Cairo University
Faculty of Engineering
CMP 4030

Advanced Database

Semantic Search Engine with Vectorized Databases

Name	section	bench number
Nesma Abd-ElKader Shafie	2	28
Yousef Osama Mohamed Tawfiq	2	32
Iman Ibrahim Mohamed	1	14
Sara Gamal Gerges	1	20

Efficient Nearest Neighbor Search Using Clustering with LSH and HNSW

Steps:

1. **Vector Clustering**:
    - initially applies clustering (such as k-means or hierarchical clustering) to group vectors into clusters based on similarity.
    - Each cluster contains vectors closer to each other than vectors in other clusters. This way, a new query only needs to compare with vectors in the closest clusters rather than all vectors in the database.
    - Each cluster is assigned a unique identifier (cluster ID) and its centroid is calculated.
2. **Building the Inverted Index Across Clusters**:
    - Build an inverted index in which each unique cluster ID is associated with a list of vectors that fall within that cluster.
3. **Using the Inverted Index** :
    - When searching compare the search vector with clusters centroids only.
    - Instead of searching through all clusters, restrict the search to only the clusters identified by the inverted index, reducing the search space.
4. **Search inside the Cluster Using One of the Following LHS or HNSW**

Locality-Sensitive Hashing (LSH)

Overview

Locality-Sensitive Hashing (LSH) is an efficient technique for approximate nearest neighbor search in high-dimensional spaces. It is particularly useful in applications like semantic search, where we want to find vectors (representing text, images, etc.) that are similar to a given query vector. LSH works by hashing input items in a way that similar items are more likely to end up in the same hash bucket.

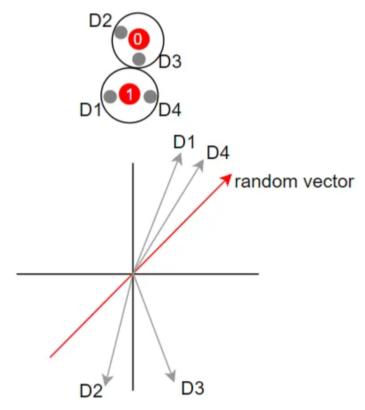
How LSH Works

1. **Generate Random Vectors (Hyperplanes)**: Create random vectors from the same high-dimensional space, each representing a random hyperplane.
2. **Projection and Thresholding Using Cosine Similarity**: For each random hyperplane, calculate the dot product with the vector being hashed.
    - If the dot product is above a threshold (often zero), output `1`.
    - If below the threshold, output `0`.
3. **Apply Multiple Hash Functions**: Repeat with multiple hyperplanes (or random vectors) to form a binary hash code for each item. Similar vectors tend to get similar binary codes.

Using Hash Codes for Bucket Assignment

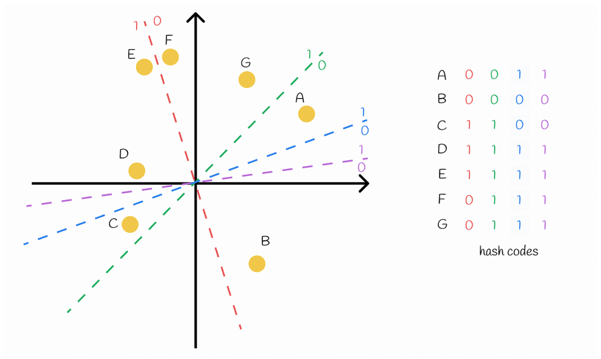
- Each unique binary hash code represents a "bucket" in the LSH table.
- During a query, the hash code of the query vector is calculated, and only the vectors in matching buckets are retrieved, narrowing down potential matches.

Examples



HASH	DOCUMENTS
1	D1, D4
0	D2, D3

LSH using one random vector



A	0	0	1	1
B	0	0	0	0
C	1	1	0	0
D	1	1	1	1
E	1	1	1	1
F	0	1	1	1
G	0	1	1	1

LSH using four random vectors

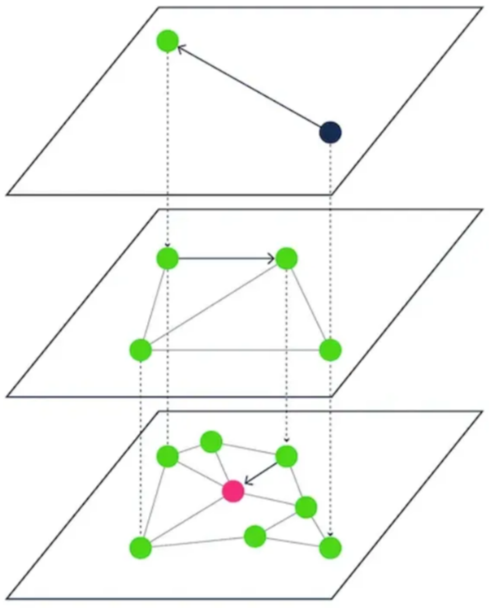
Hierarchical Navigable Small World (HNSW)

overview

Hierarchical Navigable Small World (HNSW) is a graph-based algorithm designed for efficient approximate nearest neighbor search in high-dimensional spaces.

How it works

1. **Graph Construction**:
    - Each vector is added to a multi-level graph with a random highest level.
    - Vectors in higher levels are sparsely connected; lower levels are denser.
2. **Connecting Neighbors**:
    - For each vector, only the closest vectors within each level are retained as neighbors, with a maximum set by the parameter *M*.



HNSW visualization

3. **Search Process**:
    - **Starting Point**: A search begins at the topmost level of the graph with a random entry point. From there, it performs a greedy search within the

sparse connections at that level, moving closer to the query vector.

- **Layer-by-Layer Navigation**: Once the closest point is found at the current layer, the search continues downward to the next, denser layer.
- **Greedy Search in the Bottom Layer**: In the final, densest layer, the algorithm performs a precise nearest neighbor search using the close neighbors from the previous layers as a starting point. This leads to finding the approximate nearest neighbors to the query vector.

4. **Parameters for Tuning**:
    - **M(Max neighbors)** controls the number of neighbors per node (affects memory and accuracy).
    - **ef (Exploration Factor)**: A parameter that controls the search breadth. It determines how many nodes the algorithm examines at each layer during the search.

Notes

we considered using only HNSW but according to the FAISS documentation the memory consumption, in bytes per vector, for an HNSW index is `(d * 4 + M * 2 * 4)` where `d` is the dimensionality of the indexed vectors and `M` is the number of edges per node in the constructed graph (a default value of 32 is typically used).

so using this calculation we would have broken ram constraints so that's why we considered using HNSW after clustering to decrease ram usage.

The number of clusters and which algorithm will be used(LHS or HNSW) will be tuned according to the size of the database.

References

- https://medium.com/@david.gutsch0/vector-databases-understanding-the-algorithm-part-3-bc7a8926f27c
- https://stackoverflow.com/questions/77401874/how-to-calculate-amount-of-ram-required-for-serving-x-n-dimensional-vectors-with#:~:text=According to the FAISS documentation,of 32 is typically used).
- https://lantern.dev/blog/calculator
- https://towardsdatascience.com/similarity-search-part-6-random-projections-with-lsh-forest-f2e9b31dcc47