lab4-LSH 检索　实验报告

电院 2303 徐恺阳 523030910085

December 2024

1　实验概览

本实验内容如下：

1. Hashing 的基本思想；
2. LSH 预处理；
3. 检索算法流程。

用 nearest neighbor (NN) 或 k-nearest neighbor (KNN) 在数据库中检索和输入数据距离最近的 1 个或 k 个数据，一般情况下算法复杂度为 O(n)（例如暴力搜索），优化情况下可达到 O(log n)（例如二叉树搜索），其中 n 为数据库中的数据量。当数据库很大（即 N 很大时），搜索速度很慢。

因此，我们引入 **LSH（局部敏感哈希）**。Hashing 的基本思想是按照某种规则（Hash 函数）把数据库中的数据分类，对于输入数据，先按照该规则找到相对应的类别，然后在其中进行搜索。由于某类别中的数据量相比全体数据少得多，因此搜索速度大大加快。

2　实验环境

本实验基于 Python 3.10，用到的库有 OpenCV、Numpy 和 time。

3　练习一

3.1　题目描述

利用 LSH 算法在图片数据库中搜索与目标图片最相似的图片。

3.2　解题思路

Step1: 图像数据的表示

在本实验中，每幅图像用一个 12 维的颜色直方图 p 表示，构成方式如图 1所示，其中 $H_i, i = 1, 2, 3, 4$ 是 3 维颜色直方图。

图 1: 构成方式

代码如下：

```python
def extract_color_histogram(image_path):
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([image], [0, 1, 2], None, [4, 4, 4], [0, 256, 0, 256, 0, 256])
    hist = cv2.normalize(hist, hist).flatten()
    return hist
```

图 2: 计算颜色直方图部分代码

同时，我们还要进行特征向量的量化。上述得到的特征向量 $p = (p_1, \cdots, p_{12})$，每个分量满足 $0 \leq p_j \leq 1$，将其量化成 3 个区间分别用 0, 1, 2 表示：

$$p_j = \begin{cases} 0, & \text{if } 0 \leq p_j < 0.3 \\ 1, & \text{if } 0.3 \leq p_j < 0.6 \\ 2, & \text{if } 0.6 \leq p_j \end{cases}$$

最终得到的特征向量的每个元素满足 $p_j \in \{0, 1, 2\}$，且 0, 1, 2 的分布平均。

代码如下：

```python
def quantize_features(hist):
    quantized = []
    for value in hist:
        if value < 0.3:
            quantized.append(0)
        elif value < 0.6:
            quantized.append(1)
        else:
            quantized.append(2)
    return quantized
```

图 3: 特征向量的量化部分代码

Step2: 哈希函数的计算

d 维整数向量 $\mathbf{p}$ 可用 $d' = d \times C$ 维的 Hamming 码表示：

$$v(\mathbf{p}) = \text{Unary}_C(p_1) \cdots \text{Unary}_C(p_d)$$

其中，$\text{Unary}_C(p_1)$ 表示 C 个二进制数，前 p_1 个为 1，后 $C - p_1$ 个为 0。

选取集合 $\{1,2,\ldots,d'\}$ 的 L 个子集 $\{I_i\}_{i=1}^{L}$，定义 $v(\mathbf{p})$ 在集合

$$I_i=\{i_1,i_2,\cdots,i_m\}:1\le i_1<i_2<\cdots<i_m\le d'$$

上的投影为 $g_i(\mathbf{p})=p_{i1}p_{i2}\cdots p_{im}$，其中 p_{ij} 为 $v(\mathbf{p})$ 的第 i_j 个元素。

事实上，我们不必显式的将 d 维空间中的点 $\mathbf{p}$ 映射到 d' 维 Hamming 空间向量 $\mathbf{v}(\mathbf{p})$，简化步骤如下。

定义：$I|i$ 表示 I 中范围在 $(i-1)\cdot C+1$ 到 $i\cdot C$ 中的坐标，例如：

$$I=\{1,3,7,8\},\quad I|1=\{1\},\quad I|2=\{3\},\quad I|3=\emptyset,\quad I|4=\{7,8\},\quad I|5=I|6=\emptyset$$

$\mathbf{v}(\mathbf{p})$ 在 I 上的投影即是 $\mathbf{v}(\mathbf{p})$ 在 $I|i$ $(i=1,2,\ldots,d)$ 上的投影串联，$\mathbf{v}(\mathbf{p})$ 在 $I|i$ 上的投影是一串 1 紧跟一串 0 的形式，需要求出 1 的个数：

$$|\{I|i\}-C\cdot(i-1)\le x_i|$$

再按顺序串联起来即可。

注意：这里特征向量 $p=(p_1,\cdots,p_{12})$ 的每个分量都被量化为 0，1，2，所以 C 取 2 即可。

例如，$\{I|1\}$ 中小于等于 $x_1=0$ 的个数为 0，故投影为 0；$\{I|2\}-2$ 中小于等于 $x_2=1$ 的个数为 1，故投影为 1；$\{I|4\}-3\cdot 2$ 中小于等于 $x_4=1$ 的个数为 1，故投影为 10。串联得到投影为 $(0,1,1,0)$。

代码如图 4所示。其中，量化特征向量（quantized_vector）和投影集合（hash_planes），生成一个对应的哈希值（hash_code）。

```python
def hash_function(quantized_vector, C, hash_planes):
    hash_code = []
    for i, plane in enumerate(hash_planes, start=1):
        count_ones = sum(1 for idx in plane if quantized_vector[idx - 1] <= C * (i - 1))
        hash_code.append(1 if count_ones > 0 else 0)
    return tuple(hash_code)
```

图 4: 哈希函数的计算部分代码

Step3:LSH 检索

$g(p)$ 被称作 Hash 函数，对于容量为 N 的数据集 $P=\{p_i\}_{i=1}^{N}$，$g(p)$ 可能的输出有 n 个，n 远小于 N，这样就将原先的 N 个数据分成了 n 个类别，其中每个类别中的数据具有相同的 Hash 值，不同类别的数据具有不同的 Hash 值。

对于待检索的输入 p，先计算 $g(p)$，找到其对应的类别，然后在该类别的数据集中进行搜索，速度能够大大加快。

我们先将一组量化后的特征向量插入到哈希桶（buckets）中，基于每个特征向量的哈希值（由 hash_function 计算），在哈希桶（buckets）中查找相似的图片候选集合。代码如下：

```python
def insert_to_buckets(quantized_vectors, hash_planes):
    buckets = defaultdict(list)
    for idx, q_vector in enumerate(quantized_vectors):
        hash_code = hash_function(q_vector, hash_planes)
        buckets[hash_code].append(idx + 1)
    return buckets
```

图 5: LSH 检索部分代码

```python
def query_image(target_vector, hash_planes, buckets):
    target_hash_code = hash_function(target_vector, hash_planes)
    return buckets.get(target_hash_code, [])
```

图 6: LSH 检索部分代码

Step4: 根据欧几里得距离进行匹配

原理与 nearest neighbor (NN) 相同，在数据库中检索和输入数据距离最近的 1 个数据。代码如下：

```python
# 根据欧几里得距离验证并精确匹配
if similar_image_ids:
    # 计算目标图像和当前图像之间的距离
    distances = {}
    for img_id in similar_image_ids:
        candidate_hist = extract_color_histogram(os.path.join(data_folder, f"{img_id}.jpg"))
        distances[img_id] = np.linalg.norm(candidate_hist - target_histogram)

    # 找到距离最近的匹配
    closest_match = min(distances, key=distances.get)
    print(f"最相似的图片是第{closest_match}张")
    print(f"与目标图像间的距离为: {distances[closest_match]}")
```

图 7: 根据欧几里得距离进行匹配部分代码

3.3 代码运行结果

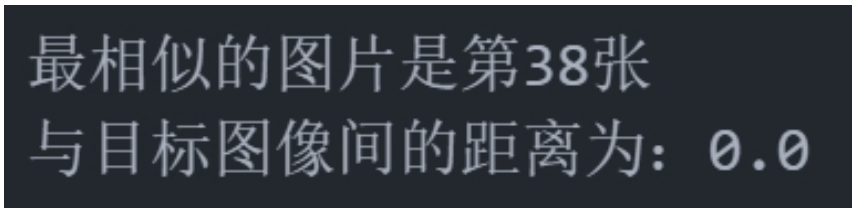

图 8: 匹配结果

3.4 分析与思考

匹配结果如图 8所示，结果非常准确！结果表明，检索出的图像在整体颜色分布上与输入图像有较高的相似性，例如主要颜色的比例和分布模式较为一致。这表明颜色直方图在表征图像全局色彩特征方面具有一定的可靠性。

4 练习二

4.1 题目描述

自行设计投影集合，尝试不同投影集合的搜索的效果。

4.2 解题思路

分别考虑以下三组投影集合的搜索的效果，利用 time 库中的相关函数，计算每组投影集合对应的检索时间和准确率。

1. 投影集合 1：$[[1,3,5],[2,4,6],[7,8,9],[10,11,12]]$

2. 投影集合 2：$[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]$

3. 投影集合 3：$[[1,6,7],[2,8,9],[3,4,5],[10,11,12]]$

4.3 代码运行结果

```
投影集合：投影集合1
检索时间：0.0000秒
检索准确率：100.00%
检索结果：第38张为最相似的图片！
--------------------------------------------------
投影集合：投影集合2
检索时间：0.0000秒
检索准确率：100.00%
检索结果：第38张为最相似的图片！
--------------------------------------------------
投影集合：投影集合3
检索时间：0.0010秒
检索准确率：100.00%
检索结果：第38张为最相似的图片！
--------------------------------------------------
```

图 9: 不同投影集合的搜索效果

4.4 分析与思考

可以看出，三组投影集合的准确率都非常高。

然而，图片数据库中的图片量太少，导致检索时间都非常短，无法很好地比较这三组投影集合的搜索效果，仅能看出第三组投影集合的搜索时间略长于前面两组。

5 练习三

5.1 题目描述

对比 NN 与 LSH 搜索的执行时间、搜索结果。

5.2 解题思路

NN 检索通过计算目标向量与数据库中每个向量之间的欧几里得距离，找到距离最小的向量，进而认为该向量最接近目标向量。代码如下：

```python
# 最近邻算法（暴力搜索）
def query_nn(target_vector, database_vectors):
    """使用最近邻算法暴力搜索"""
    distances = [np.linalg.norm(target_vector - db_vector) for db_vector in database_vectors]
    closest_idx = np.argmin(distances)
    return closest_idx, distances[closest_idx]
```

图 10: NN 检索代码

5.3 代码运行结果

```
LSH 搜索:
检索时间: 0.000000 秒
第38为最相似的图片!
--------------------------------------------------
NN 搜索:
检索时间: 0.001048 秒
第38为最相似的图片!
--------------------------------------------------
```

图 11: LSH 检索和 NN 检索的搜索效果比较结果

5.4 分析与思考

可以看出，LSH 检索的速度显著快于 NN 检索。但由于图片数据库中的图片量太少，二者准确率均非常高，很难看出差别。

6 实验感想

本次实验，我通过实现局部敏感哈希（LSH）算法，直观体验了其在高效图像检索中的优势。实验以颜色直方图为特征，通过比较传统最近邻（NN）算法和 LSH 算法的性能，进一步理解了 LSH 在大数据集中的检索效率提升。本实验让我深刻认识到合理的特征设计与哈希函数优化对结果准确性的重要性，同时激发了对探索其他特征设计的兴趣。

7 拓展思考

7.1 本练习中使用了颜色直方图特征信息，检索效果符合你的预期吗？检索出的图像与输入图像的相似性体现在哪里？

答：符合。检索出的图像在整体颜色分布上与输入图像有较高的相似性，例如主要颜色的比例和分布模式较为一致。这表明颜色直方图在表征图像全局色彩特征方面具有一定的可靠性。

7.2 能否设计其他的特征？

答：可以，例如纹理特征、形状特征、空间分布特征、深度学习特征、局部关键点特征以及多模态融合。例如 lab3 中的 SIFT 特征点匹配，就跟图像的灰度图的梯度特征有关。

7.3 PPT 的第 6 页中，x_i 指的是什么？

答：x_i 与前面所说的特征向量 p 的第 i 个分量 p_i 相对应。