昵称：侯凯
园龄：8年8个月
粉丝：251
关注：10
+加关注

< 2022年2月 >

日	一	二	三	四	五	六
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	1	2	3	4	5
6	7	8	9	10	11	12

搜索
找找看
谷歌搜索

随笔分类
C++及STL(23)
java开发(5)
python知识与应用(17)
shell编程及小工具(10)
web相关(3)
操作系统(4)
长尾知识(9)
大数据学习(8)
经典算法(12)
深度学习(12)
数据库(18)
图像算法及OpenCV(33)
系统框架(41)

Faiss教程：入门

作者：@houkai
本文为作者原创，转载请注明出处：https://www.cnblogs.com/houkai/p/9316129.html

Faiss处理固定维度d的数据，矩阵每一行表示一个向量，每列表示向量的一项。Faiss采用32-bit浮点型存储。

假设xb为数据集，维度为 ×　　；xq是查询数据，维度为 ×

```
import numpy as np
d = 64                           # dimension
nb = 100000                      # database size
nq = 10000                       # nb of queries
np.random.seed(1234)             # make reproducible
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.
```

为数据构建索引，Faiss包含非常多的索引类型，这里我们采用最简单的版本**IndexFlatL2**，基于L2距离进行brute-force搜索。

所有的索引的构建都需要知道它们操作数据的维度(d),其中大多索引需要一个训练过程，基于训练集来分析向量的分布。对IndexFlatL2，我们可以跳过训练。

索引创建后，add 和 search操作便可以基于索引来执行了。add 添加数据到索引(添加到xb)。

我们可以查看索引的属性状态，is_trained是否训练完成(有些不需要训练)，ntotal被索引数据的数目。

有一些索引，需要提供向量的整数ID，如果ID没有提供，add可以采用数据的序号数，第一个数据为0，第二个是1，以此类推。

```
import faiss                   # make faiss available
index = faiss.IndexFlatL2(d)   # build the index
print(index.is_trained)
index.add(xb)                  # add vectors to the index
print(index.ntotal)
# output
True
100000
```

基于索引便可以进行k近邻查询了，结果矩阵为 ×　　，第i行表示第i个查询向量，每行包含k个最近邻的ID，距离依次递增。同时返回相同维度的距离矩阵。

```
k = 4                          # we want to see 4 nearest neighbors
D, I = index.search(xb[:5], k) # sanity check
print(I)
print(D)
D, I = index.search(xq, k)     # actual search
print(I[:5])                   # neighbors of the 5 first queries
print(I[-5:])                  # neighbors of the 5 last queries
# output
[[  0 393 363  78]
 [  1 555 277 364]
 [  2 304 101  13]
 [  3 173  18 182]
 [  4 288 370 531]]
[[ 0.          7.17517328  7.2076292   7.25116253]
 [ 0.          6.32356453  6.6845808   6.79994535]
 [ 0.          5.79640865  6.39173603  7.28151226]
```

```
 [ 0.          7.27790546  7.52798653  7.66284657]
 [ 0.          6.76380348  7.29512024  7.36881447]]
[[ 381  207  210  477]
 [ 526  911  142   72]
 [ 838  527 1290  425]
 [ 196  184  164  359]
 [ 526  377  120  425]]
[[ 9900 10500  9309  9831]
 [11055 10895 10812 11321]
 [11353 11103 10164  9787]
 [10571 10664 10632  9638]
 [ 9628  9554 10036  9582]]
```

受向量第一项的影响，查询数据中头部数据的最近邻也在数据集的头部。

加速查询，首先可以把数据集切分成多个，我们定义Voronoi Cells，每个数据向量只能落在一个cell中。查询时只需要查询query向量落在cell中的数据了，降低了距离计算次数。

通过**IndexIVFFlat**索引，可以实现上面的思想，它需要一个训练的阶段。IndexIVFFlat需要另一个索引，称为quantizer，来判断向量属于哪个cell。

search方法也相应引入了nlist（cell的数目）和nprobe（执行搜索的cell数）

```
nlist = 100
k = 4
quantizer = faiss.IndexFlatL2(d)  # the other index
index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC_L2)
       # here we specify METRIC_L2, by default it performs inner-product search
assert not index.is_trained
index.train(xb)
assert index.is_trained

index.add(xb)                  # add may be a bit slower as well
D, I = index.search(xq, k)     # actual search
print(I[-5:])                  # neighbors of the 5 last queries
index.nprobe = 10              # default nprobe is 1, try a few more
D, I = index.search(xq, k)
print(I[-5:])                  # neighbors of the 5 last queries
# output
[[ 9900 10500  9831 10808]
 [11055 10812 11321 10260]
 [11353 10164 10719 11013]
 [10571 10203 10793 10952]
 [ 9582 10304  9622  9229]]
[[ 9900 10500  9309  9831]
 [11055 10895 10812 11321]
 [11353 11103 10164  9787]
 [10571 10664 10632  9638]
 [ 9628  9554 10036  9582]]
```

结果并不完全一致，因为落在Voronoi cell外的数据也可能离查询数据更近。适当增加nprobe可以得到和brute-force相同的结果，nprobe控制了速度和精度的平衡。

IndexFlatL2 和 IndexIVFFlat都要存储所有的向量数据，这对于大型数据集是不现实的。Faiss基于PQ提供了变体**IndexIVFPQ**来压缩数据向量（一定的精度损耗）。

向量仍是存储在Voronoi cells中，但是它们的大小可以通过m来设置(m是d的因子)。

由于向量值不在准确存储，所以search计算的距离也是近似的了。

```
nlist = 100
m = 8                             # number of bytes per vector
k = 4
quantizer = faiss.IndexFlatL2(d)  # this remains the same
index = faiss.IndexIVFPQ(quantizer, d, nlist, m, 8)
                                  # 8 specifies that each sub-vector is encoded as 8 bits
index.train(xb)
index.add(xb)
D, I = index.search(xb[:5], k) # sanity check
print(I)
print(D)
```

```
index.nprobe = 10              # make comparable with experiment above
D, I = index.search(xq, k)     # search
print(I[-5:])
# output
[[   0  424  363  278]
 [   1  555 1063   24]
 [   2  304   46  346]
 [   3  773  182 1529]
 [   4  288  754  531]]
[[ 1.45568264  6.03136778  6.18729019  6.38852692]
 [ 1.4934082   5.74254704  6.19941282  6.21501732]
 [ 1.60279942  6.20174742  6.32792568  6.78541422]
 [ 1.69804895  6.2623148   6.26956797  6.56042767]
 [ 1.30235791  6.13624859  6.33899879  6.51442146]]
[[10664 10914  9922  9380]
 [10260  9014  9458 10310]
 [11291  9380 11103 10392]
 [10856 10284  9638 11276]
 [10304  9327 10152  9229]]
```

最近距离(到自身)不再是0了，因为数据被压缩了。整理64位 32-bits向量，被分割为8份，每份用8bits表示，所以压缩因子为32。

查询数据集的结果和IVFFlat对比，大多是错误的，但是它们都在10000左右。这种策略在实际数据中是更好的：

- 均匀分布的数据是很难索引的，它们很难聚类和降维
- 自然数据，相似数据比不相干数据的距离要显著的更小。

使用工厂方法简化索引构建

```
index = faiss.index_factory(d, "IVF100,PQ8")
```

PQ8替换为Flat便得到了IndexFlat索引，工厂方法是非常有效的，尤其是对数据采用预处理的时候，如参数"PCA32,IVF100,Flat"，表示通过PCA把向量减低到32维。

Faiss可以基本无缝地在GPU上运行，首先申请GPU资源，并包括足够的显存空间。

```
res = faiss.StandardGpuResources()  # use a single GPU
```

使用GPU创建索引

```
# build a flat (CPU) index
index_flat = faiss.IndexFlatL2(d)
# make it into a gpu index
gpu_index_flat = faiss.index_cpu_to_gpu(res, 0, index_flat)
```

索引的使用和CPU上类似

```
gpu_index_flat.add(xb)         # add vectors to the index
print(gpu_index_flat.ntotal)

k = 4                          # we want to see 4 nearest neighbors
D, I = gpu_index_flat.search(xq, k)  # actual search
print(I[:5])                   # neighbors of the 5 first queries
print(I[-5:])                  # neighbors of the 5 last queries
```

使用多张GPU卡

```
ngpus = faiss.get_num_gpus()

print("number of GPUs:", ngpus)

cpu_index = faiss.IndexFlatL2(d)

gpu_index = faiss.index_cpu_to_all_gpus(  # build the index
```

```
    cpu_index
)

gpu_index.add(xb)              # add vectors to the index
print(gpu_index.ntotal)

k = 4                          # we want to see 4 nearest neighbors
D, I = gpu_index.search(xq, k) # actual search
print(I[:5])                   # neighbors of the 5 first queries
print(I[-5:])                  # neighbors of the 5 last queries
```

分类: 图像算法及OpenCV

标签: faiss

好文要顶 关注我 收藏该文

侯凯
关注 - 10
粉丝 - 251
+加关注

0 推荐 0 反对

« 上一篇： docker简易命令
» 下一篇： Faiss教程：索引(2)

posted @ 2018-07-16 09:49 侯凯 阅读(9798) 评论(0) 编辑 收藏 举报

刷新评论 刷新页面 返回顶部

登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

编辑推荐：
· 使用 Three.js 让二维图片具有3D效果
· 疑难杂症：运用 transform 导致文本模糊的现象探究
· ASP.NET Core 6框架揭秘实例演示[05]：依赖注入基本编程模式
· 走进Task（2）：Task 的回调执行与 await
· 戏说领域驱动设计（五）——子域

最新新闻：
· 未来，你的手机屏幕可能是「钻石」造的？
· 百度计算生物研究登Nature子刊！结果超斯坦福MIT，落地制药领域
· 比一粒盐还小的电池问世
· 死前真的会有「跑马灯」，人类首次同步测量大脑濒死状态
· 网传员工猝死，字节跳动内网流出回应：仍在医院抢救中
» 更多新闻...