Similarity-Based Text Retrieval

From Basics to Vector Spaces

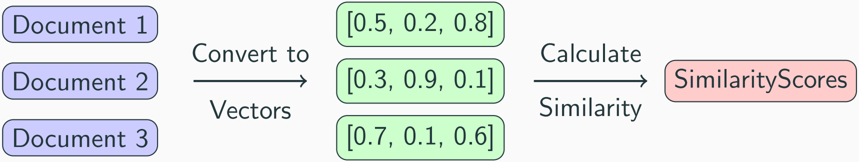
Bala Priya C

29th May, 2025

What We'll Cover Today

- What is text similarity?
- Vector space model basics
- Text representation methods (Count, TF-IDF, Word2Vec)
- Distance and similarity measures
- Visual examples and simple math

The Big Picture



What is Text Similarity?

Goal: Find how "similar" two pieces of text are

Examples:

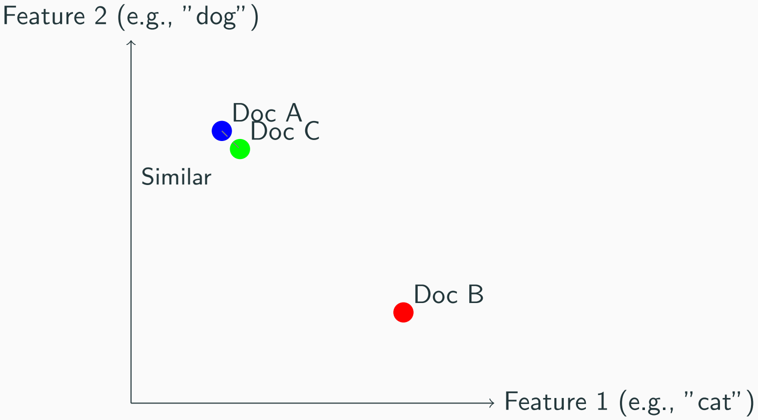
- "The cat sat on the mat" vs "A cat sits on a mat" $\rightarrow$ High similarity
- "Machine learning algorithms" vs "Cooking pasta recipes" $\rightarrow$ Low similarity

Applications:

- Search engines
- Recommendation systems
- Duplicate detection
- Document clustering

Vector Space Model - The Core Idea

Key Insight: Represent text as vectors in high-dimensional space



Closer points = More similar documents

Method 1: Simple Word Count

Idea: Count how many times each word appears

Example:

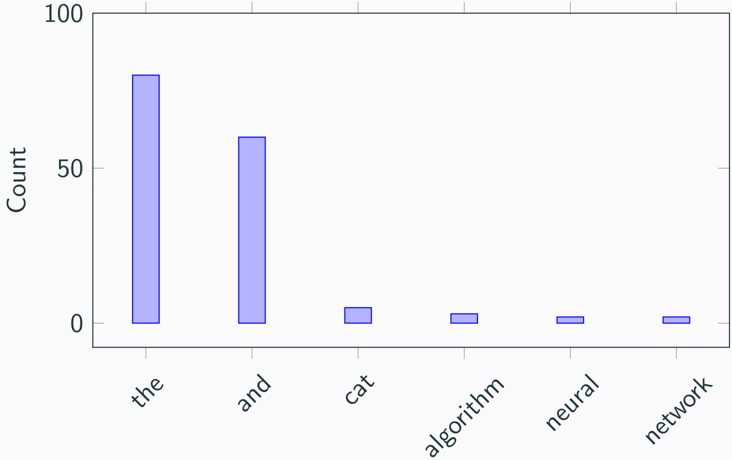
- Doc 1: "The cat sat on the mat"
- Doc 2: "The dog sat on the floor"

Word	Doc 1	Doc 2
the	2	2
cat	1	0
dog	0	1
sat	1	1
on	1	1
mat	1	0
floor	0	1

Vectors: Doc1 = [2,1,0,1,1,1,0], Doc2 = [2,0,1,1,1,0,1]

Problem with Simple Counts

Issue: Common words dominate



Solution: Use TF-IDF to balance frequency with rarity!

Method 2: TF-IDF

TF-IDF = Term Frequency × Inverse Document Frequency

Term Frequency (TF):

$$TF(t, d) = \frac{\text{count of term } t \text{ in document } d}{\text{total terms in document } d}$$

Inverse Document Frequency (IDF):

$$IDF(t) = \log\left(\frac{\text{total documents}}{\text{documents containing term } t}\right)$$

TF-IDF Score:

$$\text{TF-IDF}(t, d) = TF(t, d) \times IDF(t)$$

Intuition: Rare words get higher scores, common words get lower scores

TF-IDF Example

3 Documents:

- Doc 1: "cat sat mat"
- Doc 2: "dog sat floor"
- Doc 3: "cat dog animal"

For word "cat":

- TF in Doc 1: $\frac{1}{3} = 0.33$
- IDF: $\log\left(\frac{3}{2}\right) = 0.18$ (appears in 2 out of 3 docs)
- TF-IDF: $0.33 \times 0.18 = 0.06$

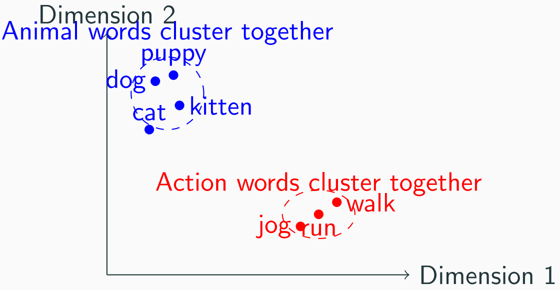
For word "sat":

- TF in Doc 1: $\frac{1}{3} = 0.33$
- IDF: $\log\left(\frac{3}{2}\right) = 0.18$
- TF-IDF: $0.33 \times 0.18 = 0.06$

Method 3: Word2Vec

Problem: "cat" and "kitten" should be similar, but TF-IDF treats them as completely different

Word2Vec Solution: Learn word meanings from context



Word2Vec: How It Works

Training Process:

"The cat sat on the mat"

$\downarrow$

Neural Network

$\downarrow$

cat = [0.2, -0.1, 0.8, 0.3, ...]
sat = [-0.1, 0.5, 0.2, -0.4, ...]

Key Idea: Words that appear in similar contexts get similar vectors

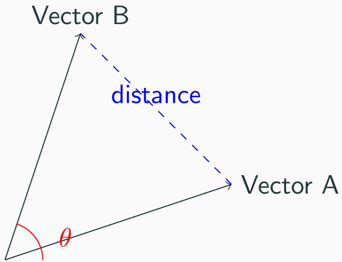
Result: Document vector = average of all word vectors in the document

Similarity Measures - The Math

Once we have vectors, how do we measure similarity?

Three main approaches:

1. Cosine Similarity - Angle between vectors
2. Euclidean Distance - Straight-line distance
3. Jaccard Similarity - Set overlap



Cosine Similarity

Measures the angle between two vectors

Formula:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| \times |\mathbf{B}|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

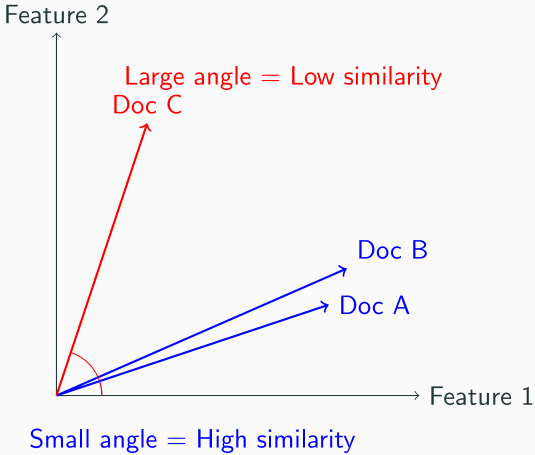
Example:

- Vector A = [1, 2, 3]
- Vector B = [2, 4, 6]

$$\cos(\theta) = \frac{1 \times 2 + 2 \times 4 + 3 \times 6}{\sqrt{1^2 + 2^2 + 3^2} \times \sqrt{2^2 + 4^2 + 6^2}} = \frac{26}{\sqrt{14} \times \sqrt{56}} = 1.0$$

Range: [-1, 1] where 1 = identical direction, 0 = perpendicular, -1 = opposite

Cosine Similarity Visualization



Key Insight: Cosine similarity ignores magnitude, only cares about direction

Euclidean Distance

Measures straight-line distance between points

Formula:

$$d(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

Example:

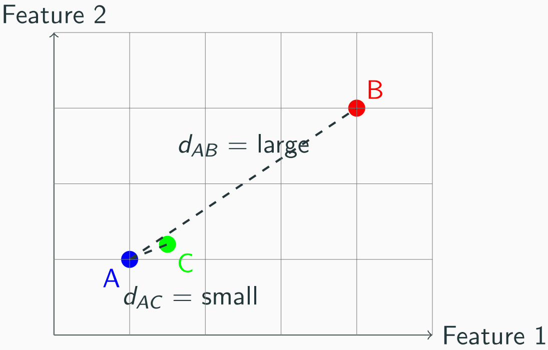
- Vector A = [1, 2]
- Vector B = [4, 6]

$$d(A, B) = \sqrt{(1-4)^2 + (2-6)^2} = \sqrt{9+16} = \sqrt{25} = 5$$

To convert to similarity:

$$\text{Similarity} = \frac{1}{1 + \text{distance}}$$

Euclidean Distance Visualization



Key Insight: Euclidean distance considers both direction and magnitude

Jaccard Similarity

Measures overlap between sets (good for binary features)

Formula:

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

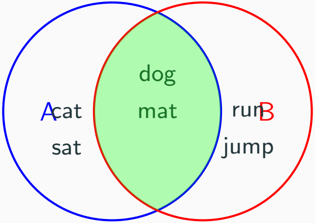
Example:

- Doc A words: {cat, sat, mat}
- Doc B words: {cat, dog, mat}

- Intersection: {cat, mat} $\rightarrow$ size $= 2$
- Union: {cat, sat, mat, dog} $\rightarrow$ size $= 4$
- Jaccard $= \frac{2}{4} = 0.5$

Range: [0, 1] where 0 = no overlap, 1 = identical sets

Jaccard Similarity Visualization



Intersection: {mat, dog}

Union: {cat, sat, mat, dog, run, jump}

Jaccard $= 2/6 = 0.33$

Putting It All Together: Complete Example

Most common in text retrieval: Cosine similarity with TF-IDF or Word2Vec **Documents:**

- Doc 1: "machine learning algorithms"
- Doc 2: "deep learning neural networks"
- Doc 3: "cooking pasta recipes"

Step 1: Convert to TF-IDF vectors

- Doc 1: [0.7, 0.5, 0.3, 0, 0, 0, 0, 0]
- Doc 2: [0, 0.6, 0, 0.8, 0.4, 0, 0, 0]
- Doc 3: [0, 0, 0, 0, 0, 0.9, 0.6, 0.8]

Step 2: Calculate cosine similarities

- Sim(Doc1, Doc2) = 0.3 (some similarity - both about learning)
- Sim(Doc1, Doc3) = 0.0 (no similarity)
- Sim(Doc2, Doc3) = 0.0 (no similarity)

Real-World Performance Tips

Making it work in practice:

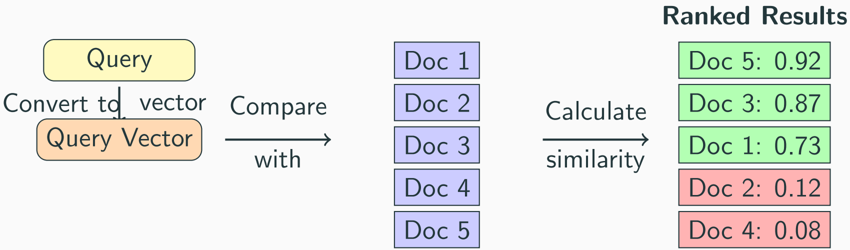
- **Preprocessing:** Remove stopwords, lowercase, stemming
- **Dimensionality:** Use 50-300 dimensions for Word2Vec
- **Speed:** Use approximate nearest neighbor search for large datasets
- **Evaluation:** Test with human judgments of similarity

Tools to try:

- Python: scikit-learn, gensim, sentence-transformers
- Libraries: Elasticsearch, Faiss, Annoy

Document Retrieval in Practice

The Search Process



Steps

1. Convert query to vector
2. Calculate similarity with all documents
3. Rank by similarity score
4. Return top documents

Key Takeaways

1. **Text → Vectors:** Convert documents to numerical representations
2. **Three main methods:**
   - Count vectors (simple but limited)
   - TF-IDF (balances frequency and rarity)
   - Word2Vec (captures semantic meaning)
3. **Three similarity measures:**
   - Cosine (direction-based, most popular)
   - Euclidean (distance-based)
   - Jaccard (set-based)
4. **Document retrieval:** Query → Vector → Compare → Rank → Return top results
5. **Choose based on your use case and data characteristics**

Questions?