Fast Near Neighbor Search in High-Dimensional Binary Data

Anshumali Shrivastava

Dept. of Computer Science

Cornell University

Ping Li

Dept. of Statistical Science

Cornell University

High Dimensional Sparse Binary Data in Practice

- Consider a Web-scale term-doc matrix X $\in R^{n\times D}$ with each row representing one Web page. Certain industry applications used 5-grams (i.e., D = O(10^{25}) is conceptually possible. Assuming 10^5 common English words).
- Usually, when using 3- to 5-grams, most of the grams only occur at most once in each document. It is thus common to utilize only binary data when using n-grams.
- Conceptually, the textual content of the Web may be viewed as a giant matrix of size $n \times D$, with $n = 10^{11}$ Web pages and each page in $D = 2^{64}$ dims.
- Image Representations for retrieval and search using vector quantization naturally leads to sparse high dimensional binary data.

Near Neighbor Search

- The Classical Problem: Given a high dimensional query vector (Document or Image) we want to search a huge database for items similar to the given query.
- The simple strategy to scan all the database and compute similarities is prohibitive when
  - The data matrix X itself may be too large for the memory.
  - Computing similarities on the fly can be too time-consuming when the dimensionality D is high.
  - The cost of scanning all n data points is prohibitive and may not meet the demand in user-facing applications (e.g., search).
  - Parallelizing linear scans will not be energy-efficient if a significant portion of the computations is not needed.

Locality Sensitive Hashing (LSH)

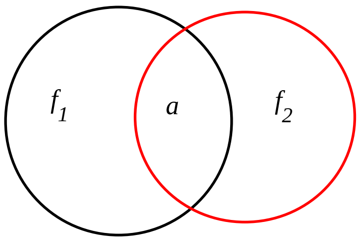
- Early space partitioning based approaches like K-D trees, R trees, etc, only good for low dimensions, typically $D < 10$, but leads to almost linear scan for higher D.
- LSH is currently one of the most popular technique in industrial practice.
- The basic idea behind LSH is to construct a randomized hash function such that similar objects are more likely to have the same hash key.
- More specifically, we are interested in hash function families $\mathcal{H}$, such that $Pr_{h\in\mathcal{H}}(h(x) = h(y)) = F(sim(x, y))$, where F is a monotonically increasing function and sim(x,y) is the similarity of interest between x and y.

Sub-linear Time Approximate Near Neighbor Search Using LSH

- For each point x, generate a hash key by concatenating K hash signatures $g(x) = \{h_1(x), h_2(x), ..., h_K(x)\}$, where each $h_i(x)$ drawn independently from the LSH family $\mathcal{H}$.
- Store data point x in a hashtable at location $g(x)$.
- Generate L such independent hashtables.
- For a given query point q, retrieve elements from the bucket $g(q) = \{h_1(q), h_2(q), ..., h_K(q)\}$ corresponding to each of the L hashtables.
- Smart choices of L, K lead to worst case approximate solution in O(n^ρ) where $\rho < 1$. (Adoni-Indyk 08)

Minwise Hashing: LSH for Set Similarity

- Binary vector can be thought of as sets. Consider two sets $S_1, S_2 \subseteq \Omega = \{0, 1, 2, ..., D-1\}$ (e.g., $D = 2^{64}$)



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

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

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

Suppose a random permutation π is performed on Ω, i.e., $\pi: \; \Omega \longrightarrow \Omega$

An elementary probability argument shows that

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

Shortcomings of Minwise Hashing

- The signatures from Minwise Hashing could be potentially 64-bits, after concatenation of K signatures, the size of hashtable will just blow up beyond feasibility.
- We are mostly interested in highly similar pairs, we probably don't need all the 64-bits.

Introduction to b-Bit Minwise Hashing

Define the minimum values to be : $z_1 = \min\left(\pi\left(S_1\right)\right), \quad z_2 = \min\left(\pi\left(S_2\right)\right).$

Recall minwise hashing: $\mathbf{Pr}\left(z_1 = z_2\right) = R$. For b-bit minwise hashing,

$$\mathbf{Pr}\left(\text{lowest b bits of } z_1 = \text{lowest b bits of } z_2\right) = C_{1,b} + \left(1 - C_{2,b}\right) R$$

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

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

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

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

Accuracy Space Tradeoff

- When the data are highly similar, a small b (e.g., 1 or 2) may be good enough. However, when the data are not very similar, b cannot be too small.
- The advantage of b-bit minwise hashing can be demonstrated through the "variance-space" trade-off: $\text{Var}\left(\hat{R}_b\right) \times b$.
- For all practical purposes, the similarities estimated from 4-bit is indistinguishable compared to 64-bit minwise hashing. (Li-Konig WWW 2010)

Our Proposal for Near Neighbor Search

- In the limiting case when the data is very sparse and r_j is typically very small then for all practical purposes we can set $A_{j,b} = \frac{1}{2^b}$ in the formula.
- b-bit minwise hashing in such case is LSH with collision probability

$$\mathbf{Pr}\left(\text{lowest b bits of } z_1 = \text{lowest b bits of } z_2\right) = \frac{1}{2^b} + (1 - \frac{1}{2^b})(R)$$

- The signatures are now at most b (1,2,4, etc.) bits, the hash table size is manageable and so it can naturally be used to build hash table for sublinear search.

Example of Hashtables

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*

Figure 1: An example of hash tables, with $b = 2$, $K = 2$, and $L = 2$.

Real Datasets used for Comparisons and Evaluations

Table 1: Data Information

Dataset	n	D
Webspam	70,000	16,609,143
NYTimes	20,000	102,660
EM30k	30,000	34,950,038

Competitor 1: Signed Random Projections (SRP)

- One of the most popular LSH is SRP (Charikar STOC 2002),

$$h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where $r \in R^d$ drawn independently from $N(0, \mathcal{I})$

- The seminal work of Geomens-Williamson showed that

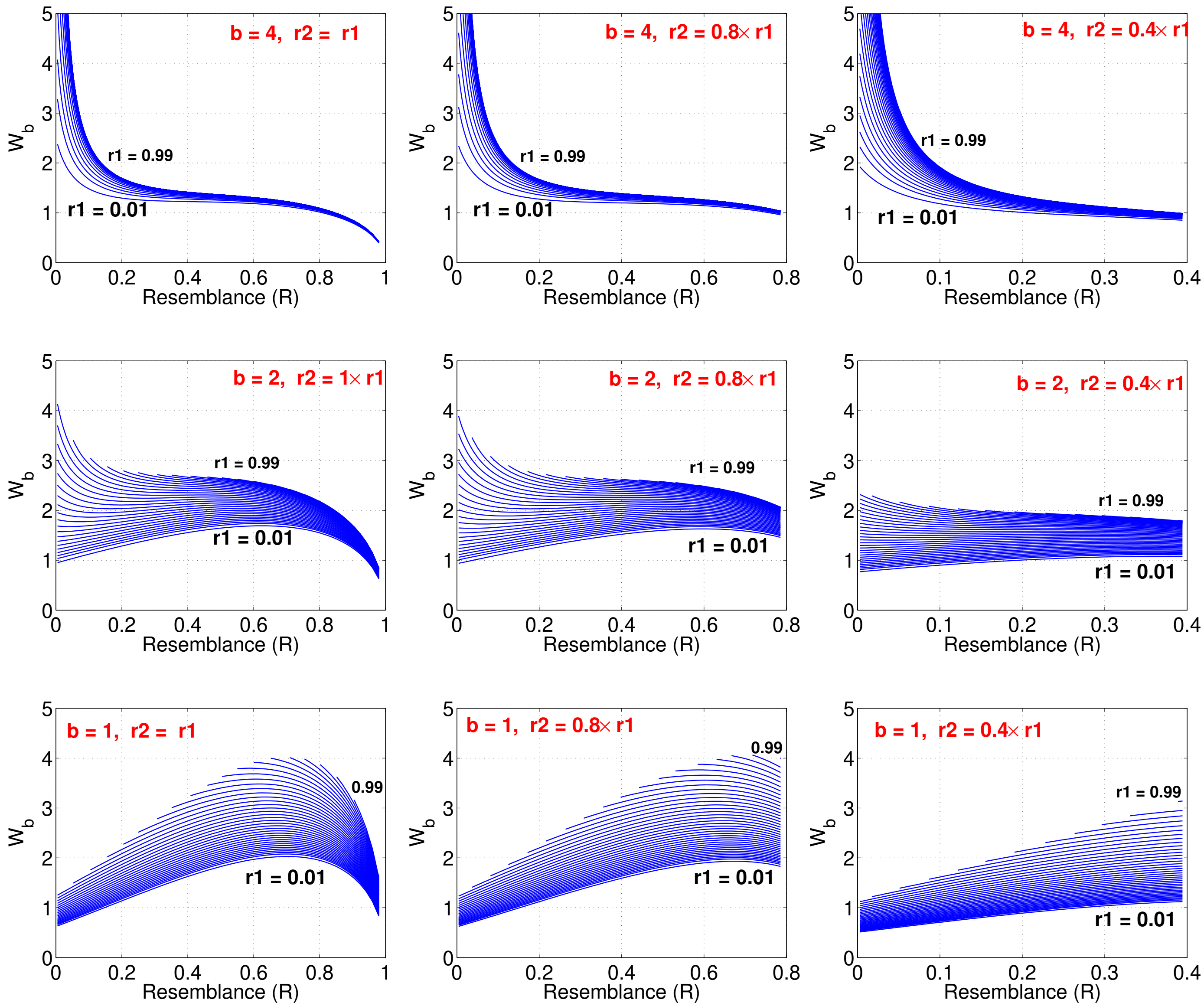
$$Pr(h(x) = h(y)) = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{x^T y}{\|x\|\|y\|}\right)$$

Why b-bit minwise hashing should be better ?

- We compare the variance of estimators of resemblance (R) using b-bit hashing $Var\left(\hat{R}_b\right)$ and signed random projections $Var\left(\hat{R}_S\right)$.
- We compute the ratio

$$W_b = \frac{Var\left(\hat{R}_S\right)}{Var\left(\hat{R}_b\right) \times b} = \frac{\theta(\pi-\theta) f_1 f_2 \sin^2(\theta)\left(\frac{f_1+f_2}{(f_1+f_2-a)^2}\right)^2}{\frac{[C_{1,b}+(1-C_{2,b})R][1-C_{1,b}-(1-C_{2,b})R]}{[1-C_{2,b}]^2}} \tag{1}$$

- $W_b > 1$ means b-bit minwise hashing is more accurate than SRP at the same storage.


b = 4, r2 = r1
b = 4, r2 = 0.8× r1
b = 4, r2 = 0.4× r1
b = 2, r2 = 1× r1
b = 2, r2 = 0.8× r1
b = 2, r2 = 0.4× r1
b = 1, r2 = r1
b = 1, r2 = 0.8× r1
b = 1, r2 = 0.4× r1
r1 = 0.99
r1 = 0.01
0.99
W_b
Resemblance (R)

Learning Approaches

- The idea is to learn a mapping from data vectors to compact binary codes which preserves the pairwise similarity.
- Unlike LSH, these approaches take into account the underlying data distribution.
- Machine learning approaches tend to outperform LSH where the data is usually sitting on some low dimensional manifolds.
- What about extremely high dimensional sparse data?
  - Most of these methods are almost impossible to train at such scale.
  - Not much is known about the performance of these approaches at such scale.

Competitor 2: Spectral Hashing

- Spectral Hashing is one of the state-of-the-art learning based hashing methods.
- Closely related to the problem of spectral graph partitioning.
- Aims to minimize the average hamming distance between the output codes of similar objects, subject to constraints that the bits are independent and uncorrelated.
- The minimization is done efficiently via one dimensional eigenfunctions.

Spectral Hashing (SH) Formulation

Let $\{y_i\}$ be the list of code words (binary vectors of length k) for each data point and $W_{ij} = e^{\frac{-||x_i - x_j||^2}{\epsilon^2}}$ be the similarity function. The SH aims to solve

$$minimize : \sum_{ij} W_{ij} ||y_i - y_j||^2$$

subject to:

$$y_i \in \{-1, 1\}^k$$

$$\sum_i y_i = 0$$

$$\frac{1}{n} \sum_i y_i y_i^T = \mathcal{I}$$

Spectral Hashing (SH) Algorithm

- Fit a multi-dimensional rectangle to the data. (Run PCA to align axes, then bound uniform distribution.)
- For each dimension, calculate k smallest eigenfunctions.
- Threshold eigenfunctions at zero to give binary codes.

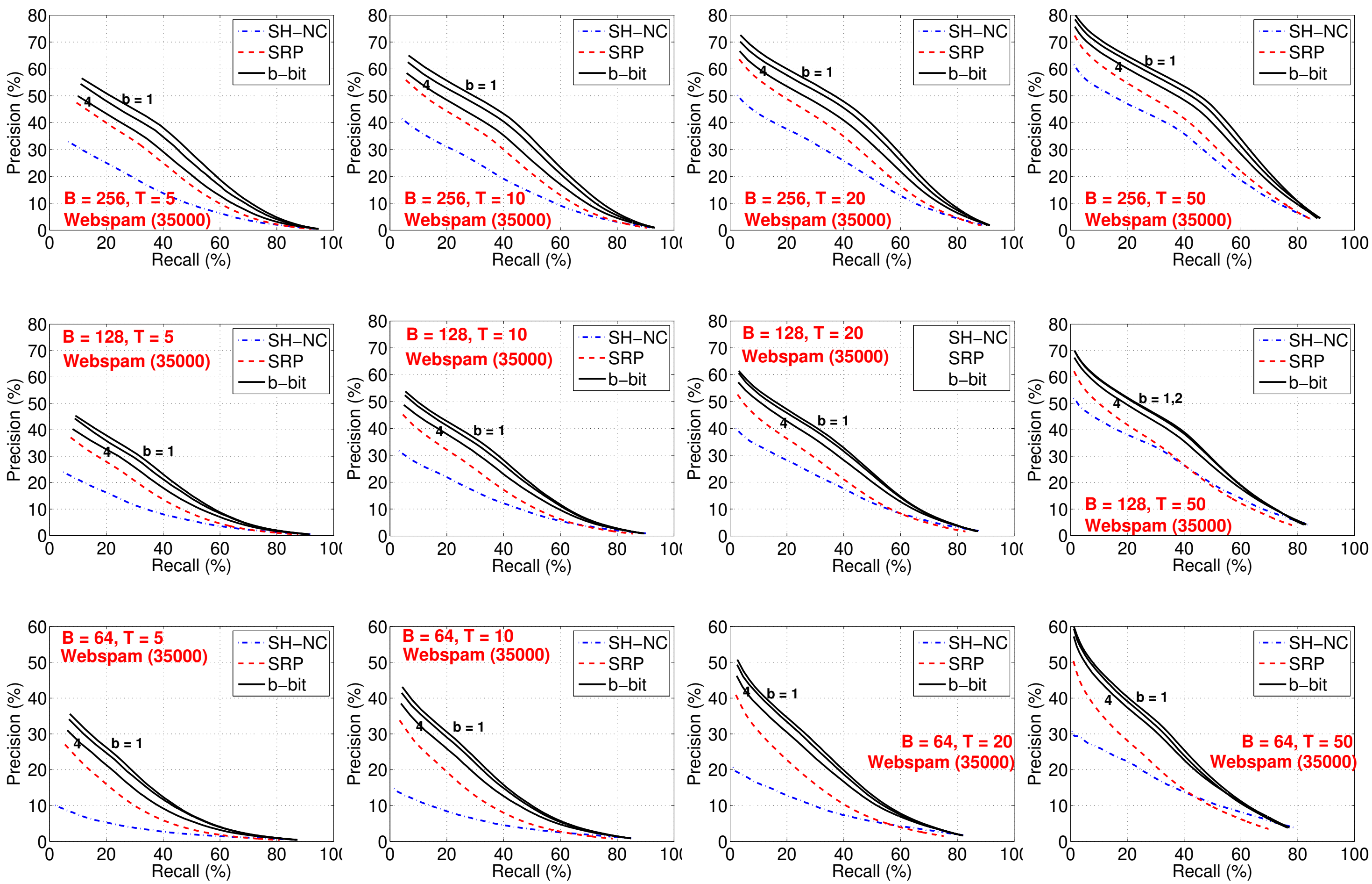
Making Spectral Hashing Work

- We replace the eigen-decomposition operations by equivalent SVD operations which avoids materializing the dense covariance matrix.
- PCA need a centering step which makes the data non-sparse and impossible to handle.
- We empirically observe that skipping centering step does not affect the performance of SH on the small subsets of data.
- Skipping the centering step made it possible to train SH on full datasets instead of small samples.

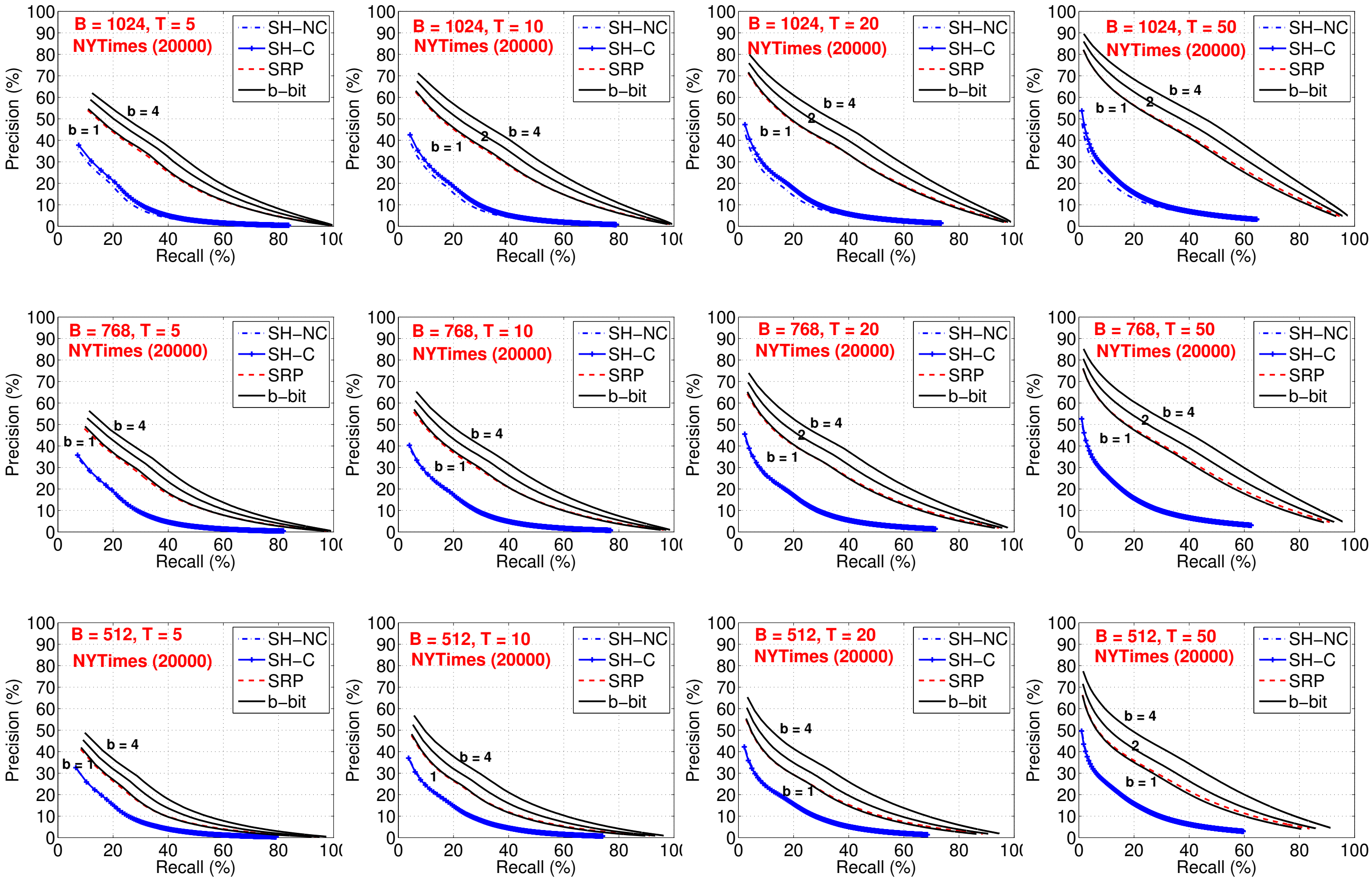
Evaluation 1: Hash Code Quality Evaluation

- We generate binary codes of fixed length using the three methodologies.
- Retrieve nearest neighbor based on the similarity between binary codes.
- Plot precision-recall curves.

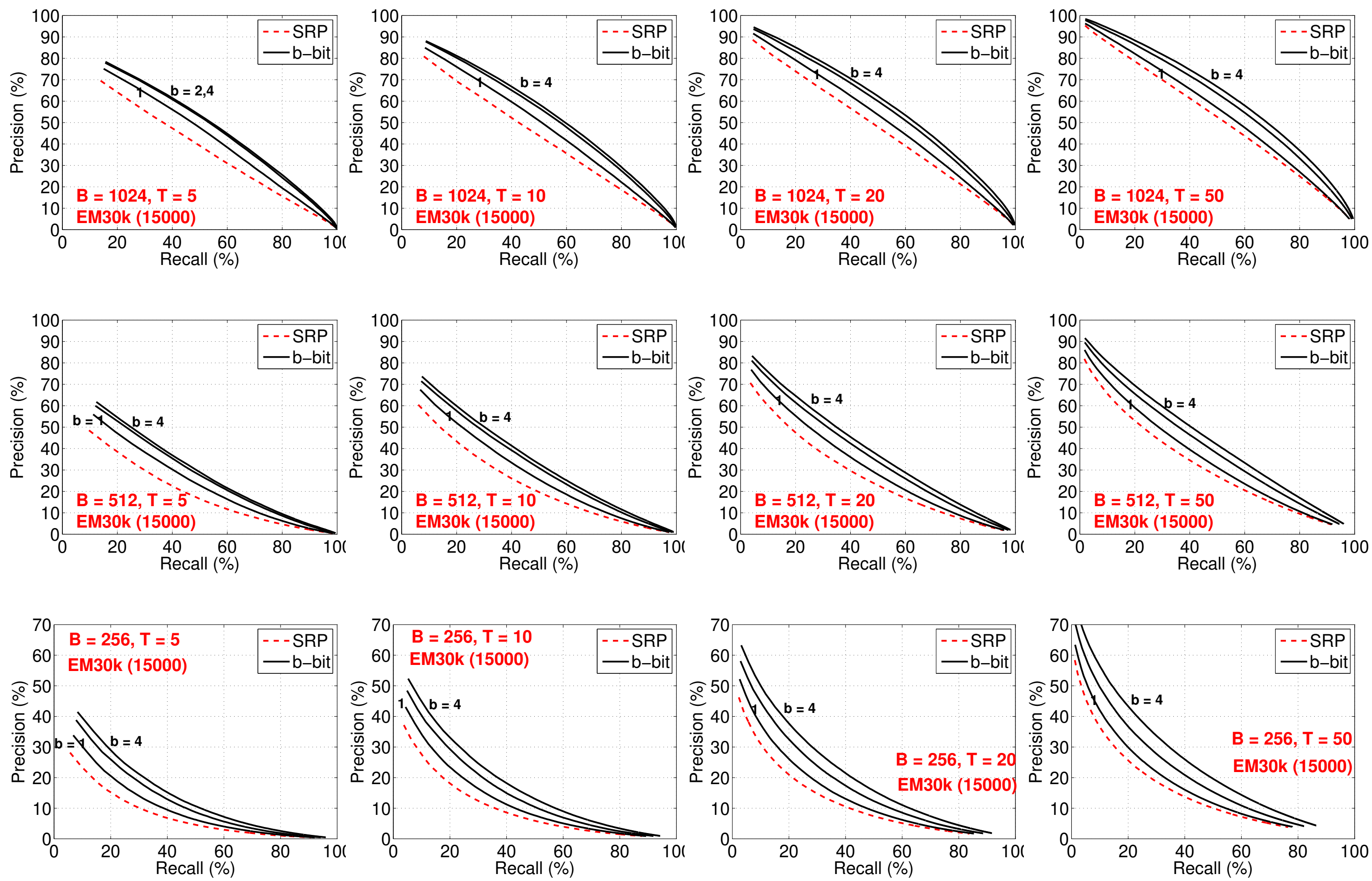
Webspam


SH–NC
SRP
b–bit
Precision (%)
Recall (%)
b = 1
4
b = 1,2
B = 256, T = 5
B = 256, T = 10
B = 256, T = 20
B = 256, T = 50
B = 128, T = 5
B = 128, T = 10
B = 128, T = 20
B = 128, T = 50
B = 64, T = 5
B = 64, T = 10
B = 64, T = 20
B = 64, T = 50
Webspam (35000)

NYTimes


B = 1024, T = 5
NYTimes (20000)
B = 1024, T = 10
NYTimes (20000)
B = 1024, T = 20
NYTimes (20000)
B = 1024, T = 50
NYTimes (20000)
B = 768, T = 5
NYTimes (20000)
B = 768, T = 10
NYTimes (20000)
B = 768, T = 20
NYTimes (20000)
B = 768, T = 50
NYTimes (20000)
B = 512, T = 5
NYTimes (20000)
B = 512, T = 10
NYTimes (20000)
B = 512, T = 20
NYTimes (20000)
B = 512, T = 50
NYTimes (20000)
SH–NC
SH–C
SRP
b–bit
b = 1
b = 4
2
Precision (%)
Recall (%)

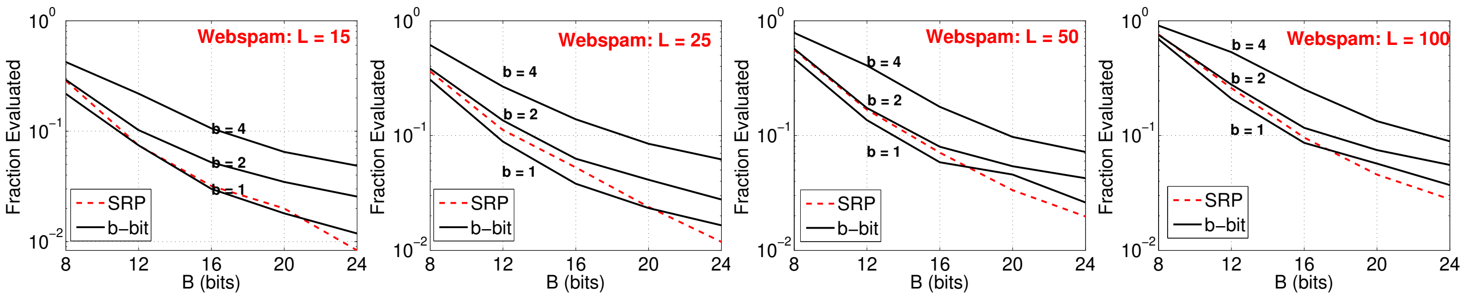
EM30k


B = 1024, T = 5
EM30k (15000)
B = 1024, T = 10
EM30k (15000)
B = 1024, T = 20
EM30k (15000)
B = 1024, T = 50
EM30k (15000)
B = 512, T = 5
EM30k (15000)
B = 512, T = 10
EM30k (15000)
B = 512, T = 20
EM30k (15000)
B = 512, T = 50
EM30k (15000)
B = 256, T = 5
EM30k (15000)
B = 256, T = 10
EM30k (15000)
B = 256, T = 20
EM30k (15000)
B = 256, T = 50
EM30k (15000)
SRP
b–bit
b = 2,4
b = 4
b = 1
1
Precision (%)
Recall (%)

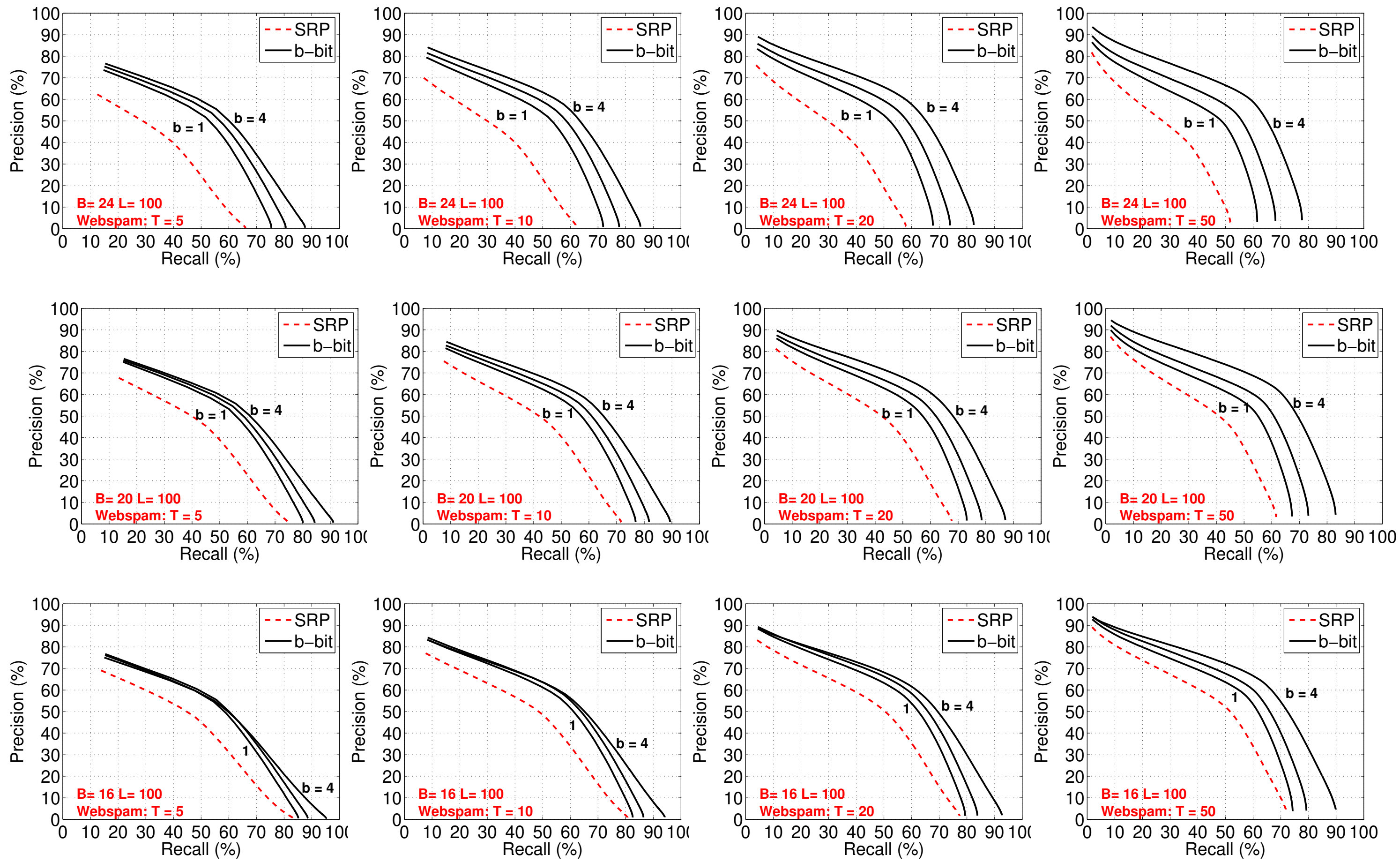
Evaluation 2: Sublinear Near Neighbor Search

- Build hashtables with parameters L and K.
- Retrieve elements for every query point.
- Rank the retrieved candidates based on the total number of signature matches (note we already have precomputed signatures while building hash tables).
- Plot precision-recall curve.
- Plot number of points retrieved.

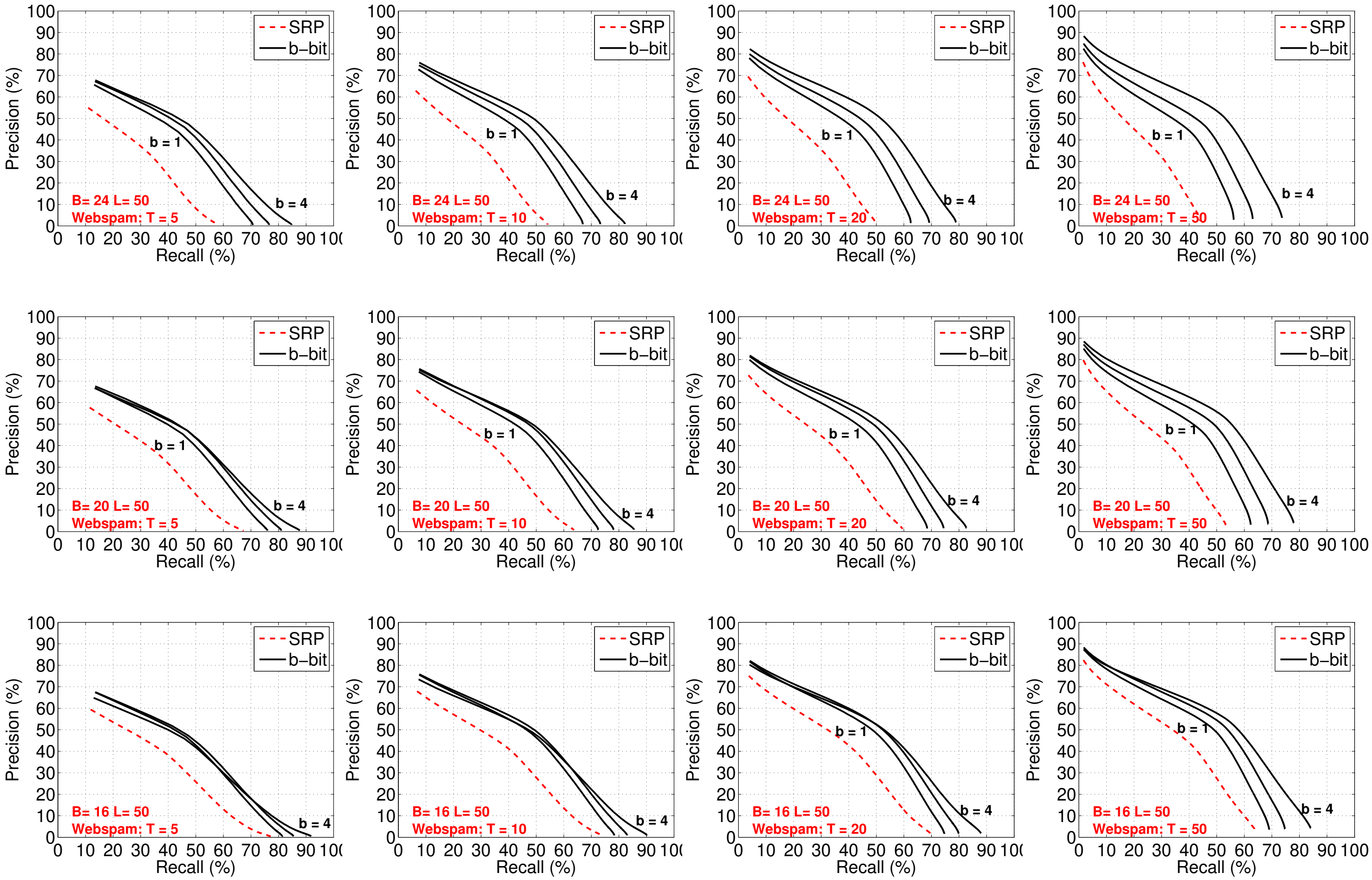
Webspam: Number of Retrieved Points



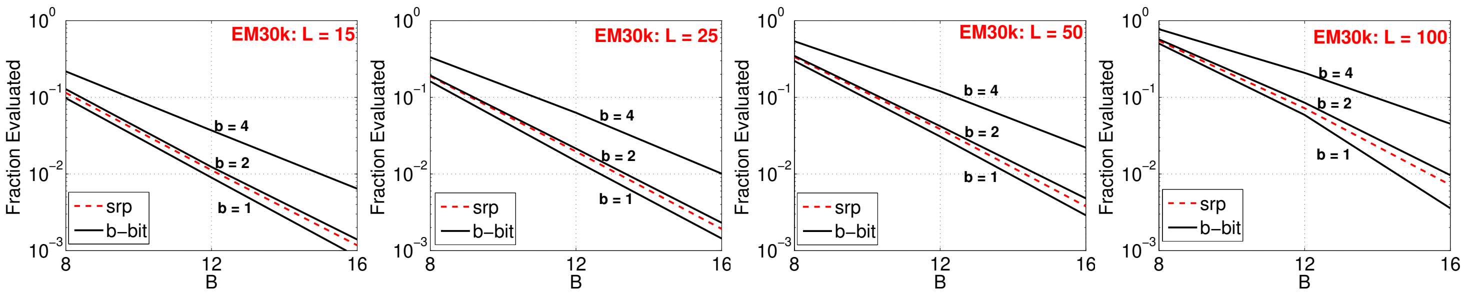
Webspam: Precision-Recall



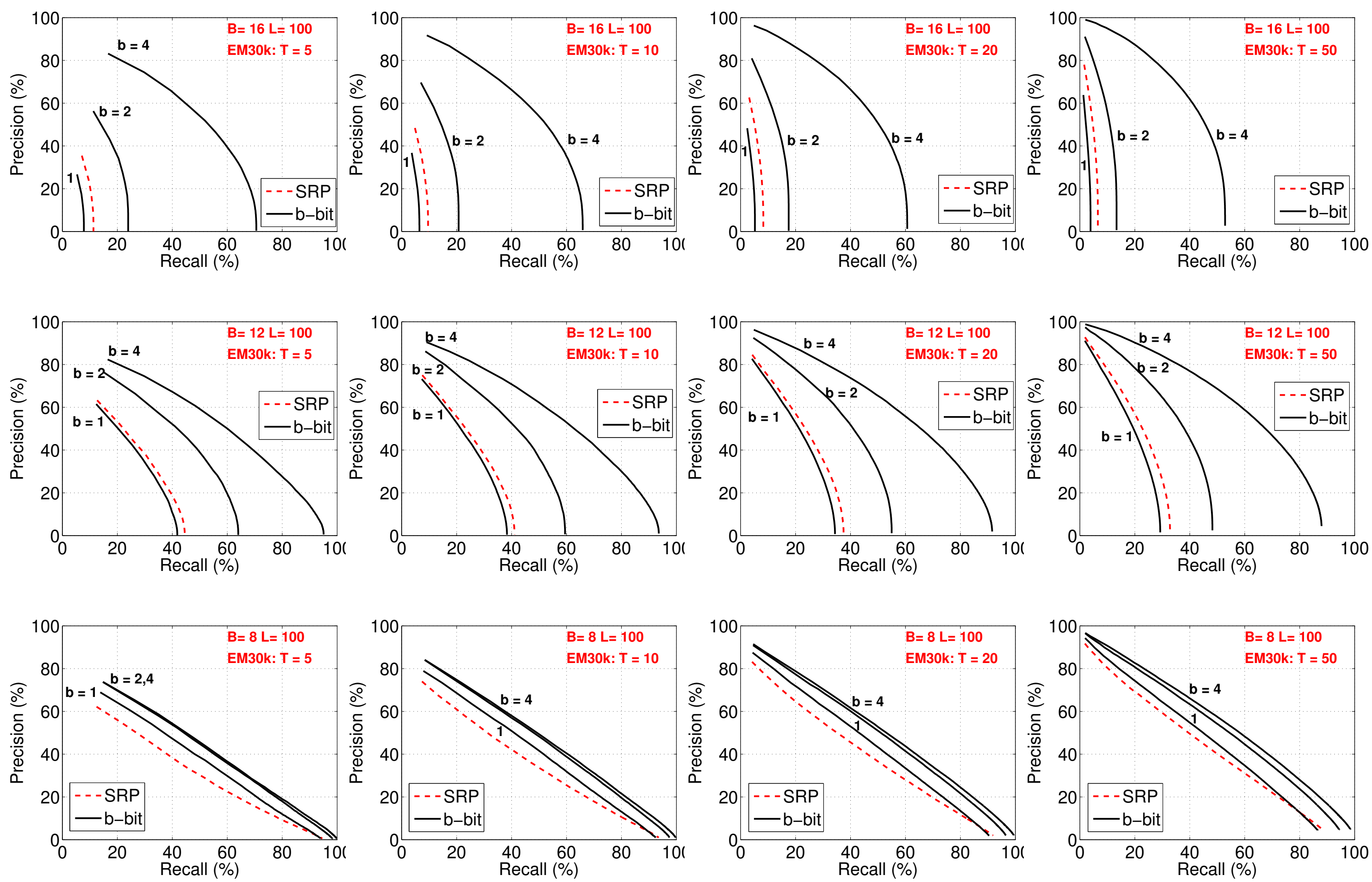
Webspam: Precision-Recall



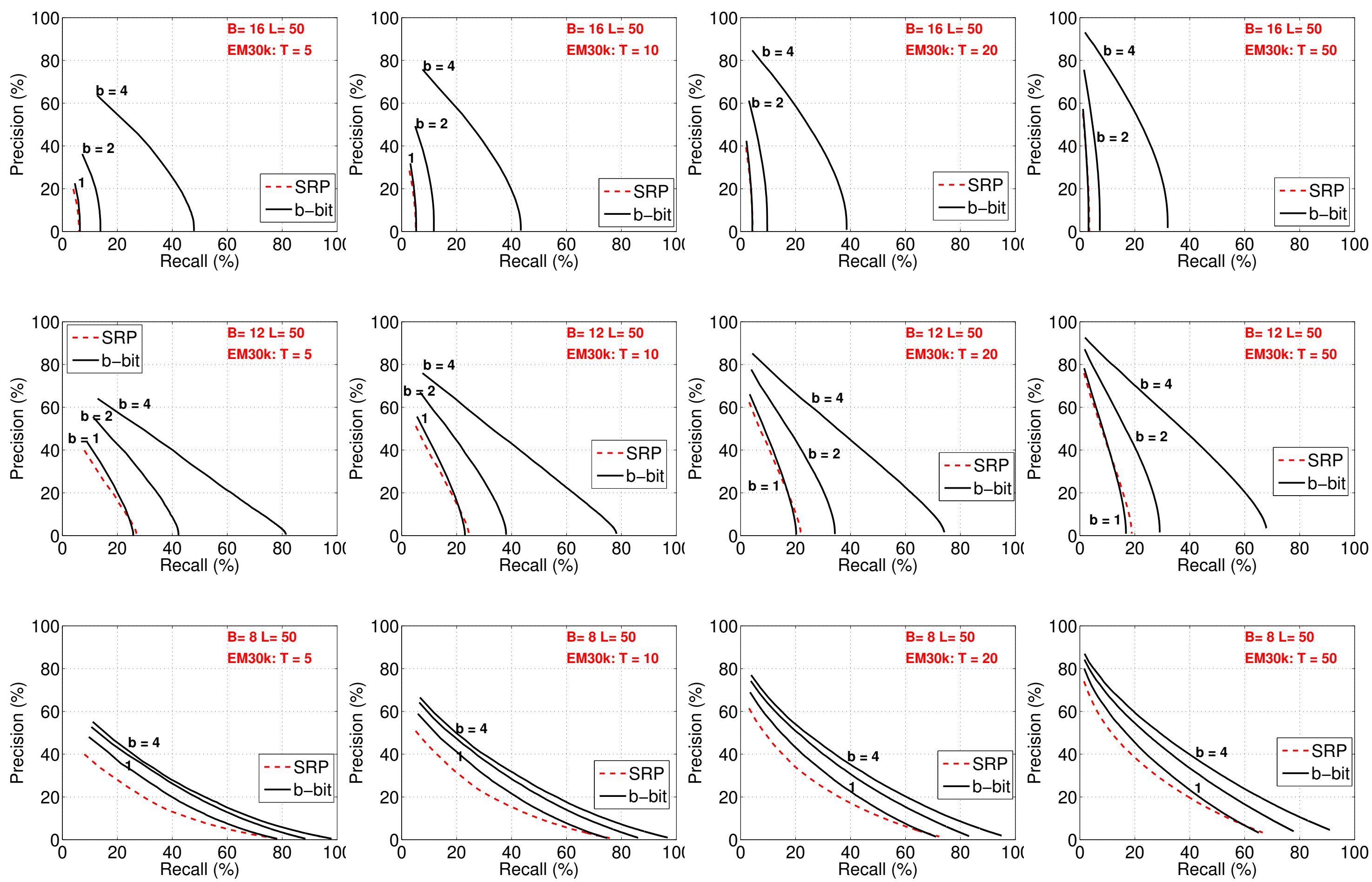
EM30k: Number of Retrieved Points



EM30k: Precision-Recall



EM30k: Precision-Recall



Analysis of b-bit minwise LSH

- b-bit minwise hashing comes with interesting behavior with parameters b, K, L.
- For fixed b, K, to guarantee approximate near neighbor with probability 1 - δ, we need

$$L \geq \frac{\log 1/\delta}{\log\left(\frac{1}{1-P_b^k(R)}\right)}$$

where $P_b(R)$ is collision probability at R.

- Expected fraction of retrieved points with similarity R, assuming uniform distribution over the similarity values is

$$1 - \sum_{i=0}^{L} \binom{L}{i} (-1)^i \frac{1}{2^{bki}} \frac{1}{(2^b-1)R} \frac{\left((2^b-1)R+1\right)^{ki+1} - 1}{ki+1}$$

Fractions Retrieved Plot

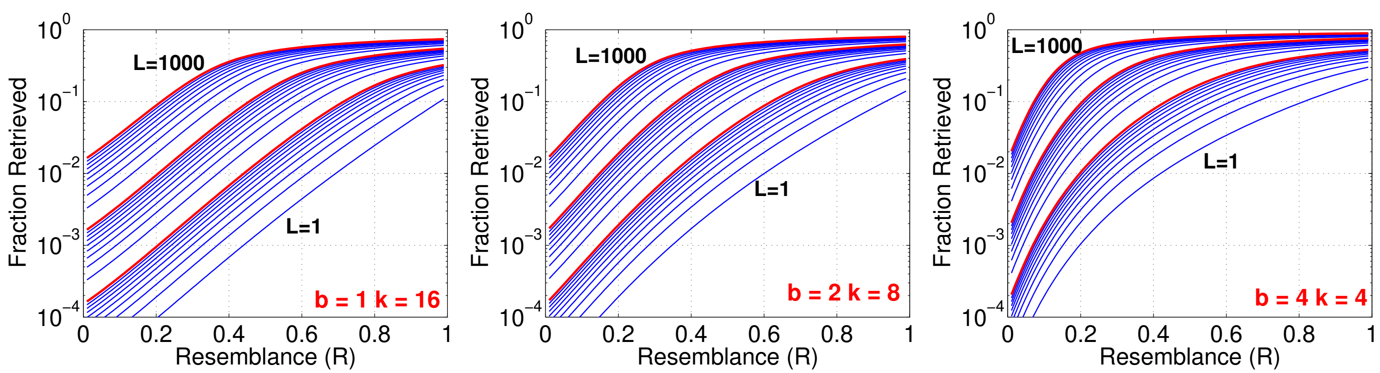


Figure 2: Numerical values for the fraction of retrieved points.

Operating Threshold

- The overall collision probability is

$$P_{b,k,L}(R) = 1 - \left(1 - P_b^k(R)\right)^L$$

- Given b, K, L, the optimum operating point is the point where the rate of change of probability is maximum or where the second derivative vanishes

$$R_0 = \frac{\left(\frac{k-1}{Lk-1}\right)^{1/k} - \frac{1}{2^b}}{1 - \frac{1}{2^b}}$$

Operating Threshold Plot

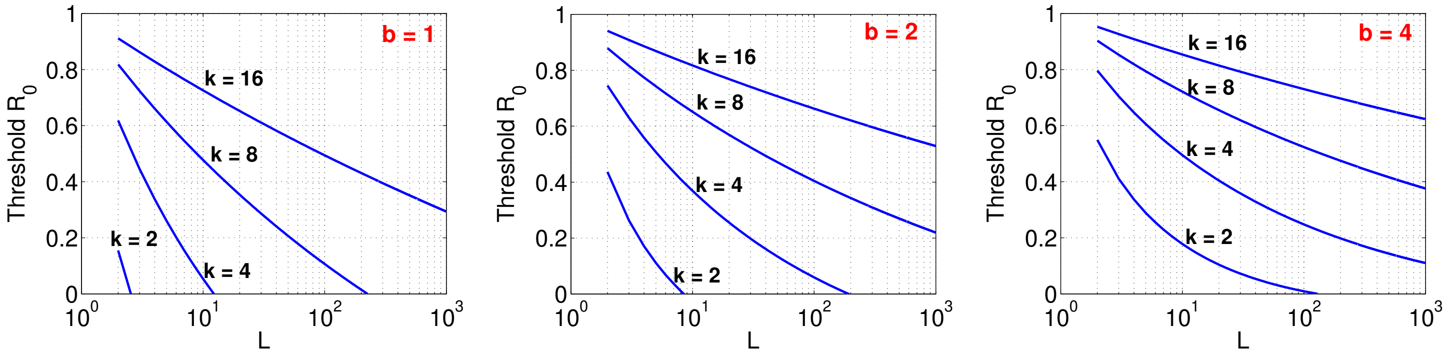


Figure 3: The threshold R_0, i.e., inflection point of $P_{b,k,L}(R)$.

Conclusions

- We present a first study of directly using the bits generated by b-bit minwise hashing to construct hashtables.
- Our proposed scheme is extremely simple and exhibits superb performance compared to two strong baselines: spectral hashing (SH) and sign random projections (SRP).
- The new scheme poses some interesting tradeoffs.

References

- Li, P., Konig, A.C.: b-bit minwise hashing. In: WWW, Raleigh, NC (2010) 671-680.
- Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of ACM 42(6) (1995) 1115-1145.
- Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: STOC, Montreal, Quebec, Canada (2002) 380-388.
- Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Commun. ACM. Volume 51. (2008) 117-122.

Thanks for your attention

Q & A