faiss/乘积量化---从一维到高维的最近邻搜索

原创 卢新来 人民程序员 7月5日

收录于话题 #机器学习 3个

算法工程师应该对youtube那篇“Deep Neural Networks for YouTube Recommendations”印象深刻，文中的算法架构思想固然很有影响力，而核心的高维向量相似性搜索看上去也是相当神奇。

facebook的faiss可以说是相似向量搜索的标杆，而它背后的算法就是乘积量化。正好我也接触过一些低维空间的类似问题，所以就写来记录一下。

本篇从一维向量搜索开始，然后说一说二维，最后推广到高维，并在这个过程中逐渐说明遇到的问题，以便更好地理解乘积量化的原理。

一维

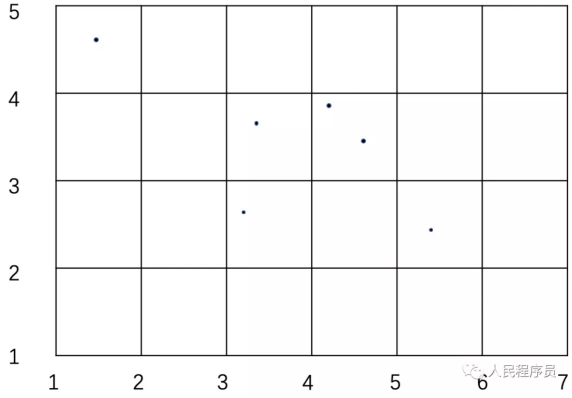
先看这个问题：假设有一个已知的无序的数组A，再给定某个值x，现在要查找A中距离x最近的值。这个应该很简单，直接遍历一遍就可以了。

现在再加一个条件：假设有一系统的值x1，x2，x3...，查询它们距离A中最近的值。这个似乎也还好，因为A是已知的不变的，所以可以进行排序预处理，然后对每个x值二分查找排序后的A即可。这个办法也很直接，但和乘积量化还不沾边。

二维

把一维的推广到二维就是：给定已知的点集合P={(x1, y1), (x2, y2), (x3, y3)...(xn, yn)}，然后随机给一个点p(x, y)，找点集P中距离点p最近的点。

这个问题似乎也不算多难，geohash，kdtree和四叉树等很多办法都可以解决它。但其实和这些“大杀器”相比，还有个相对容易实现但同样高效的办法：画方格。就是说先将连续的坐标值“量化”或“离散”成整数值并进行编码建立倒排，查找时先将点p(x,y)通过参数进行量化，再查找倒排即可找到距离最近点。当量化后落到某方格中的点不会太多时，这个过程就非常高效。

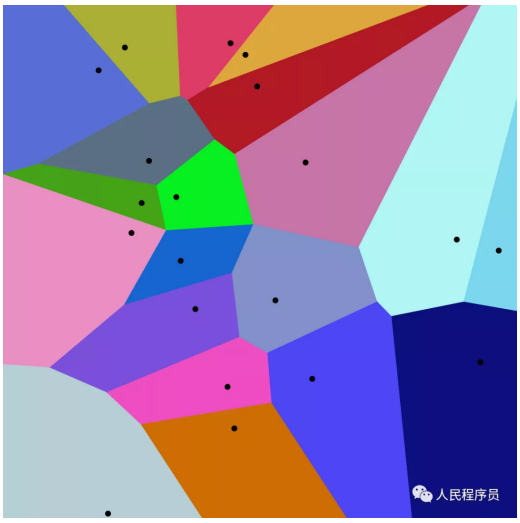


再回头看看一维，直接就有这样的新解法：将一维数组量化成多个“线段”并建倒排，对查询值也同样地量化再查找倒排。

向量量化

量化这个词，我最早在通信原理里遇到过，本质就是发送端对连续值信号进行离散化并编码，接收端再从离散值重建连续值信号。量化是有损的过程。

量化方式有很多种，画方格的只是一个特例。有人已经证明近似最优的量化方式是k-means聚类量化，并以质心表示量化结果，这一过程相当于用Voronoi单元分割向量空间。所谓Voronoi单元就是单元内任意一点到其质心的距离都小于该点到邻近Voronoi单元的质心的距离。乘积量化就用到了k-means量化。



“画方格”和k-means量化的一个区别是，画方格的量化结果正好和编码重合，这一特点使它用起来很方便，而k-means量化结果仍然是连续值的坐标，查找时就需要遍历

操作。另一个区别是，“画方格”是在各个维度上独立量化，而k-means是将向量做为整体进行量化。

高维向量的量化

画方格量化在高维向量空间会遇到问题：假设每维量化成k个值，D维空间量化后的可能值有k的D次方个。想像下k=2，D=128吧。这完全没法实现。

k-means量化的问题，一是内存占用比较高。D维向量k个质心时空间占用是**kD**。二是量化和查找过程都比较耗时。

后面的量化就**专指**k-means量化了。

乘积量化

乘积量化的思路是把高维向量分割成多段低维向量，再各个量化。比如，D维高维向量分成m段，每段维度为：

$$D^*=D/m$$

每段单独用q量化：

$$\underbrace{x_1,\ldots,x_{D^*}}_{u_1(x)},\ldots,\underbrace{x_{D-D^*+1},\ldots,x_D}_{u_m(x)} \rightarrow q_1(u_1(x)),\ldots,q_m(u_m(x))$$

这一步其实只解决了内存的问题，即用分段后质心（数量较少）的乘积表示未分段时的质心（数量较大）。假设未分段时量化质心数量为k，每个分段量化后的质心数量都为k*，则m段共可表示的质心数量为k*的m次方，它们之间关系是：

$$k = (k^*)^m$$

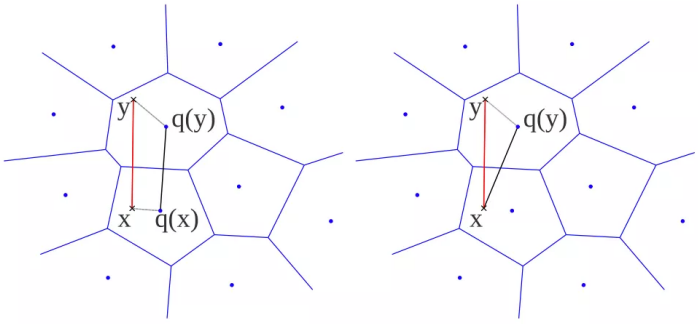
现在的**空间占用**是:

$$m\,\hat{k}^*\,D^* = k^{1/m}D$$

相比上面得出的kD减少很多。

距离度量

在查找时有两种度量查询向量与质心距离的方式，一种对查询向量x进行量化，叫做SDC(Symmetric distance computation)，另一种不对x量化，叫ADC（Asymmetric distance computa）。一般情况下ADC效果较好。



“非暴力”查找

乘积量化有效地降低空间复杂度，查找的时间复杂度只有一些降低，为**O(nm)**，其中n为输入数据量。这虽然不是暴力查找，但复杂度仍然很高。

优化时间复杂度的办法是使用倒排索引：先用一个“粗粒度的量化器”q(c)对数据进行量化，质心作为倒排的key，聚类到该质心的向量串成链表作为value。这样就能将查找过程聚焦到一小部分向量上。这被叫做IVFADC（an inverted file system with the asymmetric distance computation）。具体在value的处理上，并不是直接量化原向量，而是去量化一个残差向量：

$$r(y) = y - q_c(y)$$

其中y为原向量，q为粗粒度量化器。直观上，r(y)比y小，量化效果应该会好些---相同量化粒度的情况下，数值范围小的精度应该更高。

可以用下图表示上面的思路。其中c表示粗粒度量化质心，rpq（residual product quantizer)表示对残差的量化，箭头表示映射关系。

c1 ⟶ rpq1, rpq5, rpq20,⋯

c2 ⟶ rpq7, rpq11, rpq18,⋯

c3 ⟶ rpq2, rpq9, rpq23,⋯

这样在查找时就先对查询向量x进行粗粒度量化，再去相应倒排里搜索最近邻。假设倒排能均衡地分割n条数据，那每个倒排长度约为：n/kc，kc为粗粒度质心数据，所以总的时间复杂度为**O(n/kc*m)**，相比O(nm)有很大提高。

还要补充，粗粒度量化可能将查询向量x的最近点聚类到相邻类中，所以在查找时一般要用x的几个粗粒度邻近质心同时查找。

最后还有个问题：对查询向量x的粗粒度量化的耗时。因为按说也要遍历各粗粒度质心才能拿到最近的几个质心，这样当质心数量较多时光这一步就挺耗时的。原文中的解释是，一是有层次型量化器可以高效地完成x量化；二是当粗质心较多时，相应地倒排链表长度就减少。所以对大数据集，粗质心越多，IVFADC性能越好，而对小数据集，直接用ADC就好。

总结

总的来看，faiss背后的原理就关键的两点：1.用乘积量化减少内存，2.用倒排提高性能。

海量数据用simhash去重和这个问题应该是类似的。

补充个问题

一维、二维最近邻查找大家遇到的比较多，尤其二维的，比如在地图上查找离定位最近的厕所。之前工作中还有个相关的问题，欢迎思考：假设用矩形表示地图上的大量景区，现在在大量的定位点传送过来，要求找到落到每个景区的那些点。如果觉得简单，那把矩形换成多边形再试试。

参考文献：

1. Product quantization for nearest neighbor search

2. http://vividfree.github.io/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/2017/08/05/understanding-product-quantization