SEEM5020 Algorithms for Big Data

Nearest Neighbor Search: Locality Sensitivity Hashing

Sibo WANG

Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong

Nearest Neighbor Search (NNS)

Definition 1 (Nearest Neighbor Search)

Given a set $P = \{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n}\}$ of n points in $\mathbb{R}^d$ and a distance metric $dist(\cdot, \cdot)$, for any query point $\mathbf{q} \in \mathbb{R}^d$, the nearest neighbor search query finds the point closest to $\mathbf{q}$ in P according to the provided distance metric.

Definition 2 (r-near neighbor search (r-NNS))

Still consider the input set P of n d-dimensional points and a distance metric $dist(\cdot, \cdot)$ is given. For any query point $\mathbf{q}$, the r-near neighbor search (if exists) returns a point $\mathbf{x} \in P$ s.t. $dist(\mathbf{q}, \mathbf{x}) \leq r$

The r-near neighbor search problem can be treated as a decision problem of the nearest neighbor search problem. To solve the nearest neighbor search, we can apply the decision version with $\log(d_{max}/d_{min})$ iterations.

Nearest Neighbor Search: Exact Solutions

Consider the Euclidean distance, i.e., $dist(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2$.

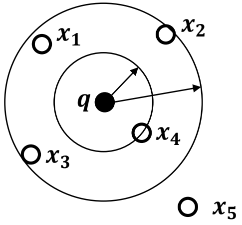
- When $d = 1$: sort the data and then for any input query point, we can do a binary search to find the closest point.
  - $O(n)$ space and $O(\log n)$ query time.
- When $d = 2$: Building a Voronoi diagram.
  - $O(n)$ space and $O(\log)$ query time.
- When $d > 2$:
  - Voronoi diagram: $O(n^{\lceil d/2 \rceil})$ space. Too expensive!
  - Linear search: $O(d \cdot n)$ search time.

Approximate Nearest Neighbor Search

Definition 3 (c-Approximate Nearest Neighbor Search (c-ANNS))

Given a set $P = \{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n}\}$ of n points in $\mathbb{R}^d$ and a distance metric $dist(\cdot, \cdot)$, for any query point $\mathbf{q} \in \mathbb{R}^d$, the c-approximate nearest neighbor search query returns an arbitrary point $\mathbf{x}$ so that $dist(\mathbf{q}, \mathbf{x}) \leq c \cdot dist(\mathbf{q}, \mathbf{x}^*)$, where $\mathbf{x}^*$ is the nearest neighbor of $\mathbf{q}$.

- Point $\mathbf{x_4}$ is the nearest neighbor of the query point $\mathbf{q}$.
- Within the range of $2 \cdot dist(\mathbf{q}, \mathbf{x_4})$, $\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$ all fall into it.
- We can return any of $\mathbf{x_1}$, $\mathbf{x_2}$, and $\mathbf{x_3}$, which is a valid answer for the 2-ANNS query.



Approximate r-Near Neighbor Search

Definition 4 (c-Approximate r-Near Neighbor Search (c, r)-ANNS)

For any query point $\mathbf{q}$, if there exists a point $\mathbf{x}$ in P such that $dist(\mathbf{q}, \mathbf{x}) \leq r$, then the c-approximate r-near neighbor search query returns a point $\mathbf{x}' \in P$ so that $dist(\mathbf{q}, \mathbf{x}') \leq c \cdot dist(\mathbf{q}, \mathbf{x})$.

Similarly, we can answer the c-approximate nearest neighbor query via a binary search on the radius r with the (c, r)-ANNS query. Next, we focus on this (c, r)-ANNS query, which will be solved by locality sensitivity hashing.

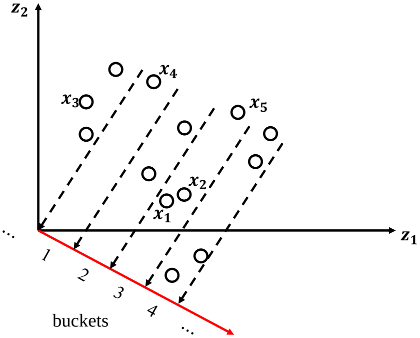
Table of Contents

Locality-Sensitive Hashing [1]

Intuition: For two points **x** and **y**, locality-sensitive hashing will hash the close data points into the same buckets with a higher probability.

Consider a simple idea that projects the data points into a random line crossing the origin.

- Close points $\mathbf{x_1}$ and $\mathbf{x_2}$ are projected into the same bucket with ID 3.
- The far point $\mathbf{x_4}$ of $\mathbf{x_1}$ is in a different bucket, with ID 1.
- The far point $\mathbf{x_5}$ is in the same bucket as $\mathbf{x_1}$. But the chances of such events are much lower.



Locality-Sensitive Hashing (LSH) Families

Definition 5 (($r, c \cdot r, p_1, p_2$)-sensitive)

Given a distance measure *dist*$(\cdot, \cdot)$, a fixed r, a family $\mathscr{H}$ of hash function is said to be $(r, c \cdot r, p_1, p_2)$-sensitive, where $p_1 > p_2$ and $c > 1$, if h randomly drawn from $\mathscr{H}$ satisfies the following:

- $\mathbb{P}[h(\mathbf{x}) = h(\mathbf{y})] \geq p_1$ when *dist*$(\mathbf{x}, \mathbf{y}) \leq r$.
  - If **x** and **y** are close, the collision probability is high.
- $\mathbb{P}[h(\mathbf{x}) = h(\mathbf{y})] \leq p_2$ when *dist*$(\mathbf{x}, \mathbf{y}) \geq c \cdot r$.
  - If **x** and **y** are sufficiently far, the probability of collision is low.

A key parameter: the gap between p_1 and p_2 measured as $\rho = \frac{\log p_1}{\log p_2}$. We will see the role of ρ later in our analysis.

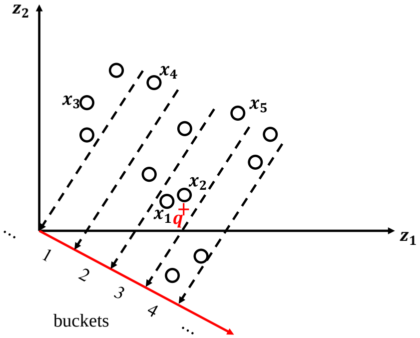
Here, we assume that we have already had the locality-sensitive family $\mathscr{H}$ for the distance measure *dist*. We will see how to design the LSH family for different metrics later.

The Issue with a Single LSH Function from $\mathscr{H}$

Given a query point $\mathbf{q}$, even though there is a small probability p_2 that a point $\mathbf{x}$ will collide with $\mathbf{q}$ if the distance is far, i.e. at least $c \cdot r$, there might exist $O(n)$ such far data points.

- There exist $O(p_2 \cdot n)$ far points that collide with $\mathbf{q}$ in expectation.
- These far data points that collide with $\mathbf{q}$ are called false positives.

- For query point $\mathbf{q}$, x_5 is a false positive



Pruning False Positives with AND Operation

We need to reduce the number of false positives!

- Solution: Pick multiple independent hash functions $h_1, h_2, \cdots, h_k$ from $\mathscr{H}$. Define a new hash function $g(\mathbf{x})$ with the AND operation:

$$g(\mathbf{x}) = \langle h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_k(\mathbf{x}) \rangle$$

where $g(\mathbf{x_1}) = g(\mathbf{x_2})$ if and only if

$$h_1(\mathbf{x_1}) = h_2(\mathbf{x_1}) \wedge h_1(\mathbf{x_2}) = h_2(\mathbf{x_2}) \wedge \cdots h_k(\mathbf{x_1}) = h_k(\mathbf{x_k})$$

Lemma 1

Given a family $\mathscr{H}$ of $(r, c \cdot r, p_1, p_2)$-sensitive hash functions, by randomly choosing k hash functions from $\mathscr{H}$ and defining it as

$$g(\mathbf{x}) = \langle h_1(\mathbf{x}), h_2(\mathbf{x}) \cdots, h_k(\mathbf{x}) \rangle,$$

$g(\mathbf{x})$ constitutes a family $\mathscr{G}$ of $(r, c \cdot r, p_1^k, p_2^k)$-sensitive hash functions.

Choosing the Appropriate k

We choose k so that the expected number of far points is ≤ 1.

- This can be done by setting $p_2^k = \frac{1}{n} \to k = \log_{1/p_2}(n)$.
- Accordingly, $p_1^k = 1/n^\rho$. If there is only one point that is the r-near neighbor of the query point $\mathbf{q}$, then in expectation we only have the probability $\frac{1}{n^\rho}$ that the point hashes to the same bucket as the query point $\mathbf{q}$.
- As we need to spend at least $O(1)$ time to do a table lookup.
  - The cost of the false positive hence can be bounded by the table lookup cost, without incurring additional cost.

Multiple Hash Tables

We now need to increase the chance that a r-near neighbor of $\mathbf{q}$ hashes to the same bucket as $\mathbf{q}$.

- We can choose a sufficiently large number L of hash functions from $(r, c \cdot r, p_1^k, p_2^k)$-sensitive hash family $\mathscr{G}$.
- Then, the r-near neighbors will hash to at least one of these L hash functions with high probability.
  - We set $L = n^\rho$. The reason will be explained shortly.
  - The storage is then: $O(n \cdot L) = O(n^{1+\rho})$.

Query Processing

The LSH includes L hash tables, where for each hash table, the hash function is chosen from the $(r, c \cdot r, p_1^k, p_2^k)$-sensitive family $\mathscr{G}$. To answer a query, it proceeds as follows:

- Retrieve data points from buckets $g_1(\mathbf{q}), g_2(\mathbf{q}), \cdots g_L(\mathbf{q})$ one by one.
- For each retrieved data point $\mathbf{x}$, we compute the distance *dist*$(\mathbf{q}, \mathbf{x})$.
- Then, *(i)* we return the first data point such that the distance to $\mathbf{q}$ is no larger than $c \cdot r$, or *(ii)* we have retrieved all data points from the L buckets but no point within distance $c \cdot r$, then we return failure.

What if there are too many data points in these L buckets? How to bound the search complexity?

- Fix: we stop the search when we have retrieved $3L$ data points (but no cr-near neighbors) and we return failure.
- This bounds the search complexity to $O(d \cdot k \cdot L)$.

Theoretical Analysis

For a query point $\mathbf{q}$, the LSH query algorithm answers correctly:

- If there is no point that is a $c \cdot r$-near neighbor of $\mathbf{q}$, hence the query returns failure.
- If the LSH query algorithm returns a data point $\mathbf{x}$, its distance to the query point $\mathbf{q}$ is always bounded by $c \cdot r$.

Theoretical Analysis

For a query point $\mathbf{q}$, the LSH query algorithm answers correctly:

- If there is no point that is a $c \cdot r$-near neighbor of $\mathbf{q}$, hence the query returns failure.
- If the LSH query algorithm returns a data point $\mathbf{x}$, its distance to the query point $\mathbf{q}$ is always bounded by $c \cdot r$.

How to bound the failure probability when there is a r-near neighbor $\mathbf{x}$ of $\mathbf{q}$ exists. When it fails?

- Event E_1: For any r-near neighbor $\mathbf{x}$, it does not collide with query point $\mathbf{q}$.
- Event E_2: There are too many (more than $3L$) far points collide with $\mathbf{q}$ in these L hash functions.

Theoretical Analysis (Cont.)

Lemma 2

Event E_1 occurs with probability no more than $\frac{1}{e}$.

Proof.

We analyze the case where there is only one r-near neighbor $\mathbf{x}$ of the query point $\mathbf{q}$. Obviously, the more r-near neighbors $\mathbf{q}$ has, the smaller the probability of event E_1 will be.

$$\mathbb{P}[E_1] = \mathbb{P}[g_1(\mathbf{x}) \neq g_1(\mathbf{q}) \wedge g_2(\mathbf{x}) \neq g_2(\mathbf{q}) \cdots g_L(\mathbf{x}) \neq g_L(\mathbf{q})]$$

$$= \mathbb{P}[g_1(\mathbf{x}) \neq g_1(\mathbf{q})] \cdot \mathbb{P}[g_2(\mathbf{x}) \neq g_2(\mathbf{q})] \cdots \mathbb{P}[g_L(\mathbf{x}) \neq g_L(\mathbf{q})]$$

$$= (1 - \frac{1}{n^\rho}) \cdot (1 - \frac{1}{n^\rho}) \cdots (1 - \frac{1}{n^\rho}) = (1 - \frac{1}{n^\rho})^L \text{ (recap: } L = n^\rho\text{)}$$

$$= (1 - 1/L)^L \leq \frac{1}{e}$$

This finishes the proof. □

Theoretical Analysis (Cont.)

Lemma 3

Event E_2 occurs with a probability no more than $\frac{1}{3}$.

Proof.

Let $\mathbf{x}$ be a bad point such that $dist(\mathbf{q}, \mathbf{x}) > c \cdot r$. Let Y be a random variable to indicate the number of bad points examined. As we have shown, for a given hash function $g_i(\cdot)$, the expected number of bad points that collide with $\mathbf{q}$ is bounded by 1. Thus, the expected number of points that collide with the L hash functions is bounded by L. Then, by Markov's inequality, we have:

$$\mathbb{P}[Y \geq 3L] \leq \frac{\mathbb{E}[Y]}{3L} \leq \frac{L}{3L} = \frac{1}{3}.$$

This finishes the proof. $\square$

Theoretical Analysis (Cont.)

Theorem 1

The LSH query algorithm correctly returns an c-approximate r-near neighbor with probability at least $\frac{2}{3} - \frac{1}{e}$.

The above theorem can be derived via a combination of Lemmas 2-3 and the union bound.

Consider the following questions:

- How to increase the success probability to a larger constant?
- In case we want to return the c-approximate nearest neighbor query with a success probability of at least $\frac{2}{3} - \frac{1}{e}$, how should we set the parameters?
- In the median trick, we need to have the success probability larger than $1/2$ to boost the probability. Here, do we need to have the same constraint? Why?

Table of Contents

Intuition: Projection onto random lines (crossing the origin) and divide them into different buckets.

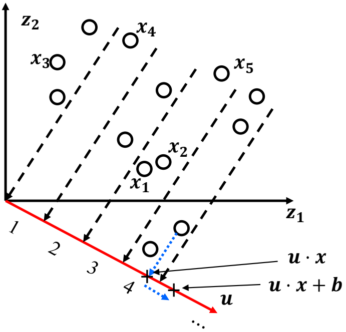
- Version 1: We choose a random line by randomly sampling a unit vector $\mathbf{u}$ and then divide them into different buckets.

$$h_{\mathbf{u}}(\mathbf{x}) = \lceil \frac{\mathbf{u} \cdot \mathbf{x}}{r} \rceil.$$

  Issue: Even if two points are very close, we might still project them into different buckets.

- Final version: We add a random offset $b \in [0, r]$ to the result.

$$h_{\mathbf{u},b}(\mathbf{x}) = \lceil \frac{\mathbf{u} \cdot \mathbf{x} + b}{r} \rceil. \qquad (1)$$



Generating the Random Gaussian Vector u

Next, we show how to generate a random Gaussian vector in $\mathbb{R}^d$:

- Pick d Independent and identically distributed (i.i.d) random variables $Z_1, Z_2, \cdots, Z_d$ from Gaussian distribution $\mathcal{N}(0,1)$. Let $\mathbf{u} = (Z_1, Z_2, \cdots, Z_d)$.
- The vector $\mathbf{u}$ is also called a random Gaussian vector.

Recap: Property of Gaussian distributions.

Lemma 4

Assume that we have two random variables X and Y that are sampled from two independent Gaussian distributions, i.e., $X \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$. Then, their sum $Z = X + Y$ also follows a Gaussian distribution specified as $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$.

Analysis of the Hashing Scheme

Recap that for the hash function:

- We pick a bucket width $r > 0$
- a number b sampled from $(0, r)$ uniformly at random
- and a random Gaussian vector $\mathbf{u}$.

The final hash function is $h_{\mathbf{u},a}(\mathbf{x}) = \lceil \frac{\mathbf{x}\cdot\mathbf{u}+b}{r} \rceil$. We omit the subscript and denote it as $h(\mathbf{x})$ directly when the context is clear.

Lemma 5

Given a vector $\mathbf{x}$ *and a random Gaussian vector* $\mathbf{u}$, *then* $Z = \mathbf{x} \cdot \mathbf{u}$ *follows a Gaussian distribution* $\mathcal{N}(0, (||\mathbf{x}||_2)^2)$ *and* $\mathbb{E}[Z^2] = (||\mathbf{x}||_2)^2$.

Analysis of the Hashing Scheme (Cont.)

Given the hashing scheme $h(\mathbf{x})$, now we consider the probability:

- $p_1 = \mathbb{P}[h(\mathbf{x_1}) = h(\mathbf{x_2})]$ if *dist*$(\mathbf{x_1}, \mathbf{x_2}) = ||\mathbf{x_1} - \mathbf{x_2}||_2 \leq r$.
- $p_2 = \mathbb{P}[h(\mathbf{x_1}) = h(\mathbf{x_2})]$ if *dist*$(\mathbf{x_1}, \mathbf{x_2}) = ||\mathbf{x_1} - \mathbf{x_2}||_2 \geq c \cdot r$

Define $\mathbf{x}' = \mathbf{x_1} - \mathbf{x_2}$.

$$h(\mathbf{x_1}) = h(\mathbf{x_2}) \Leftrightarrow \lceil \frac{\mathbf{x_1} \cdot \mathbf{u} + b}{r} \rceil = \lceil \frac{\mathbf{x_2} \cdot \mathbf{u} + b}{r} \rceil$$

As we have analyzed in Lecture 1,

- if $|\mathbf{x_1} \cdot \mathbf{u} - \mathbf{x_2} \cdot \mathbf{u}| \leq r$, the probability that $h(\mathbf{x_1}) = h(\mathbf{x_2})$ holds is $1 - |\mathbf{x_1} \cdot \mathbf{u} - \mathbf{x_2} \cdot \mathbf{u}|/r$.
- if $|\mathbf{x_1} \cdot \mathbf{u} - \mathbf{x_2} \cdot \mathbf{u}| > r$, $h(\mathbf{x_1}) \neq h(\mathbf{x_2})$.

Analysis of the Hashing Scheme (Cont.)

Define $Z = \mathbf{x_1} \cdot \mathbf{u} - \mathbf{x_2} \cdot \mathbf{u}$. Then, Z follows a Gaussian distribution of $\mathcal{N}(0, (||\mathbf{x_1} - \mathbf{x_2}||_2)^2)$.

Then, the probability distribution of the event $h(\mathbf{x_1}) = h(\mathbf{x_2})$ is:

$$\mathbb{P}[h(\mathbf{x_1}) = h(\mathbf{x_2})] = \int_0^r \mathbb{P}[h(\mathbf{x_1}) = h(\mathbf{x_2}) || Z| = y] \cdot f(y)\, dy,$$

where $f(y)$ (with $y \geq 0$) is the density function of $|\mathcal{N}(0, Z^2)|$. Since $\mathbb{P}[h(\mathbf{x_1}) = h(\mathbf{x_2}) \mid |Z| = y] = 1 - \frac{y}{r}$, we have that:

$$\begin{aligned}\mathbb{P}[h(\mathbf{x_1}) = h(\mathbf{x_2})] &= \int_0^r (1 - \frac{y}{r}) \cdot f(y)\, dy \\ &= \int_0^r \frac{1}{||\mathbf{x_1} - \mathbf{x_2}||_2} f(\frac{y}{||\mathbf{x_1} - \mathbf{x_2}||_2})(1 - \frac{y}{r})\, dy,\end{aligned}$$

where $f(y)$ is the density function of $|\mathcal{N}(0, 1)|$.

Analysis of the Hashing Scheme (Cont.)

Now we make a connection between the distance of $\mathbf{x_1}$ and $\mathbf{x_2}$ and the probability that they will collide. Define the probability derived on previous page as $p(||\mathbf{x_1} - \mathbf{x_2}||)$.

- Clearly, $p_1 = p(r)$;
- $p_2 = p(c \cdot r)$.

We want to bound $\rho = \log(1/p(r))/\log(1/p(c \cdot r))$. Here, notice that for different data points $\mathbf{x_1}$ and $\mathbf{x_2}$, p_1 and p_2 clearly depend on these two inputs and can be different for different input $\mathbf{x_1}$ and $\mathbf{x_2}$.

Lemma 6 ([2])

Given the LSH designed with Equation 1, $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$ is bounded by $\frac{1}{c}$.

Table of Contents

Cosine Distance

Cosine similarity between two vectors $\mathbf{x}$ and $\mathbf{y}$:

$$cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||_2 \cdot ||\mathbf{y}||_2}$$

The cosine distance measures the angle between the two vectors:

$$dist_{cos}(\mathbf{x}, \mathbf{y}) = \arccos\left(\frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||_2 \cdot ||\mathbf{y}||_2}\right).$$

Easy to verify:

- When $\mathbf{x}$ and $\mathbf{y}$ point to opposite directions, their cosine distance is the largest, which is π.
- When $\mathbf{x}$ and $\mathbf{y}$ point to the same direction, their cosine distance is the smallest, which is 0.

LSH for Cosine Distance: SimHash [3]

SimHash is designed for the cosine distance. Given a sampled random Gaussian unit vector $\mathbf{u}$, $h_{\mathbf{u}}(\mathbf{x})$ is defined as:

$$h_{\mathbf{u}}(\mathbf{x}) = \text{sign}(\mathbf{u} \cdot \mathbf{x}),$$

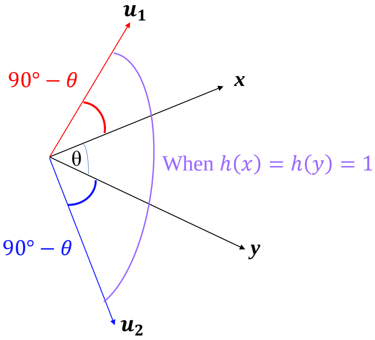
where sign outputs 1 if the input is ≥ 0 and otherwise outputs -1.

Lemma 7

Given two data points $\mathbf{x}$ *and* $\mathbf{y}$ *and a SimHash function* $h_{\mathbf{u}}(\cdot)$ *The probability that the hash values of* $\mathbf{x}$ *and* $\mathbf{y}$ *are equal, i.e., either both are 1 or both are -1, is as follows:*

$$\mathbb{P}[h_{\mathbf{u}}(\mathbf{x}) = h_{\mathbf{u}}(\mathbf{y})] = 1 - \frac{dist_{cos}(\mathbf{x}, \mathbf{y})}{\pi}$$

We can prove Lemma 7 by drawing the figure as shown on the right-hand side.



We further consider $\rho = \log(1/p_1)/\log(1/p_2)$.

$$\rho = \frac{\ln(1/(1 - r/\pi))}{\ln(1/(1 - cr/\pi))}$$

By using the following inequality:

$$\frac{x}{x+1} \leq \ln(1+x) \leq \frac{x(6+x)}{6+4x} \leq x \text{ for } x > -1.$$

We can show that $\rho \leq \frac{1}{c}$ for $c \geq 2$.

Table of Contents

Jaccard Distance for Binary Vectors

Given two binary vectors $\mathbf{x}$ and $\mathbf{y}$, we first map them to two sets X and Y, where the i-th non-zero entry indicates the existence of element i in the set. The Jaccard similarity $J(\mathbf{x}, \mathbf{y})$ is defined as:

$$J(\mathbf{x}, \mathbf{y}) = \frac{|X \cap Y|}{|X \cup Y|}.$$

The Jaccard distance $dist_{Jaccard}(\mathbf{x}, \mathbf{y})$ is defined as $1 - J(\mathbf{x}, \mathbf{y})$.

Example 1

Let $\mathbf{x} = (1,0,0,0,1,0,1,0,1,1)$ and $\mathbf{y} = (0,0,0,0,1,0,1,0,0,0)$. Then, the Jaccard similarity between $\mathbf{x}$ and $\mathbf{y}$ is

$$J(\mathbf{x}, \mathbf{y}) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{\{1,5,7,9,10\} \cap \{5,7\}}{\{1,5,7,9,10\} \cup \{5,7\}} = \frac{2}{5}$$

Then, $dist_{Jaccard}(\mathbf{x}, \mathbf{y}) = 1 - J(\mathbf{x}, \mathbf{y}) = \frac{3}{5}$.

MinHash for Jaccard Distance [4]

Generate a random permutation P for $[1 \cdots d]$. For the i-th dimension, the hashed value is $P[i]$, i.e., the value in the i-th dimension of the permutation P. Let $h(\mathbf{x}) = min_{\mathbf{x}(i)=1} P(\mathbf{x}(i))$ be the minimum hashed value among the non-zero entries. Then,

$$\mathbb{P}[h(\mathbf{x} = \mathbf{y}] = J(\mathbf{x}, \mathbf{y}) = 1 - dist_{Jaccard}(\mathbf{x}, \mathbf{y}).$$

Example 2

Let $\mathbf{x} = (1, 0, 0, 0, 1, 0, 1, 0, 1, 1)$ and $\mathbf{y} = (0, 0, 0, 0, 1, 0, 1, 0, 0, 0)$. Assume that the random permutation of $[1 \cdots 10]$ is $[4, 2, 10, 5, 1, 3, 8, 7, 9, 6]$. Then, $h(\mathbf{x}) = \min\{4, 1, 8, 9, 6\} = 1$ and $h(\mathbf{y}) = min\{1, 8\} = 1$.

We can verify that $\rho = \log(1/p_1)/\log(1/p_2)$ can be bounded by $\frac{1}{c}$ if $c \geq 2$.

References

Piotr Indyk and Rajeev Motwani.
Approximate nearest neighbors: Towards removing the curse of dimensionality.
In *STOC*, pages 604–613, 1998.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni.
Locality-sensitive hashing scheme based on p-stable distributions.
In *SOCG*, pages 253–262, 2004.

Moses Charikar.
Similarity estimation techniques from rounding algorithms.
In *STOC*, pages 380–388, 2002.

Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher.
Min-wise independent permutations (extended abstract).
In *STOC*, pages 327–336, 1998.