知乎

KD树简介

ARGO创新实验室
数据科学与统计的深度思考者，狮马帮ECU终身学习者

作者 周知航

审核 张翔

ARGO曾经推出了

ARGO：KNN算法介绍
32 赞同 · 1 评论 文章

介绍了KNN算法的原理，其核心是寻找待测样本在训练样本中的k个近邻，从而从k个近邻中获取数量最多的类别作为待测样本的类别。

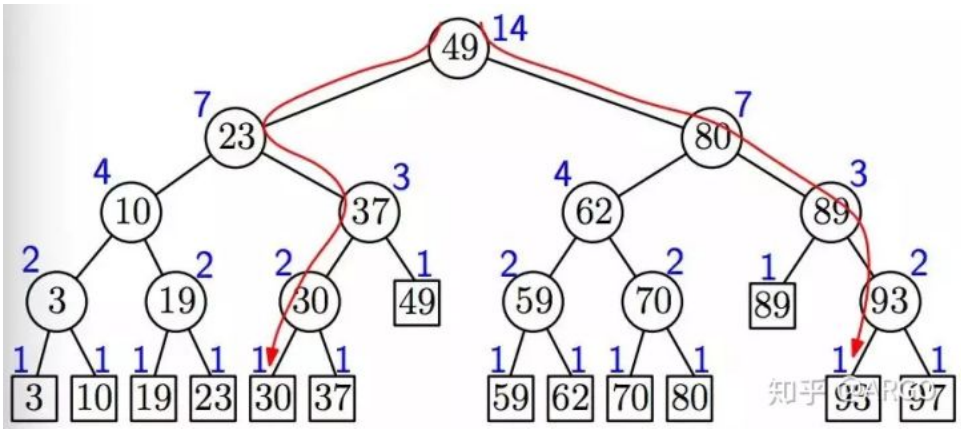
KNN算法是机器学习中最简单的算法之一，但是工程实现上，如果训练样本过大，则传统的遍历全样本寻找k近邻的方式将导致性能的急剧下降。 因此，为了优化效率，不同的训练数据存储结构被纳入到实现方式之中。在sikit-learn中的KNN算法参数也提供了'kd_tree'之类的算法选择项。本文将从如下几个方面阐述kd树：

- kd树的概念
- 从例子理解kd树的构建及查找
- ball tree 和其他树类型简介

一 kd树的概念

kd（k-dimensional）树的概念自1975年提出，试图解决的是在k维空间为数据集建立索引的问题。依上文所述，已知样本空间如何快速查询得到其近邻？唯有以空间换时间，**建立索引**便是计算机世界的解决之道。但是索引建立的方式各有不同，kd树只是是其中一种。它的思想如同分治法，即：**利用已有数据对k维空间进行切分**。

我们先回顾一下二叉查找树，这也是在一维空间中的kd树的情形：



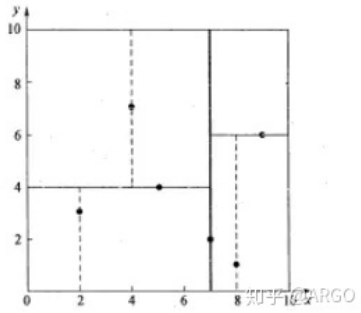
由图可知，二叉树在时间复杂度上是O(logN），远远优于全遍历算法。对于该树，可以在空间上理解：树的每个节点把对应父节点切成的空间再切分，从而形成各个不同的子空间。查找某点的所在位置时，就变成了查找点所在子空间。上图仅仅是一维，如果换到二维，或者是更高维度上，这棵树该怎么建立？ 又该怎么理解所切成的子空间呢？

我们可以用一个在网络上较为普遍的例子来演示一下二维的kd树构建及查找方式。

二 从例子理解kd树的构建及查找

假设有6个二维数据点{(2,3)，(5,4)，(9,6)，(4,7)，(8,1)，(7,2)}。

依据上文，对于一维空间的切分的最终结果实际上切出的是线段，同理，二维空间中，对于该6个点集最后的切分结果的空间应该是子平面集合。这里，kd树最终目标就是构建一棵树，能够将二维平面切分得到的最终的子平面集，如下图所示：



切分后的空间示意图

找到切分这些平面的直线便是建立kd树的重点。我们将先介绍这种构建方式，然后再介绍基于kd树的最近邻查找的查询方法。

构建

将二维的平面想像成一块方型蛋糕，kd树构建就是面点师要将蛋糕切成上面示意图的模样。先将平面上的六个点在蛋糕上做好标记。面点师举起刀，他该如何下刀呢？这就涉及到kd树在切分时要注意两个要点，**切分域和切分点的选择**。

只要明确了这两个要点的实施规则，切分蛋糕就变得十分容易。因为对于子平面的切分，依然可以按照这些规则递归进行。

a 切分域的选择

切分（split）域，即切分的维度方向，在这个例子中，也就是在二维平面上，面点师必须决定，我这第一刀是平行于x轴横着切还是平行于y轴竖着切？作为面点师，最害怕的情况就是切出的子平面

切分域。 方差较大，表明在该维度上的点的分散度较高，按该维度切分分辨率比较高。

在这里，**一轮**是指的必须依照k个维度，对每个维度对平面进行一次切分。比如面点师如果第一刀选择了切分X轴，则第二刀必须选择Y轴。如果多余两个维度，则接下来依然按照剩下维度的方差值进行计算来选择切分域。直至遍历所有维度，一轮结束。

计算第一次切分时两个维度上的方差，得到

$$\bar{x} = (2+5+9+4+8+7)/6 = 5.83, D(x) = \sum_{k=1}^{6}(x_k - \bar{x})^2 = 34.83$$

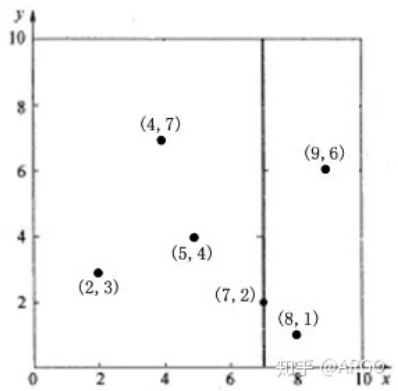
$$\bar{y} = (3+4+6+7+1+2)/6 = 3.83, D(y) = \sum_{k=1}^{6}(y_k - \bar{y})^2 = 26.71$$

在X轴的方向方差较大，所以面点师选择首先切分X轴（也就是在二维平面中沿着Y轴方向）。

b 切分点的选择

切分点的选择策略比较简单，是将带切分平面上的所有数据点按照切分域的维度进行大小排序，选择**正中间的点**作为切分点。如果正中间点是偶数个数，则任取中间左边或者右边的值作为切分点。所以，对上述X轴的点位进行排序，（5，4），（7，2）是X轴方向的中间点。这里选择（7，2）作为第一次的切分点。

根据已经选择的切分点和切分域，面点师终于畅快的下了第一刀，如下所示。

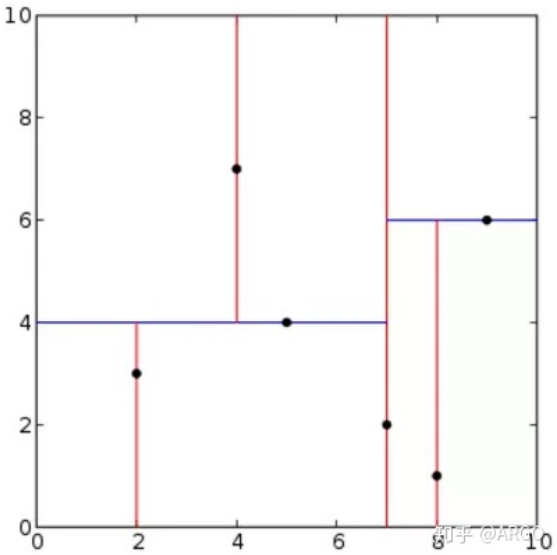


第一次切分后的结果

依照切分点进行切分后，当前切分点将从子平面的数据集中排除。

本轮并未就此结束，在X轴方向进行切分后将继续对Y方向进行切分。由于面点师第一刀把方形平面切成了两个子平面，对Y轴的切分自然需要对两个子平面各切一刀。这里再次强调，对于二维情况，Y是最后本轮最后维度，不需要再计算方差得到切分域，但是对于多维情况，此次计算是排除第一次切分域的维度后剩下维度在子平面中计算分列域下选择的方差最大的维度。

选择对Y方向上切分的两个子平面的切分点，分别选择（5,4）（也可以是（2,3））（9, 6）（也可以是8,1）。按照切分点和切分域下刀，如下图所示的两条蓝色线条，则是面点师的下刀痕迹。



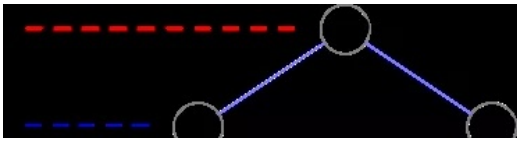
已切好的平面

在第二轮开始后，由于右上角的平面数据点数为0，不用再切分。而剩下的平面中数据点数都为1。因为此时方差相同，计算方差选择切分域没有意义，使用对X轴进行切分，如上上图的红线所示。

以上只是面点师作为kd树构建时的指导思想，在计算机中该算法究竟该如何表示？接下来看看完成对上述分割时建立的kd树的数据结构。

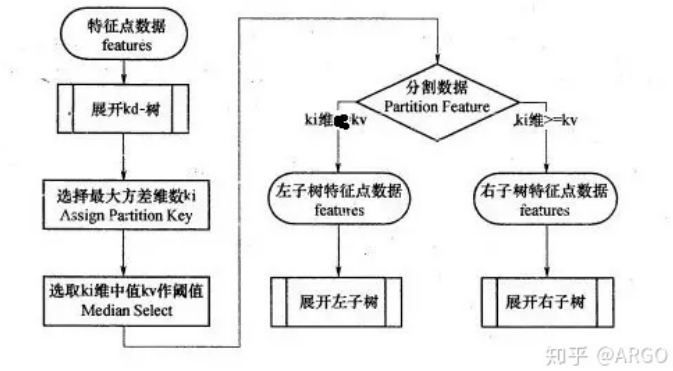
d 树形数据结构表示

如图所示，这是对应切分方法应该建立的树。第一个根节点（7, 2）是第一次的切分点，切分的是X方向。也就是树的左子树的X轴方向都比7小，右子树X方向点都比7大。再看下一级（5,4），（9,6）点切分的是Y轴方向，也就是对于（5,4）为根的子树，左子树Y轴方向都比4小，右子树都比4大。以此类推。



值得注意的是，在kd树的实现过程中，切分域的选择并不一定采用方差选择方式，而可能只是简单的以 d mod n 维度的方式，来决定该层树的切分域（d代表树节点的深度，n代表维度数量，求模来确定切分域）。这样实现简单，效果也一般不会太差。

总结一下kd树的算法流程如下



在构建完kd树之后，就是kd树的应用了。如何使用kd树找到目标点的最近邻，kd树是如何帮助实现效率的提升的呢，这时候就得观察一下kd树是如何搜索最近邻的了。

查找

回到面点师切好的糕点平面图，用目标数据在kd树中寻找最近邻时，最核心的两个部分是：

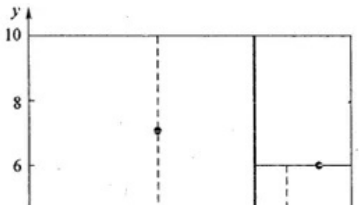
1 寻找近似点-寻找最近邻的叶子节点作为目标数据的近似最近点。

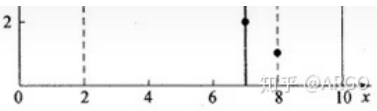
2 回溯-以目标数据和最近邻的近似点的距离沿树根部进行回溯和迭代。

以两个例子来描述这一过程。例子来源：

cnblogs.com/eyeszjwang/...

例子1 搜索点（2.1,3.1）



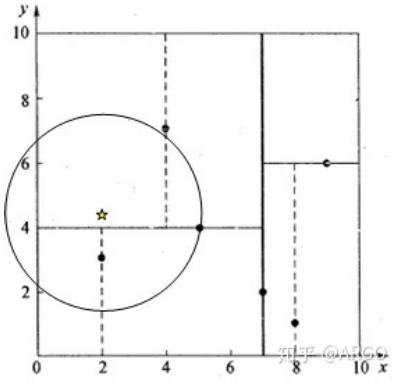


计算近似最近点--很显然，通过查找寻找kd树，很容易定位到该点将落在点(2,3)所在的叶子节点上。 此时，将（2,3）作为该点的近似最近点。计算出到该近似点的距离为0.141。

为了理解方便，将（2,3）到（2.1,3.1）的距离为直径，以（2,3）为圆心作圆。

回溯--在根据目标节点寻找近似最近点时，其路径是(7,2)->(5,4)->(2,3)。 先计算该点与近似最近点的父节点（5,4）的距离，看看是否该距离是否小于0.141，（5,4）点的距离大于0.141，说明被(5,4) 切分的另外一个子平面与（2,3）点为圆心的圆没有交集。但不能排除跟（7,2）所切开的右子平面没有交集，所以再往上回溯（7,2）点，直至确认跟右子平面没有交集，回溯结束，确认最近邻点位（2,3）。

例子2 搜索点（2, 4.5）



重复上述搜索过程。

计算近似最近点--其落在叶子节点（4,7）点上。近似距离为3.20。

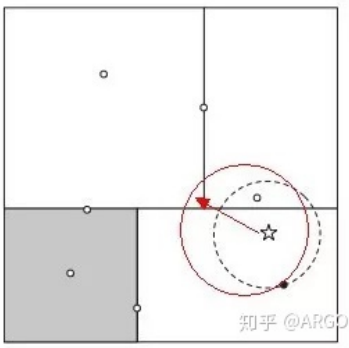
回溯--对于该点的回溯过程就会比刚才复杂一些。其父节点（5,4）与目标点的距离为3.04，小于3.20。 这就意味着（4,7）点作为最近近似点的假设不成立。以（5,4）作为最近近似点。以3.04做圆，圆是跟（5,4）所切分的上下两个平面相交的，所以需要检查（5,4）的另外一个子树的叶子节点（2,3）是否可以作为近似最近点。发现（2，3）的距离为1.5小于3.04，更新（2,3）为近似最近点。最后回溯至（7,2），确认跟（7,2）切分的右子平面无关。 回溯结束，（2,3）为其最近点。

以上两例说明了为什么kd树能够减少查询次数。但是，例子2中，目标数据点的位置有可能也会要求查询另外一颗子树，加多查询次数，降低查询性能。在维度较高时，这种性能降低十分明显，所以一般kd树要求数据维度在20维以内。通常原则是，如果维度是k, 数据点数是N， 需要满足N >> 2k。

为了应对这一问题，不但出现了对于kd树本身算法的改进如BBF算法，也出现了ball tree一类的针

三 ball tree 和其他树类型介绍

在kd tree 中，我们看到一个导致性能下降的最核心因素是因为kd树的平面是一个个的方形，求最近邻时使用的是圆形。方形平面跟圆形相交的可能性是极高的，如果方形的交汇点多的话，圆形和几个平面相交的可能性就变得更大。这样，凡是相交的平面，都需要进行检查，大大的降低运行效率。



为了解决这一个问题，ball tree抛弃了kd树画的方形，而建立球形，去掉棱角。简而言之，就是使用超球面而不是超矩形划分区域。

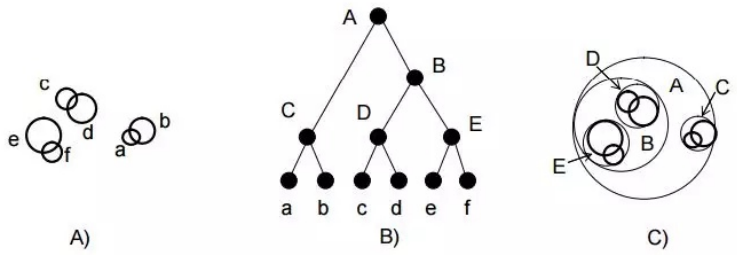


Figure 1. A) A set of balls in the plane. B) A binary tree over these balls. C) The balls in the resulting balltree.

它的构造方式是：所有的数据先构成一个最大的球体。从球中选择一个离球的中心最远的点，然后选择第二个点离第一个点最远。将圆中所有离这两个点最近的观测点都赋给这两个簇的中心，而后计算每一个簇的中心点和包含所有其所属观测点的最小半径。不断递归得到最终结果如上图C所示。在上图B中所示数据结构中A，B，C , D, E非叶子节点存储了球的中心点和半径。

它的查找方式是一样先寻找叶子节点，并计算叶子节点和目标点的距离，此值确定目标最小距离上限。根据构造方式可知，可以根据目标点计算出其叶子节点。此时，唯一需要验证的是其兄弟节点的距离，如果小于上限就更新最近目标点。

也能够看出ball tree的构建比kd tree复杂的多，也耗时的多。但是为了节省查询时间这种耗时也是

知乎

对于高维度索引树，还有VP树和MVP树，这里就不再多做解释了。

四 总结

本文重点讲述了kd树的构建和查询，也同时介绍了kd树的改进数据结构ball树。kd树是KNN算法的重要基础，KNN是最简单的机器学习算法之一，而kd树的复杂估计也是读者没有想到的，所谓细节之处皆是魔鬼，此话诚然。

参考文献

cnblogs.com/lesleysbw/p...

blog.csdn.net/wzgang123...

blog.csdn.net/app_12062...

blog.csdn.net/zjx_adstu...

blog.csdn.net/weixin_41...

好文推荐

sigAI的文章确实写的不错，大家也可以关注他们的公招号：blog.csdn.net/sigai_csd...

cnblogs.com/lesleysbw/p... 格式有点小问题，但是ball tree写的不错。

kd树就给大家介绍到这了，欢迎大家多多提建议并关注我们的公众号。

微信公众号：ARGO Finance创新实验室

知乎 @ARGO创新实验室

微信文章列表索引：

往期文章

mp.weixin.qq.com/s?__biz=MzI0MDQ0NzA…

编辑于 2020-06-22 20:21

机器学习

▲ 赞同 88 ▼

知乎

KD Tree的原理及Python实现

李小文

《
树

KN
本
个
到
高
时

ch

14 条评论 切换为时间排序

写下你的评论...

zxfzxfzxf 2019-11-11
你这篇写的是最好的，其他的文章都是画个圆判断，我高维咋画个圆用眼睛看出来？
5

printf 07-28
有些都没写明白，要不是我提前知道，根本看不懂
1

知乎用户 03-30
知乎还是有用心人的
1

哒哒 07-22
学gis的给你点个赞，讲的很棒
赞

夜曲沉醉 06-07
受益颇丰
赞

斐世 06-07
好难哟
赞

道生 06-01
非常感谢您，讲的很清楚
赞

霎臉 04-23
所以那六个点最后建的kd树的中序遍历是：（2,3），（5,4），（4,7），（7,2），（9,6），（8,1）是吧

赞

▲ 赞同 88 ▼

知乎

为了理解方便，将 (2,3) 到 (2.1,3.1) 的距离为直径，以 (2,3) 为圆心作圆。

这里应该是笔误吧，应该是以(2.1, 3.1)为圆心作圆吧?

赞

whu-jzj 回复 知乎用户 03-30

我也觉着是的

1

静寂旮旯 回复 知乎用户 09-29

应该如此,不然找下个近邻点计算的是(2,3)到父节点的距离,没有意义,而且第二个例子中感觉第一个近邻点应该是最后的叶子结点(4,7),然后再往根搜索

赞

知乎用户 02-04

判断是否回溯的时候，应该不是到父节点而是到父节点所在的超平面的距离吧

赞

JSJ 2020-12-04

请问第一轮横着切的时候为什么有两个点可以选择呢

赞

仝雪菡 2020-12-03

我看不懂但是很好奇

赞

▲ 赞同 88 ▼