Locality-Sensitive Hashing (LSH)

Shusen Wang

Revisit Collision-Resistant Hashing

- Let **h** be a collision-resistant hash function.
- Even if **x** is close to **y**, **h(x)** is entire different from **h(y)**.
    - **x = "ABCDEFTHIJKLMNOPQRST".**
    - **y = "ABCKEFTHIJKLMNOPQRST".**
    - $\|\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{y})\|$ must be very big.

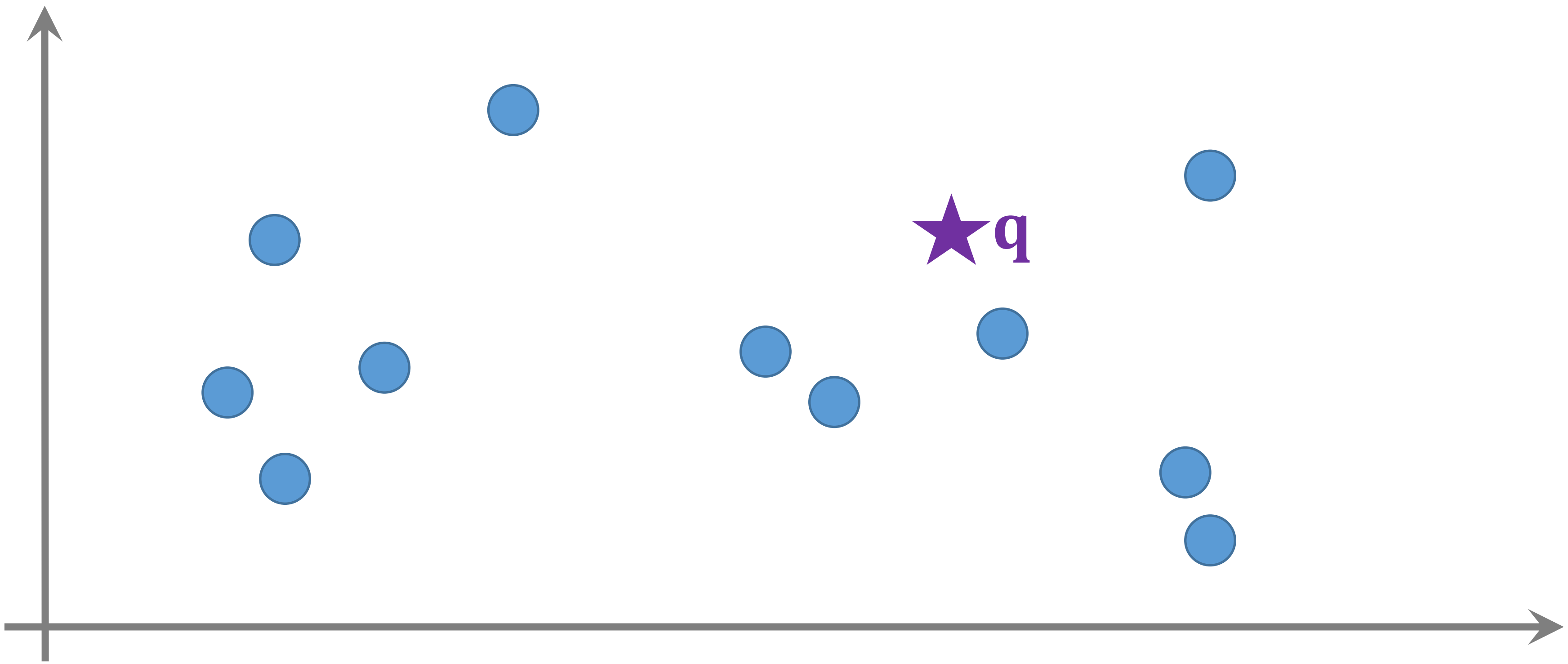
Locality Sensitive Hashing (LSH)

- Locality sensitive hashing is the opposite of collision-resistant hashing.
- If **x** is close to **y**, then **h(x)** must be close to **h(y)**.
  - **x = [3,4,1,0,9,4,8,3].**
  - **y = [3,3,3,0,9,4,8,3].**
  - $||\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{y})||$ must be small.

Application: Nearest Neighbor Search

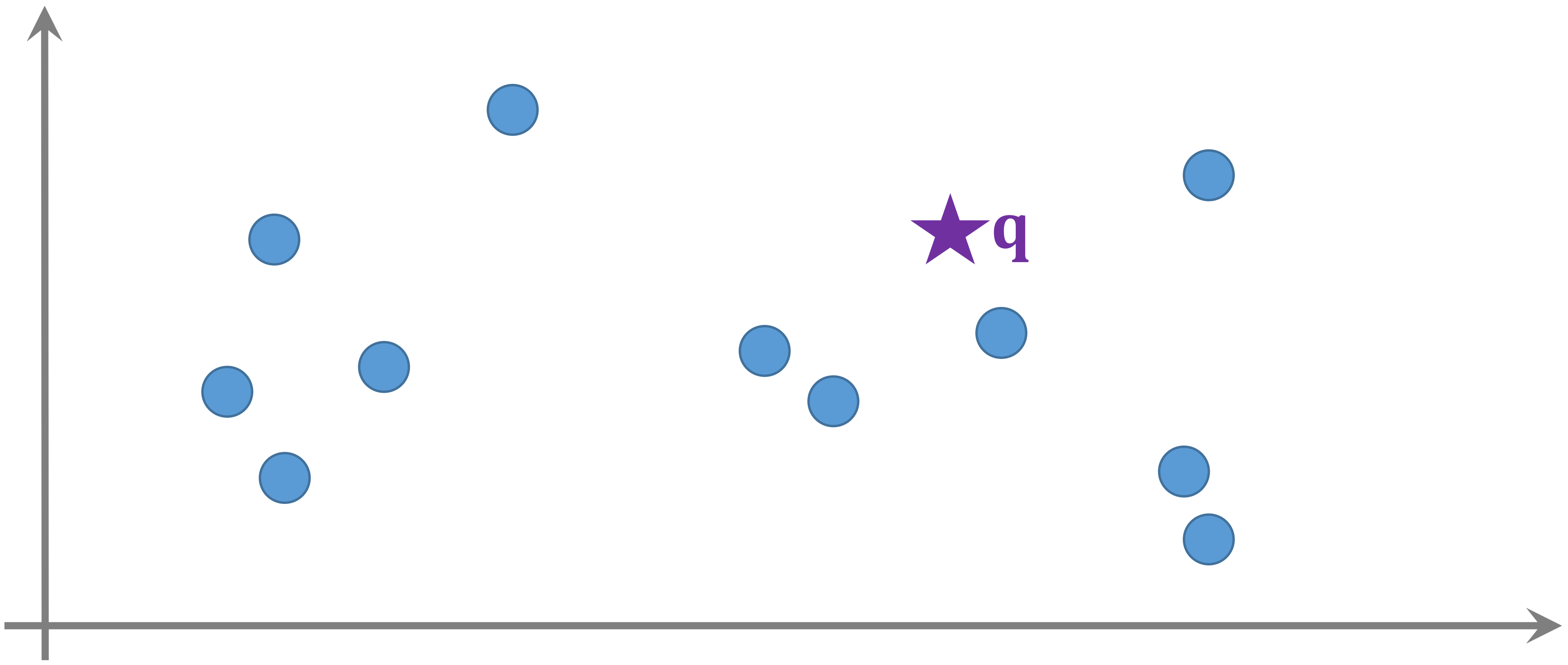
Nearest Neighbor Search

- Given a set of vectors, $\mathbf{x}_1, \cdots, \mathbf{x}_n$, e.g., features of images, documents, etc.
- Nearest Neighbor Search: For a query $\mathbf{q}$, find its nearest $\mathbf{x}$.



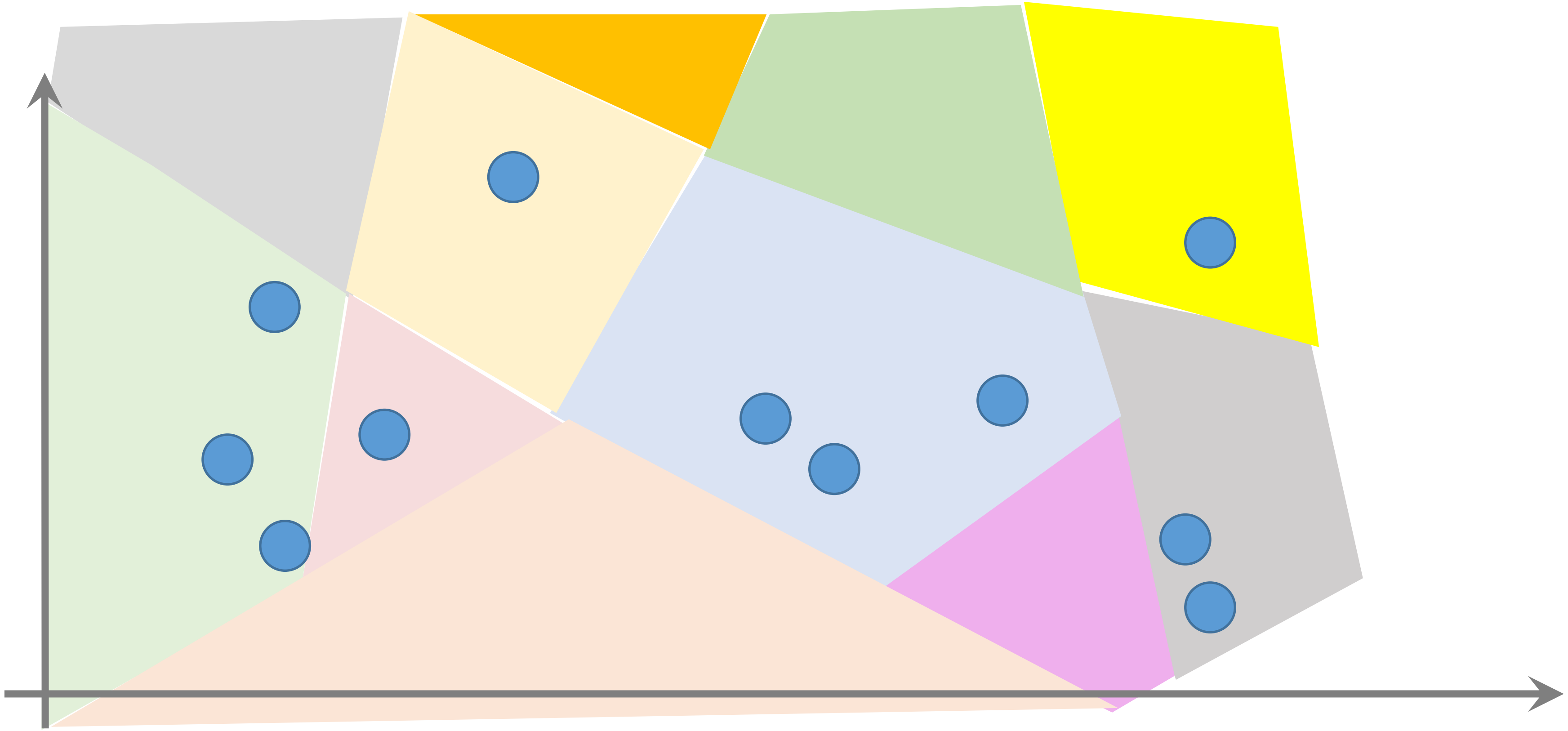
Nearest Neighbor Search

- Naïve algorithm: compare $\mathbf{q}$ with every $\mathbf{x}$; $O(n)$ time complexity.
- There may be billions of queries every day. (E.g., search engine.)



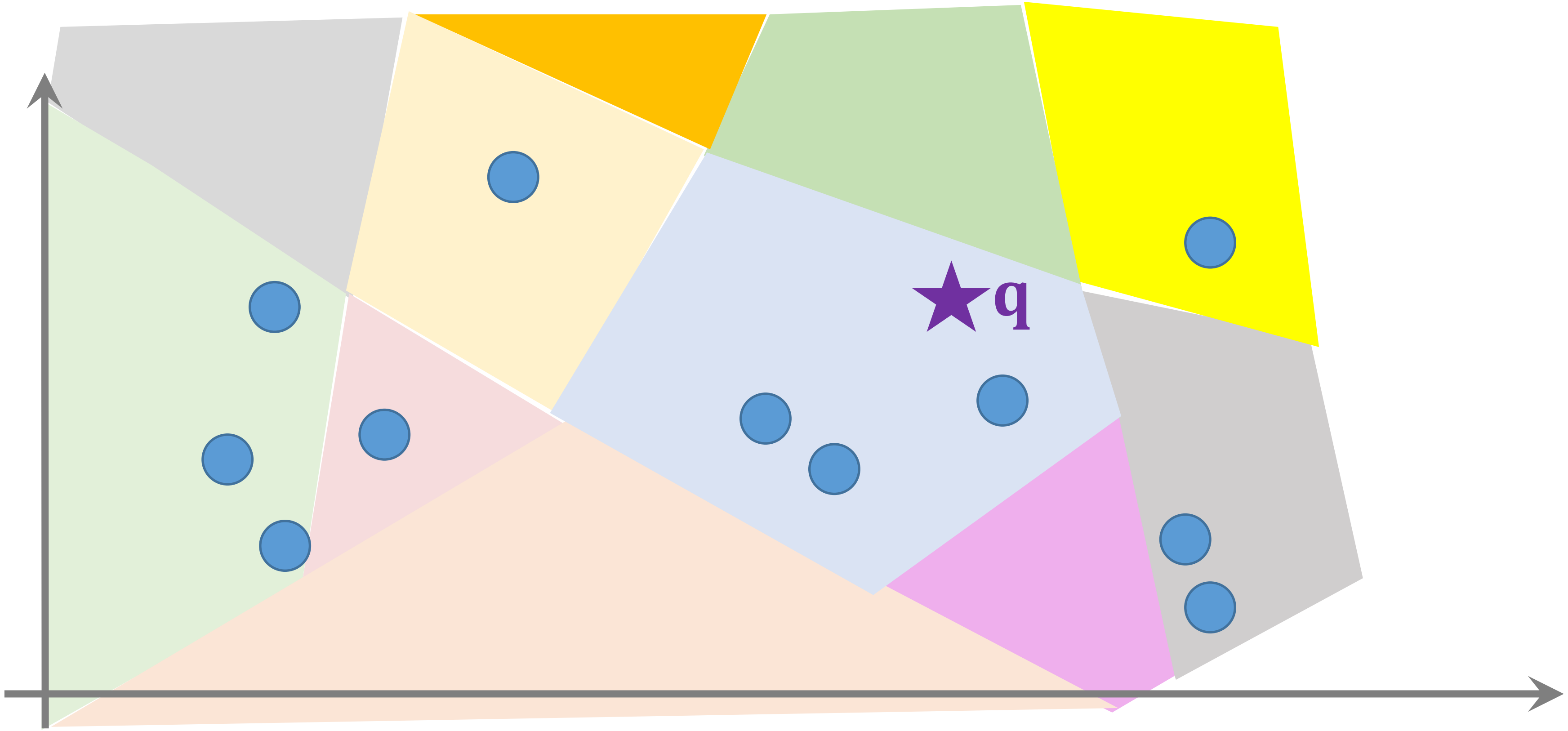
Efficient Algorithm: Main Idea

- Partition the space to many regions (buckets).



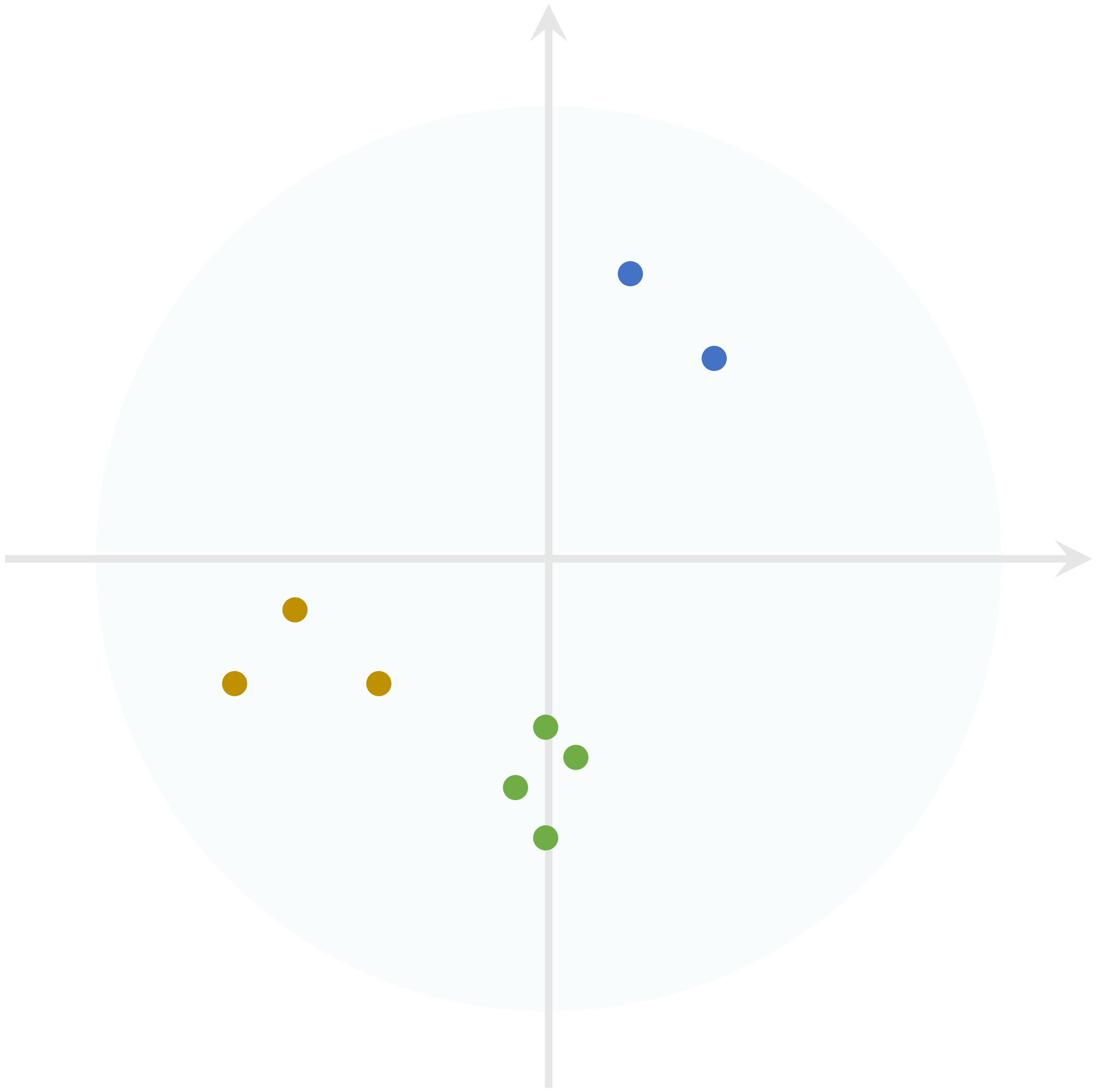
Efficient Algorithm: Main Idea

- Partition the space to many regions (buckets).
- Compare the query with only the points in the same bucket.

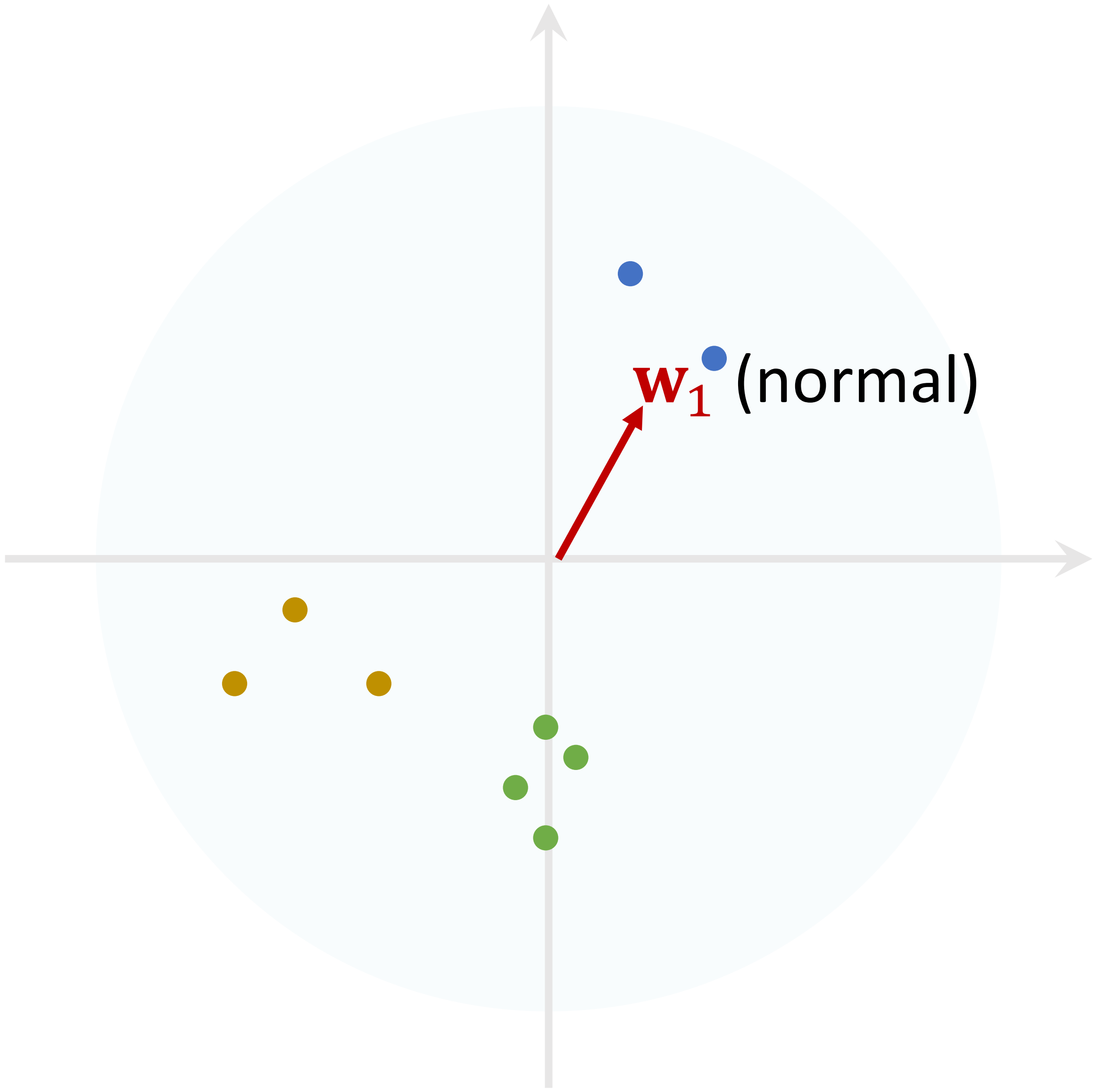


LSH via Random Projection

Input Data

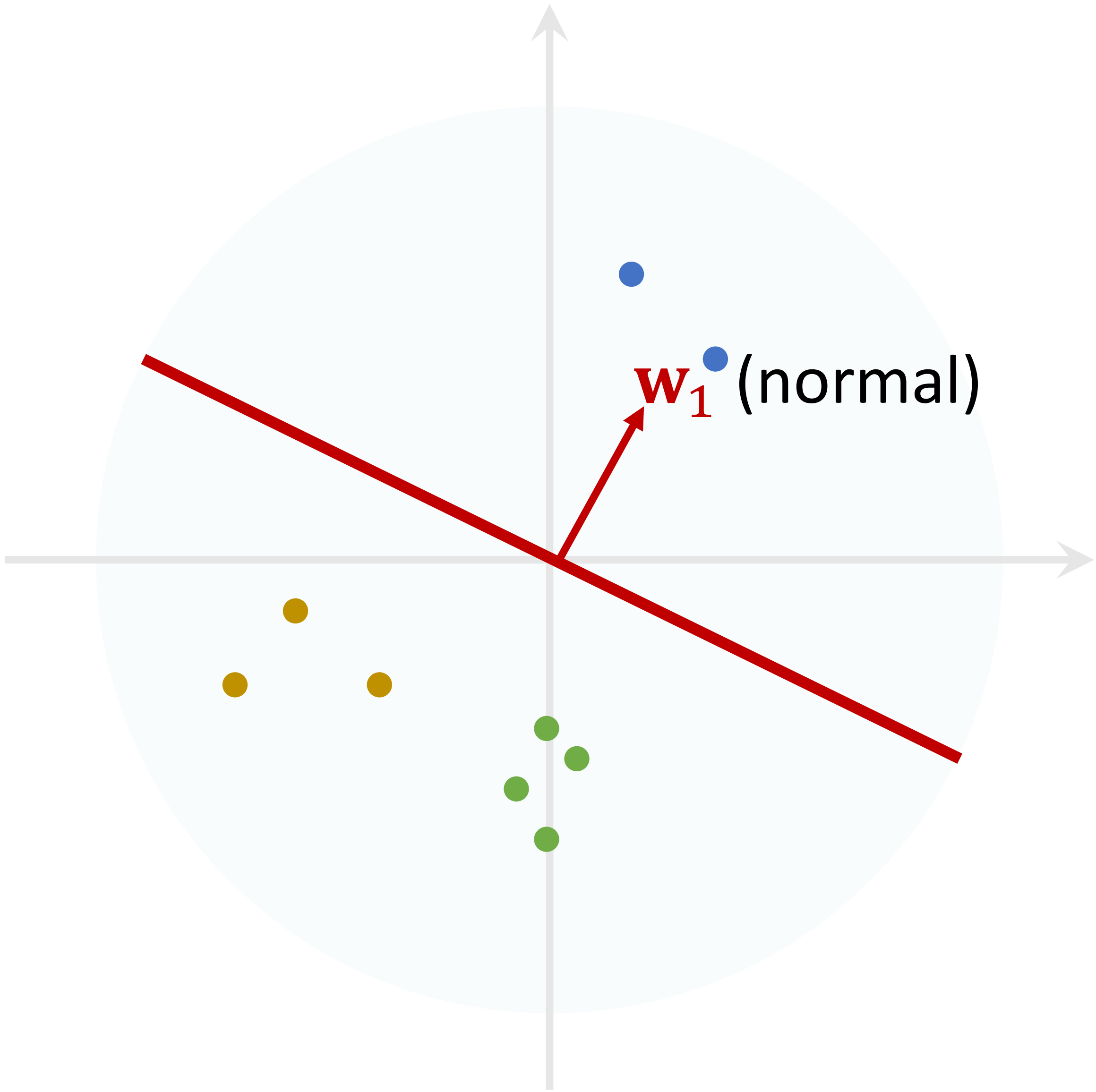


Partition the Space



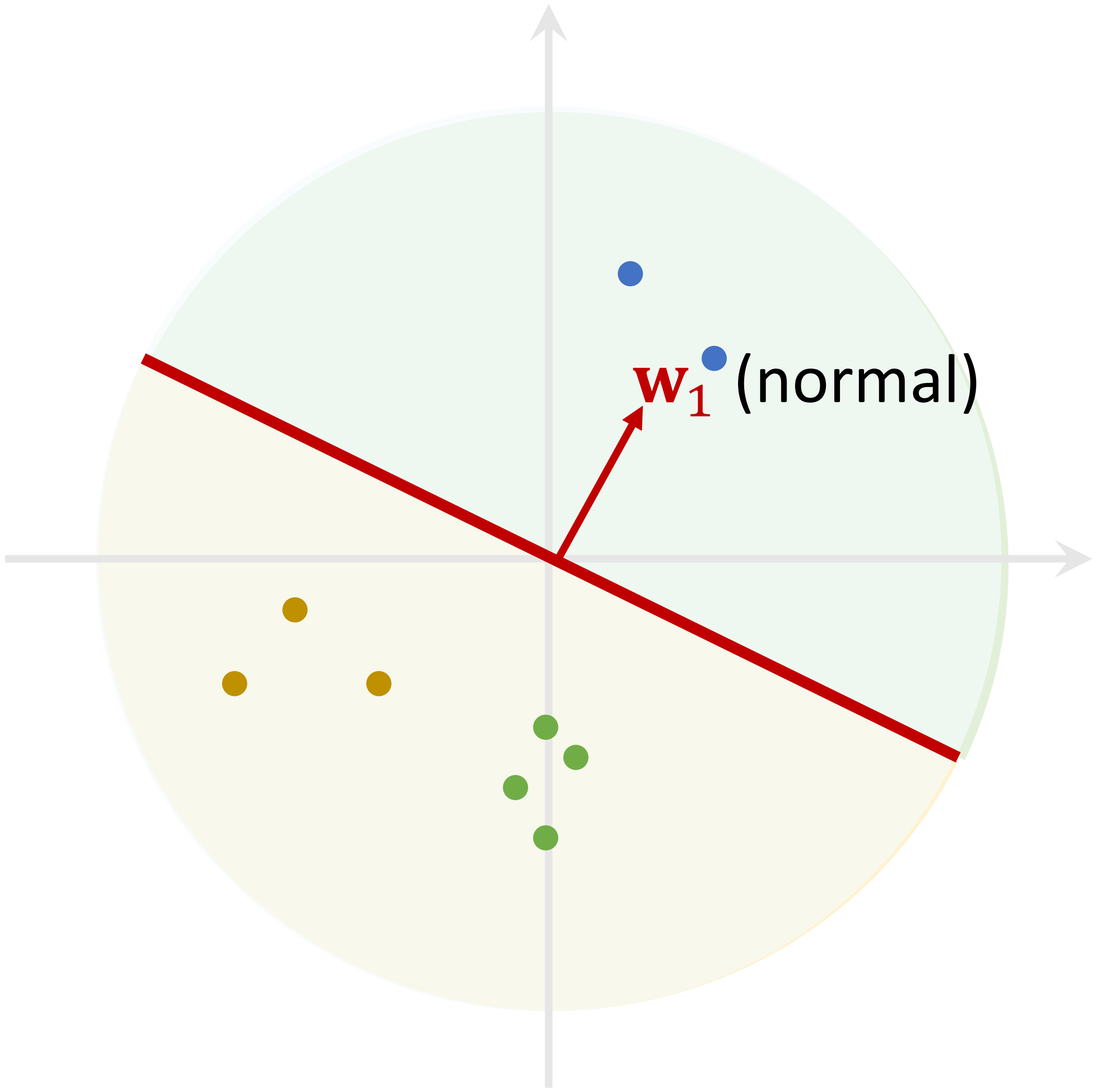
- Randomly sample a direction $\mathbf{w}_1$ as the normal.
    1. $\mathbf{v} \sim N(\mathbf{0}, \mathbf{I})$.
    2. $\mathbf{w}_1 = \mathbf{v} \;/\; ||\mathbf{v}||_2$.

Partition the Space

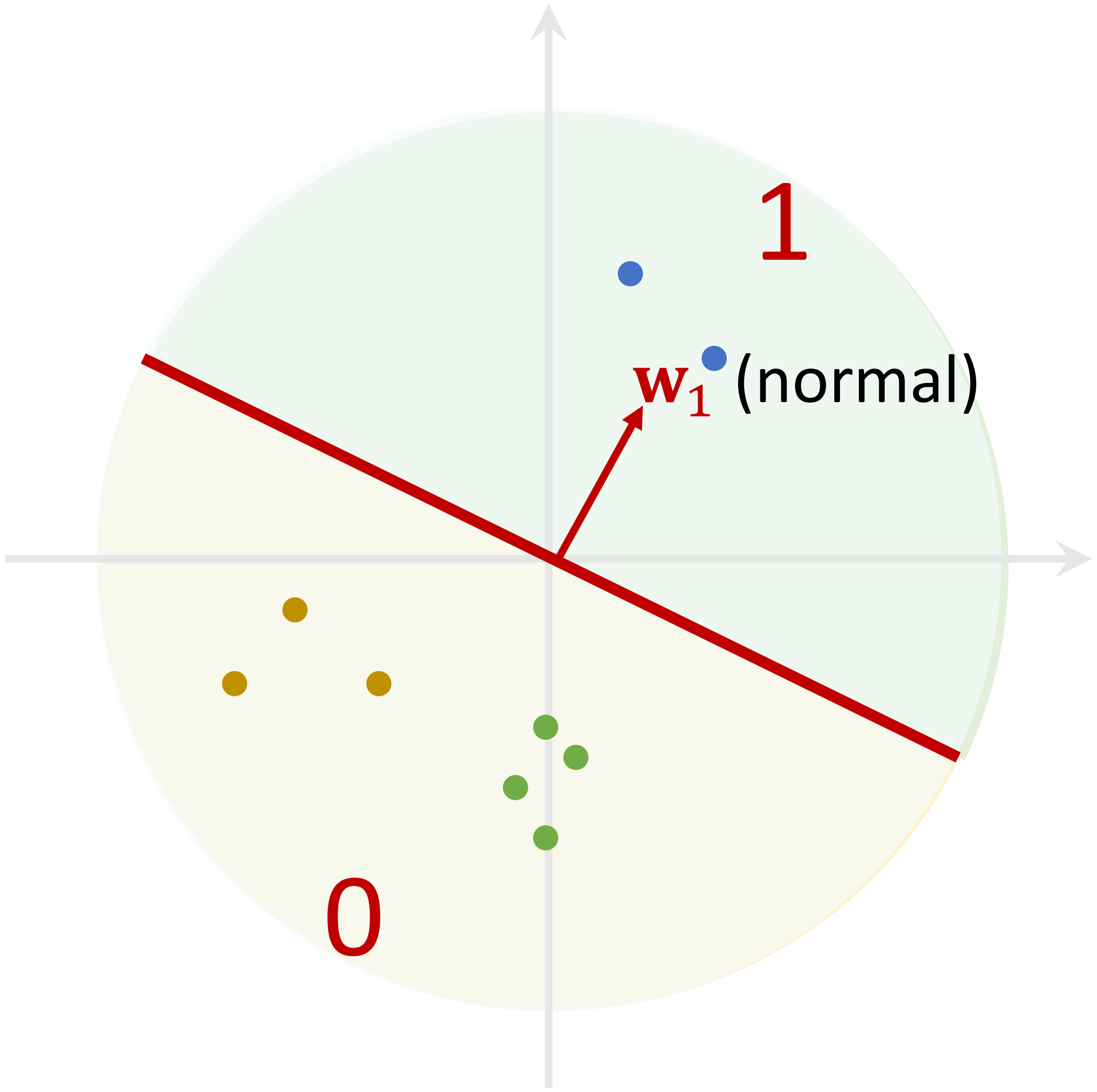


- Randomly sample a direction $\mathbf{w}_1$ as the normal.
    1. $\mathbf{v} \sim N(\mathbf{0}, \mathbf{I})$.
    2. $\mathbf{w}_1 = \mathbf{v} \,/\, \|\mathbf{v}\|_2$.

Partition the Space

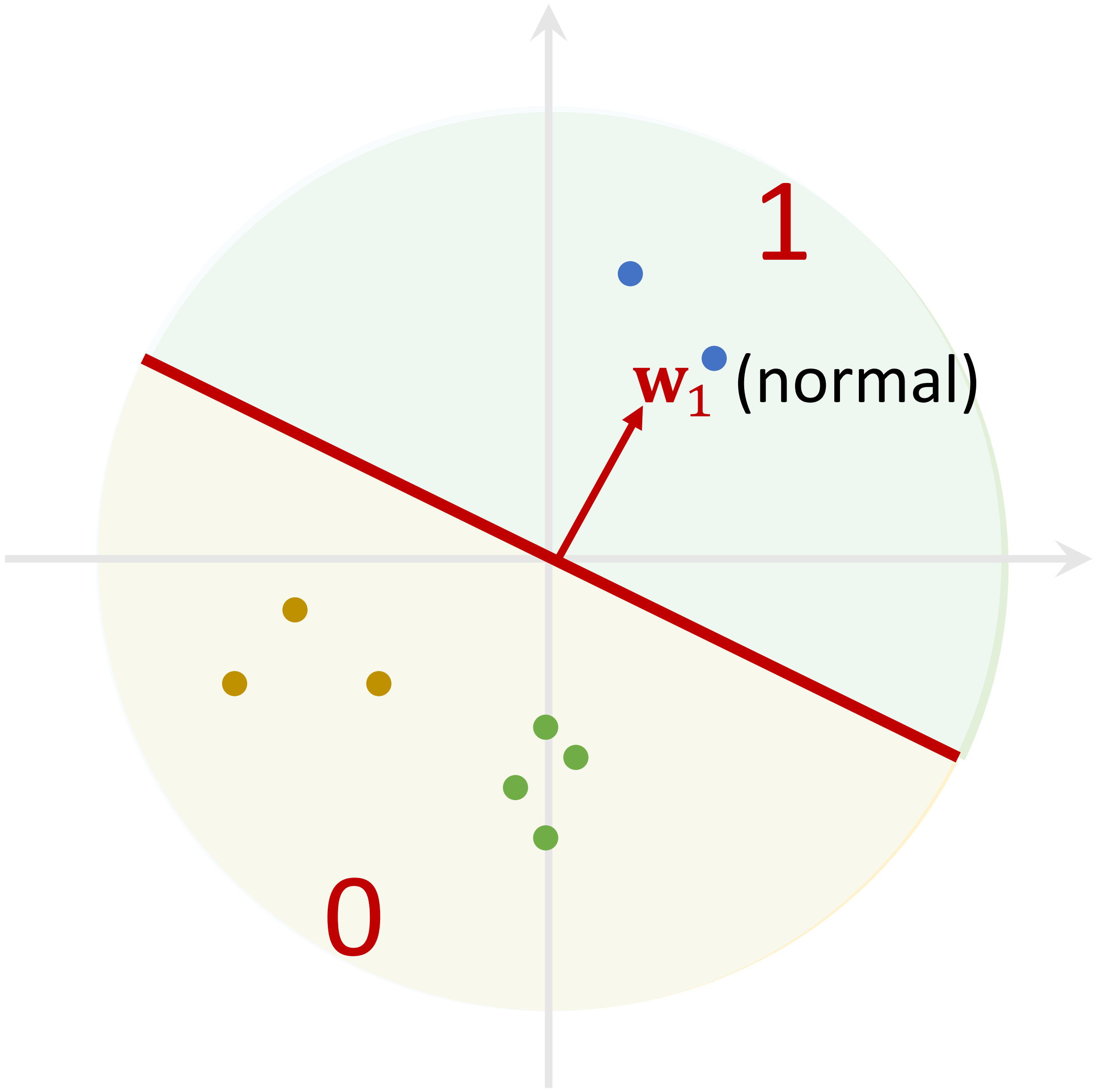


Partition the Space



- Randomly sample a direction $\mathbf{w}_1$ as the normal.
  1. $\mathbf{v} \sim N(\mathbf{0}, \mathbf{I})$.
  2. $\mathbf{w}_1 = \mathbf{v} \,/\, \left\|\mathbf{v}\right\|_2$.

Partition the Space

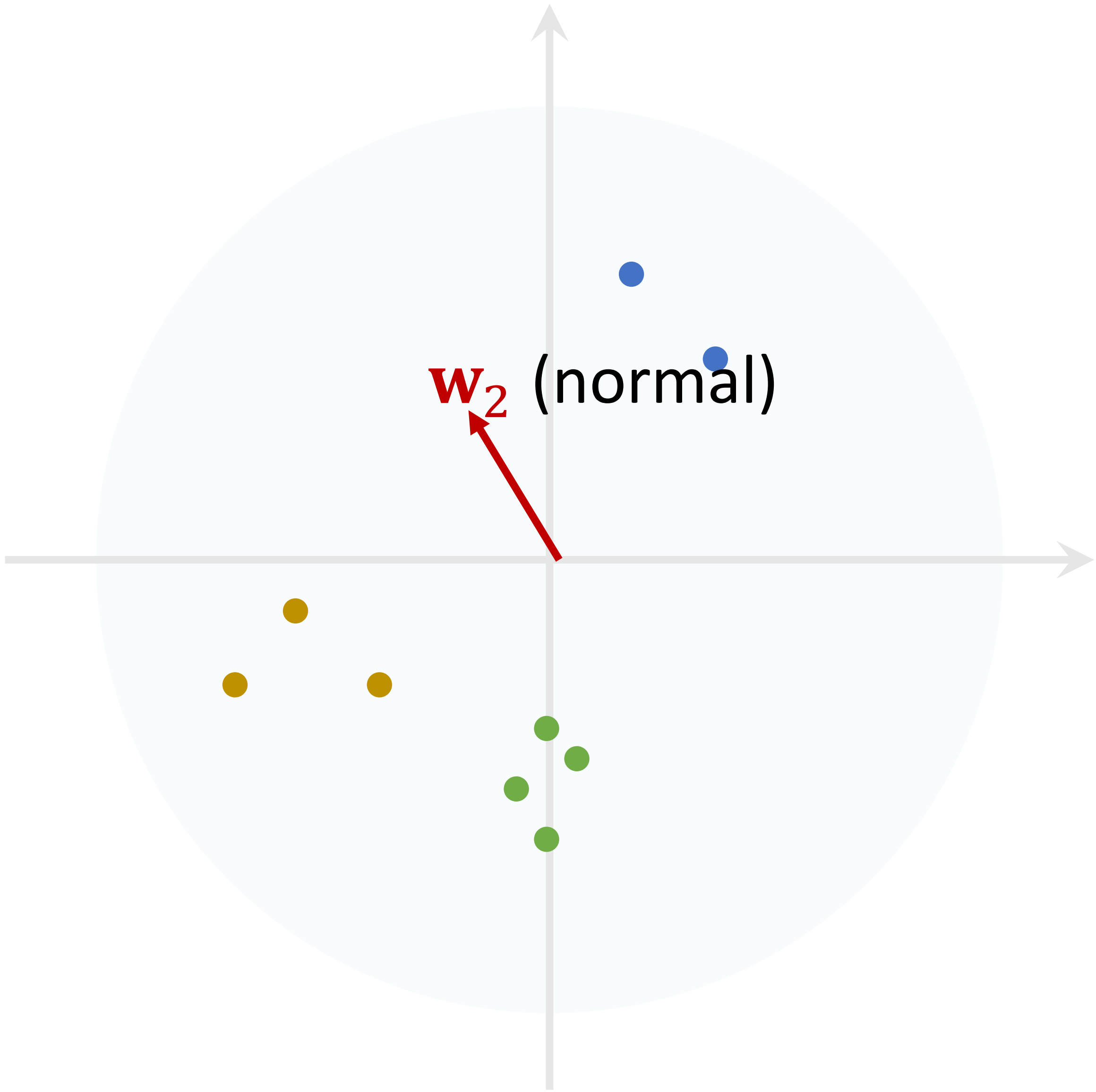


- Randomly sample a direction $\mathbf{w}_1$ as the normal.
    1. $\mathbf{v} \sim N(\mathbf{0}, \mathbf{I})$.
    2. $\mathbf{w}_1 = \mathbf{v} \,/\, \|\mathbf{v}\|_2$.

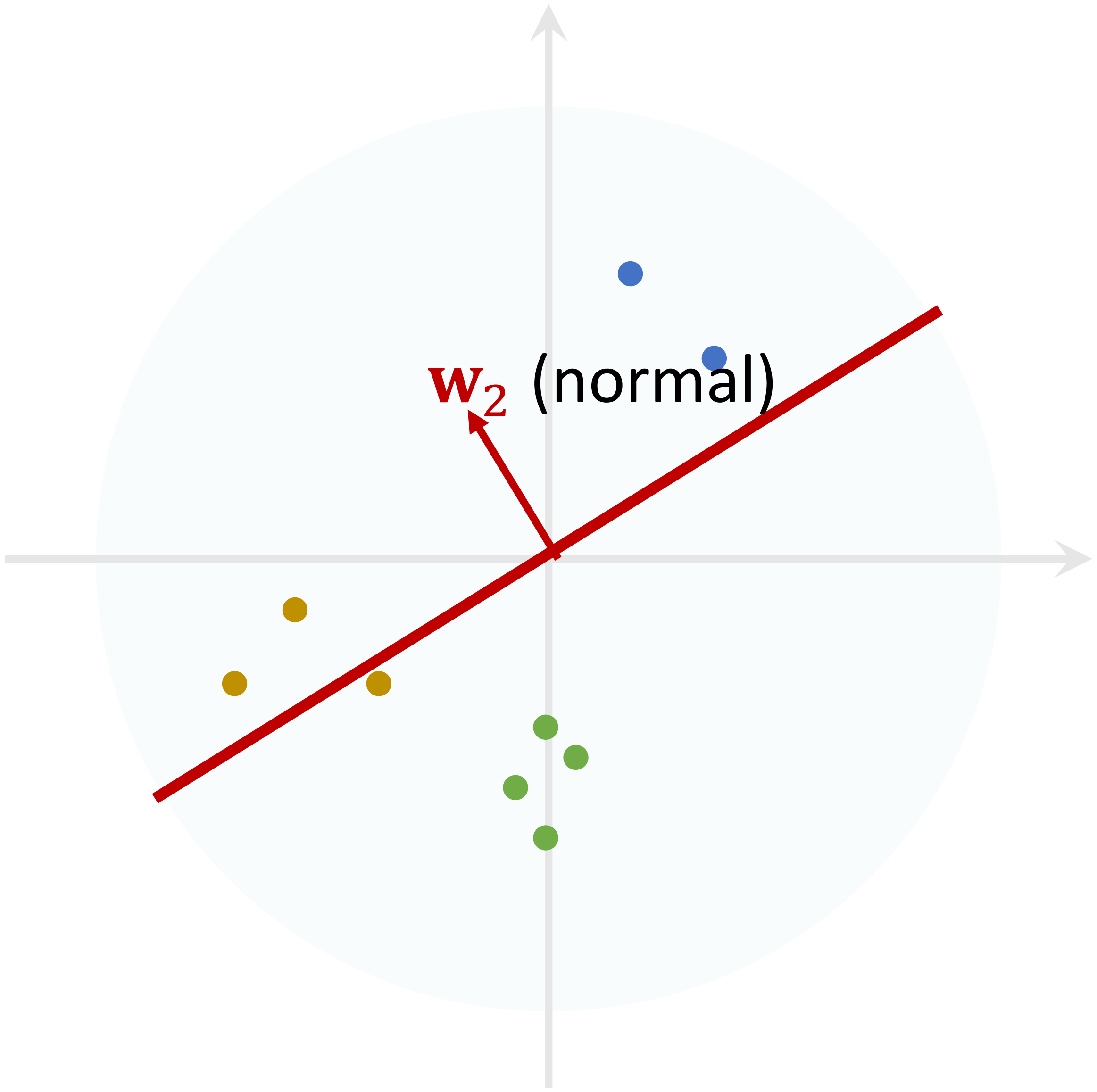
- Hash function: for any point $\mathbf{x}$,

$$h_1(\mathbf{x}) = \begin{cases} 1, & \boxed{\text{if } \mathbf{x}^T \mathbf{w}_1 \geq 0;} \\ 0, & \text{otherwise.} \end{cases}$$

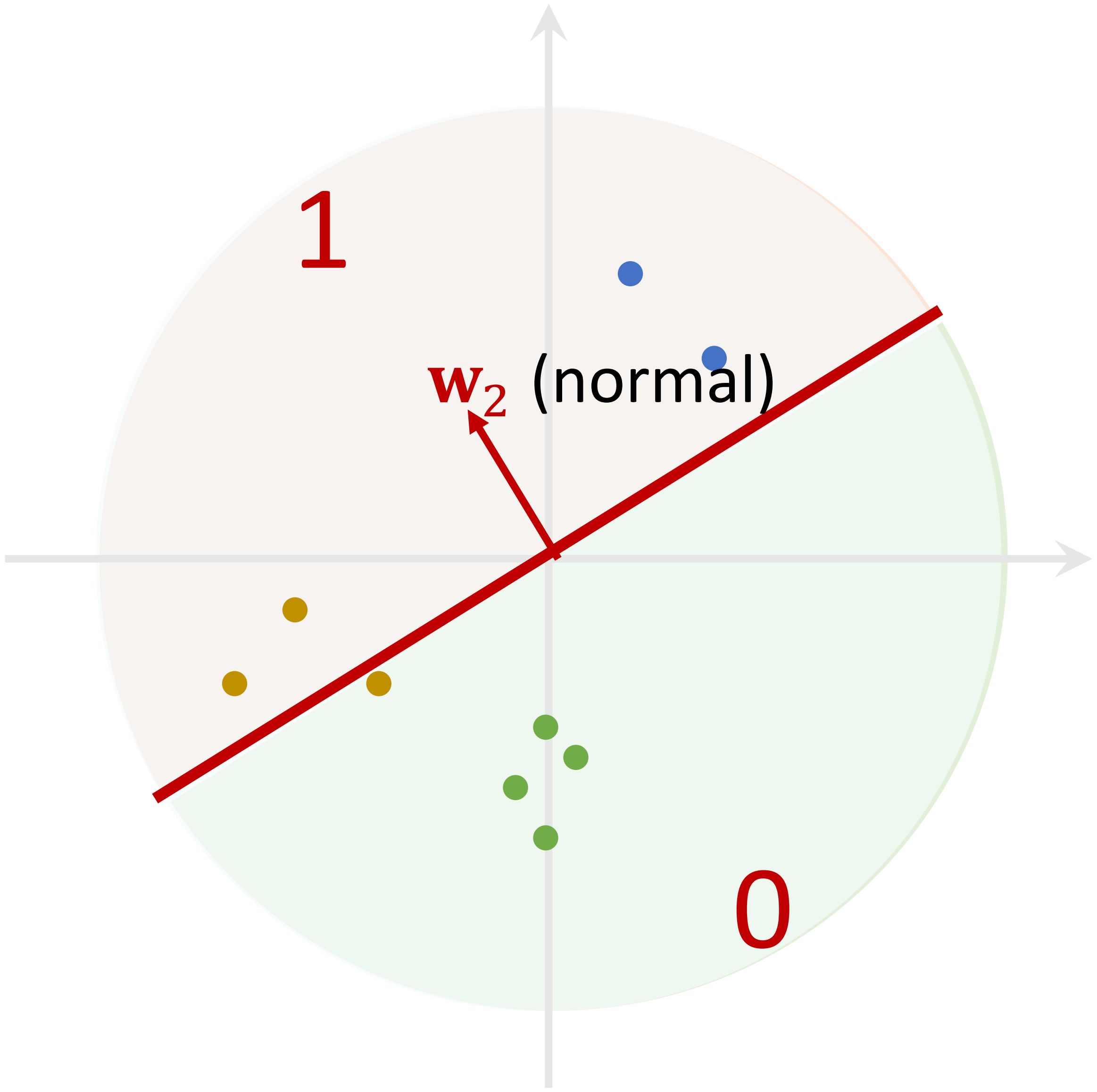
Partition the Space



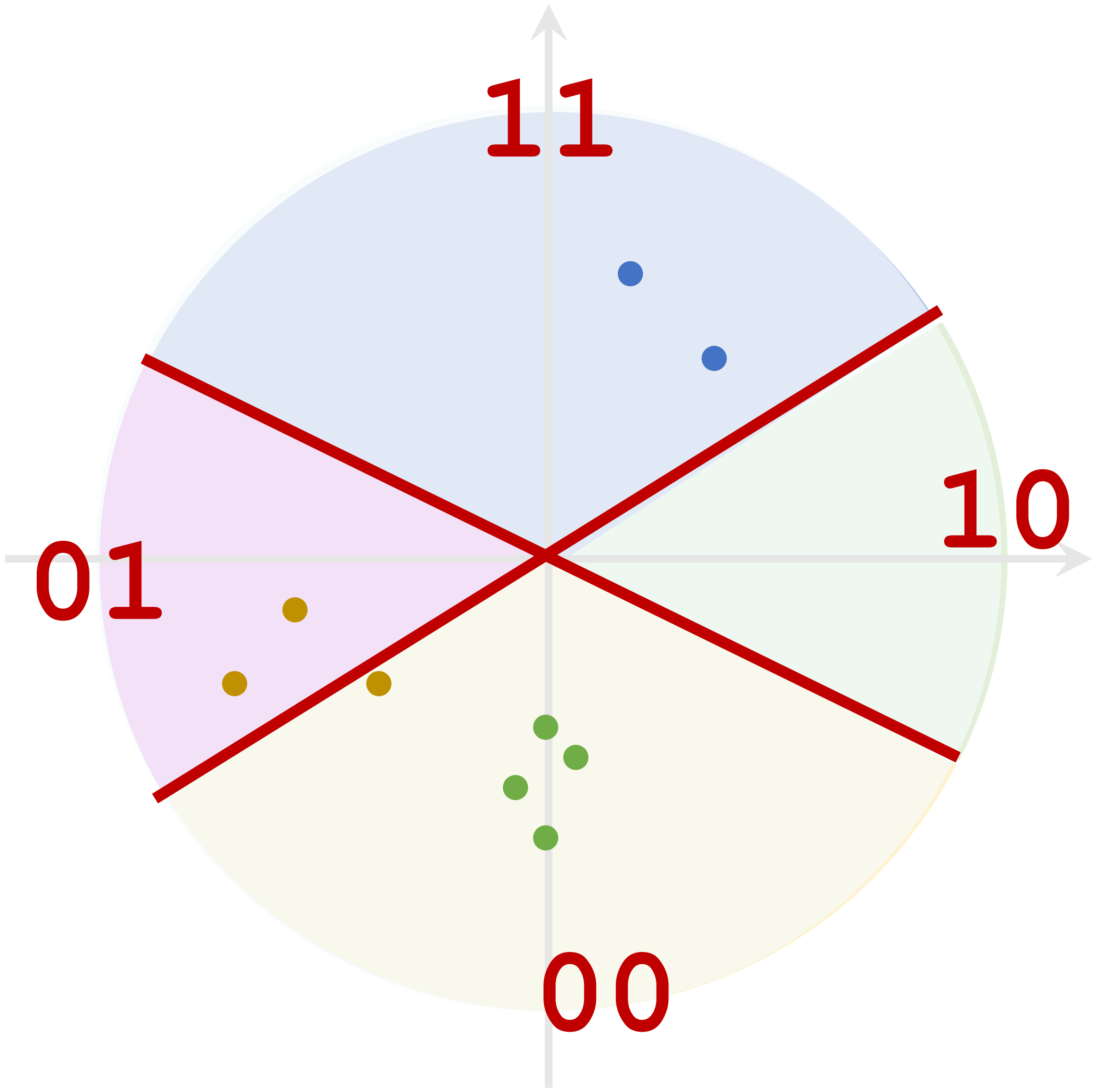
Partition the Space



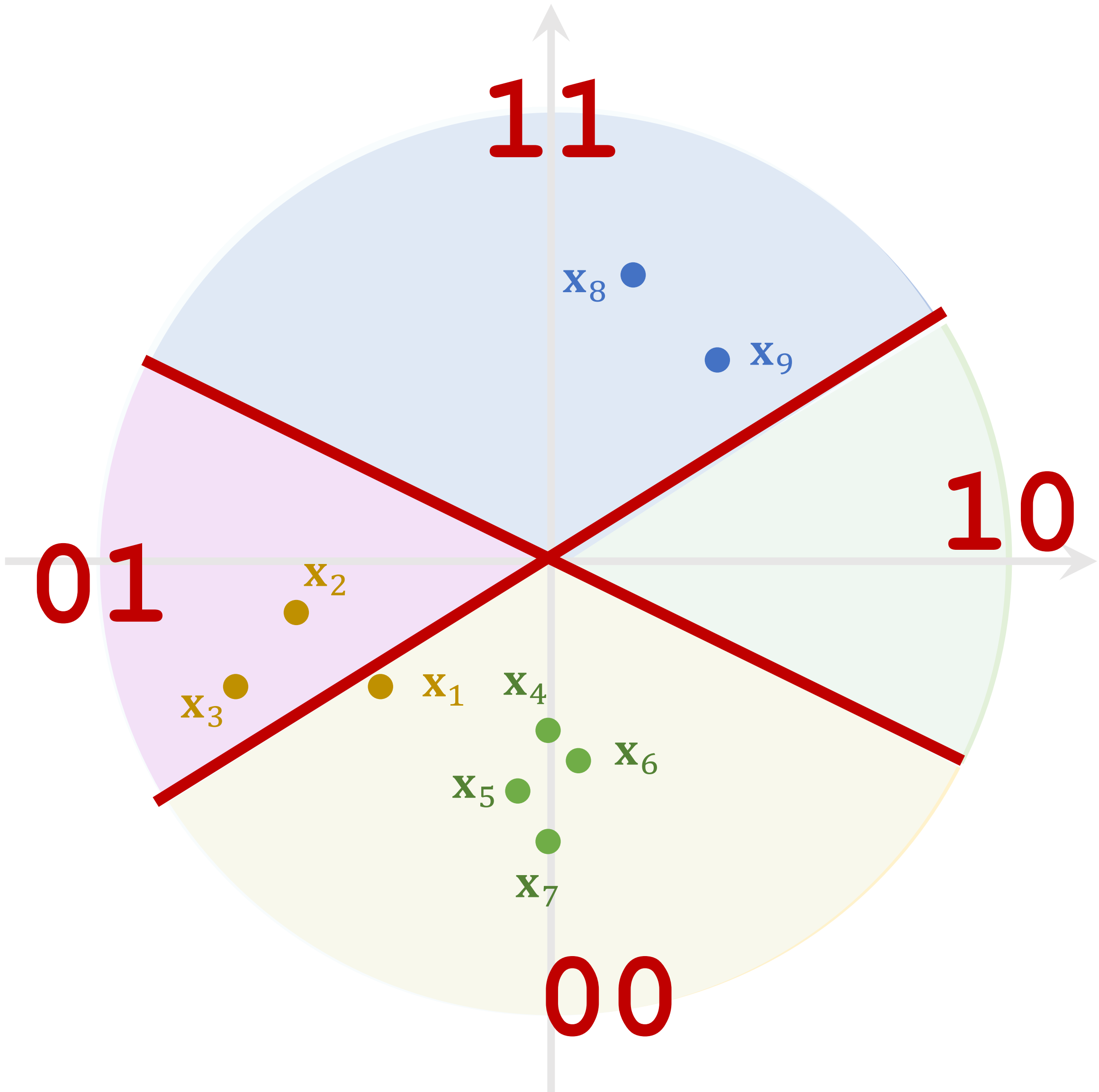
Partition the Space



Partition the Space

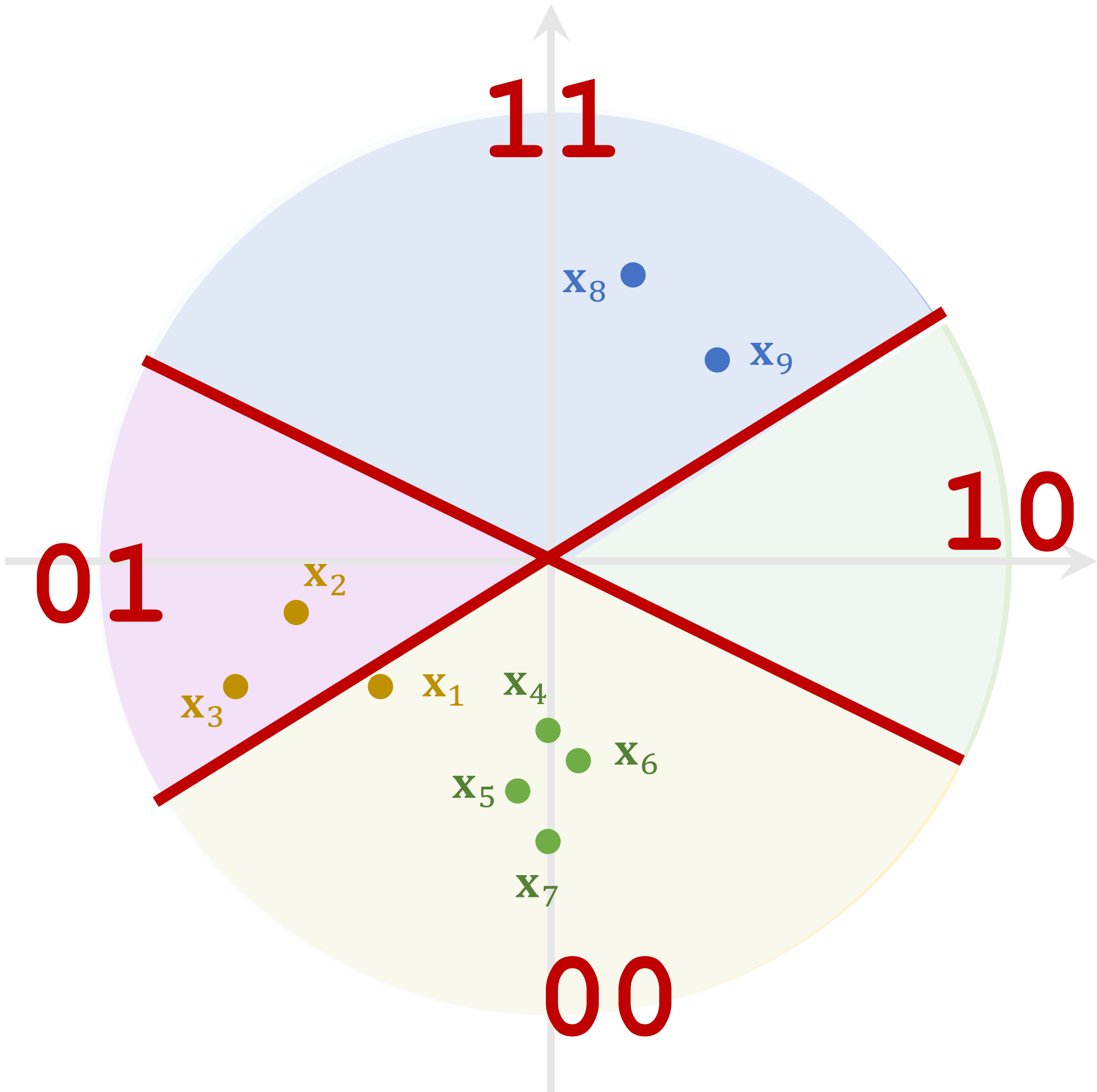


Build Buckets



Index	Data
00	x_1, x_4, x_5, x_6, x_7
01	x_2, x_3
10	
11	x_8, x_9

Build Buckets

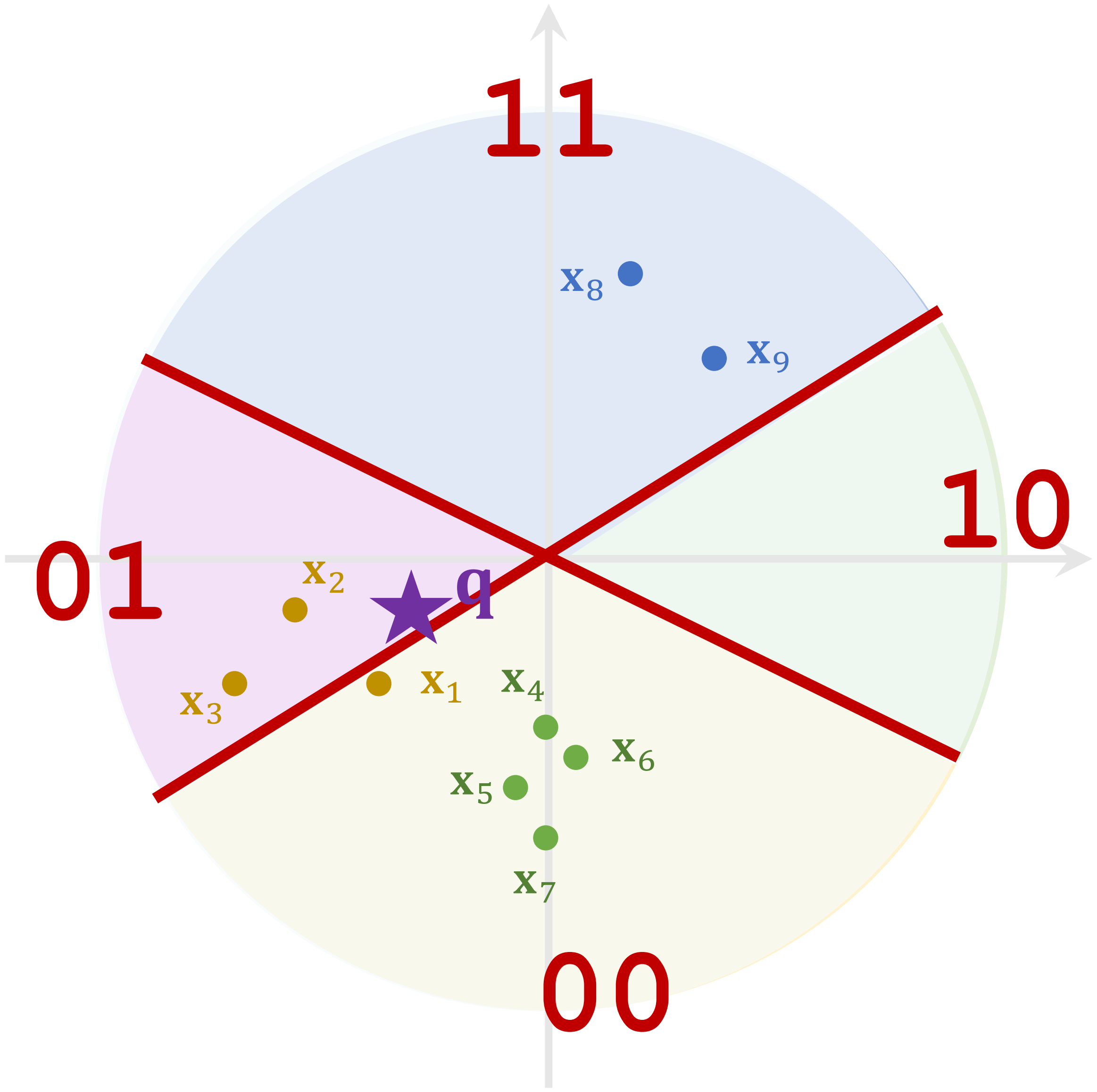


Index	Data
00	x_1, x_4, x_5, x_6, x_7
01	x_2, x_3
10	
11	x_8, x_9

One Bucket

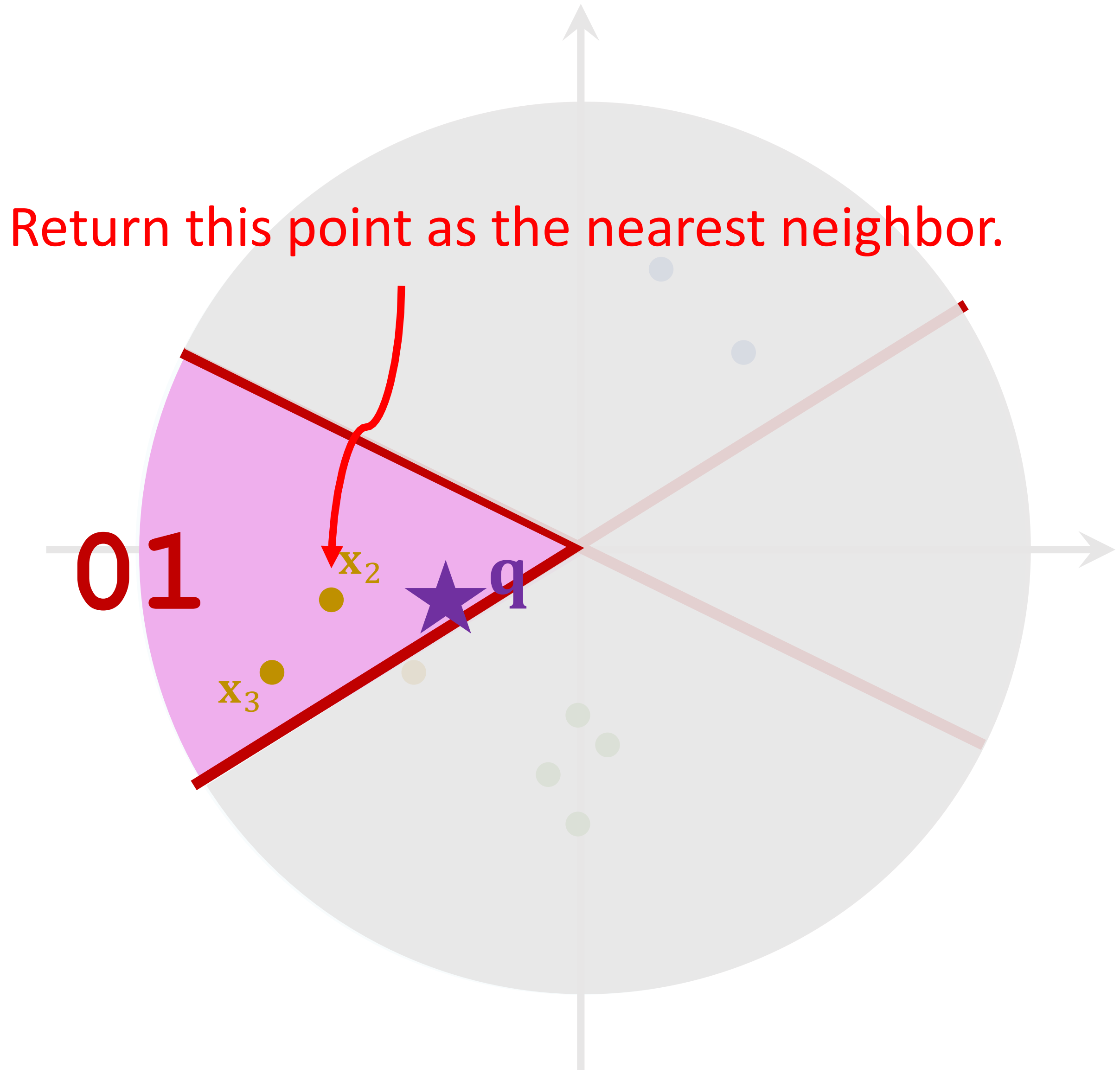
Nearest Neighbor Search Using LSH

Nearest Neighbor Search



- Given a query $\mathbf{q}$, find its nearest point.
- Hashing: $h_1(\mathbf{q}) = 0,\ h_2(\mathbf{q}) = 1$.

Nearest Neighbor Search



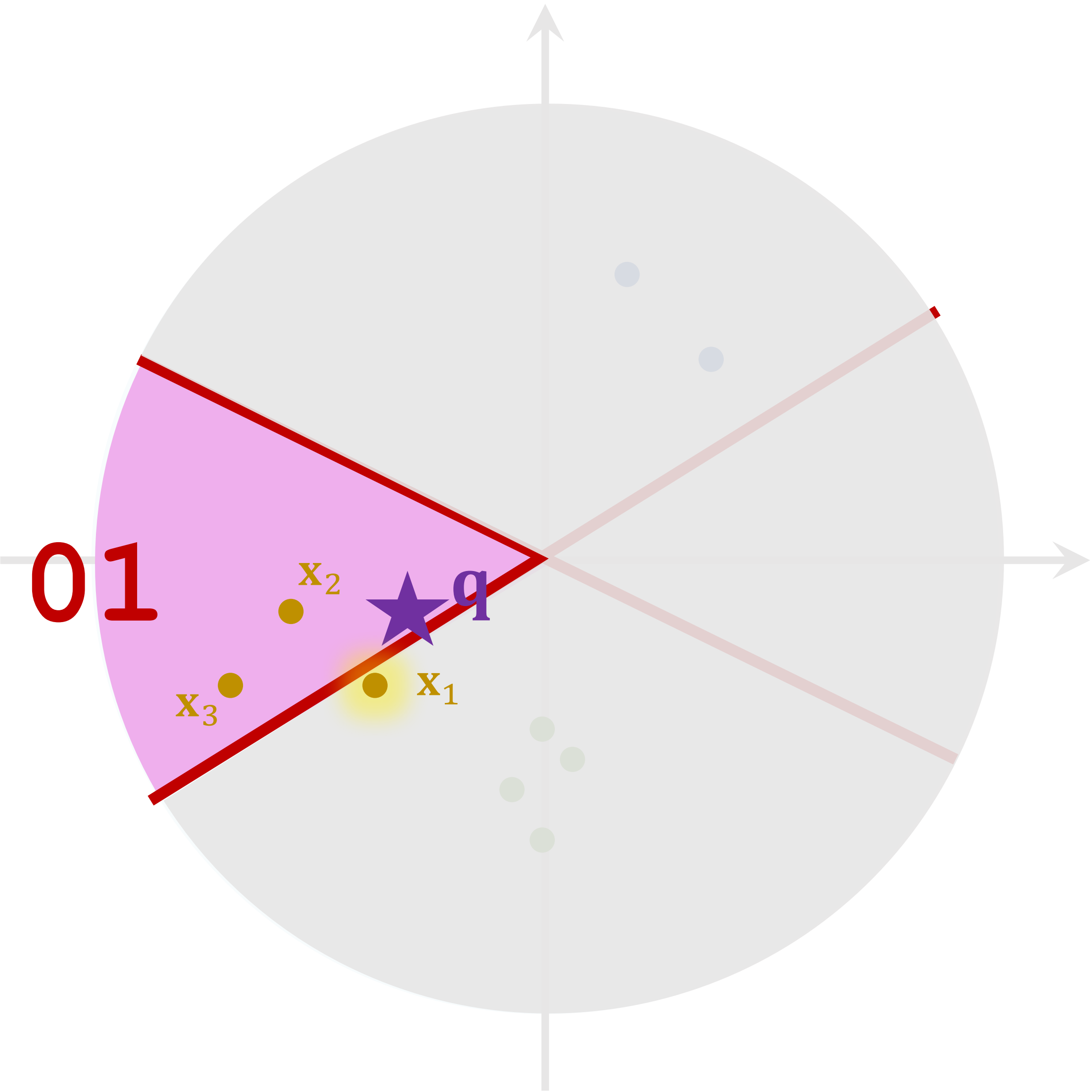
- Given a query $\mathbf{q}$, find its nearest point.
- Hashing: $h_1(\mathbf{q}) = 0, \ h_2(\mathbf{q}) = 1$.
- Thus $\mathbf{q}$ is in the "01" bucket.
- Search the nearest point in the "01" bucket.

Nearest Neighbor Search

Index	Data
00	$\mathbf{x}_1, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7$
01	$\mathbf{x}_2, \mathbf{x}_3$
10	
11	$\mathbf{x}_8, \mathbf{x}_9$

- The points in the "01" bucket can be very efficiently retrieved.

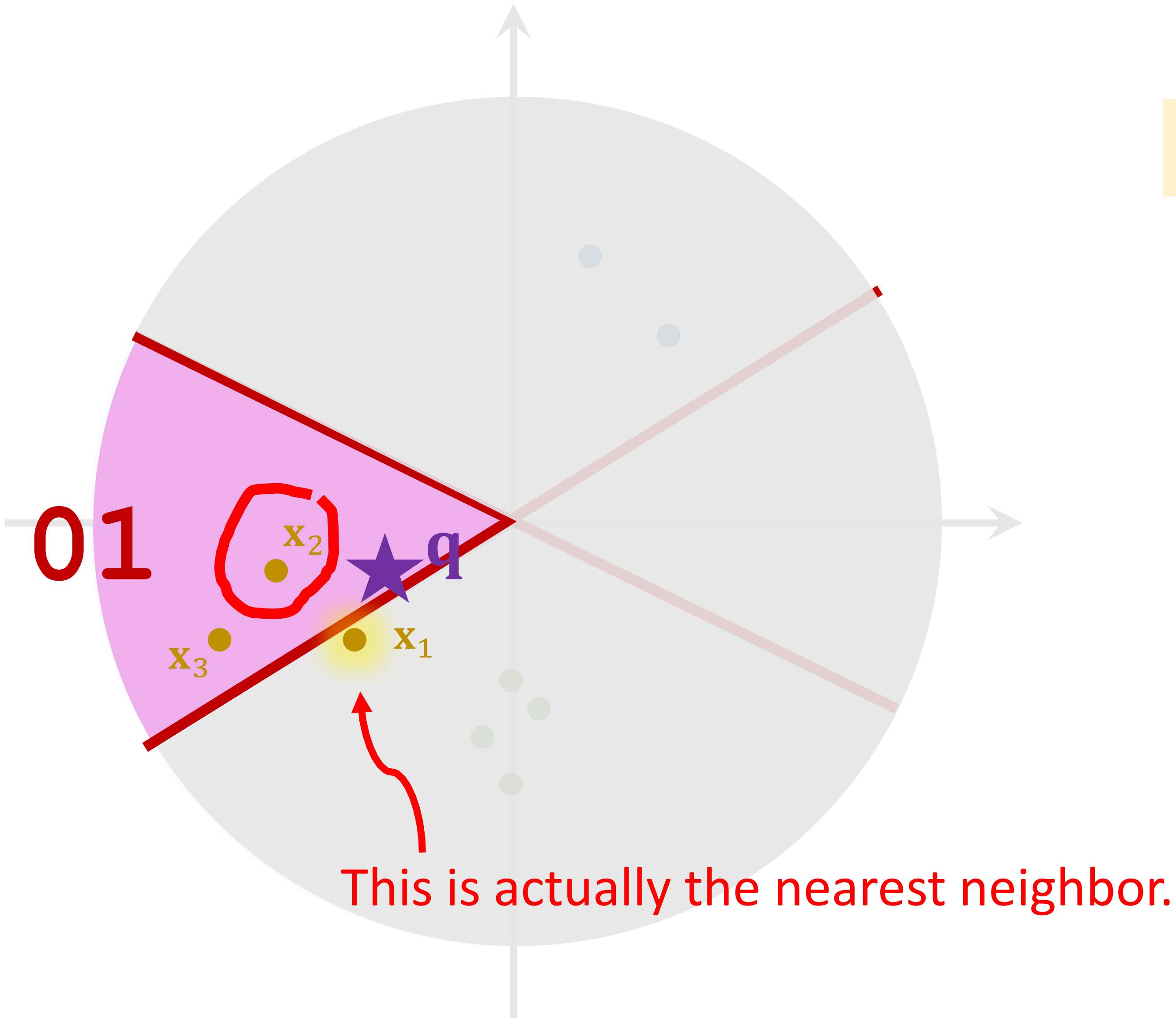
Nearest Neighbor Search



What is wrong with this approach?

- The true nearest neighbor may be in a different bucket.

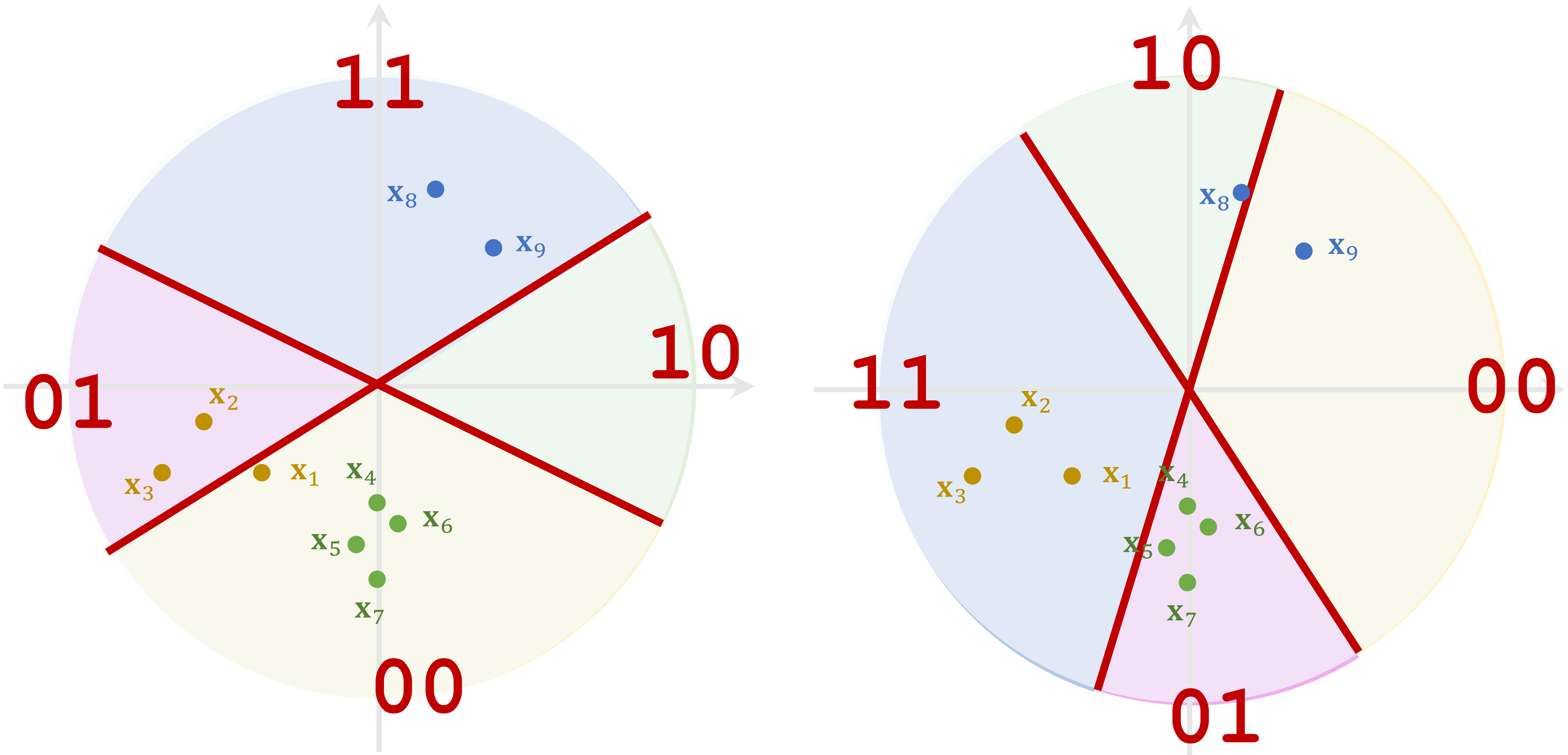
Nearest Neighbor Search



What is wrong with this approach?

- The true nearest neighbor may be in a different bucket.

Using Multiple Tables



Using Multiple Tables

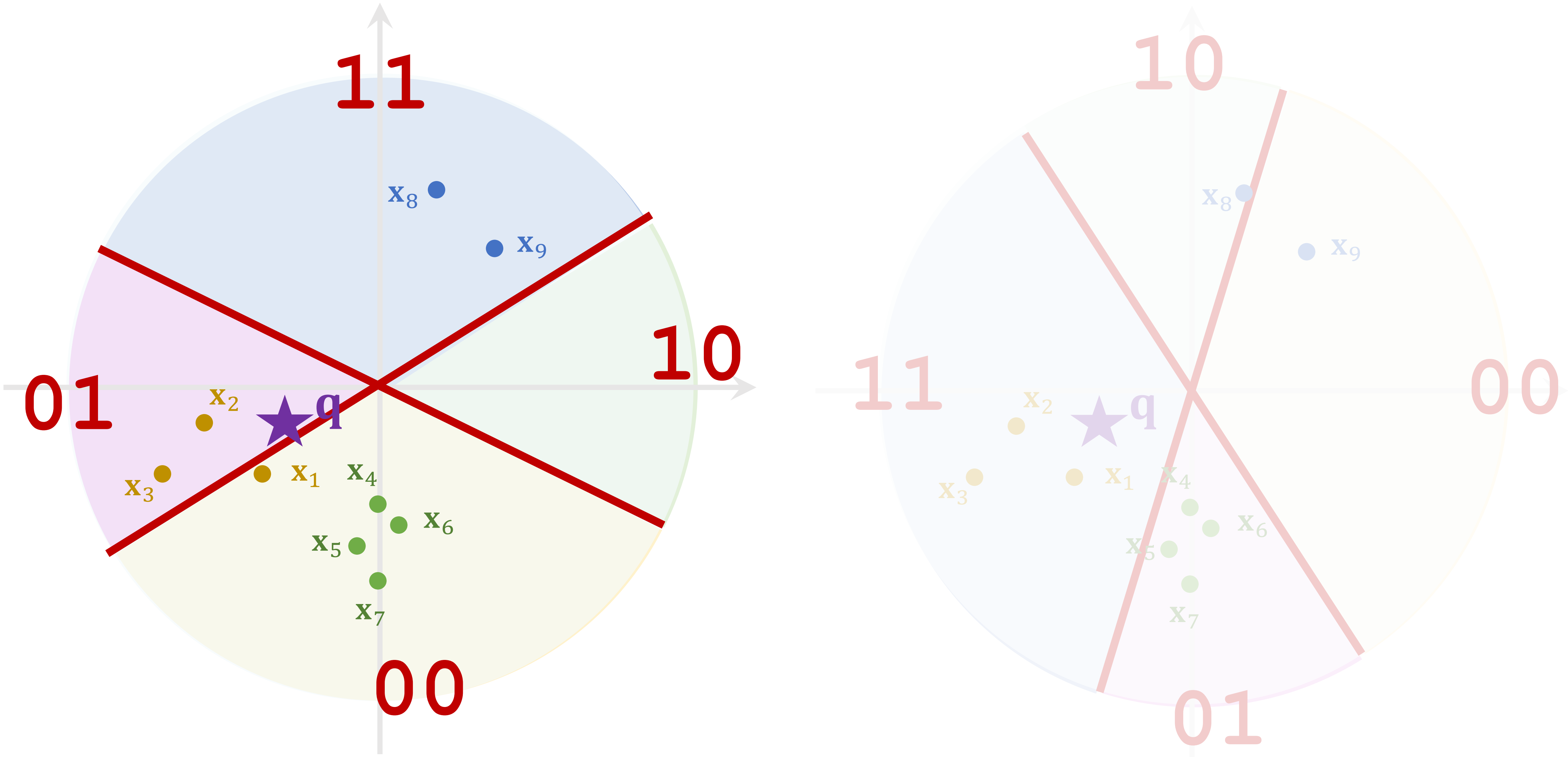
Table 1

Index	Data
00	x_1, x_4, x_5, x_6, x_7
01	x_2, x_3
10	
11	x_8, x_9

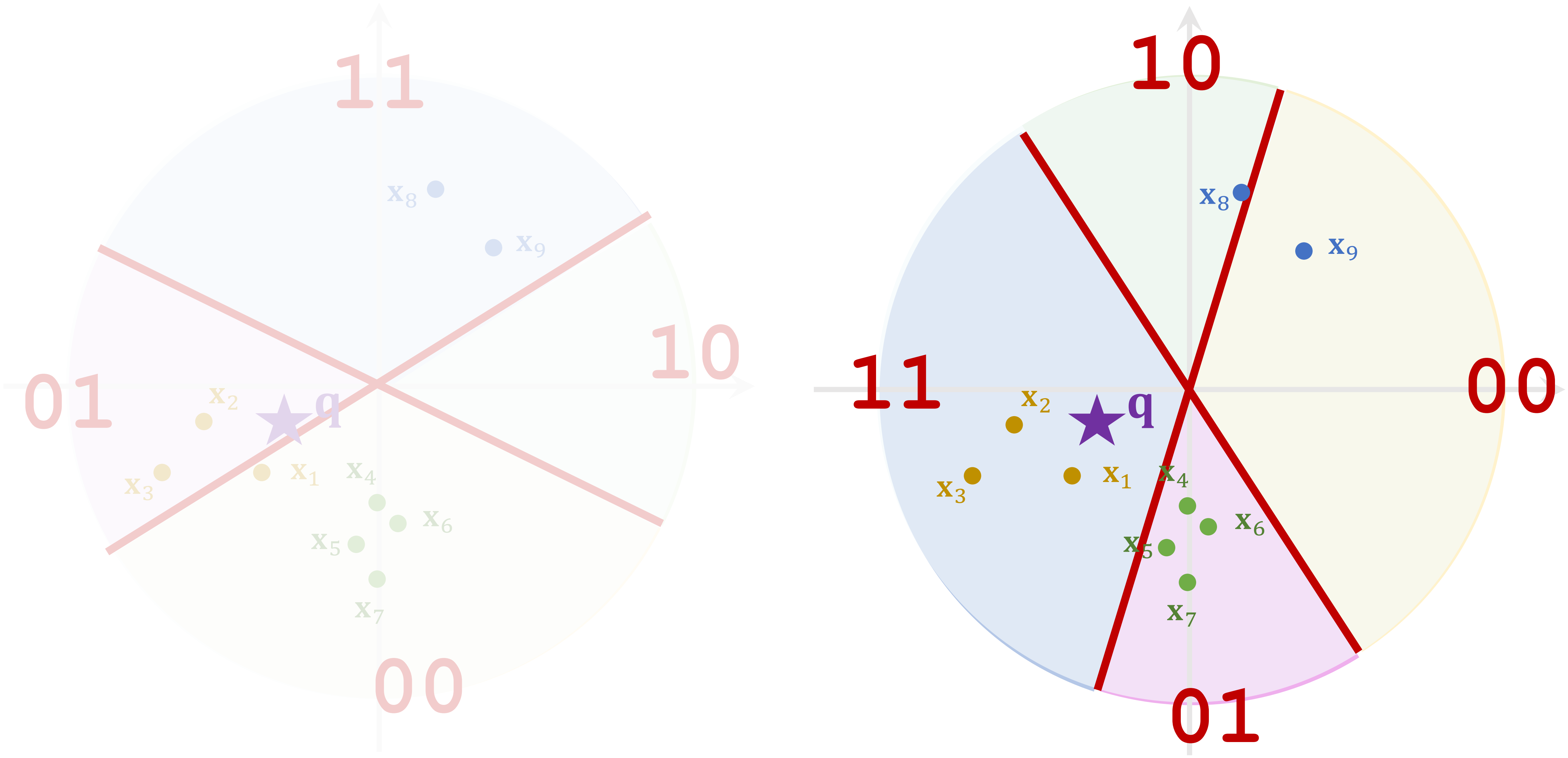
Table 2

Index	Data
00	x_9
01	x_4, x_5, x_6, x_7
10	x_8
11	x_1, x_2, x_3

Using Multiple Tables



Using Multiple Tables



Using Multiple Tables

- From Table 1, retrieve the points in the "**01**" bucket.
- From Table 2, retrieve the points in the "**11**" bucket.

Table 1

Index	Data
00	$\mathbf{x}_1, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7$
01	$\mathbf{x}_2, \mathbf{x}_3$
10	
11	$\mathbf{x}_8, \mathbf{x}_9$

Table 2

Index	Data
00	$\mathbf{x}_9$
01	$\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7$
10	$\mathbf{x}_8$
11	$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$

Time Complexity

LSH via Random Projection

Index	Data
00	
01	
10	
11	

- Build a hash table with 4 buckets.
- Random sample two unit-length vectors:

$$\mathbf{w}_1, \mathbf{w}_2.$$

LSH via Random Projection

Index	Data
00...000	
00...001	
00...010	
00...011	
⋮	
11...111	

$\underbrace{\hspace{5em}}_{k \text{ bits}}$

- Build a hash table with 2^k buckets.
- Random sample k unit-length vectors:

$$\mathbf{w}_1, \cdots, \mathbf{w}_k.$$

LSH via Random Projection

Index	Data
00...000	
00...001	
00...010	
00...011	
⋮	
11...111	

$\underbrace{\hphantom{11...111}}_{k \text{ bits}}$

- Hash function: $h_i(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x}^T \mathbf{w}_i \geq 0; \\ 0, & \text{otherwise.} \end{cases}$
- Let $g(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_k(\mathbf{x})] \in \{0, 1\}^k$.

Use multiple tables

Build $m > 1$ hash tables, each of which has 2^k buckets.

Table 1

Index	Data
00...000	
00...001	
⋮	
11...111	

2^k buckets

k bits

$\cdots$

Table m

Index	Data
00...000	
00...001	
⋮	
11...111	

Nearest Neighbor Search

- Given a query $\mathbf{q}$, find the m buckets where $\mathbf{q}$ belongs to.
- Retrieve points in the m buckets and find the nearest to $\mathbf{q}$.

Table 1

2^k buckets

Index	Data
`00...000`	
`00...001`	
⋮	
`11...111`	

k bits

⋯

Table m

Index	Data
`00...000`	
`00...001`	
⋮	
`11...111`	

Time Complexity of Brute-Force Search

- There are n data points in the database.
- Brute-force algorithm compares $\mathbf{q}$ with all the n points.
- Brute-force search has $O(n)$ per-query time complexity.

Time Complexity of LSH

- Each table has 2^k buckets.
- On average, a bucket has $\frac{n}{2^k}$ points.

- There are m tables.
- Thus $\frac{mn}{2^k}$ points are retrieved.

Time complexity (on average): $O\left(\frac{mn}{2^k}\right)$ per query.

Time Complexity of LSH

Time complexity (on average): $O\left(\frac{mn}{2^k}\right)$ per query.

- If $m < 2^k$, then LSH is faster than the brute-force search.
- Set $k = \Theta(\log n)$ and $m = \sqrt{n}$. (So that the performance is theoretically guaranteed.)
- Time complexity (on average): $O(\sqrt{n})$ per query.

Space Complexity

Space Complexity

Index	Data
`00...000`	
`00...001`	
`00...010`	
`00...011`	
$\vdots$	
`11...111`	

- There are n vectors: $\mathbf{x}_1, \cdots, \mathbf{x}_n$.
- Storing them (or simply their indices) in the table costs $O(n)$ space.

Space Complexity

Index	Data
00...000	
00...001	
00...010	
00...011	
⋮	
11...111	

2^k rows

- There are n vectors: $\mathbf{x}_1, \cdots, \mathbf{x}_n$.
- Storing them (or simply their indices) in the table costs $O(n)$ space.
- The table has 2^k rows.
- Thus $O(2^k)$ space.

Space Complexity

Index	Data
`00...000`	
`00...001`	
`00...010`	
`00...011`	
⋮	
`11...111`	

k bits

- Each table costs $O(n+2^k)$ space.
- There are m hash tables.
- Thus, the overall space complexity is

$$O\left(m(n+2^k)\right).$$

Space Complexity

Index	Data
`00...000`	
`00...001`	
`00...010`	
`00...011`	
$\vdots$	
`11...111`	

$\underbrace{\hspace{6em}}_{k \text{ bits}}$

- Set $k = \Theta(\log n)$.
- Set $m = \sqrt{n}$.
- Space: $O\left(m\left(2^k + n\right)\right) = O(n^{1.5})$.

Comparisons

- There are n data in the database.
- Build m hash tables; each table has 2^k buckets.
- Set $k = \Theta(\log n)$ and $m = \sqrt{n}$.

Comparisons

- There are n data in the database.
- Build m hash tables; each table has 2^k buckets.
- Set $k = \Theta(\log n)$ and $m = \sqrt{n}$.

Brute-Force Search	Using LSH
• Time: $O(n)$ per query. • Space: $O(n)$.	• Time: $O(\sqrt{n})$ per query. • Space: $O(n^{1.5})$.
	Less time, more space.

Thank You!