文本和图像搜索

DM-Team
2018-11

提纲

- 文本搜索-Elastic Search
  - 索引文档
    - 倒排
    - FST 有限状态转换器
    - Skip List 跳表
  - 查询文档
    - 提高Recall
    - 提高Precision
    - 相关度计算
    - 性能分析

提纲

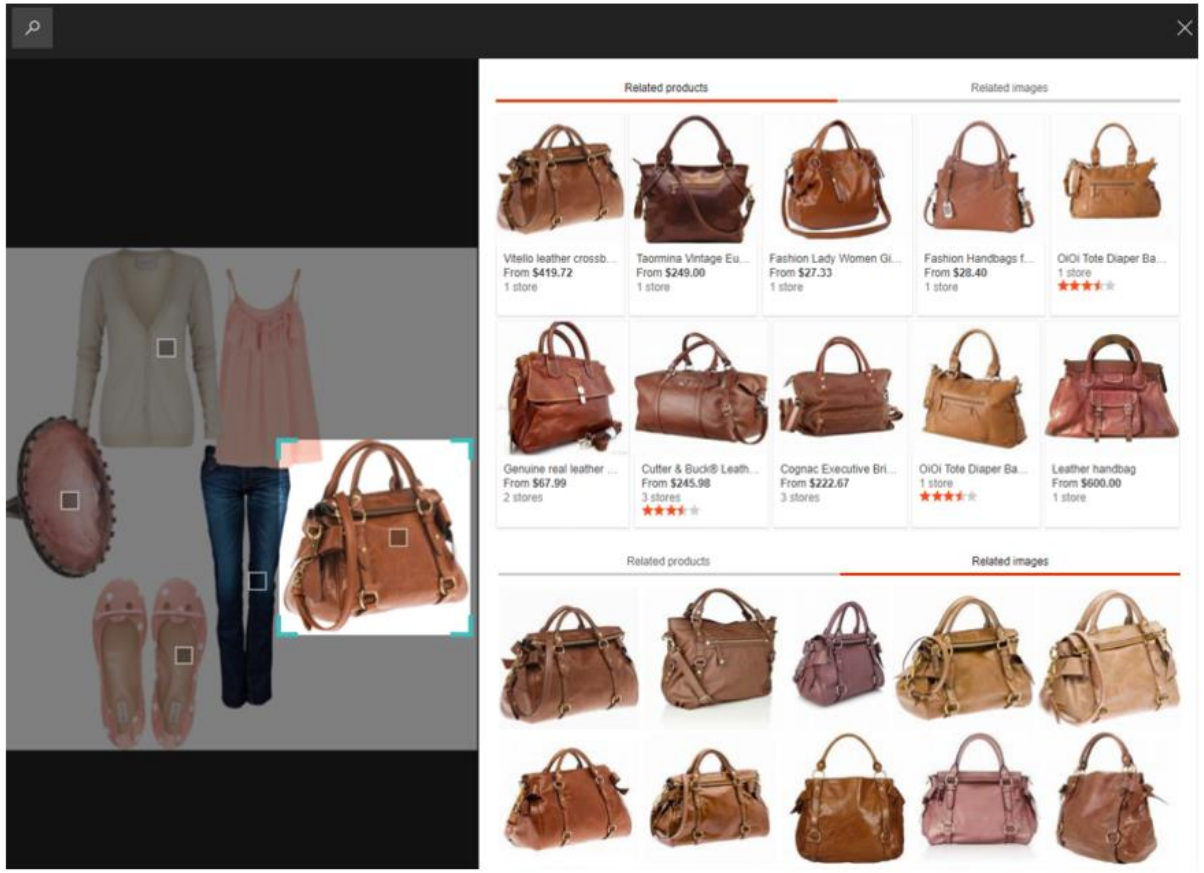
- 向量搜索-Faiss
  - NN,KNN,ANN
  - Quantization 量化
  - Product Quantization 乘积量化
  - Benchmark

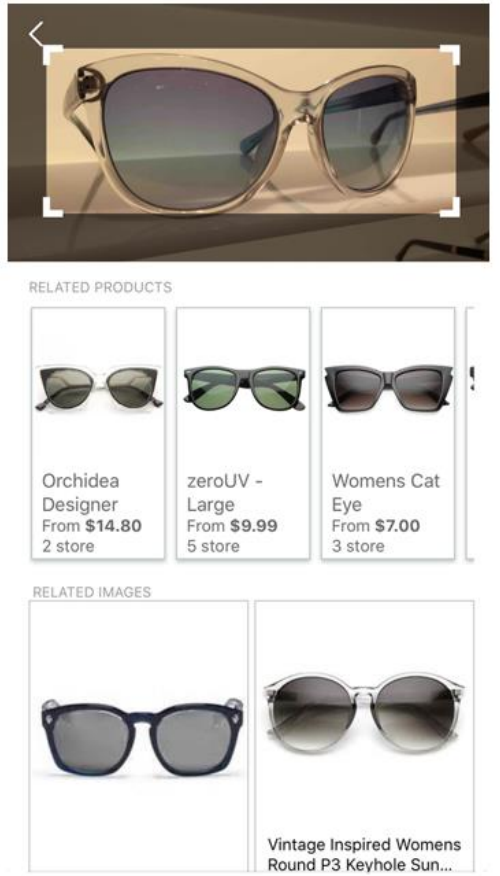
图像搜索和Faiss

Faiss

- NN,KNN,ANN
- Quantization 量化
- Product Quantization 乘积量化
- Benchmark

图像搜索


Related products
Related images
Vitello leather crossb...
From $419.72
1 store
Taormina Vintage Eu...
From $249.00
1 store
Fashion Lady Women Gi...
From $27.33
1 store
Fashion Handbags f...
From $28.40
1 store
OiOi Tote Diaper Ba...
1 store
Genuine real leather ...
From $67.99
2 stores
Cutter & Buck® Leath...
From $245.98
3 stores
Cognac Executive Bri...
From $222.67
3 stores
OiOi Tote Diaper Ba...
1 store
Leather handbag
From $600.00
1 store
Related products
Related images


RELATED PRODUCTS
Orchidea Designer
From $14.80
2 store
zeroUV - Large
From $9.99
5 store
Womens Cat Eye
From $7.00
3 store
RELATED IMAGES
Vintage Inspired Womens Round P3 Keyhole Sun...

图像搜索

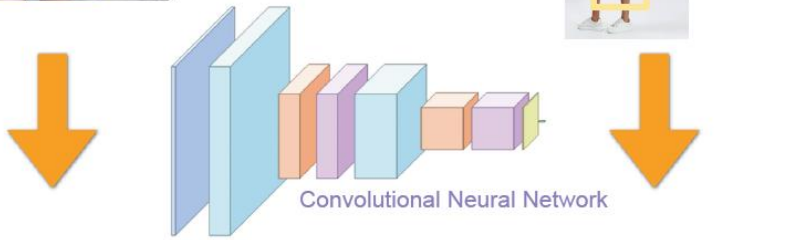
Architecture

Approach

1. Detect objects
2. Extract deep feature vectors

Query (Online)

Product Catalog (Offline)



$\mathbf{x} \in \mathbb{R}^{2048}$

$\{\mathbf{y}_i \in \mathbb{R}^{2048}; i = 1 \ldots N\}$

图像搜索

Architecture

Approach

1. Detect objects
2. Extract deep feature vectors
3. Find the closest vector in the reference set

Query (Online)

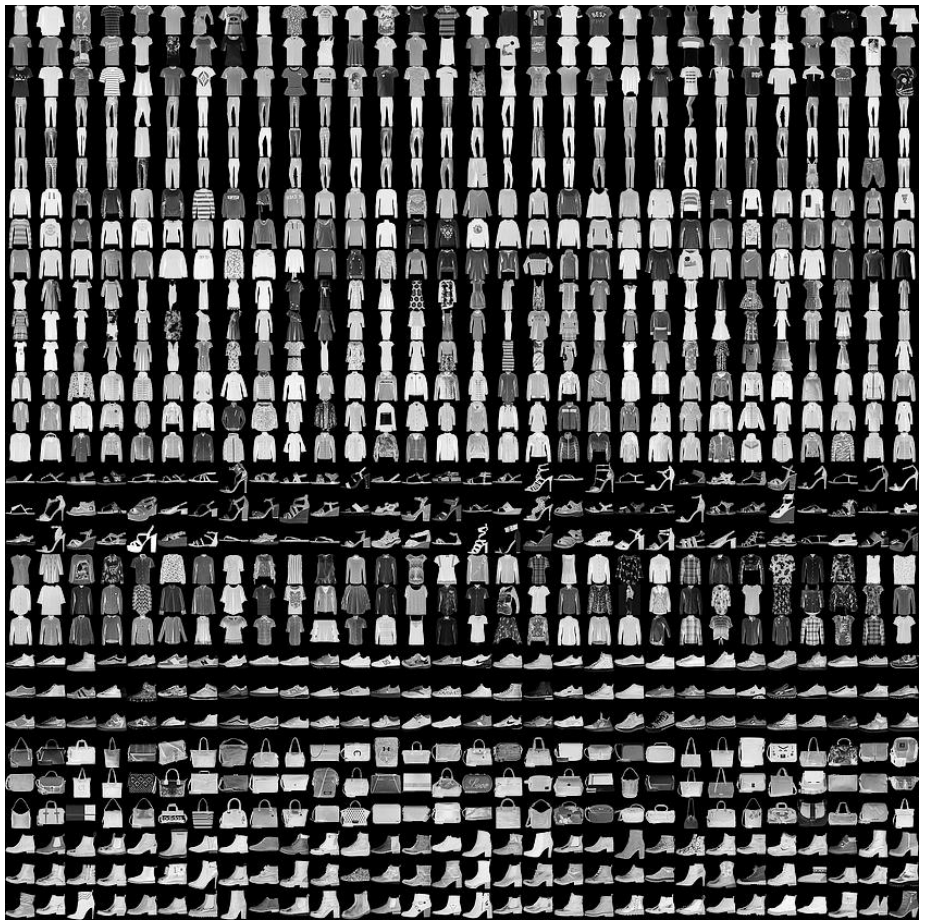
$\mathbf{x} \in \mathbb{R}^{2048}$

Product Catalog (Offline)

$\{\mathbf{y}_i \in \mathbb{R}^{2048}; i = 1 \ldots N\}$

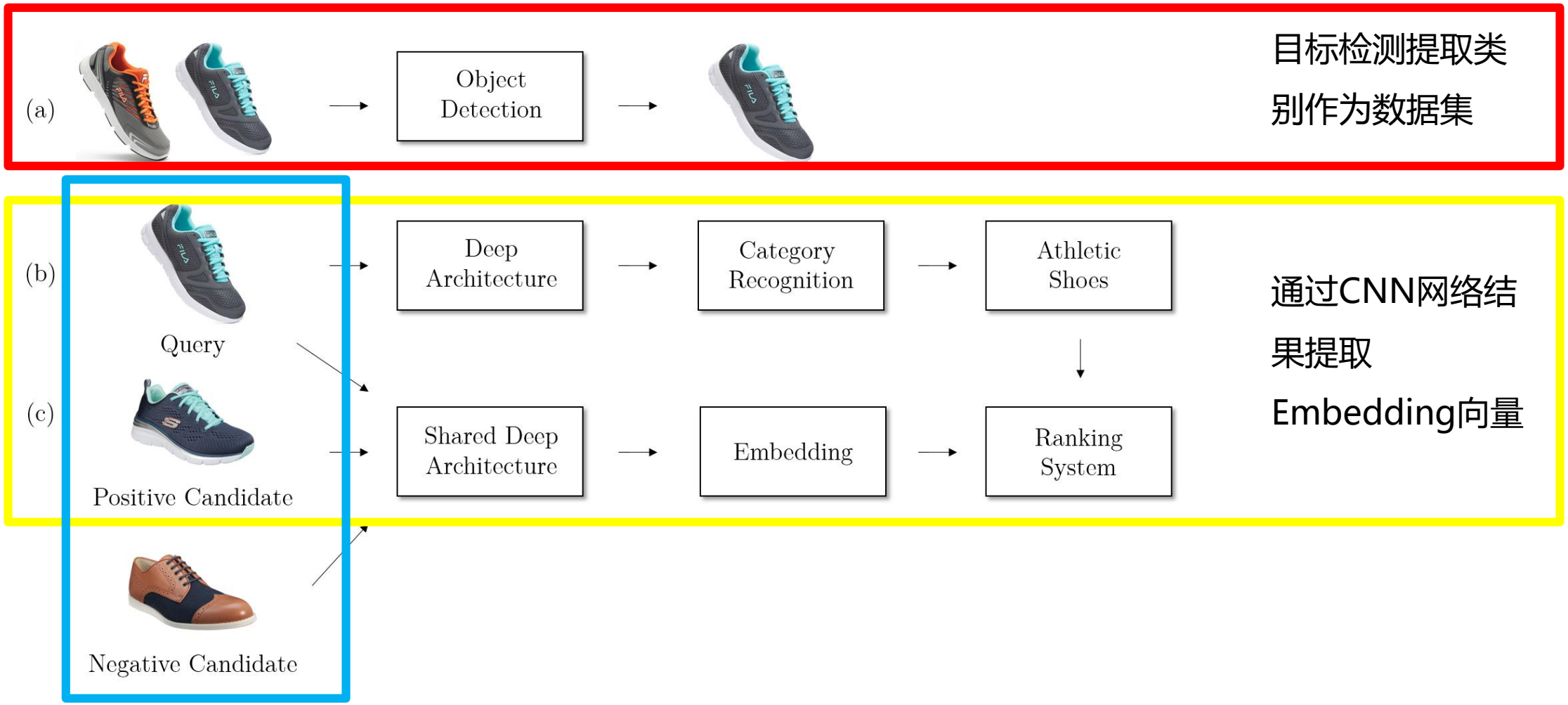
Search Result: $\underset{i}{\operatorname{argmin}} \| \mathbf{x} - \mathbf{y}_i \|$

图像搜索



一般从分类模型训练得到Embedding向量

图像搜索


(a)
Object Detection
目标检测提取类别作为数据集
(b)
Query
Deep Architecture
Category Recognition
Athletic Shoes
(c)
Positive Candidate
Shared Deep Architecture
Embedding
Ranking System
通过CNN网络结果提取Embedding向量
Negative Candidate

Nearest Neighbor 最近邻

Nearest Neighbor

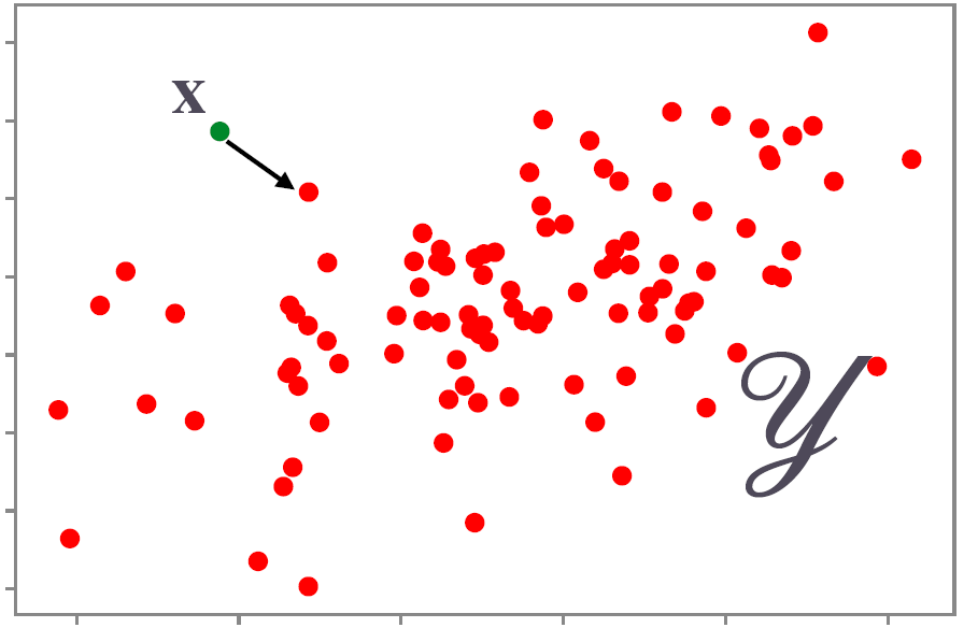
- Nearest Neighbor:

$$NN(\mathbf{x}) = \underset{i}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{y}_i)$$

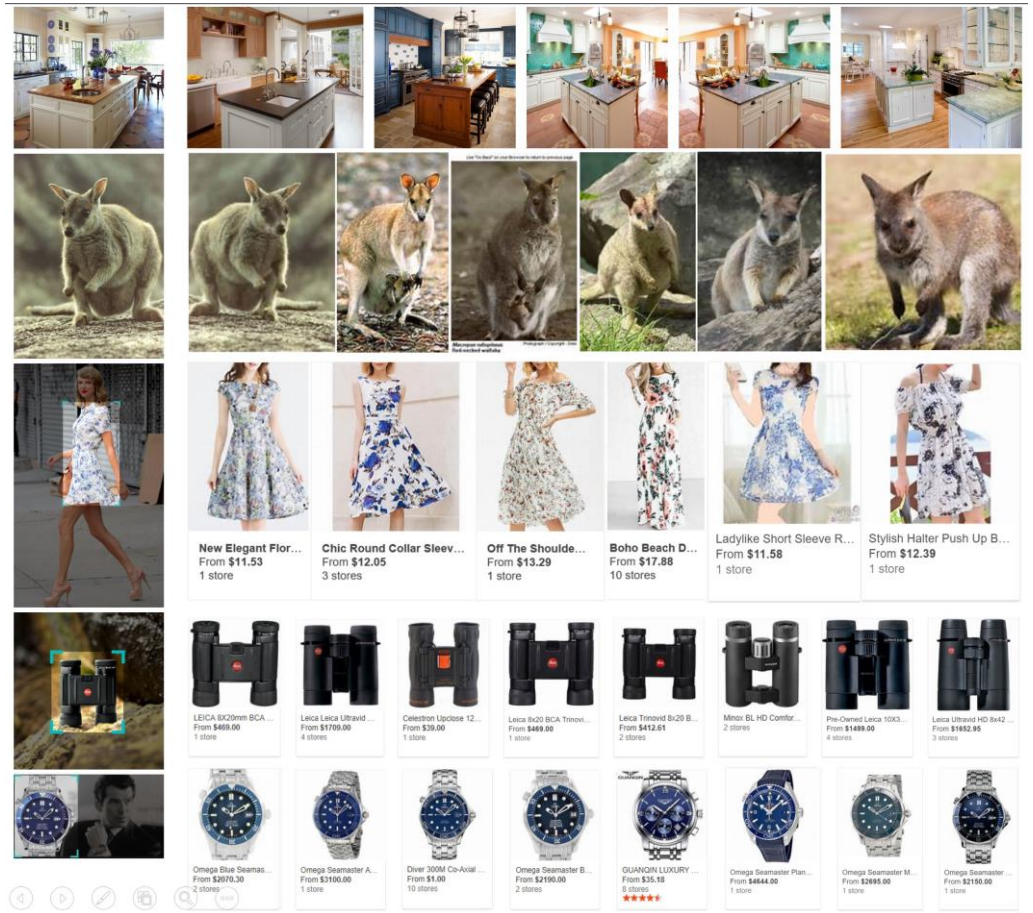
$$d(\mathbf{x}, \mathbf{y}_i) = \| \mathbf{x} - \mathbf{y}_i \|$$

- Complexity:

$$O(N),\ N = |\mathcal{Y}|$$



KNN（K最近邻）


New Elegant Flor…
From $11.53
1 store
Chic Round Collar Sleev…
From $12.05
3 stores
Off The Shoulde…
From $13.29
1 store
Boho Beach D…
From $17.88
10 stores
Ladylike Short Sleeve R…
From $11.58
1 store
Stylish Halter Push Up B…
From $12.39
1 store

KNN（K最近邻）

k-Nearest Neighbors

- Nearest Neighbor:

$$NN(\mathbf{x}) = \underset{i}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{y}_i)$$

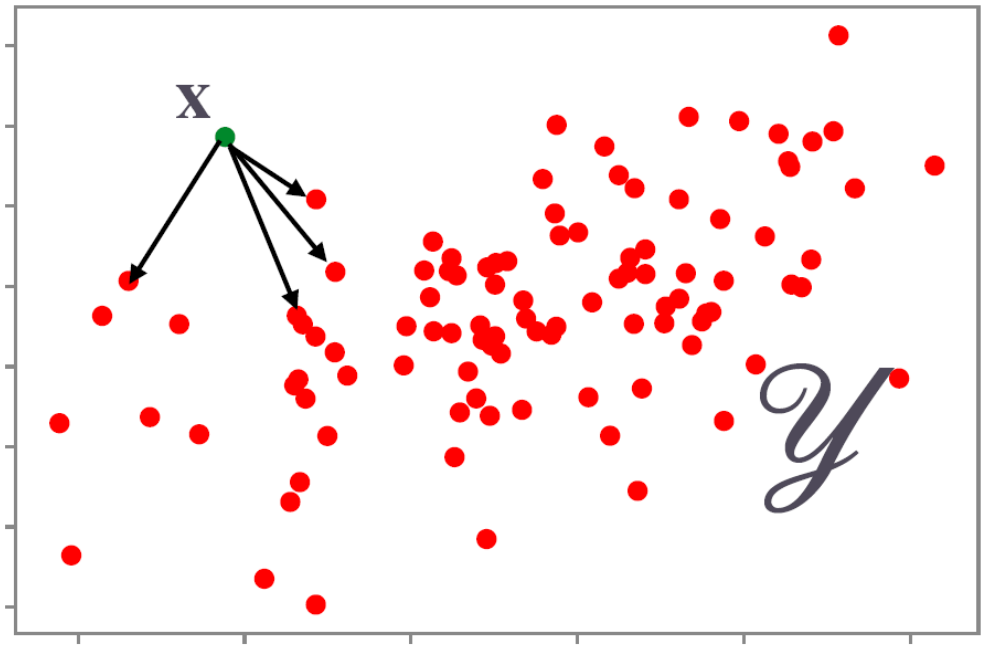
$$d(\mathbf{x}, \mathbf{y}_i) = \|\mathbf{x} - \mathbf{y}_i\|$$

- K-Nearest Neighbors:

  Finds top-k nearest neighbors

- Complexity:

$$O(N \log k)$$



O(n.logK)

堆排序

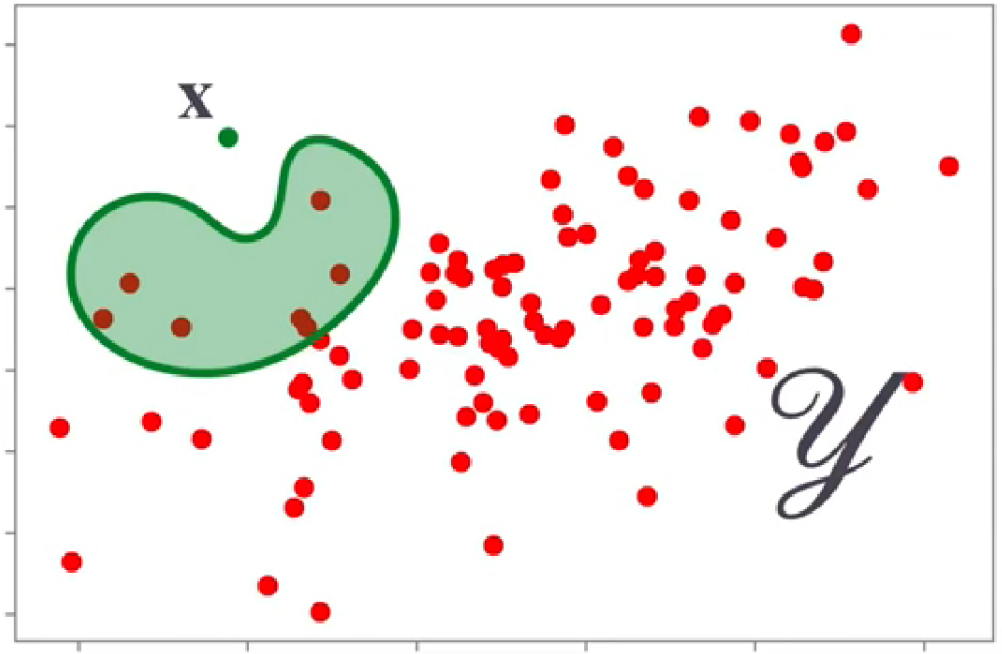
Approximate Nearest Neighbor

Approximate Nearest Neighbor

- Returns points *close* to the nearest neighbors
- Non Exhaustive
- Common accuracy metric: recall@*k*:

$$\frac{\left|\left\{\mathbf{y} \in \mathcal{Y} \,|\, NN(\mathbf{y}) \in K\tilde{N}N(\mathbf{y}, k)\right\}\right|}{|\mathcal{Y}|}$$

i.e, % of queries with correct answer in top-k approximate results.



当N非常大时，找到一些距离“最近邻”的最近的一些点也可以满足要求

非穷举搜索，用准确性来交换速度

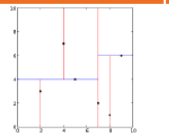
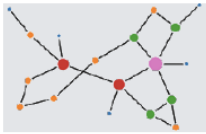
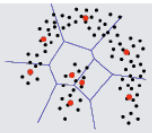
一般用recall@k来进行评价

ANN

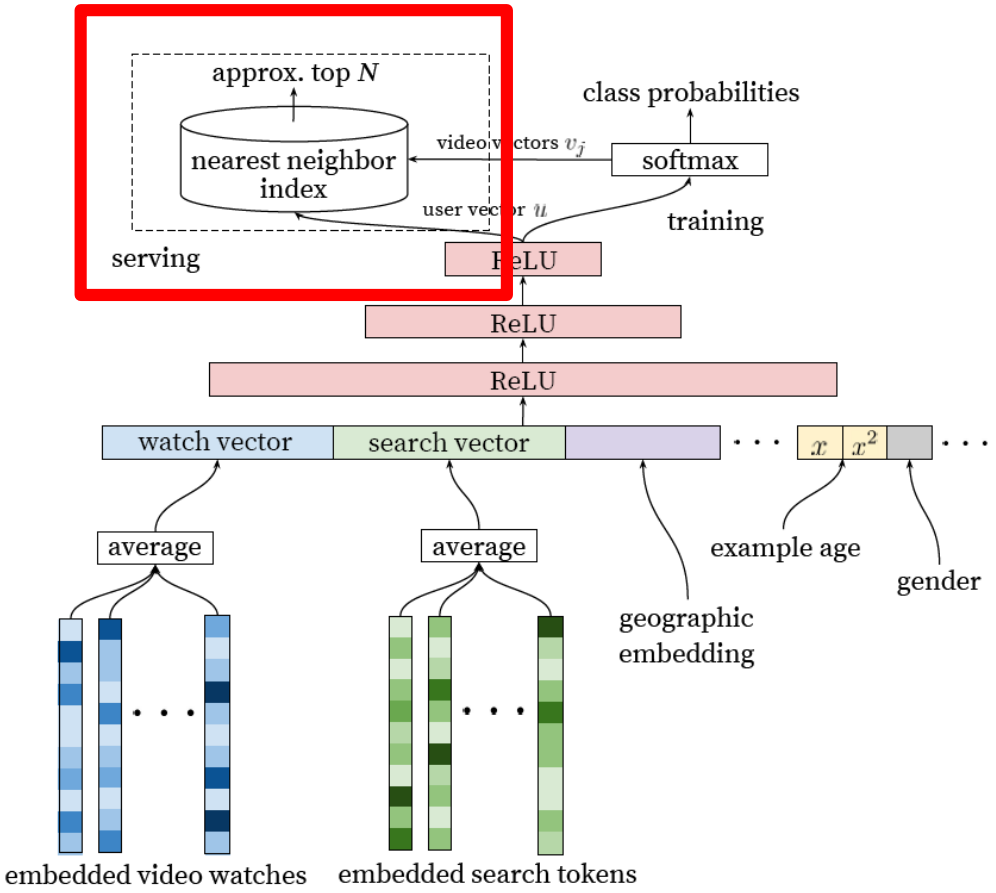
- brute-force搜索的方式是在全空间进行搜索，为了加快查找的速度，几乎所有的ANN方法都是通过对全空间分割，将其分割成很多小的子空间，在搜索的时候，通过某种方式，快速锁定在某一（几）子空间，然后在该（几个）子空间里做遍历
- 正是因为缩减了遍历的空间大小范围，从而使得ANN能够处理大规模数据的索引

ANN

Approximate Nearest Neighbor Techniques

Technique	Space	Query Time
Trees & Forests 	$O(N)$	$O(\log N)$
 Neighborhood Graphs	$O(N)$	$O(\log N)$
Locality Sensitive Hashing 	$O(N^{1+\rho})$	$O(N^{\rho} \log N)$
 Quantization	Sub-linear	Sub-linear

ANN



Youtube的DNN视频推荐

基于ANN的视频向量召回

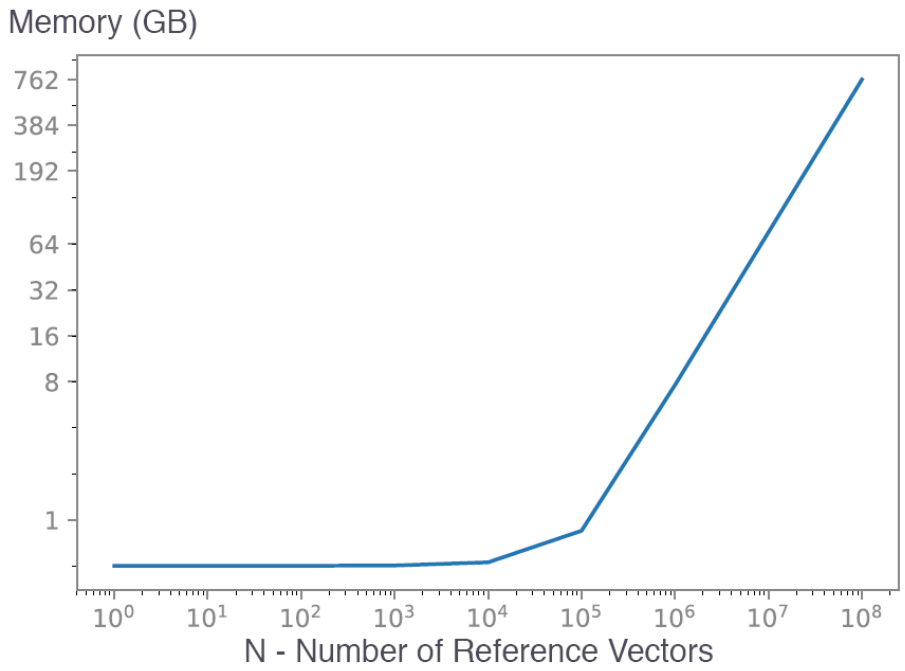
至于怎么从视频数据得到视频向量，不知道，应该是和Word2Vec类似的方法

ANN

CURALATE EXAMPLE

How bad is Linear Storage?

- d = 2048
- 8 KB storage per vector
- 1GB ~ 130K images
- 8GB ~ 1 Million images
- 7.6TB ~ 1 Billion images



存储原始向量占据比较大的空间

可以存储压缩后向量

Quantization 量化

Quantization: A Crash Course

Quantizer:

a function q

that maps a vector $\mathbf{y}$

to a vector $q(\mathbf{y}) \in C$

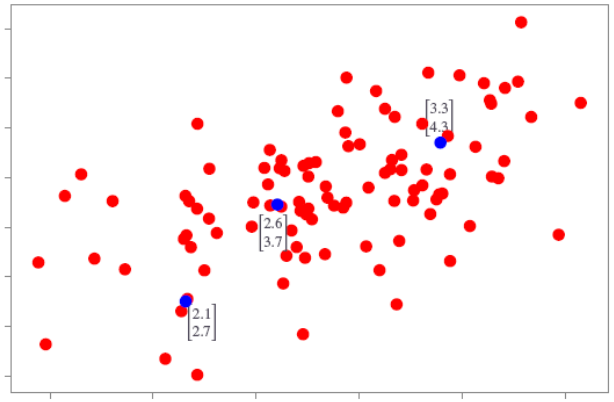
where C

is a finite subset of $\mathbb{R}^d$

with cardinality $k < N$.

$$\mathbf{y} \mapsto q(\mathbf{y}) \in C$$

$$q(\mathbf{y}) = \underset{\mathbf{c} \in C}{\operatorname{argmin}}\, d(\mathbf{y}, \mathbf{c})$$



$$C = \left\{ \begin{bmatrix} 2.1 \\ 2.7 \end{bmatrix}, \begin{bmatrix} 2.6 \\ 3.7 \end{bmatrix}, \begin{bmatrix} 3.3 \\ 4.3 \end{bmatrix} \right\}$$

Only *k* possible values.

用同一个空间下的有限个数向量代替原来所有向量

本质上是数据的一种压缩表达方法（通信学科的一个主要研究工作就是研究信号的压缩表达）

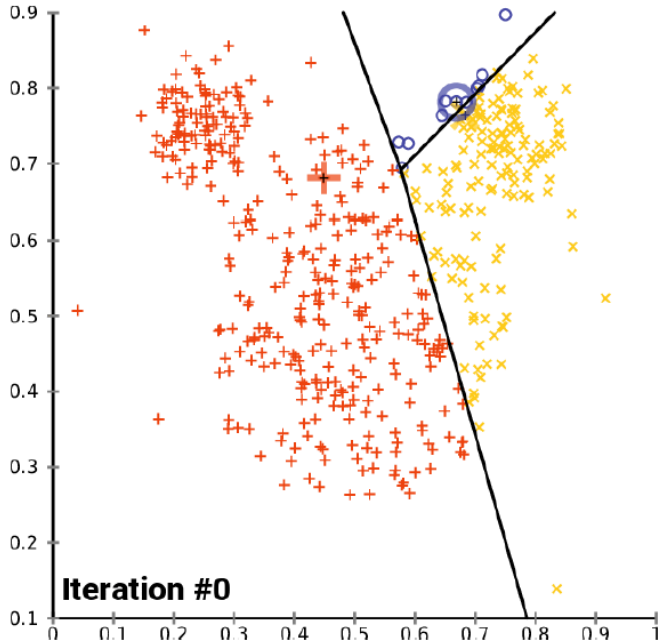
类似于向量空间的基向量

例如图中所有红色的点用三个蓝色点中最近的那个代替

Quantization-K-Means

Picking Centroids: K-Means

- Select initial centroids at random
- Until convergence:
  - Assign each point to nearest centroid
  - Compute new centroids as mean of all assignments.



“基向量”可以用K-Means来找到

Quantization-索引

Indexing with Quantization

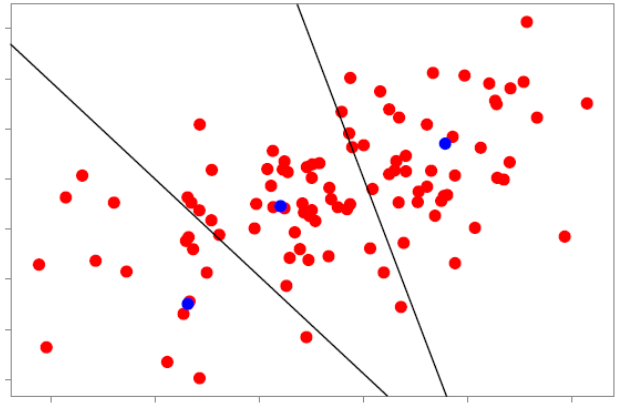
Approach

For each $\mathbf{y}_i \in \mathcal{Y}$:

- Quantize $\mathbf{y}_i \mapsto q(\mathbf{y}_i)$
- Let $z_i = \arg\min_{j \in |C|} q(\mathbf{y}_j) - \mathbf{c}_j$

  (i.e., which centroid you assigned)
- Store z_i on disk.

Visualization



Index

Image ID	z_i
1	1
2	3
3	3
4	2
5	1
6	3

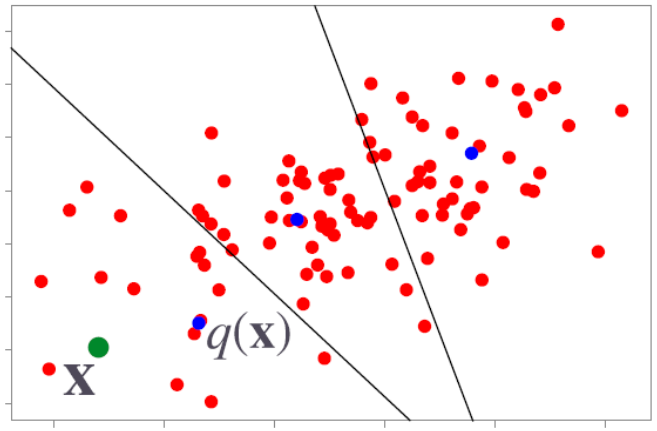
Only $\log_2 k$ bits per vector!

索引过程：量化之后用距离最近的聚类中心的id代替原来的图片id

Quantization-倒排结构

Sub-linear Search: Inverted Lists

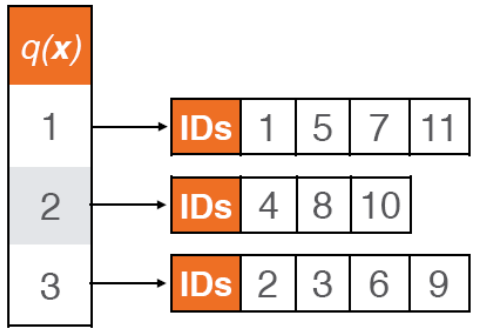
Query Quantization



Index

Image ID	z_i
1	1
2	3
3	3
4	2
5	1
6	3

Inverted List

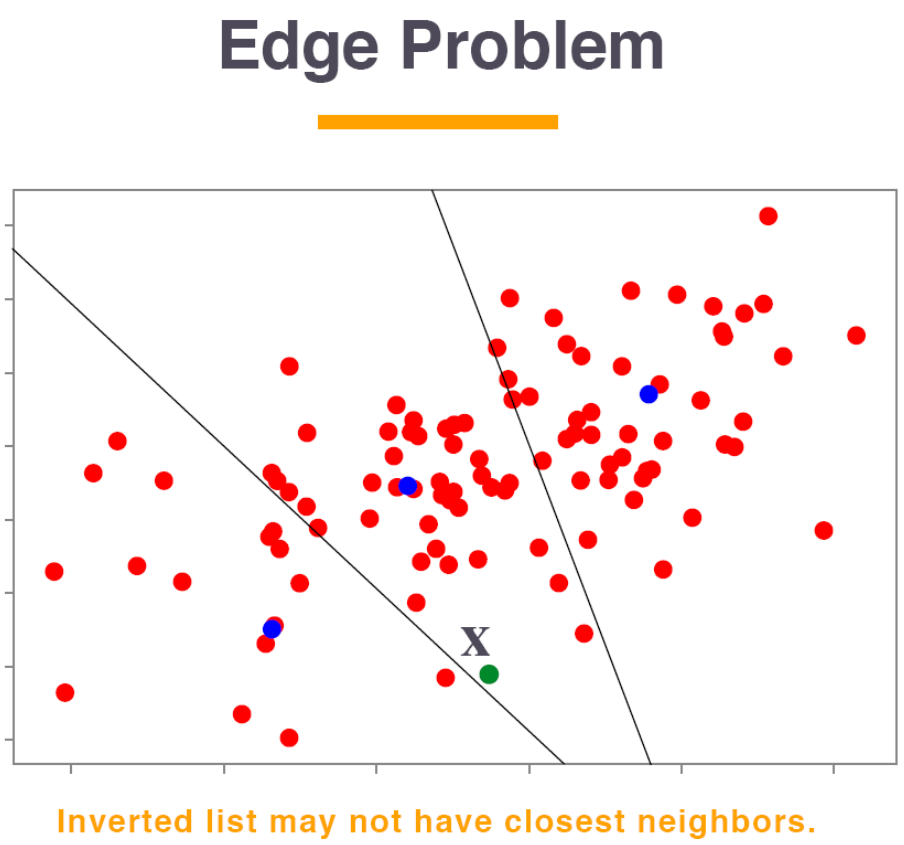


Reduces search time to $O(N/k)$

搜索的时候先建立图片id-聚类中国内心id索引和反向索引，只需要查找对应聚类中心对应的链表

平均的搜索时间是O(N/k)，在每个聚类中心对应的原始图片id分布比较均匀的情况下

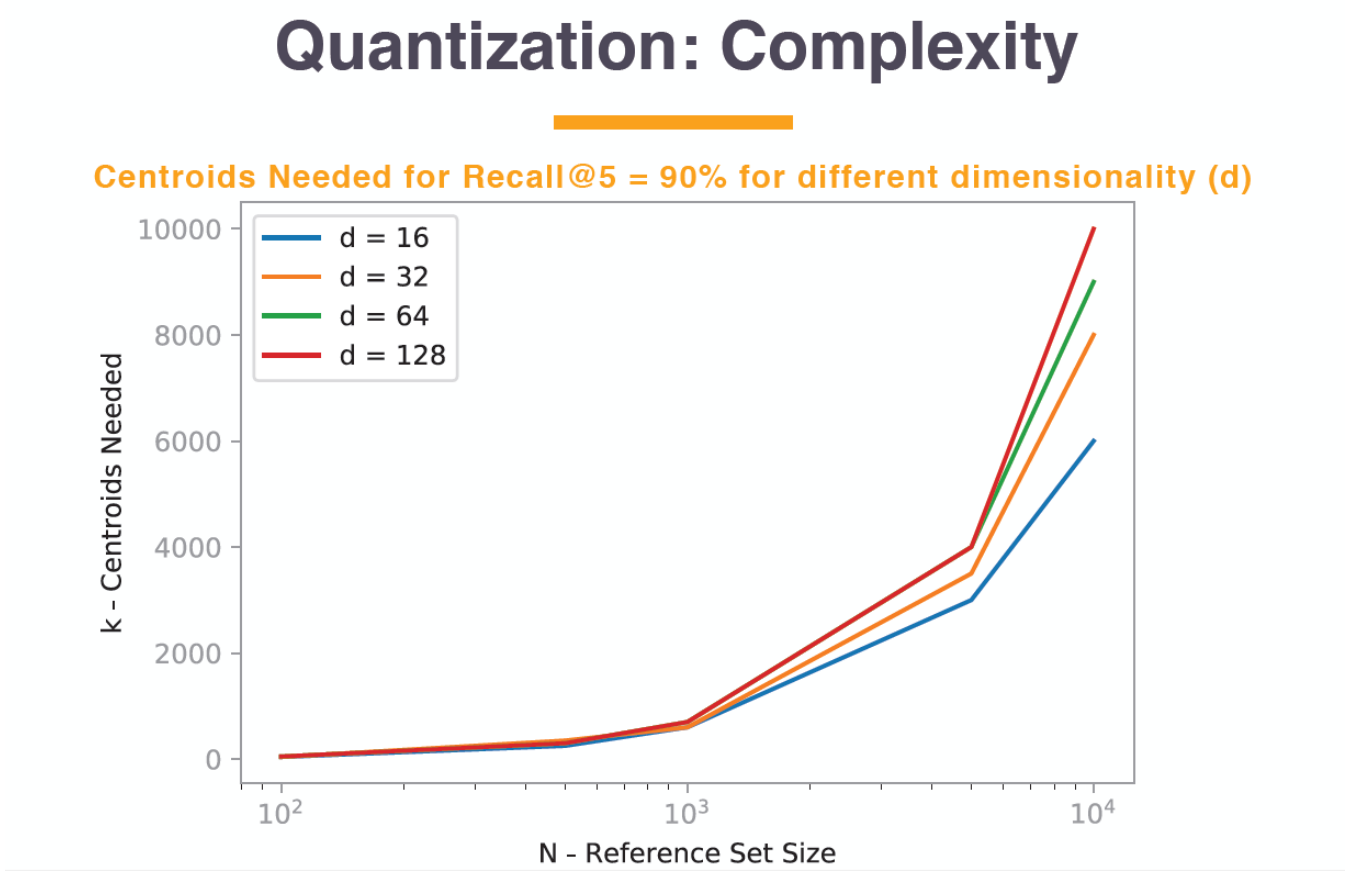
Quantization-召回率



同一个分区下的并不一定是最近邻的

因此K比较低时召回的准确度不高，需要比较多的聚类中心达到一定的准确度

Quantization-召回率



同一个分区下的并不一定是最近邻的

因此K比较低时召回的准确度不高，需要比较多的聚类中心达到一定的准确度

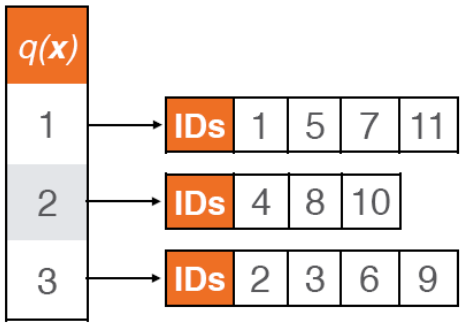
Quantization-重排序

Aside: Re-Ranking

Index

Image ID	z_i
1	1
2	3
3	3
4	2
5	1
6	3

Inverted List



Re-Ranking

Sort by actual distance:

$$d(\mathbf{x}, \mathbf{y}_i) \ \forall \mathbf{y}_i \in \text{list}(z_i)$$

Requires uncompressed storage!

提升准确度的另一个方法是存储原始向量

在倒排索引的列表中链接原始向量并进行重排序

Quantization-小结

Review: Quantization

Pros

- Great compression ratio: $N \log_2 k$ bits
- Very fast search with cached distances
- Sub-linear search with inverted lists.

Cons

- If using inverted indexes, the edge problem requires query expansion.
- If $k << N$, non unique results.
- As d or N increase, k needs to get increasing large. K-means quickly becomes intractable.

Large enough k for sufficient accuracy is computationally infeasible.

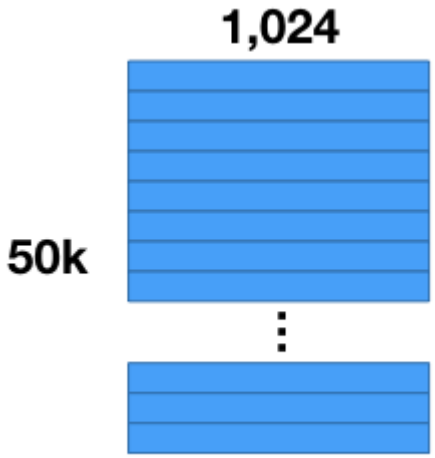
Quantization可以理解

为减少一个变量可能的值

Product Quantization

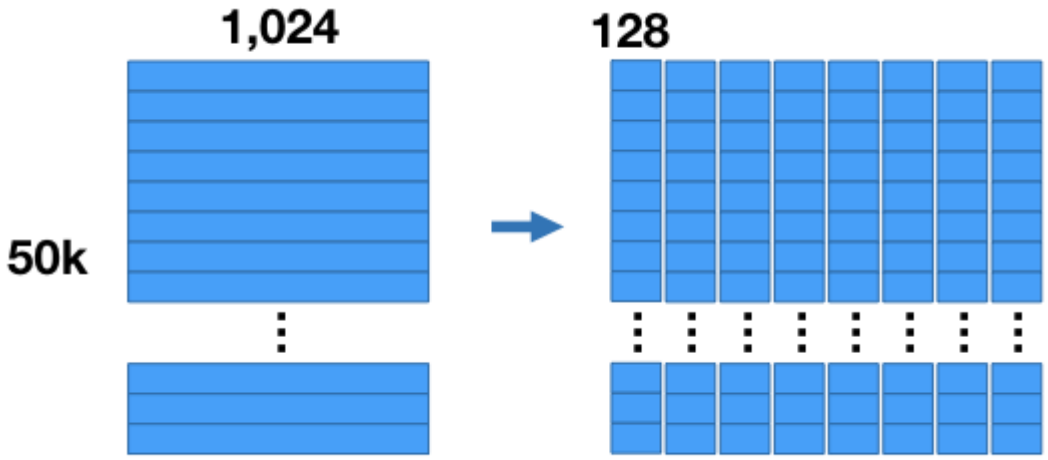
乘积量化

Product Quantization-索引数据



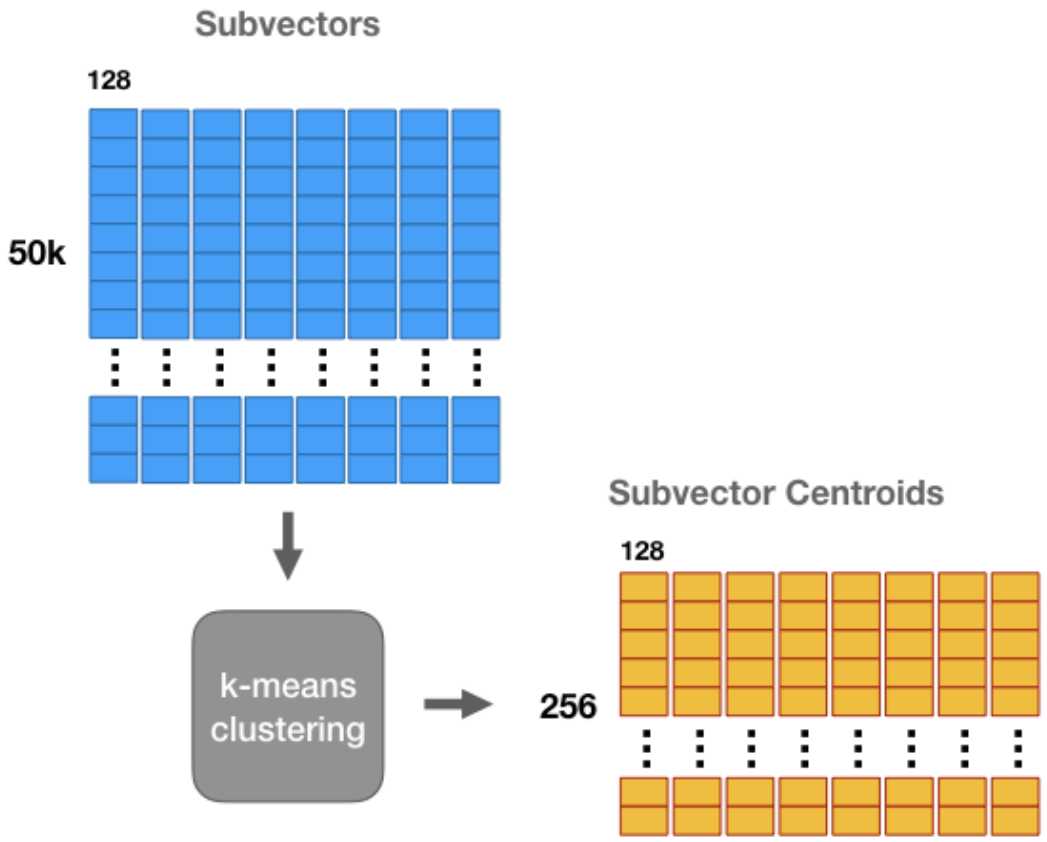
50K张图片，每张图片从神经网络中提取1024维度向量

Product Quantization-索引数据



切分1024维向量为8个124维向量

Product Quantization-索引数据



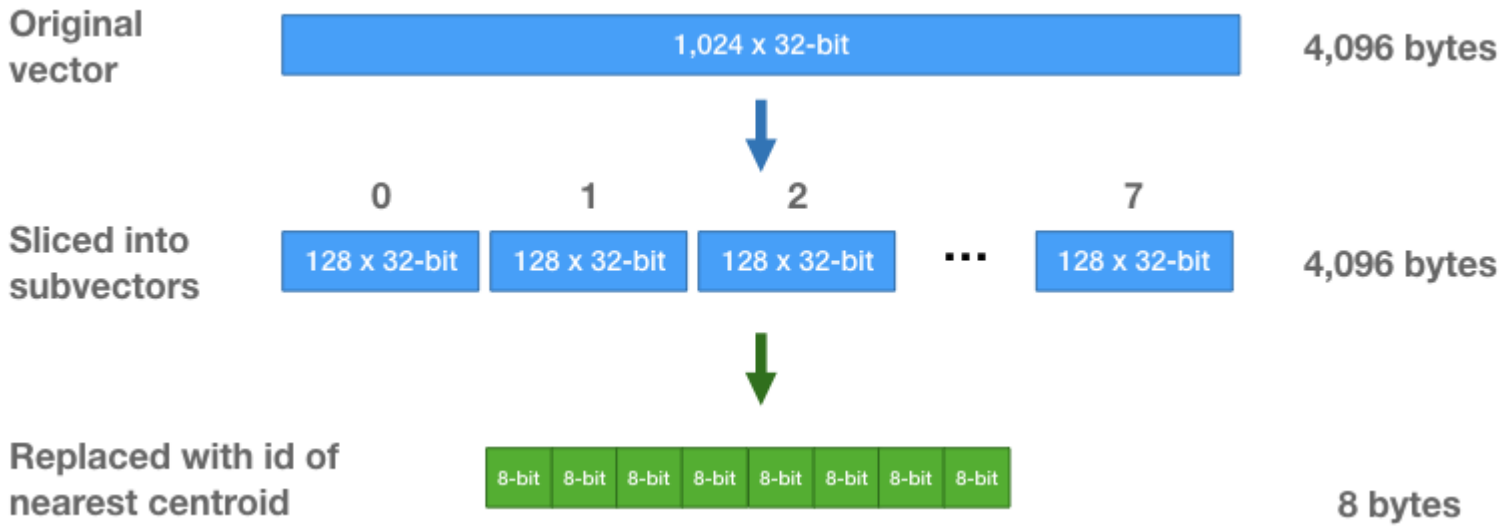
在8个128*N的子区间上分别进行K=256的k-means

8个子区间上每个区间都有256个质心，利用这些质心对原始的50K个向量进行压缩

利用质心来代替一个向量子区间，虽然和原始向量不同，但是还是比较close

不再存储原始的浮点数，而是存储对应的质心

Product Quantization-索引数据



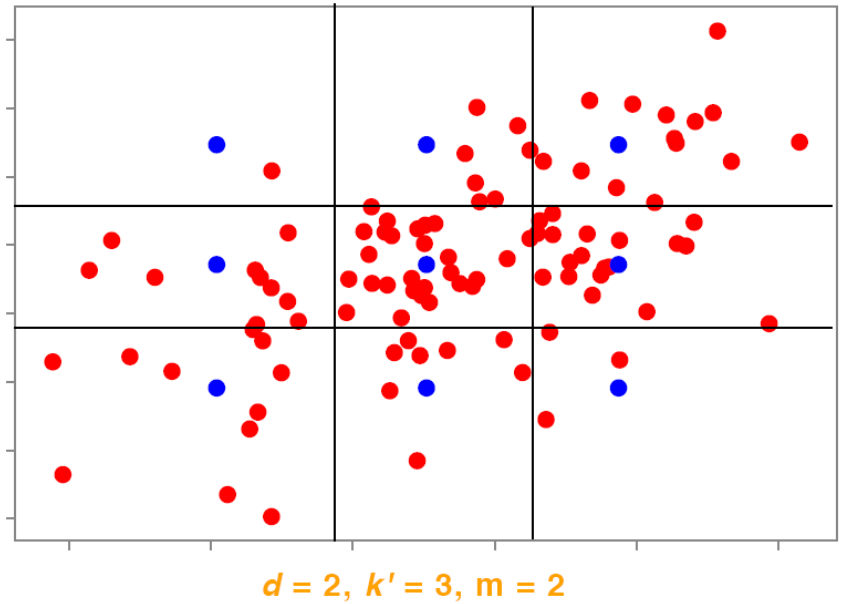
从原始向量的4096字节压缩到8个字节

4096字节=1024维度*浮点数

8字节=8组subvectors*每个subvectors的质心id

Product Quantization 乘积量化

Product Quantization: Example



$d = 2,\ k' = 3,\ \mathrm{m} = 2$

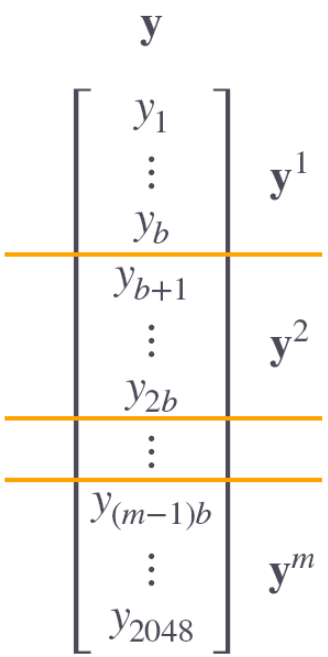
9 centroids instead of 3!

Product Quantization for Nearest Neighbor Search.
Herve Jegou, Matthijs Douze, and Cordelia Schmid.

可以理解为多个维度上的Quantization量化

乘积量化

Product Quantization



- Codebook is Cartesian product of sub-codebooks:

$$C = C^1 \times \ldots \times C^m$$

- Number of centroids very large: k'^m
- *Very* Fast: run k-means m times with low k'

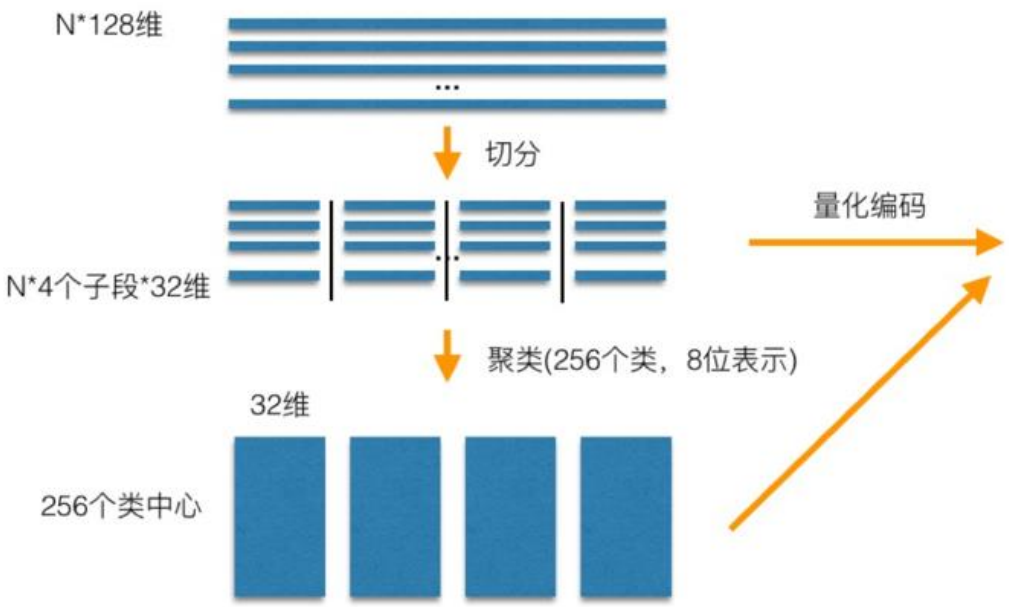
Vastly increase the number of centroids.

Product Quantization for Nearest Neighbor Search.
Herve Jegou, Matthijs Douze, and Cordelia Schmid.

相比于Quantization，增加了质心（聚类中心）的数量

可以快速地在各个子区间上进行K-Means

乘积量化-查询数据



十进制

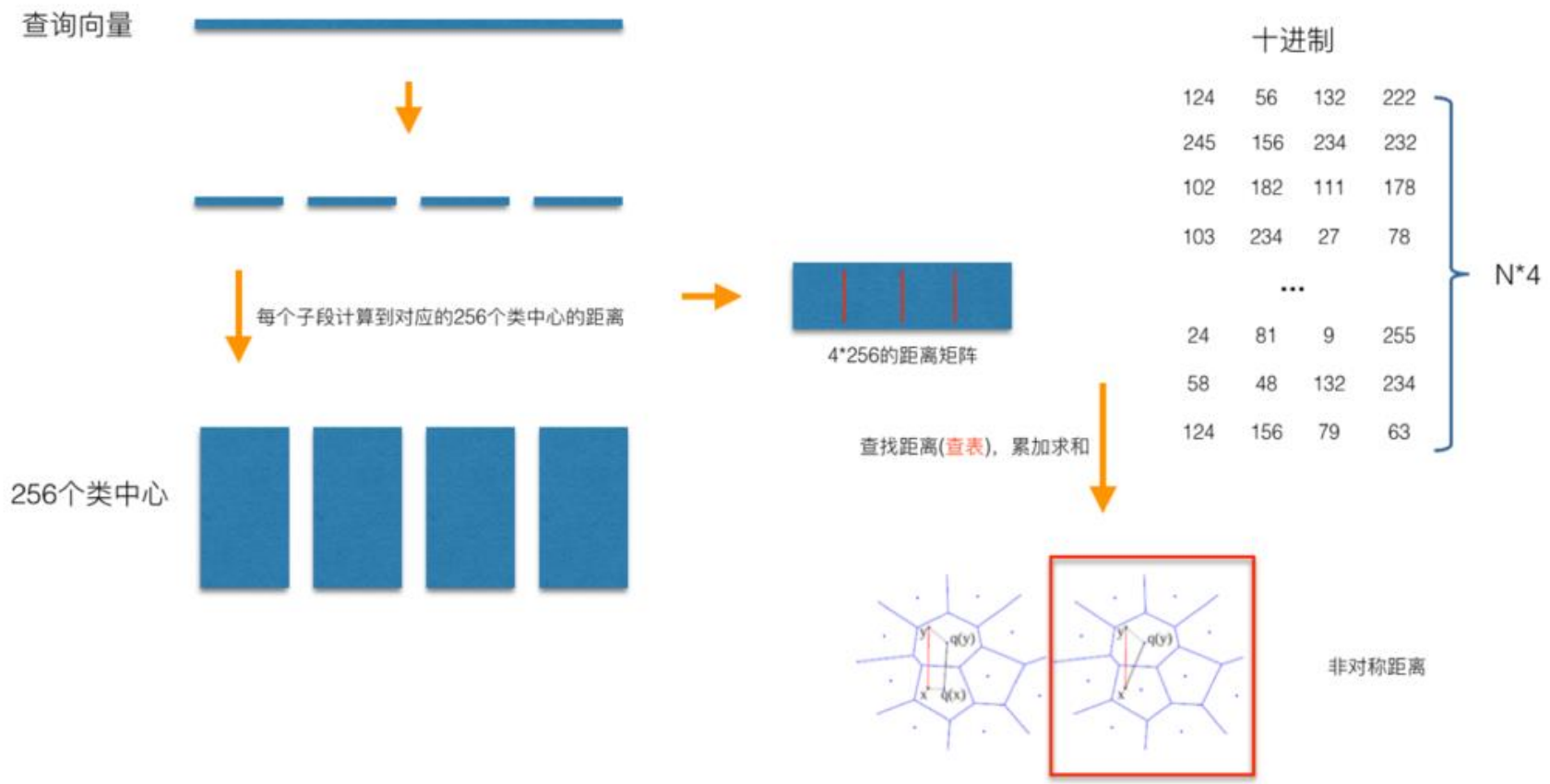
124	56	132	222
245	156	234	232
102	182	111	178
103	234	27	78
24	81	9	255
58	48	132	234
124	156	79	63

N*4

通过这样一种编码方式，训练样本仅使用的很短的一个编码得以表示，从而达到量化的目的。

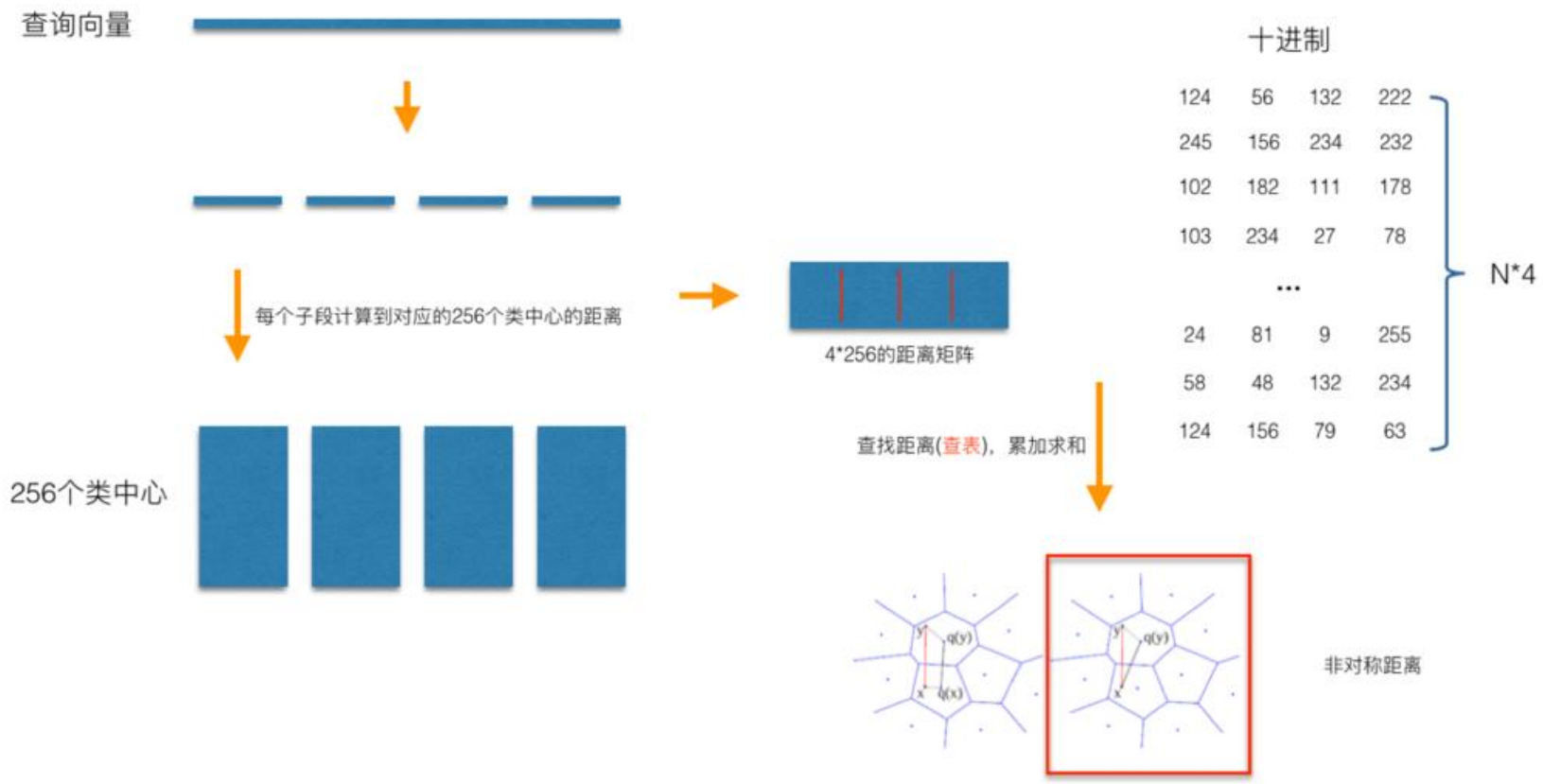
对于待编码的样本，将它进行相同的切分，然后在各个子空间里逐一找到距离它们最近的类中心，然后用类中心的id来表示它们，即完成了待编码样本的编码。

乘积量化-查询数据



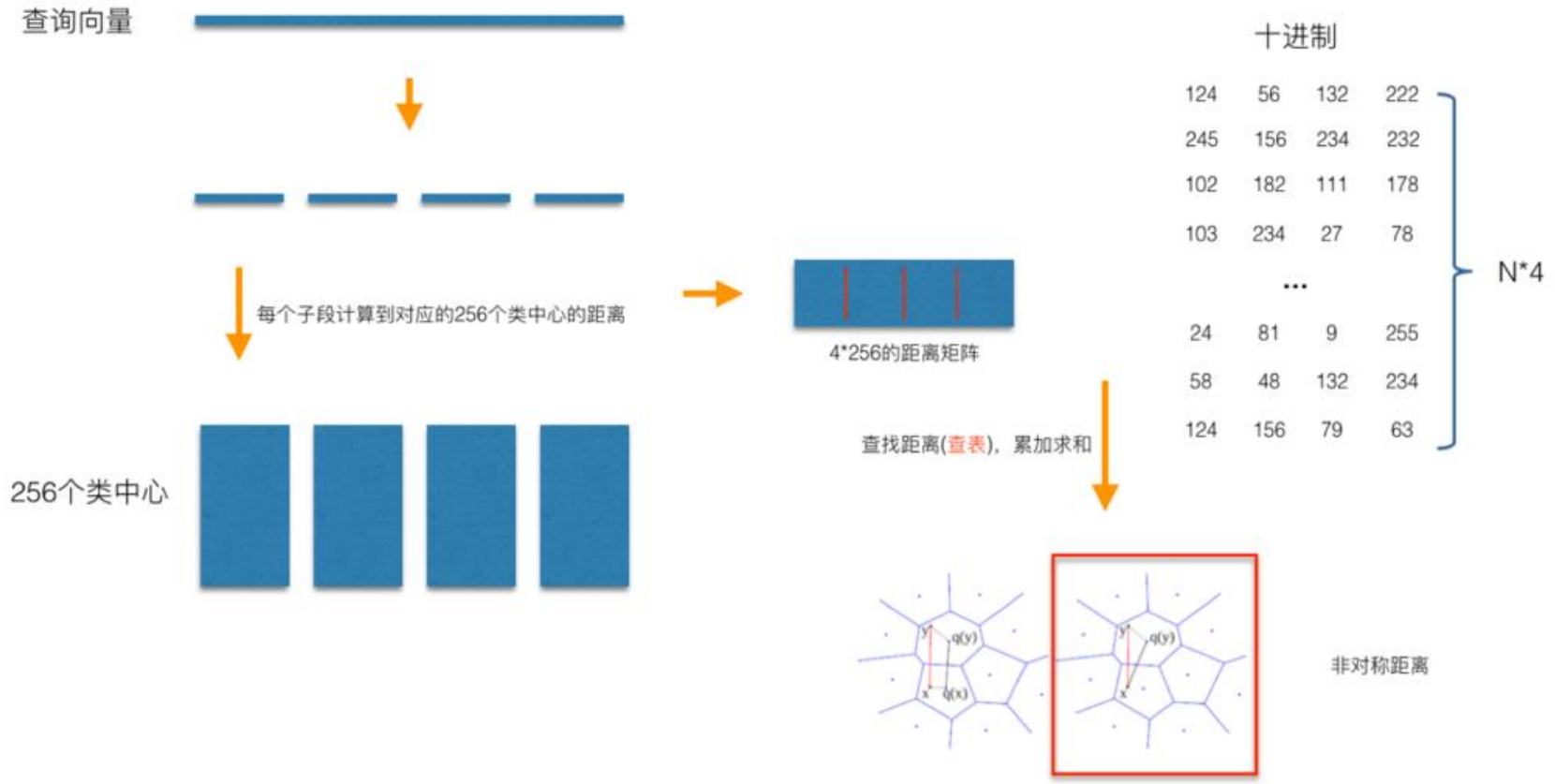
在查询阶段，PQ同样在计算查询样本与已索引数据中各个样本的距离，只不过这种距离的计算转化为间接近似的方法而获得

乘积量化-查询数据



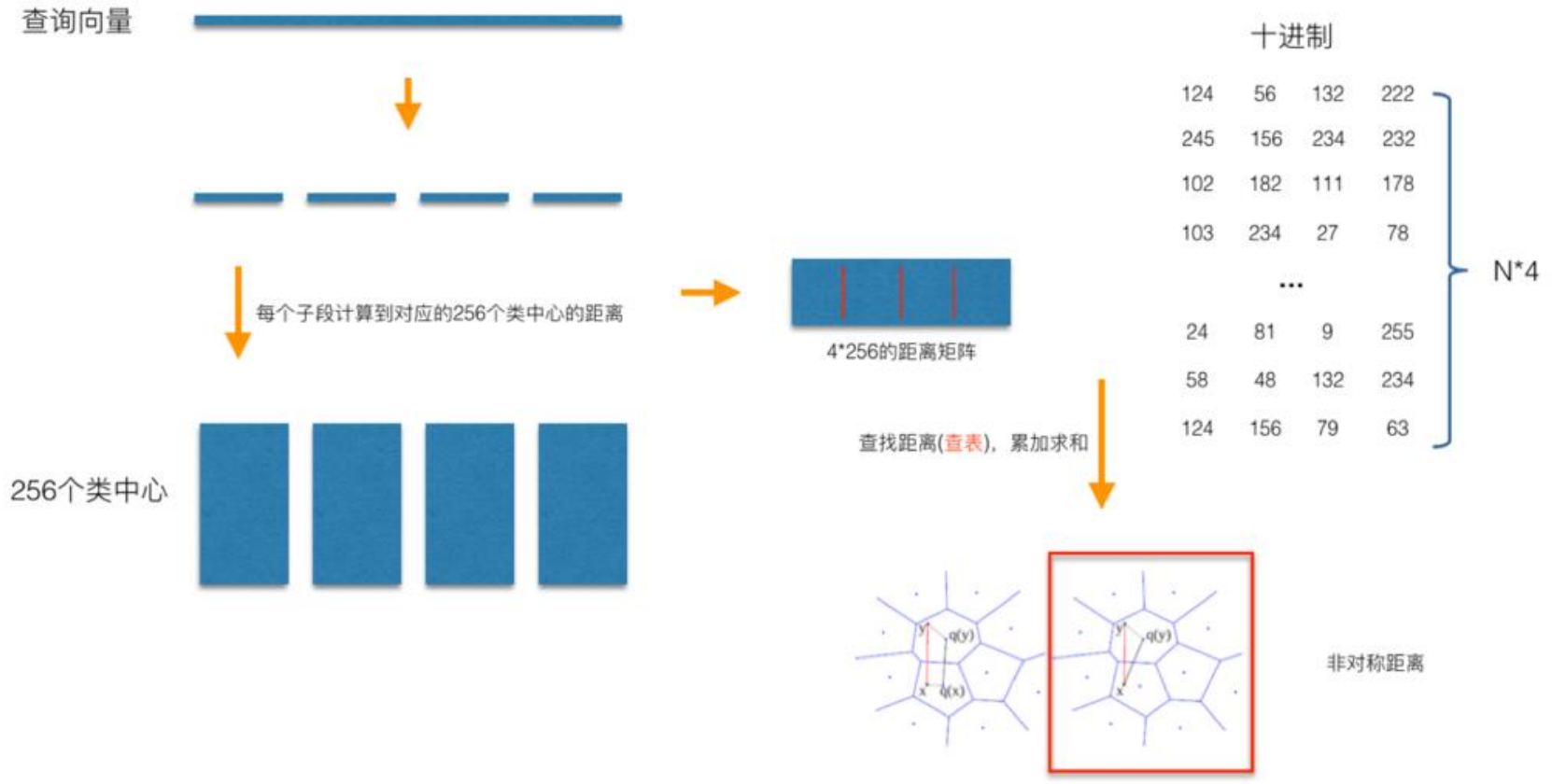
查询向量来到时，按训练样本生成码本的过程，将其同样分成相同的子段，然后在每个子空间中，计算子段到该子空间中所有聚类中心的距离，如图中所示，可以得到4*256个距离，把这些算好的距离称作距离池

乘积量化-查询数据



在计算库中某个样本到的距离时，比如编码为(124, 56, 132, 222)这个样本到查询向量的距离时，我们分别到距离池中取各个子段对应的距离即可，比如编码为124这个子段，在第1个算出的256个距离里面把编号为124的那个距离取出来就可，所有子段对应的距离取出来后，将这些子段的距离求和相加，即得到该样本到查询样本间的距离。所有距离算好后，排序后即得到我们最终想要的结果

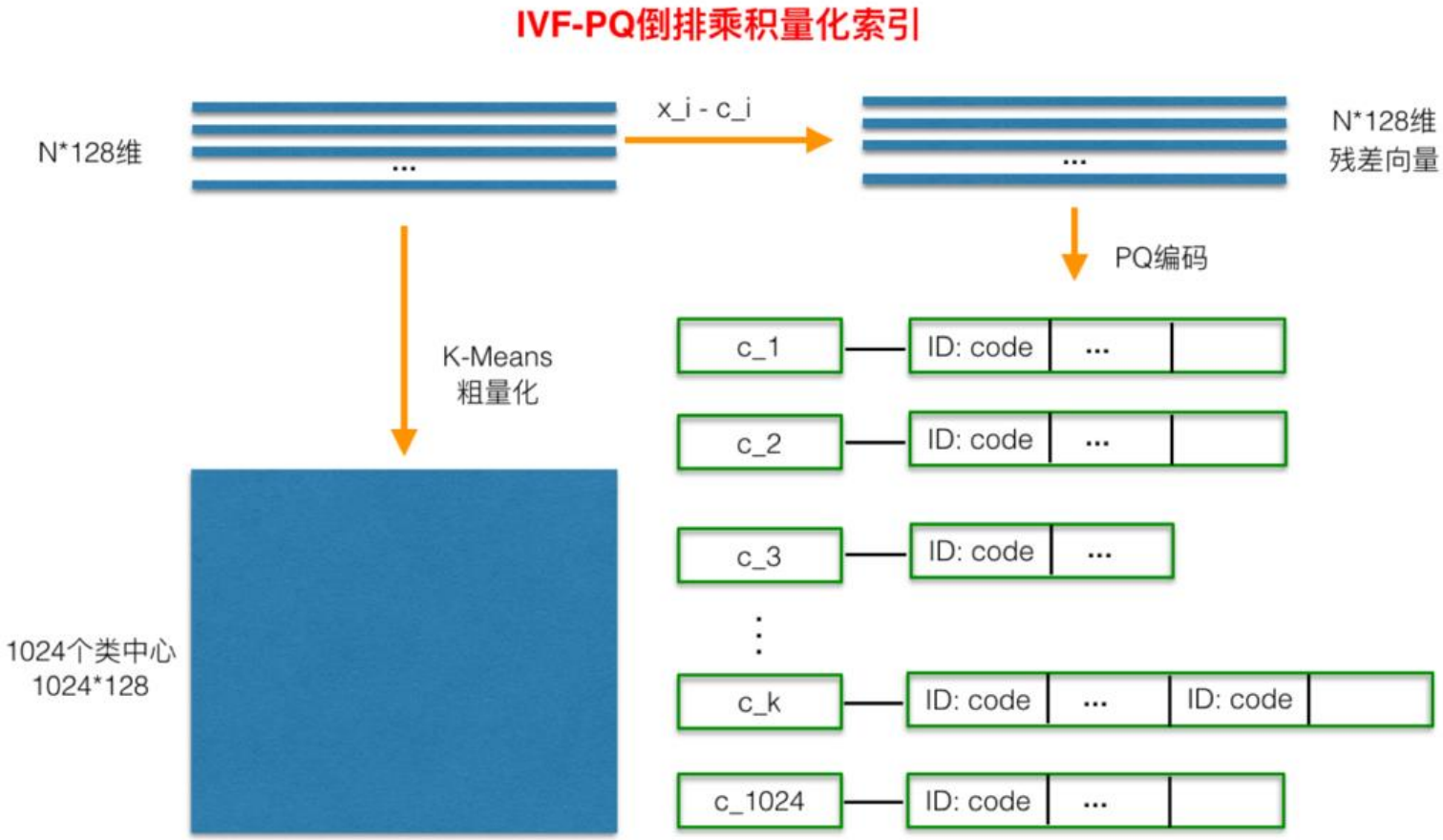
乘积量化-查询数据



PQ乘积量化能够加速索引的原理：全样本的距离计算，转化为到子空间类中心的距离计算。不过这种距离是近似距离

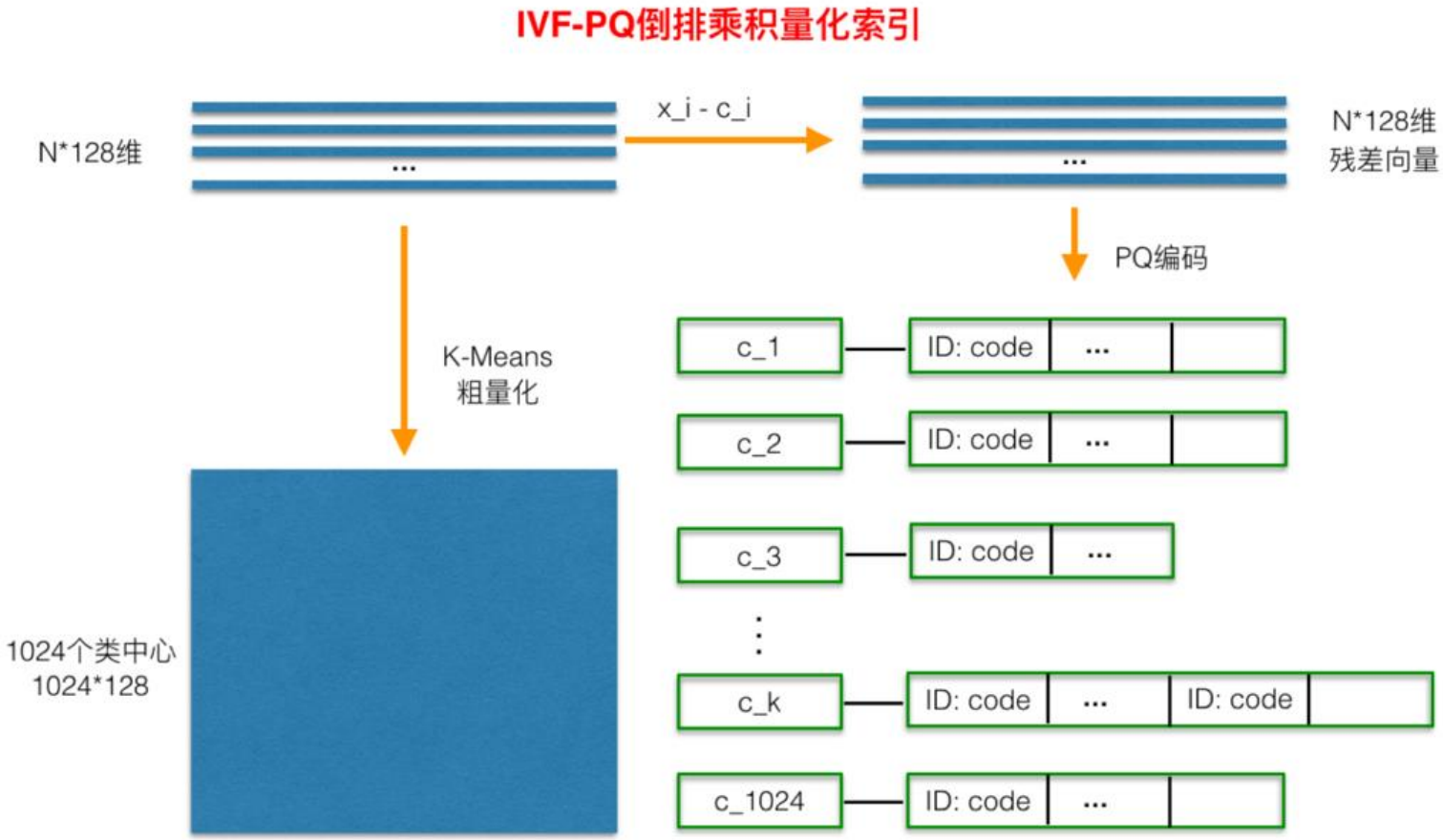
比如上面所举的例子，原本穷举搜索的方式计算距离的次数随样本数目N成线性增长，但是经过PQ编码后，对于耗时的距离计算，只要计算4*256次，几乎可以忽略此时间的消耗

乘积量化-倒排



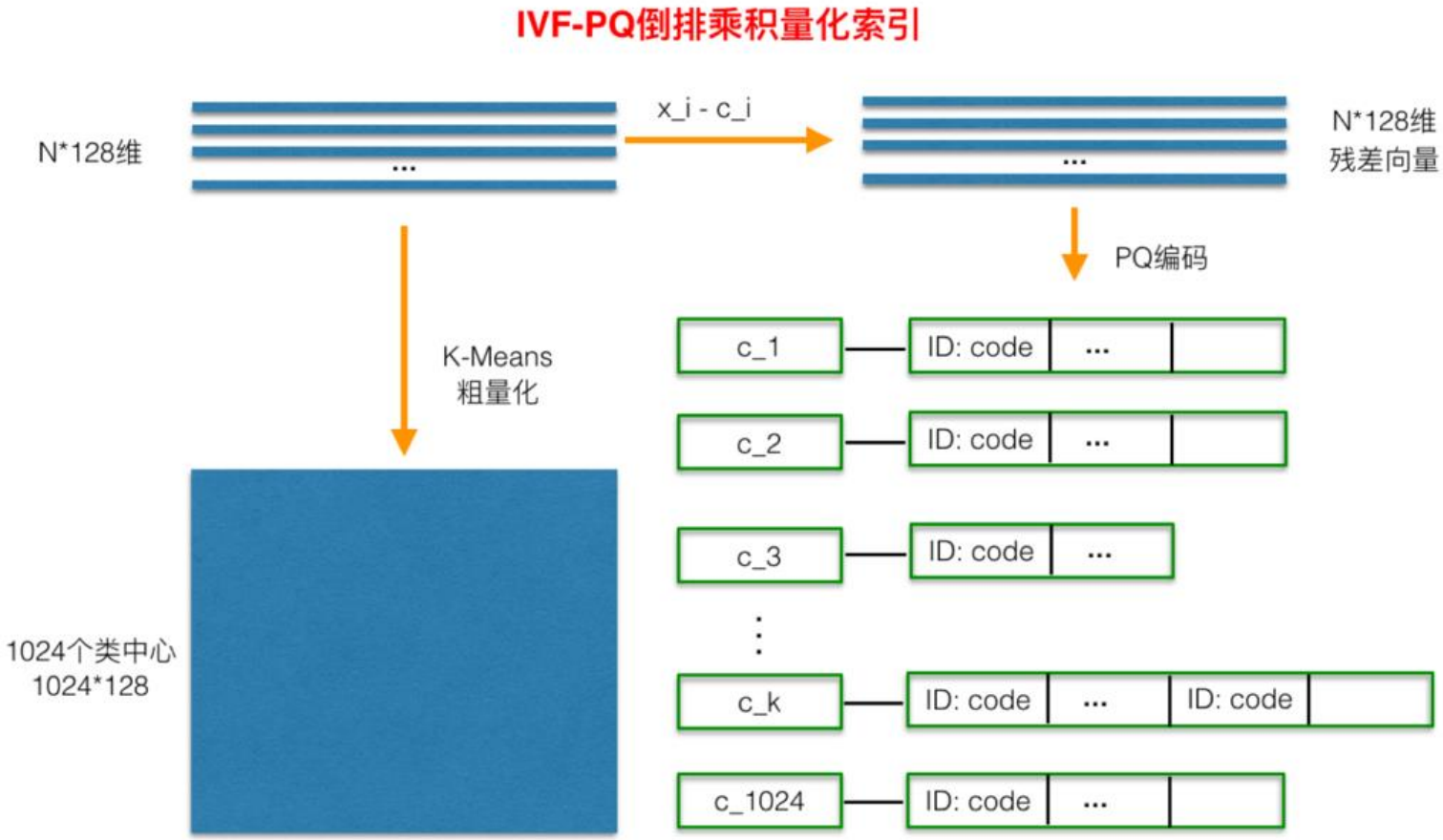
PQ乘积量化计算距离的时候，距离虽然已经预先算好了，但是对于每个样本到查询样本的距离，还是得老老实实挨个去求和相加计算距离。但是，实际上我们感兴趣的是那些跟查询样本相近的样本

乘积量化-倒排



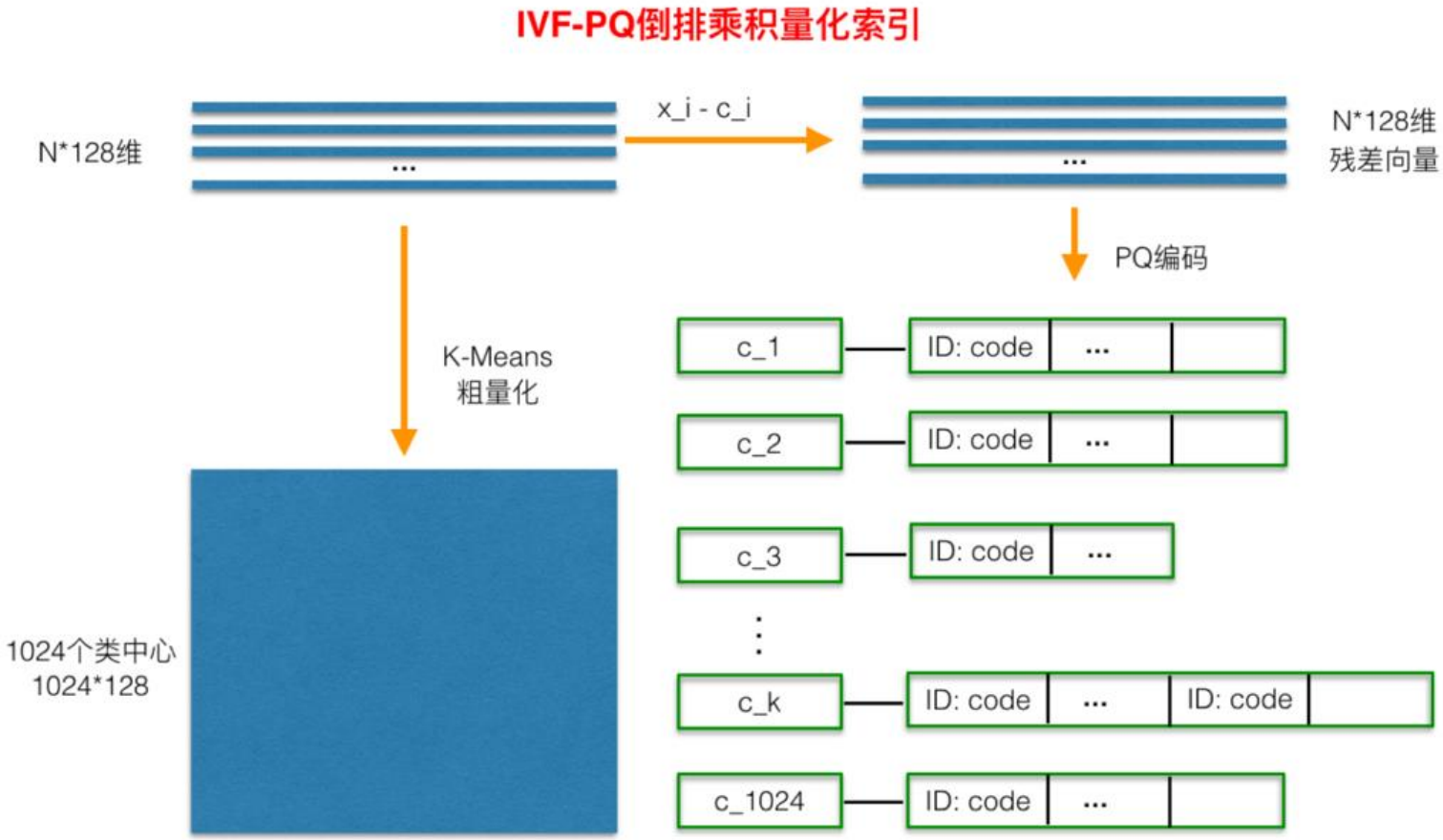
如果能够通过某种手段快速将全局遍历锁定为感兴趣区域，则可以舍去不必要的全局计算以及排序。倒排PQ乘积量化的”倒排“，正是这样一种思想的体现，在具体实施手段上，采用的是通过聚类的方式实现感兴趣区域的快速定位

乘积量化-倒排



在PQ乘积量化之前，增加了一个粗量化过程。具体地，先对N个训练样本采用K-Means进行聚类，这里聚类的数目一般设置得不应过大，一般设置为1024差不多，这种可以以比较快的速度完成聚类过程。得到了聚类中心后，针对每一个样本x_i，找到其距离最近的类中心c_i后，两者相减得到样本x_i的残差向量(x_i-c_i)，后面剩下的过程，就是针对(x_i-c_i)的PQ乘积量化过程，此过程不再赘述

乘积量化-倒排



在查询的时候，通过相同的粗量化，可以快速定位到查询向量属于哪个c_i（即在哪一个感兴趣区域），然后在该感兴趣区域按上面所述的PQ乘积量化距离计算方式计算距离

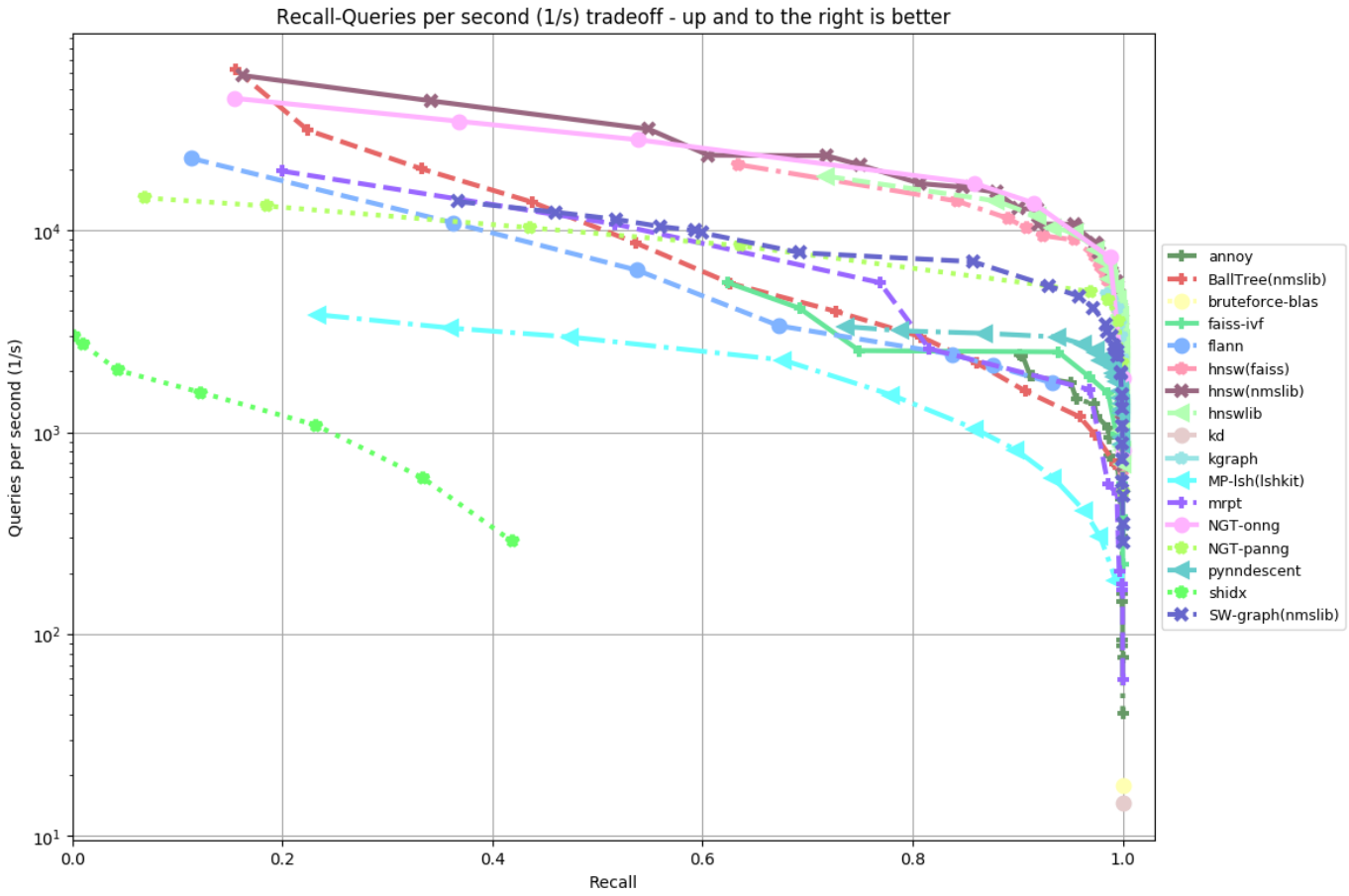
Benchmark

- 用100万条128维SIFT数据（http://corpus-texmex.irisa.fr/）作为索引数据（base向量），每次查询15个向量
- 查询每个query向量L2距离（欧几里得距离）最近的1个base向量（KNN，K=1）平均每次查询耗时0.009秒（非压缩索引建在GPU上）
- 同等情况下通过 Numpy（在CPU上）构建特征数组进行查询需要0.1秒量级的耗时，通过 Faiss 在CPU上进行查询也需要0.15秒左右的耗时

Benchmark

- 每次添加1条向量和每次添加10条向量的耗时都在0.016秒左右（所以最好是批量添加）
- 在同一个图像的相似图像比较多时，支持ANN近似搜索，可以用于人脸特征向量的快速查找

Benchmark



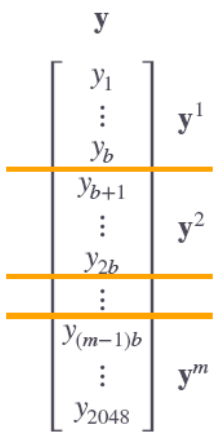
数据集是Fasion-Mnist
每个向量784维
横轴是召回率
纵轴是每秒的查询数

乘积量化

Review: Product Quantization

General Approach

- Split the vector into m sub-vectors, build a quantizer for each with k' centers.



Key Results

- *Very* large number of centroids with low computational cost.
- Storage cost is still very low: $Nm \log_2 k$
- Sub-linear search feasible with inverted lists, coarse quantizers, or multi-indexes.
- Significant improvement over hashing techniques.

小结

- 表示学习
  - 用向量表示图像
- 向量搜索
  - Optimized Product Quantization
  - Locally Optimized Product Quantization
  - ……

参考

- https://compose.com/articles/elasticsearch-query-time-strategies-and-techniques-for-relevance-part-i/
- https://compose.com/articles/elasticsearch-query-time-strategies-and-techniques-for-relevance-part-ii/
- https://www.compose.com/articles/how-scoring-works-in-elasticsearch/

参考

- Lucene 查询原理
- 基于Lucene查询原理分析Elasticsearch的性能
- 剖析Elasticsearch的IndexSorting:一种查询性能优化利器
- 时间序列数据库的秘密 (2)——索引

参考

- https://zhuanlan.zhihu.com/p/51201097
- https://www.youtube.com/watch?v=AQau4-VF64w&index=13&list=PLuRT8pAOHZrzGDI236hCtpM5NTgJNyvsm&t=0s
- http://yongyuan.name/blog/ann-search.html
- Product Quantizers for k-NN Tutorial part 1

参考

- http://yongyuan.name/blog/cbir-technique-summary.html
- MySQL索引背后的数据结构及算法原理
- https://github.com/facebookresearch/faiss
- http://mccormickml.com/2017/10/22/product-quantizer-tutorial-part-2/

参考

- 一次“高大上”模型的落地“失败”吐槽
- lucene字典实现原理
- http://www.hankcs.com/program/algorithm/aho-corasick-double-array-trie.html
- http://cs231n.github.io/transfer-learning/#tf

参考

- https://www.slideshare.net/CraigMacdonald/efficient-query-processing-infrastructures