Asymmetric Minwise Hashing for Indexing Binary Inner Products and Set Containment

Anshumali Shrivastava and Ping Li

Cornell University[1] and Rutgers University

WWW 2015
Florence, Italy

May 21st 2015

[1] **Will Join Rice Univ. as TT Asst. Prof. Fall 2015**

What are we solving ?

Minwise hashing is widely popular for search and retrieval.

Major Complaint: Document length is unnecessarily penalized.

We precisely fix this and provide a practical solution.

Other consequence: Algorithmic improvement for binary maximum inner product search (MIPS).

Outline

- Motivation
- Asymmetric LSH for General Inner Products
- Asymmetric Minwise Hashing
- Faster Sampling
- Experimental Results.

Shingle Based Representation

- Shingle based representation (Bag-of-Words) widely adopted.
- Document is represented as a set of tokens over a vocabulary Ω.

Example Sentence : "Five Kitchen Berkley".
Shingle Representation (Uni-grams): {Five, Kitchen, Berkeley}
Shingle Representation (Bi-grams): {Five Kitchen, Kitchen Berkeley}

Shingle Based Representation

- Shingle based representation (Bag-of-Words) widely adopted.
- Document is represented as a set of tokens over a vocabulary Ω.

Example Sentence : "Five Kitchen Berkley".
Shingle Representation (Uni-grams): {Five, Kitchen, Berkeley}
Shingle Representation (Bi-grams): {Five Kitchen, Kitchen Berkeley}

Sparse Binary High Dimensional Data Everywhere

- Sets can be represented as binary vector indicating presence/absence.
- Vocabulary is typically huge in practice.
- Modern "Big data" systems use only binary data matrix.

Resemblance (Jaccard) Similarity

The popular **resemblance (Jaccard) similarity** between two sets (or binary vectors) $X,\ Y \subset \Omega$ is defined as:

$$\mathcal{R} = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a},$$

where $a = |X \cap Y|$, $f_x = |X|$, $f_y = |Y|$ and $|.|$ denotes the cardinality.

For binary (0/1) vector representation $\iff$ a set (locations of nonzeros).

$$a = |X \cap Y| = x^T y; \quad f_x = nonzeros(x); \quad f_y = nonzeros(y),$$

where x and y are the binary vector equivalents of sets X and Y respectively.

Minwise Hashing (Broder 97)

The standard practice in the search industry:

Given a random permutation π (or a random hash function) over Ω, i.e.,

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

The MinHash is given by

$$h_\pi(x) = \min(\pi(x))$$

An elementary probability argument shows that

$$\mathbf{Pr}\left(\min(\pi(X)) = \min(\pi(Y))\right) = \frac{|X \cap Y|}{|X \cup Y|} = \mathcal{R}.$$

Traditional Minwise Hashing Computation[2]

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	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

[2] This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014

Traditional Minwise Hashing Computation[2]

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	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

	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

[2] This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014

Traditional Minwise Hashing Computation[2]

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	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

	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

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

[2] This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014

Traditional Minwise Hashing Computation[2]

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	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

	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

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

For any two binary vectors S_1, S_2 we always have

$$\mathbf{Pr}\left(h_\pi(S_1) = h_\pi(S_2)\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R \quad (\textbf{Jaccard Similarity.}).$$

[2] This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014

Locality Sensitive Hashing (LSH) and Sub-linear Search

Locality Sensitive : A family (randomized) of hash functions h s.t.

$$Pr_h\big[h(x) = h(y)\big] = f(sim(x, y)),$$

where f is monotonically increasing[3].

MinHash is LSH for Resemblance or Jaccard Similarity

[3] A stronger sufficient condition than the classical one

Locality Sensitive Hashing (LSH) and Sub-linear Search

Locality Sensitive : A family (randomized) of hash functions h s.t.

$$Pr_h\big[h(x) = h(y)\big] = f(sim(x, y)),$$

where f is monotonically increasing[3].

MinHash is LSH for Resemblance or Jaccard Similarity

Well Known: Existence of LSH for a similarity $\implies$ fast search algorithms with query time $O(n^\rho)$, $\rho < 1$ (Indyk & Motwani 98)

The quantity ρ:

- A property dependent f.
- Smaller is better.

[3] A stronger sufficient condition than the classical one

Known Complaints with Resemblance

$$\mathcal{R} = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a},$$

Consider "text" description of two restaurants:

1. "Five Guys Burgers and Fries Downtown Brooklyn New York"
   {**five, guys, burgers, and, fries, downtown, brooklyn, new, york**}
2. "Five Kitchen Berkley"
   {**five, kitchen, berkley**}

Search Query (Q): "Five Guys" {**five, guys**}

Known Complaints with Resemblance

$$\mathcal{R} = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a},$$

Consider "text" description of two restaurants:

1. "Five Guys Burgers and Fries Downtown Brooklyn New York"
   {**five, guys, burgers, and, fries, downtown, brooklyn, new, york**}
2. "Five Kitchen Berkley"
   {**five, kitchen, berkley**}

Search Query (Q): "Five Guys" {**five, guys**}

Resemblance with descriptions:

1. $|X \cap Q| = 2$, $|X \cup Q| = 9$, $\mathcal{R} = \frac{2}{9} = 0.22$
2. $|X \cap Q| = 1$, $|X \cup Q| = 4$, $\mathcal{R} = \frac{1}{4} = 0.25$

Resemblance penalizes the size of the document.

Alternatives: Set containment and Inner Product

For many applications (e.g. record matching, plagiarism detection etc.) Jaccard Containment more suited than Resemblance.

Jaccard Containment w.r.t. Q between X and Q

$$\mathcal{J}_{\mathcal{C}} = \frac{|X \cap Q|}{|Q|} = \frac{a}{f_q}. \tag{1}$$

Some Observations

1. Does not penalize the size of text.
2. Ordering same as the ordering of inner products a (or overlap).
3. Desirable ordering in the previous example.

LSH Framework Not Sufficient for Inner Products

Locality Sensitive Requirement:

$$Pr_h\big[h(x) = h(y)\big] = f(x^T y),$$

where f is monotonically increasing.

Theorem (Shrivastava and Li NIPS 2014): Impossible for dot products

- For inner products, we can have x and y, s.t. $x^T y > x^T x$.
  Self similarity is not the highest similarity.
- Under any hash function $Pr(h(x) = h(x)) = 1$. But we need

$$Pr(h(x) = h(y)) > Pr(h(x) = h(x)) = 1$$

LSH Framework Not Sufficient for Inner Products

Locality Sensitive Requirement:

$$Pr_h\big[h(x) = h(y)\big] = f(x^T y),$$

where f is monotonically increasing.

Theorem (Shrivastava and Li NIPS 2014): Impossible for dot products

- For inner products, we can have x and y, s.t. $x^T y > x^T x$**.**
  Self similarity is not the highest similarity.
- Under any hash function $Pr(h(x) = h(x)) = 1$. But we need

$$Pr(h(x) = h(y)) > Pr(h(x) = h(x)) = 1$$

For Binary Inner Products : Still Impossible

- $x^T y \leq x^T x$ is always true.
- We instead need $x,\ y,\ z$ such that $x^T y > z^T z$

LSH Framework Not Sufficient for Inner Products

Locality Sensitive Requirement:

$$Pr_h\big[h(x) = h(y)\big] = f(x^T y),$$

where f is monotonically increasing.

Theorem (Shrivastava and Li NIPS 2014): Impossible for dot products

- For inner products, we can have x and y, s.t. $x^T y > x^T x$.
  Self similarity is not the highest similarity.
- Under any hash function $Pr(h(x) = h(x)) = 1$. But we need

$$Pr(h(x) = h(y)) > Pr(h(x) = h(x)) = 1$$

For Binary Inner Products : Still Impossible

- $x^T y \leq x^T x$ is always true.
- We instead need $x,\ y,\ z$ such that $x^T y > z^T z$

Hopeless to find Locality Sensitive Hashing!

Asymmetric LSH (ALSH) for General Inner Products

Shrivastava and Li (NIPS 2014): Despite no LSH, **Maximum Inner Product Search (MIPS)** is still efficient via an extended framework

Asymmetric LSH (ALSH) for General Inner Products

Shrivastava and Li (NIPS 2014): Despite no LSH, **Maximum Inner Product Search (MIPS)** is still efficient via an extended framework

Asymmetric LSH Framework : Idea

1. Construct two transformations $P(.)$ and $Q(.)$ ($P \neq Q$) along with a randomized hash functions h.
2. $P(.)$, $Q(.)$ and h satisfies

$$Pr_h\left[h(P(x)) = h(Q(q))\right] = f(x^T q), \quad \text{f is monotonic}$$

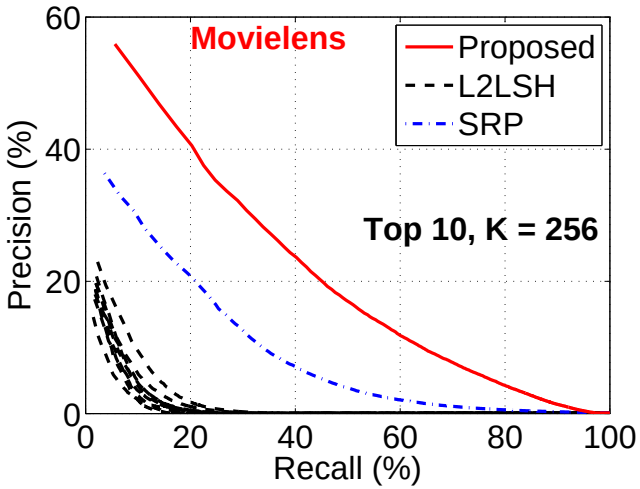
Small things that made BIG difference

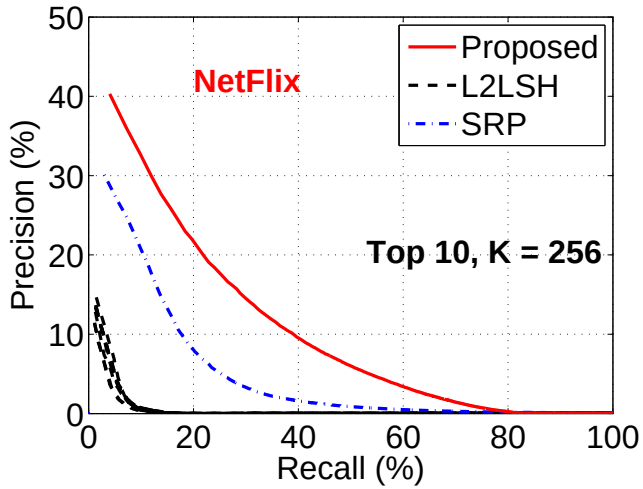
Shrivastava and Li NIPS 2014 construction (L2-ALSH)

1. **P(x): Scale** data to shrink norms < 0.83. **Append** $||x||^2$, $||x||^4$, and $||x||^8$ to vector x. (just 3 scalars)
2. **Q(q): Normalize**. **Append** three 0.5 to vector q.
3. **h:** Use standard LSH family for L_2 distance.

Caution: Scaling is asymmetry in strict sense, it changes the distribution (e.g. variance) of hashes.

First Practical and Provable Algorithm for General MIPS :





A Generic Recipe : Even better ALSH for MIPS

The Recipe:

- Start with a similarity $\mathcal{S}'(q, x)$ for which we have an LSH (or ALSH).
- Design $P(.)$ and $Q(.)$, such that $\mathcal{S}'(Q(q), P(x))$ is monotonic in $q^T x$
- Use extra dimensions.

Improved ALSH (Sign-ALSH) Construction for General MIPS

$\mathcal{S}'(q, x) = \frac{q^T x}{||q||_2 ||x||_2}$ and Simhash[4]. (**Shrivastava and Li UAI 2015**)

	Sign-ALSH	L2-ALSH	Cone Trees
MNIST	**7,944**	9,971	11,202
WEBSPAM	**2,866**	3,813	22,467
RCV1	**9,951**	11,883	38,162

[4]Expected after Shrivastava and Li ICML 2014 "Codings for Random Projections"

Binary MIPS: A Sampling based ALSH

Idea: Sample index i, if $x_i = 1$ and $q_i = 1$, make hash collision, else not.

$$\mathcal{H}_S(f(x)) = \begin{cases} 0 & \text{if } x_i = 1,\ i \text{ drawn uniformly} \\ 1 & \text{if f = Q (for query)} \\ 2 & \text{if f = P (while preprocessing)} \end{cases}$$

$$Pr(\mathcal{H}_S(P(x)) = \mathcal{H}_S(Q(y))) = \frac{a}{D},$$

$\frac{a}{D}$ is **monotonic in inner product** a**.**

Problems:

1. Only informative if $x_i = 1$, else hash just indicates query or not.
2. Sparse data, with $D \gg f$, $\frac{a}{D} \simeq 0$, almost all hashes are un-informative.

A Closer Look at MinHash

Collision Probability:

$$Pr(h_\pi(x) = h_\pi(q)) = \frac{a}{f_x + f_q - a} \gg \frac{a}{D} \simeq 0$$

Useful: $\frac{a}{f_x+f_q-a}$ very sensitive w.r.t a compared to $\frac{a}{D}$. $(D \gg f)$
The core reason why MinHash is better than random sampling.

Problem: $\frac{a}{f_x+f_q-a}$ is not monotonic in a (inner product).
Not LSH for binary inner product. (Though a good heuristic !)

Why we are biased in favor of MinHash ? :

- **SL "In defense of MinHash over Simhash" AISTATS 2014** $\Longrightarrow$ For binary data MinHash is provably superior than SimHash.
- Already some hope to beat state-of-the-art Sign-ALSH for Binary Data

The Fix: Asymmetric Minwise Hashing

Let M be the maximum sparsity of the data vectors.

$$M = \max_{x \in \mathcal{C}} |x|$$

Define $P : [0,1]^D \rightarrow [0,1]^{D+M}$ and $Q : [0,1]^D \rightarrow [0,1]^{D+M}$ as:

$$P(x) = [x; 1; 1; 1; ...; 1; 0; 0; ...; 0] \quad M - f_x \text{ 1s and } f_x \text{ zeros}$$
$$Q(q) = [x; 0; 0; 0; ...; 0], \quad \text{M zeros}$$

The Fix: Asymmetric Minwise Hashing

Let M be the maximum sparsity of the data vectors.

$$M = \max_{x \in \mathcal{C}} |x|$$

Define $P : [0,1]^D \rightarrow [0,1]^{D+M}$ and $Q : [0,1]^D \rightarrow [0,1]^{D+M}$ as:

$$P(x) = [x; 1; 1; 1; ...; 1; 0; 0; ...; 0] \quad M - f_x \text{ 1s and } f_x \text{ zeros}$$
$$Q(q) = [x; 0; 0; 0; ...; 0], \quad \text{M zeros}$$

After Transformation:

$$R' = \frac{|P(x) \cap Q(q)|}{|P(x) \cup Q(q)|} = \frac{a}{M + f_q - a}, \quad \textbf{monotonic in the inner product } a$$

Also, $M + f_q - a \ll D$ (M of order of sparsity, handle outliers separately.)

Note : To get rid of f_q change $P(.)$ to $P(Q(.))$ and $Q(.)$ to $Q(P(.))$.

Asymmetric Minwise Hashing: Alternative View

$$P'(x) = [x; M - f_x; 0]$$
$$Q'(x) = [x; 0; M - f_x]$$

The weighted Jaccard between $P'(x)$ and $Q'(q)$ is

$$\mathcal{R}_W = \frac{\sum_i \min(P'(x)_i, Q'(q)_i)}{\sum_i \max(P'(x)_i, Q'(q)_i)} = \frac{a}{2M - a}.$$

Fast Consistent Weighted Sampling (CWS) to get asymmetric MinHash in $O(f_x)$ time instead of $O(2M)$ where $M \geq f_x$.

Asymmetric Minwise Hashing: Alternative View

$$P'(x) = [x; M - f_x; 0]$$
$$Q'(x) = [x; 0; M - f_x]$$

The weighted Jaccard between $P'(x)$ and $Q'(q)$ is

$$\mathcal{R}_W = \frac{\sum_i \min(P'(x)_i, Q'(q)_i)}{\sum_i \max(P'(x)_i, Q'(q)_i)} = \frac{a}{2M - a}.$$

Fast Consistent Weighted Sampling (CWS) to get asymmetric MinHash in $O(f_x)$ time instead of $O(2M)$ where $M \geq f_x$.

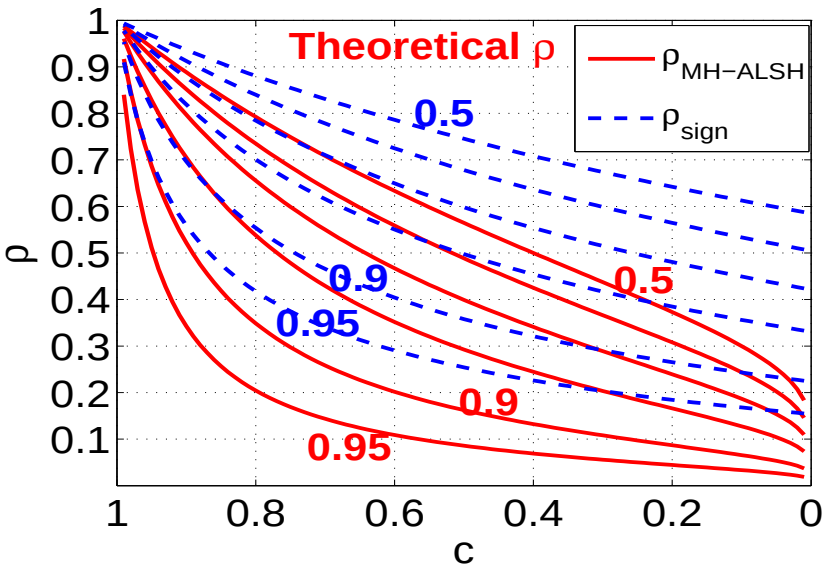
Alternative View:

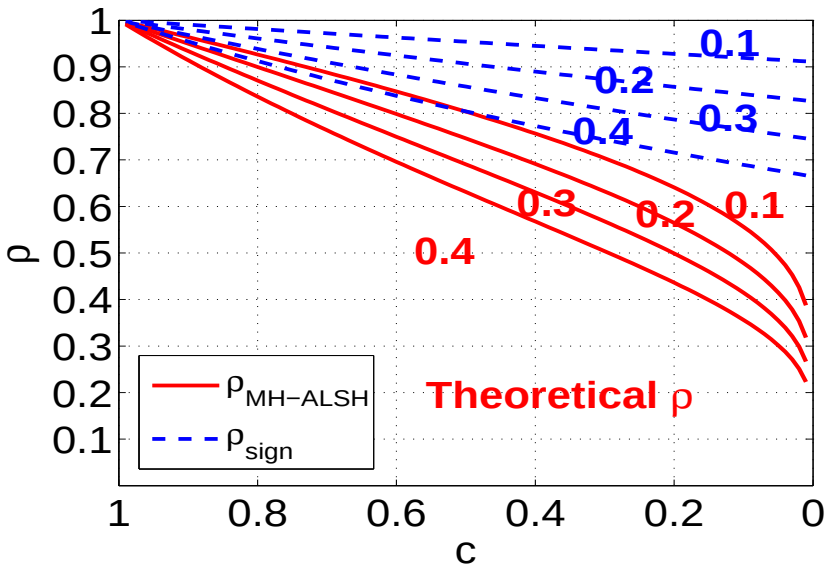
- $M - f_x$ favors larger documents in proportion to $-f_x$, which cancels the inherent bias of minhash towards smaller set.
- A novel bias correction, which works well in practice.

Theoretical Comparisons

Collision probability monotonic in inner product $\Longrightarrow$ **asymmetric minwise hashing is an ALSH for binary MIPS.**

$$\rho_{MH-ALSH} = \frac{\log \frac{S_0/M}{2-S_0/M}}{\log \frac{cS_0/M}{2-cS_0/M}}; \quad \rho_{Sign} = \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0}{M}\right)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{cS_0}{M}\right)\right)}$$

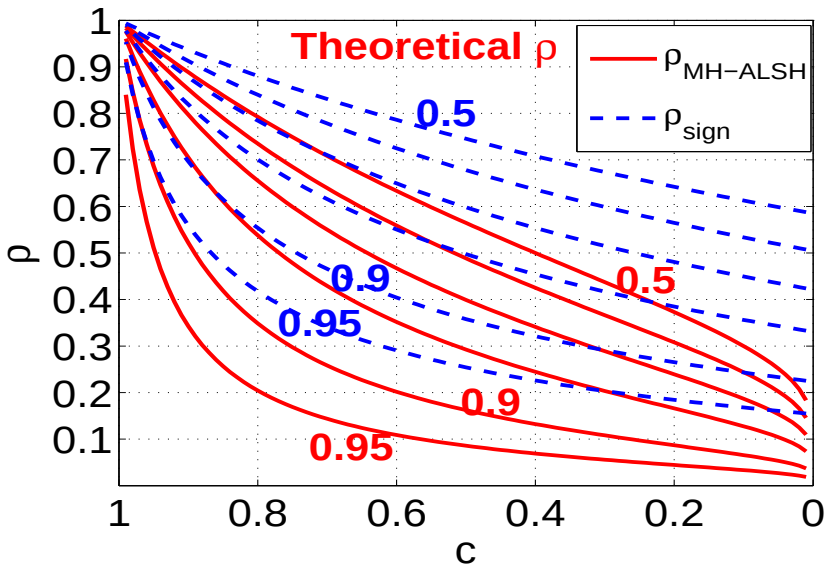


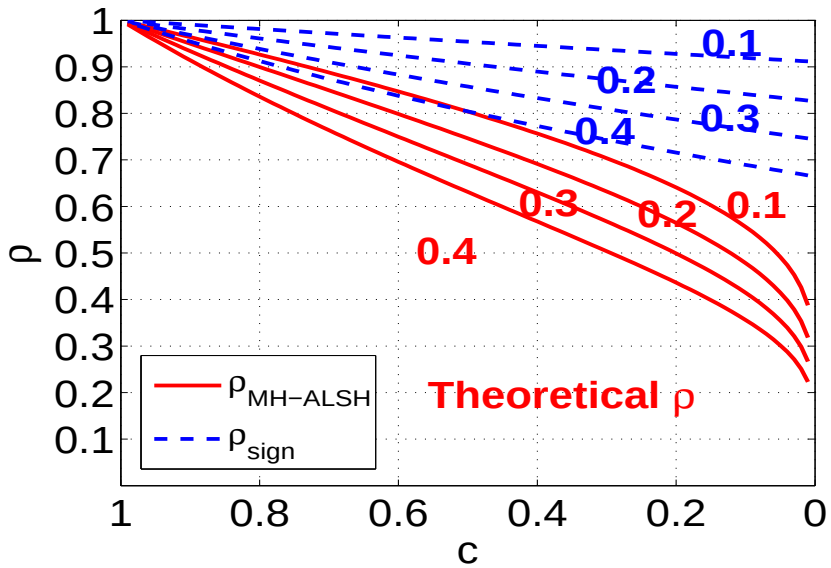


Theoretical Comparisons

Collision probability monotonic in inner product $\Longrightarrow$ **asymmetric minwise hashing is an ALSH for binary MIPS.**

$$\rho_{MH-ALSH} = \frac{\log \frac{S_0/M}{2-S_0/M}}{\log \frac{cS_0/M}{2-cS_0/M}}; \quad \rho_{Sign} = \frac{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{S_0}{M}\right)\right)}{\log\left(1 - \frac{1}{\pi}\cos^{-1}\left(\frac{cS_0}{M}\right)\right)}$$





Asymmetric Minwise Hashing is significantly better than **Sign-ALSH (SL UAI 2015)** (**Expected after SL AISTATS 2014**)

Complaints with MinHash: Costly Sampling

	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

Complaints with MinHash: Costly Sampling

	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

	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

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

Complaints with MinHash: Costly Sampling

	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

	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

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

Process the entire vector to compute one minhash $O(d)$.

- Search time is dominated by the hashing query. $O(KLd)$
- Training and Testing time dominated by the hashing time. $O(kd)$

Parallelization possible but not energy efficient. (L**S**K **WWW 2012**)

Complaints with MinHash: Costly Sampling

	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

	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

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

Process the entire vector to compute one minhash $O(d)$.

- Search time is dominated by the hashing query. $O(KLd)$
- Training and Testing time dominated by the hashing time. $O(kd)$

Parallelization possible but not energy efficient. (L**S**K **WWW 2012**)

Storing only the minimum seems quite wasteful.

Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

$\pi(S_1)$	**0000 0101 0000 0011 1010 0110**
$\pi(S_2)$	**0000 0111 0000 1010 1100 0000**

Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0

Solution: One Pass for All hashes

1. Sketching: Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0

Solution: One Pass for All hashes

1. Sketching: Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

2. **Fill Empty Bins:** Borrow from right (circular) with shift.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$H(S_1)$	E	1	E	2	0	1
$H(S_2)$	E	1	E	0	0	E

Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

2. **Fill Empty Bins:** Borrow from right (circular) with shift.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$H(S_1)$	E ←	1	E	2	0	1
$H(S_2)$	E	1	E	0	0	E

Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

2. **Fill Empty Bins:** Borrow from right (circular) with shift.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$H(S_1)$	1+C ←	1	E	2	0	1
$H(S_2)$	E	1	E	0	0	E

Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

2. **Fill Empty Bins:** Borrow from right (circular) with shift.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$H(S_1)$	1+C ← 1		2+C ← 2		0	1
$H(S_2)$	1+C ← 1		0+C ← 0		0	E

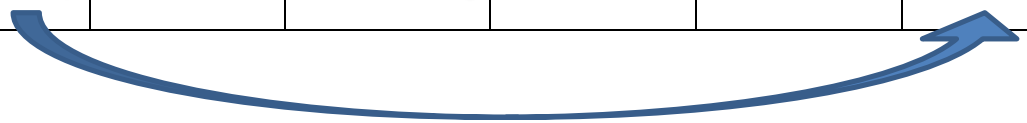
Solution: One Pass for All hashes

1. **Sketching:** Bin and compute minimum non-zero index in each bin.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$\pi(S_1)$	0 0 0 0	0 1 0 1	0 0 0 0	0 0 1 1	1 0 1 0	0 1 1 0
$\pi(S_2)$	0 0 0 0	0 1 1 1	0 0 0 0	1 0 1 0	1 1 0 0	0 0 0 0
OPH(S_1)	E	1	E	2	0	1
OPH(S_2)	E	1	E	0	0	E

2. **Fill Empty Bins:** Borrow from right (circular) with shift.

	Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5
$H(S_1)$	1+C ← 1		2+C ← 2		0	1
$H(S_2)$	1+C ← 1		0+C ← 0		0	1+2C



- $\mathbf{Pr}\left(\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)\right) = R$ for $i = \{0, 1, 2..., k\}$
- $O(d + k)$ **instead of traditional** $O(dk)$ **!**

Speedup

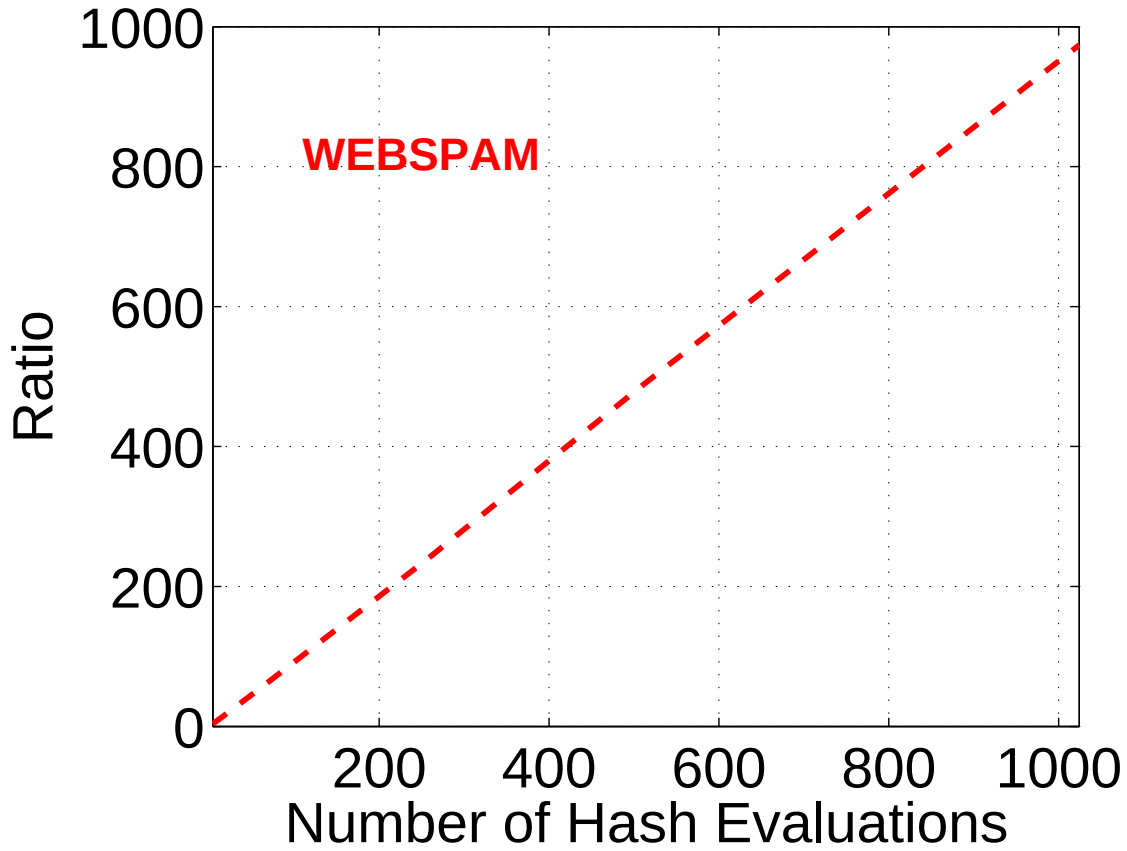


Figure: Ratio of old and new hashing time indicates a linear time speedup

Datasets and Baselines

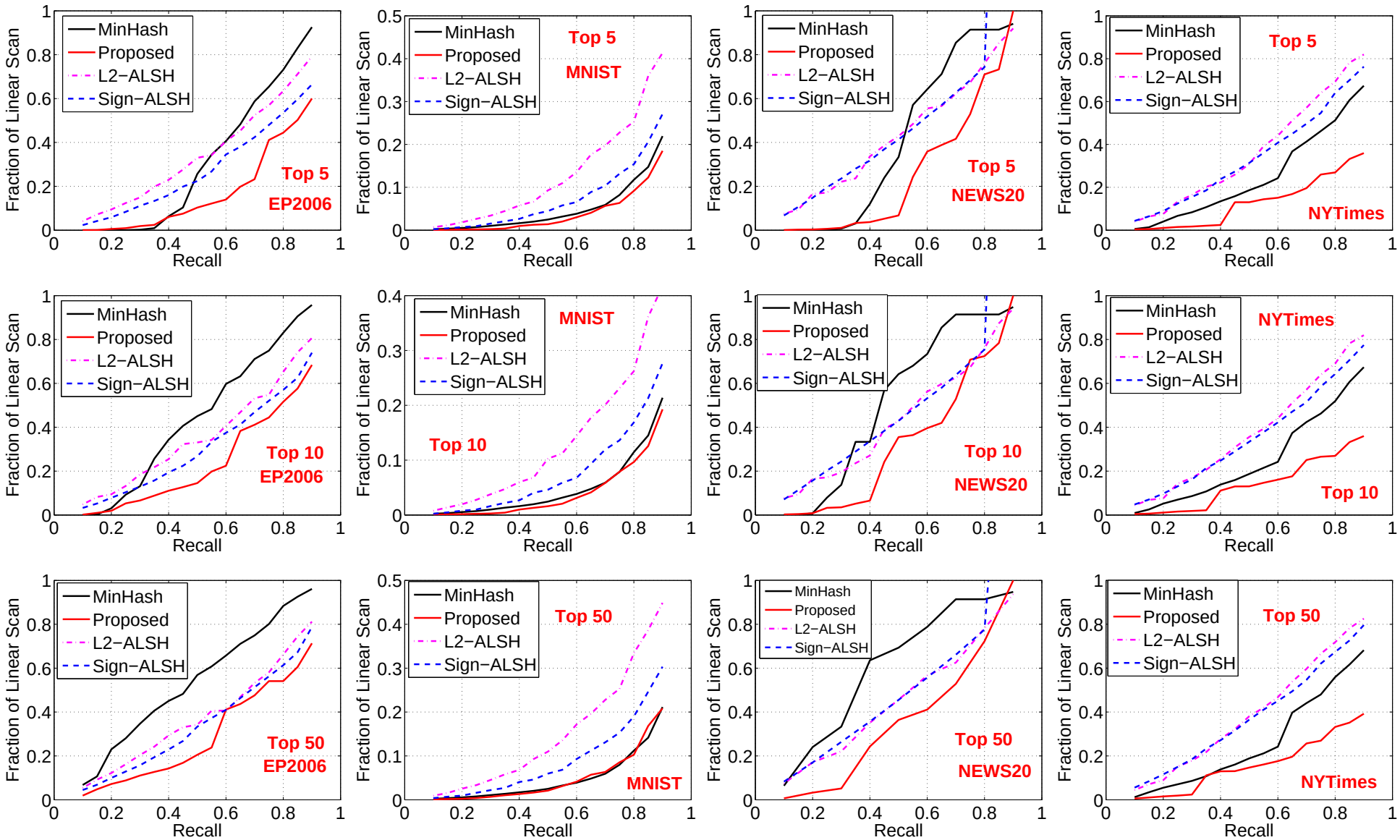
Table: Datasets

Dataset	# Query	# Train	# Dim	nonzeros (mean $\pm$ std)
EP2006	2,000	17,395	4,272,227	6072 $\pm$ 3208
MNIST	2,000	68,000	784	150 $\pm$ 41
NEWS20	2,000	18,000	1,355,191	454 $\pm$ 654
NYTIMES	2,000	100,000	102,660	232 $\pm$ 114

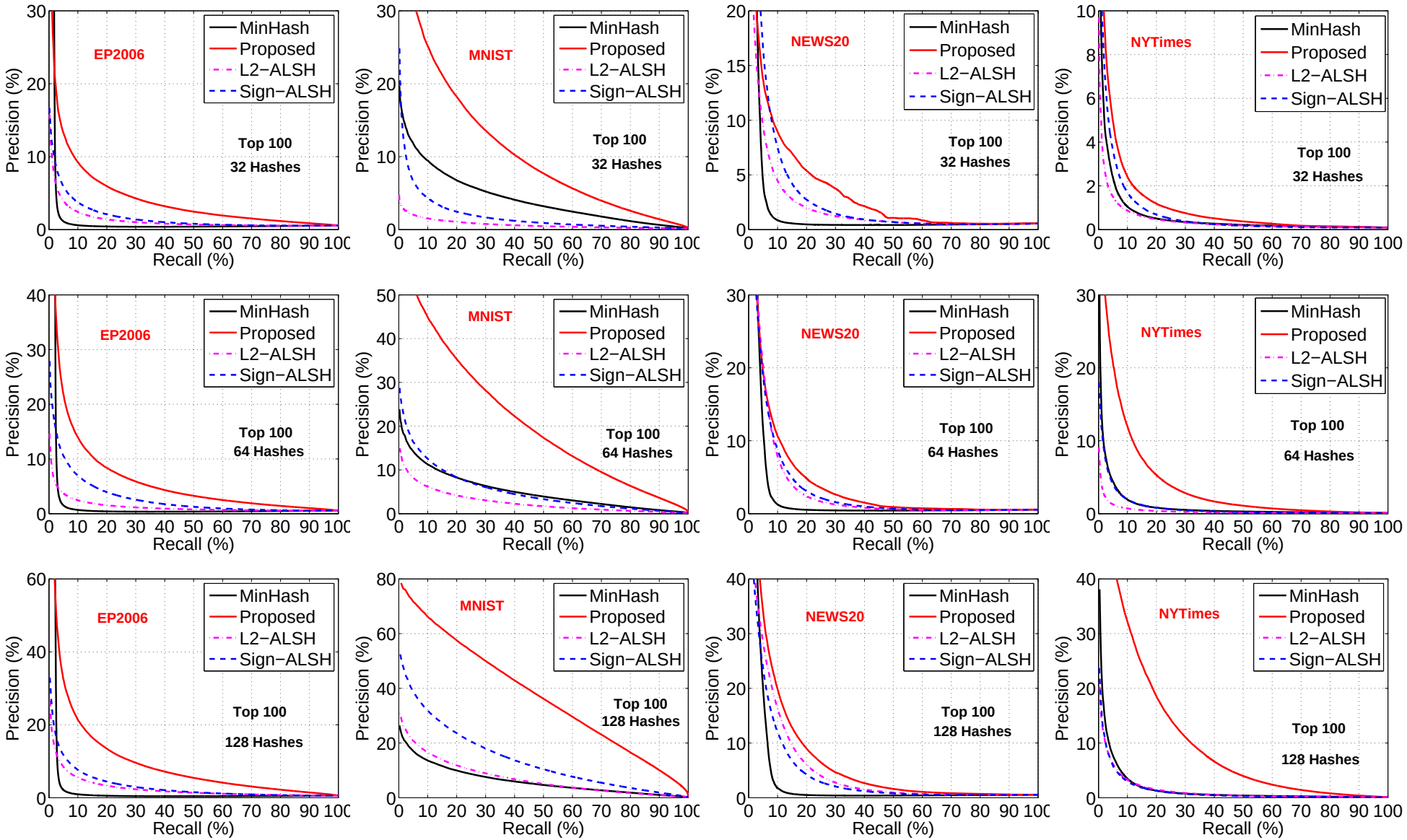
Competing Schemes

1. Asymmetric minwise hashing (Proposed)
2. Traditional minwise hashing (MinHash)
3. L2 based Asymmetric LSH for Inner products (L2-ALSH)
4. SimHash based Asymmetric LSH for Inner Products (Sign-ALSH)

Actual Savings in Retrieval



Ranking Verification



Conclusions

- Minwise hashing has inherent bias towards smaller sets.
- Using the recent line of work on asymmetric LSH, we can fix the existing bias using asymmetric transformations.
- Asymmetric minwise hashing leads to an algorithmic improvement over state-of-the-art hashing scheme for binary MIPS .
- We can obtain huge speedups using recent line of work on one permutation hashing.
- The final algorithm performs very well in practice compared to popular schemes.

References

Asymmetric LSH framework and improvements.

- Shrivastava & Li "Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS)". **NIPS 2014**
- Shrivastava & Li "Improved Asymmetric Locality Sensitive Hashing (ALSH) for Maximum Inner Product Search (MIPS)". **UAI 2015**

Efficient replacements of minwise hashing.

- Li et. al. "One Permutation Hashing" **NIPS 2012**
- Shrivastava & Li "Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search". **ICML 2014**
- Shrivastava & Li "Improved Densification of One Permutation Hashing." **UAI 2014**

Minwise hashing is superior to SimHash for binary data.

- Shrivastava & Li "In Defense of MinHash over SimHash" **AISTATS 2014**