向量检索常见面试篇

来自： AiGC面试宝典

宁静致远　　2024年01月12日 06:45

一、向量检索库总结

1.1 Annoy

github:https://github.com/spotify/annoy

1.1.1 Annoy 介绍

Annoy是高维空间求近似最近邻的一个开源库。全称：Approximate Nearest Neighbors Oh Yeah，是一种适合实际应用的快速相似查找算法。
Annoy构建一个二叉树，查询时间为O（logn）。

1.1.2 Annoy 使用

直接通过pip install annoy安装。

```
from annoy import AnnoyIndex
import random
# 向量的维度
f = 40  # Length of item vector that will be indexed
# 返回一个可读可写的存储 f 维向量的索引
t = AnnoyIndex(f, 'angular')
for i in range(1000):
    # random.gauss为随机生成高斯分布的随机数
    v = [random.gauss(0, 1) for z in range(f)]
    # 在位置 i 添加向量
```

```
    t.add_item(i, v)
# 建立一棵 n_trees的森林，树越多，精度越高
t.build(10) # 10 trees
# 保存
t.save('test.ann')

# ...

u = AnnoyIndex(f, 'angular')
# 直接加载
u.load('test.ann') # super fast, will just mmap the file
print(u.get_nns_by_item(0, 1000)) # will find the 1000 nearest neighbors
```

上面的例子为官方github提供的例子。
向量检索时用到的函数

- a.get_nns_by_item(i, n, search_k=-1, include_distances=False): 返回最接近 i 的n个item。查询过程中，将检查search_k个节点，默认为n_trees* n。serarch_k实现了准确性和速度之间的运行时间权衡。include_distances为True时将返回一个包含两个列表的2元素元组:第二个包含所有相应的距离。
- a.get_nns_by_vector(v, n, search_k=-1, include_distances=False): 和上面的根据item查询一样，只不过这里时给定一个查询向量v,比如给定一个用户embedding， 返回n个最近邻的item,一般这样用的时候， 后面的距离会带着，可能作为精排那面的强特。
- a.get_item_vector(i): 返回索引i对应的向量。
- a.get_distance(i, j): 返回item_i和item_j的平方距离。

索引属性函数

- a.get_n_items(): 返回索引中的items个数，即词典大小。
- a.get_n_trees(): 索引树的个数。

annoy接口中一般需要调整的参数有两个：树的数量n_trees和搜索过程中检查的节点数量search_k

- n_trees: 在构建期间提供，影响构建时间和索引大小。值越大，结果越准确，但索引越大。
- search_k: 在运行时提供，并影响搜索性能。值越大，结果越准确，但返回的时间越长。如果不提供，就是n_trees * n， n是最近邻的个数。

1.2 Faiss

- github:https://github.com/facebookresearch/faiss
- tutorial:https://github.com/facebookresearch/faiss/wiki/Getting-started

1.2.1 Faiss 介绍

Faiss库是由 Facebook 开发的适用于稠密向量匹配的开源库，支持 c++ 与 python 调用。Faiss提供了高效的索引类库。是向量化检索开山鼻祖的应用。
Faiss 支持多种向量检索方式，包括内积、欧氏距离等，同时支持精确检索与模糊搜索。

1.2.2 Faiss 主要特性

- 支持相似度检索和聚类；
- 支持多种索引方式；
- 支持CPU和GPU计算；

- 支持Python和C++调用；

1.2.3 Faiss 使用

直接通过pip install faiss-cpu --no-cache进行安装。
faiss的使用方法也比较简单，归纳为以下三个步骤：

1. 构建向量库，对已知的数据进行向量，最终以矩阵的形式表示
2. 为矩阵选择合适的index，将第一步得到的矩阵add到index中
3. search得到最终结果

```
import numpy as np
import faiss

d = 64
nb = 100000
nq = 10000
# 构建向量库
xb = np.random.random((nb, d)).astype('float32')
xb[:, 0] += np.arange(nb) / 1000.
xq = np.random.random((nq, d)).astype('float32')
xq[:, 0] += np.arange(nq) / 1000.

# 关键步骤，build index
index = faiss.IndexFlatL2(d)
index.add(xb)

k = 4
D, I = index.search(xq[:5], k)   # 分别返回距离和索引
```

1.3 Milvus

Milvus的更多介绍可以查看：https://gitee.com/milvus-io/mil
Milvus 是一款开源的特征向量相似度搜索引擎，使用方便、实用可靠、易于扩展、稳定高效和搜索迅速。

- 高性能：涵盖如Faiss、Annoy和hnswlib等主流第三方索引库，性能高，支持对海量向量数据进行相似搜索。
- 高可用、高可靠：Milvus支持使用Kubernetes部署，支持在云上扩展。其容灾能力能够保证服务的高可用。Milvus依照日志及数据的理念，使用如Pulsar、Kafka等消息队列的技术实现组件间的通信，对组件进行解耦，拥抱云原生。
- 混合查询：Milvus支持在向量检索过程中进行标量字段过滤，实现混合查询。
- 开发者友好：支持多语言、多工具的Milvus生态。如今Milvus已经支持Python、Java、Go和Node.js，未来可能还会扩展对更多语言的支持。Milvus提供了如Attu等工具，帮助用户简化操作。

1.4 ElasticSearch

1.4.1 ElasticSearch 介绍

Elasticsearch 是一个分布式可扩展的实时搜索和分析引擎，一个建立在全文搜索引擎 Apache Lucene(TM)基础上的搜索引擎，当然 Elasticsearch 并不仅仅是 Lucene 那么简单，它不仅包括了全文搜索功能，还可以进行以下工作：

- 分布式实时文件存储，并将每一个字段都编入索引，使其可以被搜索。
- 实时分析的分布式搜索引擎。
- 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。

ES本质上是一个支持全文搜索的分布式内存数据库，特别适用于构建搜索系统，比如内容检索、文本检索、日志检索。其原因是采用了倒排索引。

1.4.2 什么是倒排索引呢?

倒排索引是一种特别为搜索而设计的索引结构。
先对需要索引的字段进行分词，然后以分词为索引组成一个查找树，这样就把一个全文匹配的查找转换成了对树的查找。
倒排索引相比于一般数据库采用B树索引，其写入和更新的性能比较差，因此倒排索引只适合全文搜索，不适合更新频繁的交易类数据。

1.4.3 ES机制

Elasticsearch的文件存储， Elasticsearch是面向文档型数据库，一条数据在这里就是一个文档，用JSON作为文档序列化的格式，比如下面这条用户数据：

```
{
"name" : "John",
"sex" : "Male",
"age" : 25,
"birthDate": "1990/05/01",
"about" : "I love to go rock climbing",
"interests": [ "sports", "music" ]
}
```

也就是说Elasticsearch比较适合存储非结构化或半结构化数据。
用Mysql这样的数据库存储就会容易想到建立一张User表，有用户信息的字段等，在Elasticsearch里这就是一个文档，当然这个文档会属于一个User的类型，各种各样的类型存在于一个索引当中。
这里有一份将Elasticsearch和关系型数据术语对照表：
关系数据库【数据库关系系统】 ⇒ 数据库 ⇒ 表 ⇒ 行 ⇒ 列(Columns)
Elasticsearch ⇒ 索引(Index) ⇒ 类型(type) ⇒ 文档(Docments) ⇒ 字段(Fields)