COP509

Natural Language Processing

Document Similarity and Distance Measures

Prof. Georgina Cosma

Department of Computer Science
Loughborough University

g.cosma@lboro.ac.uk

Learning Outcomes

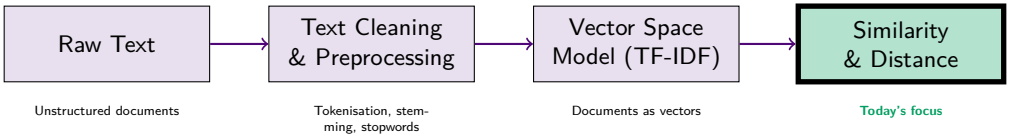
By the end of this session, you will be able to:

1. Explain why we need to measure similarity and distance between documents
2. Understand the relationship between the Vector Space Model and similarity measures
3. Calculate cosine similarity between document vectors
4. Understand and apply different distance metrics (Euclidean, Manhattan, etc.)
5. Choose appropriate similarity/distance measures for specific NLP tasks
6. Apply these concepts in document classification and information retrieval
7. Implement similarity calculations in Python

Outline

Where We Are in the NLP Pipeline

Building towards practical text applications:



- **Previous lectures:** How to represent documents as numerical vectors
- **This lecture:** How to *compare* those vectors to find similar documents
- **The payoff:** Search engines, clustering, plagiarism detection, recommendations

Key Question

We can now represent documents as vectors – but how do we determine which documents are *similar* to each other?

Part 1

Connecting VSM to Similarity

Remember from the VSM lectures: We represent documents as vectors in a high-dimensional space.

Example: Three Shakespeare plays

Document	battle	good	fool	wit
Julius Caesar	0.58	0.39	0	0.39
Henry V	0.47	0	0.36	0.36
As You Like It	0	0.30	0.51	0.51

Each row is a **document vector**:

- Julius Caesar: $(0.58, 0.39, 0, 0.39)$
- Henry V: $(0.47, 0, 0.36, 0.36)$
- As You Like It: $(0, 0.30, 0.51, 0.51)$

The Question: How do we determine which documents are similar?

Why We Need Similarity Measures

Once we have document vectors, what can we do with them?

Information Retrieval:

- User types a query
- Convert query to vector
- Find documents most similar to query
- Return ranked results

Document Clustering:

- Group similar documents
- Organise large collections
- Discover topics automatically

Plagiarism Detection:

- Compare student essays
- Find suspiciously similar texts
- Identify copied content

Recommendation Systems:

- User likes document A
- Find documents similar to A
- Recommend to user

Key Insight

Similarity measures let us **compare** document vectors to answer: "How alike are these two documents?"

Part 2

What is Similarity?
What is Distance?

Similarity vs Distance: The Relationship

Two sides of the same coin – but inverted!

Similarity:

- **Higher values** = more alike
- Usually ranges from 0 to 1
- Example: Cosine similarity
- 1 = identical
- 0 = completely different

Think: How much do they have in common?

Distance:

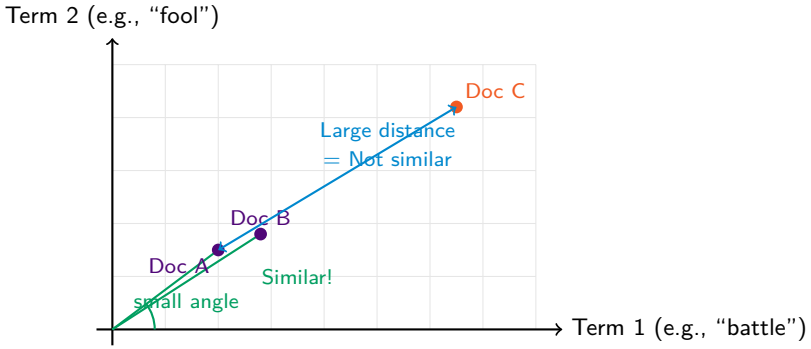
- **Lower values** = more alike
- Ranges from 0 to ∞
- Example: Euclidean distance
- 0 = identical
- Large number = very different

Think: How far apart are they?

Key Point

Similarity and distance are inversely related: high similarity means low distance, and vice versa.

Geometric intuition: Documents are points in space.



- **Angle between vectors** (cosine similarity): Docs A & B have a small angle = similar
- **Distance between points** (Euclidean): Docs A & C are far apart = not similar

Part 3

Cosine Similarity: The Most Important Measure

Why Cosine Similarity is Crucial for Text

For document vectors, cosine similarity is the standard measure. Why?

1. Ignores document length:

- A 100-word article and a 10,000-word book about the same topic should be similar
- Cosine focuses on *direction* (topic), not *magnitude* (length)

2. Handles sparse vectors well:

- Document vectors have many zeros (most words don't appear in each doc)
- Cosine ignores zero-matches (we don't care that both docs lack "xylophone")

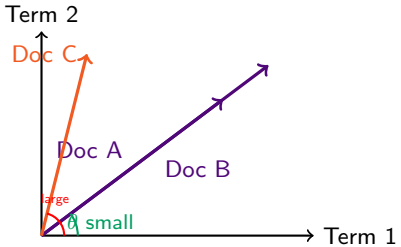
3. Mathematically convenient:

- Range is [0, 1] for positive values (or [-1, 1] generally)
- Computationally efficient with normalised vectors

Key Principle

Cosine similarity measures the **angle** between vectors. A small angle means the vectors point in the same direction = similar content.

Why does the angle matter?



Docs A & B: Small angle = **Similar topics**
Doc A & C: Large angle = **Different topics**

- **Small angle** ($\theta \approx 0°$): $\cos(\theta) \approx 1$ = Very similar
- **Right angle** ($\theta = 90°$): $\cos(\theta) = 0$ = No similarity
- **Opposite directions** ($\theta = 180°$): $\cos(\theta) = -1$ = Opposite (rare in text)

The Cosine Similarity Formula

For two document vectors $\mathbf{d}_1$ and $\mathbf{d}_2$:

$$\text{cosine\_sim}(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1|| \times ||\mathbf{d}_2||}$$

Breaking it down:

- $\mathbf{d}_1 \cdot \mathbf{d}_2$ = **Dot product** (multiply corresponding components and sum)
- $||\mathbf{d}_1||$ = **Length or (magnitude)** of vector $\mathbf{d}_1 = \sqrt{d_{11}^2 + d_{12}^2 + \cdots + d_{1n}^2}$
- $||\mathbf{d}_2||$ = **Length (or magnitude)** of vector $\mathbf{d}_2 = \sqrt{d_{21}^2 + d_{22}^2 + \cdots + d_{2n}^2}$

What it computes:

- The cosine of the angle θ between the two vectors
- Result is between -1 and +1 (but 0 to 1 for term-frequency vectors)
- Closer to 1 = more similar

Understanding the Dot Product

The numerator $\mathbf{d}_1 \cdot \mathbf{d}_2$ is the dot product:

Definition: Multiply each pair of corresponding components and add them up.

Example: If $\mathbf{d}_1 = (5, 0, 3, 0, 2)$ and $\mathbf{d}_2 = (3, 0, 2, 0, 1)$:

$$\begin{aligned}\mathbf{d}_1 \cdot \mathbf{d}_2 &= (5 \times 3) + (0 \times 0) + (3 \times 2) + (0 \times 0) + (2 \times 1) \\ &= 15 + 0 + 6 + 0 + 2 \\ &= 23\end{aligned}$$

Intuition

The dot product is **high** when:

- Both vectors have large values in the *same* positions
- = They share many important terms

The dot product is **low** when:

- Vectors have large values in *different* positions
- = They discuss different topics

Understanding Vector Length (Magnitude)

The denominator $||\mathbf{d}_1|| \times ||\mathbf{d}_2||$ normalises by vector lengths.

Definition: The length (or magnitude) of a vector is:

$$||\mathbf{d}|| = \sqrt{d_1^2 + d_2^2 + \cdots + d_n^2}$$

Note: The terms 'length', 'magnitude', and 'norm' all mean the same thing for vectors and are used interchangeably (e.g., `np.linalg.norm()` in Python).

Example: If $\mathbf{d}_1 = (3, 0, 2, 0, 1)$:

$$\begin{aligned} ||\mathbf{d}_1|| &= \sqrt{3^2 + 0^2 + 2^2 + 0^2 + 1^2} \\ &= \sqrt{9 + 0 + 4 + 0 + 1} \\ &= \sqrt{14} \\ &\approx 3.742 \end{aligned}$$

Intuition

Vector length captures the overall **magnitude** (or "size") of the document. By dividing by lengths, we focus on **direction** (content) rather than size (length).

Normalised Vectors: A Useful Shortcut

If vectors are pre-normalised, cosine similarity becomes just the dot product!

Normalising a vector: Divide each component by the vector's length:

$$\hat{\mathbf{d}} = \frac{\mathbf{d}}{||\mathbf{d}||} = \left(\frac{d_1}{||\mathbf{d}||}, \frac{d_2}{||\mathbf{d}||}, \ldots, \frac{d_n}{||\mathbf{d}||} \right)$$

After normalisation: $||\hat{\mathbf{d}}|| = 1$ (unit vector)

Example: Normalise $\mathbf{d} = (3, 0, 4)$

- $||\mathbf{d}|| = \sqrt{9 + 0 + 16} = \sqrt{25} = 5$
- $\hat{\mathbf{d}} = (3/5, 0/5, 4/5) = (0.6, 0, 0.8)$
- Check: $||\hat{\mathbf{d}}|| = \sqrt{0.36 + 0 + 0.64} = \sqrt{1} = 1$ ✓

Why This Matters

If both vectors are normalised: $\text{cosine\_sim}(\hat{\mathbf{d}}_1, \hat{\mathbf{d}}_2) = \hat{\mathbf{d}}_1 \cdot \hat{\mathbf{d}}_2$
This saves computation! In practice, documents are often stored as normalised vectors.

Part 4

Working Through Complete Examples

Example 1: Simple Term-Frequency Vectors

Scenario: We have two sports documents represented by term counts:

Document	team	coach	hockey	baseball	soccer
Document 1	5	0	3	0	2
Document 2	3	0	2	0	1

Vectors:

- $\mathbf{d}_1 = (5, 0, 3, 0, 2)$
- $\mathbf{d}_2 = (3, 0, 2, 0, 1)$

Question: How similar are these two documents?

Expectation: They should be quite similar – both about team sports (hockey/soccer), no baseball, no coach mentioned.

Step 1: Calculate $\mathbf{d}_1 \cdot \mathbf{d}_2$

$$\begin{aligned}\mathbf{d}_1 \cdot \mathbf{d}_2 &= (5 \times 3) + (0 \times 0) + (3 \times 2) + (0 \times 0) + (2 \times 1) \\ &= 15 + 0 + 6 + 0 + 2 \\ &= \mathbf{23}\end{aligned}$$

Interpretation

The dot product of 23 is **high** because both documents have large counts for "team", "hockey", and "soccer" in the same positions.

Example 1: Step 2 – Compute Vector Lengths

Step 2a: Calculate $||\mathbf{d}_1||$

$$\begin{aligned} ||\mathbf{d}_1|| &= \sqrt{5^2 + 0^2 + 3^2 + 0^2 + 2^2} \\ &= \sqrt{25 + 0 + 9 + 0 + 4} \\ &= \sqrt{38} \\ &\approx \mathbf{6.164} \end{aligned}$$

Step 2b: Calculate $||\mathbf{d}_2||$

$$\begin{aligned} ||\mathbf{d}_2|| &= \sqrt{3^2 + 0^2 + 2^2 + 0^2 + 1^2} \\ &= \sqrt{9 + 0 + 4 + 0 + 1} \\ &= \sqrt{14} \\ &\approx \mathbf{3.742} \end{aligned}$$

Step 3: Apply the formula

$$\begin{aligned}
\text{cosine_sim}(\mathbf{d}_1, \mathbf{d}_2) &= \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1|| \times ||\mathbf{d}_2||} \\
&= \frac{23}{6.164 \times 3.742} \\
&= \frac{23}{23.066} \\
&\approx \mathbf{0.997}
\end{aligned}$$

Result

Cosine similarity ≈ 0.997 which is very close to 1!

This confirms our intuition: these documents are **extremely similar** – they discuss the same sports topics with the same emphasis.

Example 2: Shakespeare Plays

Let's use the Shakespeare TF-IDF vectors from the VSM lectures:

Document	battle	good	fool	wit
Julius Caesar	0.58	0.39	0	0.39
Henry V	0.47	0	0.36	0.36
As You Like It	0	0.30	0.51	0.51

Question: Which two plays are most similar?

Let's compare:

- Julius Caesar (JC) vs Henry V (H5)
- Julius Caesar (JC) vs As You Like It (AYLI)
- Henry V (H5) vs As You Like It (AYLI)

Our intuition:

- JC and H5 are both histories/tragedies with battles → should be similar
- AYLI is a comedy → should be different from the others

Example 2: Comparing Julius Caesar and Henry V

Vectors:

- JC: $(0.58, 0.39, 0, 0.39)$
- H5: $(0.47, 0, 0.36, 0.36)$

Step 1: Dot product

$$\begin{aligned} \text{JC} \cdot \text{H5} &= (0.58 \times 0.47) + (0.39 \times 0) + (0 \times 0.36) + (0.39 \times 0.36) \\ &= 0.273 + 0 + 0 + 0.140 \\ &= \mathbf{0.413} \end{aligned}$$

Step 2: Vector lengths

$$||\text{JC}|| = \sqrt{0.58^2 + 0.39^2 + 0^2 + 0.39^2} = \sqrt{0.64} \approx \mathbf{0.801}$$

$$||\text{H5}|| = \sqrt{0.47^2 + 0^2 + 0.36^2 + 0.36^2} = \sqrt{0.480} \approx \mathbf{0.693}$$

Step 3: Cosine similarity

$$\text{cosine\_sim}(\text{JC}, \text{H5}) = \frac{0.413}{0.800 \times 0.693} \approx \frac{0.413}{0.555} \approx \mathbf{0.744}$$

Vectors:

- JC: $(0.58, 0.39, 0, 0.39)$
- AYLI: $(0, 0.30, 0.51, 0.51)$

Step 1: Dot product

$$\begin{aligned} \text{JC} \cdot \text{AYLI} &= (0.58 \times 0) + (0.39 \times 0.30) + (0 \times 0.51) + (0.39 \times 0.51) \\ &= 0 + 0.117 + 0 + 0.199 \\ &= \mathbf{0.316} \end{aligned}$$

Step 2: Vector lengths

$$||\text{JC}|| \approx 0.801 \text{ (computed earlier)}$$

$$||\text{AYLI}|| = \sqrt{0^2 + 0.30^2 + 0.51^2 + 0.51^2} = \sqrt{0.610} \approx \mathbf{0.781}$$

Step 3: Cosine similarity

$$\text{cosine\_sim}(\text{JC}, \text{AYLI}) = \frac{0.316}{0.801 \times 0.781} \approx \frac{0.316}{0.625} \approx \mathbf{0.506}$$

Vectors:

- H5: $(0.47, 0, 0.36, 0.36)$
- AYLI: $(0, 0.30, 0.51, 0.51)$

Step 1: Dot product

$$\begin{aligned} \text{H5} \cdot \text{AYLI} &= (0.47 \times 0) + (0 \times 0.30) + (0.36 \times 0.51) + (0.36 \times 0.51) \\ &= 0 + 0 + 0.184 + 0.184 \\ &= \mathbf{0.368} \end{aligned}$$

Step 2: Vector lengths

$$||\text{H5}|| \approx 0.693 \text{ (computed earlier)}$$
$$||\text{AYLI}|| \approx 0.781 \text{ (computed earlier)}$$

Step 3: Cosine similarity

$$\text{cosine\_sim}(\text{H5}, \text{AYLI}) = \frac{0.368}{0.693 \times 0.781} \approx \frac{0.368}{0.541} \approx \mathbf{0.680}$$

Example 2: Summary and Interpretation

Results:

Comparison	Cosine Similarity
Julius Caesar vs Henry V	**0.744**
Julius Caesar vs As You Like It	0.506
Henry V vs As You Like It	0.680

Interpretation:

- **Highest similarity (0.744)**: JC and H5 – both histories with battles
- **Lowest similarity (0.506)**: JC and AYLI – tragedy vs comedy
- **Middle similarity (0.680)**: H5 and AYLI – share "wit" and "fool" terms

Key Takeaway

The cosine similarity correctly captures our intuition: the two historical/battle plays are most similar, while the comedy is least similar to the tragedy.

Part 5

Other Distance Measures

Why Other Distance Measures?

Cosine similarity is the standard for text, but other measures exist:

- **Euclidean distance**: Straight-line distance (like a ruler)
- **Manhattan distance**: Grid-based distance (like city blocks)
- **Hamming distance**: Count of differences (for binary/categorical data)
- **Jaccard distance**: Set overlap (for presence/absence)

When to Use Each?

- **Text/Documents**: Cosine similarity (always!)
- **Continuous features**: Euclidean or Manhattan
- **Binary features**: Jaccard or Hamming
- **Categorical data**: Hamming

In this course: We focus on cosine for text, but you should know the others exist for different data types.

The familiar "straight-line" distance from geometry:

$$d_{\text{Euclidean}}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Example: $\mathbf{x} = (3, 0, 2)$ and $\mathbf{y} = (1, 0, 1)$

$$\begin{aligned} d_{\text{Euclidean}}(\mathbf{x}, \mathbf{y}) &= \sqrt{(3-1)^2 + (0-0)^2 + (2-1)^2} \\ &= \sqrt{4 + 0 + 1} \\ &= \sqrt{5} \\ &\approx 2.236 \end{aligned}$$

Interpretation: The straight-line distance is ≈ 2.236 units.

Why Euclidean Distance Doesn't Work Well for Text

Problem: Euclidean distance is sensitive to magnitude, not direction.

Example: Compare these two documents:

- Short article: $(5, 0, 3, 0, 2)$ (total word count: 10)
- Long book: $(50, 0, 30, 0, 20)$ (total word count: 100)

These have *identical proportions* – the book is just $10\times$ longer!

Euclidean distance:

$$\begin{aligned} d &= \sqrt{(50-5)^2 + (0-0)^2 + (30-3)^2 + (0-0)^2 + (20-2)^2} \\ &= \sqrt{45^2 + 27^2 + 18^2} = \sqrt{3078} \approx 55.48 \end{aligned}$$

Cosine similarity: Would be 1.0 (perfect similarity!)

Conclusion

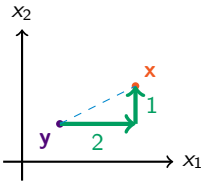
Euclidean distance incorrectly treats these as very different, while cosine correctly recognises they discuss the same topics.

"City block" distance – sum of absolute differences:

$$d_{\text{Manhattan}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

Example: $\mathbf{x} = (3, 0, 2)$ and $\mathbf{y} = (1, 0, 1)$

$$\begin{aligned} d_{\text{Manhattan}}(\mathbf{x}, \mathbf{y}) &= |3 - 1| + |0 - 0| + |2 - 1| \\ &= 2 + 0 + 1 \\ &= 3 \end{aligned}$$



Manhattan (green): $2 + 1 = 3$ **Euclidean (blue):** ≈ 2.236

Hamming Distance

Counts the number of positions where two sequences differ:
Used for: Binary strings, categorical data, spell-checking

Example 1: Binary strings

- String A: 1011101
- String B: 1001001
- Differences at positions 3 and 5:
- String A: 1011101
- String B: 1001001
- Hamming distance = 2

Example 2: Text strings

- Word A: `karolin`
- Word B: `kathrin`
- Differences at positions 3, 4, 5:
- Word A: karolin
- Word B: kathrin
- Hamming distance = 3

Note: Only works for sequences of *equal length*!

Jaccard Similarity/Distance

Measures overlap between two sets – ignores counts!

Formula:

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{size of intersection}}{\text{size of union}}$$

Example: Document similarity based on unique words

- Doc A contains words: {cat, dog, mouse, bird}
- Doc B contains words: {cat, dog, fish, bird, snake}
- Intersection $A \cap B$: {cat, dog, bird} $\rightarrow$ size $= 3$
- Union $A \cup B$: {cat, dog, mouse, bird, fish, snake} $\rightarrow$ size $= 6$
- Jaccard similarity $= \frac{3}{6} = 0.5$

Jaccard distance $= 1 -$ Jaccard similarity $= 1 - 0.5 = 0.5$

When to Use

Useful when you only care about *presence/absence* of features, not frequency. Example: comparing sets of tags, categories, or binary features.

Quick Comparison of Distance Measures

Measure	Best For	Range	Key Property
Cosine Similarity	**Text/Documents**	**[0, 1]**	**Ignores magnitude, focuses on direction**
Euclidean Distance	Continuous numerical features	$[0, \infty)$	Straight-line distance; sensitive to magnitude
Manhattan Distance	Grid-based data, outliers present	$[0, \infty)$	Sum of absolute differences; sensitive to magnitude
Hamming Distance	Binary/categorical sequences	$[0, n]$	Counts position differences; requires equal length
Jaccard Similarity	Sets, presence/absence	$[0, 1]$	Measures overlap ratio; ignores frequency

Why Cosine is Preferred for Text

Euclidean and Manhattan distances are **unbounded** and **magnitude-sensitive** – they require manual normalisation to compare documents of different lengths. Cosine similarity **automatically normalises** by dividing by vector magnitudes, making it ideal for text where document length varies.

Part 6

Practical Applications in NLP

How search engines use cosine similarity:

The Process:

1. User types a query: "machine learning algorithms"
2. Convert query to a vector using same vocabulary as documents
3. Compute cosine similarity between query vector and all document vectors
4. Rank documents by similarity score
5. Return top k most similar documents

Example:

- Query about: machine learning algorithms
- Doc 1: Article on ML techniques $\rightarrow$ high similarity (shares key terms)
- Doc 2: Article on quantum physics $\rightarrow$ zero similarity (no shared terms)
- Doc 3: Brief note on algorithms $\rightarrow$ moderate similarity (partial overlap)

Ranking: Doc 1, Doc 3, Doc 2 (by decreasing similarity)

Detailed Search Example: Step-by-Step

Scenario: User searches for "python programming tutorial"

Step 1: Build vocabulary from documents

	python	programming	tutorial	snake
Query	0.58	0.58	0.58	0
Doc 1: "Python Programming Guide"	0.7	0.7	0	0
Doc 2: "Python Tutorial for Beginners"	0.6	0	0.8	0
Doc 3: "Snakes of Australia"	0	0	0	0.9

Step 2: Compute similarities

- Query $\cdot$ Doc 1: $(0.58 \times 0.7) + (0.58 \times 0.7) + 0 + 0 = 0.812$
- $||\text{Query}|| \times ||\text{Doc 1}|| = 1.0 \times 0.99 = 0.99$
- Similarity $= 0.812/0.99 =$ **0.82**

Step 3: Rank (both Doc 1 and Doc 2 match two query terms)

1. Doc 1 (0.82) – matches "python" and "programming"
2. Doc 2 (0.81) – matches "python" and "tutorial"
3. Doc 3 (0.0) – completely irrelevant

Application 2: Document Clustering

Grouping similar documents together:

Use case: Organising a large collection of news articles by topic

Algorithm (K-means with cosine similarity):

1. Represent each article as a TF-IDF vector
2. Choose k initial cluster centres
3. **Repeat:**
   - Assign each document to the cluster with highest cosine similarity
   - Update cluster centres (average of member vectors)
4. Stop when clusters stabilise

Result: Documents naturally group by topic:

- Cluster 1: Sports articles (high similarity amongst themselves)
- Cluster 2: Politics articles
- Cluster 3: Technology articles

Application 3: Plagiarism Detection

Finding suspiciously similar student essays:

Approach:

1. Convert each essay to a TF-IDF vector
2. Compute pairwise cosine similarity for all essay pairs
3. Flag pairs with similarity above threshold (e.g., 0.85)
4. Manually review flagged pairs

Why it works:

- Copied essays will have very similar word frequencies
- Even with some paraphrasing, core vocabulary remains similar
- Cosine similarity captures this lexical overlap

Limitations:

- Doesn't catch sophisticated paraphrasing
- Doesn't detect idea theft if words are completely changed
- But good as a first-pass screening tool

"If you liked this article, you might also like..."

Content-based filtering with cosine similarity:

1. User reads/likes Article A
2. Represent Article A as a TF-IDF vector
3. Compute cosine similarity between Article A and all other articles
4. Recommend top k most similar articles

Example:

- User likes: "Introduction to Neural Networks"
- Most similar articles:
  1. "Deep Learning Basics" (similarity = 0.89)
  2. "Backpropagation Explained" (similarity = 0.82)
  3. "Machine Learning Overview" (similarity = 0.78)
- Recommend these to the user

Advantage: Works even for new articles (no need for user ratings data)

k-Nearest Neighbours (k-NN) with cosine similarity:

Problem: Classify a new document into a category (e.g., Sports, Politics, Technology)

Algorithm:

1. Have a training set of labelled documents
2. New document arrives (unlabelled)
3. Compute cosine similarity between new document and all training documents
4. Find the k most similar training documents
5. Assign the new document to the majority class among those k neighbours

Example ($k = 3$):

- New document: "The football team won the championship"
- 3 most similar training docs:
  - Sports article (similarity = 0.91)
  - Sports article (similarity = 0.87)
  - Politics article (similarity = 0.65)
- Classification: **Sports** (2 out of 3 neighbours)

Part 7

Implementation in Python

Method 1: Write your own function

```
import numpy as np

def cosine_similarity(vec1, vec2):
    """
    Compute cosine similarity between two vectors.
    """
    # Compute dot product
    dot_product = np.dot(vec1, vec2)

    # Compute magnitudes
    magnitude1 = np.linalg.norm(vec1)
    magnitude2 = np.linalg.norm(vec2)

    # Avoid division by zero
    if magnitude1 == 0 or magnitude2 == 0:
        return 0.0

    # Return cosine similarity
    return dot_product / (magnitude1 * magnitude2)

# Example usage
doc1 = np.array([5, 0, 3, 0, 2])
doc2 = np.array([3, 0, 2, 0, 1])

similarity = cosine_similarity(doc1, doc2)
print(f"Cosine similarity: {similarity:.4f}")
# Output: Cosine similarity: 0.9971
```

Method 2: Use scikit-learn's built-in function

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Single pair of documents
doc1 = np.array([[5, 0, 3, 0, 2]])  # Note: 2D array (1 row)
doc2 = np.array([[3, 0, 2, 0, 1]])

similarity = cosine_similarity(doc1, doc2)
print(f"Cosine similarity: {similarity[0, 0]:.4f}")
# Output: Cosine similarity: 0.9971

# Multiple documents (pairwise similarities)
documents = np.array([
    [5, 0, 3, 0, 2],  # Doc 1
    [3, 0, 2, 0, 1],  # Doc 2
    [0, 7, 0, 2, 1]   # Doc 3
])

# Compute all pairwise similarities
similarity_matrix = cosine_similarity(documents)
print("Similarity matrix:")
print(similarity_matrix)
# Output:
# [[1.     0.9971 0.1862]
#  [0.9971 1.     0.2236]
#  [0.1862 0.2236 1.    ]]
```

Complete Example: Document Similarity Pipeline

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Sample documents
documents = [
    "The cat sat on the mat",
    "The dog sat on the log",
    "Cats and dogs are enemies",
    "Python is a programming language"
]

# Step 1: Convert documents to TF-IDF vectors
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

# Step 2: Compute pairwise cosine similarities
similarity_matrix = cosine_similarity(tfidf_matrix)

# Step 3: Display results
print("Documents:")
for i, doc in enumerate(documents):
    print(f"{i}: {doc}")

print("\nPairwise Similarities:")
for i in range(len(documents)):
    for j in range(i+1, len(documents)):
        print(f"Doc {i} vs Doc {j}: {similarity_matrix[i, j]:.4f}")

# Output:
# Doc 0 vs Doc 1: 0.4201 (both about animals sitting)
# Doc 0 vs Doc 2: 0.2357 (both mention cats/dogs)
# Doc 0 vs Doc 3: 0.0000 (completely different topics)
# Doc 1 vs Doc 2: 0.2357
# Doc 1 vs Doc 3: 0.0000
# Doc 2 vs Doc 3: 0.0000
```

Finding Most Similar Documents

Practical function to find top-k most similar documents:

```
import numpy as np
def find_most_similar(query_idx, similarity_matrix, k=3):
    """
    Find k most similar documents to the query document.
    Args:
        query_idx: Index of query document
        similarity_matrix: Pairwise similarity matrix
        k: Number of similar documents to return
    Returns:
        List of (document_index, similarity_score) tuples
    """
    # Get similarities for query document
    similarities = similarity_matrix[query_idx]

    # Get indices of top-k most similar (excluding the query itself)
    # argsort gives indices in ascending order, so we reverse with [::-1]
    top_indices = np.argsort(similarities)[::-1][1:k+1]

    # Return list of (index, score) tuples
    return [(idx, similarities[idx]) for idx in top_indices]

# Example usage
query_document = 0  # "The cat sat on the mat"
similar_docs = find_most_similar(query_document, similarity_matrix, k=2)

print(f"Documents most similar to '{documents[query_document]}':")
for doc_idx, score in similar_docs:
    print(f"  Doc {doc_idx} (score: {score:.4f}): {documents[doc_idx]}")

# Output:
# Documents most similar to 'The cat sat on the mat':
#   Doc 1 (score: 0.4201): The dog sat on the log
#   Doc 2 (score: 0.2357): Cats and dogs are enemies
```

Part 8

Summary and Best Practices

Key Takeaways

Essential concepts from this lecture:

1. **Vector Space Model + Similarity = Information Retrieval**
   - Documents as vectors enable mathematical comparison
   - Similarity measures quantify "how alike" documents are

2. **Cosine Similarity is the Standard for Text**
   - Measures angle between vectors (direction, not magnitude)
   - Range: 0 (orthogonal) to 1 (identical direction)
   - Formula: $\frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1|| \times ||\mathbf{d}_2||}$

3. **Similarity $\neq$ Distance**
   - Similarity: higher = more alike
   - Distance: lower = more alike

Calculation Checklist

Steps to compute cosine similarity by hand:

1. **Identify the vectors**
   - Write out both document vectors explicitly
   - Check dimensions match
2. **Compute the dot product** (numerator)
   - Multiply corresponding elements
   - Sum all products
3. **Compute vector magnitudes** (denominator)
   - Square each element, sum, take square root
   - Do this for both vectors
4. **Divide numerator by product of denominators**
   - Result should be between 0 and 1 (for non-negative vectors)
   - Interpret: close to 1 = very similar

When to Use Which Measure

Task	Measure	Reason
Document similarity	Cosine	Ignores length, focuses on content
Document search/ranking	Cosine	Standard in IR systems
Document clustering	Cosine	Works with sparse, high-dim vectors
Plagiarism detection	Cosine	Captures lexical overlap
Recommender systems	Cosine	Content-based filtering
Geographic data	Euclidean	Actual spatial distance matters
Binary features	Jaccard or Hamming	Designed for binary data
Categorical features	Hamming	Counts disagreements

Golden Rule for NLP

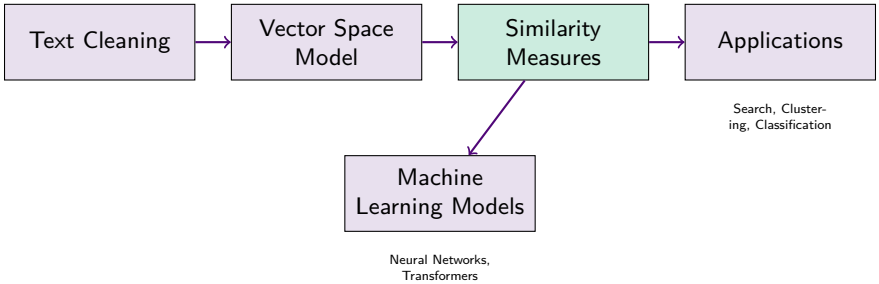
When in doubt with text data, use cosine similarity!

Mistakes students often make:

1. **Forgetting to square root in magnitude calculation**
   - Remember: $||\mathbf{d}|| = \sqrt{d_1^2 + d_2^2 + \cdots}$, not just the sum of squares
2. **Using Euclidean distance for text documents**
   - Euclidean is sensitive to magnitude – use cosine instead!
3. **Confusing similarity with distance**
   - High similarity = low distance
   - Don't accidentally invert your rankings!
4. **Not normalising vectors before comparison**
   - If using dot product alone, different-length documents aren't comparable
   - Cosine similarity includes normalisation (dividing by magnitudes)

Connecting Back to the Big Picture

How this fits into your NLP journey:



- **Past:** Text cleaning → VSM representation
- **Now:** Similarity measures connect vectors to applications
- **Future:** These foundations extend to modern deep learning

Even with advanced models (BERT, GPT), understanding vector similarity remains crucial!

Further Reading and Resources

Recommended resources:

Textbooks:

- *Introduction to Information Retrieval* by Manning, Raghavan, & Schütze (Chapter 6)
- *Speech and Language Processing* by Jurafsky & Martin (Chapter 6)

Online tutorials:

- Scikit-learn documentation: `sklearn.metrics.pairwise`
- Towards Data Science: "Understanding Cosine Similarity"

Practice:

- Implement similarity calculations in Python
- Try different distance measures on the same dataset
- Build a simple document search system

Next steps:

- Tutorial: Implement cosine similarity from scratch
- Coursework: Use these concepts for document classification

Questions to Test Your Understanding

Can you answer these?

1. Why is cosine similarity better than Euclidean distance for comparing documents?
2. If two document vectors have a cosine similarity of 0, what does this mean geometrically?
3. Given vectors $(4, 0, 2)$ and $(2, 0, 1)$, calculate their cosine similarity.
4. How would you use cosine similarity to build a simple search engine?
5. When might you prefer Jaccard similarity over cosine similarity?

If you can confidently answer these, you've mastered the material!

1. Why is cosine similarity better than Euclidean distance for comparing documents?
Cosine similarity measures the *angle* between vectors (direction/topic), not the *magnitude* (length). This means a short article and a long book about the same topic will have high similarity, whereas Euclidean distance would incorrectly treat them as very different due to the difference in word counts.

2. If two document vectors have a cosine similarity of 0, what does this mean geometrically?
The vectors are *orthogonal* (at a 90◦ angle to each other). This means they share no common terms — the documents discuss completely different topics with no vocabulary overlap.

3. Given vectors $(4, 0, 2)$ and $(2, 0, 1)$, calculate their cosine similarity.

- Dot product: $(4 \times 2) + (0 \times 0) + (2 \times 1) = 8 + 0 + 2 = 10$
- $||(4, 0, 2)|| = \sqrt{16 + 0 + 4} = \sqrt{20} \approx 4.472$
- $||(2, 0, 1)|| = \sqrt{4 + 0 + 1} = \sqrt{5} \approx 2.236$
- Cosine similarity $= \frac{10}{4.472 \times 2.236} = \frac{10}{10} = \mathbf{1.0}$

Note: $(4, 0, 2) = 2 \times (2, 0, 1)$ — they point in the same direction, so similarity is perfect!

4. How would you use cosine similarity to build a simple search engine?

1. **Index documents:** Convert all documents in your collection to TF-IDF vectors
2. **Process query:** When a user enters a search query, convert it to a TF-IDF vector using the same vocabulary
3. **Compute similarities:** Calculate cosine similarity between the query vector and every document vector
4. **Rank results:** Sort documents by similarity score (highest first)
5. **Return top-k:** Display the k most similar documents to the user

5. When might you prefer Jaccard similarity over cosine similarity?

Use Jaccard similarity when:

- You only care about *presence/absence* of terms, not their frequency
- Comparing sets of tags, categories, or binary features
- Working with very short texts where term frequency is less meaningful
- Example: Comparing hashtags in tweets, product categories, or user interests

Jaccard treats documents as *sets of words*, ignoring how many times each word appears. Cosine uses the full term frequency information.

Questions?

Prof. Georgina Cosma

g.cosma@lboro.ac.uk