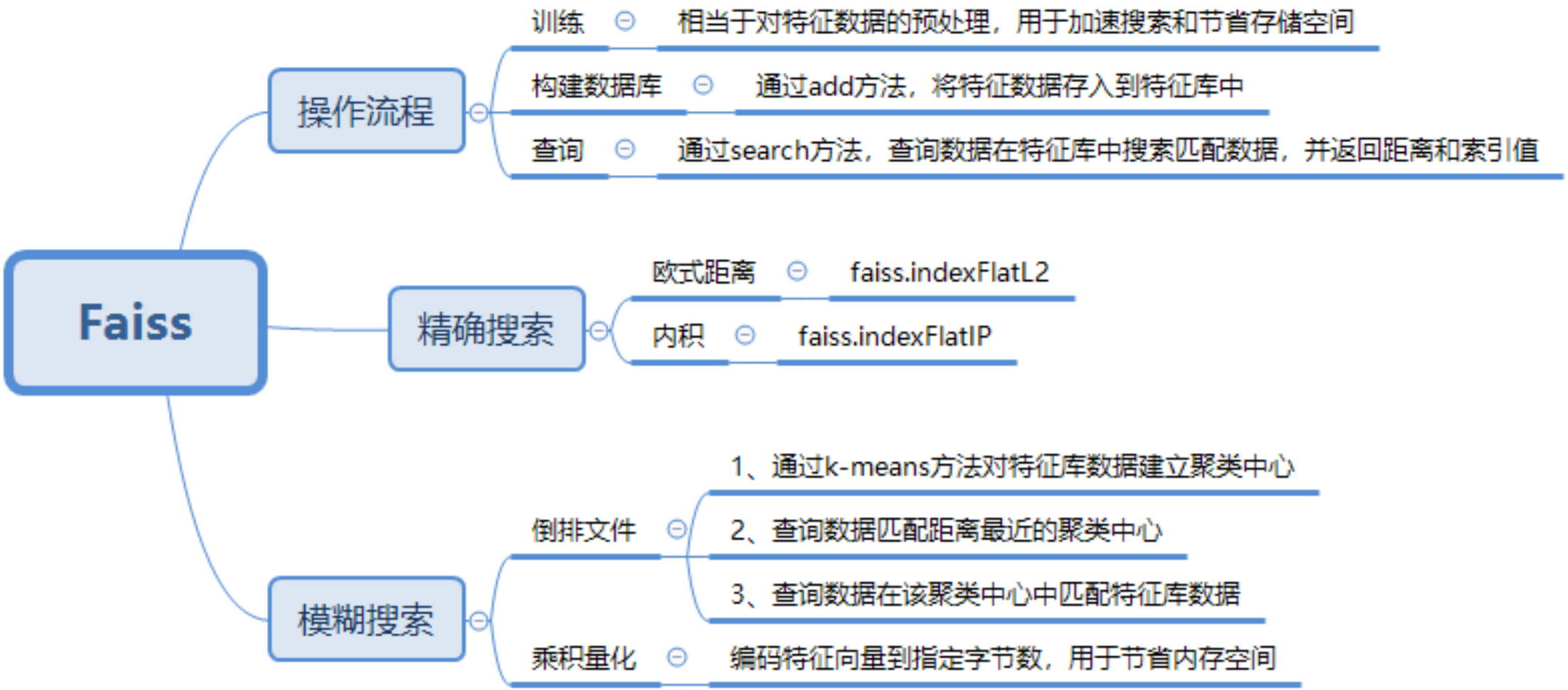
Faiss

Facebook AI Similarity Search

Faiss

- Faiss 是一个开源库，针对高维空间中海量数据，提供了高效且可靠的检索方法
- 解决2个暴力检索算法存在的问题：1）降低内存占用
  2）加快检索速度

Faiss


Faiss
操作流程
训练
相当于对特征数据的预处理，用于加速搜索和节省存储空间
构建数据库
通过add方法，将特征数据存入到特征库中
查询
通过search方法，查询数据在特征库中搜索匹配数据，并返回距离和索引值
精确搜索
欧式距离
faiss.indexFlatL2
内积
faiss.indexFlatIP
模糊搜索
倒排文件
1、通过k-means方法对特征库数据建立聚类中心
2、查询数据匹配距离最近的聚类中心
3、查询数据在该聚类中心中匹配特征库数据
乘积量化
编码特征向量到指定字节数，用于节省内存空间

精确索引

```python
d = 64       # 特征向量维度
nb = 10000   # 特征向量数目
nq = 5      # 查询向量数目
k = 4        # 查询距离最近的k个向量
np.random.seed(1234)
xb = np.random.random((nb, d)).astype('float32')  # 随机生成特征向量组
xq = np.random.random((nq, d)).astype('float32')  # 随机生成查询向量组


index = faiss.IndexFlatL2(d)    # 建立索引，搜索方法为欧式距离
# index = faiss.IndexFlatIP(d)    # 搜索方法为内积

# index.train(xb)  精准搜索不需要训练
index.add(xb)
start_time = time.time()
D, I = index.search(xq, k)
consume_time = time.time() - start_time

print('耗时：', consume_time)
print('最近距离：', D)
print('对应索引：', I)
```

```
耗时：  0.05835223197937012
最近距离：  [[5.0900965 5.670616  5.7184644 5.765298 ]
 [6.8587008 6.956443  7.088391  7.100714 ]
 [5.277209  5.452718  5.79651   5.9845552]
 [5.809805  6.09425   6.100406  6.1200604]
 [5.14467   5.5545816 5.785306  5.794691 ]]
对应索引：  [[1204 3271 2568 7824]
 [8063 2700  919 6738]
 [3919 8653 4130 6471]
 [4429  230  317 3062]
 [9103  199 6044 8418]]
```

倒排文件

```
nlist = 50   # 聚类中心个数
k = 4
quantizer = faiss.IndexFlatL2(d)
index = faiss.IndexIVFFlat(quantizer, d, nlist, faiss.METRIC_L2)
index.train(xb)
index.nprobe = 5     # 查找聚类中心的个数，默认为1
index.add(xb)
start_time = time.time()
D, I = index.search(xq, k)
consume_time = time.time() - start_time

print('耗时：', consume_time)
print('最近距离：', D)
print('对应索引：', I)
```

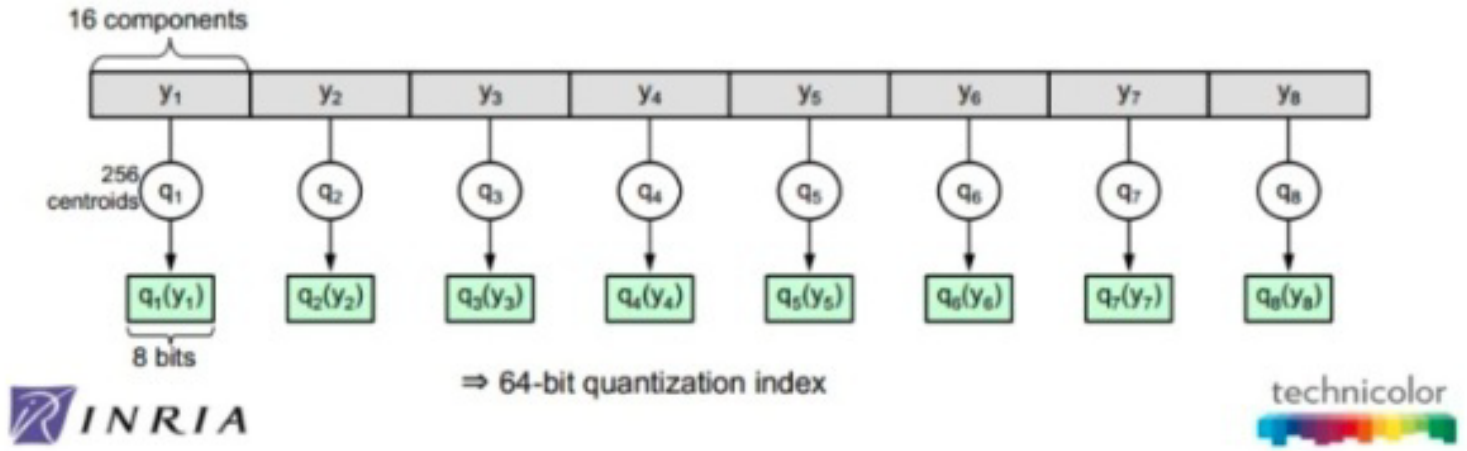
```
耗时：  0.004063606262207031
最近距离：  [[5.670616  5.7184644 5.8450646 6.0118012]
 [6.8587008 6.956443  7.100714  7.17673  ]
 [5.277209  5.79651   6.4668956 6.7035265]
 [5.809805  6.09425   6.465868  6.6459413]
 [5.14467   5.5545816 6.109751  6.222799 ]]
对应索引：  [[3271 2568 2596 3695]
 [8063 2700 6738 7603]
 [3919 4130 8483 6111]
 [4429  230 6789 5529]
 [9103  199 5916 3562]]
```

乘积量化

PQ算法是图像检索中常用的一种快速搜索算法，可以大大减少计算量。

具体做法如下：

（1）将D维空间切分为M个D/M维的子空间，假设现得到每张图片特征向量维度为128维，共256张图片。为了加速距离运算，现将128维向量切分成8段，如下图所示。



1. 分别在每一段所表示的子空间中进行K-means聚类
2. 之后在每一段中用距离该段内距离最近的聚类中心的编号来作为索引值，即每一个128维向量被压缩为8个索引值表示的向量。保留压缩后的向量与聚类中心。

乘积量化

```
nlist = 50
k = 4
m = 8  # 每个向量被编码成8个字节
quantizer = faiss.IndexFlatL2(d)
index = faiss.IndexIVFPQ(quantizer, d, nlist, m, 8)
index.train(xb)
index.add(xb)
index.nprobe = 5
start_time = time.time()
D, I = index.search(xq, k)
consume_time = time.time() - start_time

print('耗时：', consume_time)
print('最近距离：', D)
print('对应索引：', I)
```

```
耗时：  0.015917062759399414
最近距离：  [[4.9119143 5.520312  5.5522532 5.632739 ]
 [6.2797313 6.3857985 6.8672643 6.921715 ]
 [3.9875689 5.1577916 5.603181  5.680262 ]
 [5.269494  5.726007  5.7755857 5.910144 ]
 [4.663948  5.19293   5.274192  5.797719 ]]
对应索引：  [[3271 5378 6219  826]
 [8063 4424  670 3471]
 [3919 4130 1769 2363]
 [5466  230 6705 8066]
 [9103 9450  199 8670]]
```