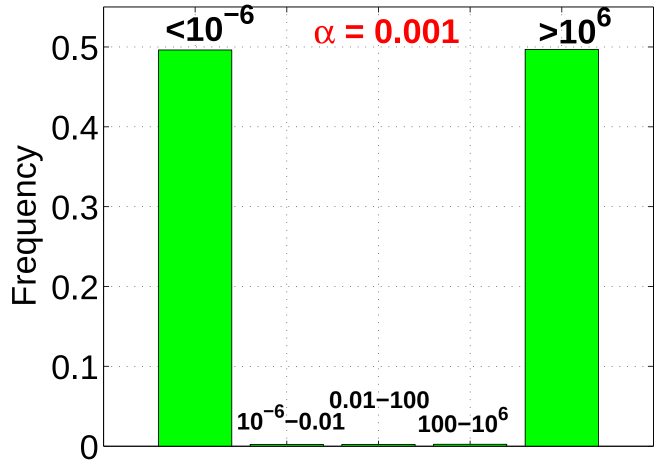
Ratio of Two Independent α-Stable Variables

$S_1, S_2 \sim S(\alpha, 1)$ independent. Ratio $|S_2/S_1|$ is either very small or very large.









Using very small α requires more precision. In this preliminary work, we simply use Matlab and let $\alpha = 0.03$.

————-

Recall, when $K = 2$, the ratio statistics are

$$z_{1,j} = y_j/s_{1j} = 1 + \frac{s_{2j}}{s_{1j}}$$

$$z_{2,j} = y_j/s_{2j} = 1 + \frac{s_{1j}}{s_{2j}}$$

Advantages of the New Compressed Sensing Framework

Our proposal uses α-stable distributions with $0 < \alpha < 1$, especially for small α.

- **Computationally very efficient**, with the main cost being one linear scan.
- **Very robust to measurement noise**, unlike traditional methods.
- **Fewer (or at most the same) measurements** compared to LP.
- **The design matrix can be made very sparse** easily especially for small α

  (Related **Ref**: Ping Li, Very Sparse Stable Random Projections, KDD'07)

—————————————-

A series of technical reports are being written, for example,

Ref: Ping Li, Cun-Hui Zhang, Tong Zhang, Compressed Counting Meets Compressed Sensing, Preprint, 2013.

Compressed Counting Meets Compressed Sensing

Most natural signals (e.g., images) are **nonnegative**. Instead of using symmetric stable projections, **skewed stable** projections have significant advantages.

——————-

Ref: Li Compressed Counting, SODA'09. (Initially written in 2007)

Ref: Li Improving Compressed Counting, UAI'09.

Ref: Li and Zhang A New Algorithm for Compressed Counting ..., COLT'11.

———————-

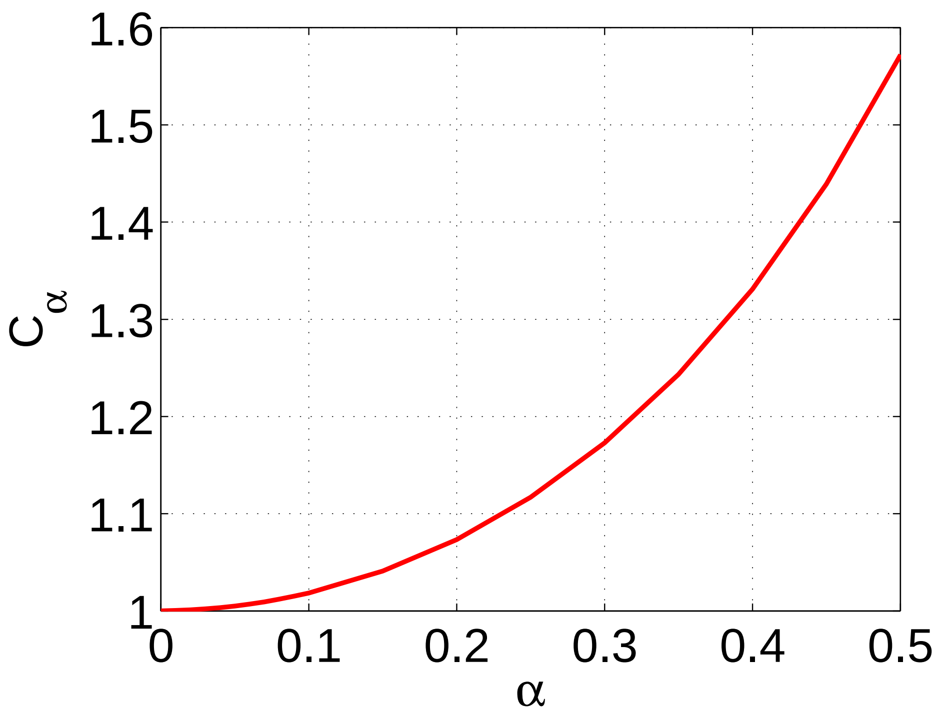
With Compressed Counting, a very simple recovery algorithm can be developed and analyzed, at least for $0 < \alpha \leq 0.5$

Sample Complexity of One-Scan Technique

Theorem: Suppose signal $\mathbf{x} \in \mathbb{R}^N$ is nonnegative, i.e., $x_i \geq 0, \forall\, i$. When $\alpha \in (0,\ 0.5]$, with α-stable maximally-skewed stable projections, it suffices to use $M = C_\alpha \epsilon^{-\alpha} \left(\sum_{i=1}^N x_i^\alpha\right) \log N/\delta$ measurements, so that all coordinates will be recovered in **one-scan** within ϵ additive precision, with probability $1-\delta$.

The constant $C_{0+} = 1$ and $C_{0.5} = \pi/2$. In particular, when $\alpha \rightarrow 0$ (exact sparse recovery), $M = K \log N/\delta$, where $K = \sum_{i=1}^N 1\{x_i \neq 0\}$.

The Constant C_α



$C_{0+} = 1$ and $C_{0.5} = \pi/2$.

Extensions and Improvements

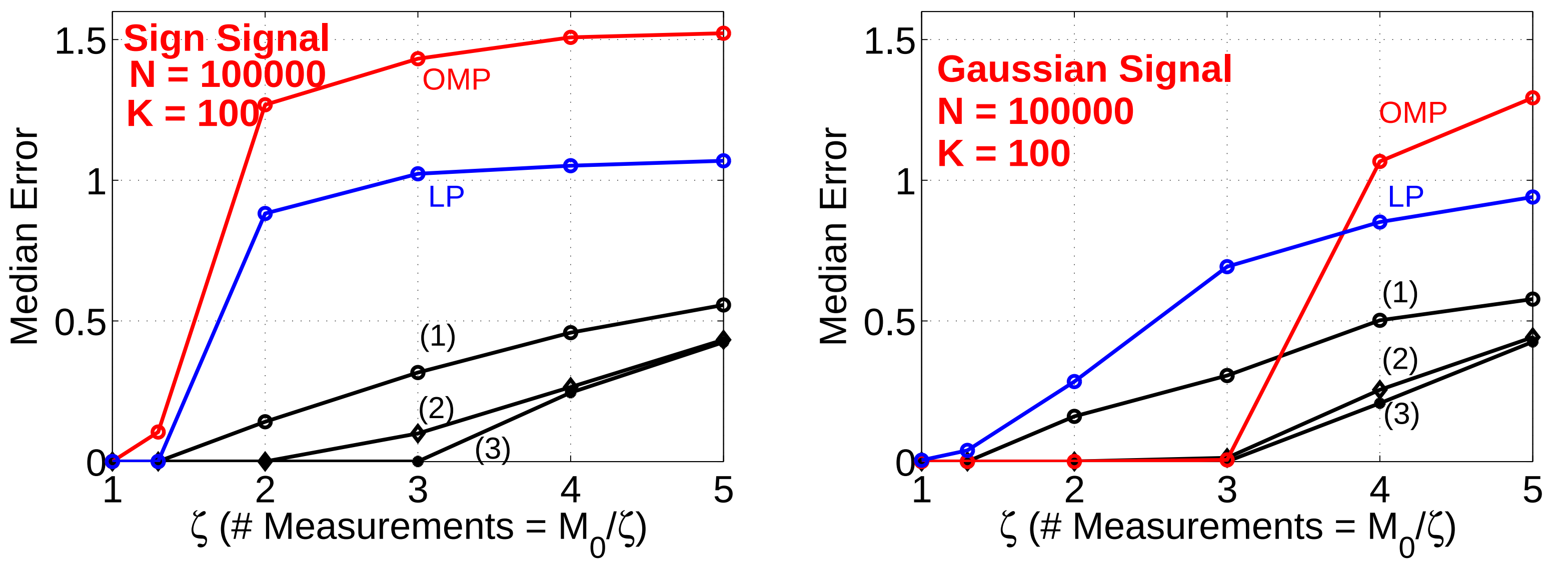
$\boxed{M = C_\alpha \epsilon^{-\alpha} \left(\sum_{i=1}^N x_i^\alpha\right) \log N/\delta}$: Complexity for one-scan method using Compressed Counting. Numerous extensions and improvements are possible:

- When the signal can be negative, use symmetric projections.
- If $\alpha \rightarrow 0+$, then $\epsilon^\alpha \rightarrow 1$. This means we can accomplish exact sparse recovery, which allows us to develop slightly more sophisticated multi-scan (ie.., with iterations) algorithm. This will substantially reduce the required number of measurements.
- Design matrix can be made extremely sparse with little impact on recovery. (Related **Ref**: Ping Li, Very Sparse Stable Random Projections, KDD'07)

Simulations

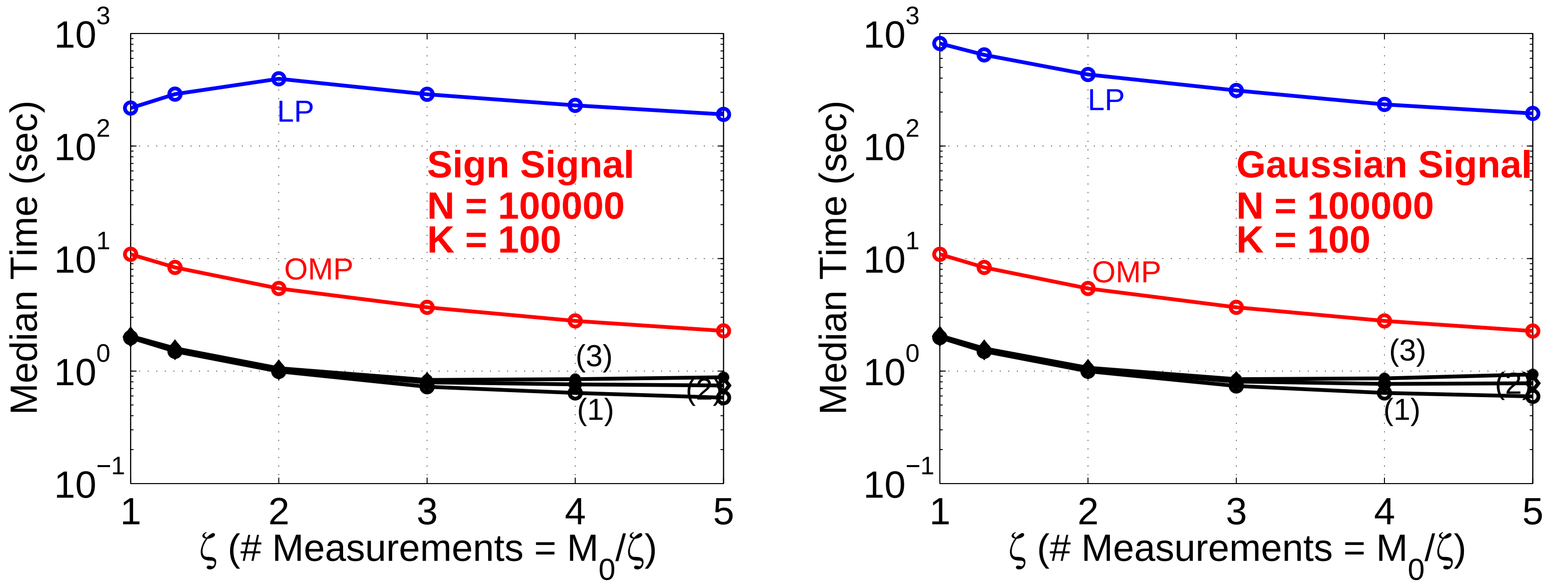
Reconstruction error

$$\text{Error} = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i - \text{estimated } x_i\right)^2}{\sum_{i=1}^{N} x_i^2}}, \qquad M_0 = K\log N/\delta, \qquad M = M_0/\zeta$$



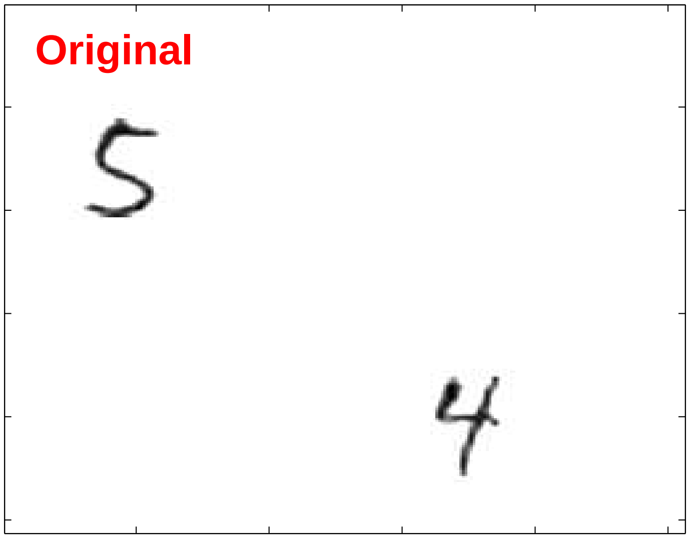
Our proposed multi-scan algorithm with (1), (2), and (3) iterations, with small α.

Decoding Time



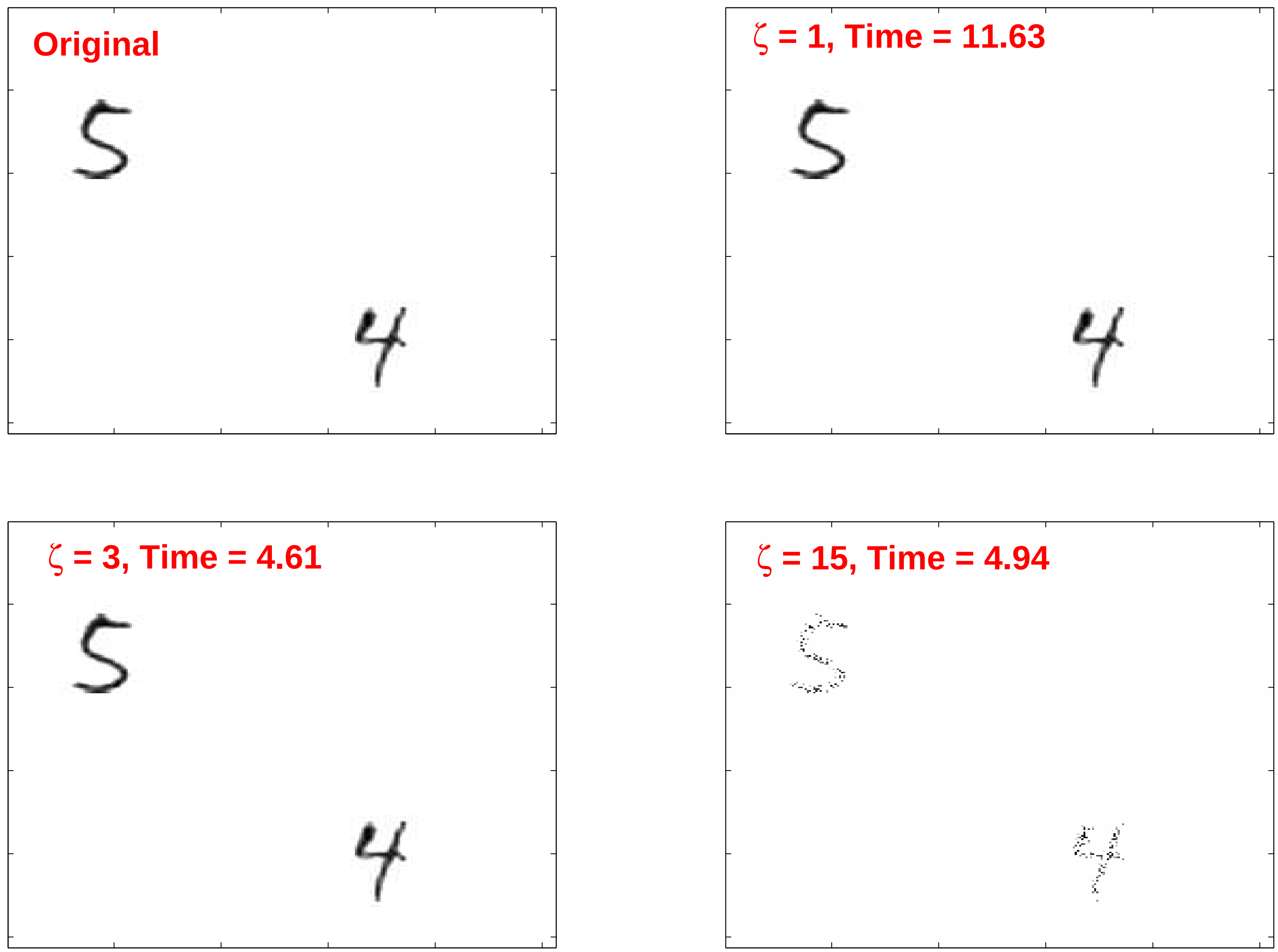
An Example of “Single-Pixel Camera” Application

The task is to take a picture using linear combination of measurements. When the scene is sparse, our method can efficiently recover the nonzero components.



In reality, natural pictures are often not as sparse, but in important scenarios, for example, the difference between consecutive frames of surveillance cameras are usually very sparse because the background remains still.

$M = K\log(N/\delta)/\zeta$ measurements.



Summary of Contributions on Compressed Sensing

- Sparse recovery is a very active area of research in many disciplines: Mathematics, EE, CS, and perhaps Statistics.
- In classical settings, the design matrix for sparse recovery is sampled from Gaussian distribution, which is $\alpha = 2$-stable distribution.
- Using α-stable distribution with $\alpha \approx 0$ leads to simple, fast, robust, accurate exact sparse recovery. Cost is one linear scan, with no catastrophic failures.
- The design matrix can be made very sparse without hurting the performance. This connects to the influential work on sparse recovery with sparse matrices.
- This is just very preliminary work. There are numerous research problems and applications which we will study in the next a few years.

Other Applications of Stable Random Projections

$\boxed{y_j = \sum_{i=1}^{N} x_i s_{ij}}$, $\quad s_{ij} \sim S(\alpha, \beta, 1)$, i.e., α-stable, β-skewed.

- Estimating the l_α norm ($\sum_{i=1}^{N} |x_i|^\alpha$) or distance in streaming data.

  Ref: Indyk JACM'06, Li SODA'08.

- Using skewed projections with $\beta = 1$ (Compressed Counting) makes entropy estimation trivial in nonnegative data streams (10 samples are needed).

  Ref: Li SODA'09, Li UAI'09, Li and Zhang COLT'11

- Using dependent symmetric ($\beta = 0$) projections also makes entropy estimation trivial in general data streams (100 samples are needed).

  Ref: Li and Zhang, Correlated Symmetric Stable Random Projections, NIPS'12.

$$\boxed{y_j = \sum_{i=1}^{N} x_i s_{ij}}$$

- Using only the signs ($sign(y_j)$, i.e., 1-bit) of the projected data leads to very efficient search and machine learning algorithms. For example, sign Cauchy (i.e., $\alpha = 1$) projection implicitly (and approximately) compute the χ^2-kernel (which is very popular in Computer Vision) with a linear kernel.

  Ref: Li, Samorodnitsky, and Hopcroft, Sign Cauchy Projections and Chi-Square Kernel, NIPS'13.

- We can make better use of more than just the signs (i.e., more than 1-bit) of the projected data.

  Ref: Li, Mitzenmacher, and Shrivastava, Coding for Random Projections, arXiv'1308.2218.

Data Streams and Entropy Estimation

- Massive data generated as streams and processed **on the fly** in **one-pass**. The problem of "scaling up for high dimensional data and high speed data streams" is among the "ten challenging problems in data mining research".

- In the standard **turnstile model**, a data stream is a vector A_t of length N, where $N = 2^{64}$ or $N = 2^{128}$ in network applications. At time t, there is an input stream $a_t = (i_t, I_t)$, $i_t \in [1,\ N]$ which updates A_t by a linear rule:

$$A_t[i_t] = A_{t-1}[i_t] + I_t.$$

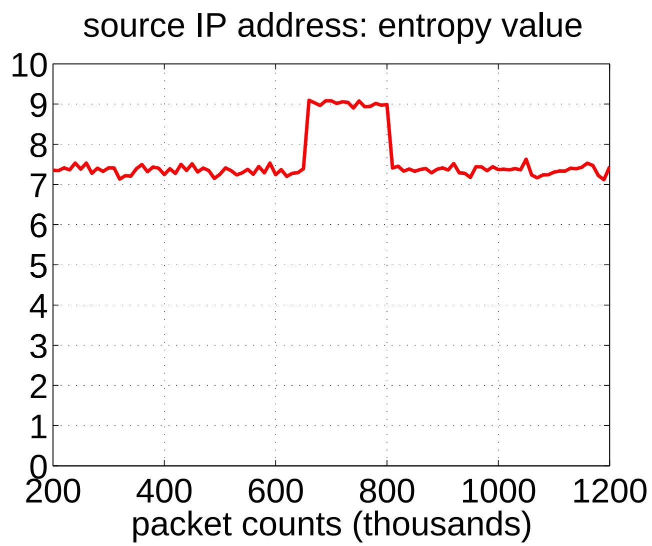
- A crucial task is to compute the α-th moment $F_{(\alpha)}$ and Shannon entropy H:

$$F_{(\alpha)} = \sum_{i=1}^{N} |A_t[i]|^{\alpha}, \qquad H = -\sum_{i=1}^{N} \frac{|A_t[i]|}{F_1} \log \frac{|A_t[i]|}{F_1},$$

Exact computation is not feasible as it requires storing the entire vector A_t.

Anomaly Detection of Network Traffic

Network traffic is a typical example of high-rate data streams. An effective measurement in real-time is crucial for anomaly detection and network diagnosis.



This plot is reproduced from a DARPA conference (Feinstein et. al. 2003). One can view x-axis as surrogate for time. Y-axis is the measured Shannon entropy, which exhibited a sudden sharp change at the time when an attack occurred.

Entropy Estimation Using Derivatives

The Shannon entropy is essentially the derivative of the frequency moment at $\alpha = 1$. A popular practice is to approximate Shannon entropy by finite difference:

$$H_\alpha = \frac{1}{\alpha - 1}\left(1 - \frac{F_{(\alpha)}}{F_{(1)}^\alpha}\right) \longrightarrow H, \quad \text{as } \alpha \rightarrow 1$$

and then estimate H by estimating the two moments $F_{(1)}^\alpha$ and $F_{(\alpha)}$

$$\hat{H}_\alpha = \frac{1}{\alpha - 1}\left(1 - \frac{\hat{F}_{(\alpha)}}{\hat{F}_{(1)}^\alpha}\right)$$

The immediate problem is that

$$Var\left(\hat{H}_\alpha\right) = O\left(\frac{1}{|\alpha - 1|^2}\right) \qquad \text{but } \alpha \rightarrow 1$$

Before our work, entropy estimation needs **lots** of samples, millions or billions.

Compressed Counting for Nonnegative Data Streams

For nonnegative data streams (as common in practice), the first moment can be computed error-free: (Recall $A_t[i_t] = A_{t-1}[i_t] + I_t$)

$$F_{(1)} = \sum_{i=1}^{N} |A_t[i]| = \sum_{i=1}^{N} A_t[i] = \sum_{s=1}^{t} I_s$$

Thus, we should expect $Var\left(\hat{F}_{(1)}\right) = 0$. It turns out using maximally-skewed stable random projections, we can achieve extremely small variance:

$$Var\left(\hat{F}_{(\alpha)}\right) = \Theta\left(|\alpha - 1|^2\right)$$

This essentially makes entropy estimation a trivial problem because now $Var\left(\hat{H}_{(\alpha)}\right) = const$ instead of $O\left(\frac{1}{|\alpha-1|^2}\right)$.

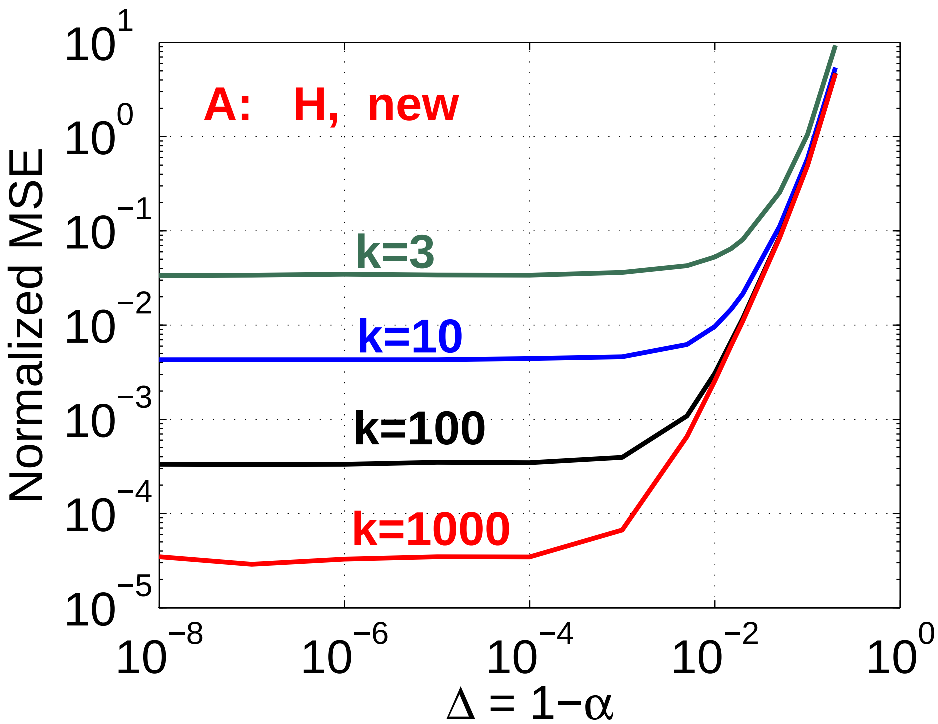
The Estimator For Nonnegative Data Streams

$$x_j = \sum_{i=1}^{N} A_t[i] s_{ij}, \qquad s_{ij} \sim S(\alpha, \beta = 1, 1)$$

$$\hat{F}_{(\alpha)} = \frac{1}{\Delta^\Delta}\left[\frac{k}{\sum_{j=1}^k x_j^{-\alpha/\Delta}}\right]^\Delta, \quad \Delta = 1-\alpha$$

$$Var\left(\hat{F}_{(\alpha)}\right) = \Theta\left(|\alpha - 1|^2\right)$$

10 Samples Provide Good Estimates



Ref: Li Compressed Counting, SODA'09. (Initially written in 2007)

Ref: Li Improving Compressed Counting, UAI'09.

Ref: Li and Zhang A New Algorithm for Compressed Counting ..., COLT'11.

Dependent Symmetric Stable Projections for General Streams

For general streams (e.g., difference between two streams), we have to use symmetric stable projections.

$$\hat{H}_\alpha = \frac{1}{\alpha - 1}\left(1 - \frac{\hat{F}_{(\alpha)}}{\hat{F}^{\alpha}_{(1)}}\right) \longrightarrow H, \quad \text{as } \alpha \rightarrow 1$$

The idea is to make the two estimators, $\hat{F}_{(1)}$ and $\hat{F}_{(\alpha)}$, highly dependent to reduce the variance.

This technique also makes $Var\left(\hat{H}_{(\alpha)}\right) = const$, by using a special sampling procedure and (seemingly) strange estimator.

Recall, If $w \sim exp(1)$, $u \sim unif(-\pi/2, \pi/2)$, w and u are independent, then

$$g(u, w; \alpha) = \frac{\sin(\alpha u)}{(\cos u)^{1/\alpha}}\left[\frac{\cos(u - \alpha u)}{w}\right]^{(1-\alpha)/\alpha} \sim S(\alpha, 1)$$

and $g(u, w; \alpha) \sim S(1, 1)$.

Thus, we can use $g(u, w; 1)$ to estimate $\hat{F}_{(1)}$, and $g(u, w; \alpha)$ to estimate $\hat{F}_{(\alpha)}$. The two estimators ought to be highly dependent.

Surprisingly, in order to remove the $O\left(\frac{1}{|\alpha-1|^2}\right)$ factor in $\hat{H}_\alpha$, we must use a special and **bad** estimator for $\hat{F}_{(\alpha)}$. Magically, the ratio of the two "bad" estimators $\frac{\hat{F}_{(\alpha)}}{\hat{F}_{(1)}^\alpha}$ leads to a good estimate of H.

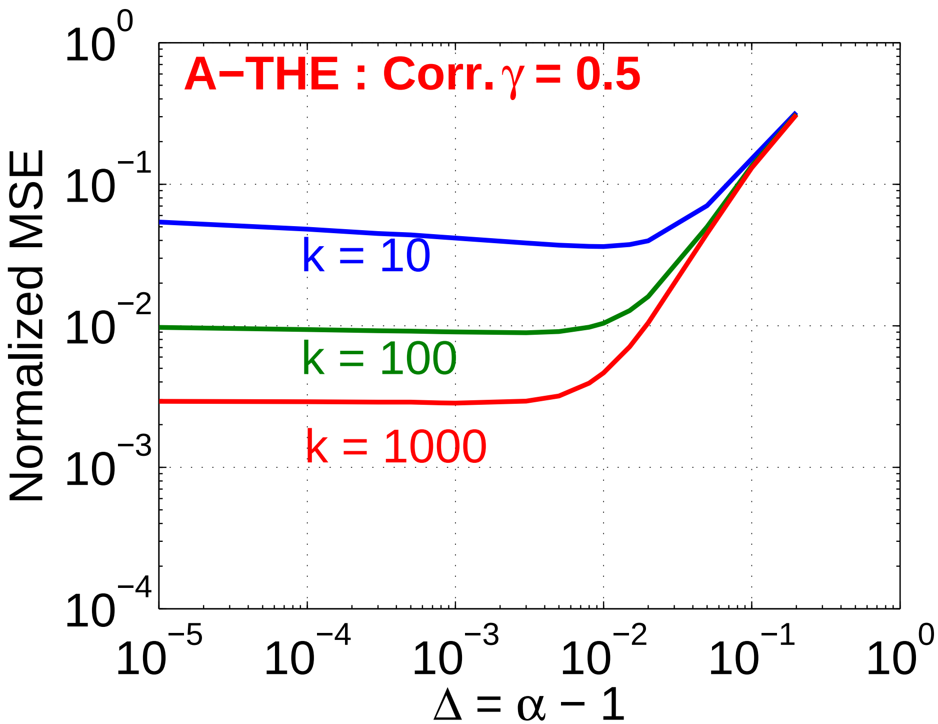
The Estimator of Entropy in General Data Streams

$$w_{ij} \sim exp(1), \qquad u_{ij} \sim uniform(-\pi/2, \pi/2)$$

$$x_j = \sum_{i=1}^{N} A_t[i] g(w_{ij}, u_{ij}, 1), \qquad\qquad y_j = \sum_{i=1}^{N} A_t[i] g(w_{ij}, u_{ij}, \alpha)$$

$$\hat{H}_\alpha = \frac{1}{\alpha - 1}\left(1 - \left(\frac{\sqrt{\pi}}{\Gamma\left(1 - \frac{1}{2\alpha}\right)} \frac{\sum_{j=1}^{k} \sqrt{|y_j|}}{\sum_{j=1}^{k} \sqrt{|x_j|}}\right)^{2\alpha}\right)$$

100 Samples Provide Good Estimate in General Streams



Ref: Li and Zhang Entropy Estimations Using Correlated Symmetric Stable Random Projections, NIPS'12.

Limitations of Stable Random Projections

When the high-dimensional data are binary and extremely sparse (as common in practice), it is often much more efficient to use **b-bit minwise hashing**.

This is illustrated by our work on **machine learning with bigdata**.

Practice of Statistical Analysis and Data Mining

Key Components:

Models (Methods) + Variables (Features) + Observations (Examples)

A Popular Practice:

Simple Models + Lots of Features + Lots of Data

or simply

Simple Methods + **BigData**

BigData Everywhere

Conceptually, consider a dataset as a matrix of size $n \times D$.

In modern applications, # examples $n = 10^6$ is common and $n = 10^9$ is not rare, for example, images, documents, spams, search click data.

High-dimensional (image, text, biological) data are common: $D = 10^6$ (million), $D = 10^9$ (billion), $D = 10^{12}$ (trillion), or even $D = 2^{64}$.

Examples of BigData Challenges: Linear Learning

Binary classification: 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 of Learning with Massive High-dimensional Data

- The data often can not fit in memory (even when the data are sparse).
- Data loading (or transmission over network) takes too long.
- Training can be expensive, even for simple linear models.
- Testing may be too slow to meet the demand, especially crucial for applications in search, high-speed trading, or interactive data visual analytics.
- Near neighbor search, for example, finding the most similar document in billions of Web pages without scanning them all.

Dimensionality Reduction and Data Reduction

Dimensionality reduction: Reducing D, for example, from 2^{64} to 10^5.

Data reduction: Reducing # nonzeros is often more important. With modern linear learning algorithms, the cost (for storage, transmission, computation) is mainly determined by # nonzeros, not much by the dimensionality.

In practice, where do high-dimensional data come from?

Webspam: Text Data

	Dim.	Time	Accuracy
1-gram	254	20 sec	93.30%
3-gram	16,609,143	200 sec	99.6%
Kernel	16,609,143	**About a Week**	99.6%

Classification experiments were based on linear SVM unless marked as "kernel"

—————-

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

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

Webspam: Binary Quantized Data

	Dim.	Time	Accuracy
1-gram	254	20 sec	93.30%
Binary 1-gram	254	20 sec	87.78%
3-gram	16,609,143	200 sec	99.6%
Binary 3-gram	16,609,143	200 sec	99.6%
Kernel	16,609,143	**About a Week**	99.6%

With high-dim representations, often only presence/absence information matters.

Major Issues with High-Dim Representations

- **High-dimensionality**
- **High storage cost**
- **Relatively high training/testing cost**

The search industry has commonly adopted the practice of

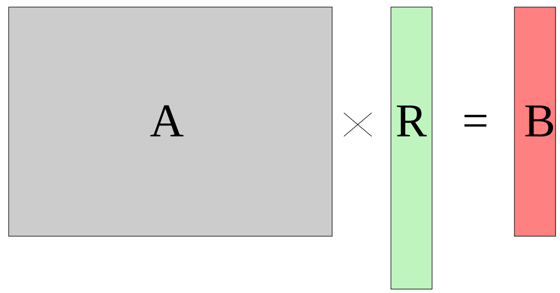
high-dimensional data + linear algorithms + **hashing**

Random projection is a standard hashing method.

Several well-known hashing algorithms are equivalent to random projections.

A Popular Solution Based on 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 $s \geq 1$:

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

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.

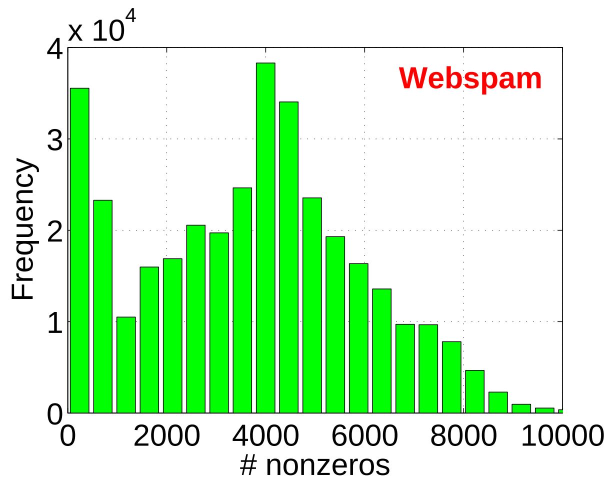
——————-

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.

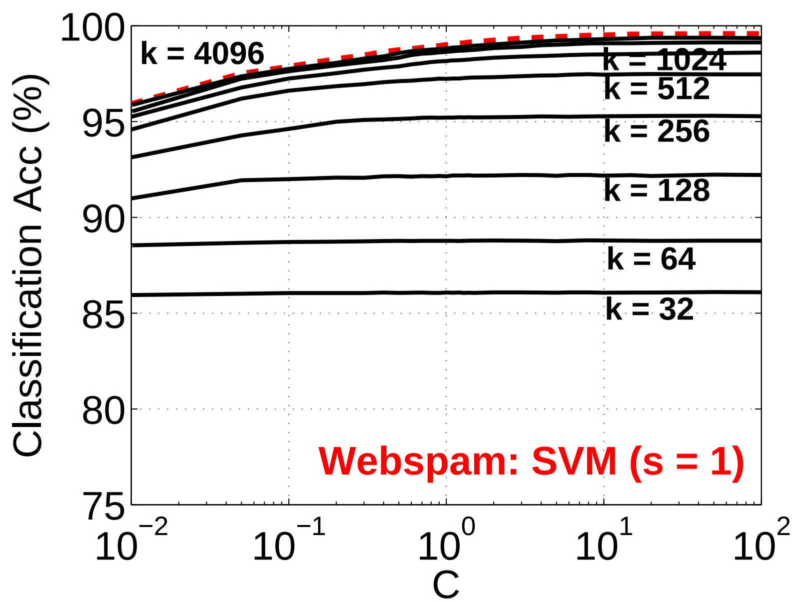


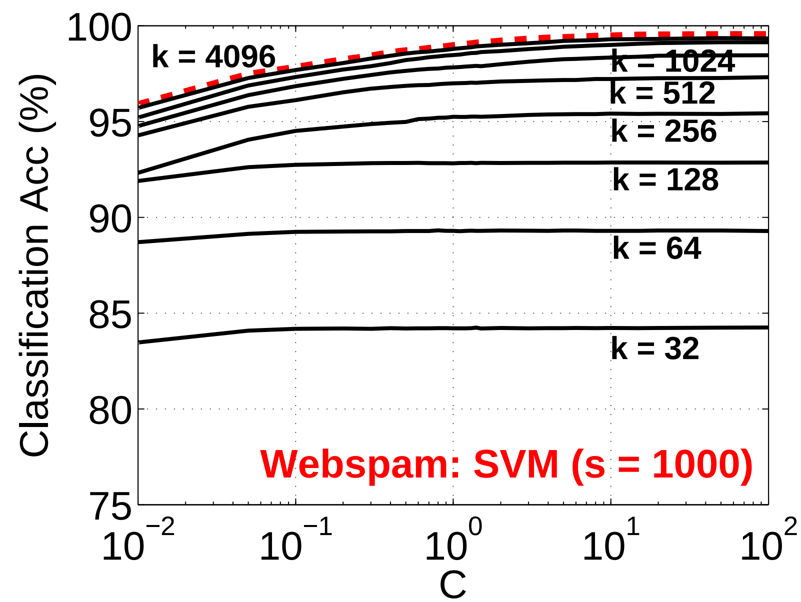
Task: Binary classification for spam vs. non-spam.

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.

Disadvantages of Random Projections (and Variants)

Inaccurate, especially on binary data.

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

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

The New Approach

- Use random permutations instead of random projections.
- Use only a small "k" (e.g.,) 200 as opposed to (e.g.,) 10^4.
- The work is inspired by minwise hashing for binary data.

This talk will focus on binary data.

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

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

Minwise Hashing in 0/1 Data Matrix

Original Data Matrix

	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

Permuted Data Matrix

	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

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

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 : ...			

....

One major problem: Need to use 64 bits to store each hashed value.

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. **Linear kernel learning**: Minwise hashing was not used for supervised learning, especially linear kernel learning.

   Our solution: We prove 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**: The industry had been using the expensive procedure since 1997 (or earlier)

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

All these are accomplished by only doing basic statistics/probability

Ref: Li and König, b-Bit Minwise Hashing, WWW'10.

Ref: Li and König, Theory and Applications of b-Bit Minwise Hashing, CACM Research Highlights, 2011 .

Ref: Li, König, Gui, b-Bit Minwise Hashing for Estimating 3-Way Similarities, NIPS'10.

Ref: Shrivastava and Li, Fast Near Neighbor Search in High-Dimensional Binary Data, ECML'12

Ref: Li, Owen, Zhang, One Permutation Hashing, NIPS'12

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

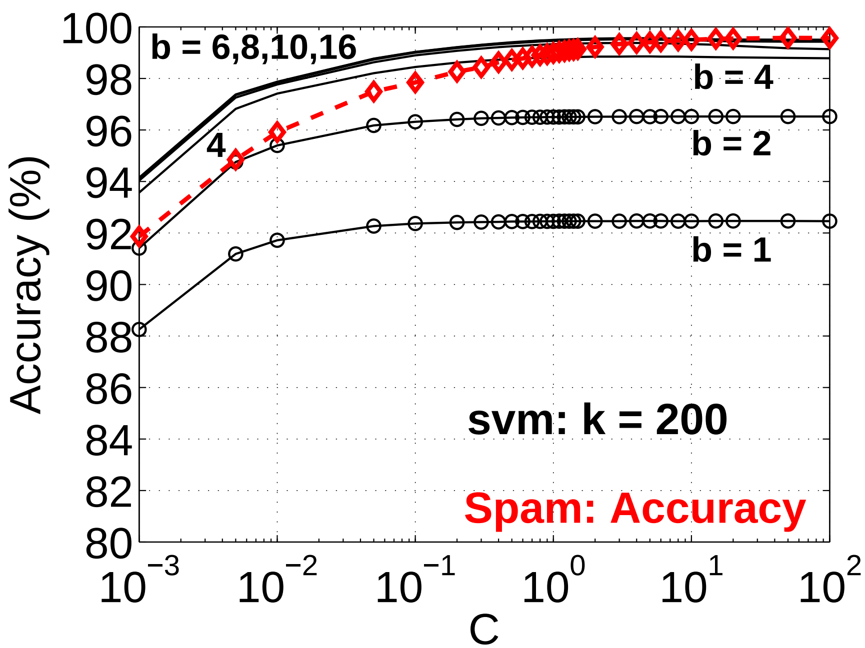
For set (vector) S_1: (Original high-dimensional binary feature vector)

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 : $[0, 1, 0, 0, \quad 1, 0, 0, 0, \quad 0, 0, 0, 1] \times \frac{1}{\sqrt{3}}$

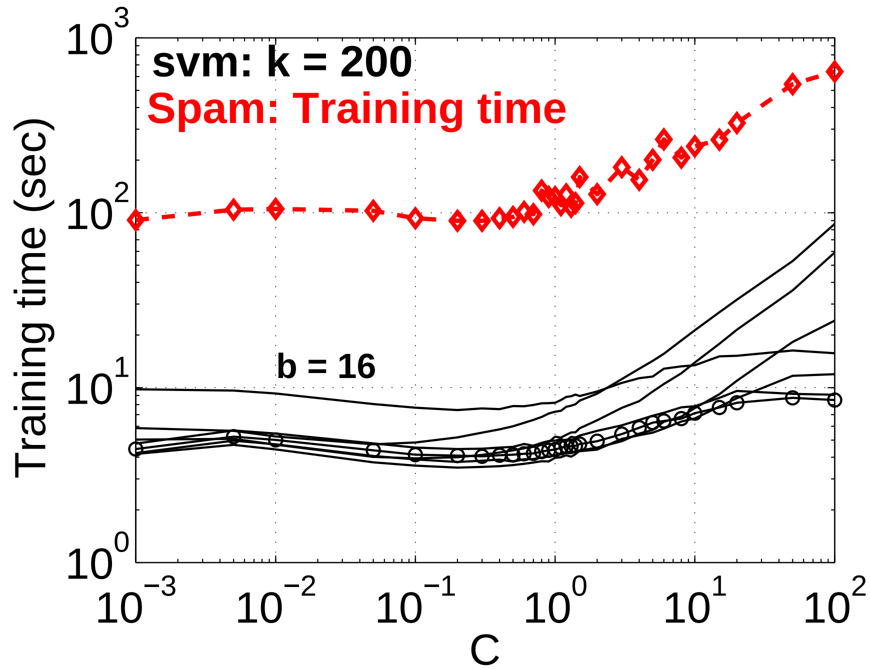
Same procedures on sets S_2, S_3, ...

Experiments on Webspam Data: Testing Accuracy



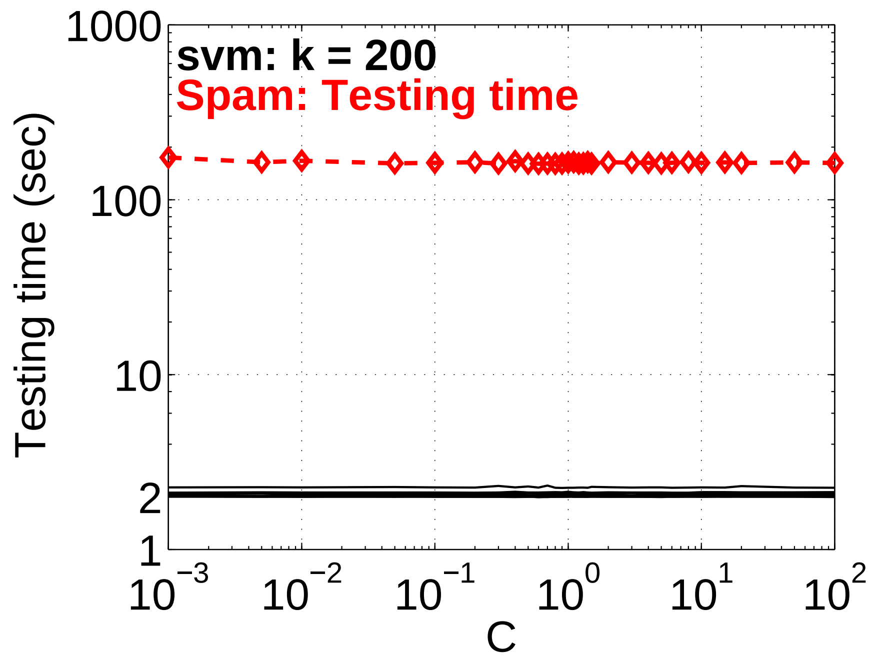
- Dashed: using the original data (**24GB** disk space).
- Solid: b-bit hashing. Using $b = 8$ and $k = 200$ achieves about the same test accuracies as using the original data. Space: **70MB** (350000×200)

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 on the entire data can be very expensive.

Particularly a serious issue when the new testing data have not been processed.

Two solutions:

1. **Parallel solution by GPUs**: Achieved up to 100-fold improvement in speed.

   Ref: Li, Shrivastava, König, GPU-Based Minwise Hashing, WWW'12 (poster)

2. **Statistical solution**: Only one permutation is needed. Even more accurate.

   Ref: Li, Owen, Zhang, One Permutation Hashing, NIPS'12

Intuition: Minwise Hashing Ought to Be Wasteful

Original Data Matrix

	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

Permuted Data Matrix

	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

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

One Permutation Hashing

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

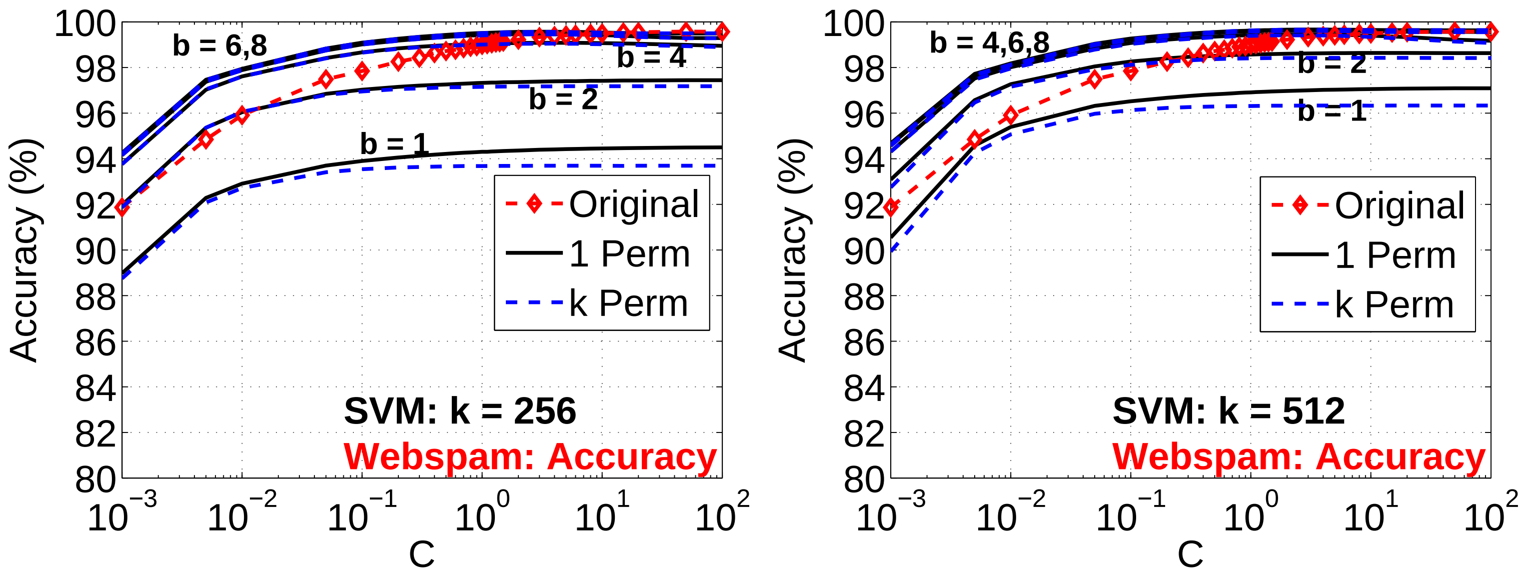
$\pi(S_1) = \{2, 4, 7, 13\}$, $\quad \pi(S_2) = \{0, 3, 6, 13\}$, $\quad \pi(S_3) = \{0, 1, 10, 12\}$

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

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

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

Experimental Results on Webspam Data



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

Summary of Contributions on b-Bit Minwise Hashing

1. **b-bit minwise hashing**: Use only the lowest b bits of hashed value, as opposed to 64 bits in the standard industry practice.

2. **Linear kernel learning**: b-bit minwise hashing results in positive definite linear kernel. The data dimensionality is reduced from 2^{64} to 2^b.

3. **One permutation hashing**: It reduces the expensive and energy-consuming process of (e.g.,) 500 permutations to merely one permutation. The results are even more accurate.

4. **Beyond Pairwise: three-way hashing and search (e.g., GoogleSets)**: **Ref:** Li, Konig, Gui, NIPS'10. **Ref:** Shrivastava and Li, NIPS'13.

5. **Others:** For example, b-bit minwise hashing naturally leads to a space-partition scheme for sub-linear time near-neighbor search.

Conclusion

- **The "Big Data" time**. Researchers & practitioners often don't have enough knowledge of the fundamental data generation process, which is fortunately often compensated by the capability of quickly collecting lots of data.
- **Approximate answers often suffice**, for many statistical problems.
- **Hashing becomes crucial in important applications**, for example, search engines, as they have lots of data and need the answers quickly & cheaply.
- **Hashing with stable random projections** leads to surprising algorithms for compressed sensing (sparse signal recovery), databases, learning, and information retrieval. Numerous research problems remain to be solved.
- **Binary data are special and crucial**. In most applications, one can expand the features to extremely high-dimensional binary vectors.

- **b-Bit minwise hashing**, in important applications, often provides significantly better results than stable random projections.
- **Probabilistic hashing is basically clever sampling**.
- **Numerous interesting research problems**. Statisticians can make great contributions in this area. We just need to treat them as statistical problems.