Similarity Search and Locality Sensitive Hashing

Marta Arias, UPC

Grau en Enginyeria Informàtica, UPC

October 05, 2025

Locality Sensitive Hashing (LSH)

Motivation

- Finding similar items efficiently in large datasets is essential for tasks like:
  - Near-duplicate detection
  - Image and document retrieval
  - Recommender systems
- Exact similarity search is expensive — often $O(n^2)$
- **Locality Sensitive Hashing (LSH)** provides a probabilistic approach for *approximate* similarity search in *sub-linear* time.

Main Idea

- LSH uses hash functions that **preserve similarity**:
  - Similar objects x, y often collide: $P[h(x) = h(y)] \geq p_1$
  - Dissimilar objects z, v rarely collide: $P[h(z) = h(v)] \leq p_2$
  - with $p_1 > p_2$

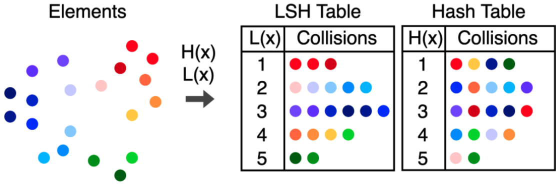

Figure 1: General hash function vs. locality-sensitive hash function.

Definition

A hash family $\mathcal{H}$ is $(s, c \cdot s, p_1, p_2)$-**sensitive** if for any two objects x, y:

- If $s(x, y) \geq s$, then $P[h(x) = h(y)] \geq p_1$
- If $s(x, y) \leq c \cdot s$, then $P[h(x) = h(y)] \leq p_2$

where the probability is taken over a random $h \in \mathcal{H}$, with $c < 1$.

For example..

- If $s(x, y) \geq 0.8$, then $P[h(x) = h(y)] \geq 0.9$
- If $s(x, y) \leq 0.5$, then $P[h(x) = h(y)] \leq 0.2$

Example 1: Fixed-size Bit Vectors

- Objects represented as $x \in \{0,1\}^d$
- **Hamming distance:** $d_H(x,y) = \sum |x_i - y_i|$
- **Similarity function:** $s(x,y) = 1 - \frac{d_H(x,y)}{d}$

Hash family

- Choose a bit position $i \in \{1, ..., d\}$ uniformly at random.
- Define $h_i(x) = x_i$

Then, $P[h(x) = h(y)] = s(x,y)$

Thus, $\mathcal{H}$ is $(s, c \cdot s, s, c \cdot s)$-sensitive.

Example Calculation

Let $x = 10010$, $y = 11011$, $d = 5$

$$d_H(x, y) = 2, \quad s(x, y) = 1 - 2/5 = 0.6$$

If $s = 0.9$, $c = 0.6$:

- $p_1 = s = 0.9$
- $p_2 = c \cdot s = 0.54$

And so:

- If $s(x, y) \geq 0.9$, then $P[h(x) = h(y)] \geq 0.9$
- If $s(x, y) \leq 0.54$, then $P[h(x) = h(y)] \leq 0.54$

Gap: $p_1 - p_2 = 0.36$

Amplifying the Gap

Stacking

- Combine k independent hash functions:

$$h(x) = (h_1(x), ..., h_k(x))$$

- Collision probabilities:
  - Similar objects: p_1^k
  - Dissimilar objects: p_2^k

Repetition

- Repeat m times with independent functions.
- Probability of at least one collision for similar items:

$$1 - (1 - p_1^k)^m$$

Resulting sensitivity:

$$(s, cs, 1 - (1 - s^k)^m, 1 - (1 - (cs)^k)^m)$$

Amplifying the Gap, cont.

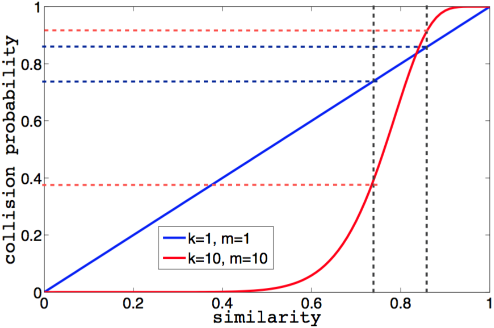

Figure 2: Amplifying the gap with a collision probability of $1-(1-s^k)^m$.

Similarity Search System

1. **Preprocessing**
   - Choose $k \times m$ random projections.
   - Build m hash tables with k-length hash functions.
   - Insert each object into corresponding buckets.
2. **Querying**
   - Compute m hash codes for the query object.
   - Retrieve candidate objects from matching buckets.
   - Compute actual similarities only among candidates.

Search Time

$$O(dm \log n)$$

This achieves *sub-linear* time in database size n.

Example 2: Fixed-size Integer Vectors

- Objects: $x \in [1..M]^d$
- Represent integers in unary: $u(x_i)$

Example: If $M = 8$, $x = (5, 2) \rightarrow (11111000, 11000000)$

- **L1 distance:** $d_{Manhattan}(x, y) = \sum |x_i - y_i|$
- Equivalent to Hamming distance on unary representations.
- **Similarity:** $s(x, y) = 1 - \frac{d_{Manhattan}(x,y)}{dM}$

Thus, reuse bit-vector LSH scheme on dM-bit representations.

Example 3: Simhashing for tf-idf Vectors

- Documents represented by tf-idf vectors in $\mathbb{R}^V$
- Normalized to unit length $\rightarrow$ all vectors on unit sphere.

Random Projections

- Random hyperplane w through origin.
- Hash function: $h_w(x) = 1_{\{w^T x \geq 0\}}$
- $P[h_w(x) = h_w(y)] = 1 - \frac{\theta}{\pi}$, where θ is the angle between x, y

Thus, collision probability corresponds to **cosine similarity**.

Example 3: Simhashing for tf-idf Vectors, cont.

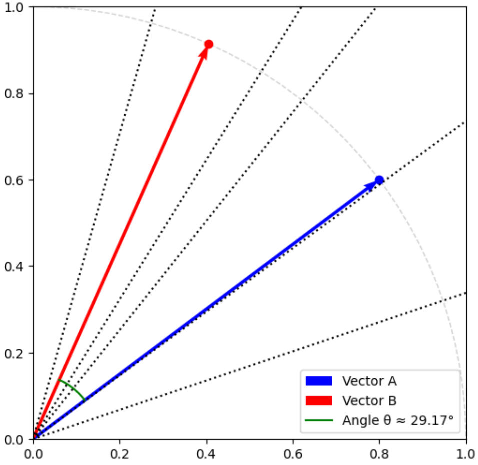

Figure 3: Random projections “separate” the two vectors w.p. $\frac{\theta}{\pi/2}$

Simhashing Implementation

Preprocessing

1. For each term in vocabulary, generate a unique bitstring (e.g., via MD5).
2. Build matrix H (vocabulary $\times$ bit-length), replacing 0s by -1s.
3. Compute document simhash: $simhash(d) = 1_{\{dH \geq 0\}}$.
4. Split simhash into m groups of k bits and index documents.

Querying

1. Compute $simhash(q)$ for query.
2. Split into m chunks of k bits.
3. Retrieve all documents sharing a bucket with q.
4. Return top matches based on actual similarity.

Summary

- **Locality Sensitive Hashing** allows efficient approximate similarity search.
- Works by mapping similar items to the same hash bucket with high probability.
- Core techniques:
  - Randomized hash functions preserving similarity.
  - Amplification using stacking and repetition.
- Applicable to:
  - Bit vectors (Hamming)
  - Integer vectors (L1 distance)
  - tf-idf vectors (Cosine similarity via simhash)

References

- Gionis, Indyk, & Motwani (1999). *Similarity Search in High Dimensions via Hashing.* VLDB.
- Charikar (2002). *Similarity Estimation Techniques from Rounding Algorithms.* STOC.
- Indyk & Motwani (1998). *Approximate Nearest Neighbors.* STOC.
- Randorithms (2019). *Visualizing Locality Sensitive Hashing.*