

Similarity Search - PQ

Lin Guosheng
School of Computer Science and Engineering
Nanyang Technological University

Outline

- Nearest Neighbour Search (NNS) Problems
- Local Sensitive Hashing (LSH) – Random Projection
- **Product Quantization (PQ)**
- Inverted File Index (**non-examinable!!**)

Product Quantization (PQ)

- Product Quantization (PQ)
 - A clustering based method
 - Compress the data
 - Speed up searching by using precomputed distances

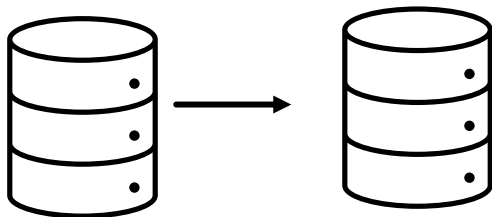
Most materials are from

<https://www.pinecone.io/learn/product-quantization/>

Paper: Product quantization for nearest neighbor search, 2011

Overview of PQ for similarity search

Step1: convert the database
into compressed vectors (offline step)

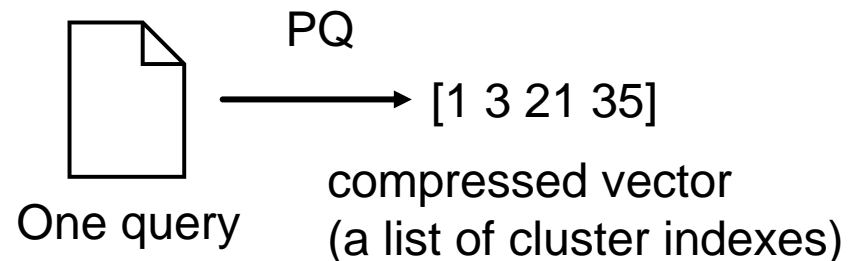


Compressed vectors

When using symmetric distance:
Precompute the
centroid distances tables
for each subspace

Step 2: process query requests
Use efficient approximate distance
(symmetric or asymmetric)
calculation to get top-k retrieved results

When using symmetric distance:
For each query, we need to
convert the query to a compressed vector



The process of PQ for compression

Pre-processing:

1. Define the number of subspaces (subvectors).
2. For each subspace, generate k cluster centroids using the samples in the database.

Generating compressed vectors using PQ

Input: the input feature vector (high-dimensional)

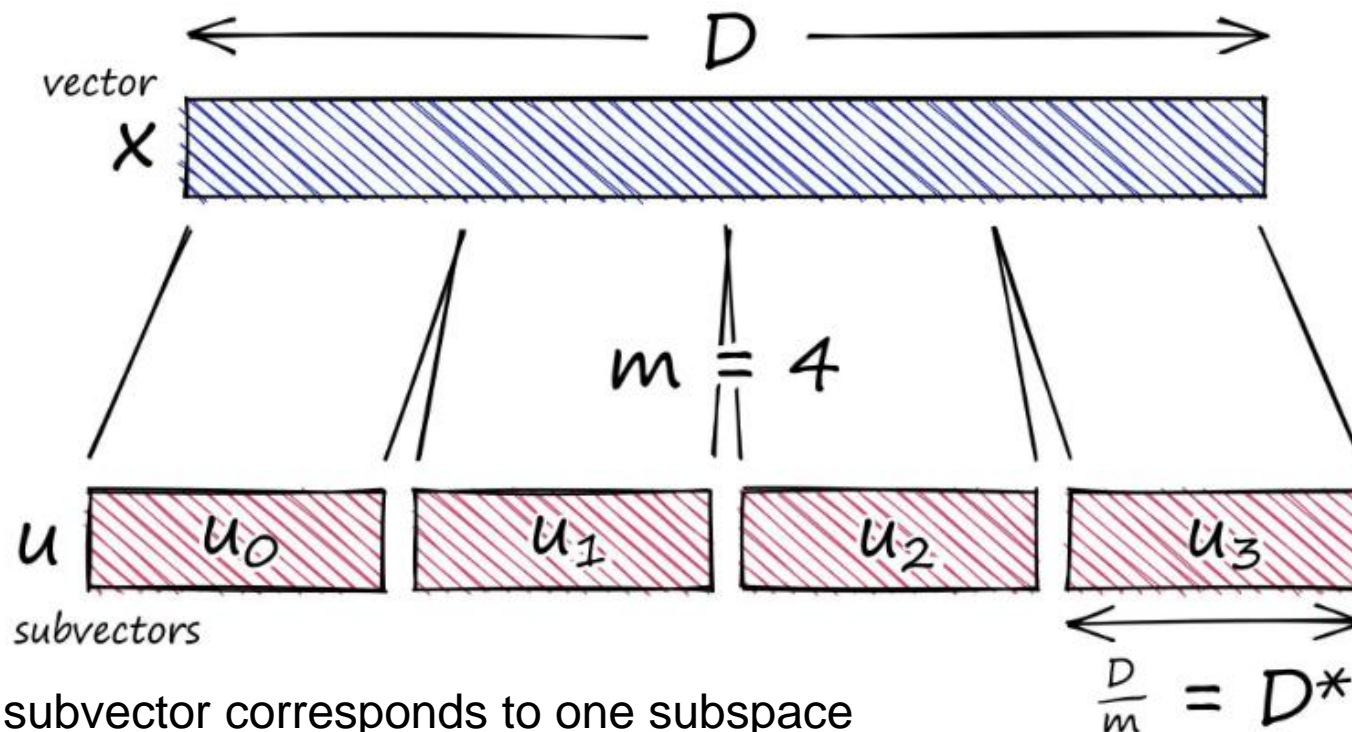
Steps:

1. Equally split the input into M sub-vectors.
Each sub-vector corresponds to one subspace.
2. Assigning each of these sub-vectors to its nearest cluster centroid (also called reproduction/reconstruction values)
3. Use the assigned cluster IDs (indexes) of all sub-vectors to construct the compressed vector (or called quantized vector).

Example

We define $M=4$ subspaces.

-> We generate 4 sub-vectors for each input.
(equally split the input vector into M sub-vectors)



One subvector corresponds to one subspace

E.g., if the input vector dimension $D=100$,
the length of each sub-vector is $100/4 = 25$

Sub-vector generation example

Compressed vector

Define 4 subspaces

-> 4 sub-vectors

Input vector

Equally
split
→

Find the nearest centroid
in subspace 1

Assigned
cluster ID (S1)

Find the nearest centroid
in subspace 2

Assigned
cluster ID (S2)

Find the nearest centroid
in subspace 3

Assigned
cluster ID (S3)

Find the nearest centroid
in subspace 4

Assigned
cluster ID (S4)

Compressed
vector:
[S1, S2, S3, S4]

Example

- Product Quantization example

Define 2 Subspaces

Each subspace has 4 centroids.

Input vectors:

A: [1.82, 5.08, 2.21, 4.21]

B: [4.96, 4.46, 4.1, 1.3]

Example

Subspace 1, We have 4 centroids: C1 to C4.

A1, B1: the sub-vectors for datapoint A and B, respectively

$$A1 = (1.82, 5.08)$$

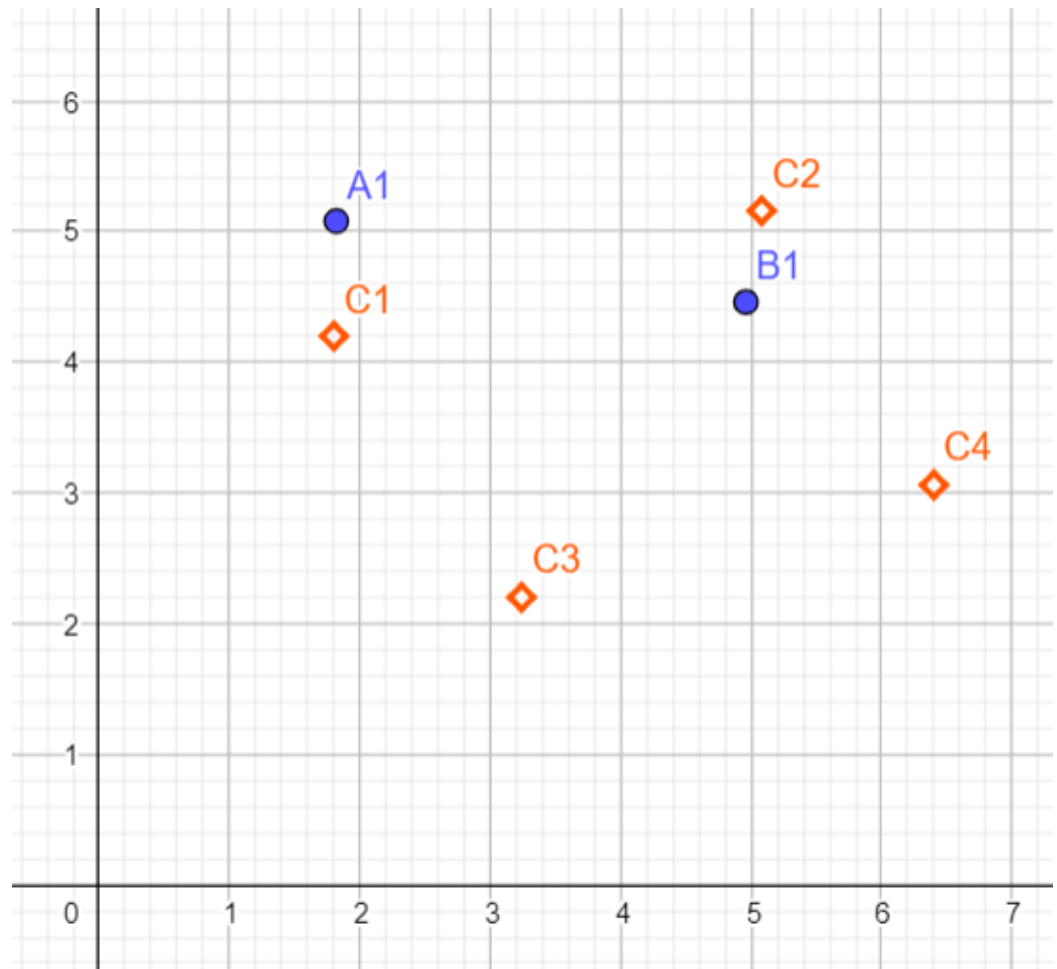
$$B1 = (4.96, 4.46)$$

$$C1 = (1.8, 4.2)$$

$$C2 = (5.08, 5.16)$$

$$C3 = (3.24, 2.2)$$

$$C4 = (6.4, 3.06)$$



Example

Subspace 2, We have 4 centroids: C1 to C4.

A2, B2: the sub-vectors for datapoint A and B, respectively

$$A2 = (2.01, 4.21)$$

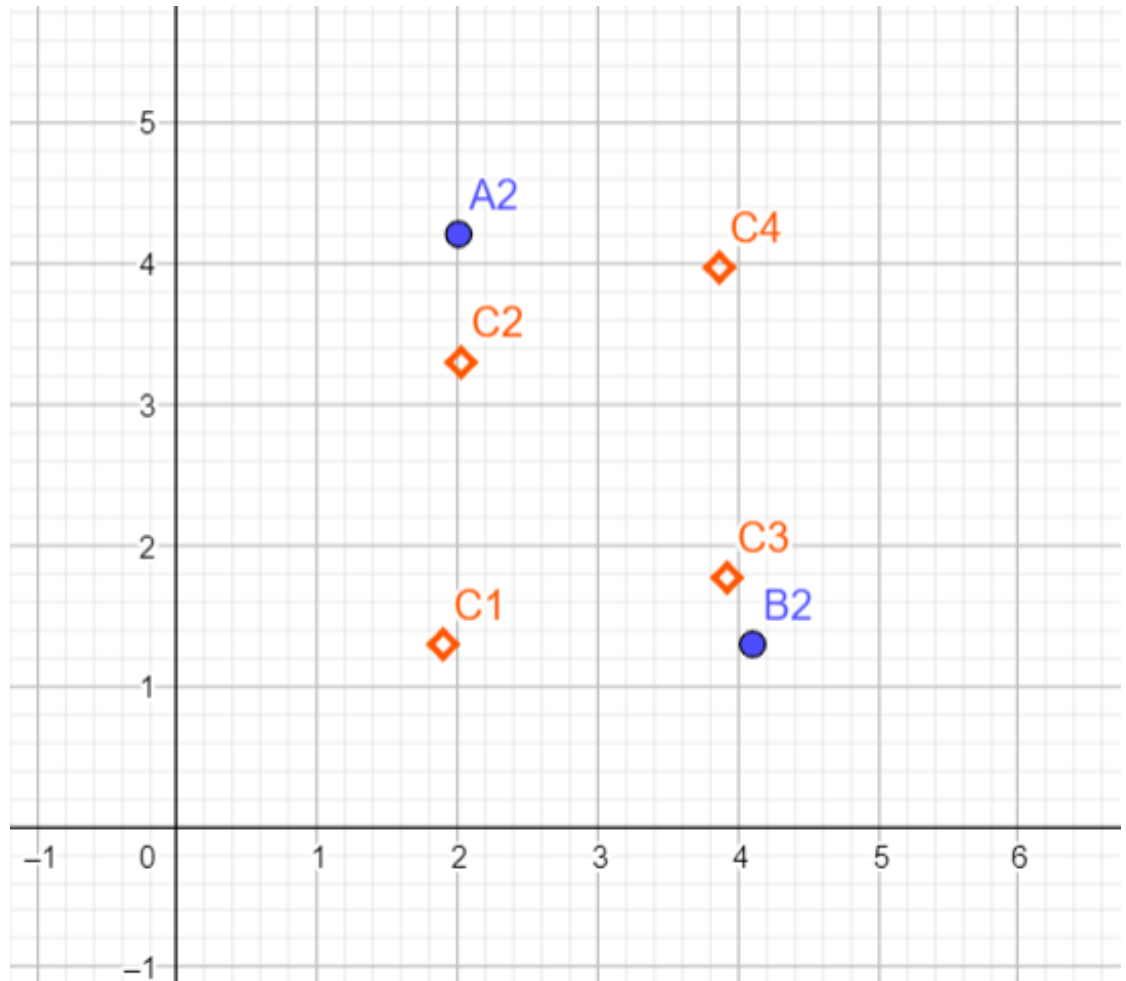
$$B2 = (4.1, 1.3)$$

$$C1 = (1.9, 1.3)$$

$$C2 = (2.02, 3.3)$$

$$C3 = (3.92, 1.77)$$

$$C4 = (3.87, 3.98)$$



Example

- Q: generate the compressed vector for datapoint A and B:

Answer:

In subspace 1, A is assigned to the centroid C1
(1st dimension of the compressed vector)

In subspace 2, A is assigned to the centroid C2
(2nd dimension of the compressed vector)

-> A: [1, 2]

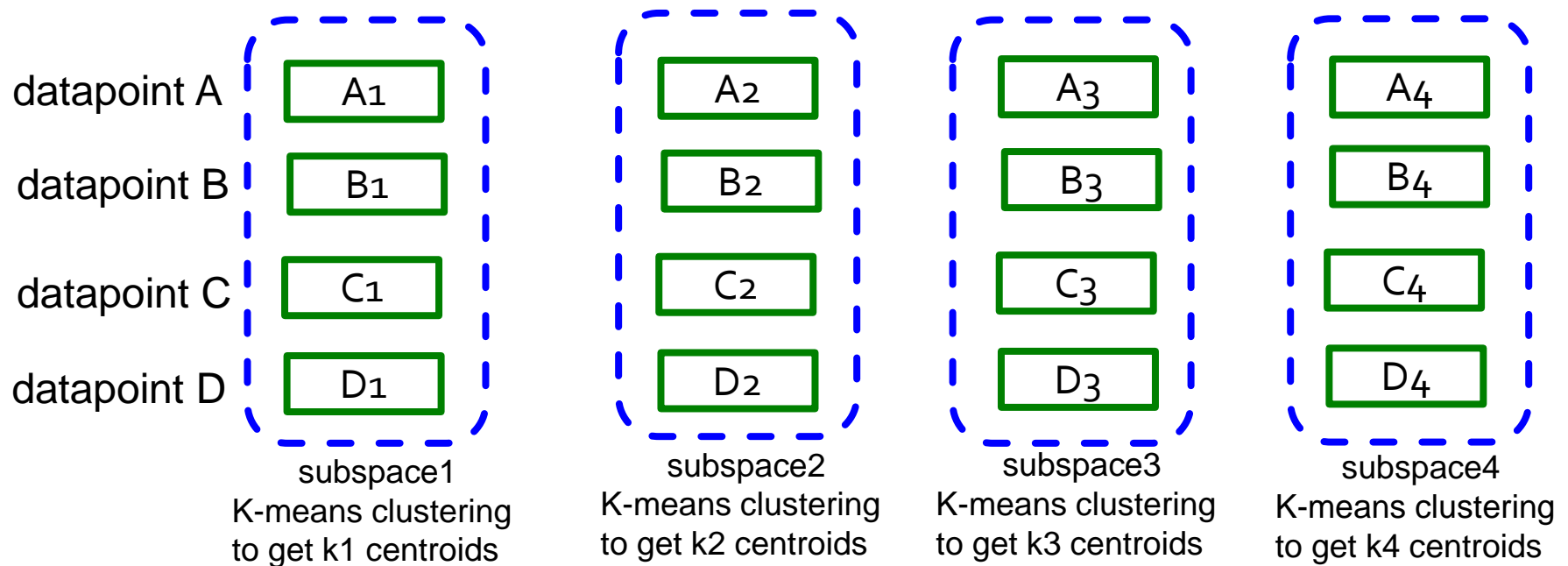
In subspace 1, B is assigned to the centroid C2
In subspace 2, B is assigned to the centroid C3

- -> B: [2, 3]

■ How to generate cluster centroids?

- Generate sub-vectors for all data examples in the database
 - Define M subspaces -> Generate M sub-vectors for one input.
- Perform K-means in each subspace to generate k centroids.
 - For example, we can generate k=256 centroids for each sub space

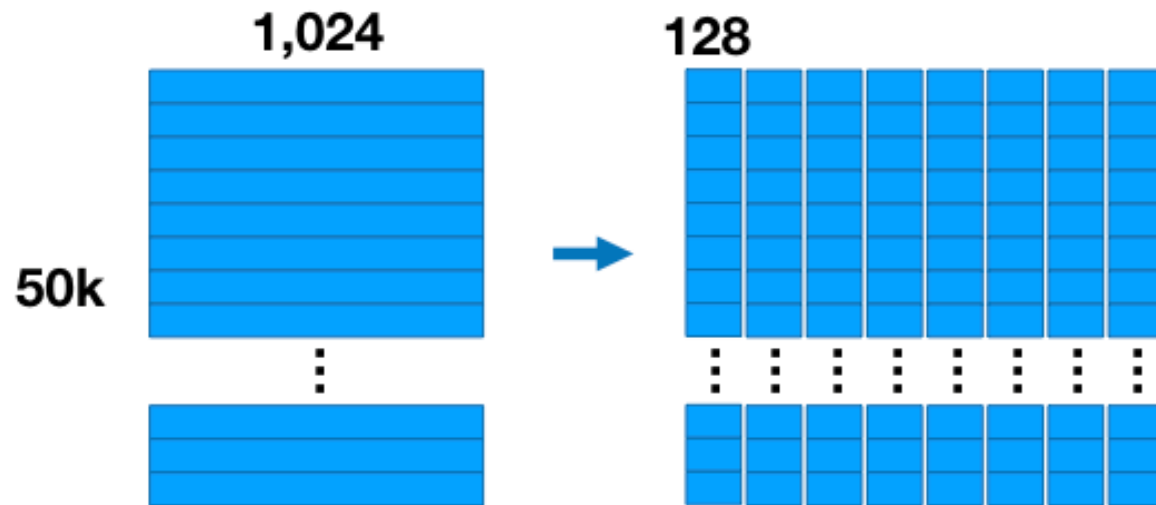
Use datapoints in the database to generate centroids for each subspace



Example: Subspace clustering

Database: 50K samples,
each has 1024 dimensions:

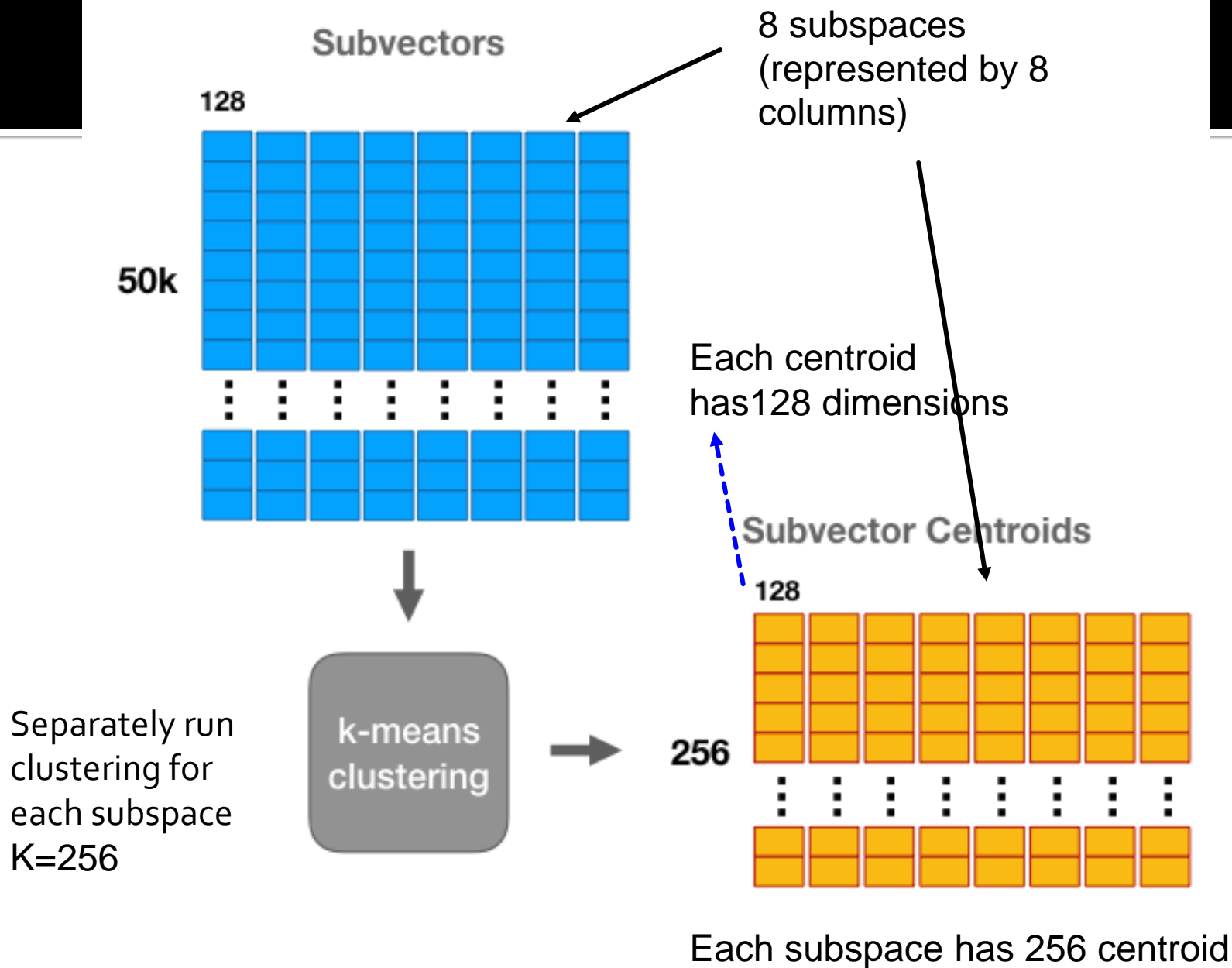
8 sub-vectors for each sample:
(each sub-vector's dimension: 128)



Each row indicates one data sample.

Define 8 subspaces. Each vector is spitted into 8 sub-vectors, each of length 128 (8 sub vectors x 128 dimensions = 1,024 dimensions).

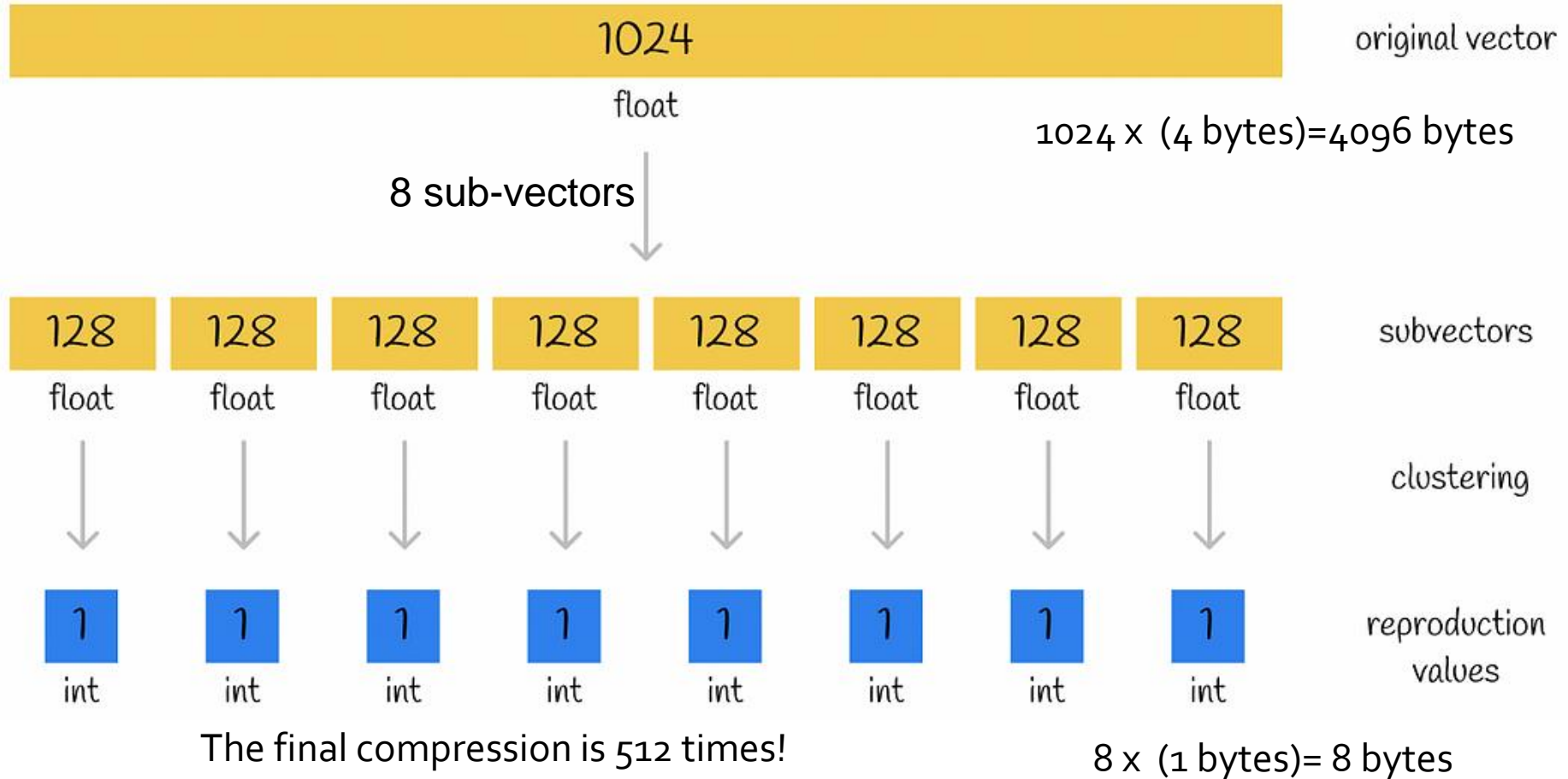
<https://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/#nearest-neighbor-search>



Compression analysis

1 float value requires 4 bytes ($4 \times 8 = 32$ bits);

1 integer value requires 1 byte (here we use 8-bit integer, value range: 0-255)



One centroid ID is an integer value. In this example, we only have 256 centroids, so we can use one 8-bit integer ($2^8 = 256$) to store one cluster ID.

Product Quantization

- Codebook based compression
 - the codebook is the centroids
 - Separate codebook (centroids) for each subspace
- Compress high-dimensional vector to a tiny vector of IDs that only requires very little memory/storage space.

Product Quantization

- Product Quantization (PQ)
 - A data dependent method
 - Need to use the data in the database to determine the parameters (cluster centres) of the algorithm.
 - LSH is a data independent method
 - The hash functions are randomly generated, not using the data in the database.

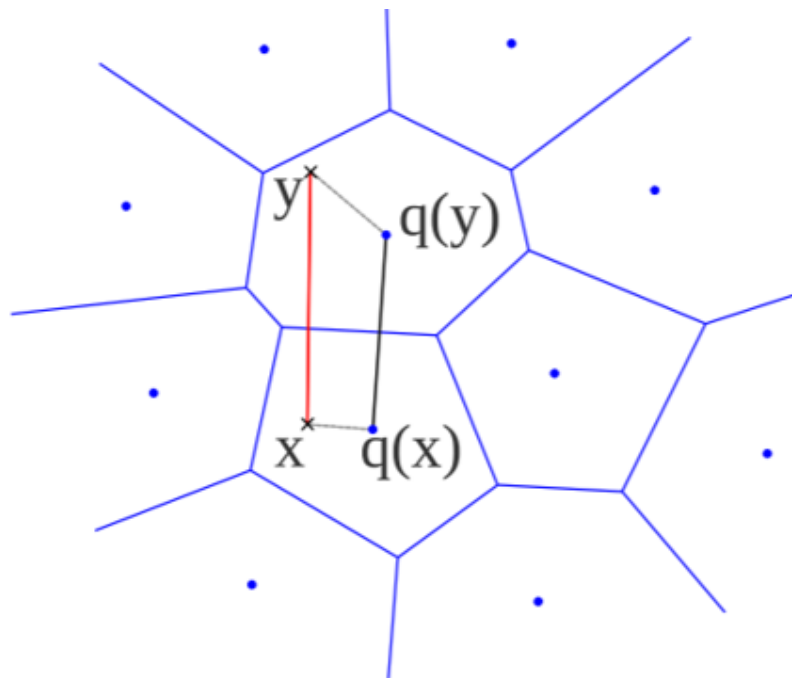
PQ for similarity search

- PQ for Similarity Search
 - Use liner search (exhaustive search)
 - Use approximate distance to speed up distance calculation
 - Calculating approximate distance
 - Symmetric distance
 - Use pre-computed distance look-up table to speed up the calculation.
 - Asymmetric distance

Symmetric distance

- Approximate distance: Symmetric distance

$$\hat{d}(x, y) = d(q(x), q(y)) = \sqrt{\sum_j d(q_j(x), q_j(y))^2},$$



symmetric case

x, y are two input vectors

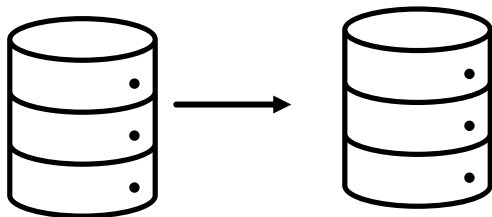
$q(x)$ and $q(y)$ are the
assigned cluster centroids

The distance of (x, y) is approximated
by the distance of cluster centroids

Recap

Overview of PQ for similarity search

Step1: convert the database
into compressed vectors (offline step)

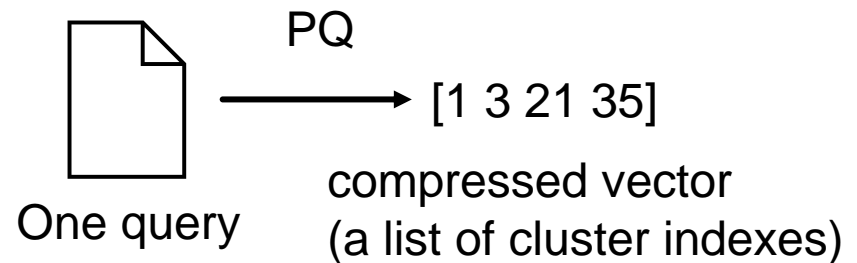


Compressed vectors

When using symmetric distance:
Precompute the
centroid distances tables
for each subspace

Step 2: process query requests
Use efficient approximate distance
(symmetric or asymmetric)
calculation to get top-k retrieved results

When using symmetric distance:
For each query, we need to
convert the query to a compressed vector



Symmetric distance example

Subspace 1, We have 4 centroids: C1 to C4.

A1, B1: the sub-vectors for datapoint A and B, respectively

$$A1 = (1.82, 5.08)$$

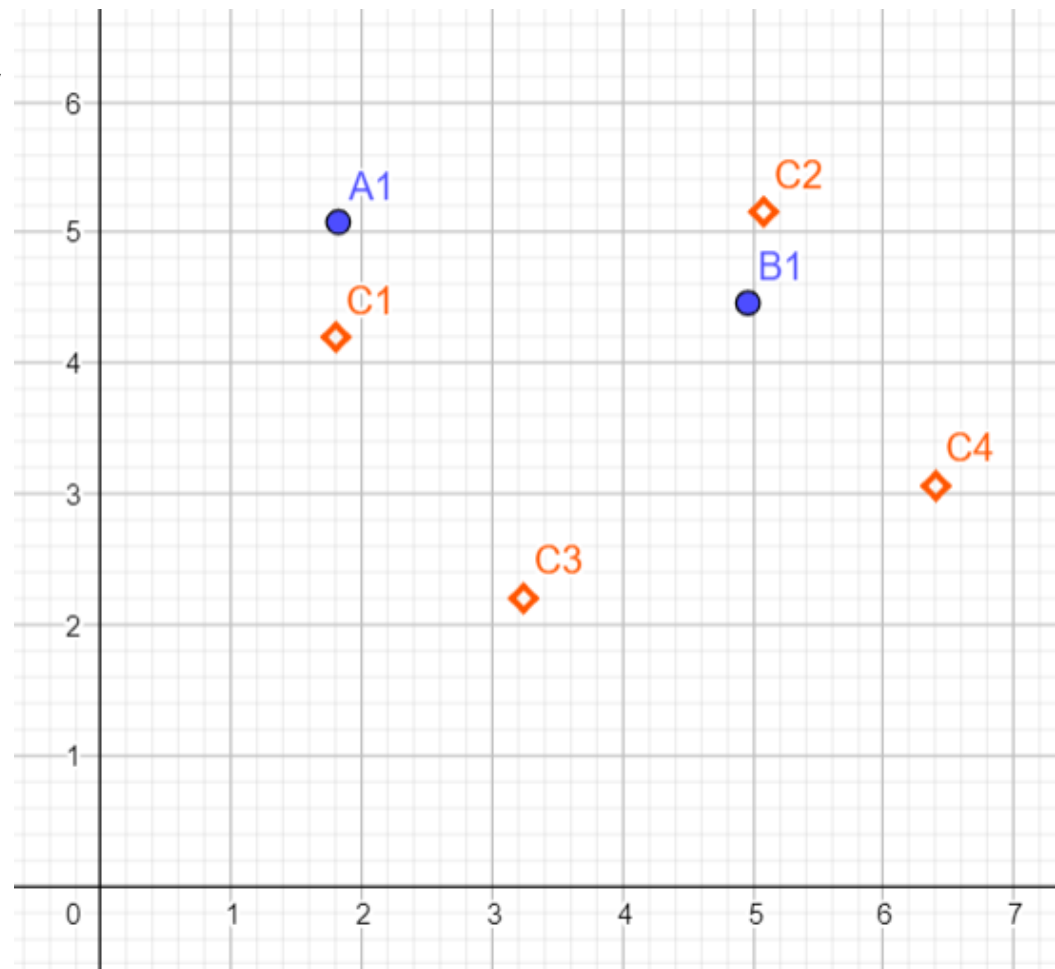
$$B1 = (4.96, 4.46)$$

$$C1 = (1.8, 4.2)$$

$$C2 = (5.08, 5.16)$$

$$C3 = (3.24, 2.2)$$

$$C4 = (6.4, 3.06)$$



Symmetric distance example

Subspace 2, We have 4 centroids: C1 to C4.

A2, B2: the sub-vectors for datapoint A and B, respectively

$$A2 = (2.01, 4.21)$$

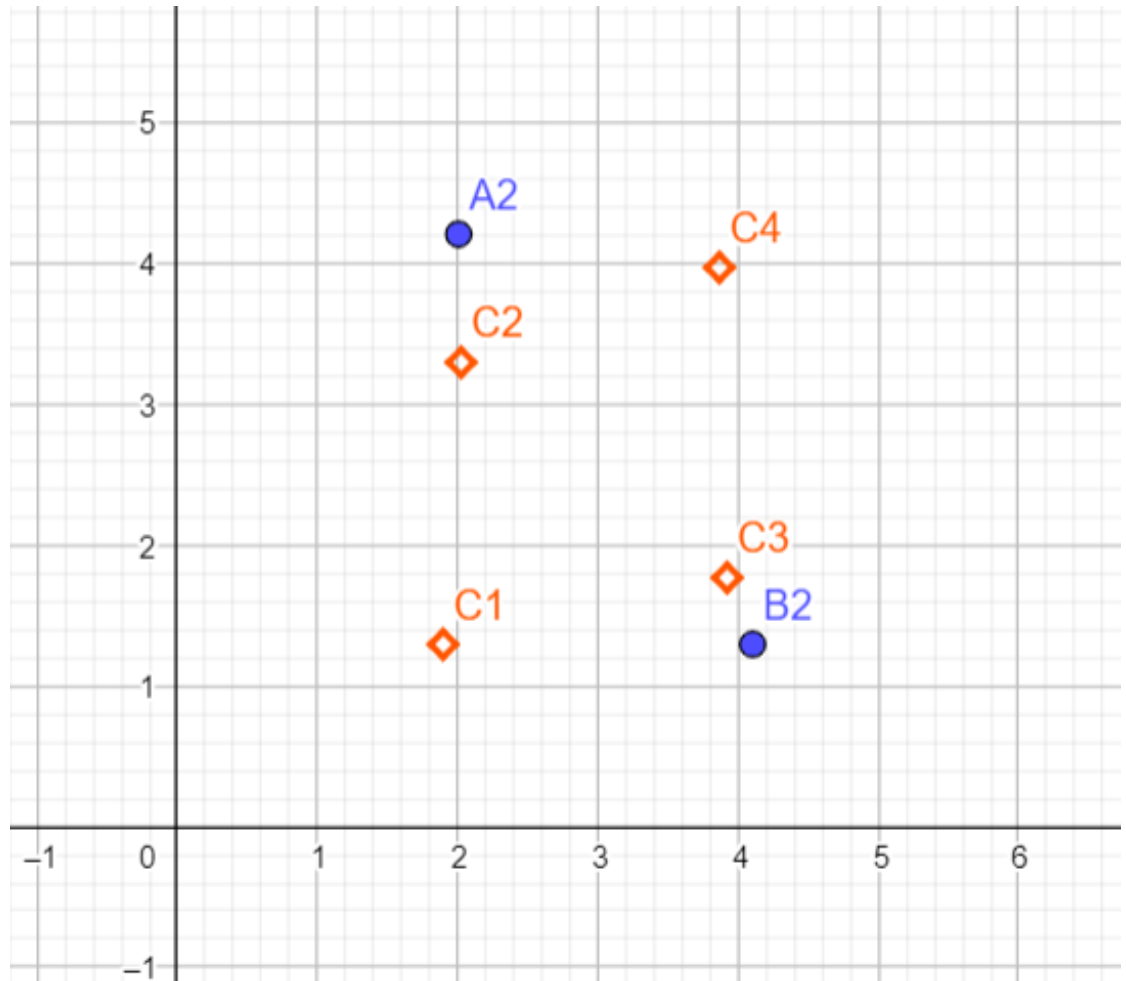
$$B2 = (4.1, 1.3)$$

$$C1 = (1.9, 1.3)$$

$$C2 = (2.02, 3.3)$$

$$C3 = (3.92, 1.77)$$

$$C4 = (3.87, 3.98)$$



Symmetric distance example

Define 2 subspaces. The centroids and datapoints are given in the above slides.

Question:

1. Construct centroid distance look up tables and
2. compute the approximate Squared L2 distance (symmetric case) between A and B.

Symmetric distance example

■ Solution to Q1: Subspace 1

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	11.68	6.07	22.46
Centroid 2	11.68	0	12.15	6.15
Centroid 3	6.07	12.15	0	10.73
Centroid 4	22.46	6.15	10.73	0

Table 1: squared L2 distance table for Subspace 1 (D1)

Distance calculation example:

$C1=[1.8, 4.2]$, $C2=[5.08, 5.16]$,

$$D1(C1, C2) = (1.8-5.08)^2 + (4.2-5.16)^2 = 10.76 + 0.92 = 11.68$$

Subspace 1, We have 4 centroids: C1 to C4.

A1, B1: the sub-vectors for datapoint A and B, respectively

$$A1 = (1.82, 5.08)$$

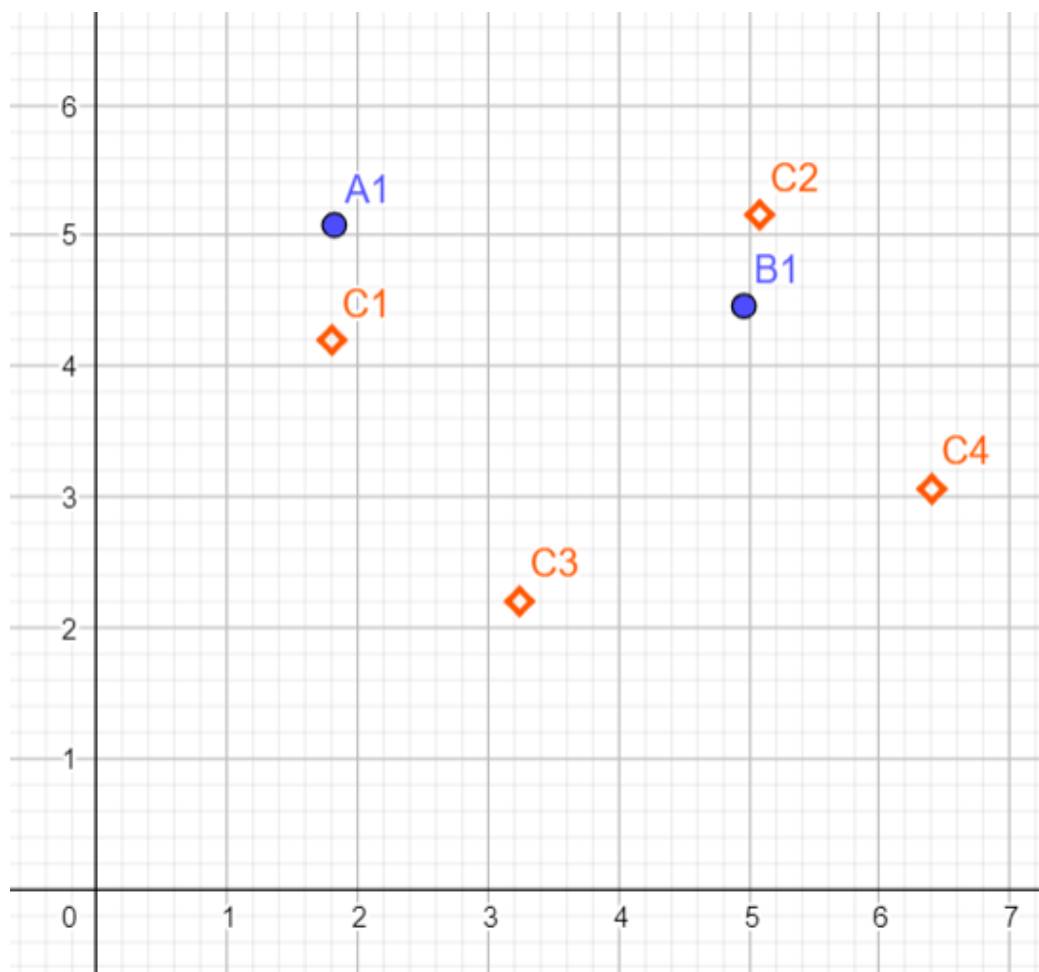
$$B1 = (4.96, 4.46)$$

$$C1 = (1.8, 4.2)$$

$$C2 = (5.08, 5.16)$$

$$C3 = (3.24, 2.2)$$

$$C4 = (6.4, 3.06)$$



Subspace 2

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	4.01	4.30	11.06
Centroid 2	4.01	0	5.95	3.88
Centroid 3	4.3	5.95	0	4.89
Centroid 4	11.06	3.88	4.89	0

Table 2: squared L2 distance table for Subspace 2 (D2)

Subspace 2, We have 4 centroids: C1 to C4.

A2, B2: the sub-vectors for
datapoint A and B, respectively

$$A2 = (2.01, 4.21)$$

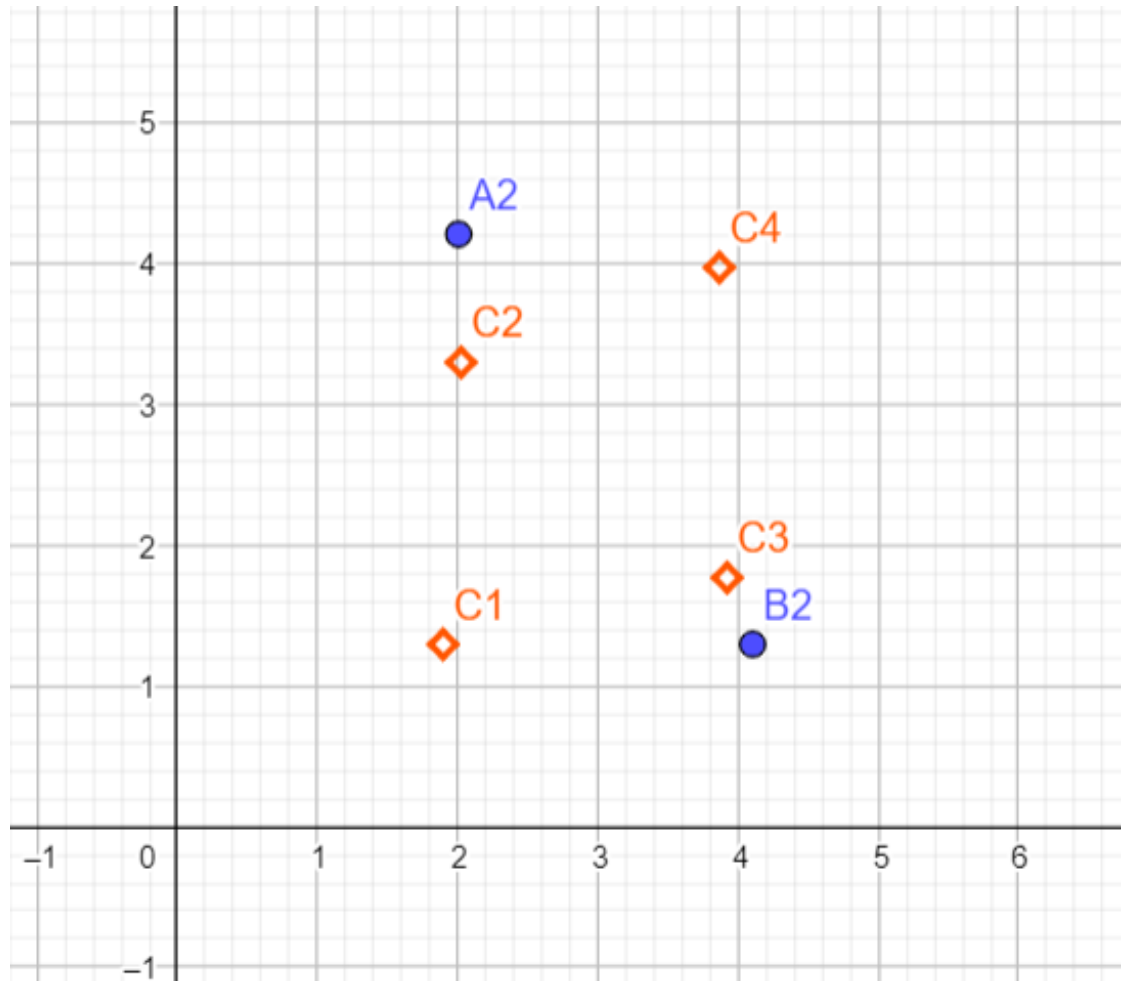
$$B2 = (4.1, 1.3)$$

$$C1 = (1.9, 1.3)$$

$$C2 = (2.02, 3.3)$$

$$C3 = (3.92, 1.77)$$

$$C4 = (3.87, 3.98)$$



Question:

2. compute the approximate Squared L2 distance (symmetric case) between A and B.

■ Solution to Q2:

Compressed vectors: A: [1, 2]; B: [2, 3]

Look up the distance values in table D1 and D2 using the cluster IDs:

$$D(A_1, B_1) = D1(\text{Centroid 1}, \text{Centroid 2}) = 11.68$$

$$D(A_2, B_2) = D2(\text{Centroid 2}, \text{Centroid 3}) = 5.95$$

$$D(A, B) \approx D(A_1, B_1) + D(A_2, B_2) = 11.68 + 5.95 = 17.63$$

Subspace 1

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	11.68	6.07	22.46
Centroid 2	11.68	0	12.15	6.15
Centroid 3	6.07	12.15	0	10.73
Centroid 4	22.46	6.15	10.73	0

Subspace 2

	Centroid 1	Centroid 2	Centroid 3	Centroid 4
Centroid 1	0	4.01	4.30	11.06
Centroid 2	4.01	0	5.95	3.88
Centroid 3	4.3	5.95	0	4.89
Centroid 4	11.06	3.88	4.89	0

Compressed vectors: A: [1, 2]; B: [2, 3]

$$D(A_1, B_1) = D_1(\text{Centroid 1}, \text{Centroid 2}) = 11.68$$

$$D(A_2, B_2) = D_2(\text{Centroid 2}, \text{Centroid 3}) = 5.95$$

$$D(A, B) \approx D(A_1, B_1) + D(A_2, B_2) = 11.68 + 5.95 = 17.63$$

Symmetric distance

- Approximate distance calculation
 - we can calculate the distances much more efficiently using just table look-ups.
 - Using centroid pre-computed distances.
Pre-compute and the distances between centroids for each subspace, stored the distances in a matrix or lookup table, use the centroid index pair to retrieve the partial distances for each subspace.
 - Sum-up the partial distances from all subspaces as the final distance output

Asymmetric distance

Asymmetric distance

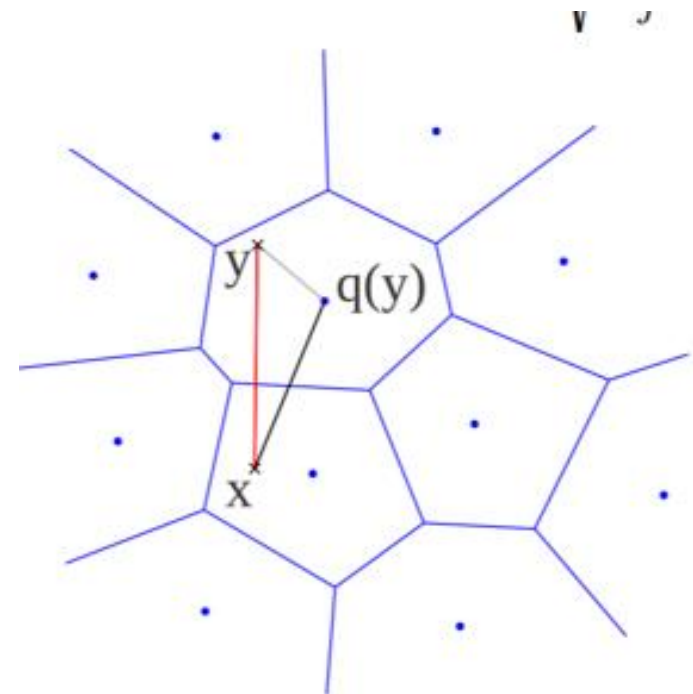
More accurate for distance calculation,
as we only quantize one input.
(compared to symmetric distance)

$$\tilde{d}(x, y) = d(x, q(y))$$

x: a query example,

y: an example in the database.

Use $q(y)$ to replace y
for distance calculation

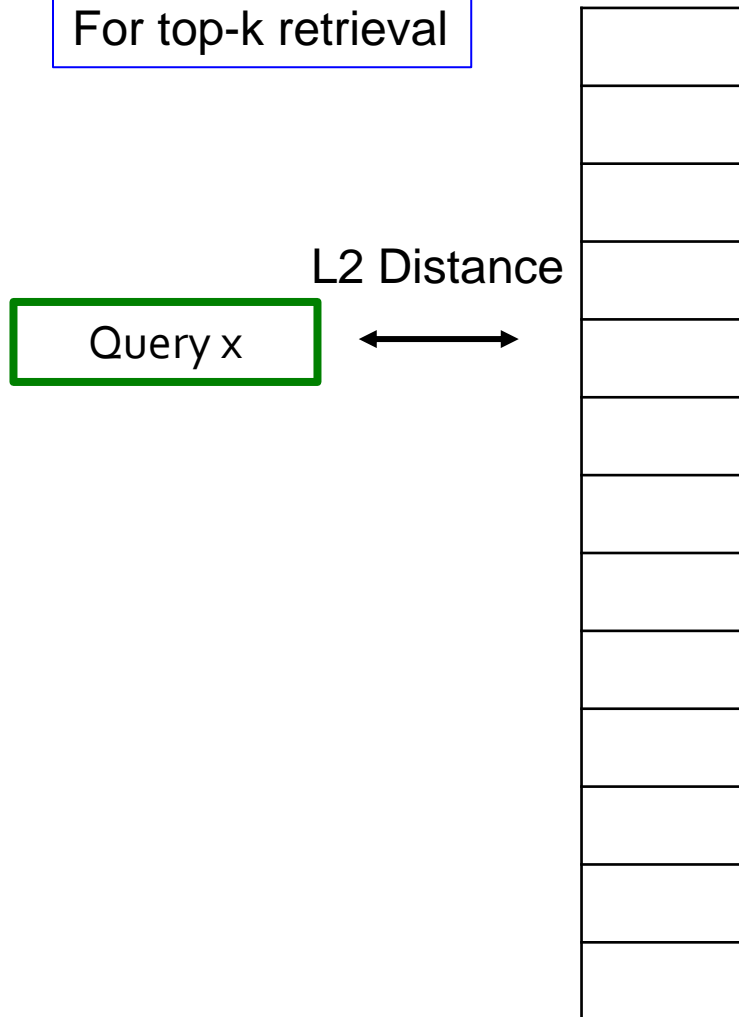


asymmetric case

Example for asymmetric distance

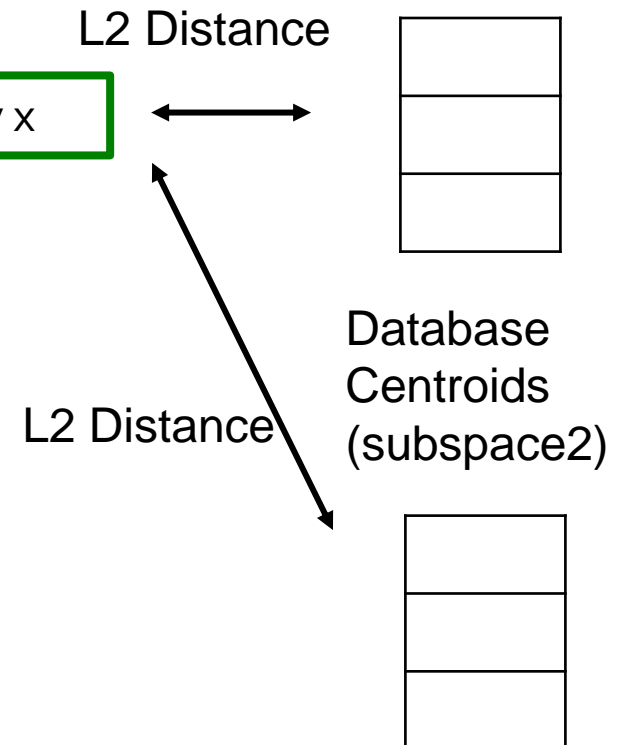
Traditional
linear search
For top-k retrieval

Database
All datapoints



PQ + asymmetric distance
for top-k retrieval

Database
Centroids
(subspace1)



Example (asymmetric distance)

Data points in the database:

A	[4, 6, 14, 4]
B	[6, 2, 8, 2]
D	[6, 8, 10, 2]
E	[4, 4, 6, 4]

Query data point:

Q	[16, 4, 4, 8]
---	---------------

Subspace 1

Centroid 1	[8,6]
Centroid 2	[18,6]

Subspace 2

Centroid 1	[10, 12]
Centroid 2	[10, 4]

Question: Calculate the approximate Squared L2 distance using **Asymmetric distance** between the query Q and all data points in the database (A, B, D, E)

Example (asymmetric distance)

Solution:

Step1: Compress the data points in the database
(In each subspace, identify the nearest centroid for each data point)

Step 2: Calculate Asymmetric distances

Example (asymmetric distance)

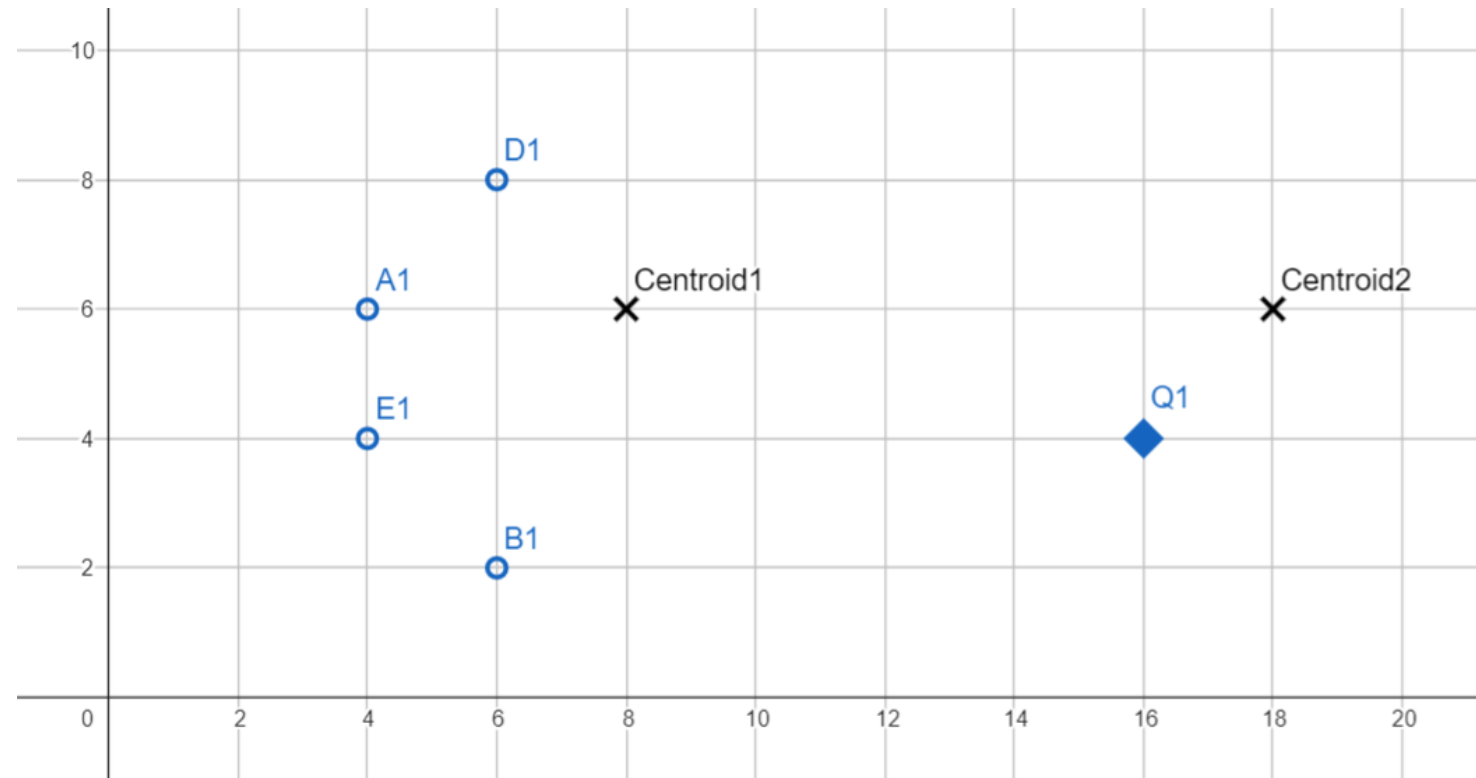
Subspace 1:

(A1, B1, D1, E1, Q1 are sub-vectors in Subspace1)

A1	[4, 6]
B1	[6, 2]
D1	[10, 2]
E1	[4, 4]

Q1	[16, 4]
----	---------

C1	[8, 6]
C2	[18, 6]

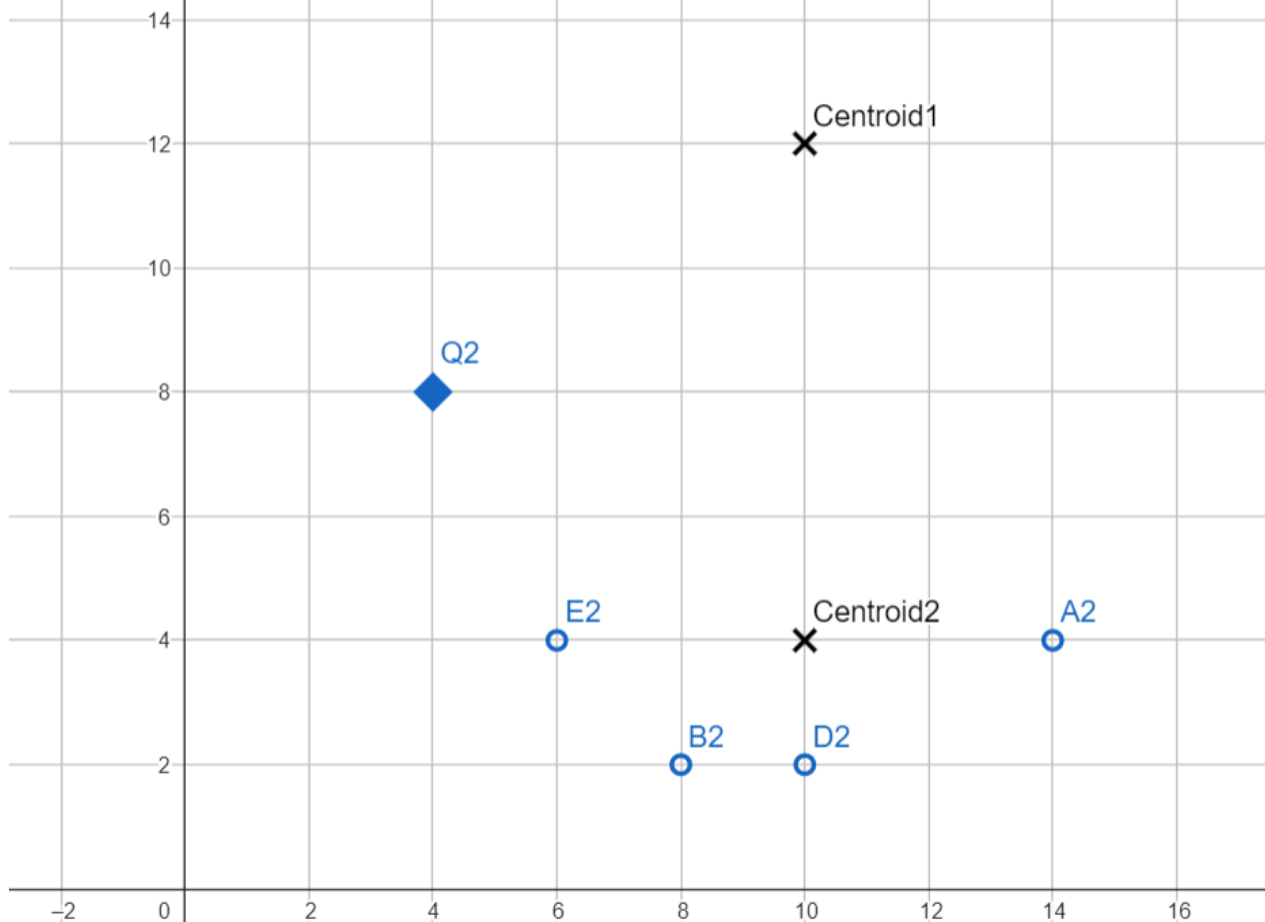


Calculate the squared L2 distances between the data points and each centroid to identify the nearest centroid

Example (asymmetric distance)

Subspace 2:

(A2, B2, D2, E2, Q2 are sub-vectors in Subspace2)



A2	[14, 4]
B2	[8, 2]
D2	[10, 2]
E2	[6, 4]

Q2	[4, 8]
----	--------

C1	[10, 12]
C2	[10, 4]

Calculate the squared L2 distances between the data points and each centroid to identify the nearest centroid

Example (asymmetric distance)

Step1: Compress the data points in the database
(In each subspace, identify the nearest centroid for each data point)

compression results:

A -> [1, 2]

B -> [1, 2]

D -> [1, 2]

E -> [1, 2]

Example (asymmetric distance)

Step2: calculate Asymmetric distances

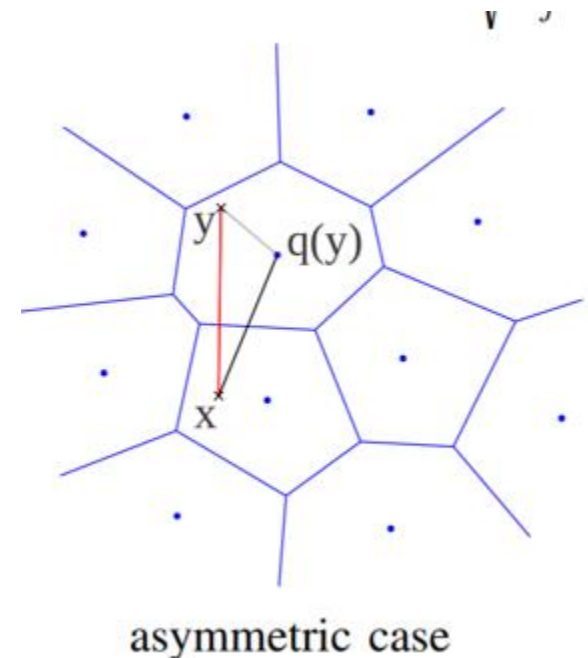
1) Generally, for the query Q, we need to calculate the distances between Q and all the centroids in each subspace.

2) For this question, as all data points are compressed as the same vector [1, 2], we only need to calculate the following distances:

Subspace 1: D (Q1, Centroid 1)

Subspace 2: D (Q2, Centroid 2)

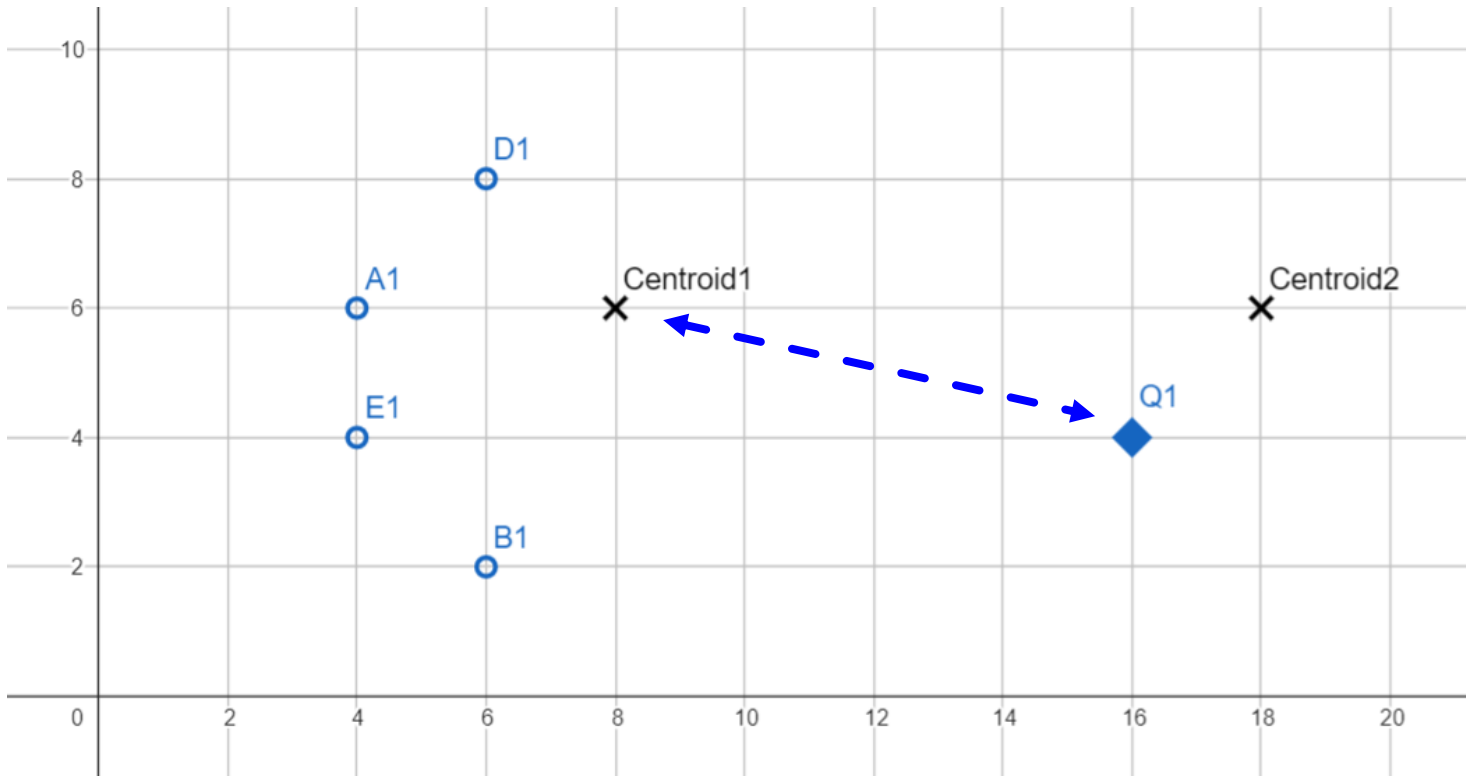
$$\tilde{d}(x, y) = d(x, q(y))$$



Example (asymmetric distance)

Subspace 1:

(A1, B1, D1, E1, Q1 are sub-vectors in Subspace1)



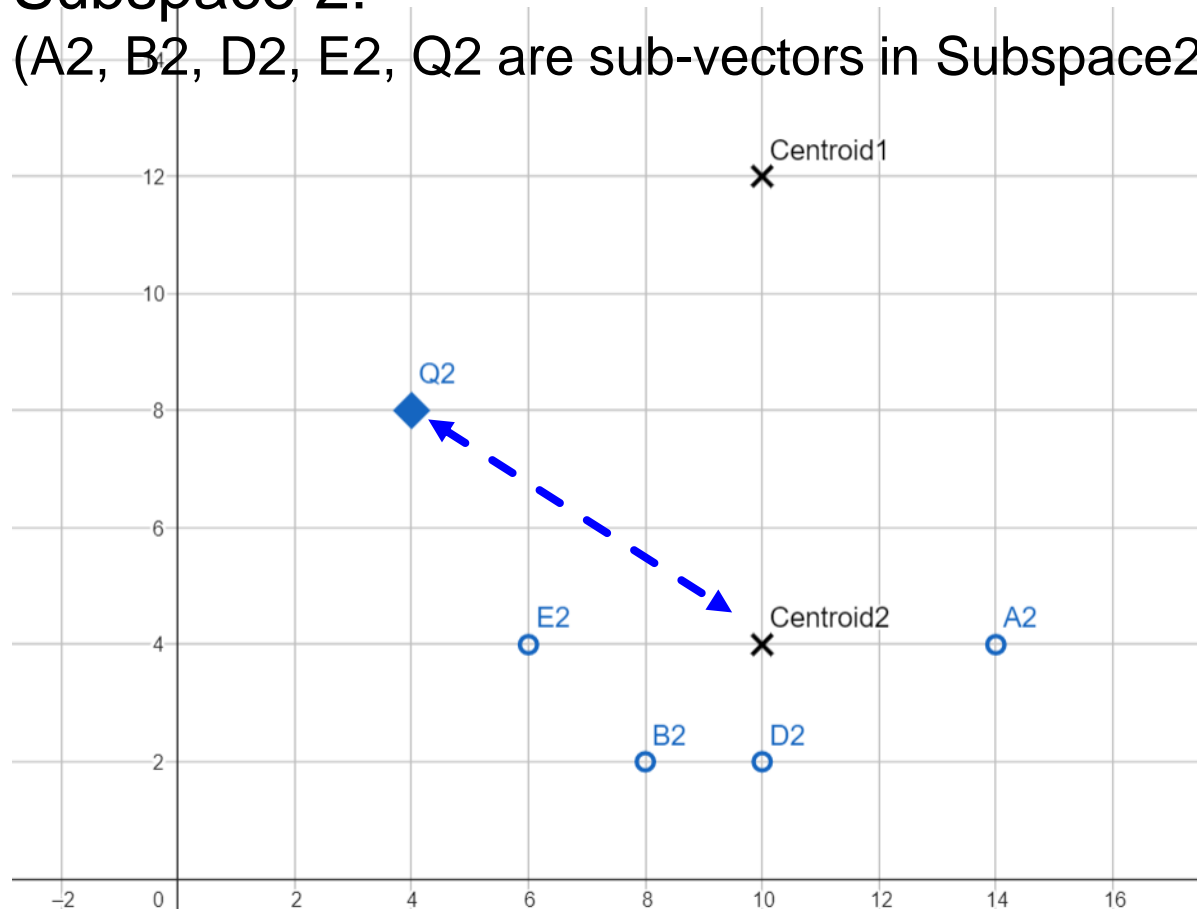
In one subspace, the data points belonging to the same centroid will have equal approximate distances to the query.

$$D(A1, Q1) = D(B1, Q1) = D(D1, Q1) = D(E1, Q1)$$

Example (asymmetric distance)

Subspace 2:

(A2, B2, D2, E2, Q2 are sub-vectors in Subspace2)



In one subspace, the data points belonging to the same centroid will have equal approximate distances to the query.

$$D(A2, Q2) = D(B2, Q2) = D(D2, Q2) = D(E2, Q2)$$

Example (asymmetric distance)

Compression results

A -> [1, 2]

B -> [1, 2]

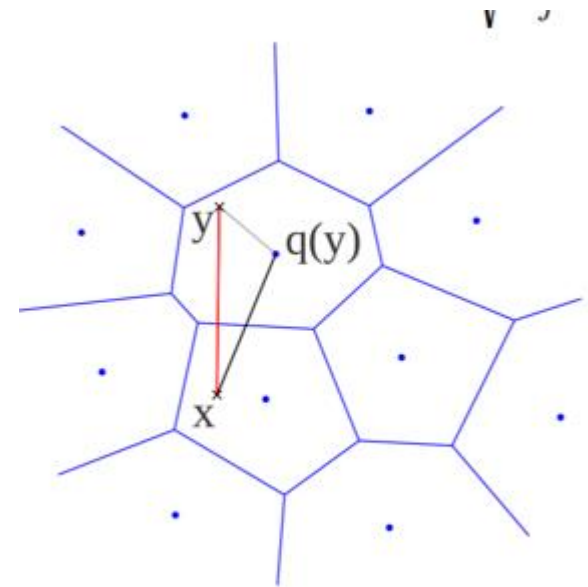
D -> [1, 2]

E -> [1, 2]

As they are all compressed to the same vector, we have Approximate Squared L2 Distances (Asymmetric):

$$D(Q, A) = D(Q, B) = D(Q, D) = D(Q, E)$$

$$\tilde{d}(x, y) = d(x, q(y))$$



asymmetric case

Example (asymmetric distance)

Subspace 1:	
Q1	[16, 4]
C1	[8, 6]
C2	[18, 6]

Subspace 2:	
Q2	[4, 8]
C1	[10, 12]
C2	[10, 4]

Compressed
vectors:

A -> [1, 2]

B -> [1, 2]

D -> [1, 2]

E -> [1, 2]

Subspace 1:

$$D_1(Q_1, \text{Centroid 1}) = 8^2 + 2^2 = 68$$

Subspace 2:

$$D_2(Q_2, \text{Centroid 2}) = 6^2 + 4^2 = 52$$

Approximate Squared L2 Distances (Asymmetric):

$$D(Q, A) = D(Q, B) = D(Q, D) = D(Q, E) = D_1 + D_2 = 120$$

Example (asymmetric distance)

- Asymmetric distance calculation is efficient
 - General case:
 - For one subspace, we only need to calculate K approximate distances. K is the number of centroids in one subspace.
 - K is much less than the total number of data points in the database. E.g., $K=128$ centroids, data points: 10 million

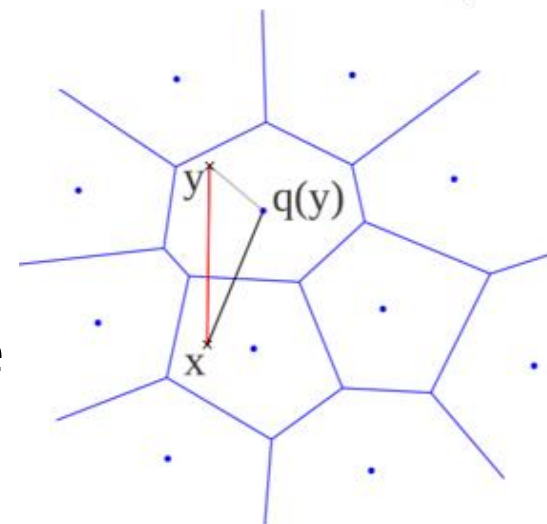
Asymmetric distance

1. Given a query in a top-k retrieval task, we don't need to compute the distance between the query and all the samples in the database.

We only need to calculate the distance between the query and all the centroids in each subspace.

2. The number of centroids are much less than the number of samples in the database, so we can speed up the retrieval process.
3. We don't need to quantize x , which reduces computation compared to the symmetric case.

$$\tilde{d}(x, y) = d(x, q(y))$$



asymmetric case

Asymmetric distance

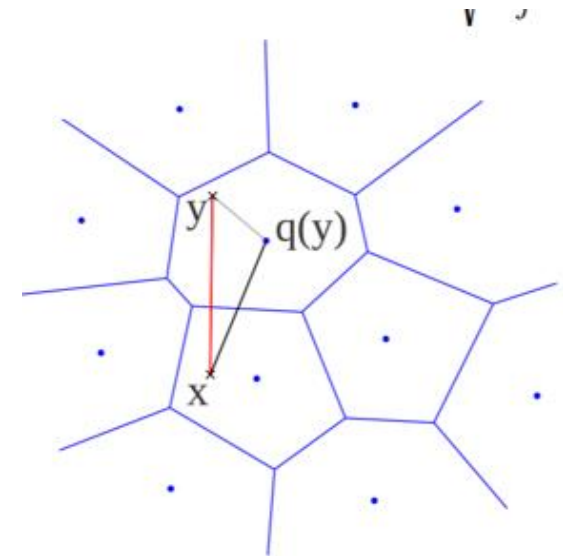
4. For each subspace, we compute the squared L2 distances between the x and all centroids.

This brings extra computation compared to the symmetric case.

5. The above extra computation cost is similar with the quantization cost for x . (cluster assignment requires distance calculation between x and all centroids)

That means symmetric and asymmetric distance have the same computation complexity.

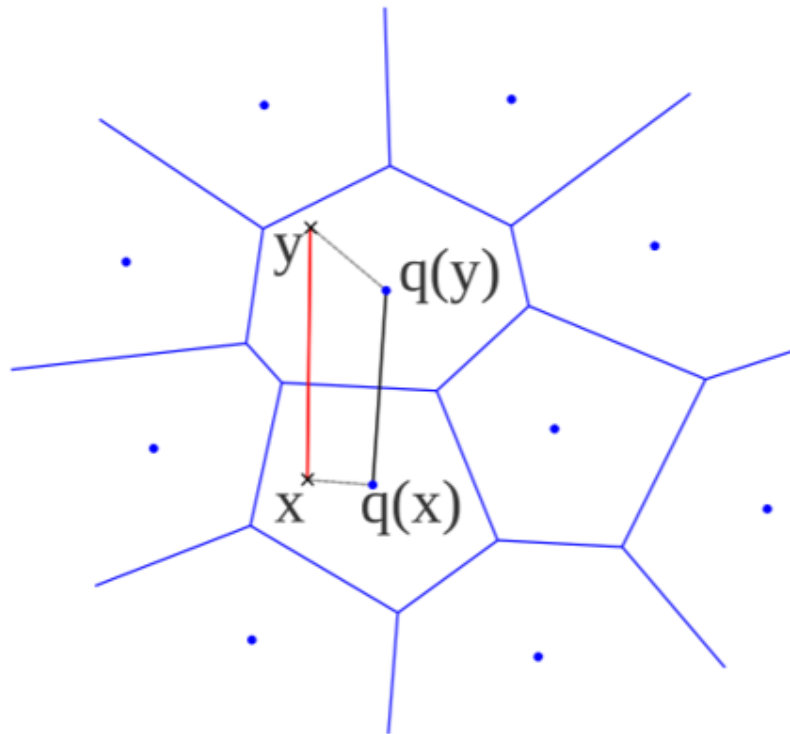
$$\tilde{d}(x, y) = d(x, q(y))$$



asymmetric case

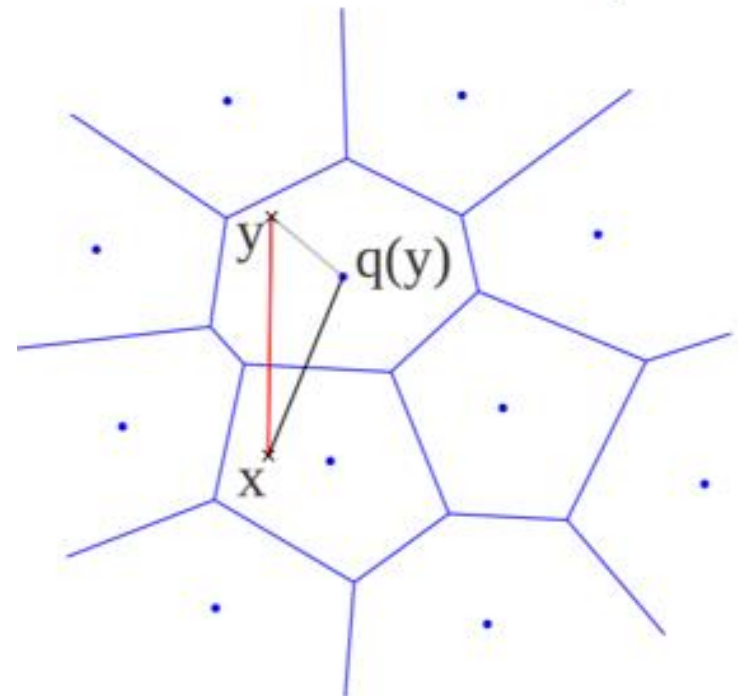
Comparison

$$\hat{d}(x, y) = d(q(x), q(y)) = \sqrt{\sum_j d(q_j(x), q_j(y))^2},$$



symmetric case

$$\tilde{d}(x, y) = d(x, q(y))$$

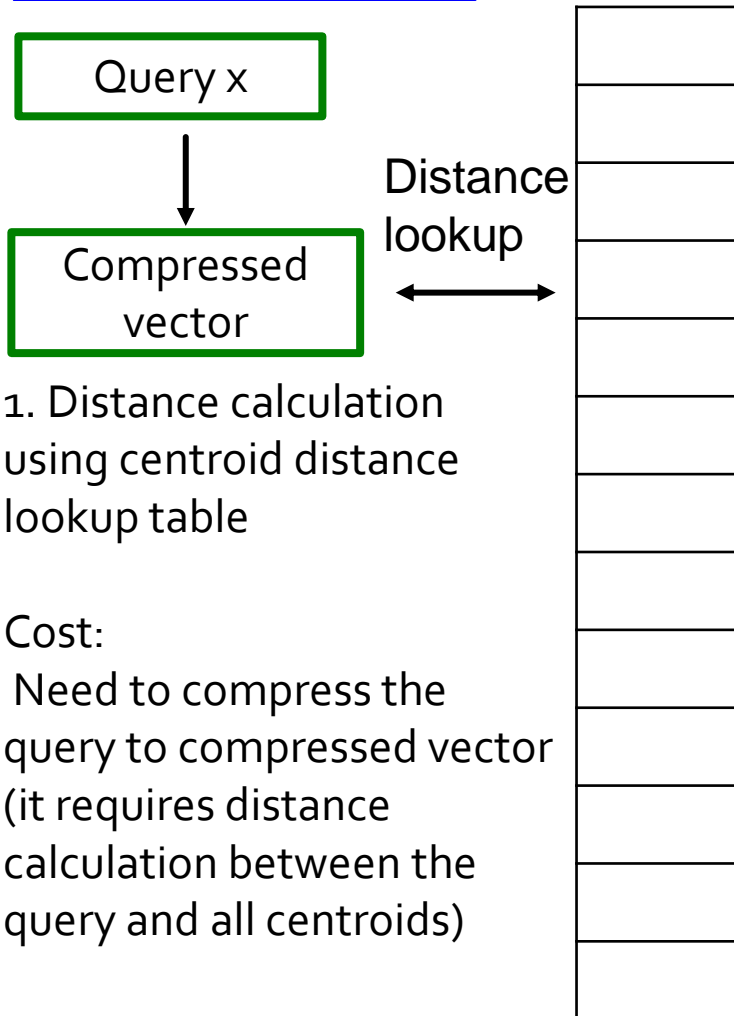


asymmetric case

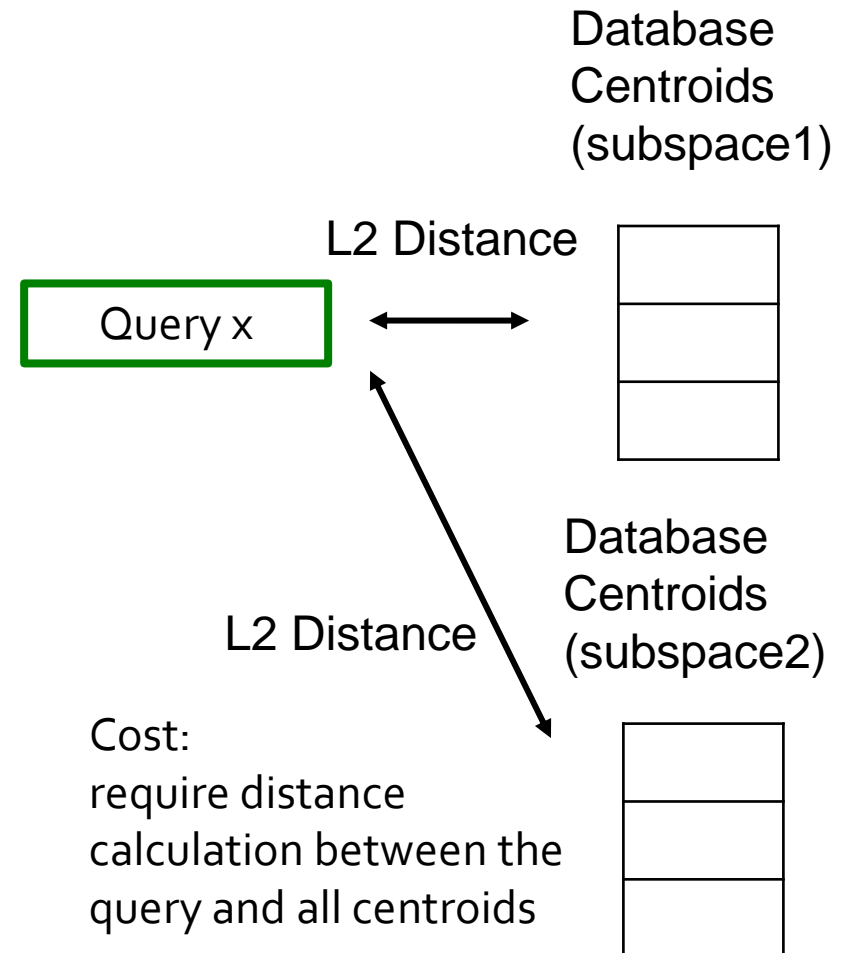
Comparison for top-k search

PQ+
Symmetric distance

Database
All datapoints



PQ + asymmetric distance
for top-k retrieval



Outline

- Nearest Neighbour Search (NNS) Problems
- Local Sensitive Hashing (LSH) – Random Projection
- Product Quantization (PQ)
- **Inverted File Index (non-examinable!!)**

Inverted File Index



Non-examinable!

- Inverted File Index
 - Clustering based method
 - Data dependent
 - Reduce the search scope

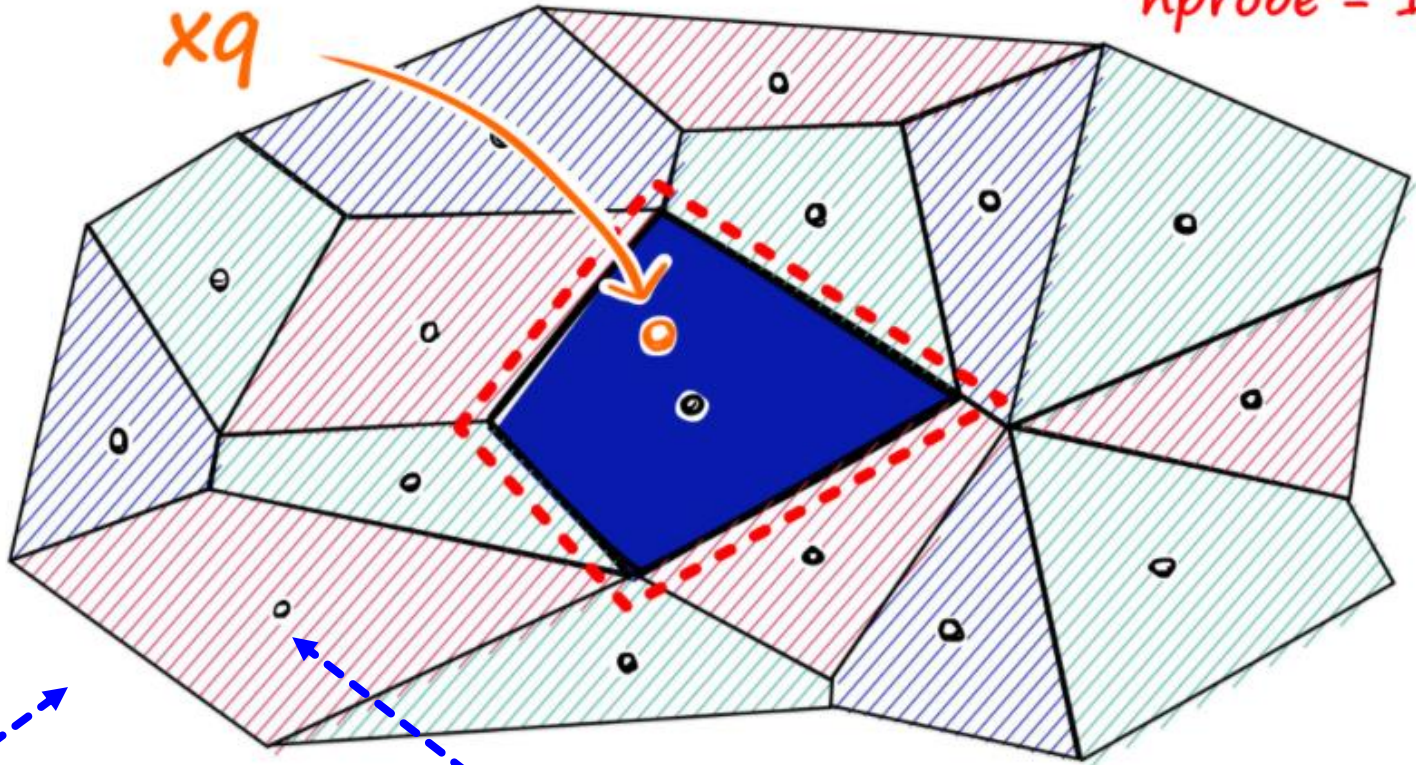


Non-
examinable!

- Inverted File Index (IVT)
 - An inverted index here means a mapping (a lookup table) from cluster centroids (words) to the cluster members.
- Steps:
 - 1. Clustering of the samples in the database, generate the centroids and cells
 - 2. Given a query vector, identify the cells for search scope
 - 3. Use linear search (exhaustive search) to retrieve results within the selected cells

x_q : the query vector

search scope
 $n_{probe} = 1$

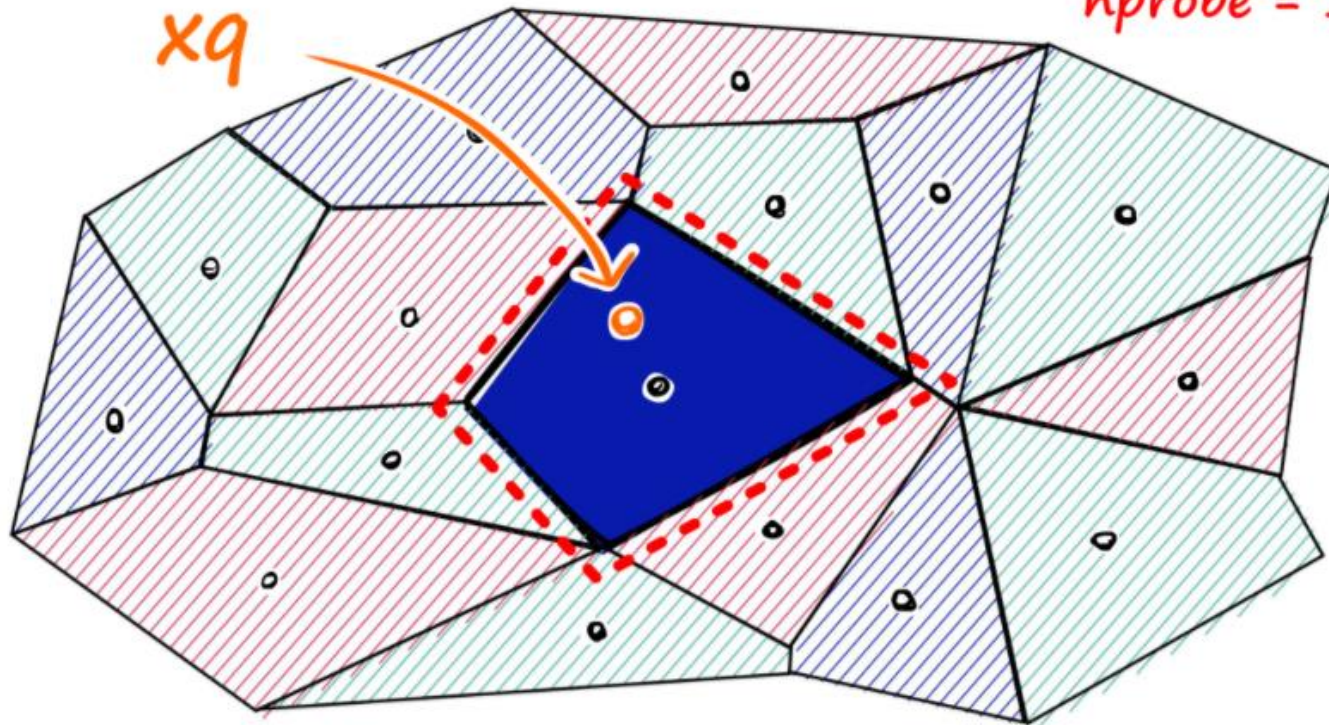


Perform k-means clustering
to group the data points in
the database (cluster
centroids and assignments)

Cluster centroid in one cell.
One cell represents one cluster

xq: the query vector

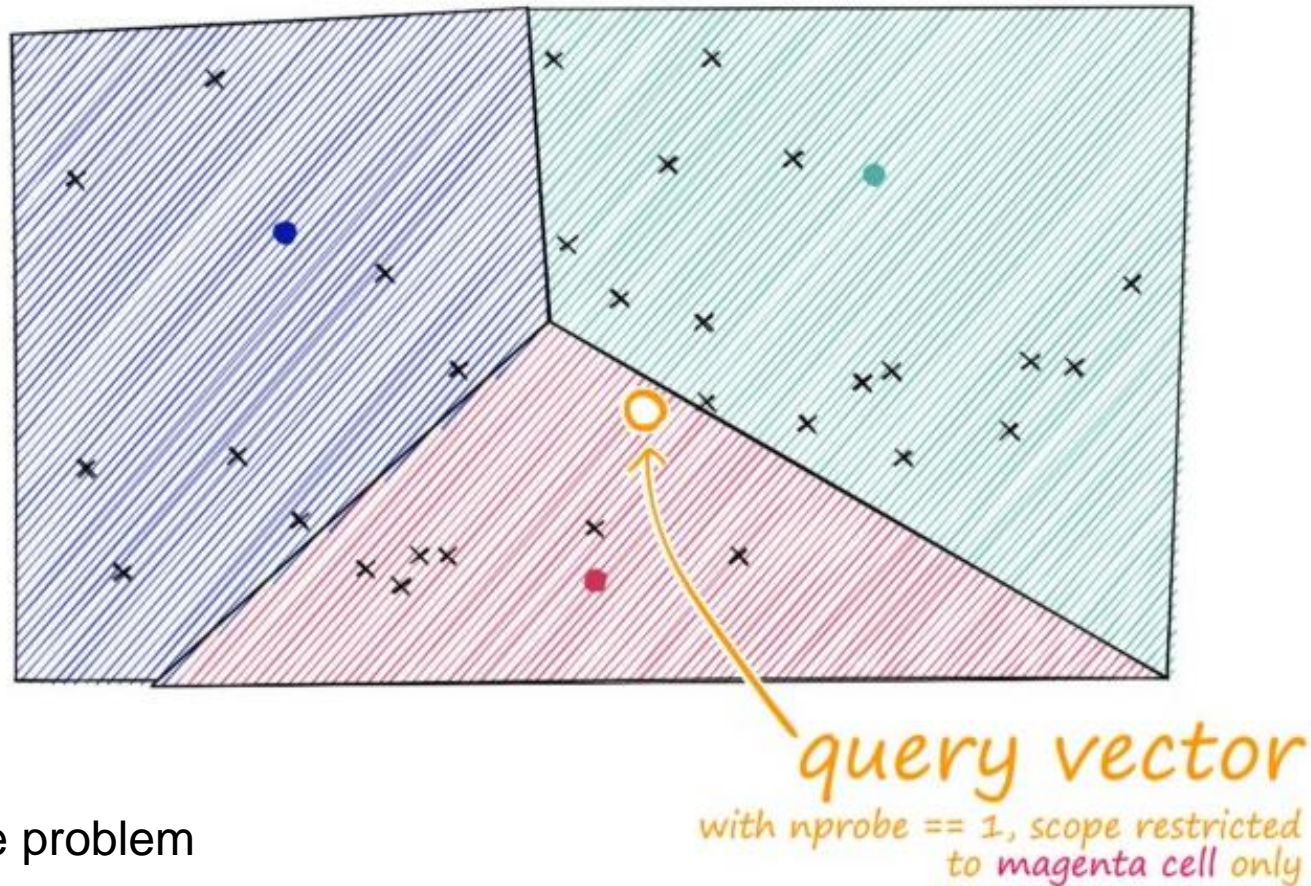
search scope
nprobe = 1



nprobe: select the top k nearest clusters (cells) as the search scope
nprobe=1: only select 1 cell for searching

There are 2 parameters in IVT:

1. nprobe: the number of cells to search
2. The number of cells (clusters) to create.

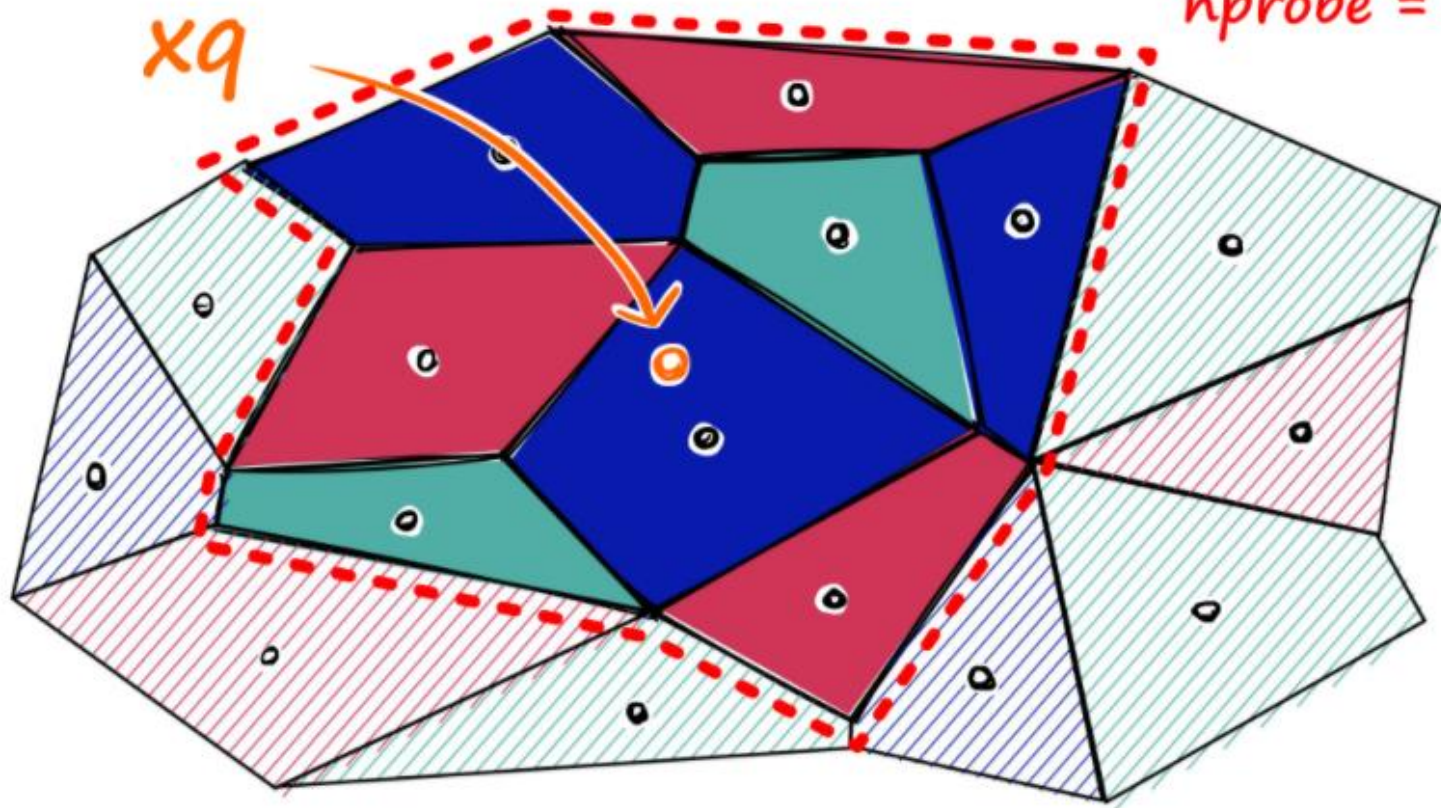


Edge problem

Our query vector \mathbf{x}_q lands on the edge of the magenta cell. Despite being closer to datapoints in the teal cell, we will not compare these if $nprobe == 1$ — as this means we would restrict search scope to the magenta cell only.

x_q : the query vector

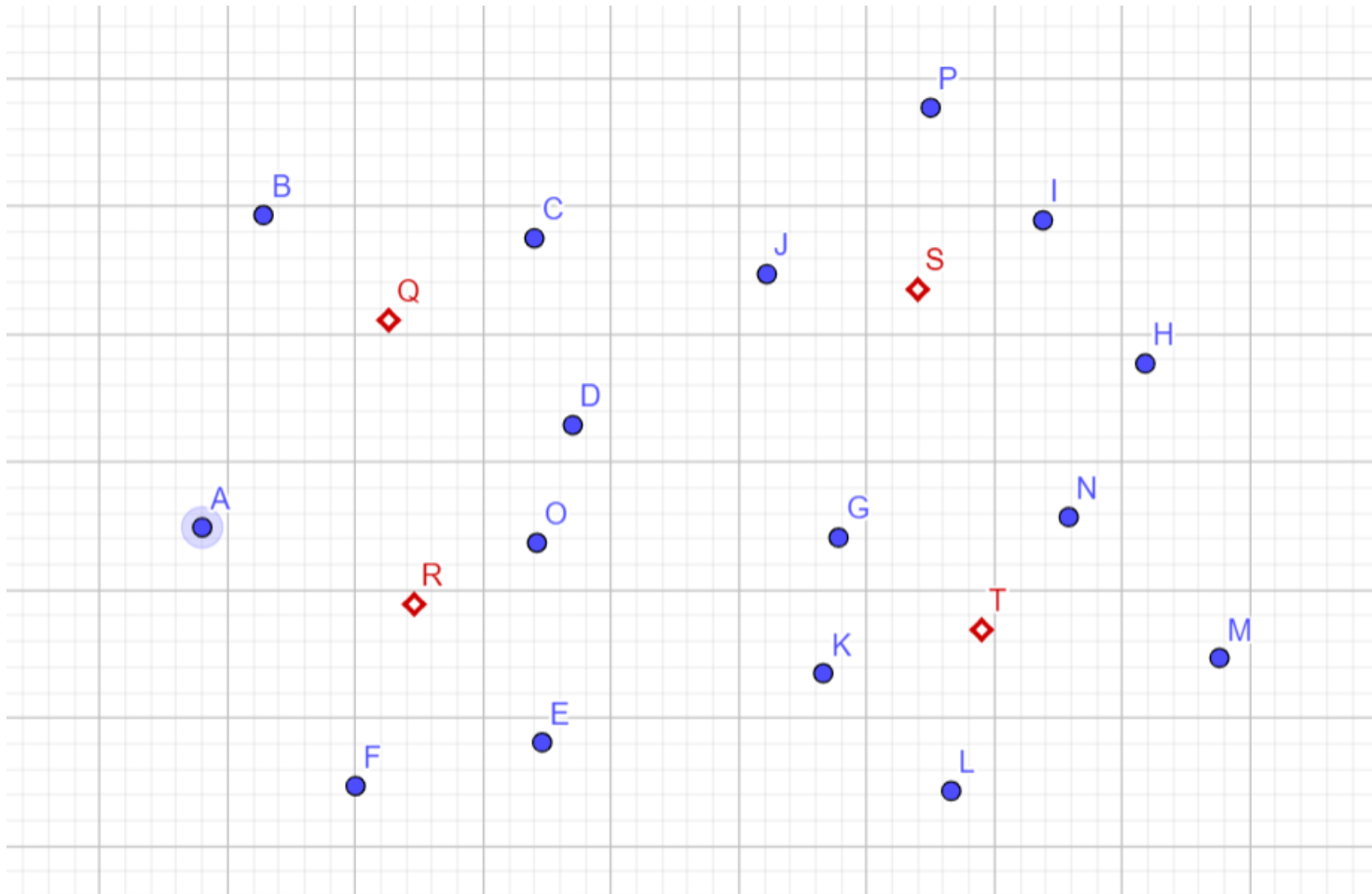
search scope
 $n_{probe} = 8$



Increasing **nprobe** increases our search scope.

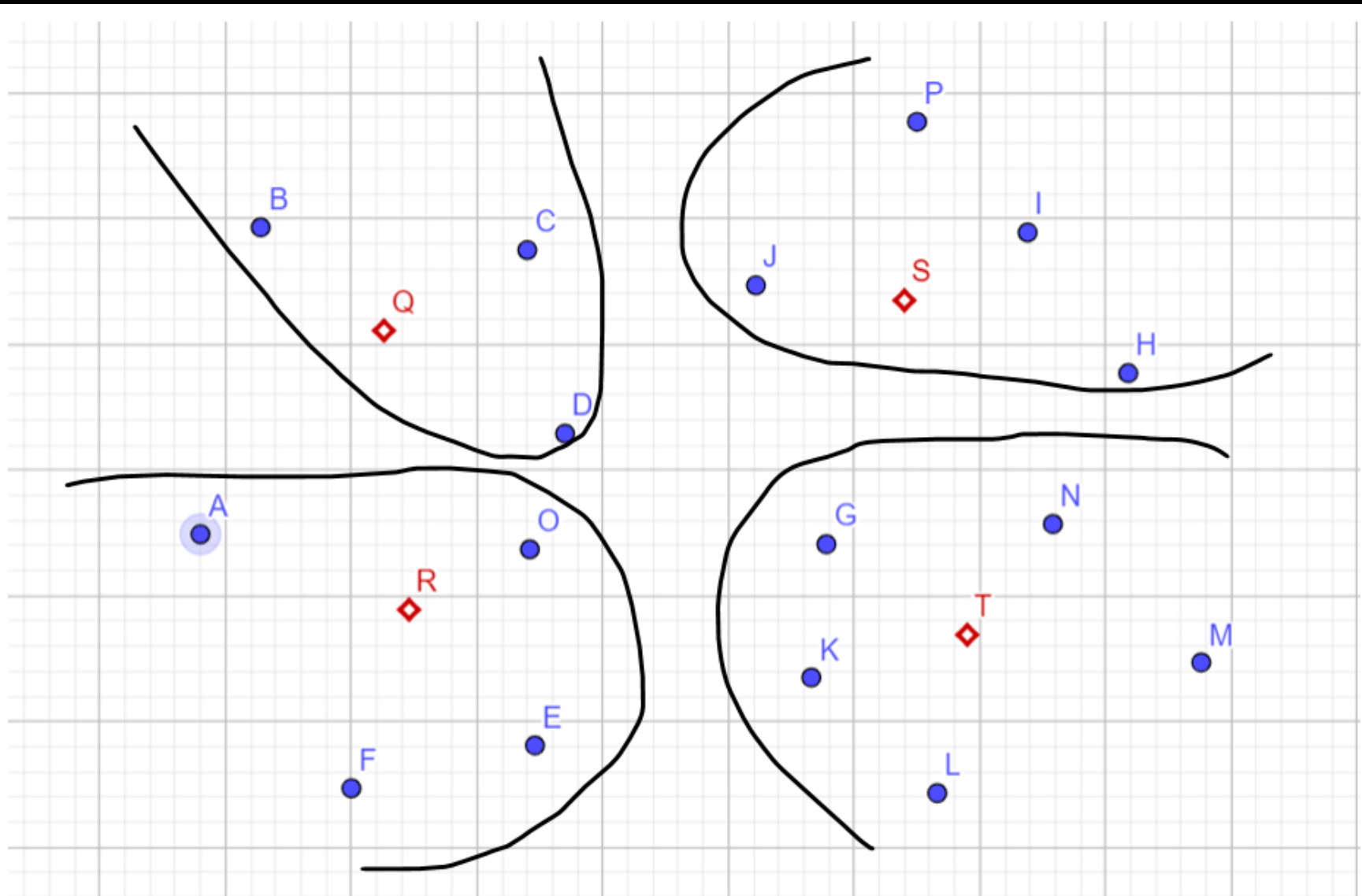
$n_{probe}=8$: select the top 8 nearby cells for searching.
(ranked by the distance between the query and the centroids)

■ Inverted File Index Example

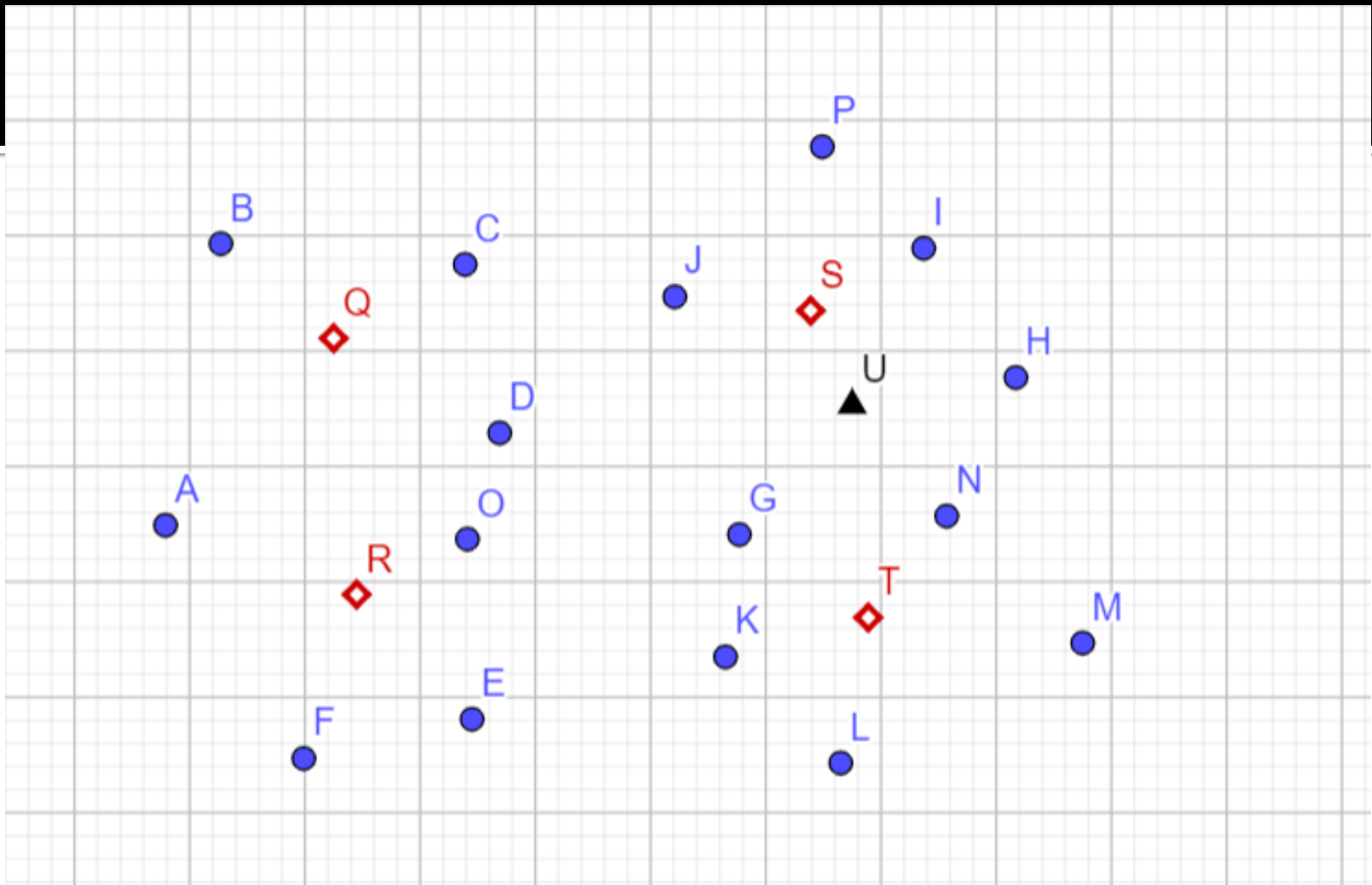


Q, R, S, T are centroids, all other points are the data samples in the database.

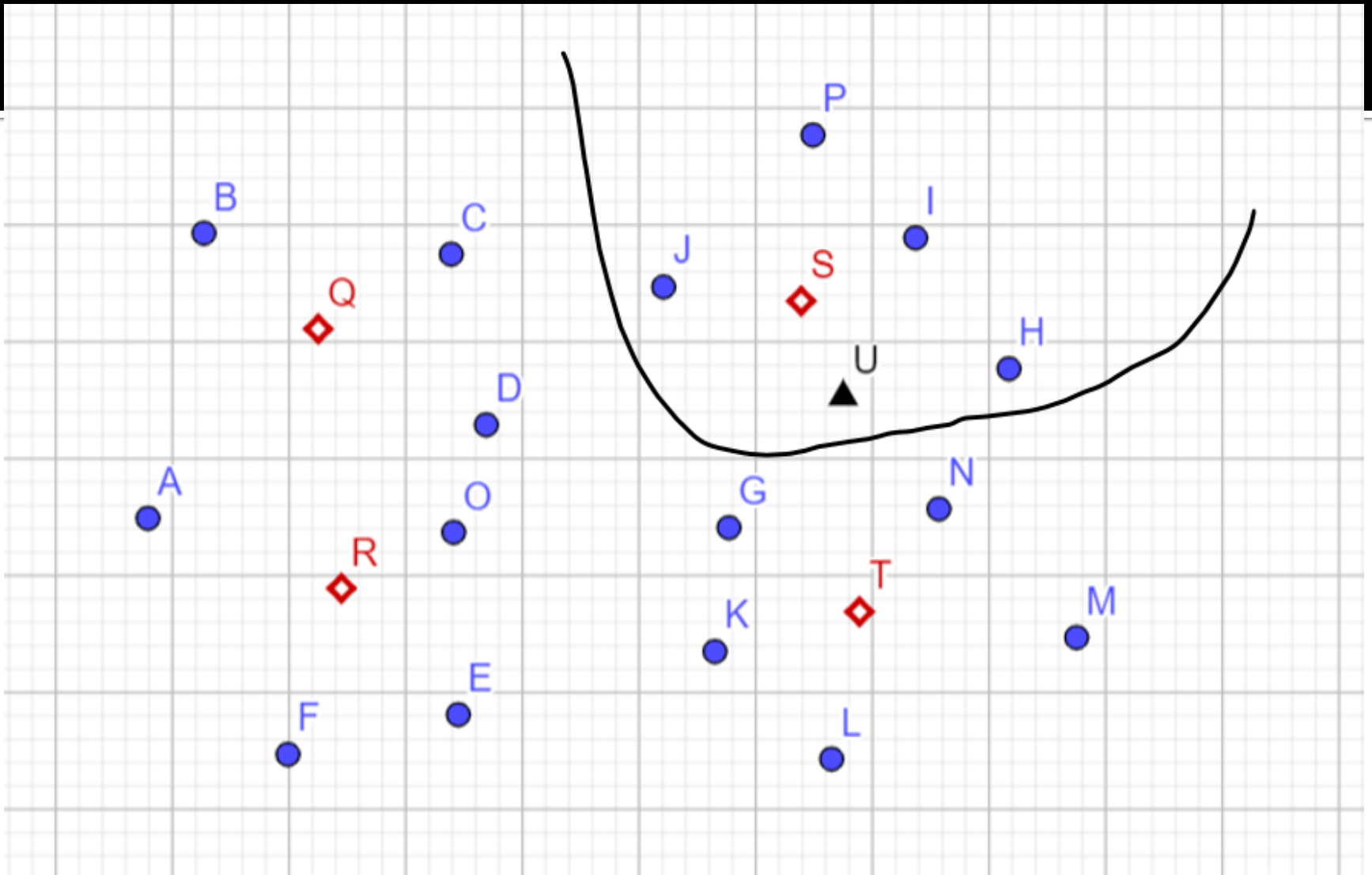
- $Q(a)$: assign the samples to the cells defined by the centroids.



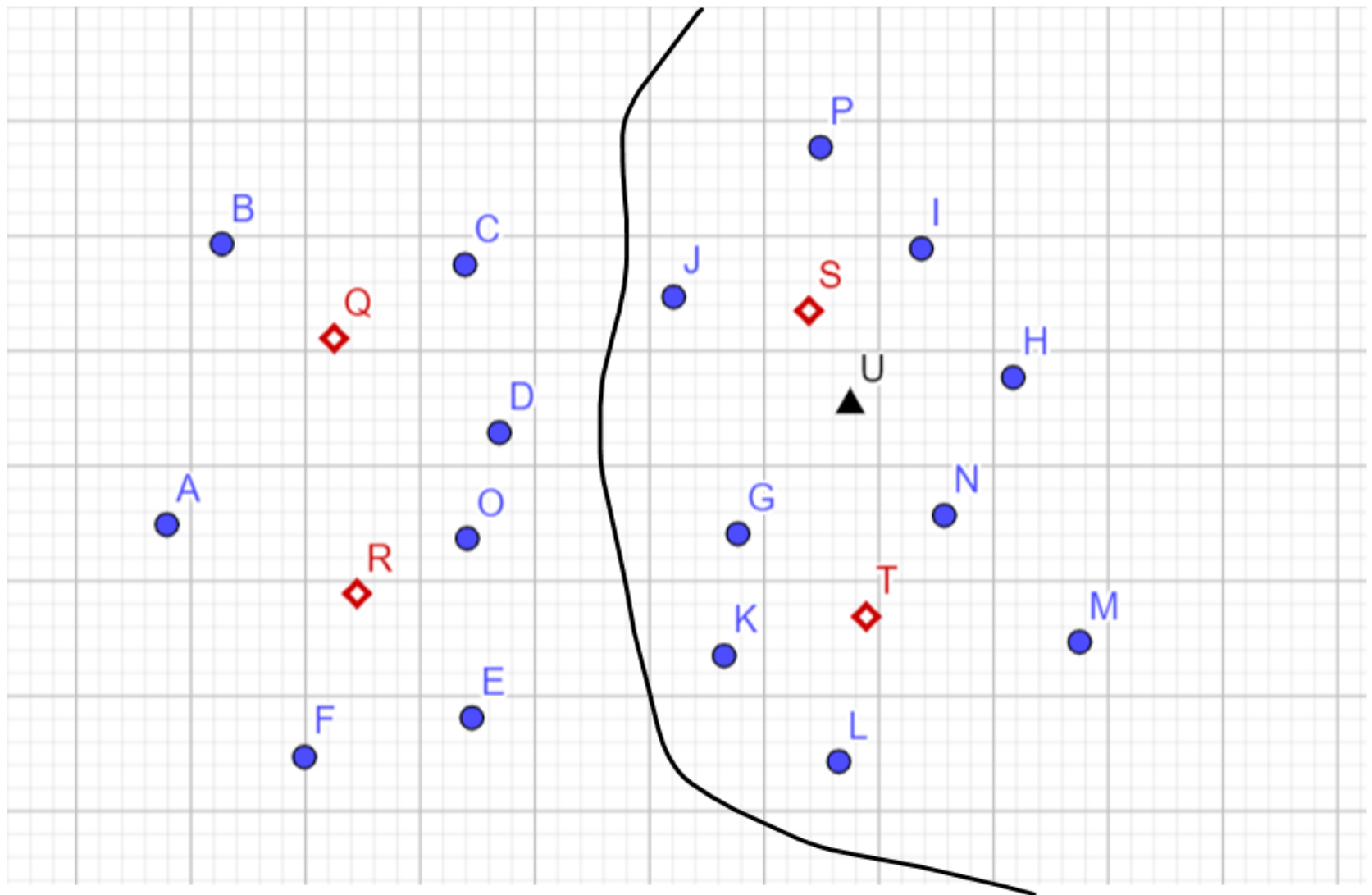
one data point is assigned to its nearest centroid based on L2 distance



Q(b) Given the query sample U,
list the candidate samples in the database for similar search
when $nProbe=1$ and $nProbe=2$, respectively



nProbe=1: only search in 1 cell

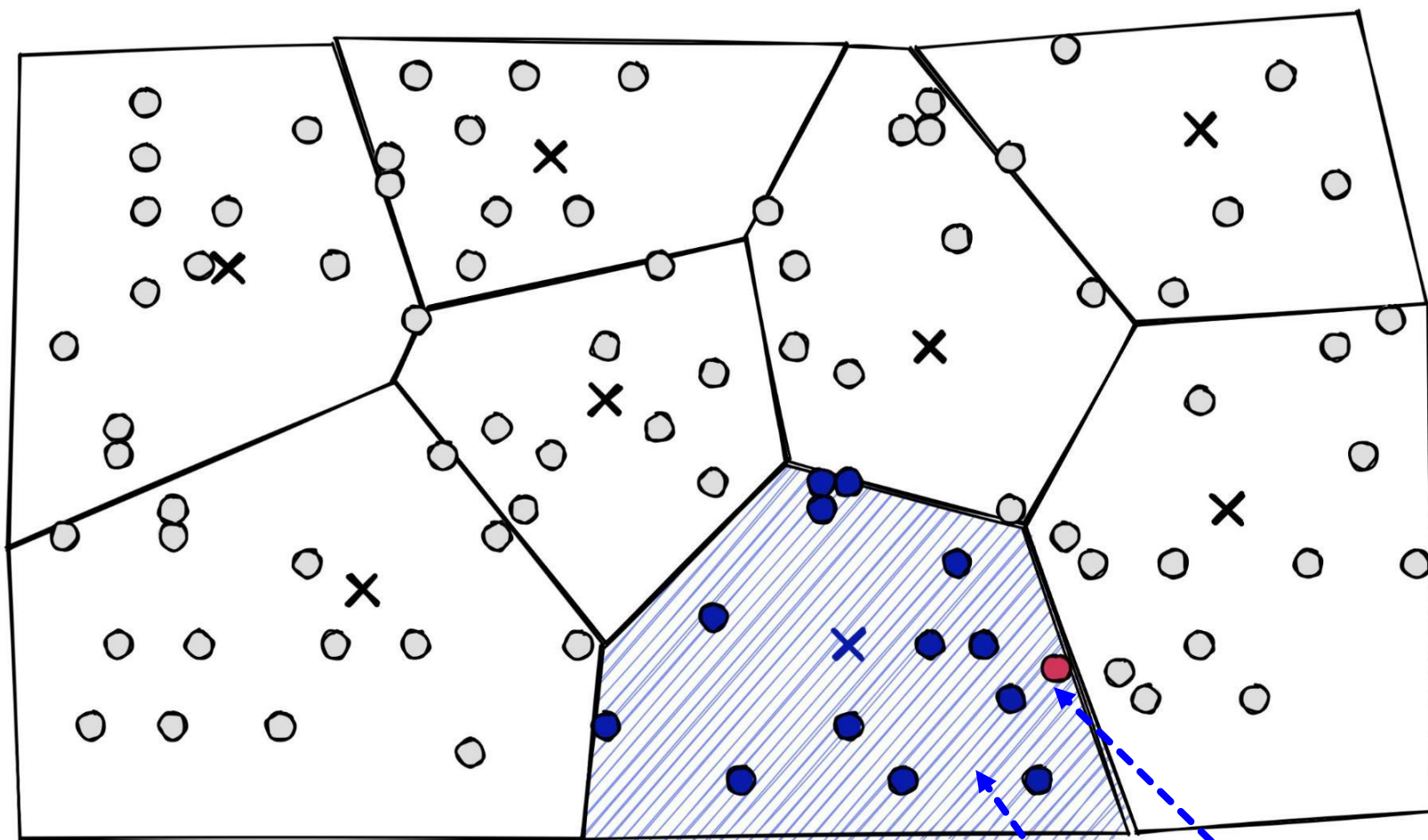


nProbe=2: search in 2 cells (the top 2 closest cells)

PQ+IVF



- PQ+IVF
 - Product Quantization + Inverted File Index
 - A hierarchical solution
 - Much better performance in searching
 - Reduce search scope by using IVF



Step1: use IVF to identify search scope (cells).
 IVF allows us to restrict our search in the target cells only.

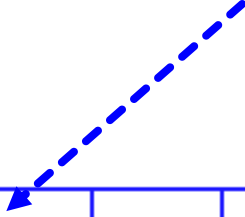
Step2: perform PQ based linear search in the target cells. PQ generates
 compresses vectors and uses fast distance calculation

query

Experiment:

Sift1M dataset, 2048 centroids for each subspace

L2 distance base linear search



	FlatL2	PQ	IVFPQ
Recall (%)	100	50	52
Speed (ms)	8.26	1.49	0.09
Memory (MB)	256	6.5	9.2

<https://www.pinecone.io/learn/product-quantization/>

Online resources

■ FAISS

- Faiss is a library for efficient similarity search and clustering of dense vectors.
- <https://github.com/facebookresearch/faiss/wiki>