Unlocking the Power of Databases: The Crucial Role of Theory and Indices in Scalable Vector Databases for Machine Learning

Sharan Sahu

Cornell University
ss4329@cornell.edu

August 28th, 2024

Overview

Introduction

- I am first-year PhD student in Statistics and Machine Learning at Cornell University. Before joining Cornell University, I was an undergraduate at UC Berkeley where I studied Computer Science.
- I am a DoD SMART Scholar and have done internships at Marine Corps Tactical Systems Support Activity (MCTSSA) in Software Engineering and Data Science.

Introduction

- Databases are fundamental pieces of software that allows us to store, manipulate, and retrieve data quickly and efficiently.
- There are many types of databases: **Relational** (MySQL, Oracle, PostgreSQL), **NoSQL** (MongoDB, Cassandra, Redis), **Cloud** (AWS RDS, GCP SQL, Azure SQL), and **Vector** (FAISS, Milvus, Pinecone).
- All these databases and their variants have important use cases and properties, but **vector databases** are dominating the field of machine learning.

Example Schema

SID	Name	Age	Major	GPA
1001	Alice Johnson	20	Computer Science	3.8
1002	Bob Smith	22	Mathematics	3.5
1003	Carol White	19	Biology	3.9
1004	David Lee	21	Physics	3.7
1005	Eva Green	20	Chemistry	3.6
⋮	⋮	⋮	⋮	⋮
41001	Bob Myers	37	Art History	2.1

Table: Student Records (With Millions of Rows)

SQL Query Examples

Filter People Based on Age Greater Than 30:

```
SELECT *
FROM Students
WHERE Age > 30;
```

Filter People Based on Age Greater Than 30 and GPA Less Than 2.5:

```
SELECT *
FROM Students
WHERE Age > 30 AND GPA < 2.5;
```

Naive Way of Filtering

Why Linear Filtering Without an Index or Sorting is Inefficient:

- We want to find students who satisfy specific conditions, e.g., age greater than 30 and GPA less than 2.5.
- **Problem:** Only a few thousands students in the entire database meet these criteria.
- **Linear Search:** Without indexing or sorting, the database engine must scan every single entry in the table.
- **Cost:** This means checking all 40K+ entries, which is time-consuming and computationally expensive.
- **Inefficiency:** The cost of scanning 40K+ records just to find a few thousand relevant results is inefficient and will take a long time.

Better Way of Filtering

- **Imagine a Textbook:**
  - When reading a textbook about databases, if you want to find a chapter or page about a particular topic, you don't scan the entire book to find the content you are interested in
  - Instead, you use a table of contents or Appendix to quickly find which page contains the content you want.
- **Applying This to Databases:**
  - Instead of scanning every entry in the 'Students' table, an "table of contents" allows the database to jump directly to the relevant entries.
  - For example, if we create an "table of contents" on the 'Age' and 'GPA' columns, the database can efficiently locate students who are older than 30 and have a GPA less than 2.5.
- **Efficiency:**
  - Using a "table of contents", the number of comparisons drops from 1,000,000 to just a few dozen, saving significant time and computational resources.

Indices: The Solution For Fast Lookups

An **index** is a data structure that enables fast **lookup** and **modification** of **data entries** by **search key**

- **Data Entries**: items stored in the index
- **Modification**: want to support fast insert and delete

B+ Trees

- **B+ Trees**: A type of self-balancing tree structure used for indexing in databases.
- **Usage:**
    - Primarily used for creating indexes on columns where efficient retrieval is critical where you need to perform equality and range queries on columns.
    - Commonly used in relational databases to speed up search operations, such as ‘WHERE‘ clauses.
- **Benefits:**
    - Supports efficient insertion, deletion, range, and lookup operations.
    - Disk-based B+ Trees keep the data organized in a way that minimizes disk I/O operations, crucial for large-scale data storage.
- **Limitations:**
    - Inefficient for multi-dimensional queries (e.g., finding students based on a combination of Age and GPA) without using composite indexes.
    - Not ideal for unstructured or high-dimensional data, such as text or image data, where more advanced data structures like R-trees or hash-based indexing might be preferable.

B+ Tree Example: Indexing by Age

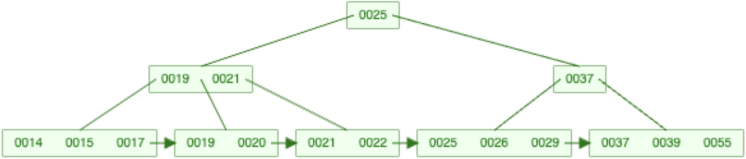


Figure: B+-Tree Construction Indexing By Age

B+ Tree Example: Indexing by GPA

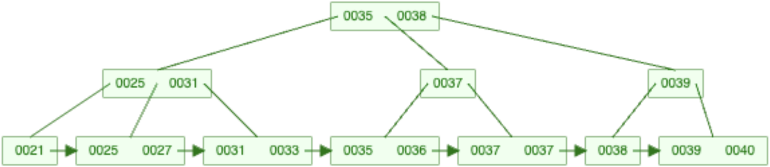


Figure: B+-Tree Construction Indexing By GPA

K-d Trees

- **K-d Trees**: A binary tree structure used for organizing points in a k-dimensional space.
- **Usage:**
  - Primarily used for spatial searches involving multi-dimensional data, such as range searches and nearest neighbor searches.
  - Commonly used in applications like machine learning for tasks such as clustering and data retrieval.
- **Benefits:**
  - Efficiently handles multi-dimensional data, making it suitable for complex queries involving multiple attributes (e.g., location-based searches).
  - Provides fast nearest neighbor searches, which are crucial in information retrieval and large language models.
- **Limitations:**
  - As the number of dimensions (k) increases, the efficiency of K-d Trees decreases, a phenomenon known as the "curse of dimensionality."
  - A K-d tree may become unbalanced with frequent updates, leading to inefficient searches.

K-d Tree Example: Indexing by Age and GPA

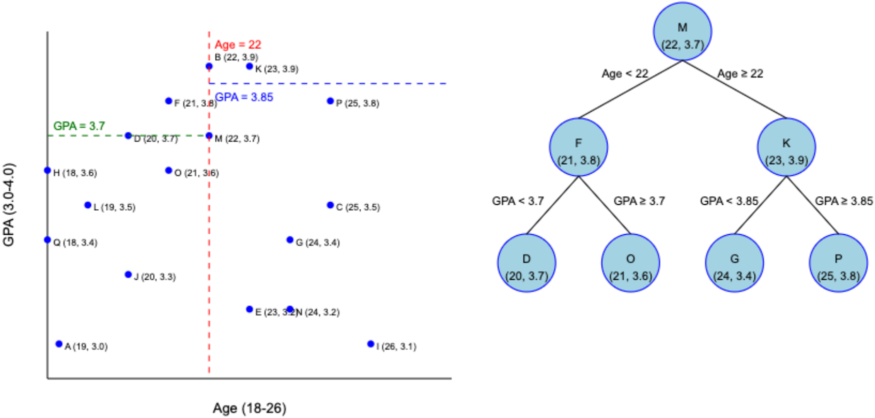
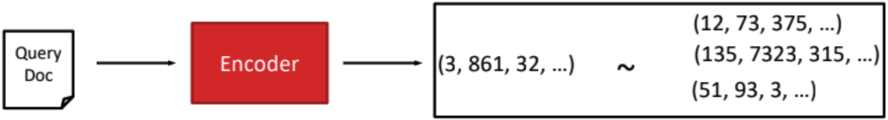


Figure: K-d Tree Construction Indexing By Age and GPA

Vector Databases

- Problem: from document corpus, find the one most similar to a "query doc"
  - Docs = news articles, query doc = a new article
  - Docs = code, query doc = buggy program
- We can place these documents into a database that has a bunch of embeddings and then use a k-d tree for doing similarity search.
- This is a vector database!



Similar search on embeddings!

Vector Databases

- **Vector Databases**: Specialized databases optimized for storing and querying high-dimensional vectors. These databases are crucial in modern applications where data is represented in vector form.
  - **Storage and Retrieval**: Vectors can represent text embeddings, image features, or user preferences, and are stored in such a way that allows for efficient retrieval based on similarity or distance metrics (e.g., cosine similarity, Euclidean distance).
- **Applications**:
  - **Recommendation Systems**: Leveraging user behavior vectors to recommend products, movies, or music based on similarity to other users or items.
  - **Image Retrieval**: Searching large image databases by comparing the vector representations of query images with those in the database to find similar images.
  - **Text Search**: Utilizing text embeddings to perform semantic searches, where results are based on meaning rather than keyword matching.

RAG and Vector Databases

- **RAG**: Combines large language models (LLMs) with vector databases for enhanced information retrieval.
- Vector databases provide efficient storage and retrieval of high-dimensional data, crucial for the real-time context-aware responses in AI applications.
- Together, they enable more accurate and relevant responses by aligning LLM outputs with the most relevant contextual information retrieved from vector databases.

Scaling Vector Databases: Moving Away From K-d Trees

- K-d trees are effective for low-dimensional vector data, offering a straightforward method for indexing and searching.
- Ideal for applications with fewer dimensions where exact nearest neighbor searches are feasible.
- As data volume or dimensionality increases, transitioning to more advanced indexing structures is crucial. Shifting from exact nearest neighbor searches to approximate methods enhances scalability and maintains performance in high-dimensional spaces.

Conclusion

- Theoretical understanding of database structures is essential for designing efficient vector databases.
- Practical applications like RAG benefit from indexing techniques like K-d trees, a foundational data structure rooted in database theory.
- Database theory is important and has a wide variety of applications.

Q&A

Questions?