Finding Similar Items: Locality Sensitive Hashing

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu
## New thread: High dim. data

<table>
<thead>
<tr>
<th>High dim. data</th>
<th>Graph data</th>
<th>Infinite data</th>
<th>Machine learning</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locality sensitive hashing</td>
<td>PageRank, SimRank</td>
<td>Filtering data streams</td>
<td>SVM</td>
<td>Recommender systems</td>
</tr>
<tr>
<td>Clustering</td>
<td>Network Analysis</td>
<td>Web advertising</td>
<td>Decision Trees</td>
<td>Association Rules</td>
</tr>
<tr>
<td>Dimensionality reduction</td>
<td>Spam Detection</td>
<td>Queries on streams</td>
<td>Perceptron, kNN</td>
<td>Duplicate document detection</td>
</tr>
</tbody>
</table>

Given a **query image patch**, find similar images

Visually similar results

![Query Image Patch](image1.jpg)

![Similar Images](image2.jpg)
Collect billions of images
Determine feature vector for each image (4k dim)
Given a query $Q$, find nearest neighbors FAST
Nearest neighbor query in the embedding space
Application: Visual Search

Visually similar results

- Shoes
- Sneakers
- Nike
- Adidas
- Fashion
- Light up shoes
- Style
- Air force

Q
Many problems can be expressed as finding “similar” sets:
- Find near-neighbors in high-dimensional space

Examples:
- Pages with similar words
  - For duplicate detection, classification by topic
- Customers who purchased similar products
  - Products with similar customer sets
- Images with similar features
  - Image completion
- Recommendations and search
Given: High dimensional data points $x_1, x_2, \ldots$
  - For example: Image is a long vector of pixel colors
  - And some distance function $d(x_1, x_2)$
    - which quantifies the “distance” between $x_1$ and $x_2$

Goal: Given $q$, find data points $x_j$ that are within distance threshold $d(q, x_j) \leq s$

Note: Naïve solution would take $O(N)$ where $N$ is the number of data points

MAGIC: This can be done in $O(1)$!! How??
Given: High dimensional data points $x_1, x_2, \ldots$

- For example: Image is a long vector of pixel colors
- And some distance function $d(x_1, x_2)$
  - which quantifies the “distance” between $x_1$ and $x_2$

Goal: Find all pairs of data points $(x_i, x_j)$ that are within distance threshold $d(x_i, x_j) \leq s$

Note: Naïve solution would take $O(N^2)$ where $N$ is the number of data points

MAGIC: This can be done in $O(N)$!! How??
Overview of LSH: The Bigfoot of CS

- LSH is really a family of related techniques
- In general, one throws items into buckets using several different “hash functions”.
- You examine only those pairs of items that share a bucket for at least one of these hashings.

**Upside:** Designed correctly, only a small fraction of points are ever examined

**Downside:** There are *false negatives* – there might be similar items that get missed.
Motivating Application: Finding Similar Documents
Motivation for Min-Hash/LSH

- Suppose we need to find near-duplicate documents among $N = 1$ million documents
  - Naïvely, we would have to compute pairwise similarities for every pair of docs
    - $N(N - 1)/2 \approx 5 \times 10^{11}$ comparisons
    - At $10^5$ secs/day and $10^6$ comparisons/sec, it would take 5 days
  - For $N = 10$ million, it takes more than a year...

- Similarly, we have a dataset of 10B documents, quickly find the document that is most similar to query document $q$. 
3 Essential Steps for Similar Docs

1. **Shingling**: Converts a document into a set representation (Boolean vector)

2. **Min-Hashing**: Convert large sets to short signatures, while preserving similarity

3. **Locality-Sensitive Hashing**: Focus on pairs of signatures likely to be from similar documents
   - **Candidate pairs!**
The set of strings of length $k$ that appear in the document

**Signatures:** short integer vectors that represent the sets, and reflect their similarity

**Candidate pairs:** those pairs of signatures that we need to test for similarity
Step 1: **Shingling:**
Convert a document into a set of strings of length $k$ that appear in the document.
Step 1: **Shingling:** Converts a document into a set

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the document
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples
- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- Represent a document by the set of hash values of its *k*-shingles
Compressing Shingles

- **Example:** $k=2$; document $D_1 = abcab$
  - Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
  - Hash the shingles: $h(D_1) = \{1, 5, 7\}$
- $k = 8, 9, \text{ or } 10$ is often used in practice

- **Benefits of shingles:**
  - Documents that are intuitively similar will have many shingles in common
  - Changing a word only affects $k$-shingles within distance $k-1$ from the word
Document $D_i$ is represented by a set of its $k$-shingles $C_i = S(D_i)$

A natural similarity measure is the Jaccard similarity:

$$sim(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Jaccard distance:

$$d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

3 in intersection.
8 in union.
Jaccard similarity
= 3/8
Encode sets using 0/1 (bit, Boolean) vectors

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 1 in row $e$ and column $s$ if and only if $e$ is a member of $s$
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
- **Typical matrix is sparse!**
- **Each document is a column:**
  - Example: $\text{sim}(C_1,C_2) =$ ?
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) = $3/6$
    - $d(C_1,C_2) = 1 - \text{(Jaccard similarity)} = 3/6$

<table>
<thead>
<tr>
<th>Shingles</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

We don’t really construct the matrix; just imagine it exists.
So far:
- Documents → Sets of shingles
- Represent sets as Boolean vectors in a matrix

Next goal: Find similar columns while computing small signatures
- Similarity of columns == similarity of signatures

Warnings:
- Comparing all pairs takes too much time: Job for LSH
  - These methods can produce false negatives, and even false positives (if the optional check is not made)
Step 2: **Min-Hashing:** Convert large sets to short signatures, while preserving similarity

**Signatures:** short integer vectors that represent the sets, and reflect their similarity

**The set of strings of length** $k$ **that appear in the document**
Key idea: “hash” each column \( C \) to a small signature \( h(C) \), such that:
- \( \text{sim}(C_1, C_2) \) is the same as the “similarity” of signatures \( h(C_1) \) and \( h(C_2) \)

Goal: Find a hash function \( h(\cdot) \) such that:
- If \( \text{sim}(C_1, C_2) \) is high, then with high prob. \( h(C_1) = h(C_2) \)
- If \( \text{sim}(C_1, C_2) \) is low, then with high prob. \( h(C_1) \neq h(C_2) \)

Idea: Hash docs into buckets. Expect that “most” pairs of near duplicate docs hash into the same bucket!
Goal: Find a hash function $h(\cdot)$ such that:

- if $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
- if $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Clearly, the hash function depends on the similarity metric:

- Not all similarity metrics have a suitable hash function
- There is a suitable hash function for the Jaccard similarity: It is called Min-Hashing
- Next lecture we'll cover what other similarity functions can be used with LSH
Min-Hashing: Overview

- Permute the rows of the Boolean matrix using some permutation $\pi$
  - Thought experiment – not real

- Define **minhash function** for this permutation $\pi$, $h_\pi(C) = \text{the number of the first (in the permuted order) row in which column } C \text{ has value 1.}$
  - Denoted this as: $h_\pi(C) = \min_\pi \pi(C)$

- Apply, to all columns, several randomly chosen permutations $\pi$ to create a signature for each column

- **Result is a signature matrix**: Columns = sets, Rows = minhash values for each permutation $\pi$
### Min-Hashing Example

**Permutation** $\pi

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**Input matrix (Shingles x Documents)**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Signature matrix** $M$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$h_\pi(C) = \min_\pi \pi(C)$

$h_2(3) = 4$ (permutation 2, column 3)

2\textsuperscript{nd} element of the permutation (row 1) is the first to map to a 1

4\textsuperscript{th} element of the permutation (row 1) is the first to map to a 1
Students sometimes ask whether the minhash value should be the original number of the row, or the number in the permuted order (as we did in our example).

**Answer:** It doesn’t matter.

- We only need to be consistent and assure that two columns get the same value if and only if their first 1’s in the permuted order are in the same row.
The Min-Hash Property (1)

- **Choose a random permutation** \( \pi \)
- **Claim:** \( \Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = \text{sim}(C_1, C_2) \)
- **Why?**
  
  - Let \( X \) be a doc (set of shingles), \( z \in X \) is a shingle
  
  - **Then:** \( \Pr[\pi(z) = \min(\pi(X))] = 1/|X| \)
    - It is equally likely that any \( z \in X \) is mapped to the \text{min} element
  
  - **Now,** let \( y \) be s.t. \( \pi(y) = \min(\pi(C_1 \cup C_2)) \)
  
  - **Then either:** \( \pi(y) = \min(\pi(C_1)) \) if \( y \in C_1 \), or \( \pi(y) = \min(\pi(C_2)) \) if \( y \in C_2 \)
  
  - So the prob. that **both** are true is the prob. \( y \in C_1 \cap C_2 \)
  
  - \( \Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2|/|C_1 \cup C_2| = \text{sim}(C_1, C_2) \)
Another Way to See This

- Given cols $C_1$ and $C_2$, rows are classified as:

  $\begin{array}{c|c|c}
  C_1 & C_2 \\
  \hline
  A & 1 & 1 \\
  B & 1 & 0 \\
  C & 0 & 1 \\
  D & 0 & 0 \\
  \end{array}$

- Define: $a = \# \text{ rows of type A}$, etc.

- Note: $\text{sim}(C_1, C_2) = \frac{a}{a + b + c}$

- Then: $\Pr[h(C_1) = h(C_2)] = \text{Sim}(C_1, C_2)$

  - Look down the permuted cols $C_1$ and $C_2$ until we see a 1
  - If it’s a type-A row, then $h(C_1) = h(C_2)$
  - If a type-B or type-C row, then not
We know: \( \Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = sim(C_1, C_2) \)

Now generalize to multiple hash functions

The *similarity of two signatures* is the fraction of the hash functions in which they agree.

Thus, the expected similarity of two signatures equals the Jaccard similarity of the columns or sets that the signatures represent:

- And the longer the signatures, the smaller will be the expected error.
**Min-Hashing Example**

**Permutation \( \pi \)**

\[
\begin{array}{ccc}
2 & 4 & 3 \\
3 & 2 & 4 \\
7 & 1 & 7 \\
6 & 3 & 2 \\
1 & 6 & 6 \\
5 & 7 & 1 \\
4 & 5 & 5 \\
\end{array}
\]

**Input matrix (Shingles x Documents)**

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
\end{array}
\]

**Signature matrix **\( M \)**

\[
\begin{array}{cccc}
2 & 1 & 2 & 1 \\
2 & 1 & 4 & 1 \\
1 & 2 & 1 & 2 \\
\end{array}
\]

**Similarities:**

**Col/Col**

\[
\begin{array}{cccc}
1-3 & 2-4 & 1-2 & 3-4 \\
0.75 & 0.75 & 0 & 0 \\
0.67 & 1.00 & 0 & 0 \\
\end{array}
\]

**Sig/Sig**

\[
\begin{array}{cccc}
1-3 & 2-4 & 1-2 & 3-4 \\
0.75 & 0.75 & 0 & 0 \\
0.67 & 1.00 & 0 & 0 \\
\end{array}
\]
Permuting rows even once is prohibitive

Row hashing!
- Pick $K = 100$ hash functions $h_i$
- Ordering under $h_i$ gives a random permutation $\pi$ of rows!

One-pass implementation
- For each column $c$ and hash-func. $h_i$ keep a “slot” $M(i, c)$ for the min-hash value of column $c$ and hash-func $i$
- Initialize all $M(i, c) = \infty$
- Scan rows looking for 1s
  - Suppose row $j$ has 1 in column $c$
  - Then for each $h_i$:
    - If $h_i(j) < M(i, c)$, then $M(i, c) \leftarrow h_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x)=((a \cdot x + b) \mod p) \mod N$
where:
a, b ... random integers
p ... prime number ($p > N$)
Step 3: **Locality Sensitive Hashing:**
Focus on pairs of signatures likely to be from similar documents

**Locality Sensitive Hashing**

- **Document**
- **Shingling**
  - The set of strings of length $k$ that appear in the document
- **Min-Hashing**
- **Locality-Sensitive Hashing**
  - **Signatures:** short integer vectors that represent the sets, and reflect their similarity
  - **Candidate pairs:** those pairs of signatures that we need to test for similarity
Goal: Find documents with Jaccard similarity at least \( s \) (for some similarity threshold, e.g., \( s=0.8 \))

**LSH – General idea:** Use a hash function that tells whether \( x \) and \( y \) is a candidate pair: a pair of elements whose similarity must be evaluated.

**For Min-Hash matrices:**
- Hash columns of signature matrix \( M \) to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.
Pick a similarity threshold $s$ ($0 < s < 1$)

Columns $x$ and $y$ of $M$ are a candidate pair if their signatures agree on at least fraction $s$ of their rows:

$M(i, x) = M(i, y)$ for at least frac. $s$ values of $i$

- We expect documents $x$ and $y$ to have the same (Jaccard) similarity as their signatures
Big idea: Hash columns of signature matrix $M$ several times

Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs are those that hash to the same bucket
Partition $M$ into $b$ Bands

Signature matrix $M$
Partition $M$ into Bands

- Divide matrix $M$ into $b$ bands of $r$ rows
- For each band, hash its portion of each column to a hash table with $k$ buckets
  - Make $k$ as large as possible
- Candidate column pairs are those that hash to the same bucket for $\geq 1$ bands
- Tune $b$ and $r$ to catch most similar pairs, but few non-similar pairs
Columns 2 and 6 are probably identical (candidate pair)

Columns 6 and 7 are surely different.

Matrix $M$
There are **enough buckets** that columns are unlikely to hash to the same bucket unless they are **identical** in a particular band.

Hereafter, we assume that “**same bucket**” means “**identical in that band**”.

Assumption needed only to simplify analysis, not for correctness of algorithm.
Assume the following case:

- Suppose 100,000 columns of $M$ (100k docs)
- Signatures of length 100, stored as integers (rows)
- Therefore, signatures take 40MB
- **Goal:** Find pairs of documents that are at least $s = 0.8$ similar
- Choose $b = 20$ bands of $r = 5$ integers/band
If $C_1$, $C_2$ are 80% Similar

- **Find pairs of** $\geq s=0.8$ similarity, set $b=20$, $r=5$
- **Assume:** $\text{sim}(C_1, C_2) = 0.8$
  - Since $\text{sim}(C_1, C_2) \geq s$, we want $C_1, C_2$ to be a **candidate pair:** We want them to hash to at least 1 common bucket (at least one band is identical)
- **Probability $C_1, C_2$ identical in one particular band:** $(0.8)^5 = 0.328$
- **Probability $C_1, C_2$ are **not** similar in all 20 bands:** $(1-0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
- **We would find 99.965% pairs of truly similar documents**
Find pairs of $\geq s=0.8$ similarity, set $b=20$, $r=5$

Assume: $\text{sim}(C_1, C_2) = 0.3$

- Since $\text{sim}(C_1, C_2) < s$ we want $C_1, C_2$ to hash to NO common buckets (all bands should be different)

Probability $C_1, C_2$ identical in one particular band: $(0.3)^5 = 0.00243$

Probability $C_1, C_2$ identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$

- In other words, approximately 4.74% pairs of docs with similarity 0.3 end up becoming candidate pairs
  - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold $s$
**LSH Involves a Tradeoff**

- **Pick:**
  - The number of Min-Hashes (rows of $M$)
  - The number of bands $b$, and
  - The number of rows $r$ per band
to balance false positives/negatives
  - Note, $M=b*r$

- **Example:** If we had only 10 bands of 10 rows, the number of false positives would go down, but the number of false negatives would go up
Analysis of LSH – What We Want

Probability of sharing a bucket

Similarity $t = \text{sim}(C_1, C_2)$ of two sets

Similarity threshold $s$

Probability $= 1$ if $t > s$

No chance if $t < s$

Say “yes” if you are below the line.
Remember:
Probability of equal hash-values = similarity

Similarity $t = \text{sim}(C_1, C_2)$ of two sets
Similarity $t = \text{sim}(C_1, C_2)$ of two sets

Probability of sharing a bucket

False positives

False negatives

Say “yes” if you are below the line.
Say columns $C_1$ and $C_2$ have similarity $t$

Pick any band ($r$ rows)
- Prob. that all rows in band equal = $t^r$
- Prob. that some row in band unequal = $1 - t^r$

Prob. that no band identical = $(1 - t^r)^b$

Prob. that at least 1 band identical = $1 - (1 - t^r)^b$
What \( b \) Bands of \( r \) Rows Gives You

Similarity \( t = \text{sim}(C_1, C_2) \) of two sets

\[
1 - (1 - t^r)^b
\]

- All rows are equal
- Some row of a band are unequal
- At least one band identical
- No bands identical

Probability of sharing a bucket
Example: $b = 20; r = 5$

- **Similarity threshold** $s$
- **Prob. that at least 1 band is identical:**

<table>
<thead>
<tr>
<th>$s$</th>
<th>$1-(1-s^r)^b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.006</td>
</tr>
<tr>
<td>0.3</td>
<td>0.047</td>
</tr>
<tr>
<td>0.4</td>
<td>0.186</td>
</tr>
<tr>
<td>0.5</td>
<td>0.470</td>
</tr>
<tr>
<td>0.6</td>
<td>0.802</td>
</tr>
<tr>
<td>0.7</td>
<td>0.975</td>
</tr>
<tr>
<td>0.8</td>
<td>0.9996</td>
</tr>
</tbody>
</table>
Picking $r$ and $b$: The S-curve

- Picking $r$ and $b$ to get the best S-curve
  - 50 hash-functions ($r=5, b=10$)

![Graph showing S-curve with yellow and blue areas indicating False Negative and False Positive rates.](image-url)
Tune $M, b, r$ to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.

Check in main memory that candidate pairs really do have similar signatures.

Optional: In another pass through data, check that the remaining candidate pairs really represent similar documents.
Summary: 3 Steps

- **Shingling:** Convert documents to set representation
  - We used hashing to assign each shingle an ID

- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used similarity preserving hashing to generate signatures with property \( \Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2) \)
  - We used hashing to get around generating random permutations

- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find candidate pairs of similarity \( \geq s \)