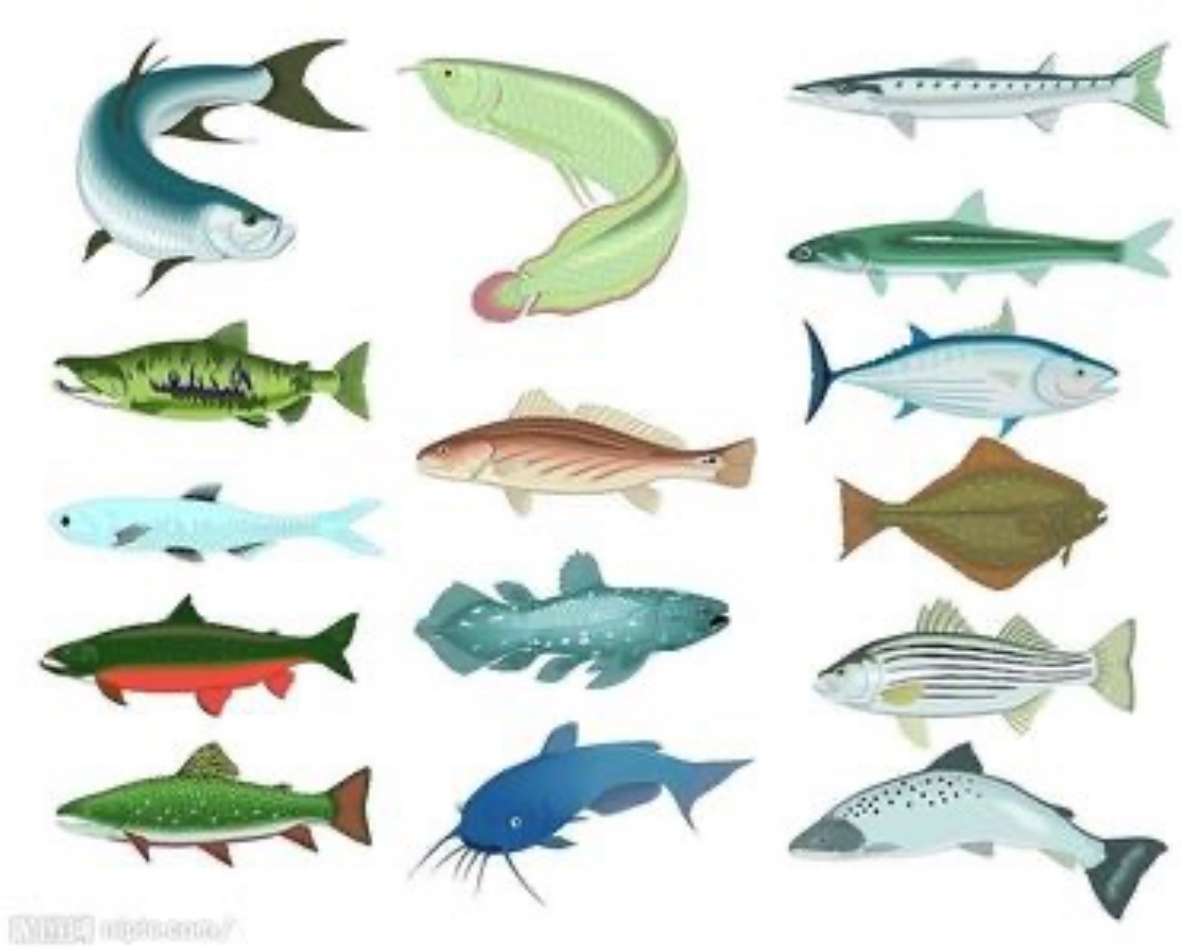
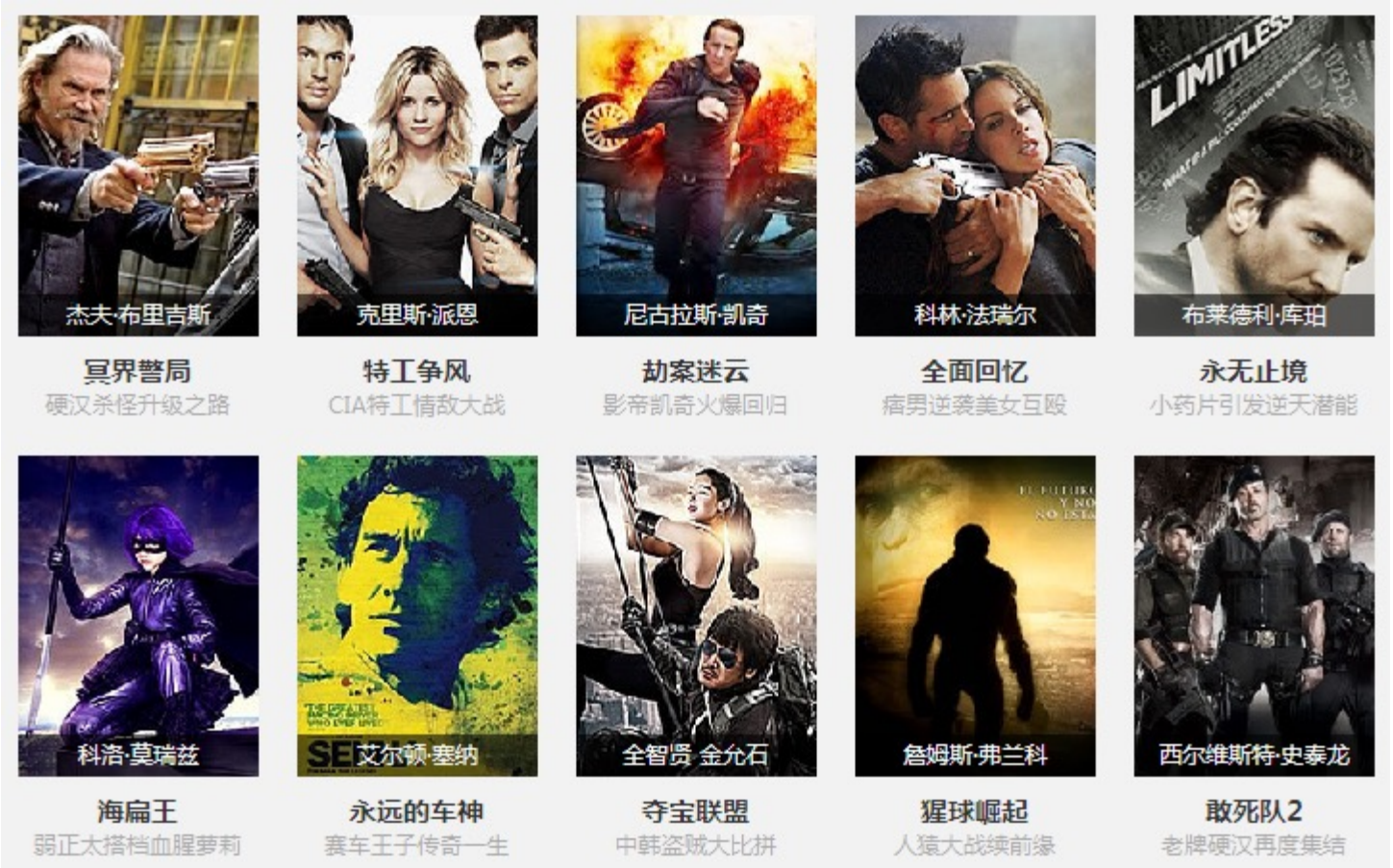
# 第三章
# k-近邻算法

# 分类问题



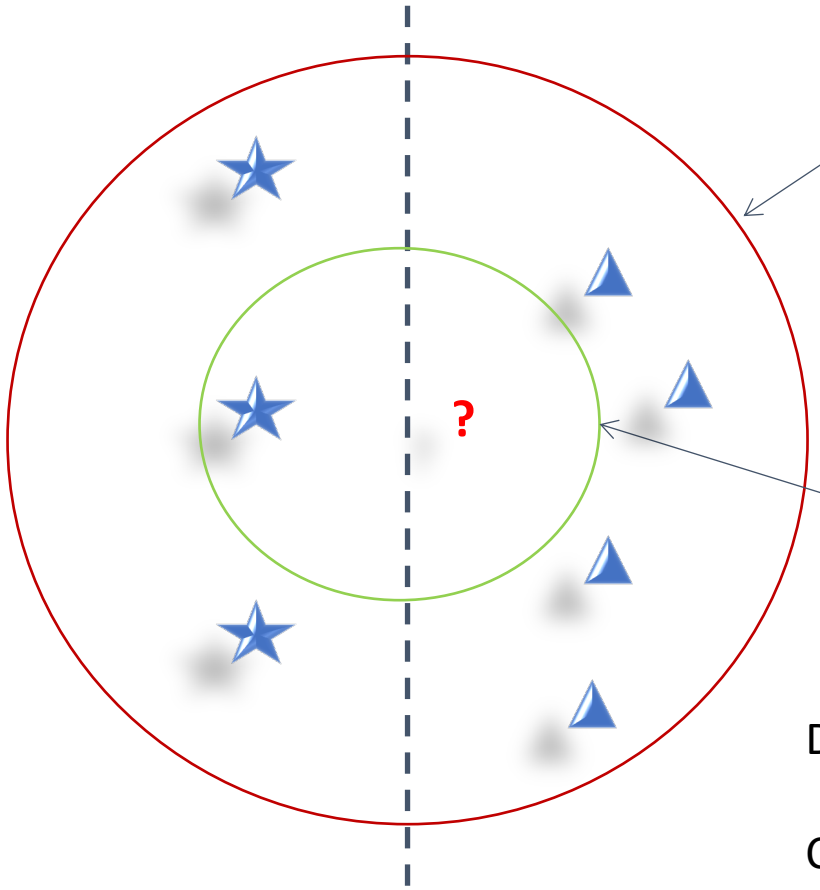
# 分类问题


杰夫·布里吉斯
克里斯·派恩
尼古拉斯·凯奇
科林·法瑞尔
LIMITLESS
布莱德利·库珀
冥界警局
硬汉杀怪升级之路
特工争风
CIA特工情敌大战
劫案迷云
影帝凯奇火爆回归
全面回忆
痞男逆袭美女互殴
永无止境
小药片引发逆天潜能
科洛·莫瑞兹
艾尔顿·塞纳
全智贤 金允石
詹姆斯·弗兰科
西尔维斯特·史泰龙
海扁王
弱正太搭档血腥萝莉
永远的车神
赛车王子传奇一生
夺宝联盟
中韩盗贼大比拼
猩球崛起
人猿大战续前缘
敢死队2
老牌硬汉再度集结

- 爱情片、剧情片、喜剧片、家庭片、伦理片、
  文艺片、音乐片、歌舞片、动漫片、
  西部片、武侠片、古装片、动作片、
  恐怖片、惊悚片、冒险片、犯罪片、悬疑片、
  记录片、战争片、历史片、传记片、体育片、
  科幻片、魔幻片、奇幻片

# Supervised learning

# K-Nearest Neighbors算法原理




K=7 Neighborhood

K=1 Neighborhood

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K个最近邻的数据。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。

# 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：

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

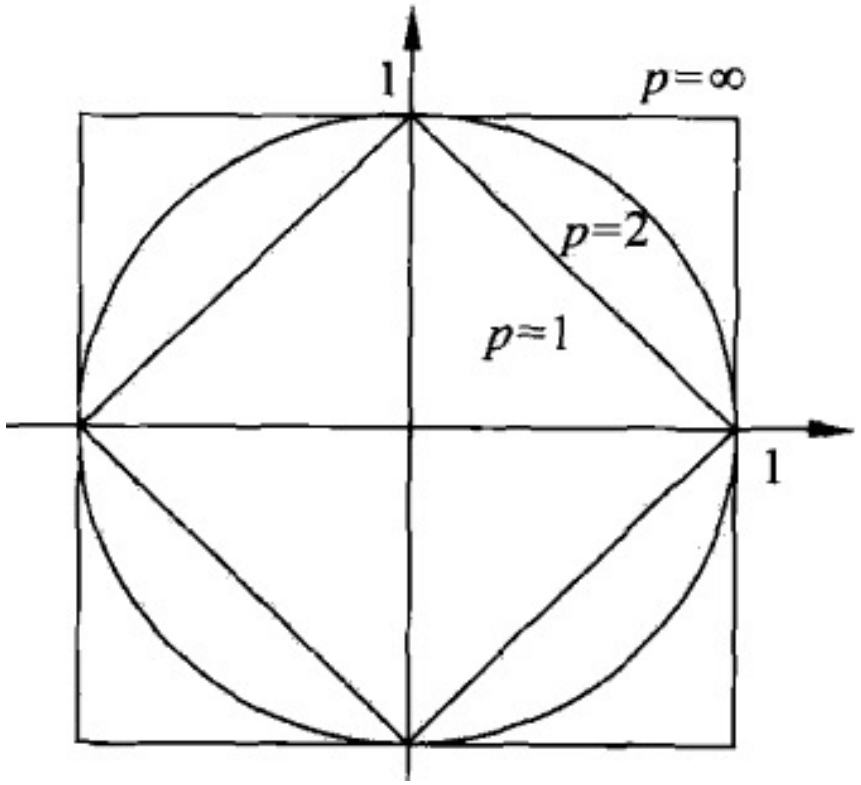
- 曼哈顿距离

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

- 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.

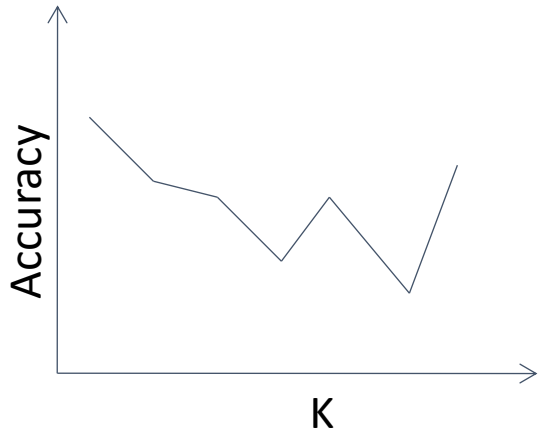
- 如果选择较大的K值，
  - 减少学习的估计误差，但缺点是学习的近似误差会增大.
  - K值的增大 就意味着整体的模型变得简单.

# 分类算法流程

- 对未知类别的数据集中的每个点依次执行以下操作
  - 计算已知类别数据集众多点与当前点之间的距离
  - 按照距离递增次序排序
  - 选取与当前点距离最小的k个点
  - 确定前k个点所在类别的出现频率
  - 返回前k个点出现频率最高的类别作为当前点的预测分类

# KNN面临挑战

- K值确定
  - Non-monotonous impact on accuracy
  - Too Big vs. Too Small
  - Rule of thumbs
- 特征的选择
  - Different features may have different impact …
- 距离函数确定
  - There are many different ways to measure the distance.
  - Euclidean, Manhattan …
- 复杂度
  - Need to calculate the distance between X’ and all training data.
  - In proportion to the size of the training data.



# KD树

- Kd树是一种对K（**和前面的K意义不一样**）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（partition).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。

**算法 3.2（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 $x_i=(x_i^{(1)},x_i^{(2)},\cdots,x_i^{(k)})^{\mathrm{T}}$，$i=1,2,\cdots,N$；

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，$l=j(\mathrm{mod}\ k)+1$，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. ■

# 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 $l = j(\mathrm{mod}\, k) + 1$
- 例： $T = \{(2,3)^{\mathrm{T}}, (5,4)^{\mathrm{T}}, (9,6)^{\mathrm{T}}, (4,7)^{\mathrm{T}}, (8,1)^{\mathrm{T}}, (7,2)^{\mathrm{T}}\}$

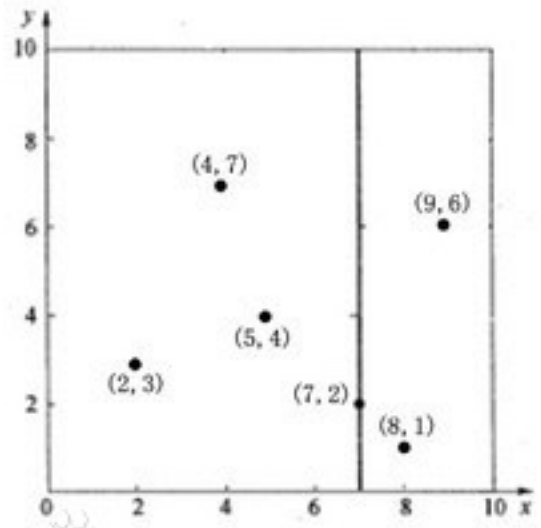



图2 x=7将整个空间分为两部分

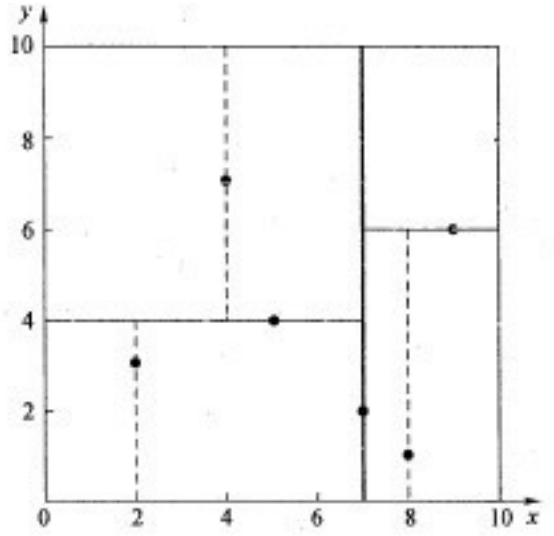
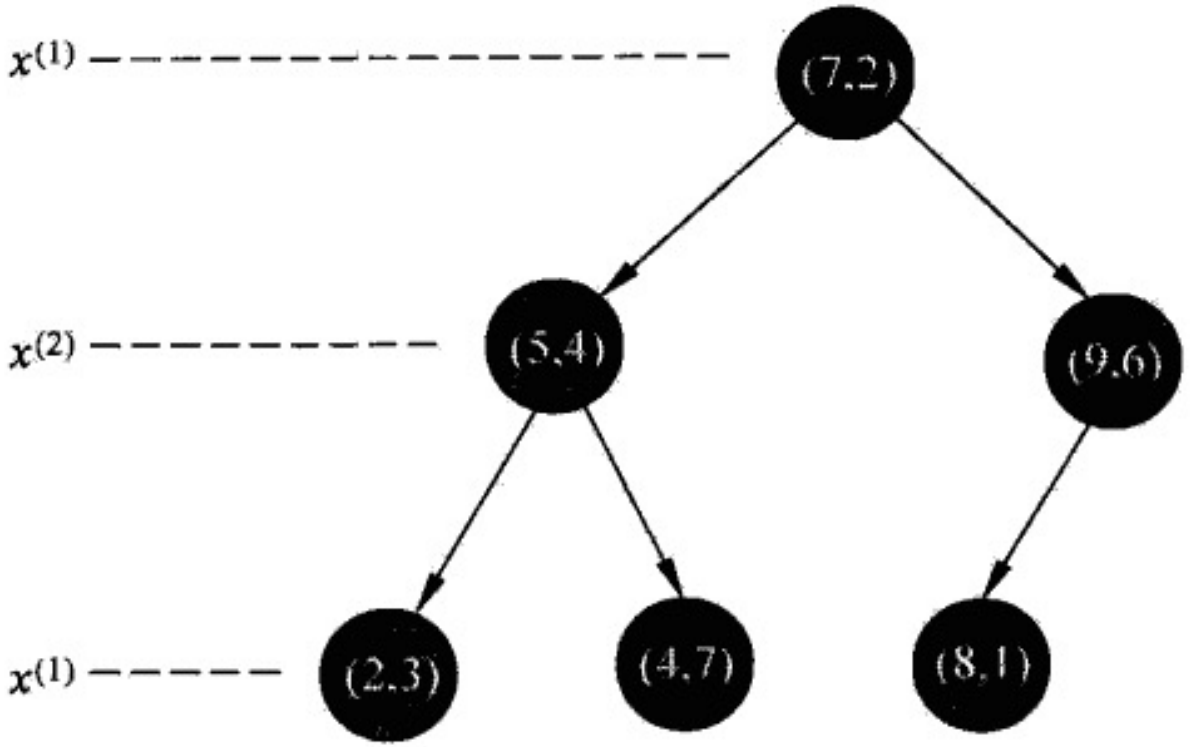



图1 二维数据k-d树空间划分示意图

# KD 树

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



**算法 3.3（用 $kd$ 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。

（1）在 $kd$ 树中找出包含目标点 $x$ 的叶结点：从根结点出发，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：

（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”。

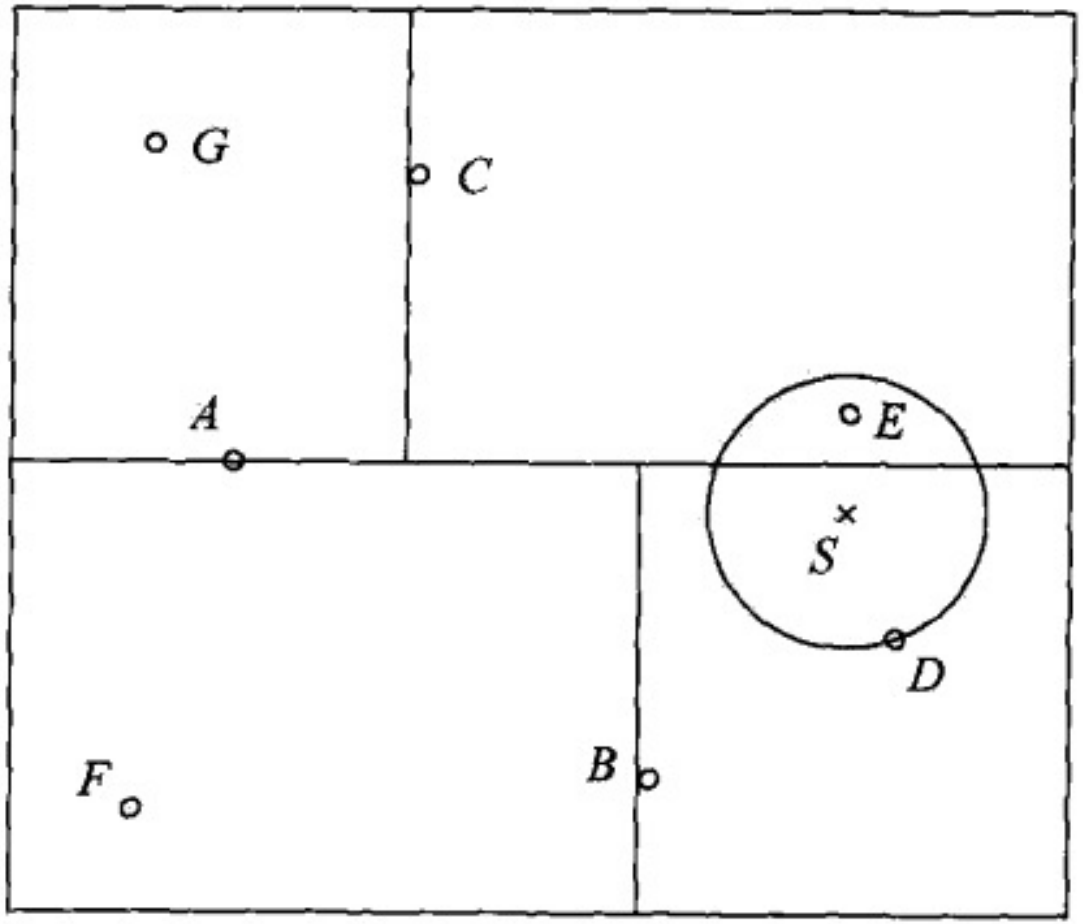
（b）当前最近点一定存在于该结点一个子结点对应的区域。检查该子结点的父结点的另一子结点对应的区域是否有更近的点。具体地，检查另一子结点对应的区域是否与以目标点为球心、以目标点与“当前最近点”间的距离为半径的超球体相交。

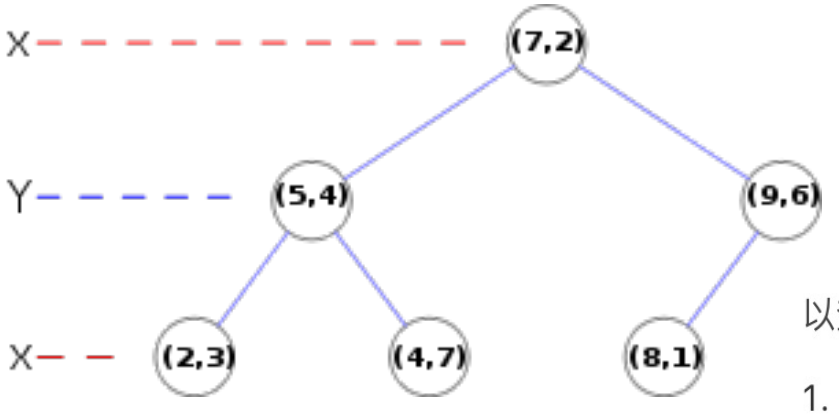
如果相交，可能在另一个子结点对应的区域内存在距目标点更近的点，移动到另一个子结点。接着，递归地进行最近邻搜索；

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 $x$ 的最近邻点。■

# KD树搜索





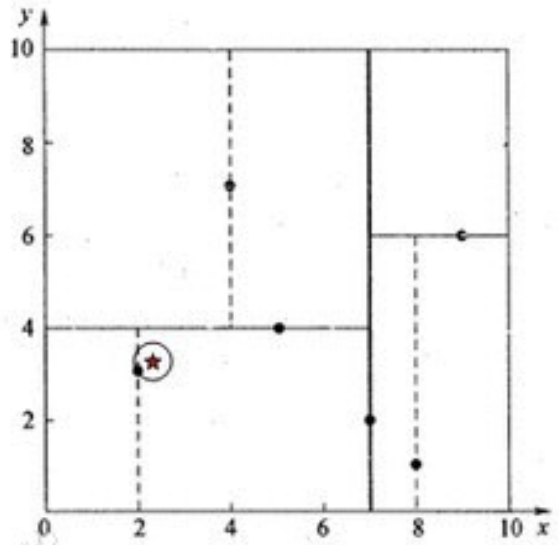
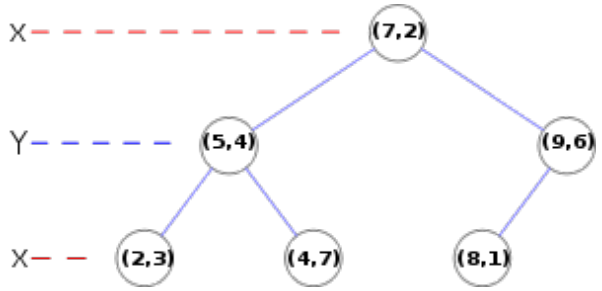




图4 查找（2.1，3.1）点的两次回溯判断

以查询（2.1,3.1）为例：

1. 二叉树搜索：先从（7,2）点开始进行二叉查找，然后到达（5,4），最后到达（2,3），此时搜索路径中的节点为<(7,2)，(5,4)，(2,3)>，首先以（2,3）作为当前最近邻点，计算其到查询点（2.1,3.1）的距离为0.1414，
2. 回溯查找：在得到（2,3）为查询点的最近点之后，回溯到其父节点（5,4），并判断在该父节点的其他子节点空间中是否有距离查询点更近的数据点。以（2.1,3.1）为圆心，以0.1414为半径画圆，如下图所示。发现该圆并不和超平面y = 4交割，因此不用进入（5,4）节点右子空间中(图中灰色区域)去搜索；
3. 最后，再回溯到（7,2），以（2.1,3.1）为圆心，以0.1414为半径的圆更不会与x = 7超平面交割，因此不用进入（7,2）右子空间进行查找。至此，搜索路径中的节点已经全部回溯完，结束整个搜索，返回最近邻点（2,3），最近距离为0.1414。



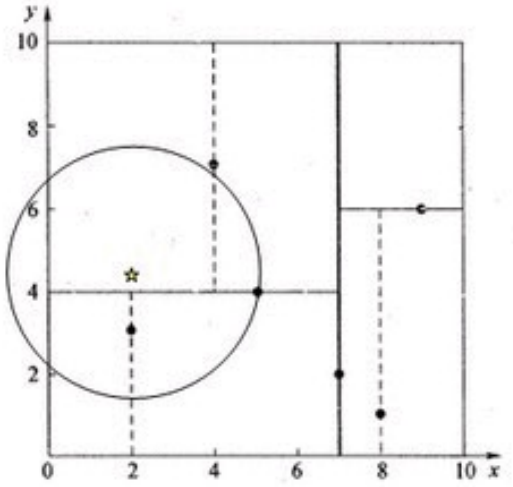




图5 查找（2，4.5）点的第一次回溯判断

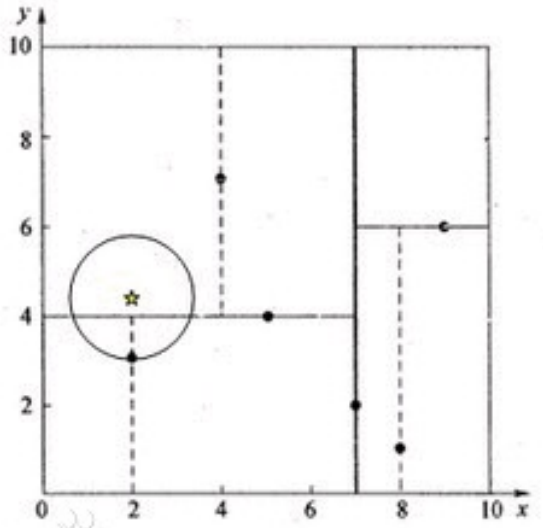



图6 查找（2，4.5）点的第二次回溯判断

1. 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)>，但（4,7）与目标查找点的距离为3.202，而（5,4）与查找点之间的距离为3.041，所以（5,4）为查询点的最近点；
2. 以（2，4.5）为圆心，以3.041为半径作圆，如下图所示。可见该圆和y = 4超平面交割，所以需要进入（5,4）左子空间进行查找，也就是将（2,3）节点加入搜索路径中得<(7,2)，(2,3)>；于是接着搜索至（2,3）叶子节点，（2,3）距离（2,4.5）比（5,4）要近，所以最近邻点更新为（2，3），最近距离更新为1.5；
3. 回溯查找至（5,4），直到最后回溯到根结点（7,2）的时候，以（2,4.5）为圆心1.5为半径作圆，并不和x = 7分割超平面交割，如下图所示。至此，搜索路径回溯完，返回最近邻点（2,3），最近距离1.5。