Vector Databases : From Embedding to Applications

Introduction

SHORTCOMING OF LLM

A trained language model doesnot have any knowledge about recent event or data it is not trained on.

RAGs solves this problem and vector databases are key component of RAGs

→ How does it solves the problem?

- proprietary or recent data is first stored in the vector database.
- When system recieves a query, it activates its retrival component. it searches in the vector database to find relevant info and retrieves its.
- Finally, this retrieved text can be passed to give a context.

Application of vector database

1. RAGs
2. Semantic Search Application
3. Recommender Systems.

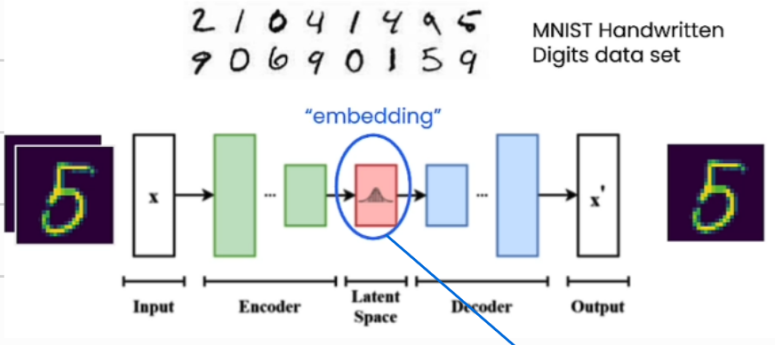
Outline

- How to obtain vector representations of data?
  - Embeddings
- Searching for Similar Vectors
  - Distance Metrics
- Approximate Nearest Neighbors
  - ANN - Trade recall for accuracy
  - HNSW
- Vector DB's
  - CRUD operations
  - Objects + Vectors
  - Inverted Index - filtered search
- Sparse vs Dense Search
  - ANN search over Dense embeddings
  - Sparse search
  - Hybrid Search
- Applications of Vector DBs in Industry

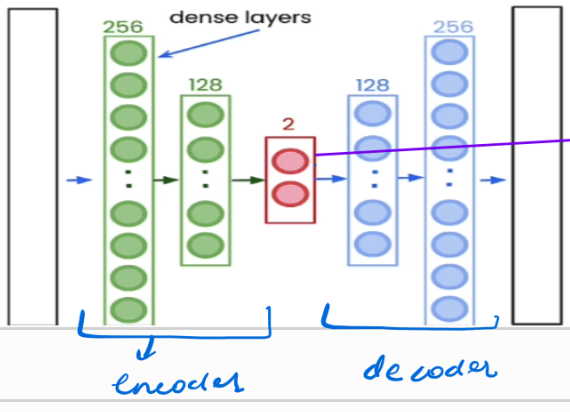
How Neural Networks can be used to embed data into numbers ?

One Example : Auto encoders
- can convert images into vector embeddings.

Creating Embeddings



Once model is trained, the o/p can be generated from embedding only. The vector embedding contains the meaning of the image.

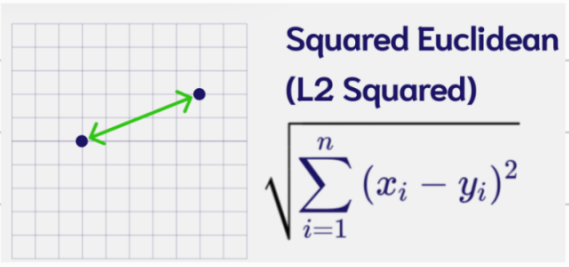


dimension of vector embedding is usually large. Typically 100 - 1000 dim.

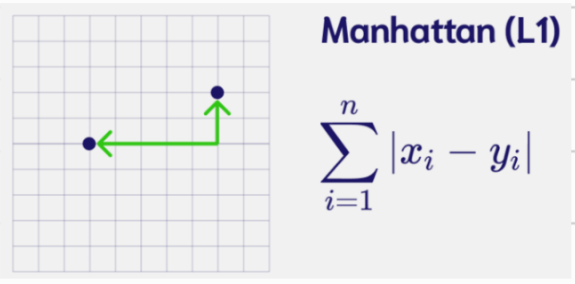
How can we measure the distance b/w embedding?

There are many ways to calculate distance b/w two vector.

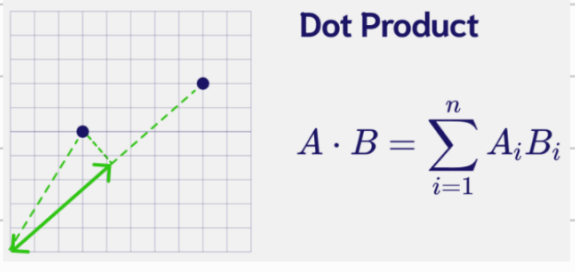
1. Euclidean distance (L2)



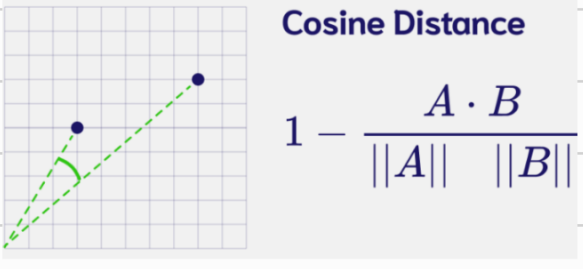
2. Manhattan distance (L1)



3. Dot Product



4. Cosine Similarity

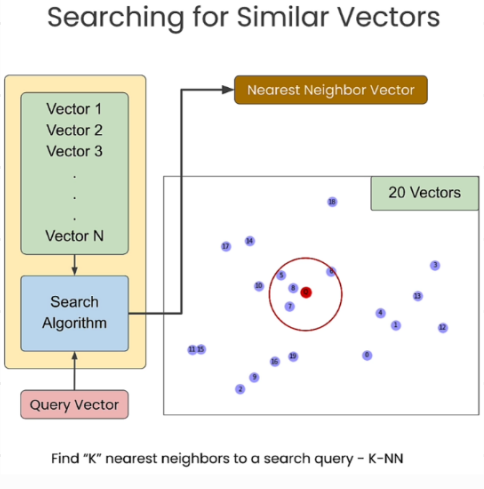


Search For Similar Vectors

vectors / embedding captures the meaning behind our data

↓

To find data points similar in meaning to our query we



can search and Retrieve the closest object to our query in the vector space and return them

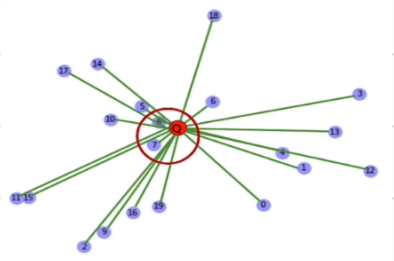
← Vector / Semantic Search

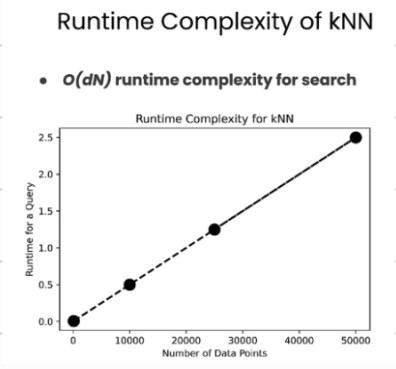
BRUTE FORCE SEARCH

Step 1 : Compute distances btw query vector and all other vectors

Step 2 : Sort the dist

Step 3 : Return top K closest objects KNN





Drawback of Brute force approach

- High computational cost
- scalability issues : size ↑ Query time ↑

query time ∝ size

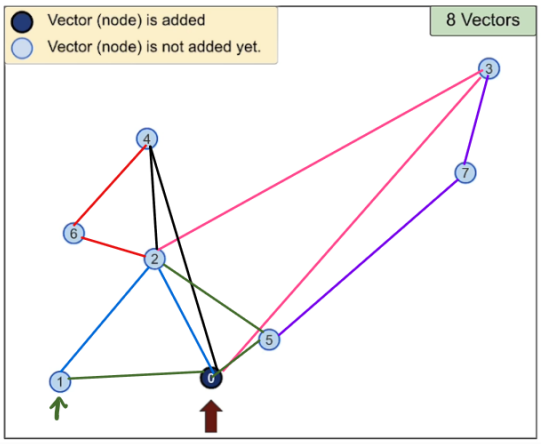
Solution ↓

Approximate Nearest Neighbours

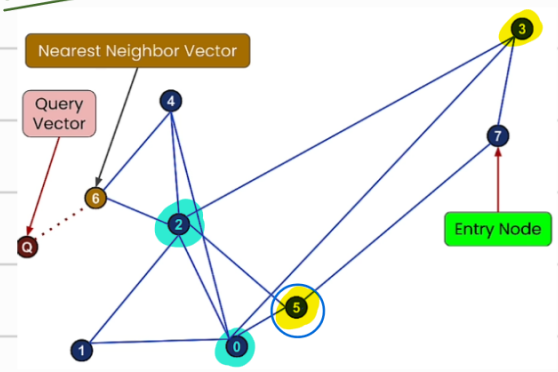
→ at the core of vector databases

- Here, we trade a little bit of accuracy or recall for a lot of performance

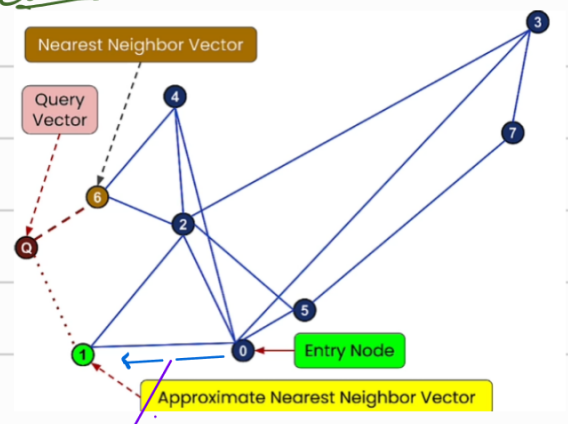
Navigable Small World (NSW)



Case I



Case II



Given: a Query vector Q

- start with a random entry node and try to move across the nearest neighbour.

The best possible movement for this entry Node

↳ Not the best Nearest Nbr, but approx Nearest Nbr. good but not perfect

Starting from, let's say, node 7 (7 is connected to 3 & 5), 5 is closer to Q than 7, move to 5

Now, 5 is connected to 0 and 2 and 2 is way closer move to 2.

Then, move from 2 ⟶ 6
6 is nearest nbr.

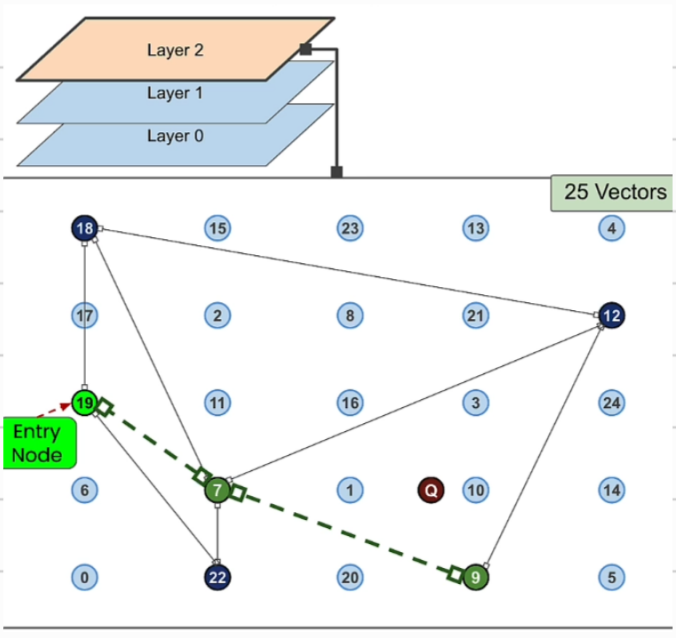
Note: Since it is a greedy algorithm, then from where you start does matters.

Hierarchical Navigable Small world

putting several layer of Navigable Small world on top of each other.

↓

Construction of each layer of HNSW is same as that of NSW.



Working

- choose a random node, (you can choose only from those at highest level.)

→ Move to node nearest to Q with in that level (like in NSW)

Find best possible match at next layer

→ keep moving until you reach bottom layer.

Once you are at bottom layer. we can go to object that is closest to Q.

TC → $O(\log n)$

Vector Databases

Vector Databases:

- **Definition:** Databases optimized for storing and querying data represented as vectors (arrays of numbers), typically used in AI, ML, and deep learning tasks.
- **Purpose:**
  - Efficiently handle high-dimensional data (like embeddings from NLP or image models).
  - Speed up similarity searches, clustering, and recommendations.
- **Use Cases:**
  - **Text embeddings** (e.g., document search, chatbots)
  - **Image embeddings** (e.g., image similarity searches)
  - **Audio embeddings** (e.g., voice recognition)
- **Key Concepts:**
  - **Vector Representation:** Convert data (text, images, etc.) into a fixed-size vector.
  - **Similarity Search:** Finding vectors closest to a query vector using metrics like cosine similarity or Euclidean distance.
- **Common Algorithms:**
  - **Approximate Nearest Neighbor (ANN):** Accelerates similarity searches in large datasets (e.g., FAISS, HNSW).

- **Popular Vector Databases:**
  - **Pinecone:** Fully managed database designed for high-scale vector search.
  - **Weaviate:** Open-source, supports semantic search and knowledge graphs.
  - **Milvus:** Optimized for AI applications, handles billion-scale vector data.
- **Benefits:**
  - Faster search in high-dimensional spaces.
  - Can handle dynamic, unstructured data.
  - Supports real-time updates for ML pipelines.
- **Challenges:**
  - High computational and memory demands.
  - Indexing strategies can be complex.

Common Applications:

- **Search Engines:** Using vector embeddings for semantic search instead of keyword matching.
- **Recommendation Systems:** Matching users with similar interests using vector-based profiles.

Sparse Search	Dense Search
- uses BOW techinques etc since most of element in vector are zero, Hence Spase.	- uses vector embedding rep of data to perform search.
- Eg BM25 Search	- capture and return semantically similar object
- good for exact search and easy to interpret	- poor performance where you need to do exact search like searching a SNO : BN2113CABB or when context is not in there in embedding training data!
- lack semantic meaning or context beyond exact search.	

Aspect	Dense Search	Sparse Search
Vector Type	Low-dimensional, dense, continuous vectors	High-dimensional, sparse, mostly zero vectors
Techniques	Embeddings (BERT, Word2Vec, etc.)	TF-IDF, Bag of Words, BM25
Semantic Ability	Captures context and meaning	Exact word matching, lacks semantics
Similarity Metric	Cosine similarity, Euclidean distance	Dot product, Jaccard similarity
Efficiency	More computationally expensive, but more meaningful	Efficient for large corpora, but limited in flexibility
Use Cases	Semantic search, recommendation systems, chatbots	Keyword search, traditional document retrieval

Hybrid Search

- combines sparse and dense search
- delivers more relevant and accurate results.

How Hybrid Search Works:

- **Parallel Search**: The system performs **both dense and sparse searches simultaneously**. Results from both methods are then ranked and combined.
- **Re-Ranking**: Results from sparse search (keyword matches) and dense search (semantic matches) are merged, and a scoring system is used to prioritize the most relevant results.
- **Custom Weighting**: Hybrid systems can assign different weights to sparse and dense results depending on the query type (e.g., more weight on dense for ambiguous queries, more weight on sparse for specific keyword searches).

Example of Hybrid Search in Action:

For the query "best affordable phone with good battery life":

- **Sparse Search** would look for documents containing the words "best", "affordable", "phone", and "battery life".
- **Dense Search** would find documents that discuss phones with **similar features** (long battery life, affordable price) even if they don't contain the exact query terms.
- **Hybrid Search** combines these results, re-ranks them, and provides the most relevant answer.