K 近邻

W.J.Z

摘要： 本文重点介绍了 K 近邻距离度量、kd 树生成、kd 树最近邻搜索。算法和标准算法有所差别但效果都是一样的，理解难点在于切分时递归和搜索时半径与维度距离大小比较，自己动手模拟过程可以帮助理解。

1 近邻模型

K 近邻算法：给定一个训练数据集, 对新的输入实例，在训练数据集中找到与该算法最邻近的 K 个实例。模型由距离度量、K 值、分类决策规定决定。

欧式距离：

$$L_p\left(x_i, x_j\right) = \left(\sum_{l=1}^{n}\left|x_i^l - x_j^l\right|^2\right)^{\frac{1}{2}} \tag{1}$$

曼哈顿距离：

$$L_p\left(x_i, x_j\right) = \sum_{l=1}^{n}\left|x_i^l - x_j^l\right| \tag{2}$$

切比雪夫距离：

$$L_p\left(x_i, x_j\right) = \max_{l}\left|x_i^l - x_j^l\right| \tag{3}$$

2 kd 树构造

k 近邻法最简单的实现方式为线性扫描，当训练集很大时，计算非常耗时，为提高搜索效率，使用特殊存储结构储存训练数据，kd 树就是其中一种方法。

Algorithm 1: 构造平衡二叉树

Input: 训练数据集

Output: kd 树

1. 构造根节点，选择实例 $(depth\%data.length)$ 维度的中位数为切分点，将对应的区域分成两个子区域。左边区域节点对应坐标小于且分点，右边区域节点大于切分点。
2. 记录该深度落在切分超平面实例的下标，重复步骤一操作，直到两个子区域没有实例存在为止。

kd 关键算法代码：

```
#kd树构造利用二叉树递归
#二叉树节点
struct Node{
    int index; #记录落在超平面实例的下标
```

```
    int axis; #记录该节点所在的维度
    Node *left; #左区域
    Node *right; #右区域
    Node():index(-1),axis(-1){left=right=nullptr}
};

/**建树
*indices记录每一个落在超平面实例的下标， 二叉树节点储存的是下标
*不是实例,indices初始值为{0,1,2,3...n},n为数据集的大小,len为
*剩余搜索长度， depth为二叉树深度
*/
Node* buildTree(int* indices,int len,int depth)
{
    if len <=0
        return nullptr;
    //依次递增维度,length为特征向量长度
    const int axis = depth % length;
    //获取中位数
    const int mid = len / 2;
    #依照数据集选定的维度数据进行indices排序获取中位数
    for(i=0;i<len,i++)
        temp[i] = *((indices+i)[axis]);
    qQsort(temp,indeices,0,len(data));
    /**qQsort函数等同于使用STL标准库的函数如下：
    std::nth_element(indices, indices + mid, indices + len
    , [&](int lhs, int rhs){
    return data[lhs][axis] < data[rhs][axis];
    });
    */
    Node * node = new Node();
    node->index = indices[mid];
    node->axis = axis;
    node->left = buildTree(indices,mid,depth+1);
    node->right = buildTree(indices+mid+1,len-mid-1,depth+1);
    #indices+mid+1移动指针
    return node;
}
void qQsort(int temp[],int* p,int low,int high)
{
    if(low >= high)
        return ;
```

```
    int first = low;
    int last = high;
    int key = temp[first];
    while(first<last)
    {
        while(first < last && temp[last] >= key)
            --last;
        temp[first]=temp[last];
        while(first < last && temp[first] <= key)
            ++first;
        temp[last] = temp[first];
    }
    a[first] = key;
    //对indices进行调整排序
    a =  p[first];
    p[first] = p[low];
    p[low] = a;
    qQsort(temp,low,first -1);
    qQsort(temp,first+1,high);
}
```

3 kd 树最近邻搜索

Algorithm 2: kd 树最近邻搜索

Input: kd 树，目标点 x

Output: x 的最近邻

1 从根节点出发，递归向下访问 kd 树，若目标点当前维的坐标小于且分点的坐标，移动到左节点，否则移动到右节点，直到子节点为叶节点未知。同时计算目标点与个切分点的欧式距离，记录最小距离和最近节点。

2 递归向上回退，计算当前点的维度与目标点之间的距离，如果小于最小距离，说明当前节点另外一个子节点可能存在离目标节点更近的节点，进入该子区域，重复步骤一。

3 当退回根节点，搜索结束，最后的最近节点为目标节点的最近邻节点。

```
void search(const int& query,const Node* node,int* result,
double* minDist)
{
    if(node==nullptr)
        return;
```

```
    //获取当前节点数据
    const int& train = data[node->index];
    //计算当前节点与目标节点的欧式距离
    //distance()计算欧式距离函数
    const double dist = distance(query,train);
    //记录最近距离和坐标
    if(dist < *minDist)
    {
        *minDist = dist;
        *result = node->index;
    }
    const int axis = node->axis;
    //进行当前节点维度比较，进而选择左右树
    const int dir = query[axis] < train[axis]?0:1;
    if(dir == 0)
        search(query,node->left,result,minDist);
    else
        search(query,node->left,result,minDist);
    //计算当前节点维度与目标节点的距离
    const double diff = fabs(query[axis]-train[axis]);
    //如果在半径之内，说明当前节点子区域可能存在更近的点
    if(diff < *miDist)
    {
        if(!dir)
            search(query,node->right,result,minDist);
        else
            search(query,node->left,result,minDist);
    }
}
```

参考资料：https://www.cnblogs.com/earendil/p/8135074.html

大佬代码参考：https://github.com/benjones/kdTree

4 性能度量

聚类性能度量大致分为两类：一类将聚类结果与某个参考模型进行比较，成为外部指标；一类直接考察聚类结果而不利用任何参考模型，称为内部指标。

4.1 外部指标

假设通过聚类给出的簇划分为 $C = C_1, C_2, C_3, \dots, C_k$, 参考模型给出的簇划分为 $C^* = C_1^*, \dots, C_s^*$, 令 λ 和 λ^* 分别表示与 C 和 C^* 对应的簇标记向量。

$$a = |SS|\,; SS = \left\{ (x_i, y_j)\,|\lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j \right\} \tag{4}$$

$$b = |SD|\,; SS = \left\{ (x_i, y_j)\,|\lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j \right\} \tag{5}$$

$$c = |DS|\,; SS = \left\{ (x_i, y_j)\,|\lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j \right\} \tag{6}$$

$$d = |DD|\,; SS = \left\{ (x_i, y_j)\,|\lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j \right\} \tag{7}$$

1. Jaccard 系数

$$JC = \frac{a}{a + b + c} \tag{8}$$

2. FM 指数

$$FMI = \sqrt{\frac{a}{a + b} \cdot \frac{a}{a + c}} \tag{9}$$

3. Rand 指数

$$RI = \frac{2\,(a + d)}{m\,(m - 1)} \tag{10}$$

上述性能度量的结果均在 $[0 - 1]$ 区间，值越大越好。

4.2 内部指标

$$avg(C) = \frac{2}{|C|\,(|C| - 1)} \sum_{1 \leq i \leq j \leq |C|} dist(x_i, x_j) \tag{11}$$

$$diam(C) = max_{1 \leq i \leq j \leq |C|} dist(x_i, x_j) \tag{12}$$

$$d_{min}\,(C_i, C_j) = min_{x_i \epsilon C_i, x_j \epsilon C_j} dist(x_i, y_j) \tag{13}$$

$$d_{cen}(C_i, C_j) = dist(\mu_i, \mu_j) \tag{14}$$

$dist()$ 表示两个样本之间的距离，μ 代表簇 C 的中心点，$avg(C)$ 对应簇 C 内样本间的平均距离，diam(C) 对应于簇 C 内样本间的最远距离，$d_{min}(C_i, C_j$ 对应簇 C_i 与簇 C_j 最近样本间的距离，$d_{cen}(C_i, C_j)$ 对应于簇 C_i 与簇 C_j 中心点的距离。

1. DB 指数

$$DBI = \frac{1}{k} \sum_{k}^{i=1} max_{j \neq i} \left(\frac{avg(C_i) + avg(C_j)}{d_{cen}(\mu_i, \mu_j)} \right) \tag{15}$$

2. Dunn 指数

$$DI = min_{1 \leq i \leq k} \left\{ min_{j \neq i} \left(\frac{d_{min}\,(C_i, C_j)}{max_{1 \leq lk} diam(C_l)} \right) \right\} \tag{16}$$

DBI 值越小越好，DI 值越大越好。