k近邻

2021-09-17

1. k近邻算法

k近邻算法（k-nearest neighbor，k-NN）简单、直观：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实例最邻近的k个实例，这k个实例的多数属于某个类，就将该类别作为输入实例的预测类别。

算法$1.1(k$近邻法$)$

输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，其中，$x_i\in\mathcal{X}\subseteq\mathbf{R^n}$为实例的特征向量，$y_i\in\mathcal{Y}=\{c_1,c_2,\cdots,c_K\}$为实例的类别，$i=1,2,\cdots,N$；实例特征向量$x$；

输出：实例x所属的类y

(1)根据给定的距离度量，在训练集T中找出与x最邻近的k个点，涵盖这k个点的x的邻域记作$N_k(x)$

(2)在$N_k(x)$中根据分类决策规则（如多数表决）决定x的类别y

$$y=\arg\max_{c_j}\sum_{x_i\in N_k(x)}I\left(y_i=c_j\right),\quad i=1,2,\cdots,N;j=1,2,\cdots,K$$其中，I为指示函数，即当$y_i=c_j$时I为1，否则为0 (4

2. k近邻模型

k近邻算法中，当训练集、距离度量、k值及分类决策规则（如多数表决）确定后，对于任何一个新的实例，它所属的类唯一地确定。

2.1. 距离度量

常用的是欧式距离、曼哈顿距离。

2.2. k值的选择

k值越小，越容易过拟合，即模型越复杂。

k值越大，最大为N，即无论输入任何实例，都将简单地预测它属于在训练实例中最多的类，即模型越简单。

实际中如何选择？先取较小的数，然后使用交叉验证来选择最优的k值。

2.3. 分类决策规则

常用的是多数表决规则。

多数表决规则有如下解释：如果分类的损失函数是0-1损失函数，分类函数为 $f:\mathbf{R}^n\to\{c_1,c_2,\cdots,c_K\}$。那么误分类的概率是：

$$P(Y\neq f(X))=1-P(Y=f(X)) \tag{5}$$

对给定的实例$x\in\mathcal{X}$，其最近邻的k个训练实例点构成集合$N_k(x)$。如果涵盖$N_k(x)$的区域的类别是c_j（真实标签），那么误分类率是：

$$\frac{1}{k}\sum_{x_i\in N_k(x)}I\left(y_i\neq c_j\right)=1-\frac{1}{k}\sum_{x_i\in N_k(x)}I\left(y_i=c_j\right) \tag{6}$$

要使误分类率最小，即经验风险最小，就要使$\sum_{x_i \in N_k(x)} I\left(y_i = c_j\right)$最大，如式（1）多数表决规则所述。因此，多数表决规则等价于经验风险最小化。

3. kd树

为了提高k近邻搜索的效率，可以考虑使用特殊的结构存储训练数据，以减少计算距离的次数。具体方法很多，比如kd树方法。

3.1. 构造kd树

kd树是一种对k维空间中的实例点进行存储以便能够快速检索的树形数据结构。kd树是二叉树，表示对k维空间的一个划分。构造kd树相当于不断地用垂直于坐标轴的超平面将k维空间划分，构成一系列的k维超矩形区域。

通常，依次选择坐标轴对空间切分，选择训练实例点在选定坐标轴上的中位数作为切分点，这样得到的kd树是平衡的。但是，平衡的kd树搜索时效率未必是最优的。

算法1.2（构造平衡kd树）

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

输出：kd树

步骤：

(1)开始：构造根结点，根结点对应于包含T的k维空间的超矩形区域。

选择$x^{(1)}$为坐标轴，以T中所有实例的$x^{(1)}$坐标的中位数为切分点，将根结点对应的超矩形区域切分为两个子区域，切分由通过切分点并与坐标轴$x^{(1)}$垂直的超平面实现。

由根结点生成深度为1的左、右子结点：左子结点对应坐标$x^{(1)}$小于切分点的子区域，右子结点对应于坐标$x^{(1)}$大于切分点的子区域。

将落在切分超平面上的实例点保存在根结点。

(2)重复：对深度为j的结点，选择$x^{(l)}$为切分的坐标轴，$l=(j-1)\%k+1$，以该结点的区域中所有实例的$x^{(l)}$坐标的中位数为切分点

将该结点对应的超矩形区域切分为两个子区域。切分由通过切分点并与坐标轴$x^{(l)}$垂直的超平面实现。

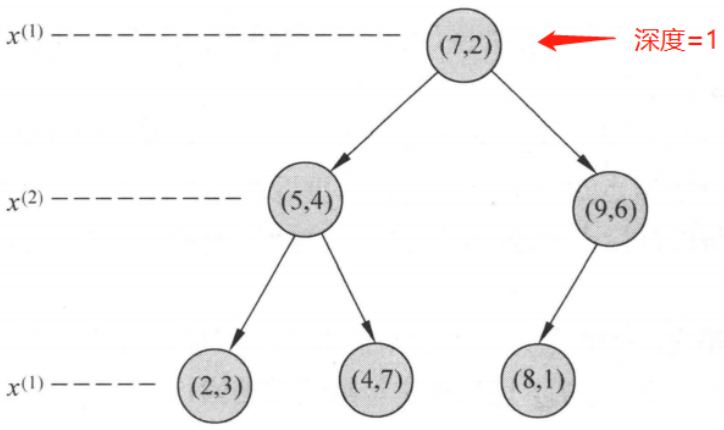
由该结点生成深度为$j+1$的左、右子结点：左子结点对应坐标$x^{(l)}$小于切分点的子区域，右子结点对应坐标$x^{(l)}$大于切分点的子区域。

将落在切分超平面上的实例点保存在该结点。

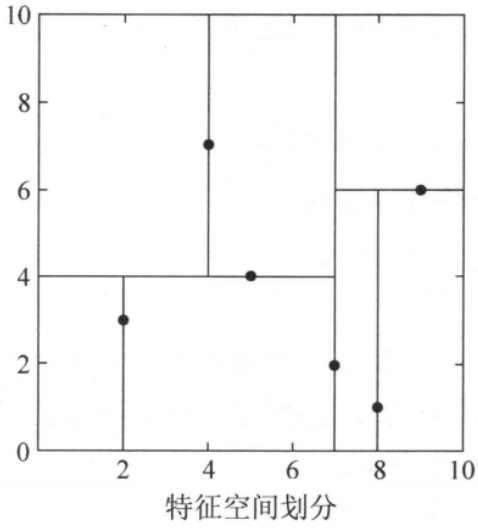
(3)直到两个子区域没有实例存在时停止。从而形成kd树的区域划分。

例题 给定一个二维空间的数据集，构造一个平衡的kd树。

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



kd 树示例



特征空间划分

3.2. 搜索kd树

利用kd树可以省去大部分数据点的搜索，从而减少搜索的计算量。以**最近邻搜索**为例，对算法进行描述。

算法1.3（基于kd树的最近邻搜索）

输入：已构造的kd树；目标点x

输出：x的最近邻点

步骤：

（1）在kd树中找出包含目标点x的叶结点：从根结点出发，使用DFS递归地向下访问kd树。若目标点x当前维的坐标小于切分点的坐标，则转移到左子结点，否则移动到右子结点。直到子结点为叶结点为止，同时在$stack$中存储已访问的结点。

（2）以此叶结点为"当前最近点"

（3）递归地向上回退，从步骤（2）的叶结点开始，在每个结点进行以下操作：

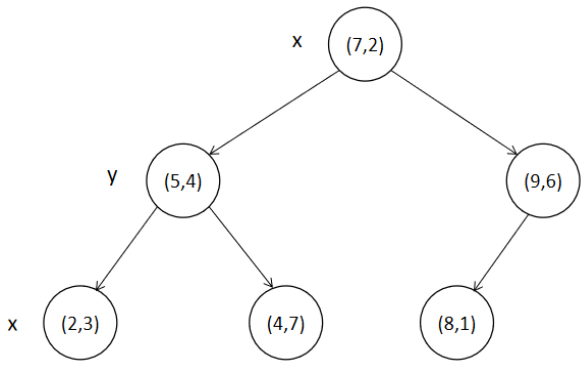
（a）如果该结点保存的实例点比当前最近点距离目标点更近，则以该实例点为"当前最近点"。

（b）检查以目标点为球心、以目标点与"当前最近点"间的距离为半径所构成的超球体是否与父节点的超平面相交。
如果相交，则到该结点的兄弟结点，同样采用DFS，开始检查最近邻结点。
如果不相交，则继续回溯，兄弟结点一侧全部被淘汰，不再考虑范围。

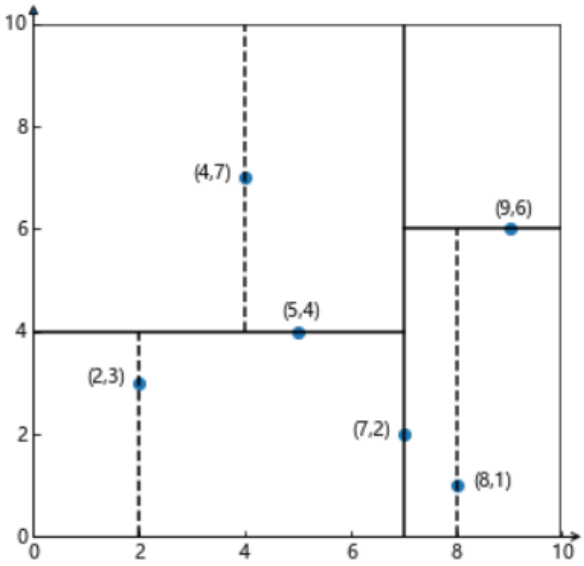
（4）当回退到根结点时，搜索结束。最后的"当前最近点"即为x的最近邻点。

3.3. kd树搜索例题

已知树形图如下：

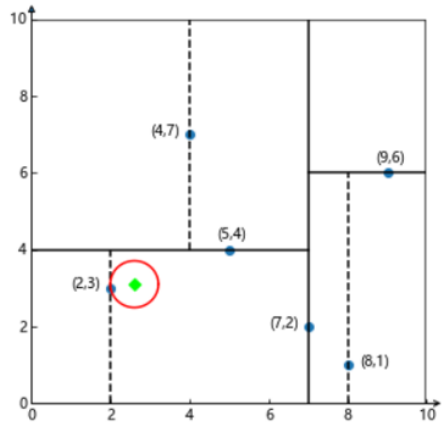


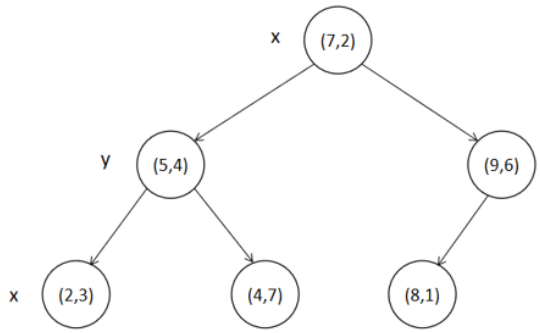
划分图如下：



请对点(2.6, 3.1)和点(2, 4.5)分别进行最近邻结点的搜索。

1. 点(2.6, 3.1)搜索

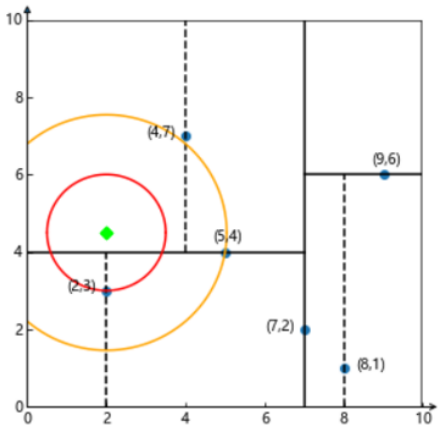


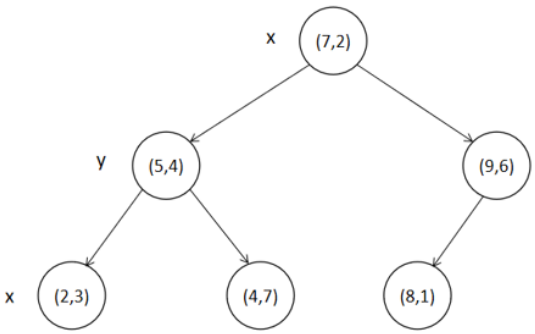


如上图所示，绿色的点为目标点。

- 寻找叶子结点：搜索路径为{(7. 2)，(5, 4)，(2, 3)}，最终定位属于叶子结点(2, 3)。
- 以(2, 3)作为当前最近邻点，计算其到查询点(2.6, 3.1)的距离为0.6083。
- 回溯到父节点(5, 4)，并判断在父结点的其它子结点空间中是否有距离目标点更近的数据点。以(2.6, 3.1)为圆心，以0.6083为半径画圆，发现该圆并不和超平面y=4相交，因此不用进入(5, 4)结点的右子结点空间中进行搜索。
- 再回溯到(7. 2)，以(2.6, 3.1)为圆心，以0.6083为半径画圆不会与x=7的超平面相交，因此不用进入(7. 2)的右子结点空间进行搜索。
- 搜索完毕，返回最近邻结点(2, 3)，最近邻距离为0.6083。

2. 点(2, 4.5)搜索





- 寻找叶子结点：搜索路径为{(7, 2)，(5, 4)，(4,7)}，最终定位属于叶子结点(4, 7)。
- 以(4, 7)为最近邻点，计算与目标点(2, 4.5)之间的距离为3.202。
- 回溯到 (5, 4)，计算与目标点(2, 4.5)之间的距离为3.041，因此 (5, 4)为当前的最近邻点。
- 以(2, 4.5)为圆心、3.041为半径画圆（上图橙色圆），可见该圆与超平面y=4相交，因此需要进入 (5, 4)结点的左子结点(2, 3)进行搜索，此时搜索的路径为{(7, 2)，(2, 3)}
- 回溯至(2, 3)叶子结点，(2, 3)距离目标点(2, 4.5)的距离为1.5，所以当前的最近邻结点为(2, 3)。

- 回溯至(7, 2)结点，以(2, 4.5)为圆心、1.5为半径画圆（上图红色圆），并不和x=7超平面相交，因此，不需要进入(7, 2)的右子结点进行搜索。
- 搜索完毕，返回最近邻结点(2, 3)，最近邻距离为1.5。