

Nhóm 5 - Chủ đề 4

FINDING SIMILAR ITEMS

Tìm kiếm tương tự

Thành viên nhóm:

Hoàng Hữu Đức	23020046
Lê Minh Tuấn	23020149
Phạm Minh Thông	23020164
Bùi An Huy	23020079
Lê Minh Đạt	23020037

Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

3. Biểu diễn dữ liệu

4. Định nghĩa tương đồng

5. Các phép đo độ tương đồng

6. Các kỹ thuật tìm cặp tương đồng

Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

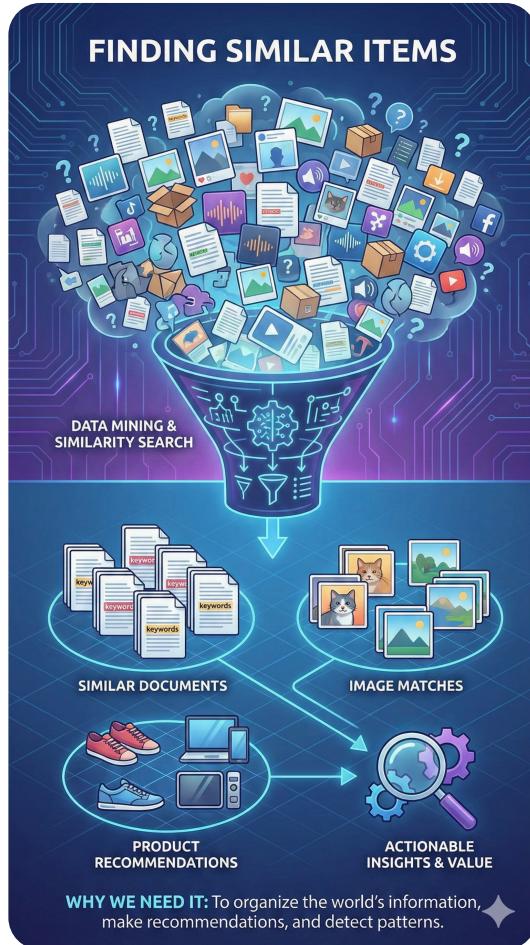
3. Biểu diễn dữ liệu

4. Định nghĩa tương đồng

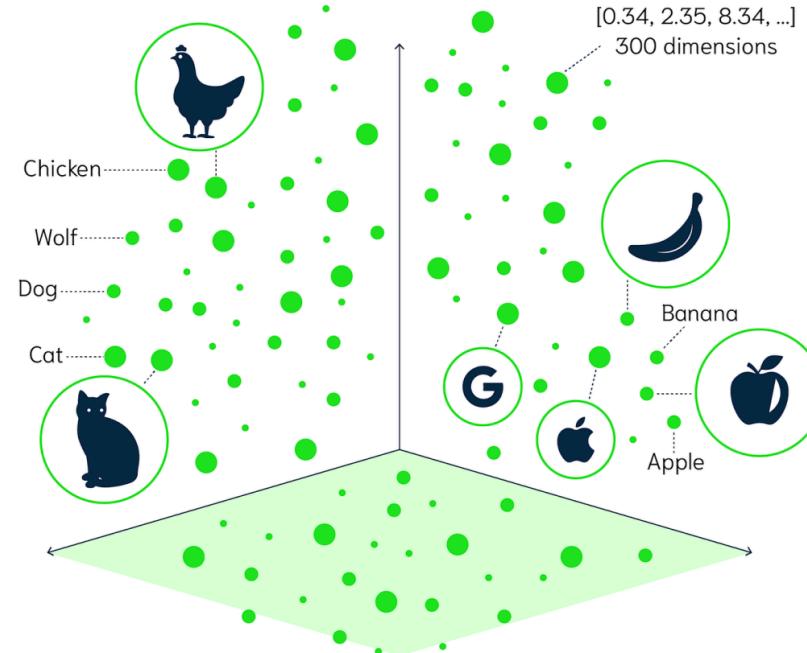
5. Các phép đo độ tương đồng

6. Các kỹ thuật tìm cặp tương đồng

Giới thiệu



Hình 1: For?



Hình 2: Present



Hình 3: Find

Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

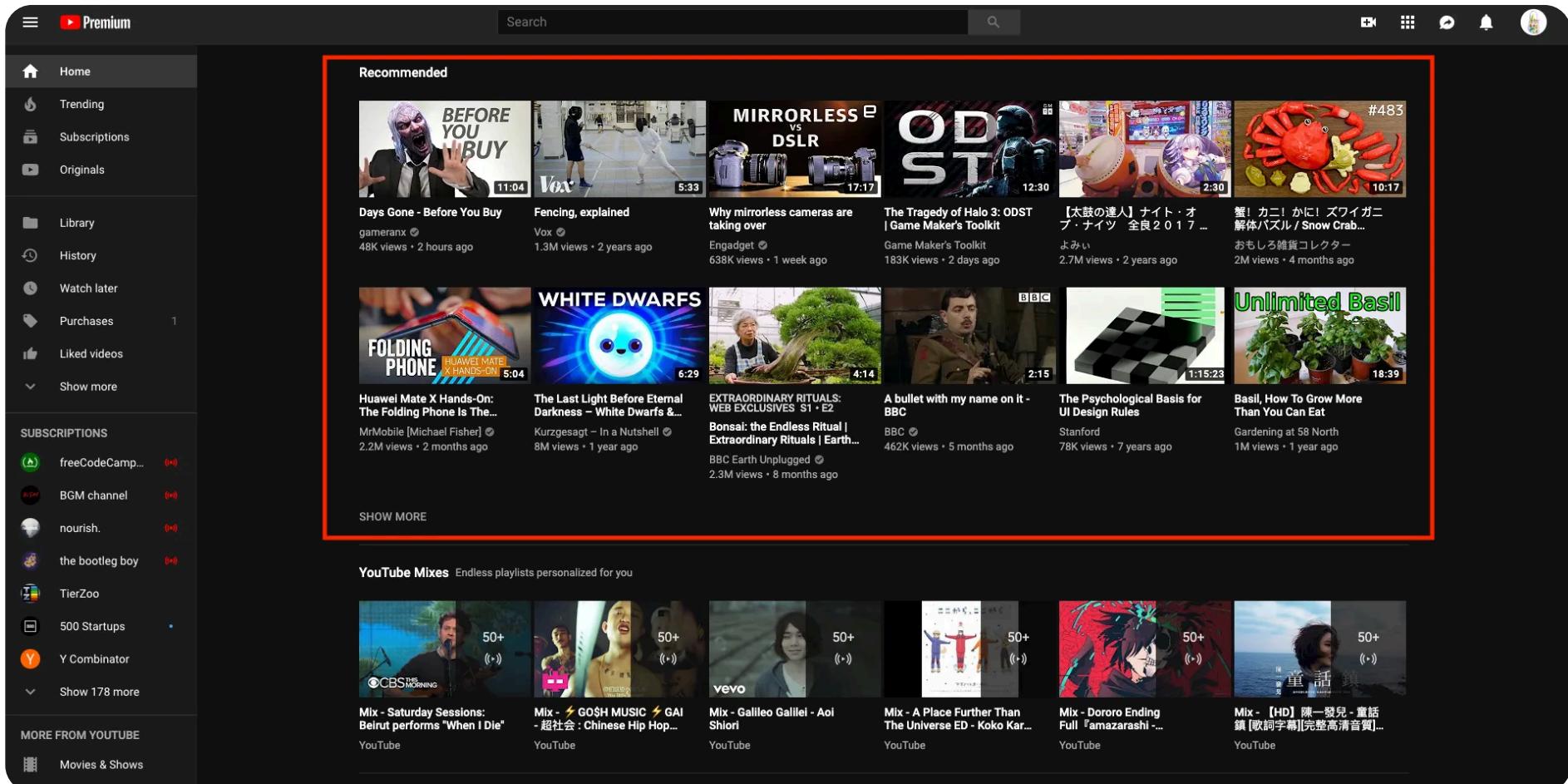
3. Biểu diễn dữ liệu

4. Định nghĩa tương đồng

5. Các phép đo độ tương đồng

6. Các kỹ thuật tìm cặp tương đồng

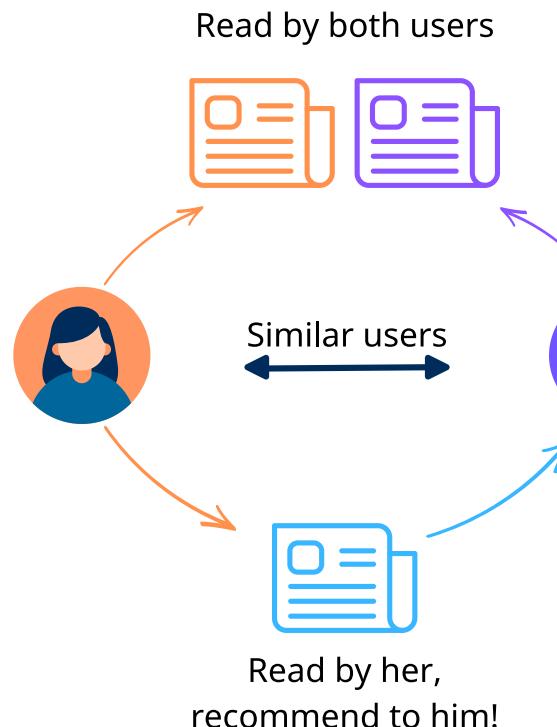
2.1. Hệ thống đề xuất



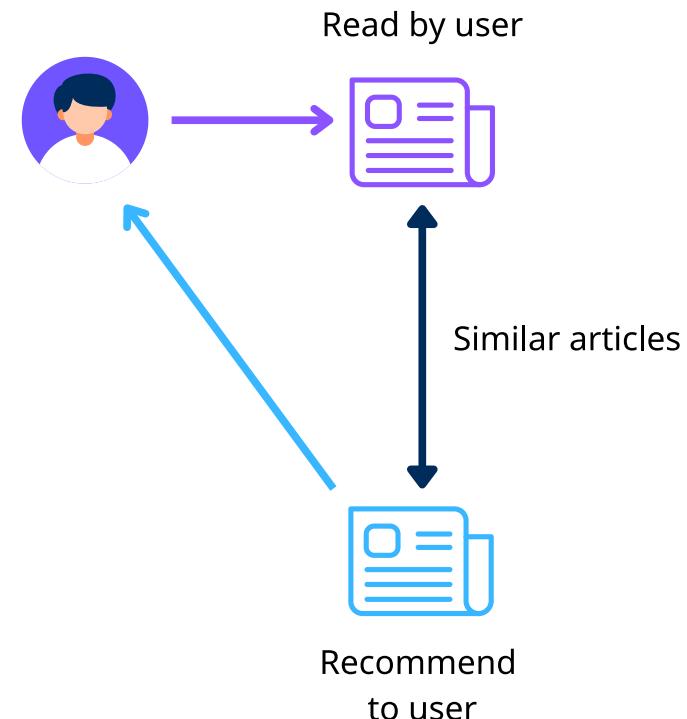
Hình 4: Hệ thống đề xuất trên YouTube

2.1. Hệ thống đề xuất

COLLABORATIVE FILTERING



CONTENT-BASED FILTERING



Hình 5: Các loại hệ thống đề xuất

2.2. Nhận diện trùng lặp văn bản

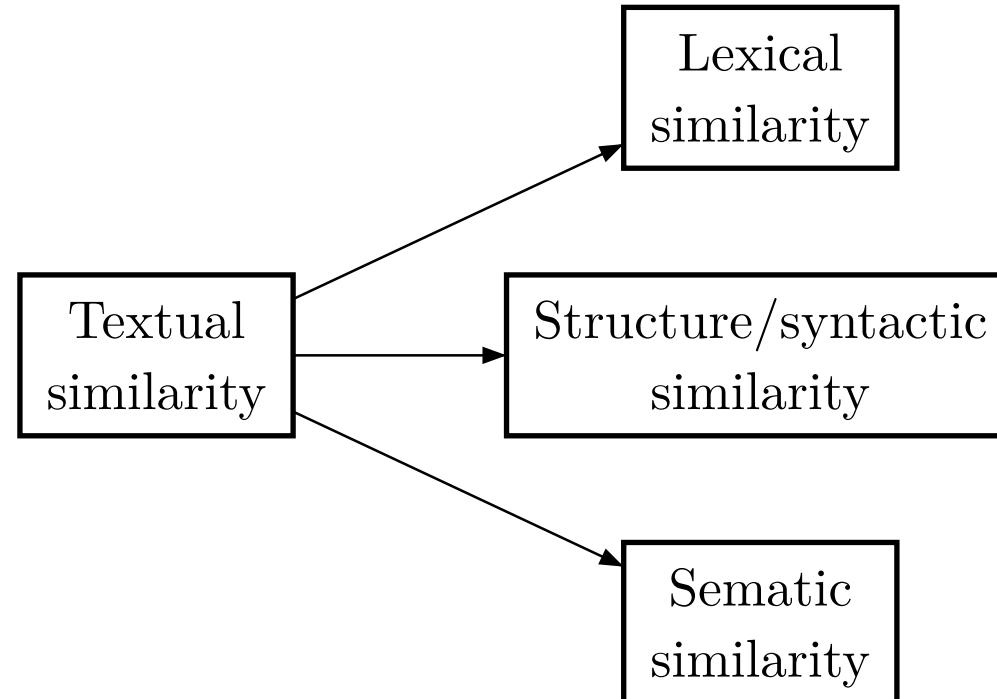
The screenshot shows the Turnitin software interface. On the left, a document titled "Exegesis paper.docx" is displayed. The first section is "INTRODUCTION", which discusses Romans 8:12-17 as one of the most remarkable paragraphs in the New Testament, quoting Paul's teaching on the Spirit's work in believers. The second section is "TEXT AND STRUCTURE", with a "Translation" section containing Greek text from Romans 8:12-17 and its English translation. Below the document are "Share" and search buttons. On the right, the "Submission Details" tab is selected, showing a similarity report. The overall similarity is 40%. The top sources are listed as follows:

Rank	Source	Similarity (%)
1	dokumen.pub INTERNET	2%
2	ebin.pub INTERNET	2%
3	sbts-wordpress-uploads.s3.amazonaws.com INTERNET	1%
4	bibliotekanauki.pl INTERNET	1%
5	blogs.corban.edu INTERNET	<1%
6	ourarchive.otago.ac.nz INTERNET	<1%
7	repository.sbts.edu INTERNET	<1%
8	dartmouthbible.org INTERNET	<1%

Page 1 of 26

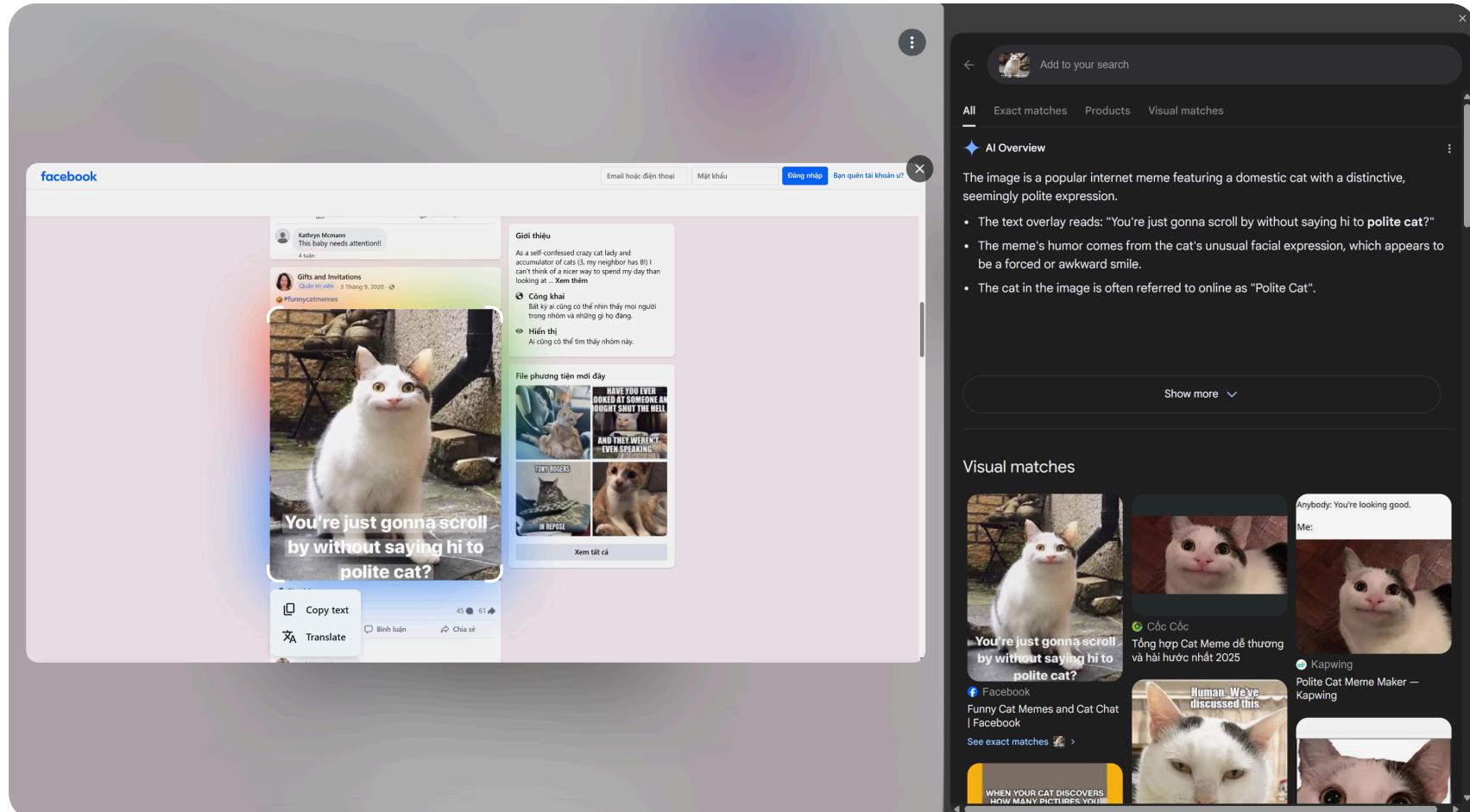
Hình 6: Giao diện phần mềm Turnitin

2.2. Nhận diện trùng lặp văn bản



Hình 7: Các loại tương đồng văn bản

2.3. Tìm hình ảnh giống nhau



Hình 8: Tìm kiếm hình ảnh tương tự sử dụng Google Lens

Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

3. Biểu diễn dữ liệu

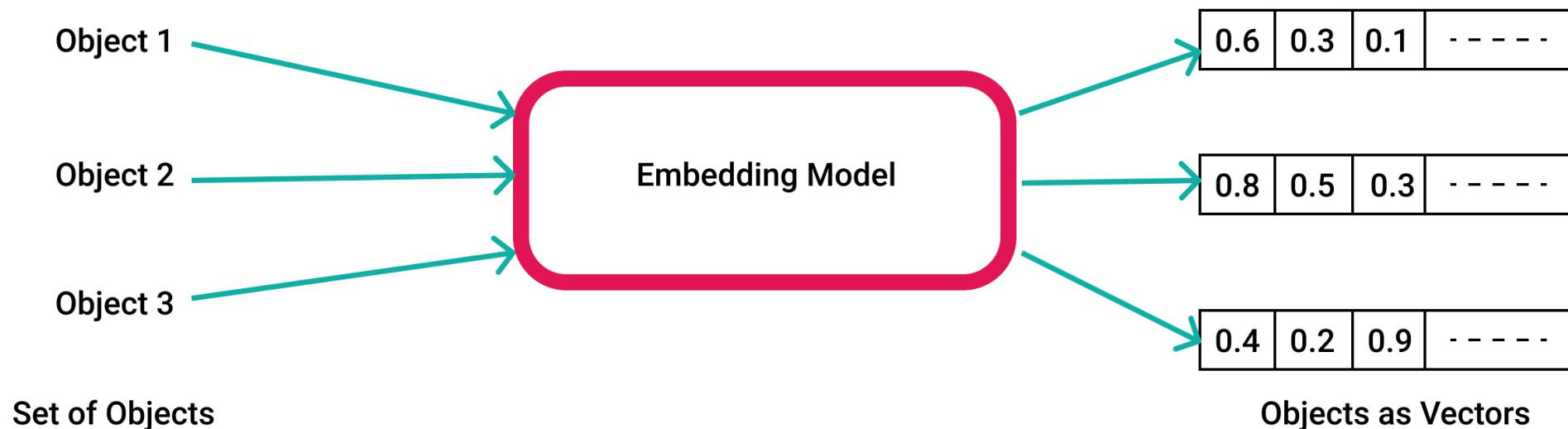
4. Định nghĩa tương đồng

5. Các phép đo độ tương đồng

6. Các kỹ thuật tìm cặp tương đồng

3.1. Các loại dữ liệu

3.1.1. Vector

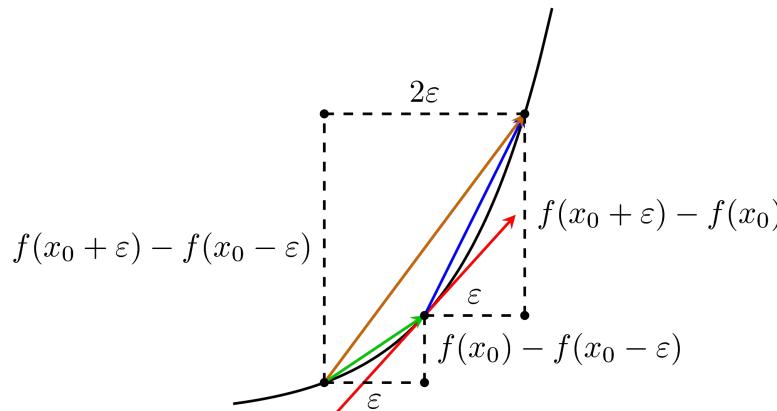


Hình 9: Một ví dụ từ Pinecone

3.1. Các loại dữ liệu

KIỂU BIỂU DIỄN PHỐ BIỀN NHẤT

- Các mô hình học máy dựa trên nền tảng của đại số:



Hình 10: Đạo hàm

a

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

b

$$m_1 = (a_1 + a_4)(b_1 + b_4)$$
$$m_2 = (a_3 + a_4)b_1$$
$$m_3 = a_1(b_2 - b_4)$$
$$m_4 = a_4(b_3 - b_1)$$
$$m_5 = (a_1 + a_2)b_4$$
$$m_6 = (a_3 - a_1)(b_1 + b_2)$$
$$m_7 = (a_2 - a_4)(b_3 + b_4)$$
$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

c

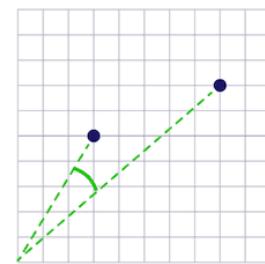
$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$
$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$
$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Hình 11: Nhân ma trận

3.1. Các loại dữ liệu

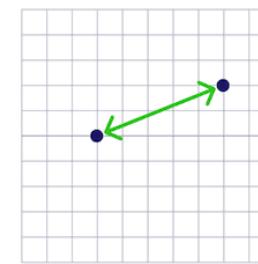
- Có thể đo khoảng cách một cách nhanh chóng và dễ dàng

Distance Metrics in Vector Search



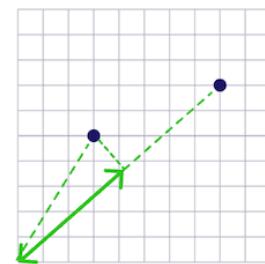
Cosine Distance

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



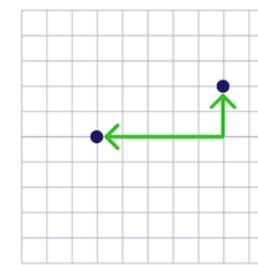
Squared Euclidean
(L2 Squared)

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



Dot Product

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



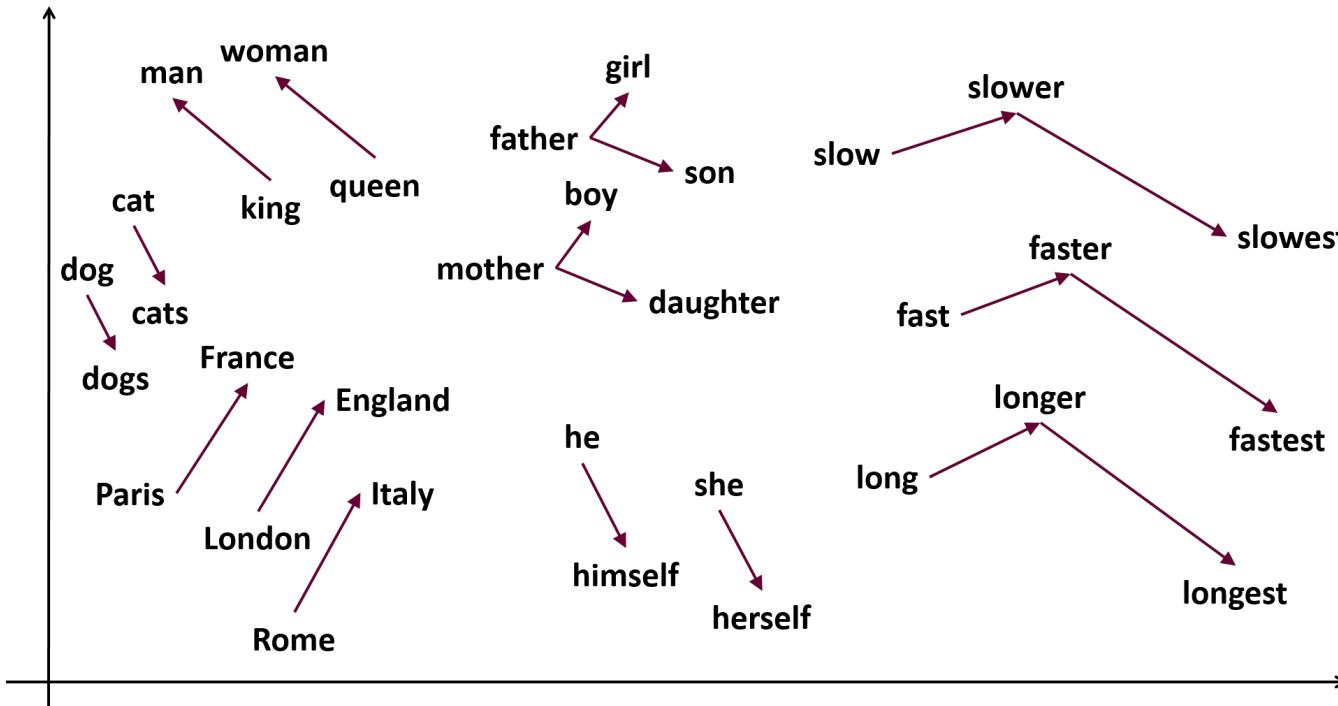
Manhattan (L1)

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

Hình 12: Khoảng cách vector trong không gian

3.1. Các loại dữ liệu

- Dễ cải tiến tốc độ sử dụng GPU
- Tạo ra không gian vector liên tục giúp mô hình có thể tự hiểu ngữ nghĩa



Hình 13: Không gian các từ

3.1. Các loại dữ liệu

3.1.2. Probability and statistic

3.1.3. Hashing

3.1.4. Token sequence

Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

3. Biểu diễn dữ liệu

4. Định nghĩa tương đồng

5. Các phép đo độ tương đồng

6. Các kỹ thuật tìm cặp tương đồng

4.1. Tương đồng là gì?

Trong data mining, “độ tương đồng” được hiểu là độ đo mức độ giống nhau giữa hai đối tượng dữ liệu hay mức độ chồng lặp khi biểu diễn chúng dưới dạng tập hợp hoặc vector đặc trưng.

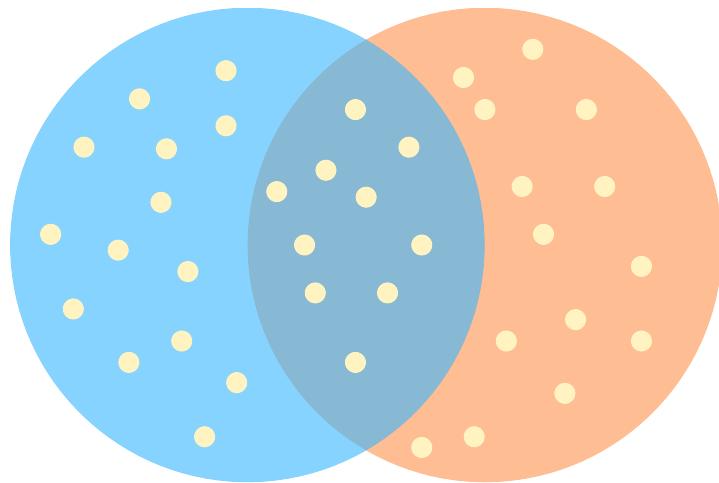
4.2. Các đối tượng như thế nào thì được coi là tương đồng?

Hai đối tượng được coi là tương đồng khi chúng chia sẻ nhiều đặc trưng chung nhau, tức là có mức độ trùng lắp cao giữa các biểu diễn của chúng.

Mục lục

1. Giới thiệu
2. Ứng dụng thực tế
3. Biểu diễn dữ liệu
4. Định nghĩa tương đồng
- 5. Các phép đo độ tương đồng**
6. Các kỹ thuật tìm cặp tương đồng

5.1. Jaccard Similarity



$$J(A, B) = \frac{A \cap B}{A \cup B}$$

Hình 14: Jaccard Similarity

5.2. Phép đo khoảng cách

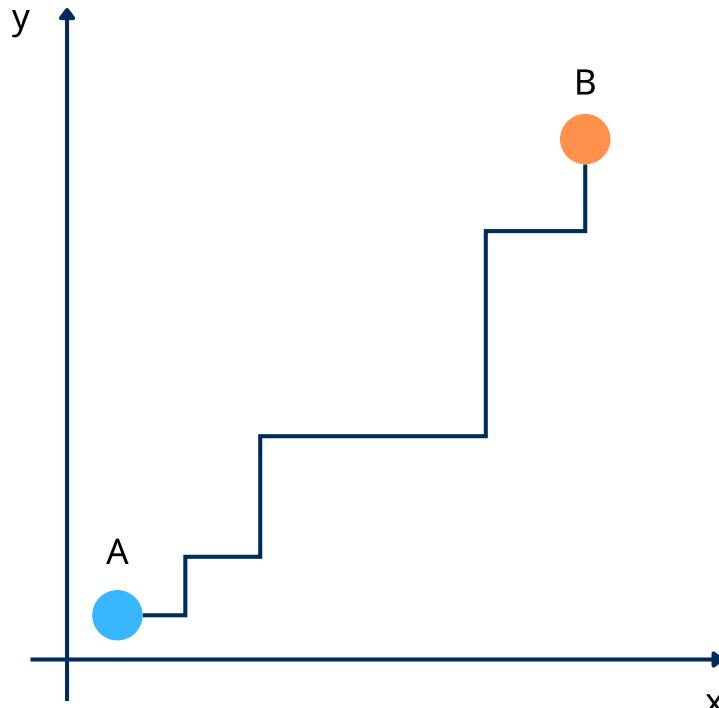
5.2.1. L_p norm

Dạng tổng quát:

$$d_p(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

5.2. Phép đo khoảng cách

Với $p = 1$, ta có khoảng cách Manhattan:

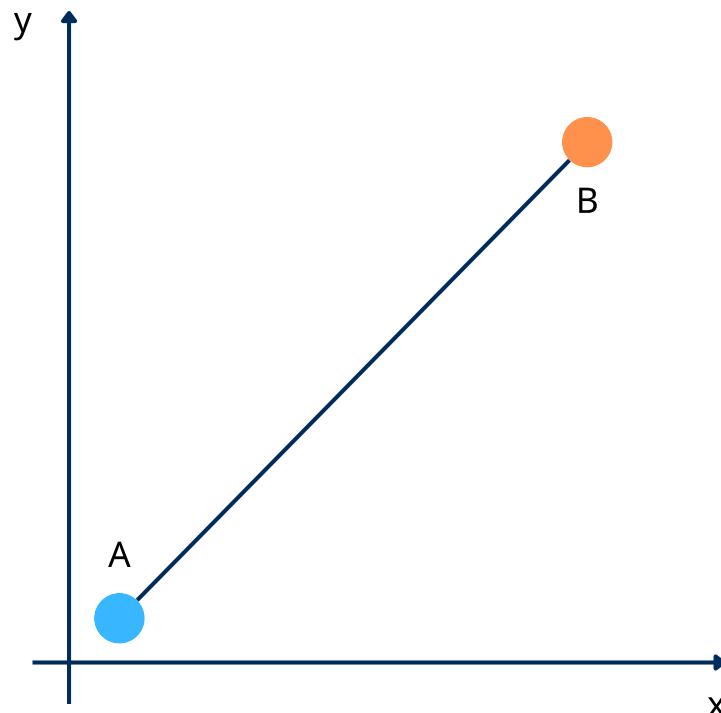


$$d_1(x, y) = \sum_{i=1}^n |x_i - y_i|$$

Hình 15: Manhattan Distance

5.2. Phép đo khoảng cách

Với $p = 2$, ta có khoảng cách Euclidean:



$$d_2(x, y) = \sqrt{\sum_{i=1}^n |x_i - y_i|^2}$$

Hình 16: Euclidean Distance

5.2. Phép đo khoảng cách

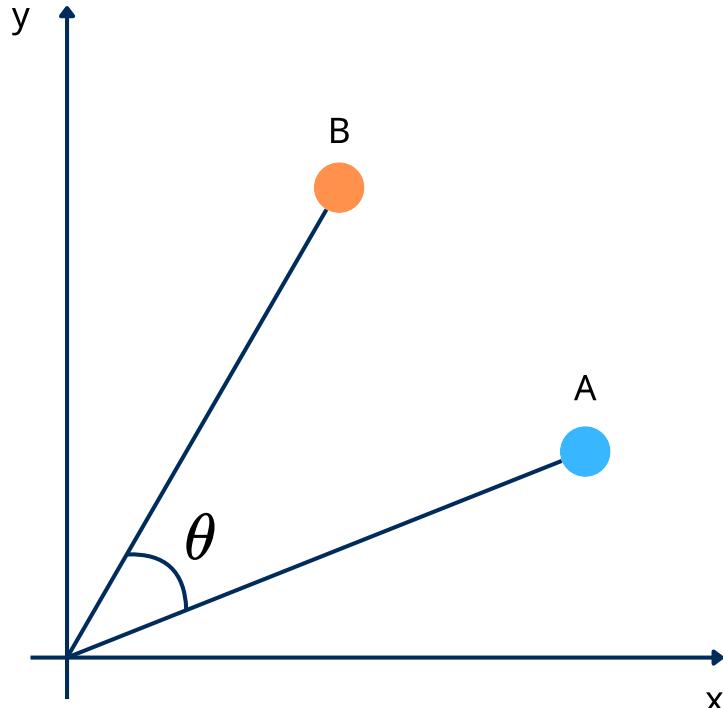
5.2.2. Jaccard Distance

$$d_J(A, B) = 1 - J(A, B)$$

Trong đó $J(A, B)$ là Jaccard Similarity giữa hai tập A và B .

5.2. Phép đo khoảng cách

5.2.3. Cosine Similarity



$$\cos(\theta) = \frac{A \cdot B}{\|A\| * \|B\|}$$

Hình 17: Cosine Similarity

5.2. Phép đo khoảng cách

5.2.4. Edit Distance

5.2.5. Hamming Distance

5.3. So sánh

Mục lục

1. Giới thiệu

2. Ứng dụng thực tế

3. Biểu diễn dữ liệu

4. Định nghĩa tương đồng

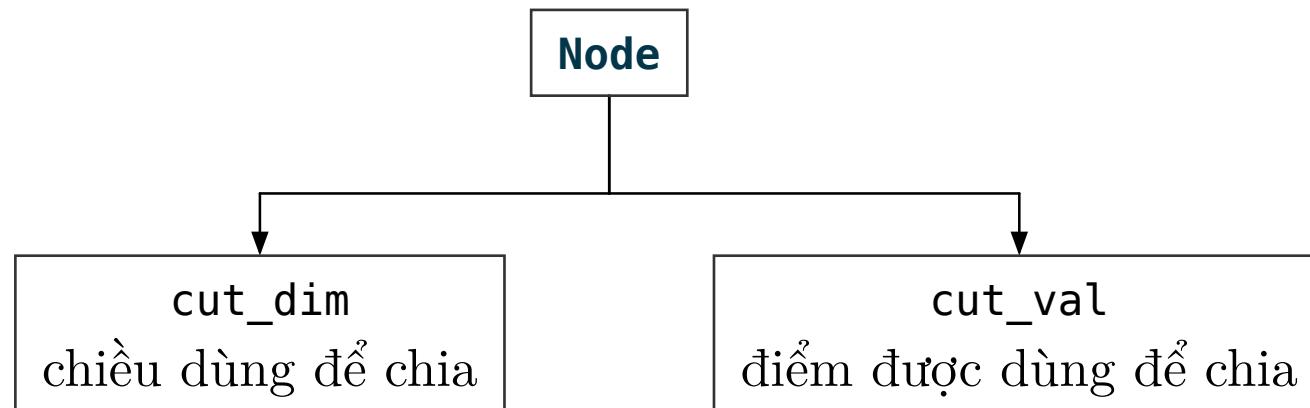
5. Các phép đo độ tương đồng

6. Các kỹ thuật tìm cặp tương đồng

6.1. Tìm chính xác

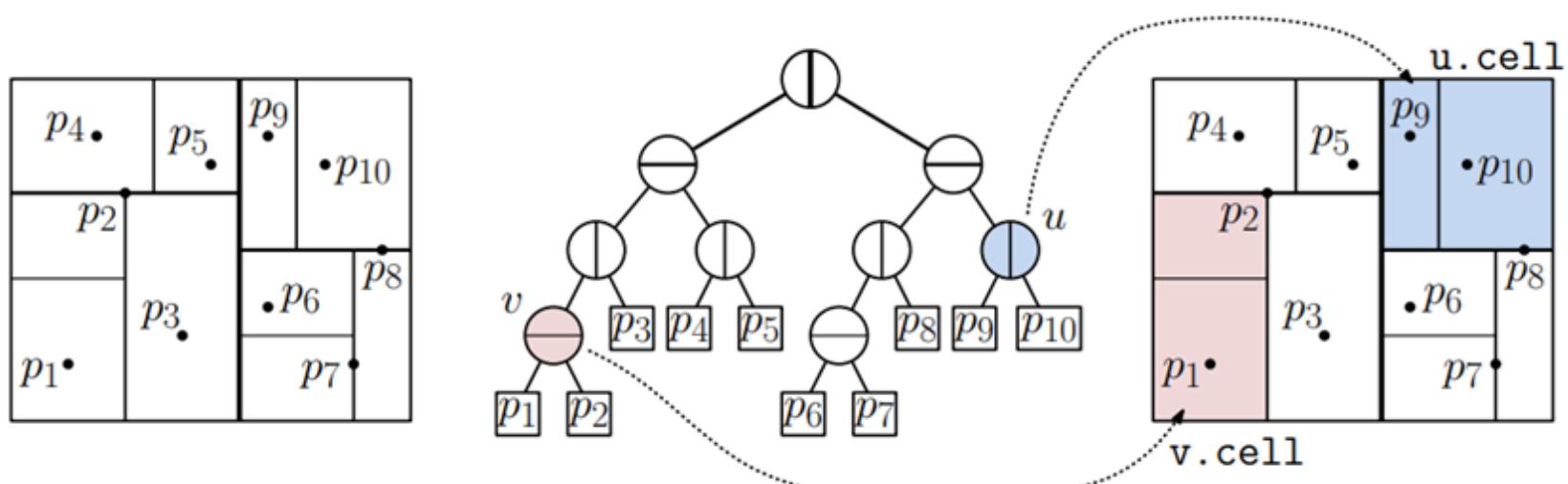
6.1.1. K-d Tree

Cấu trúc cây



Hình 18: Cấu trúc K-d Tree

6.1. Tìm chính xác



Hình 19: Ví dụ cây với một tập điểm 2D

6.1. Tìm chính xác

Xây dựng cây

Thuật toán 1: Xây dựng cây K-d Tree

```
1 function build( $P$ , depth)
2   if  $P$  is empty then
3     return None
4   if  $|P| = 1$  then
5     return Node(depth mod  $k$ ,  $P[0]$ , None, None)
6   mid  $\leftarrow$  midpoint to partition  $P$  into  $P_0$  and  $P_1$  by depth mod  $k$ 
7   left_node  $\leftarrow$  build( $P_0$ , depth + 1)
8   right_node  $\leftarrow$  build( $P_1$ , depth + 1)
9   return Node(depth mod  $k$ , mid, left_node, right_node)
```

Độ phức tạp:

- Giả sử ở mỗi bước đệ quy ta chia đôi tập điểm $\Rightarrow \mathcal{O}(n \log n)$
- Tìm trung vị:
 - Quicksort: $\mathcal{O}(n \log^2 n)$
 - Median of medians: $\mathcal{O}(n \log n)$
 - Duy trì k danh sách sắp xếp theo từng chiều: $\mathcal{O}(nk \log n)$

6.1. Tìm chính xác

Chèn

Thuật toán 2: Chèn điểm vào cây K-d Tree

```
1 function insert( $x$ , node, depth)
2   if node is empty then
3      $\sqsubset$  return Node(depth mod  $k$ ,  $x$ , None, None)
4   if  $x == \text{node.data}$  then
5      $\sqsubset$  return error “duplicate”
6   cd  $\leftarrow$  depth mod  $k$ 
7   if  $x[cd] < \text{node.data}[cd]$  then
8      $\sqsubset$  node.left  $\leftarrow$  insert( $x$ , node.left, depth + 1)
9   else
10     $\sqsubset$  node.right  $\leftarrow$  insert( $x$ , node.right, depth + 1)
11  return node
```

Độ phức tạp:

- Ngẫu nhiên: $\mathcal{O}(\log n)$
- Tối đa: $\mathcal{O}(n)$

6.1. Tìm chính xác

Tìm điểm có chiều d nhỏ nhất

Thuật toán 3: Tìm điểm có chiều d nhỏ nhất

```
1 function findMin(root, d)
2   if root is empty then
3     return None
4   if root.cut_dim == d then
5     if root.left is empty then
6       return root
7     return findMin(root.left, d)
8   else
9     left_min ← findMin(root.left, d)
10    right_min ← findMin(root.right, d)
11    res ← root
12    for candidate in [left_min, right_min] do
13      if candidate != None and candidate.point[d] < res.point[d] then
14        res ← candidate
15    return res
```

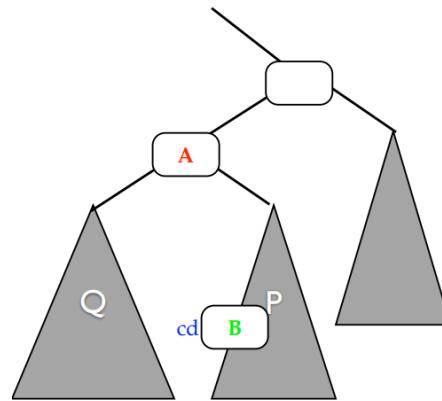
6.1. Tìm chính xác

Xóa

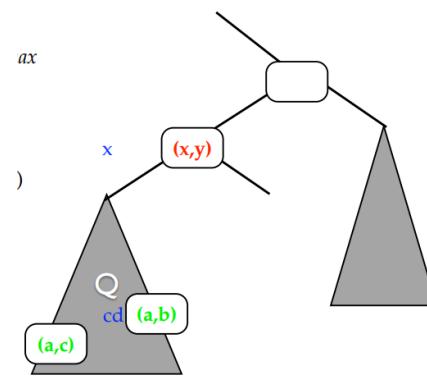
Thuật toán 4: Xóa điểm trong cây K-d Tree

```
1 function delete(root, point, depth)
2   if root is empty then
3     return None
4   if root.point == point then
5     if root.right is not empty then
6       minNode ← findMin(root.right, root.cut_dim)
7       root.point ← minNode.point
8       root.right ← delete(root.right, minNode.point, depth + 1)
9     else if root.left is not empty then
10      minNode ← findMin(root.left, root.cut_dim)
11      root.point ← minNode.point
12      root.right ← delete(root.left, minNode.point, depth + 1)
13      root.left ← None
14    else
15      return None
16    return root
17    axis ← depth mod k
18    if point[axis] < root.point[axis] then
19      root.left ← delete(root.left, point, depth + 1)
20    else
21      root.right ← delete(root.right, point, depth + 1)
22    return root
```

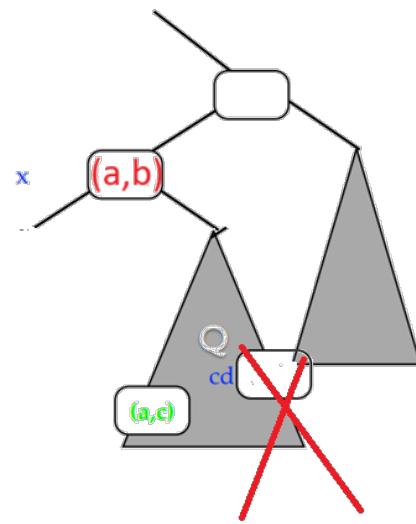
6.1. Tìm chính xác



Hình 20: Tồn tại con bên phải



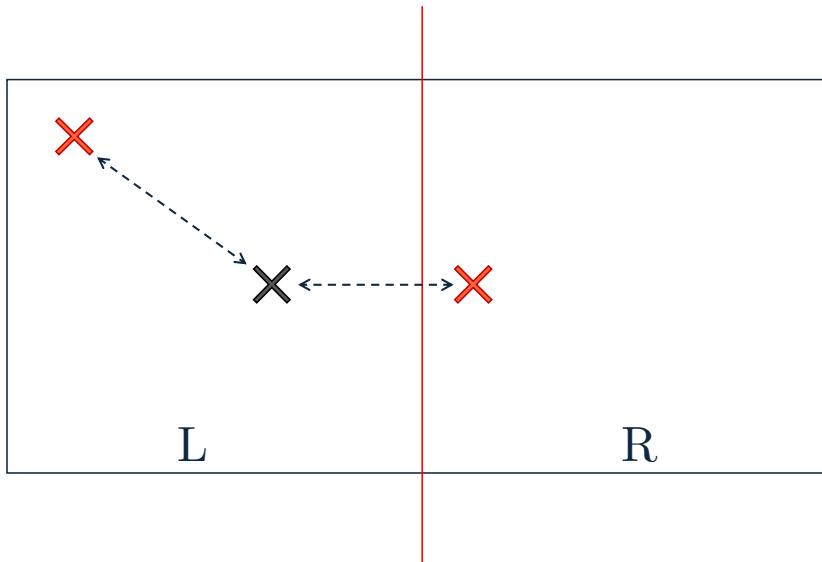
,
Hình 21: Không tồn tại
con bên phải



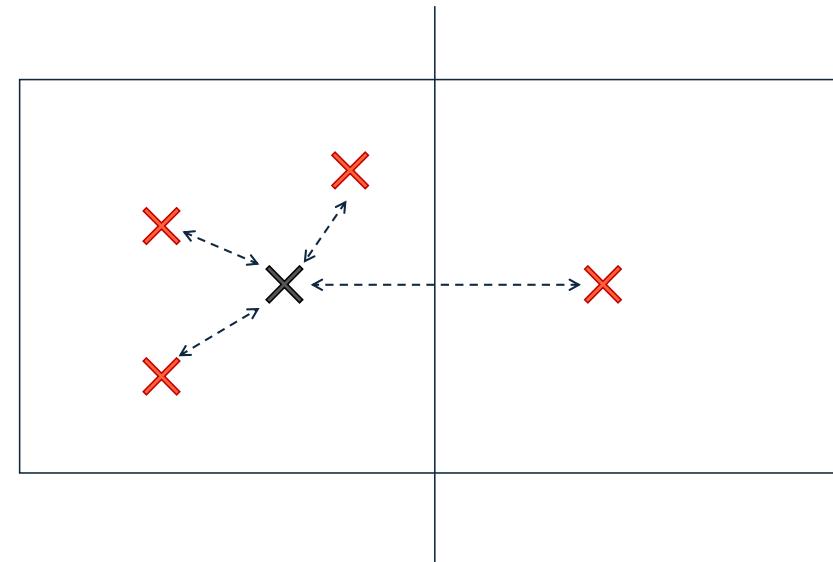
Hình 22: Không tồn tại
con bên phải

6.1. Tìm chính xác

Tìm k điểm gần nhất.



Hình 23: Cắt tia giúp giảm số lượng điểm duyệt



Hình 24: Cắt tia không làm giảm số lượng điểm cần duyệt

6.1. Tìm chính xác

Thuật toán 5: Tìm k điểm gần nhất

```
1 function kNearestNeighbor(root, depth, point, k)
2   if root is empty then
3     return
4   cut_dim ← depth mod  $d$ 
5   if point[cut_dim] < root.cut_val[cut_dim] then
6     if root.left is not empty then
7       ↘ kNearestNeighbor(root.left, depth + 1, point, k)
8     estimated_dist ← (point[cut_dim] - root.cut_val[cut_dim])2
9     if estimated_dist ≤ pq.maxDist() or len(pq) < k then
10      if root.right is not empty then
11        ↘ kNearestNeighbor(root.right, depth + 1, point, k)
12        pq.add(root.cut_val)
13      else if point[cut_dim] > root.cut_val[cut_dim] then
14        if root.right is not empty then
15          ↘ kNearestNeighbor(root.right, depth + 1, point, k)
16          estimated_dist ← (point[cut_dim] - root.cut_val[cut_dim])2
17          if estimated_dist ≤ pq.maxDist() or len(pq) < k then
18            if root.left is not empty then
19              ↘ kNearestNeighbor(root.left, depth + 1, point, k)
20              pq.add(root.cut_val)
21            else
22              if root.left is not empty then
23                ↘ kNearestNeighbor(root.left, depth + 1, point, k)
24              if root.right is not empty then
25                ↘ kNearestNeighbor(root.right, depth + 1, point, k)
26                pq.add(root.cut_val)
```

6.1. Tìm chính xác

6.1.2. Ball Tree

6.2. Tìm gần đúng

6.2.1. LSH

6.2.2. K-NNG