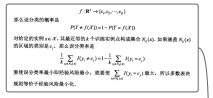
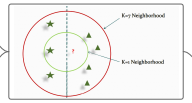
KNN

最邻近的定义

模型

KNN不具有显示的模型
但是具体的过程如下：**最近邻点**取其相同类别，或寻找半径为R内的点，采取**投票表决的方式**产生预测点的类别。方式上都一定的差异，但是本质都是使分类的经验风险最小化





K=1就是最近邻
K> =1就是投票表决方式

远近的度量就要使用距离的概念

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

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

闵氏距离 $L_\infty(x_i,x_j)=\max_l|x_i^{(l)}-x_j^{(l)}|$

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

特征空间中，对每个训练实例点Xi，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). 每个训练实例点拥有 一个单元，所有训练实例点的单元构成对特征空间的一个划分.最近邻法将实例xi的类yi作为其单元中所有点的类标记(class label). 这样，每个单元的实例点的类别是确定的，如图，灰色区域就是'x'类，白色区域就是 '.' 类.

策略

以最近邻为例

通过构造KDTree检索最邻近点

构造kDTree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.

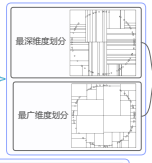
伪代码

KDTree的构造

KDTree为什么能够减少搜索

如图所示，在搜索过程中只需要搜索图中白色区域就行了，其他灰色区域的点不需要进行判断，这样就降低了搜索空间

构造KDTree及其分析



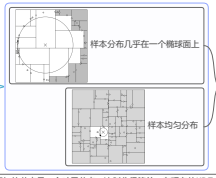
在则两中划分方法中，我们如果使用最深维度划分就会造成生成的二叉树极度不平衡,但是使用最广维度的划分就能使得二叉树的生成平衡一些。

在实际应用中，我们采取某一维度的中位数进行划分，这样的生成的KDTree最接近与平衡二叉树

但是注意，这样生成的二叉树是平衡的，但是搜索时的效率不一定时最高的

KDTree的中位数生成法

样本分布对KDTree的影响



这里可以看出，在样本划分时，如果样本基本处于一个椭球面在KDTree搜索时效率会非常低，基本会接近搜索整个状态空间 但是在样本均匀分布时，这KDTree的搜索效果就会好很多

KDTree的在探究

添加新节点

如果添加的节点是一个叶子节点，这划分很简单，在现有的KDTree基础添加一个叶子节点就好了，但是实际情况往往比这个复杂的多，如果我们通过中位数的方式创建KDTree，原有的KDTree本身就是平衡二叉树，新加入的节点必定会打破原有的平衡，所以在某些情况下将新加入的节点的父节点找到，然后重构该节点下面的所有节点，很明显在查找父节点的过程中还是会浪费大量的时间、空间，在一般情况下就会直接重构KDtee,其时间复杂度就为O(N)

删除节点

与插入新节点类似，删除节点也一定会打破原有的二叉树的平衡，所以在实际应用过程中通常不是直接删除节点，而是使用特殊的标记表示该节点已经被删除了，在删除了一定数量的节点之后我们就能直接重构KDtree

KDTree的查询

先查询左子树（实际上是划分的维度小于根结点的子树）

与根节点比较，判断是都需要遍历另外一颗子树

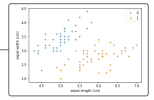
查询另外一颗子树

验证

K近邻

最近邻

使用数据集：sklearn.datasets iris



算法主体

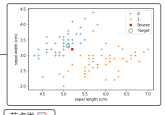
节点类

KDTree类

KD树与暴力搜索的比较

从图中明显能看出，KD树的搜索效率比纯暴力搜索要好

算法结果



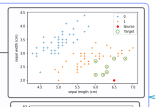
算法结果比较

算法主体

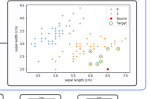
节点类

KSTree类

自己的



sklearn



运行结果相同，说明算法正确

手写识别

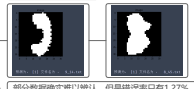
部分数据图片

需要识别的数字已经使用图形处理软件，处理成具有相同的色彩和大小：宽高是32像素x32像素的黑白图像。尽管采用本文格式存储图像不能有效地利用内存空间，但是为了方便理解，在数据存储式任然保存为文本格式，上图为可视化之后的格式

主要代码

验证结果

总共错了12个数据
错误率为1.268499%



部分数据确实难以辨认，但是错误率只有1.27%， 表明分类效果较好