k 邻近法

1 k 邻近法

1.1 距离度量

对于 $x_i, x_j \in R^n, x_i = (x_i^1, x_i^2, ..., x_i^n), x_i = (x_j^1, x_j^2, ..., x_j^n)$，两者之间的 L_p 距离定义为：

$$L_P(x_i, x_j) = (\sum_{l=1}^{n} |x_i^l - x_j^l|^p)^{\frac{1}{p}}$$

几种特殊的 L_p 距离：

- $p = 2$：欧氏距离

$$L_2(x_i, x_j) = (\sum_{l=1}^{n} |x_i^l - x_j^l|^2)^{\frac{1}{2}}$$

- $p = 1$：曼哈顿距离

$$L_1(x_i, x_j) = \sum_{l=1}^{n} |x_i^l - x_j^l|$$

- $p = +\infty$：各个坐标距离最大值

$$L_\infty(x_i, x_j) = \max |x_i^l - x_j^l|$$

1.2 k 邻近算法

k 邻近算法步骤如下：

输入：训练集 $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$。其中，$x_i \in R^n, y \in \{c_1, c_2, ...c_K\}, i = 1, 2, ..., N$ 和实例特征 x。

1. 根据给定的距离度量方式 $l(x_i, x_j)$，在训练集 T 中找出与 x 最邻近的 k 个点，记这 k 个点所构成的集合为 $N_k(x)$
2. 决定 x 的类别

$$y = argmax_{c_j} \sum_{x_i \in N_k(x)} I(y_i = c_j), i = 1, 2.., N; j = 1, 2, ..., K$$

输出：x 的类别 y。

1.3 k 的选取

通常采用交叉验证法来选取最优的 k 值。

1.4 kd 树

kd 树可以大幅度地将 k 邻近法的**平均效率**从 $O(n)$ 提升到 $O(\log n)$。

1.4.1 构造 kd 树

kd 树不断地使用垂直于坐标轴的超平面将 k 维空间切分，将空间划分为一系列 k 维的超矩形区域。构造 kd 树的步骤如下：

1. 构造根节点，根节点对应包含所有实例点的超矩形区域
2. 对 k 维空间进行划分，生成子节点。划分方法是：(1) 在超矩形区域上选择**一个坐标轴**和该坐标轴上的一个实例作为**切分点**；(2) 确定**垂直于该坐标轴**且**过切分点**的超平面；(3) 将超矩形切分为左右两个子区域
3. 对空间进行递归，直到所有子区域中没有实例

操作中，我们往往使用实例点在坐标轴上的**中位数**作为切分点，这样可以得到**平衡的** kd 树。

具体的，在数据集上构造 kd 树算法如下：

输入：k 维空间数据集 $T = \{x_1, x_2, ..., x_N\}$，其中 $x_i = (x_i^1, x_i^2, ..., x_i^k)^T, i = 1, 2, ..., N$。

1. 构造根节点，根节点对应包含 T 的 k 为空间的超矩形区域
2. 选择 x^1 作为坐标轴，选取 $\{x_1^1, x_2^1, ..., x_N^1\}$ 的中位数作为切分点 (若中位数不在集合中，则使用离中位数最近且在集合中的数作为切分点)，将根节点对应的超矩形区域划分为两个子区域，将过切分平面的实例点存储在节点
3. 以 x^l 作为切分轴，其中 $l = j \mod k + 1, j$ 为该节点的深度，重复步骤 2，直至子区域均没有实例

输出：kd 树。

1.4.2 搜索 kd 树

以 kd 树的最近邻搜索为例，其搜索最邻近的方法是：

1. 先找到**包含目标点的叶节点**
2. 从该叶节点出发，**依次回退到父节点**，不断查找与目标点最邻近的节点，直到确定不存在更近的点为止

更为具体的，依次回退继续寻找最邻近的点的步骤是 (假设目前将回退到节点 A)：

1. 比较 A 中的节点是否比当前最近点更近，若更近则将其变为当前最近点
2. 以目标点为中心，目标点和当前最近点连线为半径做一个球体
3. 判断此球体是否与 A 的兄弟节点的超矩形区域相交
4. 若不相交，则回到 1 继续回退
5. 若相交，则在相交区域内寻找是否存在更近的点
6. 若不存在，则回到 1 继续回退
7. 若存在，则在找出更近的节点 B，将其设为最近点并继续回退

具体地，在 kd 树上搜索的步骤是：

输入：已构造好的 kd 树**和**目标点 x。

1. 在 kd 树中找到包含目标点 x 的叶节点：从根节点出发，递归地向下寻找。若 x 的当前维坐标小于切分点的坐标，则移动到左节点；否则移动到右节点
2. 以该叶节点为*当前最近点*
3. 递归地向上回退，在每个节点均执行以下操作：(1) 若该节点保存的实例比*当前最近点*更近，则以*当前节点*为*当前最近点*；(2) 检查**该节点的父节点的另一子节点**所对应的区域中是否与**以目标点为球心，目标点当前最近点为半径的球体**相交；(3) 如果不相交，则继续向上回退；(4) 如果相交，则在**该节点的父节点的另一子节点**所对应的区域中**递归**使用 kd 树最邻近搜索，找出该区域中 x 的最邻近点；(4) 如果该区域中的最邻近点比当前最近点远，则继续回退；(5) 如果该区域中的最邻近点比当前最近点近，则将该区域中的最邻近点设为当前最近点，并继续回退
4. 当退回到根节点时，搜索结束，存储的*当前最近点*即为 x 的最近邻

输出：x 的最近邻

2 代码实现

考虑以下数据:

$$T = \{(2,3)^T, (5,4)^T, (9,6)^T, (4,7)^T, (8,1)^T, (7,2)^T\}$$

其对应的标签依次为 0,1,2,3,4,5。目标点为 $(3,4.5)^T$。

2.1 sklearn 实现

准备数据：

```
import numpy as np
X = np.array([[2,3],[5,4],[9,6],[4,7],[8,1],[7,2]])
Y = np.array([i for i in range(6)]).reshape(-1,1)
```

k 邻近法的 sklearn 接口位于 sklearn.neighbors.KNeighborsClassifier, 其文档可见**此处**。其中较为重要的参数有：

- n_neighbors:k 的取值
- p:L_p 距离中的 p

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=1,p=2)
clf.fit(X,Y)
```

```
clf.predict(np.array([[3,4.5]]))
```

```
## array([0])
```

2.2 构造 kd 树

首先构造树中的节点，节点中需要储存的有：

- value: 该节点的实例
- idx: 切分坐标轴的下标
- left,right,father: 节点的左、右、父节点

```
class kdNode:
    def __init__(self):
        self.value = None
        self.idx = None
        self.father = None
        self.left = None
        self.right = None
```

定义一个 make_tree 函数，该函数可递归生成一棵 kd 树。

```
import math
def make_tree(node,data,idx):
    if len(data)>1:
        N,k = data.shape
        ## 寻找切分坐标轴和切分点
        idx = (idx+1)%k
        median = sorted(data[:,idx])[math.floor(N/2)]
        ## 从数据中找到切分实例
        median_id = list(data[:,idx]).index(median)
        value = data[median_id,:]
```

```
        ## 储存
        node.value = value
        node.idx = idx
        ## 切分剩余数据集
        data = np.delete(data,median_id,axis=0)
        left_data = data[data[:,idx]<median,:]
        right_data = data[data[:,idx]>=median,:]
        ## 递归左边和右边
        if len(left_data)!=0:
            node.left = kdNode()
            node.left.father = node
            make_tree(node.left,left_data,idx)
        if len(right_data)!=0:
            node.right = kdNode()
            node.right.father = node
            make_tree(node.right,right_data,idx)
    else:
        node.value = data[0]
```

调用该函数即可构建 kd 树。

```
class kdTree:
    def __init__(self,data):
        self.tree = kdNode()
        make_tree(self.tree,data,-1)
```

使用数据集 T 构建的 kd 树如下：

```
tree = kdTree(X)

def print_tree(tree,layer):
    print('\t'*layer,tree.value)
    if tree.left is not None:
        print_tree(tree.left,layer+1)
    if tree.right is not None:
        print_tree(tree.right,layer+1)

print_tree(tree.tree,0)
```

```
##  [7 2]
##    [5 4]
```

```
##       [2 3]
##       [4 7]
##   [9 6]
##       [8 1]
```

2.3 搜索 kd 树

定义 kdFInd 函数进行递归搜索：

```
def L2_dist(x,y):
    return np.sqrt(np.sum((x-y)**2))

def FindBrother(node):
    if node.father.left is node:
        return node.father.right
    if node.father.right is node:
        return node.father.left

def kdFind(node,x):
    ## 寻找叶节点
    while node.idx is not None:
        if x[node.idx]<node.value[node.idx]:
            if node.left is not None:
                node = node.left
            else:
                break
        else:
            if node.right is not None:
                node = node.right
            else:
                break
    ## 保存当前最近点
    nearest = node.value
    ## 不断向上回退直至根节点
    while node is not None and node.father is not None:
        ## 比较距离
        if L2_dist(node.value,x)<L2_dist(nearest,x):
            nearest = node.value
        ## 是否与兄弟节点相交
```

```
        dist = np.abs((node.value-x)[node.father.idx])
        r = L2_dist(nearest,x)
        ## 判断兄弟节点中有无更近的点
        if r>dist:
            try:
                parent_node = FindBrother(node)
                parent_node.father = None
                tmp = kdFind(parent_node,x)
                ## 存在，改动当前最近点
                if L2_dist(tmp,x)<L2_dist(nearest,x):
                    nearest = tmp
                ## 不存在，继续向上递归
                else:
                    node = node.father
                    continue
            except:
                pass
        node = node.father
    return nearest
```

对于点 $(3, 4.5)^T$，其最邻近点为：

```
kdFind(tree.tree,np.array([3,4.5]))
```

```
## array([2, 3])
```