K近邻算法

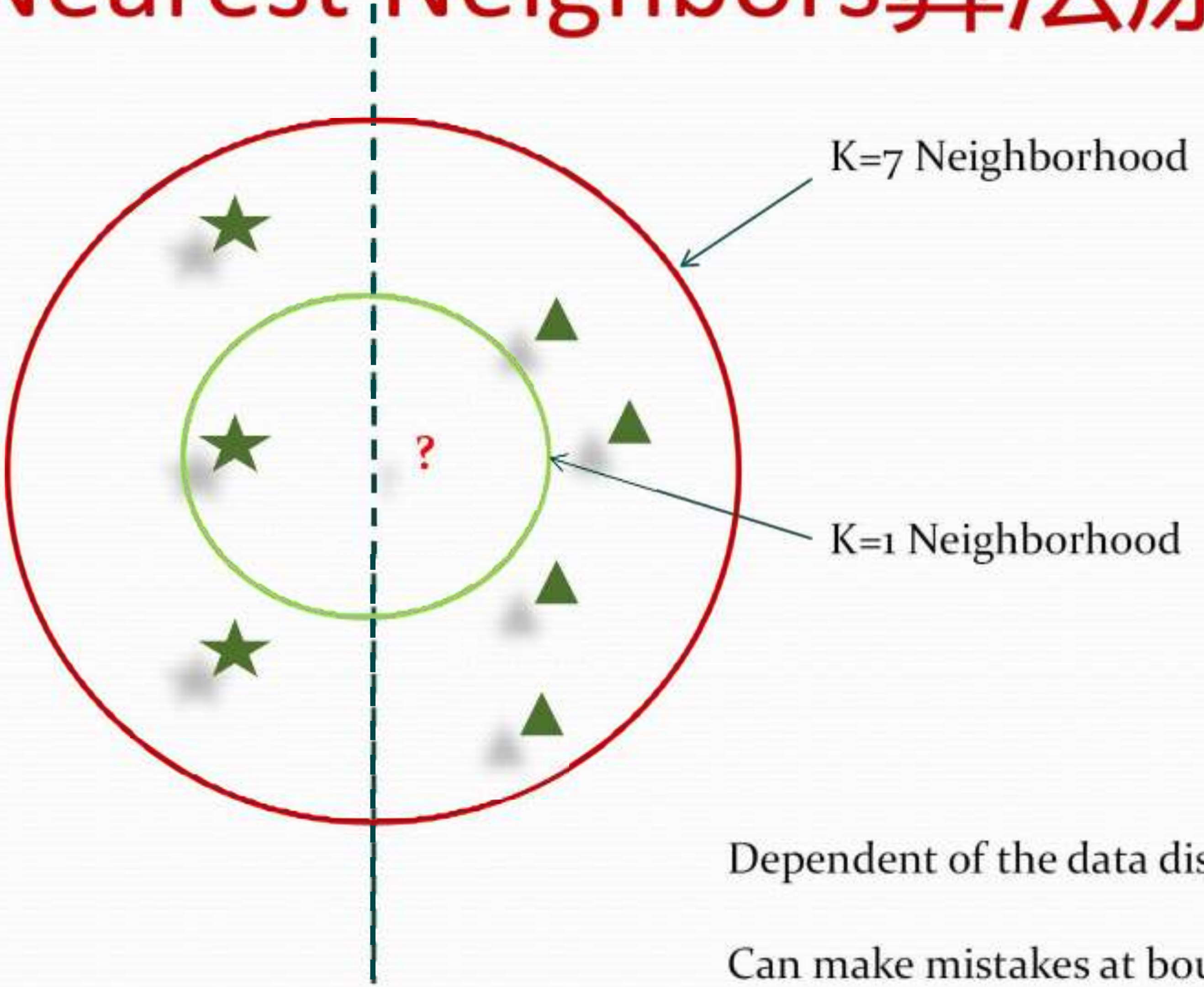
目录

1. *k* 近邻算法
2. *k* 近邻模型
3. *k* 近邻法的实现： *kd* 树

一、*k* 近邻算法

- 原理
- 特点
- 一般流程

K-Nearest Neighbors算法原理



Dependent of the data distributions.

Can make mistakes at boundaries.

K-Nearest Neighbors算法特点

- 优点
- 精度高
- 对异常值不敏感
- 无数据输入假定
- 缺点
- 计算复杂度高
- 空间复杂度高
- 适用数据范围
- 数值型和标称型

K-Nearest Neighbors Algorithm

- 工作原理
- 存在一个样本数据集合，也称作训练样本集，并且样本 集中每个数据都存在标签，即我们知道样本集中每个数 据与所属分类的对应关系。
- 输入没有标签的新数据后，将新数据的每个特征与样本 集中数据对应的特征进行比较，然后算法提取样本集中 特征最相似数据（最近邻）的分类标签。
- 一般来说，只选择样本数据集中前N个最相似的数据。K 一般不大于20，最后，选择k个中出现次数最多的分类， 作为新数据的分类

K近邻算法的一般流程

- 收集数据：可以使用任何方法
- 准备数据：距离计算所需要的数值，最后是结构化的数据格式。
- 分析数据：可以使用任何方法

训练算法：（此步骤kNN）中不适用

测试算法：计算错误率

使用算法：首先需要输入样本数据和结构化的输出结果，然后运行k-近邻算法判定输入数据分别属于哪个分类，最后应用对计算出的分类执行后续的处理。

二、*k* 近邻模型

- 模型
- 距离度量
- *k* 值的选择
- 分类决策规则

模型

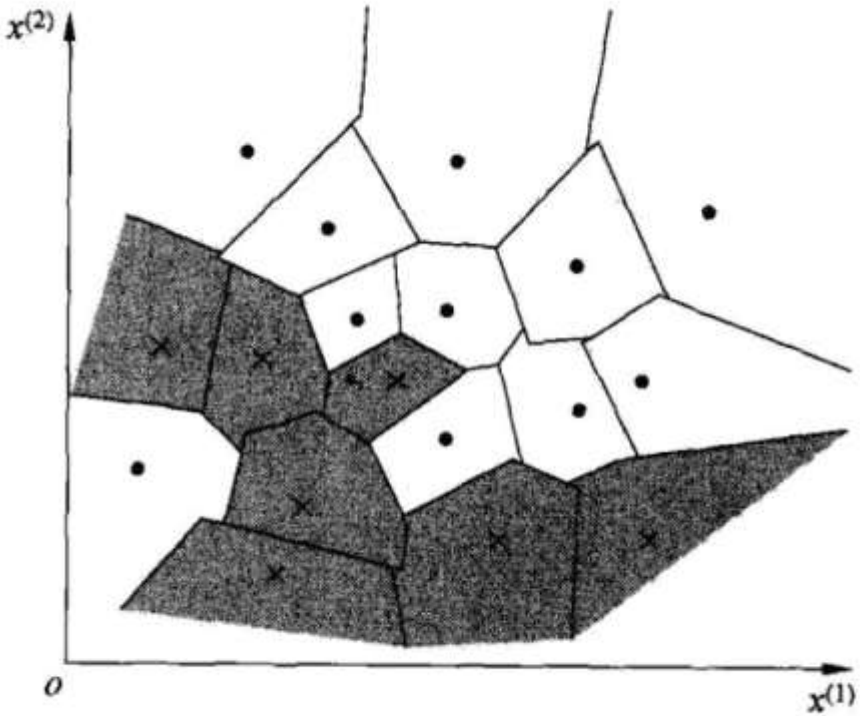


图 3.1　k 近邻法的模型对应特征空间的一个划分

距离度量

$$x_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(n)})^{\mathrm{T}}$$

- Lp距离：

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

- 欧式距离：

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

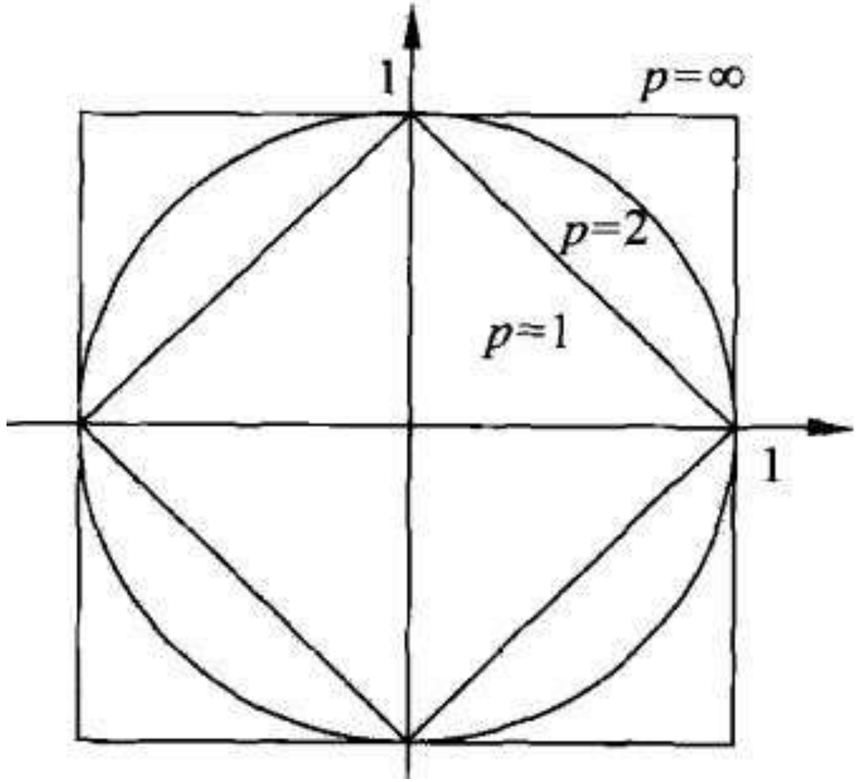
- 曼哈顿距离

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

- L∞距离

$$L_\infty(x_i,x_j)=\max_{l}|x_i^{(l)}-x_j^{(l)}|$$

距离度量



K值的选择

- 如果选择较小的K值
- “学 习” 的近似误差（approximation error)会减小，但 “学习” 的估计误差（estimation error) 会增大，
- 噪声敏感
- K值的减小就意味着整体模型变得复杂，容易发生过拟 合.

- 如果选择较大的K值，
- 减少学习的估计误差，但缺点是学习的近似误差会增大.
- K值的增大 就意味着整体的模型变得简单.

分类决策规则

- 多数表决规则（经验风险最小化）

分类函数 $f:\mathbf{R}^n \to \{c_1, c_2, \cdots, c_K\}$

误分类率 $P(Y \neq f(X)) = 1 - P(Y = f(X))$

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

三、*k* 近邻法的实现：*kd* 树

- 构造 *kd* 树
- 搜索 *kd* 树

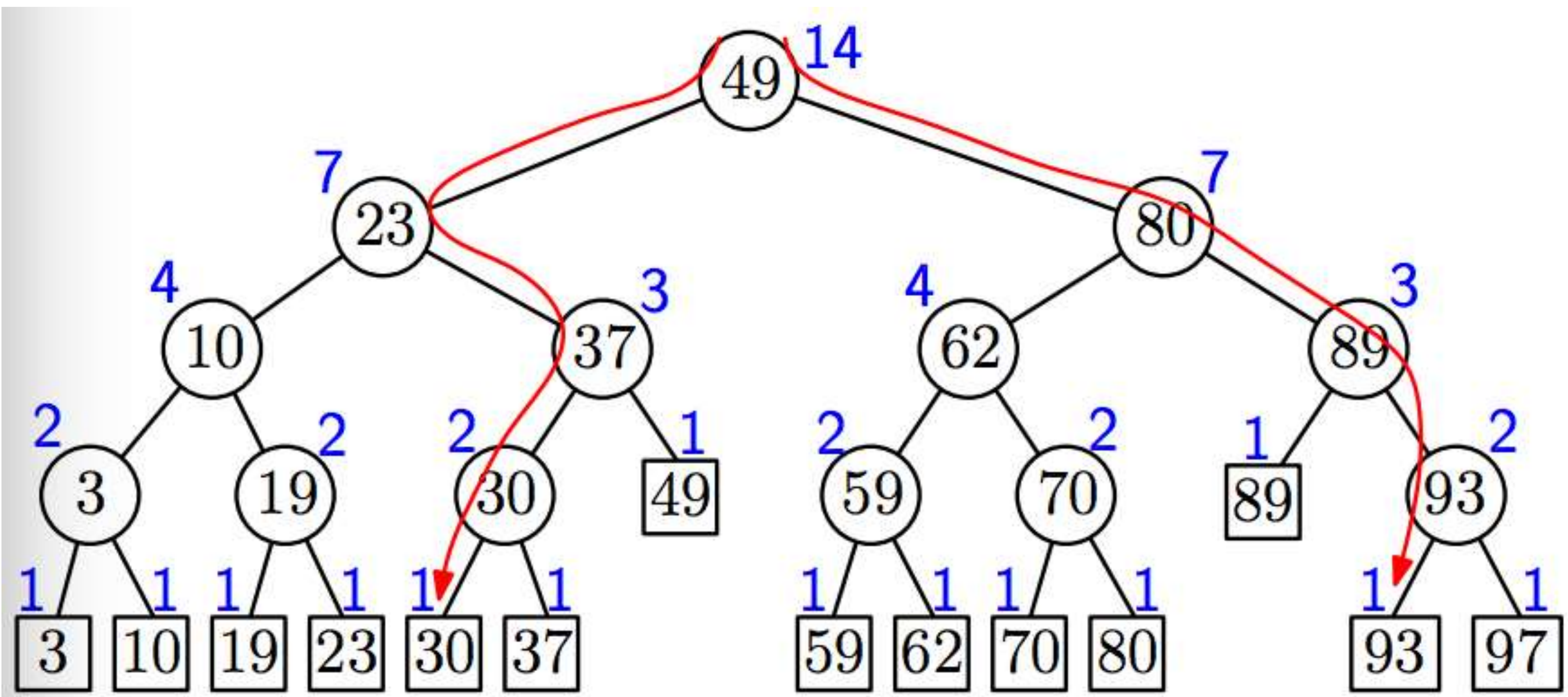
KD树

- kd树是一种对K维空间中的实例点进行存储以便对其进行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维空间切分，构成一系列的k维超矩形区域.Kd树的每个结点对应于一个k维超矩形区域.

KD树

如果要查询语文成绩介于30～93分的学生，如何处理？假设学生数量为N，如果顺序查询，则其时间复杂度为O(N)，当学生规模很大时，其效率显然很低，如果使用平衡二叉树，则其时间复杂度为O(logN)，能极大地提高查询效率。平衡二叉树示意图为：

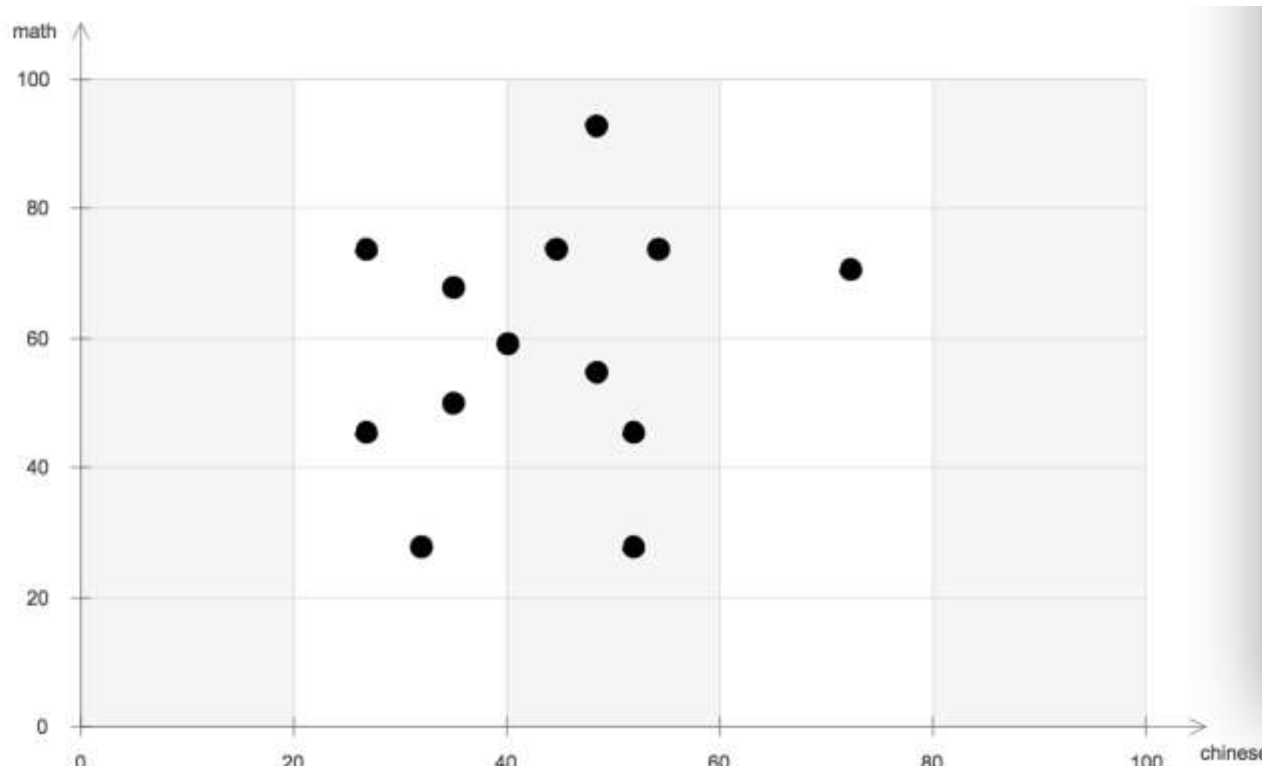


KD树

- 如果现在将查询条件变为：语文成绩介于30～93，数学成绩结余30～90，该如何处理？
- 如果分别使用平衡二叉树对语文成绩和数学成绩建立索引，则需要先在语文成绩中查询得到集合S_1，再在数学成绩中查询得到集合S_2，然后计算S_1和S_2的交集，若$|S_1|=m,|S_2|=n$，则其时间复杂度为$O(m*n)$，有没有更好的办法？

KD树

如果先根据语文成绩，将所有人的成绩分成两半，其中一半的语文成绩<=c_1，另一半的语文成绩>c_1，分别得到集合S_1,S_2;然后针对S_1，根据数学成绩分为两半，其中一半的数学成绩<=m_1,另一半的数学成绩>m_1，分别得到S_3,S_4，针对S_2，根据数学成绩分为两半，其中一半的数学成绩<=m_2,另一半的数学成绩>m_2，分别得到S_5,S_6；再根据语文成绩分别对S_3,S_4，S_5,S_6继续执行类似划分得到更小的集合，然后再在更小的集合上根据数学成绩继续,...



KD树

- 构造kd树：
- 对深度为j的节点，选择x^l为切分的坐标轴 $l = j(\bmod k)+1$
- 例： $T=\{(2,3)^T,(5,4)^T,(9,6)^T,(4,7)^T,(8,1)^T,(7,2)^T\}$

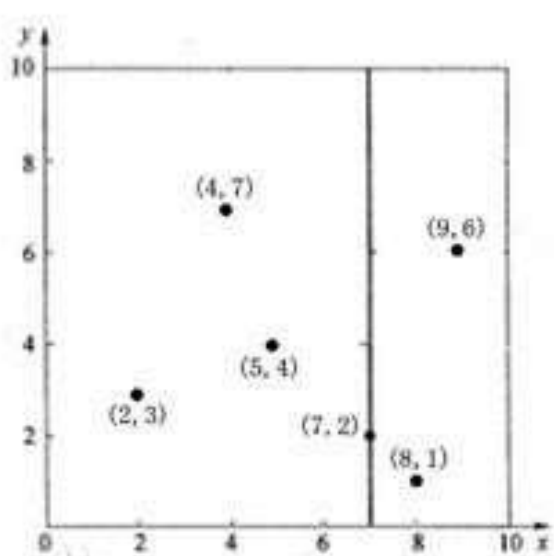


图2 x=7将整个空间分为两部分

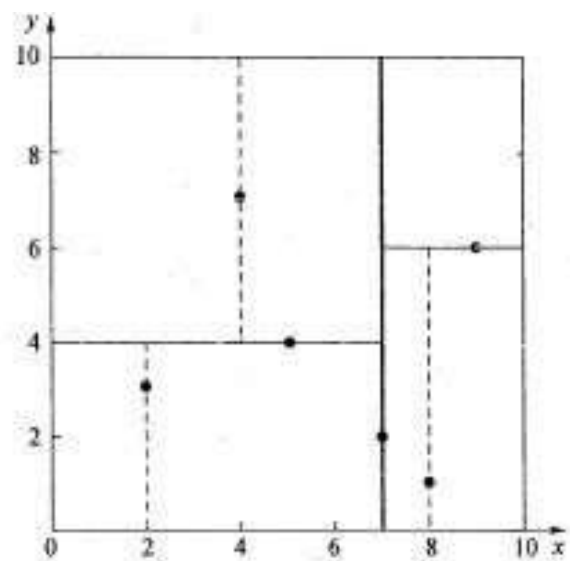
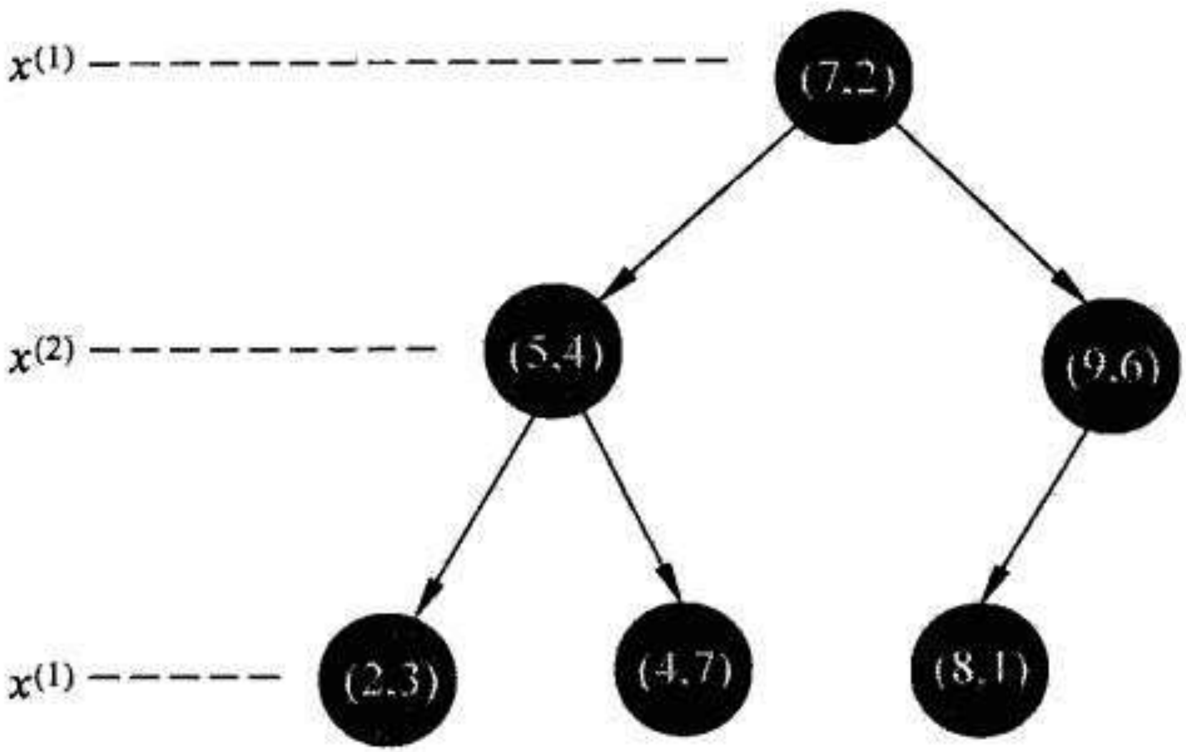


图1 二维数据k-d树空间划分示意图

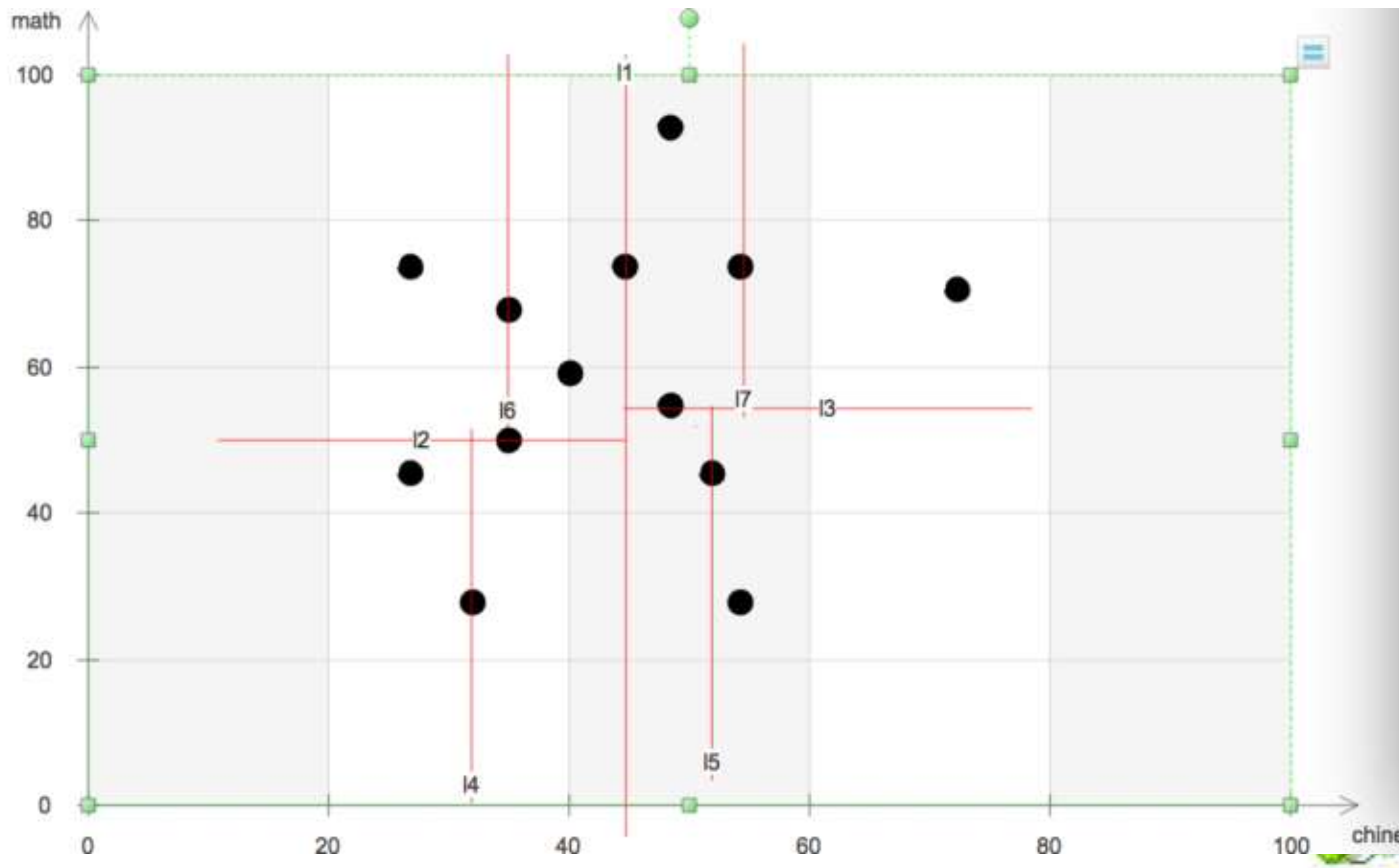
KD树

- {(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)},
- 建立索引

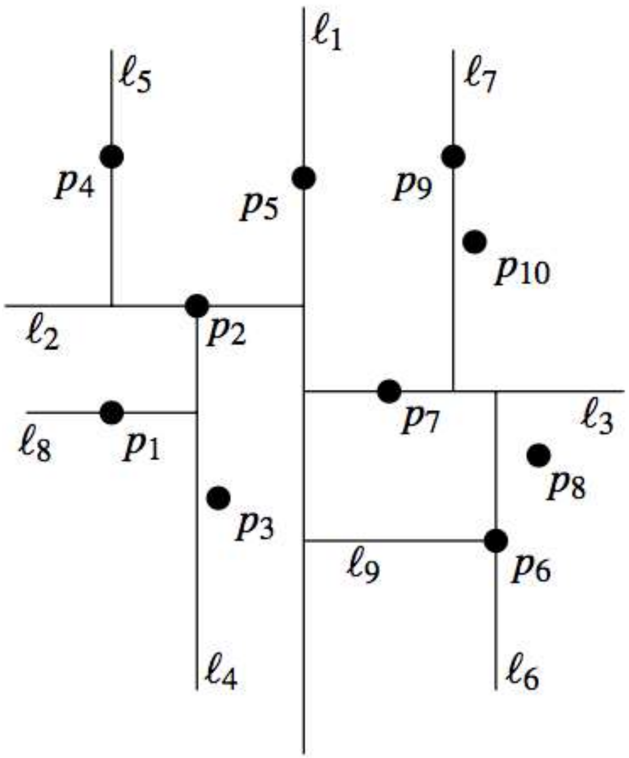


KD树

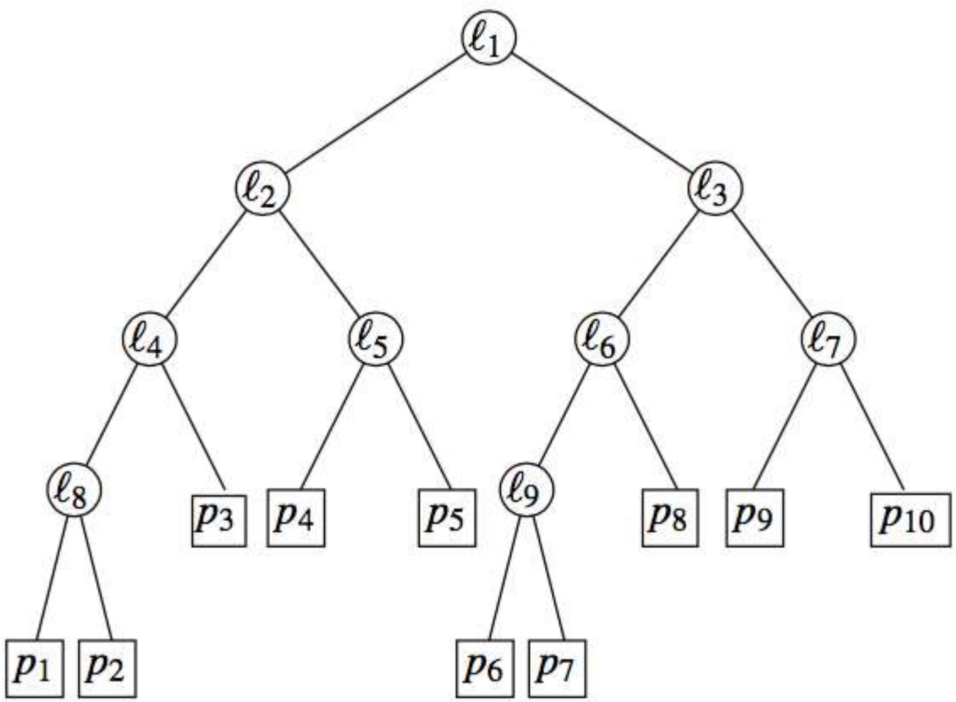
l1左边都是语文成绩低于45分，l1右边都是语文成绩高于45分的；l2下方都是语文成绩低于45分且数学成绩低于50分的，l2上方都是语文成绩低于45分且数学成绩高于50分的，后面以此类推。



KD树



ℓ1
ℓ2
ℓ3
ℓ4
ℓ5
ℓ6
ℓ7
ℓ8
ℓ9
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10



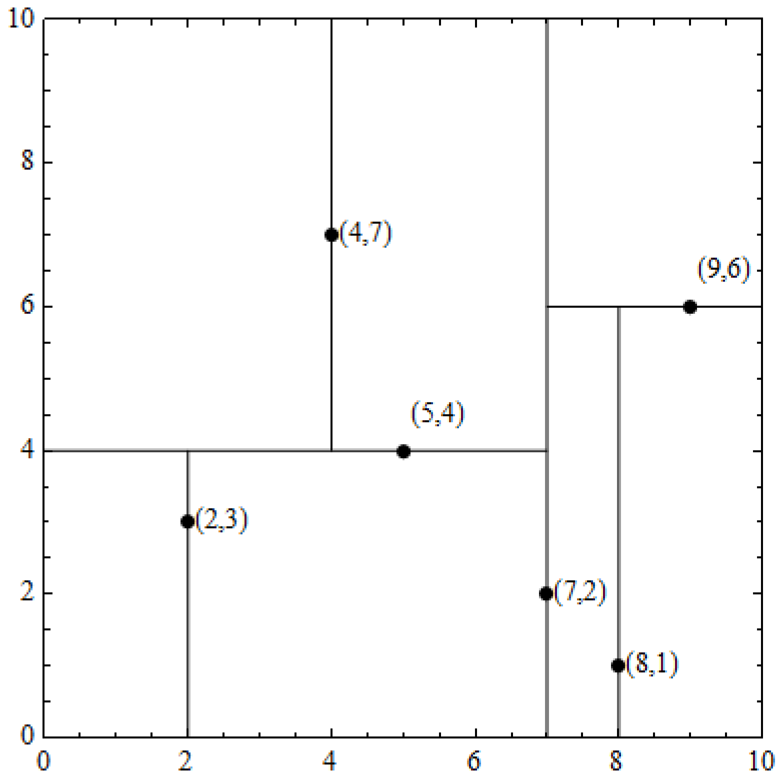
ℓ1
ℓ2
ℓ3
ℓ4
ℓ5
ℓ6
ℓ7
ℓ8
ℓ9
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10

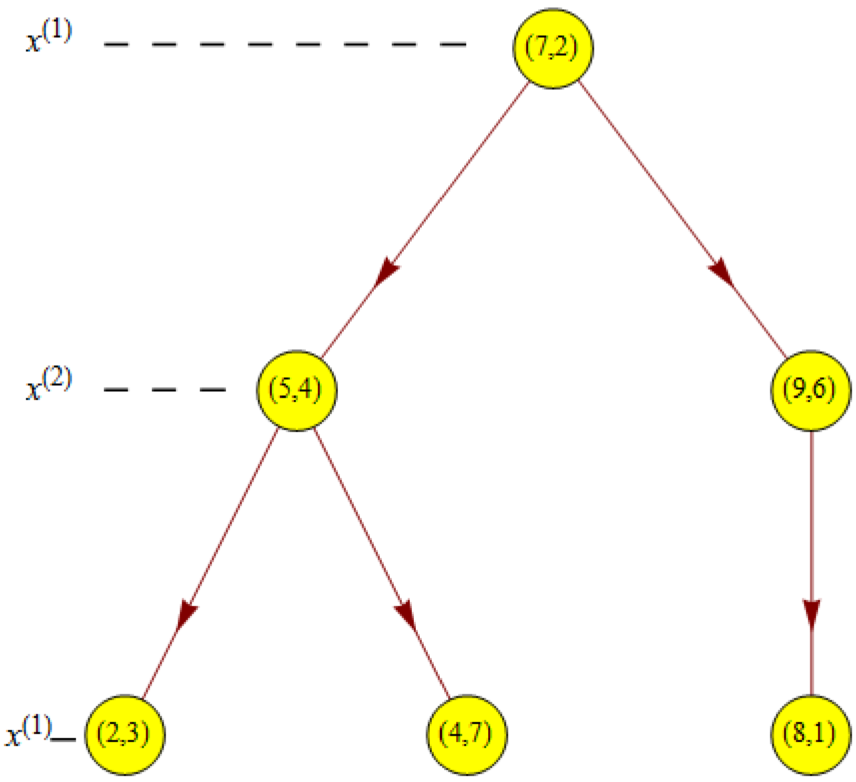
KD树

Algorithm $\textsc{BuildKdTree}(P, depth)$

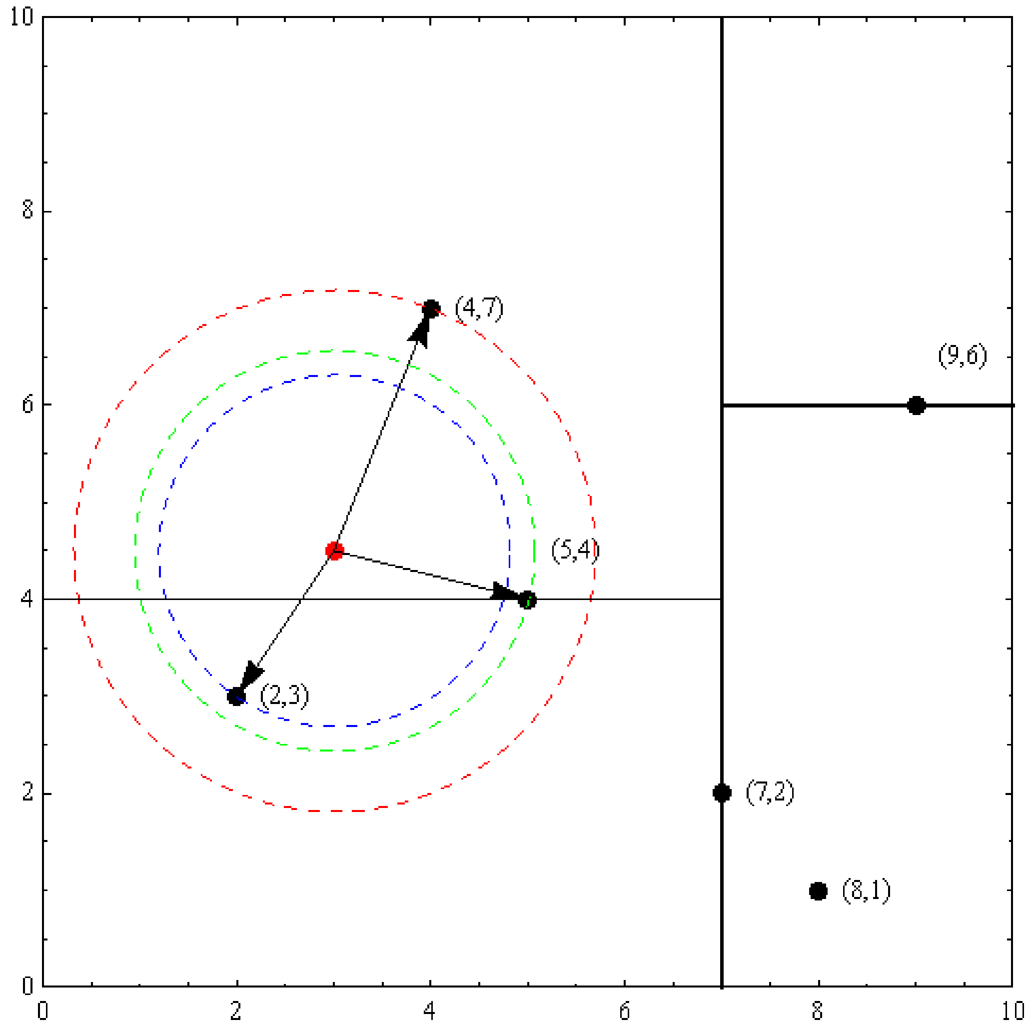
1. **if** P contains only one point
2. **then return** a leaf storing this point
3. **else if** $depth$ is even
4. **then** Split P with a vertical line ℓ through the median x-coordinate into P_1 (left of or on ℓ) and P_2 (right of ℓ)
5. **else** Split P with a horizontal line ℓ through the median y-coordinate into P_1 (below or on ℓ) and P_2 (above ℓ)
6. $\nu_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, depth+1)$
7. $\nu_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, depth+1)$
8. Create a node ν storing ℓ, make ν_{left} the left child of ν, and make ν_{right} the right child of ν.
9. **return** ν

KD树搜索





KD树搜索



KD树搜索

以先前构建好的kd树为例，查找目标点（3,4.5）的最近邻点。同样先进行二叉查找，先从（7,2）查找到（5,4）节点，在进行查找时是由y = 4为分割超平面的，由于查找点为y值为4.5，因此进入右子空间查找到（4,7），形成搜索路径：（7,2）→（5,4）→（4,7），取（4,7）为当前最近邻点。以目标查找点为圆心，目标查找点到当前最近点的距离2.69为半径确定一个红色的圆。然后回溯到（5,4），计算其与查找点之间的距离为2.06，则该结点比当前最近点距目标点更近，以(5,4)为当前最近点。用同样的方法再次确定一个绿色的圆，可见该圆和y = 4超平面相交，所以需要进入（5,4）结点的另一个子空间进行查找。（2,3）结点与目标点距离为1.8，比当前最近点要更近，所以最近邻点更新为（2，3），最近距离更新为1.8，同样可以确定一个蓝色的圆。接着根据规则回退到根结点(7,2)，蓝色圆与x=7的超平面不相交，因此不用进入（7,2）的右子空间进行查找。至此，搜索路径回溯完，返回最近邻点（2,3），最近距离1.8。

KD树

Algorithm SEARCHKDTREE(ν, R)

Input. The root of (a subtree of) a kd-tree, and a range R.

Output. All points at leaves below ν that lie in the range.

```
if ν is a leaf
    then Report the point stored at ν if it lies in R.
    else if region(lc(ν)) is fully contained in R
            then REPORTSUBTREE(lc(ν))
            else if region(lc(ν)) intersects R
                    then SEARCHKDTREE(lc(ν), R)
         if region(rc(ν)) is fully contained in R
            then REPORTSUBTREE(rc(ν))
            else if region(rc(ν)) intersects R
                    then SEARCHKDTREE(rc(ν), R)
```