k近邻算法（KNN）及kd树简介（KD-Tree）

2016年04月24日 17:40:19 标签：k近邻算法 / KNN / kd树 / kd-tree / 机器学习 5380

在使用k近邻法进行分类时，对新的实例，根据其k个最近邻的训练实例的类别，通过**多数表决**的方式进行预测。由于模型的特征空间一般是n维实数向量，所以距离的计算通常采用的是欧式距离。关键的是k值的选取，如果k值**太小**就意味着整体模型变得复杂，容易**发生过拟合**，即如果邻近的实例点恰巧是噪声，预测就会出错，极端的情况是k=1，称为算法，对于待预测点x，与x最近的点决定了x的类别。k值得增大意味着整体的模型变得简单，极端的情况是k=N，那么无论输入实例是什么，都简单地预测它属于训练集中最多的类，这样的模型过于简单。经验是，k值一般去一个比较小通常采取**交叉验证的方法来选取最优的k值**。

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，这点在特征空间的维数大以及训练数据容量大时尤其重要。k近邻法的最简单实现是线性扫描，这时要计算输入实例与每一个训练实例的距离，当训练集很大时，计算时，这种方法是不可行的。为了提高k近邻搜索的效率，**可以考虑使用特殊的结构存储训练数据**，以减少计算距离的次数。具体方法有很多，这里介绍kd树方法。

1.实例

先以一个简单直观的实例来介绍k-d树算法。假设有6个二维数据点{（2,3），（5,4），（9,6），（4,7），（8,1），（7,2）}，数据点位于二维空间内（如图2中黑点所示）。k-d树算法就是要确定图2中这些分割空间的分割线（多维空间即为分割平面，一般为超平面）。下面就要通过一步步展示k-d树是如何确定这些分割线的。

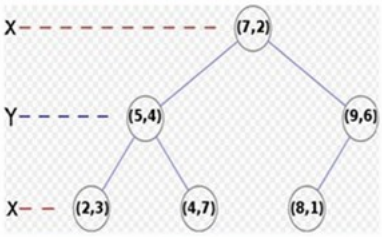


图 1. 2D KD-tree

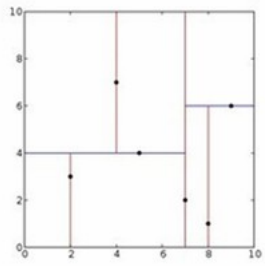


图 2. 2D KD-tree 对应的平面的切割

k-d树算法可以分为**两大部分**，一部分是有关k-d树本身这种数据结构**建立**的算法，另一部分是在建立的k-d树上如何进行最邻近**查找**的算法。

2.构造kd树

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搜索的树形数据结构。**kd树是二叉树**，表示对k维空间的一个划分。构造kd树相当于不断地用垂直于坐标轴的超平面将k维空间进行切分，构成一系列的k维超矩形区域。kd树的每一个节点对应于一个k维超矩形区域。k-d树是一个二叉树，每个节点表示一个空间范围。下表给出的是k-d树每个节点中主要包含的数据结构。

域名	数据类型	描述
Node-data	数据矢量	数据集中某个数据点，是n维矢量（这里也就是k维）
Range	空间矢量	该节点所代表的空间范围
split	整数	垂直于分割超平面的方向轴序号
Left	k-d树	由位于该节点分割超平面左子空间内所有数据点所构成的k-d树
Right	k-d树	由位于该节点分割超平面右子空间内所有数据点所构成的k-d树
parent	k-d树	父节点

从上面对k-d树节点的数据类型的描述可以看出构建k-d树是一个逐级展开的递归过程。下面给出的是构建k-d树的伪码。

输出：Kd，类型为k-d tree
1.If Data-set为空，则返回空的k-d 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），统计它们在每个维上的数据方差。假设每条数据记录为64维，可计算64个方差。挑选出最大值，对应的维就是split域的值。数据方差大表明沿该坐标轴方向上的数据分散得比较开，在这个方向上进行数据分割有较好的分辨率；
（2）确定Node-data域：数据点集Data-set按其第split域的值排序。位于正中间的那个数据点被选为Node-data。此时新的Data-set' = Data-set \ Node-data（除去其中Node-data这一点）。
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，Left_Range）建立的k-d tree，即递归调用createKDTree（dataleft，Left_Range）。并设置left的parent域为Kd；
right = 由（dataright，Right_Range）建立的k-d tree，即调用createKDTree（dataleft，Left_Range）。并设置right的parent域为Kd。

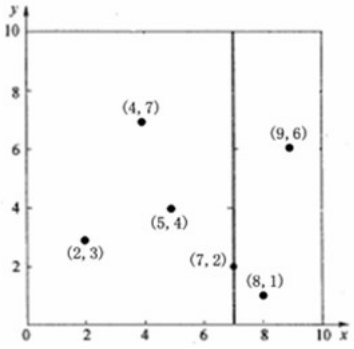
以上述举的实例来看，过程如下：

由于此例简单，数据维度只有2维，所以可以简单地给x，y两个方向轴编号为0,1，也即split={0,1}。

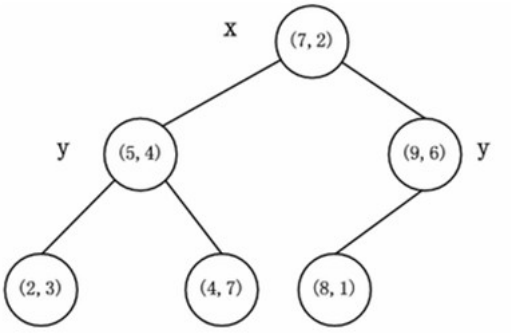
（1）**确定split域的首先该取的值。分别计算x，y方向上数据的方差得知x方向上的方差最大，所以split域值首先取0，也就是x轴方向**；

（2）确定Node-data的域值。根据x轴方向的值2,5,9,4,8,7排序选出中值为7，所以Node-data = （7,2）。这样，该节点的分割超平面就是通过（7,2）并垂直于split = 0（x轴）的直线x = 7；

（3）确定左子空间和右子空间。分割超平面x = 7将整个空间分为两部分，如下图所示。x <= 7的部分为左子空间，包含3个节点{（2,3），（5,4），（4,7）}；另一部分为右子空间，包含2个节点{（9,6），（8,1）}。



如算法所述，k-d树的构建是一个递归的过程。然后对左子空间和右子空间内的数据重复根节点的过程就可以得到下一级子节点（5,4）和（9,6）（也就是左右子空间的'根'节点），同时将空间和数据集进一步细分。如此反复直到空间中只包含一个数据点，如下图所示。最后生成的k-d树如下图所示。

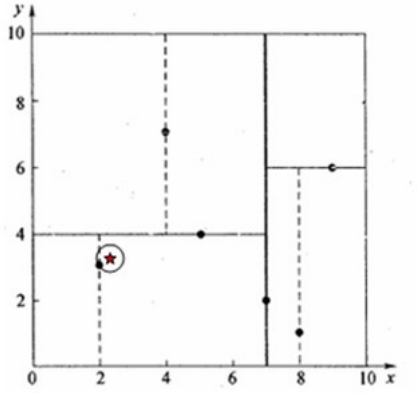


3.搜索kd树

在k-d树中进行数据的查找也是特征匹配的重要环节，其目的是检索在k-d树中与查询点距离最近的数据点。这里先以一个简单的实例来描述最邻近查找的基本思路。

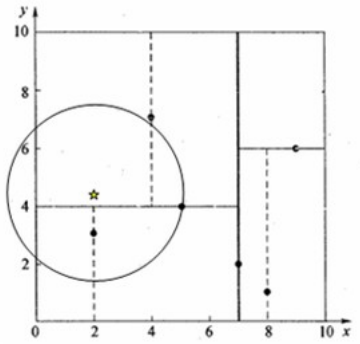
星号表示要查询的点（2.1,3.1）。**通过二叉搜索，顺着搜索路径很快就能找到最邻近的近似点**，也就是叶子节点（2,3）。而找到的叶子节点并不一定就是最邻近的，**最邻近**肯定距离查询点更近，**应该位于以查询点为圆心且通过叶子节点的圆域内**。为了找到真正的最近邻，还需要进行'回溯'操作：算法沿搜索路径反向查找是否有距离查询点更近的数据点。此例中先从（7,2）点开始进行二叉查找，然后到达（5,4），最后到达（2,3），此时搜索路径中的节点为小于（7,2）和（5,

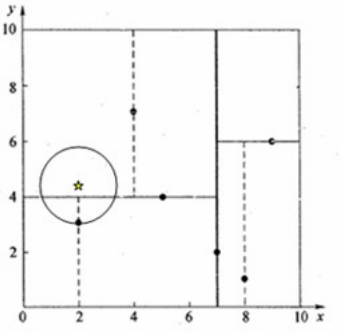
（5,4），并判断在该父节点的其他子节点空间中是否有距离查询点更近的数据点。以（2.1,3.1）为圆心，以0.1414为半径画圆，如下图所示。发现该圆并不和超平面y = 4交割，因此不用进入（5,4）节点右子空间中去搜索。



再回溯到（7,2），以（2.1,3.1）为圆心，以0.1414为半径的圆更不会与x = 7超平面交割，因此不用进入（7,2）右子空间进行查找。至此，搜索路径中的节点已经全部回溯完，结束整个搜索，返回最近邻点（2,3），最近距离为0.1414。

一个复杂点了例子如查找点为（2，4.5）。同样先进行二叉查找，先从（7,2）查找到（5,4）节点，在进行查找时是由y = 4为分割超平面的，由于查找点为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）>，取（4,7）为当前最近邻点，计算其与目标查找点的距离为3.202。然后回溯到（5,4），计算其与查找点之间的距离为3.041。以（2，4.5）为圆心，以3.041为半径作圆，如下图左所示。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。此时需将（2,3）节点加入搜索路径中得<（7,2），（2,3）>。回溯至（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。回溯至（7,2），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，如下图右所示。至此，搜索路径回溯完。返回最近邻点（2,3），最近距离1.5。





k-d树查询算法的伪代码如下所示。

```
[cpp]
算法：k-d树最邻近查找
输入：Kd，     //k-d tree类型
     target  //查询数据点
输出：nearest，  //最邻近数据点
     dist      //最邻近数据点和查询点间的距离
1. If Kd为NULL，则设dist为infinite并返回
2. //进行二叉查找，生成搜索路径
   Kd_point = &Kd;                    //Kd-point中保存k-d tree根节点地址
   nearest = Kd_point -> Node-data;   //初始化最近邻点
   while（Kd_point）
      push（Kd_point）到search_path中;  //search_path是一个堆栈结构，存储着搜索路径节点指针
 /*** If Dist（nearest，target） > Dist（Kd_point -> Node-data，target）
          nearest  = Kd_point -> Node-data;     //更新最近邻点
          Max_dist = Dist(Kd_point，target）;   //更新最近邻点与查询点间的距离  ***/
       s = Kd_point -> split;                        //确定待分割的方向
       If target[s] <= Kd_point -> Node-data[s]     //进行二叉查找
          Kd_point = Kd_point -> left;
       else
          Kd_point = Kd_point ->right;
   nearest = search_path中最后一个叶子节点;  //注意：二叉搜索时不比计算选择搜索路径中的最邻近点，这部分已被注释
   Max_dist = Dist（nearest，target）;     //直接取最后叶子节点作为回溯前的初始最近邻点
```

```
s = back_point -> split;                    //确定分割方向
If Dist（target[s]，back_point -> Node-data[s]） < Max_dist   //判断还需进入的子空间
    If target[s] <= back_point -> Node-data[s]
        Kd_point = back_point -> right;   //如果target位于左子空间，就应进入右子空间
    else
        Kd_point = back_point -> left;    //如果target位于右子空间，就应进入左子空间
    将Kd_point压入search_path堆栈;
If Dist（nearest，target） > Dist（Kd_Point -> Node-data，target）
    nearest  = Kd_point -> Node-data;                  //更新最近邻点
    Min_dist = Dist（Kd_point -> Node-data,target）;   //更新最近邻点与查询点间的距离
```

3

维数较大时，直接利用k-d树快速检索的性能急剧下降。假设数据集的维数为D，一般来说要求数据的规模N满足条远大于2的D次方，才能达到高效的搜索。

转目：http://blog.csdn.net/qll125596718/article/details/8426458

olzhy 2017-12-30 23:45 回复 1楼

关于kd树 看看这篇文章吧 说的非常详细 还有代码实现 https://leileiluoluo.com/posts/kdtree-algorithm-and-implementation.html

k ee : k近邻查询和范围查询

Dylan_Frank 2017年09月11日 16:07 932

想象一下我们有如下两个任务: 我现在想骑一辆小黄车，我想查找离我最近的k辆小黄车. 找到百度地图中显示在屏幕上区域中的所有酒店 这两个任务均可以用kd-tree来解决 kd-tree 主要两个用途:...

k ee小结

u012288458 2015年05月23日 21:59 2943

来省队集训被吊打，于是无聊学了一下kd-tree，挺好玩的东西。 kd-tree是一种支持查询平面最近点（其实是k维，但这种题目 ）的数据结构，单次理论时间复杂度O（√n），但实际是非常快的，可以...

Kd-tree的用法

qq_33690156 2016年09月06日 19:46 375

Kd-Tree算法原理和开源实现代码 本文介绍一种用于高维空间中的快速最近邻和近似最近邻查找技术——Kd-Tree（Kd树）。Kd-Tree，即K-dimensional tree，是...

kD-tree 的C语言实现 带有史上最全的注释和解释

原文转自CSDN http://blog.csdn.net/daringpig/article/details/7652891 语言cstructtreefloatinsert ...

sherry_0009 2013年06月23日 09:32 1047

kdtree,kd-tree

2008年04月15日 18:54 465KB 下载

RAR

kD-tree 的C语言实现 带有史上最全的注释和解释

kdtree的原理就是基于二叉树的形式，将高维空间用超矩形进行划分.其主要用途是用来求解高维空间中最近邻的值. 下面是kdtree.
h文件，是kdtree数据结构的头文件 #i...

daringpig 2012年06月11日 16:13 17134

K相邻和Kd tree

lanbing510 2013年01月28日 15:18 6593

动机 先前写了一篇文章《SIFT算法研究》讲了讲SIFT特征具体是如何检测和描述的，其中也提到了SIFT常见的一个用途就是物体识别，物体识别的过程如下图所示： 如上图(a)，我们先...

kd-tree实现KNN

so__sunshine 2016年09月21日 21:48 2211

转载：http://www.cnblogs.com/v-July-v/archive/2012/11/20/3125419.html 从K近邻算法、距离度量谈到KD树、...

OpenCV——KD Tree（介绍完整的flann邻近搜索）

关于OpenCV的FLANN库中KD-Tree搜索的文章并不多。 FLANN搜索的流程包括索引的建立和搜索。而所谓kd-tree搜索，指的是在建立索引这一步骤中建立的是kd-tree索引。 本文将结合...

ErenTuring 2017年06月10日 12:14 2166

KD-Tree 总结

u010697167 2014年03月18日 17:53 1959

hdu 2966 In case of failure 求每个点的最短距离点的距离。直接建KD树，然后查询最近点距离就可以了。此题是比着点击打开链接写的，可以用另一种不用删除和插入的写法。 /...

kd树简介 在matlab下VLFeat中的kd-tree使用

damant 2016年03月17日 21:51 4884

先简要介绍knn——K近邻算法和kd-tree——kd树，然后介绍matlab
环境中有关使用kd树的函数。k-d树（k-dimensional树的简称），是一种分割k维数据空间的数据结构。主要应用于多...

kd-tree in Python

lipengcn 2016年03月20日 20:33 2501

kd-tree改进 in Python

kd-tree c语言代码

u012637838 2014年07月26日 11:34 1467

/* This file is part of ``kdtree'', a library for working with kd-trees. Copyright (C) 2007-2009 Joh...

从K近邻算法、距离度量谈到KD树、SIFT+BBF算法

从K近邻算法、距离度量谈到KD树、SIFT+BBF算法前言　前两日，在微博上说：“到今天为止，我至少亏欠了3篇文章待写：1、KD树；2、神经网络；3、编程艺术第28章。你看到，blog内的文章与...

v_JULY_v 2012年11月20日 16:31 159332

KNN（三）--KD树详解及KD树最近邻算法

k-d树搜索最近点，在opencv中使用FLANN算法，，其包含：1：建树　2.查询 程序见下：　#include "kdtree.h" #include #include　#includ...

App_12062011 2016年07月21日 21:24 14359