Code7Crusaders

Software Development Team

Analisi per la scelta del DB Vettoriale

Membri del Team:
Enrico Cotti Cottini, Gabriele Di Pietro, Tommaso Diviesti
Francesco Lapenna, Matthew Pan, Eddy Pinarello, Filippo Rizzolo

Data: 15 Febbraio 2025

Indice

Elenco delle tabelle

1 Introduzione

Il seguente documento ha lo scopo di analizzare i database vettoriali e di fornire una valutazione per la scelta del database vettoriale più adatto per il progetto in corso.
Il documento è strutturato in tre sezioni principali:

- **Introduzione ai database vettoriali**: in cui vengono introdotti i database vettoriali e le loro caratteristiche principali.
- **Confronto**: In maniera tabellare vengono confrontati i database vettoriali analizzati.
- **Conclusione e Motivazione della Scelta**: in cui viene presentata la scelta del database vettoriale più adatto per il progetto in corso.

2 Introduzione ai database vettoriali

2.1 FAISS (Facebook AI Similarity Search)

FAISS è una libreria sviluppata da Facebook AI per la ricerca di similarità tra vettori ad alta dimensione. È ottimizzata per operazioni di nearest neighbor search su dataset di grandi dimensioni ed è ampiamente utilizzata in applicazioni di machine learning e intelligenza artificiale.

Pro:

- Alta velocità: ottimizzato per operazioni di nearest neighbor search su dataset di grandi dimensioni.
- Supporto GPU: usa CUDA per accelerare la ricerca su hardware Nvidia.
- Ottimizzazioni avanzate: supporta tecniche come IVF (Inverted File Index), PQ (Product Quantization), e HNSW (Hierarchical Navigable Small World).

Contro:

- Nessuna persistenza nativa: gli indici devono essere salvati e caricati manualmente.
- Uso elevato di memoria: può richiedere molta RAM, specialmente senza compressione.
- Configurazione complessa: necessita di tuning per ottimizzare velocità, accuratezza e memoria.

Risorse: GitHub | Documentazione

2.2 Annoy (Approximate Nearest Neighbors Oh Yeah)

Annoy è una libreria sviluppata da Spotify per la ricerca approssimata del nearest neighbor. È progettata per essere leggera e facile da usare, rendendola ideale per applicazioni su dispositivi con risorse limitate.

Pro:

- Leggero e facile da usare: API minimale.
- Basso consumo di memoria: ottimizzato per la RAM.
- Salvataggio dell'indice su file: permette di memorizzare e ricaricare facilmente gli indici.

Contro:

- Meno ottimizzazioni rispetto a FAISS.
- Più lento su dataset grandi.
- Solo CPU: non ha accelerazione GPU.

Risorse: GitHub

2.3 Milvus

Milvus è un database scalabile progettato specificamente per la ricerca vettoriale. Supporta architetture distribuite ed è adatto a scenari in cui sono richieste elevate prestazioni e persistenza dei dati.

Pro:

- Database scalabile per milioni di vettori con query distribuite.
- Persistenza nativa: gli indici vengono salvati automaticamente.
- Supporto a diverse tecniche di indicizzazione.

Contro:

- Complesso da configurare: richiede un database backend.
- Overhead di gestione: necessita di infrastruttura server e manutenzione.
- Latenza iniziale più alta.

Risorse: GitHub | Documentazione

2.4 Pinecone (Cloud-Managed Vector Database)

Pinecone è un database vettoriale gestito nel cloud, che offre un'infrastruttura scalabile e facile da integrare per applicazioni che richiedono ricerca di similarità su larga scala senza dover gestire manualmente server e indici.

Pro:

- Completamente gestito nel cloud.
- Facile da integrare tramite API REST.
- Persistenza automatica.

Contro:

- Servizio a pagamento.
- Meno controllo sulle ottimizzazioni.
- Dipendenza dal cloud.

Risorse: Sito ufficiale | Documentazione

3 Confronto

Database	Pro	Contro	Fonti
FAISS	Prestazioni elevate su grandi dataset, Supporto GPU per accelerazione, Ottimizzazioni avanzate (IVF, PQ, HNSW), Open-source e integrabile in Python, Eseguibile localmente	Nessuna persistenza nativa, Alto uso di memoria, Configurazione complessa	NextBrick, Zilliz, Zack Proser, FAISS, GitHub
Annoy	Leggero e facile da usare, Basso consumo di memoria, Possibilità di salvare l'indice su file, Buone prestazioni su dataset medio-piccoli	Meno ottimizzazioni rispetto a FAISS, Più lento su dataset di grandi dimensioni, Supporta solo CPU	GitHub, Zilliz
Milvus	Database scalabile con persistenza nativa, Supporta architettura distribuita, Diversi metodi di indicizzazione, Adatto a dataset di grandi dimensioni	Complesso da configurare, Maggiore overhead di gestione, Latenza iniziale più alta	Zack Proser, Documentazione, GitHub
Pinecone	Completamente gestito nel cloud, Facile da integrare tramite API REST, Persistenza automatica	Servizio a pagamento, Meno controllo sulle ottimizzazioni, Dipendenza dal cloud	Sito ufficiale, Documentazione

Tabella 1: Confronto tra database vettoriali per ricerca di similarità

4 Conclusione e Motivazione della Scelta

Dopo un'attenta analisi delle opzioni disponibili, il database vettoriale scelto per il progetto è **FAISS** (Facebook AI Similarity Search). La scelta è stata motivata dai seguenti fattori principali:

- **Prestazioni elevate e ottimizzazioni avanzate**: FAISS è altamente ottimizzato per la ricerca di similarità tra vettori di grandi dimensioni, grazie a tecniche come IVF (Inverted File Index), PQ (Product Quantization) e HNSW (Hierarchical Navigable Small World). È in grado di gestire milioni di vettori con tempi di ricerca estremamente ridotti.
- **Integrazione con Python e LangChain**: La compatibilità con Python e il supporto all'integrazione con LangChain rendono FAISS una scelta ideale per l'implementazione nel nostro progetto. La sua API è ben documentata e ampiamente utilizzata nella comunità di machine learning.
- **Open-source ed eseguibile localmente**: FAISS è completamente open-source e non richiede una dipendenza dal cloud, garantendo maggiore controllo e flessibilità sull'infrastruttura del progetto. Questo permette di ridurre i costi operativi e aumentare la sicurezza dei dati, evitando la trasmissione a servizi esterni.
- **Scalabilità ed efficienza**: FAISS supporta l'esecuzione su GPU, sfruttando CUDA per accelerare le operazioni di nearest neighbor search. Questo lo rende particolarmente adatto per applicazioni che richiedono una ricerca ad alta velocità su dataset di grandi dimensioni.

- **Ampio supporto e adozione**: FAISS è sviluppato da Meta AI ed è ampiamente utilizzato in produzione da aziende tecnologiche di primo livello, garantendo stabilità, aggiornamenti costanti e un'ampia base di conoscenze disponibile online.

Se in futuro dovesse emergere la necessità di una persistenza nativa per gli indici, **Milvus** potrebbe rappresentare un'opzione valida, grazie alle sue capacità di gestione distribuita e persistenza automatica. Per applicazioni più leggere e meno esigenti in termini di risorse, **Annoy** potrebbe essere considerato un'alternativa interessante. Infine, se si volesse evitare completamente la gestione dell'infrastruttura, **Pinecone** rappresenterebbe una soluzione cloud-managed, sebbene con il compromesso dei costi e della dipendenza da un servizio esterno.