Introduction to Vector Databases and Chroma DB Cheat Sheet

Skills Network

Estimated Reading Time: 15 minutes

Distance and Similarity Metrics

1. L2 Distance (Euclidean Distance)

Definition: The L2 distance between two vectors aa and bb is the square root of the sum of the squared differences between corresponding elements:

$$L2(a,b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

L2(a, b) = √Σ i=1n(ai − bi)2

Properties:

- The L2 distance measures the straight-line distance between two points in Euclidean space, following the principles of the Pythagorean theorem.
- It is sensitive to both the magnitude and direction of the vectors, meaning that changes in either can significantly affect the calculated distance.
- This distance metric is commonly used in applications involving spatial or geometric data, such as image analysis, computer vision, and geographic mapping.

Use Case: A common use case for L2 distance is determining the closest point to a given location in two-dimensional (2D) or three-dimensional (3D) space. This approach is frequently used in computer vision tasks, where spatial proximity between features or objects plays a critical role.

2. Dot Product (Inner Product) Similarity and Distance

Definition: The dot product of two vectors is:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

a · b = Σ i=1n aibi

Alternative Calculation: The dot product can also be calculated using magnitudes and angles:

$$a \cdot b = ||a||\,||b||cos(\alpha)$$

a · b = | | a | | | | b | | cos(α)

here $||a|| = \sqrt{\sum_{k=1}^{n} a_k^2}$ | | a | | = √Σ k=1n ak2

is the L2 norm of vector aa, and αα is the angle between vectors aa and bb.

Properties:

- The dot product of two vectors can be positive, negative, or zero, depending on the angle between them.
- Larger dot product values typically indicate a higher degree of similarity between the vectors, especially when they are pointing in similar directions.
- If a **distance-like metric** is needed, the **negative of the dot product** can be used. In this case, larger (more negative) values correspond to greater dissimilarity or distance.
- The dot product is sensitive to both the magnitude and direction of the vectors, meaning that changes in either will affect the result.
- It is frequently used in machine learning models, such as in neural networks for computing activations or in matrix factorization for recommendation systems.

Use Case: When the length of the vector (representing something such as relevance, confidence, or quantity) is meaningful. For instance, in recommendation systems, the direction of vectors typically indicates two products are about the same topic, but larger magnitudes might indicate that a product is more popular. In this case using dot product similarity makes sense, because one would want to recommend the products that are not only about the same topic, but also popular.

3. Cosine Similarity and Distance

Definition: Cosine similarity measures the cosine of the angle between two vectors:

$$cosine\_similarity(a,b) = \frac{a \cdot b}{||a||\,||b||}$$

cosine_similarity(a, b) = | | a | | | | b | | a·b

Distance Conversion: To convert cosine similarity to cosine distance:

$$\text{cosine\_distance}(a, b) = 1 - \text{cosine\_similarity}(a, b)$$

cosine_distance(a, b) = 1 − cosine_similarity(a, b)

Alternative Calculation with Normalized Vectors: Cosine similarity can be calculated more efficiently if the vectors are normalized. To normalize a vector, divide it by its L2 norm:

$$norm(a) = \frac{a}{||a||}$$

norm(a) = ∣ ∣ a ∣ ∣ a

A normalized vector has the property that the sum of the squared components sums to one:

$$\sum_{i=1}^{n} norm(a)_i^2 = 1$$

∑ i=1n norm(a)i2 = 1

Calculating cosine similarity between two normalized vectors is as easy as calculating the dot product between them:

$$\text{cosine\_similarity}(a, b) = \frac{a \cdot b}{||a||\,||b||} = \frac{a}{||a||} \cdot \frac{b}{||b||} = norm(a) \cdot norm(b)$$

cosine_similarity(a, b) = ∣ ∣ a ∣ ∣ ∣ ∣ b ∣ ∣ a⋅b = ∣ ∣ a ∣ ∣ a⋅ ∣ ∣ b ∣ ∣ b = norm(a) ⋅ norm(b)

Properties:

- Cosine similarity focuses on the orientation of vectors rather than their magnitude, measuring the cosine of the angle between them.
- It is particularly well-suited for high-dimensional, sparse data, such as text embeddings or term-frequency vectors in natural language processing.
- This metric is invariant to vector length, meaning that scaling a vector up or down does not affect the similarity score.

Use Case: A common use case for cosine similarity is measuring document similarity in natural language processing (NLP), where it helps identify texts with similar content regardless of their length.

Choosing the Right Metric

Metric	Sensitive to Magnitude	Normalized	Best For
L2 Distance	✅ Yes	❌ No	Spatial data, clustering
Cosine Distance	❌ No	✅ Yes	Text, embeddings, NLP
Dot Product	✅ Yes	❌ No	Neural networks, recommender systems

Practical Considerations

- **Normalization**: Normalize vectors if you expect to only need to calculate cosine similarity. For many natural language processing tasks and text embedding models, this is the default option.
- **High-Dimensional Data**:
  - L2 distance can suffer from the "curse of dimensionality." Consider using a different metric or using a dimensionality reduction technique if the data exhibits high dimensionality.
  - Cosine distance often performs better in high dimensions, and is the default choice in many natural language processing and text-based tasks.
  - Dot product can be computed efficiently using matrix operations, and can be used to calculate cosine similarity if the vectors are normalized.

Vector Databases versus Traditional Databases

Function	Traditional Databases	Vector Databases
Data Representation	Structured format using tables, rows, columns	Multi-dimensional vectors encoding complex/unstructured data
Data Search	SQL queries for structured data	Similarity searches for vectorized data
Indexing	B-trees for efficient retrieval	Specialized indices such as the graph-based HNSW for approximate nearest neighbor search
Scalability	Resource augmentation or data sharding	Distributed architectures for horizontal scaling
Applications	Business applications, transactional systems	Context-aware AI applications, similarity search, NLP, multimedia analysis

Vector Database Fundamentals

Vector Database: Specialized database designed to store and query vectorized data rapidly. Data is represented as vectors in multi-dimensional space, where each vector dimension corresponds to a specific attribute.

Vector Libraries vs Vector Databases:

- **Vector Libraries**: In-memory with similarity capabilities

- **Vector Databases**: Full CRUD operations (create, read, update, delete) + enterprise production deployments

Chroma DB Key Tips and Best Practices

Filtering Best Practices

- **Document filtering is case-sensitive** in Chroma DB - searching for "Pandas" will not find "pandas"
- Combine metadata and document filters for precise results using both `where` and `where_document` parameters
- Use `$and` and `$or` operators for complex filtering logic
- Use `$in` and `$nin` for list-based filtering

Key Chroma DB Features

- **Dual-filtering approach**: Supports both metadata filtering (structured attributes) and document filtering (full text search)
- **Metadata filtering**: Similar to SQL `WHERE` clauses, but more flexible and can be combined with vector search
- **Document filtering**: Comparable to SQL's `CONTAINS` or `LIKE` operators, but more powerful when integrated with vector search
- **Full text search**: Document filtering is also referred to as full text search in Chroma DB

Chroma DB Setup & Configuration

Basic Setup

```
import chromadb
from chromadb.utils import embedding_functions
# Define embedding function
ef = embedding_functions.SentenceTransformerEmbeddingFunction(
    model_name="all-MiniLM-L6-v2"
)
# Create client
client = chromadb.Client()
```

Collection Creation with Typical HNSW Configuration

```
collection = client.create_collection(
    name="my_collection_name",
    metadata={"topic": "query testing"},
    configuration={
        "hnsw": {
            "space": "cosine",    # Distance metric
        },
        "embedding_function": ef
    }
)
```

What is a Vector Index?

Finding semantically similar vectors using brute-force comparison (checking every vector in the database) becomes inefficient at scale. **Vector indexes** solve this by using specialized data structures that enable algorithms to compute similarity scores with only a small subset of vectors, significantly speeding up search while still returning exact or near-optimal results.

A **vector index** organizes high-dimensional embeddings to reflect the geometry of the vector space, clustering similar vectors together or linking them through proximity-based graphs. This allows the search algorithm to **prune large portions of the dataset** early in the search process, enabling scaling to millions or billions of vectors while maintaining low-latency performance.

What is HNSW?

HNSW (Hierarchical Navigable Small World) is a fast, scalable graph-based vector index designed for **approximate nearest neighbor (ANN)** search in high-dimensional spaces. It is the sole indexing method supported by Chroma DB and is widely adopted in other vector databases due to its performance and reliability.

How It Works: HNSW builds a **multi-layered graph** where:

- The **upper layers** contain a sparse overview of the data for fast navigation.
- The **bottom layer** holds all vectors for detailed search.
- Each vector connects to a few nearby neighbors, forming a **"small world" network**—meaning most vectors can be reached in just a few steps.

Search Process: The algorithm starts at the top layer and moves closer to the query vector as it descends, refining the search at each level. This structure allows it to skip most of the dataset and still find highly similar vectors quickly.

Why Use HNSW?:

- **Fast**: Avoids scanning the entire dataset
- **Accurate**: Delivers near-exact results
- **Scalable**: Handles millions to billions of vectors
- **Versatile**: Works with various similarity metrics

HNSW Distance Metric Parameters

- `space`: Distance metric options
  - `l2`: Squared L2 (Euclidean) distance (default)
  - `ip`: Inner (dot) product distance
  - `cosine`: Cosine distance

HNSW Performance Parameters: Key Trade-offs

- **Higher** `ef_search`: Better accuracy, slower performance
- **Higher** `ef_construction`: Better index quality, longer build time
- **Higher** `max_neighbors`: Better search performance, more memory usage

Data Operations

Adding Documents

```
collection.add(
    documents=[
        "Document text 1",
        "Document text 2"
    ],
    metadatas=[
        {"source": "source1", "category": "type1"},
        {"source": "source2", "category": "type2"}
    ],
    ids=["id1", "id2"]
)
```

Retrieving Documents

```
# Get all documents
all_items = collection.get()
# Get with metadata filter
filtered_items = collection.get(
    where={"source": "source1"}
)
```

Filtering in Chroma DB

Metadata Filtering Operators

```
# Basic equality
where={"key": "value"}
# Equivalent to
where={"key": {"$eq": "value"}}
# Comparison operators
"$eq"   # equal to (string, int, float)
"$ne"   # not equal to
"$gt"   # greater than (int, float)
"$gte"  # greater than or equal to
"$lt"   # less than (int, float)
"$lte"  # less than or equal to
"$in"   # in list
"$nin"  # not in list
```

Complex Filters with Logical Operators

```
# AND operation; you can replace `$and` with `$or` to make this an OR operation
collection.get(
    where={
        "$and": [
            {"source": {"$eq": "langchain.com"}},
            {"version": {"$lt": 0.3}}
        ]
    }
)
# Using a list to perform an OR operation on the values of a metadata key
collection.get(
    where={
        "$and": [
            {"source": {"$in": ["langchain.com", "llamaindex.ai"]}},
            {"version": {"$lt": 0.3}}
        ]
    }
)
```

Document Content Filtering

```
# Contains text
where_document={"$contains": "pandas"}
# Does not contain text
where_document={"$not_contains": "library"}
# Combined with logical operators
where_document={
    "$or": [
        {"$contains": "LangChain"},
        {"$contains": "Python"}
    ]
}
```

Similarity Search

Basic Query

```
results = collection.query(
    query_texts=["search term"],
    n_results=3  # Number of results to return
)
```

Query with Filters

```
# With metadata filter
results = collection.query(
    query_texts=["polar bear"],
    n_results=1,
    where={'topic': 'animals'}
)
# With document filter
results = collection.query(
    query_texts=["polar bear"],
    n_results=1,
    where_document={'$not_contains': 'library'}
)
# Combined filters
results = collection.query(
    query_texts=["polar bear"],
    n_results=1,
    where={'topic': 'animals'},
    where_document={'$not_contains': 'library'}
)
```

Common Workflow Pattern

```
# 1. Setup
import chromadb
from chromadb.utils import embedding_functions
# 2. Create embedding function and client
ef = embedding_functions.SentenceTransformerEmbeddingFunction(model_name="all-MiniLM-L6-v2")
client = chromadb.Client()
# 3. Create collection with configuration
collection = client.create_collection(
    name="collection_name",
    configuration={"hnsw": {"space": "cosine"}, "embedding_function": ef}
)
# 4. Add documents
collection.add(documents=texts, metadatas=metadata, ids=ids)
# 5. Perform similarity search
results = collection.query(query_texts=["query"], n_results=5)
# 6. Process results
for i, (doc_id, score, text) in enumerate(zip(results['ids'][0], results['distances'][0], results['documents'][0])):
    print(f"Rank {i+1}: {doc_id}, Score: {score:.4f}, Text: {text}")
```

Author(s)

Wojciech "Victor" Fulmyk

Skills Network