机器学习

Machine Learning

主讲人：张敏 清华大学长聘副教授

ML 机器学习

MACHINE LEARNING-MIN ZHANG

Unit.06

基于实例的学习(II)

*图片均来自网络或已发表刊物

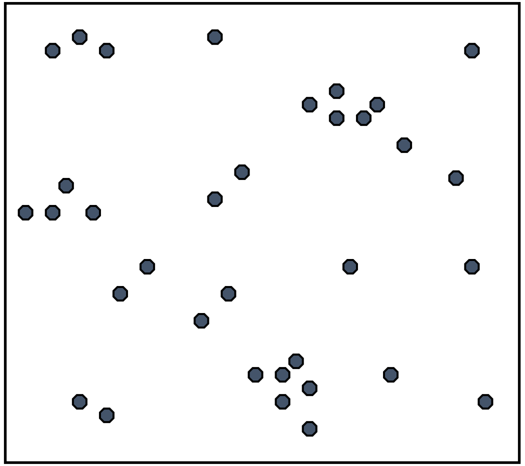
KNN 总览

- 基本算法
- 讨论
  - 更多距离度量
  - 属性：归一化、加权
  - 连续取值目标函数
  - k 的选择
  - 打破平局
  - 关于效率

KNN 讨论 6: 关于效率

- KNN算法把所有的计算放在新实例来到时，实时计算开销大
- 加速对最近邻居的选择
  - 先检验临近的点
  - 忽略比目前找到最近的点更远的点
- 通过 KD-tree 来实现：
  - KD-tree: k 维度的树（数据点的维度是 k）
  - 基于树的数据结构
  - 递归地将点划分到和坐标轴平行的方形区域内

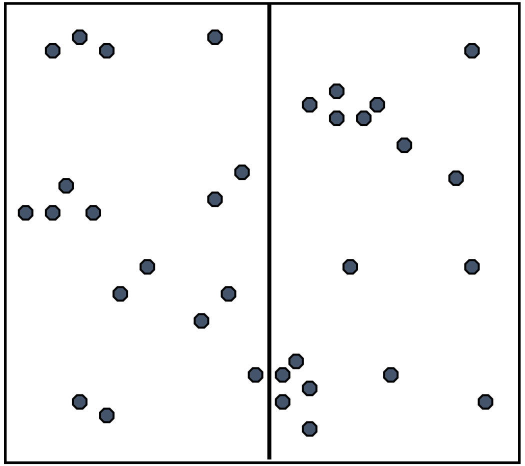
KD-Tree: （1）构建



Pt	X	Y
1	0.00	0.00
2	1.00	4.31
3	0.13	2.85
…	…	…

从一系列数据点出发

KD-Tree: （1）构建



X>.5

NO

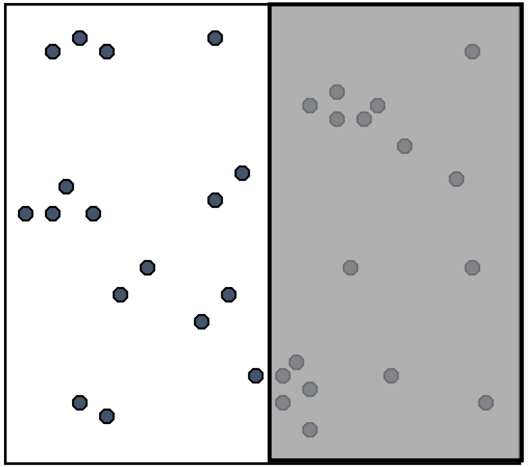
Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

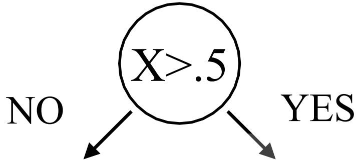
YES

Pt	X	Y
2	1.00	4.31
…	…	…

我们可以选择一个维度 X 和分界值 V 将数据点分为两组：X > V 和 X <= V

KD-Tree:（1）构建



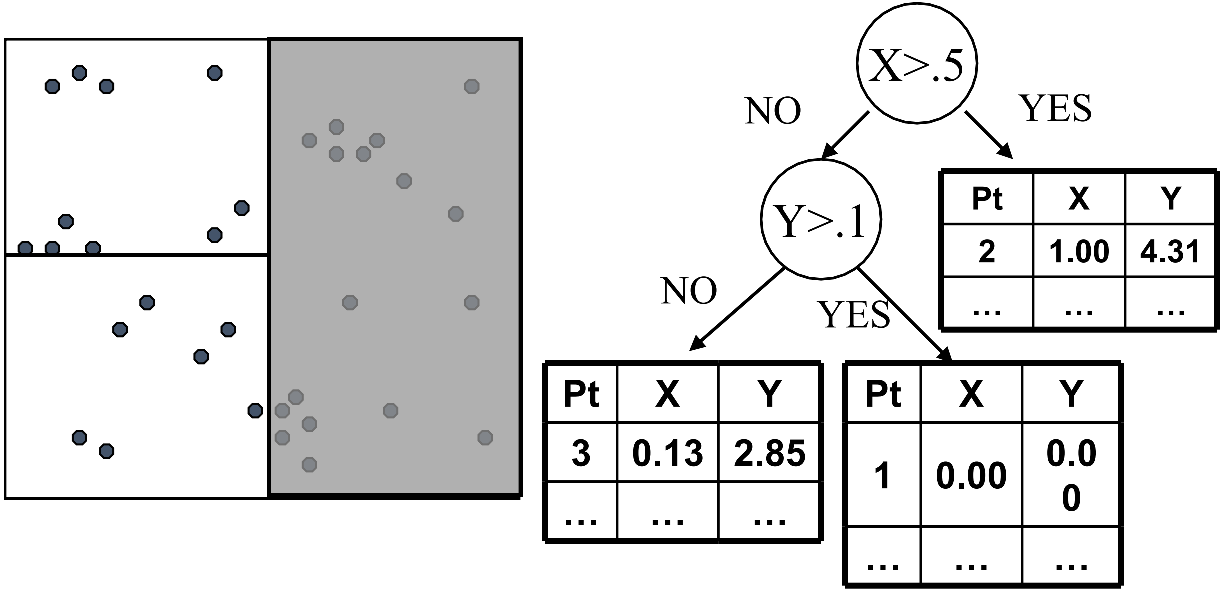


Pt	X	Y
1	0.00	0.00
3	0.13	2.85
…	…	…

Pt	X	Y
2	1.00	4.31
…	…	…

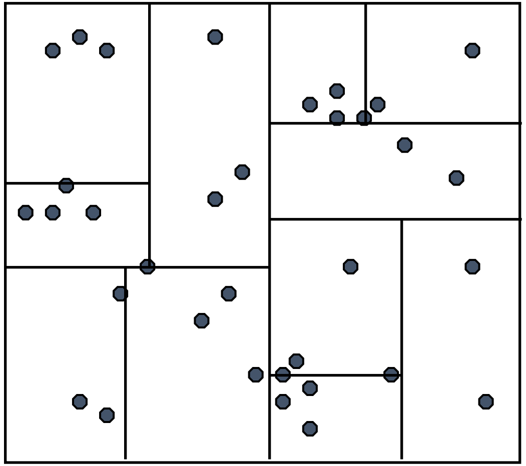
接下来分别考虑每个组，并再次分割（可以沿相同或不同的维度）

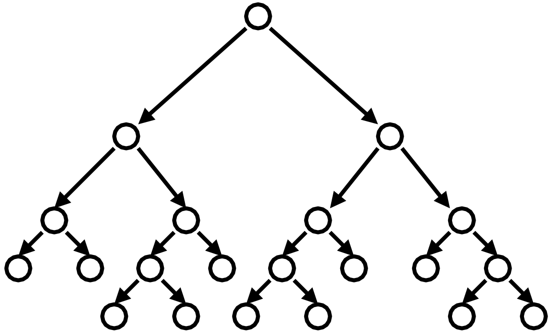
KD-Tree: （1） 构建



接下来分别考虑每个组，并再次分割（可以沿相同或不同的维度）

KD-Tree: （1） 构建

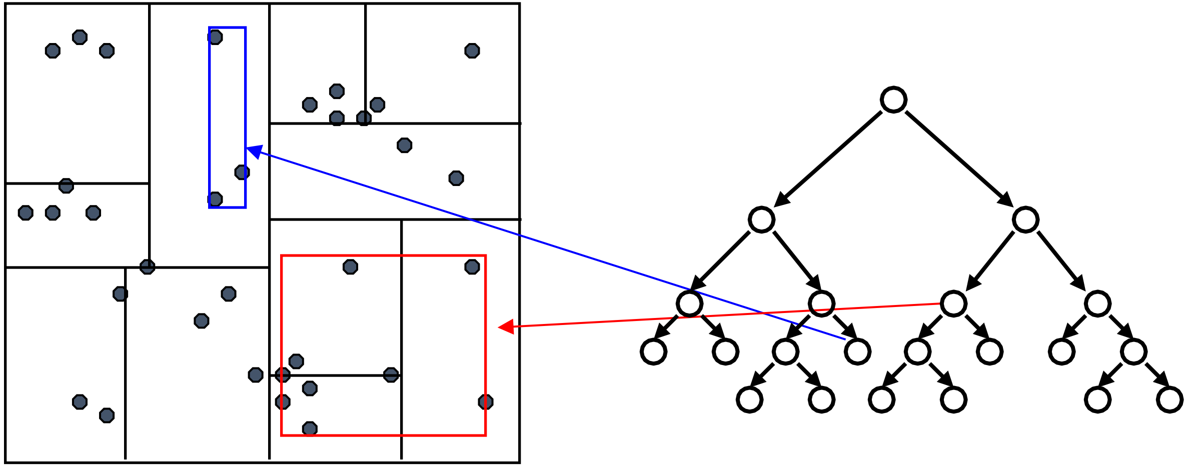




持续分割每个集合中的数据点，从而构建一个树形结构

每个叶节点表示为一系列数据点的列表

KD-Tree: （1） 构建



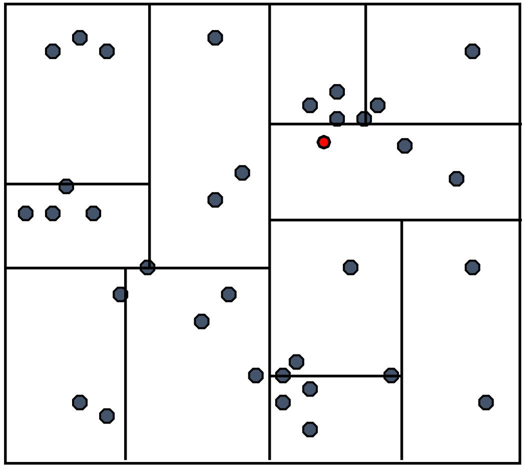
在每个叶节点维护一个额外信息：**这个节点下所有数据点的 (紧) 边界**

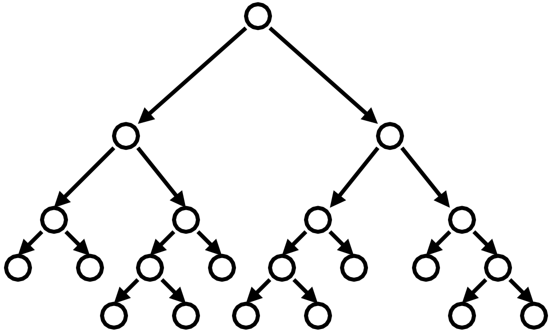
KD-Tree:（1）构建

用启发式的方法去决定如何分割

- 沿哪个维度分割？
  - 范围最宽的维度
- 分割的值怎么取？
  - 数据点在分割维度的中位数
  - 为什么是「中位数」而不是「均值」？
- 什么时候停止分割？
  - 当剩余的数据点少于 m，或者
  - 区域的宽度达到最小值

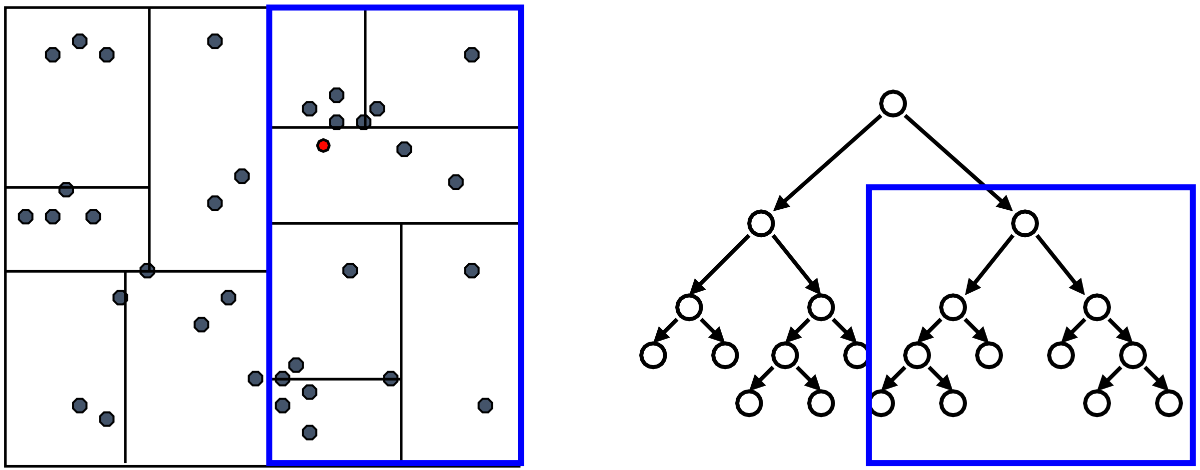
KD-Tree: （2） 查询





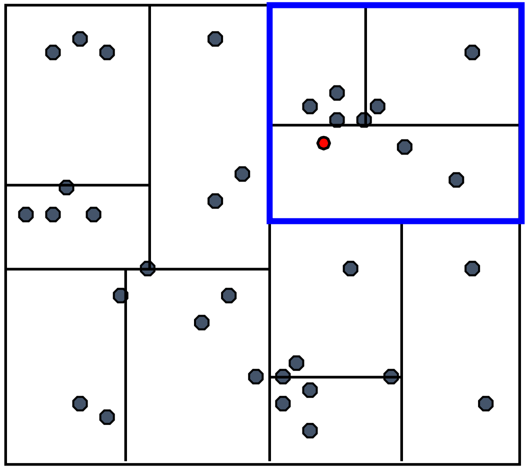
遍历树，来查找所查询数据点的最近邻居

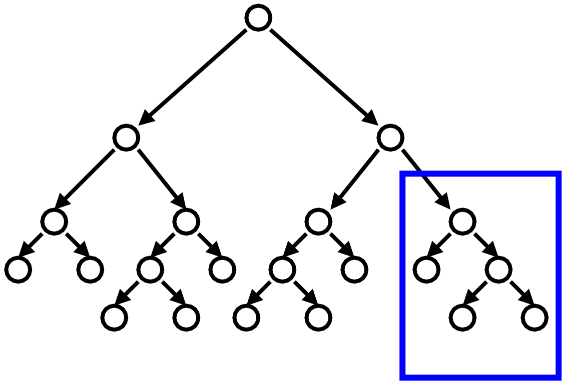
KD-Tree: （2） 查询



先检验临近的点：关注距离所查询数据点最近的树的分支

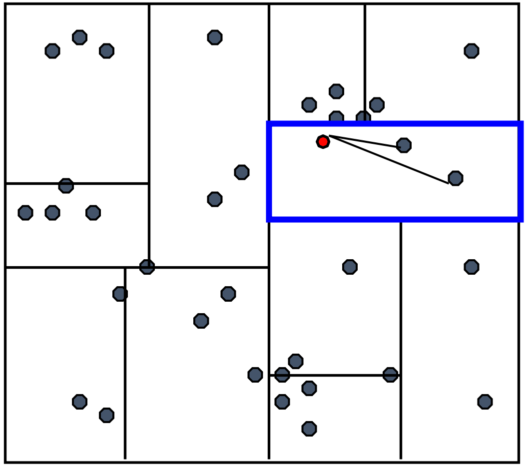
KD-Tree:（2） 查询

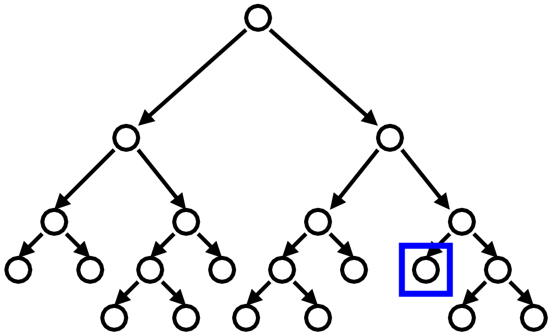




先检验临近的点：关注距离所查询数据点最近的树的分支

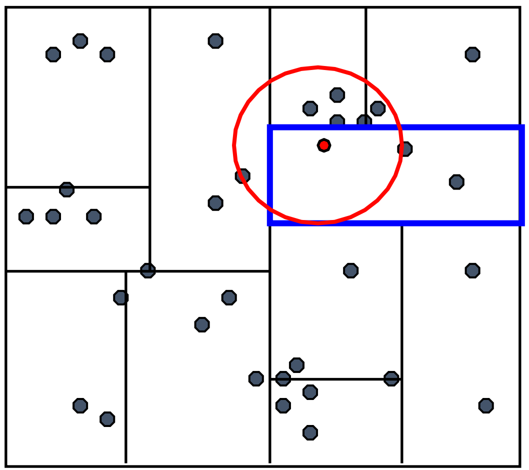
KD-Tree: （2） 查询

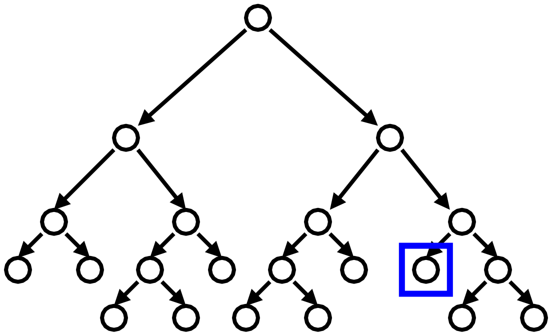




达到一个叶节点后：计算节点中每个数据点距离目标点的距离

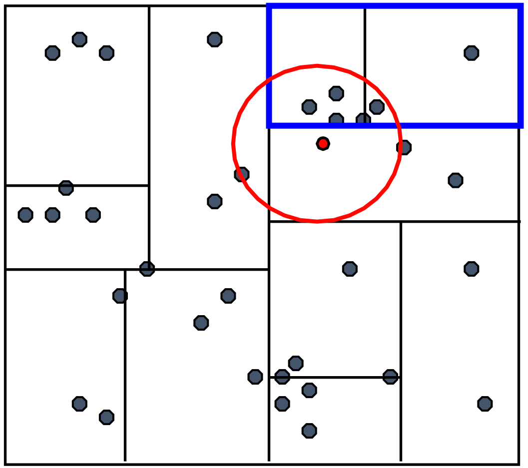
KD-Tree:（2） 查询

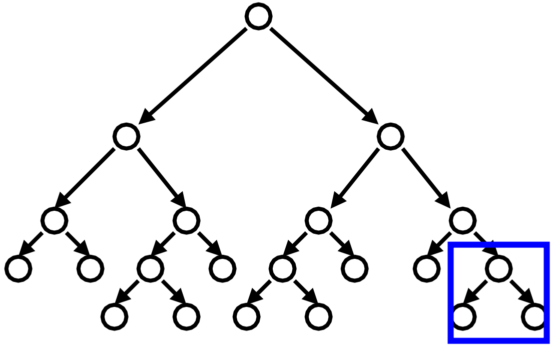




达到一个叶节点后：计算节点中每个数据点距离目标点的距离

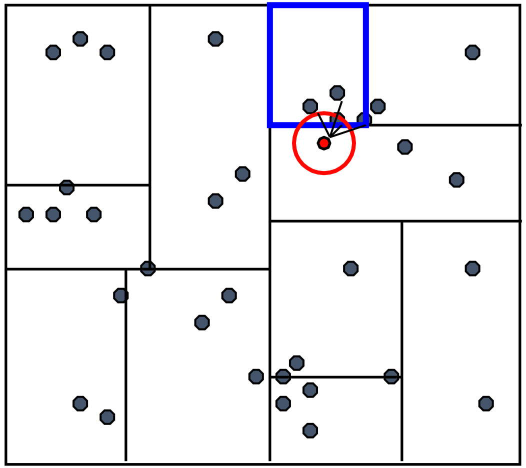
KD-Tree: （2） 查询

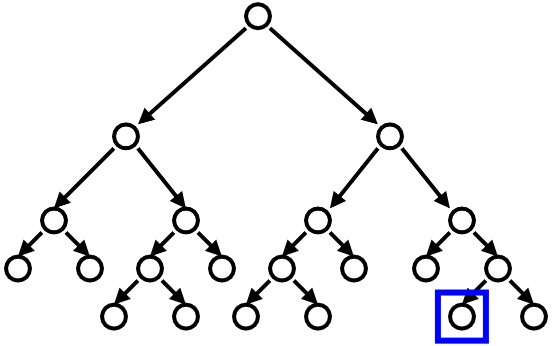




接着**回溯**检验我们访问过的每个树节点的另一个分支

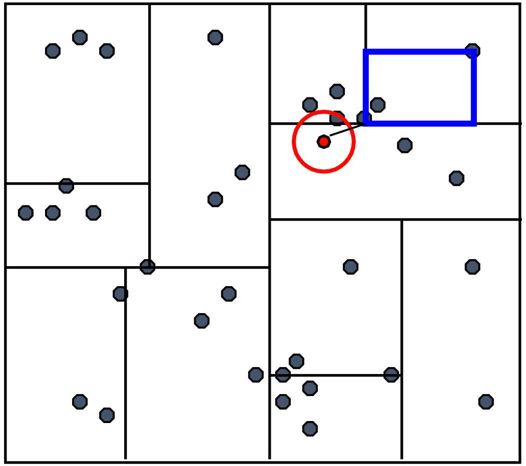
KD-Tree: （2） 查询

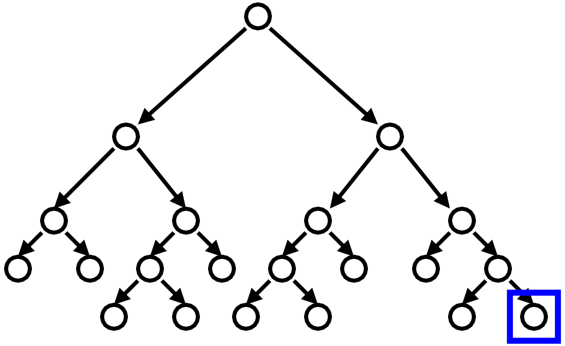




每次我们找到一个最近的点，就更新距离的上界

KD-Tree: （2） 查询

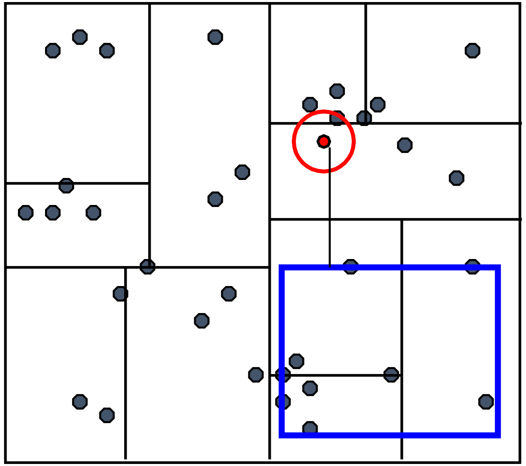


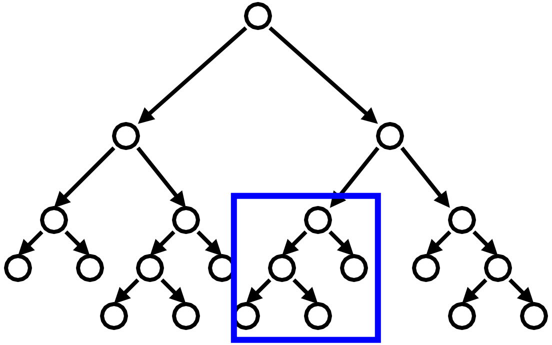


利用这个最近距离以及每个树节点下数据的边界信息，

我们可以对一部分不可能包含最近邻居的分支进行剪枝

KD-Tree: （2） 查询

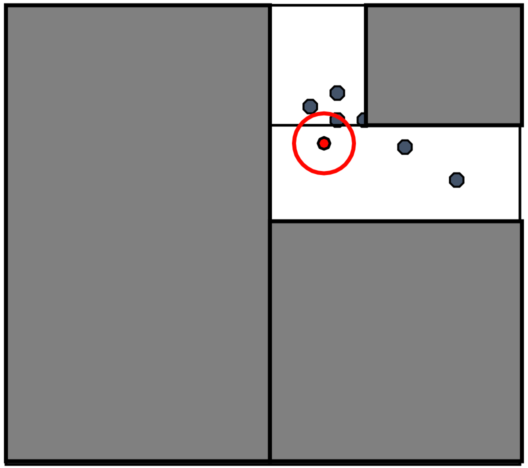


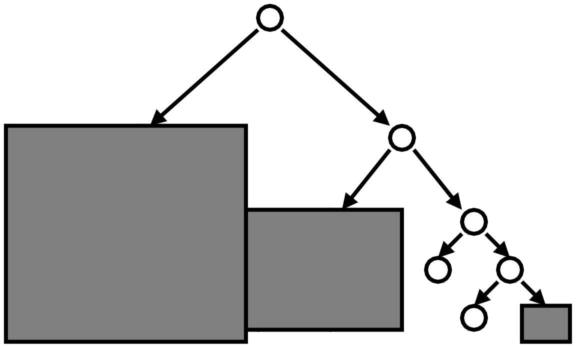


利用这个最近距离以及每个树节点下数据的边界信息，

我们可以对一部分不可能包含最近邻居的分支进行剪枝

KD-Tree: （2） 查询





利用这个最近距离以及每个树节点下数据的边界信息，
我们可以对一部分不可能包含最近邻居的分支进行剪枝

KNN 总览

- 基本算法
- 讨论
  - 更多距离度量
  - 属性：归一化、加权
  - 连续取值目标函数
  - k 的选择
  - 打破平局
  - 关于效率 – KD-tree
- 优点与缺点

KNN 总览：优点

- 概念上很简单，但可以处理复杂的问题（以及复杂的目标函数）
  - e.g. 图片分类
- 通过对k-近邻的平均，对噪声数据更鲁棒
- 容易理解：预测结果可解释（最近邻居）
- 训练样例中呈现的信息不会丢失
  - 因为样例本身被显式地存储下来了
- 实现简单、稳定、没有参数（除了 k）
- 方便进行 leave-one-out 测试

KNN 总览：缺点

- **内存开销**
  - 需要大量的空间存储所有样例
  - 通常来说，需要存储任意两个点之间的距离 $O(n^2)$ ； K-DTrees $O(n\log n)$
- **CPU 开销**
  - 分类新样本需要更多的时间（因此多用在离线场景）
- 很难确定一个合适的**距离函数**
  - 特别是当样本是由复杂的符号表示时
- **不相关的特征** 对距离的度量有负面的影响

下一个问题

- 回忆：用多个邻居使得对噪声数据鲁棒

这些邻居的贡献是一样的吗？

- 解决方案
  - 对数据加权
  - 更接近所查询数据点的邻居赋予更大的权重

↓

距离加权近邻

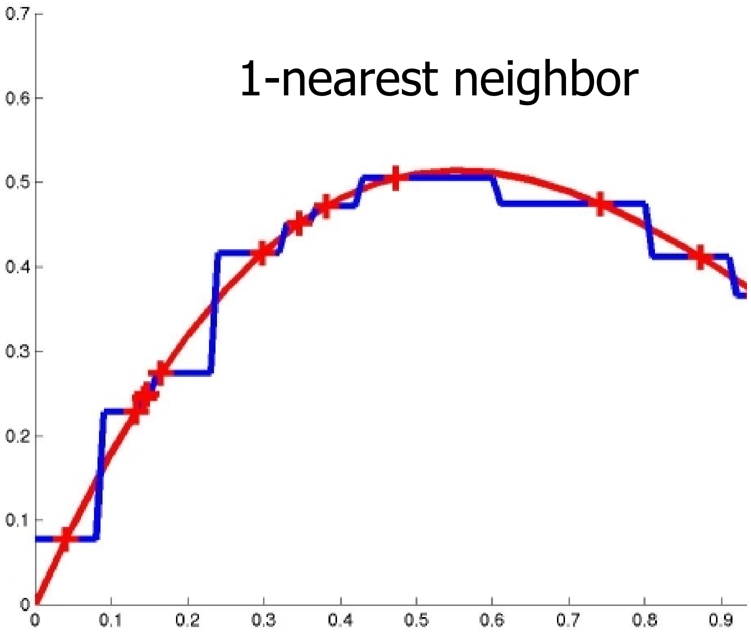
三、距离加权 KNN

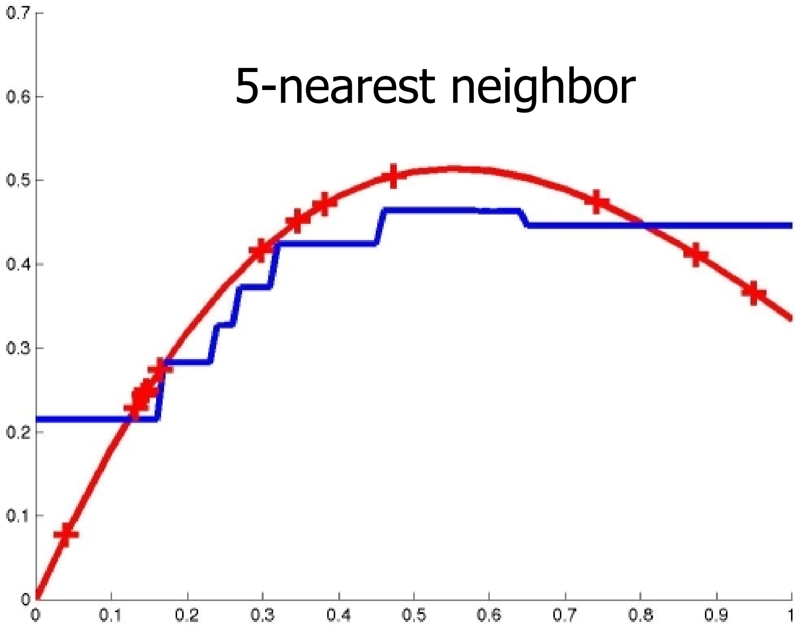
Distance-weighted KNN

距离加权 KNN

- 一种加权函数
  - $w_i = \mathrm{K}(\mathrm{d}(x_i, x_q))$
  - $\mathrm{d}(x_i, x_q)$ ：查询数据点与 x_i 之间的关系
  - $K(\cdot)$ ：决定每个数据点权重的核函数
- 输出：　加权平均：$\mathrm{predict} = \Sigma w_i y_i \,/\, \Sigma w_i$
- 核函数 $\mathrm{K}(\mathrm{d}(x_i, x_q))$
  - $1/d^2$, e^{-d}, $1/(1+d)$, …　应该和距离 d 成反比

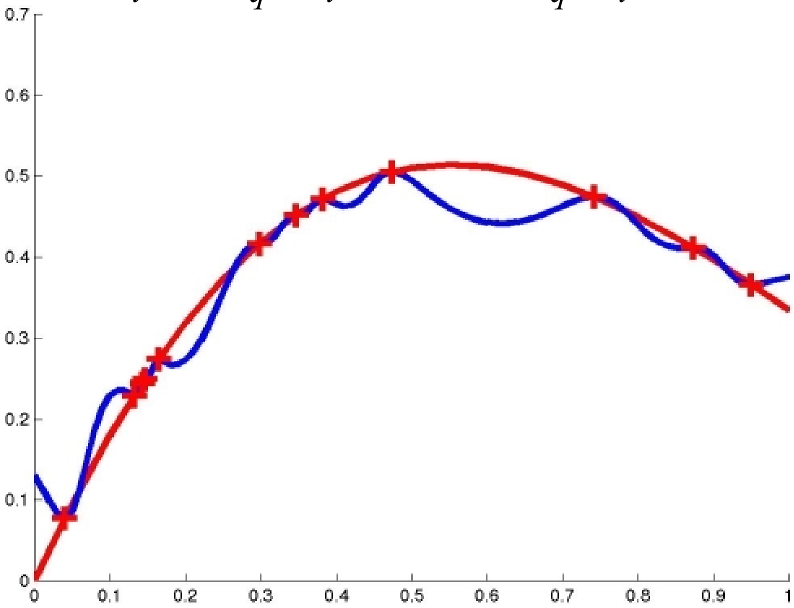
回顾


1-nearest neighbor
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

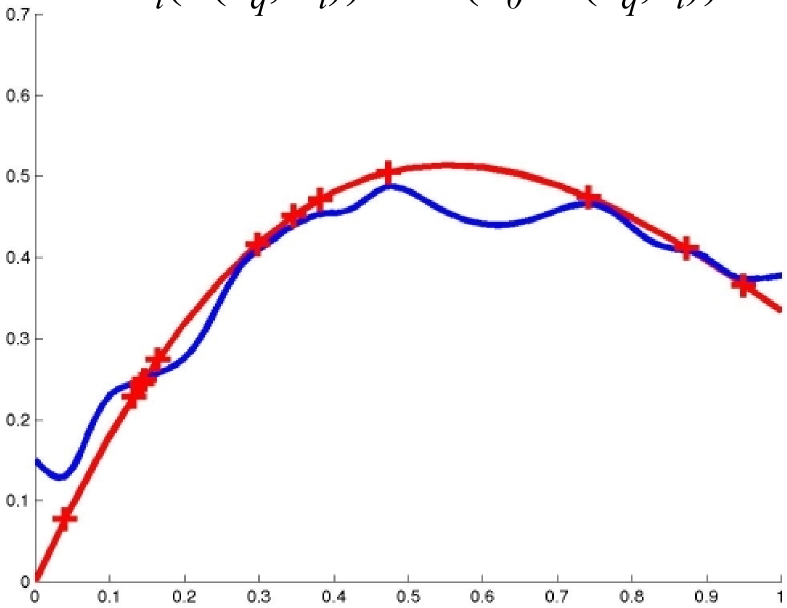

5-nearest neighbor
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

距离加权 NN

$W_i(d(x_q, x_i)) = 1/d(x_q, x_i)^2$

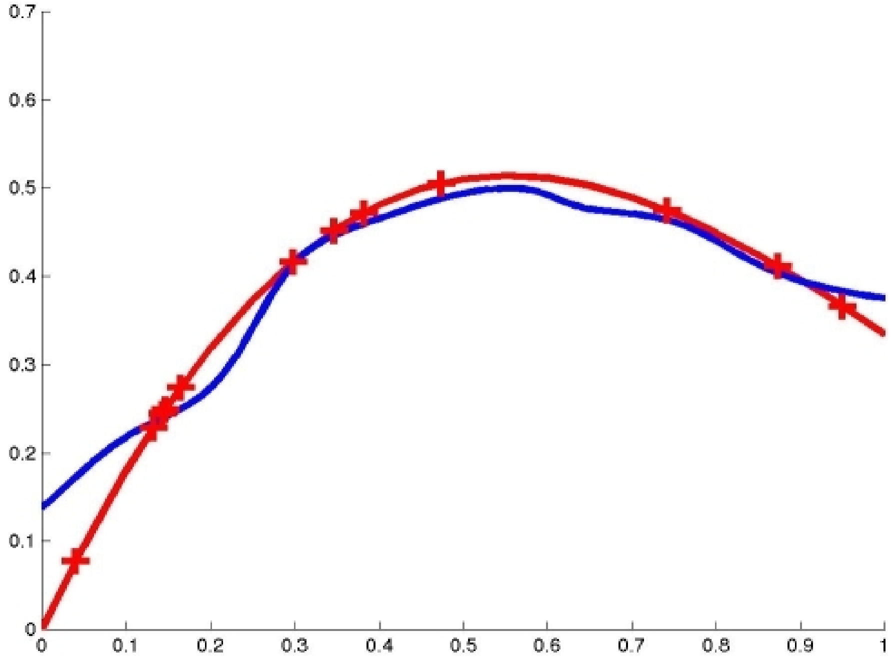


$W_i(d(x_q, x_i)) = 1/(d_0+d(x_q,x_i))^2$



距离加权 NN

$$W_i(d(x_q, x_i)) = e^{-(d(x_q, x_i)/\sigma_0)^2}$$



四、总览：

基于实例/记忆的学习器：4 个要素

基于记忆的学习器：4 个要素

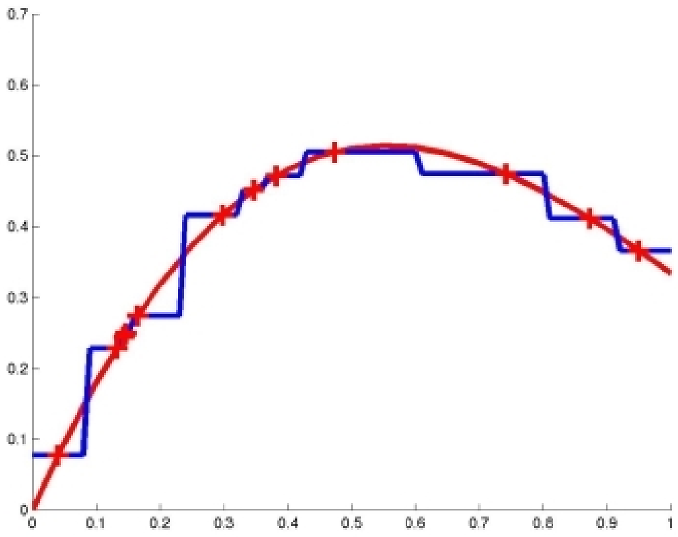
1. 一种距离度量
2. 使用多少个邻居？
3. 一个加权函数（可选）
4. 如何使用已知的邻居节点？

1-NN

基于记忆的学习器：4 个要素

1. 一种距离度量 欧式距离
2. 使用多少个邻居？ 一个
3. 一个加权函数（加权） 无
4. 如何使用已知的邻居节点？

和邻居节点相同

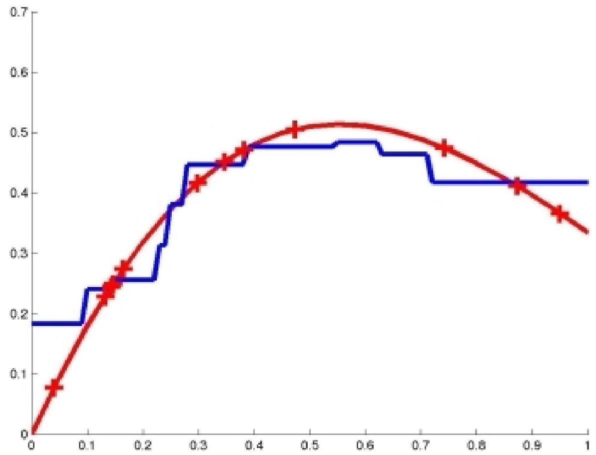


K-NN

基于记忆的学习器：4 个要素

1. 一种距离度量　　欧式距离
2. 使用多少个邻居？　　K 个
3. 一个加权函数（加权）　　无
4. 如何使用已知的邻居节点？

K 个邻居节点投票



距离加权 KNN

基于记忆的学习器：4 个要素

1. 一种距离度量　　缩放的欧式距离
2. 使用多少个邻居？　　所有的，或K 个
3. 一个加权函数（可选）

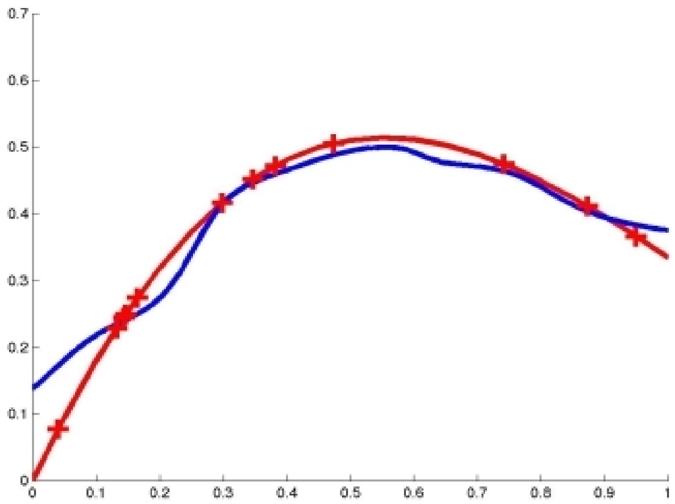
$$w_i = exp(-D(x_i, query)^2 / K_w^2)$$

K_w：核宽度。非常重要

4. 如何使用已知的邻居节点？

每个输出的加权平均 predict = $\Sigma w_i y_i / \Sigma w_i$

$$W_i(d(x_q, x_i)) = e^{-(d(x_q, x_i)/\sigma_0)^2}$$



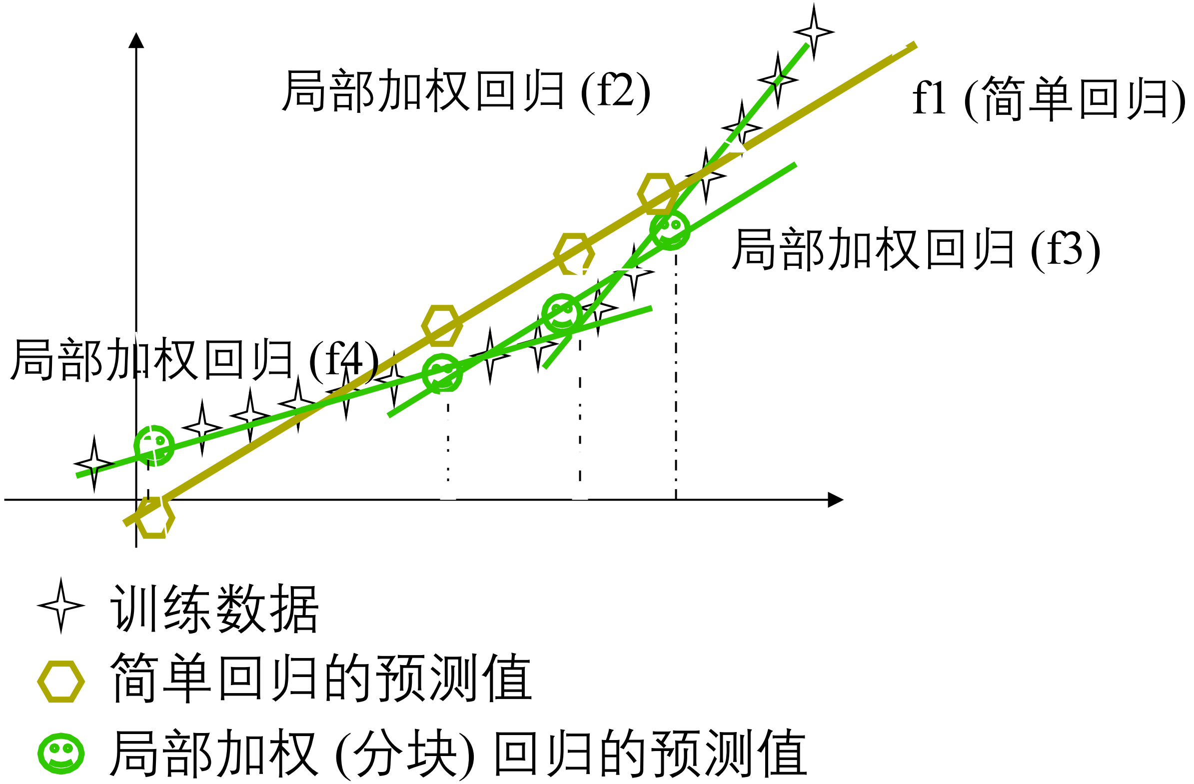
五、扩展：

局部加权回归

Locally weighted regression

- 回归：对实数值目标函数做估计/预测
- 局部：因为函数的估计是基于与所查询数据点相近的数据
- 加权：每个数据点的贡献由它们与所查询数据点的距离决定

局部加权回归（例子）


局部加权回归 (f2)
f1 (简单回归)
局部加权回归 (f3)
局部加权回归 (f4)
训练数据
简单回归的预测值
局部加权 (分块) 回归的预测值

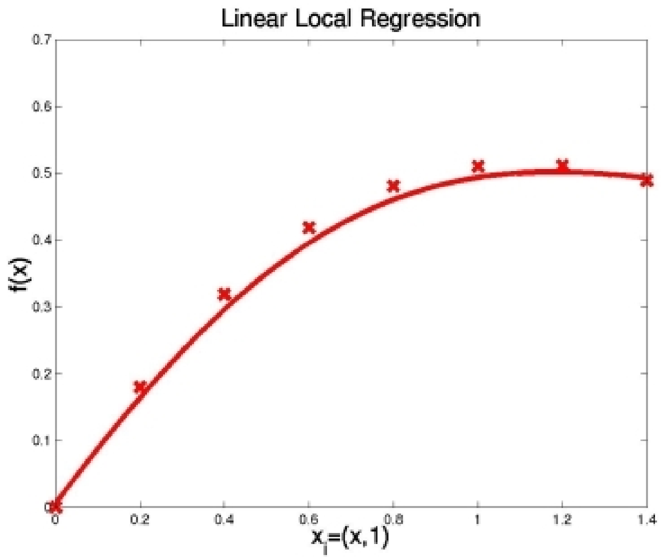
局部加权回归

基于记忆的学习器：4 个要素

1. 一种距离度量　　缩放的欧式距离
2. 使用多少个邻居？　所有的，或 K个
3. 一个加权函数（可选）

   e.g. $w_i = exp(-D(x_i, query)^2 / K_w^2)$

   K_w: 核宽度。非常重要
4. 如何使用已知的邻居节点？



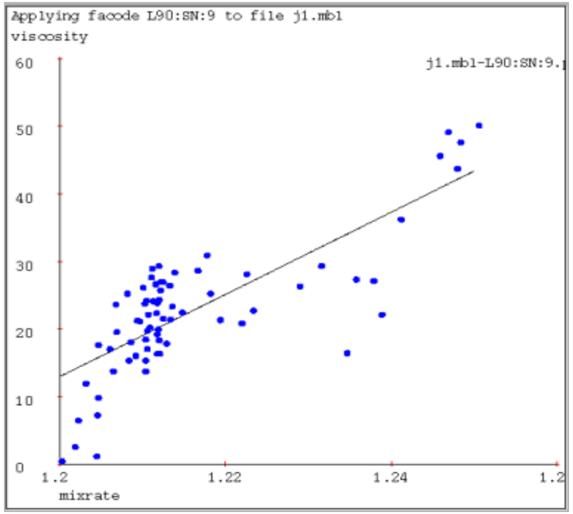
首先构建一个局部的线性模型。拟合 $\underline{\beta}$ 最小化局部的加权平方误差和：

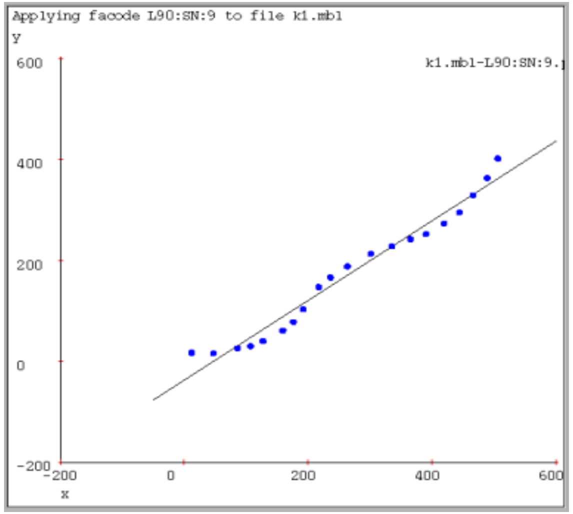
$$\underline{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{k=1}^{N} w_k^{\,2} \left(y_k - \beta^T x_k\right)^2$$

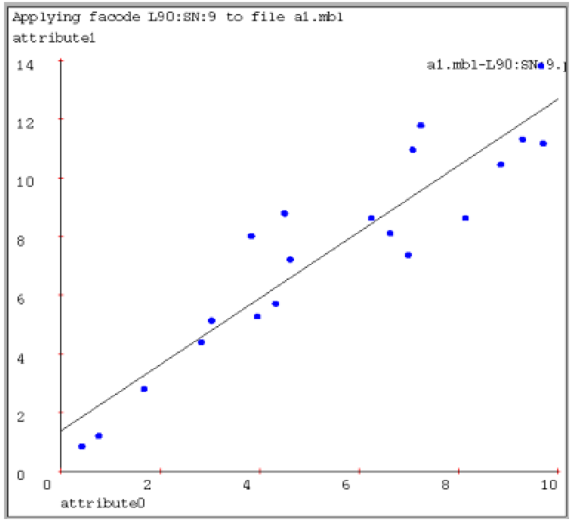
那么 $y_{predict} = \underline{\beta}^T x_{query}$

六、真实测试样例下
不同基于实例的算法表现举例

线性回归





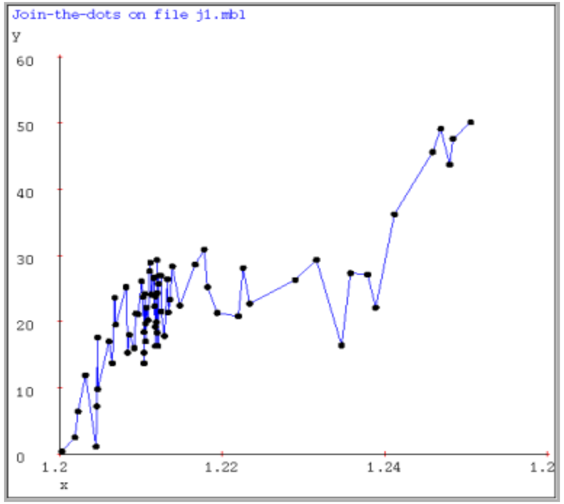


有明显的偏差

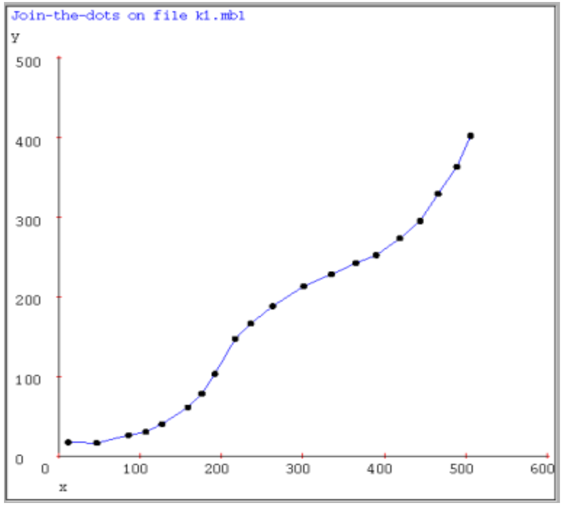
线性回归有较好的拟合结果，但偏差仍十分明显

线性回归可能确实是正确的选择

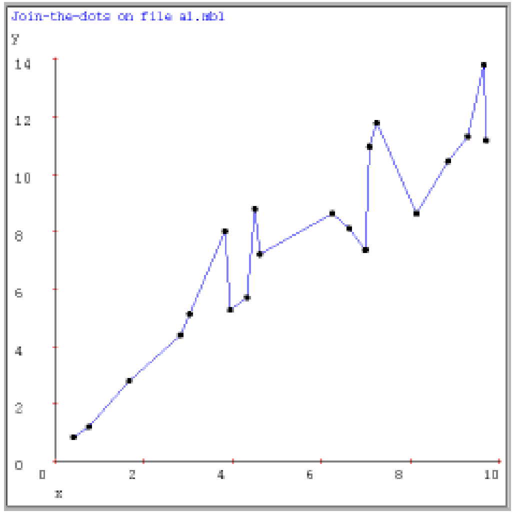
连接所有点



明显拟合了噪声

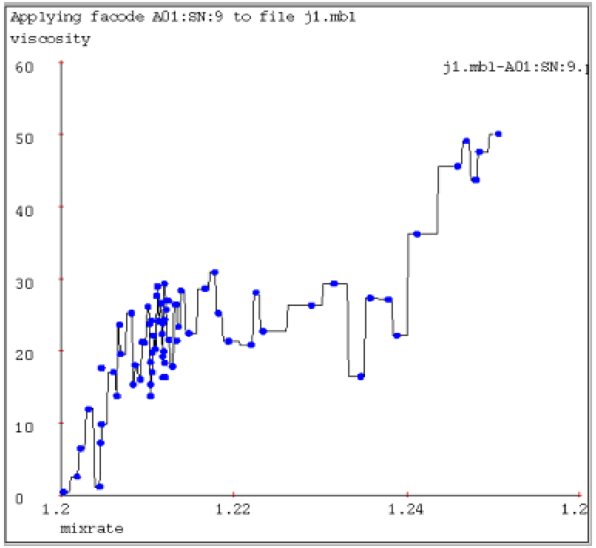


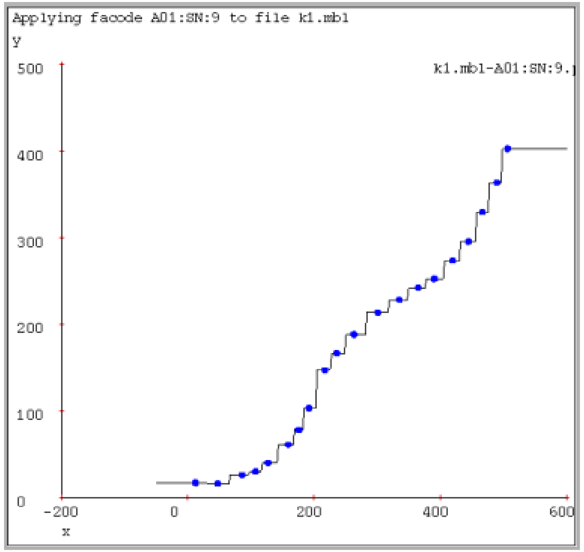
这里连接所有点看起来很正确

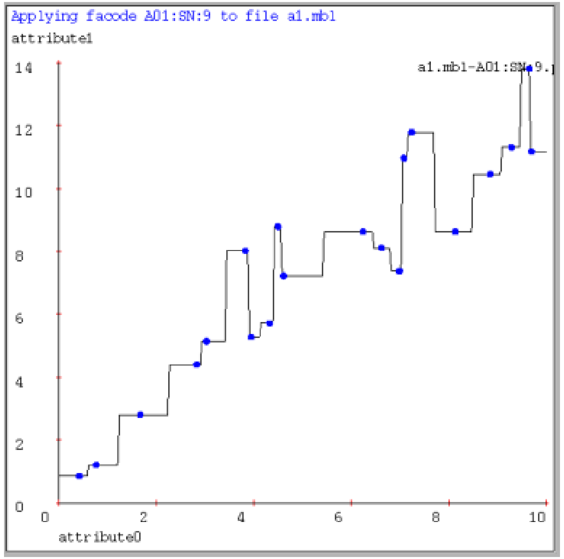


同样明显拟合了噪声

1-近邻

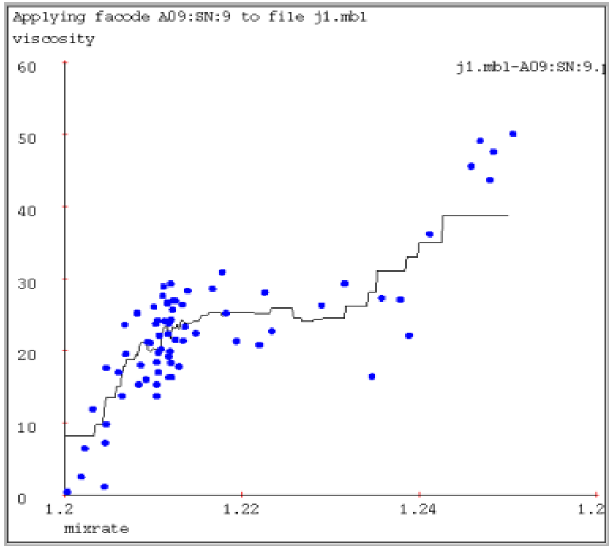




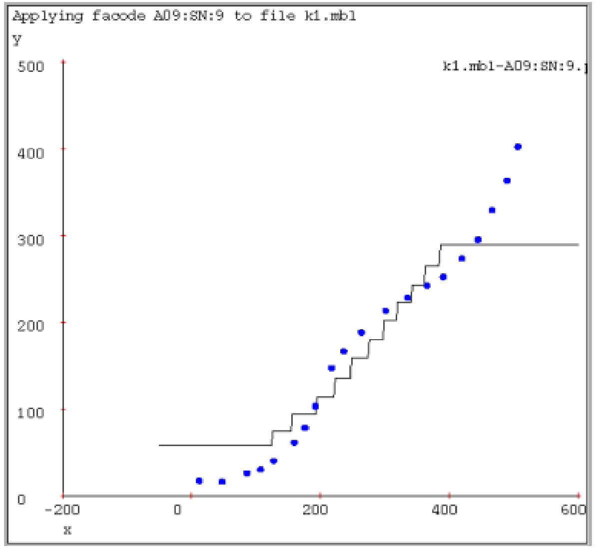


和连接所有点的结果很像

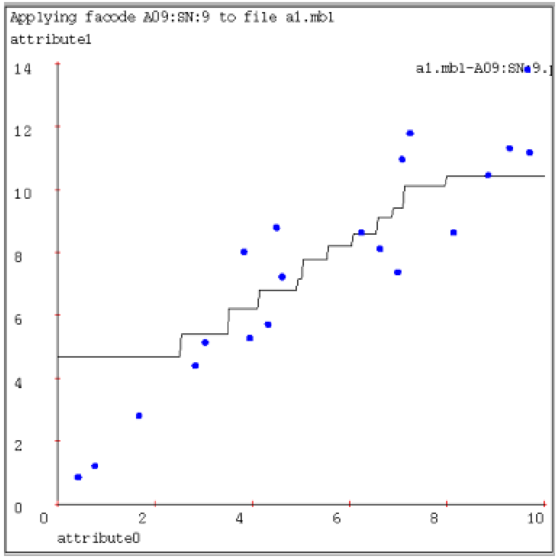
k-近邻（k=9）



很好地平滑了噪音，但不可微以及数值的抖动不是我们想要的

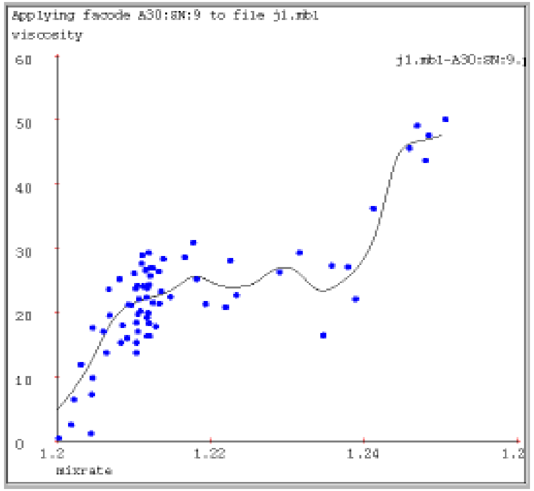


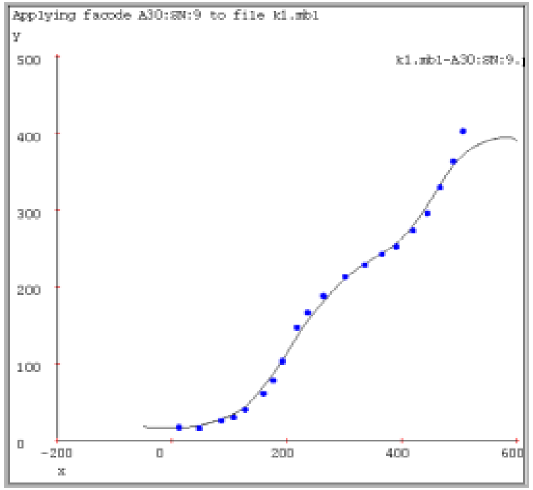
能够刻画总体趋势，对噪声更鲁棒一些，但仍然不如线性回归

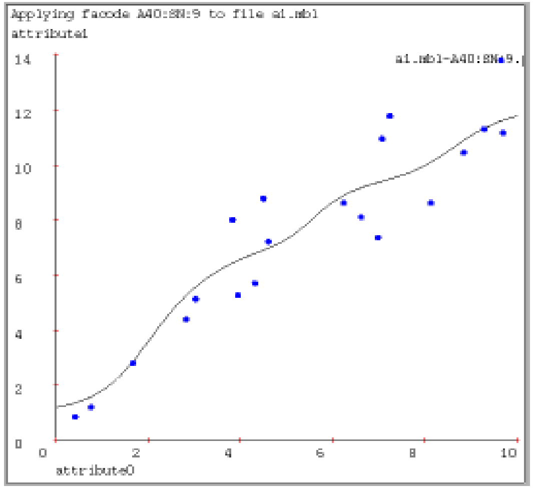


能够刻画总体趋势，对噪声更鲁棒一些，但仍然不如线性回归

距离加权 KNN（核回归）







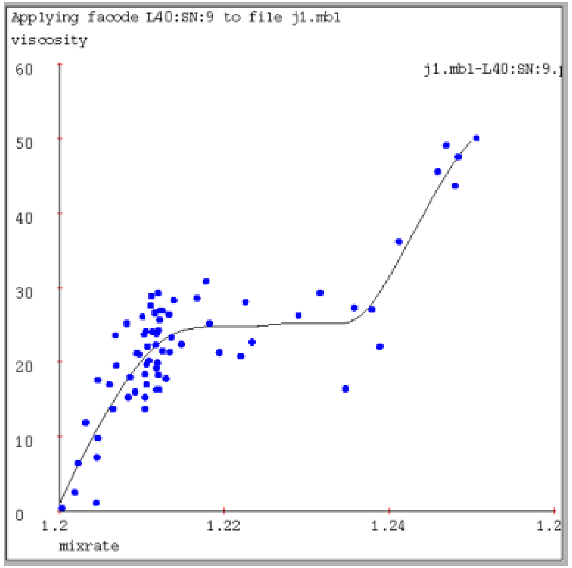
K_W= x轴宽度的1/32
最终得到了一条光滑的曲线，但抖动很大。如果 K_w 变大，拟合会更差

K_W= x轴宽度的1/32
完美！

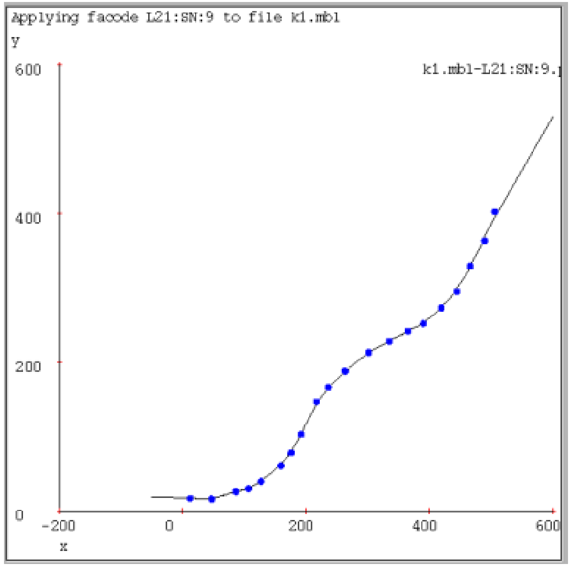
K_W= x轴宽度的1/16
很好且光滑。但抖动的问题似乎没有解决。或者也许过拟合了？

选择一个合适的 K_w 非常重要，不仅是对核回归，对所有局部加权学习器都很重要

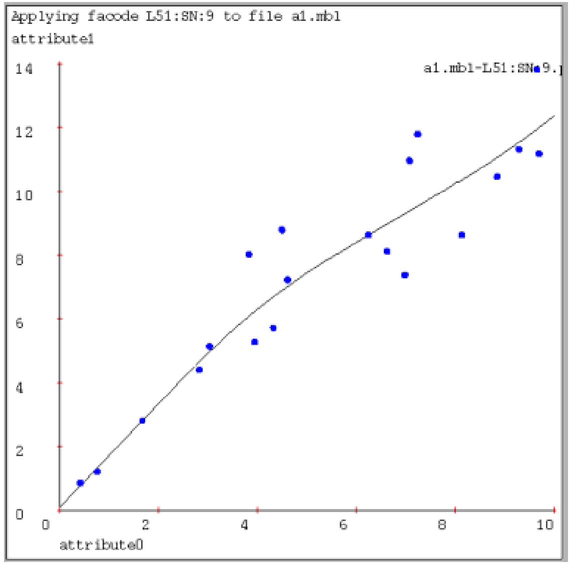
局部加权回归



K_W= x轴宽度的 1/16



K_W= x轴宽度的 1/32



K_W= x轴宽度的 1/8

拟合更好并且更光滑。

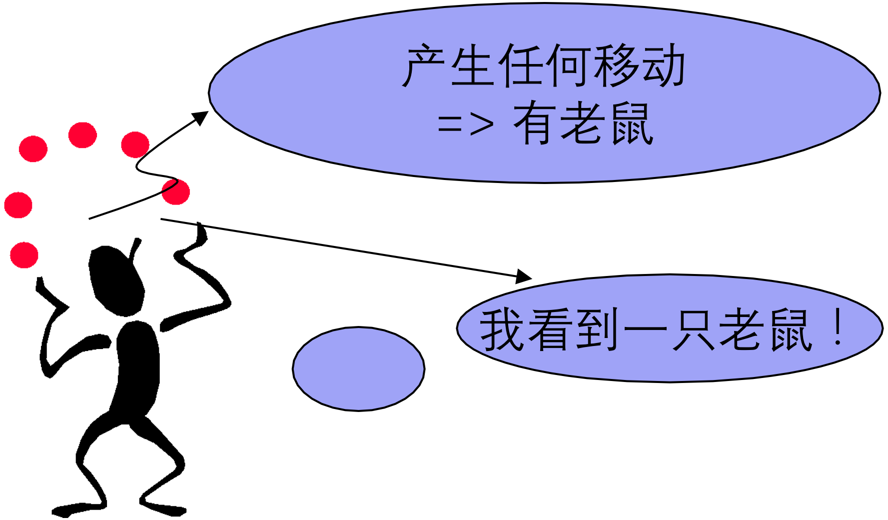
抖动的问题也好多了。

七、懒惰学习与贪婪学习

Lazy learner and Eager Learner

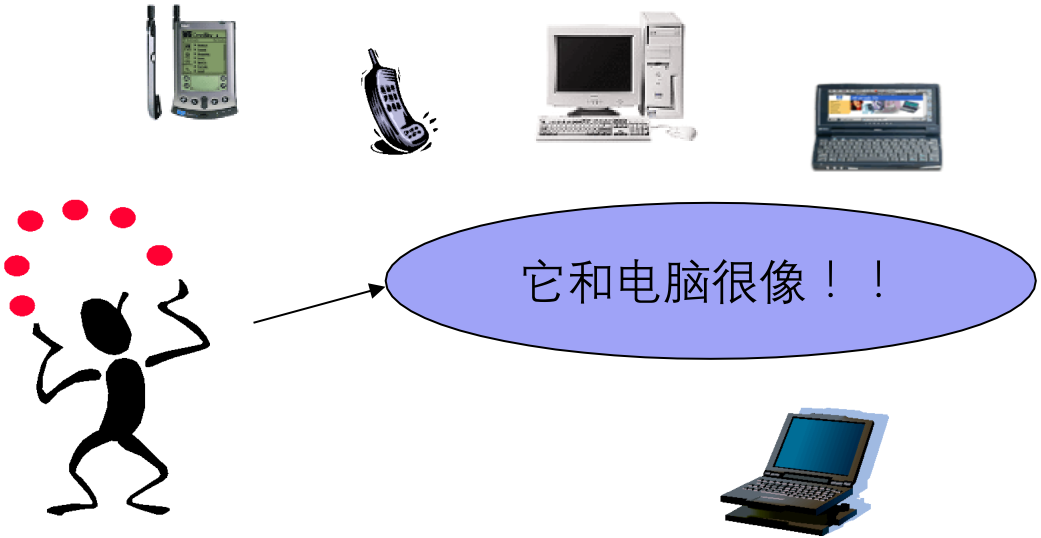
不同的学习方法

- 贪婪学习



不同的学习方法

- 懒惰学习 （例如基于实例的学习）



懒惰学习 vs. 贪婪学习

- 懒惰：等待查询再泛化
  - 训练时间：短
  - 测试时间：很长
- 懒惰学习器
  - 可以得到局部估计

- 贪婪：查询之前就泛化
  - 训练时间：长
  - 测试时间：短
- 贪婪学习器
  - 对于每个查询使用相同的模型
  - 倾向于给出全局估计

如果它们共享相同的假设空间，懒惰学习可以表示更复杂的函数

（e.g. H=线性函数）

基于实例的学习总结

- 基本概念与最近邻方法
- K近邻方法
  - 基本算法
  - 讨论：更多距离度量；属性：归一化、加权；连续取值目标函数；k 的选择；打破平局；关于效率(K-Dtree的构建与查询)
- 距离加权的KNN
- 基于实例的学习器的四要素
- 扩展：局部加权回归
- 真实测试样例下的算法表现举例
- 懒惰学习与贪婪学习