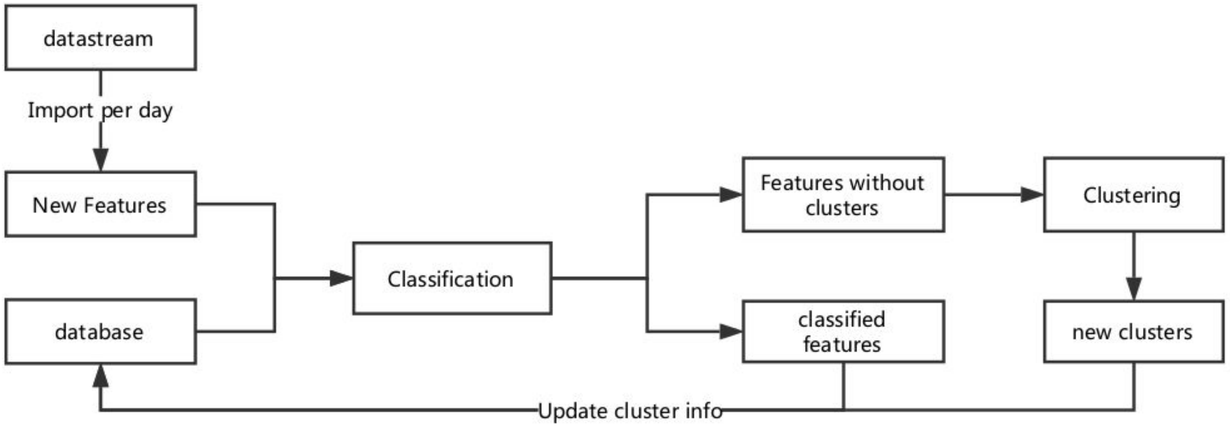
监控人脸聚类

窦浩轩-研究院 智能视频

Problem Description & Motivations

- 增量聚类描述
  - 每日输入一批监控人脸图像特征
  - 底库中存放已经累积的监控人脸图像特征, 假定底库人脸图像已经含有类别信息
  - 增量聚类的目的是给新输入的人脸图像特征赋予类别信息, 类别可能是底库中已有的类别, 或者产生全新的类别
  - 新输入特征被赋予类别信息后与底库人脸图像进行类别合并
- 增量聚类通常分为两步:
  - 将新输入数据归类到已有类别
  - 将无法归类到已有类别的数据进行聚类产生新的类别



Problem Description & Motivations

- 底层聚类算法基本需求
    - Easy to generalize to incremental scenario
    - Scalable, at least to billions of images and hundreds of millions of clusters
    - Distributable
    - Memory-efficient
    - Good guarantee on the homogeneity of the cluster result

Incremental Clustering

Distributed incremental clustering using KNN search

Design Goal

- 1 billion clusters
- 100 million features per day
- good homogeneity and no guarantee about completeness
- linear scalability for less than 100 gpu devices
- fault tolerance for recoverable failures
- in-memory computing

Motivations

- 人脸检索
  - 提升检索的召回率 : 利用监控人脸多模态之间的相似性
  - 建立一人一档机制 , 有更大的应用空间 (轨迹追踪)

Motivations

- 人脸检索
  - 提升检索的召回率：利用监控人脸多模态之间的相似性，提供更多信息
  - 建立一人一档机制，有更大的应用空间(轨迹追踪)

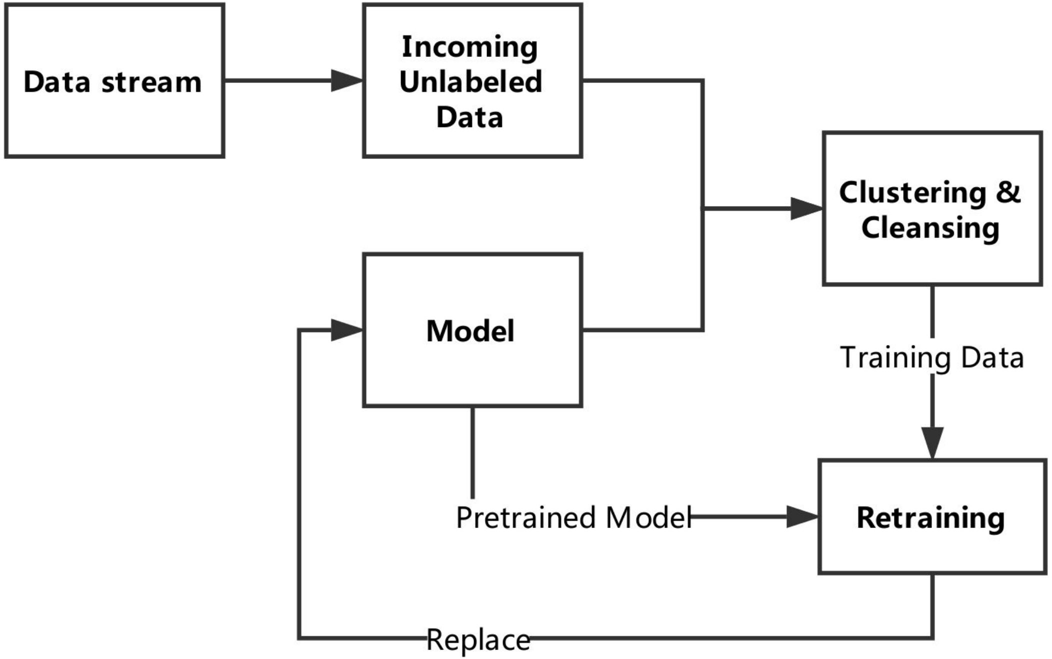
Search:

Results: (stats)

rank	score	id	face	matched pedestrian	details
0	0.90857846	{"camera_id":{"region_id":106,"camera_idx":1},"captured_time":"2018-06-20T03:02:13.000Z","sequence":389}			Show Details
1	0.8909461	{"camera_id":{"region_id":105,"camera_idx":111},"captured_time":"2018-06-20T03:53:34.000Z","sequence":275}			Show Details
2	0.87588704	{"camera_id":{"region_id":106,"camera_idx":1},"captured_time":"2018-06-20T03:10:27.000Z","sequence":115}			Show Details
3	0.86521745	{"camera_id":{"region_id":105,"camera_idx":111},"captured_time":"2018-06-20T03:44:44.000Z","sequence":335}			Show Details

Motivations

- 增量训练
  - 人脸聚类可为监控数据流提供自动标注
  - 布控模型可使用自动标注的监控数据流不间断进行增量训练
  - 每一次训练强化模型的聚类能力，从而提升下一次自动标注的质量
  - 增量训练形成正反馈循环，不断提高布控模型的performance

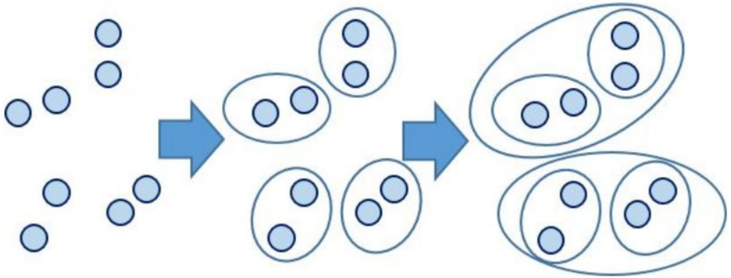


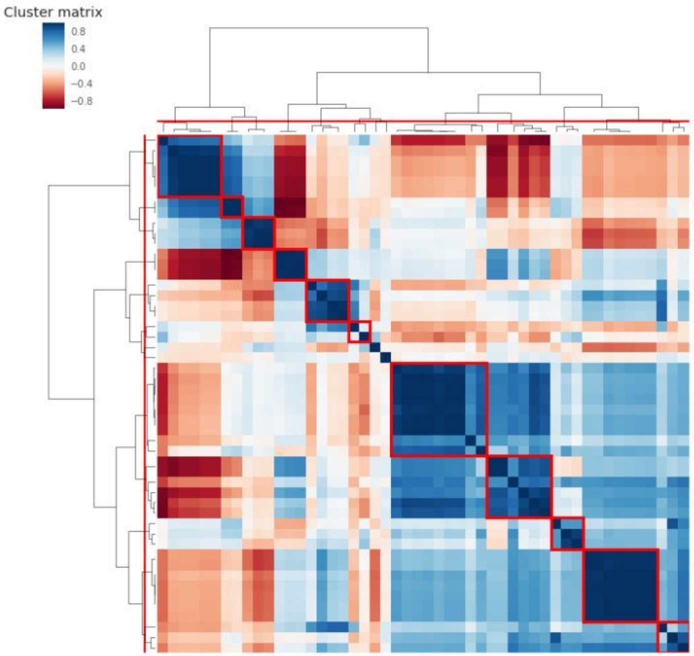
Related works

- Incremental K-means
- Affinity propagation
- Spectral clustering: poor scalability for above
- Hierarchical clustering
- Density-based clustering
- . . .

Hierarchical Clustering

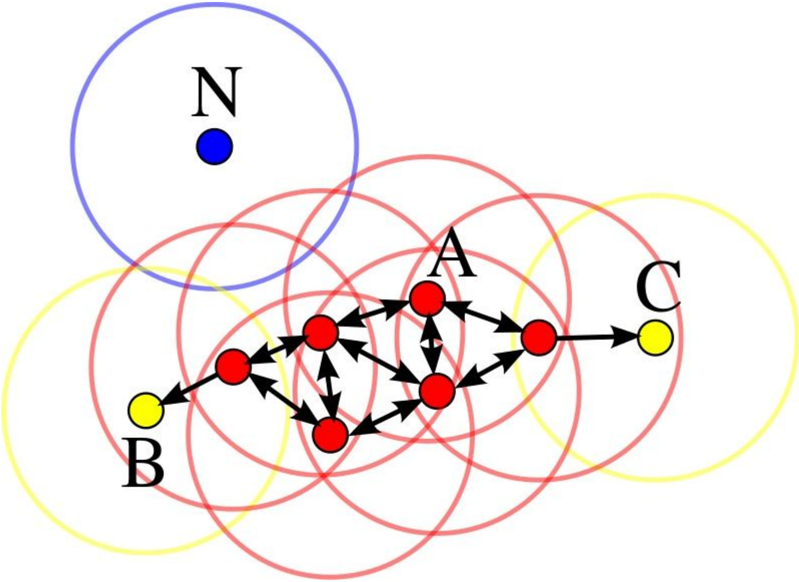
- Starts with clusters of one element, and gradually combine clusters that are close to each other based on a predefined distance metric.
- Need to keep the similarity matrix, and become computationally prohibitive as scale goes to hundreds of millions without connectivity constraints.





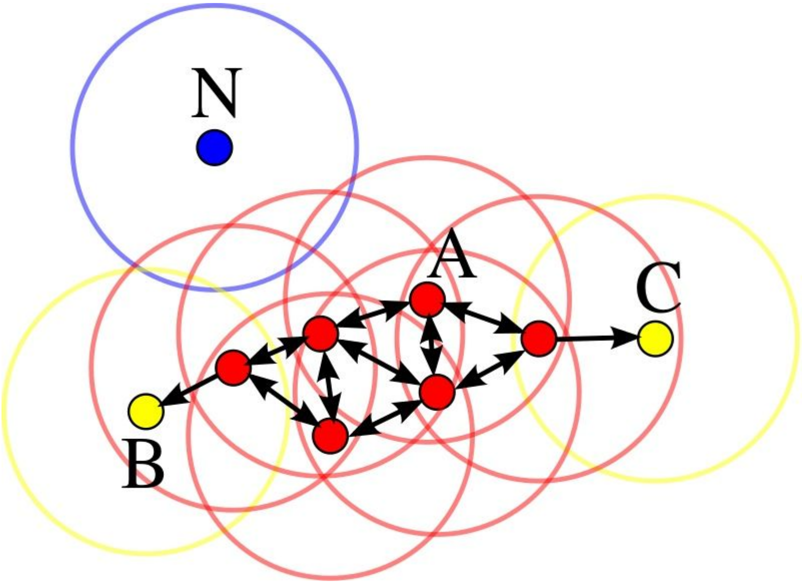
DBSCAN: Density-based Clustering

- A point p1 is neighbor to point p2 if p1 is in p2's **epsilon-neighborhood**
- A point A is **a core point** if it has more than **minPts** neighbors
- Two points B and C are **reachable** to each other if a sequence of core points p1, …, pT exist such that B is neighbor to p1 and C is neighbor to pT, and pi is neighbor to pj for |i-j| = 1

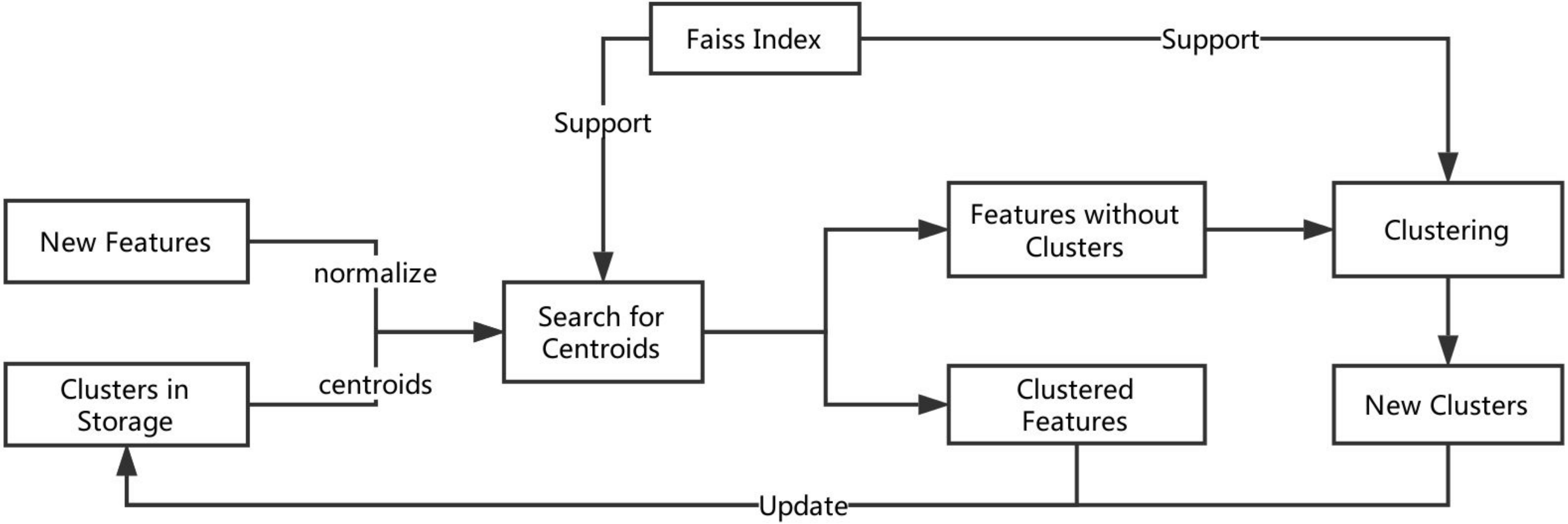


DBSCAN: Density-based Clustering

- Does not have to calculate whole similarity matrix, memory efficient
- Computation scalable
- We implement a loose version without minPts constraint using approximate knn search and affinity graph



The Current Overall Pipeline



人脸增量聚类分为两步:

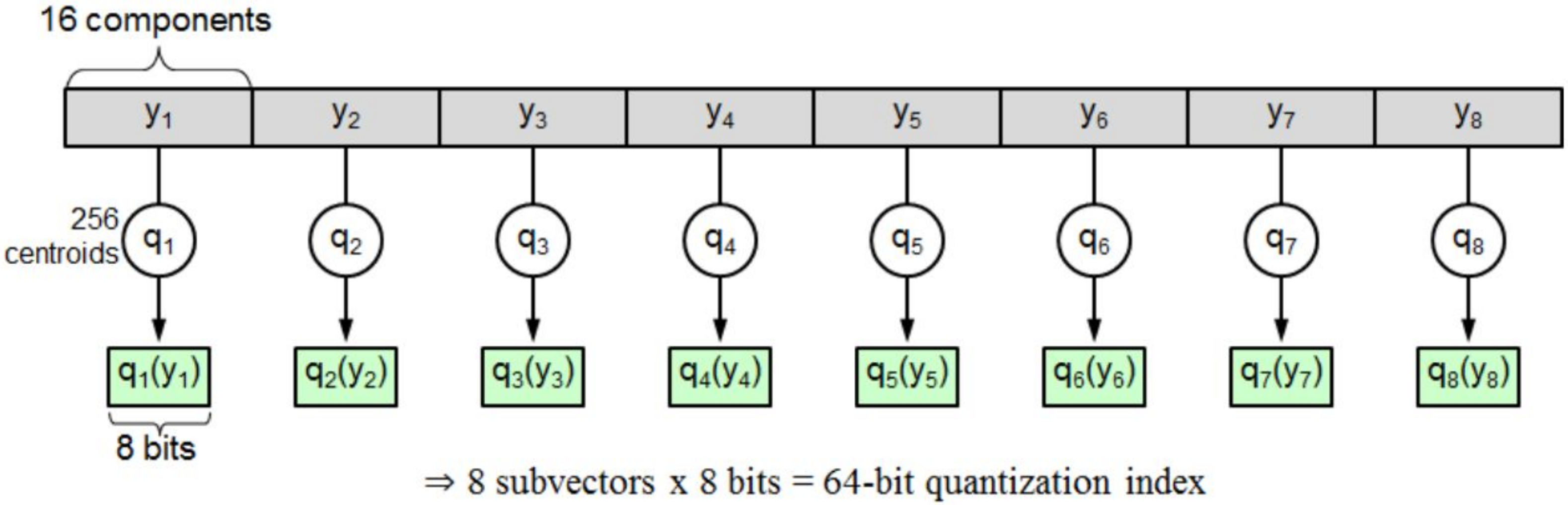
1. 新输入特征与底库类中心进行最近邻搜索, 并通过对距离卡阈值确定所属类别(属于底库中某一类 or 不属于现有类别), 更新底库类中心
2. 对所有无类别的特征进行聚类, 确定类别, 将新聚类中心加入到底库类中心中

Approximate KNN Search by FAISS Index

- 算法目标: 给定X为底库数据, Y为query数据, 期望为Y中每一条向量在X中尽快找到近似的k近邻
- IVFADC算法使用两层量化: coarse quantizer用于划分搜索空间并对向量进行粗略量化; product quantizer用于精细量化向量, 从而减少search的计算量
- 使用kmeans算法作为coarse quantizer, 将底库中的数据Y分成k个簇, 经过粗略量化后, 计算每条向量与量化中心的残差, 对残差进行product quantization

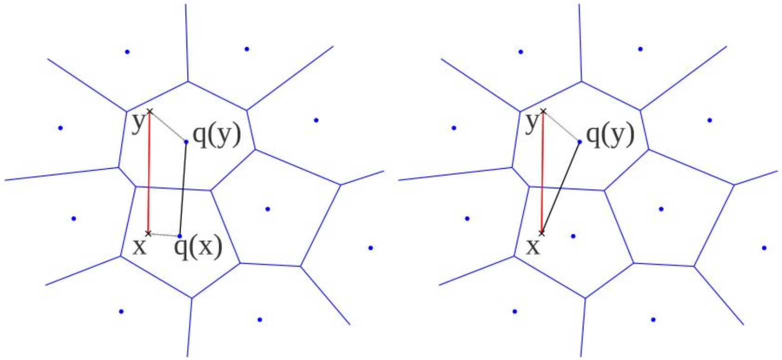
Approximate KNN Search by FAISS Index

- Product quatization 将D维向量沿维度分成M个子向量并在每一份上进行coarse quantization, 将D维向量压缩到M维



Approximate KNN Search by FAISS Index

- 使用asymmetric distance computation计算query向量与底库向量的近似距离



symmetric case　　　　asymmetric case

Fig. 2. Illustration of the symmetric and asymmetric distance computation. The distance $d(x, y)$ is estimated with either the distance $d(q(x), q(y))$ (*left*) or the distance $d(x, q(y))$ (*right*). The mean squared error on the distance is on average bounded by the quantization error.

$$\|x - q(y)\|_2^2 = \|x - q_1(y) - q_2(y - q_1(y))\|_2^2.$$

$$\underbrace{\|q_2(\ldots)\|_2^2 + 2\langle q_1(y), q_2(\ldots)\rangle}_{\text{term 1}} + \underbrace{\|x - q_1(y)\|_2^2}_{\text{term 2}} \underbrace{-2\langle x, q_2(\ldots)\rangle}_{\text{term 3}}.$$

- 量化中心之间的距离与norm, 也就是term 1, 可事先计算, 所以计算近似距离只需计算查询向量与所有子向量量化中心的距离，并且查表累加计算即可

- 使用底库训练量化器, 并对底库向量进行量化并存入GPU中
- 搜索时计算查询向量与量化后底库向量的近似欧式距离 , 并通过排序找出k近邻
- 对于每次使用faiss index进行approximate knn search返回的topk结果, 会在topk中再进行一次精确的内积距离计算, 以修正近似搜索的误差
- O(N*m*w/k + N*d)

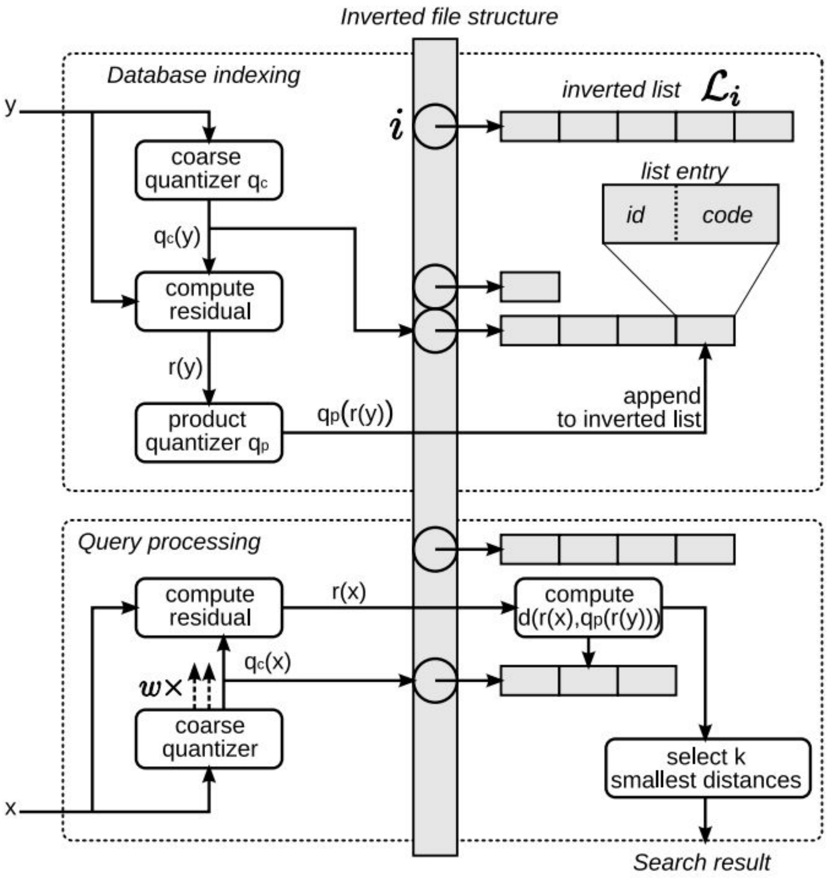


Fig. 5. Overview of the *inverted file with asymmetric distance computation* (IVFADC) indexing system. *Top*: insertion of a vector. *Bottom*: search.

Clustering in details

- 根据knn search返回的topk similarity(cosine distance), 来绘制affinity graph, 并通过在affinity graph上寻找连通量来确定聚类
    - 具体实现是通过DFS递归染色来进行, 对于similarity大于阈值(通常为0.6~0.7)的, 进行染色, 对于similarity小于阈值的跳过
    - 工程化实现使用Spark Graph中的 connectedComponents
    - O(k*N)

```python
num_cent = -1
# 递归标记连通
def _scc(i, c):
    q = deque()
    q.append(i)
    I[i] = c
    while len(q) > 0:
        j = q.pop()
        for k,d in zip(new_cent[j], distance[j]):
            if I[k] == -1:
                if d > threshold:
                    I[k] = c
                    q.append(k)
                else:
                    break

t0 = time.time()
for i in range(len(I)):
    if I[i] == -1:
        num_cent += 1
        _scc(i, num_cent)
t1 = time.time()
print "Time elapsed for connecting: %.3f s"%(t1-t0)
```

性能指标

- B-cubed precision-recall = averaged per-item precision-recall

- Precision indicates homogeneity 同质性
- Recall indicates completeness 完备性
- 聚类算法应当在保证同质性足够好的情况下,尽可能提高完备性
- 聚类测试使用一个已标注的小probe测试集混入干扰集后进行聚类, 并在probe集上计算B-cubed precision-recall

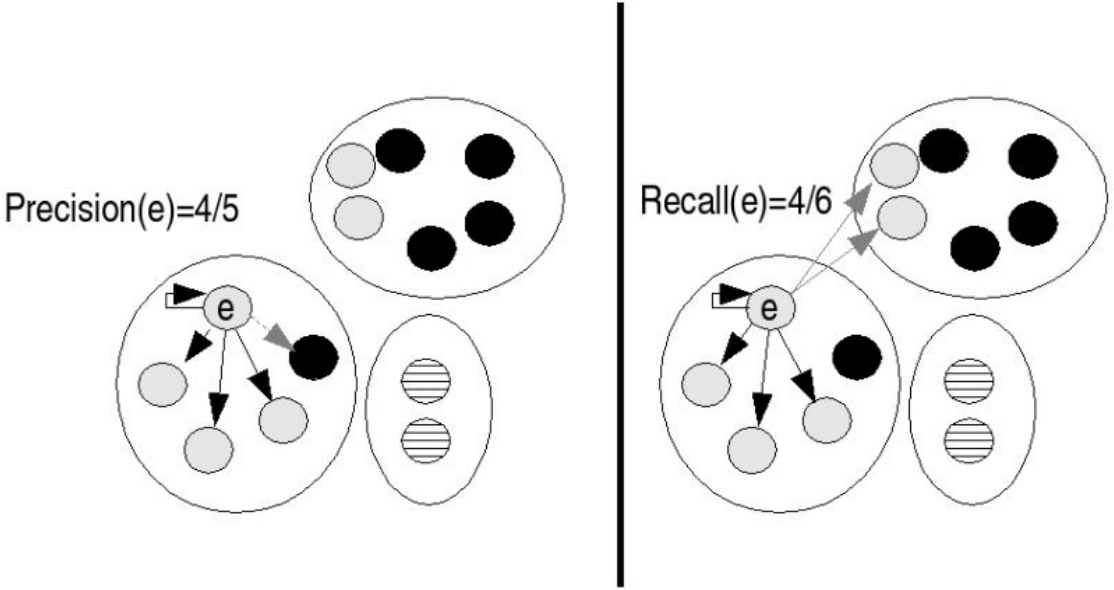


Figure 10: Example of computing the BCubed precision and recall for one item

性能指标

GPU: V100*1	CPU:Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz				Faiss index: IVF1024,PQ32	
order of distractor	training(s)	adding(s)	searching(s)	searching per item(ms)	precesie searching(s)	connecting(s)
1000w	12.162	19.587	36.104	0.004	196.472	159.081
2000w	21.917	37.996	90.269	0.005	405.277	358.609
3000w	27.716	49.056	152.852	0.005	626.701	995.403
4000w	32.276	65.515	233.884	0.006	787.458	1699.562
5000w	39.771	77.697	328.535	0.007	987.063	713.966
6000w	47.844	94.18	802.098	0.013	1430.137	885.577
GPU: V100*8	CPU:Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz				Faiss index: IVF262144,PQ32	
1000w	1700.076	2467.682	2489.59	0.249	207.789	190.134
2000w	3394.072	4536.591	4580.274	0.229	419.808	475.241
3000w	5484.167	7022.083	7088.769	0.231	646.616	1333.365

性能指标

使用大约30w张图, 6464个类组成的probe聚类测试集, 混入不同数量级的干扰集

order of distractor	Faiss index	recall	precision
1000w	IVF1024,PQ32	0.37991	0.983059
2000w	IVF1024,PQ32	0.31262	0.983489
3000w	IVF1024,PQ32	0.283941	0.983477
1000w	IVF262144,PQ32	0.480993	0.978951
2000w	IVF262144,PQ32	0.641345	0.978849
3000w	IVF262144,PQ32	0.709545	0.978985