CS 7800/4810: Locality Sensitive Hashing (LSH) & Nearest Neighbor Search

Lecture Notes

1 Classical Hashing

A hash function maps elements from a large universe $\mathcal{U}$ into a hash table T of much smaller size:

$$h : \mathcal{U} \to [0, |T| - 1], \qquad |T| \ll |\mathcal{U}|.$$

The family of all such functions is denoted $\mathcal{H} = \{h : \mathcal{U} \to [0, |T| - 1]\}$.

Properties of a perfect hash function:

- If $x = y$, then $h(x) = h(y)$.
- If $x \neq y$, then $h(x) \neq h(y)$.

In practice, the second property cannot always be satisfied because $|T| \ll |\mathcal{U}|$, so collisions are inevitable.

Hashing is useful because it allows us to map items drawn from a large universe to a much smaller one. A key observation is that hashed values are still ordered—you can think of a hash function as a random permutation from an input space to an output space.

2 Universal Hashing

The goal of universal hashing is to make the number of collisions *as rare as possible.*

A family $\mathcal{H}$ is **universal** if, for all $x, y \in \mathcal{U}$ with $x \neq y$,

$$\Pr_{h \in \mathcal{H}}\big[h(x) = h(y)\big] = \frac{1}{|T|}.$$

Note: The probability is taken over the *random choice* of $h \in \mathcal{H}$, **not** over the elements x, y.

3 Locality Sensitive Hashing (LSH)

In classical hashing, we want to avoid collisions at all costs. With LSH, our goal is the opposite—we *want* similar items to hash to the same value.

3.1 Definition

A family $\mathcal{H}$ is (d_1, d_2, p_1, p_2)-**sensitive** if, for all $x, y \in \mathcal{U}$:

$$E_d(x, y) \leq d_1 \implies \Pr_{h \in \mathcal{H}}\big[h(x) = h(y)\big] \geq p_1, \tag{1}$$

$$E_d(x, y) \geq d_2 \implies \Pr_{h \in \mathcal{H}}\big[h(x) = h(y)\big] \leq p_2, \tag{2}$$

where $E_d(x, y)$ is a distance function and $d_1 < d_2$, $p_1 > p_2$.

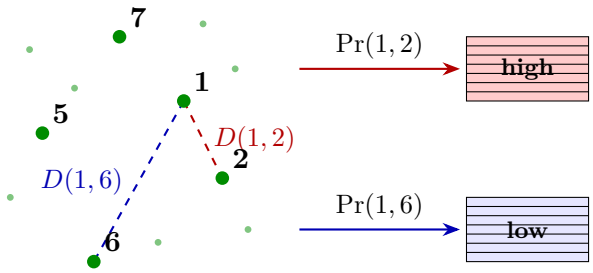
Intuition:

- Low distance $\Leftrightarrow$ high collision probability.
- High distance $\Leftrightarrow$ low collision probability.
- For distances between d_1 and d_2, there is no guarantee.

When $d_1 = d_2$, the family is called an **ungapped LSH**. When there is a gap between d_1 and d_2, we call this **gapped LSH**.

3.2 Visual Intuition

The figure below illustrates the core idea. Points 1 and 2 are close together (small distance $D(1, 2)$), so they have a high probability of hashing to the same bucket. Points 1 and 6 are far apart (large distance $D(1, 6)$), so their collision probability is much lower.



Since $D(1, 2) < D(1, 6)$, we have $\Pr(1, 2) \gg \Pr(1, 6)$.

4 Notion of Similarity: Jaccard Index

Given two sets S_1 and S_2, the **Jaccard similarity** (or Jaccard index) is defined as:

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{\text{cardinality of intersection}}{\text{cardinality of union}}.$$

The **Jaccard distance** is defined as $D(S_1, S_2) = 1 - J(S_1, S_2)$.

4.1 Example

Let $S_1 = \{3, 10, 15, 19\}$ and $S_2 = \{4, 10, 15\}$.

$$J(S_1, S_2) = \frac{|\{10, 15\}|}{|\{3, 4, 10, 15, 19\}|} = \frac{2}{5}.$$

4.2 Weighted Jaccard Similarity

For weighted sets where each element k has weights $s_1^{(k)}$ and $s_2^{(k)}$:

$$J_w(S_1, S_2) = \frac{\sum_k \min\big(s_1^{(k)},\, s_2^{(k)}\big)}{\sum_k \max\big(s_1^{(k)},\, s_2^{(k)}\big)}.$$

5 MinHash (Min-wise Hashing)

Introduced by Andrei Broder, 1997.

Let h be a random universal hash function mapping strings to natural numbers:

$$h : \text{Strings} \to \mathbb{N}.$$

The **MinHash** of a set S under h is:

$$\text{MinHash}_h(S) = \min_{x \in S} h(x).$$

Procedure: Every time we want to generate a new MinHash value, we draw a new universal hash function and compute the minimum.

5.1 Key Property

For any two sets S_1 and S_2:

$$\Pr\big[\text{MinHash}(S_1) = \text{MinHash}(S_2)\big] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = J(S_1, S_2).$$

That is, the probability of a MinHash collision equals the Jaccard similarity exactly.

- If the similarity between two sets increases, the probability of collision also increases.
- MinHash is an **unbiased estimator** of the Jaccard similarity.
- This makes it an **ungapped LSH** for the Jaccard distance.

5.2 Intuition

Consider two sets S_1 and S_2. We use a universal hash function, so any item in $S_1 \cup S_2$ is equally likely to receive the smallest hash value:

$$\Pr(\text{any item in } S_1 \cup S_2 \text{ is the smallest}) = \frac{1}{|S_1 \cup S_2|}.$$

For $\text{MinHash}(S_1) = \text{MinHash}(S_2)$, the element achieving the global minimum must belong to $S_1 \cap S_2$. Since there are $|S_1 \cap S_2|$ such elements, each equally likely to be the minimum:

$$\Pr\big[\text{MinHash}(S_1) = \text{MinHash}(S_2)\big] = \sum_{|S_1 \cap S_2|} \frac{1}{|S_1 \cup S_2|} = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}.$$

5.3 Example

With $S_1 = \{3, 10, 15, 19\}$ and $S_2 = \{4, 10, 15\}$:

$$J(S_1, S_2) = \frac{|\{10, 15\}|}{|\{3, 4, 10, 15, 19\}|} = \frac{2}{5}.$$

6 Estimating Jaccard with a Sketch

Question: How can we estimate the Jaccard similarity if we have k MinHash values for S_1 and S_2?

Instead of dealing with large sets (which require significant time and memory), MinHash provides a **sketch**—a small, fixed-size summary—that approximates the Jaccard similarity in a scalable way. We take a sketch of k MinHash values and count the fraction that collide.

6.1 Variance of the Estimator

Each MinHash trial is a Bernoulli random variable:

$$X_i = \begin{cases} 1 & \text{with probability } J(S_1, S_2), \\ 0 & \text{with probability } 1 - J(S_1, S_2). \end{cases}$$

The estimator $\hat{J} = \frac{1}{k} \sum_{i=1}^{k} X_i$ has variance:

$$\text{Var}(\hat{J}) = \frac{J(1-J)}{k}.$$

6.2 Example

If $J = 0.8$ and $|\text{sketch}| = 50$:

$$\text{Standard Error} = \sqrt{\frac{0.8 \times 0.2}{50}} = \sqrt{\frac{0.16}{50}} \approx 0.05.$$

7 Parity of MinHash

A space-saving variant: instead of storing the full MinHash value, store only its **parity** (0 if even, 1 if odd).

7.1 Collision Probability Under Parity

Let $M(S)$ denote the MinHash of set S. Define $J = J(S_1, S_2)$. Then:

$$\begin{aligned} p' &= \Pr\big[\text{Parity}(M(S_1)) = \text{Parity}(M(S_2))\big] \\ &= \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} + \left(1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}\right) \times 0.5 \\ &= 0.5 \times \left(1 + \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}\right) \\ &= \frac{1+J}{2}. \end{aligned}$$

7.2 Variance Under Parity

$$\text{Var}' = \frac{p'(1-p')}{k} = \frac{0.5(1+J)\big(1 - 0.5(1+J)\big)}{k}.$$

8 Extension: One-Permutation Hashing

Li et al., 2012.

Instead of using k independent hash functions, use a *single* universal hash function $h : \text{Strings} \to \mathbb{N}$ and:

1. Divide the range $[0, N]$ into k equal-sized bins.
2. Take the minimum hash value within each bin.

Advantage: This achieves the same quality of sampling with $1/k$ the preprocessing cost of standard MinHash.

8.1 Handling Empty Bins

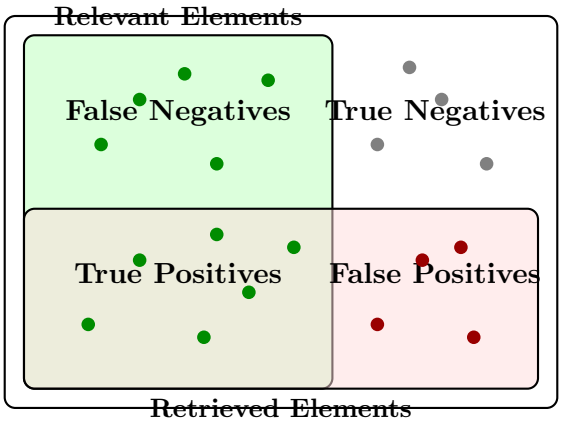
Shrivastava & Li, 2014.

When a bin is empty (no elements hash into it), a **rotation scheme** is used:

- Borrow the value from the closest non-empty bin in the clockwise direction.
- Add an offset C to the borrowed value.

9 Precision and Recall

Before designing a data structure for nearest neighbor search, we must understand the notions of **precision** and **recall**. When querying an approximate data structure, all returned (and not returned) values fall into one of four categories:



Definitions:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}.$$

In words: **precision** is the probability that a retrieved element is actually relevant. **Recall** is the probability that a relevant element was successfully retrieved. When designing and querying a nearest neighbor search data structure, we must balance these two quantities. For instance, increasing the query scope raises recall but may also increase false positives, reducing precision.

10 Nearest Neighbor Search Using LSH

The LSH search framework uses two parameters:

- K hash functions per table (controls **precision**).
- L hash tables (controls **recall**).

10.1 Construction

1. Each data point is hashed using K LSH hash functions.
2. The K hash values are concatenated to form a **hash signature**.
3. To improve recall, use L independent hash tables, each with its own K hash functions.

10.2 Query

For a query element x, compute its hash signature for each of the L tables, and return the **union** of all elements found at those L bucket locations.

Intuition: By using multiple (K) hash functions, we decrease the likelihood that distant elements hash to the same location. By using multiple (L) hash tables, we increase the likelihood that similar elements are hashed to the same location at least once.

10.3 Worked Example

Suppose we have sets S_1, S_2, S_3, S_4, S_5 hashed into $L = 4$ hash tables. Each table uses its own set of K hash functions. The contents of each table might look like:

Table 1	Table 2	Table 3	Table 4
S_1	S_4	S_1	S_5
S_5	S_1, S_2	S_2	S_2, S_1
S_2	S_3	S_3	S_3
S_3, S_4	S_5	S_4	S_4
$\cdots$	$\cdots$	S_5	$\cdots$

Query S_6: highlighted buckets $\Rightarrow$ result $= \{S_5, S_3, S_5, S_2, S_1\}$

When we query S_6, we compute its hash signature for each table and look up the corresponding bucket (highlighted in yellow). Taking the union of all retrieved elements gives the candidate set $\{S_1, S_2, S_3, S_5\}$. To increase recall, we could use more tables (L). To increase precision, we could use more hash functions (K).

10.4 Analysis

With K hash functions (using MinHash), the probability that two items x and y land in the same bucket in a specific table is:

$$\Pr(x \text{ and } y \text{ in same bucket}) = J(x, y)^K.$$

With L hash tables, the probability that y is retrieved when querying x becomes:

$$\Pr(y \text{ is retrieved}) = 1 - \left(1 - J(x, y)^K\right)^L.$$

Example: Let $K = 10$, $L = 10$, and $J(x, y) = 0.9$:

$$\Pr(x, y \text{ same bucket in one table}) = 0.9^{10} \approx 0.35,$$

$$\Pr(y \text{ is retrieved}) = 1 - (1 - 0.9^{10})^{10} = 1 - (1 - 0.35)^{10} \approx 0.986.$$

So when the similarity is high, the likelihood that we identify y as an approximate nearest neighbor is also very high.

10.5 Role of K and L

- K **controls precision:** A higher K reduces the number of collisions (fewer false positives), since distant elements must agree on *all* K hashes to collide.
- L **improves recall:** A larger L increases the chance of finding a true neighbor (fewer false negatives), since similar elements only need to collide in *one* of L tables.

11 The Power of Two Choices

Mitzenmacher, 2001.

We briefly sketch the analysis of the "power of two choices" in balls-and-bins, which has applications to load balancing and hash table design.

11.1 Setup

Consider throwing n balls into n bins uniformly at random. It is known that the maximum load (the most balls in any single bin) is $O\left(\frac{\log n}{\log \log n}\right)$ with high probability.

Now consider the **two-choice strategy**: for each ball, pick two bins uniformly at random and place the ball into the bin with the *lesser* current load. This simple change dramatically reduces the maximum load to $O(\log \log n)$.

11.2 Proof Sketch

Let the **height** of a ball be the number of balls (including itself) in the bin where it is placed. Similarly, let the height of a bin be the number of balls it contains. After throwing n balls:

Base case (height ≥ 2): The maximum number of bins with height at least 2 is $n/2$ (since n balls distributed among n bins means at most half can have ≥ 2). Therefore:

$$\Pr(\text{a randomly selected bin has height} \geq 2) \leq \frac{n/2}{n} = \frac{1}{2}.$$

Height ≥ 3: For a ball to reach height 3, it must select two bins that *both* have height ≥ 2. The probability of this is at most $\left(\frac{1}{2}\right)^2 = \frac{1}{4}$, so at most 1/4 of bins have height ≥ 3.

Height ≥ 4: Similarly, the ball must select two bins both of height ≥ 3: $\left(\frac{1}{4}\right)^2 = \frac{1}{16}$.

General pattern: The expected fraction of bins with height $\geq h$ is at most:

$$\frac{1}{2^{2^{(h-2)}}}.$$

This double-exponential decay means the fraction drops to $1/n$ (i.e., at most one such bin exists) when:

$$2^{2^{(h-2)}} = n \quad \Longrightarrow \quad h = \log\log n + 2.$$

From this point on, at most one bin has that height, and no ball can find two bins of that height to choose between. Therefore, the maximum bin height is bounded by $O(\log\log n)$.

11.3 Summary

Strategy	**Max Load**
Random placement (1 choice)	$O\left(\frac{\log n}{\log\log n}\right)$
Two choices (pick lesser load)	$O(\log\log n)$

The power of two choices shows that even a small amount of additional information (checking one extra bin) yields an exponential improvement in load balance.

References

[1] A. Broder. On the Resemblance and Containment of Documents. *Proceedings, Compression and Complexity of SEQUENCES*, 1997.

[2] P. Li, A. Owen, C.-H. Zhang. One Permutation Hashing. *NIPS*, 2012.

[3] A. Shrivastava, P. Li. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search. *ICML*, 2014.

[4] M. Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. *IEEE Trans. Parallel and Distributed Systems*, 12(10):1094–1104, 2001.