深蓝学院
shenlanxueyuan.com

Nearest Neighbors Problem

主讲人 冯冠植

ResultSet

- 结果集的成员变量有：
- KNN的K，对应capacity
- count表示当前已经找到多少个neighbors
- 初始化最差距离极大
- 维护一个存放当前已经找到的答案的索引及其距离query点距离的列表

```python
class DistIndex:
    def __init__(self, distance, index):
        self.distance = distance
        self.index = index

    def __lt__(self, other):
        return self.distance < other.distance
```

```python
class KNNResultSet:
    def __init__(self, capacity):
        self.capacity = capacity
        self.count = 0
        self.worst_dist = 1e10
        self.dist_index_list = []
        for i in range(capacity):
            self.dist_index_list.append(DistIndex(self.worst_dist, 0))

        self.comparison_counter = 0
```

ResultSet

- 尝试添加新的点，如不符合条件（dist>worst_dist）直接返回。
- 如何添加：保存结果的列表按照里面每个点到query点的距离升序排列，于是可以找到待添加的点对应合适的位置并加入列表，其他元素后移。
- 最差距离始终选择最后一个元素，即最大距离

```python
def add_point(self, dist, index):
    self.comparison_counter += 1
    if dist > self.worst_dist:
        return

    if self.count < self.capacity:
        self.count += 1

    i = self.count - 1
    while i > 0:
        if self.dist_index_list[i-1].distance > dist:
            self.dist_index_list[i] = copy.deepcopy(self.dist_index_list[i-1])
            i -= 1
        else:
            break

    self.dist_index_list[i].distance = dist
    self.dist_index_list[i].index = index
    self.worst_dist = self.dist_index_list[self.capacity-1].distance
```

KDTree

- 节点类的成员分别有：下一次切割的轴，切割轴上的虚拟点，切割之后的左子节点和右子节点，以及当前节点里存放的点集

```
class Node:
    def __init__(self, axis, value, left, right, point_indices):
        self.axis = axis
        self.value = value
        self.left = left
        self.right = right
        self.point_indices = point_indices
```

KDTree

- sort_key_by_value根据值的升序排列返回索引和对应的值

- axis_round_robin功能类似axis++, 主要实现了在0到dim-1区间的循环递增

```
def sort_key_by_value(key, value):
    assert key.shape == value.shape
    assert len(key.shape) == 1
    sorted_idx = np.argsort(value)
    key_sorted = key[sorted_idx]
    value_sorted = value[sorted_idx]
    return key_sorted, value_sorted


def axis_round_robin(axis, dim):
    if axis == dim-1:
        return 0
    else:
        return axis + 1
```

KDTree_Construction

深蓝学院 shenlanxueyuan.com

- 每次递归创建一个新节点并把当前所有点放入
- 如果满足叶子节点要求则返回，如果不满足则需要进一步切割和点集划分
- 先确定切割轴的位置：找点集在切割方向上的中点，并把它赋值给当前节点的value
- 进入递归，左子节点为切割轴左侧点集的建树结果，右子节点为切割轴右侧点集的建树结果

```python
def kdtree_recursive_build(root, db, point_indices, axis, leaf_size):
    if root is None:
        root = Node(axis, None, None, None, point_indices)

    # determine whether to split into left and right
    if len(point_indices) > leaf_size:
        # --- get the split position ---
        point_indices_sorted, _ = sort_key_by_vale(point_indices, db[point_indices, axis])  # M

        # 作业1
        # 屏蔽开始
        middle_left_index = math.ceil(point_indices_sorted.shape[0] / 2) - 1
        middle_left_point_index = point_indices_sorted[middle_left_index]
        middle_left_point_value = db[middle_left_point_index]

        middle_right_index = middle_left_index + 1
        middle_right_point_index = point_indices_sorted[middle_right_index]
        middle_right_point_value = db[middle_right_point_index]

        root.value = (middle_right_point_value + middle_left_point_value) / 2
        root.left = kdtree_recursive_build(root.left,
                                           db,
                                           point_indices_sorted[0:middle_right_index],
                                           axis_round_robin(axis, dim = db.shape[1]),
                                           leaf_size)
        root.right = kdtree_recursive_build(root.right,
                                            db,
                                            point_indices_sorted[middle_right_index:],
                                            axis_round_robin(axis, dim = db.shape[1]),
                                            leaf_size)
        # 屏蔽结束
    return root
```

KDTree_KNNSearch

- 如果当前节点是叶子节点就尝试将其包含的所有点加入结果集
- 如果不是叶子节点，根据query位置选择往哪个分枝搜索，如果搜索完成后query距离仍小于最差距离，说明另一个分枝还有机会，需要搜索

```python
def kdtree_knn_search(root: Node, db: np.ndarray, result_set: KNNResultSet, query: np.ndarray):
    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
    if query[root.axis] >= root.value[root.axis]:
        kdtree_knn_search(root.right, db, result_set, query)
        if math.fabs(query[root.axis] - root.value[root.axis]) < result_set.worstDist():
            kdtree_knn_search(root.left, db, result_set, query)
    else:
        kdtree_knn_search(root.left, db, result_set, query)
        if math.fabs(query[root.axis] - root.value[root.axis]) < result_set.worstDist():
            kdtree_knn_search(root.right, db, result_set, query)
    # 屏蔽结束

    return False
```

OcTree

深蓝学院
shenlanxueyuan.com

- Octant类的成员分别有：八个子节点，当前节点中心，当前节点（立方体）的1/2边长，以及当前立方体包含的所有点的索引

```python
class Octant:
    def __init__(self, children, center, extent, point_indices, is_leaf):
        self.children = children
        self.center = center
        self.extent = extent
        self.point_indices = point_indices
        self.is_leaf = is_leaf
```

OcTree_Construction

- 每次递归创建新节点，存放当前点集。
- 依据点集里每个点到当前中点的相对位置，将点集划分到八个子节点当中并进入下一次递归

```python
def octree_recursive_build(root, db, center, extent, point_indices, leaf_size, min_extent):
    if len(point_indices) == 0:
        return None

    if root is None:
        root = Octant([None for i in range(8)], center, extent, point_indices, is_leaf=True)
```

```python
    if len(point_indices) <= leaf_size or extent <= min_extent:
        root.is_leaf = True
    else:
        # 作业4
        # 屏蔽开始
        root.is_leaf = False
        children_point_indices = [[] for i in range(8)]
        for point_idx in point_indices:
            point_db = db[point_idx]
            morton_code = 0
            if point_db[0] > center[0]:
                morton_code = morton_code | 1
            if point_db[1] > center[1]:
                morton_code = morton_code | 2
            if point_db[2] > center[2]:
                morton_code = morton_code | 4
            children_point_indices[morton_code].append(point_idx)
        factor = [-0.5, 0.5]
        for i in range(8):
            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
            child_extent = 0.5 * extent
            child_center = np.asarray([child_center_x, child_center_y, child_center_z])
            root.children[i] = octree_recursive_build(root.children[i],
                                                      db,
                                                      child_center,
                                                      child_extent,
                                                      children_point_indices[i],
                                                      leaf_size,
                                                      min_extent)
        # 屏蔽结束
    return root
```

OcTree_KNN

- 和KDTree的搜索相同，如果是叶子节点则遍历其中所有点如果不是叶子节点，则递归搜索八个子节点，其中如果query球和子节点立方体没有交集则可以跳过此子节点
- RadiusNN和KNN基本相同

```python
def octree_knn_search(root: Octant, db: np.ndarray, result_set: KNNResultSet, query: np.ndarray):
    if root is None:
        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)

    # 作业7
    # 屏蔽开始
    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

    if octree_knn_search(root.children[morton_code], db, result_set, query):
        return True
    # 屏蔽结束
    for c, child in enumerate(root.children):
        if c == morton_code or child is None:
            continue
        if False == overlaps(query, result_set.worstDist(), child):
            continue
        if octree_knn_search(child, db, result_set, query):
            return True

    # final check of if we can stop search
    return inside(query, result_set.worstDist(), root)
```

Numpy暴力搜索

- 计算query点距离点云里每个点的距离并排序，取前K个即可

```
diff = np.linalg.norm(np.expand_dims(query, 0) - db_np, axis=1)
nn_idx = np.argsort(diff)
nn_dist = diff[nn_idx]
```

python调库

深蓝学院
shenlanxueyuan.com

- scipy

```
from scipy import spatial
```

```
begin_t = time.time()
tree = spatial.KDTree(db_np)
construction_time_sum += time.time() - begin_t

query = db_np[0,:]

begin_t = time.time()
result = tree.query(query, k = 8)
knn_time_sum += time.time() - begin_t
```

- open3d

```
import open3d as o3d
```

```
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(db_np)

begin_t = time.time()
o3d_tree = o3d.geometry.KDTreeFlann(pcd)
construction_time_sum += time.time() - begin_t

query = db_np[0,:]

begin_t = time.time()
[x1, idx, _] = o3d_tree.search_knn_vector_3d(query, k)
knn_time_sum += time.time() - begin_t

begin_t = time.time()
[x2, idx, _] = o3d_tree.search_radius_vector_3d(query, radius)
radius_time_sum += time.time() - begin_t
```

C++调库

```cpp
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/kdtree/kdtree_flann.h>
```

- 使用pcl库。将Bin文件转换为pcd文件

```cpp
void convertBin2Pcd(const std::string &path, pcl::PointCloud<pcl::PointXYZI>::Ptr point_cloud, const std::string &path2){
    std::fstream inputFile_(path, std::ios::in | std::ios::binary);
    while(inputFile_){
        pcl::PointXYZI point;
        inputFile_.read((char *) &point.x, 3*sizeof(float));
        //inputFile_.read((char *) &point.y, sizeof(float));
        //inputFile_.read((char *) &point.z, sizeof(float));
        inputFile_.read((char *) &point.intensity, sizeof(float));
        point_cloud->push_back(point);
    }
    inputFile_.close();
    pcl::io::savePCDFileBinary(path2, *point_cloud);
    return;
}
```

C++调库

```cpp
void pclKnnSearch(pcl::PointCloud<pcl::PointXYZI>::Ptr point_cloud, int K, std::vector<int> &neighborIndices_, std::vector<float> &neighborDistances_){

    pcl::KdTreeFLANN<pcl::PointXYZI> kdtree;
    kdtree.setInputCloud(point_cloud);
    pcl::PointXYZI searchPoint = point_cloud->points[0];
    std::cout<<"searching "<<K<<" nearest neighbors of point "<<searchPoint<<'\n';
    std::unique_ptr<Timer> timer = std::make_unique<Timer>();
    kdtree.nearestKSearch(searchPoint, K, neighborIndices_, neighborDistances_);
    return;
}
```

C++调库

```cpp
int main(int argc, char **argv){
    //std::string path = argv[1];
    //std::string outputFile_ = "/home/gfeng/gfeng_ws/point_cloud_processing/ch2_nearest_neighbor_problem/data/0.pcd";
    pcl::PointCloud<pcl::PointXYZI>::Ptr point_cloud(new pcl::PointCloud<pcl::PointXYZI>);
    //convertBin2Pcd(path, point_cloud, outputFile_);
    if(pcl::io::loadPCDFile<pcl::PointXYZI>("/home/gfeng/gfeng_ws/point_cloud_processing/ch2_nearest_neighbor_problem/data/0.pcd", *point_cloud) == -1){
        PCL_ERROR ("Couldn't read file\n");
        return (-1);
    }
    //visPointCloud(point_cloud);
    int K = 8;
    std::vector<int> neighborIndices_(K);
    std::vector<float> neighborDistances_(K);
    pclKnnSearch(point_cloud, K, neighborIndices_, neighborDistances_);

    for(int i = 0;i < K; i++){
        int index = neighborIndices_[i];
        std::cout<<index<<' '<<point_cloud->points[index]<<' '<<neighborDistances_[i]<<'\n';
    }
    return 0;
}
```

计算时间比较

- 12万个点，搜索索引为0的点的8个最邻近点

```
searching 8 nearest neighbors of point (52.8979,0.0229897,1.99799 - 0.08)
starting to record time
Timer took 0.025061ms
0 (52.8979,0.0229897,1.99799 - 0.08) 0
3943 (52.8103,-0.113949,1.66003 - 0) 0.140658
5884 (52.7904,-0.168925,1.41104 - 0) 0.392907
3944 (53.6181,-0.0319858,1.68201 - 0) 0.621469
5885 (53.484,-0.00199911,1.426 - 0) 0.671272
1 (53.7505,0.192914,2.02695 - 0) 0.756614
5883 (53.3032,-0.504776,1.42212 - 0) 0.774437
1966 (52.3758,-0.716679,1.98017 - 0) 0.820059
```

```
octree --------------
0 - 0.00
3943 - 0.38
5884 - 0.63
3944 - 0.79
5885 - 0.82
1 - 0.87
5883 - 0.88
1966 - 0.91
In total 22 comparison operations.
Octree: build 6106.511, knn 0.849, radius 0.941, brute 9.175
kdtree --------------
Kdtree: build 116.174, knn 2.608, radius 0.290, brute 8.777
scipy --------------
Scipy: build 563.690, knn 0.541, brute 8.403
open3d --------------
Open3d: build 11.224, knn 0.031, radius 0.010, brute 8.903
```

在线问答

Q&A

感谢各位聆听！

Thanks for Listening