Asymmetric Minwise Hashing for Indexing Binary Inner Products and Set Containment

Anshumali Shrivastava and Ping Li

Cornell University\textsuperscript{1} and Rutgers University

WWW 2015
Florence, Italy

May 21st 2015

\textsuperscript{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 $\Omega$.

**Example Sentence**: “Five Kitchen Berkley”.

**Shingle Representation (Uni-grams)**: \{Five, Kitchen, Berkeley\}

**Shingle Representation (Bi-grams)**: \{Five Kitchen, Kitchen Berkeley\}
Shingle based representation (Bag-of-Words) widely adopted.

Document is represented as a set of tokens over a vocabulary $\Omega$.

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).

\[
a = |X \cap Y| = x^T y; \quad f_x = \text{nonzeros}(x); \quad f_y = \text{nonzeros}(y),
\]

where \(x\) and \(y\) are the binary vector equivalents of sets \(X\) and \(Y\) respectively.
The standard practice in the search industry:

Given a random permutation \( \pi \) (or a random hash function) over \( \Omega \), i.e.,

\[
\pi : \Omega \longrightarrow \Omega, \quad \text{where} \quad \Omega = \{0, 1, \ldots, D - 1\}.
\]

The MinHash is given by

\[
h_\pi(x) = \min(\pi(x))
\]

An elementary probability argument shows that

\[
\Pr(\min(\pi(X)) = \min(\pi(Y))) = \frac{|X \cap Y|}{|X \cup Y|} = R.
\]
1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under $\pi$.
3. The hash value is **smallest index which is not zero**.

For any two binary vectors $S_1, S_2$ we always have $\Pr(h_\pi(S_1) = h_\pi(S_2)) = |S_1 \cap S_2| / |S_1 \cup S_2| = R(Jaccard Similarity)$.

This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014
Traditional Minwise Hashing Computation

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

\begin{align*}
S_1: & \quad 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
\pi(S_1): & \quad 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \\
S_2: & \quad 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\
\pi(S_2): & \quad 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \\
S_3: & \quad 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\
\pi(S_3): & \quad 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0
\end{align*}

\textsuperscript{2}This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014
1. Uniformly sample a permutation over attributes \( \pi : [0, D] \mapsto [0, D] \).
2. Shuffle the vectors under \( \pi \).
3. The hash value is \textit{smallest index which is not zero}.

\[
\begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
S_1: & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
S_3: & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\quad
\begin{array}{cccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\pi(S_1): & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(S_2): & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\pi(S_3): & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[h_\pi(S_1) = 2, \quad h_\pi(S_2) = 0, \quad h_\pi(S_3) = 0\]
Traditional Minwise Hashing Computation

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

For any two binary vectors $S_1, S_2$ we always have

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

---

$^2$This is very inefficient, we recently found faster ways ICML 2014 and UAI 2014
Locality Sensitive: A family (randomized) of hash functions $h$ s.t.

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

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.

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

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 $\rho$:**

- 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.
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 $Q$

$$J_C = \frac{|X \cap Q|}{|Q|} = \frac{a}{f_q}. \quad (1)$$

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:**

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

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
  \[
  Pr(h(x) = h(y)) > Pr(h(x) = h(x)) = 1
  \]
**Locality Sensitive Requirement:**

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

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 \( x^T y > z^T z \)
Locality Sensitive Requirement:

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

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 \( Pr(h(x) = h(y)) = 1 \). 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!
Shrivastava and Li (NIPS 2014): Despite no LSH, Maximum Inner Product Search (MIPS) is still efficient via an extended framework
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

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

![Graphs showing Recall and Precision for MovieLens and NetFlix datasets with Top 10, K = 256]
A Generic Recipe: Even better ALSH for MIPS

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

Improved ALSH (Sign-ALSH) Construction for General MIPS

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sign-ALSH</th>
<th>L2-ALSH</th>
<th>Cone Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>7,944</td>
<td>9,971</td>
<td>11,202</td>
</tr>
<tr>
<td>WEBSPAM</td>
<td>2,866</td>
<td>3,813</td>
<td>22,467</td>
</tr>
<tr>
<td>RCV1</td>
<td>9,951</td>
<td>11,883</td>
<td>38,162</td>
</tr>
</tbody>
</table>

$^4$Expected after Shrivastava and Li ICML 2014 “Codings for Random Projections”
**Idea:** Sample index $i$, if $x_i = 1$ and $q_i = 1$, make hash collision, else not.

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

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

$\frac{a}{D}$ 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 $D \gg f$, $\frac{a}{D} \approx 0$, 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 \( \Rightarrow \)
  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:

$P(x) = [x; 1; 1; 1; \ldots; 1; 0; 0; \ldots; 0]$ $M - f_x$ 1s and $f_x$ zeros

$Q(q) = [x; 0; 0; 0; \ldots; 0]$, $M$ 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:

$$P(x) = [x; 1; 1; 1; \ldots; 1; 0; 0; \ldots; 0] \quad M - f_x \ 1s \ and \ f_x \ zeros$$

$$Q(q) = [x; 0; 0; 0; \ldots; 0], \ M \ zeros$$

After Transformation:

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

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 $Q(P(.))$. 

Anshumali Shrivastava and Ping Li
WWW 2015
May 21st 2015 17 / 27
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

\[
R_W = \frac{\sum_i \min(P'(x)_i, Q'(q)_i)}{\sum_i \max(P'(x)_i, Q'(q)_i)} = \frac{a}{2M - a}.
\]

**Fast Consistent Weighted Sampling (CWS)** to get asymmetric MinHash in \( O(f_x) \) time instead of \( O(2M) \) where \( M \geq f_x \).
$P'(x) = [x; M - f_x; 0]$

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

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

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

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

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

Asymmetric Minwise Hashing is significantly better than Sign-ALSH (SL UAI 2015) (Expected after SL AISTATS 2014)
Complaints with MinHash: Costly Sampling

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$:</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$S_2$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$S_3$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Search time is dominated by the hashing query, which takes $O(K d)$. 

Training and testing time are dominated by the hashing time, also taking $O(K d)$. 

Parallelization is possible but not energy efficient. (L斯基 WWWW 2012) 

Storing only the minimum seems quite wasteful.

Anshumali Shrivastava and Ping Li

WWW 2015
May 21st 2015
20 / 27
Complaints with MinHash: Costly Sampling

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$:</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_2$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$S_3$:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(S_1)$:</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\pi(S_2)$:</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$\pi(S_3)$:</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$h_\pi(S_1) = 2, \quad h_\pi(S_2) = 0, \quad h_\pi(S_3) = 0$
Complaints with MinHash: Costly Sampling

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$: 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$S_2$: 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0</td>
</tr>
<tr>
<td>$S_3$: 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(S_1)$: 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$\pi(S_2)$: 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>$\pi(S_3)$: 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0</td>
</tr>
</tbody>
</table>

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

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. (LSK WWW 2012)
Complaints with MinHash: Costly Sampling

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

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

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. (LSK WWW 2012)

Storing only the minimum seems quite wasteful.
1. **Sketching**: Bin and compute minimum non-zero index in each bin.

<table>
<thead>
<tr>
<th></th>
<th>$\pi(S_1)$</th>
<th>$\pi(S_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1</td>
<td>0 0 0 0 0 1 1 1 0 0 0 0 1 0 1 0 1 1</td>
</tr>
</tbody>
</table>

Solution: One Pass for All hashes
1. **Sketching:** Bin and compute minimum non-zero index in each bin.

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\pi(S_1))</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>(\pi(S_2))</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
1. **Sketching:** Bin and compute minimum non-zero index in each bin.

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(S_1)$</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>$\pi(S_2)$</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>
Solution: One Pass for All hashes

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(S_1)$</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>$\pi(S_2)$</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>OPH($S_1$)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OPH($S_2$)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>
### Solution: One Pass for All hashes

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi(S_1) )</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>( \pi(S_2) )</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>OPH(S₁)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OPH(S₂)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( H(S_1) )</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( H(S_2) )</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>
Solution: One Pass for All hashes

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(S_1)$</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>$\pi(S_2)$</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>OPH($S_1$)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OPH($S_2$)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H(S_1)$</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$H(S_2)$</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>
Solution: One Pass for All hashes

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\pi(S_1))</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>(\pi(S_2))</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>OPH((S_1))</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OPH((S_2))</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H(S_1))</td>
<td>1+C</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(H(S_2))</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>
Solution: One Pass for All hashes

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi(S_1)$</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>$\pi(S_2)$</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>OPH($S_1$)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OPH($S_2$)</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H(S_1)$</td>
<td>1+C</td>
<td>1</td>
<td>2+C</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$H(S_2)$</td>
<td>1+C</td>
<td>1</td>
<td>0+C</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>
Solution: One Pass for All hashes

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\pi(S_1))</td>
<td>0 0 0 0</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
<td>0 0 1 1</td>
<td>1 0 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td>(\pi(S_2))</td>
<td>0 0 0 0</td>
<td>0 1 1 1</td>
<td>0 0 0 0</td>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>OPH( (S_1))</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OPH( (S_2))</td>
<td>E</td>
<td>1</td>
<td>E</td>
<td>0</td>
<td>0</td>
<td>E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Bin 0</th>
<th>Bin 1</th>
<th>Bin 2</th>
<th>Bin 3</th>
<th>Bin 4</th>
<th>Bin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H(S_1))</td>
<td>1+C</td>
<td>1</td>
<td>2+C</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(H(S_2))</td>
<td>1+C</td>
<td>1</td>
<td>0+C</td>
<td>0</td>
<td>0</td>
<td>1+2C</td>
</tr>
</tbody>
</table>

- \(\Pr(\mathcal{H}_j(S_1) = \mathcal{H}_j(S_2)) = R \) for \(i = \{0, 1, 2, ..., k\}\)
- \(O(d + k)\) instead of traditional \(O(dk)\)!
Figure: Ratio of old and new hashing time indicates a linear time speedup
Datasets and Baselines

Table: Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Query</th>
<th># Train</th>
<th># Dim</th>
<th>nonzeros (mean ± std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP2006</td>
<td>2,000</td>
<td>17,395</td>
<td>4,272,227</td>
<td>6072 ± 3208</td>
</tr>
<tr>
<td>MNIST</td>
<td>2,000</td>
<td>68,000</td>
<td>784</td>
<td>150 ± 41</td>
</tr>
<tr>
<td>NEWS20</td>
<td>2,000</td>
<td>18,000</td>
<td>1,355,191</td>
<td>454 ± 654</td>
</tr>
<tr>
<td>NYTIMES</td>
<td>2,000</td>
<td>100,000</td>
<td>102,660</td>
<td>232 ± 114</td>
</tr>
</tbody>
</table>

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

Anshumali Shrivastava and Ping Li
WWW 2015
May 21st 2015 24 / 27
Anshumali Shrivastava and Ping Li
WWW 2015
May 21st 2015 25 / 27

Ranking Verification

Recall (%)
Precision (%)
Top 100
EP2006
32 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
MNIST
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
NEWS20
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
NYTimes
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
EP2006
64 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
MNIST
64 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
NEWS20
64 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
NYTimes
64 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
EP2006
128 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
MNIST
128 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
NEWS20
128 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH

Recall (%)
Precision (%)
Top 100
NYTimes
128 Hashes
MinHash
Proposed
L2−ALSH
Sign−ALSH
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*