VP 树

Mr Wu

2019 年 5 月 8 日

1 介绍

VP 树是更具体的度量树，可以有效划分 n 维度量空间中的数据。VP 树在执行范围查询时有优势。

考虑二维平面上一个包含 1000 个点的数据集。VP 树的每个结点包含以下五方面信息：

数据：它包含的数据点列表；

vp: 从数据中选取的一个优势点；

mu：定义这个结点范围的一个半径值；

inside: 左子树；

outside: 右子树。

2 创建

现在开始创建根结点。根结点的数据包含所有数据点。选择一个好的 vp 可以直接影响树的效率，但为了方便起见，我们随机选取一个数据点。

选择好 vp 之后，开始计算 mu，使得一半数据在半径范围之内。

接下来再细分数据点。创建左子树，包含所有圆内的数据点；创建右子树，包含所有圆外的点。如此递归，直到子树中不包含数据点。对于 1000 个点的数据集，只需要划分 9 次。

3 分析

每一层递归将原始问题划分为两个相同规模的子问题。在每个结点，必须计算出结点所包含数据的中位数，用来得到 mu 值。对于每一层，这个操作的复杂度为 $O(n)$。

树的深度为 $O(logn)$，因此创建一个 VP 树的总复杂度为 $O(nlogn)$。

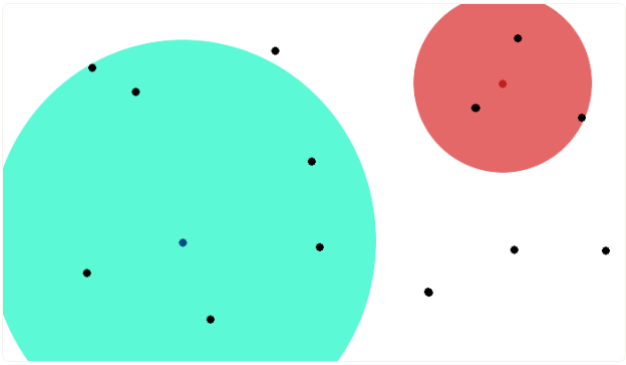
4 搜索

搜索目标是通过查询尽可能少的结点来找到查询点的最近邻。

考虑一个半径为 tau 的以查询点为圆心的圆，包含它的所有最近邻。假设要找 k 近邻，则 tau 会包含最近的 k 个数据点。

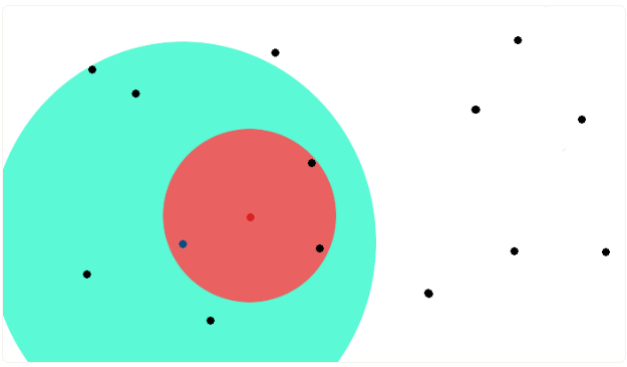
考虑 $k = 3$ 的情况：14 个数据点 (黑点)，查询点 (红点)，vp(蓝点)，mu(蓝圈)，tau(红圈)。

情形 1：



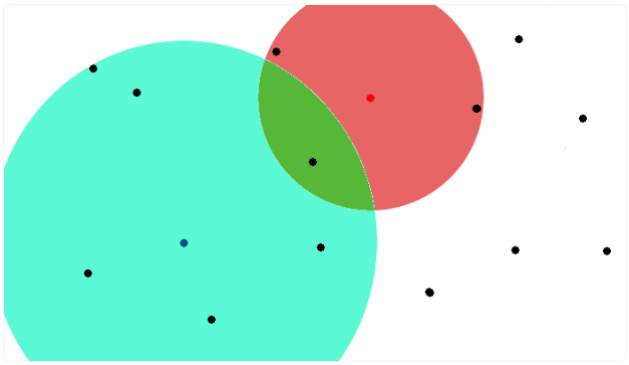
注意到 tau 完全在 mu 之外，也就是查询点的最近邻全在右子树中。所以可以剪去左子树，工作量减半。

情形 2：



与情形 1 正好相反，可以剪去右子树。

情形 3：



此时 tau 和 mu 相交，无法剪枝，必须搜索左右子树。

搜索树的时候，可以把 tau 看成查询点周围感兴趣的区域。从无穷大开始，通过剪去感兴趣区域之外的点，来逐步缩小 tau。代码：

```
// 给定根结点，k近邻，查询点。
function knn (root, k, query) {
    // 从根结点开始查询。
    var tau = Infinity;
    var toSearch = [root];

    // 将结果存储在长度为k的优先队列中。
    // 按到查询点的距离来排序。
    var results = new DistanceQueue(query, k);

    while (toSearch.length > 0) {
        // 出队一个结点，然后搜索。
        var currentNode = toSearch.splice(0, 1)[0];//
        var dist = query.dist(currentNode.vp);//计算当前结点的vp到查询点的距离

        // 若当前结点的vp在查询点的范围tau内，则加入该点，减小tau;
        if (dist < tau) {
            toSearch.push(currentNode.vp);
            var farthest = results.last();
            tau = query.dist(farthest);
        }

        // tau的一部分区域在mu中，所以要检查左子树。
        if (dist < currentNode.mu + tau)
            toSearch.push(currentNode.left);

        // tau的一部分区域在mu外，所以要检查右子树。
        if (dist >= currentNode.mu - tau)
            toSearch.push(currentNode.right);
    }

    return results;
}
```