机器学习

苏州大学计算机科学与技术学院
自然语言处理实验室
主讲：周夏冰
邮箱：zhouxiabing@suda.edu.cn

CONTENTS

01

K近邻

K近邻（K-nearest neighbor, k-NN）

- **1968年，Cover 和 Hart 提出了最初的近邻法**
- **基本思想：根据k个最近邻的训练实例的类别，通过多数表决等方式进行预测**

k-NN

- 主要思想：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以该实例最邻近的k个实例，这k个实例多数属于某个类，就把该输入实例分为这个类。

训练数据集$T=\{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$

$x_i \in \mathcal{X} \subseteq R^d, y_i \in \mathcal{Y}=\{c_1,\cdots,c_K\}$

$x_{new}=(x_{new}^1,\cdots,x_{new}^d)$

(x_1,y_1)

(x_2,y_2)

…

(x_k,y_k)

y_{new}

注：这里角标1不代表数据集中第一个样本，单纯从k个实例角度进行重新编号而已

k-NN-算法描述

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*:

$$y = \arg\max_{c_j} \sum_{x_i \in N_k(x)} I(y_i = c_j)$$

其中*i*=1,2,…,*N*；*j*=1,2,..,*K*；

*I*为指示函数，即当$y_i = c_j$时*I*=1，否则*I*=0

- *K*=1为*K*近邻的特殊情况，称为最近邻算法

k-NN

- 特点
  - 没有训练过程 → 懒惰学习 Lazy Learning
  - 关键计算：距离度量

k-NN—距离度量

- 特征空间中两个实例点的距离是两个实例点**相似程度的反映**
- 近邻模型的特征空间一般是**n**维实数向量空间R^n，能使用的距离是欧氏距离，但也可以是其他距离，如更一般的L_p距离或**Minkowski**距离
  - 设特征空间$\mathcal{X}$是**d**维实数向量空间R^d，$x_i, x_j \in \mathcal{X}$，$x_i = \left(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(d)}\right)^T$，$x_j = \left(x_j^{(1)}, x_j^{(2)}, \ldots, x_j^{(d)}\right)^T$，$x_i$, x_j的L_P距离定义为：

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

k-NN—距离度量

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

- 这里$p \geq 1$. 当p =2时，称为欧氏距离，即：

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

- 当p =1时，称为曼哈顿距离，即：

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

- 当$p = \infty$时，它是各个坐标距离的最大值，称为切比雪夫距离，即：

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

k-NN—距离度量

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

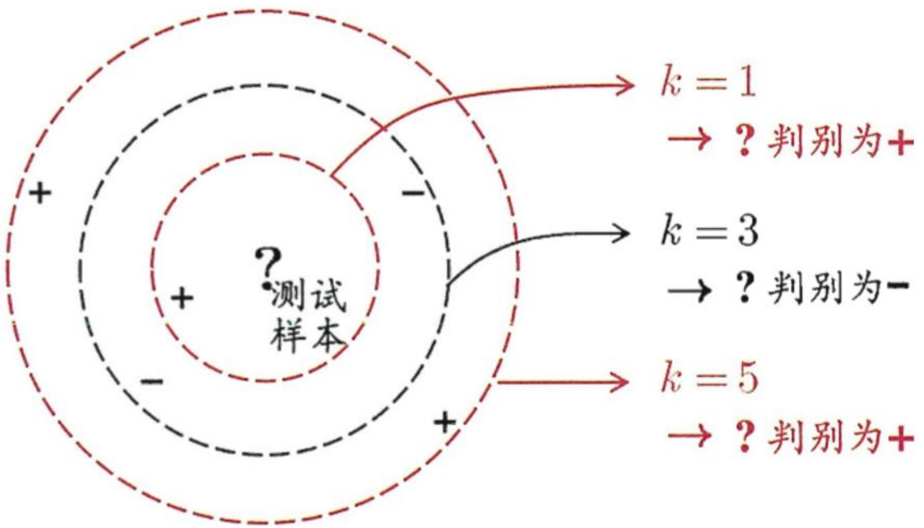
- **已知二维空间的三个点$x_1 = (1,1)^T$，$x_2 = (5,1)^T$，$x_3 = (4,4)^T$，试求在p取不同值时，L_p距离下x_1的最近邻点**
- **解：**

$$L_p(x_1, x_2) = \left(\left|x_1^1 - x_2^1\right|^p + 0\right)^{\frac{1}{p}}$$

  - 因为x_1和x_2只有第一维上值不同，所以**p**为任何值时，$L_p(x_1, x_2) = 4$
  - 而$L_1(x_1, x_3) = 6$，$L_2(x_1, x_3) = 4.24$，$L_3(x_1, x_3) = 3.78$，$L_4(x_1, x_3) = 3.57$，$f(p) = L_p(x_1, x_3)$单调递减
- 于是得到：**p**等于1或2时，x_2是x_1的最近邻点；**p**大于等于3时，x_3是x_1的最近邻点

K值的选择

- ***k*值的选择会对*k*近邻法的结果产生重大影响**

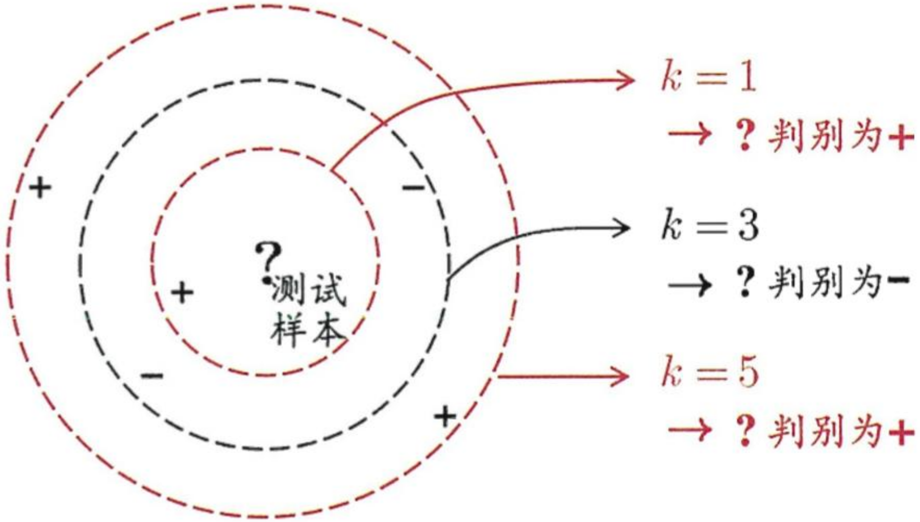


如果选择较小的*k*值，就相当于用较小的邻域中的训练实例进行预测，“学习”的近似误差会减小，只有与输入实例较近的（相似的）训练实例才会对预测结果起作用

缺点是“学习”的估计误差会增大，预测结果会对近邻的实例点非常敏感。如果邻近的实例点恰巧是噪声，预测就会出错；换句话说，*k*值的减小就意味着整体模型变得复杂，容易发生过拟合

K值的选择

- ***k*值的选择会对*k*近邻法的结果产生重大影响**



如果*k*选择较大的值，就相当于用较大邻域中的训练实例进行预测

优点是可以减少学习的估计误差
缺点是学习的近似误差会增大，这时与输入实例较远的（不相似的）训练实例也会对预测起作用，使预测发生错误，*k*值的增大就意味着整体的模型变得简单

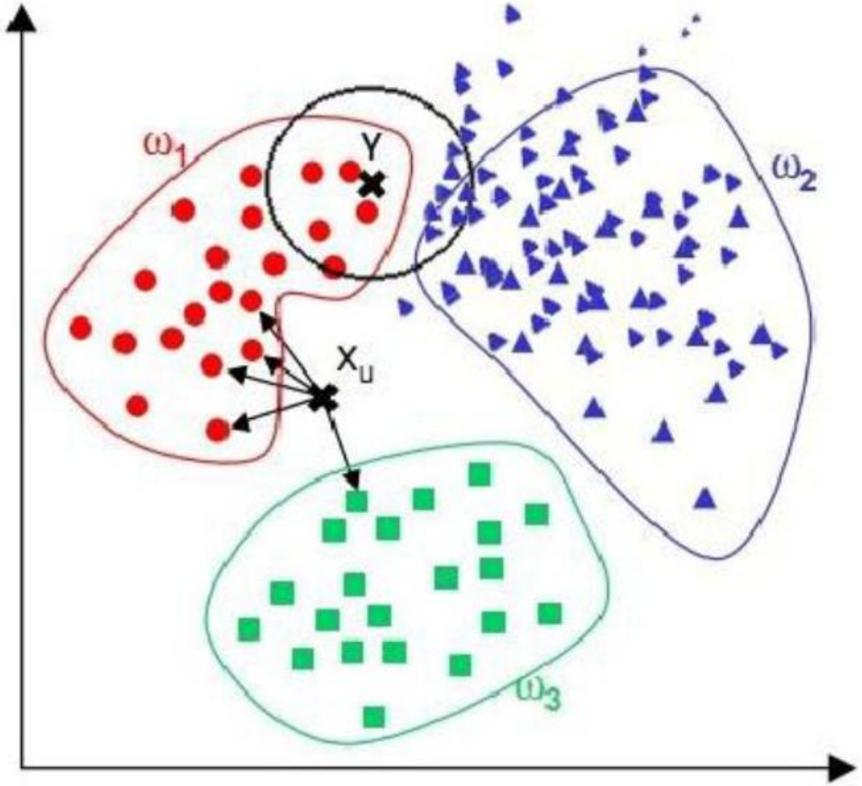
考虑：如果$k=N$，那么无论输入实例是什么，都将简单地预测它属于在训练实例中最多的类，这时模型过于简单，完全忽略训练实例中的大量有用信息，是不可取的

在应用中，k值一般取一个比较小的数值，通常采用交叉验证法来选取最优的k值

K-NN

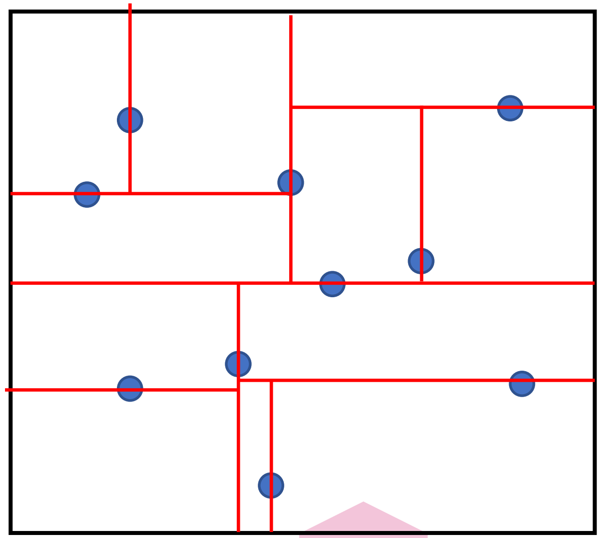
- 样本不平衡敏感

增加样本权重权重



kd 树

- 实现k近邻法时，主要考虑的问题是如何**对训练数据进行快速k近邻搜索**
- kd树(**K-dimension** tree)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
  - 二叉树



Kd树-构造-算法描述

- **例1：给定一个二维空间的数据集：**

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

构造一个平衡kd树

构造根节点

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，将根节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。坐标$x^{(1)}$小于中位数的作为左节点，大于的做右节点，等于的放在根节点

左子区域

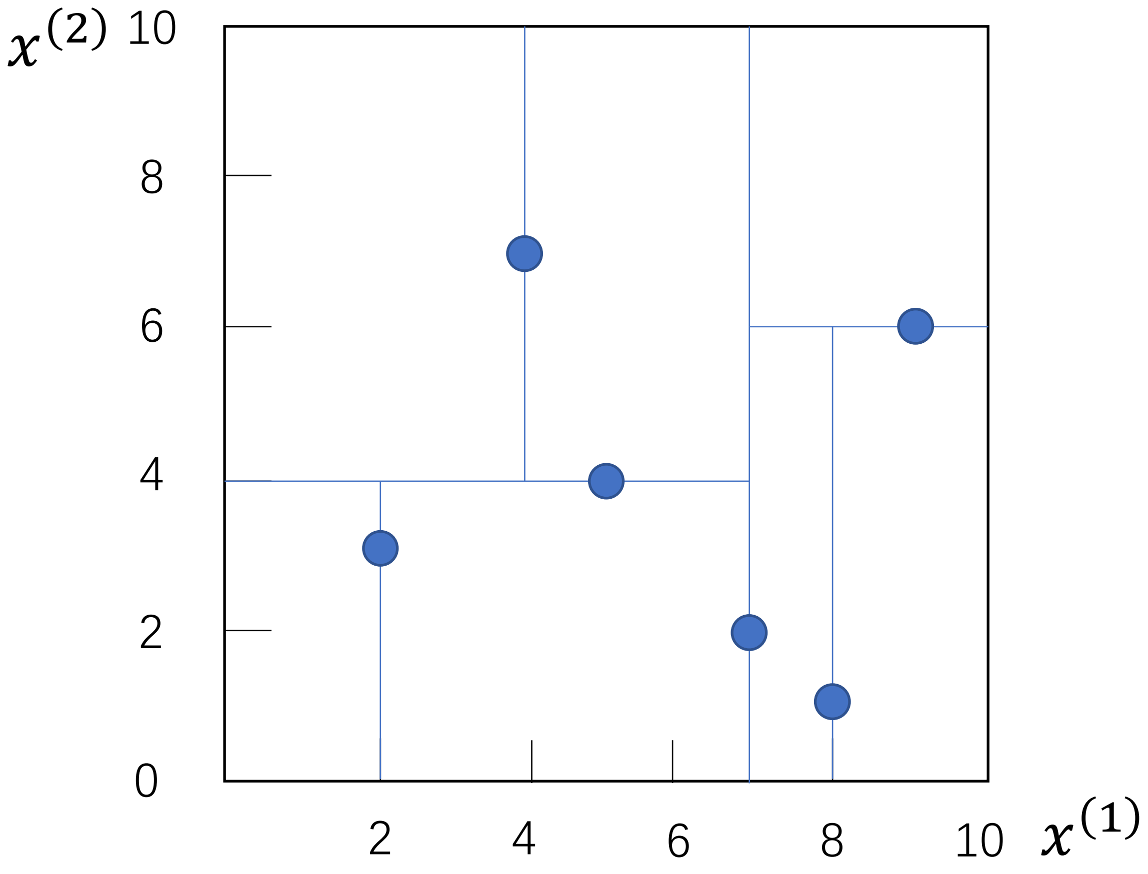
右子区域

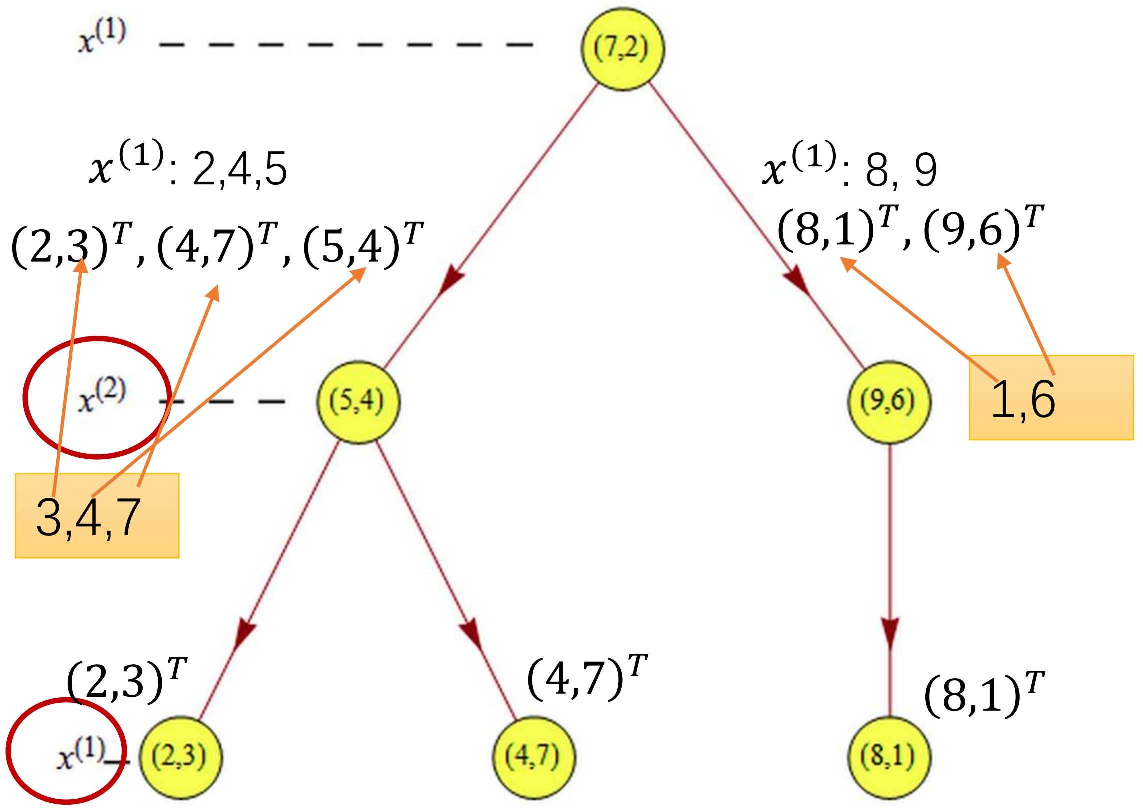
$x^{(1)}$: 2,5,9,4,8,7 → 2,4,5,7,8,9 → 2,4,5,7,8,9 → 2,4,5,7,8,9

排序

中位数：列表长度/2，向下取整

Kd树-构造-算法描述





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

课堂作业

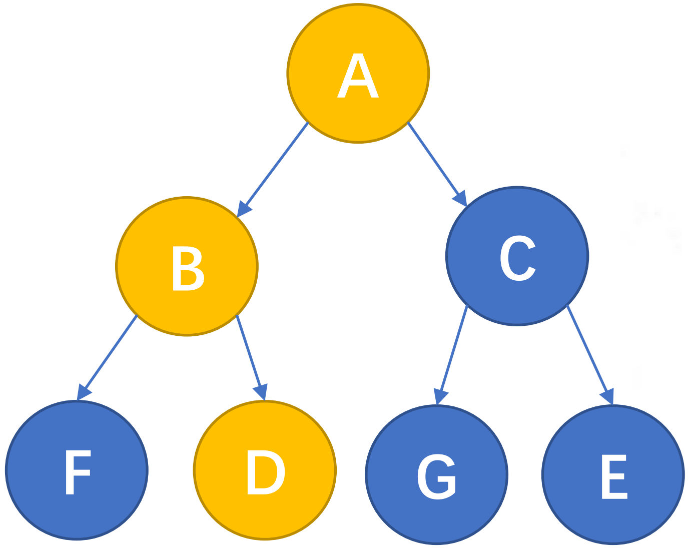
- 进阶练习：给定一个三维空间的数据集：T={(1,9,6), (4,5,7), (8,1,9), (3,7,2), (7,6,4), (5,2,5), (6,3,1)}构造一颗平衡kd树。
  - 给出计算过程，画出KD树

搜索kd树

- 输入:已构造的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z 的叶结点:从根结点出发，递归地向下访问kd 树。若目标点z 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"。

搜索kd树

- <A,B,D>
- 最近点D



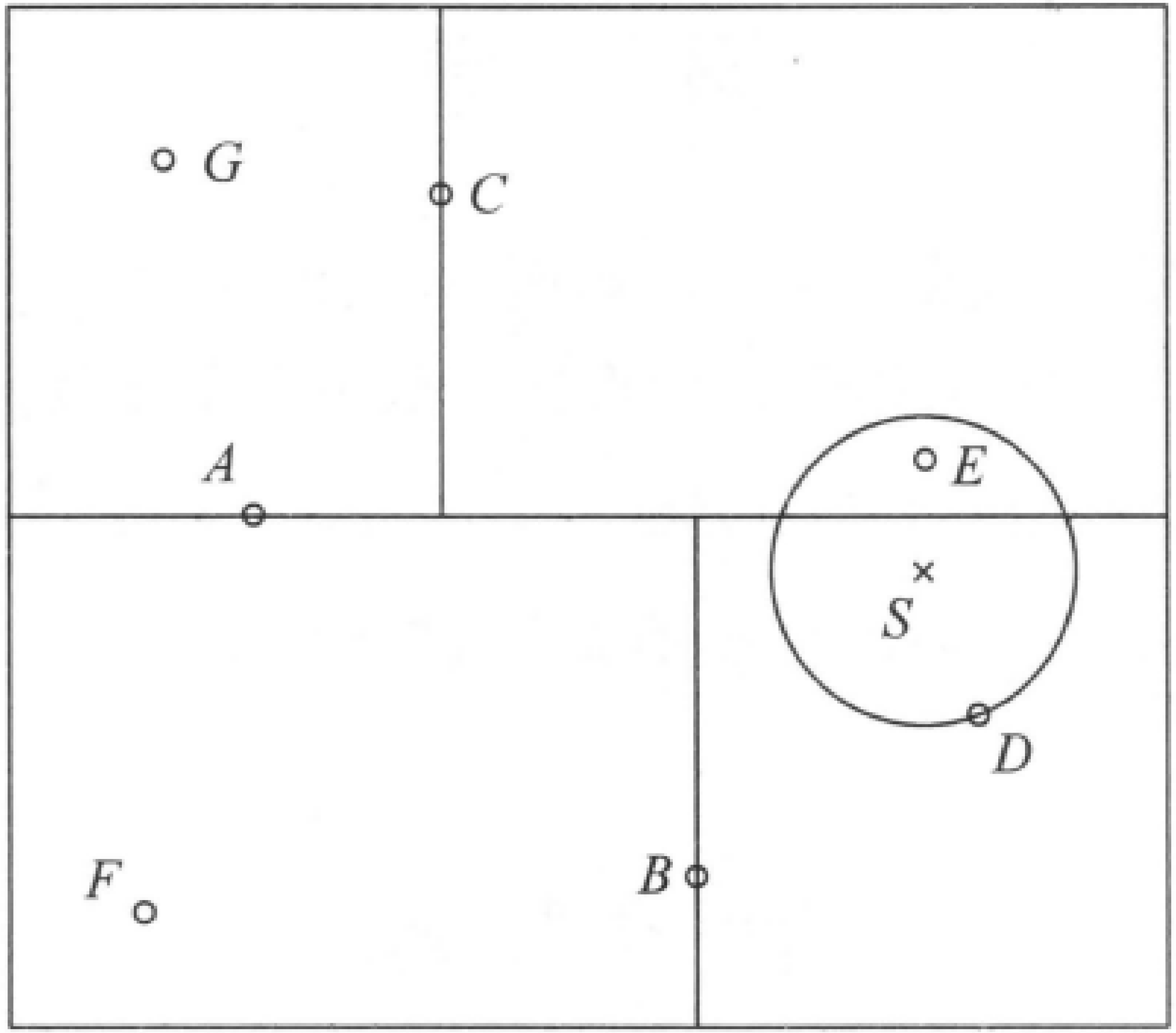
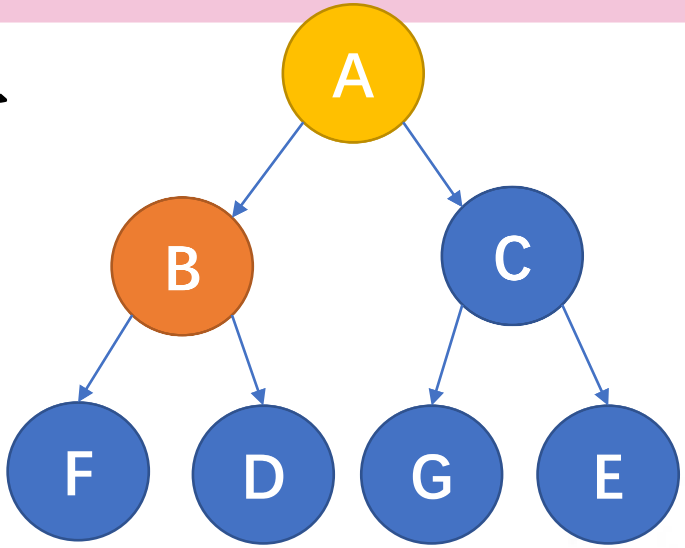


图 3.5　通过 *kd* 树搜索最近邻

搜索kd树

- 输入:已构造的kd 树， 目标点X;
- 输出： X 的最近邻。
- (3) 递归地向上回退，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) 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(2)的最近邻点。

搜索kd树



- <A,B,D>
- 最近点D
- 返回父节点B，考虑B对应的另一子节点F。不相交

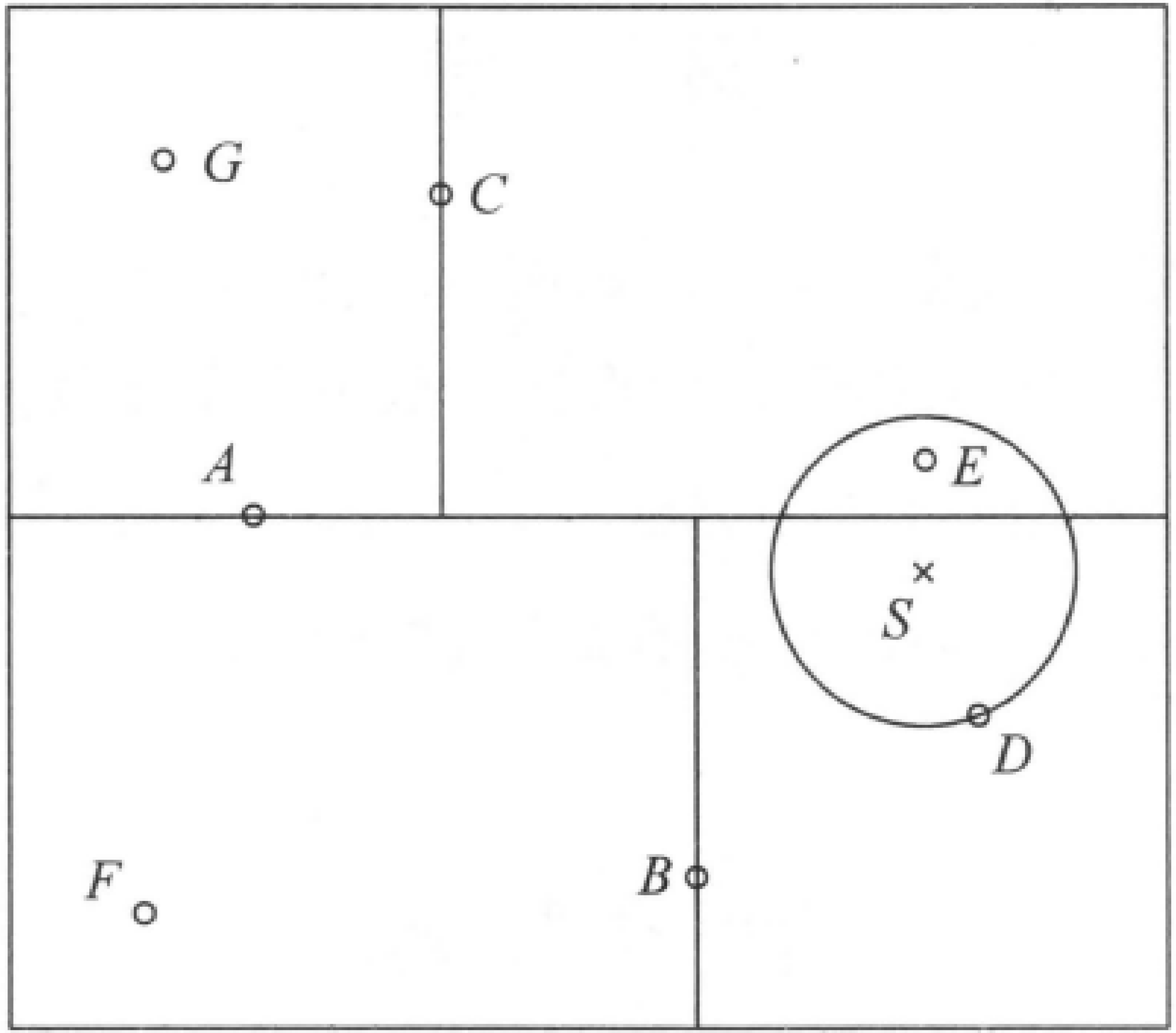
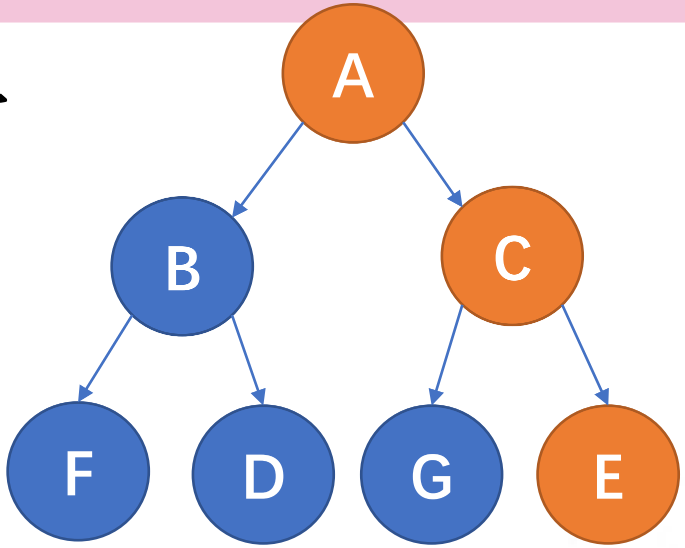


图 3.5 通过 kd 树搜索最近邻

搜索kd树



- <A,B,D>
- 最近点D
- 返回父节点B，考虑B对应的另一子节点F。不相交
- 返回A，考虑A另一区域C，C的区域与之相交，存在E点最近
- 更新最近点E

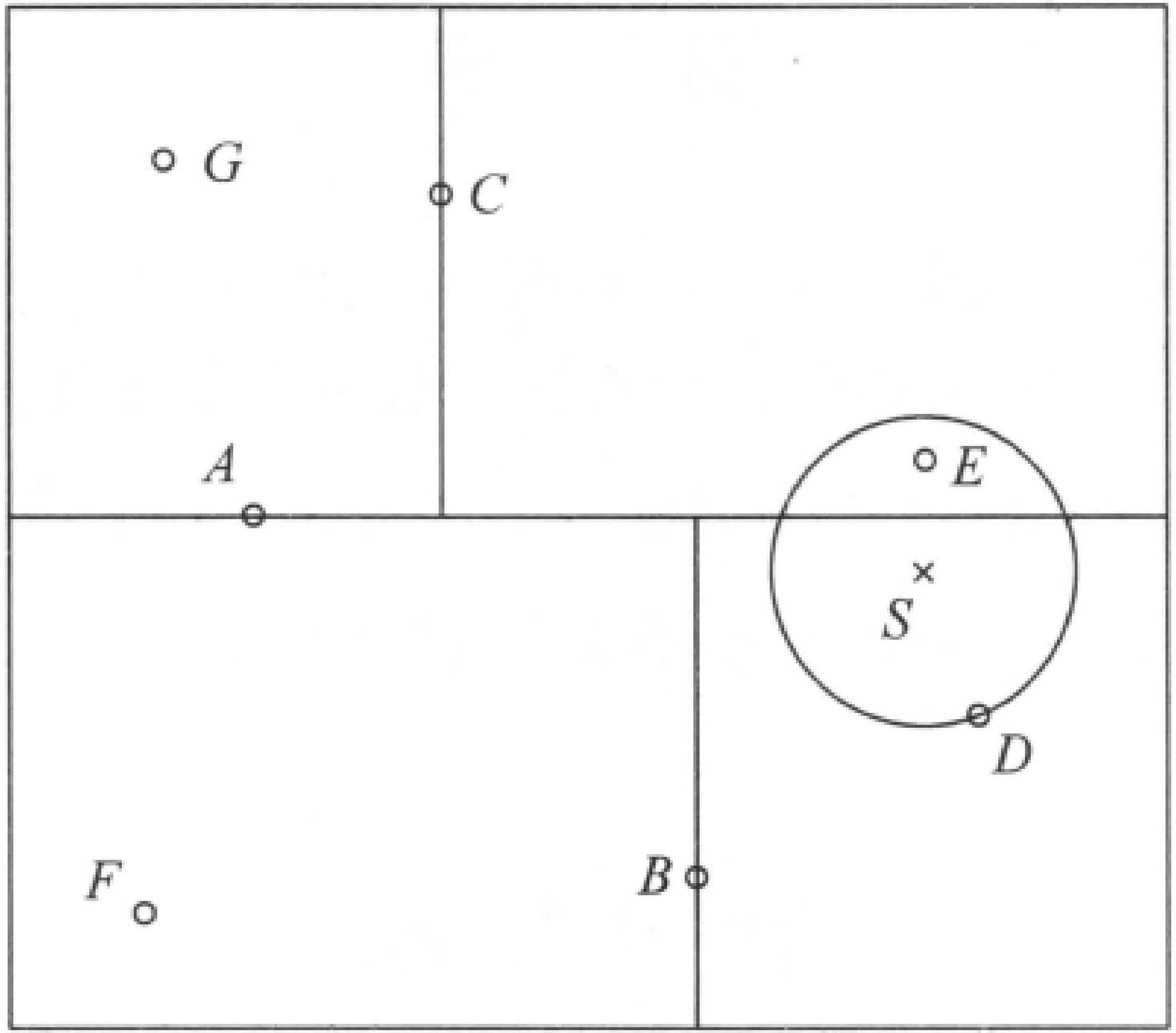
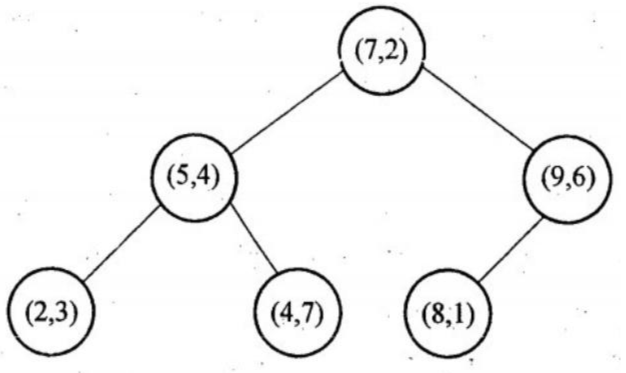
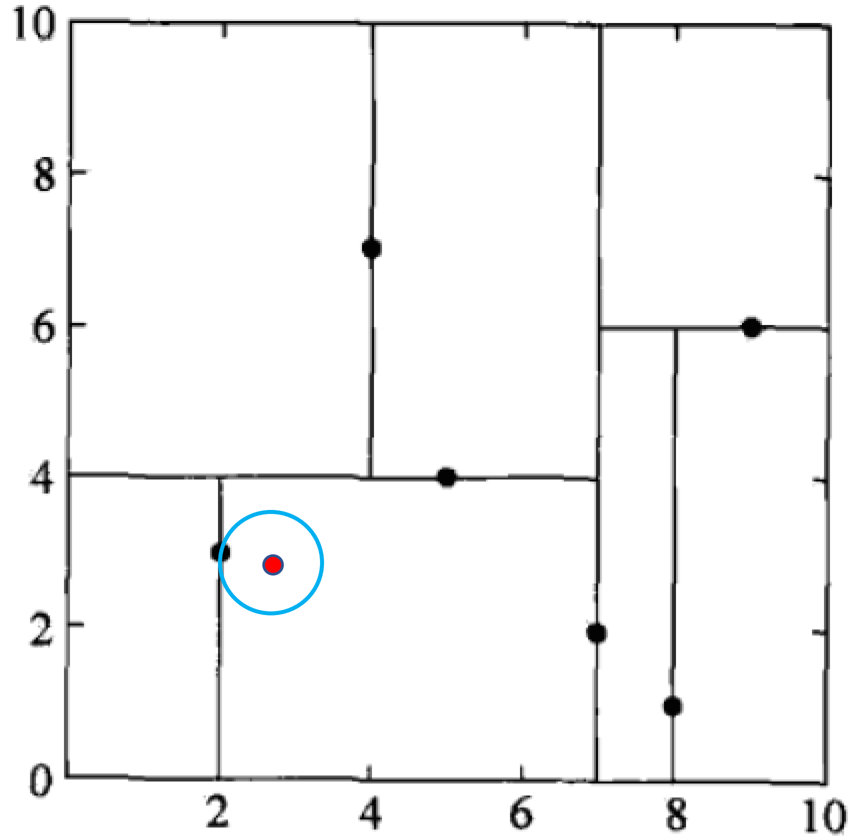


图 3.5 通过 kd 树搜索最近邻

kd 树

- 例 给定一个二维空间数据集：T={(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)}, 构造一个平衡kd树





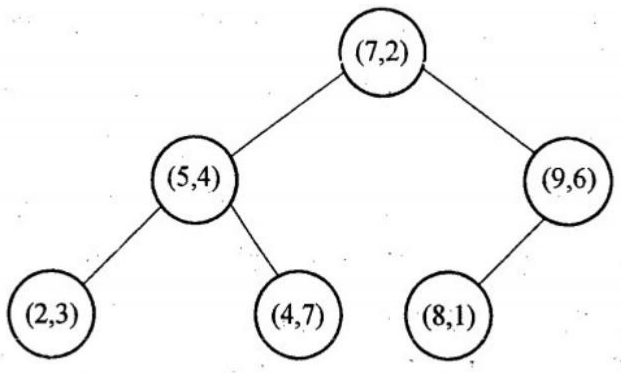
- (2.1, 3.1)
- 最近邻（2，3）
- trackList = [(7, 2), (5, 4), (2, 3)]
- trackList = [(7, 2), (5, 4)]
- trackList = [(7, 2)]
- trackList =[]

Kd树运行实例

- 例 给定一个二维空间数据集：T={(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)}, 构造一个平衡kd树，查找（3，4.5）最近邻

最近邻叶子节点value = (4,7)

路径 trackList = [(7, 2), (5, 4), (4, 7)]



Kd树运行实例

- 例 给定一个二维空间数据集：T={(2,3),(5,4),(9,6),(4,7),(8,1),(7,2)}, 构造一个平衡kd树，查找（3，4.5）最近邻

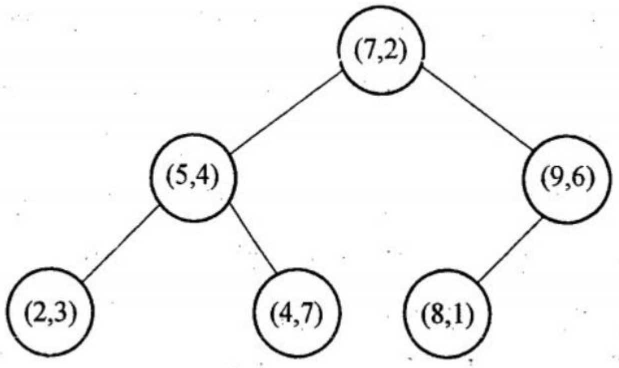
trackList = [(7, 2), (5, 4), (4, 7)]

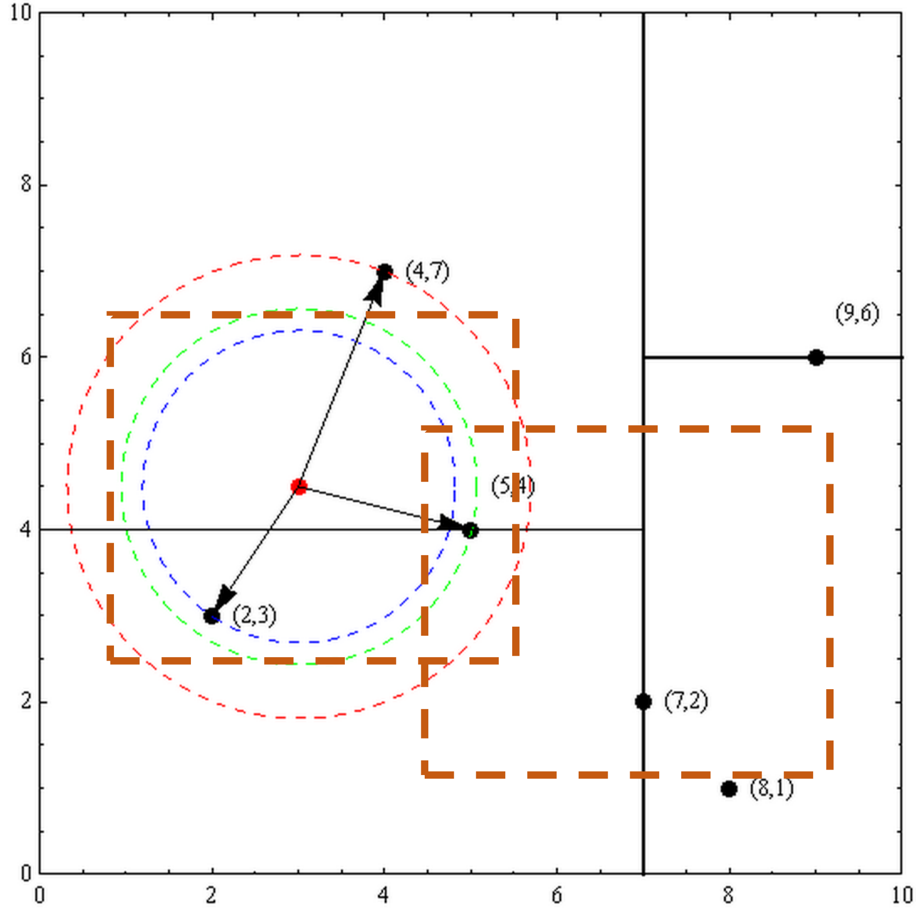
trackList = [(7, 2), (5, 4)]

trackList = [(7, 2), (2, 3)]

trackList = [(7, 2)]

trackList = []





```python
def create(self, dataSet, depth=0):
    if len(dataSet) > 0:
        m, n = np.shape(dataSet)
        self.n = n - 1
        axis = depth % self.n
        mid = int(m / 2)
        dataSetcopy = sorted(dataSet, key=lambda x: x[axis])
        node = Node(dataSetcopy[mid], depth)
        if depth == 0:
            self.KdTree = node
        node.lchild = self.create(dataSetcopy[:mid], depth+1)
        node.rchild = self.create(dataSetcopy[mid+1:], depth+1)
        return node
    return None
```

```python
# DFS algorithm
def dfs(kdTree, target, tracklist=[]):
    tracklistCopy = tracklist[:]
    if not kdTree:
        return None, tracklistCopy
    elif not kdTree['Left']:
        tracklistCopy.append(kdTree['Value'])
        return kdTree['Value'], tracklistCopy
    elif kdTree['Left']:
        pointValue = kdTree['Value']
        feature = kdTree['feature']
        tracklistCopy.append(pointValue)
        # return kdTree['Value'], tracklistCopy

        if target[feature] <= pointValue[feature]:
            return dfs(kdTree['Left'], target, tracklistCopy)
        elif target[feature] > pointValue[feature]:
            return dfs(kdTree['Right'], target, tracklistCopy)
```

```python
def kdTreeSearch(tracklist, target, usedPoint=[], minDistance=float('inf'), minDistancePoint=None):
    tracklistCopy = tracklist[:]
    usedPointCopy = usedPoint[:]

    if not minDistancePoint:
        minDistancePoint = tracklistCopy[-1]

    if len(tracklistCopy) == 1:
        return minDistancePoint
    else:
        point = findPoint(kdTree, tracklist[-1])

        if calDistance(point['Value'], target) < minDistance:
            minDistance = calDistance(point['Value'], target)
            minDistancePoint = point['Value']

        fatherPoint = findPoint(kdTree, tracklistCopy[-2])
        fatherPointval = fatherPoint['Value']
        fatherPointfea = fatherPoint['feature']

        if calDistance(fatherPoint['Value'], target) < minDistance:
            minDistance = calDistance(fatherPoint['Value'], target)
            minDistancePoint = fatherPoint['Value']
```

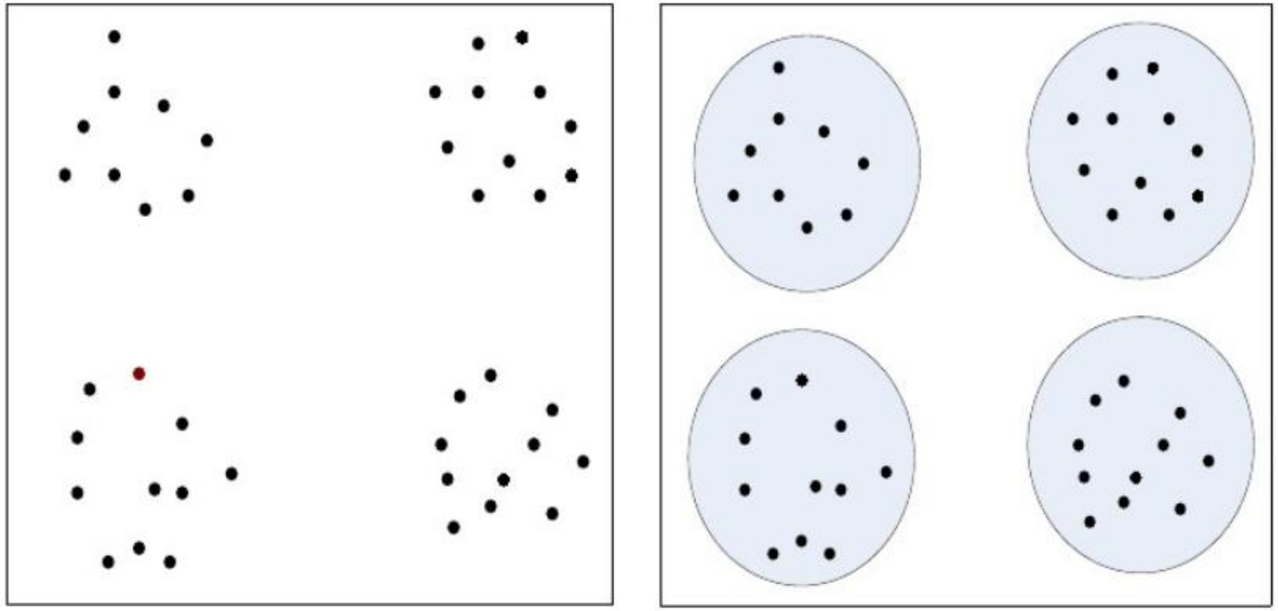
```
# check another side
    if point == fatherPoint['Left']:
        anotherPoint = fatherPoint['Right']
    elif point == fatherPoint['Right']:
        anotherPoint = fatherPoint['Left']

    if (anotherPoint == None or anotherPoint['Value'] in usedPointCopy or
            abs(fatherPointval[fatherPointfea] - target[fatherPointfea]) > minDistance):  # non-intersect
        usedPoint = tracklistCopy.pop()
        usedPointCopy.append(usedPoint)
return kdTreeSearch(tracklistCopy, target, usedPointCopy, minDistance, minDistancePoint)
    else:  # intersect
        usedPoint = tracklistCopy.pop()
        usedPointCopy.append(usedPoint)
subvalue, subtrackList = dfs(
            anotherPoint, target)
        tracklistCopy.extend(subtrackList)
        print('tracklistCopy'+str(tracklistCopy))
        return kdTreeSearch(tracklistCopy, target, usedPointCopy, minDistance, minDistancePoint)
```

02

聚类方法

聚类

- 聚类（Clustering）：对大量未标注的数据集，按数据的内在相似性将数据集进行划分，形成多个簇（cluster)，使得内部数据相似度尽可能大而类别间的数据相似度尽可能小



聚类相关概念

- 相似度/距离
  - 闵可夫斯基距离
  - 马哈拉诺比斯距离 $d_{ij}=\left[(x_i-x_j)^T S^{-1}(x_i-x_j)\right]^{\frac{1}{2}}$，S是协方差矩阵
  - 相关系数 $$r_{ij}=\frac{\sum_{k=1}^{m}(x_{ki}-\bar{x}_i)(x_{kj}-\bar{x}_j)}{\left[\sum_{k=1}^{m}(x_{ki}-\bar{x}_i)^2\sum_{k=1}^{m}(x_{kj}-\bar{x}_j)^2\right]^{\frac{1}{2}}}$$
  - 夹角余弦

$$s_{ij}=\frac{\sum_{k=1}^{m}x_{ki}x_{kj}}{\left[\sum_{k=1}^{m}x_{ki}^2\sum_{k=1}^{m}x_{kj}^2\right]^{\frac{1}{2}}}$$

聚类相关概念

- 类/簇
  - 类之间没有交集——硬聚类
  - 类之间存在交集——软聚类
- 类均值(中心） $\bar{x}_G = \frac{1}{n_G}\sum_{i=1}^{n_G} x_i$
- 类直径：任意两个样本之间的最大距离 $D_G = \max_{x_i,x_j \in G} d_{ij}$
- 类散度: $A_G = \sum_{i=1}^{n_G}(x - \bar{x}_G)(x - \bar{x}_G)^T$
- 协方差矩阵: $S_G = \frac{1}{m-1}\sum_{i=1}^{n_G}(x - \bar{x}_G)(x - \bar{x}_G)^T$

聚类相关概念

- 类与类之间的距离
  - 最短距离: $D_{pq} = min\{d_{ij}|x_i \in G_p, x_j \in G_q\}$
  - 最长距离: $D_{pq} = max\{d_{ij}|x_i \in G_p, x_j \in G_q\}$
  - 中心距离: $D_{pq} = d_{\bar{x}_p\bar{x}_q}$
  - 平均聚类: $D_{pq} = \frac{1}{n_p n_q}\sum_{x_i \in G_p}\sum_{x_j \in G_q} d_{ij}$

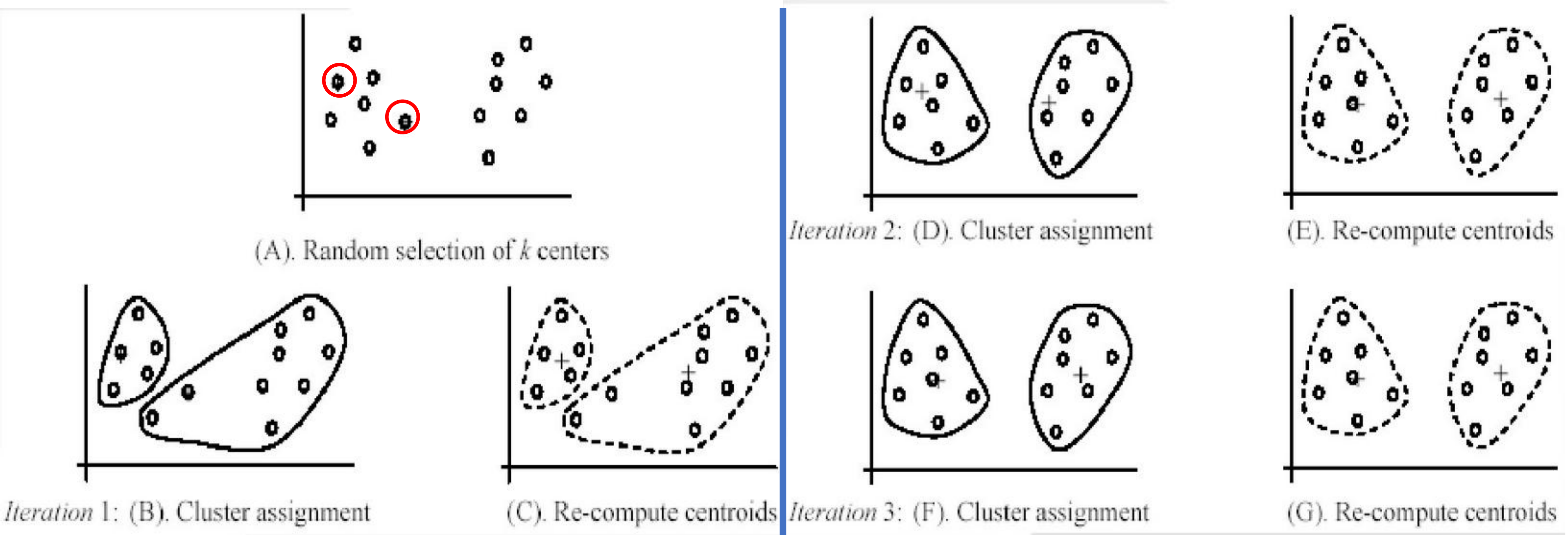
聚类方法——k-means

- **k：指定定义多少个簇**
- **基本思想：通过迭代把数据集划分为不同的类别，使得评价聚类性能的准则函数达到最优，使得每个聚类类内紧凑，类间独立**
- **最优化问题：**
  - $C^* = \underset{C}{\operatorname{argmin}} \sum_{l=1}^{k} \sum_{C(i)=l} \|\boldsymbol{x}_i - \overline{\boldsymbol{x}}_l\|^2$
  - 组合优化问题，NP难

聚类方法——k-means

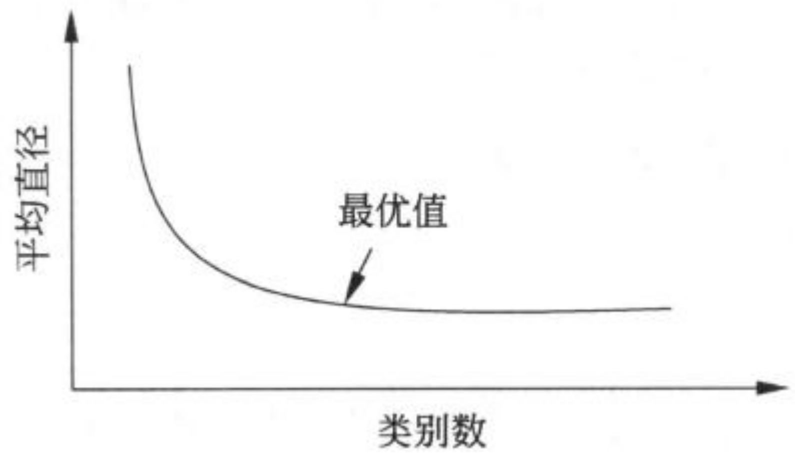
- k：指定定义多少个簇
- 基本思想：通过迭代把数据集划分为不同的类别，使得评价聚类性能的准则函数达到最优，使得每个聚类类内紧凑，类间独立
- 步骤
  - 初始随机选择k个点作为簇的中心
  - 根据某个距离函数反复地把数据分入K个聚类中
  - 距离函数：欧式距离，曼哈顿距离等

k-means


(A). Random selection of k centers
Iteration 1: (B). Cluster assignment
(C). Re-compute centroids
Iteration 2: (D). Cluster assignment
(E). Re-compute centroids
Iteration 3: (F). Cluster assignment
(G). Re-compute centroids

k-means

- **初始点的选择**
  - 随机
  - 根据经验选择
  - 选择距离较大的点
- **k值的选择**
  - 尝试不同的k值，检验各种得到的聚类结果质量，比如用平均直径来衡量



k-means

- 例：学生兴趣划分

学生编号	喜欢吃零食	喜欢看韩剧	喜欢打篮球	喜欢玩游戏	工资
A	8	8	0	0	5000
B	7	8	0	1	5100
C	8	7	0	1	5080
D	8	8	1	0	5030
E	0	0	10	8	5010
F	0	2	9	8	5090
G	1	2	9	9	5020
H	2	1	8	9	5040

- 分组结果

C1	C2
B、C、F	A、D、E、G、H

被工资主导

k-means

- 距离度量

$$\sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

- 解决方案：归一化

$$v' = \frac{v - v_{\min}}{v_{\max} - v_{\min}}$$

学生编号	喜欢吃零食	喜欢看韩剧	喜欢打篮球	喜欢玩游戏	工资
A	1	1	0	0	0
B	0.875	1	0	0.111111	1
C	1	0.875	0	0.111111	0.8
D	1	1	0.1	0	0.3
E	0	0	1	0.888888	0.1
F	0	0.25	0.9	0.888888	0.9
G	0.125	0.25	0.9	1	0.2
H	0.25	0.125	0.8	1	0.4

k-means

- 例：学生兴趣划分

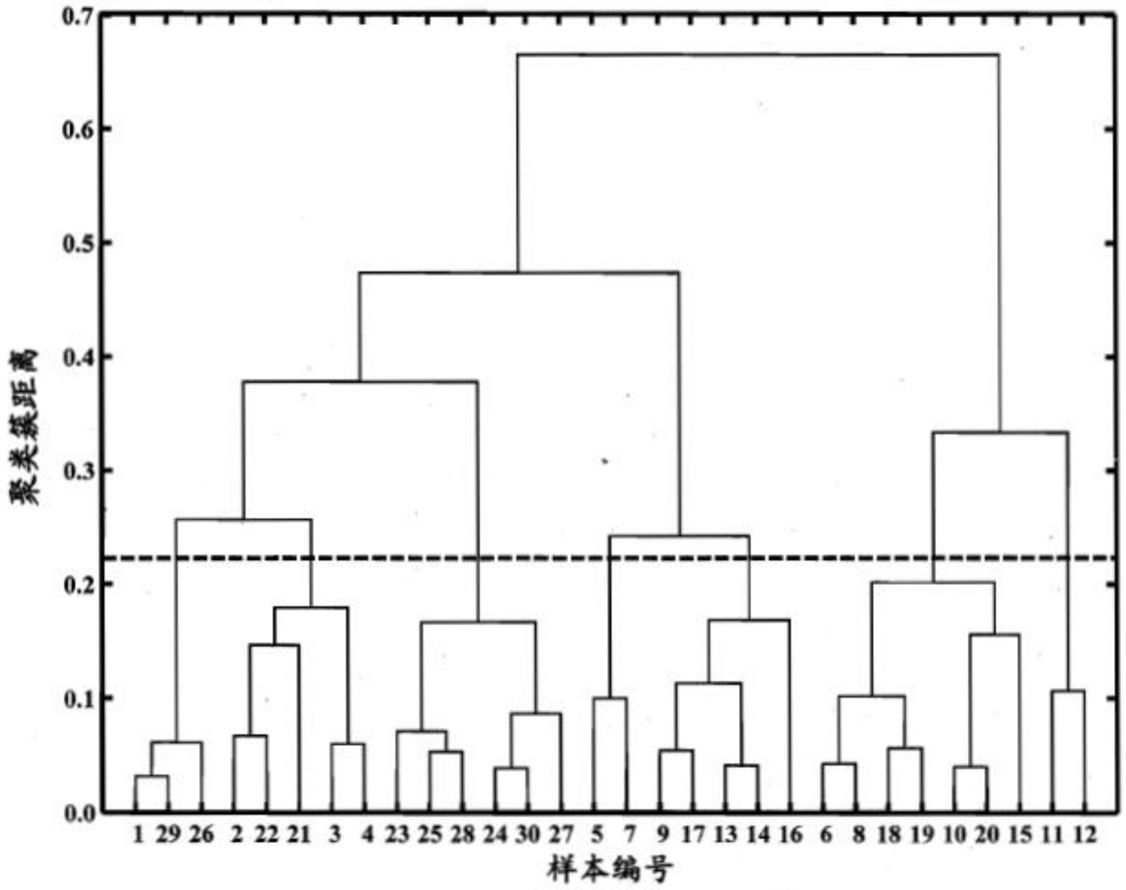
学生编号	喜欢吃零食	喜欢看韩剧	喜欢打篮球	喜欢玩游戏	工资
A	8	8	0	0	5000
B	7	8	0	1	5100
C	8	7	0	1	5080
D	8	8	1	0	5030
E	0	0	10	8	5010
F	0	2	9	8	5090
G	1	2	9	9	5020
H	2	1	8	9	5040

- 分组结果

C1	C2
A、B、C、D	E、F、G、H

聚类方法——层次聚类

- 从不同层次对数据集进行划分，从而形成树形的聚类结构
- 自下而上（聚合）
- 自上而下（分裂）



层次聚类

- **输入：** 包含n个对象的数据库。

  输出： 满足终止条件的若干个簇。

  (1) 将每个对象当成一个初始簇；

  (2) REPEAT

  (3) 计算任意两个簇的距离，并找到最近的两个簇；

  (4) 合并两个簇，生成新的簇的集合；

  (5) UNTIL 终止条件得到满足。

层次聚类

- **例14.1**

$$
D=[d_{ij}]_{5\times 5}=
\begin{array}{c c}
\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} & \\
\begin{bmatrix}
0 & 7 & 2 & 9 & 3 \\
7 & 0 & 5 & 4 & 6 \\
2 & 5 & 0 & 8 & 1 \\
9 & 4 & 8 & 0 & 5 \\
3 & 6 & 1 & 5 & 0
\end{bmatrix} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}
\end{array}
$$

$\cancel{D_{61}=2,}\quad \cancel{D_{62}=5,}\quad \cancel{D_{64}=5}$

$D_{72}=5,\quad D_{74}=5$