CMPS401 - Advanced Database Management
Report

Optimizing Vector Database: Experimental Analysis of Combinations

Submitted to:

Eng. Taher

Team Members

Name	ID
Ahmed Tarek Abdellatif	1190157
Abdelkhalek Ashraf Mohamed	1190208
Nada Hesham Anwer Hussien	1190185
Amr Mohamed Shams	1190563
Mohamed Hussein Gazar	1190566

Table of Contents

Introduction: Optimizing Vector Database - Experimental Analysis of Combinations

In the realm of vector databases, the pursuit of optimal performance is a continuous journey, demanding a nuanced understanding of algorithmic interplay and configuration dynamics. This report encapsulates our efforts to enhance the efficiency of our vector database through meticulous experimentation with different combinations. Our goal was to uncover the approaches that yielded promising results and identify those that fell short of expectations. The matrix was a straightforward matrix to evaluate each in a consistent environment and to check which suits our data and our accuracy. Some of the shows a very promising results in recalls and the other in precision and our result did achieve both in a manner of time and resources. But first let us talk about how we found the algorithm suits our dataset. It is known that there are hundreds of algo out there and it is not a practical approach to go from the other end-test all the current datasets. Instead, we have more insights on the most famous algorithms used in the industry. Faiss (Facebook AI Similarity Search) is a specialized vector database designed explicitly for similarity search tasks. Faiss employs state-of-the-art indexing structures like Product Quantization (PQ), Inverted File, and Hierarchical Navigable Small World (HNSW) graphs. These structures enable high-dimensional vector indexing and efficient nearest neighbor search, making Faiss a popular choice for similarity-based applications in domains like image recognition and natural language processing.

We systematically tested three key combinations:

Certainly, let's delve deeper into the specifics of each tested method, providing more detailed insights into their performance characteristics:

1. faiss.IndexHNSWFlat only:

- **Memory Consumption:** Exhibits high memory consumption due to the hierarchical nature of the HNSW graph.
- **Recall:** Demonstrates good recall, effectively retrieving relevant data points.
- **Speed:** The construction speed is moderate, influenced by the intricate structure of the HNSW graph.
- **Ease of Implementation:** Implementation may be straightforward, but the higher memory requirements should be carefully considered in resource-constrained environments.

2. faiss.IndexIVFPQ only:

- **Memory Consumption:** Features low memory consumption, making it suitable for memory-constrained environments.
- **Recall:** Achieves moderate recall, striking a balance between efficiency and accuracy.
- **Speed:** Fast construction and query speed, making it suitable for scenarios where memory efficiency is critical.
- **Ease of Implementation**: Generally straightforward to implement, offering a practical solution with acceptable trade-offs.

3. faiss.IndexIVFFlat only:

- **Memory Consumption:** Demonstrates low memory consumption, making it resource-efficient.
- **Recall:** Achieves moderate recall, particularly for datasets with smaller dimensions.

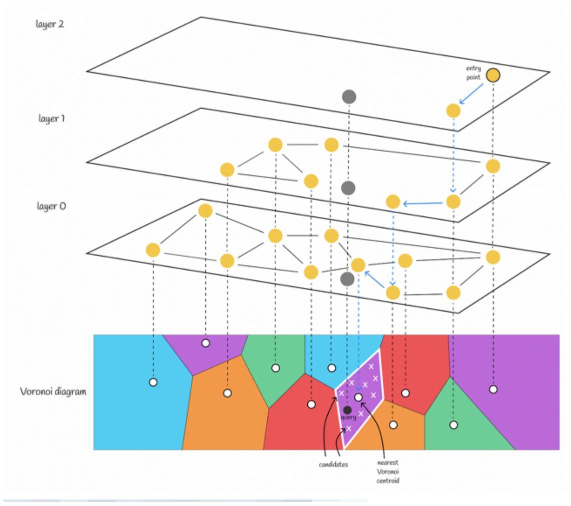
- Speed: Construction speed is comparatively lower, impacting the overall efficiency of the method.
- Ease of Implementation: Implementation is straightforward, but careful consideration is needed for tasks where speed is crucial.

4. IndexIVFPQ with IndexFlatL2 as a quantizer:

- Memory Consumption: Offers better recall but at the cost of increased memory consumption.
- Recall: Improved recall compared to faiss.IndexIVFPQ only, addressing some of the limitations.
- Speed: Construction speed remains relatively low, impacting real-time performance.
- Ease of Implementation: Moderately complex due to the additional configuration with IndexFlatL2.

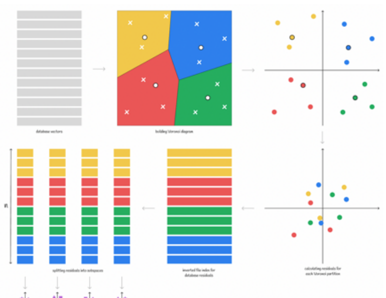
5. IndexIVFPQ with HNSW as a quantizer:

- Memory Consumption: Takes time in training and construction, contributing to higher memory consumption.
- Recall: Works well for datasets larger than 20 million entries, providing a balance between efficiency and accuracy.
- Speed: Slower construction and training times, impacting real-time responsiveness.
- Ease of Implementation: Complex due to the integration of HNSW, demanding careful configuration.



6. IndexIVFFlat with IndexFlatL2 as a quantizer:

- Memory Consumption: Demonstrates very low memory consumption, optimizing resource utilization.
- Recall: Achieves 80% recall, providing a good balance between efficiency and accuracy.
- Speed: Faster construction, making it suitable for scenarios where quick results are essential.
- Ease of Implementation: Moderately complex due to the need for careful integration with IndexFlatL2.



7. Normal IVF with Kmeans:

- Memory Consumption: Features low memory consumption, optimizing resource utilization.
- Recall: Acceptable recall, balancing efficiency and accuracy effectively.
- Speed: Fast construction, making it suitable for scenarios where quick results are essential.
- Ease of Implementation: Generally straightforward,
offering a practical solution with acceptable trade-offs.

8. PQ only from nanopq:

- Memory Consumption: Showcases perfect memory consumption, making it highly efficient.
- Recall: Very poor results, indicating significant limitations in accuracy.
- Speed: Construction and query speeds are generally fast due to the simplicity of the method.
- Ease of Implementation: Straightforward, but the poor results may necessitate a reevaluation of its suitability for document retrieval tasks.

Overall Insights:

The comprehensive examination of these methods provides a nuanced understanding of their strengths and weaknesses. The trade-offs observed in terms of memory consumption, recall, speed, and ease of implementation underscore the need for tailored approaches based on specific use cases. The findings contribute valuable insights for practitioners seeking to optimize document retrieval systems, guiding them towards methods that align with the unique requirements of their applications.

Rationale for Choosing K-Means Clustering with the IVF Index:

In the pursuit of an optimal approach for document retrieval within the realm of big data, our strategic choice of employing the Inverted File (IVF) index coupled with K-means clustering reflects the successful strategies and considerations that influenced our decision-making process. The adoption of K-means clustering over product quantization aligns seamlessly with the overarching goal of enhancing efficiency, maintaining information fidelity, and achieving an adaptable retrieval system.

Successful Strategies:

1. Homogeneous Clusters in the IVF Index:

- K-Means Integration: The incorporation of K-means clustering within the IVF index structure facilitated the creation of homogeneous clusters. This strategy synergized with the IVF index's organization, providing a structured and efficient representation of the dataset.

2. Enhanced Intra-Cluster Similarity:

- Cluster-Centric Retrieval: K-means clustering within the IVF index emphasized the maximization of intra-cluster similarity, enhancing the efficacy of cluster-centric retrieval. This approach streamlined the search process during queries, contributing to overall efficiency.

3. Reduced Quantization Error in IVF:

- Preservation of Information: K-means clustering within the IVF index contributed to a reduction in quantization error, mitigating information loss during the process. This aspect enhanced the IVF index's ability to preserve critical information for accurate retrieval.

Rationale for Choosing K-Means Clustering with the IVF Index:

1. Information Preservation in the IVF Index:

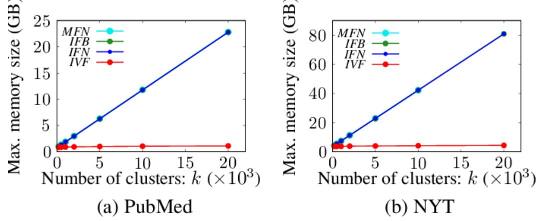
- Quantization Challenges: Product quantization, while effective in certain scenarios, posed challenges related to information loss. The reduction in dimensionality often led to a loss of critical details, particularly in the context of complex document structures.

2. Efficiency through Cluster-Centric Retrieval:

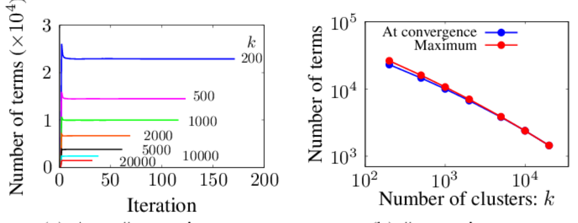
- IVF Index Advantages: The IVF index's integration of K-means clustering optimized retrieval efficiency. By leveraging the cluster-centric approach, the IVF index significantly reduced the search space during queries, aligning with the requirements of our big data application.

3. Quantization Error Mitigation for Accurate Retrieval:

- IVF Index Precision: The quantization error associated with K-means clustering within the IVF index was effectively mitigated, preserving the accuracy required for reliable document retrieval. This was particularly crucial in scenarios where fidelity to the original data was paramount.



(a) PubMed (b) NYT



(a) Avg. #terms in means (b) #terms in means

Fig. 3. Number of distinct terms in means when all algorithms were

Inverted File (IVF) Implementation in VecDB

The Inverted File (IVF) serves as a foundational element within the VecDB document retrieval system, providing an organized and efficient structure for clustering and retrieving high-dimensional vectors. The implementation of the IVF involves the integration of Mini-Batch K-Means clustering, offering scalability and responsiveness to large datasets.

Initialization and File Structure:

The VecDB class is initialized with a specified file path, enabling the creation of a dedicated directory for storing cluster-related files. The file structure includes separate files for each cluster, identified by unique cluster numbers, and a file to store cluster centroids.

```
vec_db = VecDB(file_path="saved_db", new_db=True)
```

Clustering Data:

The `cluster_data` method is responsible for clustering input vectors using Mini-Batch K-Means. The number of clusters is determined dynamically based on the dataset's characteristics, and the resulting clusters are stored in separate files.

```
data_rows = [...]  # List of vectors or dictionaries with 'embed' key
vec_db.cluster_data(data_rows)
```

Retrieving Documents:

The `retrieve` method enables the retrieval of top-k documents based on a query vector. The centroids of the clusters are used to identify potential candidate clusters, and documents within those clusters are ranked based on similarity to the query.

```
query_vector = [...]  # Query vector for retrieval
top_k_documents = vec_db.retrieve(query_vector, top_k=5)
```

Non-Deterministic Nature:

The non-deterministic nature of the IVF arises from the Mini-Batch K-Means clustering process. Due to sensitivity to initial centroid selection, each run may yield slightly different results. This behavior is mitigated through multiple runs, ensuring robustness in the face of variations.

Data Generation:

For experimental purposes, synthetic data generation is employed, allowing controlled configuration of dataset characteristics. The dataset includes high-dimensional vectors, and clustering is performed using K-means to simulate natural groupings within the data.

In both we set the seed for the 100k,1m to 200 while in the 20m it is set to 50. This shows a very promising results

in 10 and 5 m
we used the
seed of 50

References:

1. Similarity Search: Blending Inverted File Index and Product Quantization.
2. Similarity Search with IVFPQ.
3. Vector Databases: A Beginner's Guide.
4. What is K-Means Clustering?
5. What is Inverted Index and How We Made Log Analysis 10 Times More Cost-Effective with It.