06 Cosine Similarity in Vector Similarity Search

Cosine Similarity in Vector Similarity Search

1. Introduction to Vector Similarity

In the realm of information retrieval, recommendation systems, and machine learning, we often need to determine how similar two items are to each other. Vector similarity metrics provide mathematical frameworks for quantifying this similarity when items are represented as vectors in a high-dimensional space.

These lecture notes focus on cosine similarity, one of the most widely used similarity measures, especially in the context of vector databases and text embeddings. We'll explore its mathematical foundations, practical applications, advantages, limitations, and implementation considerations.

2. Vector Representations and Embeddings

2.1 Vector Spaces

Before discussing similarity, we need to understand what vectors represent:

- A vector is an ordered list of numbers that defines a point in a multi-dimensional space
- Each dimension in this space represents a distinct feature or attribute
- The vector $\vec{v} = [v_1, v_2, \ldots, v_n]$ has n dimensions, each capturing some aspect of the entity it represents

2.2 Embeddings

Embeddings are special vector representations that capture semantic meaning:

- An embedding is a learned vector representation that maps entities (words, documents, images, etc.) to points in a continuous vector space
- Similar entities are positioned closer together in this space
- Modern embedding techniques include Word2Vec, GloVe, BERT, Sentence Transformers, etc.
- The dimensionality of embeddings typically ranges from 100 to 1024 or more

Example: In a word embedding space, vectors for "king", "queen", "man", and "woman" might satisfy the relationship: $\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$

3. Mathematical Definition of Cosine Similarity

3.1 Basic Definition

Cosine similarity measures the cosine of the angle between two non-zero vectors:

$$\text{cosine similarity}(\vec{A}, \vec{B}) = \cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \cdot ||\vec{B}||}$$

Where:

- $\vec{A} \cdot \vec{B}$ is the dot product of vectors $\vec{A}$ and $\vec{B}$
- $||\vec{A}||$ and $||\vec{B}||$ are the Euclidean norms (magnitudes) of vectors $\vec{A}$ and $\vec{B}$

3.2 Expanded Formula

For vectors $\vec{A} = [A_1, A_2, \ldots, A_n]$ and $\vec{B} = [B_1, B_2, \ldots, B_n]$:

$$\text{cosine similarity}(\vec{A}, \vec{B}) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

3.3 Properties of Cosine Similarity

- **Range**: The output ranges from -1 to 1
  - 1 indicates perfectly aligned vectors (identical direction)
  - 0 indicates orthogonal vectors (perpendicular, no correlation)
  - -1 indicates exactly opposite vectors
- **Invariance to scaling**: Multiplying either vector by a positive scalar does not change the similarity
- **Symmetric**: $\text{cosine similarity}(\vec{A}, \vec{B}) = \text{cosine similarity}(\vec{B}, \vec{A})$

3.4 Cosine Distance

Cosine distance is derived from cosine similarity:

$$\text{cosine distance}(\vec{A}, \vec{B}) = 1 - \text{cosine similarity}(\vec{A}, \vec{B})$$

For normalized vectors (with non-negative components), this gives a distance measure in the range [0, 2].

4. Geometric Interpretation

Cosine similarity has an intuitive geometric interpretation:

- It measures the cosine of the angle between two vectors
- Two vectors pointing in the same direction have cosine similarity = 1
- Two vectors at right angles (90°) have cosine similarity = 0
- Two vectors pointing in opposite directions have cosine similarity = -1

This makes it particularly useful for measuring directional similarity, regardless of magnitude.

5. Cosine Similarity in Text Analysis

5.1 Text Representation as Vectors

There are several ways to represent text as vectors:

- **Bag-of-Words (BoW)**: Each dimension represents the frequency of a word
- **TF-IDF**: Weight terms by their frequency in the document and inverse frequency in the corpus

- **Word Embeddings**: Average or aggregate word vectors (Word2Vec, GloVe)
- **Sentence/Document Embeddings**: Generated by models like BERT, Sentence-BERT, or OpenAI's text embeddings

5.2 Example: Document Similarity

Consider two documents represented as TF-IDF vectors:

Document 1: "Vector databases store embeddings efficiently" Document 2: "Embedding vectors require special storage systems"

After vectorization, we might have:

- $\vec{D}_1 = [0.3, 0.5, 0.4, 0.6, 0.0, 0.0]$
- $\vec{D}_2 = [0.0, 0.4, 0.0, 0.5, 0.6, 0.4]$

Cosine similarity calculation:

$$\text{cosine similarity}(\vec{D}_1, \vec{D}_2) = \frac{0.5 \times 0.4 + 0.6 \times 0.5}{\sqrt{0.3^2 + 0.5^2 + 0.4^2 + 0.6^2} \times \sqrt{0.4^2 + 0.5^2 + 0.6^2 + 0.4^2}} \approx 0.54$$

This moderate similarity (0.54) indicates some topical overlap between the documents.

6. Vector Databases and Similarity Search

6.1 Vector Databases

Vector databases are specialized systems designed to store and query vector embeddings efficiently:

- **Examples**: Pinecone, Milvus, Weaviate, Qdrant, Faiss, Chroma
- **Key features**:
  - Optimized for high-dimensional vector storage
  - Fast similarity search capabilities
  - Support for various distance metrics
  - Scaling to billions of vectors
  - Metadata filtering

6.2 Approximate Nearest Neighbor (ANN) Search

Vector databases implement ANN algorithms to perform efficient similarity search:

- **Exact k-NN**: Find the k most similar vectors to a query vector
- **Approximate k-NN**: Trade perfect accuracy for significant speed improvements
- **Common indexing techniques**:
  - Hierarchical Navigable Small World (HNSW)
  - Inverted File Index (IVF)
  - Product Quantization (PQ)
  - Locality-Sensitive Hashing (LSH)

6.3 Cosine Similarity Implementation in Vector Databases

Many vector databases implement cosine similarity search as follows:

1. **Vector normalization**: Normalize vectors during indexing to convert cosine similarity to dot product comparison
    - For normalized vectors: $\cos(\theta) = \vec{A} \cdot \vec{B}$ when $||\vec{A}|| = ||\vec{B}|| = 1$
2. **Index construction**: Build an index structure optimized for the similarity metric
3. **Query processing**: Convert the query vector to the same space and perform similarity search

7. Practical Applications of Cosine Similarity

7.1 Information Retrieval

- **Semantic search**: Finding documents semantically similar to a query
- **Question answering**: Retrieving relevant passages for extractive QA

7.2 Recommendation Systems

- **Content-based filtering**: Recommending items with similar feature vectors
- **Item-item similarity**: Finding products/articles/movies similar to ones a user liked

7.3 Natural Language Processing

- **Document clustering**: Grouping similar documents
- **Plagiarism detection**: Identifying similar text passages
- **Text classification**: Comparing document vectors to category centroids

7.4 Multi-modal Search

- **Image-text similarity**: Comparing image embeddings with text embeddings
- **Cross-modal retrieval**: Finding images matching text descriptions or vice versa

8. Advantages and Limitations of Cosine Similarity

8.1 Advantages

- **Scale invariance**: Focus on orientation rather than magnitude
- **Effective for sparse high-dimensional spaces**: Works well with text data
- **Computational efficiency**: Relatively simple to implement and optimize
- **Intuitive interpretation**: Angle between vectors is conceptually clear

8.2 Limitations

- **Insensitive to absolute magnitudes**: May miss important magnitude differences
- **Limited to angular distance**: Cannot capture all types of semantic relationships
- **Challenges with negative values**: Interpretation becomes less intuitive
- **Not a proper distance metric**: Doesn't satisfy the triangle inequality

9. Alternative Similarity Measures

9.1 Euclidean Distance

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

- Measures the straight-line distance between points
- Sensitive to both direction and magnitude

9.2 Manhattan Distance

$$d(\vec{A}, \vec{B}) = \sum_{i=1}^{n}|A_i - B_i|$$

- Sum of absolute differences along each dimension
- Less sensitive to outliers than Euclidean distance

9.3 Dot Product

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

- Simple and efficient but affected by vector magnitudes
- Often used with normalized vectors (equivalent to cosine similarity)

9.4 Jaccard Similarity

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

- Useful for binary vectors or sets
- Measures overlap between sets

10. Implementation Considerations

10.1 Normalization

Pre-normalizing vectors offers computational advantages:

$$\vec{A}_{\text{norm}} = \frac{\vec{A}}{||\vec{A}||}$$

For normalized vectors, cosine similarity reduces to the dot product:

$$\text{cosine similarity}(\vec{A}_{\text{norm}}, \vec{B}_{\text{norm}}) = \vec{A}_{\text{norm}} \cdot \vec{B}_{\text{norm}}$$

10.2 Performance Optimization

For large-scale applications:

- **Dimensionality reduction**: Techniques like PCA can reduce vector size

- **Quantization**: Reducing precision of vector components
- **Specialized hardware**: GPUs and TPUs for parallel computation
- **Caching**: Storing frequently accessed vectors in memory

10.3 Python Implementation Example

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

# Define two document vectors
doc1_vector = np.array([0.2, 0.5, 0.7, 0.1, 0.8]).reshape(1, -1)
doc2_vector = np.array([0.1, 0.4, 0.5, 0.2, 0.9]).reshape(1, -1)

# Calculate cosine similarity
similarity = cosine_similarity(doc1_vector, doc2_vector)[0][0]
print(f"Cosine similarity: {similarity:.4f}")

# Manual calculation
def manual_cosine_similarity(vec1, vec2):
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)

manual_sim = manual_cosine_similarity(doc1_vector.flatten(), doc2_vector.flatten())
print(f"Manually calculated similarity: {manual_sim:.4f}")
```

12. Advanced Topics

12.1 Improving Embedding Quality

The quality of similarity search depends heavily on the embedding quality:

- **Contrastive learning**: Training embeddings to specifically optimize for similarity tasks
- **Fine-tuning**: Adapting pre-trained models to specific domains
- **Domain-specific models**: Using or training embeddings for specialized fields

12.2 Handling the Curse of Dimensionality

As dimensions increase, similarity measures become less discriminative:

- **Dimensionality reduction**: Using techniques like PCA, t-SNE, or UMAP
- **Feature selection**: Identifying and keeping only the most informative dimensions
- **Specialized distance metrics**: Using metrics designed for high-dimensional spaces

12.3 Time Series Similarity

For comparing time series data:

- **Dynamic Time Warping (DTW)**: Measuring similarity between temporal sequences
- **Windowed cosine similarity**: Applying cosine similarity to sliding windows

13. Conclusion and Future Directions

Cosine similarity remains a fundamental tool in vector similarity search due to its simplicity, efficiency, and effectiveness, particularly for text data. Modern vector databases have made it practical to deploy similarity search at scale.

Emerging research directions include:

- **Learned similarity metrics**: Models that learn optimal similarity functions
- **Context-aware similarity**: Adapting similarity measures based on query context
- **Efficient multi-modal similarity**: Comparing vectors from different modalities
- **Privacy-preserving similarity search**: Computing similarity without revealing the raw vectors

As models and embedding techniques continue to evolve, so too will the methods for comparing and retrieving them, with cosine similarity remaining a cornerstone of these approaches.