# k-NN

k-近邻算法（k-nearest neighbor, k-NN），一种基本分类与回归方法，理论上比较成熟。从分类模型的构造先后顺序来看，K-NN直至给定一个测试元组才开始构造泛化模型，即属于懒惰学习[1]的范畴，后者也称基于实例的学习。

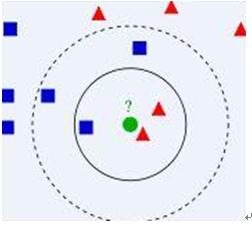


图1.1

该方法的思路是：如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。 KNN方法虽然从原理上也依赖于极限定理，但在类别决策时，只与极少量的相邻样本有关。由于KNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

## K-近邻算法

如图，k=3，就表示我们选择离绿色圆圈最近的3个点来判断，由于红色三角形所占比例为2/3，所以我们认为绿色圆是和红色三角形同类。如果k=5，由于蓝色四方形比例为3/5，因此绿色圆被赋予蓝色四方形类。从这里可以看到，k的值还是很重要的。

算法：

|  |
| --- |
| Input: 训练数据集  T= {（x1,y1），（x2,y2），…，（xN,yN）}  其中，  是实例的特征向量， 为实例的类别，i = 1,2,…,N;实例特征向量x;  Output: 实例x的所属类别   1. 根据所给定的的距离度量，在训练集T中找出与x最邻近的k个点，涵盖了这 k个点的x邻域记做； 2. 在中根据分类决策规则（如多数表决）决定x的类别y：   （1.1）  上式中，I为指示函数，当时I为1，否则I为0.  k-NN的特殊情况是k=1的情形，当k=1 时称为最邻近算法.对于输入的实例点（特征向量）x，最近邻算法将训练数据集中与x最邻近的点作为x的类. |

k-NN算法没有显示的学习过程.

## K-近邻模型

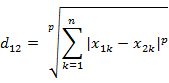
### 2.1 距离度量

我们看到，K近邻算法的核心在于找到实例点的邻居，这个时候，问题就接踵而至了，如何找到邻居，邻居的判定标准是什么，用什么来度量。这一系列问题便是下面要讲的距离度量表示法。下面就来具体阐述下都有哪些距离度量的表示法，权当扩展。

2.1.1 **闵可夫斯基距离(Minkowski Distance)**

闵氏距离不是一种距离，而是一组距离的定义。

两个n维变量a(x11,x12,…,x1n)与 b(x21,x22,…,x2n)间的闵可夫斯基距离定义为：



其中p是一个变参数。当p=1时，就是曼哈顿距离；当p=2时，就是欧氏距离；当p→∞时，就是切比雪夫距离

根据变参数的不同，闵氏距离可以表示一类的距离。

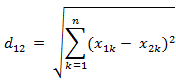
2.1.2  **欧氏距离**

* 最常见的两点之间或多点之间的距离表示法，又称之为欧几里得度量，它定义于欧几里得空间中，如点 x = (x1,...,xn) 和 y = (y1,...,yn) 之间的距离为：http://img.my.csdn.net/uploads/201211/20/1353398777_7638.png

(1)二维平面上两点a(x1,y1)与b(x2,y2)间的欧氏距离：

http://img.my.csdn.net/uploads/201211/20/1353399552_4107.png

(2)两个n维向量a(x11,x12,…,x1n)与 b(x21,x22,…,x2n)间的欧氏距离：



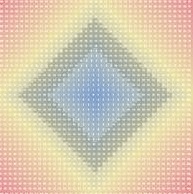
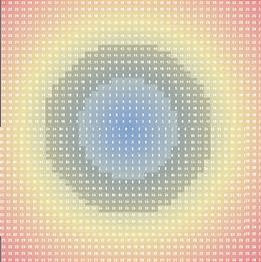
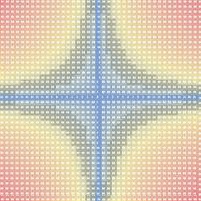
　　也可以用表示成向量运算的形式：

http://img.my.csdn.net/uploads/201211/20/1353399664_2255.png

2.1.3 **曼哈顿距离**

两个n维向量a(x11,x12,…,x1n)与 b(x21,x22,…,x2n)间的曼哈顿距离

http://img.my.csdn.net/uploads/201211/20/1353399924_2304.png



2.1.4 **切比雪夫距离**

(1)二维平面两点a(x1,y1)与b(x2,y2)间的切比雪夫距离

http://img.my.csdn.net/uploads/201211/20/1353400142_8660.png

(2)两个n维向量a(x11,x12,…,x1n)与 b(x21,x22,…,x2n)间的切比雪夫距离

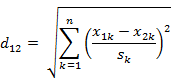
http://img.my.csdn.net/uploads/201211/20/1353400159_5706.png

这个公式的另一种等价形式是

http://img.my.csdn.net/uploads/201211/20/1353400175_3976.png

2.1.5 **标准化欧氏距离 (Standardized Euclidean distance )**

经过简单的推导就可以得到两个n维向量a(x11,x12,…,x1n)与 b(x21,x22,…,x2n)间的标准化欧氏距离的公式：



如果将方差的倒数看成是一个权重，这个公式可以看成是一种加权欧氏距离(Weighted Euclidean distance)。

**2.1.6 马氏距离(Mahalanobis Distance)**

有M个样本向量X1~Xm，[协方差矩阵](http://zh.wikipedia.org/wiki/%E5%8D%8F%E6%96%B9%E5%B7%AE%E7%9F%A9%E9%98%B5" \t "_blank)记为S，均值记为向量μ，则其中样本向量X到u的马氏距离表示为：

http://img.my.csdn.net/uploads/201211/21/1353469107_5540.png

（协方差矩阵中每个元素是各个矢量元素之间的协方差Cov(X,Y)，Cov(X,Y) = E{ [X-E(X)] [Y-E(Y)]}，其中E为数学期望）

而其中向量Xi与Xj之间的马氏距离定义为：

http://img.my.csdn.net/uploads/201211/21/1353469122_3539.png

若协方差矩阵是单位矩阵（各个样本向量之间独立同分布）,则公式就成了：

http://img.my.csdn.net/uploads/201211/21/1353469133_1179.png

若协方差矩阵是对角矩阵，公式变成了标准化欧氏距离。

**2.1.7 夹角余弦(Cosine)**

2.1.9 **汉明距离(Hamming distance)**

### 2.2 k值

如果选择较小的K值，就相当于用较小的领域中的训练实例进行预测，“学习”近似误差会减小，只有与输入实例较近或相似的训练实例才会对预测结果起作用，与此同时带来的问题是“学习”的估计误差会增大，换句话说，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，就相当于用较大领域中的训练实例进行预测，其优点是可以减少学习的估计误差，但缺点是学习的近似误差会增大。这时候，与输入实例较远（不相似的）训练实例也会对预测器作用，使预测发生错误，且K值的增大就意味着整体的模型变得简单。

K=N，则完全不足取，因为此时无论输入实例是什么，都只是简单的预测它属于在训练实例中最多的类，模型过于简单，忽略了训练实例中大量有用信息。

### 2.3 分类决策规则

DVF决策规则，经典

即寻求距离测试实例最近的k个样本点中，所属类别最多的那个类为测试实例的类别。

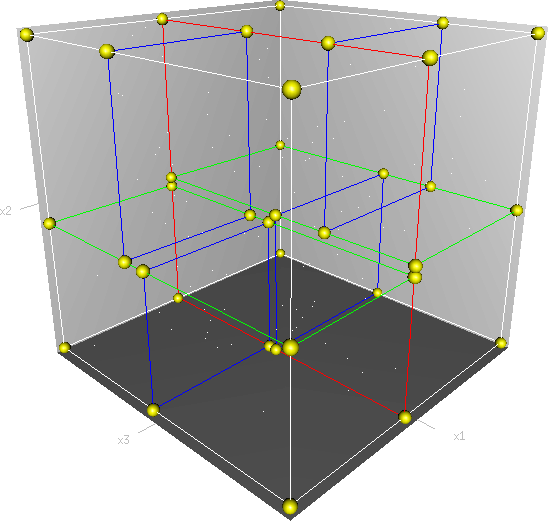
SWF决策规则，使用广泛

距离测试实例最近的K个样本点中，计算同一个类别的所有点到测试实例点的平均距离，并将平均距离最小的类当做测试实例的类。

## K-近邻实现----KD树

Kd-树是K-dimension tree的缩写，是对数据点在k维空间（如二维(x，y)，三维(x，y，z)，k维(x1，y，z..)）中划分的一种数据结构，主要应用于多维空间关键数据的搜索（如：范围搜索和最近邻搜索）。本质上说，Kd-树就是一种平衡二叉树。

首先必须搞清楚的是，k-d树是一种空间划分树，说白了，就是把整个空间划分为特定的几个部分，然后在特定空间的部分内进行相关搜索操作。想像一个三维(多维有点为难你的想象力了)空间，kd树按照一定的划分规则把这个三维空间划分了多个空间，如下图所示：



3.1 KD树的建立

建Kd-树的伪码为：

算法：构建Kd-tree

输入：数据点集Data\_Set，和其所在的空间。

输出：Kd，类型为Kd-tree

1 if data-set is null ,return 空的Kd-tree

2 调用节点生成程序

（1）确定split域：对于所有描述子数据（特征矢量），统计他们在每个维度上的数据方差，挑选出方差中最大值，对应的维就是split域的值。数据方差大说明沿该坐标轴方向上数据点分散的比较开。这个方向上，进行数据分割可以获得最好的分辨率。

（2）确定Node-Data域，数据点集Data-Set按照第split维的值排序，位于正中间的那个数据点 被选为Node-Data，Data-Set` =Data-Set\Node-data

3 dataleft = {d 属于Data-Set` & d[:split]<=Node-data[:split]}

Left-Range ={Range && dataleft}

dataright = {d 属于Data-Set` & d[:split]>Node-data[:split]}

Right-Range ={Range && dataright}

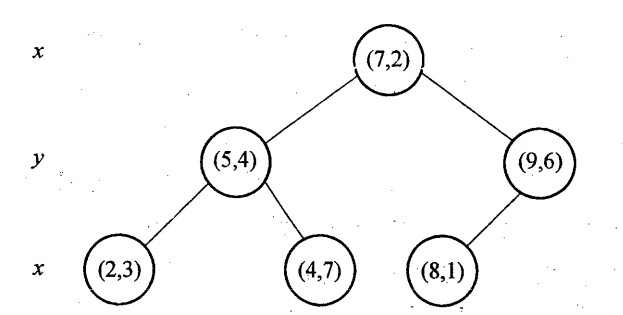
4 :left =由（dataleft,LeftRange）建立的Kd-tree

设置:left的parent域（父节点）为Kd

：right =由（dataright,RightRange）建立的Kd-tree

设置：right的parent域为kd。

这个构建过程是一个递归过程。重复上述过程，直至只包含一个节点。



如图，（1）确定：split 域=x，6个数据点在x,y 维度上的数据方差为39,28.63.在x轴方向上的方差大，所以split域值为x。

（2）确定：Node-Data=（7,2），根据x维上的值将数据排序，6个数据的中值为7，所以node-data域为数据点（7,2）。这样该节点的分割超面就是通过（7,2）并垂直于：split=x轴的直线x=7.

(3)左子空间和右子空间，分割超面x=7将整个空间分为两部分。x<=7 为左子空间，包含节点（2,3），（5,4），（4，7），另一部分为右子空间。包含节点（9,6），（8,1）

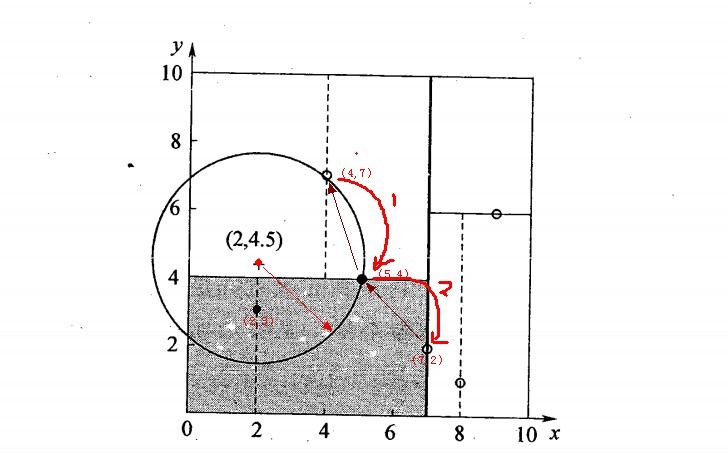
这个构建过程是一个递归过程。重复上述过程，直至只包含一个节点。

3.2 KD树的搜索

|  |
| --- |
| *输入*：以构造的kd树，目标点x； *输出*：x 的最近邻 *算法步骤如下*：   1. 在kd树中找出包含目标点x的叶结点：从根结点出发，递归地向下搜索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. 递归的向上回溯，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，则更新“当前最近点”，也就是说以该实例点为“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。具体做法是，检查另一子结点对应的区域是否以目标点位球心，以目标点与“当前最近点”间的距离为半径的圆或超球体相交： 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，继续递归地进行最近邻搜索； 如果不相交，向上回溯。 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。 |

 先给一个简单的例子，比如要查找点（2，4.5）的最近邻点，具体步骤依次如下：

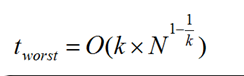
1. 先进行二叉查找，从根节点开始，比较目标点与当前结点在当前结点split维度上的大小，于是，先从（7,2）查找到（5,4）节点，此时由当前结点（5,4）的split为y, y = 4为分割超平面的，由于目标点为y值为4.5，因此进入右子空间查找到（4,7），形成搜索路径<(7,2)，(5,4)，(4,7)>，但 （4,7）与目标查找点的距离为3.202，而（5,4）与查找点之间的距离为3.041，所以（5,4）为查询点的最近点；
2. 以（2，4.5）为圆心，以3.041为半径作圆，如下图所示。可见该圆和y = 4超平面交割，所以需要进入（5,4）左子空间进行查找，也就是将（2,3）节点加入搜索路径中得<(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；



1. 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。

然而，在实际的应用中，如SIFT特征矢量128维，SURF特征矢量64维，维度都比较大，直接利用k-d树快速检索的性能急剧下降，几乎接近贪婪线性扫描。假设数据集的维数为D，一般来说要求数据的规模N满足N»2D，才能达到高效的搜索。所以这就引出了一系列对k-d树算法的改进：BBF算法，和一系列M树、VP树、MVP树等高维空间索引树。

**最坏的情况下搜索N个结点的k维kd-tree所花费的时间为：**



3.3 BBF算法

|  |
| --- |
| 算法： BBF最近邻查询 |
| 输入： Kd, /\* \*Kd-tree类型\*/  target, /\*查询目标点\*/ |
| 输出： Nearest, /\*最近邻数据点\*/  Dist, /\*最近邻和目标点的距离\*/ |
| 1. If Kd == NULL   Dist = INF;   1. Nearest = Kd.Node->data;   将 &Kd压入priority\_list优先级堆栈中；  /\*建立优先级队列，首先压入根节点，优先级队列中记录的都是Kd树结点，他们都是需要回溯的树节点，回溯这些树结点的优先级取决于它们离查询点的距离，距离越近，优先级越高\*/  While(priority\_list is not null)  /\* 优先检查这个树节点表示的空间中是否有更好的最近邻\*/  提取优先级最高的结点的结点top\_Kd;  Kd\_point = top\_Kd;  While(Kd\_point is not null)  s = Kd\_point ->split;  If (target[s]<=Kd\_point->Node\_data[s])  Current\_data = Kd\_point->Node\_data;  将Kd\_data->right按照优先级插入priority\_list中  Kd\_point = Kd\_point->left;  Else  Current\_data = Kd\_point ->Node\_data;  将Kd\_point->left按照优先级插入到priority\_list中  Kd\_point = Kd\_point->right;  If Distance(Nearest, target) > Distance(current\_data, target)  Nearst = current\_data;  Max\_dist = distance(current\_data, target);  EndWhile  EndWhile |

算法优缺点  
1、优点  
简单，易于理解，易于实现，无需估计参数，无需训练  
适合对稀有事件进行分类（例如当流失率很低时，比如低于0.5%，构造流失预测模型）  
特别适合于多分类问题(multi-modal,对象具有多个类别标签)，例如根据基因特征来判断其功能分类，kNN比SVM的表现要好  
2、缺点  
懒惰算法，对测试样本分类时的计算量大，内存开销大，评分慢  
可解释性较差，无法给出决策树那样的规则。