高精度低时延向量检索算法

数据结构课程PJ

2025/11/05

01 向量检索算法背景

- 向量(Vector)是 AI 模型理解、学习和表达数据的基本形式。
- 在 AI 应用系统中，通常会通过特征提取，将文本、图像、视频、用户行为等数据转换成 AI 模型易于理解的特征向量(Embedding)，如词向量 word embedding，图像特征向量 image embedding 等。
- 在实际业务场景中，对向量进行“相似性检索”是一种常用的操作，在大模型 RAG、推荐系统、图像搜索、智能问答、向量数据库、分子结构分析等场景中都有广泛的应用。
- 当前，向量检索已成为 AI 应用系统必备的基础设施之一。

02 向量检索算法定义

向量检索是指从一个实数向量集合：

$$\mathcal{B} = \{v_1, v_2, \dots v_n\} \subseteq \mathbb{R}^{n \times d}$$

检索出与给定查询向量：

$$q \in \mathbb{R}^{1 \times d}$$

最相似的 K 个向量，其中向量集合 $\mathcal{B}$ 被称为向量底库，向量$v_1, v_2, \dots v_n \in \mathcal{B}$被称为底库向量，$n = |\mathcal{B}|$ 被称为底库规模，d 是向量维度，底库向量和查询向量的维度必须相等。

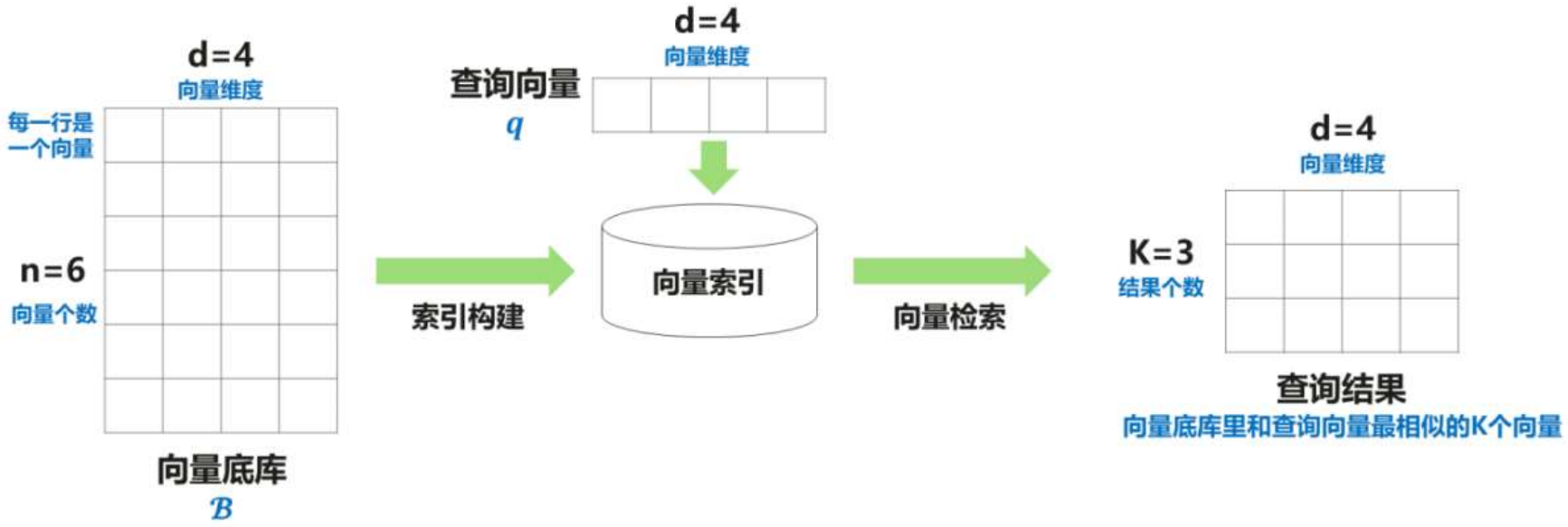
在实际业务场景中，向量之间的相似度通常有以下几种定义：

- L2 相似度：$sim(v, q) = -\|v - q\|_2 = -\sqrt{(v-q)^2}$
- 内积相似度：$sim(v, q) = v \cdot q$
- 余弦相似度：$sim(v, q) = cosine(v, q) = \frac{v \cdot q}{\|v\|_2 \|q\|_2}$

02 向量检索算法定义

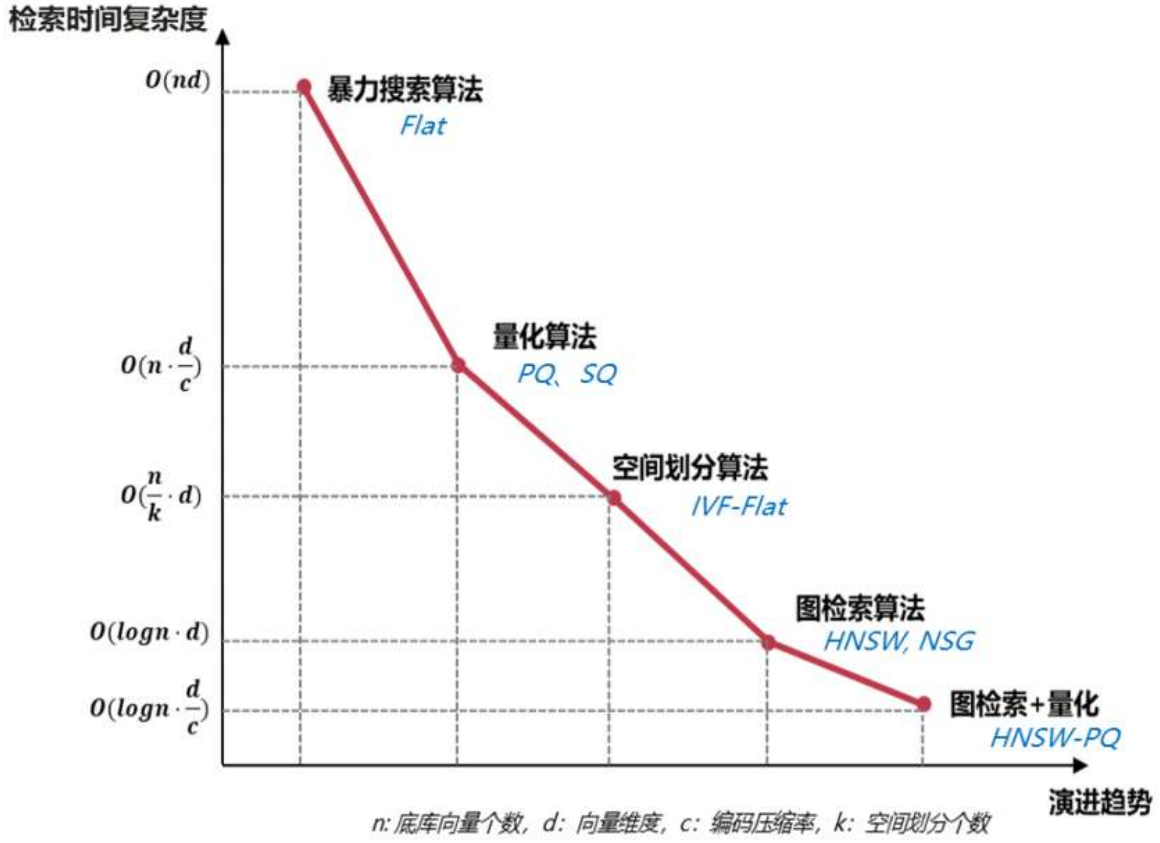
- 在实际业务场景中，如果将查询向量 q和向量底库 $\mathcal{B}$中的向量进行逐一比对，向量检索的计算量会非常大，无法满足客户业务场景的时延要求。

- 因此，业界主流方案是在向量底库上构建一个高性能的向量索引，检索时基于向量索引快速查询到对应的向量，从而能够大幅降低向量检索的计算开销，满足客户业务的时延要求。



03 向量检索主流算法

- 向量检索算法当前有 5 条主流技术路线：暴力搜索算法、量化算法、空间划分算法、图检索算法、图检索+量化融合算法，其中图检索算法较其他类型的算法具有压倒性的性能优势，是当前业界的 SOTA 算法，但图检索算法也存在内存开销大，索引构建耗时长等问题。



04 向量检索算法参考资料

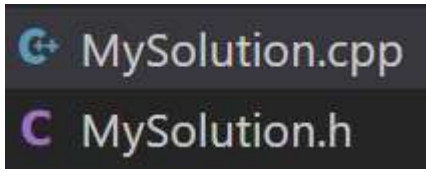
- 向量检索算法是工业界和学术界研究的热点方向，在网上有大量的学习资源。以下论文和算法库可供您参考。

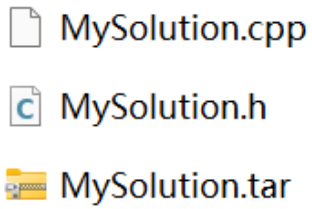
- Wang, Mengzhao, et al. "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search." arXiv preprint arXiv:2101.12631 (2021).
  - https://arxiv.org/abs/2101.12631

- Malkov Y A, Yashunin D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs[J]. IEEE transactions on pattern analysis and machine intelligence, 2018, 42(4): 824-836.
  - https://ieeexplore.ieee.org/document/8594636/

05 题目流程

- 网站： http://10.176.56.208:5000
- 上传代码：上传算法代码，代码需实现 Solution 类以及类里的 build 和search 函数，类和函数实现需要遵循接口定义：
  - a) build 函数：读入向量底库，完成向量索引构建；
  - b) search 函数：基于构建好的向量索引，返回向量检索结果。





Solution 类	
类定义	class Solution
成员函数定义	void build(int d, const vector<float>& base) 输入底库向量 *base* 和向量维度 *d*，完成索引构建 *base* 中的第 *0~d-1* 个元素是第一个向量，第 *d~2d-1* 个元素是第二个向量，以此类推
	void search(const vector<float>& query, int* res) 返回向量底库中距离查询向量 *query* 最近的 10 个向量 ID 其中 *res* 是用来存放结果的数组，长度为 10，内存空间已申请好

05 题目流程

FUDAN UNIVERSITY

MySolution.h

```h
#ifndef CPP_SOLUTION_H
#define CPP_SOLUTION_H

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <cmath>

using namespace std;

class Solution {
public:
    void build(int d, const vector<float>& base);
    void search(const vector<float>& query, int* res);

private:
    int dim;                                    // 向量维度
    vector<vector<float>> base_vectors;         // 存储底库向量
};

#endif // CPP_SOLUTION_H
```

MySolution.cpp

```cpp
#include "MySolution.h"
#include <limits>  // for numeric_limits<float>::max()

void Solution::build(int d, const vector<float>& base) {
    dim = d;
    int N = base.size() / d;
    base_vectors.resize(N);

    for (int i = 0; i < N; ++i) {
        base_vectors[i].assign(base.begin() + i * d, base.begin() + (i + 1) * d);
    }
}

void Solution::search(const vector<float>& query, int* res) {
    vector<pair<float, int>> dist_id;  // <distance, id>

    for (int i = 0; i < (int)base_vectors.size(); ++i) {
        const vector<float>& v = base_vectors[i];
        float dist = 0.0f;
        for (int j = 0; j < dim; ++j) {
            float diff = query[j] - v[j];
            dist += diff * diff;
        }
        dist_id.emplace_back(dist, i);
    }

    // 取前10个最小距离
    int k = min(10, (int)dist_id.size());
    nth_element(dist_id.begin(), dist_id.begin() + k, dist_id.end());
    sort(dist_id.begin(), dist_id.begin() + k);

    for (int i = 0; i < k; ++i) {
        res[i] = dist_id[i].second;
    }
}
```

05 题目流程

- 执行代码：平台执行上传的算法代码，并记录以下关键指标
  - a) 检索时延：即每个查询向量的检索平均时延；
  - b) 检索精度：本赛题将 Top-10 召回率作为检索精度指标；
  - c) build时延：即索引构建时延；
- 返回结果：判题器将记录指标返回给平台网页前端，可视化后呈现

目标：给定包含 n 个向量的底库 $\mathcal{B}$ 和包含 m 个查询向量的集合 Q，设计向量检索算法，从向量底库中检索出和每个查询向量最相似的 10 个向量，目标使向量检索算法的平均时延 T_{avg} 最小。要求算法的检索精度 δ 大于或等于 0.99：

$$obj: min(T_{avg})$$

$$s.t.\ \delta \geq 0.99$$

概念定义：

- **平均时延：**设每个查询向量的检索时延分别是$\{L_1, L_2, \dots L_m\}$，则平均时延为：

$$T_{avg} = \frac{L_1 + L_2 + \cdots + L_m}{m}$$

- **检索精度：**对于查询向量 $q \in Q$，假设选手算法检索出的 10 个底库向量 ID 为 $\{v_1, v_2, \dots v_{10}\}$，而与查询向量 q 最相似的 10 个底库向量 ID（即正确答案）为：$\{v_1^*, v_2^*, \dots v_{10}^*\}$，则检索精度为：

$$\delta = \frac{1}{\mathrm{m}} \sum_{q \in Q} \frac{\{v_1, v_2, \dots v_{10}\} \cap \{v_1^*, v_2^*, \dots v_{10}^*\}}{10}$$

- **向量相似度：**对于任意两个向量 v 和 q，定义其向量相似度为：

$$sim(v, q) = -\|v - q\|_2 = -\sqrt{(v-q)^2}$$

05 题目流程


```
read_data(cas);

auto start = std::chrono::high_resolution_clock::now();
solution_.build(dimension_, base_);
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
```

```
auto start = std::chrono::high_resolution_clock::now();
solution_.search(query_vector, res_);
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
total_time += duration.count();
```

05 题目流程


```
if (cas == 0) {
    filePath = "/home/yanhan/DataStructure/pJudge/data/glove/base.txt";
    pointsnum = 1183514;
    dimension_ = 100;
} else if (cas == 1) {
    filePath = "/home/yanhan/DataStructure/pJudge/data/sift/base.txt";
    pointsnum = 1000000;
    dimension_ = 128;
} else {
    std::cerr << "Invalid dataset case: " << cas << std::endl;
    return;
}

base_.resize(dimension_ * pointsnum);
{
    std::ifstream fin(filePath);
    if (!fin.is_open()) {
        std::cerr << "Error opening " << filePath << std::endl;
        return;
    }
    for (int i = 0; i < dimension_ * pointsnum; ++i) {
        fin >> base_[i];
    }
    fin.close();
}
```

06 任务时间点

- 第九周 11 / 05　发布PJ
- 第十二周 11 / 26　检查第一组数据集SIFT通过情况　　20%（检查完成情况
- 第十四周 12 / 10　检查第二组数据集GLOVE通过情况　　10%（检查算法性能
- 第十五、十六周 12 / 17 ， 12/24　方法分享

07 任务注意点

- 每天只允许提交一次
- 将会检查代码，有投机取巧、卡bug等将可能作零分处理
- PJ最终报告，不鼓励过多，一般在10页左右，过多不会有额外加分但也不会扣分
- 若大模型生成无意义话语，视情况扣分

感谢倾听！