第3章 k近邻算法

k近邻算法

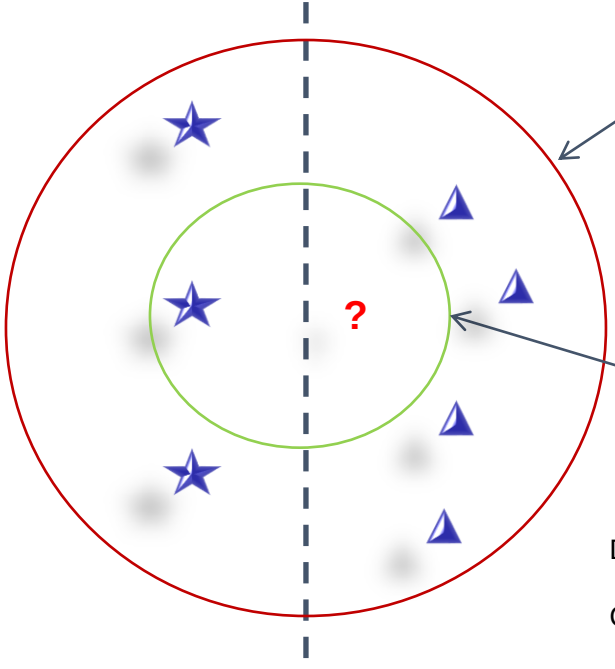
- k近邻法(k-nearest neighbor, k-NN) 是一种基本分类与回归方法。只讨论分类问题中的k近邻法
- k近邻法
  - 输入为实例的特征向量，对应于特征空间的点
  - 输出为实例的类别，可以取多类
  - k近邻法假设给定一个训练数据集，其中的实例类别已定

- 分类时，对新的实例，根据其k个最近邻的确定实例的类别，通过多数表决等方式进行预测
  - 因此，近邻法不具有显式的学习过程
  - 【是一个过程决定的方法，不是公式描述的方法，如果用公式描述，参数不确定】

1 k近邻算法

k-Nearest Neighbors算法原理

- 从众？！



K=7 Neighborhood

K=1 Neighborhood

Dependent of the data distributions.

Can make mistakes at boundaries.

k近邻法

算法3.1（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：

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

I为指示函数

【计算器的形式化写法】

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

2 k近邻模型

k近邻模型

- 模型三个基本要素
  - 距离度量
  - k值的选择
  - 分类决策规则

模型

- 根据模型要素将特征空间划分为一些子空间，确定子空间里的每个点所属的类
- 特征空间中，对每个训练实例点 x_i ，距离该点比其他点更近的所有点组成一个区域，叫作单元(cell)每个训练实例点x_i拥有一个单元(以x_i为中心)，所有训练实例点的单元构成对特征空间的一个划分
- 每个单元的实例点的类别是确定的
- 维诺图(Voronoi diagram)

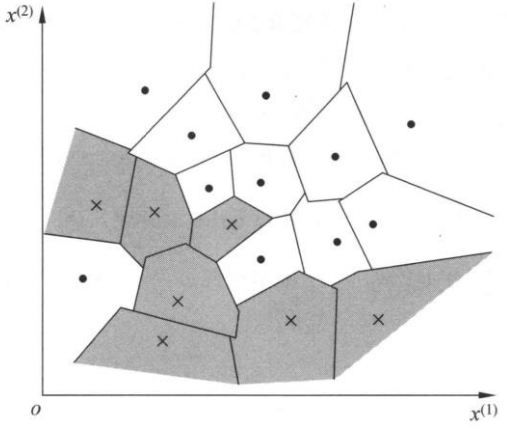


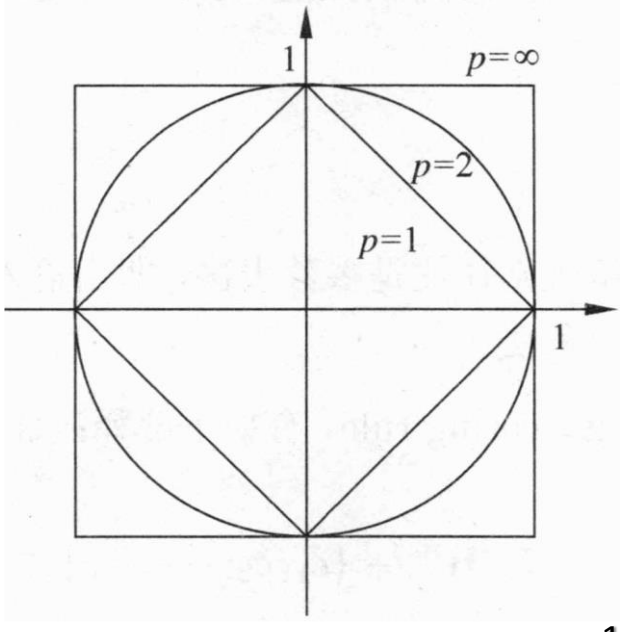
图 3.1 k 近邻法的模型对应特征空间的一个划分

距离度量

- 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
- L_p距离(Minkowski距离)

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

  - $p = 2$，欧式距离
  - $p = 1$，曼哈顿距离
  - $p = \infty$， $L_\infty(x_i, x_j) = \max_l \left| x_i^{(l)} - x_j^{(l)} \right|$



k值的选择

- 选择较小的k值
  - 近似误差(approximation error)会减小，但估计误差(estimation error) 会增大
  - 对噪声敏感
  - k值的减小意味着整体模型变得复杂，容易发生过拟合
- 选择较大的k值，
  - 减少学习的估计误差，但学习的近似误差会增大.
  - k值的增大意味着整体的模型变得简单
- 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

3 k近邻的实现：kd树

每个维度上构建平衡二叉树

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。在特征空间的维数大及训练数据容量大时尤其必要
- kd树
  - 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
  - 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

- 【注意】此处kd的k是k维，并不是k近邻的k个邻居【沿用惯例，可将k换成n】

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

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，$i=j \bmod k+1$,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3.2

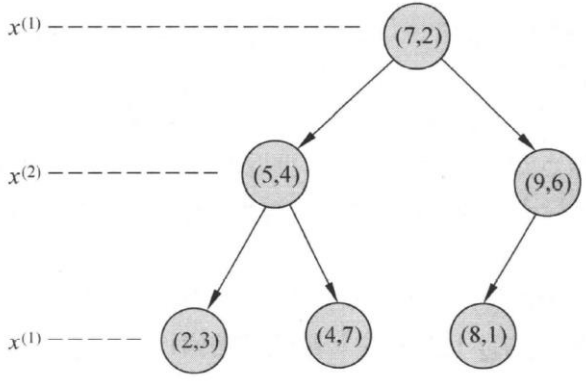
给定一个二维空间的数据集：

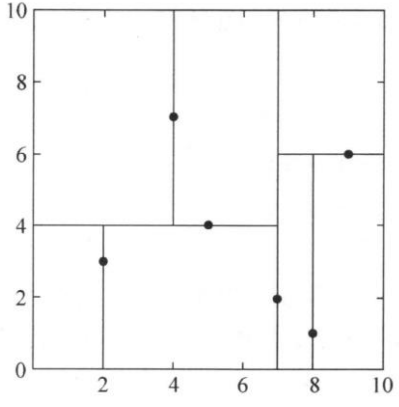
$$T = \{(2,3)^{\mathrm{T}}, (5,4)^{\mathrm{T}}, (9,6)^{\mathrm{T}}, (4,7)^{\mathrm{T}}, (8,1)^{\mathrm{T}}, (7,2)^{\mathrm{T}}\}$$

构造一个平衡kd树

根结点对应包含数据集T的矩形，选择$x^{(1)}$轴，6个数据点的$x^{(1)}$坐标的中位数是7，以平面$x^{(1)} = 7$将空间分为左、右两个子矩形（子结点）；

左矩形以$x^{(2)}$ ... 右矩形以$x^{(2)}$ ... 如此递归





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

- 问题：kd树根结点为A，其子结点为B, C等。树例上共存储7个实例点;另有一个输入目标实例点S,求S的最近邻
  - 首先在kd树中找到包含点S的叶结点D(图中的右下区域),以点D作为近似最近邻。真正最近邻一定在以点S为中心通过点D的圆的内部。
  - 然后返回结点D的父结点B，在结点B的另一子结点F的区域内搜索最近邻。结点F的区域与圆不相交，不可能有最近邻点。
  - 继续返回上一级父结点A，在结点A的另一子结点C的区域内搜索最近邻。结点C的区域与圆相交;该区域在圆内的实例点有点E，点E比点D更近，成为新的最近邻近似。最后得到点E是点S的最近邻