深蓝学院
shenlanxueyuan.com

第二章作业讲评

主讲人 陆一帆

作业1


```
# --- get the split position ---
point_indices_sorted, _ = sort_key_by_vale(point_indices, db[point_indices, axis])

# 作业1
# 屏蔽开始
# 取排序后中间点
middle_left_idx = math.ceil(point_indices_sorted.shape[0] / 2) - 1
# 取排序后中间点的原始索引
middle_left_point_idx = point_indices_sorted[middle_left_idx]
# 取排序后中间点的值
middle_left_point_value = db[middle_left_point_idx, axis]

#取右边一个点的上述信息
middle_right_idx = middle_left_idx + 1
middle_right_point_idx = point_indices_sorted[middle_right_idx]
middle_right_point_value = db[middle_right_point_idx, axis]
```

看注释读代码

作业1

深蓝学院 shenlanxueyuan.com

```
#根节点的值赋值为上述两个点的平均值
root.value = (middle_left_point_value + middle_right_point_value) * 0.5
# === get the split position ===
# 二分，构建左子树
root.left = kdtree_recursive_build(root.left,
                                   db,
                                   point_indices_sorted[0:middle_right_idx],
                                   axis_round_robin(axis, dim=db.shape[1]),
                                   leaf_size)
# 二分，构建右子树
root.right = kdtree_recursive_build(root.right,
                                    db,
                                    point_indices_sorted[middle_right_idx:],
                                    axis_round_robin(axis, dim=db.shape[1]),
                                    leaf_size)
```

看注释读代码

显然，作业里提供的划分axis的方法不是最优的，应该是哪个轴上的数据分布方差大，就垂直于哪个轴分割，这样才能用最少的层数划分最多的点。

作业2

深蓝学院 shenlanxueyuan.com

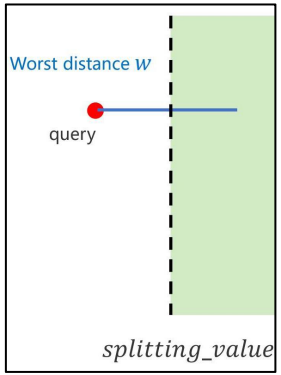
```
if root is None:
    return False

if root.is_leaf():
    # compare the contents of a leaf
    leaf_points = db[root.point_indices, :]
    diff = np.linalg.norm(np.expand_dims(query, 0) - leaf_points, axis=1)
    for i in range(diff.shape[0]):
        result_set.add_point(diff[i], root.point_indices[i])
    return False
```

```
# 作业2
# 提示：仍通过递归的方式实现搜索
# 屏蔽开始
# 如果当前axis的值小于根节点的值，则搜索左子树
if query[root.axis] <= root.value:
    kdtree_knn_search(root.left, db, result_set, query)
    if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
        kdtree_knn_search(root.right, db, result_set, query)
else:
    # 如果当前axis的值大于根节点的值，则搜索右子树
    kdtree_knn_search(root.right, db, result_set, query)
    if math.fabs(query[root.axis] - root.value) < result_set.worstDist():
        kdtree_knn_search(root.left, db, result_set, query)
# 屏蔽结束
```

1. 首先要记住，遇到叶子节点，直接把节点内所有点压入result set。result set会根据压入的点自动收缩worst_dist，然后把不符合要求的数剔除掉，剩下的数就是我们要求的解。

2. 如果不是叶子结点，那直接去搜它的子节点。目标点在哪一边先搜哪个（搜到的概率大）。

3. 搜完一边之后，worst dist就自动缩小了，这个时候比较一下worst dist和目标点到分割轴的距离，看看是不是可以不用搜另一边（如右图）。



作业3

深蓝学院
shenlanxueyuan.com

这里不放代码了，跟作业二几乎一样。

Radius和KNN搜索的唯一区别，本质区别，就是：

- KNN搜索的时候，worst dist在不断的缩小。随着你往result set中塞的点越来越多， worst dist越来越小，直到最近的k个点找到， worst dist 达到最小。
- Radius搜索的时候， worst dist就等于radius的值，永远不变。

作业4

深蓝学院 shenlanxueyuan.com

```
# 作业4
# 屏蔽开始

#root有子节点，is_leaf置为False
root.is_leaf = False
#创建8个子节点
children_point_indices = [[] for i in range(8)]
#遍历每一个点
for point_idx in point_indices:
    point_db = db[point_idx]
    #计算当前点该放置到哪个子节点
    morton_code = 0
    #判断该放到x轴的哪一侧
    if point_db[0] > center[0]:
        morton_code = morton_code | 1
    #判断该放到y轴的哪一侧
    if point_db[1] > center[1]:
        morton_code = morton_code | 2
    #判断该放到z轴的哪一侧
    if point_db[2] > center[2]:
        morton_code = morton_code | 4
    #子节点存储点的索引
    children_point_indices[morton_code].append(point_idx)
```

莫顿码（三位二进制数e.g. 001，010），对点的归属进行编码。

编码的第1位表示axis 1的归属
编码的第2位表示axis 2的归属
……

最终确定了把点放到哪个象限。

作业4

深蓝学院 shenlanxueyuan.com

```
#计算每一个子节点的center坐标
child_center_x = center[0] + factor[(i & 1) > 0] * extent
child_center_y = center[1] + factor[(i & 2) > 0] * extent
child_center_z = center[2] + factor[(i & 4) > 0] * extent
#子节点的extent
child_extent = 0.5 * extent
child_center = np.asarray([child_center_x, child_center_y, child_center_z])

#递归创建子节点的八叉树
root.children[i] = octree_recursive_build(root.children[i],
                                          db,
                                          child_center,
                                          child_extent,
                                          children_point_indices[i],
                                          leaf_size,
                                          min_extent)
```

1. 遍历每 8个子树（对应8个象限），构造子树的 center和extent。

2. 把上面分配好的children_point_indices传下去去递归建子树。

作业5

深蓝学院 shenlanxueyuan.com

```
if contains(query, result_set.worstDist(), root):
    # compare the contents of the octant
    leaf_points = db[root.point_indices, :]
    diff = np.linalg.norm(np.expand_dims(query, 0) - leaf_points, axis=1)
    for i in range(diff.shape[0]):
        result_set.add_point(diff[i], root.point_indices[i])
    # don't need to check any child
    return False
```

```
if root.is_leaf and len(root.point_indices) > 0:
    # compare the contents of a leaf
    leaf_points = db[root.point_indices, :]
    diff = np.linalg.norm(np.expand_dims(query, 0) - leaf_points, axis=1)
    for i in range(diff.shape[0]):
        result_set.add_point(diff[i], root.point_indices[i])
    # check whether we can stop search now
    return inside(query, result_set.worstDist(), root)
```

如果当前节点整个被包含在worst dist里，那就要把每个点都压入result set，关键是，直接返回false，因为不可能在这个节点就停止搜索，因为radius搜索不会缩小worstdist。

如果该结点没有被完全包住，同时它是个叶子节点，那就把该节点的点都压入result set，也别去找子节点了，因为叶子节点没儿子了。最后看一下是不是worstdist被包含在节点里，可以提前终止了。

1. 递归搜索儿子节点的代码不提供了。注意这里用不着先去算距离目标点最近的儿子节点。因为radius搜索，worstdist不变，所以所有的儿子节点都需要查，不存在说先查了哪个，后面就可以不查了。
2. 最后的最后，遍历完儿子节点后，别忘了查一下是不是可以在该节点提前终止搜索了。

作业6&7

作业6是作业5的简化版本，就是没有contain判断了，其他一样，不重要，主要看作业5的耗时优化。

```
# 作业7
# 屏蔽开始
# go to the relevant child first
# 根据query找到当前所属象限
morton_code = 0
if query[0] > root.center[0]:
    morton_code = morton_code | 1
if query[1] > root.center[1]:
    morton_code = morton_code | 2
if query[2] > root.center[2]:
    morton_code = morton_code | 4

#去八叉树的相应象限中递归查找
if octree_knn_search(root.children[morton_code], db, result_set, query):
    return True
```

作业7和作业6很像。

关键在于，knn搜索的时候，**要先去搜距离目标点最近的儿子节点（如上图）**，这和radius搜索不一样。因为knn搜索，worstdist会慢慢变小，有可能你搜完第一个儿子节点后，发现worstdist很小了，被第一个儿子节点包住了，就不用搜剩下的7个了。最容易让worstdist变小的节点当然是距离目标点最近的儿子节点啦！

benchmark

最后，大家拿我提供的的benchmark.py文件，跑一下自己的octree和kdtree。

大概做到kdtree和octree的radius/knn都是暴力算法的耗时的1/9左右时就是比较好的作业了。

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening