Hamming distance example

numerical example of **Hamming Distance** used in **kNN for categorical variables**.

🔹 Formula:

$$\text{dist}(x, x') = \sum_{i=1}^{d} (1 - \delta_{x_i = x_i'})$$

where

- d = length of the string (or number of attributes)
- $\delta_{x_i = x_i'} = 1$ if values are equal, otherwise 0.

So, Hamming Distance = **number of mismatches**.

🔹 Example 1 (Strings)

Compare:

- **String 1:** `abcde`
- **String 2:** `abfde`

Step-by-step:

Position	String 1	String 2	Same?
1	a	a	✅ Yes
2	b	b	✅ Yes
3	c	f	❌ No
4	d	d	✅ Yes
5	e	e	✅ Yes

Differences = 1 (only at position 3)

👉 **Hamming Distance = 1**

🔹 Example 2 (Binary Data, often used in kNN)

Compare:

- **x = 10101**
- **x′ = 11110**

Step-by-step:

Position	x	x′	Same?
1	1	1	✅ Yes
2	0	1	❌ No
3	1	1	✅ Yes
4	0	1	❌ No
5	1	0	❌ No

Differences = 3 (positions 2, 4, 5)
👉 **Hamming Distance = 3**

✅ in kNN, when working with **categorical variables**, instead of Euclidean distance we can use **Hamming Distance** to measure how
different two samples are.

a **full kNN classification example using Hamming Distance** step by step with
a small dataset.

🔹 Step 1: Dataset (Categorical Attributes)

We'll classify fruits based on **color** and **shape**.

Fruit	Color	Shape	Label
Sample 1	Red	Round	Apple
Sample 2	Green	Round	Apple
Sample 3	Yellow	Long	Banana
Sample 4	Green	Long	Banana
Sample 5	Red	Long	Banana

🔹 Step 2: New Point to Classify

New fruit:

- **Color = Green**
- **Shape = Round**

We want to predict: **Apple or Banana?**

🔹 Step 3: Compute Hamming Distance

👉 For categorical attributes, Hamming Distance = number of mismatches.

We compare new fruit with each sample:

1. **(Green, Round) vs (Red, Round)**
    - Color: ❌ different (Green ≠ Red)
    - Shape: ✅ same (Round = Round)

    → Distance = 1
2. **(Green, Round) vs (Green, Round)**
    - Color: ✅ same
    - Shape: ✅ same

    → Distance = 0
3. **(Green, Round) vs (Yellow, Long)**
    - Color: ❌ different
    - Shape: ❌ different

    → Distance = 2
4. **(Green, Round) vs (Green, Long)**

- Color: ✅ same
- Shape: ❌ different

  → Distance = 1

5. **(Green, Round) vs (Red, Long)**
   - Color: ❌ different
   - Shape: ❌ different

     → Distance = 2

🔹 Step 4: Pick k Nearest Neighbors

Suppose **k = 3** (3 nearest neighbors).

The **three lowest distances** are:

- Sample 2 → Distance = 0 → **Apple**
- Sample 1 → Distance = 1 → **Apple**
- Sample 4 → Distance = 1 → **Banana**

🔹 Step 5: Majority Voting

Among the 3 nearest neighbors:

- **Apple = 2 votes**
- **Banana = 1 vote**

👉 Predicted label = **Apple** 🍎

✅ This is how **kNN with Hamming Distance** works for **categorical data**.

Great 👍 let's now do a **binary (0/1 encoded dataset example)** with **Hamming Distance inside kNN**.

🔹 Step 1: Encode the Dataset

Suppose we have fruits with 2 categorical features: **Color** and **Shape**.

We one-hot encode them into binary form:

- **Color**:
  - Red = (1,0,0)
  - Green = (0,1,0)
  - Yellow = (0,0,1)
- **Shape**:
  - Round = (1,0)
  - Long = (0,1)

So each fruit is represented as a **5-bit vector**.

Fruit	Color	Shape	Encoded Vector	Label
Sample 1	Red	Round	(1,0,0, 1,0)	Apple
Sample 2	Green	Round	(0,1,0, 1,0)	Apple
Sample 3	Yellow	Long	(0,0,1, 0,1)	Banana
Sample 4	Green	Long	(0,1,0, 0,1)	Banana
Sample 5	Red	Long	(1,0,0, 0,1)	Banana

🔹 Step 2: New Fruit to Classify

New fruit: **(Green, Round)**

Encoded vector = **(0,1,0, 1,0)**

🔹 Step 3: Compute Hamming Distance

Hamming distance = count of differing bits.

1. Compare with Sample 1 (1,0,0, 1,0):
   - (0,1,0,1,0) vs (1,0,0,1,0) → differences = 2

     → Distance = 2
2. Compare with Sample 2 (0,1,0, 1,0):

- Exactly the same → differences = 0
  → Distance = 0
3. Compare with Sample 3 (0,0,1, 0,1):
   - Differences = 4
     → Distance = 4
4. Compare with Sample 4 (0,1,0, 0,1):
   - Differences = 2
     → Distance = 2
5. Compare with Sample 5 (1,0,0, 0,1):
   - Differences = 4
     → Distance = 4

🔹 Step 4: Pick k Nearest Neighbors

For **k = 3**, nearest neighbors are:

- Sample 2 → Distance 0 → Apple
- Sample 1 → Distance 2 → Apple
- Sample 4 → Distance 2 → Banana

🔹 Step 5: Majority Voting

Votes:

- **Apple = 2**
- **Banana = 1**

👉 Predicted label = **Apple** 🍎

✅ This shows how **Hamming Distance works directly on binary-encoded categorical features** in kNN.

Would you like me to also create a **step-by-step numeric table (bit-by-bit comparison like your slide)** for one of these distances so it looks visually similar to the diagram you uploaded?

📄

🔹 Problem with Euclidean Distance in Text

When documents are represented as **term-frequency vectors** (word counts), Euclidean distance has issues:

- **Longer documents** (with more words) appear farther from the origin.
- **Shorter documents** appear closer to the origin.
- So even unrelated short documents may seem "closer" than long but similar ones.

👉 That's why we use **Cosine Distance** (or **Cosine Similarity**) instead.

🔹 Cosine Similarity Formula

Given two vectors x and x':

$$\cos\theta = \frac{x \cdot x'}{\|x\|\|x'\|}$$

- $x \cdot x'$ = dot product = sum of element-wise multiplication.
- $\|x\|$ = length (magnitude) of vector.
- Value of $\cos\theta$ ranges between **0 and 1** (for non-negative vectors).
  - 1 → documents are identical in direction (same content pattern).
  - 0 → documents are orthogonal (completely different).

Cosine **Distance** = $1 - \cos\theta$.

🔹 Example with Documents

Suppose we have a vocabulary of 3 words: **tea, time, two**.

We represent each document as a vector of word counts.

Document	**Vector (tea, time, two)**
Doc1	(2, 0, 0)
Doc2	(2, 2, 0)
Doc3	(0, 0, 2)
Doc4	(3, 0, 7)

Step 1: Compare Doc1 vs Doc2

- Doc1 = (2,0,0)
- Doc2 = (2,2,0)

Dot Product:

$$(2)(2) + (0)(2) + (0)(0) = 4$$

Magnitudes:

$$\|Doc1\| = \sqrt{2^2 + 0^2 + 0^2} = 2$$

$$\|Doc2\| = \sqrt{2^2 + 2^2 + 0^2} = \sqrt{8} = 2.828$$

Cosine Similarity:

$$\cos\theta = \frac{4}{2 \times 2.828} = \frac{4}{5.656} \approx 0.707$$

👉 They are **fairly similar**.

Step 2: Compare Doc1 vs Doc3

- Doc1 = (2,0,0)
- Doc3 = (0,0,2)

Dot Product:

$$(2)(0) + (0)(0) + (0)(2) = 0$$

$$\cos\theta = \frac{0}{(2)(2)} = 0$$

👉 They are **completely different**.

Step 3: Compare Doc2 vs Doc4

- Doc2 = (2,2,0)
- Doc4 = (3,0,7)

Dot Product:

$$(2)(3) + (2)(0) + (0)(7) = 6$$

Magnitudes:

$$\|Doc2\| = 2.828, \quad \|Doc4\| = \sqrt{3^2 + 0^2 + 7^2} = \sqrt{58} \approx 7.616$$

Cosine Similarity:

$$\cos\theta = \frac{6}{(2.828)(7.616)} = \frac{6}{21.52} \approx 0.279$$

👉 They are **not very similar**.

🔹 Interpretation

- **Cosine Similarity focuses on angle (direction), not length.**
  - So, long documents and short documents with the same word distribution look similar.
- **Euclidean distance** would incorrectly judge long documents as "farther away."
- That’s why **Cosine Distance is better for text similarity in kNN**.

Nice — here's a **complete, numeric kNN example for document similarity** using **Cosine Similarity / Cosine Distance**, shown step-by-step so you can use it directly in class or a worksheet.

Problem setup — vectors (term counts)

Vocabulary (3 terms): **[tea, time, two]**

Training documents (term-frequency vectors) and labels:

- **Doc1** = (2, 0, 0) → **Label = Tea**
- **Doc2** = (2, 2, 0) → **Label = Tea**
- **Doc3** = (0, 0, 3) → **Label = Two**
- **Doc4** = (0, 1, 1) → **Label = Two**

New (query) document to classify:

- **NewDoc** = (1, 1, 0) (contains "tea" and "time")

We will use **cosine similarity**:

$$\cos\theta = \frac{x \cdot y}{\|x\|\,\|y\|} \qquad \text{and} \qquad \text{cosine distance} = 1 - \cos\theta$$

Step 1 — magnitudes (norms)

Compute $\|v\| = \sqrt{\sum v_i^2}$.

1. $\|\text{New}\| = \sqrt{1^2 + 1^2 + 0^2} = \sqrt{1 + 1 + 0} = \sqrt{2} = 1.4142135624$
2. $\|\text{Doc1}\| = \sqrt{2^2 + 0^2 + 0^2} = \sqrt{4} = 2.0000000000$
3. $\|\text{Doc2}\| = \sqrt{2^2 + 2^2 + 0^2} = \sqrt{4 + 4} = \sqrt{8} = 2.8284271247$
4. $\|\text{Doc3}\| = \sqrt{0^2 + 0^2 + 3^2} = \sqrt{9} = 3.0000000000$
5. $\|\text{Doc4}\| = \sqrt{0^2 + 1^2 + 1^2} = \sqrt{1 + 1} = \sqrt{2} = 1.4142135624$

(kept 10 decimal places for clarity)

Step 2 — dot products $x \cdot y$ (element-wise multiply and sum)

- New · Doc1 = $(1)(2) + (1)(0) + (0)(0) = 2 + 0 + 0 = 2$

- New · Doc2 = $(1)(2) + (1)(2) + (0)(0) = 2 + 2 + 0 = 4$
- New · Doc3 = $(1)(0) + (1)(0) + (0)(3) = 0 + 0 + 0 = 0$
- New · Doc4 = $(1)(0) + (1)(1) + (0)(1) = 0 + 1 + 0 = 1$

Step 3 — cosine similarities (compute numerator / denominator)

We compute denominator = $\|\text{New}\| \times \|\text{Doci}\|$ and then divide.

1. **New vs Doc1**
- numerator = 2
- denominator = $1.4142135624 \times 2.0000000000 = 2.8284271248$
- $\cos\theta = 2 \div 2.8284271248 = 0.7071067812$
2. **New vs Doc2**
- numerator = 4
- denominator = $1.4142135624 \times 2.8284271247 = 4.0000000000$ (exact: $\sqrt{2} \cdot 2\sqrt{2} = 4$)
- $\cos\theta = 4 \div 4.0000000000 = 1.0000000000$
3. **New vs Doc3**
- numerator = 0
- denominator = $1.4142135624 \times 3.0000000000 = 4.2426406872$
- $\cos\theta = 0 \div 4.2426406872 = 0.0000000000$
4. **New vs Doc4**
- numerator = 1
- denominator = $1.4142135624 \times 1.4142135624 = 2.0000000000$ (since both are $\sqrt{2}$)
- $\cos\theta = 1 \div 2.0000000000 = 0.5000000000$

Cosine similarities summary (rounded)

- New–Doc2 = **1.0000**
- New–Doc1 = **0.7071**
- New–Doc4 = **0.5000**
- New–Doc3 = **0.0000**

Step 4 — cosine distances

$$\text{cosine distance} = 1 - \cos\theta$$

- New–Doc2: $1 - 1.0000 = 0.0000$

- New–Doc1: $1 - 0.7071067812 = 0.2928932188$
- New–Doc4: $1 - 0.5000 = 0.5000$
- New–Doc3: $1 - 0.0000 = 1.0000$

Smaller distance ⇒ more similar.

Step 5 — choose k and find nearest neighbors

Choose **k = 3**.

Order by increasing cosine distance:

1. Doc2 — distance 0.0000 — Label **Tea**
2. Doc1 — distance 0.2929 — Label **Tea**
3. Doc4 — distance 0.5000 — Label **Two**

(k = 3 picks Doc2, Doc1, Doc4)

Step 6 — majority voting

Neighbors' labels: **Tea**, **Tea**, **Two** → counts: Tea = 2, Two = 1

Prediction for NewDoc = Tea

Notes, interpretation & teaching points

- **Why cosine, not Euclidean?** Cosine ignores document length and focuses on direction (relative word frequencies). Doc2 (2,2,0) and NewDoc (1,1,0) are in the same direction (scalar multiples), so cosine similarity = 1 even though lengths differ. Euclidean would treat them as different distances.
- **Scaling/TF-IDF:** In real text tasks you usually use TF-IDF vectors (same cosine formula) to downweight common words.
- **Tie-breaking:** If k produces a tie (e.g., k=2 with one Tea and one Two), common tie-breakers: choose nearest among the tied, reduce k, or use weighted voting (weight by 1/distance).
- **Weighted kNN variant:** Instead of simple vote, weight each neighbor by cosine similarity (or 1/distance) for smoother decisions.
