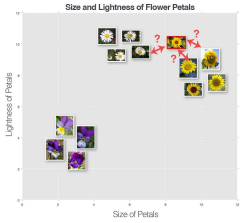
K-近邻算法

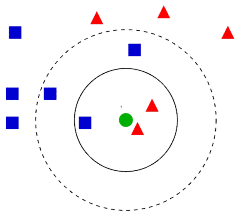
刘杰
人工智能学院
南开大学 Nankai University

提纲

KNN算法原理
KNN算法流程
距离函数
K值的选取
KD树

K-Nearest Neighbors算法





K-Nearest Neighbors算法

- 载入数据
- 初始化k值
- 预测类别，遍历每一个训练样本
  - 计算测试数据到每一个训练数据的距离；
- 按距离排序，取Top k个最相近的训练数据；
- 取频率最高的类别作为测试数据的预测类别

K-Nearest Neighbors算法

令z为测试样本x的最近邻，那么最近邻分类器出错的概率是x与z类别标记不同的概率

$$P(err) = 1 - \sum_{c\in Y} P(c|x)P(c|z)$$

$$P(err) = 1 - \sum_{c\in Y} P(c|x)P(c|z)$$
$$\approx 1 - \sum_{c\in Y} P^2(c|x)$$
$$\leq 1 - P^2(c^*|x)$$
$$= (1 + P(c^*|x))(1 - P(c^*|x))$$
$$\leq 2 \times (1 - P(c^*|x))$$

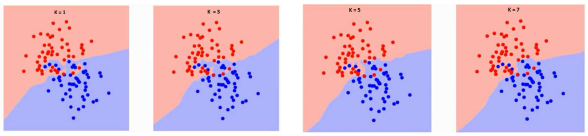
$c^* = \arg\max_{c\in Y} P(c^*|x)$
表示贝叶斯最优分类器的结果

KNN模型-关键之处

- 载入数据
- 初始化***k*值**
- 预测类别，遍历每一个训练样本
  - 计算测试数据到每一个训练数据的**距离**；
  - 按距离升序排序，**取Top *k*个最相近的训练数据**；
  - 取频率最高的类别作为测试数据的预测类别

→ *k*值确定
距离函数确定
复杂度

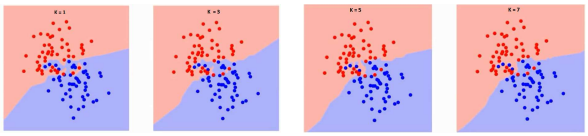
K值的选择



$k=1$ $k=3$ $k=5$ $k=7$

观察分类界面的变化

K值的选择



$k=1$ $k=3$ $k=5$ $k=7$

观察分类界面的变化：随着k的增大，分类界面边的更加平滑

k 大：学习的近似误差会增大，模型变得简单

k 小：敏感性增强，模型变得复杂，容易发生过拟合

距离度量

(x_1, y_1) 和(x_2, y_2)之间的距离计算

$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

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

$|x_1-x_2|+|y_1-y_2|$

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

距离度量

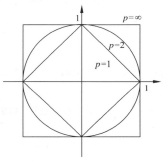
$$x_i=(x_i^{(1)},x_i^{(2)},...,x_i^{(n)})^T$$

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

欧式距离：$L_2(x_i,x_j)=(\sum_{l=1}^{n}|x_i^l-x_j^l|^2)^{\frac{1}{2}}$

曼哈顿距离：$L_1(x_i,x_j)=\sum_{l=1}^{n}|x_i^l-x_j^l|$

切比雪夫距离：$L_\infty(x_i,x_j)=\max_l|x_i^l-x_j^l|$



KNN的时间复杂度

假设训练数据为n个d维数据样本，

训练时，k-NN只需记住它看到的每个数据点的标签。

这意味着每添加一个数据点是$O(d)$。

测试时，我们需要计算新数据点和我们训练的所有数据点之间的距离。

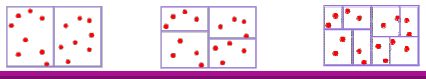
那么对一个测试输入进行分类是$O(dn)$。

为了达到最佳精度，我们希望我们的训练数据集非常大($n \gg 0$)，但这很快将成为测试期间的严重瓶颈。

目标：我们能在测试期间使k-NN更快吗？如果我们使用聪明的数据结构，我们可以。

KD树 (K-Dimension Tree)

- KD树是一种二叉树，其中每个节点都是一个k维数值点，每个节点都表示一个与当前划分维度的坐标轴垂直的超平面，并将空间分为两部分。
- 一部分在它的左子树，另一部分在它的右子树。
- 如果当前节点的划分维度为d，则d维度左子树上所有点的坐标值小于当前值，d维度右子树上所有点的坐标值大于或等于当前值。
- 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。这个结构允许可以有效地修剪我们的搜索空间，这样我们就不必访问每一个数据点。

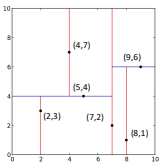


KD树

对深度为j的节点，选择x^l为切分的坐标轴$l = j(mod(k)) + 1$

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

构造kd树：

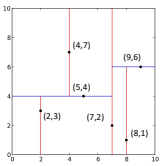


KD树

对深度为j的节点，选择x^l为切分的坐标轴$l = j(mod(k)) + 1$

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

构造kd树：

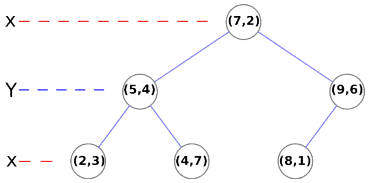


2, 3
5, 4
9, 6
4, 7
8, 1
7, 2

KD树

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

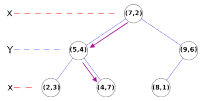
建立索引



KD树搜索

算法：（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。

例：查找（2，4.5）的最近邻

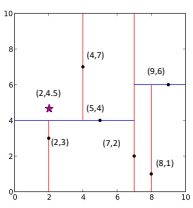


KD树搜索

算法：（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。

例：查找（2，4.5）的最近邻

路径：（7,2）(5,4)(4,7)

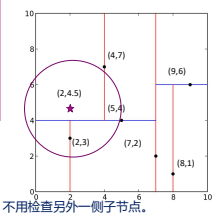


KD树搜索

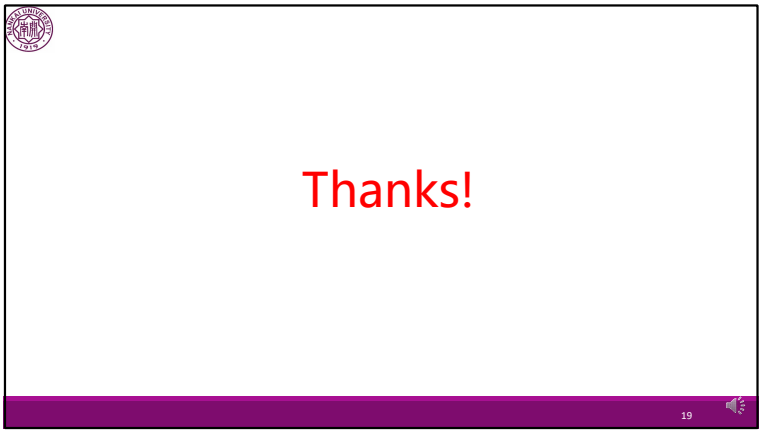
算法：（用kd 树的最近邻搜索）
输入：已构造的kd 树，目标点x;
输出：x的最近邻。
(1)在kd 树中找出包含目标点x 的叶结点:从根结点出发，递归地向下访问kd树。若目标点x当前维的坐标小于切分点的坐标，则移动到左子结点，否则移动到右子结点。直到子结点为叶结点为止。
(2) 以此叶结点为"当前最近点"。
(3) 递归地向上回退，在每个结点进行以下操作:
(a) 如果该结点保存的实例点比当前最近点距离目标点更近，则以该实例点为"当前最近点"。
(b) 当前最近点一定存在于该结点一个子结点对应的区域。检查该子结点的父结点的另一子结点对应的区域是否有更近的点。具体地，检查另一子结点对应的区域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2 的最近邻点。

例：查找（2，4.5）的最近邻

路径：（7,2）(5,4)(4,7)



递归搜索：令叶子结点(4,7)为“当前最近”；
回溯：(5,4)，距离比 (4,7)更近，更新为“当前最近”；
检查：更新“当前最近”为(2,3)；
回溯：(7,2)，距离比 (2,3)更远，“当前最近”不变；
检查：（2,3）为圆心，（2,3）与（2, 4.5）距离为半径的圆与x=7不相交，不用检查另外一侧子节点。


Thanks!
19