Part 1

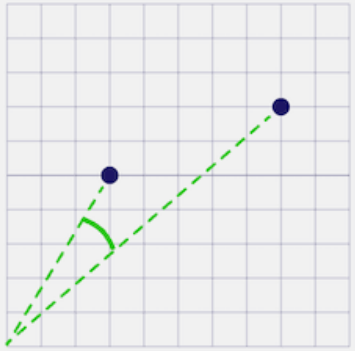
Tasks

- **Task 1: Precise KNN Top 10**
- **Task 2: KMeans and ANN**

Task 1.1: Distance

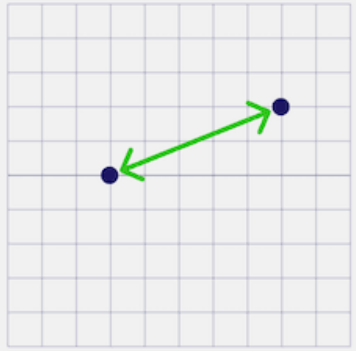
Implement distance functions for two vectors in cupy/torch/triton.
Input:
D: dimension of the vector.
X[D], Y[D]: two vectors
Output: the distance between X and Y

There are four distinct types of distance, so the implementation of four separate functions.



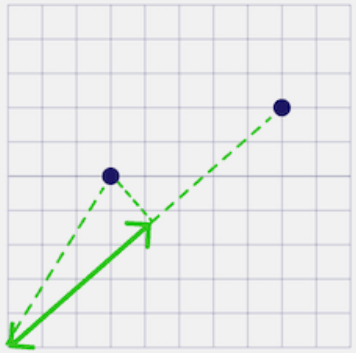
Cosine Distance

$$1 - \frac{A \cdot B}{||A|| \quad ||B||}$$



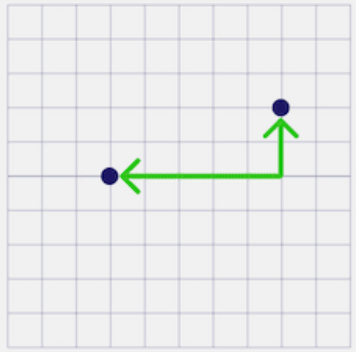
Squared Euclidean (L2 Squared)

$$\sum_{i=1}^{n} (x_i - y_i)^2$$



Dot Product

$$A \cdot B = \sum_{i=1}^{n} A_i B_i$$



Manhattan (L1)

$$\sum_{i=1}^{n} |x_i - y_i|$$

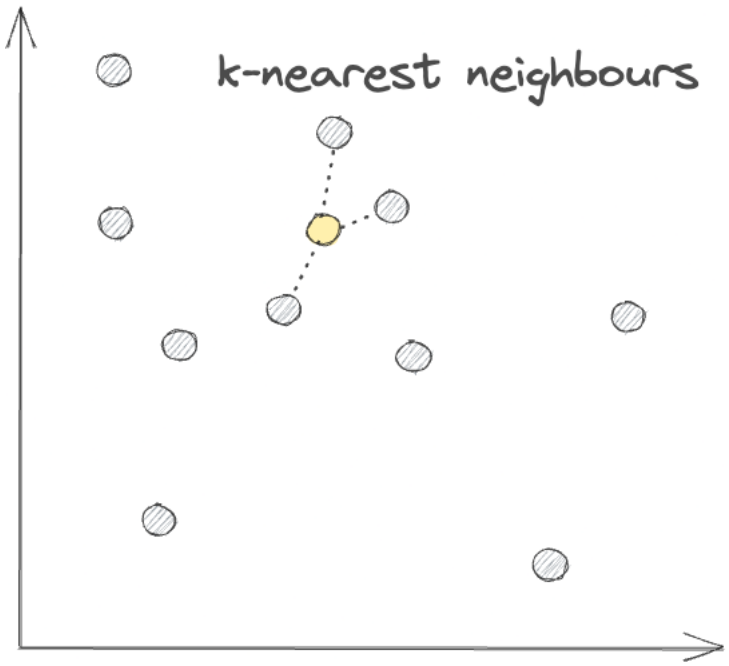
Task 1.2: Top-K with GPU

Under the same environment as task 1.1

Identify the K nearest vectors within a set of vectors.

Input:
- N: Number of vectors
- D: Dimension of vectors
- A[N, D]: A collection of vectors
- X: A specified vector
- K: Top K

Output:
- Result[K, D]: The top K nearest vectors



Task 1: KNN

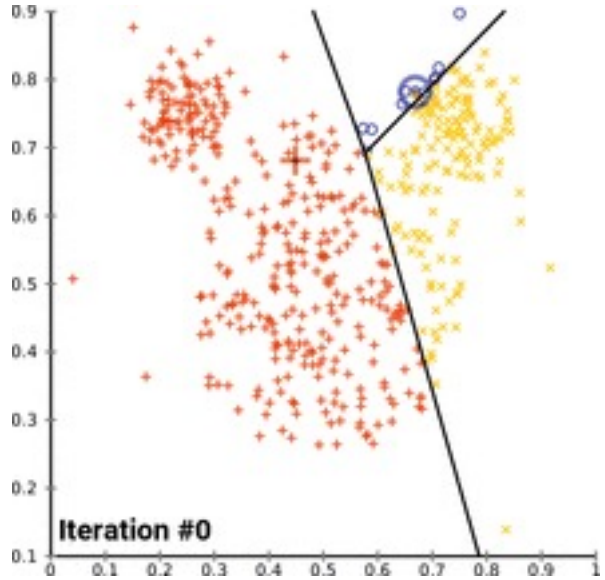
- In the Task 1 report, you are required to address the following questions:
  1. How did you implement four distinct distance functions on the GPU?
  2. What is the speed advantage of the GPU over the CPU version when the dimension is 2? Additionally, what is the speed advantage when the dimension is 1024?
  3. Please provide a detailed description of your Top K algorithm.
  4. What steps did you undertake to implement the Top K on the GPU? How do you manage data within GPU memory?
  5. When processing 4,000 vectors, how many seconds does the operation take? Furthermore, when handling 4,000,000 vectors, what modifications did you implement to ensure the effective functioning of your code?

Task 2: Kmeans and ANN

- What is Kmeans algorithm:
- https://en.wikipedia.org/wiki/K-means_clustering

- K-means clustering is an unsupervised learning algorithm that groups similar data points into K clusters. It works by iteratively assigning points to the nearest cluster center (centroid) and updating centroids based on the mean of assigned points until convergence.
- In this task we only use L2 distance and cosine similarity.

1. Select K random points as initial centroids
2. REPEAT:
   - Assign each point to nearest centroid
   - Update centroids by calculating mean of points
3. UNTIL centroids don't change



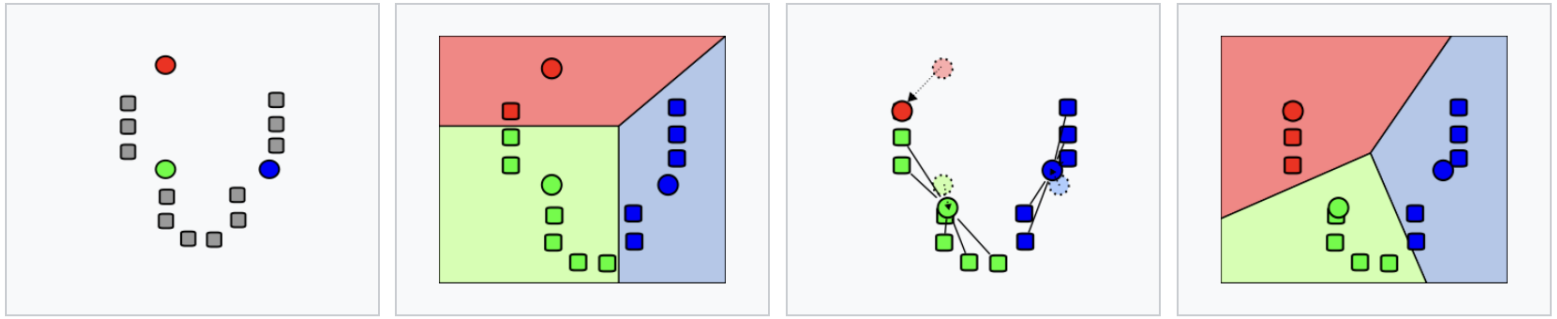
Task 2.1: Implement Kmeans on GPU

Kmeans cluster algorithm

Input:
- N: Number of vectors
- D: Dimension of vectors
- A[N, D]: A collection of vectors
- K: number of clusters

Output:
- R[N]: cluster ID for each vector

Demonstration of the standard algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

Task 2.2: Implement ANN algorithm on GPU

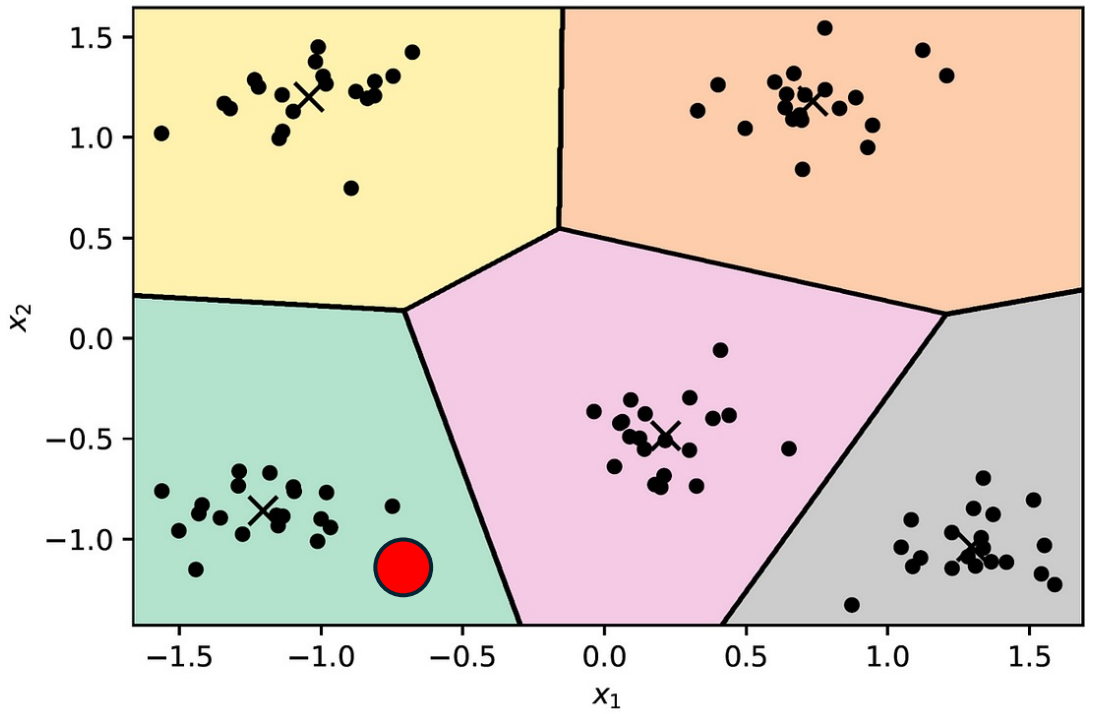
- **Input:**
  - N: Number of vectors
  - D: Dimension of vectors
  - A[N, D]: A collection of vectors
  - X: A specified vector
  - K: Top K

- **Output:**
  - Result[K]: The top K nearest vectors ID (index of the vector in A)

Task 2.2 ANN

- 🔴 is our query target
- If we need to find the top 2 vectors in the entire graph, it might be quicker to scan just 2 groups:

🟩 and 🟪



The clustering result could become a “vector search” index.

Recall rate

- Keep in mind that the ANN algorithm is merely an ***approximation algorithm.*** We determine the recall rate by comparing the results with your KNN results.

- Recall rate = (#Same vectors in KNN result and ANN result) / K

- If the recall rate exceeds ***70%*** across all data, we consider your result to be correct.

(You can also implement other ANN algorithms such as HNSW or IVFPQ. However, you cannot use libraries other than cupy/triton/pytorch. In other words, you cannot use libraries like faiss/milvus/cuvs to complete the task in just one line of code.)

Task 2: Kmeans and ANN

- In the Task 2 report, you are required to address the following questions:
  1. How did you implement your K-means algorithm on the GPU?
  2. What is the speed advantage of the GPU over the CPU version when the dimension is 2? Additionally, what is the speed advantage when the dimension is 1024?
  3. Please provide a detailed description of your ANN algorithm.
  4. If you implemented another clustering algorithm/ANN algorithm, which algorithm did you use?

Code Template:

- The code template is at:

  https://github.com/ed-aisys/edin-mls-25-spring/tree/main/task-1