《数据挖掘技术》

⭐ CH03 K近邻法

➡️ Created by *Wang JingHui*

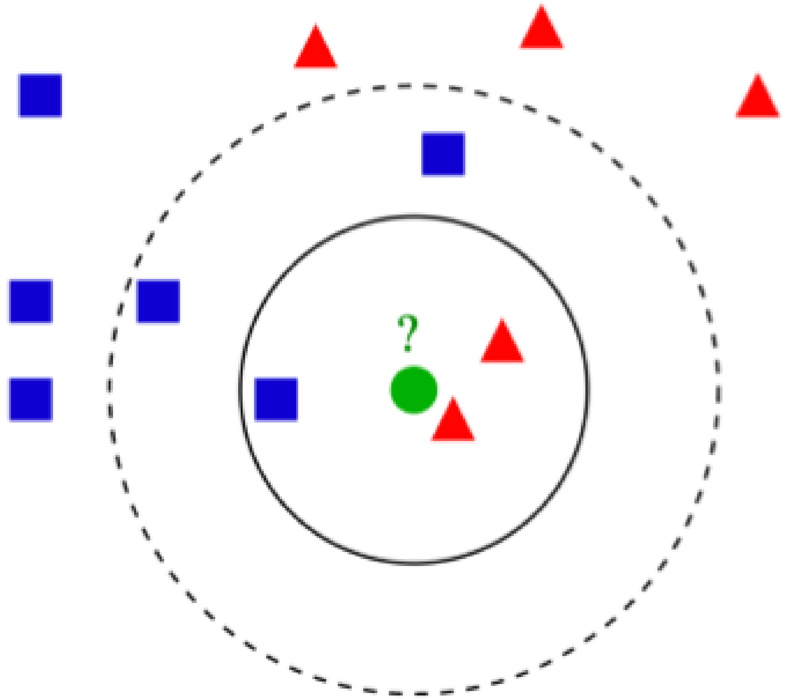
➡️ Version *4.0*

主要内容

1. k近邻算法
2. k近邻模型
    i. 模型
    ii. 距离度量
    iii. k值选择
    iv. 分类决策规则
3. k近邻法的实现: KDTree
    i. 构造KDTree
    ii. 搜索KDTree

概述

- k 近邻法是一种基本分类与回归方法；
- 三个基本要素；
- 1968年由 Cover 和 Hart 提出。



Example of *k*-NN classification

The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles.

- If $k = 3$ (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle.
- If $k = 5$ (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

最近邻算法

$k=1$的情形, 称为最近邻算法. 书中后面的分析都是按照最近邻做例子, 这样不用判断类别, 可以略去一些细节.

k近邻模型

> **输入:** 训练数据集$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$，$x_i\in\mathcal{X}\subseteq\mathbf{R}^{\mathbf{n}}$
> $y_i\in\mathcal{Y}=\{c_1,c_2,\ldots,c_k\}$;
> 实例特征向量x

> **输出:** 实例所属的y

步骤:

1. 根据指定的**距离度量**，在T中查找x的**最近邻的k个点**，覆盖这k个点的x的邻域定义为$N_k(x)$
2. 在$N_k(x)$中应用**分类决策规则**决定x的类别y

$$y=\arg\max_{c_j}\sum_{x_i\in N_k(x)}I(y_i=c_j),i=1,2,\ldots,N,j=1,2,\ldots,K$$

距离度量

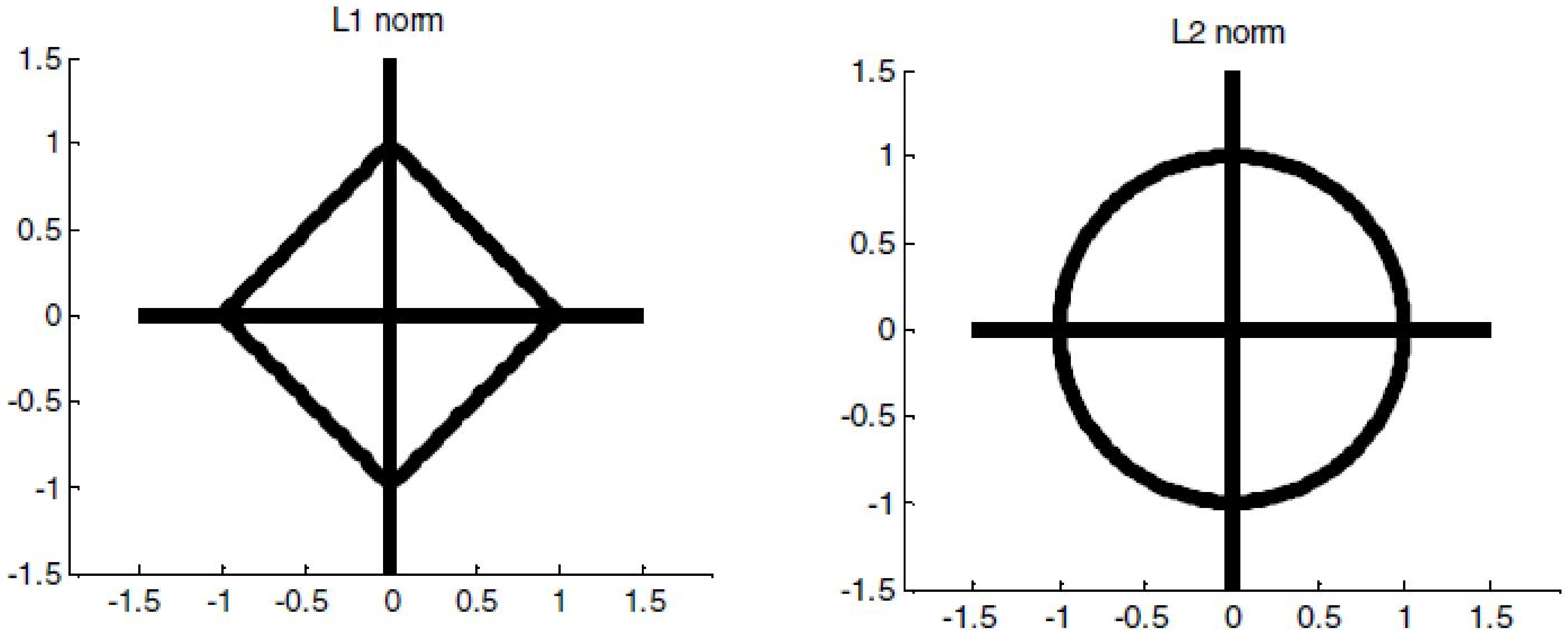
> **特征空间**中的两个实例点的距离是两个实例点相似程度的反映。

距离越近(数值越小), 相似度越大。

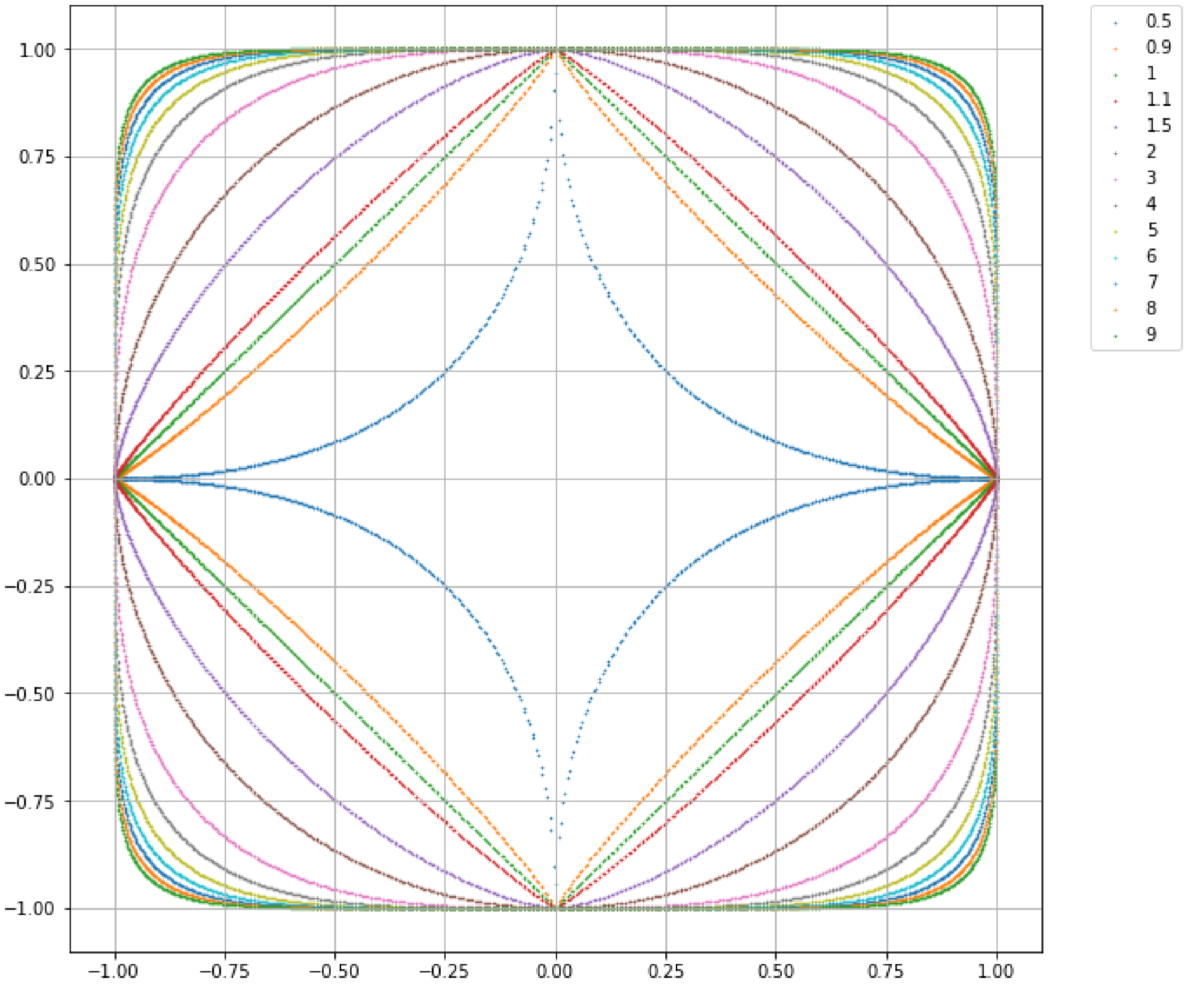
这里用到了L_p距离, 可以参考Wikipedia上L_p Space词条[^1]

1. $p=1$ 对应 曼哈顿距离
2. $p=2$ 对应 欧氏距离
3. 任意p 对应 闵可夫斯基距离

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



L1 norm
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
L2 norm
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5



考虑二维的情况, 上图给出了不同的p值情况下与原点距离为1的点的图形。

这个图有几点理解下:

1. 与原点的距离
2. 与原点距离为1的点
3. 前一点换个表达方式, 图中的点向量(x_1, x_2)的p范数都为1
4. 图中包含多条曲线, 关于p=1并没有对称关系
5. 定义中$p \geqslant 1$，这一组曲线中刚好是凸的

范数是对向量或者矩阵的度量，是一个标量，这个里面两个点之间的L_p距离可以认为是两个点坐标差值的p范数。

例3.1

已知二维空间的3个点，$x_1 = (1,1)^T, x_2 = (5,1)^T, x_3 = (4,4)^T$，求在$p$取不同值时的最近邻点。

k值选择

1. 如果选择较小的k值，相当于用较小的邻域中的训练实例进行预测，整体模型复杂，容易过拟合；较大的k值，整体模型变得简单。
2. 通过**交叉验证**选取最优k, 算是超参数。
3. 二分类问题, k选择奇数有助于避免平票。

分类决策规则(Majority Voting Rule)

常常采用**多数表决**

- 误分类率

$$\frac{1}{k}\sum_{x_i \in N_k(x)} I(y_i \neq c_i) = 1 - \frac{1}{k}\sum_{x_i \in N_k(x)} I(y_i = c_i)$$

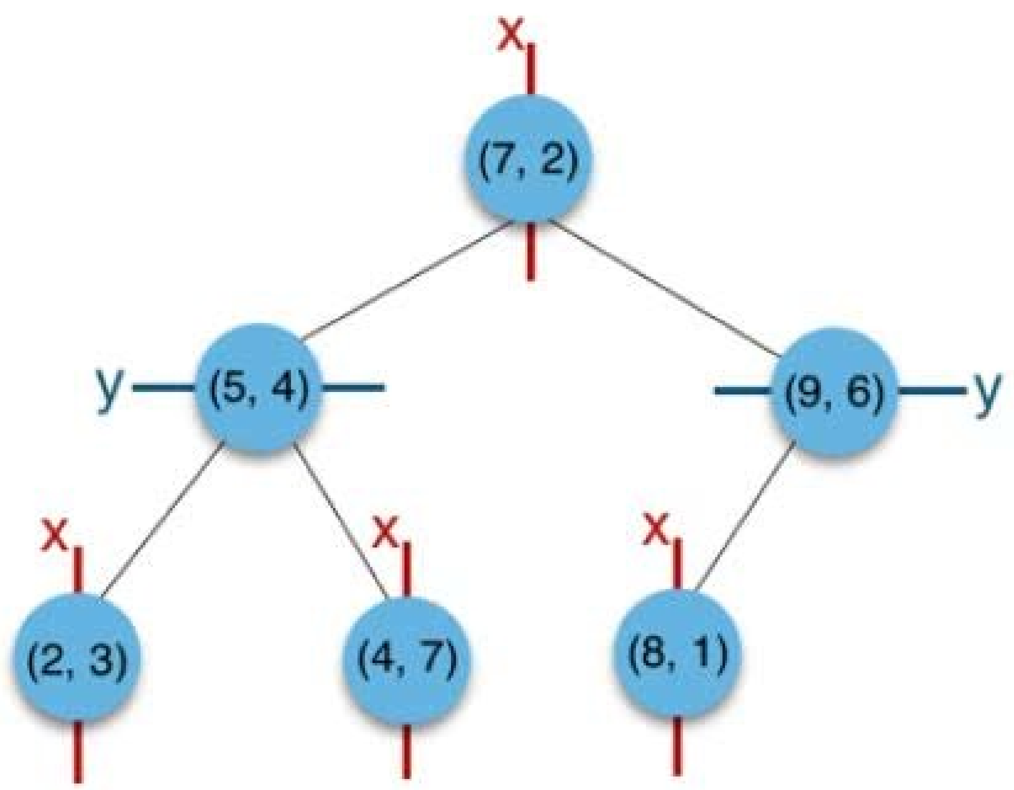
- 如果分类损失函数是0-1损失，误分类率最低即经验风险最小。

⚛️ KD树实现

kNN在实现的时候，要考虑多维数据的存储，这里会用到树结构。

kd树（k-dimensional树的简称），是一种对k维空间（注意，k不是k个邻居的意思）中的实例点进行存储以便对其进行快速搜索的二叉树结构。

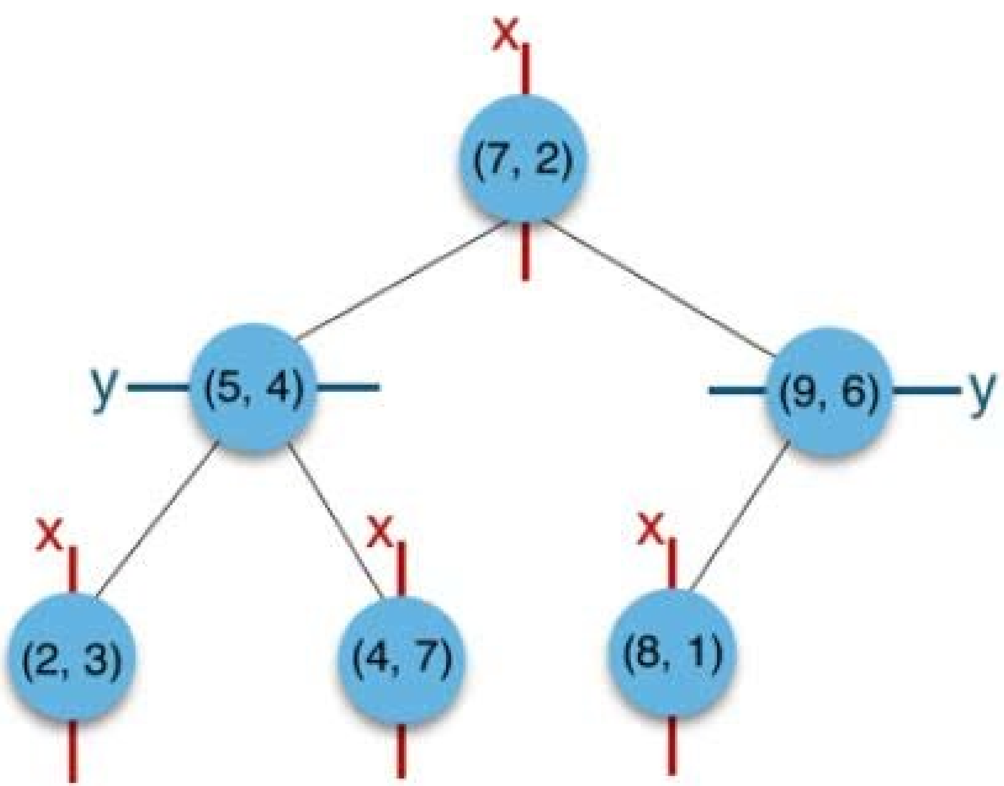
在Scipy Cookbook里面有个kd树具体的实现[^2]可参考

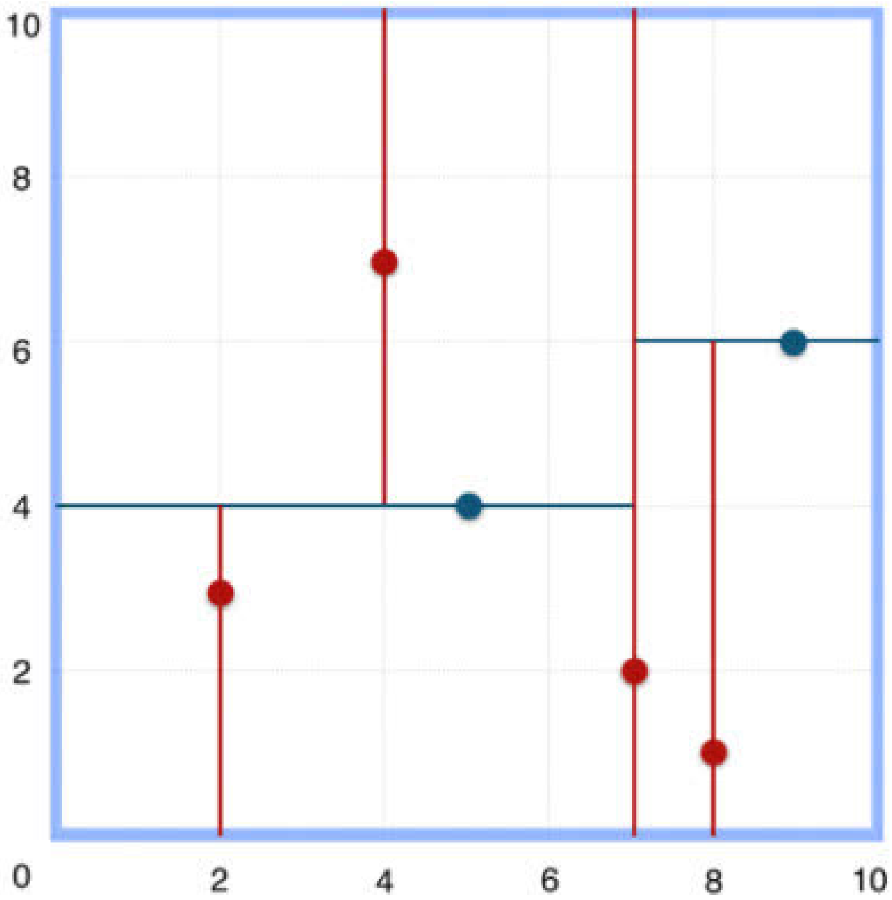


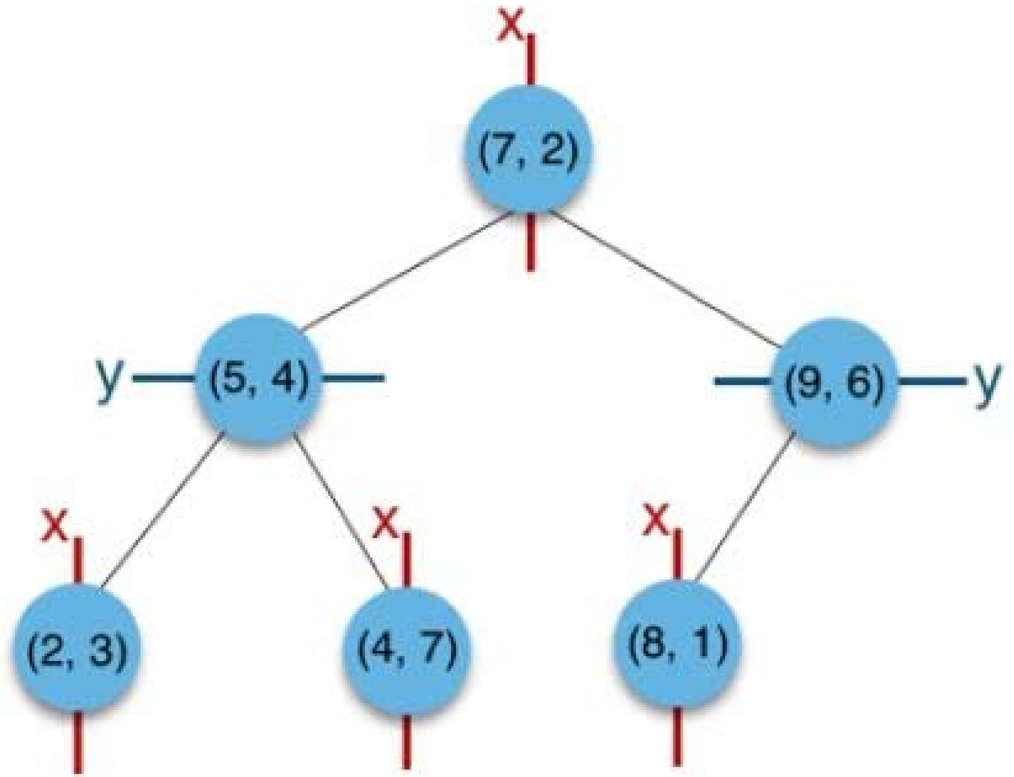
KD树创建

集合(2,3)，(5,4)，(9,6)，(4,7)，(8,1)，(7,2)。

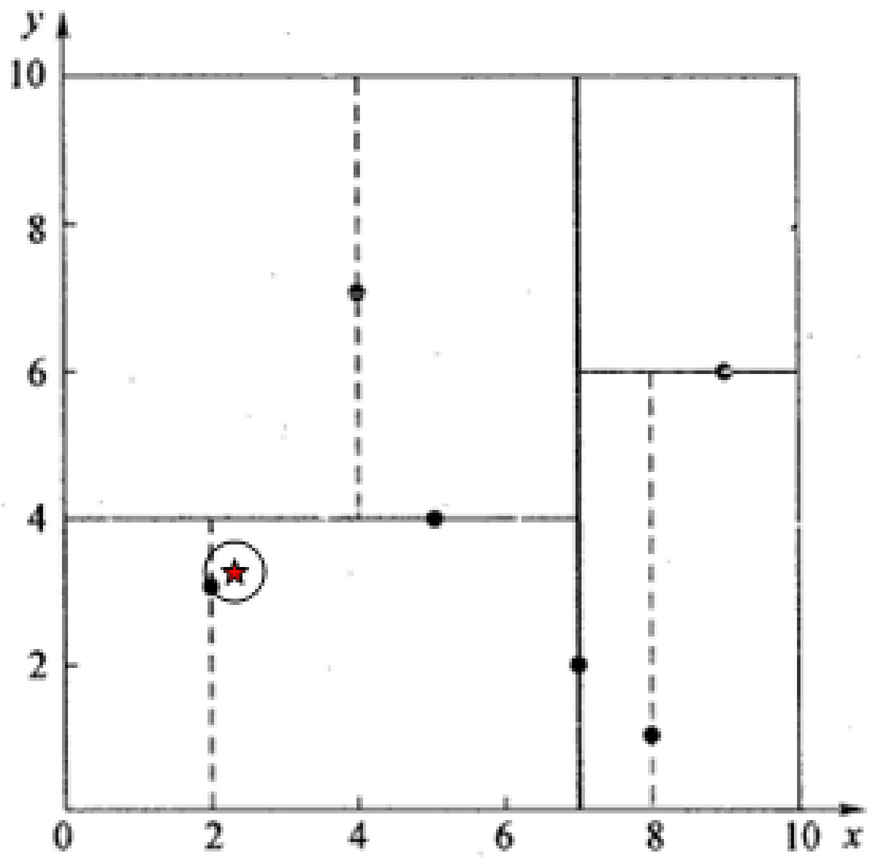
- 构建根节点时，切分维度为x，集合在x维从小到大排序为(2,3)，(4,7)，(5,4)，(7,2)，(8,1)，(9,6)；x维中位数为（5+7）/2=6，但是点必须在集合中，因此选择点(7,2)。
- (2,3)，(4,7)，(5,4)挂在(7,2)节点的左子树，(8,1)，(9,6)挂在(7,2)节点的右子树。
- 以此类推。


x
(7, 2)
y
(5, 4)
(9, 6)
y
x
x
x
(2, 3)
(4, 7)
(8, 1)


10
8
6
4
2
0
2
4
6
8
10



KD树查找（2.1,3.1)



平衡kd树构造算法：

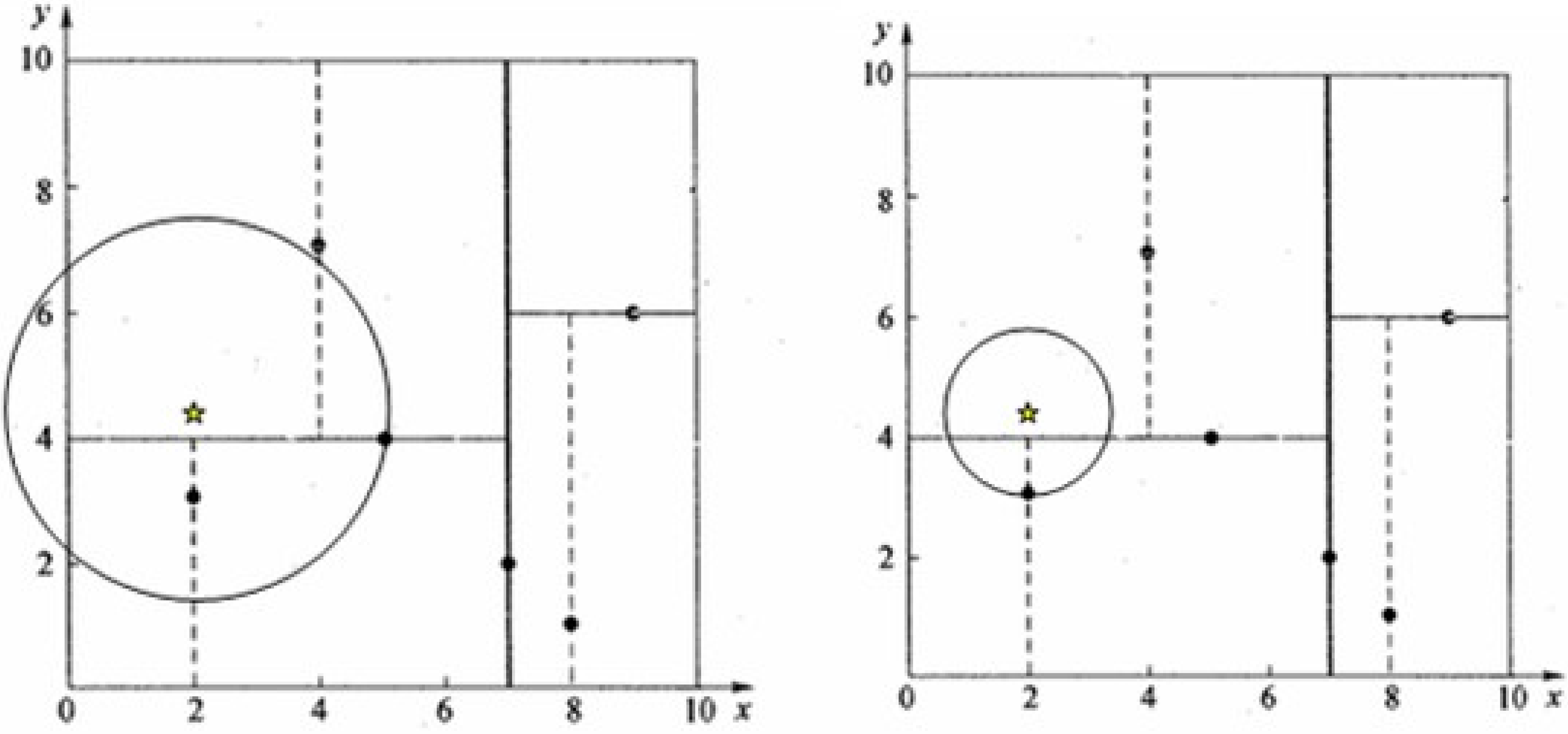
输入：k维空间数据集$T = \{x_1, x_2, \cdots, x_N\}$,

- 其中$x_i = \left(x_i^{(1)}, x_i^{(1)}, \cdots, x_i^{(k)}\right)^T, i = 1, 2, \cdots, N$;

输出：kd树

1. **开始：** 构造根结点，根结点对应于包涵T的k维空间的超矩形区域。选择$x^{(1)}$为坐标轴，以T中所欲实例的$x^{(1)}$坐标的中位数为切分点，将根结点对应的超矩形区域切分成两个子区域。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：坐子结点对应坐标$x^{(1)}$小于切分点的子区域，右子结点对应于坐标$x^{(1)}$大与切分点的子区域。
   将落在切分超平面上的实例点保存在跟结点。
2. **重复：** 对深度为j的结点，选择$x^{(l)}$为切分坐标轴，$l = j \pmod k + 1$，以该结点的区域中所由实例的$x^{(l)}$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。
   由根结点生成深度为$j+1$的左、右子结点：坐子结点对应坐标$x^{(l)}$小于切分点的子区域，右子结点对应于坐标$x^{(l)}$大与切分点的子区域。
   将落在切分超平面上的实例点保存在跟结点。
3. 直到两个子区域没有实例存在时停止。

KD树查找（2,4.5)



KD树的最近邻搜索算法：

输入：kd树；目标点x
输出：x的最近邻

1. 在kd树中找出包含目标点x的叶结点：从跟结点出发，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，则移动到左子结点，否则移动到右子结点。直到子结点为叶结点为止。
2. 以此叶结点为“当前最近点”。

3. 递归地向上回退，在每个结点进行以下操作：

   3.1 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”。

   3.2 当前最近点一定存在于该结点一个子结点对应的区域。检查该子结点的父结点的另一子结点对应的区域是否有更近的点。具体地，检查另一子结点对应的区域是否与以目标点为球心、以目标点与“当前最近点”间的距离为半径的超球体相交。

   如果相交，可能在另一个子结点对应的区域内存在距目标点更近的点，移动到另一个子结点。接着，递归地进行最近邻搜索；

   如果不相交，向上回退。

4. 当回退到根结点时，搜索结束。最后的“当前最近点”即为x的当前最近邻点。

KDTree总结

- KDTree的构建是一个递归的过程
- 注意: KDTree左边的点比父节点小，右边的点比父节点大。
- 这里面有提到，KDTree搜索时效率未必是最优的，这个和样本分布有关系. 随机分布样本**KDTree搜索**(这里应该是**最**近邻搜索)的平均计算复杂度是$O(\log N)$, 空间维数K接近训练样本数N时, 搜索效率急速下降, 几乎$O(N)$。
- 如果维度比较高, 搜索效率很低. 当然, 在考虑维度的同时也要考虑样本的规模，这种查找方法就是二分查找，其算法复杂度为$O(Log2(N))$。

参考

1. Lp Space
2. ESL
3. KDTree
4. KD Tree: k近邻查询和范围查询
5. HUD4347

↘ Enjoy your machine learning!

https://github.com/wjssx/

E-mail: csr_dsp@sina.com