

使用 HNSW 算法优化向量相似性搜索

叶家炜

Apr 08, 2025

在大数据时代，向量相似性搜索已成为推荐系统、图像检索和自然语言处理等场景的核心技术。传统方法如暴力搜索（Brute-force）虽能保证 100% 召回率，但其 $O(N)$ 的时间复杂度难以应对百万级以上数据规模；KD-Tree 和 LSH（Locality-Sensitive Hashing）虽提升了效率，却在高维空间中面临「维度灾难」的挑战。而 **HNSW（Hierarchical Navigable Small World）** 算法通过分层可导航小世界网络结构，在精度与效率之间实现了突破性平衡，成为当前近似最近邻搜索（ANN）领域的热门选择。

1 向量相似性搜索基础

向量相似性搜索的核心目标是从海量数据中找到与查询向量最接近的候选集。例如在自然语言处理中，BERT 模型生成的 768 维词向量可通过余弦相似度衡量语义关联性；在图像检索中，ResNet 提取的特征向量通过欧氏距离比较视觉相似性。近似最近邻搜索（ANN）通过允许微小误差，将时间复杂度从 $O(N)$ 降低至 $O(\log N)$ 级别，但其难点在于如何在精度损失可控的前提下实现高效检索。

2 HNSW 算法原理

2.1 算法背景

HNSW 继承自 NSW（Navigable Small World）算法，后者受小世界网络理论启发，通过构建具备短路径和高聚类系数的图结构实现高效搜索。HNSW 在此基础上引入分层设计，将图结构分解为多个层级，高层用于快速定位区域，底层用于精细化搜索，从而显著降低搜索路径长度。

2.2 核心思想

HNSW 的分层结构可类比于交通网络：高层如同高速公路，稀疏连接但能快速跨越远距离；底层如同城市道路，密集连接以精确抵达目标。算法采用贪心搜索策略，从高层开始逐层向下导航，结合「跳表」机制跳过无关节点。数学上，搜索过程可形式化为：

$$\text{搜索路径} = \arg \min_{v \in V} \sum_{i=1}^L \|q - v_i\|^2$$

其中 q 为查询向量， v_i 为路径上的节点， L 为路径长度。

2.3 算法流程

插入过程动态构建分层图：新节点以概率 $p = 1 / \ln(1 + level)$ 分配到不同层级，高层级连接数较少以保持稀疏性。搜索过程从最高层开始，每层找到局部最近邻后，将其作为下一层的入口点，直至底层完成精确搜索。关键参数包括：

1. M ：控制节点最大连接数，影响内存占用和搜索效率。
2. $efConstruction$ ：构建时的候选池大小，决定索引质量。
3. $efSearch$ ：搜索时的候选池大小，影响召回率。

2.4 算法优势

HNSW 的时间复杂度接近 $O(\log N)$ ，且支持动态数据更新。相比 Faiss 的 IVF-PQ 需要预训练码本，HNSW 无需离线训练即可适应数据分布变化。此外，其对高维数据的鲁棒性显著优于 KD-Tree 等空间划分方法。

3 HNSW 与其他 ANN 算法的对比

在公开数据集 ANN-Benchmarks 的测试中，HNSW 在召回率与查询速度（QPS）的综合表现优于主流算法。例如在 Glove-100 数据集上，HNSW 在 90% 召回率时 QPS 达到 10,000，而 LSH 仅为 3,000。与同为图结构的 NGT 相比，HNSW 的内存占用降低约 30%，因后者需要维护额外的边信息。

4 HNSW 优化实践

4.1 参数调优指南

以电商推荐场景为例，用户画像向量维度通常为 100-300 维。此时建议设置 $M=16$ 以平衡连接密度， $efConstruction=200$ 确保构建质量， $efSearch=100$ 实现 95% 以上召回率。若数据分布极度不均，需适当增大 $efSearch$ 避免陷入局部最优。

4.2 工程实现优化

内存优化可通过标量量化（SQ8）将浮点向量转换为 8 位整型，减少 75% 内存占用。分布式场景下，可采用分片策略将索引划分为多个子图，通过协调节点路由查询请求。以下为使用 hnsplib 库的 Python 示例：

```
1 import hnsplib
2 import numpy as np
3
4 # 初始化索引
5 dim = 128
6 p = hnsplib.Index(space='l2', dim=dim)
7 p.init_index(max_elements=10000, ef_construction=200, M=16)
```

```
9 # 插入数据
data = np.random.randn(10000, dim)
11 p.add_items(data)

13 # 设置搜索参数
p.set_ef(100)
15 labels, distances = p.knn_query(data[:5], k=10)
```

代码中 `space='l2'` 指定欧氏距离，`init_index` 定义初始容量和构建参数，`set_ef` 控制搜索时的候选池大小。实际部署时需监控 `max_elements` 避免溢出，可通过动态扩容接口解决。

4.3 实际应用案例

某头部电商平台采用 HNSW 优化推荐系统，将用户实时行为向量（点击、加购）与商品特征向量匹配，响应时间从 50ms 降至 8ms，推荐转化率提升 12%。另一案例中，医学影像系统通过 HNSW 实现相似病例检索，支持医生快速定位历史诊断记录。

5 挑战与解决方案

HNSW 的索引构建时间随数据规模线性增长，可通过增量构建策略分批插入数据。针对数据分布不均问题，可结合 K-Means 聚类预划分区域，每个子簇独立构建 HNSW 索引。开源工具如 FAISS 提供 HNSW 的 GPU 加速版本，可将构建速度提升 5-10 倍。

6 未来展望

随着多模态大模型兴起，HNSW 在跨模态检索中的应用值得关注。例如将 CLIP 生成的图文联合向量纳入统一索引空间，实现「以图搜文」或「以文搜图」。硬件层面，基于 CXL 协议的持久化内存可突破传统 RAM 容量限制，支持万亿级向量实时检索。

HNSW 凭借其分层图结构与动态更新能力，已成为高维向量检索的首选方案。开发者应根据数据规模、维度分布和实时性需求调整参数，必要时结合量化、分片等技术突破性能瓶颈。随着算法与硬件的协同进化，HNSW 将在更多场景中释放潜力。